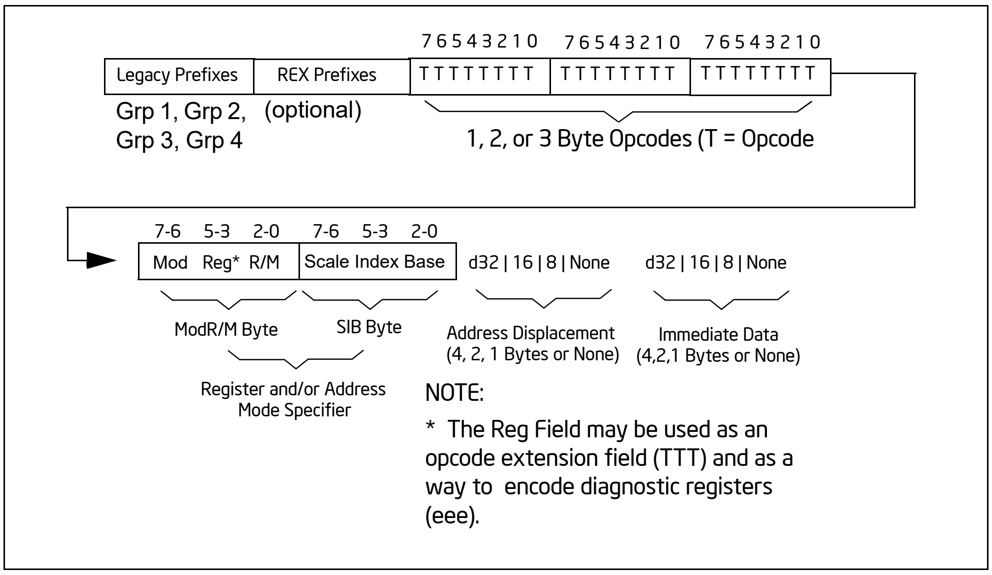
APPENDIX B
INSTRUCTION FORMATS AND ENCODINGS

This appendix provides machine instruction formats and encodings of IA-32 instructions. The first section describes the IA-32 architecture's machine instruction format. The remaining sections show the formats and encoding of general-purpose, MMX, P6 family, SSE/SSE2/SSE3, x87 FPU instructions, and VMX instructions. Those instruction formats also apply to Intel 64 architecture. Instruction formats used in 64-bit mode are provided as supersets of the above.

B.1 MACHINE INSTRUCTION FORMAT

All Intel Architecture instructions are encoded using subsets of the general machine instruction format shown in Figure B-1. Each instruction consists of:

- an opcode
- a register and/or address mode specifier consisting of the ModR/M byte and sometimes the scale-index-base (SIB) byte (if required)
- a displacement and an immediate data field (if required)



7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
Legacy Prefixes
REX Prefixes
T T T T T T T T T T T T T T T T T T T T T T T T
Grp 1, Grp 2, Grp 3, Grp 4
(optional)
1, 2, or 3 Byte Opcodes (T = Opcode
7-6 5-3 2-0 7-6 5-3 2-0
Mod Reg* R/M
Scale Index Base
d32 | 16 | 8 | None
d32 | 16 | 8 | None
ModR/M Byte
SIB Byte
Address Displacement (4, 2, 1 Bytes or None)
Immediate Data (4,2,1 Bytes or None)
Register and/or Address Mode Specifier
NOTE:
* The Reg Field may be used as an opcode extension field (TTT) and as a way to encode diagnostic registers (eee).

Figure B-1. General Machine Instruction Format

The following sections discuss this format.

B.1.1 Legacy Prefixes

The legacy prefixes noted in Figure B-1 include 66H, 67H, F2H and F3H. They are optional, except when F2H, F3H and 66H are used in new instruction extensions. Legacy prefixes must be placed before REX prefixes.

Refer to Chapter 2, "Instruction Format," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*, for more information on legacy prefixes.

B.1.2 REX Prefixes

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 IA-32 operating modes and in compatibility mode. In 64-bit mode, the same opcodes represent the instruction prefix REX and are not treated as individual instructions.

Refer to Chapter 2, “Instruction Format,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*, for more information on REX prefixes.

B.1.3 Opcode Fields

The primary opcode for an instruction is encoded in one to three bytes of the instruction. Within the primary opcode, smaller encoding fields may be defined. These fields vary according to the class of operation being performed.

Almost all instructions that refer to a register and/or memory operand have a register and/or address mode byte following the opcode. This byte, the ModR/M byte, consists of the mod field (2 bits), the reg field (3 bits; this field is sometimes an opcode extension), and the R/M field (3 bits). Certain encodings of the ModR/M byte indicate that a second address mode byte, the SIB byte, must be used.

If the addressing mode specifies a displacement, the displacement value is placed immediately following the ModR/M byte or SIB byte. Possible sizes are 8, 16, or 32 bits. If the instruction specifies an immediate value, the immediate value follows any displacement bytes. The immediate, if specified, is always the last field of the instruction.

Refer to Chapter 2, “Instruction Format,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*, for more information on opcodes.

B.1.4 Special Fields

Table B-1 lists bit fields that appear in certain instructions, sometimes within the opcode bytes. All of these fields (except the d bit) occur in the general-purpose instruction formats in Table B-13.

Table B-1. Special Fields Within Instruction Encodings

Field Name	Description	Number of Bits
reg	General-register specifier (see Table B-4 or B-5).	3
w	Specifies if data is byte or full-sized, where full-sized is 16 or 32 bits (see Table B-6).	1
s	Specifies sign extension of an immediate field (see Table B-7).	1
sreg2	Segment register specifier for CS, SS, DS, ES (see Table B-8).	2
sreg3	Segment register specifier for CS, SS, DS, ES, FS, GS (see Table B-8).	3
eee	Specifies a special-purpose (control or debug) register (see Table B-9).	3
tttn	For conditional instructions, specifies a condition asserted or negated (see Table B-12).	4
d	Specifies direction of data operation (see Table B-11).	1

B.1.4.1 Reg Field (reg) for Non-64-Bit Modes

The reg field in the ModR/M byte specifies a general-purpose register operand. The group of registers specified is modified by the presence and state of the w bit in an encoding (refer to Section B.1.4.3). Table B-2 shows the encoding of the reg field when the w bit is not present in an encoding; Table B-3 shows the encoding of the reg field when the w bit is present.

Table B-2. Encoding of reg Field When w Field is Not Present in Instruction

reg Field	Register Selected during 16-Bit Data Operations	Register Selected during 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Table B-3. Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations		
	Function of w Field	
reg	When w = 0	When w = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
	Function of w Field	
reg	When w = 0	When w = 1
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

B.1.4.2 Reg Field (reg) for 64-Bit Mode

Just like in non-64-bit modes, the reg field in the ModR/M byte specifies a general-purpose register operand. The group of registers specified is modified by the presence of and state of the w bit in an encoding (refer to Section B.1.4.3). Table B-4 shows the encoding of the reg field when the w bit is not present in an encoding; Table B-5 shows the encoding of the reg field when the w bit is present.

Table B-4. Encoding of reg Field When w Field is Not Present in Instruction

reg Field	Register Selected during 16-Bit Data Operations	Register Selected during 32-Bit Data Operations	Register Selected during 64-Bit Data Operations
000	AX	EAX	RAX
001	CX	ECX	RCX
010	DX	EDX	RDX
011	BX	EBX	RBX
100	SP	ESP	RSP
101	BP	EBP	RBP
110	SI	ESI	RSI
111	DI	EDI	RDI

Table B-5. Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations		
	Function of w Field	
reg	When w = 0	When w = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH[1]	SP
101	CH[1]	BP
110	DH[1]	SI
111	BH[1]	DI

Register Specified by reg Field During 32-Bit Data Operations		
	Function of w Field	
reg	When w = 0	When w = 1
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH*	ESP
101	CH*	EBP
110	DH*	ESI
111	BH*	EDI

NOTES:

1. AH, CH, DH, BH can not be encoded when REX prefix is used. Such an expression defaults to the low byte.

B.1.4.3 Encoding of Operand Size (w) Bit

The current operand-size attribute determines whether the processor is performing 16-bit, 32-bit or 64-bit operations. Within the constraints of the current operand-size attribute, the operand-size bit (w) can be used to indicate operations on 8-bit operands or the full operand size specified with the operand-size attribute. Table B-6 shows the encoding of the w bit depending on the current operand-size attribute.

Table B-6. Encoding of Operand Size (w) Bit

w Bit	Operand Size When Operand-Size Attribute is 16 Bits	Operand Size When Operand-Size Attribute is 32 Bits
0	8 Bits	8 Bits
1	16 Bits	32 Bits

B.1.4.4 Sign-Extend (s) Bit

The sign-extend (s) bit occurs in instructions with immediate data fields that are being extended from 8 bits to 16 or 32 bits. See Table B-7.

Table B-7. Encoding of Sign-Extend (s) Bit

s	Effect on 8-Bit Immediate Data	Effect on 16- or 32-Bit Immediate Data
0	None	None
1	Sign-extend to fill 16-bit or 32-bit destination	None

B.1.4.5 Segment Register (sreg) Field

When an instruction operates on a segment register, the reg field in the ModR/M byte is called the sreg field and is used to specify the segment register. Table B-8 shows the encoding of the sreg field. This field is sometimes a 2-bit field (sreg2) and other times a 3-bit field (sreg3).

Table B-8. Encoding of the Segment Register (sreg) Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	Reserved[1]
111	Reserved

NOTES:

1. Do not use reserved encodings.

B.1.4.6 Special-Purpose Register (eee) Field

When control or debug registers are referenced in an instruction they are encoded in the eee field, located in bits 5 though 3 of the ModR/M byte (an alternate encoding of the sreg field). See Table B-9.

Table B-9. Encoding of Special-Purpose Register (eee) Field

eee	Control Register	Debug Register
000	CR0	DR0
001	Reserved[1]	DR1
010	CR2	DR2
011	CR3	DR3
100	CR4	Reserved
101	Reserved	Reserved
110	Reserved	DR6
111	Reserved	DR7

NOTES:

1. Do not use reserved encodings.

B.1.4.7 Condition Test (tttn) Field

For conditional instructions (such as conditional jumps and set on condition), the condition test field (tttn) is encoded for the condition being tested. The ttt part of the field gives the condition to test and the n part indicates whether to use the condition (n = 0) or its negation (n = 1).

- For 1-byte primary opcodes, the tttn field is located in bits 3, 2, 1, and 0 of the opcode byte.
- For 2-byte primary opcodes, the tttn field is located in bits 3, 2, 1, and 0 of the second opcode byte.

Table B-10 shows the encoding of the tttn field.

Table B-10. Encoding of Conditional Test (tttn) Field

t t t n	Mnemonic	Condition
0000	O	Overflow
0001	NO	No overflow
0010	B, NAE	Below, Not above or equal
0011	NB, AE	Not below, Above or equal
0100	E, Z	Equal, Zero
0101	NE, NZ	Not equal, Not zero
0110	BE, NA	Below or equal, Not above
0111	NBE, A	Not below or equal, Above
1000	S	Sign
1001	NS	Not sign
1010	P, PE	Parity, Parity Even
1011	NP, PO	Not parity, Parity Odd
1100	L, NGE	Less than, Not greater than or equal to
1101	NL, GE	Not less than, Greater than or equal to
1110	LE, NG	Less than or equal to, Not greater than
1111	NLE, G	Not less than or equal to, Greater than

B.1.4.8 Direction (d) Bit

In many two-operand instructions, a direction bit (d) indicates which operand is considered the source and which is the destination. See Table B-11.

- When used for integer instructions, the d bit is located at bit 1 of a 1-byte primary opcode. Note that this bit does not appear as the symbol “d” in Table B-13; the actual encoding of the bit as 1 or 0 is given.
- When used for floating-point instructions (in Table B-16), the d bit is shown as bit 2 of the first byte of the primary opcode.

Table B-11. Encoding of Operation Direction (d) Bit

d	Source	Destination
0	reg Field	ModR/M or SIB Byte
1	ModR/M or SIB Byte	reg Field

B.1.5 Other Notes

Table B-12 contains notes on particular encodings. These notes are indicated in the tables shown in the following sections by superscripts.

Table B-12. Notes on Instruction Encoding

Symbol	Note
A	A value of 11B in bits 7 and 6 of the ModR/M byte is reserved.
B	A value of 01B (or 10B) in bits 7 and 6 of the ModR/M byte is reserved.

B.2 GENERAL-PURPOSE INSTRUCTION FORMATS AND ENCODINGS FOR NON-64-BIT MODES

Table B-13 shows machine instruction formats and encodings for general purpose instructions in non-64-bit modes.

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes

Instruction and Format	Encoding
AAA – ASCII Adjust after Addition	0011 0111
AAD – ASCII Adjust AX before Division	1101 0101 : 0000 1010
AAM – ASCII Adjust AX after Multiply	1101 0100 : 0000 1010
AAS – ASCII Adjust AL after Subtraction	0011 1111
ADC – ADD with Carry	
register1 to register2	0001 000w : 11 reg1 reg2
register2 to register1	0001 001w : 11 reg1 reg2
memory to register	0001 001w : mod reg r/m
register to memory	0001 000w : mod reg r/m
immediate to register	1000 00sw : 11 010 reg : immediate data
immediate to AL, AX, or EAX	0001 010w : immediate data
immediate to memory	1000 00sw : mod 010 r/m : immediate data
ADD – Add	
register1 to register2	0000 000w : 11 reg1 reg2
register2 to register1	0000 001w : 11 reg1 reg2
memory to register	0000 001w : mod reg r/m
register to memory	0000 000w : mod reg r/m
immediate to register	1000 00sw : 11 000 reg : immediate data
immediate to AL, AX, or EAX	0000 010w : immediate data
immediate to memory	1000 00sw : mod 000 r/m : immediate data
AND – Logical AND	
register1 to register2	0010 000w : 11 reg1 reg2
register2 to register1	0010 001w : 11 reg1 reg2
memory to register	0010 001w : mod reg r/m
register to memory	0010 000w : mod reg r/m
immediate to register	1000 00sw : 11 100 reg : immediate data
immediate to AL, AX, or EAX	0010 010w : immediate data
immediate to memory	1000 00sw : mod 100 r/m : immediate data

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
ARPL – Adjust RPL Field of Selector	
from register	0110 0011 : 11 reg1 reg2
from memory	0110 0011 : mod reg r/m
BOUND – Check Array Against Bounds	0110 0010 : mod^A reg r/m
BSF – Bit Scan Forward	
register1, register2	0000 1111 : 1011 1100 : 11 reg1 reg2
memory, register	0000 1111 : 1011 1100 : mod reg r/m
BSR – Bit Scan Reverse	
register1, register2	0000 1111 : 1011 1101 : 11 reg1 reg2
memory, register	0000 1111 : 1011 1101 : mod reg r/m
BSWAP – Byte Swap	0000 1111 : 1100 1 reg
BT – Bit Test	
register, immediate	0000 1111 : 1011 1010 : 11 100 reg: imm8 data
memory, immediate	0000 1111 : 1011 1010 : mod 100 r/m : imm8 data
register1, register2	0000 1111 : 1010 0011 : 11 reg2 reg1
memory, reg	0000 1111 : 1010 0011 : mod reg r/m
BTC – Bit Test and Complement	
register, immediate	0000 1111 : 1011 1010 : 11 111 reg: imm8 data
memory, immediate	0000 1111 : 1011 1010 : mod 111 r/m : imm8 data
register1, register2	0000 1111 : 1011 1011 : 11 reg2 reg1
memory, reg	0000 1111 : 1011 1011 : mod reg r/m
BTR – Bit Test and Reset	
register, immediate	0000 1111 : 1011 1010 : 11 110 reg: imm8 data
memory, immediate	0000 1111 : 1011 1010 : mod 110 r/m : imm8 data
register1, register2	0000 1111 : 1011 0011 : 11 reg2 reg1
memory, reg	0000 1111 : 1011 0011 : mod reg r/m
BTS – Bit Test and Set	
register, immediate	0000 1111 : 1011 1010 : 11 101 reg: imm8 data
memory, immediate	0000 1111 : 1011 1010 : mod 101 r/m : imm8 data
register1, register2	0000 1111 : 1010 1011 : 11 reg2 reg1
memory, reg	0000 1111 : 1010 1011 : mod reg r/m
CALL – Call Procedure (in same segment)	
direct	1110 1000 : full displacement
register indirect	1111 1111 : 11 010 reg
memory indirect	1111 1111 : mod 010 r/m
CALL – Call Procedure (in other segment)	
direct	1001 1010 : unsigned full offset, selector
indirect	1111 1111 : mod 011 r/m

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
CBW – Convert Byte to Word	1001 1000
CDQ – Convert Doubleword to Qword	1001 1001
CLC – Clear Carry Flag	1111 1000
CLD – Clear Direction Flag	1111 1100
CLI – Clear Interrupt Flag	1111 1010
CLTS – Clear Task-Switched Flag in CR0	0000 1111 : 0000 0110
CMC – Complement Carry Flag	1111 0101
CMP – Compare Two Operands	
register1 with register2	0011 100w : 11 reg1 reg2
register2 with register1	0011 101w : 11 reg1 reg2
memory with register	0011 100w : mod reg r/m
register with memory	0011 101w : mod reg r/m
immediate with register	1000 00sw : 11 111 reg : immediate data
immediate with AL, AX, or EAX	0011 110w : immediate data
immediate with memory	1000 00sw : mod 111 r/m : immediate data
CMPS/CMPSB/CMPSW/CMPSD – Compare String Operands	1010 011w
CMPXCHG – Compare and Exchange	
register1, register2	0000 1111 : 1011 000w : 11 reg2 reg1
memory, register	0000 1111 : 1011 000w : mod reg r/m
CPUID – CPU Identification	0000 1111 : 1010 0010
CWD – Convert Word to Doubleword	1001 1001
CWDE – Convert Word to Doubleword	1001 1000
DAA – Decimal Adjust AL after Addition	0010 0111
DAS – Decimal Adjust AL after Subtraction	0010 1111
DEC – Decrement by 1	
register	1111 111w : 11 001 reg
register (alternate encoding)	0100 1 reg
memory	1111 111w : mod 001 r/m
DIV – Unsigned Divide	
AL, AX, or EAX by register	1111 011w : 11 110 reg
AL, AX, or EAX by memory	1111 011w : mod 110 r/m
HLT – Halt	1111 0100
IDIV – Signed Divide	
AL, AX, or EAX by register	1111 011w : 11 111 reg
AL, AX, or EAX by memory	1111 011w : mod 111 r/m

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
IMUL – Signed Multiply	
AL, AX, or EAX with register	1111 011w : 11 101 reg
AL, AX, or EAX with memory	1111 011w : mod 101 reg
register1 with register2	0000 1111 : 1010 1111 : 11 : reg1 reg2
register with memory	0000 1111 : 1010 1111 : mod reg r/m
register1 with immediate to register2	0110 10s1 : 11 reg1 reg2 : immediate data
memory with immediate to register	0110 10s1 : mod reg r/m : immediate data
IN – Input From Port	
fixed port	1110 010w : port number
variable port	1110 110w
INC – Increment by 1	
reg	1111 111w : 11 000 reg
reg (alternate encoding)	0100 0 reg
memory	1111 111w : mod 000 r/m
INS – Input from DX Port	0110 110w
INT n – Interrupt Type n	1100 1101 : type
INT – Single-Step Interrupt 3	1100 1100
INTO – Interrupt 4 on Overflow	1100 1110
INVD – Invalidate Cache	0000 1111 : 0000 1000
INVLPG – Invalidate TLB Entry	0000 1111 : 0000 0001 : mod 111 r/m
INVPCID – Invalidate Process-Context Identifier	0110 0110:0000 1111:0011 1000:1000 0010: mod reg r/m
IRET/IRETD – Interrupt Return	1100 1111
J*cc* – Jump if Condition is Met	
8-bit displacement	0111 tttn : 8-bit displacement
full displacement	0000 1111 : 1000 tttn : full displacement
JCXZ/JECXZ – Jump on CX/ECX Zero Address-size prefix differentiates JCXZ and JECXZ	1110 0011 : 8-bit displacement
JMP – Unconditional Jump (to same segment)	
short	1110 1011 : 8-bit displacement
direct	1110 1001 : full displacement
register indirect	1111 1111 : 11 100 reg
memory indirect	1111 1111 : mod 100 r/m
JMP – Unconditional Jump (to other segment)	
direct intersegment	1110 1010 : unsigned full offset, selector
indirect intersegment	1111 1111 : mod 101 r/m
LAHF – Load Flags into AHRegister	1001 1111

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
LAR – Load Access Rights Byte	
from register	0000 1111 : 0000 0010 : 11 reg1 reg2
from memory	0000 1111 : 0000 0010 : mod reg r/m
LDS – Load Pointer to DS	1100 0101 : mod^{A,B} reg r/m
LEA – Load Effective Address	1000 1101 : mod^{A} reg r/m
LEAVE – High Level Procedure Exit	1100 1001
LES – Load Pointer to ES	1100 0100 : mod^{A,B} reg r/m
LFS – Load Pointer to FS	0000 1111 : 1011 0100 : mod^{A} reg r/m
LGDT – Load Global Descriptor Table Register	0000 1111 : 0000 0001 : mod^{A} 010 r/m
LGS – Load Pointer to GS	0000 1111 : 1011 0101 : mod^{A} reg r/m
LIDT – Load Interrupt Descriptor Table Register	0000 1111 : 0000 0001 : mod^{A} 011 r/m
LLDT – Load Local Descriptor Table Register	
LDTR from register	0000 1111 : 0000 0000 : 11 010 reg
LDTR from memory	0000 1111 : 0000 0000 : mod 010 r/m
LMSW – Load Machine Status Word	
from register	0000 1111 : 0000 0001 : 11 110 reg
from memory	0000 1111 : 0000 0001 : mod 110 r/m
LOCK – Assert LOCK# Signal Prefix	1111 0000
LODS/LODSB/LODSW/LODSD – Load String Operand	1010 110w
LOOP – Loop Count	1110 0010 : 8-bit displacement
LOOPZ/LOOPE – Loop Count while Zero/Equal	1110 0001 : 8-bit displacement
LOOPNZ/LOOPNE – Loop Count while not Zero/Equal	1110 0000 : 8-bit displacement
LSL – Load Segment Limit	
from register	0000 1111 : 0000 0011 : 11 reg1 reg2
from memory	0000 1111 : 0000 0011 : mod reg r/m
LSS – Load Pointer to SS	0000 1111 : 1011 0010 : mod^{A} reg r/m
LTR – Load Task Register	
from register	0000 1111 : 0000 0000 : 11 011 reg
from memory	0000 1111 : 0000 0000 : mod 011 r/m
MOV – Move Data	
register1 to register2	1000 100w : 11 reg1 reg2
register2 to register1	1000 101w : 11 reg1 reg2
memory to reg	1000 101w : mod reg r/m
reg to memory	1000 100w : mod reg r/m
immediate to register	1100 011w : 11 000 reg : immediate data
immediate to register (alternate encoding)	1011 w reg : immediate data
immediate to memory	1100 011w : mod 000 r/m : immediate data
memory to AL, AX, or EAX	1010 000w : full displacement

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
AL, AX, or EAX to memory	1010 001w : full displacement
MOV – Move to/from Control Registers	
CR0 from register	0000 1111 : 0010 0010 : -- 000 reg
CR2 from register	0000 1111 : 0010 0010 : -- 010reg
CR3 from register	0000 1111 : 0010 0010 : -- 011 reg
CR4 from register	0000 1111 : 0010 0010 : -- 100 reg
register from CR0-CR4	0000 1111 : 0010 0000 : -- eee reg
MOV – Move to/from Debug Registers	
DR0-DR3 from register	0000 1111 : 0010 0011 : -- eee reg
DR4-DR5 from register	0000 1111 : 0010 0011 : -- eee reg
DR6-DR7 from register	0000 1111 : 0010 0011 : -- eee reg
register from DR6-DR7	0000 1111 : 0010 0001 : -- eee reg
register from DR4-DR5	0000 1111 : 0010 0001 : -- eee reg
register from DR0-DR3	0000 1111 : 0010 0001 : -- eee reg
MOV – Move to/from Segment Registers	
register to segment register	1000 1110 : 11 sreg3 reg
register to SS	1000 1110 : 11 sreg3 reg
memory to segment reg	1000 1110 : mod sreg3 r/m
memory to SS	1000 1110 : mod sreg3 r/m
segment register to register	1000 1100 : 11 sreg3 reg
segment register to memory	1000 1100 : mod sreg3 r/m
MOVBE – Move data after swapping bytes	
memory to register	0000 1111 : 0011 1000:1111 0000 : mod reg r/m
register to memory	0000 1111 : 0011 1000:1111 0001 : mod reg r/m
MOVS/MOVSB/MOVSW/MOVSD – Move Data from String to String	1010 010w
MOVSX – Move with Sign-Extend	
memory to reg	0000 1111 : 1011 111w : mod reg r/m
MOVZX – Move with Zero-Extend	
register2 to register1	0000 1111 : 1011 011w : 11 reg1 reg2
memory to register	0000 1111 : 1011 011w : mod reg r/m
MUL – Unsigned Multiply	
AL, AX, or EAX with register	1111 011w : 11 100 reg
AL, AX, or EAX with memory	1111 011w : mod 100 r/m
NEG – Two's Complement Negation	
register	1111 011w : 11 011 reg
memory	1111 011w : mod 011 r/m
NOP – No Operation	1001 0000

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
NOP – Multi-byte No Operation[1]	
register	0000 1111 0001 1111 : 11 000 reg
memory	0000 1111 0001 1111 : mod 000 r/m
NOT – One's Complement Negation	
register	1111 011w : 11 010 reg
memory	1111 011w : mod 010 r/m
OR – Logical Inclusive OR	
register1 to register2	0000 100w : 11 reg1 reg2
register2 to register1	0000 101w : 11 reg1 reg2
memory to register	0000 101w : mod reg r/m
register to memory	0000 100w : mod reg r/m
immediate to register	1000 00sw : 11 001 reg : immediate data
immediate to AL, AX, or EAX	0000 110w : immediate data
immediate to memory	1000 00sw : mod 001 r/m : immediate data
OUT – Output to Port	
fixed port	1110 011w : port number
variable port	1110 111w
OUTS – Output to DX Port	0110 111w
POP – Pop a Word from the Stack	
register	1000 1111 : 11 000 reg
register (alternate encoding)	0101 1 reg
memory	1000 1111 : mod 000 r/m
POP – Pop a Segment Register from the Stack (Note: CS cannot be sreg2 in this usage.)	
segment register DS, ES	000 sreg2 111
segment register SS	000 sreg2 111
segment register FS, GS	0000 1111: 10 sreg3 001
POPA/POPAD – Pop All General Registers	0110 0001
POPF/POPFD – Pop Stack into FLAGS or EFLAGS Register	1001 1101
PUSH – Push Operand onto the Stack	
register	1111 1111 : 11 110 reg
register (alternate encoding)	0101 0 reg
memory	1111 1111 : mod 110 r/m
immediate	0110 10s0 : immediate data
PUSH – Push Segment Register onto the Stack	
segment register CS,DS,ES,SS	000 sreg2 110
segment register FS,GS	0000 1111: 10 sreg3 000

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
PUSHA/PUSHAD – Push All General Registers	0110 0000
PUSHF/PUSHFD – Push Flags Register onto the Stack	1001 1100
RCL – Rotate thru Carry Left	
register by 1	1101 000w : 11 010 reg
memory by 1	1101 000w : mod 010 r/m
register by CL	1101 001w : 11 010 reg
memory by CL	1101 001w : mod 010 r/m
register by immediate count	1100 000w : 11 010 reg : imm8 data
memory by immediate count	1100 000w : mod 010 r/m : imm8 data
RCR – Rotate thru Carry Right	
register by 1	1101 000w : 11 011 reg
memory by 1	1101 000w : mod 011 r/m
register by CL	1101 001w : 11 011 reg
memory by CL	1101 001w : mod 011 r/m
register by immediate count	1100 000w : 11 011 reg : imm8 data
memory by immediate count	1100 000w : mod 011 r/m : imm8 data
RDMSR – Read from Model-Specific Register	0000 1111 : 0011 0010
RDPMC – Read Performance Monitoring Counters	0000 1111 : 0011 0011
RDTSC – Read Time-Stamp Counter	0000 1111 : 0011 0001
RDTSCP – Read Time-Stamp Counter and Processor ID	0000 1111 : 0000 0001: 1111 1001
REP INS – Input String	1111 0011 : 0110 110w
REP LODS – Load String	1111 0011 : 1010 110w
REP MOVS – Move String	1111 0011 : 1010 010w
REP OUTS – Output String	1111 0011 : 0110 111w
REP STOS – Store String	1111 0011 : 1010 101w
REPE CMPS – Compare String	1111 0011 : 1010 011w
REPE SCAS – Scan String	1111 0011 : 1010 111w
REPNE CMPS – Compare String	1111 0010 : 1010 011w
REPNE SCAS – Scan String	1111 0010 : 1010 111w
RET – Return from Procedure (to same segment)	
no argument	1100 0011
adding immediate to SP	1100 0010 : 16-bit displacement
RET – Return from Procedure (to other segment)	
intersegment	1100 1011
adding immediate to SP	1100 1010 : 16-bit displacement

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
ROL – Rotate Left	
register by 1	1101 000w : 11 000 reg
memory by 1	1101 000w : mod 000 r/m
register by CL	1101 001w : 11 000 reg
memory by CL	1101 001w : mod 000 r/m
register by immediate count	1100 000w : 11 000 reg : imm8 data
memory by immediate count	1100 000w : mod 000 r/m : imm8 data
ROR – Rotate Right	
register by 1	1101 000w : 11 001 reg
memory by 1	1101 000w : mod 001 r/m
register by CL	1101 001w : 11 001 reg
memory by CL	1101 001w : mod 001 r/m
register by immediate count	1100 000w : 11 001 reg : imm8 data
memory by immediate count	1100 000w : mod 001 r/m : imm8 data
RSM – Resume from System Management Mode	0000 1111 : 1010 1010
SAHF – Store AH into Flags	1001 1110
SAL – Shift Arithmetic Left	same instruction as SHL
SAR – Shift Arithmetic Right	
register by 1	1101 000w : 11 111 reg
memory by 1	1101 000w : mod 111 r/m
register by CL	1101 001w : 11 111 reg
memory by CL	1101 001w : mod 111 r/m
register by immediate count	1100 000w : 11 111 reg : imm8 data
memory by immediate count	1100 000w : mod 111 r/m : imm8 data
SBB – Integer Subtraction with Borrow	
register1 to register2	0001 100w : 11 reg1 reg2
register2 to register1	0001 101w : 11 reg1 reg2
memory to register	0001 101w : mod reg r/m
register to memory	0001 100w : mod reg r/m
immediate to register	1000 00sw : 11 011 reg : immediate data
immediate to AL, AX, or EAX	0001 110w : immediate data
immediate to memory	1000 00sw : mod 011 r/m : immediate data
SCAS/SCASB/SCASW/SCASD – Scan String	1010 111w
SETcc – Byte Set on Condition	
register	0000 1111 : 1001 tttn : 11 000 reg
memory	0000 1111 : 1001 tttn : mod 000 r/m
SGDT – Store Global Descriptor Table Register	0000 1111 : 0000 0001 : mod^{A} 000 r/m

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
SHL – Shift Left	
register by 1	1101 000w : 11 100 reg
memory by 1	1101 000w : mod 100 r/m
register by CL	1101 001w : 11 100 reg
memory by CL	1101 001w : mod 100 r/m
register by immediate count	1100 000w : 11 100 reg : imm8 data
memory by immediate count	1100 000w : mod 100 r/m : imm8 data
SHLD – Double Precision Shift Left	
register by immediate count	0000 1111 : 1010 0100 : 11 reg2 reg1 : imm8
memory by immediate count	0000 1111 : 1010 0100 : mod reg r/m : imm8
register by CL	0000 1111 : 1010 0101 : 11 reg2 reg1
memory by CL	0000 1111 : 1010 0101 : mod reg r/m
SHR – Shift Right	
register by 1	1101 000w : 11 101 reg
memory by 1	1101 000w : mod 101 r/m
register by CL	1101 001w : 11 101 reg
memory by CL	1101 001w : mod 101 r/m
register by immediate count	1100 000w : 11 101 reg : imm8 data
memory by immediate count	1100 000w : mod 101 r/m : imm8 data
SHRD – Double Precision Shift Right	
register by immediate count	0000 1111 : 1010 1100 : 11 reg2 reg1 : imm8
memory by immediate count	0000 1111 : 1010 1100 : mod reg r/m : imm8
register by CL	0000 1111 : 1010 1101 : 11 reg2 reg1
memory by CL	0000 1111 : 1010 1101 : mod reg r/m
SIDT – Store Interrupt Descriptor Table Register	0000 1111 : 0000 0001 : mod^{A} 001 r/m
SLDT – Store Local Descriptor Table Register	
to register	0000 1111 : 0000 0000 : 11 000 reg
to memory	0000 1111 : 0000 0000 : mod 000 r/m
SMSW – Store Machine Status Word	
to register	0000 1111 : 0000 0001 : 11 100 reg
to memory	0000 1111 : 0000 0001 : mod 100 r/m
STC – Set Carry Flag	1111 1001
STD – Set Direction Flag	1111 1101
STI – Set Interrupt Flag	1111 1011
STOS/STOSB/STOSW/STOSD – Store String Data	1010 101w
STR – Store Task Register	
to register	0000 1111 : 0000 0000 : 11 001 reg
to memory	0000 1111 : 0000 0000 : mod 001 r/m

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
SUB – Integer Subtraction	
register1 to register2	0010 100w : 11 reg1 reg2
register2 to register1	0010 101w : 11 reg1 reg2
memory to register	0010 101w : mod reg r/m
register to memory	0010 100w : mod reg r/m
immediate to register	1000 00sw : 11 101 reg : immediate data
immediate to AL, AX, or EAX	0010 110w : immediate data
immediate to memory	1000 00sw : mod 101 r/m : immediate data
TEST – Logical Compare	
register1 and register2	1000 010w : 11 reg1 reg2
memory and register	1000 010w : mod reg r/m
immediate and register	1111 011w : 11 000 reg : immediate data
immediate and AL, AX, or EAX	1010 100w : immediate data
immediate and memory	1111 011w : mod 000 r/m : immediate data
UD0 – Undefined instruction	0000 1111 : 1111 1111
UD1 – Undefined instruction	0000 1111 : 0000 1011
UD2 – Undefined instruction	0000 FFFF : 0000 1011
VERR – Verify a Segment for Reading	
register	0000 1111 : 0000 0000 : 11 100 reg
memory	0000 1111 : 0000 0000 : mod 100 r/m
VERW – Verify a Segment for Writing	
register	0000 1111 : 0000 0000 : 11 101 reg
memory	0000 1111 : 0000 0000 : mod 101 r/m
WAIT – Wait	1001 1011
WBINVD – Writeback and Invalidate Data Cache	0000 1111 : 0000 1001
WRMSR – Write to Model-Specific Register	0000 1111 : 0011 0000
XADD – Exchange and Add	
register1, register2	0000 1111 : 1100 000w : 11 reg2 reg1
memory, reg	0000 1111 : 1100 000w : mod reg r/m
XCHG – Exchange Register/Memory with Register	
register1 with register2	1000 011w : 11 reg1 reg2
AX or EAX with reg	1001 0 reg
memory with reg	1000 011w : mod reg r/m
XLAT/XLATB – Table Look-up Translation	1101 0111
XOR – Logical Exclusive OR	
register1 to register2	0011 000w : 11 reg1 reg2
register2 to register1	0011 001w : 11 reg1 reg2
memory to register	0011 001w : mod reg r/m

Table B-13. General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
register to memory	0011 000w : mod reg r/m
immediate to register	1000 00sw : 11 110 reg : immediate data
immediate to AL, AX, or EAX	0011 010w : immediate data
immediate to memory	1000 00sw : mod 110 r/m : immediate data
Prefix Bytes	
address size	0110 0111
LOCK	1111 0000
operand size	0110 0110
CS segment override	0010 1110
DS segment override	0011 1110
ES segment override	0010 0110
FS segment override	0110 0100
GS segment override	0110 0101
SS segment override	0011 0110

NOTES:

1. The multi-byte NOP instruction does not alter the content of the register and will not issue a memory operation.

B.2.1 General Purpose Instruction Formats and Encodings for 64-Bit Mode

Table B-15 shows machine instruction formats and encodings for general purpose instructions in 64-bit mode.

Table B-14. Special Symbols

Symbol	Application
S	If the value of REX.W. is 1, it overrides the presence of 66H.
w	The value of bit W. in REX is has no effect.

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode

Instruction and Format	Encoding
ADC – ADD with Carry	
register1 to register2	0100 0R0B : 0001 000w : 11 reg1 reg2
qwordregister1 to qwordregister2	0100 1R0B : 0001 0001 : 11 qwordreg1 qwordreg2
register2 to register1	0100 0R0B : 0001 001w : 11 reg1 reg2
qwordregister1 to qwordregister2	0100 1R0B : 0001 0011 : 11 qwordreg1 qwordreg2
memory to register	0100 0RXB : 0001 001w : mod reg r/m
memory to qwordregister	0100 1RXB : 0001 0011 : mod qwordreg r/m
register to memory	0100 0RXB : 0001 000w : mod reg r/m
qwordregister to memory	0100 1RXB : 0001 0001 : mod qwordreg r/m
immediate to register	0100 000B : 1000 00sw : 11 010 reg : immediate
immediate to qwordregister	0100 100B : 1000 0001 : 11 010 qwordreg : imm32
immediate to qwordregister	0100 1R0B : 1000 0011 : 11 010 qwordreg : imm8

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
immediate to AL, AX, or EAX	0001 010w : immediate data
immediate to RAX	0100 1000 : 0000 0101 : imm32
immediate to memory	0100 00XB : 1000 00sw : mod 010 r/m : immediate
immediate32 to memory64	0100 10XB : 1000 0001 : mod 010 r/m : imm32
immediate8 to memory64	0100 10XB : 1000 0031 : mod 010 r/m : imm8
ADD – Add	
register1 to register2	0100 0R0B : 0000 000w : 11 reg1 reg2
qwordregister1 to qwordregister2	0100 1R0B 0000 0000 : 11 qwordreg1 qwordreg2
register2 to register1	0100 0R0B : 0000 001w : 11 reg1 reg2
qwordregister1 to qwordregister2	0100 1R0B 0000 0010 : 11 qwordreg1 qwordreg2
memory to register	0100 0RXB : 0000 001w : mod reg r/m
memory64 to qwordregister	0100 1RXB : 0000 0000 : mod qwordreg r/m
register to memory	0100 0RXB : 0000 000w : mod reg r/m
qwordregister to memory64	0100 1RXB : 0000 0011 : mod qwordreg r/m
immediate to register	0100 0000B : 1000 00sw : 11 000 reg : immediate data
immediate32 to qwordregister	0100 100B : 1000 0001 : 11 010 qwordreg : imm
immediate to AL, AX, or EAX	0000 010w : immediate8
immediate to RAX	0100 1000 : 0000 0101 : imm32
immediate to memory	0100 00XB : 1000 00sw : mod 000 r/m : immediate
immediate32 to memory64	0100 10XB : 1000 0001 : mod 010 r/m : imm32
immediate8 to memory64	0100 10XB : 1000 0011 : mod 010 r/m : imm8
AND – Logical AND	
register1 to register2	0100 0R0B 0010 000w : 11 reg1 reg2
qwordregister1 to qwordregister2	0100 1R0B 0010 0001 : 11 qwordreg1 qwordreg2
register2 to register1	0100 0R0B 0010 001w : 11 reg1 reg2
register1 to register2	0100 1R0B 0010 0011 : 11 qwordreg1 qwordreg2
memory to register	0100 0RXB 0010 001w : mod reg r/m
memory64 to qwordregister	0100 1RXB : 0010 0011 : mod qwordreg r/m
register to memory	0100 0RXB : 0010 000w : mod reg r/m
qwordregister to memory64	0100 1RXB : 0010 0001 : mod qwordreg r/m
immediate to register	0100 000B : 1000 00sw : 11 100 reg : immediate
immediate32 to qwordregister	0100 100B 1000 0001 : 11 100 qwordreg : imm32
immediate to AL, AX, or EAX	0010 010w : immediate
immediate32 to RAX	0100 1000 0010 1001 : imm32
immediate to memory	0100 00XB : 1000 00sw : mod 100 r/m : immediate
immediate32 to memory64	0100 10XB : 1000 0001 : mod 100 r/m : immediate32
immediate8 to memory64	0100 10XB : 1000 0011 : mod 100 r/m : imm8
BSF – Bit Scan Forward	

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
register1, register2	0100 0R0B 0000 1111 : 1011 1100 : 11 reg1 reg2
qwordregister1, qwordregister2	0100 1R0B 0000 1111 : 1011 1100 : 11 qwordreg1 qwordreg2
memory, register	0100 0RXB 0000 1111 : 1011 1100 : mod reg r/m
memory64, qwordregister	0100 1RXB 0000 1111 : 1011 1100 : mod qwordreg r/m
BSR – Bit Scan Reverse	
register1, register2	0100 0R0B 0000 1111 : 1011 1101 : 11 reg1 reg2
qwordregister1, qwordregister2	0100 1R0B 0000 1111 : 1011 1101 : 11 qwordreg1 qwordreg2
memory, register	0100 0RXB 0000 1111 : 1011 1101 : mod reg r/m
memory64, qwordregister	0100 1RXB 0000 1111 : 1011 1101 : mod qwordreg r/m
BSWAP – Byte Swap	0000 1111 : 1100 1 reg
BSWAP – Byte Swap	0100 100B 0000 1111 : 1100 1 qwordreg
BT – Bit Test	
register, immediate	0100 000B 0000 1111 : 1011 1010 : 11 100 reg: imm8
qwordregister, immediate8	0100 100B 1111 : 1011 1010 : 11 100 qwordreg: imm8 data
memory, immediate	0100 00XB 0000 1111 : 1011 1010 : mod 100 r/m : imm8
memory64, immediate8	0100 10XB 0000 1111 : 1011 1010 : mod 100 r/m : imm8 data
register1, register2	0100 0R0B 0000 1111 : 1010 0011 : 11 reg2 reg1
qwordregister1, qwordregister2	0100 1R0B 0000 1111 : 1010 0011 : 11 qwordreg2 qwordreg1
memory, reg	0100 0RXB 0000 1111 : 1010 0011 : mod reg r/m
memory, qwordreg	0100 1RXB 0000 1111 : 1010 0011 : mod qwordreg r/m
BTC – Bit Test and Complement	
register, immediate	0100 000B 0000 1111 : 1011 1010 : 11 111 reg: imm8
qwordregister, immediate8	0100 100B 0000 1111 : 1011 1010 : 11 111 qwordreg: imm8
memory, immediate	0100 00XB 0000 1111 : 1011 1010 : mod 111 r/m : imm8
memory64, immediate8	0100 10XB 0000 1111 : 1011 1010 : mod 111 r/m : imm8
register1, register2	0100 0R0B 0000 1111 : 1011 1011 : 11 reg2 reg1
qwordregister1, qwordregister2	0100 1R0B 0000 1111 : 1011 1011 : 11 qwordreg2 qwordreg1
memory, register	0100 0RXB 0000 1111 : 1011 1011 : mod reg r/m
memory, qwordreg	0100 1RXB 0000 1111 : 1011 1011 : mod qwordreg r/m
BTR – Bit Test and Reset	
register, immediate	0100 000B 0000 1111 : 1011 1010 : 11 110 reg: imm8
qwordregister, immediate8	0100 100B 0000 1111 : 1011 1010 : 11 110 qwordreg: imm8
memory, immediate	0100 00XB 0000 1111 : 1011 1010 : mod 110 r/m : imm8
memory64, immediate8	0100 10XB 0000 1111 : 1011 1010 : mod 110 r/m : imm8
register1, register2	0100 0R0B 0000 1111 : 1011 0011 : 11 reg2 reg1

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
qwordregister1, qwordregister2	0100 1R0B 0000 1111 : 1011 0011 : 11 qwordreg2 qwordreg1
memory, register	0100 0RXB 0000 1111 : 1011 0011 : mod reg r/m
memory64, qwordreg	0100 1RXB 0000 1111 : 1011 0011 : mod qwordreg r/m
BTS – Bit Test and Set	
register, immediate	0100 000B 0000 1111 : 1011 1010 : 11 101 reg: imm8
qwordregister, immediate8	0100 100B 0000 1111 : 1011 1010 : 11 101 qwordreg: imm8
memory, immediate	0100 00XB 0000 1111 : 1011 1010 : mod 101 r/m : imm8
memory64, immediate8	0100 10XB 0000 1111 : 1011 1010 : mod 101 r/m : imm8
register1, register2	0100 0R0B 0000 1111 : 1010 1011 : 11 reg2 reg1
qwordregister1, qwordregister2	0100 1R0B 0000 1111 : 1010 1011 : 11 qwordreg2 qwordreg1
memory, register	0100 0RXB 0000 1111 : 1010 1011 : mod reg r/m
memory64, qwordreg	0100 1RXB 0000 1111 : 1010 1011 : mod qwordreg r/m
CALL – Call Procedure (in same segment)	
direct	1110 1000 : displacement32
register indirect	0100 WR00^{w} 1111 1111 : 11 010 reg
memory indirect	0100 W0XB^{w} 1111 1111 : mod 010 r/m
CALL – Call Procedure (in other segment)	
indirect	1111 1111 : mod 011 r/m
indirect	0100 10XB 0100 1000 1111 1111 : mod 011 r/m
CBW – Convert Byte to Word	1001 1000
CDQ – Convert Doubleword to Qword+	1001 1001
CDQE – RAX, Sign-Extend of EAX	0100 1000 1001 1001
CLC – Clear Carry Flag	1111 1000
CLD – Clear Direction Flag	1111 1100
CLI – Clear Interrupt Flag	1111 1010
CLTS – Clear Task-Switched Flag in CR0	0000 1111 : 0000 0110
CMC – Complement Carry Flag	1111 0101
CMP – Compare Two Operands	
register1 with register2	0100 0R0B 0011 100w : 11 reg1 reg2
qwordregister1 with qwordregister2	0100 1R0B 0011 1001 : 11 qwordreg1 qwordreg2
register2 with register1	0100 0R0B 0011 101w : 11 reg1 reg2
qwordregister2 with qwordregister1	0100 1R0B 0011 101w : 11 qwordreg1 qwordreg2
memory with register	0100 0RXB 0011 100w : mod reg r/m
memory64 with qwordregister	0100 1RXB 0011 1001 : mod qwordreg r/m
register with memory	0100 0RXB 0011 101w : mod reg r/m
qwordregister with memory64	0100 1RXB 0011 101w1 : mod qwordreg r/m
immediate with register	0100 000B 1000 00sw : 11 111 reg : imm

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
immediate32 with qwordregister	0100 100B 1000 0001 : 11 111 qwordreg : imm64
immediate with AL, AX, or EAX	0011 110w : imm
immediate32 with RAX	0100 1000 0011 1101 : imm32
immediate with memory	0100 00XB 1000 00sw : mod 111 r/m : imm
immediate32 with memory64	0100 1RXB 1000 0001 : mod 111 r/m : imm64
immediate8 with memory64	0100 1RXB 1000 0011 : mod 111 r/m : imm8
CMPS/CMPSB/CMPSW/CMPSD/CMPSQ – Compare String Operands	
compare string operands [X at DS:(E)SI with Y at ES:(E)DI]	1010 011w
qword at address RSI with qword at address RDI	0100 1000 1010 0111
CMPXCHG – Compare and Exchange	
register1, register2	0000 1111 : 1011 000w : 11 reg2 reg1
byteregister1, byteregister2	0100 000B 0000 1111 : 1011 0000 : 11 bytereg2 reg1
qwordregister1, qwordregister2	0100 100B 0000 1111 : 1011 0001 : 11 qwordreg2 reg1
memory, register	0000 1111 : 1011 000w : mod reg r/m
memory8, byteregister	0100 00XB 0000 1111 : 1011 0000 : mod bytereg r/m
memory64, qwordregister	0100 10XB 0000 1111 : 1011 0001 : mod qwordreg r/m
CPUID – CPU Identification	0000 1111 : 1010 0010
CQO – Sign-Extend RAX	0100 1000 1001 1001
CWD – Convert Word to Doubleword	1001 1001
CWDE – Convert Word to Doubleword	1001 1000
DEC – Decrement by 1	
register	0100 000B 1111 111w : 11 001 reg
qwordregister	0100 100B 1111 1111 : 11 001 qwordreg
memory	0100 00XB 1111 111w : mod 001 r/m
memory64	0100 10XB 1111 1111 : mod 001 r/m
DIV – Unsigned Divide	
AL, AX, or EAX by register	0100 000B 1111 011w : 11 110 reg
Divide RDX:RAX by qwordregister	0100 100B 1111 0111 : 11 110 qwordreg
AL, AX, or EAX by memory	0100 00XB 1111 011w : mod 110 r/m
Divide RDX:RAX by memory64	0100 10XB 1111 0111 : mod 110 r/m
ENTER – Make Stack Frame for High Level Procedure	1100 1000 : 16-bit displacement : 8-bit level (L)
HLT – Halt	1111 0100
IDIV – Signed Divide	
AL, AX, or EAX by register	0100 000B 1111 011w : 11 111 reg
RDX:RAX by qwordregister	0100 100B 1111 0111 : 11 111 qwordreg
AL, AX, or EAX by memory	0100 00XB 1111 011w : mod 111 r/m
RDX:RAX by memory64	0100 10XB 1111 0111 : mod 111 r/m
IMUL – Signed Multiply	

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
AL, AX, or EAX with register	0100 000B 1111 011w : 11 101 reg
RDX:RAX <- RAX with qwordregister	0100 100B 1111 0111 : 11 101 qwordreg
AL, AX, or EAX with memory	0100 00XB 1111 011w : mod 101 r/m
RDX:RAX <- RAX with memory64	0100 10XB 1111 0111 : mod 101 r/m
register1 with register2	0000 1111 : 1010 1111 : 11 : reg1 reg2
qwordregister1 <- qwordregister1 with qwordregister2	0100 1R0B 0000 1111 : 1010 1111 : 11 : qwordreg1 qwordreg2
register with memory	0100 0RXB 0000 1111 : 1010 1111 : mod reg r/m
qwordregister <- qwordregister withmemory64	0100 1RXB 0000 1111 : 1010 1111 : mod qwordreg r/m
register1 with immediate to register2	0100 0R0B 0110 10s1 : 11 reg1 reg2 : imm
qwordregister1 <- qwordregister2 with sign-extended immediate8	0100 1R0B 0110 1011 : 11 qwordreg1 qwordreg2 : imm8
qwordregister1 <- qwordregister2 with immediate32	0100 1R0B 0110 1001 : 11 qwordreg1 qwordreg2 : imm32
memory with immediate to register	0100 0RXB 0110 10s1 : mod reg r/m : imm
qwordregister <- memory64 with sign-extended immediate8	0100 1RXB 0110 1011 : mod qwordreg r/m : imm8
qwordregister <- memory64 with immediate32	0100 1RXB 0110 1001 : mod qwordreg r/m : imm32
IN – Input From Port	
fixed port	1110 010w : port number
variable port	1110 110w
INC – Increment by 1	
reg	0100 000B 1111 111w : 11 000 reg
qwordreg	0100 100B 1111 1111 : 11 000 qwordreg
memory	0100 00XB 1111 111w : mod 000 r/m
memory64	0100 10XB 1111 1111 : mod 000 r/m
INS – Input from DX Port	0110 110w
INT n – Interrupt Type n	1100 1101 : type
INT – Single-Step Interrupt 3	1100 1100
INTO – Interrupt 4 on Overflow	1100 1110
INVD – Invalidate Cache	0000 1111 : 0000 1000
INVLPG – Invalidate TLB Entry	0000 1111 : 0000 0001 : mod 111 r/m
INVPCID – Invalidate Process-Context Identifier	0110 0110:0000 1111:0011 1000:1000 0010: mod reg r/m
IRETO – Interrupt Return	1100 1111
***Jcc* – Jump if Condition is Met**	
8-bit displacement	0111 tttn : 8-bit displacement
displacements (excluding 16-bit relative offsets)	0000 1111 : 1000 tttn : displacement32
JCXZ/JECXZ – Jump on CX/ECX Zero	
Address-size prefix differentiates JCXZ and JECXZ	1110 0011 : 8-bit displacement
JMP – Unconditional Jump (to same segment)	
short	1110 1011 : 8-bit displacement

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
direct	1110 1001 : displacement32
register indirect	0100 W00B^{W} : 1111 1111 : 11 100 reg
memory indirect	0100 W0XB^{W} : 1111 1111 : mod 100 r/m

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
JMP – Unconditional Jump (to other segment)	
indirect intersegment	0100 00XB : 1111 1111 : mod 101 r/m
64-bit indirect intersegment	0100 10XB : 1111 1111 : mod 101 r/m
LAR – Load Access Rights Byte	
from register	0100 0R0B : 0000 1111 : 0000 0010 : 11 reg1 reg2
from dwordregister to qwordregister, masked by 00FxFF00H	0100 WR0B : 0000 1111 : 0000 0010 : 11 qwordreg1 dwordreg2
from memory	0100 0RXB : 0000 1111 : 0000 0010 : mod reg r/m
from memory32 to qwordregister, masked by 00FxFF00H	0100 WRXB 0000 1111 : 0000 0010 : mod r/m
LEA – Load Effective Address	
in wordregister/dwordregister	0100 0RXB : 1000 1101 : mod^A reg r/m
in qwordregister	0100 1RXB : 1000 1101 : mod^A qwordreg r/m
LEAVE – High Level Procedure Exit	1100 1001
LFS – Load Pointer to FS	
FS:r16/r32 with far pointer from memory	0100 0RXB : 0000 1111 : 1011 0100 : mod^A reg r/m
FS:r64 with far pointer from memory	0100 1RXB : 0000 1111 : 1011 0100 : mod^A qwordreg r/m
LGDT – Load Global Descriptor Table Register	0100 10XB : 0000 1111 : 0000 0001 : mod^A 010 r/m
LGS – Load Pointer to GS	
GS:r16/r32 with far pointer from memory	0100 0RXB : 0000 1111 : 1011 0101 : mod^A reg r/m
GS:r64 with far pointer from memory	0100 1RXB : 0000 1111 : 1011 0101 : mod^A qwordreg r/m
LIDT – Load Interrupt Descriptor Table Register	0100 10XB : 0000 1111 : 0000 0001 : mod^A 011 r/m
LLDT – Load Local Descriptor Table Register	
LDTR from register	0100 000B : 0000 1111 : 0000 0000 : 11 010 reg
LDTR from memory	0100 00XB :0000 1111 : 0000 0000 : mod 010 r/m
LMSW – Load Machine Status Word	
from register	0100 000B : 0000 1111 : 0000 0001 : 11 110 reg
from memory	0100 00XB :0000 1111 : 0000 0001 : mod 110 r/m
LOCK – Assert LOCK# Signal Prefix	1111 0000
LODS/LODSB/LODSW/LODSD/LODSQ – Load String Operand	
at DS:(E)SI to AL/EAX/EAX	1010 110w
at (R)SI to RAX	0100 1000 1010 1101
LOOP – Loop Count	
if count ≠ 0, 8-bit displacement	1110 0010
if count ≠ 0, RIP + 8-bit displacement sign-extended to 64-bits	0100 1000 1110 0010
LOOPE – Loop Count while Zero/Equal	
if count ≠ 0 & ZF =1, 8-bit displacement	1110 0001
if count ≠ 0 & ZF = 1, RIP + 8-bit displacement sign-extended to 64-bits	0100 1000 1110 0001

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
LOOPNE/LOOPNZ – Loop Count while not Zero/Equal	
if count ≠ 0 & ZF = 0, 8-bit displacement	1110 0000
if count ≠ 0 & ZF = 0, RIP + 8-bit displacement sign-extended to 64-bits	0100 1000 1110 0000
LSL – Load Segment Limit	
from register	0000 1111 : 0000 0011 : 11 reg1 reg2
from qwordregister	0100 1R00 0000 1111 : 0000 0011 : 11 qwordreg1 reg2
from memory16	0000 1111 : 0000 0011 : mod reg r/m
from memory64	0100 1RXB 0000 1111 : 0000 0011 : mod qwordreg r/m
LSS – Load Pointer to SS	
SS:r16/r32 with far pointer from memory	0100 0RXB : 0000 1111 : 1011 0010 : mod^A reg r/m
SS:r64 with far pointer from memory	0100 1WXB : 0000 1111 : 1011 0010 : mod^A qwordreg r/m
LTR – Load Task Register	
from register	0100 0R00 : 0000 1111 : 0000 0000 : 11 011 reg
from memory	0100 00XB : 0000 1111 : 0000 0000 : mod 011 r/m
MOV – Move Data	
register1 to register2	0100 0R0B : 1000 100w : 11 reg1 reg2
qwordregister1 to qwordregister2	0100 1R0B 1000 1001 : 11 qwordeg1 qwordreg2
register2 to register1	0100 0R0B : 1000 101w : 11 reg1 reg2
qwordregister2 to qwordregister1	0100 1R0B 1000 1011 : 11 qwordreg1 qwordreg2
memory to reg	0100 0RXB : 1000 101w : mod reg r/m
memory64 to qwordregister	0100 1RXB 1000 1011 : mod qwordreg r/m
reg to memory	0100 0RXB : 1000 100w : mod reg r/m
qwordregister to memory64	0100 1RXB 1000 1001 : mod qwordreg r/m
immediate to register	0100 000B : 1100 011w : 11 000 reg : imm
immediate32 to qwordregister (zero extend)	0100 100B 1100 0111 : 11 000 qwordreg : imm32
immediate to register (alternate encoding)	0100 000B : 1011 w reg : imm
immediate64 to qwordregister (alternate encoding)	0100 100B 1011 1000 reg : imm64
immediate to memory	0100 00XB : 1100 011w : mod 000 r/m : imm
immediate32 to memory64 (zero extend)	0100 10XB 1100 0111 : mod 000 r/m : imm32
memory to AL, AX, or EAX	0100 0000 : 1010 000w : displacement
memory64 to RAX	0100 1000 1010 0001 : displacement64
AL, AX, or EAX to memory	0100 0000 : 1010 001w : displacement
RAX to memory64	0100 1000 1010 0011 : displacement64
MOV – Move to/from Control Registers	
CR0-CR4 from register	0100 0R0B : 0000 1111 : 0010 0010 : 11 eee reg (eee = CR#)
CRx from qwordregister	0100 1R0B : 0000 1111 : 0010 0010 : 11 eee qwordreg (Reee = CR#)
register from CR0-CR4	0100 0R0B : 0000 1111 : 0010 0000 : 11 eee reg (eee = CR#)

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
qwordregister from CRx	0100 1R0B 0000 1111 : 0010 0000 : 11 eee qwordreg (Reee = CR#)
MOV – Move to/from Debug Registers	
DR0-DR7 from register	0000 1111 : 0010 0011 : 11 eee reg (eee = DR#)
DR0-DR7 from quadregister	0100 100B 0000 1111 : 0010 0011 : 11 eee reg (eee = DR#)
register from DR0-DR7	0000 1111 : 0010 0001 : 11 eee reg (eee = DR#)
quadregister from DR0-DR7	0100 100B 0000 1111 : 0010 0001 : 11 eee quadreg (eee = DR#)
MOV – Move to/from Segment Registers	
register to segment register	0100 W00B^{W} : 1000 1110 : 11 sreg reg
register to SS	0100 000B : 1000 1110 : 11 sreg reg
memory to segment register	0100 00XB : 1000 1110 : mod sreg r/m
memory64 to segment register (lower 16 bits)	0100 10XB 1000 1110 : mod sreg r/m
memory to SS	0100 00XB : 1000 1110 : mod sreg r/m
segment register to register	0100 000B : 1000 1100 : 11 sreg reg
segment register to qwordregister (zero extended)	0100 100B 1000 1100 : 11 sreg qwordreg
segment register to memory	0100 00XB : 1000 1100 : mod sreg r/m
segment register to memory64 (zero extended)	0100 10XB 1000 1100 : mod sreg3 r/m
MOVBE – Move data after swapping bytes	
memory to register	0100 0RXB : 0000 1111 : 0011 1000:1111 0000 : mod reg r/m
memory64 to qwordregister	0100 1RXB : 0000 1111 : 0011 1000:1111 0000 : mod reg r/m
register to memory	0100 0RXB :0000 1111 : 0011 1000:1111 0001 : mod reg r/m
qwordregister to memory64	0100 1RXB :0000 1111 : 0011 1000:1111 0001 : mod reg r/m
MOVS/MOVSB/MOVSW/MOVSD/MOVSQ – Move Data from String to String	
Move data from string to string	1010 010w
Move data from string to string (qword)	0100 1000 1010 0101
MOVSX/MOVSXD – Move with Sign-Extend	
register2 to register1	0100 0R0B : 0000 1111 : 1011 111w : 11 reg1 reg2
byteregister2 to qwordregister1 (sign-extend)	0100 1R0B 0000 1111 : 1011 1110 : 11 quadreg1 bytereg2
wordregister2 to qwordregister1	0100 1R0B 0000 1111 : 1011 1111 : 11 quadreg1 wordreg2
dwordregister2 to qwordregister1	0100 1R0B 0110 0011 : 11 quadreg1 dwordreg2
memory to register	0100 0RXB : 0000 1111 : 1011 111w : mod reg r/m
memory8 to qwordregister (sign-extend)	0100 1RXB 0000 1111 : 1011 1110 : mod qwordreg r/m
memory16 to qwordregister	0100 1RXB 0000 1111 : 1011 1111 : mod qwordreg r/m
memory32 to qwordregister	0100 1RXB 0110 0011 : mod qwordreg r/m
MOVZX – Move with Zero-Extend	
register2 to register1	0100 0R0B : 0000 1111 : 1011 011w : 11 reg1 reg2
dwordregister2 to qwordregister1	0100 1R0B 0000 1111 : 1011 0111 : 11 qwordreg1 dwordreg2

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
memory to register	0100 0RXB : 0000 1111 : 1011 011w : mod reg r/m
memory32 to qwordregister	0100 1RXB 0000 1111 : 1011 0111 : mod qwordreg r/m
MUL – Unsigned Multiply	
AL, AX, or EAX with register	0100 000B : 1111 011w : 11 100 reg
RAX with qwordregister (to RDX:RAX)	0100 100B 1111 0111 : 11 100 qwordreg
AL, AX, or EAX with memory	0100 00XB 1111 011w : mod 100 r/m
RAX with memory64 (to RDX:RAX)	0100 10XB 1111 0111 : mod 100 r/m
NEG – Two's Complement Negation	
register	0100 000B : 1111 011w : 11 011 reg
qwordregister	0100 100B 1111 0111 : 11 011 qwordreg
memory	0100 00XB : 1111 011w : mod 011 r/m
memory64	0100 10XB 1111 0111 : mod 011 r/m
NOP – No Operation	1001 0000
NOT – One's Complement Negation	
register	0100 000B : 1111 011w : 11 010 reg
qwordregister	0100 000B 1111 0111 : 11 010 qwordreg
memory	0100 00XB : 1111 011w : mod 010 r/m
memory64	0100 1RXB 1111 0111 : mod 010 r/m
OR – Logical Inclusive OR	
register1 to register2	0000 100w : 11 reg1 reg2
byteregister1 to byteregister2	0100 0R0B 0000 1000 : 11 bytereg1 bytereg2
qwordregister1 to qwordregister2	0100 1R0B 0000 1001 : 11 qwordreg1 qwordreg2
register2 to register1	0000 101w : 11 reg1 reg2
byteregister2 to byteregister1	0100 0R0B 0000 1010 : 11 bytereg1 bytereg2
qwordregister2 to qwordregister1	0100 0R0B 0000 1011 : 11 qwordreg1 qwordreg2
memory to register	0000 101w : mod reg r/m
memory8 to byteregister	0100 0RXB 0000 1010 : mod bytereg r/m
memory8 to qwordregister	0100 0RXB 0000 1011 : mod qwordreg r/m
register to memory	0000 100w : mod reg r/m
byteregister to memory8	0100 0RXB 0000 1000 : mod bytereg r/m
qwordregister to memory64	0100 1RXB 0000 1001 : mod qwordreg r/m
immediate to register	1000 00sw : 11 001 reg : imm
immediate8 to byteregister	0100 000B 1000 0000 : 11 001 bytereg : imm8
immediate32 to qwordregister	0100 000B 1000 0001 : 11 001 qwordreg : imm32
immediate8 to qwordregister	0100 000B 1000 0011 : 11 001 qwordreg : imm8
immediate to AL, AX, or EAX	0000 110w : imm
immediate64 to RAX	0100 1000 0000 1101 : imm64
immediate to memory	1000 00sw : mod 001 r/m : imm

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
immediate8 to memory8	0100 00XB 1000 0000 : mod 001 r/m : imm8
immediate32 to memory64	0100 00XB 1000 0001 : mod 001 r/m : imm32
immediate8 to memory64	0100 00XB 1000 0011 : mod 001 r/m : imm8
OUT – Output to Port	
fixed port	1110 011w : port number
variable port	1110 111w
OUTS – Output to DX Port	
output to DX Port	0110 111w
POP – Pop a Value from the Stack	
wordregister	0101 0101 : 0100 000B : 1000 1111 : 11 000 reg16
qwordregister	0100 W00B^S : 1000 1111 : 11 000 reg64
wordregister (alternate encoding)	0101 0101 : 0100 000B : 0101 1 reg16
qwordregister (alternate encoding)	0100 W00B : 0101 1 reg64
memory64	0100 W0XB^S : 1000 1111 : mod 000 r/m
memory16	0101 0101 : 0100 00XB 1000 1111 : mod 000 r/m
POP – Pop a Segment Register from the Stack (Note: CS cannot be sreg2 in this usage.)	
segment register FS, GS	0000 1111: 10 sreg3 001
POPF/POPFQ – Pop Stack into FLAGS/RFLAGS Register	
pop stack to FLAGS register	0101 0101 : 1001 1101
pop Stack to RFLAGS register	0100 1000 1001 1101
PUSH – Push Operand onto the Stack	
wordregister	0101 0101 : 0100 000B : 1111 1111 : 11 110 reg16
qwordregister	0100 W00B^S : 1111 1111 : 11 110 reg64
wordregister (alternate encoding)	0101 0101 : 0100 000B : 0101 0 reg16
qwordregister (alternate encoding)	0100 W00B^S : 0101 0 reg64
memory16	0101 0101 : 0100 000B : 1111 1111 : mod 110 r/m
memory64	0100 W00B^S : 1111 1111 : mod 110 r/m
immediate8	0110 1010 : imm8
immediate16	0101 0101 : 0110 1000 : imm16
immediate64	0110 1000 : imm64
PUSH – Push Segment Register onto the Stack	
segment register FS,GS	0000 1111: 10 sreg3 000
PUSHF/PUSHFD – Push Flags Register onto the Stack	1001 1100
RCL – Rotate thru Carry Left	
register by 1	0100 000B : 1101 000w : 11 010 reg
qwordregister by 1	0100 100B 1101 0001 : 11 010 qwordreg
memory by 1	0100 00XB : 1101 000w : mod 010 r/m
memory64 by 1	0100 10XB 1101 0001 : mod 010 r/m

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
register by CL	0100 000B : 1101 001w : 11 010 reg
qwordregister by CL	0100 100B 1101 0011 : 11 010 qwordreg
memory by CL	0100 00XB : 1101 001w : mod 010 r/m
memory64 by CL	0100 10XB 1101 0011 : mod 010 r/m
register by immediate count	0100 000B : 1100 000w : 11 010 reg : imm
qwordregister by immediate count	0100 100B 1100 0001 : 11 010 qwordreg : imm8
memory by immediate count	0100 00XB : 1100 000w : mod 010 r/m : imm
memory64 by immediate count	0100 10XB 1100 0001 : mod 010 r/m : imm8
RCR – Rotate thru Carry Right	
register by 1	0100 000B : 1101 000w : 11 011 reg
qwordregister by 1	0100 100B 1101 0001 : 11 011 qwordreg
memory by 1	0100 00XB : 1101 000w : mod 011 r/m
memory64 by 1	0100 10XB 1101 0001 : mod 011 r/m
register by CL	0100 000B : 1101 001w : 11 011 reg
qwordregister by CL	0100 000B 1101 0010 : 11 011 qwordreg
memory by CL	0100 00XB : 1101 001w : mod 011 r/m
memory64 by CL	0100 10XB 1101 0011 : mod 011 r/m
register by immediate count	0100 000B : 1100 000w : 11 011 reg : imm8
qwordregister by immediate count	0100 100B 1100 0001 : 11 011 qwordreg : imm8
memory by immediate count	0100 00XB : 1100 000w : mod 011 r/m : imm8
memory64 by immediate count	0100 10XB 1100 0001 : mod 011 r/m : imm8
RDMSR – Read from Model-Specific Register	
load ECX-specified register into EDX:EAX	0000 1111 : 0011 0010
RDPMC – Read Performance Monitoring Counters	
load ECX-specified performance counter into EDX:EAX	0000 1111 : 0011 0011
RDTSC – Read Time-Stamp Counter	
read time-stamp counter into EDX:EAX	0000 1111 : 0011 0001
RDTSCP – Read Time-Stamp Counter and Processor ID	0000 1111 : 0000 0001: 1111 1001
REP INS – Input String	
REP LODS – Load String	
REP MOVS – Move String	
REP OUTS – Output String	
REP STOS – Store String	
REPE CMPS – Compare String	
REPE SCAS – Scan String	
REPNE CMPS – Compare String	
REPNE SCAS – Scan String	
RET – Return from Procedure (to same segment)	

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
no argument	1100 0011
adding immediate to SP	1100 0010 : 16-bit displacement
RET – Return from Procedure (to other segment)	
intersegment	1100 1011
adding immediate to SP	1100 1010 : 16-bit displacement
ROL – Rotate Left	
register by 1	0100 000B 1101 000w : 11 000 reg
byteregister by 1	0100 000B 1101 0000 : 11 000 bytereg
qwordregister by 1	0100 100B 1101 0001 : 11 000 qwordreg
memory by 1	0100 00XB 1101 000w : mod 000 r/m
memory8 by 1	0100 00XB 1101 0000 : mod 000 r/m
memory64 by 1	0100 10XB 1101 0001 : mod 000 r/m
register by CL	0100 000B 1101 001w : 11 000 reg
byteregister by CL	0100 000B 1101 0010 : 11 000 bytereg
qwordregister by CL	0100 100B 1101 0011 : 11 000 qwordreg
memory by CL	0100 00XB 1101 001w : mod 000 r/m
memory8 by CL	0100 00XB 1101 0010 : mod 000 r/m
memory64 by CL	0100 10XB 1101 0011 : mod 000 r/m
register by immediate count	1100 000w : 11 000 reg : imm8
byteregister by immediate count	0100 000B 1100 0000 : 11 000 bytereg : imm8
qwordregister by immediate count	0100 100B 1100 0001 : 11 000 bytereg : imm8
memory by immediate count	1100 000w : mod 000 r/m : imm8
memory8 by immediate count	0100 00XB 1100 0000 : mod 000 r/m : imm8
memory64 by immediate count	0100 10XB 1100 0001 : mod 000 r/m : imm8
ROR – Rotate Right	
register by 1	0100 000B 1101 000w : 11 001 reg
byteregister by 1	0100 000B 1101 0000 : 11 001 bytereg
qwordregister by 1	0100 100B 1101 0001 : 11 001 qwordreg
memory by 1	0100 00XB 1101 000w : mod 001 r/m
memory8 by 1	0100 00XB 1101 0000 : mod 001 r/m
memory64 by 1	0100 10XB 1101 0001 : mod 001 r/m
register by CL	0100 000B 1101 001w : 11 001 reg
byteregister by CL	0100 000B 1101 0010 : 11 001 bytereg
qwordregister by CL	0100 100B 1101 0011 : 11 001 qwordreg
memory by CL	0100 00XB 1101 001w : mod 001 r/m
memory8 by CL	0100 00XB 1101 0010 : mod 001 r/m
memory64 by CL	0100 10XB 1101 0011 : mod 001 r/m
register by immediate count	0100 000B 1100 000w : 11 001 reg : imm8

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
byteregister by immediate count	0100 000B 1100 0000 : 11 001 reg : imm8
qwordregister by immediate count	0100 100B 1100 0001 : 11 001 qwordreg : imm8
memory by immediate count	0100 00XB 1100 000w : mod 001 r/m : imm8
memory8 by immediate count	0100 00XB 1100 0000 : mod 001 r/m : imm8
memory64 by immediate count	0100 10XB 1100 0001 : mod 001 r/m : imm8
RSM – Resume from System Management Mode	0000 1111 : 1010 1010
SAL – Shift Arithmetic Left	same instruction as SHL
SAR – Shift Arithmetic Right	
register by 1	0100 000B 1101 000w : 11 111 reg
byteregister by 1	0100 000B 1101 0000 : 11 111 bytereg
qwordregister by 1	0100 100B 1101 0001 : 11 111 qwordreg
memory by 1	0100 00XB 1101 000w : mod 111 r/m
memory8 by 1	0100 00XB 1101 0000 : mod 111 r/m
memory64 by 1	0100 10XB 1101 0001 : mod 111 r/m
register by CL	0100 000B 1101 001w : 11 111 reg
byteregister by CL	0100 000B 1101 0010 : 11 111 bytereg
qwordregister by CL	0100 100B 1101 0011 : 11 111 qwordreg
memory by CL	0100 00XB 1101 001w : mod 111 r/m
memory8 by CL	0100 00XB 1101 0010 : mod 111 r/m
memory64 by CL	0100 10XB 1101 0011 : mod 111 r/m
register by immediate count	0100 000B 1100 000w : 11 111 reg : imm8
byteregister by immediate count	0100 000B 1100 0000 : 11 111 bytereg : imm8
qwordregister by immediate count	0100 100B 1100 0001 : 11 111 qwordreg : imm8
memory by immediate count	0100 00XB 1100 000w : mod 111 r/m : imm8
memory8 by immediate count	0100 00XB 1100 0000 : mod 111 r/m : imm8
memory64 by immediate count	0100 10XB 1100 0001 : mod 111 r/m : imm8
SBB – Integer Subtraction with Borrow	
register1 to register2	0100 0R0B 0001 100w : 11 reg1 reg2
byteregister1 to byteregister2	0100 0R0B 0001 1000 : 11 bytereg1 bytereg2
quadregister1 to quadregister2	0100 1R0B 0001 1001 : 11 quadreg1 quadreg2
register2 to register1	0100 0R0B 0001 101w : 11 reg1 reg2
byteregister2 to byteregister1	0100 0R0B 0001 1010 : 11 reg1 bytereg2
byteregister2 to byteregister1	0100 1R0B 0001 1011 : 11 reg1 bytereg2
memory to register	0100 0RXB 0001 101w : mod reg r/m
memory8 to byteregister	0100 0RXB 0001 1010 : mod bytereg r/m
memory64 to byteregister	0100 1RXB 0001 1011 : mod quadreg r/m
register to memory	0100 0RXB 0001 100w : mod reg r/m
byteregister to memory8	0100 0RXB 0001 1000 : mod reg r/m

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
quadregister to memory64	0100 1RXB 0001 1001 : mod reg r/m
immediate to register	0100 000B 1000 00sw : 11 011 reg : imm
immediate8 to byteregister	0100 000B 1000 0000 : 11 011 bytereg : imm8
immediate32 to qwordregister	0100 100B 1000 0001 : 11 011 qwordreg : imm32
immediate8 to qwordregister	0100 100B 1000 0011 : 11 011 qwordreg : imm8
immediate to AL, AX, or EAX	0100 000B 0001 110w : imm
immediate32 to RAL	0100 1000 0001 1101 : imm32
immediate to memory	0100 00XB 1000 00sw : mod 011 r/m : imm
immediate8 to memory8	0100 00XB 1000 0000 : mod 011 r/m : imm8
immediate32 to memory64	0100 10XB 1000 0001 : mod 011 r/m : imm32
immediate8 to memory64	0100 10XB 1000 0011 : mod 011 r/m : imm8
SCAS/SCASB/SCASW/SCASD – Scan String	
scan string	1010 111w
scan string (compare AL with byte at RDI)	0100 1000 1010 1110
scan string (compare RAX with qword at RDI)	0100 1000 1010 1111
SETcc – Byte Set on Condition	
register	0100 000B 0000 1111 : 1001 tttn : 11 000 reg
register	0100 0000 0000 1111 : 1001 tttn : 11 000 reg
memory	0100 00XB 0000 1111 : 1001 tttn : mod 000 r/m
memory	0100 0000 0000 1111 : 1001 tttn : mod 000 r/m
SGDT – Store Global Descriptor Table Register	0000 1111 : 0000 0001 : mod^{A} 000 r/m
SHL – Shift Left	
register by 1	0100 000B 1101 000w : 11 100 reg
byteregister by 1	0100 000B 1101 0000 : 11 100 bytereg
qwordregister by 1	0100 100B 1101 0001 : 11 100 qwordreg
memory by 1	0100 00XB 1101 000w : mod 100 r/m
memory8 by 1	0100 00XB 1101 0000 : mod 100 r/m
memory64 by 1	0100 10XB 1101 0001 : mod 100 r/m
register by CL	0100 000B 1101 001w : 11 100 reg
byteregister by CL	0100 000B 1101 0010 : 11 100 bytereg
qwordregister by CL	0100 100B 1101 0011 : 11 100 qwordreg
memory by CL	0100 00XB 1101 001w : mod 100 r/m
memory8 by CL	0100 00XB 1101 0010 : mod 100 r/m
memory64 by CL	0100 10XB 1101 0011 : mod 100 r/m
register by immediate count	0100 000B 1100 000w : 11 100 reg : imm8
byteregister by immediate count	0100 000B 1100 0000 : 11 100 bytereg : imm8
quadregister by immediate count	0100 100B 1100 0001 : 11 100 quadreg : imm8
memory by immediate count	0100 00XB 1100 000w : mod 100 r/m : imm8

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
memory8 by immediate count	0100 00XB 1100 0000 : mod 100 r/m : imm8
memory64 by immediate count	0100 10XB 1100 0001 : mod 100 r/m : imm8
SHLD – Double Precision Shift Left	
register by immediate count	0100 0R0B 0000 1111 : 1010 0100 : 11 reg2 reg1 : imm8
qwordregister by immediate8	0100 1R0B 0000 1111 : 1010 0100 : 11 qworddreg2 qwordreg1 : imm8
memory by immediate count	0100 0RXB 0000 1111 : 1010 0100 : mod reg r/m : imm8
memory64 by immediate8	0100 1RXB 0000 1111 : 1010 0100 : mod qwordreg r/m : imm8
register by CL	0100 0R0B 0000 1111 : 1010 0101 : 11 reg2 reg1
quadregister by CL	0100 1R0B 0000 1111 : 1010 0101 : 11 quadreg2 quadreg1
memory by CL	0100 00XB 0000 1111 : 1010 0101 : mod reg r/m
memory64 by CL	0100 1RXB 0000 1111 : 1010 0101 : mod quadreg r/m
SHR – Shift Right	
register by 1	0100 000B 1101 000w : 11 101 reg
byteregister by 1	0100 000B 1101 0000 : 11 101 bytereg
qwordregister by 1	0100 100B 1101 0001 : 11 101 qwordreg
memory by 1	0100 00XB 1101 000w : mod 101 r/m
memory8 by 1	0100 00XB 1101 0000 : mod 101 r/m
memory64 by 1	0100 10XB 1101 0001 : mod 101 r/m
register by CL	0100 000B 1101 001w : 11 101 reg
byteregister by CL	0100 000B 1101 0010 : 11 101 bytereg
qwordregister by CL	0100 100B 1101 0011 : 11 101 qwordreg
memory by CL	0100 00XB 1101 001w : mod 101 r/m
memory8 by CL	0100 00XB 1101 0010 : mod 101 r/m
memory64 by CL	0100 10XB 1101 0011 : mod 101 r/m
register by immediate count	0100 000B 1100 000w : 11 101 reg : imm8
byteregister by immediate count	0100 000B 1100 0000 : 11 101 reg : imm8
qwordregister by immediate count	0100 100B 1100 0001 : 11 101 reg : imm8
memory by immediate count	0100 00XB 1100 000w : mod 101 r/m : imm8
memory8 by immediate count	0100 00XB 1100 0000 : mod 101 r/m : imm8
memory64 by immediate count	0100 10XB 1100 0001 : mod 101 r/m : imm8
SHRD – Double Precision Shift Right	
register by immediate count	0100 0R0B 0000 1111 : 1010 1100 : 11 reg2 reg1 : imm8
qwordregister by immediate8	0100 1R0B 0000 1111 : 1010 1100 : 11 qwordreg2 qwordreg1 : imm8
memory by immediate count	0100 00XB 0000 1111 : 1010 1100 : mod reg r/m : imm8
memory64 by immediate8	0100 1RXB 0000 1111 : 1010 1100 : mod qwordreg r/m : imm8
register by CL	0100 000B 0000 1111 : 1010 1101 : 11 reg2 reg1

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
qwordregister by CL	0100 1R0B 0000 1111 : 1010 1101 : 11 qwordreg2 qwordreg1
memory by CL	0000 1111 : 1010 1101 : mod reg r/m
memory64 by CL	0100 1RXB 0000 1111 : 1010 1101 : mod qwordreg r/m
SIDT – Store Interrupt Descriptor Table Register	0000 1111 : 0000 0001 : mod^{A} 001 r/m
SLDT – Store Local Descriptor Table Register	
to register	0100 000B 0000 1111 : 0000 0000 : 11 000 reg
to memory	0100 00XB 0000 1111 : 0000 0000 : mod 000 r/m
SMSW – Store Machine Status Word	
to register	0100 000B 0000 1111 : 0000 0001 : 11 100 reg
to memory	0100 00XB 0000 1111 : 0000 0001 : mod 100 r/m
STC – Set Carry Flag	1111 1001
STD – Set Direction Flag	1111 1101
STI – Set Interrupt Flag	1111 1011
STOS/STOSB/STOSW/STOSD/STOSQ – Store String Data	
store string data	1010 101w
store string data (RAX at address RDI)	0100 1000 1010 1011
STR – Store Task Register	
to register	0100 000B 0000 1111 : 0000 0000 : 11 001 reg
to memory	0100 00XB 0000 1111 : 0000 0000 : mod 001 r/m
SUB – Integer Subtraction	
register1 from register2	0100 0R0B 0010 100w : 11 reg1 reg2
byteregister1 from byteregister2	0100 0R0B 0010 1000 : 11 bytereg1 bytereg2
qwordregister1 from qwordregister2	0100 1R0B 0010 1000 : 11 qwordreg1 qwordreg2
register2 from register1	0100 0R0B 0010 101w : 11 reg1 reg2
byteregister2 from byteregister1	0100 0R0B 0010 1010 : 11 bytereg1 bytereg2
qwordregister2 from qwordregister1	0100 1R0B 0010 1011 : 11 qwordreg1 qwordreg2
memory from register	0100 00XB 0010 101w : mod reg r/m
memory8 from byteregister	0100 0RXB 0010 1010 : mod bytereg r/m
memory64 from qwordregister	0100 1RXB 0010 1011 : mod qwordreg r/m
register from memory	0100 0RXB 0010 100w : mod reg r/m
byteregister from memory8	0100 0RXB 0010 1000 : mod bytereg r/m
qwordregister from memory8	0100 1RXB 0010 1000 : mod qwordreg r/m
immediate from register	0100 000B 1000 00sw : 11 101 reg : imm
immediate8 from byteregister	0100 000B 1000 0000 : 11 101 bytereg : imm8
immediate32 from qwordregister	0100 100B 1000 0001 : 11 101 qwordreg : imm32
immediate8 from qwordregister	0100 100B 1000 0011 : 11 101 qwordreg : imm8
immediate from AL, AX, or EAX	0100 000B 0010 110w : imm
immediate32 from RAX	0100 1000 0010 1101 : imm32

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
immediate from memory	0100 00XB 1000 00sw : mod 101 r/m : imm
immediate8 from memory8	0100 00XB 1000 0000 : mod 101 r/m : imm8
immediate32 from memory64	0100 10XB 1000 0001 : mod 101 r/m : imm32
immediate8 from memory64	0100 10XB 1000 0011 : mod 101 r/m : imm8
SWAPGS – Swap GS Base Register	
Exchanges the current GS base register value for value in MSR C0000102H	0000 1111 0000 0001 1111 1000
SYSCALL – Fast System Call	
fast call to privilege level 0 system procedures	0000 1111 0000 0101
SYSRET – Return From Fast System Call	
return from fast system call	0000 1111 0000 0111
TEST – Logical Compare	
register1 and register2	0100 0R0B 1000 010w : 11 reg1 reg2
byteregister1 and byteregister2	0100 0R0B 1000 0100 : 11 bytereg1 bytereg2
qwordregister1 and qwordregister2	0100 1R0B 1000 0101 : 11 qwordreg1 qwordreg2
memory and register	0100 0R0B 1000 010w : mod reg r/m
memory8 and byteregister	0100 0RXB 1000 0100 : mod bytereg r/m
memory64 and qwordregister	0100 1RXB 1000 0101 : mod qwordreg r/m
immediate and register	0100 000B 1111 011w : 11 000 reg : imm
immediate8 and byteregister	0100 000B 1111 0110 : 11 000 bytereg : imm8
immediate32 and qwordregister	0100 100B 1111 0111 : 11 000 bytereg : imm8
immediate and AL, AX, or EAX	0100 000B 1010 100w : imm
immediate32 and RAX	0100 1000 1010 1001 : imm32
immediate and memory	0100 00XB 1111 011w : mod 000 r/m : imm
immediate8 and memory8	0100 1000 1111 0110 : mod 000 r/m : imm8
immediate32 and memory64	0100 1000 1111 0111 : mod 000 r/m : imm32
UD2 – Undefined instruction	0000 FFFF : 0000 1011
VERR – Verify a Segment for Reading	
register	0100 000B 0000 1111 : 0000 0000 : 11 100 reg
memory	0100 00XB 0000 1111 : 0000 0000 : mod 100 r/m
VERW – Verify a Segment for Writing	
register	0100 000B 0000 1111 : 0000 0000 : 11 101 reg
memory	0100 00XB 0000 1111 : 0000 0000 : mod 101 r/m
WAIT – Wait	1001 1011
WBINVD – Writeback and Invalidate Data Cache	0000 1111 : 0000 1001
WRMSR – Write to Model-Specific Register	
write EDX:EAX to ECX specified MSR	0000 1111 : 0011 0000
write RDX[31:0]:RAX[31:0] to RCX specified MSR	0100 1000 0000 1111 : 0011 0000

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
XADD – Exchange and Add	
register1, register2	0100 0R0B 0000 1111 : 1100 000w : 11 reg2 reg1
byteregister1, byteregister2	0100 0R0B 0000 1111 : 1100 0000 : 11 bytereg2 bytereg1
qwordregister1, qwordregister2	0100 0R0B 0000 1111 : 1100 0001 : 11 qwordreg2 qwordreg1
memory, register	0100 0RXB 0000 1111 : 1100 000w : mod reg r/m
memory8, bytereg	0100 1RXB 0000 1111 : 1100 0000 : mod bytereg r/m
memory64, qwordreg	0100 1RXB 0000 1111 : 1100 0001 : mod qwordreg r/m
XCHG – Exchange Register/Memory with Register	
register1 with register2	1000 011w : 11 reg1 reg2
AX or EAX with register	1001 0 reg
memory with register	1000 011w : mod reg r/m
XLAT/XLATB – Table Look-up Translation	
AL to byte DS:[(E)BX + unsigned AL]	1101 0111
AL to byte DS:[RBX + unsigned AL]	0100 1000 1101 0111
XOR – Logical Exclusive OR	
register1 to register2	0100 0RXB 0011 000w : 11 reg1 reg2
byteregister1 to byteregister2	0100 0R0B 0011 0000 : 11 bytereg1 bytereg2
qwordregister1 to qwordregister2	0100 1R0B 0011 0001 : 11 qwordreg1 qwordreg2
register2 to register1	0100 0R0B 0011 001w : 11 reg1 reg2
byteregister2 to byteregister1	0100 0R0B 0011 0010 : 11 bytereg1 bytereg2
qwordregister2 to qwordregister1	0100 1R0B 0011 0011 : 11 qwordreg1 qwordreg2
memory to register	0100 0RXB 0011 001w : mod reg r/m
memory8 to byteregister	0100 0RXB 0011 0010 : mod bytereg r/m
memory64 to qwordregister	0100 1RXB 0011 0011 : mod qwordreg r/m
register to memory	0100 0RXB 0011 000w : mod reg r/m
byteregister to memory8	0100 0RXB 0011 0000 : mod bytereg r/m
qwordregister to memory8	0100 1RXB 0011 0001 : mod qwordreg r/m
immediate to register	0100 000B 1000 00sw : 11 110 reg : imm
immediate8 to byteregister	0100 000B 1000 0000 : 11 110 bytereg : imm8
immediate32 to qwordregister	0100 100B 1000 0001 : 11 110 qwordreg : imm32
immediate8 to qwordregister	0100 100B 1000 0011 : 11 110 qwordreg : imm8
immediate to AL, AX, or EAX	0100 000B 0011 010w : imm
immediate to RAX	0100 1000 0011 0101 : immediate data
immediate to memory	0100 00XB 1000 00sw : mod 110 r/m : imm
immediate8 to memory8	0100 00XB 1000 0000 : mod 110 r/m : imm8
immediate32 to memory64	0100 10XB 1000 0001 : mod 110 r/m : imm32
immediate8 to memory64	0100 10XB 1000 0011 : mod 110 r/m : imm8

Table B-15. General Purpose Instruction Formats and Encodings for 64-Bit Mode (Contd.)

Instruction and Format	Encoding
Prefix Bytes	
address size	0110 0111
LOCK	1111 0000
operand size	0110 0110
CS segment override	0010 1110
DS segment override	0011 1110
ES segment override	0010 0110
FS segment override	0110 0100
GS segment override	0110 0101
SS segment override	0011 0110

B.3 PENTIUM® PROCESSOR FAMILY INSTRUCTION FORMATS AND ENCODINGS

The following table shows formats and encodings introduced by the Pentium processor family.

Table B-16. Pentium Processor Family Instruction Formats and Encodings, Non-64-Bit Modes

Instruction and Format	Encoding
CMPXCHG8B – Compare and Exchange 8 Bytes	
EDX:EAX with memory64	0000 1111 : 1100 0111 : mod 001 r/m

Table B-17. Pentium Processor Family Instruction Formats and Encodings, 64-Bit Mode

Instruction and Format	Encoding
CMPXCHG8B/CMPXCHG16B – Compare and Exchange Bytes	
EDX:EAX with memory64	0000 1111 : 1100 0111 : mod 001 r/m
RDX:RAX with memory128	0100 10XB 0000 1111 : 1100 0111 : mod 001 r/m

B.4 64-BIT MODE INSTRUCTION ENCODINGS FOR SIMD INSTRUCTION EXTENSIONS

Non-64-bit mode instruction encodings for MMX Technology, SSE, SSE2, and SSE3 are covered by applying these rules to Table B-19 through Table B-31. Table B-34 lists special encodings (instructions that do not follow the rules below).

1. The REX instruction has no effect:
    - On immediates.
    - If both operands are MMX registers.
    - On MMX registers and XMM registers.
    - If an MMX register is encoded in the reg field of the ModR/M byte.
2. If a memory operand is encoded in the r/m field of the ModR/M byte, REX.X and REX.B may be used for encoding the memory operand.

3. If a general-purpose register is encoded in the r/m field of the ModR/M byte, REX.B may be used for register encoding and REX.W may be used to encode the 64-bit operand size.
4. If an XMM register operand is encoded in the reg field of the ModR/M byte, REX.R may be used for register encoding. If an XMM register operand is encoded in the r/m field of the ModR/M byte, REX.B may be used for register encoding.

B.5 MMX INSTRUCTION FORMATS AND ENCODINGS

MMX instructions, except the EMMS instruction, use a format similar to the 2-byte Intel Architecture integer format. Details of subfield encodings within these formats are presented below.

B.5.1 Granularity Field (gg)

The granularity field (gg) indicates the size of the packed operands that the instruction is operating on. When this field is used, it is located in bits 1 and 0 of the second opcode byte. Table B-18 shows the encoding of the gg field.

Table B-18. Encoding of Granularity of Data Field (gg)

gg	Granularity of Data
00	Packed Bytes
01	Packed Words
10	Packed Doublewords
11	Quadword

B.5.2 MMX Technology and General-Purpose Register Fields (mmxreg and reg)

When MMX technology registers (mmxreg) are used as operands, they are encoded in the ModR/M byte in the reg field (bits 5, 4, and 3) and/or the R/M field (bits 2, 1, and 0).

If an MMX instruction operates on a general-purpose register (reg), the register is encoded in the R/M field of the ModR/M byte.

B.5.3 MMX Instruction Formats and Encodings Table

Table B-19 shows the formats and encodings of the integer instructions.

Table B-19. MMX Instruction Formats and Encodings

Instruction and Format	Encoding
EMMS – Empty MMX technology state	0000 1111:01110111
MOVD – Move doubleword	
reg to mmxreg	0000 1111:0110 1110: 11 mmxreg reg
reg from mmxreg	0000 1111:0111 1110: 11 mmxreg reg
mem to mmxreg	0000 1111:0110 1110: mod mmxreg r/m
mem from mmxreg	0000 1111:0111 1110: mod mmxreg r/m
MOVQ – Move quadword	
mmxreg2 to mmxreg1	0000 1111:0110 1111: 11 mmxreg1 mmxreg2
mmxreg2 from mmxreg1	0000 1111:0111 1111: 11 mmxreg1 mmxreg2
mem to mmxreg	0000 1111:0110 1111: mod mmxreg r/m

Table B-19. MMX Instruction Formats and Encodings (Contd.)

Instruction and Format	Encoding
mem from mmxreg	0000 1111:0111 1111: mod mmxreg r/m
PACKSSDW[1] – Pack dword to word data (signed with saturation)	
mmxreg2 to mmxreg1	0000 1111:0110 1011: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:0110 1011: mod mmxreg r/m
PACKSSWB[1] – Pack word to byte data (signed with saturation)	
mmxreg2 to mmxreg1	0000 1111:0110 0011: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:0110 0011: mod mmxreg r/m
PACKUSWB[1] – Pack word to byte data (unsigned with saturation)	
mmxreg2 to mmxreg1	0000 1111:0110 0111: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:0110 0111: mod mmxreg r/m
PADD – Add with wrap-around	
mmxreg2 to mmxreg1	0000 1111: 1111 11gg: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111: 1111 11gg: mod mmxreg r/m
PADDS – Add signed with saturation	
mmxreg2 to mmxreg1	0000 1111: 1110 11gg: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111: 1110 11gg: mod mmxreg r/m
PADDUS – Add unsigned with saturation	
mmxreg2 to mmxreg1	0000 1111: 1101 11gg: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111: 1101 11gg: mod mmxreg r/m
PAND – Bitwise And	
mmxreg2 to mmxreg1	0000 1111:1101 1011: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:1101 1011: mod mmxreg r/m
PANDN – Bitwise AndNot	
mmxreg2 to mmxreg1	0000 1111:1101 1111: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:1101 1111: mod mmxreg r/m
PCMPEQ – Packed compare for equality	
mmxreg1 with mmxreg2	0000 1111:0111 01gg: 11 mmxreg1 mmxreg2
mmxreg with memory	0000 1111:0111 01gg: mod mmxreg r/m
PCMPGT – Packed compare greater (signed)	
mmxreg1 with mmxreg2	0000 1111:0110 01gg: 11 mmxreg1 mmxreg2
mmxreg with memory	0000 1111:0110 01gg: mod mmxreg r/m
PMADDWD – Packed multiply add	
mmxreg2 to mmxreg1	0000 1111:1111 0101: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:1111 0101: mod mmxreg r/m
PMULHUW – Packed multiplication, store high word (unsigned)	
mmxreg2 to mmxreg1	0000 1111: 1110 0100: 11 mmxreg1 mmxreg2

Table B-19. MMX Instruction Formats and Encodings (Contd.)

Instruction and Format	Encoding
memory to mmxreg	0000 1111: 1110 0100: mod mmxreg r/m
PMULHW – Packed multiplication, store high word	
mmxreg2 to mmxreg1	0000 1111:1110 0101: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:1110 0101: mod mmxreg r/m
PMULLW – Packed multiplication, store low word	
mmxreg2 to mmxreg1	0000 1111:1101 0101: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:1101 0101: mod mmxreg r/m
POR – Bitwise Or	
mmxreg2 to mmxreg1	0000 1111:1110 1011: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:1110 1011: mod mmxreg r/m
PSLL[2] – Packed shift left logical	
mmxreg1 by mmxreg2	0000 1111:1111 00gg: 11 mmxreg1 mmxreg2
mmxreg by memory	0000 1111:1111 00gg: mod mmxreg r/m
mmxreg by immediate	0000 1111:0111 00gg: 11 110 mmxreg: imm8 data
PSRA[2] – Packed shift right arithmetic	
mmxreg1 by mmxreg2	0000 1111:1110 00gg: 11 mmxreg1 mmxreg2
mmxreg by memory	0000 1111:1110 00gg: mod mmxreg r/m
mmxreg by immediate	0000 1111:0111 00gg: 11 100 mmxreg: imm8 data
PSRL[2] – Packed shift right logical	
mmxreg1 by mmxreg2	0000 1111:1101 00gg: 11 mmxreg1 mmxreg2
mmxreg by memory	0000 1111:1101 00gg: mod mmxreg r/m
mmxreg by immediate	0000 1111:0111 00gg: 11 010 mmxreg: imm8 data
PSUB – Subtract with wrap-around	
mmxreg2 from mmxreg1	0000 1111:1111 10gg: 11 mmxreg1 mmxreg2
memory from mmxreg	0000 1111:1111 10gg: mod mmxreg r/m
PSUBS – Subtract signed with saturation	
mmxreg2 from mmxreg1	0000 1111:1110 10gg: 11 mmxreg1 mmxreg2
memory from mmxreg	0000 1111:1110 10gg: mod mmxreg r/m
PSUBUS – Subtract unsigned with saturation	
mmxreg2 from mmxreg1	0000 1111:1101 10gg: 11 mmxreg1 mmxreg2
memory from mmxreg	0000 1111:1101 10gg: mod mmxreg r/m
PUNPCKH – Unpack high data to next larger type	
mmxreg2 to mmxreg1	0000 1111:0110 10gg: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:0110 10gg: mod mmxreg r/m
PUNPCKL – Unpack low data to next larger type	
mmxreg2 to mmxreg1	0000 1111:0110 00gg: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:0110 00gg: mod mmxreg r/m

Table B-19. MMX Instruction Formats and Encodings (Contd.)

Instruction and Format	Encoding
PXOR – Bitwise Xor	
mmxreg2 to mmxreg1	0000 1111:1110 1111: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:1110 1111: mod mmxreg r/m

NOTES:

1. The pack instructions perform saturation from signed packed data of one type to signed or unsigned data of the next smaller type.
2. The format of the shift instructions has one additional format to support shifting by immediate shift-counts. The shift operations are not supported equally for all data types.

B.6 PROCESSOR EXTENDED STATE INSTRUCTION FORMATS AND ENCODINGS

Table B-20 shows the formats and encodings for several instructions that relate to processor extended state management.

Table B-20. Formats and Encodings of XSAVE/XRSTOR/XGETBV/XSETBV Instructions

Instruction and Format	Encoding
XGETBV – Get Value of Extended Control Register	0000 1111:0000 0001: 1101 0000
XRSTOR – Restore Processor Extended States[1]	0000 1111:1010 1110: mod^A 101 r/m
XSAVE – Save Processor Extended States[1]	0000 1111:1010 1110: mod^A 100 r/m
XSETBV – Set Extended Control Register	0000 1111:0000 0001: 1101 0001

NOTES:

1. For XSAVE and XRSTOR, “mod = 11” is reserved.

B.7 P6 FAMILY INSTRUCTION FORMATS AND ENCODINGS

Table B-20 shows the formats and encodings for several instructions that were introduced into the IA-32 architecture in the P6 family processors.

Table B-21. Formats and Encodings of P6 Family Instructions

Instruction and Format	Encoding
CMOVcc – Conditional Move	
register2 to register1	0000 1111: 0100 tttn : 11 reg1 reg2
memory to register	0000 1111 : 0100 tttn : mod reg r/m
FCMOVcc – Conditional Move on EFLAG Register Condition Codes	
move if below (B)	11011 010 : 11 000 ST(i)
move if equal (E)	11011 010 : 11 001 ST(i)
move if below or equal (BE)	11011 010 : 11 010 ST(i)
move if unordered (U)	11011 010 : 11 011 ST(i)
move if not below (NB)	11011 011 : 11 000 ST(i)
move if not equal (NE)	11011 011 : 11 001 ST(i)

Table B-21. Formats and Encodings of P6 Family Instructions (Contd.)

Instruction and Format	Encoding
move if not below or equal (NBE)	11011 011 : 11 010 ST(i)
move if not unordered (NU)	11011 011 : 11 011 ST(i)
FCOMI – Compare Real and Set EFLAGS	11011 011 : 11 110 ST(i)
FXRSTOR – Restore x87 FPU, MMX, SSE, and SSE2 State[1]	0000 1111:1010 1110: mod^A 001 r/m
FXSAVE – Save x87 FPU, MMX, SSE, and SSE2 State[1]	0000 1111:1010 1110: mod^A 000 r/m
SYSENTER – Fast System Call	0000 1111:0011 0100
SYSEXIT – Fast Return from Fast System Call	0000 1111:0011 0101

NOTES:

1. For FXSAVE and FXRSTOR, “mod = 11” is reserved.

B.8 SSE INSTRUCTION FORMATS AND ENCODINGS

The SSE instructions use the ModR/M format and are preceded by the 0FH prefix byte. In general, operations are not duplicated to provide two directions (that is, separate load and store variants).

The following three tables (Tables B-22, B-23, and B-24) show the formats and encodings for the SSE SIMD floating-point, SIMD integer, and cacheability and memory ordering instructions, respectively. Some SSE instructions require a mandatory prefix (66H, F2H, F3H) as part of the two-byte opcode. Mandatory prefixes are included in the tables.

Table B-22. Formats and Encodings of SSE Floating-Point Instructions

Instruction and Format	Encoding
ADDPS—Add Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 1000:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 1000: mod xmmreg r/m
ADDSS—Add Scalar Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	1111 0011:0000 1111:01011000:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:01011000: mod xmmreg r/m
ANDNPS—Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 0101:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 0101: mod xmmreg r/m
ANDPS—Bitwise Logical AND of Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 0100: mod xmmreg r/m
CMPPS—Compare Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1, imm8	0000 1111:1100 0010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0000 1111:1100 0010: mod xmmreg r/m: imm8
CMPSS—Compare Scalar Single-Precision Floating-Point Values	

Table B-22. Formats and Encodings of SSE Floating-Point Instructions (Contd.)

Instruction and Format	Encoding
xmmreg2 to xmmreg1, imm8	1111 0011:0000 1111:1100 0010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	1111 0011:0000 1111:1100 0010: mod xmmreg r/m: imm8
COMISS—Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS	
xmmreg2 to xmmreg1	0000 1111:0010 1111:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0010 1111: mod xmmreg r/m
CVTPI2PS—Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values	
mmreg to xmmreg	0000 1111:0010 1010:11 xmmreg1 mmreg1
mem to xmmreg	0000 1111:0010 1010: mod xmmreg r/m
CVTPS2PI—Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to mmreg	0000 1111:0010 1101:11 mmreg1 xmmreg1
mem to mmreg	0000 1111:0010 1101: mod mmreg r/m
CVTSI2SS—Convert Doubleword Integer to Scalar Single-Precision Floating-Point Value	
r32 to xmmreg1	1111 0011:0000 1111:00101010:11 xmmreg1 r32
mem to xmmreg	1111 0011:0000 1111:00101010: mod xmmreg r/m
CVTSS2SI—Convert Scalar Single-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	1111 0011:0000 1111:0010 1101:11 r32 xmmreg
mem to r32	1111 0011:0000 1111:0010 1101: mod r32 r/m
CVTTPS2PI—Convert with Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to mmreg	0000 1111:0010 1100:11 mmreg1 xmmreg1
mem to mmreg	0000 1111:0010 1100: mod mmreg r/m
CVTTSS2SI—Convert with Truncation Scalar Single-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	1111 0011:0000 1111:0010 1100:11 r32 xmmreg1
mem to r32	1111 0011:0000 1111:0010 1100: mod r32 r/m
DIVPS—Divide Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 1110:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 1110: mod xmmreg r/m
DIVSS—Divide Scalar Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0101 1110:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:0101 1110: mod xmmreg r/m
LDMXCSR—Load MXCSR Register State	
m32 to MXCSR	0000 1111:1010 1110:mod^{A} 010 mem
MAXPS—Return Maximum Packed Single-Precision Floating-Point Values	

Table B-22. Formats and Encodings of SSE Floating-Point Instructions (Contd.)

Instruction and Format	Encoding
xmmreg2 to xmmreg1	0000 1111:0101 1111:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 1111: mod xmmreg r/m
MAXSS—Return Maximum Scalar Double-Precision Floating-Point Value	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0101 1111:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:0101 1111: mod xmmreg r/m
MINPS—Return Minimum Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 1101:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 1101: mod xmmreg r/m
MINSS—Return Minimum Scalar Double-Precision Floating-Point Value	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0101 1101:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:0101 1101: mod xmmreg r/m
MOVAPS—Move Aligned Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0010 1000:11 xmmreg2 xmmreg1
mem to xmmreg1	0000 1111:0010 1000: mod xmmreg r/m
xmmreg1 to xmmreg2	0000 1111:0010 1001:11 xmmreg1 xmmreg2
xmmreg1 to mem	0000 1111:0010 1001: mod xmmreg r/m
MOVHLPS—Move Packed Single-Precision Floating-Point Values High to Low	
xmmreg2 to xmmreg1	0000 1111:0001 0010:11 xmmreg1 xmmreg2
MOVHPS—Move High Packed Single-Precision Floating-Point Values	
mem to xmmreg	0000 1111:0001 0110: mod xmmreg r/m
xmmreg to mem	0000 1111:0001 0111: mod xmmreg r/m
MOVLHPS—Move Packed Single-Precision Floating-Point Values Low to High	
xmmreg2 to xmmreg1	0000 1111:00010110:11 xmmreg1 xmmreg2
MOVLPS—Move Low Packed Single-Precision Floating-Point Values	
mem to xmmreg	0000 1111:0001 0010: mod xmmreg r/m
xmmreg to mem	0000 1111:0001 0011: mod xmmreg r/m
MOVMSKPS—Extract Packed Single-Precision Floating-Point Sign Mask	
xmmreg to r32	0000 1111:0101 0000:11 r32 xmmreg
MOVSS—Move Scalar Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0001 0000:11 xmmreg2 xmmreg1
mem to xmmreg1	1111 0011:0000 1111:0001 0000: mod xmmreg r/m

Table B-22. Formats and Encodings of SSE Floating-Point Instructions (Contd.)

Instruction and Format	Encoding
xmmreg1 to xmmreg2	1111 0011:0000 1111:0001 0001:11 xmmreg1 xmmreg2
xmmreg1 to mem	1111 0011:0000 1111:0001 0001: mod xmmreg r/m
MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0001 0000:11 xmmreg2 xmmreg1
mem to xmmreg1	0000 1111:0001 0000: mod xmmreg r/m
xmmreg1 to xmmreg2	0000 1111:0001 0001:11 xmmreg1 xmmreg2
xmmreg1 to mem	0000 1111:0001 0001: mod xmmreg r/m
MULPS—Multiply Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 1001:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 1001: mod xmmreg r/m
MULSS—Multiply Scalar Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0101 1001:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:0101 1001: mod xmmreg r/m
ORPS—Bitwise Logical OR of Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 0110:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 0110: mod xmmreg r/m
RCPPS—Compute Reciprocals of Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 0011:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 0011: mod xmmreg r/m
RCPSS—Compute Reciprocals of Scalar Single-Precision Floating-Point Value	
xmmreg2 to xmmreg1	1111 0011:0000 1111:01010011:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:01010011: mod xmmreg r/m
RSQRTPS—Compute Reciprocals of Square Roots of Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 0010:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 0010: mode xmmreg r/m
RSQRTSS—Compute Reciprocals of Square Roots of Scalar Single-Precision Floating-Point Value	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0101 0010:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:0101 0010: mod xmmreg r/m
SHUFPS—Shuffle Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1, imm8	0000 1111:1100 0110:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0000 1111:1100 0110: mod xmmreg r/m: imm8
SQRTPS—Compute Square Roots of Packed Single-Precision Floating-Point Values	

Table B-22. Formats and Encodings of SSE Floating-Point Instructions (Contd.)

Instruction and Format	Encoding
xmmreg2 to xmmreg1	0000 1111:0101 0001:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 0001: mod xmmreg r/m
SQRTSS—Compute Square Root of Scalar Single-Precision Floating-Point Value	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0101 0001:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:0101 0001:mod xmmreg r/m
STMXCSR—Store MXCSR Register State	
MXCSR to mem	0000 1111:1010 1110:mod^{A} 011 mem
SUBPS—Subtract Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 1100:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 1100:mod xmmreg r/m
SUBSS—Subtract Scalar Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0101 1100:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:0101 1100:mod xmmreg r/m
UCOMISS—Unordered Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS	
xmmreg2 to xmmreg1	0000 1111:0010 1110:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0010 1110: mod xmmreg r/m
UNPCKHPS—Unpack and Interleave High Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0001 0101:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0001 0101: mod xmmreg r/m
UNPCKLPS—Unpack and Interleave Low Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0001 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0001 0100: mod xmmreg r/m
XORPS—Bitwise Logical XOR of Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 0111:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 0111: mod xmmreg r/m

Table B-23. Formats and Encodings of SSE Integer Instructions

Instruction and Format	Encoding
PAVGB/PAVGW—Average Packed Integers	
mmreg2 to mmreg1	0000 1111:1110 0000:11 mmreg1 mmreg2
	0000 1111:1110 0011:11 mmreg1 mmreg2
mem to mmreg	0000 1111:1110 0000: mod mmreg r/m
	0000 1111:1110 0011: mod mmreg r/m
PEXTRW—Extract Word	
mmreg to reg32, imm8	0000 1111:1100 0101:11 r32 mmreg: imm8
PINSRW—Insert Word	
reg32 to mmreg, imm8	0000 1111:1100 0100:11 mmreg r32: imm8
m16 to mmreg, imm8	0000 1111:1100 0100: mod mmreg r/m: imm8
PMAXSW—Maximum of Packed Signed Word Integers	
mmreg2 to mmreg1	0000 1111:1110 1110:11 mmreg1 mmreg2
mem to mmreg	0000 1111:1110 1110: mod mmreg r/m
PMAXUB—Maximum of Packed Unsigned Byte Integers	
mmreg2 to mmreg1	0000 1111:1101 1110:11 mmreg1 mmreg2
mem to mmreg	0000 1111:1101 1110: mod mmreg r/m
PMINSW—Minimum of Packed Signed Word Integers	
mmreg2 to mmreg1	0000 1111:1110 1010:11 mmreg1 mmreg2
mem to mmreg	0000 1111:1110 1010: mod mmreg r/m
PMINUB—Minimum of Packed Unsigned Byte Integers	
mmreg2 to mmreg1	0000 1111:1101 1010:11 mmreg1 mmreg2
mem to mmreg	0000 1111:1101 1010: mod mmreg r/m
PMOVMSKB—Move Byte Mask To Integer	
mmreg to reg32	0000 1111:1101 0111:11 r32 mmreg
PMULHUW—Multiply Packed Unsigned Integers and Store High Result	
mmreg2 to mmreg1	0000 1111:1110 0100:11 mmreg1 mmreg2
mem to mmreg	0000 1111:1110 0100: mod mmreg r/m
PSADBW—Compute Sum of Absolute Differences	
mmreg2 to mmreg1	0000 1111:1111 0110:11 mmreg1 mmreg2
mem to mmreg	0000 1111:1111 0110: mod mmreg r/m
PSHUFW—Shuffle Packed Words	
mmreg2 to mmreg1, imm8	0000 1111:0111 0000:11 mmreg1 mmreg2: imm8
mem to mmreg, imm8	0000 1111:0111 0000: mod mmreg r/m: imm8

Table B-24. Format and Encoding of SSE Cacheability & Memory Ordering Instructions

Instruction and Format	Encoding
MASKMOVQ—Store Selected Bytes of Quadword	
mmreg2 to mmreg1	0000 1111:1111 0111:11 mmreg1 mmreg2
MOVNTPS—Store Packed Single-Precision Floating-Point Values Using Non-Temporal Hint	
xmmreg to mem	0000 1111:0010 1011: mod xmmreg r/m
MOVNTQ—Store Quadword Using Non-Temporal Hint	
mmreg to mem	0000 1111:1110 0111: mod mmreg r/m
PREFETCHT0—Prefetch Temporal to All Cache Levels	0000 1111:0001 1000:mod^A 001 mem
PREFETCHT1—Prefetch Temporal to First Level Cache	0000 1111:0001 1000:mod^A 010 mem
PREFETCHT2—Prefetch Temporal to Second Level Cache	0000 1111:0001 1000:mod^A 011 mem
PREFETCHNTA—Prefetch Non-Temporal to All Cache Levels	0000 1111:0001 1000:mod^A 000 mem
SFENCE—Store Fence	0000 1111:1010 1110:11 111 000

B.9 SSE2 INSTRUCTION FORMATS AND ENCODINGS

The SSE2 instructions use the ModR/M format and are preceded by the 0FH prefix byte. In general, operations are not duplicated to provide two directions (that is, separate load and store variants).

The following three tables show the formats and encodings for the SSE2 SIMD floating-point, SIMD integer, and cacheability instructions, respectively. Some SSE2 instructions require a mandatory prefix (66H, F2H, F3H) as part of the two-byte opcode. These prefixes are included in the tables.

B.9.1 Granularity Field (gg)

The granularity field (gg) indicates the size of the packed operands that the instruction is operating on. When this field is used, it is located in bits 1 and 0 of the second opcode byte. Table B-25 shows the encoding of this gg field.

Table B-25. Encoding of Granularity of Data Field (gg)

gg	Granularity of Data
00	Packed Bytes
01	Packed Words
10	Packed Doublewords
11	Quadword

Table B-26. Formats and Encodings of SSE2 Floating-Point Instructions

Instruction and Format	Encoding
ADDPD—Add Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 1000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 1000: mod xmmreg r/m
ADDSD—Add Scalar Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	1111 0010:0000 1111:0101 1000:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0010:0000 1111:0101 1000: mod xmmreg r/m
ANDNPD—Bitwise Logical AND NOT of Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 0101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 0101: mod xmmreg r/m
ANDPD—Bitwise Logical AND of Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 0100: mod xmmreg r/m
CMPPD—Compare Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:1100 0010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:1100 0010: mod xmmreg r/m: imm8
CMPSD—Compare Scalar Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1, imm8	1111 0010:0000 1111:1100 0010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	11110 010:0000 1111:1100 0010: mod xmmreg r/m: imm8
COMISD—Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0010 1111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0010 1111: mod xmmreg r/m
CVTPI2PD—Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values	
mmreg to xmmreg	0110 0110:0000 1111:0010 1010:11 xmmreg1 mmreg1
mem to xmmreg	0110 0110:0000 1111:0010 1010: mod xmmreg r/m
CVTPD2PI—Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to mmreg	0110 0110:0000 1111:0010 1101:11 mmreg1 xmmreg1
mem to mmreg	0110 0110:0000 1111:0010 1101: mod mmreg r/m
CVTSI2SD—Convert Doubleword Integer to Scalar Double-Precision Floating-Point Value	
r32 to xmmreg1	1111 0010:0000 1111:0010 1010:11 xmmreg r32
mem to xmmreg	1111 0010:0000 1111:0010 1010: mod xmmreg r/m
CVTSD2SI—Convert Scalar Double-Precision Floating-Point Value to Doubleword Integer	

Table B-26. Formats and Encodings of SSE2 Floating-Point Instructions (Contd.)

Instruction and Format	Encoding
xmmreg to r32	1111 0010:0000 1111:0010 1101:11 r32 xmmreg
mem to r32	1111 0010:0000 1111:0010 1101: mod r32 r/m
CVTTPD2PI—Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to mmreg	0110 0110:0000 1111:0010 1100:11 mmreg xmmreg
mem to mmreg	0110 0110:0000 1111:0010 1100: mod mmreg r/m
CVTTSD2SI—Convert with Truncation Scalar Double-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	1111 0010:0000 1111:0010 1100:11 r32 xmmreg
mem to r32	1111 0010:0000 1111:0010 1100: mod r32 r/m
CVTPD2PS—Covert Packed Double-Precision Floating-Point Values to Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 1010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 1010: mod xmmreg r/m
CVTPS2PD—Covert Packed Single-Precision Floating-Point Values to Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 1010:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 1010: mod xmmreg r/m
CVTSD2SS—Covert Scalar Double-Precision Floating-Point Value to Scalar Single-Precision Floating-Point Value	
xmmreg2 to xmmreg1	1111 0010:0000 1111:0101 1010:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0010:0000 1111:0101 1010: mod xmmreg r/m
CVTSS2SD—Covert Scalar Single-Precision Floating-Point Value to Scalar Double-Precision Floating-Point Value	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0101 1010:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:00001 111:0101 1010: mod xmmreg r/m
CVTPD2DQ—Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg2 to xmmreg1	1111 0010:0000 1111:1110 0110:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0010:0000 1111:1110 0110: mod xmmreg r/m
CVTTPD2DQ—Convert With Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1110 0110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:1110 0110: mod xmmreg r/m

Table B-26. Formats and Encodings of SSE2 Floating-Point Instructions (Contd.)

Instruction and Format	Encoding
CVTDQ2PD—Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	1111 0011:0000 1111:1110 0110:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:1110 0110: mod xmmreg r/m
CVTPS2DQ—Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 1011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 1011: mod xmmreg r/m
CVTTPS2DQ—Convert With Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0101 1011:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:0101 1011: mod xmmreg r/m
CVTDQ2PS—Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0000 1111:0101 1011:11 xmmreg1 xmmreg2
mem to xmmreg	0000 1111:0101 1011: mod xmmreg r/m
DIVPD—Divide Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 1110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 1110: mod xmmreg r/m
DIVSD—Divide Scalar Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	1111 0010:0000 1111:0101 1110:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0010:0000 1111:0101 1110: mod xmmreg r/m
MAXPD—Return Maximum Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 1111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 1111: mod xmmreg r/m
MAXSD—Return Maximum Scalar Double-Precision Floating-Point Value	
xmmreg2 to xmmreg1	1111 0010:0000 1111:0101 1111:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0010:0000 1111:0101 1111: mod xmmreg r/m
MINPD—Return Minimum Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 1101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 1101: mod xmmreg r/m
MINSD—Return Minimum Scalar Double-Precision Floating-Point Value	
xmmreg2 to xmmreg1	1111 0010:0000 1111:0101 1101:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0010:0000 1111:0101 1101: mod xmmreg r/m

Table B-26. Formats and Encodings of SSE2 Floating-Point Instructions (Contd.)

Instruction and Format	Encoding
MOVAPD—Move Aligned Packed Double-Precision Floating-Point Values	
xmmreg1 to xmmreg2	0110 0110:0000 1111:0010 1001:11 xmmreg2 xmmreg1
xmmreg1 to mem	0110 0110:0000 1111:0010 1001: mod xmmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0010 1000:11 xmmreg1 xmmreg2
mem to xmmreg1	0110 0110:0000 1111:0010 1000: mod xmmreg r/m
MOVHPD—Move High Packed Double-Precision Floating-Point Values	
xmmreg to mem	0110 0110:0000 1111:0001 0111: mod xmmreg r/m
mem to xmmreg	0110 0110:0000 1111:0001 0110: mod xmmreg r/m
MOVLPD—Move Low Packed Double-Precision Floating-Point Values	
xmmreg to mem	0110 0110:0000 1111:0001 0011: mod xmmreg r/m
mem to xmmreg	0110 0110:0000 1111:0001 0010: mod xmmreg r/m
MOVMSKPD—Extract Packed Double-Precision Floating-Point Sign Mask	
xmmreg to r32	0110 0110:0000 1111:0101 0000:11 r32 xmmreg
MOVSD—Move Scalar Double-Precision Floating-Point Values	
xmmreg1 to xmmreg2	1111 0010:0000 1111:0001 0001:11 xmmreg2 xmmreg1
xmmreg1 to mem	1111 0010:0000 1111:0001 0001: mod xmmreg r/m
xmmreg2 to xmmreg1	1111 0010:0000 1111:0001 0000:11 xmmreg1 xmmreg2
mem to xmmreg1	1111 0010:0000 1111:0001 0000: mod xmmreg r/m
MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0001 0001:11 xmmreg2 xmmreg1
mem to xmmreg1	0110 0110:0000 1111:0001 0001: mod xmmreg r/m
xmmreg1 to xmmreg2	0110 0110:0000 1111:0001 0000:11 xmmreg1 xmmreg2
xmmreg1 to mem	0110 0110:0000 1111:0001 0000: mod xmmreg r/m
MULPD—Multiply Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 1001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 1001: mod xmmreg r/m
MULSD—Multiply Scalar Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	1111 0010:00001111:01011001:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0010:00001111:01011001: mod xmmreg r/m
ORPD—Bitwise Logical OR of Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 0110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 0110: mod xmmreg r/m

Table B-26. Formats and Encodings of SSE2 Floating-Point Instructions (Contd.)

Instruction and Format	Encoding
SHUFPD—Shuffle Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:1100 0110:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:1100 0110: mod xmmreg r/m: imm8
SQRTPD—Compute Square Roots of Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 0001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 0001: mod xmmreg r/m
SQRTSD—Compute Square Root of Scalar Double-Precision Floating-Point Value	
xmmreg2 to xmmreg1	1111 0010:0000 1111:0101 0001:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0010:0000 1111:0101 0001: mod xmmreg r/m
SUBPD—Subtract Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 1100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 1100: mod xmmreg r/m
SUBSD—Subtract Scalar Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	1111 0010:0000 1111:0101 1100:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0010:0000 1111:0101 1100: mod xmmreg r/m
UCOMISD—Unordered Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0010 1110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0010 1110: mod xmmreg r/m
UNPCKHPD—Unpack and Interleave High Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0001 0101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0001 0101: mod xmmreg r/m
UNPCKLPD—Unpack and Interleave Low Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0001 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0001 0100: mod xmmreg r/m
XORPD—Bitwise Logical OR of Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0101 0111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0101 0111: mod xmmreg r/m

Table B-27. Formats and Encodings of SSE2 Integer Instructions

Instruction and Format	Encoding
MOVD—Move Doubleword	
reg to xmmreg	0110 0110:0000 1111:0110 1110: 11 xmmreg reg
reg from xmmreg	0110 0110:0000 1111:0111 1110: 11 xmmreg reg
mem to xmmreg	0110 0110:0000 1111:0110 1110: mod xmmreg r/m
mem from xmmreg	0110 0110:0000 1111:0111 1110: mod xmmreg r/m
MOVDQA—Move Aligned Double Quadword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0110 1111:11 xmmreg1 xmmreg2
xmmreg2 from xmmreg1	0110 0110:0000 1111:0111 1111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0110 1111: mod xmmreg r/m
mem from xmmreg	0110 0110:0000 1111:0111 1111: mod xmmreg r/m
MOVDQU—Move Unaligned Double Quadword	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0110 1111:11 xmmreg1 xmmreg2
xmmreg2 from xmmreg1	1111 0011:0000 1111:0111 1111:11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:0110 1111: mod xmmreg r/m
mem from xmmreg	1111 0011:0000 1111:0111 1111: mod xmmreg r/m
MOVQ2DQ—Move Quadword from MMX to XMM Register	
mmreg to xmmreg	1111 0011:0000 1111:1101 0110:11 mmreg1 mmreg2
MOVDQ2Q—Move Quadword from XMM to MMX Register	
xmmreg to mmreg	1111 0010:0000 1111:1101 0110:11 mmreg1 mmreg2
MOVQ—Move Quadword	
xmmreg2 to xmmreg1	1111 0011:0000 1111:0111 1110: 11 xmmreg1 xmmreg2
xmmreg2 from xmmreg1	0110 0110:0000 1111:1101 0110: 11 xmmreg1 xmmreg2
mem to xmmreg	1111 0011:0000 1111:0111 1110: mod xmmreg r/m
mem from xmmreg	0110 0110:0000 1111:1101 0110: mod xmmreg r/m
PACKSSDW[1]—Pack Dword To Word Data (signed with saturation)	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0110 1011: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:0110 1011: mod xmmreg r/m
PACKSSWB—Pack Word To Byte Data (signed with saturation)	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0110 0011: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:0110 0011: mod xmmreg r/m
PACKUSWB—Pack Word To Byte Data (unsigned with saturation)	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0110 0111: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:0110 0111: mod xmmreg r/m
PADDQ—Add Packed Quadword Integers	
mmreg2 to mmreg1	0000 1111:1101 0100:11 mmreg1 mmreg2

Table B-27. Formats and Encodings of SSE2 Integer Instructions (Contd.)

Instruction and Format	Encoding
mem to mmreg	0000 1111:1101 0100: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:1101 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:1101 0100: mod xmmreg r/m
PADD—Add With Wrap-around	
xmmreg2 to xmmreg1	0110 0110:0000 1111: 1111 11gg: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111: 1111 11gg: mod xmmreg r/m
PADDS—Add Signed With Saturation	
xmmreg2 to xmmreg1	0110 0110:0000 1111: 1110 11gg: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111: 1110 11gg: mod xmmreg r/m
PADDUS—Add Unsigned With Saturation	
xmmreg2 to xmmreg1	0110 0110:0000 1111: 1101 11gg: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111: 1101 11gg: mod xmmreg r/m
PAND—Bitwise And	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1101 1011: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:1101 1011: mod xmmreg r/m
PANDN—Bitwise AndNot	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1101 1111: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:1101 1111: mod xmmreg r/m
PAVGB—Average Packed Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:11100 000:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11100000 mod xmmreg r/m
PAVGW—Average Packed Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1110 0011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:1110 0011 mod xmmreg r/m
PCMPEQ—Packed Compare For Equality	
xmmreg1 with xmmreg2	0110 0110:0000 1111:0111 01gg: 11 xmmreg1 xmmreg2
xmmreg with memory	0110 0110:0000 1111:0111 01gg: mod xmmreg r/m
PCMPGT—Packed Compare Greater (signed)	
xmmreg1 with xmmreg2	0110 0110:0000 1111:0110 01gg: 11 xmmreg1 xmmreg2
xmmreg with memory	0110 0110:0000 1111:0110 01gg: mod xmmreg r/m
PEXTRW—Extract Word	
xmmreg to reg32, imm8	0110 0110:0000 1111:1100 0101:11 r32 xmmreg: imm8
PINSRW—Insert Word	
reg32 to xmmreg, imm8	0110 0110:0000 1111:1100 0100:11 xmmreg r32: imm8
m16 to xmmreg, imm8	0110 0110:0000 1111:1100 0100: mod xmmreg r/m: imm8
PMADDWD—Packed Multiply Add	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1111 0101: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:1111 0101: mod xmmreg r/m

Table B-27. Formats and Encodings of SSE2 Integer Instructions (Contd.)

Instruction and Format	Encoding
PMAXSW—Maximum of Packed Signed Word Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1110 1110:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11101110: mod xmmreg r/m
PMAXUB—Maximum of Packed Unsigned Byte Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1101 1110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:1101 1110: mod xmmreg r/m
PMINSW—Minimum of Packed Signed Word Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1110 1010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:1110 1010: mod xmmreg r/m
PMINUB—Minimum of Packed Unsigned Byte Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1101 1010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:1101 1010 mod xmmreg r/m
PMOVMSKB—Move Byte Mask To Integer	
xmmreg to reg32	0110 0110:0000 1111:1101 0111:11 r32 xmmreg
PMULHUW—Packed multiplication, store high word (unsigned)	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1110 0100: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:1110 0100: mod xmmreg r/m
PMULHW—Packed Multiplication, store high word	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1110 0101: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:1110 0101: mod xmmreg r/m
PMULLW—Packed Multiplication, store low word	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1101 0101: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:1101 0101: mod xmmreg r/m
PMULUDQ—Multiply Packed Unsigned Doubleword Integers	
mmreg2 to mmreg1	0000 1111:1111 0100:11 mmreg1 mmreg2
mem to mmreg	0000 1111:1111 0100: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:00001111:1111 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:00001111:1111 0100: mod xmmreg r/m
POR—Bitwise Or	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1110 1011: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:1110 1011: mod xmmreg r/m
PSADBW—Compute Sum of Absolute Differences	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1111 0110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:1111 0110: mod xmmreg r/m
PSHUFLW—Shuffle Packed Low Words	

Table B-27. Formats and Encodings of SSE2 Integer Instructions (Contd.)

Instruction and Format	Encoding
xmmreg2 to xmmreg1, imm8	1111 0010:0000 1111:0111 0000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	1111 0010:0000 1111:0111 0000:11 mod xmmreg r/m: imm8
PSHUFHW—Shuffle Packed High Words	
xmmreg2 to xmmreg1, imm8	1111 0011:0000 1111:0111 0000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	1111 0011:0000 1111:0111 0000: mod xmmreg r/m: imm8
PSHUFD—Shuffle Packed Doublewords	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0111 0000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0111 0000: mod xmmreg r/m: imm8
PSLLDQ—Shift Double Quadword Left Logical	
xmmreg, imm8	0110 0110:0000 1111:0111 0011:11 111 xmmreg: imm8
PSLL—Packed Shift Left Logical	
xmmreg1 by xmmreg2	0110 0110:0000 1111:1111 00gg: 11 xmmreg1 xmmreg2
xmmreg by memory	0110 0110:0000 1111:1111 00gg: mod xmmreg r/m
xmmreg by immediate	0110 0110:0000 1111:0111 00gg: 11 110 xmmreg: imm8
PSRA—Packed Shift Right Arithmetic	
xmmreg1 by xmmreg2	0110 0110:0000 1111:1110 00gg: 11 xmmreg1 xmmreg2
xmmreg by memory	0110 0110:0000 1111:1110 00gg: mod xmmreg r/m
xmmreg by immediate	0110 0110:0000 1111:0111 00gg: 11 100 xmmreg: imm8
PSRLDQ—Shift Double Quadword Right Logical	
xmmreg, imm8	0110 0110:00001111:01110011:11 011 xmmreg: imm8
PSRL—Packed Shift Right Logical	
xmmreg1 by xmmreg2	0110 0110:0000 1111:1101 00gg: 11 xmmreg1 xmmreg2
xmmreg by memory	0110 0110:0000 1111:1101 00gg: mod xmmreg r/m
xmmreg by immediate	0110 0110:0000 1111:0111 00gg: 11 010 xmmreg: imm8
PSUBQ—Subtract Packed Quadword Integers	
mmreg2 to mmreg1	0000 1111:11111 011:11 mmreg1 mmreg2
mem to mmreg	0000 1111:1111 1011: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:1111 1011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:1111 1011: mod xmmreg r/m
PSUB—Subtract With Wrap-around	
xmmreg2 from xmmreg1	0110 0110:0000 1111:1111 10gg: 11 xmmreg1 xmmreg2
memory from xmmreg	0110 0110:0000 1111:1111 10gg: mod xmmreg r/m
PSUBS—Subtract Signed With Saturation	
xmmreg2 from xmmreg1	0110 0110:0000 1111:1110 10gg: 11 xmmreg1 xmmreg2
memory from xmmreg	0110 0110:0000 1111:1110 10gg: mod xmmreg r/m
PSUBUS—Subtract Unsigned With Saturation	
xmmreg2 from xmmreg1	0000 1111:1101 10gg: 11 xmmreg1 xmmreg2
memory from xmmreg	0000 1111:1101 10gg: mod xmmreg r/m

Table B-27. Formats and Encodings of SSE2 Integer Instructions (Contd.)

Instruction and Format	Encoding
PUNPCKH—Unpack High Data To Next Larger Type	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0110 10gg:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0110 10gg: mod xmmreg r/m
PUNPCKHQDQ—Unpack High Data	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0110 1101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0110 1101: mod xmmreg r/m
PUNPCKL—Unpack Low Data To Next Larger Type	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0110 00gg:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0110 00gg: mod xmmreg r/m
PUNPCKLQDQ—Unpack Low Data	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0110 1100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0110 1100: mod xmmreg r/m
PXOR—Bitwise Xor	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1110 1111: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:1110 1111: mod xmmreg r/m

Table B-28. Format and Encoding of SSE2 Cacheability Instructions

Instruction and Format	Encoding
MASKMOVDQU—Store Selected Bytes of Double Quadword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1111 0111:11 xmmreg1 xmmreg2
CLFLUSH—Flush Cache Line	
mem	0000 1111:1010 1110: mod 111 r/m
MOVNTPD—Store Packed Double-Precision Floating-Point Values Using Non-Temporal Hint	
xmmreg to mem	0110 0110:0000 1111:0010 1011: mod xmmreg r/m
MOVNTDQ—Store Double Quadword Using Non-Temporal Hint	
xmmreg to mem	0110 0110:0000 1111:1110 0111: mod xmmreg r/m
MOVNTI—Store Doubleword Using Non-Temporal Hint	
reg to mem	0000 1111:1100 0011: mod reg r/m
PAUSE—Spin Loop Hint	1111 0011:1001 0000
LFENCE—Load Fence	0000 1111:1010 1110: 11 101 000
MFENCE—Memory Fence	0000 1111:1010 1110: 11 110 000

B.10 SSE3 FORMATS AND ENCODINGS TABLE

The tables in this section provide SSE3 formats and encodings. Some SSE3 instructions require a mandatory prefix (66H, F2H, F3H) as part of the two-byte opcode. These prefixes are included in the tables.

When in IA-32e mode, use of the REX.R prefix permits instructions that use general purpose and XMM registers to access additional registers. Some instructions require the REX.W prefix to promote the instruction to 64-bit operation. Instructions that require the REX.W prefix are listed (with their opcodes) in Section B.13.

Table B-29. Formats and Encodings of SSE3 Floating-Point Instructions

Instruction and Format	Encoding
ADDSUBPD—Add /Sub packed DP FP numbers from XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	01100110:00001111:11010000:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11010000: mod xmmreg r/m
ADDSUBPS—Add /Sub packed SP FP numbers from XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	11110010:00001111:11010000:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:11010000: mod xmmreg r/m
HADDPD—Add horizontally packed DP FP numbers XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	01100110:00001111:01111100:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01111100: mod xmmreg r/m
HADDPS—Add horizontally packed SP FP numbers XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	11110010:00001111:01111100:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01111100: mod xmmreg r/m
HSUBPD—Sub horizontally packed DP FP numbers XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	01100110:00001111:01111101:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01111101: mod xmmreg r/m
HSUBPS—Sub horizontally packed SP FP numbers XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	11110010:00001111:01111101:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01111101: mod xmmreg r/m

Table B-30. Formats and Encodings for SSE3 Event Management Instructions

Instruction and Format	Encoding
MONITOR—Set up a linear address range to be monitored by hardware	
eax, ecx, edx	0000 1111 : 0000 0001:11 001 000
MWAIT—Wait until write-back store performed within the range specified by the instruction MONITOR	
eax, ecx	0000 1111 : 0000 0001:11 001 001

Table B-31. Formats and Encodings for SSE3 Integer and Move Instructions

Instruction and Format	Encoding
FISTTP—Store ST in int16 (chop) and pop	
m16int	11011 111 : mod^A 001 r/m
FISTTP—Store ST in int32 (chop) and pop	
m32int	11011 011 : mod^A 001 r/m
FISTTP—Store ST in int64 (chop) and pop	
m64int	11011 101 : mod^A 001 r/m
LDDQU—Load unaligned integer 128-bit	
xmm, m128	11110010:00001111:11110000: mod^A xmmreg r/m
MOVDDUP—Move 64 bits representing one DP data from XMM2/Mem to XMM1 and duplicate	
xmmreg2 to xmmreg1	11110010:00001111:00010010:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:00010010: mod xmmreg r/m
MOVSHDUP—Move 128 bits representing 4 SP data from XMM2/Mem to XMM1 and duplicate high	
xmmreg2 to xmmreg1	11110011:00001111:00010110:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:00010110: mod xmmreg r/m
MOVSLDUP—Move 128 bits representing 4 SP data from XMM2/Mem to XMM1 and duplicate low	
xmmreg2 to xmmreg1	11110011:00001111:00010010:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:00010010: mod xmmreg r/m

B.11 SSSE3 FORMATS AND ENCODING TABLE

The tables in this section provide SSSE3 formats and encodings. Some SSSE3 instructions require a mandatory prefix (66H) as part of the three-byte opcode. These prefixes are included in the table below.

Table B-32. Formats and Encodings for SSSE3 Instructions

Instruction and Format	Encoding
PABSB—Packed Absolute Value Bytes	
mmreg2 to mmreg1	0000 1111:0011 1000: 0001 1100:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0001 1100: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0001 1100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0001 1100: mod xmmreg r/m
PABSD—Packed Absolute Value Double Words	
mmreg2 to mmreg1	0000 1111:0011 1000: 0001 1110:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0001 1110: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0001 1110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0001 1110: mod xmmreg r/m
PABSW—Packed Absolute Value Words	

Table B-32. Formats and Encodings for SSSE3 Instructions (Contd.)

Instruction and Format	Encoding
mmreg2 to mmreg1	0000 1111:0011 1000: 0001 1101:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0001 1101: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0001 1101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0001 1101: mod xmmreg r/m
PALIGNR—Packed Align Right	
mmreg2 to mmreg1, imm8	0000 1111:0011 1010: 0000 1111:11 mmreg1 mmreg2: imm8
mem to mmreg, imm8	0000 1111:0011 1010: 0000 1111: mod mmreg r/m: imm8
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0000 1111:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1111: mod xmmreg r/m: imm8
PHADDD—Packed Horizontal Add Double Words	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 0010:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 0010: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 0010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 0010: mod xmmreg r/m
PHADDSW—Packed Horizontal Add and Saturate	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 0011:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 0011: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 0011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 0011: mod xmmreg r/m
PHADDW—Packed Horizontal Add Words	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 0001:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 0001: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 0001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 0001: mod xmmreg r/m
PHSUBD—Packed Horizontal Subtract Double Words	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 0110:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 0110: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 0110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 0110: mod xmmreg r/m
PHSUBSW—Packed Horizontal Subtract and Saturate	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 0111:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 0111: mod mmreg r/m

Table B-32. Formats and Encodings for SSSE3 Instructions (Contd.)

Instruction and Format	Encoding
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 0111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 0111: mod xmmreg r/m
PHSUBW—Packed Horizontal Subtract Words	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 0101:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 0101: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 0101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 0101: mod xmmreg r/m
PMADDUBSW—Multiply and Add Packed Signed and Unsigned Bytes	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 0100:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 0100: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 0100: mod xmmreg r/m
PMULHRSW—Packed Multiply HIgn with Round and Scale	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 1011:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 1011: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 1011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 1011: mod xmmreg r/m
PSHUFB—Packed Shuffle Bytes	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 0000:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 0000: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 0000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 0000: mod xmmreg r/m
PSIGNB—Packed Sign Bytes	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 1000:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 1000: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 1000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 1000: mod xmmreg r/m
PSIGND—Packed Sign Double Words	
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 1010:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 1010: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 1010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 1010: mod xmmreg r/m
PSIGNW—Packed Sign Words	

Table B-32. Formats and Encodings for SSSE3 Instructions (Contd.)

Instruction and Format	Encoding
mmreg2 to mmreg1	0000 1111:0011 1000: 0000 1001:11 mmreg1 mmreg2
mem to mmreg	0000 1111:0011 1000: 0000 1001: mod mmreg r/m
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0000 1001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0000 1001: mod xmmreg r/m

B.12 AESNI AND PCLMULQDQ INSTRUCTION FORMATS AND ENCODINGS

Table B-33 shows the formats and encodings for AESNI and PCLMULQDQ instructions.

Table B-33. Formats and Encodings of AESNI and PCLMULQDQ Instructions

Instruction and Format	Encoding
AESDEC—Perform One Round of an AES Decryption Flow	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000:1101 1110: mod xmmreg r/m
AESDECLAST—Perform Last Round of an AES Decryption Flow	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000:1101 1111: mod xmmreg r/m
AESENC—Perform One Round of an AES Encryption Flow	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000:1101 1100: mod xmmreg r/m
AESENCLAST—Perform Last Round of an AES Encryption Flow	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000:1101 1101: mod xmmreg r/m
AESIMC—Perform the AES InvMixColumn Transformation	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000:1101 1011: mod xmmreg r/m
AESKEYGENASSIST—AES Round Key Generation Assist	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010:1101 1111:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010:1101 1111: mod xmmreg r/m: imm8
PCLMULQDQ—Carry-Less Multiplication Quadword	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010:0100 0100:11 xmmreg1 xmmreg2: imm8

Table B-33. Formats and Encodings of AESNI and PCLMULQDQ Instructions

Instruction and Format	Encoding
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010:0100 0100: mod xmmreg r/m: imm8

B.13 SPECIAL ENCODINGS FOR 64-BIT MODE

The following Pentium, P6, MMX, SSE, SSE2, SSE3 instructions are promoted to 64-bit operation in IA-32e mode by using REX.W. However, these entries are special cases that do not follow the general rules (specified in Section B.4).

Table B-34. Special Case Instructions Promoted Using REX.W

Instruction and Format	Encoding
CMOVcc—Conditional Move	
register2 to register1	0100 0R0B 0000 1111: 0100 tttn : 11 reg1 reg2
qwordregister2 to qwordregister1	0100 1R0B 0000 1111: 0100 tttn : 11 qwordreg1 qwordreg2
memory to register	0100 0RXB 0000 1111 : 0100 tttn : mod reg r/m
memory64 to qwordregister	0100 1RXB 0000 1111 : 0100 tttn : mod qwordreg r/m
CVTSD2SI—Convert Scalar Double-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	0100 0R0B 1111 0010:0000 1111:0010 1101:11 r32 xmmreg
xmmreg to r64	0100 1R0B 1111 0010:0000 1111:0010 1101:11 r64 xmmreg
mem64 to r32	0100 0R0XB 1111 0010:0000 1111:0010 1101: mod r32 r/m
mem64 to r64	0100 1RXB 1111 0010:0000 1111:0010 1101: mod r64 r/m
CVTSI2SS—Convert Doubleword Integer to Scalar Single-Precision Floating-Point Value	
r32 to xmmreg1	0100 0R0B 1111 0011:0000 1111:0010 1010:11 xmmreg r32
r64 to xmmreg1	0100 1R0B 1111 0011:0000 1111:0010 1010:11 xmmreg r64
mem to xmmreg	0100 0RXB 1111 0011:0000 1111:0010 1010: mod xmmreg r/m
mem64 to xmmreg	0100 1RXB 1111 0011:0000 1111:0010 1010: mod xmmreg r/m
CVTSI2SD—Convert Doubleword Integer to Scalar Double-Precision Floating-Point Value	
r32 to xmmreg1	0100 0R0B 1111 0010:0000 1111:0010 1010:11 xmmreg r32
r64 to xmmreg1	0100 1R0B 1111 0010:0000 1111:0010 1010:11 xmmreg r64
mem to xmmreg	0100 0RXB 1111 0010:0000 1111:00101 010: mod xmmreg r/m

Table B-34. Special Case Instructions Promoted Using REX.W (Contd.)

Instruction and Format	Encoding
mem64 to xmmreg	0100 1RXB 1111 0010:0000 1111:0010 1010: mod xmmreg r/m
CVTSS2SI—Convert Scalar Single-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	0100 0R0B 1111 0011:0000 1111:0010 1101:11 r32 xmmreg
xmmreg to r64	0100 1R0B 1111 0011:0000 1111:0010 1101:11 r64 xmmreg
mem to r32	0100 0RXB 11110011:00001111:00101101: mod r32 r/m
mem32 to r64	0100 1RXB 1111 0011:0000 1111:0010 1101: mod r64 r/m
CVTTSD2SI—Convert with Truncation Scalar Double-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	0100 0R0B 11110010:00001111:00101100:11 r32 xmmreg
xmmreg to r64	0100 1R0B 1111 0010:0000 1111:0010 1100:11 r64 xmmreg
mem64 to r32	0100 0RXB 1111 0010:0000 1111:0010 1100: mod r32 r/m
mem64 to r64	0100 1RXB 1111 0010:0000 1111:0010 1100: mod r64 r/m
CVTTSS2SI—Convert with Truncation Scalar Single-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	0100 0R0B 1111 0011:0000 1111:0010 1100:11 r32 xmmreg1
xmmreg to r64	0100 1R0B 1111 0011:0000 1111:0010 1100:11 r64 xmmreg1
mem to r32	0100 0RXB 1111 0011:0000 1111:0010 1100: mod r32 r/m
mem32 to r64	0100 1RXB 1111 0011:0000 1111:0010 1100: mod r64 r/m
MOVD/MOVQ—Move doubleword	
reg to mmxreg	0100 0R0B 0000 1111:0110 1110: 11 mmxreg reg
qwordreg to mmxreg	0100 1R0B 0000 1111:0110 1110: 11 mmxreg qwordreg
reg from mmxreg	0100 0R0B 0000 1111:0111 1110: 11 mmxreg reg
qwordreg from mmxreg	0100 1R0B 0000 1111:0111 1110: 11 mmxreg qwordreg
mem to mmxreg	0100 0RXB 0000 1111:0110 1110: mod mmxreg r/m
mem64 to mmxreg	0100 1RXB 0000 1111:0110 1110: mod mmxreg r/m
mem from mmxreg	0100 0RXB 0000 1111:0111 1110: mod mmxreg r/m
mem64 from mmxreg	0100 1RXB 0000 1111:0111 1110: mod mmxreg r/m
mmxreg with memory	0100 0RXB 0000 1111:0110 01gg: mod mmxreg r/m
MOVMSKPS—Extract Packed Single-Precision Floating-Point Sign Mask	
xmmreg to r32	0100 0R0B 0000 1111:0101 0000:11 r32 xmmreg
xmmreg to r64	0100 1R0B 00001111:01010000:11 r64 xmmreg
PEXTRW—Extract Word	
mmreg to reg32, imm8	0100 0R0B 0000 1111:1100 0101:11 r32 mmreg: imm8

Table B-34. Special Case Instructions Promoted Using REX.W (Contd.)

Instruction and Format	Encoding
mmreg to reg64, imm8	0100 1R0B 0000 1111:1100 0101:11 r64 mmreg: imm8
xmmreg to reg32, imm8	0100 0R0B 0110 0110 0000 1111:1100 0101:11 r32 xmmreg: imm8
xmmreg to reg64, imm8	0100 1R0B 0110 0110 0000 1111:1100 0101:11 r64 xmmreg: imm8
PINSRW—Insert Word	
reg32 to mmreg, imm8	0100 0R0B 0000 1111:1100 0100:11 mmreg r32: imm8
reg64 to mmreg, imm8	0100 1R0B 0000 1111:1100 0100:11 mmreg r64: imm8
m16 to mmreg, imm8	0100 0R0B 0000 1111:1100 0100 mod mmreg r/m: imm8
m16 to mmreg, imm8	0100 1RXB 0000 1111:11000100 mod mmreg r/m: imm8
reg32 to xmmreg, imm8	0100 0RXB 0110 0110 0000 1111:1100 0100:11 xmmreg r32: imm8
reg64 to xmmreg, imm8	0100 0RXB 0110 0110 0000 1111:1100 0100:11 xmmreg r64: imm8
m16 to xmmreg, imm8	0100 0RXB 0110 0110 0000 1111:1100 0100 mod xmmreg r/m: imm8
m16 to xmmreg, imm8	0100 1RXB 0110 0110 0000 1111:1100 0100 mod xmmreg r/m: imm8
PMOVMSKB—Move Byte Mask To Integer	
mmreg to reg32	0100 0RXB 0000 1111:1101 0111:11 r32 mmreg
mmreg to reg64	0100 1R0B 0000 1111:1101 0111:11 r64 mmreg
xmmreg to reg32	0100 0RXB 0110 0110 0000 1111:1101 0111:11 r32 mmreg
xmmreg to reg64	0110 0110 0000 1111:1101 0111:11 r64 xmmreg

B.14 SSE4.1 FORMATS AND ENCODING TABLE

The tables in this section provide SSE4.1 formats and encodings. Some SSE4.1 instructions require a mandatory prefix (66H, F2H, F3H) as part of the three-byte opcode. These prefixes are included in the tables.

In 64-bit mode, some instructions requires REX.W, the byte sequence of REX.W prefix in the opcode sequence is shown.

Table B-35. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
BLENDPD — Blend Packed Double-Precision Floats	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1010: 0000 1101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0000 1101: mod xmmreg r/m
BLENDPS — Blend Packed Single-Precision Floats	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1010: 0000 1100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0000 1100: mod xmmreg r/m
BLENDVPD — Variable Blend Packed Double-Precision Floats	

Table B-35. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
xmmreg2 to xmmreg1 <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0101:11 xmmreg1 xmmreg2
mem to xmmreg <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0101: mod xmmreg r/m
BLENDVPS — Variable Blend Packed Single-Precision Floats	
xmmreg2 to xmmreg1 <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0100:11 xmmreg1 xmmreg2
mem to xmmreg <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0100: mod xmmreg r/m
DPPD — Packed Double-Precision Dot Products	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0100 0001:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0100 0001: mod xmmreg r/m: imm8
DPPS — Packed Single-Precision Dot Products	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0100 0000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0100 0000: mod xmmreg r/m: imm8
EXTRACTPS — Extract From Packed Single-Precision Floats	
reg from xmmreg , imm8	0110 0110:0000 1111:0011 1010: 0001 0111:11 xmmreg reg: imm8
mem from xmmreg , imm8	0110 0110:0000 1111:0011 1010: 0001 0111: mod xmmreg r/m: imm8
INSERTPS — Insert Into Packed Single-Precision Floats	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0010 0001:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0001: mod xmmreg r/m: imm8
MOVNTDQA — Load Double Quadword Non-temporal Aligned	
m128 to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1010:11 r/m xmmreg2
MPSADBW — Multiple Packed Sums of Absolute Difference	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0100 0010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0100 0010: mod xmmreg r/m: imm8
PACKUSDW — Pack with Unsigned Saturation	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0010 1011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1011: mod xmmreg r/m
PBLENDVB — Variable Blend Packed Bytes	
xmmreg2 to xmmreg1 <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0000:11 xmmreg1 xmmreg2
mem to xmmreg <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0000: mod xmmreg r/m

Table B-35. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
PBLENDW — Blend Packed Words	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0001 1110:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1110: mod xmmreg r/m: imm8
PCMPEQQ — Compare Packed Qword Data of Equal	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0010 1001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1001: mod xmmreg r/m
PEXTRB — Extract Byte	
reg from xmmreg , imm8	0110 0110:0000 1111:0011 1010: 0001 0100:11 xmmreg reg: imm8
xmmreg to mem, imm8	0110 0110:0000 1111:0011 1010: 0001 0100: mod xmmreg r/m: imm8
PEXTRD — Extract DWord	
reg from xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0001 0110:11 xmmreg reg: imm8
xmmreg to mem, imm8	0110 0110:0000 1111:0011 1010: 0001 0110: mod xmmreg r/m: imm8
PEXTRQ — Extract QWord	
r64 from xmmreg, imm8	0110 0110:REX.W:0000 1111:0011 1010: 0001 0110:11 xmmreg reg: imm8
m64 from xmmreg, imm8	0110 0110:REX.W:0000 1111:0011 1010: 0001 0110: mod xmmreg r/m: imm8
PEXTRW — Extract Word	
reg from xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0001 0101:11 reg xmmreg: imm8
mem from xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0001 0101: mod xmmreg r/m: imm8
PHMINPOSUW — Packed Horizontal Word Minimum	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0100 0001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0100 0001: mod xmmreg r/m
PINSRB — Extract Byte	
reg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0000:11 xmmreg reg: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0000: mod xmmreg r/m: imm8
PINSRD — Extract DWord	
reg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0010:11 xmmreg reg: imm8

Table B-35. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0010: mod xmmreg r/m: imm8
PINSRQ — Extract QWord	
r64 to xmmreg, imm8	0110 0110:REX.W:0000 1111:0011 1010: 0010 0010:11 xmmreg reg: imm8
m64 to xmmreg, imm8	0110 0110:REX.W:0000 1111:0011 1010: 0010 0010: mod xmmreg r/m: imm8
PMAXSB — Maximum of Packed Signed Byte Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 1100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1100: mod xmmreg r/m
PMAXSD — Maximum of Packed Signed Dword Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 1101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1101: mod xmmreg r/m
PMAXUD — Maximum of Packed Unsigned Dword Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 1111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1111: mod xmmreg r/m
PMAXUW — Maximum of Packed Unsigned Word Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 1110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1110: mod xmmreg r/m
PMINSB — Minimum of Packed Signed Byte Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 1000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1000: mod xmmreg r/m
PMINSD — Minimum of Packed Signed Dword Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 1001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1001: mod xmmreg r/m
PMINUD — Minimum of Packed Unsigned Dword Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 1011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1011: mod xmmreg r/m
PMINUW — Minimum of Packed Unsigned Word Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 1010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1010: mod xmmreg r/m
PMOVSXBD — Packed Move Sign Extend - Byte to Dword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0010 0001:11 xmmreg1 xmmreg2

Table B-35. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0001: mod xmmreg r/m
PMOVSXBQ — Packed Move Sign Extend - Byte to Qword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0010 0010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0010: mod xmmreg r/m
PMOVSXBW — Packed Move Sign Extend - Byte to Word	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0010 0000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0000: mod xmmreg r/m
PMOVSXWD — Packed Move Sign Extend - Word to Dword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0010 0011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0011: mod xmmreg r/m
PMOVSXWQ — Packed Move Sign Extend - Word to Qword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0010 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0100: mod xmmreg r/m
PMOVSXDQ — Packed Move Sign Extend - Dword to Qword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0010 0101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0101: mod xmmreg r/m
PMOVZXBD — Packed Move Zero Extend - Byte to Dword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 0001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0001: mod xmmreg r/m
PMOVZXBQ — Packed Move Zero Extend - Byte to Qword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 0010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0010: mod xmmreg r/m
PMOVZXBW — Packed Move Zero Extend - Byte to Word	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 0000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0000: mod xmmreg r/m
PMOVZXWD — Packed Move Zero Extend - Word to Dword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 0011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0011: mod xmmreg r/m
PMOVZXWQ — Packed Move Zero Extend - Word to Qword	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0100: mod xmmreg r/m
PMOVZXDQ — Packed Move Zero Extend - Dword to Qword	

Table B-35. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0011 0101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0101: mod xmmreg r/m
PMULDQ — Multiply Packed Signed Dword Integers	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0010 1000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1000: mod xmmreg r/m
PMULLD — Multiply Packed Signed Dword Integers, Store low Result	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0100 0000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0100 0000: mod xmmreg r/m
PTEST — Logical Compare	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000: 0001 0111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0001 0111: mod xmmreg r/m
ROUNDPD — Round Packed Double-Precision Values	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0000 1001:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1001: mod xmmreg r/m: imm8
ROUNDPS — Round Packed Single-Precision Values	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0000 1000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1000: mod xmmreg r/m: imm8
ROUNDSD — Round Scalar Double-Precision Value	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0000 1011:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1011: mod xmmreg r/m: imm8
ROUNDSS — Round Scalar Single-Precision Value	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0000 1010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1010: mod xmmreg r/m: imm8

B.15 SSE4.2 FORMATS AND ENCODING TABLE

The tables in this section provide SSE4.2 formats and encodings. Some SSE4.2 instructions require a mandatory prefix (66H, F2H, F3H) as part of the three-byte opcode. These prefixes are included in the tables. In 64-bit mode, some instructions requires REX.W, the byte sequence of REX.W prefix in the opcode sequence is shown.

Table B-36. Encodings of SSE4.2 instructions

Instruction and Format	Encoding
CRC32 — Accumulate CRC32	
reg2 to reg1	1111 0010:0000 1111:0011 1000: 1111 000w :11 reg1 reg2
mem to reg	1111 0010:0000 1111:0011 1000: 1111 000w : mod reg r/m
bytereg2 to reg1	1111 0010:0100 WR0B:0000 1111:0011 1000: 1111 0000 :11 reg1 bytereg2
m8 to reg	1111 0010:0100 WR0B:0000 1111:0011 1000: 1111 0000 : mod reg r/m
qwreg2 to qwreg1	1111 0010:0100 1R0B:0000 1111:0011 1000: 1111 0001 :11 qwreg1 qwreg2
mem64 to qwreg	1111 0010:0100 1R0B:0000 1111:0011 1000: 1111 0001 : mod qwreg r/m
PCMPESTRI— Packed Compare Explicit-Length Strings To Index	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0001:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0001: mod xmmreg r/m
PCMPESTRM— Packed Compare Explicit-Length Strings To Mask	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0000: mod xmmreg r/m
PCMPISTRI— Packed Compare Implicit-Length String To Index	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0011:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0011: mod xmmreg r/m
PCMPISTRM— Packed Compare Implicit-Length Strings To Mask	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0010: mod xmmreg r/m
PCMPGTQ— Packed Compare Greater Than	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0111: mod xmmreg r/m
POPCNT— Return Number of Bits Set to 1	
reg2 to reg1	1111 0011:0000 1111:1011 1000:11 reg1 reg2
mem to reg1	1111 0011:0000 1111:1011 1000:mod reg1 r/m
qwreg2 to qwreg1	1111 0011:0100 1R0B:0000 1111:1011 1000:11 reg1 reg2
mem64 to qwreg1	1111 0011:0100 1R0B:0000 1111:1011 1000:mod reg1 r/m

B.16 AVX FORMATS AND ENCODING TABLE

The tables in this section provide AVX formats and encodings. A mixed form of bit/hex/symbolic forms are used to express the various bytes:

The C4/C5 and opcode bytes are expressed in hex notation; the first and second payload byte of VEX, the modR/M byte is expressed in combination of bit/symbolic form. The first payload byte of C4 is expressed as combination of bits and hex form, with the hex value preceded by an underscore. The VEX bit field to encode upper register 8-15 uses 1's complement form, each of those bit field is expressed as lower case notation rxb, instead of RXB.

The hybrid bit-nibble-byte form is depicted below:

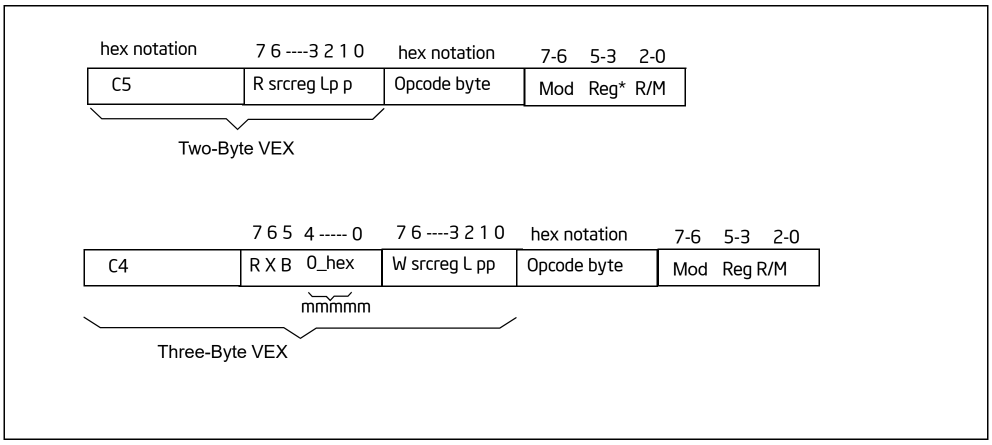


Figure B-2. Hybrid Notation of VEX-Encoded Key Instruction Bytes

Table B-37. Encodings of AVX instructions

Instruction and Format	Encoding
VBLENDPD — Blend Packed Double-Precision Floats	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_3: w xmmreg2 001:0D:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1	C4: rxb0_3: w xmmreg2 001:0D:mod xmmreg1 r/m: imm
ymmreg2 with ymmreg3 into ymmreg1	C4: rxb0_3: w ymmreg2 101:0D:11 ymmreg1 ymmreg3: imm
ymmreg2 with mem to ymmreg1	C4: rxb0_3: w ymmreg2 101:0D:mod ymmreg1 r/m: imm
VBLENDPS — Blend Packed Single-Precision Floats	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_3: w xmmreg2 001:0C:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1	C4: rxb0_3: w xmmreg2 001:0C:mod xmmreg1 r/m: imm
ymmreg2 with ymmreg3 into ymmreg1	C4: rxb0_3: w ymmreg2 101:0C:11 ymmreg1 ymmreg3: imm
ymmreg2 with mem to ymmreg1	C4: rxb0_3: w ymmreg2 101:0C:mod ymmreg1 r/m: imm
VBLENDVPD — Variable Blend Packed Double-Precision Floats	
xmmreg2 with xmmreg3 into xmmreg1 using xmmreg4 as mask	C4: rxb0_3: 0 xmmreg2 001:4B:11 xmmreg1 xmmreg3: xmmreg4
xmmreg2 with mem to xmmreg1 using xmmreg4 as mask	C4: rxb0_3: 0 xmmreg2 001:4B:mod xmmreg1 r/m: xmmreg4
ymmreg2 with ymmreg3 into ymmreg1 using ymmreg4 as mask	C4: rxb0_3: 0 ymmreg2 101:4B:11 ymmreg1 ymmreg3: ymmreg4
ymmreg2 with mem to ymmreg1 using ymmreg4 as mask	C4: rxb0_3: 0 ymmreg2 101:4B:mod ymmreg1 r/m: ymmreg4
VBLENDVPS — Variable Blend Packed Single-Precision Floats	

Instruction and Format	Encoding
xmmreg2 with xmmreg3 into xmmreg1 using xmmreg4 as mask	C4: rxb0_3: 0 xmmreg2 001:4A:11 xmmreg1 xmmreg3: xmmreg4
xmmreg2 with mem to xmmreg1 using xmmreg4 as mask	C4: rxb0_3: 0 xmmreg2 001:4A:mod xmmreg1 r/m: xmmreg4
ymmreg2 with ymmreg3 into ymmreg1 using ymmreg4 as mask	C4: rxb0_3: 0 ymmreg2 101:4A:11 ymmreg1 ymmreg3: ymmreg4
ymmreg2 with mem to ymmreg1 using ymmreg4 as mask	C4: rxb0_3: 0 ymmreg2 101:4A:mod ymmreg1 r/m: ymmreg4
VDPPD — Packed Double-Precision Dot Products	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_3: w xmmreg2 001:41:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1	C4: rxb0_3: w xmmreg2 001:41:mod xmmreg1 r/m: imm
VDPPS — Packed Single-Precision Dot Products	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_3: w xmmreg2 001:40:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1	C4: rxb0_3: w xmmreg2 001:40:mod xmmreg1 r/m: imm
ymmreg2 with ymmreg3 into ymmreg1	C4: rxb0_3: w ymmreg2 101:40:11 ymmreg1 ymmreg3: imm
ymmreg2 with mem to ymmreg1	C4: rxb0_3: w ymmreg2 101:40:mod ymmreg1 r/m: imm
VEXTRACTPS — Extract From Packed Single-Precision Floats	
reg from xmmreg1 using imm	C4: rxb0_3: w_F 001:17:11 xmmreg1 reg: imm
mem from xmmreg1 using imm	C4: rxb0_3: w_F 001:17:mod xmmreg1 r/m: imm
VINSERTPS — Insert Into Packed Single-Precision Floats	
use imm to merge xmmreg3 with xmmreg2 into xmmreg1	C4: rxb0_3: w xmmreg2 001:21:11 xmmreg1 xmmreg3: imm
use imm to merge mem with xmmreg2 into xmmreg1	C4: rxb0_3: w xmmreg2 001:21:mod xmmreg1 r/m: imm
VMOVNTDQA — Load Double Quadword Non-temporal Aligned	
m128 to xmmreg1	C4: rxb0_2: w_F 001:2A:11 xmmreg1 r/m
VMPSADBW — Multiple Packed Sums of Absolute Difference	
xmmreg3 with xmmreg2 into xmmreg1	C4: rxb0_3: w xmmreg2 001:42:11 xmmreg1 xmmreg3: imm
m128 with xmmreg2 into xmmreg1	C4: rxb0_3: w xmmreg2 001:42:mod xmmreg1 r/m: imm
VPACKUSDW — Pack with Unsigned Saturation	
xmmreg3 and xmmreg2 to xmmreg1	C4: rxb0_2: w xmmreg2 001:2B:11 xmmreg1 xmmreg3: imm
m128 and xmmreg2 to xmmreg1	C4: rxb0_2: w xmmreg2 001:2B:mod xmmreg1 r/m: imm
VPBLENDVB — Variable Blend Packed Bytes	
xmmreg2 with xmmreg3 into xmmreg1 using xmmreg4 as mask	C4: rxb0_3: w xmmreg2 001:4C:11 xmmreg1 xmmreg3: xmmreg4
xmmreg2 with mem to xmmreg1 using xmmreg4 as mask	C4: rxb0_3: w xmmreg2 001:4C:mod xmmreg1 r/m: xmmreg4
VPBLENDW — Blend Packed Words	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_3: w xmmreg2 001:0E:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1	C4: rxb0_3: w xmmreg2 001:0E:mod xmmreg1 r/m: imm
VPCMPEQQ — Compare Packed Qword Data of Equal	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_2: w xmmreg2 001:29:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:29:mod xmmreg1 r/m:

Instruction and Format	Encoding
VPEXTRB — Extract Byte	
reg from xmmreg1 using imm	C4: rxb0_3: 0_F 001:14:11 xmmreg1 reg: imm
mem from xmmreg1 using imm	C4: rxb0_3: 0_F 001:14:mod xmmreg1 r/m: imm
VPEXTRD — Extract DWord	
reg from xmmreg1 using imm	C4: rxb0_3: 0_F 001:16:11 xmmreg1 reg: imm
mem from xmmreg1 using imm	C4: rxb0_3: 0_F 001:16:mod xmmreg1 r/m: imm
VPEXTRQ — Extract QWord	
reg from xmmreg1 using imm	C4: rxb0_3: 1_F 001:16:11 xmmreg1 reg: imm
mem from xmmreg1 using imm	C4: rxb0_3: 1_F 001:16:mod xmmreg1 r/m: imm
VPEXTRW — Extract Word	
reg from xmmreg1 using imm	C4: rxb0_3: 0_F 001:15:11 xmmreg1 reg: imm
mem from xmmreg1 using imm	C4: rxb0_3: 0_F 001:15:mod xmmreg1 r/m: imm
VPHMINPOSUW — Packed Horizontal Word Minimum	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:41:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:41:mod xmmreg1 r/m
VPINSRB — Insert Byte	
reg with xmmreg2 to xmmreg1, imm8	C4: rxb0_3: 0 xmmreg2 001:20:11 xmmreg1 reg: imm
mem with xmmreg2 to xmmreg1, imm8	C4: rxb0_3: 0 xmmreg2 001:20:mod xmmreg1 r/m: imm
VPINSRD — Insert DWord	
reg with xmmreg2 to xmmreg1, imm8	C4: rxb0_3: 0 xmmreg2 001:22:11 xmmreg1 reg: imm
mem with xmmreg2 to xmmreg1, imm8	C4: rxb0_3: 0 xmmreg2 001:22:mod xmmreg1 r/m: imm
VPINSRQ — Insert QWord	
r64 with xmmreg2 to xmmreg1, imm8	C4: rxb0_3: 1 xmmreg2 001:22:11 xmmreg1 reg: imm
m64 with xmmreg2 to xmmreg1, imm8	C4: rxb0_3: 1 xmmreg2 001:22:mod xmmreg1 r/m: imm
VPMAXSB — Maximum of Packed Signed Byte Integers	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_2: w xmmreg2 001:3C:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:3C:mod xmmreg1 r/m
VPMAXSD — Maximum of Packed Signed Dword Integers	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_2: w xmmreg2 001:3D:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:3D:mod xmmreg1 r/m
VPMAXUD — Maximum of Packed Unsigned Dword Integers	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_2: w xmmreg2 001:3F:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:3F:mod xmmreg1 r/m
VPMAXUW — Maximum of Packed Unsigned Word Integers	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_2: w xmmreg2 001:3E:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:3E:mod xmmreg1 r/m
VPMINSB — Minimum of Packed Signed Byte Integers	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_2: w xmmreg2 001:38:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:38:mod xmmreg1 r/m

Instruction and Format	Encoding
VPMINSD — Minimum of Packed Signed Dword Integers	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_2: w xmmreg2 001:39:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:39:mod xmmreg1 r/m
VPMINUD — Minimum of Packed Unsigned Dword Integers	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_2: w xmmreg2 001:3B:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:3B:mod xmmreg1 r/m
VPMINUW — Minimum of Packed Unsigned Word Integers	
xmmreg2 with xmmreg3 into xmmreg1	C4: rxb0_2: w xmmreg2 001:3A:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:3A:mod xmmreg1 r/m
VPMOVSXBD — Packed Move Sign Extend - Byte to Dword	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:21:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:21:mod xmmreg1 r/m
VPMOVSXBQ — Packed Move Sign Extend - Byte to Qword	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:22:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:22:mod xmmreg1 r/m
VPMOVSXBW — Packed Move Sign Extend - Byte to Word	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:20:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:20:mod xmmreg1 r/m
VPMOVSXWD — Packed Move Sign Extend - Word to Dword	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:23:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:23:mod xmmreg1 r/m
VPMOVSXWQ — Packed Move Sign Extend - Word to Qword	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:24:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:24:mod xmmreg1 r/m
VPMOVSXDQ — Packed Move Sign Extend - Dword to Qword	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:25:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:25:mod xmmreg1 r/m
VPMOVZXBD — Packed Move Zero Extend - Byte to Dword	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:31:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:31:mod xmmreg1 r/m
VPMOVZXBQ — Packed Move Zero Extend - Byte to Qword	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:32:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:32:mod xmmreg1 r/m
VPMOVZXBW — Packed Move Zero Extend - Byte to Word	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:30:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:30:mod xmmreg1 r/m
VPMOVZXWD — Packed Move Zero Extend - Word to Dword	

Instruction and Format	Encoding
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:33:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:33:mod xmmreg1 r/m
VPMOVZXWQ — Packed Move Zero Extend - Word to Qword	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:34:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:34:mod xmmreg1 r/m
VPMOVZXDQ — Packed Move Zero Extend - Dword to Qword	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:35:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:35:mod xmmreg1 r/m
VPMULDQ — Multiply Packed Signed Dword Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:28:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:28:mod xmmreg1 r/m
VPMULLD — Multiply Packed Signed Dword Integers, Store low Result	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:40:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:40:mod xmmreg1 r/m
VPTEST — Logical Compare	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:17:11 xmmreg1 xmmreg2
mem to xmmreg	C4: rxb0_2: w_F 001:17:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_2: w_F 101:17:11 ymmreg1 ymmreg2
mem to ymmreg	C4: rxb0_2: w_F 101:17:mod ymmreg1 r/m
VROUNDPD — Round Packed Double-Precision Values	
xmmreg2 to xmmreg1, imm8	C4: rxb0_3: w_F 001:09:11 xmmreg1 xmmreg2: imm
mem to xmmreg1, imm8	C4: rxb0_3: w_F 001:09:mod xmmreg1 r/m: imm
ymmreg2 to ymmreg1, imm8	C4: rxb0_3: w_F 101:09:11 ymmreg1 ymmreg2: imm
mem to ymmreg1, imm8	C4: rxb0_3: w_F 101:09:mod ymmreg1 r/m: imm
VROUNDPS — Round Packed Single-Precision Values	
xmmreg2 to xmmreg1, imm8	C4: rxb0_3: w_F 001:08:11 xmmreg1 xmmreg2: imm
mem to xmmreg1, imm8	C4: rxb0_3: w_F 001:08:mod xmmreg1 r/m: imm
ymmreg2 to ymmreg1, imm8	C4: rxb0_3: w_F 101:08:11 ymmreg1 ymmreg2: imm
mem to ymmreg1, imm8	C4: rxb0_3: w_F 101:08:mod ymmreg1 r/m: imm
VROUNDSD — Round Scalar Double-Precision Value	
xmmreg2 and xmmreg3 to xmmreg1, imm8	C4: rxb0_3: w xmmreg2 001:0B:11 xmmreg1 xmmreg3: imm
xmmreg2 and mem to xmmreg1, imm8	C4: rxb0_3: w xmmreg2 001:0B:mod xmmreg1 r/m: imm
VROUNDSS — Round Scalar Single-Precision Value	
xmmreg2 and xmmreg3 to xmmreg1, imm8	C4: rxb0_3: w xmmreg2 001:0A:11 xmmreg1 xmmreg3: imm
xmmreg2 and mem to xmmreg1, imm8	C4: rxb0_3: w xmmreg2 001:0A:mod xmmreg1 r/m: imm

Instruction and Format	Encoding
VPCMPESTRI — Packed Compare Explicit Length Strings, Return Index	
xmmreg2 with xmmreg1, imm8	C4: rxb0_3: w_F 001:61:11 xmmreg1 xmmreg2: imm
mem with xmmreg1, imm8	C4: rxb0_3: w_F 001:61:mod xmmreg1 r/m: imm
VPCMPESTRM — Packed Compare Explicit Length Strings, Return Mask	
xmmreg2 with xmmreg1, imm8	C4: rxb0_3: w_F 001:60:11 xmmreg1 xmmreg2: imm
mem with xmmreg1, imm8	C4: rxb0_3: w_F 001:60:mod xmmreg1 r/m: imm
VPCMPGTQ — Compare Packed Data for Greater Than	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:28:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:28:mod xmmreg1 r/m
VPCMPISTRI — Packed Compare Implicit Length Strings, Return Index	
xmmreg2 with xmmreg1, imm8	C4: rxb0_3: w_F 001:63:11 xmmreg1 xmmreg2: imm
mem with xmmreg1, imm8	C4: rxb0_3: w_F 001:63:mod xmmreg1 r/m: imm
VPCMPISTRM — Packed Compare Implicit Length Strings, Return Mask	
xmmreg2 with xmmreg1, imm8	C4: rxb0_3: w_F 001:62:11 xmmreg1 xmmreg2: imm
mem with xmmreg, imm8	C4: rxb0_3: w_F 001:62:mod xmmreg1 r/m: imm
VAESDEC — Perform One Round of an AES Decryption Flow	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:DE:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:DE:mod xmmreg1 r/m
VAESDECLAST — Perform Last Round of an AES Decryption Flow	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:DF:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:DF:mod xmmreg1 r/m
VAESENC — Perform One Round of an AES Encryption Flow	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:DC:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:DC:mod xmmreg1 r/m
VAESENCLAST — Perform Last Round of an AES Encryption Flow	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:DD:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:DD:mod xmmreg1 r/m
VAESIMC — Perform the AES InvMixColumn Transformation	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:DB:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:DB:mod xmmreg1 r/m
VAESKEYGENASSIST — AES Round Key Generation Assist	
xmmreg2 to xmmreg1, imm8	C4: rxb0_3: w_F 001:DF:11 xmmreg1 xmmreg2: imm
mem to xmmreg, imm8	C4: rxb0_3: w_F 001:DF:mod xmmreg1 r/m: imm
VPABSB — Packed Absolute Value	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:1C:11 xmmreg1 xmmreg2

Instruction and Format	Encoding
mem to xmmreg1	C4: rxb0_2: w_F 001:1C:mod xmmreg1 r/m
VPABSD — Packed Absolute Value	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:1E:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:1E:mod xmmreg1 r/m
VPABSW — Packed Absolute Value	
xmmreg2 to xmmreg1	C4: rxb0_2: w_F 001:1D:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_2: w_F 001:1D:mod xmmreg1 r/m
VPALIGNR — Packed Align Right	
xmmreg2 with xmmreg3 to xmmreg1, imm8	C4: rxb0_3: w xmmreg2 001:DD:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1, imm8	C4: rxb0_3: w xmmreg2 001:DD:mod xmmreg1 r/m: imm
VPHADDD — Packed Horizontal Add	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:02:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:02:mod xmmreg1 r/m
VPHADDW — Packed Horizontal Add	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:01:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:01:mod xmmreg1 r/m
VPHADDSW — Packed Horizontal Add and Saturate	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:03:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:03:mod xmmreg1 r/m
VPHSUBD — Packed Horizontal Subtract	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:06:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:06:mod xmmreg1 r/m
VPHSUBW — Packed Horizontal Subtract	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:05:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:05:mod xmmreg1 r/m
VPHSUBSW — Packed Horizontal Subtract and Saturate	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:07:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:07:mod xmmreg1 r/m
VPMADDUBSW — Multiply and Add Packed Signed and Unsigned Bytes	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:04:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:04:mod xmmreg1 r/m
VPMULHRSW — Packed Multiply High with Round and Scale	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:0B:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:0B:mod xmmreg1 r/m
VPSHUFB — Packed Shuffle Bytes	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:00:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:00:mod xmmreg1 r/m
VPSIGNB — Packed SIGN	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:08:11 xmmreg1 xmmreg3

Instruction and Format	Encoding
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:08:mod xmmreg1 r/m
VPSIGND — Packed SIGN	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:0A:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:0A:mod xmmreg1 r/m
VPSIGNW — Packed SIGN	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: w xmmreg2 001:09:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: w xmmreg2 001:09:mod xmmreg1 r/m
VADDSUBPD — Packed Double-FP Add/Subtract	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:D0:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:D0:mod xmmreg1 r/m
xmmreglo2[1] with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:D0:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:D0:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:D0:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:D0:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:D0:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:D0:mod ymmreg1 r/m
VADDSUBPS — Packed Single-FP Add/Subtract	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:D0:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:D0:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:D0:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:D0:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 111:D0:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 111:D0:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 111:D0:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 111:D0:mod ymmreg1 r/m
VHADDPD — Packed Double-FP Horizontal Add	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:7C:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:7C:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:7C:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:7C:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:7C:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:7C:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:7C:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:7C:mod ymmreg1 r/m
VHADDPS — Packed Single-FP Horizontal Add	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:7C:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:7C:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:7C:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:7C:mod xmmreg1 r/m

Instruction and Format	Encoding
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 111:7C:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 111:7C:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 111:7C:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 111:7C:mod ymmreg1 r/m
VHSUBPD — Packed Double-FP Horizontal Subtract	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:7D:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:7D:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:7D:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:7D:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:7D:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:7D:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:7D:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:7D:mod ymmreg1 r/m
VHSUBPS — Packed Single-FP Horizontal Subtract	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:7D:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:7D:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:7D:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:7D:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 111:7D:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 111:7D:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 111:7D:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 111:7D:mod ymmreg1 r/m
VLDDQU — Load Unaligned Integer 128 Bits	
mem to xmmreg1	C4: rxb0_1: w_F 011:F0:mod xmmreg1 r/m
mem to xmmreg1	C5: r_F 011:F0:mod xmmreg1 r/m
mem to ymmreg1	C4: rxb0_1: w_F 111:F0:mod ymmreg1 r/m
mem to ymmreg1	C5: r_F 111:F0:mod ymmreg1 r/m
VMOVDDUP — Move One Double-FP and Duplicate	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 011:12:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 011:12:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 011:12:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 011:12:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 111:12:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 111:12:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_ F 111:12:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 111:12:mod ymmreg1 r/m
VMOVHLPS — Move Packed Single-Precision Floating-Point Values High to Low	
xmmreg2 and xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:12:11 xmmreg1 xmmreg3

Instruction and Format	Encoding
xmmreglo2 and xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:12:11 xmmreg1 xmmreglo3
VMOVSHDUP — Move Packed Single-FP High and Duplicate	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 010:16:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 010:16:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 010:16:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 010:16:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 110:16:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 110:16:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 110:16:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 110:16:mod ymmreg1 r/m
VMOVSLDUP — Move Packed Single-FP Low and Duplicate	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 010:12:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 010:12:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 010:12:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 010:12:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 110:12:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 110:12:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 110:12:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 110:12:mod ymmreg1 r/m
VADDPD — Add Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:58:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:58:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:58:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:58:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:58:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:58:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:58:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:58:mod ymmreg1 r/m
VADDSD — Add Scalar Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:58:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:58:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:58:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5 r_xmmreglo2 011:58:mod xmmreg1 r/m
VANDPD — Bitwise Logical AND of Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:54:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:54:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:54:11 xmmreg1 xmmreglo3

Instruction and Format	**Encoding**
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:54:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:54:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:54:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:54:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:54:mod ymmreg1 r/m
VANDNPD — Bitwise Logical AND NOT of Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:55:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:55:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:55:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:55:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:55:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:55:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:55:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:55:mod ymmreg1 r/m
VCMPPD — Compare Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:C2:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:C2:mod xmmreg1 r/m: imm
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:C2:11 xmmreg1 xmmreglo3: imm
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:C2:mod xmmreg1 r/m: imm
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:C2:11 ymmreg1 ymmreg3: imm
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:C2:mod ymmreg1 r/m: imm
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:C2:11 ymmreg1 ymmreglo3: imm
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:C2:mod ymmreg1 r/m: imm
VCMPSD — Compare Scalar Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:C2:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:C2:mod xmmreg1 r/m: imm
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:C2:11 xmmreg1 xmmreglo3: imm
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:C2:mod xmmreg1 r/m: imm
VCOMISD — Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 001:2F:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 001:2F:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 001:2F:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 001:2F:mod xmmreg1 r/m
VCVTDQ2PD— Convert Packed Dword Integers to Packed Double-Precision FP Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 010:E6:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 010:E6:mod xmmreg1 r/m

Instruction and Format	Encoding
xmmreglo to xmmreg1	C5: r_F 010:E6:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 010:E6:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 110:E6:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 110:E6:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 110:E6:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 110:E6:mod ymmreg1 r/m
VCVTDQ2PS— Convert Packed Dword Integers to Packed Single-Precision FP Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 000:5B:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 000:5B:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 000:5B:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 000:5B:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 100:5B:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 100:5B:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 100:5B:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 100:5B:mod ymmreg1 r/m
VCVTPD2DQ— Convert Packed Double-Precision FP Values to Packed Dword Integers	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 011:E6:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 011:E6:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 011:E6:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 011:E6:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 111:E6:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 111:E6:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 111:E6:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 111:E6:mod ymmreg1 r/m
VCVTPD2PS— Convert Packed Double-Precision FP Values to Packed Single-Precision FP Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 001:5A:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 001:5A:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 001:5A:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 001:5A:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 101:5A:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 101:5A:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 101:5A:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 101:5A:mod ymmreg1 r/m
VCVTPS2DQ— Convert Packed Single-Precision FP Values to Packed Dword Integers	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 001:5B:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 001:5B:mod xmmreg1 r/m

Instruction and Format	Encoding
xmmreglo to xmmreg1	C5: r_F 001:5B:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 001:5B:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 101:5B:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 101:5B:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 101:5B:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 101:5B:mod ymmreg1 r/m
VCVTPS2PD— Convert Packed Single-Precision FP Values to Packed Double-Precision FP Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 000:5A:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 000:5A:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 000:5A:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 000:5A:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 100:5A:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 100:5A:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 100:5A:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 100:5A:mod ymmreg1 r/m
VCVTSD2SI— Convert Scalar Double-Precision FP Value to Integer	
xmmreg1 to reg32	C4: rxb0_1: 0_F 011:2D:11 reg xmmreg1
mem to reg32	C4: rxb0_1: 0_F 011:2D:mod reg r/m
xmmreglo to reg32	C5: r_F 011:2D:11 reg xmmreglo
mem to reg32	C5: r_F 011:2D:mod reg r/m
ymmreg1 to reg64	C4: rxb0_1: 1_F 111:2D:11 reg ymmreg1
mem to reg64	C4: rxb0_1: 1_F 111:2D:mod reg r/m
VCVTSD2SS — Convert Scalar Double-Precision FP Value to Scalar Single-Precision FP Value	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:5A:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:5A:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:5A:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:5A:mod xmmreg1 r/m
VCVTSI2SD— Convert Dword Integer to Scalar Double-Precision FP Value	
xmmreg2 with reg to xmmreg1	C4: rxb0_1: 0 xmmreg2 011:2A:11 xmmreg1 reg
xmmreg2 with mem to xmmreg1	C4: rxb0_1: 0 xmmreg2 011:2A:mod xmmreg1 r/m
xmmreglo2 with reglo to xmmreg1	C5: r_xmmreglo2 011:2A:11 xmmreg1 reglo
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:2A:mod xmmreg1 r/m
ymmreg2 with reg to ymmreg1	C4: rxb0_1: 1 ymmreg2 111:2A:11 ymmreg1 reg
ymmreg2 with mem to ymmreg1	C4: rxb0_1: 1 ymmreg2 111:2A:mod ymmreg1 r/m
VCVTSS2SD — Convert Scalar Single-Precision FP Value to Scalar Double-Precision FP Value	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:5A:11 xmmreg1 xmmreg3

Instruction and Format	Encoding
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:5A:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:5A:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:5A:mod xmmreg1 r/m
VCVTTPD2DQ— Convert with Truncation Packed Double-Precision FP Values to Packed Dword Integers	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 001:E6:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 001:E6:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 001:E6:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 001:E6:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 101:E6:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 101:E6:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 101:E6:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 101:E6:mod ymmreg1 r/m
VCVTTPS2DQ— Convert with Truncation Packed Single-Precision FP Values to Packed Dword Integers	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 010:5B:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 010:5B:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 010:5B:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 010:5B:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 110:5B:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 110:5B:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 110:5B:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 110:5B:mod ymmreg1 r/m
VCVTTSD2SI— Convert with Truncation Scalar Double-Precision FP Value to Signed Integer	
xmmreg1 to reg32	C4: rxb0_1: 0_F 011:2C:11 reg xmmreg1
mem to reg32	C4: rxb0_1: 0_F 011:2C:mod reg r/m
xmmreglo to reg32	C5: r_F 011:2C:11 reg xmmreglo
mem to reg32	C5: r_F 011:2C:mod reg r/m
xmmreg1 to reg64	C4: rxb0_1: 1_F 011:2C:11 reg xmmreg1
mem to reg64	C4: rxb0_1: 1_F 011:2C:mod reg r/m
VDIVPD — Divide Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:5E:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:5E:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:5E:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:5E:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:5E:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:5E:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:5E:11 ymmreg1 ymmreglo3

Instruction and Format	Encoding
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:5E:mod ymmreg1 r/m
VDIVSD — Divide Scalar Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:5E:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:5E:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:5E:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:5E:mod xmmreg1 r/m
VMASKMOVDQU— Store Selected Bytes of Double Quadword	
xmmreg1 to mem; xmmreg2 as mask	C4: rxb0_1: w_F 001:F7:11 r/m xmmreg1: xmmreg2
xmmreg1 to mem; xmmreg2 as mask	C5: r_F 001:F7:11 r/m xmmreg1: xmmreg2
VMAXPD — Return Maximum Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:5F:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:5F:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:5F:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:5F:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:5F:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:5F:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:5F:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:5F:mod ymmreg1 r/m
VMAXSD — Return Maximum Scalar Double-Precision Floating-Point Value	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:5F:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:5F:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:5F:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:5F:mod xmmreg1 r/m
VMINPD — Return Minimum Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:5D:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:5D:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:5D:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:5D:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:5D:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:5D:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:5D:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:5D:mod ymmreg1 r/m
VMINSD — Return Minimum Scalar Double-Precision Floating-Point Value	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:5D:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:5D:mod xmmreg1 r/m

Instruction and Format	Encoding
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:5D:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:5D:mod xmmreg1 r/m
VMOVAPD — Move Aligned Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 001:28:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 001:28:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 001:28:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 001:28:mod xmmreg1 r/m
xmmreg1 to xmmreg2	C4: rxb0_1: w_F 001:29:11 xmmreg2 xmmreg1
xmmreg1 to mem	C4: rxb0_1: w_F 001:29:mod r/m xmmreg1
xmmreg1 to xmmreglo	C5: r_F 001:29:11 xmmreglo xmmreg1
xmmreg1 to mem	C5: r_F 001:29:mod r/m xmmreg1
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 101:28:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 101:28:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 101:28:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 101:28:mod ymmreg1 r/m
ymmreg1 to ymmreg2	C4: rxb0_1: w_F 101:29:11 ymmreg2 ymmreg1
ymmreg1 to mem	C4: rxb0_1: w_F 101:29:mod r/m ymmreg1
ymmreg1 to ymmreglo	C5: r_F 101:29:11 ymmreglo ymmreg1
ymmreg1 to mem	C5: r_F 101:29:mod r/m ymmreg1
VMOVD — Move Doubleword	
reg32 to xmmreg1	C4: rxb0_1: 0_F 001:6E:11 xmmreg1 reg32
mem32 to xmmreg1	C4: rxb0_1: 0_F 001:6E:mod xmmreg1 r/m
reg32 to xmmreg1	C5: r_F 001:6E:11 xmmreg1 reg32
mem32 to xmmreg1	C5: r_F 001:6E:mod xmmreg1 r/m
xmmreg1 to reg32	C4: rxb0_1: 0_F 001:7E:11 reg32 xmmreg1
xmmreg1 to mem32	C4: rxb0_1: 0_F 001:7E:mod mem32 xmmreg1
xmmreglo to reg32	C5: r_F 001:7E:11 reg32 xmmreglo
xmmreglo to mem32	C5: r_F 001:7E:mod mem32 xmmreglo
VMOVQ — Move Quadword	
reg64 to xmmreg1	C4: rxb0_1: 1_F 001:6E:11 xmmreg1 reg64
mem64 to xmmreg1	C4: rxb0_1: 1_F 001:6E:mod xmmreg1 r/m
xmmreg1 to reg64	C4: rxb0_1: 1_F 001:7E:11 reg64 xmmreg1
xmmreg1 to mem64	C4: rxb0_1: 1_F 001:7E:mod r/m xmmreg1
VMOVDQA — Move Aligned Double Quadword	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 001:6F:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 001:6F:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 001:6F:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 001:6F:mod xmmreg1 r/m

Instruction and Format	Encoding
xmmreg1 to xmmreg2	C4: rxb0_1: w_F 001:7F:11 xmmreg2 xmmreg1
xmmreg1 to mem	C4: rxb0_1: w_F 001:7F:mod r/m xmmreg1
xmmreg1 to xmmreglo	C5: r_F 001:7F:11 xmmreglo xmmreg1
xmmreg1 to mem	C5: r_F 001:7F:mod r/m xmmreg1
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 101:6F:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 101:6F:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 101:6F:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 101:6F:mod ymmreg1 r/m
ymmreg1 to ymmreg2	C4: rxb0_1: w_F 101:7F:11 ymmreg2 ymmreg1
ymmreg1 to mem	C4: rxb0_1: w_F 101:7F:mod r/m ymmreg1
ymmreg1 to ymmreglo	C5: r_F 101:7F:11 ymmreglo ymmreg1
ymmreg1 to mem	C5: r_F 101:7F:mod r/m ymmreg1
VMOVDQU — Move Unaligned Double Quadword	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 010:6F:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 010:6F:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 010:6F:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 010:6F:mod xmmreg1 r/m
xmmreg1 to xmmreg2	C4: rxb0_1: w_F 010:7F:11 xmmreg2 xmmreg1
xmmreg1 to mem	C4: rxb0_1: w_F 010:7F:mod r/m xmmreg1
xmmreg1 to xmmreglo	C5: r_F 010:7F:11 xmmreglo xmmreg1
xmmreg1 to mem	C5: r_F 010:7F:mod r/m xmmreg1
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 110:6F:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 110:6F:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 110:6F:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 110:6F:mod ymmreg1 r/m
ymmreg1 to ymmreg2	C4: rxb0_1: w_F 110:7F:11 ymmreg2 ymmreg1
ymmreg1 to mem	C4: rxb0_1: w_F 110:7F:mod r/m ymmreg1
ymmreg1 to ymmreglo	C5: r_F 110:7F:11 ymmreglo ymmreg1
ymmreg1 to mem	C5: r_F 110:7F:mod r/m ymmreg1
VMOVHPD — Move High Packed Double-Precision Floating-Point Value	
xmmreg1 and mem to xmmreg2	C4: rxb0_1: w xmmreg1 001:16:11 xmmreg2 r/m
xmmreg1 and mem to xmmreglo2	C5: r_xmmreg1 001:16:11 xmmreglo2 r/m
xmmreg1 to mem	C4: rxb0_1: w_F 001:17:mod r/m xmmreg1
xmmreglo to mem	C5: r_F 001:17:mod r/m xmmreglo
VMOVLPD — Move Low Packed Double-Precision Floating-Point Value	
xmmreg1 and mem to xmmreg2	C4: rxb0_1: w xmmreg1 001:12:11 xmmreg2 r/m
xmmreg1 and mem to xmmreglo2	C5: r_xmmreg1 001:12:11 xmmreglo2 r/m
xmmreg1 to mem	C4: rxb0_1: w_F 001:13:mod r/m xmmreg1

Instruction and Format	Encoding
xmmreglo to mem	C5: r_F 001:13:mod r/m xmmreglo
VMOVMSKPD — Extract Packed Double-Precision Floating-Point Sign Mask	
xmmreg2 to reg	C4: rxb0_1: w_F 001:50:11 reg xmmreg1
xmmreglo to reg	C5: r_F 001:50:11 reg xmmreglo
ymmreg2 to reg	C4: rxb0_1: w_F 101:50:11 reg ymmreg1
ymmreglo to reg	C5: r_F 101:50:11 reg ymmreglo
VMOVNTDQ — Store Double Quadword Using Non-Temporal Hint	
xmmreg1 to mem	C4: rxb0_1: w_F 001:E7:11 r/m xmmreg1
xmmreglo to mem	C5: r_F 001:E7:11 r/m xmmreglo
ymmreg1 to mem	C4: rxb0_1: w_F 101:E7:11 r/m ymmreg1
ymmreglo to mem	C5: r_F 101:E7:11 r/m ymmreglo
VMOVNTPD — Store Packed Double-Precision Floating-Point Values Using Non-Temporal Hint	
xmmreg1 to mem	C4: rxb0_1: w_F 001:2B:11 r/m xmmreg1
xmmreglo to mem	C5: r_F 001:2B:11 r/m xmmreglo
ymmreg1 to mem	C4: rxb0_1: w_F 101:2B:11r/m ymmreg1
ymmreglo to mem	C5: r_F 101:2B:11r/m ymmreglo
VMOVSD — Move Scalar Double-Precision Floating-Point Value	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:10:11 xmmreg1 xmmreg3
mem to xmmreg1	C4: rxb0_1: w_F 011:10:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:10:11 xmmreg1 xmmreglo3
mem to xmmreg1	C5: r_F 011:10:mod xmmreg1 r/m
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:11:11 xmmreg1 xmmreg3
xmmreg1 to mem	C4: rxb0_1: w_F 011:11:mod r/m xmmreg1
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:11:11 xmmreg1 xmmreglo3
xmmreglo to mem	C5: r_F 011:11:mod r/m xmmreglo
VMOVUPD — Move Unaligned Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 001:10:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 001:10:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 001:10:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 001:10:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 101:10:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 101:10:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 101:10:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 101:10:mod ymmreg1 r/m
xmmreg1 to xmmreg2	C4: rxb0_1: w_F 001:11:11 xmmreg2 xmmreg1
xmmreg1 to mem	C4: rxb0_1: w_F 001:11:mod r/m xmmreg1

Instruction and Format	Encoding
xmmreg1 to xmmreglo	C5: r_F 001:11:11 xmmreglo xmmreg1
xmmreg1 to mem	C5: r_F 001:11:mod r/m xmmreg1
ymmreg1 to ymmreg2	C4: rxb0_1: w_F 101:11:11 ymmreg2 ymmreg1
ymmreg1 to mem	C4: rxb0_1: w_F 101:11:mod r/m ymmreg1
ymmreg1 to ymmreglo	C5: r_F 101:11:11 ymmreglo ymmreg1
ymmreg1 to mem	C5: r_F 101:11:mod r/m ymmreg1
VMULPD — Multiply Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:59:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:59:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:59:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:59:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:59:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:59:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:59:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:59:mod ymmreg1 r/m
VMULSD — Multiply Scalar Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:59:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:59:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:59:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:59:mod xmmreg1 r/m
VORPD — Bitwise Logical OR of Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:56:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:56:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:56:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:56:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:56:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:56:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:56:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:56:mod ymmreg1 r/m
VPACKSSWB— Pack with Signed Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:63:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:63:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:63:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:63:mod xmmreg1 r/m
VPACKSSDW— Pack with Signed Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:6B:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:6B:mod xmmreg1 r/m

Instruction and Format	Encoding
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:6B:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:6B:mod xmmreg1 r/m
VPACKUSWB— Pack with Unsigned Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:67:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:67:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:67:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:67:mod xmmreg1 r/m
VPADDB — Add Packed Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:FC:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:FC:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:FC:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:FC:mod xmmreg1 r/m
VPADDW — Add Packed Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:FD:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:FD:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:FD:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:FD:mod xmmreg1 r/m
VPADDD — Add Packed Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:FE:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:FE:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:FE:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:FE:mod xmmreg1 r/m
VPADDQ — Add Packed Quadword Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:D4:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:D4:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:D4:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:D4:mod xmmreg1 r/m
VPADDSB — Add Packed Signed Integers with Signed Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:EC:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:EC:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:EC:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:EC:mod xmmreg1 r/m
VPADDSW — Add Packed Signed Integers with Signed Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:ED:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:ED:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:ED:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:ED:mod xmmreg1 r/m

Instruction and Format	Encoding
VPADDUSB — Add Packed Unsigned Integers with Unsigned Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:DC:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:DC:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:DC:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:DC:mod xmmreg1 r/m
VPADDUSW — Add Packed Unsigned Integers with Unsigned Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:DD:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:DD:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:DD:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:DD:mod xmmreg1 r/m
VPAND — Logical AND	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:DB:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:DB:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:DB:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:DB:mod xmmreg1 r/m
VPANDN — Logical AND NOT	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:DF:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:DF:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:DF:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:DF:mod xmmreg1 r/m
VPAVGB — Average Packed Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:E0:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:E0:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:E0:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:E0:mod xmmreg1 r/m
VPAVGW — Average Packed Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:E3:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:E3:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:E3:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:E3:mod xmmreg1 r/m
VPCMPEQB — Compare Packed Data for Equal	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:74:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:74:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:74:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:74:mod xmmreg1 r/m
VPCMPEQW — Compare Packed Data for Equal	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:75:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:75:mod xmmreg1 r/m

Instruction and Format	Encoding
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:75:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:75:mod xmmreg1 r/m
VPCMPEQD — Compare Packed Data for Equal	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:76:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:76:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:76:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:76:mod xmmreg1 r/m
VPCMPGTB — Compare Packed Signed Integers for Greater Than	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:64:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:64:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:64:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:64:mod xmmreg1 r/m
VPCMPGTW — Compare Packed Signed Integers for Greater Than	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:65:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:65:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:65:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:65:mod xmmreg1 r/m
VPCMPGTD — Compare Packed Signed Integers for Greater Than	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:66:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:66:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:66:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:66:mod xmmreg1 r/m
VPEXTRW — Extract Word	
xmmreg1 to reg using imm	C4: rxb0_1: 0_F 001:C5:11 reg xmmreg1: imm
xmmreg1 to reg using imm	C5: r_F 001:C5:11 reg xmmreg1: imm
VPINSRW — Insert Word	
xmmreg2 with reg to xmmreg1	C4: rxb0_1: 0 xmmreg2 001:C4:11 xmmreg1 reg: imm
xmmreg2 with mem to xmmreg1	C4: rxb0_1: 0 xmmreg2 001:C4:mod xmmreg1 r/m: imm
xmmreglo2 with reglo to xmmreg1	C5: r_xmmreglo2 001:C4:11 xmmreg1 reglo: imm
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:C4:mod xmmreg1 r/m: imm
VPMADDWD — Multiply and Add Packed Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:F5:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:F5:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:F5:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:F5:mod xmmreg1 r/m
VPMAXSW — Maximum of Packed Signed Word Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:EE:11 xmmreg1 xmmreg3

Instruction and Format	Encoding
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:EE:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:EE:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:EE:mod xmmreg1 r/m
VPMAXUB — Maximum of Packed Unsigned Byte Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:DE:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:DE:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:DE:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:DE:mod xmmreg1 r/m
VPMINSW — Minimum of Packed Signed Word Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:EA:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:EA:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:EA:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:EA:mod xmmreg1 r/m
VPMINUB — Minimum of Packed Unsigned Byte Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:DA:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:DA:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:DA:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:DA:mod xmmreg1 r/m
VPMOVMSKB — Move Byte Mask	
xmmreg1 to reg	C4: rxb0_1: w_F 001:D7:11 reg xmmreg1
xmmreg1 to reg	C5: r_F 001:D7:11 reg xmmreg1
VPMULHUW — Multiply Packed Unsigned Integers and Store High Result	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:E4:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:E4:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:E4:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:E4:mod xmmreg1 r/m
VPMULHW — Multiply Packed Signed Integers and Store High Result	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:E5:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:E5:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:E5:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:E5:mod xmmreg1 r/m
VPMULLW — Multiply Packed Signed Integers and Store Low Result	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:D5:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:D5:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:D5:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:D5:mod xmmreg1 r/m

Instruction and Format	Encoding
VPMULUDQ — Multiply Packed Unsigned Doubleword Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:F4:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:F4:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:F4:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:F4:mod xmmreg1 r/m
VPOR — Bitwise Logical OR	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:EB:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:EB:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:EB:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:EB:mod xmmreg1 r/m
VPSADBW — Compute Sum of Absolute Differences	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:F6:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:F6:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:F6:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:F6:mod xmmreg1 r/m
VPSHUFD — Shuffle Packed Doublewords	
xmmreg2 to xmmreg1 using imm	C4: rxb0_1: w_F 001:70:11 xmmreg1 xmmreg2: imm
mem to xmmreg1 using imm	C4: rxb0_1: w_F 001:70:mod xmmreg1 r/m: imm
xmmreglo to xmmreg1 using imm	C5: r_F 001:70:11 xmmreg1 xmmreglo: imm
mem to xmmreg1 using imm	C5: r_F 001:70:mod xmmreg1 r/m: imm
VPSHUFHW — Shuffle Packed High Words	
xmmreg2 to xmmreg1 using imm	C4: rxb0_1: w_F 010:70:11 xmmreg1 xmmreg2: imm
mem to xmmreg1 using imm	C4: rxb0_1: w_F 010:70:mod xmmreg1 r/m: imm
xmmreglo to xmmreg1 using imm	C5: r_F 010:70:11 xmmreg1 xmmreglo: imm
mem to xmmreg1 using imm	C5: r_F 010:70:mod xmmreg1 r/m: imm
VPSHUFLW — Shuffle Packed Low Words	
xmmreg2 to xmmreg1 using imm	C4: rxb0_1: w_F 011:70:11 xmmreg1 xmmreg2: imm
mem to xmmreg1 using imm	C4: rxb0_1: w_F 011:70:mod xmmreg1 r/m: imm
xmmreglo to xmmreg1 using imm	C5: r_F 011:70:11 xmmreg1 xmmreglo: imm
mem to xmmreg1 using imm	C5: r_F 011:70:mod xmmreg1 r/m: imm
VPSLLDQ — Shift Double Quadword Left Logical	
xmmreg2 to xmmreg1 using imm	C4: rxb0_1: w_F 001:73:11 xmmreg1 xmmreg2: imm
xmmreglo to xmmreg1 using imm	C5: r_F 001:73:11 xmmreg1 xmmreglo: imm
VPSLLW — Shift Packed Data Left Logical	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:F1:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:F1:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:F1:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:F1:mod xmmreg1 r/m

Instruction and Format	Encoding
xmmreg2 to xmmreg1 using imm8	C4: rxb0_1: w_F 001:71:11 xmmreg1 xmmreg2: imm
xmmreglo to xmmreg1 using imm8	C5: r_F 001:71:11 xmmreg1 xmmreglo: imm
VPSLLD — Shift Packed Data Left Logical	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:F2:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:F2:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:F2:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:F2:mod xmmreg1 r/m
xmmreg2 to xmmreg1 using imm8	C4: rxb0_1: w_F 001:72:11 xmmreg1 xmmreg2: imm
xmmreglo to xmmreg1 using imm8	C5: r_F 001:72:11 xmmreg1 xmmreglo: imm
VPSLLQ — Shift Packed Data Left Logical	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:F3:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:F3:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:F3:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:F3:mod xmmreg1 r/m
xmmreg2 to xmmreg1 using imm8	C4: rxb0_1: w_F 001:73:11 xmmreg1 xmmreg2: imm
xmmreglo to xmmreg1 using imm8	C5: r_F 001:73:11 xmmreg1 xmmreglo: imm
VPSRAW — Shift Packed Data Right Arithmetic	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:E1:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:E1:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:E1:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:E1:mod xmmreg1 r/m
xmmreg2 to xmmreg1 using imm8	C4: rxb0_1: w_F 001:71:11 xmmreg1 xmmreg2: imm
xmmreglo to xmmreg1 using imm8	C5: r_F 001:71:11 xmmreg1 xmmreglo: imm
VPSRAD — Shift Packed Data Right Arithmetic	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:E2:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:E2:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:E2:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:E2:mod xmmreg1 r/m
xmmreg2 to xmmreg1 using imm8	C4: rxb0_1: w_F 001:72:11 xmmreg1 xmmreg2: imm
xmmreglo to xmmreg1 using imm8	C5: r_F 001:72:11 xmmreg1 xmmreglo: imm
VPSRLDQ — Shift Double Quadword Right Logical	
xmmreg2 to xmmreg1 using imm8	C4: rxb0_1: w_F 001:73:11 xmmreg1 xmmreg2: imm
xmmreglo to xmmreg1 using imm8	C5: r_F 001:73:11 xmmreg1 xmmreglo: imm
VPSRLW — Shift Packed Data Right Logical	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:D1:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:D1:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:D1:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:D1:mod xmmreg1 r/m
xmmreg2 to xmmreg1 using imm8	C4: rxb0_1: w_F 001:71:11 xmmreg1 xmmreg2: imm

Instruction and Format	Encoding
xmmreglo to xmmreg1 using imm8	C5: r_F 001:71:11 xmmreg1 xmmreglo: imm
VPSRLD — Shift Packed Data Right Logical	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:D2:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:D2:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:D2:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:D2:mod xmmreg1 r/m
xmmreg2 to xmmreg1 using imm8	C4: rxb0_1: w_F 001:72:11 xmmreg1 xmmreg2: imm
xmmreglo to xmmreg1 using imm8	C5: r_F 001:72:11 xmmreg1 xmmreglo: imm
VPSRLQ — Shift Packed Data Right Logical	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:D3:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:D3:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:D3:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:D3:mod xmmreg1 r/m
xmmreg2 to xmmreg1 using imm8	C4: rxb0_1: w_F 001:73:11 xmmreg1 xmmreg2: imm
xmmreglo to xmmreg1 using imm8	C5: r_F 001:73:11 xmmreg1 xmmreglo: imm
VPSUBB — Subtract Packed Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:F8:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:F8:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:F8:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:F8:mod xmmreg1 r/m
VPSUBW — Subtract Packed Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:F9:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:F9:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:F9:11 xmmreg1 xmmreglo3
xmmrelog2 with mem to xmmreg1	C5: r_xmmreglo2 001:F9:mod xmmreg1 r/m
VPSUBD — Subtract Packed Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:FA:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:FA:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:FA:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:FA:mod xmmreg1 r/m
VPSUBQ — Subtract Packed Quadword Integers	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:FB:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:FB:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:FB:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:FB:mod xmmreg1 r/m
VPSUBSB — Subtract Packed Signed Integers with Signed Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:E8:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:E8:mod xmmreg1 r/m

Instruction and Format	Encoding
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:E8:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:E8:mod xmmreg1 r/m
VPSUBSW — Subtract Packed Signed Integers with Signed Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:E9:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:E9:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:E9:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:E9:mod xmmreg1 r/m
VPSUBUSB — Subtract Packed Unsigned Integers with Unsigned Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:D8:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:D8:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:D8:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:D8:mod xmmreg1 r/m
VPSUBUSW — Subtract Packed Unsigned Integers with Unsigned Saturation	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:D9:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:D9:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:D9:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:D9:mod xmmreg1 r/m
VPUNPCKHBW — Unpack High Data	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:68:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:68:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:68:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:68:mod xmmreg1 r/m
VPUNPCKHWD — Unpack High Data	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:69:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:69:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:69:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:69:mod xmmreg1 r/m
VPUNPCKHDQ — Unpack High Data	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:6A:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:6A:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:6A:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:6A:mod xmmreg1 r/m
VPUNPCKHQDQ — Unpack High Data	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:6D:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:6D:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:6D:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:6D:mod xmmreg1 r/m

Instruction and Format	Encoding
VPUNPCKLBW — Unpack Low Data	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:60:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:60:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:60:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:60:mod xmmreg1 r/m
VPUNPCKLWD — Unpack Low Data	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:61:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:61:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:61:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:61:mod xmmreg1 r/m
VPUNPCKLDQ — Unpack Low Data	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:62:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:62:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:62:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:62:mod xmmreg1 r/m
VPUNPCKLQDQ — Unpack Low Data	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:6C:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:6C:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:6C:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:6C:mod xmmreg1 r/m
VPXOR — Logical Exclusive OR	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:EF:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:EF:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:EF:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:EF:mod xmmreg1 r/m
VSHUFPD — Shuffle Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1 using imm8	C4: rxb0_1: w xmmreg2 001:C6:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1 using imm8	C4: rxb0_1: w xmmreg2 001:C6:mod xmmreg1 r/m: imm
xmmreglo2 with xmmreglo3 to xmmreg1 using imm8	C5: r_xmmreglo2 001:C6:11 xmmreg1 xmmreglo3: imm
xmmreglo2 with mem to xmmreg1 using imm8	C5: r_xmmreglo2 001:C6:mod xmmreg1 r/m: imm
ymmreg2 with ymmreg3 to ymmreg1 using imm8	C4: rxb0_1: w ymmreg2 101:C6:11 ymmreg1 ymmreg3: imm
ymmreg2 with mem to ymmreg1 using imm8	C4: rxb0_1: w ymmreg2 101:C6:mod ymmreg1 r/m: imm
ymmreglo2 with ymmreglo3 to ymmreg1 using imm8	C5: r_ymmreglo2 101:C6:11 ymmreg1 ymmreglo3: imm
ymmreglo2 with mem to ymmreg1 using imm8	C5: r_ymmreglo2 101:C6:mod ymmreg1 r/m: imm
VSQRTPD — Compute Square Roots of Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 001:51:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 001:51:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 001:51:11 xmmreg1 xmmreglo

Instruction and Format	Encoding
mem to xmmreg1	C5: r_F 001:51:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 101:51:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 101:51:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 101:51:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 101:51:mod ymmreg1 r/m
VSQRTSD — Compute Square Root of Scalar Double-Precision Floating-Point Value	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:51:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:51:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:51:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:51:mod xmmreg1 r/m
VSUBPD — Subtract Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:5C:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:5C:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:5C:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:5C:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:5C:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:5C:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:5C:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:5C:mod ymmreg1 r/m
VSUBSD — Subtract Scalar Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 011:5C:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 011:5C:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 011:5C:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 011:5C:mod xmmreg1 r/m
VUCOMISD — Unordered Compare Scalar Double-Precision Floating-Point Values and Set EFLAGS	
xmmreg2 with xmmreg1, set EFLAGS	C4: rxb0_1: w_F xmmreg1 001:2E:11 xmmreg2
mem with xmmreg1, set EFLAGS	C4: rxb0_1: w_F xmmreg1 001:2E:mod r/m
xmmreglo with xmmreg1, set EFLAGS	C5: r_F xmmreg1 001:2E:11 xmmreglo
mem with xmmreg1, set EFLAGS	C5: r_F xmmreg1 001:2E:mod r/m
VUNPCKHPD — Unpack and Interleave High Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:15:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:15:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:15:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:15:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:15:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:15:mod ymmreg1 r/m

Instruction and Format	Encoding
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:15:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:15:mod ymmreg1 r/m
VUNPCKHPS — Unpack and Interleave High Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:15:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:15:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:15:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:15:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:15:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:15:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:15:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:15:mod ymmreg1 r/m
VUNPCKLPD — Unpack and Interleave Low Packed Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:14:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:14:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:14:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:14:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:14:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:14:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:14:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:14:mod ymmreg1 r/m
VUNPCKLPS — Unpack and Interleave Low Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:14:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:14:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:14:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:14:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:14:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:14:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:14:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:14:mod ymmreg1 r/m
VXORPD — Bitwise Logical XOR for Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 001:57:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 001:57:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 001:57:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 001:57:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 101:57:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 101:57:mod ymmreg1 r/m

Instruction and Format	Encoding
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 101:57:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 101:57:mod ymmreg1 r/m
VADDPS — Add Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:58:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:58:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:58:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:58:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:58:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:58:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:58:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:58:mod ymmreg1 r/m
VADDSS — Add Scalar Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:58:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:58:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:58:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:58:mod xmmreg1 r/m
VANDPS — Bitwise Logical AND of Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:54:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:54:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:54:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:54:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:54:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:54:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:54:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:54:mod ymmreg1 r/m
VANDNPS — Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:55:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:55:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:55:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:55:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:55:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:55:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:55:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:55:mod ymmreg1 r/m
VCMPPS — Compare Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:C2:11 xmmreg1 xmmreg3: imm

Instruction and Format	Encoding
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:C2:mod xmmreg1 r/m: imm
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:C2:11 xmmreg1 xmmreglo3: imm
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:C2:mod xmmreg1 r/m: imm
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:C2:11 ymmreg1 ymmreg3: imm
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:C2:mod ymmreg1 r/m: imm
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:C2:11 ymmreg1 ymmreglo3: imm
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:C2:mod ymmreg1 r/m: imm
VCMPSS — Compare Scalar Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:C2:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:C2:mod xmmreg1 r/m: imm
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:C2:11 xmmreg1 xmmreglo3: imm
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:C2:mod xmmreg1 r/m: imm
VCOMISS — Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS	
xmmreg2 with xmmreg1	C4: rxb0_1: w_F 000:2F:11 xmmreg1 xmmreg2
mem with xmmreg1	C4: rxb0_1: w_F 000:2F:mod xmmreg1 r/m
xmmreglo with xmmreg1	C5: r_F 000:2F:11 xmmreg1 xmmreglo
mem with xmmreg1	C5: r_F 000:2F:mod xmmreg1 r/m
VCVTSI2SS — Convert Dword Integer to Scalar Single-Precision FP Value	
xmmreg2 with reg to xmmreg1	C4: rxb0_1: 0 xmmreg2 010:2A:11 xmmreg1 reg
xmmreg2 with mem to xmmreg1	C4: rxb0_1: 0 xmmreg2 010:2A:mod xmmreg1 r/m
xmmreglo2 with reglo to xmmreg1	C5: r_xmmreglo2 010:2A:11 xmmreg1 reglo
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:2A:mod xmmreg1 r/m
xmmreg2 with reg to xmmreg1	C4: rxb0_1: 1 xmmreg2 010:2A:11 xmmreg1 reg
xmmreg2 with mem to xmmreg1	C4: rxb0_1: 1 xmmreg2 010:2A:mod xmmreg1 r/m
VCVTSS2SI — Convert Scalar Single-Precision FP Value to Dword Integer	
xmmreg1 to reg	C4: rxb0_1: 0_F 010:2D:11 reg xmmreg1
mem to reg	C4: rxb0_1: 0_F 010:2D:mod reg r/m
xmmreglo to reg	C5: r_F 010:2D:11 reg xmmreglo
mem to reg	C5: r_F 010:2D:mod reg r/m
xmmreg1 to reg	C4: rxb0_1: 1_F 010:2D:11 reg xmmreg1
mem to reg	C4: rxb0_1: 1_F 010:2D:mod reg r/m
VCVTTSS2SI — Convert with Truncation Scalar Single-Precision FP Value to Dword Integer	
xmmreg1 to reg	C4: rxb0_1: 0_F 010:2C:11 reg xmmreg1
mem to reg	C4: rxb0_1: 0_F 010:2C:mod reg r/m
xmmreglo to reg	C5: r_F 010:2C:11 reg xmmreglo
mem to reg	C5: r_F 010:2C:mod reg r/m

Instruction and Format	Encoding
xmmreg1 to reg	C4: rxb0_1: 1_F 010:2C:11 reg xmmreg1
mem to reg	C4: rxb0_1: 1_F 010:2C:mod reg r/m
VDIVPS — Divide Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:5E:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:5E:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:5E:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:5E:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:5E:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:5E:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:5E:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:5E:mod ymmreg1 r/m
VDIVSS — Divide Scalar Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:5E:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:5E:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:5E:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:5E:mod xmmreg1 r/m
VLDMXCSR — Load MXCSR Register	
mem to MXCSR reg	C4: rxb0_1: w_F 000:AEmod 011 r/m
mem to MXCSR reg	C5: r_F 000:AEmod 011 r/m
VMAXPS — Return Maximum Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:5F:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:5F:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:5F:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:5F:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:5F:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:5F:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:5F:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:5F:mod ymmreg1 r/m
VMAXSS — Return Maximum Scalar Single-Precision Floating-Point Value	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:5F:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:5F:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:5F:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:5F:mod xmmreg1 r/m
VMINPS — Return Minimum Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:5D:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:5D:mod xmmreg1 r/m

Instruction and Format	Encoding
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:5D:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:5D:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:5D:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:5D:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:5D:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:5D:mod ymmreg1 r/m
VMINSS — Return Minimum Scalar Single-Precision Floating-Point Value	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:5D:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:5D:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:5D:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:5D:mod xmmreg1 r/m
VMOVAPS— Move Aligned Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 000:28:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 000:28:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 000:28:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 000:28:mod xmmreg1 r/m
xmmreg1 to xmmreg2	C4: rxb0_1: w_F 000:29:11 xmmreg2 xmmreg1
xmmreg1 to mem	C4: rxb0_1: w_F 000:29:mod r/m xmmreg1
xmmreg1 to xmmreglo	C5: r_F 000:29:11 xmmreglo xmmreg1
xmmreg1 to mem	C5: r_F 000:29:mod r/m xmmreg1
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 100:28:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 100:28:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 100:28:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 100:28:mod ymmreg1 r/m
ymmreg1 to ymmreg2	C4: rxb0_1: w_F 100:29:11 ymmreg2 ymmreg1
ymmreg1 to mem	C4: rxb0_1: w_F 100:29:mod r/m ymmreg1
ymmreg1 to ymmreglo	C5: r_F 100:29:11 ymmreglo ymmreg1
ymmreg1 to mem	C5: r_F 100:29:mod r/m ymmreg1
VMOVHPS — Move High Packed Single-Precision Floating-Point Values	
xmmreg1 with mem to xmmreg2	C4: rxb0_1: w xmmreg1 000:16:mod xmmreg2 r/m
xmmreg1 with mem to xmmreglo2	C5: r_xmmreg1 000:16:mod xmmreglo2 r/m
xmmreg1 to mem	C4: rxb0_1: w_F 000:17:mod r/m xmmreg1
xmmreglo to mem	C5: r_F 000:17:mod r/m xmmreglo
VMOVLHPS — Move Packed Single-Precision Floating-Point Values Low to High	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:16:11 xmmreg1 xmmreg3
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:16:11 xmmreg1 xmmreglo3

Instruction and Format	Encoding
VMOVLPS — Move Low Packed Single-Precision Floating-Point Values	
xmmreg1 with mem to xmmreg2	C4: rxb0_1: w xmmreg1 000:12:mod xmmreg2 r/m
xmmreg1 with mem to xmmreglo2	C5: r_xmmreg1 000:12:mod xmmreglo2 r/m
xmmreg1 to mem	C4: rxb0_1: w_F 000:13:mod r/m xmmreg1
xmmreglo to mem	C5: r_F 000:13:mod r/m xmmreglo
VMOVMSKPS — Extract Packed Single-Precision Floating-Point Sign Mask	
xmmreg2 to reg	C4: rxb0_1: w_F 000:50:11 reg xmmreg2
xmmreglo to reg	C5: r_F 000:50:11 reg xmmreglo
ymmreg2 to reg	C4: rxb0_1: w_F 100:50:11 reg ymmreg2
ymmreglo to reg	C5: r_F 100:50:11 reg ymmreglo
VMOVNTPS — Store Packed Single-Precision Floating-Point Values Using Non-Temporal Hint	
xmmreg1 to mem	C4: rxb0_1: w_F 000:2B:mod r/m xmmreg1
xmmreglo to mem	C5: r_F 000:2B:mod r/m xmmreglo
ymmreg1 to mem	C4: rxb0_1: w_F 100:2B:mod r/m ymmreg1
ymmreglo to mem	C5: r_F 100:2B:mod r/m ymmreglo
VMOVSS — Move Scalar Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:10:11 xmmreg1 xmmreg3
mem to xmmreg1	C4: rxb0_1: w_F 010:10:mod xmmreg1 r/m
xmmreg2 with xmmreg3 to xmmreg1	C5: r_xmmreg2 010:10:11 xmmreg1 xmmreg3
mem to xmmreg1	C5: r_F 010:10:mod xmmreg1 r/m
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:11:11 xmmreg1 xmmreg3
xmmreg1 to mem	C4: rxb0_1: w_F 010:11:mod r/m xmmreg1
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:11:11 xmmreg1 xmmreglo3
xmmreglo to mem	C5: r_F 010:11:mod r/m xmmreglo
VMOVUPS— Move Unaligned Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 000:10:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 000:10:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 000:10:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 000:10:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 100:10:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 100:10:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 100:10:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 100:10:mod ymmreg1 r/m
xmmreg1 to xmmreg2	C4: rxb0_1: w_F 000:11:11 xmmreg2 xmmreg1
xmmreg1 to mem	C4: rxb0_1: w_F 000:11:mod r/m xmmreg1
xmmreg1 to xmmreglo	C5: r_F 000:11:11 xmmreglo xmmreg1

Instruction and Format	Encoding
xmmreg1 to mem	C5: r_F 000:11:mod r/m xmmreg1
ymmreg1 to ymmreg2	C4: rxb0_1: w_F 100:11:11 ymmreg2 ymmreg1
ymmreg1 to mem	C4: rxb0_1: w_F 100:11:mod r/m ymmreg1
ymmreg1 to ymmreglo	C5: r_F 100:11:11 ymmreglo ymmreg1
ymmreg1 to mem	C5: r_F 100:11:mod r/m ymmreg1
VMULPS — Multiply Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:59:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:59:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:59:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:59:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:59:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:59:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:59:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:59:mod ymmreg1 r/m
VMULSS — Multiply Scalar Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:59:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:59:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:59:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:59:mod xmmreg1 r/m
VORPS — Bitwise Logical OR of Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:56:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:56:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:56:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:56:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:56:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:56:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:56:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:56:mod ymmreg1 r/m
VRCPPS — Compute Reciprocals of Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 000:53:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 000:53:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 000:53:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 000:53:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 100:53:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 100:53:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 100:53:11 ymmreg1 ymmreglo

Instruction and Format	Encoding
mem to ymmreg1	C5: r_F 100:53:mod ymmreg1 r/m
VRCPSS — Compute Reciprocal of Scalar Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:53:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:53:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:53:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:53:mod xmmreg1 r/m
VRSQRTPS — Compute Reciprocals of Square Roots of Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 000:52:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 000:52:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 000:52:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 000:52:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 100:52:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 100:52:mod ymmreg1 r/m
ymmreglo to ymmreg1	C5: r_F 100:52:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 100:52:mod ymmreg1 r/m
VRSQRTSS — Compute Reciprocal of Square Root of Scalar Single-Precision Floating-Point Value	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:52:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:52:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:52:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:52:mod xmmreg1 r/m
VSHUFPS — Shuffle Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1, imm8	C4: rxb0_1: w xmmreg2 000:C6:11 xmmreg1 xmmreg3: imm
xmmreg2 with mem to xmmreg1, imm8	C4: rxb0_1: w xmmreg2 000:C6:mod xmmreg1 r/m: imm
xmmreglo2 with xmmreglo3 to xmmreg1, imm8	C5: r_xmmreglo2 000:C6:11 xmmreg1 xmmreglo3: imm
xmmreglo2 with mem to xmmreg1, imm8	C5: r_xmmreglo2 000:C6:mod xmmreg1 r/m: imm
ymmreg2 with ymmreg3 to ymmreg1, imm8	C4: rxb0_1: w ymmreg2 100:C6:11 ymmreg1 ymmreg3: imm
ymmreg2 with mem to ymmreg1, imm8	C4: rxb0_1: w ymmreg2 100:C6:mod ymmreg1 r/m: imm
ymmreglo2 with ymmreglo3 to ymmreg1, imm8	C5: r_ymmreglo2 100:C6:11 ymmreg1 ymmreglo3: imm
ymmreglo2 with mem to ymmreg1, imm8	C5: r_ymmreglo2 100:C6:mod ymmreg1 r/m: imm
VSQRTPS — Compute Square Roots of Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	C4: rxb0_1: w_F 000:51:11 xmmreg1 xmmreg2
mem to xmmreg1	C4: rxb0_1: w_F 000:51:mod xmmreg1 r/m
xmmreglo to xmmreg1	C5: r_F 000:51:11 xmmreg1 xmmreglo
mem to xmmreg1	C5: r_F 000:51:mod xmmreg1 r/m
ymmreg2 to ymmreg1	C4: rxb0_1: w_F 100:51:11 ymmreg1 ymmreg2
mem to ymmreg1	C4: rxb0_1: w_F 100:51:mod ymmreg1 r/m

Instruction and Format	Encoding
ymmreglo to ymmreg1	C5: r_F 100:51:11 ymmreg1 ymmreglo
mem to ymmreg1	C5: r_F 100:51:mod ymmreg1 r/m
VSQRTSS — Compute Square Root of Scalar Single-Precision Floating-Point Value	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:51:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:51:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:51:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:51:mod xmmreg1 r/m
VSTMXCSR — Store MXCSR Register State	
MXCSR to mem	C4: rxb0_1: w_F 000:AE:mod 011 r/m
MXCSR to mem	C5: r_F 000:AE:mod 011 r/m
VSUBPS — Subtract Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:5C:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:5C:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:5C:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:5C:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:5C:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:5C:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:5C:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:5C:mod ymmreg1 r/m
VSUBSS — Subtract Scalar Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 010:5C:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 010:5C:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 010:5C:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 010:5C:mod xmmreg1 r/m
VUCOMISS — Unordered Compare Scalar Single-Precision Floating-Point Values and Set EFLAGS	
xmmreg2 with xmmreg1	C4: rxb0_1: w_F 000:2E:11 xmmreg1 xmmreg2
mem with xmmreg1	C4: rxb0_1: w_F 000:2E:mod xmmreg1 r/m
xmmreglo with xmmreg1	C5: r_F 000:2E:11 xmmreg1 xmmreglo
mem with xmmreg1	C5: r_F 000:2E:mod xmmreg1 r/m
UNPCKHPS — Unpack and Interleave High Packed Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:15:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:15mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:15:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:15mod ymmreg1 r/m
UNPCKLPS — Unpack and Interleave Low Packed Single-Precision Floating-Point Value	

Instruction and Format	Encoding
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:14:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:14mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:14:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:14mod ymmreg1 r/m
VXORPS — Bitwise Logical XOR for Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_1: w xmmreg2 000:57:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_1: w xmmreg2 000:57:mod xmmreg1 r/m
xmmreglo2 with xmmreglo3 to xmmreg1	C5: r_xmmreglo2 000:57:11 xmmreg1 xmmreglo3
xmmreglo2 with mem to xmmreg1	C5: r_xmmreglo2 000:57:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_1: w ymmreg2 100:57:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_1: w ymmreg2 100:57:mod ymmreg1 r/m
ymmreglo2 with ymmreglo3 to ymmreg1	C5: r_ymmreglo2 100:57:11 ymmreg1 ymmreglo3
ymmreglo2 with mem to ymmreg1	C5: r_ymmreglo2 100:57:mod ymmreg1 r/m
VBROADCAST —Load with Broadcast	
mem to xmmreg1	C4: rxb0_2: 0_F 001:18:mod xmmreg1 r/m
mem to ymmreg1	C4: rxb0_2: 0_F 101:18:mod ymmreg1 r/m
mem to ymmreg1	C4: rxb0_2: 0_F 101:19:mod ymmreg1 r/m
mem to ymmreg1	C4: rxb0_2: 0_F 101:1A:mod ymmreg1 r/m
VEXTRACTF128 — Extract Packed Floating-Point Values	
ymmreg2 to xmmreg1, imm8	C4: rxb0_3: 0_F 001:19:11 xmmreg1 ymmreg2: imm
ymmreg2 to mem, imm8	C4: rxb0_3: 0_F 001:19:mod r/m ymmreg2: imm
VINSERTF128 — Insert Packed Floating-Point Values	
xmmreg3 and merge with ymmreg2 to ymmreg1, imm8	C4: rxb0_3: 0 ymmreg2101:18:11 ymmreg1 xmmreg3: imm
mem and merge with ymmreg2 to ymmreg1, imm8	C4: rxb0_3: 0 ymmreg2 101:18:mod ymmreg1 r/m: imm
VPERMILPD — Permute Double-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: 0 xmmreg2 001:0D:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: 0 xmmreg2 001:0D:mod xmmreg1 r/m
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_2: 0 ymmreg2 101:0D:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_2: 0 ymmreg2 101:0D:mod ymmreg1 r/m
xmmreg2 to xmmreg1, imm	C4: rxb0_3: 0_F 001:05:11 xmmreg1 xmmreg2: imm
mem to xmmreg1, imm	C4: rxb0_3: 0_F 001:05:mod xmmreg1 r/m: imm
ymmreg2 to ymmreg1, imm	C4: rxb0_3: 0_F 101:05:11 ymmreg1 ymmreg2: imm
mem to ymmreg1, imm	C4: rxb0_3: 0_F 101:05:mod ymmreg1 r/m: imm
VPERMILPS — Permute Single-Precision Floating-Point Values	
xmmreg2 with xmmreg3 to xmmreg1	C4: rxb0_2: 0 xmmreg2 001:0C:11 xmmreg1 xmmreg3
xmmreg2 with mem to xmmreg1	C4: rxb0_2: 0 xmmreg2 001:0C:mod xmmreg1 r/m
xmmreg2 to xmmreg1, imm	C4: rxb0_3: 0_F 001:04:11 xmmreg1 xmmreg2: imm

Instruction and Format	**Encoding**
mem to xmmreg1, imm	C4: rxb0_3: 0_F 001:04:mod xmmreg1 r/m: imm
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_2: 0 ymmreg2 101:0C:11 ymmreg1 ymmreg3
ymmreg2 with mem to ymmreg1	C4: rxb0_2: 0 ymmreg2 101:0C:mod ymmreg1 r/m
ymmreg2 to ymmreg1, imm	C4: rxb0_3: 0_F 101:04:11 ymmreg1 ymmreg2: imm
mem to ymmreg1, imm	C4: rxb0_3: 0_F 101:04:mod ymmreg1 r/m: imm
VPERM2F128 — Permute Floating-Point Values	
ymmreg2 with ymmreg3 to ymmreg1	C4: rxb0_3: 0 ymmreg2 101:06:11 ymmreg1 ymmreg3: imm
ymmreg2 with mem to ymmreg1	C4: rxb0_3: 0 ymmreg2 101:06:mod ymmreg1 r/m: imm
VTESTPD/VTESTPS — Packed Bit Test	
xmmreg2 to xmmreg1	C4: rxb0_2: 0_F 001:0E:11 xmmreg2 xmmreg1
mem to xmmreg1	C4: rxb0_2: 0_F 001:0E:mod xmmreg2 r/m
ymmreg2 to ymmreg1	C4: rxb0_2: 0_F 101:0E:11 ymmreg2 ymmreg1
mem to ymmreg1	C4: rxb0_2: 0_F 101:0E:mod ymmreg2 r/m
xmmreg2 to xmmreg1	C4: rxb0_2: 0_F 001:0F:11 xmmreg1 xmmreg2: imm
mem to xmmreg1	C4: rxb0_2: 0_F 001:0F:mod xmmreg1 r/m: imm
ymmreg2 to ymmreg1	C4: rxb0_2: 0_F 101:0F:11 ymmreg1 ymmreg2: imm
mem to ymmreg1	C4: rxb0_2: 0_F 101:0F:mod ymmreg1 r/m: imm

NOTES:

1. The term “lo” refers to the lower eight registers, 0-7

B.17 FLOATING-POINT INSTRUCTION FORMATS AND ENCODINGS

Table B-38 shows the five different formats used for floating-point instructions. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011.

Table B-38. General Floating-Point Instruction Formats

	Instruction										Optional Fields	
	First Byte				Second Byte							
1	11011	OPA		1	mod		1	OPB		r/m	s-i-b	disp
2	11011	MF		OPA	mod		OPB			r/m	s-i-b	disp
3	11011	d	P	OPA	1	1	OPB	R		ST(i)		
4	11011	0	0	1	1	1	1	OP				
5	11011	0	1	1	1	1	1	OP				
	15–11	10	9	8	7	6	5	4	3	2 1 0		

MF = Memory Format
- 00 — 32-bit real
- 01 — 32-bit integer
- 10 — 64-bit real
- 11 — 16-bit integer

P = Pop
- 0 — Do not pop stack
- 1 — Pop stack after operation

d = Destination
- 0 — Destination is ST(0)
- 1 — Destination is ST(i)

R XOR d = 0 — Destination OP Source
 R XOR d = 1 — Source OP Destination

ST(i) = Register stack element *i*
- 000 = Stack Top
- 001 = Second stack element
- .
- .
- .
- 111 = Eighth stack element

The Mod and R/M fields of the ModR/M byte have the same interpretation as the corresponding fields of the integer instructions. The SIB byte and disp (displacement) are optionally present in instructions that have Mod and R/M fields. Their presence depends on the values of Mod and R/M, as for integer instructions.

Table B-39 shows the formats and encodings of the floating-point instructions.

Table B-39. Floating-Point Instruction Formats and Encodings

Instruction and Format	Encoding
F2XM1 – Compute $2^{ST(0)}$ – 1	11011 001 : 1111 0000
FABS – Absolute Value	11011 001 : 1110 0001
FADD – Add	
ST(0) ← ST(0) + 32-bit memory	11011 000 : mod 000 r/m
ST(0) ← ST(0) + 64-bit memory	11011 100 : mod 000 r/m
ST(d) ← ST(0) + ST(i)	11011 d00 : 11 000 ST(i)
FADDP – Add and Pop	
ST(0) ← ST(0) + ST(i)	11011 110 : 11 000 ST(i)
FBLD – Load Binary Coded Decimal	11011 111 : mod 100 r/m
FBSTP – Store Binary Coded Decimal and Pop	11011 111 : mod 110 r/m
FCHS – Change Sign	11011 001 : 1110 0000
FCLEX – Clear Exceptions	11011 011 : 1110 0010
FCOM – Compare Real	

Table B-39. Floating-Point Instruction Formats and Encodings (Contd.)

Instruction and Format	Encoding
32-bit memory	11011 000 : mod 010 r/m
64-bit memory	11011 100 : mod 010 r/m
ST(i)	11011 000 : 11 010 ST(i)
FCOMP – Compare Real and Pop	
32-bit memory	11011 000 : mod 011 r/m
64-bit memory	11011 100 : mod 011 r/m
ST(i)	11011 000 : 11 011 ST(i)
FCOMPP – Compare Real and Pop Twice	11011 110 : 11 011 001
FCOMIP – Compare Real, Set EFLAGS, and Pop	11011 111 : 11 110 ST(i)
FCOS – Cosine of ST(0)	11011 001 : 1111 1111
FDECSTP – Decrement Stack-Top Pointer	11011 001 : 1111 0110
FDIV – Divide	
ST(0) ← ST(0) ÷ 32-bit memory	11011 000 : mod 110 r/m
ST(0) ← ST(0) ÷ 64-bit memory	11011 100 : mod 110 r/m
ST(d) ← ST(0) ÷ ST(i)	11011 d00 : 1111 R ST(i)
FDIVP – Divide and Pop	
ST(0) ← ST(0) ÷ ST(i)	11011 110 : 1111 1 ST(i)
FDIVR – Reverse Divide	
ST(0) ← 32-bit memory ÷ ST(0)	11011 000 : mod 111 r/m
ST(0) ← 64-bit memory ÷ ST(0)	11011 100 : mod 111 r/m
ST(d) ← ST(i) ÷ ST(0)	11011 d00 : 1111 R ST(i)
FDIVRP – Reverse Divide and Pop	
ST(0) ¨ ST(i) ÷ ST(0)	11011 110 : 1111 0 ST(i)
FFREE – Free ST(i) Register	11011 101 : 1100 0 ST(i)
FIADD – Add Integer	
ST(0) ← ST(0) + 16-bit memory	11011 110 : mod 000 r/m
ST(0) ← ST(0) + 32-bit memory	11011 010 : mod 000 r/m
FICOM – Compare Integer	
16-bit memory	11011 110 : mod 010 r/m
32-bit memory	11011 010 : mod 010 r/m
FICOMP – Compare Integer and Pop	
16-bit memory	11011 110 : mod 011 r/m
32-bit memory	11011 010 : mod 011 r/m
FIDIV – Divide	
ST(0) ← ST(0) 16-bit memory	11011 110 : mod 110 r/m
ST(0) ← ST(0) 32-bit memory	11011 010 : mod 110 r/m
FIDIVR – Reverse Divide	
ST(0) ← 16-bit memory ST(0)	11011 110 : mod 111 r/m

Table B-39. Floating-Point Instruction Formats and Encodings (Contd.)

Instruction and Format	Encoding
ST(0) ← 32-bit memory ST(0)	11011 010 : mod 111 r/m
FILD – Load Integer	
16-bit memory	11011 111 : mod 000 r/m
32-bit memory	11011 011 : mod 000 r/m
64-bit memory	11011 111 : mod 101 r/m
FIMUL– Multiply	
ST(0) ← ST(0) 16-bit memory	11011 110 : mod 001 r/m
ST(0) ← ST(0) 32-bit memory	11011 010 : mod 001 r/m
FINCSTP – Increment Stack Pointer	11011 001 : 1111 0111
FINIT – Initialize Floating-Point Unit	
FIST – Store Integer	
16-bit memory	11011 111 : mod 010 r/m
32-bit memory	11011 011 : mod 010 r/m
FISTP – Store Integer and Pop	
16-bit memory	11011 111 : mod 011 r/m
32-bit memory	11011 011 : mod 011 r/m
64-bit memory	11011 111 : mod 111 r/m
FISUB – Subtract	
ST(0) ← ST(0) - 16-bit memory	11011 110 : mod 100 r/m
ST(0) ← ST(0) - 32-bit memory	11011 010 : mod 100 r/m
FISUBR – Reverse Subtract	
ST(0) ← 16-bit memory – ST(0)	11011 110 : mod 101 r/m
ST(0) ← 32-bit memory – ST(0)	11011 010 : mod 101 r/m
FLD – Load Real	
32-bit memory	11011 001 : mod 000 r/m
64-bit memory	11011 101 : mod 000 r/m
80-bit memory	11011 011 : mod 101 r/m
ST(i)	11011 001 : 11 000 ST(i)
FLD1 – Load +1.0 into ST(0)	11011 001 : 1110 1000
FLDCW – Load Control Word	11011 001 : mod 101 r/m
FLDENV – Load FPU Environment	11011 001 : mod 100 r/m
FLDL2E – Load $\log_2(\varepsilon)$ into ST(0)	11011 001 : 1110 1010
FLDL2T – Load $\log_2(10)$ into ST(0)	11011 001 : 1110 1001
FLDLG2 – Load $\log_{10}(2)$ into ST(0)	11011 001 : 1110 1100
FLDLN2 – Load $\log_\varepsilon(2)$ into ST(0)	11011 001 : 1110 1101
FLDPI – Load π into ST(0)	11011 001 : 1110 1011
FLDZ – Load +0.0 into ST(0)	11011 001 : 1110 1110
FMUL – Multiply	

Table B-39. Floating-Point Instruction Formats and Encodings (Contd.)

Instruction and Format	Encoding
ST(0) ← ST(0) 32-bit memory	11011 000 : mod 001 r/m
ST(0) ← ST(0) 64-bit memory	11011 100 : mod 001 r/m
ST(d) ← ST(0) ST(i)	11011 d00 : 1100 1 ST(i)
FMULP – Multiply	
ST(i) ← ST(0) ST(i)	11011 110 : 1100 1 ST(i)
FNOP – No Operation	11011 001 : 1101 0000
FPATAN – Partial Arctangent	11011 001 : 1111 0011
FPREM – Partial Remainder	11011 001 : 1111 1000
FPREM1 – Partial Remainder (IEEE)	11011 001 : 1111 0101
FPTAN – Partial Tangent	11011 001 : 1111 0010
FRNDINT – Round to Integer	11011 001 : 1111 1100
FRSTOR – Restore FPU State	11011 101 : mod 100 r/m
FSAVE – Store FPU State	11011 101 : mod 110 r/m
FSCALE – Scale	11011 001 : 1111 1101
FSIN – Sine	11011 001 : 1111 1110
FSINCOS – Sine and Cosine	11011 001 : 1111 1011
FSQRT – Square Root	11011 001 : 1111 1010
FST – Store Real	
32-bit memory	11011 001 : mod 010 r/m
64-bit memory	11011 101 : mod 010 r/m
ST(i)	11011 101 : 11 010 ST(i)
FSTCW – Store Control Word	11011 001 : mod 111 r/m
FSTENV – Store FPU Environment	11011 001 : mod 110 r/m
FSTP – Store Real and Pop	
32-bit memory	11011 001 : mod 011 r/m
64-bit memory	11011 101 : mod 011 r/m
80-bit memory	11011 011 : mod 111 r/m
ST(i)	11011 101 : 11 011 ST(i)
FSTSW – Store Status Word into AX	11011 111 : 1110 0000
FSTSW – Store Status Word into Memory	11011 101 : mod 111 r/m
FSUB – Subtract	
ST(0) ← ST(0) – 32-bit memory	11011 000 : mod 100 r/m
ST(0) ← ST(0) – 64-bit memory	11011 100 : mod 100 r/m
ST(d) ← ST(0) – ST(i)	11011 d00 : 1110 R ST(i)
FSUBP – Subtract and Pop	
ST(0) ← ST(0) – ST(i)	11011 110 : 1110 1 ST(i)
FSUBR – Reverse Subtract	
ST(0) ← 32-bit memory – ST(0)	11011 000 : mod 101 r/m

Table B-39. Floating-Point Instruction Formats and Encodings (Contd.)

Instruction and Format	Encoding
ST(0) ← 64-bit memory – ST(0)	11011 100 : mod 101 r/m
ST(d) ← ST(i) – ST(0)	11011 d00 : 1110 R ST(i)
FSUBRP – Reverse Subtract and Pop	
ST(i) ← ST(i) – ST(0)	11011 110 : 1110 0 ST(i)
FTST – Test	11011 001 : 1110 0100
FUCOM – Unordered Compare Real	11011 101 : 1110 0 ST(i)
FUCOMP – Unordered Compare Real and Pop	11011 101 : 1110 1 ST(i)
FUCOMPP – Unordered Compare Real and Pop Twice	11011 010 : 1110 1001
FUCOMI – Unorderd Compare Real and Set EFLAGS	11011 011 : 11 101 ST(i)
FUCOMIP – Unorderd Compare Real, Set EFLAGS, and Pop	11011 111 : 11 101 ST(i)
FXAM – Examine	11011 001 : 1110 0101
FXCH – Exchange ST(0) and ST(i)	11011 001 : 1100 1 ST(i)
FXTRACT – Extract Exponent and Significand	11011 001 : 1111 0100
FYL2X – ST(1) $\log_2$(ST(0))	11011 001 : 1111 0001
FYL2XP1 – ST(1) $\log_2$(ST(0) + 1.0)	11011 001 : 1111 1001
FWAIT – Wait until FPU Ready	1001 1011 (same instruction as WAIT)

B.18 VMX INSTRUCTIONS

Table B-40 describes virtual-machine extensions (VMX).

Table B-40. Encodings for VMX Instructions

Instruction and Format	Encoding
INVEPT—Invalidate Cached EPT Mappings	
Descriptor m128 according to reg	01100110 00001111 00111000 10000000: mod reg r/m
INVVPID—Invalidate Cached VPID Mappings	
Descriptor m128 according to reg	01100110 00001111 00111000 10000001: mod reg r/m
VMCALL—Call to VM Monitor	
Call VMM: causes VM exit.	00001111 00000001 11000001
VMCLEAR—Clear Virtual-Machine Control Structure	
mem32:VMCS_data_ptr	01100110 00001111 11000111: mod 110 r/m
mem64:VMCS_data_ptr	01100110 00001111 11000111: mod 110 r/m
VMFUNC—Invoke VM Function	
Invoke VM function specified in EAX	00001111 00000001 11010100
VMLAUNCH—Launch Virtual Machine	
Launch VM managed by Current_VMCS	00001111 00000001 11000010
VMRESUME—Resume Virtual Machine	
Resume VM managed by Current_VMCS	00001111 00000001 11000011
VMPTRLD—Load Pointer to Virtual-Machine Control Structure	

Table B-40. Encodings for VMX Instructions

Instruction and Format	Encoding
mem32 to Current_VMCS_ptr	00001111 11000111: mod 110 r/m
mem64 to Current_VMCS_ptr	00001111 11000111: mod 110 r/m
VMPTRST—Store Pointer to Virtual-Machine Control Structure	
Current_VMCS_ptr to mem32	00001111 11000111: mod 111 r/m
Current_VMCS_ptr to mem64	00001111 11000111: mod 111 r/m
VMREAD—Read Field from Virtual-Machine Control Structure	
r32 *(VMCS_fieldn)* to r32	00001111 01111000: 11 reg2 reg1
r32 *(VMCS_fieldn)* to mem32	00001111 01111000: mod r32 r/m
r64 *(VMCS_fieldn)* to r64	00001111 01111000: 11 reg2 reg1
r64 *(VMCS_fieldn)* to mem64	00001111 01111000: mod r64 r/m
VMWRITE—Write Field to Virtual-Machine Control Structure	
r32 to r32 *(VMCS_fieldn)*	00001111 01111001: 11 reg1 reg2
mem32 to r32 *(VMCS_fieldn)*	00001111 01111001: mod r32 r/m
r64 to r64 *(VMCS_fieldn)*	00001111 01111001: 11 reg1 reg2
mem64 to r64 *(VMCS_fieldn)*	00001111 01111001: mod r64 r/m
VMXOFF—Leave VMX Operation	
Leave VMX.	00001111 00000001 11000100
VMXON—Enter VMX Operation	
Enter VMX.	11110011 000011111 11000111: mod 110 r/m

B.19 SMX INSTRUCTIONS

Table B-38 describes Safer Mode extensions (VMX). **GETSEC leaf functions are selected by a valid value in EAX on input.**

Table B-41. Encodings for SMX Instructions

Instruction and Format	Encoding
GETSEC—GETSEC leaf functions are selected by the value in EAX on input	
GETSEC[CAPABILITIES]	00001111 00110111 (EAX= 0)
GETSEC[ENTERACCS]	00001111 00110111 (EAX= 2)
GETSEC[EXITAC]	00001111 00110111 (EAX= 3)
GETSEC[SENTER]	00001111 00110111 (EAX= 4)
GETSEC[SEXIT]	00001111 00110111 (EAX= 5)
GETSEC[PARAMETERS]	00001111 00110111 (EAX= 6)
GETSEC[SMCTRL]	00001111 00110111 (EAX= 7)
GETSEC[WAKEUP]	00001111 00110111 (EAX= 8)

APPENDIX C
INTEL® C/C++ COMPILER INTRINSICS AND FUNCTIONAL EQUIVALENTS

The two tables in this appendix itemize the Intel C/C++ compiler intrinsics and functional equivalents for the Intel MMX technology, SSE, SSE2, SSE3, and SSSE3 instructions.

There may be additional intrinsics that do not have an instruction equivalent. It is strongly recommended that the reader reference the compiler documentation for the complete list of supported intrinsics. Please refer to http://www.intel.com/support/performancetools/.

Table C-1 presents simple intrinsics and Table C-2 presents composite intrinsics. Some intrinsics are "composites" because they require more than one instruction to implement them.

Intel C/C++ Compiler intrinsic names reflect the following naming conventions:

```
_mm_<intrin_op>_<suffix>
```

where:

<intrin_op>	Indicates the intrinsics basic operation; for example, add for addition and sub for subtraction
<suffix>	Denotes the type of data operated on by the instruction. The first one or two letters of each suffix denotes whether the data is packed (p), extended packed (ep), or scalar (s).

The remaining letters denote the type:

s	single-precision floating point
d	double-precision floating point
i128	signed 128-bit integer
i64	signed 64-bit integer
u64	unsigned 64-bit integer
i32	signed 32-bit integer
u32	unsigned 32-bit integer
i16	signed 16-bit integer
u16	unsigned 16-bit integer
i8	signed 8-bit integer
u8	unsigned 8-bit integer

The variable r is generally used for the intrinsic's return value. A number appended to a variable name indicates the element of a packed object. For example, r0 is the lowest word of r.

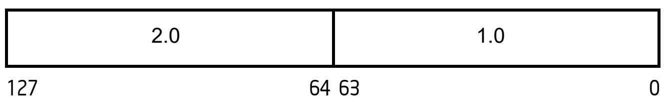
The packed values are represented in right-to-left order, with the lowest value being used for scalar operations. Consider the following example operation:

```
double a[2] = {1.0, 2.0};
__m128d t = _mm_load_pd(a);
```

The result is the same as either of the following:

```
__m128d t = _mm_set_pd(2.0, 1.0);
__m128d t = _mm_setr_pd(1.0, 2.0);
```

In other words, the XMM register that holds the value t will look as follows:



The “scalar” element is 1.0. Due to the nature of the instruction, some intrinsics require their arguments to be immediates (constant integer literals).

To use an intrinsic in your code, insert a line with the following syntax:

```
data_type intrinsic_name (parameters)
```

Where:

data_type	Is the return data type, which can be either void, int, __m64, __m128, __m128d, or __m128i. Only the _mm_empty intrinsic returns void.
intrinsic_name	Is the name of the intrinsic, which behaves like a function that you can use in your C/C++ code instead of in-lining the actual instruction.
parameters	Represents the parameters required by each intrinsic.

C.1 SIMPLE INTRINSICS

NOTE

For detailed descriptions of the intrinsics in Table C-1, see the corresponding mnemonic in Chapter 3, “Instruction Set Reference, A-L” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*, Chapter 4, “Instruction Set Reference, M-U” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*, or Chapter 5, “Instruction Set Reference, V-Z,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2C*.

Table C-1. Simple Intrinsics

Mnemonic	Intrinsic
ADDPD	__m128d _mm_add_pd(__m128d a, __m128d b)
ADDPS	__m128 _mm_add_ps(__m128 a, __m128 b)
ADDSD	__m128d _mm_add_sd(__m128d a, __m128d b)
ADDSS	__m128 _mm_add_ss(__m128 a, __m128 b)
ADDSUBPD	__m128d _mm_addsub_pd(__m128d a, __m128d b)
ADDSUBPS	__m128 _mm_addsub_ps(__m128 a, __m128 b)
AESDEC	__m128i _mm_aesdec (__m128i, __m128i)
AESDECLAST	__m128i _mm_aesdeclast (__m128i, __m128i)
AESENC	__m128i _mm_aesenc (__m128i, __m128i)
AESENCLAST	__m128i _mm_aesenclast (__m128i, __m128i)
AESIMC	__m128i _mm_aesimc (__m128i)
AESKEYGENASSIST	__m128i _mm_aesimc (__m128i, const int)
ANDNPD	__m128d _mm_andnot_pd(__m128d a, __m128d b)
ANDNPS	__m128 _mm_andnot_ps(__m128 a, __m128 b)
ANDPD	__m128d _mm_and_pd(__m128d a, __m128d b)
ANDPS	__m128 _mm_and_ps(__m128 a, __m128 b)
BLENDPD	__m128d _mm_blend_pd(__m128d v1, __m128d v2, const int mask)
BLENDPS	__m128 _mm_blend_ps(__m128 v1, __m128 v2, const int mask)
BLENDVPD	__m128d _mm_blendv_pd(__m128d v1, __m128d v2, __m128d v3)
BLENDVPS	__m128 _mm_blendv_ps(__m128 v1, __m128 v2, __m128 v3)
CLFLUSH	void _mm_clflush(void const *p)
CMPPD	__m128d _mm_cmpeq_pd(__m128d a, __m128d b)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
	__m128d _mm_cmplt_pd(__m128d a, __m128d b)
	__m128d _mm_cmple_pd(__m128d a, __m128d b)
	__m128d _mm_cmpgt_pd(__m128d a, __m128d b)
	__m128d _mm_cmpge_pd(__m128d a, __m128d b)
	__m128d _mm_cmpneq_pd(__m128d a, __m128d b)
	__m128d _mm_cmpnlt_pd(__m128d a, __m128d b)
	__m128d _mm_cmpngt_pd(__m128d a, __m128d b)
	__m128d _mm_cmpnge_pd(__m128d a, __m128d b)
	__m128d _mm_cmpord_pd(__m128d a, __m128d b)
	__m128d _mm_cmpunord_pd(__m128d a, __m128d b)
	__m128d _mm_cmpnle_pd(__m128d a, __m128d b)
CMPPS	__m128 _mm_cmpeq_ps(__m128 a, __m128 b)
	__m128 _mm_cmplt_ps(__m128 a, __m128 b)
	__m128 _mm_cmple_ps(__m128 a, __m128 b)
	__m128 _mm_cmpgt_ps(__m128 a, __m128 b)
	__m128 _mm_cmpge_ps(__m128 a, __m128 b)
	__m128 _mm_cmpneq_ps(__m128 a, __m128 b)
	__m128 _mm_cmpnlt_ps(__m128 a, __m128 b)
	__m128 _mm_cmpngt_ps(__m128 a, __m128 b)
	__m128 _mm_cmpnge_ps(__m128 a, __m128 b)
	__m128 _mm_cmpord_ps(__m128 a, __m128 b)
	__m128 _mm_cmpunord_ps(__m128 a, __m128 b)
	__m128 _mm_cmpnle_ps(__m128 a, __m128 b)
CMPSD	__m128d _mm_cmpeq_sd(__m128d a, __m128d b)
	__m128d _mm_cmplt_sd(__m128d a, __m128d b)
	__m128d _mm_cmple_sd(__m128d a, __m128d b)
	__m128d _mm_cmpgt_sd(__m128d a, __m128d b)
	__m128d _mm_cmpge_sd(__m128d a, __m128d b)
	__m128 _mm_cmpneq_sd(__m128d a, __m128d b)
	__m128 _mm_cmpnlt_sd(__m128d a, __m128d b)
	__m128d _mm_cmpnle_sd(__m128d a, __m128d b)
	__m128d _mm_cmpngt_sd(__m128d a, __m128d b)
	__m128d _mm_cmpnge_sd(__m128d a, __m128d b)
	__m128d _mm_cmpord_sd(__m128d a, __m128d b)
	__m128d _mm_cmpunord_sd(__m128d a, __m128d b)
CMPSS	__m128 _mm_cmpeq_ss(__m128 a, __m128 b)
	__m128 _mm_cmplt_ss(__m128 a, __m128 b)
	__m128 _mm_cmple_ss(__m128 a, __m128 b)
	__m128 _mm_cmpgt_ss(__m128 a, __m128 b)
	__m128 _mm_cmpge_ss(__m128 a, __m128 b)
	__m128 _mm_cmpneq_ss(__m128 a, __m128 b)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
	__m128 _mm_cmpnlt_ss(__m128 a, __m128 b)
	__m128 _mm_cmpnle_ss(__m128 a, __m128 b)
	__m128 _mm_cmpngt_ss(__m128 a, __m128 b)
	__m128 _mm_cmpnge_ss(__m128 a, __m128 b)
	__m128 _mm_cmpord_ss(__m128 a, __m128 b)
	__m128 _mm_cmpunord_ss(__m128 a, __m128 b)
COMISD	int _mm_comieq_sd(__m128d a, __m128d b)
	int _mm_comilt_sd(__m128d a, __m128d b)
	int _mm_comile_sd(__m128d a, __m128d b)
	int _mm_comigt_sd(__m128d a, __m128d b)
	int _mm_comige_sd(__m128d a, __m128d b)
	int _mm_comineq_sd(__m128d a, __m128d b)
COMISS	int _mm_comieq_ss(__m128 a, __m128 b)
	int _mm_comilt_ss(__m128 a, __m128 b)
	int _mm_comile_ss(__m128 a, __m128 b)
	int _mm_comigt_ss(__m128 a, __m128 b)
	int _mm_comige_ss(__m128 a, __m128 b) int _mm_comineq_ss(__m128 a, __m128 b)
CRC32	unsigned int _mm_crc32_u8(unsigned int crc, unsigned char data)
	unsigned int _mm_crc32_u16(unsigned int crc, unsigned short data)
	unsigned int _mm_crc32_u32(unsigned int crc, unsigned int data)
	unsigned __int64 _mm_crc32_u64(unsigned __int64 crc, unsigned __int64 data)
CVTDQ2PD	__m128d _mm_cvtepi32_pd(__m128i a)
CVTDQ2PS	__m128 _mm_cvtepi32_ps(__m128i a)
CVTPD2DQ	__m128i _mm_cvtpd_epi32(__m128d a)
CVTPD2PI	__m64 _mm_cvtpd_pi32(__m128d a)
CVTPD2PS	__m128 _mm_cvtpd_ps(__m128d a)
CVTPI2PD	__m128d _mm_cvtpi32_pd(__m64 a)
CVTPI2PS	__m128 _mm_cvt_pi2ps(__m128 a, __m64 b) __m128 _mm_cvtpi32_ps(__m128 a, __m64 b)
CVTPS2DQ	__m128i _mm_cvtps_epi32(__m128 a)
CVTPS2PD	__m128d _mm_cvtps_pd(__m128 a)
CVTPS2PI	__m64 _mm_cvt_ps2pi(__m128 a) __m64 _mm_cvtps_pi32(__m128 a)
CVTSD2SI	int _mm_cvtsd_si32(__m128d a)
CVTSD2SS	__m128 _mm_cvtsd_ss(__m128 a, __m128d b)
CVTSI2SD	__m128d _mm_cvtsi32_sd(__m128d a, int b)
CVTSI2SS	__m128 _mm_cvt_si2ss(__m128 a, int b) __m128 _mm_cvtsi32_ss(__m128 a, int b) __m128 _mm_cvtsi64_ss(__m128 a, __int64 b)
CVTSS2SD	__m128d _mm_cvtss_sd(__m128d a, __m128 b)
CVTSS2SI	int _mm_cvt_ss2si(__m128 a) int _mm_cvtss_si32(__m128 a)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
CVTTPD2DQ	__m128i _mm_cvttpd_epi32(__m128d a)
CVTTPD2PI	__m64 _mm_cvttpd_pi32(__m128d a)
CVTTPS2DQ	__m128i _mm_cvttps_epi32(__m128 a)
CVTTPS2PI	__m64 _mm_cvtt_ps2pi(__m128 a) __m64 _mm_cvttps_pi32(__m128 a)
CVTTSD2SI	int _mm_cvttsd_si32(__m128d a)
CVTTSS2SI	int _mm_cvtt_ss2si(__m128 a) int _mm_cvttss_si32(__m128 a) __m64 _mm_cvtsi32_si64(int i) int _mm_cvtsi64_si32(__m64 m)
DIVPD	__m128d _mm_div_pd(__m128d a, __m128d b)
DIVPS	__m128 _mm_div_ps(__m128 a, __m128 b)
DIVSD	__m128d _mm_div_sd(__m128d a, __m128d b)
DIVSS	__m128 _mm_div_ss(__m128 a, __m128 b)
DPPD	__m128d _mm_dp_pd(__m128d a, __m128d b, const int mask)
DPPS	__m128 _mm_dp_ps(__m128 a, __m128 b, const int mask)
EMMS	void _mm_empty()
EXTRACTPS	int _mm_extract_ps(__m128 src, const int ndx)
HADDPD	__m128d _mm_hadd_pd(__m128d a, __m128d b)
HADDPS	__m128 _mm_hadd_ps(__m128 a, __m128 b)
HSUBPD	__m128d _mm_hsub_pd(__m128d a, __m128d b)
HSUBPS	__m128 _mm_hsub_ps(__m128 a, __m128 b)
INSERTPS	__m128 _mm_insert_ps(__m128 dst, __m128 src, const int ndx)
LDDQU	__m128i _mm_lddqu_si128(__m128i const *p)
LDMXCSR	__mm_setcsr(unsigned int i)
LFENCE	void _mm_lfence(void)
MASKMOVDQU	void _mm_maskmoveu_si128(__m128i d, __m128i n, char *p)
MASKMOVQ	void _mm_maskmove_si64(__m64 d, __m64 n, char *p)
MAXPD	__m128d _mm_max_pd(__m128d a, __m128d b)
MAXPS	__m128 _mm_max_ps(__m128 a, __m128 b)
MAXSD	__m128d _mm_max_sd(__m128d a, __m128d b)
MAXSS	__m128 _mm_max_ss(__m128 a, __m128 b)
MFENCE	void _mm_mfence(void)
MINPD	__m128d _mm_min_pd(__m128d a, __m128d b)
MINPS	__m128 _mm_min_ps(__m128 a, __m128 b)
MINSD	__m128d _mm_min_sd(__m128d a, __m128d b)
MINSS	__m128 _mm_min_ss(__m128 a, __m128 b)
MONITOR	void _mm_monitor(void const *p, unsigned extensions, unsigned hints)
MOVAPD	__m128d _mm_load_pd(double * p)
	void_mm_store_pd(double *p, __m128d a)
MOVAPS	__m128 _mm_load_ps(float * p)
	void_mm_store_ps(float *p, __m128 a)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
MOVD	__m128i _mm_cvtsi32_si128(int a)
	int _mm_cvtsi128_si32(__m128i a)
	__m64 _mm_cvtsi32_si64(int a)
	int _mm_cvtsi64_si32(__m64 a)
MOVDDUP	__m128d _mm_movedup_pd(__m128d a)
	__m128d _mm_loaddup_pd(double const * dp)
MOVDQA	__m128i _mm_load_si128(__m128i * p)
	void_mm_store_si128(__m128i *p, __m128i a)
MOVDQU	__m128i _mm_loadu_si128(__m128i * p)
	void_mm_storeu_si128(__m128i *p, __m128i a)
MOVDQ2Q	__m64 _mm_movepi64_pi64(__m128i a)
MOVHLPS	__m128 _mm_movehl_ps(__m128 a, __m128 b)
MOVHPD	__m128d _mm_loadh_pd(__m128d a, double * p)
	void _mm_storeh_pd(double * p, __m128d a)
MOVHPS	__m128 _mm_loadh_pi(__m128 a, __m64 * p)
	void _mm_storeh_pi(__m64 * p, __m128 a)
MOVLPD	__m128d _mm_loadl_pd(__m128d a, double * p)
	void _mm_storel_pd(double * p, __m128d a)
MOVLPS	__m128 _mm_loadl_pi(__m128 a, __m64 *p)
	void_mm_storel_pi(__m64 * p, __m128 a)
MOVLHPS	__m128 _mm_movelh_ps(__m128 a, __m128 b)
MOVMSKPD	int _mm_movemask_pd(__m128d a)
MOVMSKPS	int _mm_movemask_ps(__m128 a)
MOVNTDQA	__m128i _mm_stream_load_si128(__m128i *p)
MOVNTDQ	void_mm_stream_si128(__m128i * p, __m128i a)
MOVNTPD	void_mm_stream_pd(double * p, __m128d a)
MOVNTPS	void_mm_stream_ps(float * p, __m128 a)
MOVNTI	void_mm_stream_si32(int * p, int a)
MOVNTQ	void_mm_stream_pi(__m64 * p, __m64 a)
MOVQ	__m128i _mm_loadl_epi64(__m128i * p)
	void_mm_storel_epi64(_m128i * p, __m128i a) __m128i _mm_move_epi64(__m128i a)
MOVQ2DQ	__m128i _mm_movpi64_epi64(__m64 a)
MOVSD	__m128d _mm_load_sd(double * p)
	void_mm_store_sd(double * p, __m128d a)
	__m128d _mm_move_sd(__m128d a, __m128d b)
MOVSHDUP	__m128 _mm_movehdup_ps(__m128 a)
MOVSLDUP	__m128 _mm_moveldup_ps(__m128 a)
MOVSS	__m128 _mm_load_ss(float * p)
	void_mm_store_ss(float * p, __m128 a)
	__m128 _mm_move_ss(__m128 a, __m128 b)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
MOVUPD	__m128d _mm_loadu_pd(double * p)
	void_mm_storeu_pd(double *p, __m128d a)
MOVUPS	__m128 _mm_loadu_ps(float * p)
	void_mm_storeu_ps(float *p, __m128 a)
MPSADBW	__m128i _mm_mpsadbw_epu8(__m128i s1, __m128i s2, const int mask)
MULPD	__m128d _mm_mul_pd(__m128d a, __m128d b)
MULPS	__m128 _mm_mul_ss(__m128 a, __m128 b)
MULSD	__m128d _mm_mul_sd(__m128d a, __m128d b)
MULSS	__m128 _mm_mul_ss(__m128 a, __m128 b)
MWAIT	void _mm_mwait(unsigned extensions, unsigned hints)
ORPD	__m128d _mm_or_pd(__m128d a, __m128d b)
ORPS	__m128 _mm_or_ps(__m128 a, __m128 b)
PABSB	__m64 _mm_abs_pi8 (__m64 a)
	__m128i _mm_abs_epi8 (__m128i a)
PABSD	__m64 _mm_abs_pi32 (__m64 a)
	__m128i _mm_abs_epi32 (__m128i a)
PABSW	__m64 _mm_abs_pi16 (__m64 a)
	__m128i _mm_abs_epi16 (__m128i a)
PACKSSWB	__m128i _mm_packs_epi16(__m128i m1, __m128i m2)
PACKSSWB	__m64 _mm_packs_pi16(__m64 m1, __m64 m2)
PACKSSDW	__m128i _mm_packs_epi32 (__m128i m1, __m128i m2)
PACKSSDW	__m64 _mm_packs_pi32 (__m64 m1, __m64 m2)
PACKUSDW	__m128i _mm_packus_epi32(__m128i m1, __m128i m2)
PACKUSWB	__m128i _mm_packus_epi16(__m128i m1, __m128i m2)
PACKUSWB	__m64 _mm_packs_pu16(__m64 m1, __m64 m2)
PADDB	__m128i _mm_add_epi8(__m128i m1, __m128i m2)
PADDB	__m64 _mm_add_pi8(__m64 m1, __m64 m2)
PADDW	__m128i _mm_add_epi16(__m128i m1, __m128i m2)
PADDW	__m64 _mm_add_pi16(__m64 m1, __m64 m2)
PADDD	__m128i _mm_add_epi32(__m128i m1, __m128i m2)
PADDD	__m64 _mm_add_pi32(__m64 m1, __m64 m2)
PADDQ	__m128i _mm_add_epi64(__m128i m1, __m128i m2)
PADDQ	__m64 _mm_add_si64(__m64 m1, __m64 m2)
PADDSB	__m128i _mm_adds_epi8(__m128i m1, __m128i m2)
PADDSB	__m64 _mm_adds_pi8(__m64 m1, __m64 m2)
PADDSW	__m128i _mm_adds_epi16(__m128i m1, __m128i m2)
PADDSW	__m64 _mm_adds_pi16(__m64 m1, __m64 m2)
PADDUSB	__m128i _mm_adds_epu8(__m128i m1, __m128i m2)
PADDUSB	__m64 _mm_adds_pu8(__m64 m1, __m64 m2)
PADDUSW	__m128i _mm_adds_epu16(__m128i m1, __m128i m2)
PADDUSW	__m64 _mm_adds_pu16(__m64 m1, __m64 m2)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
PALIGNR	__m64 _mm_alignr_pi8 (__m64 a, __m64 b, int n)
	__m128i _mm_alignr_epi8 (__m128i a, __m128i b, int n)
PAND	__m128i _mm_and_si128(__m128i m1, __m128i m2)
PAND	__m64 _mm_and_si64(__m64 m1, __m64 m2)
PANDN	__m128i _mm_andnot_si128(__m128i m1, __m128i m2)
PANDN	__m64 _mm_andnot_si64(__m64 m1, __m64 m2)
PAUSE	void _mm_pause(void)
PAVGB	__m128i _mm_avg_epu8(__m128i a, __m128i b)
PAVGB	__m64 _mm_avg_pu8(__m64 a, __m64 b)
PAVGW	__m128i _mm_avg_epu16(__m128i a, __m128i b)
PAVGW	__m64 _mm_avg_pu16(__m64 a, __m64 b)
PBLENDVB	__m128i _mm_blendv_epi (__m128i v1, __m128i v2, __m128i mask)
PBLENDW	__m128i _mm_blend_epi16(__m128i v1, __m128i v2, const int mask)
PCLMULQDQ	__m128i _mm_clmulepi64_si128 (__m128i, __m128i, const int)
PCMPEQB	__m128i _mm_cmpeq_epi8(__m128i m1, __m128i m2)
PCMPEQB	__m64 _mm_cmpeq_pi8(__m64 m1, __m64 m2)
PCMPEQQ	__m128i _mm_cmpeq_epi64(__m128i a, __m128i b)
PCMPEQW	__m128i _mm_cmpeq_epi16 (__m128i m1, __m128i m2)
PCMPEQW	__m64 _mm_cmpeq_pi16 (__m64 m1, __m64 m2)
PCMPEQD	__m128i _mm_cmpeq_epi32(__m128i m1, __m128i m2)
PCMPEQD	__m64 _mm_cmpeq_pi32(__m64 m1, __m64 m2)
PCMPESTRI	int _mm_cmpestri (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestra (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestrc (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestro (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestrs (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestrz (__m128i a, int la, __m128i b, int lb, const int mode)
PCMPESTRM	__m128i _mm_cmpestrm (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestra (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestrc (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestro (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestrs (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestrz (__m128i a, int la, __m128i b, int lb, const int mode)
PCMPGTB	__m128i _mm_cmpgt_epi8 (__m128i m1, __m128i m2)
PCMPGTB	__m64 _mm_cmpgt_pi8 (__m64 m1, __m64 m2)
PCMPGTW	__m128i _mm_cmpgt_epi16(__m128i m1, __m128i m2)
PCMPGTW	__m64 _mm_cmpgt_pi16 (__m64 m1, __m64 m2)
PCMPGTD	__m128i _mm_cmpgt_epi32(__m128i m1, __m128i m2)
PCMPGTD	__m64 _mm_cmpgt_pi32(__m64 m1, __m64 m2)
PCMPISTRI	__m128i _mm_cmpestrm (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestra (__m128i a, int la, __m128i b, int lb, const int mode)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
	int _mm_cmpestrc (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestro (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpestrs (__m128i a, int la, __m128i b, int lb, const int mode)
	int _mm_cmpistrz (__m128i a, __m128i b, const int mode)
PCMPISTRM	__m128i _mm_cmpistrm (__m128i a, __m128i b, const int mode)
	int _mm_cmpistra (__m128i a, __m128i b, const int mode)
	int _mm_cmpistrc (__m128i a, __m128i b, const int mode)
	int _mm_cmpistro (__m128i a, __m128i b, const int mode)
	int _mm_cmpistrs (__m128i a, __m128i b, const int mode)
	int _mm_cmpistrz (__m128i a, __m128i b, const int mode)
PCMPGTQ	__m128i _mm_cmpgt_epi64(__m128i a, __m128i b)
PEXTRB	int _mm_extract_epi8 (__m128i src, const int ndx)
PEXTRD	int _mm_extract_epi32 (__m128i src, const int ndx)
PEXTRQ	__int64 _mm_extract_epi64 (__m128i src, const int ndx)
PEXTRW	int _mm_extract_epi16(__m128i a, int n)
PEXTRW	int _mm_extract_pi16(__m64 a, int n)
	int _mm_extract_epi16 (__m128i src, int ndx)
PHADDD	__m64 _mm_hadd_pi32 (__m64 a, __m64 b)
	__m128i _mm_hadd_epi32 (__m128i a, __m128i b)
PHADDSW	__m64 _mm_hadds_pi16 (__m64 a, __m64 b)
	__m128i _mm_hadds_epi16 (__m128i a, __m128i b)
PHADDW	__m64 _mm_hadd_pi16 (__m64 a, __m64 b)
	__m128i _mm_hadd_epi16 (__m128i a, __m128i b)
PHMINPOSUW	__m128i _mm_minpos_epu16(__m128i packed_words)
PHSUBD	__m64 _mm_hsub_pi32 (__m64 a, __m64 b)
	__m128i _mm_hsub_epi32 (__m128i a, __m128i b)
PHSUBSW	__m64 _mm_hsubs_pi16 (__m64 a, __m64 b)
	__m128i _mm_hsubs_epi16 (__m128i a, __m128i b)
PHSUBW	__m64 _mm_hsub_pi16 (__m64 a, __m64 b)
	__m128i _mm_hsub_epi16 (__m128i a, __m128i b)
PINSRB	__m128i _mm_insert_epi8(__m128i s1, int s2, const int ndx)
PINSRD	__m128i _mm_insert_epi32(__m128i s2, int s, const int ndx)
PINSRQ	__m128i _mm_insert_epi64(__m128i s2, __int64 s, const int ndx)
PINSRW	__m128i _mm_insert_epi16(__m128i a, int d, int n)
PINSRW	__m64 _mm_insert_pi16(__m64 a, int d, int n)
PMADDUBSW	__m64 _mm_maddubs_pi16 (__m64 a, __m64 b)
	__m128i _mm_maddubs_epi16 (__m128i a, __m128i b)
PMADDWD	__m128i _mm_madd_epi16(__m128i m1 __m128i m2)
PMADDWD	__m64 _mm_madd_pi16(__m64 m1, __m64 m2)
PMAXSB	__m128i _mm_max_epi8(__m128i a, __m128i b)
PMAXSD	__m128i _mm_max_epi32(__m128i a, __m128i b)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
PMAXSW	__m128i _mm_max_epi16(__m128i a, __m128i b)
PMAXSW	__m64 _mm_max_pi16(__m64 a, __m64 b)
PMAXUB	__m128i _mm_max_epu8(__m128i a, __m128i b)
PMAXUB	__m64 _mm_max_pu8(__m64 a, __m64 b)
PMAXUD	__m128i _mm_max_epu32(__m128i a, __m128i b)
PMAXUW	__m128i _mm_max_epu16(__m128i a, __m128i b)
PMINSB	_m128i _mm_min_epi8(__m128i a, __m128i b)
PMINSD	__m128i _mm_min_epi32(__m128i a, __m128i b)
PMINSW	__m128i _mm_min_epi16(__m128i a, __m128i b)
PMINSW	__m64 _mm_min_pi16(__m64 a, __m64 b)
PMINUB	__m128i _mm_min_epu8(__m128i a, __m128i b)
PMINUB	__m64 _mm_min_pu8(__m64 a, __m64 b)
PMINUD	__m128i _mm_min_epu32 (__m128i a, __m128i b)
PMINUW	__m128i _mm_min_epu16 (__m128i a, __m128i b)
PMOVMSKB	int _mm_movemask_epi8(__m128i a)
PMOVMSKB	int _mm_movemask_pi8(__m64 a)
PMOVSXBW	__m128i _mm_ cvtepi8_epi16(__m128i a)
PMOVSXBD	__m128i _mm_ cvtepi8_epi32(__m128i a)
PMOVSXBQ	__m128i _mm_ cvtepi8_epi64(__m128i a)
PMOVSXWD	__m128i _mm_ cvtepi16_epi32(__m128i a)
PMOVSXWQ	__m128i _mm_ cvtepi16_epi64(__m128i a)
PMOVSXDQ	__m128i _mm_ cvtepi32_epi64(__m128i a)
PMOVZXBW	__m128i _mm_ cvtepu8_epi16(__m128i a)
PMOVZXBD	__m128i _mm_ cvtepu8_epi32(__m128i a)
PMOVZXBQ	__m128i _mm_ cvtepu8_epi64(__m128i a)
PMOVZXWD	__m128i _mm_ cvtepu16_epi32(__m128i a)
PMOVZXWQ	__m128i _mm_ cvtepu16_epi64(__m128i a)
PMOVZXDQ	__m128i _mm_ cvtepu32_epi64(__m128i a)
PMULDQ	__m128i _mm_mul_epi32(__m128i a, __m128i b)
PMULHRSW	__m64 _mm_mulhrs_pi16 (__m64 a, __m64 b)
	__m128i _mm_mulhrs_epi16 (__m128i a, __m128i b)
PMULHUW	__m128i _mm_mulhi_epu16(__m128i a, __m128i b)
PMULHUW	__m64 _mm_mulhi_pu16(__m64 a, __m64 b)
PMULHW	__m128i _mm_mulhi_epi16(__m128i m1, __m128i m2)
PMULHW	__m64 _mm_mulhi_pi16(__m64 m1, __m64 m2)
PMULLUD	__m128i _mm_mullo_epi32(__m128i a, __m128i b)
PMULLW	__m128i _mm_mullo_epi16(__m128i m1, __m128i m2)
PMULLW	__m64 _mm_mullo_pi16(__m64 m1, __m64 m2)
PMULUDQ	__m64 _mm_mul_su32(__m64 m1, __m64 m2)
	__m128i _mm_mul_epu32(__m128i m1, __m128i m2)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
POPCNT	int _mm_popcnt_u32(unsigned int a)
	int64_t _mm_popcnt_u64(unsigned __int64 a)
POR	__m64 _mm_or_si64(__m64 m1, __m64 m2)
POR	__m128i _mm_or_si128(__m128i m1, __m128i m2)
PREFETCHh	void _mm_prefetch(char *a, int sel)
PSADBW	__m128i _mm_sad_epu8(__m128i a, __m128i b)
PSADBW	__m64 _mm_sad_pu8(__m64 a, __m64 b)
PSHUFB	__m64 _mm_shuffle_pi8 (__m64 a, __m64 b)
	__m128i _mm_shuffle_epi8 (__m128i a, __m128i b)
PSHUFD	__m128i _mm_shuffle_epi32(__m128i a, int n)
PSHUFHW	__m128i _mm_shufflehi_epi16(__m128i a, int n)
PSHUFLW	__m128i _mm_shufflelo_epi16(__m128i a, int n)
PSHUFW	__m64 _mm_shuffle_pi16(__m64 a, int n)
PSIGNB	__m64 _mm_sign_pi8 (__m64 a, __m64 b)
	__m128i _mm_sign_epi8 (__m128i a, __m128i b)
PSIGND	__m64 _mm_sign_pi32 (__m64 a, __m64 b)
	__m128i _mm_sign_epi32 (__m128i a, __m128i b)
PSIGNW	__m64 _mm_sign_pi16 (__m64 a, __m64 b)
	__m128i _mm_sign_epi16 (__m128i a, __m128i b)
PSLLW	__m128i _mm_sll_epi16(__m128i m, __m128i count)
PSLLW	__m128i _mm_slli_epi16(__m128i m, int count)
PSLLW	__m64 _mm_sll_pi16(__m64 m, __m64 count)
	__m64 _mm_slli_pi16(__m64 m, int count)
PSLLD	__m128i _mm_slli_epi32(__m128i m, int count)
	__m128i _mm_sll_epi32(__m128i m, __m128i count)
PSLLD	__m64 _mm_slli_pi32(__m64 m, int count)
	__m64 _mm_sll_pi32(__m64 m, __m64 count)
PSLLQ	__m64 _mm_sll_si64(__m64 m, __m64 count)
	__m64 _mm_slli_si64(__m64 m, int count)
PSLLQ	__m128i _mm_sll_epi64(__m128i m, __m128i count)
	__m128i _mm_slli_epi64(__m128i m, int count)
PSLLDQ	__m128i _mm_slli_si128(__m128i m, int imm)
PSRAW	__m128i _mm_sra_epi16(__m128i m, __m128i count)
	__m128i _mm_srai_epi16(__m128i m, int count)
PSRAW	__m64 _mm_sra_pi16(__m64 m, __m64 count)
	__m64 _mm_srai_pi16(__m64 m, int count)
PSRAD	__m128i _mm_sra_epi32 (__m128i m, __m128i count)
	__m128i _mm_srai_epi32 (__m128i m, int count)
PSRAD	__m64 _mm_sra_pi32 (__m64 m, __m64 count)
	__m64 _mm_srai_pi32 (__m64 m, int count)
PSRLW	_m128i _mm_srl_epi16 (__m128i m, __m128i count)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
	__m128i _mm_srli_epi16 (__m128i m, int count)
	__m64 _mm_srl_pi16 (__m64 m, __m64 count)
	__m64 _mm_srli_pi16(__m64 m, int count)
PSRLD	__m128i _mm_srl_epi32 (__m128i m, __m128i count)
	__m128i _mm_srli_epi32 (__m128i m, int count)
PSRLD	__m64 _mm_srl_pi32 (__m64 m, __m64 count)
	__m64 _mm_srli_pi32 (__m64 m, int count)
PSRLQ	__m128i _mm_srl_epi64 (__m128i m, __m128i count)
	__m128i _mm_srli_epi64 (__m128i m, int count)
PSRLQ	__m64 _mm_srl_si64 (__m64 m, __m64 count)
	__m64 _mm_srli_si64 (__m64 m, int count)
PSRLDQ	__m128i _mm_srli_si128(__m128i m, int imm)
PSUBB	__m128i _mm_sub_epi8(__m128i m1, __m128i m2)
PSUBB	__m64 _mm_sub_pi8(__m64 m1, __m64 m2)
PSUBW	__m128i _mm_sub_epi16(__m128i m1, __m128i m2)
PSUBW	__m64 _mm_sub_pi16(__m64 m1, __m64 m2)
PSUBD	__m128i _mm_sub_epi32(__m128i m1, __m128i m2)
PSUBD	__m64 _mm_sub_pi32(__m64 m1, __m64 m2)
PSUBQ	__m128i _mm_sub_epi64(__m128i m1, __m128i m2)
PSUBQ	__m64 _mm_sub_si64(__m64 m1, __m64 m2)
PSUBSB	__m128i _mm_subs_epi8(__m128i m1, __m128i m2)
PSUBSB	__m64 _mm_subs_pi8(__m64 m1, __m64 m2)
PSUBSW	__m128i _mm_subs_epi16(__m128i m1, __m128i m2)
PSUBSW	__m64 _mm_subs_pi16(__m64 m1, __m64 m2)
PSUBUSB	__m128i _mm_subs_epu8(__m128i m1, __m128i m2)
PSUBUSB	__m64 _mm_subs_pu8(__m64 m1, __m64 m2)
PSUBUSW	__m128i _mm_subs_epu16(__m128i m1, __m128i m2)
PSUBUSW	__m64 _mm_subs_pu16(__m64 m1, __m64 m2)
PTEST	int _mm_testz_si128(__m128i s1, __m128i s2)
	int _mm_testc_si128(__m128i s1, __m128i s2)
	int _mm_testnzc_si128(__m128i s1, __m128i s2)
PUNPCKHBW	__m64 _mm_unpackhi_pi8(__m64 m1, __m64 m2)
PUNPCKHBW	__m128i _mm_unpackhi_epi8(__m128i m1, __m128i m2)
PUNPCKHWD	__m64 _mm_unpackhi_pi16(__m64 m1,__m64 m2)
PUNPCKHWD	__m128i _mm_unpackhi_epi16(__m128i m1, __m128i m2)
PUNPCKHDQ	___m64 _mm_unpackhi_pi32(__m64 m1, __m64 m2)
PUNPCKHDQ	__m128i _mm_unpackhi_epi32(__m128i m1, __m128i m2)
PUNPCKHQDQ	__m128i _mm_unpackhi_epi64(__m128i m1, __m128i m2)
PUNPCKLBW	__m64 _mm_unpacklo_pi8 (__m64 m1, __m64 m2)
PUNPCKLBW	__m128i _mm_unpacklo_epi8 (__m128i m1, __m128i m2)
PUNPCKLWD	__m64 _mm_unpacklo_pi16(__m64 m1, __m64 m2)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
PUNPCKLWD	__m128i _mm_unpacklo_epi16(__m128i m1, __m128i m2)
PUNPCKLDQ	__m64 _mm_unpacklo_pi32(__m64 m1, __m64 m2)
PUNPCKLDQ	__m128i _mm_unpacklo_epi32(__m128i m1, __m128i m2)
PUNPCKLQDQ	__m128i _mm_unpacklo_epi64(__m128i m1, __m128i m2)
PXOR	__m64 _mm_xor_si64(__m64 m1, __m64 m2)
PXOR	__m128i _mm_xor_si128(__m128i m1, __m128i m2)
RCPPS	__m128 _mm_rcp_ps(__m128 a)
RCPSS	__m128 _mm_rcp_ss(__m128 a)
ROUNDPD	__m128 mm_round_pd(__m128d s1, int iRoundMode)
	__m128 mm_floor_pd(__m128d s1)
	__m128 mm_ceil_pd(__m128d s1)
ROUNDPS	__m128 mm_round_ps(__m128 s1, int iRoundMode)
	__m128 mm_floor_ps(__m128 s1)
	__m128 mm_ceil_ps(__m128 s1)
ROUNDSD	__m128d mm_round_sd(__m128d dst, __m128d s1, int iRoundMode)
	__m128d mm_floor_sd(__m128d dst, __m128d s1)
	__m128d mm_ceil_sd(__m128d dst, __m128d s1)
ROUNDSS	__m128 mm_round_ss(__m128 dst, __m128 s1, int iRoundMode)
	__m128 mm_floor_ss(__m128 dst, __m128 s1)
	__m128 mm_ceil_ss(__m128 dst, __m128 s1)
RSQRTPS	__m128 _mm_rsqrt_ps(__m128 a)
RSQRTSS	__m128 _mm_rsqrt_ss(__m128 a)
SFENCE	void_mm_sfence(void)
SHUFPD	__m128d _mm_shuffle_pd(__m128d a, __m128d b, unsigned int imm8)
SHUFPS	__m128 _mm_shuffle_ps(__m128 a, __m128 b, unsigned int imm8)
SQRTPD	__m128d _mm_sqrt_pd(__m128d a)
SQRTPS	__m128 _mm_sqrt_ps(__m128 a)
SQRTSD	__m128d _mm_sqrt_sd(__m128d a)
SQRTSS	__m128 _mm_sqrt_ss(__m128 a)
STMXCSR	_mm_getcsr(void)
SUBPD	__m128d _mm_sub_pd(__m128d a, __m128d b)
SUBPS	__m128 _mm_sub_ps(__m128 a, __m128 b)
SUBSD	__m128d _mm_sub_sd(__m128d a, __m128d b)
SUBSS	__m128 _mm_sub_ss(__m128 a, __m128 b)
UCOMISD	int _mm_ucomieq_sd(__m128d a, __m128d b)
	int _mm_ucomilt_sd(__m128d a, __m128d b)
	int _mm_ucomile_sd(__m128d a, __m128d b)
	int _mm_ucomigt_sd(__m128d a, __m128d b)
	int _mm_ucomige_sd(__m128d a, __m128d b)
	int _mm_ucomineq_sd(__m128d a, __m128d b)
UCOMISS	int _mm_ucomieq_ss(__m128 a, __m128 b)

Table C-1. Simple Intrinsics (Contd.)

Mnemonic	Intrinsic
	int _mm_ucomilt_ss(__m128 a, __m128 b)
	int _mm_ucomile_ss(__m128 a, __m128 b)
	int _mm_ucomigt_ss(__m128 a, __m128 b)
	int _mm_ucomige_ss(__m128 a, __m128 b)
	int _mm_ucomineq_ss(__m128 a, __m128 b)
UNPCKHPD	__m128d _mm_unpackhi_pd(__m128d a, __m128d b)
UNPCKHPS	__m128 _mm_unpackhi_ps(__m128 a, __m128 b)
UNPCKLPD	__m128d _mm_unpacklo_pd(__m128d a, __m128d b)
UNPCKLPS	__m128 _mm_unpacklo_ps(__m128 a, __m128 b)
XORPD	__m128d _mm_xor_pd(__m128d a, __m128d b)
XORPS	__m128 _mm_xor_ps(__m128 a, __m128 b)

C.2 COMPOSITE INTRINSICS

Table C-2. Composite Intrinsics

Mnemonic	Intrinsic
(composite)	__m128i _mm_set_epi64(__m64 q1, __m64 q0)
(composite)	__m128i _mm_set_epi32(int i3, int i2, int i1, int i0)
(composite)	__m128i _mm_set_epi16(short w7,short w6, short w5, short w4, short w3, short w2, short w1,short w0)
(composite)	__m128i _mm_set_epi8(char w15,char w14, char w13, char w12, char w11, char w10, char w9, char w8, char w7,char w6, char w5, char w4, char w3, char w2,char w1, char w0)
(composite)	__m128i _mm_set1_epi64(__m64 q)
(composite)	__m128i _mm_set1_epi32(int a)
(composite)	__m128i _mm_set1_epi16(short a)
(composite)	__m128i _mm_set1_epi8(char a)
(composite)	__m128i _mm_setr_epi64(__m64 q1, __m64 q0)
(composite)	__m128i _mm_setr_epi32(int i3, int i2, int i1, int i0)
(composite)	__m128i _mm_setr_epi16(short w7,short w6, short w5, short w4, short w3, short w2, short w, short w0)
(composite)	__m128i _mm_setr_epi8(char w15,char w14, char w13, char w12, char w11, char w10, char w9, char w8,char w7, char w6,char w5, char w4, char w3, char w2,char w1,char w0)
(composite)	__m128i _mm_setzero_si128()
(composite)	__m128 _mm_set_ps1(float w) __m128 _mm_set1_ps(float w)
(composite)	__m128cmm_set1_pd(double w)
(composite)	__m128d _mm_set_sd(double w)
(composite)	__m128d _mm_set_pd(double z, double y)
(composite)	__m128 _mm_set_ps(float z, float y, float x, float w)
(composite)	__m128d _mm_setr_pd(double z, double y)
(composite)	__m128 _mm_setr_ps(float z, float y, float x, float w)
(composite)	__m128d _mm_setzero_pd(void)
(composite)	__m128 _mm_setzero_ps(void)

Table C-2. Composite Intrinsics (Contd.)

Mnemonic	Intrinsic
MOVSD + shuffle	__m128d _mm_load_pd(double * p) __m128d _mm_load1_pd(double *p)
MOVSS + shuffle	__m128 _mm_load_ps1(float * p) __m128 _mm_load1_ps(float *p)
MOVAPD + shuffle	__m128d _mm_loadr_pd(double * p)
MOVAPS + shuffle	__m128 _mm_loadr_ps(float * p)
MOVSD + shuffle	void _mm_store1_pd(double *p, __m128d a)
MOVSS + shuffle	void _mm_store_ps1(float * p, __m128 a) void _mm_store1_ps(float *p, __m128 a)
MOVAPD + shuffle	_mm_storer_pd(double * p, __m128d a)
MOVAPS + shuffle	_mm_storer_ps(float * p, __m128 a)