MOV—Move

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
88 /r	MOV *r/m8,r8*	Valid	Valid	Move *r8* to *r/m8.*
REX + 88 /r	MOV *r/m8****,*r8****	Valid	N.E.	Move *r8* to *r/m8.*
89 /r	MOV *r/m16,r16*	Valid	Valid	Move *r16* to *r/m16.*
89 /r	MOV *r/m32,r32*	Valid	Valid	Move *r32* to *r/m32.*
REX.W + 89 /r	MOV *r/m64,r64*	Valid	N.E.	Move *r64* to *r/m64.*
8A /r	MOV *r8,r/m8*	Valid	Valid	Move *r/m8* to *r8.*
REX + 8A /r	MOV *r8***,*r/m8****	Valid	N.E.	Move *r/m8* to *r8.*
8B /r	MOV *r16,r/m16*	Valid	Valid	Move *r/m16* to *r16.*
8B /r	MOV *r32,r/m32*	Valid	Valid	Move *r/m32* to *r32.*
REX.W + 8B /r	MOV *r64,r/m64*	Valid	N.E.	Move *r/m64* to *r64.*
8C /r	MOV *r/m16,Sreg***	Valid	Valid	Move segment register to *r/m16.*
REX.W + 8C /r	MOV *r/m64,Sreg***	Valid	Valid	Move zero extended 16-bit segment register to *r/m64.*
8E /r	MOV *Sreg,r/m16***	Valid	Valid	Move *r/m16* to segment register.
REX.W + 8E /r	MOV *Sreg,r/m64***	Valid	Valid	Move *lower 16 bits of r/m64* to segment register.
A0	MOV AL,*moffs8**	Valid	Valid	Move byte at (*seg:offset*) to AL.
REX.W + A0	MOV AL,*moffs8**	Valid	N.E.	Move byte at (*offset*) to AL.
A1	MOV AX,*moffs16**	Valid	Valid	Move word at (*seg:offset*) to AX.
A1	MOV EAX,*moffs32**	Valid	Valid	Move doubleword at (*seg:offset*) to EAX.
REX.W + A1	MOV RAX,*moffs64**	Valid	N.E.	Move quadword at (*offset*) to RAX.
A2	MOV *moffs8*,AL	Valid	Valid	Move AL to (*seg:offset*).
REX.W + A2	MOV *moffs8****,AL	Valid	N.E.	Move AL to (*offset*).
A3	MOV *moffs16**,AX	Valid	Valid	Move AX to (*seg:offset*).

A3	MOV *moffs32**,EAX	Valid	Valid	Move EAX to (*seg:offset*).

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
REX.W + A3	MOV *moffs64**,RAX	Valid	N.E.	Move RAX to (*offset*).
B0+ *rb*	MOV *r8, imm8*	Valid	Valid	Move *imm8* to *r8.*
REX + B0+ *rb*	MOV *r8*****, *imm8*	Valid	N.E.	Move *imm8* to *r8.*
B8+ *rw*	MOV *r16, imm16*	Valid	Valid	Move *imm16* to *r16.*
B8+ *rd*	MOV *r32, imm32*	Valid	Valid	Move *imm32* to *r32.*
REX.W + B8+ *rd*	MOV *r64, imm64*	Valid	N.E.	Move *imm64* to *r64.*
C6 */0*	MOV *r/m8, imm8*	Valid	Valid	Move *imm8* to *r/m8.*
REX + C6 */0*	MOV *r/m8****, *imm8*	Valid	N.E.	Move *imm8* to *r/m8.*
C7 */0*	MOV *r/m16, imm16*	Valid	Valid	Move *imm16* to *r/m16.*
C7 */0*	MOV *r/m32, imm32*	Valid	Valid	Move *imm32* to *r/m32.*
REX.W + C7 */0*	MOV *r/m64, imm32*	Valid	N.E.	Move *imm32 sign extended to 64-bits* to *r/m64.*

NOTES:

* The *moffs8*, *moffs16*, *moffs32* and *moffs64* operands specify a simple offset relative to the segment base, where 8, 16, 32 and 64 refer to the size of the data. The address-size attribute of the instruction determines the size of the offset, either 16, 32 or 64 bits.

** In 32-bit mode, the assembler may insert the 16-bit operand-size prefix with this instruction (see the following "Description" section for further information).

***In 64-bit mode, *r/m8* can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Description

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

The MOV instruction cannot be used to load the CS register. Attempting to do so results in an invalid opcode exception (#UD). To load the CS register, use the far JMP, CALL, or RET instruction.

If the destination operand is a segment register (DS, ES, FS, GS, or SS), the source operand must be a valid segment selector. In protected mode, moving a segment

selector into a segment register automatically causes the segment descriptor information associated with that segment selector to be loaded into the hidden (shadow) part of the segment register. While loading this information, the segment selector and segment descriptor information is validated (see the "Operation" algorithm below). The segment descriptor data is obtained from the GDT or LDT entry for the specified segment selector.

A NULL segment selector (values 0000-0003) can be loaded into the DS, ES, FS, and GS registers without causing a protection exception. However, any subsequent attempt to reference a segment whose corresponding segment register is loaded with a NULL value causes a general protection exception (#GP) and no memory reference occurs.

Loading the SS register with a MOV instruction inhibits all interrupts until after the execution of the next instruction. This operation allows a stack pointer to be loaded into the ESP register with the next instruction (MOV ESP, **stack-pointer value**) before an interrupt occurs[1]. Be aware that the LSS instruction offers a more efficient method of loading the SS and ESP registers.

When operating in 32-bit mode and moving data between a segment register and a general-purpose register, the 32-bit IA-32 processors do not require the use of the 16-bit operand-size prefix (a byte with the value 66H) with this instruction, but most assemblers will insert it if the standard form of the instruction is used (for example, MOV DS, AX). The processor will execute this instruction correctly, but it will usually require an extra clock. With most assemblers, using the instruction form MOV DS, EAX will avoid this unneeded 66H prefix. When the processor executes the instruction with a 32-bit general-purpose register, it assumes that the 16 least-significant bits of the general-purpose register are the destination or source operand. If the register is a destination operand, the resulting value in the two high-order bytes of the register is implementation dependent. For the Pentium 4, Intel Xeon, and P6 family processors, the two high-order bytes are filled with zeros; for earlier 32-bit IA-32 processors, the two high order bytes are undefined.

In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.R prefix permits access to additional registers (R8-R15). Use of the REX.W prefix promotes operation to 64 bits. See the summary chart at the beginning of this section for encoding data and limits.

---

1. If a code instruction breakpoint (for debug) is placed on an instruction located immediately after a MOV SS instruction, the breakpoint may not be triggered. However, in a sequence of instructions that load the SS register, only the first instruction in the sequence is guaranteed to delay an interrupt.

   In the following sequence, interrupts may be recognized before MOV ESP, EBP executes:

   ```
   MOV SS, EDX
   MOV SS, EAX
   MOV ESP, EBP
   ```