X86-64 Instruction Encoding

From OSDev Wiki

This article describes how x86 and x86-64 instructions are encoded.

Contents

General Overview

An x86-64 instruction may be at most 15 bytes in length. It consists of the following components in the given order, where the prefixes are at the least-significant (lowest) address in memory:

- Legacy prefixes (1-4 bytes, optional)
- Opcode with prefixes (1-4 bytes, required)
- ModR/M (1 byte, if required)
- SIB (1 byte, if required)
- Displacement (1, 2, 4 or 8 bytes, if required)
- Immediate (1, 2, 4 or 8 bytes, if required)

Registers

The registers are encoded using the 4-bit values in the X.Reg column of the following table. *X.Reg* is in binary.

X.Reg	8-bit GP	16-bit GP	32-bit GP	64-bit GP	80-bit x87	64-bit MMX	128-bit XMM	256-bit YMM	16-bit Segment	32-bit Control	32-bit Debug
0.000 (0)	AL	AX	EAX	RAX	ST0	MMX0	XMM0	YMM0	ES	CR0	DR0
0.001 (1)	CL	CX	ECX	RCX	ST1	MMX1	XMM1	YMM1	CS	CR1	DR1
0.010 (2)	DL	DX	EDX	RDX	ST2	MMX2	XMM2	YMM2	SS	CR2	DR2
0.011 (3)	BL	BX	EBX	RBX	ST3	MMX3	XMM3	YMM3	DS	CR3	DR3
0.100 (4)	AH, SPL[1]	SP	ESP	RSP	ST4	MMX4	XMM4	YMM4	FS	CR4	DR4
0.101 (5)	CH, BPL[1]	BP	EBP	RBP	ST5	MMX5	XMM5	YMM5	GS	CR5	DR5
0.110 (6)	DH, SIL[1]	SI	ESI	RSI	ST6	MMX6	XMM6	YMM6	-	CR6	DR6
0.111 (7)	BH, DIL[1]	DI	EDI	RDI	ST7	MMX7	XMM7	YMM7	-	CR7	DR7
1.000 (8)	R8L	R8W	R8D	R8	-	MMX0	XMM8	YMM8	ES	CR8	DR8
1.001 (9)	R9L	R9W	R9D	R9	-	MMX1	XMM9	YMM9	CS	CR9	DR9
1.010 (10)	R10L	R10W	R10D	R10	-	MMX2	XMM10	YMM10	SS	CR10	DR10
1.011 (11)	R11L	R11W	R11D	R11	-	MMX3	XMM11	YMM11	DS	CR11	DR11
1.100 (12)	R12L	R12W	R12D	R12	-	MMX4	XMM12	YMM12	FS	CR12	DR12
1.101 (13)	R13L	R13W	R13D	R13	-	MMX5	XMM13	YMM13	GS	CR13	DR13

1.110 (14)	R14L	R14W	R14D	R14	-	MMX6	XMM14	YMM14	-	CR14	DR14
1.111 (15)	R15L	R15W	R15D	R15	-	MMX7	XMM15	YMM15	-	CR15	DR15

1: When any REX prefix is used, SPL, BPL, SIL and DIL are used. Otherwise, without any REX prefix AH, CH, DH and BH are used.

Legacy Prefixes

Each instruction can have up to four prefixes. Sometimes a prefix is required for the instruction while it loses it's original meaning (i.e. a 'mandatory prefix'). The following prefixes can be used, the order does not matter:

- Prefix group 1
  - 0xF0: LOCK prefix
  - 0xF2: REPNE/REPNZ prefix
  - 0xF3: REP or REPE/REPZ prefix
- Prefix group 2
  - 0x2E: CS segment override
  - 0x36: SS segment override
  - 0x3E: DS segment override
  - 0x26: ES segment override
  - 0x64: FS segment override
  - 0x65: GS segment override
  - 0x2E: Branch not taken
  - 0x3E: Branch taken
- Prefix group 3
  - 0x66: Operand-size override prefix
- Prefix group 4
  - 0x67: Address-size override prefix

When there are two or more prefixes from a single group, the behavior is undefined. Some processors ignore the subsequent prefixes from the same group, or use only the last prefix specified for any group.

LOCK prefix

With the LOCK prefix, certain read-modify-write instructions are executed atomically. The LOCK prefix can only be used with the following instructions or an Invalid Opcode Exception occurs: ADC, ADD, AND, BTC, BTR, BTS, CMPXCHG, CMPXCHG8B, CMPXCHG16B, DEC, INC, NEG, NOT, OR, SBB, SUB, XADD, XCHG and XOR.

REPNE/REPNZ, REP and REPE/REPZ prefixes

The repeat prefixes cause string handling instructions to be repeated.

The **REP** prefix will repeat the associated instruction up to CX times, decreasing CX with every repetition. It can be used with the INS, LODS, MOVS, OUTS and STOS instructions.

REPE and **REPZ** are synonyms and repeat the instruction until CX reaches 0 or when ZF is set to 0. It can be

used with the CMPS, CMPSB, CMPSD, CMPSW, SCAS, SCASB, SCASD and SCASW instructions.

REPNE and **REPNZ** also are synonyms and repeat the instruction until CX reaches 0 or when ZF is set to 1. It can be used with the CMPS, CMPSB, CMPSD, CMPSW, SCAS, SCASB, SCASD and SCASW instructions

CS, SS, DS, ES, FS and GS segment override prefixes

Segment overrides are used with instructions that reference non-stack memory. The default segment is implied by the instruction, and using a specific override forces the use of the specified segment for memory operands.

In 64-bit the CS, SS, DS and ES segment overrides are ignored.

Branch taken/not taken prefixes

Branch hints may be used to lessen the impact of branch misprediction somewhat. The 'branch taken' hint is a strong hint, while the 'branch not taken' hint is a weak hint. The branch hints are only supported by Intel since the Pentium 4. Whether using them on AMD architectures has any (positive or negative) effect at all is not known.

Operand-size and address-size override prefix

The default operand-size and address-size can be overridden using these prefix. See the following table:

	CS.d	REX.W	Prefix (0x66 if operand, 0x67 if address)	Operand size	Address size
Real mode / Virtual 8086 mode	N/A	N/A	No	16-bit	16-bit
	N/A	N/A	Yes	32-bit	32-bit
Protected mode / Long compatibility mode	0	N/A	No	16-bit	16-bit
	0	N/A	Yes	32-bit	32-bit
	1	N/A	No	32-bit	32-bit
	1	N/A	Yes	16-bit	16-bit
Long 64-bit mode	Ignored	0	No	32-bit	64-bit
	Ignored	0	Yes	16-bit	32-bit
	Ignored	1	No	64-bit[1]	64-bit
	Ignored	1	Yes	64-bit	32-bit

1: Certain instructions default to (or are fixed at) 64-bit operands and do not need the REX prefix for this, see this table.

NASM

NASM determines the operand size by looking at the *MODRM.reg* or (for a register) *MODRM.rm* fields. When they are both 32-bit, the operand size becomes 32-bit. Same for 16-bit and 64-bit. When they differ, an error occurs at compile time. The address size is determined by looking at (for a memory operand) the *MODRM.rm* field, or the *SIB.base*, *SIB.index* and displacement, in that order. So when *SIB.base* uses a 16-bit register (such as AX), the address size becomes 16-bit. Using a 32-bit displacement will result in the displacement being

truncated.

Opcode

The x86-64 instruction set defines many opcodes and many ways to encode them, depending on several factors.

Legacy opcodes

Legacy (and x87) opcodes consist of, in this order:

- mandatory prefix;
- REX prefix;
- opcode.

Mandatory prefix

Certain instructions (most notably the SIMD instructions) require a mandatory prefix (0x66, 0xF2 or 0xF3), which looks like a normal modifier prefix. When a mandatory prefix is required, it is put with the modifier prefixes before the REX prefix (if any).

REX prefix

The REX prefix is only available in long mode.

Usage

A REX prefix must be encoded when:

- using 64-bit operand size and the instruction does not default to 64-bit operand size; or
- using one of the extended registers (R8 to R15, XMM8 to XMM15, YMM8 to YMM15, CR8 to CR15 and DR8 to DR15); or
- using one of the uniform byte registers SPL, BPL, SIL or DIL.

A REX prefix must not be encoded when:

- using one of the high byte registers AH, CH, BH or DH.

In all other cases, the REX prefix is ignored. The use of multiple REX prefixes is undefined, although processors seem to use only the last REX prefix.

Instructions that default to 64-bit operand size in long mode are:

CALL (near)	ENTER	Jcc
JrCXZ	JMP (near)	LEAVE
LGDT	LIDT	LLDT
LOOP	LOOPcc	LTR
MOV CR(n)	MOV DR(n)	POP reg/mem

POP reg	POP FS	POP GS
POPFQ	PUSH imm8	PUSH imm32
PUSH reg/mem	PUSH reg	PUSH FS
PUSH GS	PUSHFQ	RET (near)

Encoding

The layout is as follows:

Field	Length	Description
0100	4 bits	Fixed bit pattern
W	1 bit	When 1, a 64-bit operand size is used. Otherwise, when 0, the default operand size is used (which is 32-bit for most but not all instructions, see this table).
R	1 bit	This 1-bit value is an extension to the *MODRM.reg* field. See Registers.
X	1 bit	This 1-bit value is an extension to the *SIB.index* field. See 64-bit addressing.
B	1 bit	This 1-bit value is an extension to the *MODRM.rm* field or the *SIB.base* field. See 64-bit addressing.

Because the first four bits always equal 4, the existence of the REX prefix wipes out opcodes 0x40-0x4F, which were previously individual increment and decrement instructions for all eight registers. The Intel 64 and IA-32 Architectures Software Developer's Manual volume 2 states "The single-byte-opcode forms of the INC/DEC instructions are not available in 64-bit mode. INC/DEC functionality is still available using ModR/M forms of the same instructions (opcodes FF/0 and FF/1)."

Opcode

The opcode can be 1, 2 or 3 bytes in length. Depending on the opcode escape sequence, a different opcode map is selected. Possible opcode sequences are:

- <op>
- 0x0F <op>
- 0x0F 0x38 <op>
- 0x0F 0x3A <op>

Note that opcodes can specify that the REG field in the ModR/M byte is fixed at a particular value.

VEX/XOP opcodes

A VEX/XOP prefix must be encoded when:

- the instruction has only its VEX/XOP opcode and no legacy opcode; or

- 256-bit YMM registers are used; or
- more than three operands are used (e.g. *nondestructive-source operations*); or
- when using 128-bit XMM destination registers, bits 128-255 of the corresponding YMM register must be cleared.

A VEX/XOP prefix must not be encoded when:

- when using 128-bit XMM destination registers, bits 128-255 of the corresponding YMM register must not be changed.

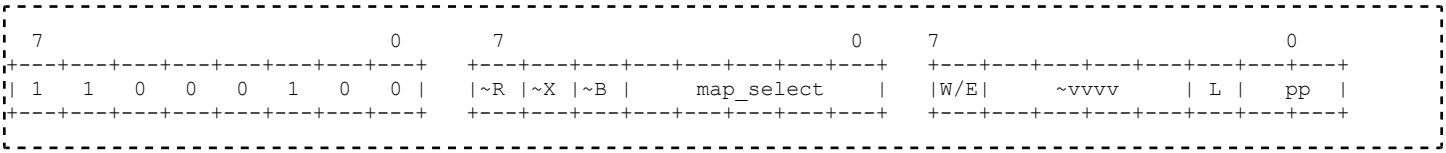
There are many VEX and XOP instructions, all of which can be encoded using the three byte VEX/XOP escape prefix. The VEX and XOP escape prefixes use fields with the following semantics:

<table>
<tr><th>Field</th><th>Length</th><th>Description</th></tr>
<tr><td>VEX/XOP prefix</td><td>8 bits</td><td>Prefix.
<table>
<tr><th>Prefix</th><th>Opcode map and encoding</th></tr>
<tr><td>0xC4</td><td>Three-byte VEX</td></tr>
<tr><td>0xC5</td><td>Two-byte VEX</td></tr>
<tr><td>0x8F</td><td>Three-byte XOP</td></tr>
</table></td></tr>
<tr><td>~R</td><td>1 bit</td><td>This 1-bit value is an 'inverted' extension to the MODRM.reg field. The inverse of REX.R. See Registers.</td></tr>
<tr><td>~X</td><td>1 bit</td><td>This 1-bit value is an 'inverted' extension to the SIB.index field. The inverse of REX.X. See 64-bit addressing.</td></tr>
<tr><td>~B</td><td>1 bit</td><td>This 1-bit value is an 'inverted' extension to the MODRM.rm field or the SIB.base field. The inverse of REX.B. See 64-bit addressing.</td></tr>
<tr><td>map_select</td><td>5 bits</td><td>Specifies the opcode map to use.</td></tr>
<tr><td>W/E</td><td>1 bit</td><td>For integer instructions: when 1, a 64-bit operand size is used; otherwise, when 0, the default operand size is used (equivalent with REX.W). For non-integer instructions, this bit is a general opcode extension bit.</td></tr>
<tr><td>~vvvv</td><td>4 bits</td><td>An additional operand for the instruction. The value of the XMM or YMM register (see Registers) is 'inverted'.</td></tr>
<tr><td>L</td><td>1 bit</td><td>When 0, a 128-bit vector lengh is used. Otherwise, when 1, a 256-bit vector length is used.</td></tr>
</table>

pp	2 bits	Specifies an implied mandatory prefix for the opcode. **Value (binary)** / **Implied mandatory prefix** 00 / none 01 / 0x66 10 / 0xF3 11 / 0xF2

Three byte VEX escape prefix

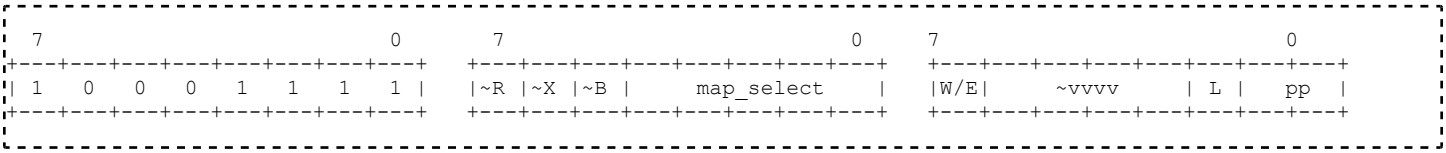
The layout is as follows, starting with a byte with value 0xC4:



A VEX instruction whose values for certain fields are VEX.~X == 1, VEX.~B == 1, VEX.W/E == 0 and map_select == b00001 may be encoded using the two byte VEX escape prefix.

Three byte XOP escape prefix

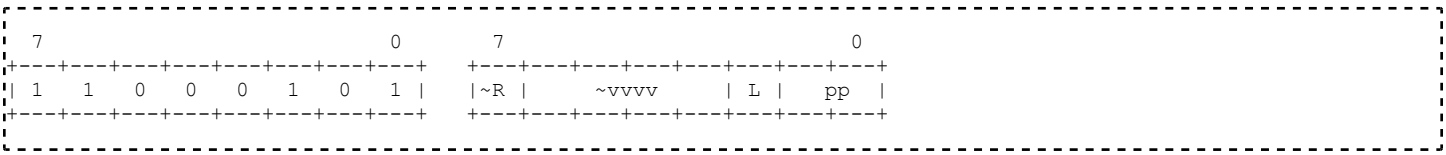
The layout is the same as the three-byte VEX escape prefix, but with initial byte value 0x8F:



The AMD64 Architecture Programmer's Manual Volume 6 states that the map_select field must be equal to or greater than 8, to differentiate the XOP prefix from the POP instruction that formerly used opcode 0x8F.

Two byte VEX escape prefix

A VEX instruction whose values for certain fields are VEX.~X == 1, VEX.~B == 1, VEX.W/E == 0 and map_select == b00001 may be encoded using the two byte VEX escape prefix. The layout is as follows:



3DNow! opcodes

3DNow! opcodes consist of, in this order:

- fixed opcode;

- (ModR/M, SIB, displacement);
- immediate opcode byte.

Fixed opcode

All 3DNow! opcodes have a fixed two-byte sequence equal to 0x0F 0x0F in the opcode position of the instruction.

Immediate opcode byte

3DNow! instructions encode the actual opcode as an 8-bit immediate value trailing the instruction (thus after the ModR/M, SIB and displacement).

ModR/M and SIB bytes

The ModR/M and SIB bytes are used to encode up to two operands of an instruction, each of which is a direct register or effective memory address.

ModR/M

The ModR/M byte encodes a register or an opcode extension, and a register or a memory address. It has the following fields:

Field	Length	Description
MODRM.mod	2 bits	In general, when this field is b11, then register-direct addressing mode is used; otherwise register-indirect addressing mode is used.
MODRM.reg	3 bits	This field can have one of two values: ▪ A 3-bit opcode extension, which is used by some instructions but has no further meaning other than distinguishing the instruction from other instructions. ▪ A 3-bit register reference, which can be used as the source or the destination of an instruction (depending on the instruction). The referenced register depends on the operand-size of the instruction and the instruction itself. See Registers for the values to use for each of the registers. The REX.R, VEX.~R or XOP.~R field can extend this field with 1 most-significant bit to 4 bits total.
MODRM.rm	3 bits	Specifies a direct or indirect register operand, optionally with a displacement. The REX.B, VEX.~B or XOP.~B field can extend this field with 1 most-significant bit to 4 bits total.

16-bit addressing

These are the meanings of the *Mod* (vertically) and *REX/VEX/XOP.B* and *R/M* bits (horizontally) for 16-bit addressing. *B.R/M* and *Mod* are in binary. The SIB-byte is not used in 16-bit addressing. In *Long processing mode* there is no way to specify 16-bit addresses.

16-bit	**B.R/M**							
Mod	**x.000 AX, R8W**	**x.001 CX, R9W**	**x.010 DX, R10W**	**x.011 BX, R11W**	**x.100 SP, R12W**	**x.101 BP, R13W**	**x.110 SI, R14W**	**x.111 DI, R15W**
00	[BX + SI]	[BX + DI]	[BP + SI]	[BP + DI]	[SI]	[DI]	[disp16]	[BX]
01	[BX + SI + disp8]	[BX + DI + disp8]	[BP + SI + disp8]	[BP + DI + disp8]	[SI + disp8]	[DI + disp8]	[BP + disp8]	[BX + disp8]
10	[BX + SI + disp16]	[BX + DI + disp16]	[BP + SI + disp16]	[BP + DI + disp16]	[SI + disp16]	[DI + disp16]	[BP + disp16]	[BX + disp16]
11	r/m							

32/64-bit addressing

These are the meanings of the *Mod* (vertically) and *REX/VEX/XOP.B* and *R/M* bits (horizontally) for 32 and 64-bit addressing. *B.R/M* and *Mod* are in binary.

32/64-bit	**B.R/M**															
Mod	**0.000 AX**	**0.001 CX**	**0.010 DX**	**0.011 BX**	**0.100 SP**	**0.101 BP**	**0.110 SI**	**0.111 DI**	**1.000 R8**	**1.001 R9**	**1.010 R10**	**1.011 R11**	**1.100 R12**	**1.101 R13**	**1.110 R14**	**1.111 R15**
00	[r/m]				[SIB]	[RIP/EIP[1,2] + disp32]	[r/m]						[SIB]	[RIP/EIP[1,2] + disp32]	[r/m]	
01	[r/m + disp8]				[SIB + disp8]	[r/m + disp8]							[SIB + disp8]	[r/m + disp8]		
10	[r/m + disp32]				[SIB + disp32]	[r/m + disp32]							[SIB + disp32]	[r/m + disp32]		
11	r/m															

1: In protected/compatibility mode, this is just *disp32*, but in long mode this is *[RIP]+disp32* (for 64-bit addresses) or *[EIP]+disp32* (for 32-bit addresses, i.e. with address-size override prefix, see here (http://objectmix.com/asm-x86-asm-370/69055-effect-address-size-prefix-rip-relative-addressing.html)).
2: In long mode, to encode *disp32* as in protected/compatibility mode, use the SIB byte.

RIP/EIP-relative addressing

Addressing in x86-64 can be relative to the current instruction pointer value. This is indicated with the *RIP* (64-bit) and *EIP* (32-bit) instruction pointer registers, which are not otherwise exposed to the program and may not exist physically. RIP-relative addressing allows object files to be location independent.

SIB

The SIB byte has the following fields:

Field	Length	Description
SIB.scale	2 bits	This field indicates the scaling factor of SIB.index, where *s* (as used in the tables) equals $2^{\text{SIB.scale}}$. **SIB.scale** / **factor *s*** b00 / 1 b01 / 2 b10 / 4 b11 / 8
SIB.index	3 bits	The index register to use. See Registers for the values to use for each of the registers. The REX.X, VEX.~X or XOP.~X field can extend this field with 1 most-significant bit to 4 bits total.
SIB.base	3 bits	The base register to use. See Registers for the values to use for each of the registers. The REX.B, VEX.~B or XOP.~B field can extend this field with 1 most-significant bit to 4 bits total.

32/64-bit addressing

The meaning of the SIB byte while using 32 or 64-bit addressing is as follows. The ModR/M byte's *Mod* field and the SIB byte's *index* field are used vertically, the SIB byte's *base* field and REX/VEX/XOP.B bit horizontally. The *s* is the scaling factor. *B.Base*, *X.Index* and *Mod* are in binary.

		B.Base															
Mod	**X.Index**	**0.000 AX**	**0.001 CX**	**0.010 DX**	**0.011 BX**	**0.100 SP**	**0.101[1] BP**	**0.110 SI**	**0.111 DI**	**1.000 R8**	**1.001 R9**	**1.010 R10**	**1.011 R11**	**1.100 R12**	**1.101[1] R13**	**1.110 R14**	**1.111 R15**
00	**0.000 AX**	[base + (index * s)]					[(index * s) + disp32]	[base + (index * s)]							[(index * s) + disp32]	[base + (index * s)]	
	0.001 CX																
	0.010 DX																
	0.011 BX																
	0.100[2] SP	[base]					[disp32]	[base]							[disp32]	[base]	
	0.101 BP	[base + (index * s)]					[(index * s) + disp32]	[base + (index * s)]							[(index * s) + disp32]	[base + (index * s)]	
	0.110 SI																
	0.111 DI																
	1.000 R8																
	1.001 R9																
	1.010 R10																
	1.011 R11																

	1.100 R12																
	1.101 R13																
	1.110 R14																
	1.111 R15																

		B.Base															
Mod	**X.Index**	**0.000 AX**	**0.001 CX**	**0.010 DX**	**0.011 BX**	**0.100 SP**	**0.101 BP**	**0.110 SI**	**0.111 DI**	**1.000 R8**	**1.001 R9**	**1.010 R10**	**1.011 R11**	**1.100 R12**	**1.101 R13**	**1.110 R14**	**1.111 R15**
01	**0.000 AX**	[base + (index * s) + disp8]															
	0.001 CX																
	0.010 DX																
	0.011 BX																
	0.100[2] SP	[base + disp8]															
	0.101 BP	[base + (index * s) + disp8]															
	0.110 SI																
	0.111 DI																
	1.000 R8																
	1.001 R9																
	1.010 R10																
	1.011 R11																
	1.100 R12																
	1.101 R13																
	1.110 R14																
	1.111 R15																

		B.Base															
Mod	**X.Index**	**0.000 AX**	**0.001 CX**	**0.010 DX**	**0.011 BX**	**0.100 SP**	**0.101 BP**	**0.110 SI**	**0.111 DI**	**1.000 R8**	**1.001 R9**	**1.010 R10**	**1.011 R11**	**1.100 R12**	**1.101 R13**	**1.110 R14**	**1.111 R15**
10	**0.000 AX**	[base + (index * s) + disp32]															
	0.001 CX																
	0.010 DX																
	0.011 BX																

0.100^2 SP	[base + disp32]
0.101 BP	[base + (index * s) + disp32]
0.110 SI	
0.111 DI	
1.000 R8	
1.001 R9	
1.010 R10	
1.011 R11	
1.100 R12	
1.101 R13	
1.110 R14	
1.111 R15	

1: No base register is encoded.

2: No index register is encoded.

Displacement

A displacement value is a 1, 2, 4, or 8 byte offset added to the calculated address. When an 8 byte displacement is used, no immediate operand is encoded.

The displacement value, if any, follows the ModR/M and SIB bytes discussed above. When the ModR/M or SIB tables state that a *disp* value is required, or without a ModR/M byte the use of *moffset* (AMD) or *moffs* (Intel) in the mnemonic syntax of the instruction, then the displacement bytes are required.

Immediate

Some instructions require an immediate value. The instruction (and the operand-size column in the above table) determine the length of the immediate value. The *imm8* mnemonic (or 8-bit operand-size) means a one byte immediate value, *imm16* (or 16-bit operand-size) means a two byte immediate value, *imm32* (or 32-bit operand-size) a four byte value and *imm64* (or 64-bit operand-size) an eight byte value. When an 8 byte immediate value is encoded, no displacement can be encoded.

See Also

External References

- AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions (https://www.amd.com/system/files/TechDocs/24594.pdf)
- Intel 64 and IA-32 Architectures Software Developer's Manuals (http://www.intel.com/products/processor/manuals/)