

MOV — Move

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
88 /r	MOV r/m8, r8	MR	Valid	Valid	Move r8 to r/m8.
REX + 88 /r	MOV r/m8 ¹ , r8 ¹	MR	Valid	N.E.	Move r8 to r/m8.
89 /r	MOV r/m16, r16	MR	Valid	Valid	Move r16 to r/m16.
89 /r	MOV r/m32, r32	MR	Valid	Valid	Move r32 to r/m32.
REX.W + 89 /r	MOV r/m64, r64	MR	Valid	N.E.	Move r64 to r/m64.
8A /r	MOV r8, r/m8	RM	Valid	Valid	Move r/m8 to r8.
REX + 8A /r	MOV r8 ¹ , r/m8 ¹	RM	Valid	N.E.	Move r/m8 to r8.
8B /r	MOV r16, r/m16	RM	Valid	Valid	Move r/m16 to r16.
8B /r	MOV r32, r/m32	RM	Valid	Valid	Move r/m32 to r32.
REX.W + 8B /r	MOV r64, r/m64	RM	Valid	N.E.	Move r/m64 to r64.
8C /r	MOV r/m16, Sreg ²	MR	Valid	Valid	Move segment register to r/m16.
8C /r	MOV r16/r32/m16, Sreg ²	MR	Valid	Valid	Move zero extended 16-bit segment register to r16/r32/m16.
REX.W + 8C /r	MOV r64/m16, Sreg ²	MR	Valid	Valid	Move zero extended 16-bit segment register to r64/m16.
8E /r	MOV Sreg, r/m16 ²	RM	Valid	Valid	Move r/m16 to segment register.
REX.W + 8E /r	MOV Sreg, r/m64 ²	RM	Valid	Valid	Move lower 16 bits of r/m64 to segment register.
A0	MOV AL, moffs8 ³	FD	Valid	Valid	Move byte at (seg:offset) to AL.
REX.W + A0	MOV AL, moffs8 ³	FD	Valid	N.E.	Move byte at (offset) to AL.
A1	MOV AX, moffs16 ³	FD	Valid	Valid	Move word at (seg:offset) to AX.
A1	MOV EAX, moffs32 ³	FD	Valid	Valid	Move doubleword at (seg:offset) to EAX.
REX.W + A1	MOV RAX, moffs64 ³	FD	Valid	N.E.	Move quadword at (offset) to RAX.
A2	MOV moffs8, AL	TD	Valid	Valid	Move AL to (seg:offset).
REX.W + A2	MOV moffs8 ¹ , AL	TD	Valid	N.E.	Move AL to (offset).
A3	MOV moffs16 ³ , AX	TD	Valid	Valid	Move AX to (seg:offset).
A3	MOV moffs32 ³ , EAX	TD	Valid	Valid	Move EAX to (seg:offset).
REX.W + A3	MOV moffs64 ³ , RAX	TD	Valid	N.E.	Move RAX to (offset).
B0+ rb ib	MOV r8, imm8	OI	Valid	Valid	Move imm8 to r8.
REX + B0+ rb ib	MOV r8 ¹ , imm8	OI	Valid	N.E.	Move imm8 to r8.
B8+ rw iw	MOV r16, imm16	OI	Valid	Valid	Move imm16 to r16.
B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.
REX.W + B8+ rd io	MOV r64, imm64	OI	Valid	N.E.	Move imm64 to r64.
C6 /0 ib	MOV r/m8, imm8	MI	Valid	Valid	Move imm8 to r/m8.
REX + C6 /0 ib	MOV r/m8 ¹ , imm8	MI	Valid	N.E.	Move imm8 to r/m8.
C7 /0 iw	MOV r/m16, imm16	MI	Valid	Valid	Move imm16 to r/m16.
C7 /0 id	MOV r/m32, imm32	MI	Valid	Valid	Move imm32 to r/m32.
REX.W + C7 /0 id	MOV r/m64, imm32	MI	Valid	N.E.	Move imm32 sign extended to 64-bits to r/m64.

1. 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.

2. 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).

3. 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.

Instruction Operand Encoding ¶

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MR	ModRM:r/m (w)	ModRM:reg (r)	N/A	N/A
RM	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

FD	AL/AX/EAX/RAX	Moffs	N/A	N/A
TD	Moffs (w)	AL/AX/EAX/RAX	N/A	N/A
OI	opcode + rd (w)	imm8/16/32/64	N/A	N/A
MI	ModRM:r/m (w)	imm8/16/32/64	N/A	N/A

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 suppresses or inhibits some debug exceptions and inhibits interrupts on the following instruction boundary. (The inhibition ends after delivery of an exception or the execution of the next instruction.) This behavior allows a stack pointer to be loaded into the ESP register with the next instruction (MOV ESP, **stack-pointer value**) before an event can be delivered. See Section 6.8.3, “Masking Exceptions and Interrupts When Switching Stacks,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. Intel recommends that software use the LSS instruction to load the SS register and ESP together.

When executing MOV Reg, Sreg, the processor copies the content of Sreg to the 16 least significant bits of the general-purpose register. The upper bits of the destination register are zero for most IA-32 processors (Pentium Pro processors and later) and all Intel 64 processors, with the exception that bits 31:16 are undefined for Intel Quark X1000 processors, Pentium, and earlier processors.

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.

Operation ¶

DEST := SRC;

Loading a segment register while in protected mode results in special checks and actions, as described in the following listing. These checks are performed on the segment selector and the segment descriptor to which it points.

```

IF SS is loaded
  THEN
    IF segment selector is NULL
      THEN #GP(0); FI;
    IF segment selector index is outside descriptor table limits
      OR segment selector's RPL ≠ CPL
      OR segment is not a writable data segment
      OR DPL ≠ CPL
      THEN #GP(selector); FI;
    IF segment not marked present
      THEN #SS(selector);
    ELSE
      SS := segment selector;
      SS := segment descriptor; FI;
  FI;
IF DS, ES, FS, or GS is loaded with non-NULL selector
  THEN
    IF segment selector index is outside descriptor table limits
      OR segment is not a data or readable code segment
      OR ((segment is a data or nonconforming code segment) AND ((RPL > DPL) or (CPL > DPL)))
      THEN #GP(selector); FI;
    IF segment not marked present
      THEN #NP(selector);
    ELSE
      SegmentRegister := segment selector;
      SegmentRegister := segment descriptor; FI;
  FI;
IF DS, ES, FS, or GS is loaded with NULL selector
  THEN
    SegmentRegister := segment selector;
    SegmentRegister := segment descriptor;
  FI;

```

Flags Affected ¶

None.

Protected Mode Exceptions ¶

#GP(0)	If attempt is made to load SS register with NULL segment selector.
--------	--------------------------------------------------------------------

	If the destination operand is in a non-writable segment.
	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a NULL segment selector.
#GP(selector)	If segment selector index is outside descriptor table limits.
	If the SS register is being loaded and the segment selector's RPL and the segment descriptor's DPL are not equal to the CPL.
	If the SS register is being loaded and the segment pointed to is a non-writable data segment.
	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.
	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, and either the RPL or the CPL is greater than the DPL.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#SS(selector)	If the SS register is being loaded and the segment pointed to is marked not present.
#NP	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is marked not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If attempt is made to load the CS register.
	If the LOCK prefix is used.

Real-Address Mode Exceptions ¶

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If attempt is made to load the CS register.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions ¶

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If attempt is made to load the CS register.
	If the LOCK prefix is used.

Compatibility Mode Exceptions ¶

Same exceptions as in protected mode.

64-Bit Mode Exceptions ¶

#GP(0)	If the memory address is in a non-canonical form.
	If an attempt is made to load SS register with NULL segment selector when CPL = 3.
	If an attempt is made to load SS register with NULL segment selector when CPL < 3 and CPL ≠ RPL.
#GP(selector)	If segment selector index is outside descriptor table limits.
	If the memory access to the descriptor table is non-canonical.
	If the SS register is being loaded and the segment selector's RPL and the segment descriptor's DPL are not equal to the CPL.
	If the SS register is being loaded and the segment pointed to is a nonwritable data segment.
	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.
	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.
#SS(0)	If the stack address is in a non-canonical form.
#SS(selector)	If the SS register is being loaded and the segment pointed to is marked not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If attempt is made to load the CS register.
	If the LOCK prefix is used.

This UNOFFICIAL, mechanically-separated, non-verified reference is provided for convenience, but it may be incomplete or b₇o_ke in various obvious or non-obvious ways. Refer to [Intel® 64 and IA-32 Architectures](#)

[Software Developer's Manual](#) for anything serious.