

---

# Befehlsliste

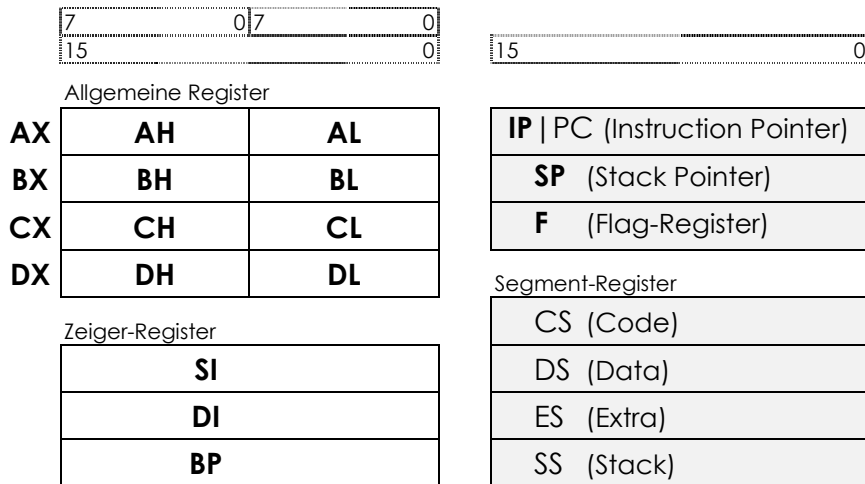
## 16-Bit-Prozessor

### INTEL 8086

#### Inhaltsverzeichnis:

Registersatz 8086.....	2
Befehlsliste .....	2
Transportbefehle .....	3
Ein- und Ausgabebefehle .....	4
Arithmetische Befehle.....	4
Logische Befehle.....	7
Rotations- und Schiebefehle.....	7
Sprungbefehle .....	8
Unterprogrammbefehle.....	9
Interrupts .....	9
Stringbefehle .....	10
Flag-Befehle .....	11
Sonstige Befehle .....	11
Das Flagregister.....	12
Bedingte Sprungbefehle .....	13
Adressierungsarten und Taktangaben .....	14
Logische Funktionen.....	14
Zahlen und Zeichen.....	15
Der 8086-Mikrorechner aus Programmierersicht .....	16

## Registersatz 8086



## Verwendete Abkürzungen

acc	Akkumulator, Register AX oder AL
reg	ein 8- oder 16-Bit-Register (reg8   reg16)
reg8	eines der 8-Bit-Register AL, AH, BL, BH, CL, CH, DL oder DH
reg16	eines der 16-Bit-Register AX, BX, CX, DX, SI, DI oder BP
seg	eines der Segmentregister CS, DS, ES oder SS
const	8- oder 16-Bit-Konstante (const8   16)
const8	ausschließlich eine 8-Bit-Konstante
const16	ausschließlich eine 16-Bit-Konstante
[mem]	Speicheradresse eines 8- oder 16-Bit-Operanden
[mem8]	Speicheradresse eines 8-Bit-Operanden
[mem16]	Speicheradresse eines 16-Bit-Operanden
[mem32]	Speicheradresse zweier 16-Bit-Operanden
port8	8-Bit-Adresse eines Datenports (Ein- oder Ausgabeport)
shortlabel	Marke im aktuellen Codesegment, 8-Bit-Sprungdistanz (-128 bis +127 Byte)
label	Marke im aktuellen Codesegment, 16-Bit-Sprungdistanz

## Befehlsliste

Diese Befehlsliste enthält die Befehle des Mikroprozessors Intel 8086 (beschränkt auf folgende Konfiguration: max. Speicher 64 KByte, keine Segmentierung).

Befehl / Befehlsgruppe allgemein		
Befehl	Takte	Erläuterung, Funktionsbeschreibung
<b>MOV dest,source</b>		Move Data
MOV reg,[mem]	8+ea	Der Inhalt von <source> wird nach <dest> kopiert.

Alle in der Befehlssyntax groß geschriebenen Bestandteile sind wie angegeben zu verwenden, die klein geschriebenen Operanden müssen (siehe „Verwendete Abkürzungen“) angepasst werden. Die Verwendung der Taktangaben ist auf Seite 14 erläutert.

Ist der Befehl / die Befehlsgruppe nicht fett gedruckt, so spielt dieser in den Lehrveranstaltungen „Rechnerarchitektur“ und „Mikroprozessortechnik“ keine Rolle.

## Transportbefehle

<b>MOV dest,source</b>		Move Data
MOV reg,reg	2	Der Inhalt von <source> bzw. die Konstante wird nach <dest> kopiert.
MOV reg,[mem]	8+ea	
MOV [mem],reg	9+ea	
MOV acc,[mem]	10	
MOV [mem],acc	10	
MOV reg,const	4	
MOV [mem],const	10+ea	
MOV seg,reg16	2	
MOV seg,[mem16]	8+ea	
MOV reg16,seg	2	
MOV [mem16],seg	9+ea	
<b>XCHG dest,source</b>		Exchange Data
XCHG reg,reg	4	Die Inhalte von <source> und <dest> werden getauscht.
XCHG reg,[mem]	17+ea	
XCHG [mem],reg	17+ea	
XCHG acc,reg	3	
XCHG reg,acc	3	
<b>PUSH source</b>		Push Operand onto the Stack (16 Bit)
PUSH reg16	11	Der Inhalt von <source> wird in den Stack geschrieben (SP-2).
PUSH [mem16]	16+ea	
PUSH seg	10	
PUSHF	10	Push Flagregister (SP-2)
<b>POP dest</b>		Pop Operand from Stack (16 Bit)
POP reg16	8	Aus dem Stack wird ein Wort gelesen und nach <dest> geschrieben (SP+2).
POP [mem16]	17+ea	
POP seg	8	
POPF	8	Pop Flagregister (SP+2)
<b>LAHF / SAHF</b>		Load Flags into AH / Store AH into Flags
LAHF	4	Das LOW-Byte des Flagregisters wird nach AH kopiert.
SAHF	4	AH wird in das LOW-Byte des Flagregisters kopiert.
<b>XLAT</b>		Table Look-Up Translation
XLAT	11	BX (zeigt auf die erste Adresse einer Tabelle) und AL (enthält einen positiven Index) werden addiert und das Ergebnis als Adresse genutzt. Der so adressierte Speicheroperand wird nach AL kopiert.

<b>LEA reg,mem</b>		Load Effective Address
LEA reg16,mem MOV reg16,mem	2+ea	Die (effektive) Adresse <mem> eines Speicheroperanden wird nach <reg16> geladen. Die Syntax hängt vom verwendeten Assembler ab. Hinweis: <mem> ohne Klammern [ ]
<b>LDS reg,mem</b>		Load Pointer using DS
LDS reg16,[mem32]	16+ea	Das LOW-Wort des Speicheroperanden <mem32> wird nach <reg16> und das HI-Wort in das Segmentregister DS übertragen.
<b>LES reg,mem</b>		Load Pointer using ES
LES reg16,[mem32]	16+ea	Wie LDS, überträgt aber das HI-Wort in das Segmentregister ES

## Ein- und Ausgabebefehle

<b>IN dest,source</b>		Input Data from Port
IN acc,port8 IN acc,DX	10 8	Vom Datenport mit der Adresse port8   [DX] wird ein Operand gelesen und nach AL   AX geschrieben.
<b>OUT dest,source</b>		Output Data to Port
OUT port8,acc OUT DX,acc	10 8	Der Inhalt von AL   AX wird zum Datenport mit der Adresse port8   [DX] übertragen (ausgegeben).

## Arithmetische Befehle

<b>ADD dest,source</b>		Add Integers
ADD reg,reg ADD reg,[mem] ADD [mem],reg ADD reg,const ADD [mem],const	3 9+ea 16+ea 4 17+ea	Die Inhalte von <dest> und <source> bzw. der Inhalt von <dest> und die Konstante werden addiert, <dest> wird mit dem Ergebnis überschrieben.
<b>ADC dest,source</b>		Add Integers with Carry
ADC reg,reg ADC reg,[mem] ADC [mem],reg ADC reg,const ADC [mem],const	3 9+ea 16+ea 4 17+ea	Wie ADD, allerdings wird zum Ergebnis noch der Wert des Carry-Flag hinzuaddiert.
<b>DAA</b>		Decimal Adjust AL after Addition
DAA	4	Nach ADD   ADC gepackter BCD-Zahlen wird AL in das korrekte BCD-Ergebnis korrigiert.
<b>AAA</b>		ASCII Adjust after Addition
AAA	8	Wie DAA, allerdings für ungepackte BCD-Zahlen. Ein Übertrag wird zum Inhalt von AH addiert.
<b>INC dest</b>		Increment
INC reg INC [mem]	3 15+ea	Der Inhalt von <dest> wird um eins erhöht.

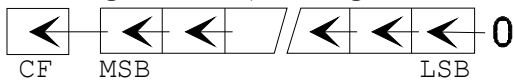
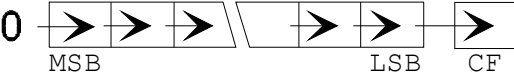
<b>SUB dest,source</b>		Subtract Integers
SUB reg,reg	3	Der Inhalt von <source> bzw. die Konstante wird vom Inhalt von <dest> subtrahiert, <dest> wird mit dem Ergebnis überschrieben.
SUB reg,[mem]	9+ea	
SUB [mem],reg	16+ea	
SUB reg,const	4	
SUB [mem],const	17+ea	
<b>SBB dest,source</b>		Subtract Integers with Borrow
SBB reg,reg	3	Wie SUB, allerdings wird vom Ergebnis noch der Wert des Carry-Flag subtrahiert.
SBB reg,[mem]	9+ea	
SBB [mem],reg	16+ea	
SBB reg,const	4	
SBB [mem],const	17+ea	
<b>DAS</b>		Decimal Adjust AL after Subtraction
DAS	4	Nach SUB SBC gepackter BCD-Zahlen wird AL in das korrekte BCD-Ergebnis korrigiert.
<b>AAS</b>		ASCII Adjust after Subtraction
AAS	8	Wie DAS, allerdings für ungepackte BCD-Zahlen. Ein negativer Übertrag wird von AH abgezogen.
<b>DEC dest</b>		Decrement
DEC reg	3	Der Inhalt von <dest> wird um eins erniedrigt.
DEC [mem]	15+ea	
<b>CMP dest,source</b>		Compare
CMP reg,reg	3	Wie SUB, aber <dest> wird <u>nicht</u> mit dem Ergebnis überschrieben, sondern es werden lediglich die Flags beeinflusst.
CMP reg,[mem]	9+ea	
CMP [mem],reg	9+ea	
CMP reg,const	4	
CMP [mem],const	10+ea	
<b>NEG dest</b>		Negate (Two's Complement)
NEG reg	3	Der vorzeichenbehaftete Inhalt von <dest> wird von 0 subtrahiert und durch das Ergebnis ersetzt (Umwandlung eines positiven in einen negativen Wert und umgekehrt).
NEG [mem]	16+ea	
<b>CBW</b>		Convert Byte to Word
CBW	2	Wenn der Inhalt des Registers AL eine positive Zahl ist, wird AH mit 0 geladen, andernfalls mit FF.
<b>CWD</b>		Convert Word to Doubleword
CWD	5	Wenn der Inhalt des Registers AX eine positive Zahl ist, wird DX mit 0 geladen, andernfalls mit FFFF.

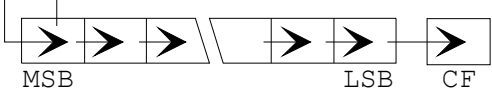
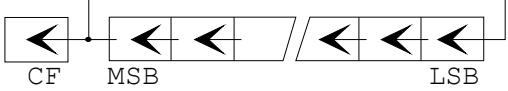
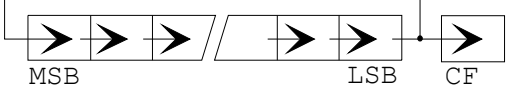
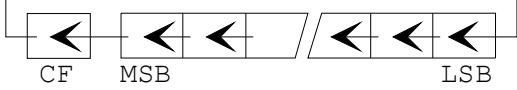
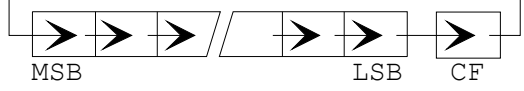
MUL source		Unsigned Multiply
MUL reg8	77	Der Registerinhalt von AL AX wird mit dem 8 16-Bit-Inhalt von <source> multipliziert. Das Produkt von doppelter Länge wird in AX DX:AX abgelegt.
MUL reg16	133	
MUL [mem8]	83+ea	
MUL [mem16]	139+ea	
IMUL source		Signed Multiply
IMUL reg8	98	Wie MUL, allerdings unter Beachtung der Vorzeichen.
IMUL reg16	154	
IMUL [mem8]	104+ea	
IMUL [mem16]	160+ea	
AAM		ASCII Adjust after Multiplication
AAM	83	Nach einer Multiplikation ungepackter BCD-Zahlen wird der Inhalt des Registers AX so korrigiert, dass sich wiederum eine korrekte BCD-Zahl ergibt.
DIV source		Unsigned Divide
DIV reg8	90	Der Registerinhalt von AX DX:AX wird durch den 8 16-Bit-Inhalt von <source> dividiert. Der Quotient wird in AL AX und der Rest in AH DX abgelegt. Bei Division durch 0 oder wenn das Ergebnis für AL AX zu groß ist, wird ein INT 0 erzeugt.
DIV reg16	162	
DIV [mem8]	96+ea	
DIV [mem16]	168+ea	
IDIV source		Signed Division
IDIV reg8	112	Wie DIV, allerdings unter Beachtung der Vorzeichen.
IDIV reg16	184	
IDIV [mem8]	118+ea	
IDIV [mem16]	190+ea	
AAD		ASCII Adjust before Division
AAD	60	Der Inhalt des Registers AX wird <u>vor</u> einer Division ungepackter BCD-Zahlen so geändert, dass der Quotient wiederum eine BCD-Zahl ergibt.

## Logische Befehle

<b>NOT dest</b>		Logical NOT (One's Complement)
NOT reg NOT [mem]	3 16+ea	Der Inhalt von <dest> wird bitweise komplementiert (invertiert).
<b>AND dest,source</b>		Logical AND
AND reg,reg AND reg,[mem] AND [mem],reg AND reg,const AND [mem],const	3 9+ea 16+ea 4 17+ea	Die Inhalte von <dest> und <source> bzw. der Inhalt von <dest> und die Konstante werden bitweise UND-verknüpft, <dest> wird mit dem Ergebnis überschrieben.
<b>TEST dest,source</b>		Logical Compare
TEST reg,reg TEST reg,[mem] TEST [mem],reg TEST reg,const TEST [mem],const	3 9+ea 16+ea 4 17+ea	Wie AND, aber <dest> wird <u>nicht</u> mit dem Ergebnis überschrieben, sondern es werden lediglich die Flags beeinflusst.
<b>OR dest,source</b>		Logical OR
OR reg,reg OR reg,[mem] OR [mem],reg OR reg,const OR [mem],const	3 9+ea 16+ea 4 17+ea	Die Inhalte von <dest> und <source> bzw. der Inhalt von <dest> und die Konstante werden bitweise ODER-verknüpft, <dest> wird mit dem Ergebnis überschrieben.
<b>XOR dest,source</b>		Logical Exklusiv OR
XOR reg,reg XOR reg,[mem] XOR [mem],reg XOR reg,const XOR [mem],const	3 9+ea 16+ea 4 17+ea	Die Inhalte von <source> und <dest> bzw. der Inhalt von <dest> und die Konstante werden bitweise XOR-verknüpft (Antivalenz, Exklusiv-Oder), <dest> wird mit dem Ergebnis überschrieben.

## Rotations- und Schiebefehle

<b>SHL   SAL source,count</b>		Shift Logical   Arithmetic Left
SHL reg,1 SHL reg,CL SHL [mem],1 SHL [mem],CL	2 8+4CL 15+ea 20+ea+4CL	Der Inhalt von <source> wird <count> mal (1 oder Inhalt des Registers CL) links geschoben. 
<b>SHR source,count</b>		Shift Logical Right
SHR reg,1 SHR reg,CL SHR [mem],1 SHR [mem],CL	2 8+4CL 15+ea 20+ea+4CL	Der Inhalt von <source> wird <count> mal (1 oder Inhalt des Registers CL) rechts geschoben. 

<b>SAR source,count</b>		Shift Arithmetic Right
SAR reg,1	2	Der Inhalt von <source> wird <count> mal (1 oder Inhalt CL) rechts geschoben, das MSB bleibt erhalten. 
SAR reg,CL	8+4CL	
SAR [mem],1	15+ea	
SAR [mem],CL	20+ea+4CL	
<b>ROL source,count</b>		Rotate Left
ROL reg,1	2	Der Inhalt von <source> wird <count> mal (1 oder Inhalt des Registers CL) links rotiert. 
ROL reg,CL	8+4CL	
ROL [mem],1	15+ea	
ROL [mem],CL	20+ea+4CL	
<b>ROR source,count</b>		Rotate Right
ROR reg,1	2	Der Inhalt von <source> wird <count> mal (1 oder Inhalt des Registers CL) rechts rotiert. 
ROR reg,CL	8+4CL	
ROR [mem],1	15+ea	
ROR [mem],CL	20+ea+4CL	
<b>RCL source,count</b>		Rotate Left through Carry
RCL reg,1	2	Der Inhalt von <source> wird <count> mal (1 oder Inhalt von CL) links durch das Carry-Flag rotiert. 
RCL reg,CL	8+4CL	
RCL [mem],1	15+ea	
RCL [mem],CL	20+ea+4CL	
<b>RCR source,count</b>		Rotate Right through Carry
RCR reg,1	2	Der Inhalt von <source> wird <count> mal (1 oder Inhalt von CL) rechts durch das Carry-Flag rotiert. 
RCR reg,CL	8+4CL	
RCR [mem],1	15+ea	
RCR [mem],CL	20+ea+4CL	

## Sprungbefehle

<b>JMP ...</b>		Unconditional Jump
JMP label	15	Unbedingter (relativer) Sprung zur Zieladresse <label>   <shortlabel>
JMP shortlabel		
JMP reg16	11	Unbedingter Sprung, die Zieladresse wird dem Register <reg16> entnommen.
JMP [mem16]	18+ea	Unbedingter Sprung, die Zieladresse wird den durch <mem16> adressierten Speicherstellen entnommen.
<b>Jcond label</b>		Conditional Jump
Jcond shortlabel	16/4	Es wird zu <shortlabel> gesprungen, wenn die Bedingung <cond> erfüllt ist (siehe Tabelle hinten).
<b>JCXZ label</b>		Jump if CX is Zero
JCXZ shortlabel	18/6	Es wird zu <shortlabel> gesprungen, wenn der Inhalt des Registers CX = 0 ist.



<b>LOOP label</b>		Loop
LOOP shortlabel	17/5	1. Der Registerinhalt von CX wird dekrementiert. 2. Es wird zu <shortlabel> gesprungen, wenn CX != 0 ist, sonst weiter beim folgenden Befehl.
LOOPZ shortlabel LOOPE shortlabel	18/6	wie LOOP, aber es wird zu <shortlabel> gesprungen, wenn CX != 0 und ZF=1 sind.
LOOPNZ shortlabel LOOPNE shortlabel	19/5	wie LOOP, aber es wird zu <shortlabel> gesprungen, wenn CX != 0 und ZF=0 sind.

## Unterprogrammbefehle

<b>CALL procedure</b>		Call Procedure
CALL label	19	Die Adresse des folgenden Befehls (IP) wird in den Stack geschrieben (SP-2), anschließend wird zu <label> gesprungen.
CALL reg16	16	Wie oben, die Zieladresse wird dem Register <reg16> entnommen.
CALL [mem16]	21+ea	Wie oben, die Zieladresse wird den durch <mem16> adressierten Speicherstellen entnommen.
<b>RET</b>		Return (from Procedure)
RET RET const	16 26	Aus dem Stack wird die (Rücksprung-) Adresse gelesen (nach IP) und dorthin gesprungen (SP+2). Ist <const> angegeben, wird zusätzlich zum Stack-pointer <const> addiert (SP+<const>).

## Interrupts

<b>INT const8</b>		Call Interrupt Procedure
INT const8	51	Die (physische, 20 Bit-) Adresse des folgenden Befehls (CS:IP) und das Flagregister werden in den Stack geschrieben (SP-6). Danach wird in der Interrupt-Vektor-Tabelle die Adresse der zugehörigen Interrupt-Service-Routine gelesen und dorthin gesprungen. Das Interrupt-Flag wird auf „0“ gesetzt.
INT 3	52	wie oben, der Befehl erzeugt einen INT 3.
INTO	53	wie oben, erzeugt einen INT 4, wenn Overflow-Flag=1.
<b>IRET</b>		Return from Interrupt
IRET	24	Aus dem Stack werden das Flagregister und die (physische, 20-Bit-Rücksprung-) Adresse gelesen und restauriert (F, CS:IP, SP+6), es wird dorthin (zurück) gesprungen.

## Stringbefehle

Stringbefehle		
<p>Für alle Stringbefehle gilt: Speicheroperanden werden indirekt über die Register DS:SI (&lt;source&gt;) und ES:DI (&lt;dest&gt;) adressiert. Der Typ des Operanden wird im Assemblerbefehl durch B (Byte) oder W (Wort) gekennzeichnet.</p> <p>Nach der Befehlsausführung werden die Inhalte der Zeigerregister SI und DI automatisch erhöht (Byte: +1, Word: +2 wenn Destination-Flag DF=0) bzw. erniedrigt (wenn DF=1) und der Inhalt des Registers CX wird um eins erniedrigt.</p> <p>Ein vorangestellter REP-Präfix wiederholt die String-Befehle solange, bis die Abbruchbedingung erfüllt ist.</p>		
MOVS		Move Data from String to String
MOVSB	18	Ein Byte   Wort wird von <source> nach <dest> kopiert.
MOVSW	18	
LODS		Load String Operand
LODSB	12	Ein Byte   Wort wird von <source> nach AL   AX kopiert.
LODSW	12	
STOS		Store String Operand
STOSB	11	Ein Byte   Wort wird von AL   AX nach <dest> kopiert.
STOSW	11	
SCAS		Scan (Compare) String Data
SCASB	15	Die Inhalte von AL   AX und <dest> werden verglichen (siehe Befehl CMP).
SCASW	15	
CMPS		Compare String Operand
CMPSB	22	Die Inhalte von <source> und AL   AX werden verglichen (siehe Befehl CMP).
CMPSW	22	
REP		Repeat
REP MOVS		Wiederholung des Stringbefehls, solange der Inhalt des Registers CX != 0.
REP STOS		
REPZ SCAS		Wiederholung des Vergleichsbefehls, solange der Inhalt des Registers CX != 0 und ZF=1 (Gleichheit).
REPE SCAS		
REPZ CMPS		
REPE CMPS		
REPNZ SCAS		Wiederholung des Vergleichsbefehls, solange der Inhalt des Registers CX != 0 und ZF=0 (Ungleichheit).
REPNE SCAS		
REPNZ CMPS		
REPNE CMPS		

## Flag-Befehle

<b>CLC</b>		Clear Carry Flag
CLC	2	Das Carry-Flag wird auf 0 gesetzt.
<b>STC</b>		Set Carry Flag
STC	2	Das Carry-Flag wird auf 1 gesetzt.
<b>CMC</b>		Complement Carry Flag
CMC	2	Das Carry-Flag wird invertiert.
<b>CLI</b>		Clear Interrupt Flag
CLI	2	Das Interrupt-Flag wird auf 0 gesetzt, maskierbare Interrupts werden gesperrt.
<b>STI</b>		Set Interrupt Flag
STI	2	Das Interrupt-Flag wird auf 1 gesetzt, maskierbare Interrupts werden zugelassen.
<b>CLD</b>		Clear Direction Flag
CLD	2	Das Direction-Flag wird auf 0 gesetzt, die String-Befehle erhöhen die Zeiger.
<b>STD</b>		Set Direction Flag
STD	2	Das Direction-Flag wird auf 1 gesetzt, die String-Befehle erniedrigen die Zeiger.

## Sonstige Befehle

<b>NOP</b>		No Operation
NOP	3	Ein Leerbefehl, drei Takte lang wird nichts getan.
<b>SR:</b>		Segment Override
CS:	2	Es wird bei Speicherzugriffen nicht mit dem Standardsegment sondern mit dem angegebenen Segment gearbeitet.
DS:	2	
ES:	2	
SS:	2	
<b>HLT</b>		Halt
HLT	2	Die Befehlsausführung wird unterbrochen und erst nach einem Interrupt oder Reset wieder aufgenommen.
<b>WAIT</b>		Wait
WAIT	4	Der Prozessor wird solange in einen Wartezustand versetzt, bis der BUSY-Eingang inaktiv wird.
<b>ESC code,source</b>		Escape
ESC const,reg ESC const,[mem]	2 8+ea	Es wird bewirkt, dass ein anderer Prozessor Befehle empfangen und den Adressbus benutzen kann.
<b>LOCK code</b>		Lock Bus
LOCK code	2	Ein Präfixbyte, welches bewirkt, dass während der Befehlsausführung von <code> der Bus für andere Anwendungen gesperrt bleibt.

## Das Flagregister

x	x	x	x	<b>O</b>	D	I	T	<b>S</b>	<b>Z</b>	x	A	x	<b>P</b>	x	<b>C</b>
15															0

Im Folgenden finden nur die Flags Beachtung, die im Zusammenhang mit den bedingten Sprüngen für die Gestaltung des Programmablaufes genutzt werden können:

### CF: Carry-Flag (Übertragsflag)

- wird auf "1" gesetzt, wenn bei einer arithmetischen Operation ein Übertrag bzw. ein Borgen erfolgt, andernfalls auf "0"
- dient bei Schiebe- und Rotationsbefehlen als 1-Bit-Zwischenspeicher und wird in Abhängigkeit von der zu verschiebenden Bitkombination beeinflusst

### PF: Parity-Flag (Paritätsflag)

- wird auf "1" gesetzt, wenn im Ergebnis einer Operation die Anzahl der Einsen gerade ist, andernfalls auf "0"

### ZF: Zero-Flag (Nullflag)

- wird auf "1" gesetzt, wenn das Ergebnis einer Operation Null ist, andernfalls auf "0"

### SF: Sign-Flag (Vorzeichenflag)

- wird auf "1" gesetzt, wenn das höchstwertige Bit (MSB) des Ergebnisses gleich eins ist, andernfalls auf "0"

### OF: Overflow-Flag (Überlaufflag)

- wird auf "1" gesetzt, wenn das Ergebnis einer (vorzeichenbehafteten) arithmetischen Operation den gültigen Zahlenbereich überschreitet, andernfalls auf "0"

Die einzelnen Befehle beeinflussen die Flags in der nachfolgenden Art und Weise (hier nicht aufgeführte Befehle: siehe jeweilige Funktionsbeschreibung):

Befehle	SF	CF	ZF	OF	PF
MOV, XCHG, IN, OUT, PUSH, POP, LAHF, XLAT, LEA, LDS, LES, CBW, CWD, JMP, Jcond, LOOP, CALL, RET	—	—	—	—	—
ADD, ADC, SUB, SBB, CMP	x	x	x	x	x
INC, DEC	x	—	x	x	x
NEG	x	a	x	x	x
DAA, DAS	x	x	x	x	x
AAA, AAS	?	x	?	?	?
MUL, IMUL	?	x	?	x	?
DIV, IDIV	?	?	?	?	?
AAD, AAM	x	?	x	?	x
NOT	—	—	—	—	—
AND, TEST, OR, XOR	x	0	x	0	x
SHL, SAL, SHR, SAR	x	x	x	x	x
ROL, ROR, RCL, RCR	—	x	—	x	—
MOVS, LODS, STOS	—	—	—	—	—
SCAS, CMPS	x	x	x	x	x

- x Flag wird in Abhängigkeit des Operationsergebnisses nach obigen Regeln beeinflusst
- Flag wird nicht geändert
- ? Flag wird unbestimmt verändert
- 0 Flag wird unabhängig vom Operationsergebnis auf "0" gesetzt
- a Flag wird auf "0" gesetzt, wenn der Operand den Wert 0 hat, sonst auf "1"

## Bedingte Sprungbefehle

Befehl	Bedingung	Jump if ...
JC	CF=1	... Carry
JNC	CF=0	... not Carry
JO	OF=1	... Overflow
JNO	OF=0	... not Overflow
JS	SF=1	... Sign
JNS	SF=0	... not Sign
JP   JPE	PF=1	... Parity   Parity Even
JNP   JPO	PF=0	... not Parity   Parity Odd
JZ	ZF=1	... Zero
JNZ	ZF=0	... not Zero

Speziell für die Anwendung nach dem Vergleich mit CMP kann auf folgende Befehlssyntax zurückgegriffen werden:

Vergleich			Jump if ...
JE	=	ZF=1 (wie JZ)	... equal
JNE	!=	ZF=0 (wie JNZ)	... not equal
Vergleich <u>vorzeichenbehafteter</u> Zahlen			
JG	>	((SF xor OF) or ZF) = 0	... greater
JNLE	>		... not (less or equal)
JGE	=>	(SF xor OF) =0	... greater or equal
JNL	=>		... not less
JNG	<=	((SF xor OF) or ZF) = 1	... not greater
JLE	<=		... less or equal
JNGE	<	(SF xor OF) =1	... not (greater or equal)
JL	<		... less
Vergleich <u>vorzeichenloser</u> Zahlen			Jump if ...
JA	>	(CF or ZF) = 0	... above
JNBE	>		... not (below or equal)
JAE	=>	CF=0 (wie JNC)	... above or equal
JNB	=>		... not below
JNA	<=	(CF or ZF) = 1	... not above
JBE	<=		... below or equal
JNAE	<	CF=1 (wie JC)	... not (above or equal)
JB	<		... below

## Adressierungsarten und Taktangaben

Adressierungsarten		
Register-adressierung	MOV AX, CX	Quelle und Ziel einer Operation sind Register
implizit	DAA NOP	Operand ist durch Befehl eindeutig festgelegt, keine separate Angabe
immediate, unmittelbar	MOV AL, 200	Befehl enthält unmittelbar den Operanden (Konstante)
direkt	MOV AL, [200H] OUT 90H, AL	Befehl enthält direkt die Adresse eines Speicheroperanden oder I/O-Operanden
indirekt	MOV AL, [BX] IN AL, DX	Befehl enthält Information zur Bestimmung der Adresse des Speicheroperanden oder I/O-Operanden

Die in der Befehlsliste angegebenen Taktzahlen gelten ausschließlich für die Befehlsabarbeitung. Zu der angegebenen Taktzahl sind bei Speicherzugriffen die nachfolgend aufgeführten ea-Takte hinzuzuzählen. Steht das niederwertige Byte eines 16-Bit-Operanden auf einer ungeraden Adresse, so sind weitere 4 Takte zu addieren.

Registerverwendung und zusätzliche ea-Takte bei Speicherzugriffen		ea-Takte
direkte Adr.	[mem] = [Adresse] (Konstante)	6
indirekte Adr.	[mem] = [BX]   [BP]   [SI]   [DI]	5
	[BX+const]   [BP+const]   [SI+const]   [DI+const]	9
	[BX+SI]   [BP+DI]	7
	[BX+DI]   [BP+SI]	8
	[BX+SI+const]   [BP+DI+const]	11
	[BX+DI+const]   [BP+SI+const]	12

## Logische Funktionen

NOT

(bitweise Negation)

x

•

y

x	y
0	1
1	0

AND, TEST

(bitweise UND)

x<sub>0</sub>

x<sub>1</sub>

&

y

x <sub>1</sub>	x <sub>0</sub>	y
0	0	0
0	1	0
1	0	0
1	1	1

OR

(bitweise ODER)

x<sub>0</sub>

x<sub>1</sub>

≥1

y

x <sub>1</sub>	x <sub>0</sub>	y
0	0	0
0	1	1
1	0	1
1	1	1

XOR (bitweise Anti-valenz, Exklusiv-OR)

x<sub>0</sub>

x<sub>1</sub>

=1

y

x <sub>1</sub>	x <sub>0</sub>	y
0	0	0
0	1	1
1	0	1
1	1	0

Zahlen und Zeichen

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-128	-127	-126	-125	-124	-123	-122	-121	-120	-119	-118	-117	-116	-115	-114	-113
9	-112	-111	-110	-109	-108	-107	-106	-105	-104	-103	-102	-101	-100	-99	-98	-97
A	-96	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86	-85	-84	-83	-82	-81
B	-80	-79	-78	-77	-76	-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65
C	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	-50	-49
D	-48	-47	-46	-45	-44	-43	-42	-41	-40	-39	-38	-37	-36	-35	-34	-33
E	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17
F	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

n	5	6	7	8	9	10	11	12	13	14	15	16
2 <sup>n</sup>	32	64	128	256	512	1.024	2.048	4.096	8.192	16.384	32.768	65.536
HEX	20	40	80	100	200	400	800	1000	2000	4000	8000	1.0000

n	18	20	24	28	32	64
2 <sup>n</sup>	262.144	1.048.576	16.777.216	268.435.456	4.294.967.296	18.446.744.073.709.600.000
HEX	4.0000	10.0000	100.0000	1000.0000	1.0000.0000	1.0000.0000.0000.0000

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

# Der 8086-Mikrorechner aus Programmiersicht

