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.

NOTES:

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.

MOV—Move to/from Control Registers

Opcode/ Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 20/r MOV r32, CR0–CR7	MR	N.E.	Valid	Move control register to r32.
0F 20/r MOV r64, CR0–CR7	MR	Valid	N.E.	Move extended control register to r64.
REX.R + 0F 20 /0 MOV r64, CR8	MR	Valid	N.E.	Move extended CR8 to r64.[1]
0F 22 /r MOV CR0–CR7, r32	RM	N.E.	Valid	Move r32 to control register.
0F 22 /r MOV CR0–CR7, r64	RM	Valid	N.E.	Move r64 to extended control register.
REX.R + 0F 22 /0 MOV CR8, r64	RM	Valid	N.E.	Move r64 to extended CR8.[1]

NOTES:

1. MOV CR* instructions, except for MOV CR8, are serializing instructions. MOV CR8 is not architecturally defined as a serializing instruction. For more information, see Chapter 9 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

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

Description

Moves the contents of a control register (CR0, CR2, CR3, CR4, or CR8) to a general-purpose register or the contents of a general-purpose register to a control register. The operand size for these instructions is always 32 bits in non-64-bit modes, regardless of the operand-size attribute. On a 64-bit capable processor, an execution of MOV to CR outside of 64-bit mode zeros the upper 32 bits of the control register. (See "Control Registers" in Chapter 2 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for a detailed description of the flags and fields in the control registers.) This instruction can be executed only when the current privilege level is 0.

At the opcode level, the *reg* field within the ModR/M byte specifies which of the control registers is loaded or read. The 2 bits in the *mod* field are ignored. The *r/m* field specifies the general-purpose register loaded or read. Some of the bits in CR0, CR3, and CR4 are reserved and must be written with zeros. Attempting to set any reserved bits in CR0[31:0] is ignored. Attempting to set any reserved bits in CR0[63:32] results in a general-protection exception, #GP(0). When PCIDs are not enabled, bits 2:0 and bits 11:5 of CR3 are not used and attempts to set them are ignored. Attempting to set any reserved bits in CR3[63:MAXPHYADDR] results in #GP(0). Attempting to set any reserved bits in CR4 results in #GP(0). On Pentium 4, Intel Xeon and P6 family processors, CR0.ET remains set after any load of CR0; attempts to clear this bit have no impact.

In certain cases, these instructions have the side effect of invalidating entries in the TLBs and the paging-structure caches. See Section 4.10.4.1, "Operations that Invalidate TLBs and Paging-Structure Caches," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for details.

The following side effects are implementation-specific for the Pentium 4, Intel Xeon, and P6 processor family: when modifying PE or PG in register CR0, or PSE or PAE in register CR4, all TLB entries are flushed, including global entries. Software should not depend on this functionality in all Intel 64 or IA-32 processors.

In 64-bit mode, the instruction's default operation size is 64 bits. The REX.R prefix must be used to access CR8. Use of REX.B permits access to additional registers (R8-R15). Use of the REX.W prefix or 66H prefix is ignored. Use

of the REX.R prefix to specify a register other than CR8 causes an invalid-opcode exception. See the summary chart at the beginning of this section for encoding data and limits.

If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 determines whether the instruction invalidates entries in the TLBs and the paging-structure caches (see Section 4.10.4.1, "Operations that Invalidate TLBs and Paging-Structure Caches," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A). The instruction does not modify bit 63 of CR3, which is reserved and always 0.

See "Changes to Instruction Behavior in VMX Non-Root Operation" in Chapter 26 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C, for more information about the behavior of this instruction in VMX non-root operation.

Operation

```
DEST := SRC;
```

Flags Affected

The OF, SF, ZF, AF, PF, and CF flags are undefined.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0, or setting the CD flag to 0 when the NW flag is set to 1).
	If an attempt is made to write a 1 to any reserved bit in CR4.
	If an attempt is made to write 1 to CR4.PCIDE.
	If any of the reserved bits are set in the page-directory pointers table (PDPT) and the loading of a control register causes the PDPT to be loaded into the processor.
	If an attempt is made to activate IA-32e mode and either the current CS has the L-bit set or the TR references a 16-bit TSS.
#UD	If the LOCK prefix is used.
	If an attempt is made to access CR1, CR5, CR6, CR7, or CR9–CR15.

Real-Address Mode Exceptions

#GP	If an attempt is made to write a 1 to any reserved bit in CR4.
	If an attempt is made to write 1 to CR4.PCIDE.
	If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0).
	If an attempt is made to activate IA-32e mode and either the current CS has the L-bit set or the TR references a 16-bit TSS.
#UD	If the LOCK prefix is used.
	If an attempt is made to access CR1, CR5, CR6, CR7, or CR9–CR15.

Virtual-8086 Mode Exceptions

#GP(0)	These instructions cannot be executed in virtual-8086 mode.

Compatibility Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0, or setting the CD flag to 0 when the NW flag is set to 1).
	If an attempt is made to change CR4.PCIDE from 0 to 1 while CR3[11:0] ≠ 000H.
	If an attempt is made to clear CR0.PG[bit 31] while CR4.PCIDE = 1.
	If an attempt is made to leave IA-32e mode by clearing CR4.PAE[bit 5].

#UD	If the LOCK prefix is used. If an attempt is made to access CR1, CR5, CR6, CR7, or CR9–CR15.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0. If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0, or setting the CD flag to 0 when the NW flag is set to 1). If an attempt is made to change CR4.PCIDE from 0 to 1 while CR3[11:0] ≠ 000H. If an attempt is made to clear CR0.PG[bit 31]. If an attempt is made to write a 1 to any reserved bit in CR4. If an attempt is made to write a 1 to any reserved bit in CR8. If an attempt is made to write a 1 to any reserved bit in CR3[63:MAXPHYADDR]. If an attempt is made to leave IA-32e mode by clearing CR4.PAE[bit 5].
#UD	If the LOCK prefix is used. If an attempt is made to access CR1, CR5, CR6, CR7, or CR9–CR15. If the REX.R prefix is used to specify a register other than CR8.

MOV—Move to/from Debug Registers

Opcode/ Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 21/r MOV r32, DR0–DR7	MR	N.E.	Valid	Move debug register to r32.
0F 21/r MOV r64, DR0–DR7	MR	Valid	N.E.	Move extended debug register to r64.
0F 23 /r MOV DR0–DR7, r32	RM	N.E.	Valid	Move r32 to debug register.
0F 23 /r MOV DR0–DR7, r64	RM	Valid	N.E.	Move r64 to extended debug register.

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

Description

Moves the contents of a debug register (DR0, DR1, DR2, DR3, DR4, DR5, DR6, or DR7) to a general-purpose register or vice versa. The operand size for these instructions is always 32 bits in non-64-bit modes, regardless of the operand-size attribute. (See Section 18.2, "Debug Registers", of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for a detailed description of the flags and fields in the debug registers.)

The instructions must be executed at privilege level 0 or in real-address mode.

When the debug extension (DE) flag in register CR4 is clear, these instructions operate on debug registers in a manner that is compatible with Intel386 and Intel486 processors. In this mode, references to DR4 and DR5 refer to DR6 and DR7, respectively. When the DE flag in CR4 is set, attempts to reference DR4 and DR5 result in an undefined opcode (#UD) exception. (The CR4 register was added to the IA-32 Architecture beginning with the Pentium processor.)

At the opcode level, the *reg* field within the ModR/M byte specifies which of the debug registers is loaded or read. The two bits in the *mod* field are ignored. The *r/m* field specifies the general-purpose register loaded or read.

In 64-bit mode, the instruction's default operation size is 64 bits. Use of the REX.B prefix permits access to additional registers (R8–R15). Use of the REX.W or 66H prefix is ignored. Use of the REX.R prefix causes an invalid-opcode exception. See the summary chart at the beginning of this section for encoding data and limits.

Operation

```
IF ((DE = 1) and (SRC or DEST = DR4 or DR5))
    THEN
        #UD;
    ELSE
        DEST := SRC;
FI;
```

Flags Affected

The OF, SF, ZF, AF, PF, and CF flags are undefined.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
#UD	If CR4.DE[bit 3] = 1 (debug extensions) and a MOV instruction is executed involving DR4 or DR5.
	If the LOCK prefix is used.
#DB	If any debug register is accessed while the DR7.GD[bit 13] = 1.

Real-Address Mode Exceptions

#UD	If CR4.DE[bit 3] = 1 (debug extensions) and a MOV instruction is executed involving DR4 or DR5.
	If the LOCK prefix is used.
#DB	If any debug register is accessed while the DR7.GD[bit 13] = 1.

Virtual-8086 Mode Exceptions

#GP(0)	The debug registers cannot be loaded or read when in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If an attempt is made to write a 1 to any of bits 63:32 in DR6.
	If an attempt is made to write a 1 to any of bits 63:32 in DR7.
#UD	If CR4.DE[bit 3] = 1 (debug extensions) and a MOV instruction is executed involving DR4 or DR5.
	If the LOCK prefix is used.
	If the REX.R prefix is used.
#DB	If any debug register is accessed while the DR7.GD[bit 13] = 1.