The x86 Instruction Set

The 8086 was the first of the x86 series of processor to be produced. Although the 8088 was the first to obtain market sales as the XT computer, both have the same instruction set. This is a list and description of available instructions on the 8086 and higher processor. Instruction that end with a question mark have many possible suffixes. Check their respective description for more details. Clock timings are not shown here since they are different on every processor and they don't take into consideration pre-fetch time, interrupt overrides and that kind of stuff.

AAA	AAD	AAM	AAS	ADC	ADD	AND	CALL
CBW	CLC	CLD	CLI	CMC	CMP	CMPS?	CWD
DAA	DAS	DEC	DIV	ESC	HLT	IDIV	IMUL
IN	INC	INT	INTO	IRET	J?	JMP	LAHF
LDS	LEA	LES	LOCK	LODS?	LOOP	LOOP?	MOV
MOVS?	MUL	NEG	NOP	NOT	OR	OUT	POP
POPF	PUSH	PUSHF	RLC	RCR	REP	REP?	RET
ROL	ROR	SAHF	SAL	SAR	SBB	SCAS?	SHL
SHR	STC	STD	STI	STOS?	SUB	TEST	WAIT
XCHG	XLAT	XOR					

AAA ASCII adjust for addition.

Adjusts the value in AL or AH after an addition of unpacked binary coded decimals. Use only after an ADD instruction. Use the instruction OR al,30h to get an ascii result.

AAD ASCII adjust for division.

Adjusts the value in AL or AH after a division of unpacked binary coded decimals. Use only after an DIV instruction. Use the instruction OR al,30h to get an ascii result.

AAM ASCII adjust for multiplication.

Adjusts the value in AL or AH after a multiplication of unpacked binary coded decimals. Use only after an MUL instruction. Use the instruction OR al,30h to get an ascii result.

AAS ASCII adjust for subtraction.

Adjusts the value in AL or AH after a subtraction of unpacked binary coded decimals. Use only after an SUB instruction. Use the instruction OR al,30h to get an ascii result.

ADC Add with carry.

Adds the two operand and stores the result in the first operand. If there is a result with carry, it will be set in CF. ADC takes care of a previous carry result in the operation by adding one if the carry flag is set.

ADD Addition.

Adds the two operand and stores the result in the first operand. If there is a result with carry, it will be set in CF. ADD doesn't take care of a previous carry result in the operation.

AND Logical AND.

Performs a logical and operation on two operands and stores the result in the first operand. In a logical AND, the result bit will equal 1 if both bits are equal to 1.

CALL Perform Subroutine.

Calls another procedure and leaves control to it until it returns.

CBW Convert byte to word.

Converts a signed byte in AL into a signed word in AX. It is used to extend the sign.

CLC Clear Carry Flag.

Set the carry flag to zero.

CLD Clear Direction Flag.

Sets the direction flag to zero.

CLI Clear Interrupt Flag.

Sets the interrupt enable flag to zero. This also disables all maskable interrupts. Use STI to restore them.

CMC Complement Carry Flag.

Equivalent to NOT CF. If CF equals 1, it is set to zero and vice-versa.

CMP Arithmetic Compare.

Subtracts the second operand from the first but does not store the value. It will set the flags accordinly and these can be tested by a jump, rep or set instruction.

CMPS? Compare String

Like CMP but uses (E)SI and (E)DI pointers to compare. It will update both pointers according to the size of the of the values compared and depending on the direction flag, it will either increment (DF = 1) or decrement (DF = 0) the pointers. The size of the values compared depends on the instruction used : CMPSB for byte values, CMPS or CMPSW for word values and CMPSD for dword values. It may be used with a conditional repetition (i.e. REP?) the check if two strings are equivalent. You must set both (E)SI and (E)DI pointers before using the CMPS? instruction.

CWD Convert word to double-word.

Converts a word in AX to a double-word in EAX. It is used to extend the sign.

DAA Decimal Adjust for Additon.

Adjusts the value in AL or AH after an addition of packed binary coded decimals. Use only after an ADD instruction.

DAS Decimal Adjust for Subtraction.

Adjusts the value in AL or AH after an subtraction of packed binary coded decimals. Use only after an SUB instruction.

DEC Decrement.

Subtracts 1 to the operand.

DIV Unsigned divide.

Divides the accumulator register (AX, EAX, EDX:EAX) by the operand. The size of the register used is twice the size of the divisor. Depending on this size (the divisor's) , the result will go to specific areas: the quotient is stored in AL and the remainder in AH for a byte divisor, quotient in AX and remainder in DX for a word divisor, quotient in EAX and remainder in EDX for a dword divisor.

ESC Co-processor escape into CPU.

This instruction is listed but I have no documentation on it.

HLT HALT.

Halts the processor. It will resume after a NMI execution. If you are in protected mode (and you're not wrtitting an OS) or in V86 mode (i.e. EMS manager loaded) this will cause a general protection exception. In short, do not use this instruction, the computer will most certainly hang.

IDIV Signed divide.

Performs like DIV but takes the signs of the operand and accumulator register into consideration.

IMUL Unsigned multiply.

Performs an unsigned multiplication. The instruction allows up to three operands. In the case of one operand, the operand is multiplied by the corresponding accumulator register size (EAX,AX,AL) and stored there. In the case of two operands, the two are multiplied together and stored in the first operand. In the case of three operands, the two last are multiplied together and the result is stored in the first operand.

IN Input from port.

Inputs data from a port. The size of the data depends on the first operand, which must be AL, AX, or EAX. The second operand may be either a port number constant or DX containing the port number to be accessed. The data will be stored in the first operand.

INC Increment.

Adds 1 to the operand

INT Interrupt.

Calls the interrupt specified by the operand.

INTO Interrupt on Overflow.

Calls interrupt 4 if the overflow flag is set.

IRET Interupt Return.

Returns from an interrupt routine. Use in an interrupt handler or use RET instead.

J? Conditional jump

Jumps to a specified address or a label if the condition is met. J? will test the needed flags and will jump depending on the condition. Here is a list of the available conditions. Note the some conditions a equivalent to another.

Mnemonic	**Meaning**	**Flag operation**	**Equivalent**
JA	Above	CF=0 and ZF= 0	JNBE
JAE	Above or Equal	CF = 0	JNB
JB	Jump if Below	CF = 1	JC, JNE
JBE	Below or Equal	CF=1 or ZF=1	JNA
JC	Carry	CF = 1	JB, JNAE
JCXZ	CX is Zero		
JE	Equal	ZF = 0	JZ
JG	Greater	(SF = OF) and ZF = 0	JNLE
JGE	Greater or Equal	SF = OF	JNL
JL	Less	SF <> OF	JNGE
JLE	Less or Equal	(SF <> OF) and ZF = 1	JNG
JNA	Not Above	CF = 1 or ZF = 1	JBE
JNAE	Not Above or Equal	CF = 1	JB, JC
JNB	Not Below	CF = 0	JNC
JNBE	Not Below or Equal	CF = 0 and ZF = 0	JA
JNE	Not Equal	ZF = 0	JNZ
JNG	Not Greater	(SF <> OF) and ZF =1	JLE
JNGE	Not Greater or Equal	SF <> OF	JL
JNL	Not Less	SF = OF	JGE
JNLE	Not Less or Equal	(SF = OF) and ZF = 0	JG
JNO	No Overflow	OF = 0	
JNP	No Parity	PF = 1	JPO
JNS	No Sign	SF = 0	
JNZ	Not Zero	ZF = 1	JNE
JO	Overflow	OF = 1	

JP	Parity	PF = 1	JPE
JPE	Parity Even	PF = 1	JP
JPO	Parity Odd	PF = 0	JNP
JS	Sign	SF = 1	
JZ	Zero	ZF = 1	JE

JMP Unconditional jump.

Jumps to a specified address or a label

LAHF Load AH with FLAGS.

Loads the low-byte part of the Flags register into AH.

LDS Load DS register.

Loads the Data Segment and the offset in the first operand with the second operand. The address format of the second operand must be 0000:0000.

LEA Load Effective Address.

Takes the offset part of the adsress of the second operand and stores it in the first operand.

LES Load ES register.

Loads the Extra Segment and the offset in the first operand with the second operand. The address format of the second operand must be 0000:0000.

LOCK LOCK bus for next instruction.

Locks the processor for the next instruction. The instruction executed after a LOCK must be a specific one or an interrupt 6 will be generated. This instruction is useful for multi-processing environments.

LODS? Load form string.

Load a value from DS:SI, stores it in the accumulator register and updates SI. The size of the value depends on the suffix used (B: Byte, W : Word, D : DWord). It may be used with a conditional REP but it may be better to used it with a conditional LOOP to allow treatment of the data.

LOOP Unconditional loop.

Jumps back to a label and decrements CX until CX equals zero.

LOOP? Conditional loop

Jumps back to a label and decrements CX until CX equals zero or the condition is not met. Here is a list of the conditional LOOPs available :

LOOPE	Loop while equal
LOOPNE	Loop while not equal
LOOPNZ	Loop while not zero
LOOPZ	Loop while zero

MOV Move Contents.

Stores the second operand's value in the first operand

MOVS? Move string.

Moves the content at DS:SI depending on the size of the move to the memory pointed by ES:DI and updates SI and DI. It may be used with a REP instruction to set a block of memory to a certain value. It may be used in a string copy routine.

MUL Signed multiply.

Performs a signed multiply. The first operand must be an accumulator register (AL,AX,EAX). The two operands are multiplied together and the result is store in the first operand.

NEG Negate.

Sets the operand with it's negative value.

NOP No Operation.

Performs no operation. Used for code alignment and debugging.

NOT Logical NOT.

Performs a logical NOT on the operand. In a logical NOT, the result bit is the opposite of its value.

OR Logical inclusive OR.

Performs a logical OR between to operands and stores the value in the first operand. In a logical OR, the result bit will be equal to zero if both bits are equal to zero. Otherwise, the result bit is equal to 1.

OUT Output to port.

Outputs data to a port. The first operand must be DX, containing the address of the port to access. The second operand is either AL, AX or EAX, depending on the size of the data to send.

POP POP stack to register.

Restores a value from the stack and into the operand.

POPF POP stack to FLAGS.

Restores a value from the stack and into the FLAGS register.

PUSH PUSH register/memory to stack.

Stores a value in the stack.

PUSHF PUSH flags to stack.

Stores the FLAGS register in the stack.

RCL Rotate operand left through CF.

Rotates the bits in the first operand to the left by the value in the second operand. the bits that overflow is stores in CF and, if necessary, is pushed back in the operand as the right-most (lowest) bit.

RCR Rotate operand right through CF.

Rotates the bits in the first operand to the right by the value in the second operand. the bits that overflow is stores in CF and, if necessary, is pushed back in the operand as the left-most (highest) bit.

REP Unconditional repeat

Repeats the following instruction and decrements CX until CX equals zero. Use with MOVS? and STOS? instructions.

REP? Conditional repeat

Repeats the following instruction and decrements CX until CX equals zero or the condition is not met. Use with CMPS? and SCAS? instructions.

RET Return to caller.

Returns to the caller. Use IRET in an interrupt handler.

ROL Rotate Operand Left.

Rotates the bits in the first operand to the left by the value in the second operand but does not store the overflowing bit in CF like RCL does.

ROR Rotate Operand Right.

Rotates the bits in the first operand to the right by the value in the second operand but does not store the overflowing bit in CF like RCR does.

SAHF Store AH into FLAGS.

Moves the content of AH into the flag register.

SAL Signed Shift Left.

Shift the bits in the first operand to the left by the value in the second operand and takes into consideration the sign of the operand. Similar to a signed multiply by 2 "second operand" times.

SAR Signed Shift Right.

Shift the bits in the first operand to the right by the value in the second operand and takes into consideration the sign of the operand. Similar to a signed division by 2 "second operand" times.

SBB Subtract with Borrow.

The carry flags is added to the first operand and the second operand is subtracted from the result. The result is stored in the first operand.

SCAS? Scan String.

Compares the values between the accumulator register (EAX, AX, AL) and the content at ES:DI. The flags are set accordingly and DI is updated.

SHL Shift Operand Left.

Shift the bits in the first operand to the left by the value in the second operand but does not takes into consideration the sign of the operand. Similar to an unsigned multiply by 2 "second operand" times.

SHR Shift Operand Right.

Shift the bits in the first operand to the right by the value in the second operand but does not takes into consideration the sign of the operand. Similar to an unsigned division by 2 "second operand" times.

STC Set Carry Flag.

Sets the carry flag to 1.

STD Set Direction Flag.

Sets the Direction Flag to 1.

STI Set Interrupt Flag.

Set the interrupt flag to 1. This also enables all maskable interrupts after a CLI instruction.

STOS? String Set.

Sets the content at ES:DI with the value in EAX,AX or AL, depending on the suffix and updates DI.

SUB Subtract.

Subtracts one operand from another and stores the remainder in the first operand.

TEST Logical compare

Performs a logical AND on two operands and sets the flags accordingly. The result is discarded.

WAIT Wait for external interrupt.

Tells the processor to wait for a response from the co-processor.

XCHG Swap contents of two operands.

Exchanges the values of two operands

XLAT Table look-up translation

Stores the content at address DS:[BX+AL] in AL.

Logical exclusive OR

Performs a logical exlusive OR on two operand and stores the result in the first operand. In a logical XOR, the result bit is equal to 0 if both bits have the same value. If they have different values, the result bit is equal to 1. It is used as a fast wait to set a register to zero by performing a logical XOR on itself.