X86 Instruction Reference
32-bit Edition

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

ISBN 978-80-254-2347-9

X86 Instruction Reference, 32-bit Edition
general, system, x87 FPU, MMX, SSE(1), SSE2, SSE3, SSSE3 instructions

Second Edition, September 2008

Errata:
http://ref.x86asm.net/errata/32/instruction

Karel Lejska
Bayerova 8
Brno 60200
Czech Republic

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

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

ISBN 978-80-254-2347-9

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 *SS:[eSP]* 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 8086 processor.
- *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 and protected mode. *P* applies for protected 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 EFlags 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 EFlags 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
AAA	***AL,***	***AH***					37								.....a..	o..szapc	.....a.c	o..sz.p.		ASCII Adjust After Addition
AAD	***AL,***	***AH***					D5	0A								o..szapc	...sz.p.	o....a.c		ASCII Adjust AX Before Division
AAM	***AL,***	***AH***					D4	0A								o..szapc	...sz.p.	o....a.c		ASCII Adjust AX After Multiply
AAS	***AL,***	***AH***					3F								.....a..	o..szapc	.....a.c	o..sz.p.		ASCII Adjust AL After Subtraction
ADC	**r/m8,**	r8					10		r					L	c	o..szapc	o..szapc			Add with Carry
ADC	**r/m16/32,**	r16/32					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,**	r/m16/32					13		r						c	o..szapc	o..szapc			Add with Carry
ADC	**AL,**	imm8					14								c	o..szapc	o..szapc			Add with Carry
ADC	**eAX,**	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,**	imm16/32					81		2					L	c	o..szapc	o..szapc			Add with Carry
ADC	**r/m8,**	imm8					82		2					L	c	o..szapc	o..szapc			Add with Carry
ADC	**r/m16/32,**	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,**	r16/32					01		r					L		o..szapc	o..szapc			Add
ADD	**r8,**	r/m8					02		r							o..szapc	o..szapc			Add
ADD	**r16/32,**	r/m16/32					03		r							o..szapc	o..szapc			Add
ADD	**AL,**	imm8					04									o..szapc	o..szapc			Add
ADD	**eAX,**	imm16/32					05									o..szapc	o..szapc			Add
ADD	**r/m8,**	imm8					80		0					L		o..szapc	o..szapc			Add
ADD	**r/m16/32,**	imm16/32					81		0					L		o..szapc	o..szapc			Add
ADD	**r/m8,**	imm8					82		0					L		o..szapc	o..szapc			Add
ADD	**r/m16/32,**	imm8					83		0					L		o..szapc	o..szapc			Add
ADX	**AL,**	***AH,***	imm8				D5									o..szapc	...sz.p.	o....a.c		Adjust AX Before Division
AMX	**AL,**	***AH,***	imm8				D4									o..szapc	...sz.p.	o....a.c		Adjust AX After Multiply
AND	**r/m8,**	r8					20		r					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**r/m16/32,**	r16/32					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,**	r/m16/32					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	**eAX,**	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,**	imm16/32					81		4					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**r/m8,**	imm8					82		4					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	**r/m16/32,**	imm8					83		4	03+				L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
ARPL	r/m16,	r16					63		r	02+						....z...	....z...			Adjust RPL Field of Segment Selector
BOUND	r16/32,	m16/32&16/32, *eFlags*					62		r	01+			f			..i.....	..i.....		..i.....	Check Array Index Against Bounds
BSF	**r16/32,**	r/m16/32				0F	BC			03+	D					o..szapc	....z...	o..s.apc		Bit Scan Forward
BSR	**r16/32,**	r/m16/32				0F	BD			03+	D					o..szapc	....z...	o..s.apc		Bit Scan Reverse
BSWAP	**r16/32**					0F	C8+r			04+										Byte Swap
BT	r/m16/32,	r16/32				0F	A3			03+						o..szapc	c	o..szap.		Bit Test

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
BT	r/m16/32,	imm8					0F	BA		4	03+						o..szapc	c	o..szap.		Bit Test
BTC	**r/m16/32,**	imm8					0F	BA		7	03+				L		o..szapc	c	o..szap.		Bit Test and Complement
BTC	**r/m16/32,**	r16/32					0F	BB			03+				L		o..szapc	c	o..szap.		Bit Test and Complement
BTR	**r/m16/32,**	r16/32					0F	B3			03+				L		o..szapc	c	o..szap.		Bit Test and Reset
BTR	**r/m16/32,**	imm8					0F	BA		6	03+				L		o..szapc	c	o..szap.		Bit Test and Reset
BTS	**r/m16/32,**	r16/32					0F	AB			03+				L		o..szapc	c	o..szap.		Bit Test and Set
BTS	**r/m16/32,**	imm8					0F	BA		5	03+				L		o..szapc	c	o..szap.		Bit Test and Set
CALL	rel16/32							E8													Call Procedure
CALL	r/m16/32							FF		2											Call Procedure
CALLF	ptr16:16/32							9A													Call Procedure
CALLF	r/m16:16/32							FF		3		D									Call Procedure
CBW	***AH,***	*AL*						98													Convert Byte to Word
CDQ	***EDX,***	*EAX*						99			03+										Convert Doubleword to Quadword
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 CMOVNAE CMOVC	**r16/32,** **r16/32,** **r16/32,**	r/m16/32 r/m16/32 r/m16/32					0F	42		r	PP+					c					Conditional Move - below/not above or equal/carry (CF=1)
CMOVBE CMOVNA	**r16/32,** **r16/32,**	r/m16/32 r/m16/32					0F	46		r	PP+					....z..c					Conditional Move - below or equal/not above (CF=1 AND ZF=1)
CMOVL CMOVNGE	**r16/32,** **r16/32,**	r/m16/32 r/m16/32					0F	4C		r	PP+					o..s....					Conditional Move - less/not greater (SF!=OF)
CMOVLE CMOVNG	**r16/32,** **r16/32,**	r/m16/32 r/m16/32					0F	4E		r	PP+					o..sz...					Conditional Move - less or equal/not greater ((ZF=1) OR (SF!=OF))
CMOVNB CMOVAE CMOVNC	**r16/32,** **r16/32,** **r16/32,**	r/m16/32 r/m16/32 r/m16/32					0F	43		r	PP+					c					Conditional Move - not below/above or equal/not carry (CF=0)
CMOVNBE CMOVA	**r16/32,** **r16/32,**	r/m16/32 r/m16/32					0F	47		r	PP+					....z..c					Conditional Move - not below or equal/above (CF=0 AND ZF=0)
CMOVNL CMOVGE	**r16/32,** **r16/32,**	r/m16/32 r/m16/32					0F	4D		r	PP+					o..s....					Conditional Move - not less/greater or equal (SF=OF)
CMOVNLE CMOVG	**r16/32,** **r16/32,**	r/m16/32 r/m16/32					0F	4F		r	PP+					o..sz...					Conditional Move - not less nor equal/greater ((ZF=0) AND (SF=OF))
CMOVNO	**r16/32,**	r/m16/32					0F	41		r	PP+					o.....					Conditional Move - not overflow (OF=0)
CMOVNP CMOVPO	**r16/32,** **r16/32,**	r/m16/32 r/m16/32					0F	4B		r	PP+					......p.					Conditional Move - not parity/parity odd
CMOVNS	**r16/32,**	r/m16/32					0F	49		r	PP+					...s....					Conditional Move - not sign (SF=0)
CMOVNZ CMOVNE	**r16/32,** **r16/32,**	r/m16/32 r/m16/32					0F	45		r	PP+					....z...					Conditional Move - not zero/not equal (ZF=1)
CMOVO	**r16/32,**	r/m16/32					0F	40		r	PP+					o.....					Conditional Move - overflow (OF=1)
CMOVP CMOVPE	**r16/32,** **r16/32,**	r/m16/32 r/m16/32					0F	4A		r	PP+					......p.					Conditional Move - parity/parity even (PF=1)
CMOVS	**r16/32,**	r/m16/32					0F	48		r	PP+					...s....					Conditional Move - sign (SF=1)