X86 Instruction Reference 64-bit Edition

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

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

For comments, suggestions, questions or bug reports, please contact mazegen@gmail.com

For booking a computer-related ad in this reference, please contact 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 oficial 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* incidates that the opcode performs additional pu**s**h of a value to the register stack, *p* incidates that the opcode performs additional **p**op of the register stack, *P* pops twice
- *iext*: The instruction extension group, which was the opcode released on
- *tested f*, *modif f*, *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.

General and System Instructions

mnemonic	op1	op2	op3	op4	iext	pf 0F	po	so	o	proc	st	m	rl	x	tested f	modif f	def f	undef f	f values	description, notes
ADC	**r/m8,**	r8					10		r					L	c	o..szapc	o..szapc			Add with Carry
ADC	**r/m16/32/64,**	r16/32/64					11		r					L	c	o..szapc	o..szapc			Add with Carry
ADC	**r8,**	r/m8					12		r						c	o..szapc	o..szapc			Add with Carry
ADC	**r16/32/64,**	r/m16/32/64					13		r						c	o..szapc	o..szapc			Add with Carry
ADC	**AL,**	imm8					14								c	o..szapc	o..szapc			Add with Carry
ADC	**rAX,**	imm16/32					15								c	o..szapc	o..szapc			Add with Carry
ADC	**r/m8,**	imm8					80		2					L	c	o..szapc	o..szapc			Add with Carry
ADC	**r/m16/32/64,**	imm16/32					81		2					L	c	o..szapc	o..szapc			Add with Carry
ADC	**r/m16/32/64,**	imm8					83		2					L	c	o..szapc	o..szapc			Add with Carry
ADD	**r/m8,**	r8					00		r					L		o..szapc	o..szapc			Add
ADD	**r/m16/32/64,**	r16/32/64					01		r					L		o..szapc	o..szapc			Add
ADD	**r8,**	r/m8					02		r							o..szapc	o..szapc			Add
ADD	**r16/32/64,**	r/m16/32/64					03		r							o..szapc	o..szapc			Add
ADD	**AL,**	imm8					04									o..szapc	o..szapc			Add
ADD	**rAX,**	imm16/32					05									o..szapc	o..szapc			Add
ADD	**r/m8,**	imm8					80		0					L		o..szapc	o..szapc			Add
ADD	**r/m16/32/64,**	imm16/32					81		0					L		o..szapc	o..szapc			Add
ADD	**r/m16/32/64,**	imm8					83		0					L		o..szapc	o..szapc			Add
AND	**r/m8,**	r8					20		r					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**r/m16/32/64,**	r16/32/64					21		r					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**r8,**	r/m8					22		r							o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**r16/32/64,**	r/m16/32/64					23		r							o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**AL,**	imm8					24									o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**rAX,**	imm16/32					25									o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**r/m8,**	imm8					80		4					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**r/m16/32/64,**	imm16/32					81		4					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**r/m16/32/64,**	imm8					83		4					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
BSF	**r16/32/64,**	r/m16/32/64				0F	BC				D^{14}					o..szapc	....z...	o..s.apc		Bit Scan Forward
BSR	**r16/32/64,**	r/m16/32/64				0F	BD				D^{14}					o..szapc	....z...	o..s.apc		Bit Scan Reverse
BSWAP	**r16/32/64**					0F	C8+r													Byte Swap
BT	r/m16/32/64,	r16/32/64				0F	A3									o..szapc	c	o..szap.		Bit Test
BT	r/m16/32/64,	imm8				0F	BA		4							o..szapc	c	o..szap.		Bit Test
BTC	**r/m16/32/64,**	imm8				0F	BA		7					L		o..szapc	c	o..szap.		Bit Test and Complement
BTC	**r/m16/32/64,**	r16/32/64				0F	BB							L		o..szapc	c	o..szap.		Bit Test and Complement
BTR	**r/m16/32/64,**	r16/32/64				0F	B3							L		o..szapc	c	o..szap.		Bit Test and Reset
BTR	**r/m16/32/64,**	imm8				0F	BA		6					L		o..szapc	c	o..szap.		Bit Test and Reset
BTS	**r/m16/32/64,**	r16/32/64				0F	AB							L		o..szapc	c	o..szap.		Bit Test and Set
BTS	**r/m16/32/64,**	imm8				0F	BA		5					L		o..szapc	c	o..szap.		Bit Test and Set
CALL	rel32						E8				D^{16}	E								Call Procedure
CALL	r/m64						FF		2		D^{16}	E								Call Procedure
CALLF	r/m16:16/32/64						FF		3		D^{6}									Call Procedure

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