

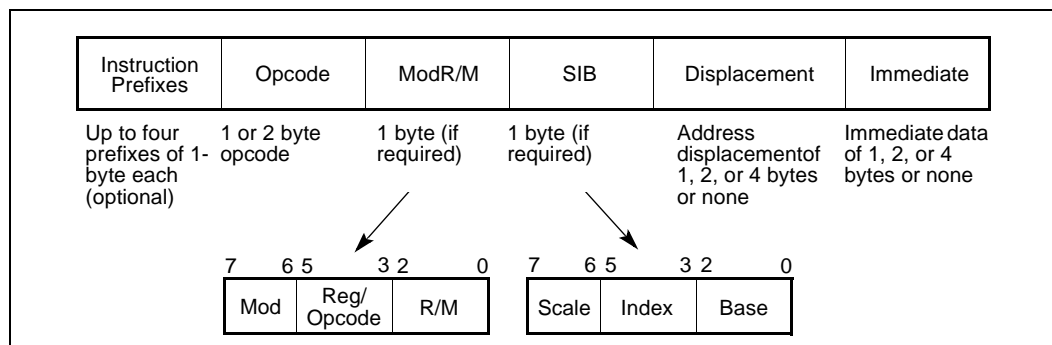
Instruction Formats and Encodings 36

This chapter describes the instruction format for all Intel Architecture processors.

36.1 General Instruction Format

All Intel Architecture instruction encodings are subsets of the general instruction format shown in Figure 36-1. Instructions consist of optional instruction prefixes (in any order), one or two primary opcode bytes, an addressing-form specifier (if required) consisting of the ModR/M byte and sometimes the SIB (Scale-Index-Base) byte, a displacement (if required), and an immediate data field (if required).

Figure 36-1. Intel Architecture Instruction Format



36.2 Instruction Prefixes

The instruction prefixes are divided into four groups, each with a set of allowable prefix codes:

- Lock and repeat prefixes.
 - F0H—LOCK prefix.
 - F2H—REPNE/REPZ prefix (used only with string instructions).
 - F3H—REP prefix (used only with string instructions).
 - F3H—REPE/REPZ prefix (used only with string instructions).
- Segment override.
 - 2EH—CS segment override prefix.
 - 36H—SS segment override prefix.
 - 3EH—DS segment override prefix.
 - 26H—ES segment override prefix.
 - 64H—FS segment override prefix.

— 65H—GS segment override prefix.

- Operand-size override, 66H
- Address-size override, 67H

For each instruction, one prefix may be used from each of these groups and be placed in any order. The effect of redundant prefixes (more than one prefix from a group) is undefined and may vary from processor to processor.

36.3 Opcode

The primary opcode is either 1 or 2 bytes. An additional 3-bit opcode field is sometimes encoded in the ModR/M byte. Smaller encoding fields can be defined within the primary opcode. These fields define the direction of the operation, the size of displacements, the register encoding, condition codes, or sign extension. The encoding of fields in the opcode varies, depending on the class of operation.

36.4 ModR/M and SIB Bytes

Most instructions that refer to an operand in memory have an addressing-form specifier byte (called the ModR/M byte) following the primary opcode. The ModR/M byte contains three fields of information:

- The *mod* field combines with the *r/m* field to form 32 possible values: eight registers and 24 addressing modes.
- The *reg/opcode* field specifies either a register number or three more bits of opcode information. The purpose of the *reg/opcode* field is specified in the primary opcode.
- The *r/m* field can specify a register as an operand or can be combined with the *mod* field to encode an addressing mode.

Certain encodings of the ModR/M byte require a second addressing byte, the SIB byte, to fully specify the addressing form. The base-plus-index and scale-plus-index forms of 32-bit addressing require the SIB byte. The SIB byte includes the following fields:

- The *scale* field specifies the scale factor.
- The *index* field specifies the register number of the index register.
- The *base* field specifies the register number of the base register.

See “Addressing-Mode Encoding of ModR/M and SIB Bytes”, for the encodings of the ModR/M and SIB bytes.

36.5 Displacement and Immediate Bytes

Some addressing forms include a displacement immediately following either the ModR/M or SIB byte. If a displacement is required, it can be 1, 2, or 4 bytes.

If the instruction specifies an immediate operand, the operand always follows any displacement bytes. An immediate operand can be 1, 2 or 4 bytes.

36.6 Addressing-Mode Encoding of ModR/M and SIB Bytes

The values and the corresponding addressing forms of the ModR/M and SIB bytes are shown in Tables 36-1 through 36-3. The 16-bit addressing forms specified by the ModR/M byte are in Table 36-1, and the 32-bit addressing forms specified by the ModR/M byte are in Table 36-2. Table 36-3 shows the 32-bit addressing forms specified by the SIB byte.

In Tables 36-1 and 36-2, the first column (labeled “Effective Address”) lists 32 different effective addresses that can be assigned to one operand of an instruction by using the Mod and R/M fields of the ModR/M byte. The first 24 give the different ways of specifying a memory location; the last 8 (specified by the Mod field encoding 11B) give the ways of specifying the general purpose and MMX registers. Each of the register encodings list four possible registers. For example, the first register-encoding (selected by the R/M field encoding of 000B) indicates the general-purpose registers EAX, AX or AL, or the MMX register MM0. Which of these four registers is used is determined by the opcode byte and the operand-size attribute, which select either the EAX register (32 bits) or AX register (16 bits).

The second and third columns in Tables 36-1 and 36-2 gives the binary encodings of the Mod and R/M fields in the ModR/M byte, respectively, required to obtain the associated effective address listed in the first column. All 32 possible combinations of the Mod and R/M fields are listed.

Across the top of Tables 36-1 and 36-2, the 8 possible values of the 3-bit Reg/Opcode field are listed, in decimal (fifth row from top) and in binary (sixth row from top). The sixth row is labeled “REG=” which represents the use of these 3 bits to give the location of a second operand, which must be a general-purpose register or an MMX register. If the instruction does not require a second operand to be specified, then the 3 bits of the Reg/Opcode field may be used as an extension of the opcode, which is represented by the fifth row, labeled “/digit (Opcode)”. The four rows above give the byte, word and doubleword general-purpose registers and the MMX registers that correspond to the register numbers, with the same assignments as for the R/M field when Mod field encoding is 11B. As with the R/M field register options, which of the four possible registers is used is determined by the opcode byte along with the operand-size attribute.

The body of Tables 2-1 and 2-2 (under the label “Value of ModR/M Byte (in Hexadecimal)”) contains a 32 by 8 array giving all of the 256 values of the ModR/M byte, in hexadecimal. Bits 3, 4 and 5 are specified by the column of the table in which a byte resides, and the row specifies bits 0, 1 and 2, and also bits 6 and 7.

Table 36-1. 16-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) /digit (Opcode) REG =			AL AX EAX MM0 0 000	CL CX ECX MM1 1 001	DL DX EDX MM2 2 010	BL BX EBX MM3 3 011	AH SP ESP MM4 4 100	CH BP ¹ EBP MM5 5 101	DH SI ESI MM6 6 110	BH DI EDI MM7 7 111
Effective Address	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)							
[BX+SI] [BX+DI] [BP+SI] [BP+DI] [SI] [DI] disp16 ² [BX]	00	000 001 010 011 100 101 110 111	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	10 11 12 13 14 15 16 17	18 19 1A 1B 1C 1D 1E 1F	20 21 22 23 24 25 26 27	28 29 2A 2B 2C 2D 2E 2F	30 31 32 33 34 35 36 37	38 39 3A 3B 3C 3D 3E 3F
[BX+SI]+disp8 ³ [BX+DI]+disp8 [BP+SI]+disp8 [BP+DI]+disp8 [SI]+disp8 [DI]+disp8 [BP]+disp8 [BX]+disp8		000 001 010 011 100 101 110 111	40 41 42 43 44 45 46 47	48 49 4A 4B 4C 4D 4E 4F	50 51 52 53 54 55 56 57	58 59 5A 5B 5C 5D 5E 5F	60 61 62 63 64 65 66 67	68 69 6A 6B 6C 6D 6E 6F	70 71 72 73 74 75 76 77	78 79 7A 7B 7C 7D 7E 7F
[BX+SI]+disp16 [BX+DI]+disp16 [BP+SI]+disp16 [BP+DI]+disp16 [SI]+disp16 [DI]+disp16 [BP]+disp16 [BX]+disp16		000 001 010 011 100 101 110 111	80 81 82 83 84 85 86 87	88 89 8A 8B 8C 8D 8E 8F	90 91 92 93 94 95 96 97	98 99 9A 9B 9C 9D 9E 9F	A0 A1 A2 A3 A4 A5 A6 A7	A8 A9 AA AB AC AD AE AF	B0 B1 B2 B3 B4 B5 B6 B7	B8 B9 BA BB BC BD BE BF
EAX/AX/AL/MM0 ECX/CX/CL/MM1 EDX/DX/DL/MM2 EBX/BX/BL/MM3 ESP/SP/AHMM4 EBP/BP/CH/MM5 ESI/SI/DH/MM6 EDI/DI/BH/MM7		000 001 010 011 100 101 110 111	C0 C1 C2 C3 C4 C5 C6 C7	C8 C9 CA CB CC CD CE CF	D0 D1 D2 D3 D4 D5 D6 D7	D8 D9 DA DB DC DD DE DF	E0 E1 E2 E3 E4 E5 E6 E7	E8 E9 EA EB EC ED EE EF	F0 F1 F2 F3 F4 F5 F6 F7	F8 F9 FA FB FC FD FE FF

Notes:

1. The default segment register is SS for the effective addresses containing a BP index, DS for other effective addresses.
2. The "disp16" nomenclature denotes a 16-bit displacement following the ModR/M byte, to be added to the index.
3. The "disp8" nomenclature denotes an 8-bit displacement following the ModR/M byte, to be sign-extended and added to the index.

Table 36-2. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) /digit (Opcode) REG =			AL AX EAX MM0 0 000	CL CX ECX MM1 1 001	DL DX EDX MM2 2 010	BL BX EBX MM3 3 011	AH SP ESP MM4 4 100	CH BP EBP MM5 5 101	DH SI ESI MM6 6 110	BH DI EDI MM7 7 111
Effective Address	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)							
[EAX] [ECX] [EDX] [EBX] [--][--] ¹ disp32 ² [ESI] [EDI]	00	000	00	08	10	18	20	28	30	38
		001	01	09	11	19	21	29	31	39
		010	02	0A	12	1A	22	2A	32	3A
		011	03	0B	13	1B	23	2B	33	3B
		100	04	0C	14	1C	24	2C	34	3C
		101	05	0D	15	1D	25	2D	35	3D
		110	06	0E	16	1E	26	2E	36	3E
		111	07	0F	17	1F	27	2F	37	3F
disp8[EAX] ³ disp8[ECX] disp8[EDX] disp8[EBX]; disp8[--][--] disp8[EBP] disp8[ESI] disp8[EDI]	01	000	40	48	50	58	60	68	70	78
		001	41	49	51	59	61	69	71	79
		010	42	4A	52	5A	62	6A	72	7A
		011	43	4B	53	5B	63	6B	73	7B
		100	44	4C	54	5C	64	6C	74	7C
		101	45	4D	55	5D	65	6D	75	7D
		110	46	4E	56	5E	66	6E	76	7E
		111	47	4F	57	5F	67	6F	77	7F
disp32[EAX] disp32[ECX] disp32[EDX] disp32[EBX] disp32[--][--] disp32[EBP] disp32[ESI] disp32[EDI]	10	000	80	88	90	98	A0	A8	B0	B8
		001	81	89	91	99	A1	A9	B1	B9
		010	82	8A	92	9A	A2	AA	B2	BA
		011	83	8B	93	9B	A3	AB	B3	BB
		100	84	8C	94	9C	A4	AC	B4	BC
		101	85	8D	95	9D	A5	AD	B5	BD
		110	86	8E	96	9E	A6	AE	B6	BE
		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL/MM0 ECX/CX/CL/MM1 EDX/DX/DL/MM2 EBX/BX/BL/MM3 ESP/SP/AH/MM4 EBP/BP/CH/MM5 ESI/SI/DH/MM6 EDI/DI/BH/MM7	11	000	C0	C8	D0	D8	E0	E8	F0	F8
		001	C1	C9	D1	D9	E1	E9	F1	F9
		010	C2	CA	D2	DA	E2	EA	F2	FA
		011	C3	CB	D3	DB	E3	EB	F3	FB
		100	C4	CC	D4	DC	E4	EC	F4	FC
		101	C5	CD	D5	DD	E5	ED	F5	FD
		110	C6	CE	D6	DE	E6	EE	F6	FE
		111	C7	CF	D7	DF	E7	EF	F7	FF

NOTES:

- The [--][--] nomenclature means a SIB follows the ModR/M byte.
- The disp32 nomenclature denotes a 32-bit displacement following the SIB byte, to be added to the index.
- The disp8 nomenclature denotes an 8-bit displacement following the SIB byte, to be sign-extended and added to the index.

Table 2-3 is organized similarly to Tables 2-1 and 2-2, except that its body gives the 256 possible values of the SIB byte, in hexadecimal. Which of the 8 general-purpose registers will be used as base is indicated across the top of the table, along with the corresponding values of the base field (bits 0, 1 and 2) in decimal and binary. The rows indicate which register is used as the index (determined by bits 3, 4 and 5) along with the scaling factor (determined by bits 6 and 7).

Table 36-3. 32-Bit Addressing Forms with the SIB Byte

r32 Base = Base =			EAX 0 000	ECX 1 001	EDX 2 010	EBX 3 011	ESP 4 100	[*] 5 101	ESI 6 110	EDI 7 111
Scaled Index	SS	Index	Value of SIB Byte (in Hexadecimal)							
[EAX]	00	000	00	01	02	03	04	05	06	07
[ECX]		001	08	09	0A	0B	0C	0D	0E	0F
[EDX]		010	10	11	12	13	14	15	16	17
[EBX]		011	18	19	1A	1B	1C	1D	1E	1F
none		100	20	21	22	23	24	25	26	27
[EBP]		101	28	29	2A	2B	2C	2D	2E	2F
[ESI]		110	30	31	32	33	34	35	36	37
[EDI]		111	38	39	3A	3B	3C	3D	3E	3F
[EAX*2]	01	000	40	41	42	43	44	45	46	47
[ECX*2]		001	48	49	4A	4B	4C	4D	4E	4F
[EDX*2]		010	50	51	52	53	54	55	56	57
[EBX*2]		011	58	59	5A	5B	5C	5D	5E	5F
none		100	60	61	62	63	64	65	66	67
[EBP*2]		101	68	69	6A	6B	6C	6D	6E	6F
[ESI*2]		110	70	71	72	73	74	75	76	77
[EDI*2]		111	78	79	7A	7B	7C	7D	7E	7F
[EAX*4]	10	000	80	81	82	83	84	85	86	87
[ECX*4]		001	88	89	8A	8B	8C	8D	8E	8F
[EDX*4]		010	90	91	92	93	94	95	96	97
[EBX*4]		011	98	99	9A	9B	9C	9D	9E	9F
none		100	A0	A1	A2	A3	A4	A5	A6	A7
[EBP*4]		101	A8	A9	AA	AB	AC	AD	AE	AF
[ESI*4]		110	B0	B1	B2	B3	B4	B5	B6	B7
[EDI*4]		111	B8	B9	BA	BB	BC	BD	BE	BF
[EAX*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7
[ECX*8]		001	C8	C9	CA	CB	CC	CD	CE	CF
[EDX*8]		010	D0	D1	D2	D3	D4	D5	D6	D7
[EBX*8]		011	D8	D9	DA	DB	DC	DD	DE	DF
none		100	E0	E1	E2	E3	E4	E5	E6	E7
[EBP*8]		101	E8	E9	EA	EB	EC	ED	EE	EF
[ESI*8]		110	F0	F1	F2	F3	F4	F5	F6	F7
[EDI*8]		111	F8	F9	FA	FB	FC	FD	FE	FF

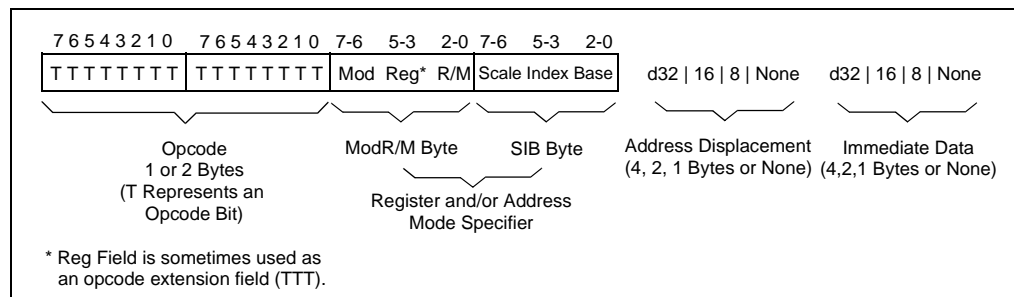
Note:

- The [*] nomenclature means a disp32 with no base if MOD is 00, [EBP] otherwise. This provides the following addressing modes:

disp32[index] (MOD=00).
 disp8[EBP][index] (MOD=01).
 disp32[EBP][index] (MOD=10).

36.7 Machine Instruction Format

All Intel Architecture instructions are encoded using subsets of the general machine instruction format shown in Figure 36-2. Each instruction consists of an opcode, a register and/or address mode specifier (if required) consisting of the ModR/M byte and sometimes the scale-index-base (SIB) byte, a displacement (if required), and an immediate data field (if required).

Figure 36-2. General Machine Instruction Format


The primary opcode for an instruction is encoded in one or two bytes of the instruction. Some instructions also use an opcode extension field encoded in bits 5, 4, and 3 of the ModR/M byte. Within the primary opcode, smaller encoding fields may be defined. These fields vary according to the class of operation being performed. The fields define such information as register encoding, conditional test performed, or sign extension of immediate byte.

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, the reg field, and the R/M field. Certain encodings of the ModR/M byte indicate that a second address mode byte, the SIB byte, must be used.

If the selected addressing mode specifies a displacement, the displacement value is placed immediately following the ModR/M byte or SIB byte. If a displacement is present, the possible sizes are 8, 16, or 32 bits.

If the instruction specifies an immediate operand, the immediate value follows any displacement bytes. An immediate operand, if specified, is always the last field of the instruction.

Table 36-4 lists several smaller fields or bits that appear in certain instructions, sometimes within the opcode bytes themselves. The following tables describe these fields and bits and list the allowable values. All of these fields (except the d bit) are shown in the integer instruction formats given in Table 36-13.

Table 36-4. Special Fields Within Instruction Encodings

Field Name	Description	Number of Bits
reg	General-register specifier (see Table 36-5 or 36-6)	3
w	Specifies if data is byte or full-sized, where full-sized is either 16 or 32 bits (see Table 36-7)	1
s	Specifies sign extension of an immediate data field (see Table 36-8)	1
sreg2	Segment register specifier for CS, SS, DS, ES (see Table 36-9)	2
sreg3	Segment register specifier for CS, SS, DS, ES, FS, GS (see Table 36-9)	3
eee	Specifies a special-purpose (control or debug) register (see Table 36-10)	3
tttn	For conditional instructions, specifies a condition asserted or a condition negated (see Table 36-11)	4
d	Specifies direction of data operation (see Table 36-12)	1

36.7.1 Reg Field (reg)

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 (see Table 36-7). Table 36-5 shows the encoding of the reg field when the w bit is not present in an encoding, and Table 36-6 shows the encoding of the reg field when the w bit is present.

Table 36-5. 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 36-6. Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field during 16-Bit Data Operations			Register Specified by reg Field during 32-Bit Data Operations		
reg	Function of w Field		reg	Function of w Field	
	When w = 0	When w = 1		When w = 0	When w = 1
000	AL	AX	000	AL	EAX
001	CL	CX	001	CL	ECX
010	DL	DX	010	DL	EDX
011	BL	BX	011	BL	EBX
100	AH	SP	100	AH	ESP
101	CH	BP	101	CH	EBP
110	DH	SI	110	DH	ESI
111	BH	DI	111	BH	EDI

36.7.2 Encoding of Operand Size Bit (w)

The current operand-size attribute determines whether the processor is performing 16- or 32-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 (16 bits or 32 bits). Table 36-7 shows the encoding of the w bit depending on the current operand-size attribute.

Table 36-7. 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

36.7.3 Sign Extend (s) Bit

The sign-extend (s) bit occurs primarily in instructions with immediate data fields that are being extended from 8 bits to 16 or 32 bits. Table 36-8 shows the encoding of the s bit.

Table 36-8. 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

36.7.4 Segment Register Field (sreg)

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 36-9 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 36-9. Encoding of the Segment Register (sreg) Field

2-Bit sreg2 Field	Segment Register Selected	3-Bit sreg3 Field	Segment Register Selected
00	ES	000	ES
01	CS	001	CS
10	SS	010	SS
11	DS	011	DS
		100	FS
		101	GS
		110	Reserved*
		111	Reserved*

* Do not use reserved encodings.

36.7.5 Special-Purpose Register (eee) Field

When the control or debug registers are referenced in an instruction they are encoded in the eee field, which is located in bits 5, 4, and 3 of the ModR/M byte. Table 36-10 shows the encoding of the eee field.

Table 36-10. Encoding of Special-Purpose Register (eee) Field

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

* Do not use reserved encodings.

36.7.6 Condition Test Field (ttn)

For conditional instructions (such as conditional jumps and set on condition), the condition test field (ttn) is encoded for the condition being tested for. 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 ttn field is located in bits 3,2,1, and 0 of the opcode byte; for 2-byte primary opcodes, the ttn field is located in bits 3,2,1, and 0 of the second opcode byte. Table 36-11 shows the encoding of the ttn field.

Table 36-11. Encoding of Conditional Test (ttn) 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

36.7.7 Direction (d) Bit

In many two-operand instructions, a direction bit (d) indicates which operand is considered the source and which is the destination. Table 36-12 shows the encoding of the d bit. When used for integer instructions, the d bit is located at bit 1 of a 1-byte primary opcode. This bit does not appear as the symbol “d” in Table 36-13; instead, 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 36-12. 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

36.8 Integer Instruction Formats and Encodings

Table 36-13 shows the formats and encodings of the integer instructions.

Table 36-13. Integer Instruction Formats and Encodings (Sheet 1 of 12)

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	

Table 36-13. Integer Instruction Formats and Encodings (Sheet 2 of 12)

Instruction and Format	Encoding
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	
immediate to AL, AX, or EAX	0010 010w : immediate data
immediate to memory	1000 00sw : mod 100 r/m : immediate data
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 reg r/m
BSF – Bit Scan Forward	
register1, register2	0000 1111 : 1011 1100 : 11 reg2 reg1
memory, register	0000 1111 : 1011 1100 : mod reg r/m
BSR – Bit Scan Reverse	
register1, register2	0000 1111 : 1011 1101 : 11 reg2 reg1
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

Table 36-13. Integer Instruction Formats and Encodings (Sheet 3 of 12)

Instruction and Format	Encoding
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
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
CMOVcc – Conditional Move	
register2 to register1	0000 1111: 0100 ttn : 11 reg1 reg2
memory to register	0000 1111: 0100 ttn : mod mem r/m
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
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
CMPXCHG8B – Compare and Exchange 8 Bytes	
memory, register	0000 1111 : 1100 0111 : 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	

Table 36-13. Integer Instruction Formats and Encodings (Sheet 4 of 12)

Instruction and Format	Encoding
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
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	1111 011w : 11 111 reg
AL, AX, or EAX by memory	1111 011w : mod 111 r/m
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
IRET/IRETD – Interrupt Return	1100 1111

Table 36-13. Integer Instruction Formats and Encodings (Sheet 5 of 12)

Instruction and Format	Encoding
Jcc – Jump if Condition is Met	
8-bit displacement	0111 ttn : 8-bit displacement
full displacement	0000 1111 : 1000 ttn : 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 AH Register	1001 1111
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 reg r/m
LEA – Load Effective Address	1000 1101 : mod reg r/m
LEAVE – High Level Procedure Exit	1100 1001
LES – Load Pointer to ES	1100 0100 : mod reg r/m
LFS – Load Pointer to FS	0000 1111 : 1011 0100 : mod reg r/m
LGDT – Load Global Descriptor Table Register	0000 1111 : 0000 0001 : mod 010 r/m
LGS – Load Pointer to GS	0000 1111 : 1011 0101 : mod reg r/m
LIDT – Load Interrupt Descriptor Table Register	
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

Table 36-13. Integer Instruction Formats and Encodings (Sheet 6 of 12)

Instruction and Format	Encoding
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 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
AL, AX, or EAX to memory	1010 001w : full displacement
MOV – Move to/from Control Registers	
CR0 from register	0000 1111 : 0010 0010 : 11 000 reg
CR2 from register	0000 1111 : 0010 0010 : 11 010reg
CR3 from register	0000 1111 : 0010 0010 : 11 011 reg
CR4 from register	0000 1111 : 0010 0010 : 11 100 reg
register from CR0-CR4	0000 1111 : 0010 0000 : 11 eee reg
MOV – Move to/from Debug Registers	
DR0-DR3 from register	0000 1111 : 0010 0011 : 11 eee reg
DR4-DR5 from register	0000 1111 : 0010 0011 : 11 eee reg
DR6-DR7 from register	0000 1111 : 0010 0011 : 11 eee reg
register from DR6-DR7	0000 1111 : 0010 0001 : 11 eee reg
register from DR4-DR5	0000 1111 : 0010 0001 : 11 eee reg
register from DR0-DR3	0000 1111 : 0010 0001 : 11 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

Table 36-13. Integer Instruction Formats and Encodings (Sheet 7 of 12)

Instruction and Format	Encoding
MOVS/MOVSb/MOVSsw/MOVSd – Move Data from String to String	1010 010w
MOVSX – Move with Sign-Extend	
register2 to register1	0000 1111 : 1011 111w : 11 reg1 reg2
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 reg
NEG – Two's Complement Negation	
register	1111 011w : 11 011 reg
memory	1111 011w : mod 011 r/m
NOP – No Operation	1001 0000
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	
segment register CS, DS, ES	000 sreg2 111
segment register SS	000 sreg2 111
segment register FS, GS	0000 1111: 10 sreg3 001

Table 36-13. Integer Instruction Formats and Encodings (Sheet 8 of 12)

Instruction and Format	Encoding
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
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
RDPMS – Read Performance Monitoring Counters	0000 1111 : 0011 0011
RDTS – Read Time-Stamp Counter	0000 1111 : 0011 0001
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

Table 36-13. Integer Instruction Formats and Encodings (Sheet 9 of 12)

Instruction and Format	Encoding
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
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

Table 36-13. Integer Instruction Formats and Encodings (Sheet 10 of 12)

Instruction and Format	Encoding
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	1101 111w
SETcc – Byte Set on Condition	
register	0000 1111 : 1001 ttn : 11 000 reg
memory	0000 1111 : 1001 ttn : mod 000 r/m
SGDT – Store Global Descriptor Table Register	0000 1111 : 0000 0001 : mod 000 r/m
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

Table 36-13. Integer Instruction Formats and Encodings (Sheet 11 of 12)

Instruction and Format	Encoding
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 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
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
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

Table 36-13. Integer Instruction Formats and Encodings (Sheet 12 of 12)

Instruction and Format	Encoding
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
AL, 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
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

36.9 MMX™ Instruction Formats and Encodings

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

36.9.1 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 36-14 shows the encoding of this gg field.

Table 36-14. Encoding of Granularity of Data Field (gg)

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

36.9.2 MMX™ and General-Purpose Register Fields (mmxreg and reg)

When MMX 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). Table 36-15 shows the 3-bit encodings used for mmxreg fields.

Table 36-15. Encoding of the MMX™ Register Field (mmxreg)

mmxreg Field Encoding	MMX™ Register
000	MM0
001	MM1
010	MM2
011	MM3
100	MM4
101	MM5
110	MM6
111	MM7

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. Table 36-16 shows the encoding of general-purpose registers when used in MMX instructions.

Table 36-16. Encoding of the General-Purpose Register Field (reg) When Used in MMX™ Instructions.

reg Field Encoding	Register Selected
000	EAX
001	ECX
010	EDX
011	EBX
100	ESP

Table 36-16. Encoding of the General-Purpose Register Field (reg) When Used in MMX™ Instructions.

reg Field Encoding	Register Selected
101	EBP
110	ESI
111	EDI

36.9.3 MMX™ Instruction Formats and Encodings Table

Table 36-17 shows the formats and encodings for MMX instructions for the data types supported—packed byte (B), packed word (W), packed doubleword (D), and quadword (Q). Figure 36-3 describes the nomenclature used in columns (3 through 6) of the table.

Figure 36-3. Key to Codes for MMX™ Data Type Cross-Reference

Code	Meaning
Y	Supported
N	Not supported
O	Output
I	Input
n/a	Not Applicable

Table 36-17. MMX™ Instruction Formats and Encodings (Sheet 1 of 4)

Instruction and Format	Encoding	B	W	D	Q
EMMS - Empty MMX™ state	0000 1111:01110111	n/a	n/a	n/a	n/a
MOVD - Move doubleword		N	N	Y	N
reg to mmxreg	0000 1111:01101110: 11 mmxreg reg				
reg from mmxreg	0000 1111:01111110: 11 mmxreg reg				
mem to mmxreg	0000 1111:01101110: mod mmxreg r/m				
mem from mmxreg	0000 1111:01111110: mod mmxreg r/m				
MOVQ - Move quadword		N	N	N	Y
mmxreg2 to mmxreg1	0000 1111:01101111: 11 mmxreg1 mmxreg2				
mmxreg2 from mmxreg1	0000 1111:01111111: 11 mmxreg1 mmxreg2				
mem to mmxreg	0000 1111:01101111: mod mmxreg r/m				
mem from mmxreg	0000 1111:01111111: mod mmxreg r/m				
PACKSSDW ¹ - Pack dword to word data (signed with saturation)		n/a	O	I	n/a
mmxreg2 to mmxreg1	0000 1111:01101011: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:01101011: mod mmxreg r/m				
PACKSSWB ¹ - Pack word to byte data (signed with saturation)		O	I	n/a	n/a

Table 36-17. MMX™ Instruction Formats and Encodings (Sheet 2 of 4)

Instruction and Format	Encoding	B	W	D	Q
mmxreg2 to mmxreg1	0000 1111:01100011: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:01100011: mod mmxreg r/m				
PACKUSWB ¹ - Pack word to byte data (unsigned with saturation)		O	I	n/a	n/a
mmxreg2 to mmxreg1	0000 1111:01100111: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:01100111: mod mmxreg r/m				
PADD - Add with wrap-around		Y	Y	Y	N
mmxreg2 to mmxreg1	0000 1111: 111111gg: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111: 111111gg: mod mmxreg r/m				
PADDs - Add signed with saturation		Y	Y	N	N
mmxreg2 to mmxreg1	0000 1111: 111011gg: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111: 111011gg: mod mmxreg r/m				
PADDUS - Add unsigned with saturation		Y	Y	N	N
mmxreg2 to mmxreg1	0000 1111: 110111gg: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111: 110111gg: mod mmxreg r/m				
PAND - Bitwise And		N	N	N	Y
mmxreg2 to mmxreg1	0000 1111:11011011: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:11011011: mod mmxreg r/m				
PANDN - Bitwise AndNot		N	N	N	Y
mmxreg2 to mmxreg1	0000 1111:11011111: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:11011111: mod mmxreg r/m				
PCMPEQ - Packed compare for equality		Y	Y	Y	N
mmxreg1 with mmxreg2	0000 1111:011101gg: 11 mmxreg1 mmxreg2				
mmxreg with memory	0000 1111:011101gg: mod mmxreg r/m				
PCMPGT - Packed compare greater (signed)		Y	Y	Y	N
mmxreg1 with mmxreg2	0000 1111:011001gg: 11 mmxreg1 mmxreg2				
mmxreg with memory	0000 1111:011001gg: mod mmxreg r/m				
PMADD - Packed multiply add		n/a	I	O	n/a
mmxreg2 to mmxreg1	0000 1111:11110101: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:11110101: mod mmxreg r/m				
PMULH - Packed multiplication		N	Y	N	N
mmxreg2 to mmxreg1	0000 1111:11100101: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:11100101: mod mmxreg r/m				
PMULL - Packed multiplication		N	Y	N	N
mmxreg2 to mmxreg1	0000 1111:11010101: 11 mmxreg1 mmxreg2				

Table 36-17. MMX™ Instruction Formats and Encodings (Sheet 3 of 4)

Instruction and Format	Encoding	B	W	D	Q
memory to mmxreg	0000 1111:11010101: mod mmxreg r/m				
POR - Bitwise Or		N	N	N	Y
mmxreg2 to mmxreg1	0000 1111:11101011: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:11101011: mod mmxreg r/m				
PSLL ² - Packed shift left logical		N	Y	Y	Y
mmxreg1 by mmxreg2	0000 1111:111100gg: 11 mmxreg1 mmxreg2				
mmxreg by memory	0000 1111:111100gg: mod mmxreg r/m				
mmxreg by immediate	0000 1111:011100gg: 11 110 mmxreg: imm8 data				
PSRA ² - Packed shift right arithmetic		N	Y	Y	N
mmxreg1 by mmxreg2	0000 1111:111000gg: 11 mmxreg1 mmxreg2				
mmxreg by memory	0000 1111:111000gg: mod mmxreg r/m				
mmxreg by immediate	0000 1111:011100gg: 11 100 mmxreg: imm8 data				

Table 36-17. MMX™ Instruction Formats and Encodings (Sheet 4 of 4)

Instruction and Format	Encoding	B	W	D	Q
PSRL ² - Packed shift right logical		N	Y	Y	Y
mmxreg1 by mmxreg2	0000 1111:110100gg: 11 mmxreg1 mmxreg2				
mmxreg by memory	0000 1111:110100gg: mod mmxreg r/m				
mmxreg by immediate	0000 1111:011100gg: 11 010 mmxreg: imm8 data				
PSUB - Subtract with wrap-around		Y	Y	Y	N
mmxreg2 from mmxreg1	0000 1111:111110gg: 11 mmxreg1 mmxreg2				
memory from mmxreg	0000 1111:111110gg: mod mmxreg r/m				
PSUBS - Subtract signed with saturation		Y	Y	N	N
mmxreg2 from mmxreg1	0000 1111:111010gg: 11 mmxreg1 mmxreg2				
memory from mmxreg	0000 1111:111010gg: mod mmxreg r/m				
PSUBUS - Subtract unsigned with saturation		Y	Y	N	N
mmxreg2 from mmxreg1	0000 1111:110110gg: 11 mmxreg1 mmxreg2				
memory from mmxreg	0000 1111:110110gg: mod mmxreg r/m				
PUNPCKH - Unpack high data to next larger type		Y	Y	Y	N
mmxreg2 to mmxreg1	0000 1111:011010gg: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:011010gg: mod mmxreg r/m				
PUNPCKL - Unpack low data to next larger type		Y	Y	Y	N
mmxreg2 to mmxreg1	0000 1111:011000gg: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:011000gg: mod mmxreg r/m				
PXOR - Bitwise Xor		N	N	N	Y
mmxreg2 to mmxreg1	0000 1111:11101111: 11 mmxreg1 mmxreg2				
memory to mmxreg	0000 1111:11101111: mod mmxreg r/m				

NOTE:

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.

36.10 Floating-Point Instruction Formats and Encodings

Table 36-18 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 36-18. General Floating-Point Instruction Formats

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

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 36-19 shows the formats and encodings of the floating-point instructions.

Table 36-19. Floating-Point Instruction Formats and Encodings (Sheet 1 of 5)

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

Table 36-19. Floating-Point Instruction Formats and Encodings (Sheet 2 of 5)

Instruction and Format	Encoding
FCFS – Change Sign	11011 001 : 1110 0000
FCLEX – Clear Exceptions	11011 011 : 1110 0010
FCMOVcc – Conditional Move on EFLAGS 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)
move if not below or equal (NBE)	11011 011 : 11 010 ST(i)
move if not unordered (NU)	11011 011 : 11 011 ST(i)
FCOM – Compare Real	
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
FCOMI – Compare Real and Set EFLAGS	11011 011 : 11 110 ST(i)
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)

Table 36-19. Floating-Point Instruction Formats and Encodings (Sheet 3 of 5)

Instruction and Format	Encoding
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	
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	
ST(0) ← ST(0) + 16-bit memory	11011 110 : mod 111 r/m
ST(0) ← ST(0) + 32-bit memory	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	
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	
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	
ST(0) ← ST(0) + 16-bit memory	11011 110 : mod 101 r/m
ST(0) ← ST(0) + 32-bit memory	11011 010 : mod 101 r/m
FLD – Load Real	

Table 36-19. Floating-Point Instruction Formats and Encodings (Sheet 4 of 5)

Instruction and Format	Encoding
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 $\log_2(\epsilon)$ 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 $\log_e(2)$ 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	
ST(0) \leftarrow ST(0) \times 32-bit memory	11011 000 : mod 001 r/m
ST(0) \leftarrow ST(0) \times 64-bit memory	11011 100 : mod 001 r/m
ST(d) \leftarrow ST(0) \times ST(i)	11011 d00 : 1100 1 ST(i)
FMULP – Multiply	
ST(0) \leftarrow ST(0) \times 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	

Table 36-19. Floating-Point Instruction Formats and Encodings (Sheet 5 of 5)

Instruction and Format	Encoding
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) \leftarrow ST(0) – 32-bit memory	11011 000 : mod 100 r/m
ST(0) \leftarrow ST(0) – 64-bit memory	11011 100 : mod 100 r/m
ST(d) \leftarrow ST(0) – ST(i)	11011 d00 : 1110 R ST(i)
FSUBP – Subtract and Pop	
ST(0) \leftarrow ST(0) – ST(i)	11011 110 : 1110 1 ST(i)
FSUBR – Reverse Subtract	
ST(0) \leftarrow 32-bit memory – ST(0)	11011 000 : mod 101 r/m
ST(0) \leftarrow 64-bit memory – ST(0)	11011 100 : mod 101 r/m
ST(d) \leftarrow ST(i) – ST(0)	11011 d00 : 1110 R ST(i)
FSUBRP – Reverse Subtract and Pop	
ST(i) \leftarrow 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)
FYXTRACT – Extract Exponent and Significand	11011 001 : 1111 0100
FYL2X – ST(1) $\times \log_2$ (ST(0))	11011 001 : 1111 0001
FYL2XP1 – ST(1) $\times \log_2$(ST(0) + 1.0)	11011 001 : 1111 1001
FWAIT – Wait until FPU Ready	1001 1011