

stc cmp subss imul fsqrtstosd jnle psubq unpcklps hswap cmovna inc repnz sbb  
out shr jnle phaddsw mulss psadbw idiv fncstp mov bswap pmaxsw sysenter mfence cmovpe fstp outsw  
fldz aaa phadd add sal pushad movsxd emms hlt rdtsc daa fbld fiadd push verr int bts sgdt hop cwde  
movntdq orpd rcpss xchg not lidt lss comisd setnq pmovmskb mov wait xlat cdq clflush movddup divps pmuludq  
movntdq orpd rcpss xchg not lidt lss comisd setnq pmovmskb mov wait xlat cdq clflush movddup divps pmuludq

# X86 Instruction Reference

## 64-bit Edition

general, system, x87 FPU, MMX, SSE(1), SSE2, SSE3, SSSE3 instructions

ref.x86asm.net

Advertisement



ISBN 978-80-254-2348-6



9 788025 423486

**X86 Instruction Reference, 64-bit Edition**  
**general, system, x87 FPU, MMX, SSE(1), SSE2, SSE3, SSSE3 instructions**

Copyright © MazeGen

Second Edition, September 2008

Errata:

<http://ref.x86asm.net/errata/64/instruction>

Karel Lejska  
Bayerova 8  
Brno 60200  
Czech Republic

Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation, without intent to infringe.

For comments, suggestions, questions or bug reports, please contact [mazegen@gmail.com](mailto:mazegen@gmail.com)

For booking a computer-related ad in this reference, please contact [mazegen@gmail.com](mailto:mazegen@gmail.com)

Credits: Christian Ludloff, Martin Mocko (*vid*), Anthony Lopes, *Aquila*, *EliCZ*, *Cephexin*

ISBN 978-80-254-2348-6

## Quick Guide

- *mnemonic*: Instruction mnemonic itself. If the mnemonic is set up using *italic*, there is no official mnemonic and the present one is just a suggested one
- *op1–op4*: Up to four instruction operands. Implicate operands are set up using *italic*. Modified operands are **bold**. Implicate *[RSP]* operand is not indicated. If the *op4* column contains only three dots '...', there are more than four operands
- *pf*: Prefix value, or if Primary opcode is present, fixed extraordinary prefix
- *0F*: Dedicated for *0x0F* two-byte prefix
- *po*: Primary opcode. Second opcode byte in case of multi-byte opcodes. *+r* means a register code, from 0 through 7, added to the value
- *so*: Secondary opcode. Fixed appended value to the primary opcode
- *o*: Register/Opcode field. Either the value of an opcode extension (values from 0 through 7) or *r* indicates that the ModR/M byte contains a register operand and an r/m operand
- *proc*: Indicates the instruction's introductory processor. If the column is empty, it means that the instruction is supported since first implementation of Intel EM64T architecture.
- *st*: Indicates how is the instruction documented in the Intel manuals. *D* means fully documented. *M* means documented only marginally. *U* undocumented at all. Empty column means *D*
- *m*: Indicates the mode in which is the instruction valid. Virtual-8086 Mode and SMM is not taken into account. *R* applies for real, protected and 64-bit mode. *P* applies for protected and 64-bit mode. *E* applies for 64-bit mode. If this column is empty, it means *R*
- *rl*: The ring level, which is the instruction valid from (3 or 0). *f* indicates that the level depends on further flag(s)
- *x*: For general instructions, *L* indicates that the instruction is basically valid with *LOCK (0xF0)* prefix. For x87 FPU instructions, *s* indicates that the opcode performs additional push of a value to the register stack, *p* indicates that the opcode performs additional pop of the register stack, *P* pops twice
- *iext*: The instruction extension group, which was the opcode released on
- *tested f, modiff, def f, undef f*: For RFlags register, indicates these flags using *odiszapc* pattern. Present flag fits in with the appropriate group. For x87 FPU flags, indicates these flags using *1234* x87 FPU flag pattern. Present flag fits in with the appropriate group.
- *f values*: For RFlags register, indicates the values of flags, which are always set or cleared, using case-sensitive *odiszapc* flag pattern. Lower-case flag means cleared flag, upper-case means set flag. For x87 FPU flags, indicates these flags using *1234* x87 FPU flag pattern. Present flag holds its value
- *description, notes*: Generic description

Visit <http://ref.x86asm.net> for detailed guide.



mnemonic	op1	op2	op3	op4	iext	pf	OF	po	so	o	proc	st	m	rl	x	tested f	modif f	def f	undef f	f values	description, notes	
CBW	<b>AH,</b>	AL																				
CWDE	<b>EAX,</b>	AX						98						E								Convert
CDQE	<b>RAX,</b>	EAX																				
CLC								F8								.....c	.....c			.....c	Clear Carry Flag	
CLD								FC								.d.....	.d.....			.d.....	Clear Direction Flag	
CLI								FA						f <sup>1</sup>		..i.....	..i.....			..i.....	Clear Interrupt Flag	
CMC								F5								.....c	.....c			.....c	Complement Carry Flag	
CMOVB	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVNAE	<b>r16/32/64,</b>	r/m16/32/64						OF 42		r						.....c					Conditional Move - below/not above or equal/carry (CF=1)	
CMOVC	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVBE	<b>r16/32/64,</b>	r/m16/32/64						OF 46		r						....z.c					Conditional Move - below or equal/not above (CF=1 AND ZF=1)	
CMOVNA	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVL	<b>r16/32/64,</b>	r/m16/32/64						OF 4C		r						O..S....					Conditional Move - less/not greater (SF!=OF)	
CMOVNGE	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVLE	<b>r16/32/64,</b>	r/m16/32/64						OF 4E		r						O..sz...					Conditional Move - less or equal/not greater ((ZF=1) OR (SF!=OF))	
CMOVNG	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVNB	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVAE	<b>r16/32/64,</b>	r/m16/32/64						OF 43		r						.....c					Conditional Move - not below/above or equal/not carry (CF=0)	
CMOVNC	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVNBE	<b>r16/32/64,</b>	r/m16/32/64						OF 47		r						....z.c					Conditional Move - not below or equal/above (CF=0 AND ZF=0)	
CMOVA	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVNL	<b>r16/32/64,</b>	r/m16/32/64						OF 4D		r						O..S....					Conditional Move - not less/greater or equal (SF=OF)	
CMOVGE	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVNLE	<b>r16/32/64,</b>	r/m16/32/64						OF 4F		r						O..sz...					Conditional Move - not less nor equal/greater ((ZF=0) AND (SF=OF))	
CMOVG	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVNO	<b>r16/32/64,</b>	r/m16/32/64						OF 41		r						O.....					Conditional Move - not overflow (OF=0)	
CMOVNP	<b>r16/32/64,</b>	r/m16/32/64						OF 4B		r						.....P.					Conditional Move - not parity/parity odd	
CMOVPO	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVNS	<b>r16/32/64,</b>	r/m16/32/64						OF 49		r						...S....					Conditional Move - not sign (SF=0)	
CMOVNZ	<b>r16/32/64,</b>	r/m16/32/64						OF 45		r						....z...					Conditional Move - not zero/not equal (ZF=1)	
CMOVNE	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVO	<b>r16/32/64,</b>	r/m16/32/64						OF 40		r						O.....					Conditional Move - overflow (OF=1)	
CMOVPP	<b>r16/32/64,</b>	r/m16/32/64						OF 4A		r						.....P.					Conditional Move - parity/parity even (PF=1)	
CMOVPE	<b>r16/32/64,</b>	r/m16/32/64																				
CMOVSS	<b>r16/32/64,</b>	r/m16/32/64						OF 48		r						...S....					Conditional Move - sign (SF=1)	
CMOVZ	<b>r16/32/64,</b>	r/m16/32/64						OF 44		r						....z...					Conditional Move - zero/equal (ZF=0)	
CMOVE	<b>r16/32/64,</b>	r/m16/32/64																				
CMP	r/m8,	r8						38		r						O..szapc	O..szapc				Compare Two Operands	
CMP	r/m16/32/64,	r16/32/64						39		r						O..szapc	O..szapc				Compare Two Operands	
CMP	r8,	r/m8						3A		r						O..szapc	O..szapc				Compare Two Operands	
CMP	r16/32/64,	r/m16/32/64						3B		r						O..szapc	O..szapc				Compare Two Operands	
CMP	AL,	imm8						3C								O..szapc	O..szapc				Compare Two Operands	
CMP	rAX,	imm16/32						3D								O..szapc	O..szapc				Compare Two Operands	
CMP	r/m8,	imm8						80		7						O..szapc	O..szapc				Compare Two Operands	
CMP	r/m16/32/64,	imm16/32						81		7						O..szapc	O..szapc				Compare Two Operands	