x86_32 Instruction Set (ver 2.0)

- **Data Transfer Instructions (General):**

Mnemonic	Format	Operation	Flags Affected
MOV	MOV r/m, r/m/i	r/m ← r/m/i	None
LEA	LEA r16_32, m	r16_32 ← EA(m)	None
XLAT	XLAT	AL ← [(E)BX+ZeroExtend(AL)]	None
CMOVcc	CMOVcc r16_32, r/m16_32	if (cc)then r16_32 ←r/m16_32	None

- **Data Transfer Instructions (Exchange):**

Mnemonic	Format	Operation	Flags Affected
XCHG	XCHG r/m, r/m	r/m ↔ r/m	None
BSWAP	BSWAP r32	TEMP ← r32; r32[7:0] ← TEMP[31:24]; r32[15:8] ← TEMP[23:16]; r32[23:16] ← TEMP[15:8]; r32[31:24] ← TEMP[7:0];	None
XADD	XADD r/m, r	TEMP ← r + r/m; r ←r/m; r/m←TEMP	All status flags
CMPXCHG	CMPXCHG r/m, r	if AL/AX/EAX = r/m then r/m ← r else AL/AX/EAX ←r/m	All status flags
CMPXCHGS8	CMPXCHGS8 m64	if EDX:EAX = m64 then m64 ←ECX:EBX else EDX:EAX ← m64	ZF only *Not affected: CF, OF, SF, AF, PF

- **Data Transfer Instructions (Data type conversion):**

Mnemonic	Format	Operation	Flags Affected
CBW	CBW	AH ← SignExtend(AL)	None
CWDE	CWDE	EAX ← SignExtend(AX)	None
CWD	CWD	DX ← SignExtend(AX)	None
CDQ	CDQ	EDX ← SignExtend(EAX)	None
MOVSX	MOVSX r16_32, r/m8_16	R16_32 ← SignExtend(r/m8_16)	None
MOVZX	MOVZX r16_32, r/m8_16	R16_32 ← ZeroExtend(r/m8_16)	None

- **Data Transfer Instructions (Stack):**

Mnemonic	Format	Operation	Flags Affected
PUSH	PUSH r/m/i16 PUSH r/m/i32	(E)SP ← (ESP) - 2; [SS:(ESP)]← r/m/i16 (E)SP ← (ESP) - 4; [SS:(ESP)] ← r/m/i32	None
POP	POP r/m16 POP r/m32	r/m16 ← [SS:(ESP)]; (E)SP ← (ESP) + 2 (r/m32 ← [SS:(ESP)]; E)SP ← (ESP) + 4	None
PUSHA	PUSHA	TEMP ← SP; PUSH(AX); PUSH(CX); PUSH(DX); PUSH (BX); PUSH(TEMP); PUSH(BP); PUSH(SI); PUSH(DI)	None
PUSHAD	PUSHAD	TEMP ← ESP; PUSH(EAX); PUSH(ECX); PUSH(EDX); PUSH (EBX); PUSH(TEMP); PUSH(EBP); PUSH(ESI); PUSH(EDI)	None
POPA	POPA	DI ← pop(); SI ←pop(); BP ←pop(); (E)SP+2; BX ←pop(); DX ←pop(); CX ←pop(); AX ←pop()	none
POPAD	POPAD	EDI ← pop(); ESI ←pop(); EBP ←pop(); (E)SP+4; EBX ←pop(); EDX ←pop(); ECX ←pop(); EAX ←pop()	none

- **Data Transfer Instructions (Segment register):**

Mnemonic	Format	Operation	Flags Affected
LDS	LDS r16, m32 LDS r32, m48	r16 ← m32; DS ← m32+2 r32 ← m48; DS ← m48+4	None
LES	LES r16, m32 LES r32, m48	r16 ← m32; ES ← m32+2 r32 ← m48; ES ← m48+4	None
LFS	LFS r16, m32 LFS r32, m48	r16 ← m32; FS ← m32+2 r32 ← m48; FS ← m48+4	None
LGS	LGS r16, m32 LGS r32, m48	r16 ← m32; GS ← m32+2 r32 ← m48; GS ← m48+4	None
LSS	LSS r16, m32 LSS r32, m48	r16 ← m32; SS ← m32+2 r32 ← m48; SS ← m48+4	None

- **Binary Arithmetic Instructions**

Mnemonic	Format	Operation	Flags Affected
ADD	ADD r/m, r/m/i	r/m ← r/m + r/m/i	all status flags
ADC	ADC r/m, r/m/i	r/m ← r/m + r/m/i + CF	all status flags
ADCX	ADCX r32, r/m32	R32 ← r32 + r/m32 + CF	CF; *not affected: OF, SF, ZF, AF, PF;
ADOX	ADOX r32, r/m32	R32 ← r32 + r/m32 + OF	OF; *not affected: CF, SF, ZF, AF, PF;
INC	INC r/m	r/m ← r/m + 1	OF, SF, ZF, AF, PF; *not affected: CF
SUB	SUB r/m, r/m/i	r/m ← r/m - r/m/i	all status flag
SBB	SBB r/m, r/m/i	r/m ← r/m – (r/m/i + CF)	all status flags
CMP	CMP r/m, r/m/i	r/m - r/m/i (result discarded)	all status flag
DEC	DEC r/m	r/m ← r/m – 1	OF, SF, ZF, AF, PF; *not affected: CF
NEG	NEG r/m	r/m ← 0 - (r/m)	all status flags
MUL	MUL r/m	AX ← AL*r/m8 DX:AX ← AX*r/m16 EDX:EAX ← EAX*r/m32	CF, OF; *undefined: SF, ZF, AF, PF
IMUL (1-operand)	IMUL r/m	AX ← AL*r/m8 DX:AX ← AX*r/m16 EDX:EAX ← EAX*r/m32	CF, OF; *undefined: SF, ZF, AF, PF
IMUL (2-operand)	IMUL **r16**, **r16/m16/i8_16** IMUL **r32**, **r32/m32/i**	r16 ← truncate(r16*****r16/m16/i8_16) r32 ← truncate(r32*****r32/m32/i_8_16_32)	CF, OF; *undefined: SF, ZF, AF, PF
IMUL (3-operand)	IMUL **r16**, **r16/m16**, **i8_16** IMUL **r32**, **r32/m32**, **i**	r16 ← truncate(r16/m16 * i8_16) r32 ← truncate(r32/m32 * i_8_16_32)	CF, OF; *undefined: SF, ZF, AF, PF
DIV	DIV r/m	AL ← AX div r/m8 AH ← AX mod r/m8 AX ← DX:AX div r/m16 DX ← DX:AX mod r/m16 EAX ← EDX:EAX div r/m32 EDX ← EDX:EAX mod r/m32	All status flags undefined
IDIV	IDIV r/m	AL ← AX div r/m8 AH ← AX mod r/m8 AX ← DX:AX div r/m16 DX ← DX:AX mod r/m16 EAX ← EDX:EAX div r/m32 EDX ← EDX:EAX mod r/m32	All status flags undefined

- **Decimal Arithmetic Instructions**

Mnemonic	Format	Operation	Flags Affected
DAA	DAA	The instruction will adjust the result in register AL as follows: • If the lower nibble is greater than 9 or AF = 1 then AL ← AL + 06h, AF ← 1 • If the upper nibble is greater than 9 or CF = 1 then AL ← AL + 60h, CF ← 1	CF, AF, SF, ZF, PF * undefined: OF
DAS	DAS	The instruction will adjust the result in register AL as follows: • if the least significant nibble is greater than 9 or AF = 1 then AL ← AL - 06h, AF ← 1 • If the most significant nibble is greater than 9 or CF = 1 then AL ← AL - 60h, CF ← 1	CF, AF, SF, ZF, PF * undefined: OF
AAA	AAA	The instruction will adjust the result in register AL as follows: if the least significant nibble is greater than 9 or AF=1, then 6 is added to AL, and 1 is added to AH. Both AF and CF are set to 1. If no adjustment is made, both AF and CF are set to 0. Regardless of the value of the least significant nibble of register AL, the most significant nibble is always zeroed out.	AF, CF *Undefined: OF, SF, ZF, PF
AAS	AAS	The instruction will adjust the result in register AL as follows: if the least significant nibble is greater than 9 or AF = 1, then 6 is subtracted from AL, and 1 is subtracted from AH. If no adjustment is made, both AF and CF are set to 0. Regardless of the value of the least significant nibble in register AL, the most significant nibble is always zeroed out.	AF, CF *Undefined: OF, SF, ZF, PF
AAM (no operand)	AAM	AH ← AL div 0Ah AL ← AL mod 0Ah	SF, ZF, PF *undefined: CF, AF, OF
AAM (1-operand)	AAM i8	AH ← AL div i8 AL ← AL mod i8	SF, ZF, PF *undefined: CF, AF, OF
AAD (no operand)	AAD	AL ← AH*0Ah + AL AH ← 0	SF, ZF, PF *undefined: CF, AF, OF
AAD (1-operand)	AAD i8	AL ← AH*i8 + AL AH ← 0	SF, ZF, PF *undefined: CF, AF, OF

- **Control Transfer Instructions**

Mnemonic	Format	Operation	Flags Affected
JMP	JMP short rel8 JMP near rel16 JMP r/m16_32	Jump to the address specified by the operand. The prefix *short* and *near* are optional.	None
Jcc	Jcc short rel8 Jcc near rel16	If the specified condition *cc* is true then jump to the address specified in the operand; otherwise, the next instruction is executed. The prefix *short* and *near* are optional.	None
LOOP	LOOP rel8	(E)CX ← (E)CX -1 if (E)CX < > 0 then jump to the address specified by the operand; otherwise execute the instruction following LOOP.	None
LOOPE/ LOOPZ	LOOPE rel8 LOOPZ rel8	(E)CX ← (E)CX - 1 if (E)CX < > 0) and (ZF = 1) then jump to the address specified by the operand; otherwise execute the instruction following LOOPE/LOOPZ.	None
LOOPNE/ LOOPNZ	LOOPE rel8 LOOPZ rel8	(E)CX ← (E)CX - 1 if (E)CX < > 0 and (ZF = 0) then jump to the address specified by the operand; otherwise execute the instruction following LOOPNE/LOOPNZ.	None
JCXZ	JCXZ rel8	if CX = 0 then jump to the address specified by the operand.	None
JECXZ	JECXZ rel8	if ECX = 0 then jump to the address specified by the operand.	None
CALL	CALL rel16 CALL r/m16_32	The return address is saved onto the stack then jump to the address of the subroutine specified by the operand.	None
RET	RET RET i16	Transfers program control to a return address located on the top of the stack. The optional i16 is added to register (E)SP after restoring the return address.	None

- **Flag Transfer Instructions**

Mnemonic	Format	Operation	Flags Affected
LAHF	LAHF	AH ← SF ZF 0 AF 0 PF 1 CF	None
SAHF	SAHF	SF ZF 0 AF 0 PF 1 CF ← AH	SF, ZF, AF, PF, CF
CLC	CLC	CF ← 0	CF
STC	STC	CF ← 1	CF
CMC	CMC	CF ← ~CF	CF
CLI	CLI	IF ← 0	IF
STI	STI	IF ← 1	IF
CLD	CLD	DF ← 0	DF
STD	STD	DF ← 1	DF
PUSHF	PUSHF	PUSH(lower 16 bits of EFLAGS register)	None
PUSHFD	PUSHFD	PUSH(EFLAGS register)	None
POPF	POPF	lower 16 bits of EFLAGS register ← pop()	All status flags
POPFD	POPFD	EFLAGS register ← pop()	All status flags

- **Logical Instructions**

Mnemonic	Format	Operation	Flags Affected
AND	AND r/m, r/m/i	r/m ← r/m • r/m/i [r/m ← r/m & r/m/i]	CF = 0; OF = 0 *SF, ZF, PF *Undefined: AF
OR	OR r/m, r/m/i	r/m ← r/m + r/m/i [r/m ← r/m | r/m/i]	CF = 0; OF = 0 *SF, ZF, PF *Undefined: AF
XOR	XOR r/m, r/m/i	r/m ← r/m ⊕ r/m/i	CF = 0; OF = 0 *SF, ZF, PF *Undefined: AF
NOT	NOT r/m	r/m ← !(r/m) [r/m ← ~(r/m)]	none

- **Bit and Byte Instructions**

Mnemonic	Format	Operation	Flags Affected
TEST	TEST r/m, r/i	r/m • r/ i [r/m & r/i] *result discarded	CF = 0; OF = 0 *SF, ZF, PF *Undefined: AF
BT	BT r/m16_32, r16_32 BT r/m16_32, i8	CF ← Bit(BitBase, BitOffset) *BitBase: r/m16_32 *BitOffset: {r16_32 | i8 }	CF Unaffected: ZF Undefined: OF, SF, AF, PF
BTS	BTS r/m16_32, r16_32 BTS r/m16_32, i8	CF ← Bit(BitBase, BitOffset) Bit(BitBase, BitOffset) ← 1 *BitBase: r/m16_32 *BitOffset: {r16_32 | i8 }	CF Unaffected: ZF Undefined: OF, SF, AF, PF
BTR	BTC r/m16_32, r16_32 BTC r/m16_32, i8	CF ← Bit(BitBase, BitOffset) Bit(BitBase, BitOffset) ← 0 *BitBase: r/m16_32 *BitOffset: {r16_32 | i8 }	CF Unaffected: ZF Undefined: OF, SF, AF, PF
BTC	BTC r/m16_32, r16_32 BTC r/m16_32, i8	CF ← Bit(BitBase, BitOffset) Bit(BitBase, BitOffset) ← not(Bit(BitBase, BitOffset)) *BitBase: r/m16_32 *BitOffset: {r16_32 | i8 }	CF Unaffected: ZF Undefined: OF, SF, AF, PF
BSF	BSF r16_32, r/m16_32	r16_32 ← bit position containing least significant 1 bit	ZF Undefined: CF, OF, SF, AF, PF
BSR	BSR r16_32, r/m16_32	r16_32 ← bit position containing most significant 1 bit	ZF Undefined: CF, OF, SF, AF, PF
SETcc	SETcc r/m8	If *(cc)* then r/m8 ←1 else r/m8 ← 0	none

- **Shift Instructions**

Mnemonic	Format	Operation	Flags Affected
SHL/ SAL	SAL/SHL r/m, i8 SAL SHL r/m, CL	r/m ← r/m << i8 r/m ← r/m << CL	CF, OF, SF, ZF, PF *undefined: AF
SHR	SHR r/m, i8 SHR r/m, CL	r/m ← r/m >> i8 r/m ← r/m >> CL	CF, OF, SF, ZF, PF *undefined: AF
SAR	SHR r/m, i8 SHR r/m, CL	Shift r/m to the right by the number of bit positions equal to {i8 | CL} and fill the vacated bits positions on the left with the most significant bit.	CF, OF, SF, ZF, PF *undefined: AF
SHLD	SHLD r/m16_32, r16_32, i8	Shift r/m16_32 to the left by the number of bit positions equal to {i8 | CL} and fill the vacated bits positions on the left with the bits from r16_32.	CF, OF, SF, ZF, PF *undefined: AF
SHRD	SHRD r/m16_32, r16_32, i8	Shift r/m16_32 to the right by the number of bit positions equal to {i8 | CL} and fill the vacated bits positions on the right with the bits from r16_32.	CF, OF, SF, ZF, PF *undefined: AF

- **Rotate Instructions**

Mnemonic	Format	Operation	Flags Affected
ROL	ROL r/m, i8 ROL r/m, CL	Rotate r/m to the left by the number of bit positions equal to {i8|CL}. Each bit rotated out from the leftmost bit goes back into the rightmost bit position.	CF, OF *Not affected: SF, ZF, AF, PF
ROR	ROR r/m, i8 ROR r/m, CL	Rotate r/m to the right by the number of bit positions equal to {i8|CL}. Each bit rotated out from the rightmost bit goes back into the leftmost bit position.	CF, OF *Not affected: SF, ZF, AF, PF
RCL	RCL r/m, i8 RCL r/m, CL	Rotate r/m together with the CF to the left by the number of bit positions equal to {i8|CL}. Each bit rotated out from the leftmost bit goes back into the rightmost bit position.	CF, OF *Not affected: SF, ZF, AF, PF
RCR	RCR r/m, i8 RCR r/m, CL	Rotate r/m together with the CF to the right by the number of bit positions equal to {i8|CL}. Each bit rotated out from the rightmost bit goes back into the leftmost bit position.	CF, OF *Not affected: SF, ZF, AF, PF

- **String Prefix**

Mnemonic	Format	Operation	Flags Affected
REP	REP	While (E)CX <> 0 { Execute string instruction; (E)CX ← (E)CX -1}	None
REPE/REPZ	REPE/REPZ	While (E)CX <> 0 { Execute string instruction; (E)CX ← (E)CX -1 if ZF = 0 terminate loop}	None
REPNE/REPNZ	REPNE/REPNZ	While (E)CX <> 0 { Execute string instruction; (E)CX ← (E)CX -1 if ZF = 1 terminate loop}	None

- **String Instructions**

Mnemonic	Format	Operation	Flags Affected
STOSB	STOSB	[(E)DI] ← AL If DF=0 then (E)DI ← (E)DI +1 else (E)DI ← (E)DI -1	None
STOSW	STOSW	[(E)DI] ← AX If DF=0 then (E)DI ← (E)DI +2 else (E)DI ← (E)DI – 2	None
STOSD	STOSD	[(E)DI] ← EAX If DF=0 then (E)DI ← (E)DI +4 else (E)DI ← (E)DI – 4	None
LODSB	LODSB	AL ← [(E)SI] if DF=0 then (E)SI ← (E)SI +1 else (E)SI ← (E)SI -1	None
LODSW	LODSW	AX ← [(E)SI] if DF=0 then (E)SI ← (E)SI +2 else (E)SI ← (E)SI -2	None
LODSD	LODSD	EAX ← [(E)SI] if DF=0 then (E)SI ← (E)SI +4 else (E)SI ← (E)SI -4	None
MOVSB	MOVSB	[(E)DI)] ← [(E)SI] If DF=0 then (E)SI ← (E)SI + 1; (E)DI ← (E)DI + 1 else (E)SI ← (E)SI - 1; (E)DI ← (E)DI - 1	None
MOVSW	MOVSW	[(E)DI)] ← [(E)SI] If DF=0 then (E)SI ← (E)SI +2 ; (E)DI ← (E)DI + 2 else (E)SI ← (E)SI - 2; (E)DI ← (E)DI – 2	None
MOVSD	MOVSD	[(E)DI)] ← [(E)SI] If DF=0 then (E)SI ← (E)SI +4 ; (E)DI ← (E)DI + 4 else (E)SI ← (E)SI - 4; (E)DI ← (E)DI – 4	None
SCASB	SCASB	CMP AL, [(E)DI] if DF=0 then (E)DI ← (E)DI +1 else (E)DI ← (E)DI -1	All status flags
SCASW	SCASW	CMP AX, [(E)DI] if DF=0 then (E)DI ← (E)DI +2 else (E)DI ← (E)DI -2	All status flags
SCASD	SCASD	CMP EAX, [(E)DI] if DF=0 then (E)DI ← (E)DI +4 else (E)DI ← (E)DI -4	All status flags
CMPSB	CMPSB	CMP [(E)DI)], [(E)SI] If DF=0 then (E)SI ← (E)SI + 1; (E)DI ← (E)DI + 1 else (E)SI ← (E)SI - 1; (E)DI ← (E)DI - 1	All status flags
CMPSW	CMPSW	CMP [(E)DI)], [(E)SI] If DF=0 then (E)SI ← (E)SI + 2; (E)DI ← (E)DI + 2 else (E)SI ← (E)SI - 2; (E)DI ← (E)DI - 2	All status flags
CMPSD	CMPSD	CMP [(E)DI)], [(E)SI] If DF=0 then (E)SI ← (E)SI + 4; (E)DI ← (E)DI + 4 else (E)SI ← (E)SI - 4; (E)DI ← (E)DI - 4	All status flags

- **Condition codes (cc)**

Condition codes (cc)	Description	Condition Tested
Comparing Unsigned Numbers		
A/NBE	above/not below nor equal	(CF = 0) and (ZF = 0)
AE/NB	above or equal/not below	CF = 0
B/NAE	below/not above nor equal	CF = 1
BE/NA	below or equal/not above	(CF = 1) or (ZF = 1)
Comparing Signed Numbers		
G/NLE	greater than/not less than nor equal	(SF = OF) and (ZF = 0)
GE/NL	greater than or equal/not less than	SF = OF
L/NGE	less than/not greater than nor equal	SF < > OF
LE/NG	less than or equal/not greater than	(SF < > OF) or (ZF = 1)
Test Flags		
E/JZ	equal to/zero	ZF = 1
NE/NZ	not equal to/not zero	ZF = 0
C	carry	CF = 1
NC	not carry	CF = 0
S	sign	SF = 1
NS	not sign	SF = 0
P/PE	parity/parity even	PF = 1
NP/PO	no parity/parity odd	PF = 0
O	overflow	OF = 1
NO	no overflow	OF = 0

0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Lower 16-bit EFLAGS Register

- **Legend:**

r	8-bit, 16-bit or 32-bit register
r8	8-bit register
r16	16-bit register
r32	32-bit register
m	8-bit, 16-bit or 32-bit data in the memory
m8	8-bit data in the memory
m16	16-bit data in the memory
m32	32-bit data in the memory
i	8-bit, 16-bit or 32-bit immediate data
i8	8-bit immediate data
i16	16-bit immediate data
i32	32-bit immediate data
r/m	8-bit, 16-bit or 32-bit register or memory
r/m/i	8-bit, 16-bit or 32-bit register, memory or immediate
rx_y	x-bit or y-bit register
mx_y	x-bit or y-bit memory
ix_y	x-bit or y-bit immediate
rel8	short label (8-bit, transfer control within -128 to +127 from the current (E)IP)
rel16	Near label (16-bit, transfer control within the current code segment)