

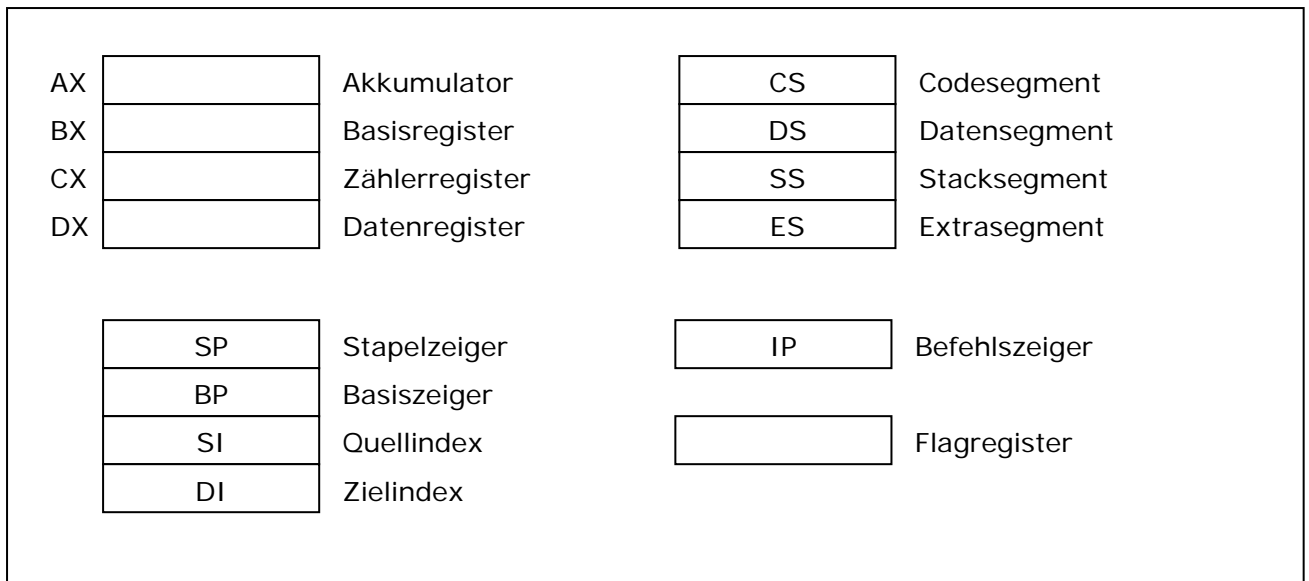
# **Der Intel 8086**

Reto Gurtner

2005

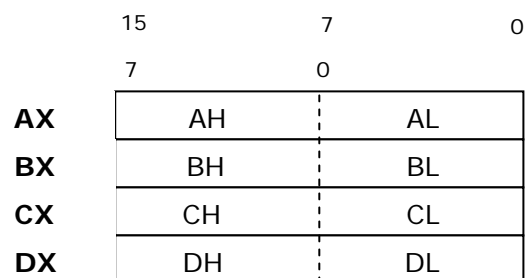
<b>1. DIE INTERNEN REGISTER .....</b>	<b>3</b>
<b>1.1 ALLGEMEINE REGISTER AX, BX, CX UND DX .....</b>	<b>3</b>
DAS AX-REGISTER .....	4
DAS BX-REGISTER .....	4
DAS CX-REGISTER .....	5
DAS DX-REGISTER .....	5
<b>1.2 DIE ADRESSREGISTER BP UND SP .....</b>	<b>6</b>
BP (BASE-POINTER-REGISTER) .....	6
SP (STACK-POINTER-REGISTER) .....	6
<b>1.3 INDEX-REGISTER SI UND DI .....</b>	<b>6</b>
<b>1.4 DAS FLAG-REGISTER (STATUS-REGISTER) SR .....</b>	<b>6</b>
<b>1.5 SEGMENT-REGISTER CS, DS, ES UND SS .....</b>	<b>7</b>
<b>2. ADRESSBILDUNG .....</b>	<b>8</b>
<b>2.1 SEGMENT .....</b>	<b>9</b>
<b>2.2 BEISPIELE .....</b>	<b>10</b>
BEISPIEL 1: .....	10
BEISPIEL 2: .....	10
BEISPIEL 3: .....	11
BEISPIEL 4: .....	11
BEISPIEL 5: .....	11
<b>3. SPEICHER-MODELLE .....</b>	<b>12</b>
<b>3.1 TINY .....</b>	<b>12</b>
<b>3.2 SMALL .....</b>	<b>12</b>
<b>3.3 MEDIUM .....</b>	<b>12</b>
<b>3.4 COMPACT .....</b>	<b>12</b>
<b>3.5 LARGE .....</b>	<b>12</b>
<b>3.6 HUGE .....</b>	<b>12</b>
<b>4. ADRESSIERUNGSARTEN .....</b>	<b>13</b>
<b>4.1 KURZÜBERSICHT .....</b>	<b>13</b>

# 1. Die internen Register



## 1.1 Allgemeine Register AX, BX, CX und DX

- Register für arithmetische, logische und Ein-/Ausgabeoperationen
- Zwischenspeicherung beliebiger Programmdateien
- Parameterübergabe bei Aufrufen von Betriebssystemroutinen
- Können als 16 oder 8-Bit Register eingesetzt werden
- Jedes Register ist unterteilbar in ein höherwertige und eine niederwertige 8-Bit Hälfte



## Das AX-Register

- AX-Register = Akkumulator
- Ist hauptsächlich für das Durchführen von Rechenoperationen gedacht.
- Von allen 4 Registern hat es eine gewisse Sonderstellung → Es gibt arithmetische Operationen, die nur über das AX Register ausgeführt werden können
- Ebenso gibt es Befehle die sich auf das AX-Register beziehen, ohne dass im Befehl das AX-Register auftaucht.

### Beispiel:

angenommen, daß *length* und *width* zwei Byte-Variablen sind, könnte die Fläche des entsprechenden Vierecks wie folgt berechnet werden:

```
MOV AL, length
MUL width           //erster Operand implizit in AL...
                   //...und Resultat implizit in AX
```

---

## Das BX-Register

- BX-Register = **Basisregister**
- Kann als **Adressregister bei indirekten Adressierung** verwendet werden.
- Ansonsten ist es **frei verwendbar!**

### Beispiel:

angenommen, daß die Zeichenkette *text* schon als 'ABCD' definiert ist, kann das dritte Zeichen von links wie folgt in ein 'A' geändert werden:

```
MOV BX, OFFSET text //Bekomme die Startadresse des String ins BX
ADD BX, 2           //springe zu der dritten Position des Strings
MOV [BX], 'A'      // Ersetze was an dieser Position ist mit einem A
```

### Erklärungen:

- BX = Inhalt des Registers
- [BX] = Inhalt der Speicherzelle an Adresse gleich des Inhalts von BX

Adressen	....	n	n+1	n+2	n+3	n+4	....
Inhalte	-	,A'	,B'	,C'	,D'	,E'	-

Nach Durchführung der ersten Zeile beinhaltet BX n.  
Vor Durchführung der dritten Zeile ist [BX] gleich 'C'.  
Nach Durchführung der dritten Zeile ist *text* gleich 'ABADE'.

## Das CX-Register

- CX-Register = **Zählregister**
- Dient für einige Maschinenbefehle als Zählregister
- Der im CX-Register gespeicherten Wert wird verwendet um z.B als Schleifenzähler innerhalb einer Programmschleife die Anzahl Durchläufe festzulegen
- Dient das CX nicht als Zähler, so ist es frei verwendbar

### Beispiel:

Ohne das CX-Register:

```
MOV BX, 10           // Schreibe 10 in das BX
start_of_loop:        // <Befehle im Körper der Schleife>
DEC BX              // Dekrementiere das BX
CMP BX, 0           // Vergleiche das BX mit 0
JNE start_of_loop   // jumpe zu start_of_loop, wenn das BX != 0 ist
```

Mit dem CX-Register:

```
MOV CX, 10          // Schreibe 10 in das BX
start_of_loop:        // < Befehle im Körper der Schleife >
LOOP start_of_loop  // Macht das gleiche wie oben: Dec, CMP, JNE
```

---

## Das DX-Register

- Unterstützt das AX-Register bei bestimmten arithmetischen Operationen, die mit 32-Bit operanden arbeiten.

### Beispiel:

angenommen, daß *length* und *width* zwei Wort-Variablen sind, könnte die Fläche des entsprechenden Vierecks wie folgt berechnet werden:

```
MOV AX, length      // Bringe length in das AX
MUL width           // erster Operand implizit in AX...
                    // ..und Resultat implizit in DX:AX
```

## 1.2 Die Adressregister BP und SP

### BP (Base-Pointer-Register)

- Wird gebraucht für die **Adressberechnung**
- Steht ansonsten **frei zur Verfügung**

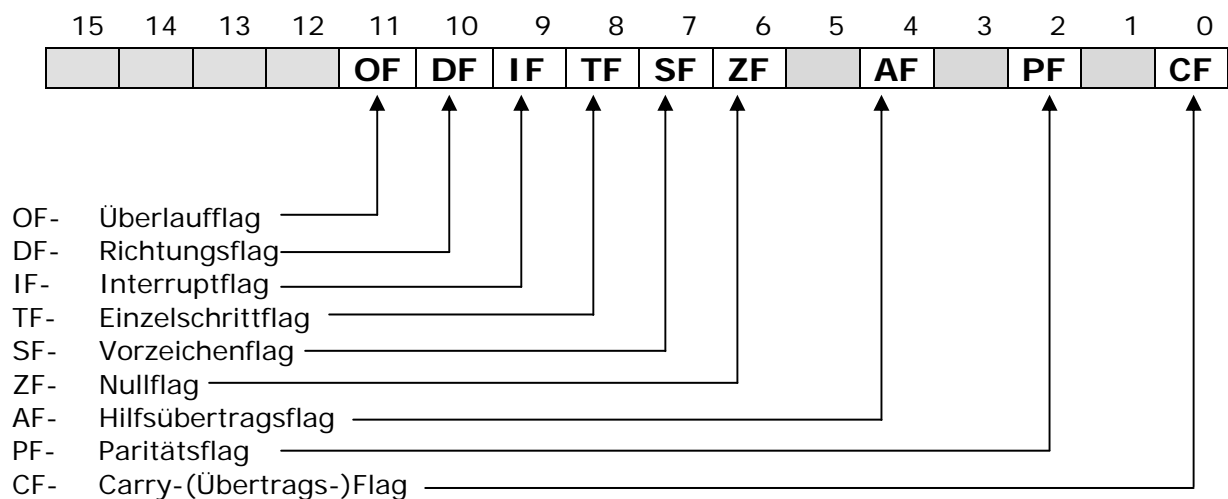
### SP (Stack-Pointer-Register)

- Dient zusammen **mit dem SS-Register zur Verwaltung des Stacks**
- Sollte **nicht für andere Zwecke** eingesetzt werden!

## 1.3 Index-Register SI und DI

- SI = Index-Register
- DI = Destination Index
- Können zur indirekten Adressierungsberechnung werden

## 1.4 Das Flag-Register (Status-Register) SR

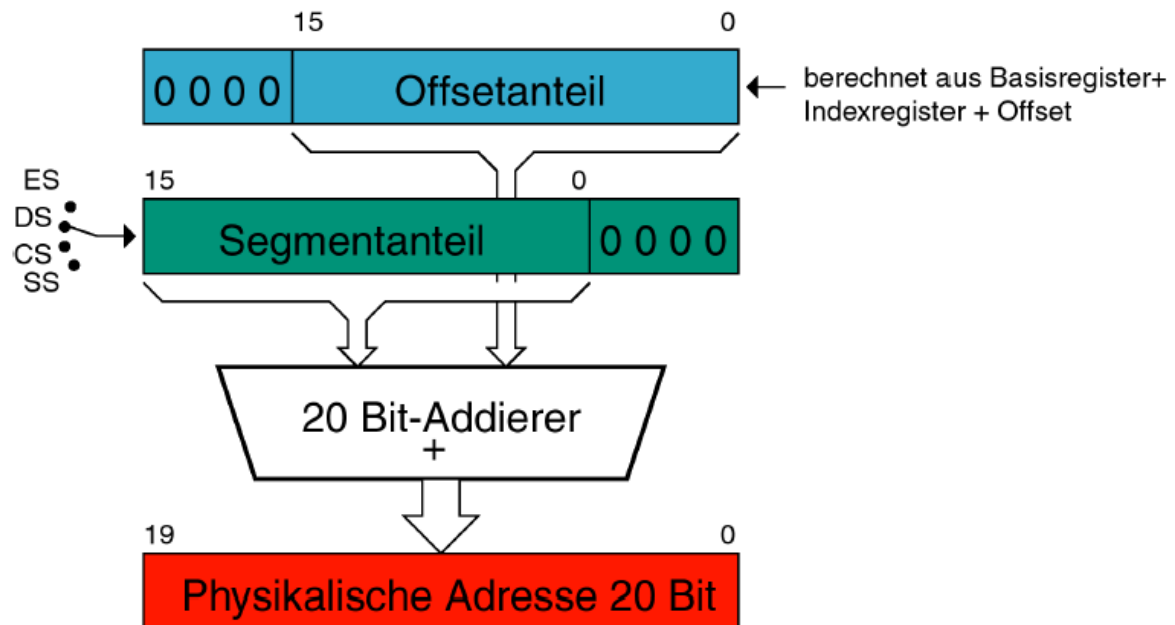


## **1.5 Segment-Register CS, DS, ES und SS**

- CS = Code Segment
- DS = Daten Segment
- ES = Extra Segment
- SS = Stack Segment

## 2. Adressbildung

- Der Intel 8086 hat 16 Bit-Register und einen 20 Bit Adressbus
- Eine Adresse kann nicht in einem Register gespeichert werden → Sie werden aus 16-Bit Registern zusammengesetzt!



- Der Segmentanteil kommt aus einem speziellen Register, einem der Segmentregister (CS, DS, SS, ES):
- Der Offsetanteil wird aus den Registern (BX, BP, DI, SI) und/oder Konstanten während der Laufzeit berechnet!
- Der Intel 8086 rechnet:

$$\text{Physikalische Adresse} = \text{Offsetadresse} + \text{Segmentadresse} * 16$$

- Ebenso gibt es die:

**Logische Adresse:**  
Segment : Offset

## 2.1 Segment

- Den Bereich, den man adressieren kann, ohne ein Segmentregister zu verändern, nennt man ein Segment
- **Daten, Befehle und momentane Variablen** sind in einem separaten Teil des Speichers abgelegt → **in den Segmenten**
- **Ein Segment** umfasst maximal **65536 Byte** → **64 KB!**
- Ein Programm kann **nur 4 Segmente gleichzeitig adressieren**
- **Ein Segmentanfang liegt immer auf einem Vielfachen von 16!**
- **Segmente können überlappen! (Bsp. 1)**
- Der abstand eines Speicherplatzes relativ zum Segmentanfang ist der Offset!
- Die Berechnung von Segment und Offset aus einer physikalischen Adresse ist mehrdeutig:

**<segmentanfang>:<Offset> → logische Adresse**

Die Adresse 035B0h:

▪ 0350:00B0	}	z.B: 03500
▪ 035B:0000		
▪ 024A:1110		
		+ 00B0

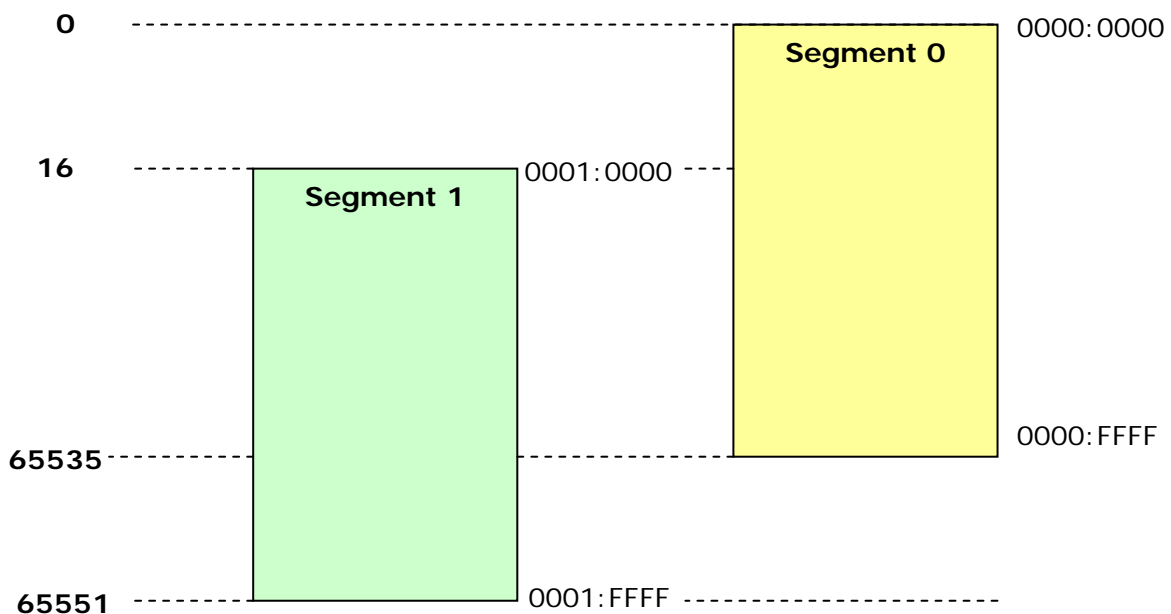
- Man braucht 2 Register (4 Byte) um eine Adresse abzuspeichern (32 belegte Bit um eine 20-Bit-Adresse aufzunehmen!)
- Man braucht 2 Register um den Programm Counter darzustellen: CS und IP. Der nächste auszuführende Befehl wird geladen von CS:IP

## 2.2 Beispiele

### Beispiel 1:

- Ein Segment besitzt die **Segmentadresse 0**
- Durch den 16-Bit-Offset lassen sich die **Adressen 0:0 bis 0:65535** adressieren (ersten 65535 Bytes)
- Das zweite Segment besitzt die **Segmentadresse 1**
- Durch den 16-Bit-Offset sich die Adressen **1:0 bis 1:65535** adressieren
- Ergibt umgerechnet: von **16 bis 65551**

#### physikalische Adresse



### Beispiel 2:

- Der Inhalt der Speicherzelle mit der Adresse 700'000 soll gelesen werden
- Um diese Adresse auf dem Adressbus zu erzeugen muss ein Segment-Register und ein Offset miteinander verknüpft werden

#### Mögliche Lösung:

IP-Register (Offset)      60000  
 CS-Register (Segment)   40000 (x 16)  
 Resultat:                    700 000

CS-Register (Segment)	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0			
IP-Register (Offset)							1	1	1	0	1	0	1	0	0	0	1	1	0	0	0	0
Physikalische Adresse	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0



Entspricht dem dezimalwert von 700 000

### Beispiel 3:

- Welche physikalische Adresse wird durch folgende Angaben festgelegt:  
2100:0003 ?

$$\begin{array}{r} 2100\text{h} \\ 0003\text{h} \\ \hline 21003\text{h} \end{array} = 135171$$

### Beispiel 4:

Segmentanfang und Ende bestimmen:

Segment Anfang:

Logische Adresse: A200:0000

Physikalische Adresse: A200h

Segment Ende:

Logische Adresse: A200:FFFF

Physikalische Adresse: B1FFFh

### Beispiel 5:

Mehrdeutigkeit von der Adresse B8012h

Logische Adresse	Physikalische Adresse
B800:0012	B8012h
B000:8012	B8012h
B401:4002	B8012h
...	...

## 3. Speicher-Modelle

### 3.1 *Tiny*

**CS = DS = SS**

- Code, Daten und Stapel im gleichen Segment!

### 3.2 *Small*

**SS = DS ; CS**

- Code in einem eigenen Segment
- Daten und Stapel zusammen in einem separaten Segment

### 3.3 *Medium*

**mehrere CS; DS = SS**

- Code eventuell in mehreren Segmenten
- Daten und Stapel zusammen in einem separaten Segment

### 3.4 *Compact*

**CS; SS; mehrere DS**

- Code, Daten und Stapel in separaten Segmenten

### 3.5 *Large*

**mehrere CS; mehrere DS; SS**

- Code and Daten in möglicherweise mehreren Segmenten
- Stapel in einem separaten Segment

### 3.6 *Huge*

- Wie Large, aber komplexe Daten können sich über mehrere Segmente dehnen

## 4. Adressierungsarten

### 4.1 Kurzübersicht

Adressierungsart	Format	Segment-Register
Register	Register	Keines
Unmittelbar	Wert	Keines
Register-Indirekt	[BX]	DS
	[BP]	SS
	[DI]	DS
	[SI]	DS
	Label[BX]	DS
Basisrelativ *	Label[BP]	SS
	Label[DI]	DS
Direkt indiziert *	Label[SI]	DS
	Label[BX + SI]	DS
Basis indiziert *	Label[BX + DI]	DS
	Label[BP + SI]	SS
	Label[BP + DI]	SS

\* Das Label kann durch ein Displacement ersetzt werden, so dass zum Beispiel anstelle von TABELLE[DI] auch [10+DI] geschrieben werden kann.