TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik, Professur für Prozessleittechnik

Mikrorechentechnik 1

Befehlssatzarchitektur

Professur für Prozessleittechnik
Wintersemester 2011/2012

Qualifikationsziele

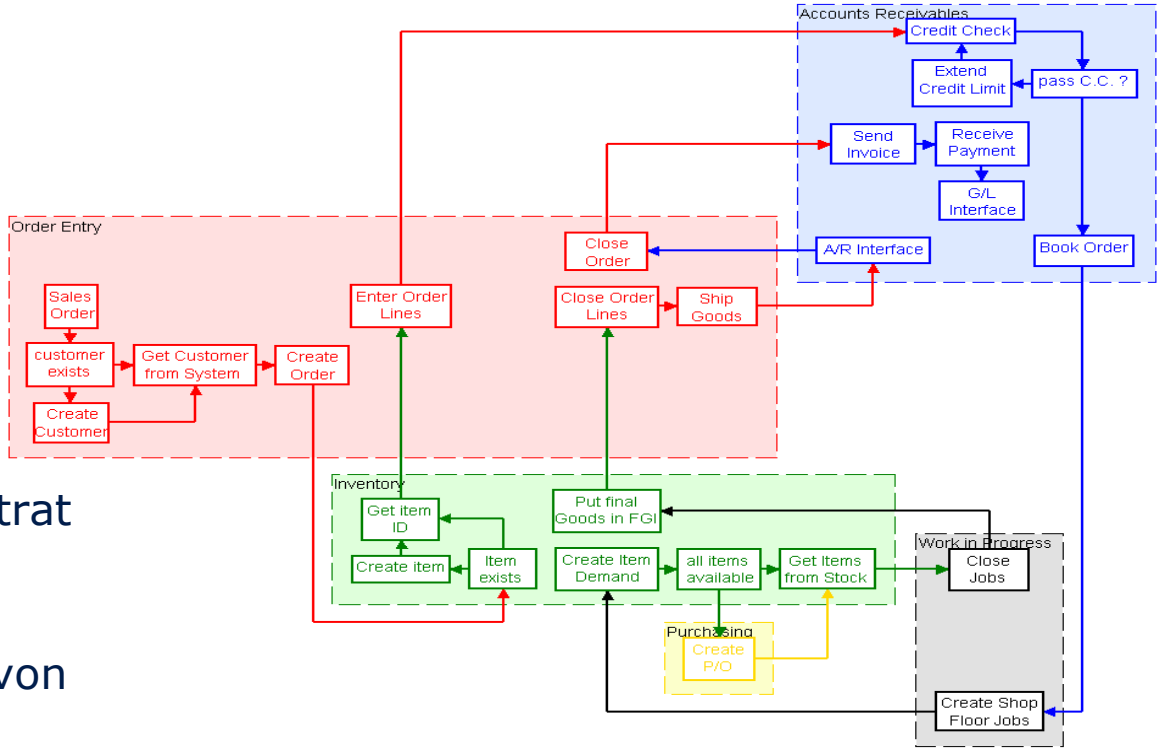
- Sie können sich die notwendigen Kenntnisse zur Programmierung eines neuen Mikroprozessors selbstständig erarbeiten, d.h.
  - Sie verstehen das grundlegende Architekturkonzept,
  - Sie können sich die Register und ggf. deren Besonderheiten erarbeiten,
  - Sie kennen die Adressierungsarten,
  - (sie wissen wo sie nachsehen müssen)
- Sie kennen die Grundstruktur der 80x86-Familie
  - ... und das 16-Bit-Erbe, das diese Prozessoren mit sich rumschleppen.

Definition Befehlssatzarchitektur

- ... the attributes of a system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation (Amdahl, Blaaw, und Brooks, 1964)

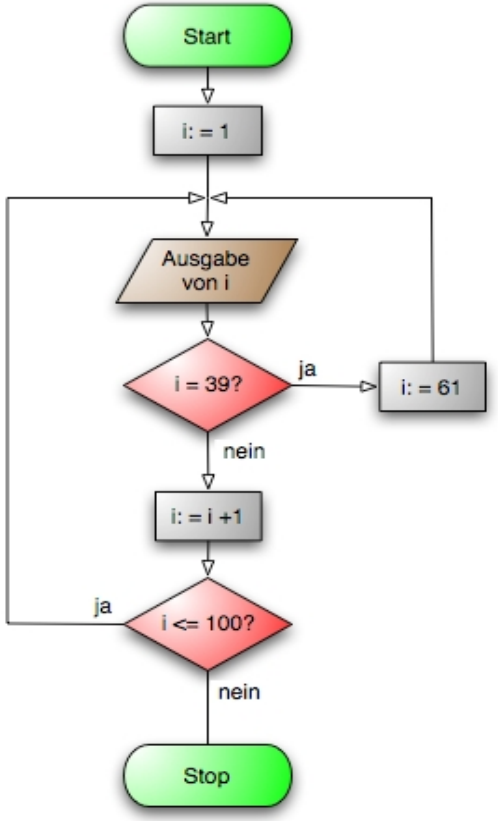
Objektorientierte Sichtweise (Java, C++)

- Denken in Systemen aus Objekten
  - Objekt = Instanz einer Klasse mit Attributen & Methoden
  - Unabhängig von technischem Substrat
  - In ersten Entwurfsschritten auch unabhängig von Berechnung



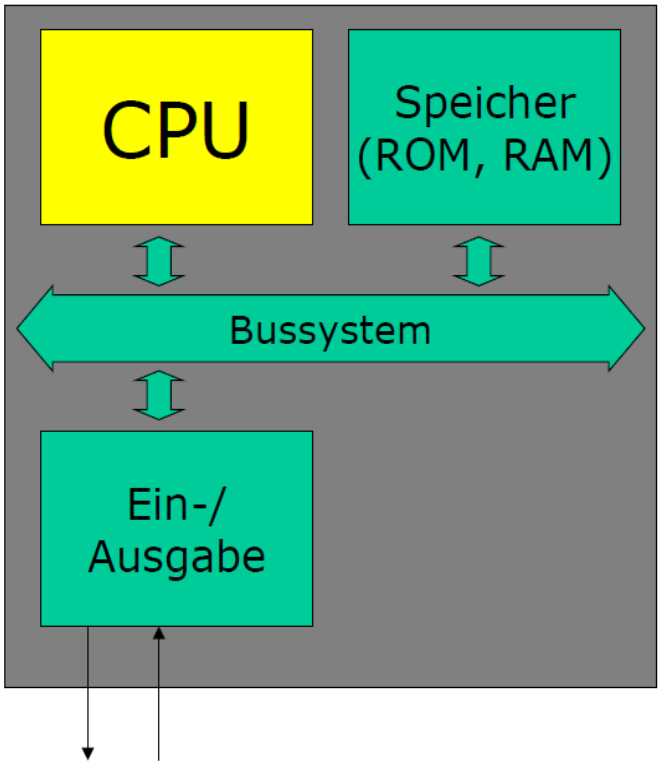
Prozedurale Sichtweise (C, Fortran)

- Denken in Variablen und Prozeduren
  - Algorithmus = Rezept zur Manipulation von Variablen (= Datenstrukturen) mit Hilfe von geeigneten Instruktionen und Prozeduren
  - Weitgehend unabhängig vom technischen Substrat
  - Berechnungsorientiert



Befehlssatzorientiert (Assembler)

- Zusammenspiel der Komponenten der CPU
  - Verschieben von Speicherstellen oder IOs in Register
  - Manipulation von Registerinhalten
  - Verschieben von Registerinhalten in Speicher oder IOs
  - Hochspezifisch für die jeweilige Rechnerarchitektur



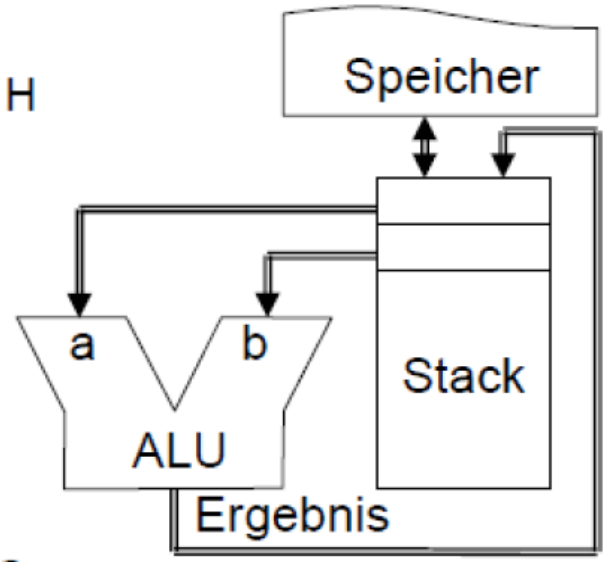
Klassifizierungskriterium

- Art und Menge der an ALU und Speicher angebundenen Register
  - Stack-Architektur
  - Akkumulator-Architektur
  - Allzweckregister-Architekturen
    - Register-Register-Maschinen (Load-Store-Architektur)
    - Register-Speicher-Maschinen
    - Speicher-Speicher-Maschinen
- Reale Prozessoren sind häufig „Mischlinge“

Stack-Architektur

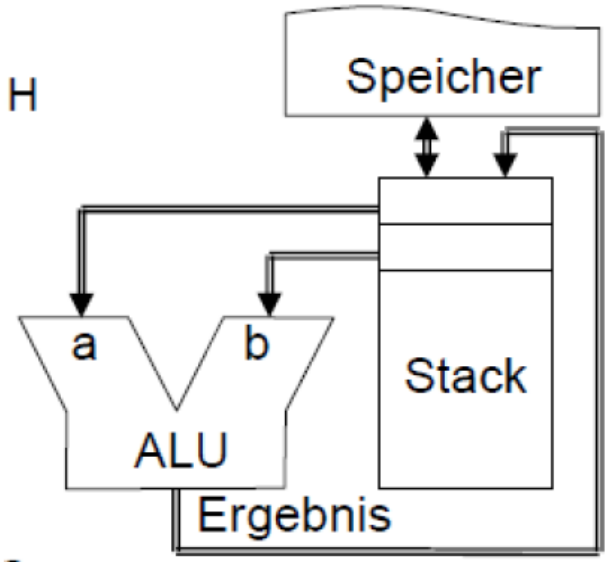
- Ergebnisspeicher sind als Stack (LIFO) realisiert
- Häufig nur Zugriff auf oberste Speicherstelle des Stacks realisiert (PUSH, POP)
- Beispiel: C ← A + B

```
PUSH A ; Stack ← A
PUSH B ; Stack ← B
ADD ; Stack ← pop + pop
POP C ; C ← Stack.
```



Diskussion Stack-Architektur

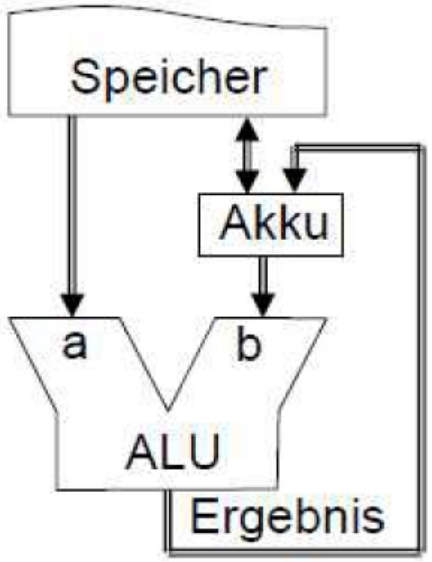
- Einfache Realisierung
- Wenige Befehle mit fester Länge
- Null-Adress-Maschine
  - Für Operationen werden **keine** Adressen angeben.
  - Operatoren und Ergebnis liegen an vordefinierten Stellen im Stack



Akkumulator-Architektur (Praktikum Compi16)

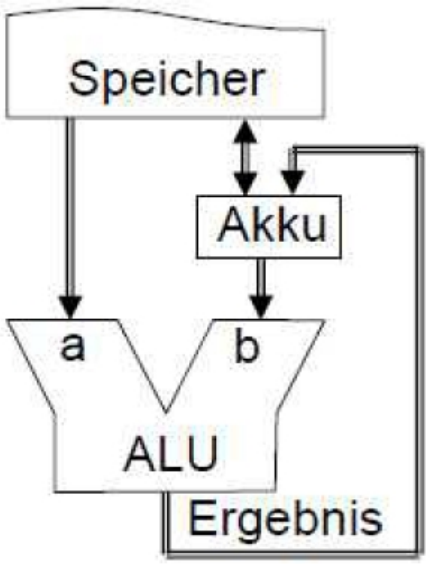
- Ein ausgezeichnetes Register: **Akkumulator** (Akku)
- **Akku** wirkt implizit an allen Operationen mit
- LOAD (LDA) und STORE (STA) wirken nur auf Akku.
- Beispiel: C ← A + B

```
LDA A       ; AR ← A
ADD B       ; AR ← AR + B
STA C       ; C ← AR
```



Diskussion Akkumulator-Architektur

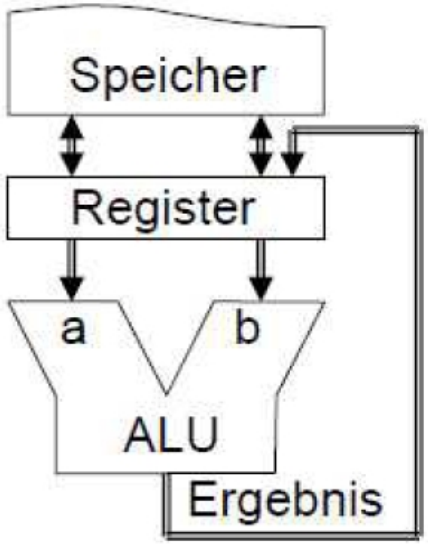
- Sehr kompaktes Befehlsformat möglich
- Ein-Adress-Maschine
  - Es muss nur **ein** Operand angegeben werden,
  - Ein ggf. zweiter und das Ergebnis liegen in einem vordefiniertem Speicher, dem Akku



Register-Register-Architektur (DLX Informatik 2)

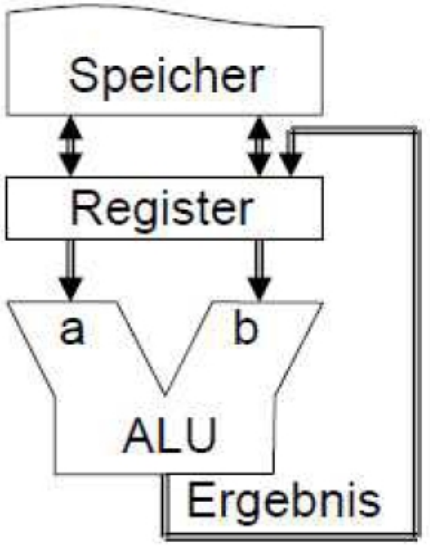
- Operationen arbeiten mit Registern
- Zugriff auf Speicher/Register mit LOAD und STORE
- Üblicherweise viele Register (32-512)
- Beispiel: C ← A+B

```
LOAD R1, A          ; R1 ← A
LOAD R2, B          ; R2 ← A
ADD R3, R1, R2      ; R3 ← R1 + R2
STORE C, R3         ; C  ← R3
```



Diskussion Register-Register-Architektur

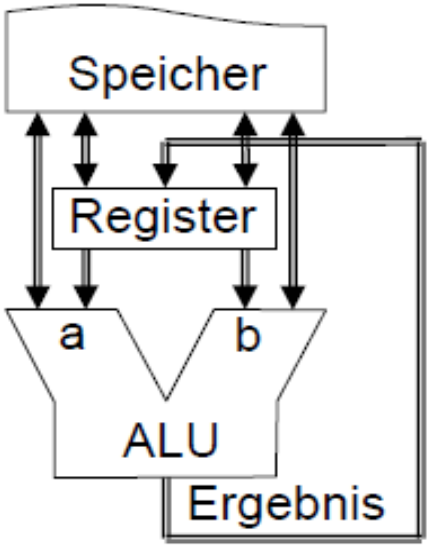
- wenige Befehle fester Länge
- Abgesehen von LOAD/STORE brauchen alle Instruktionen in etwa gleich viele Takte



Register-Speicher Architektur

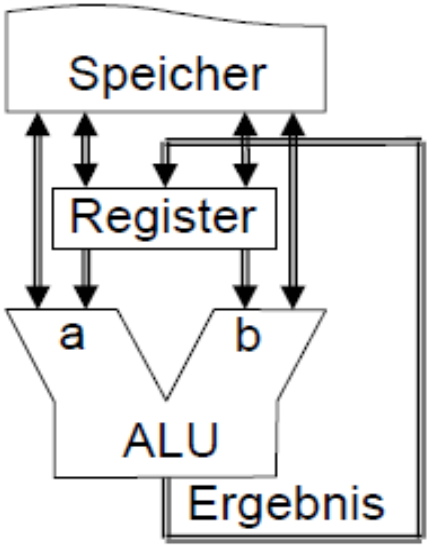
- Operationen greifen auf Register und/oder Speicher zu
- üblicherweise wenige Register
- Beispiel: C ← A + B

```
MOV AX, A   ; AX ← A
ADD AX, B   ; AX ← AX + B
MOV C, AX   ; C  ← AX
```



Diskussion Speicher-Register Architektur

- Viele unterschiedliche Wege
- Viele, zum Teil sehr komplexe Befehle variabler Länge und Ausführungszeit (CISC)
- Komplexes Steuerwerk notwendig



Modellprozessor 80x86

Warum 80x86?

- Geringe zusätzliche Kosten:
  - Hohe Verfügbarkeit von x86-Prozessoren (bzw. Emulatoren für mac, ppc) bei den Studierenden
- Gute Dokumentation:
  - Vielfältige Literatur und Software.
- Angemessenheit:
  - Grundprinzipien der Befehlsarchitektur, der Besonderheiten/Ungereimtheiten und der Programmierung sind auf andere Architekturen gut übertragbar.

c't 13/2003 zur x86-Prozessor-Architektur

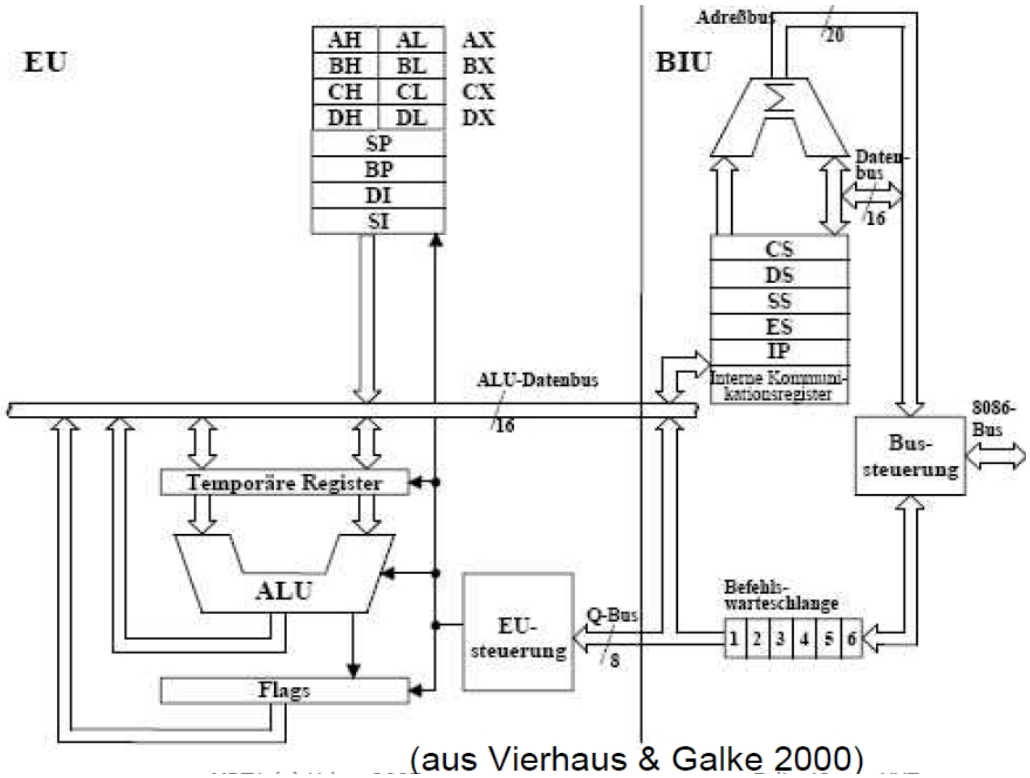
- „Vieles hat sich in den letzten Jahren in der Computerszene verändert. Das Diskettenlaufwerk ist längst passé, der DVD-Brenner gehört mittlerweile zum Standard. MS-DOS ist in einer winzigen Nische von Windows XP verschwunden, und statt PacMan spielt man heute Egoshooter in 3D.
- Nur die x86er-Prozessor-Architektur steckt noch in jedem PC. Dabei war der 8086-Prozessor damals gar nicht revolutionär, sondern nur eine logische Fortentwicklung bestehender 8-Bit-Systeme. Durchgesetzt hat er sich einige Jahre später vor allem, weil IBM den ersten Personal Computer mit einem 8088-Prozessor ausstattete […]“

Befehlssatzarchitektur 8086

- Struktureller Aufbau der CPU (soweit relevant für das Verständnis von Besonderheiten)
- Anbindung der Register an Speicher, ALU (und andere Rechenwerke)
- Funktion der Register

Blockschaltbild 8086 CPU

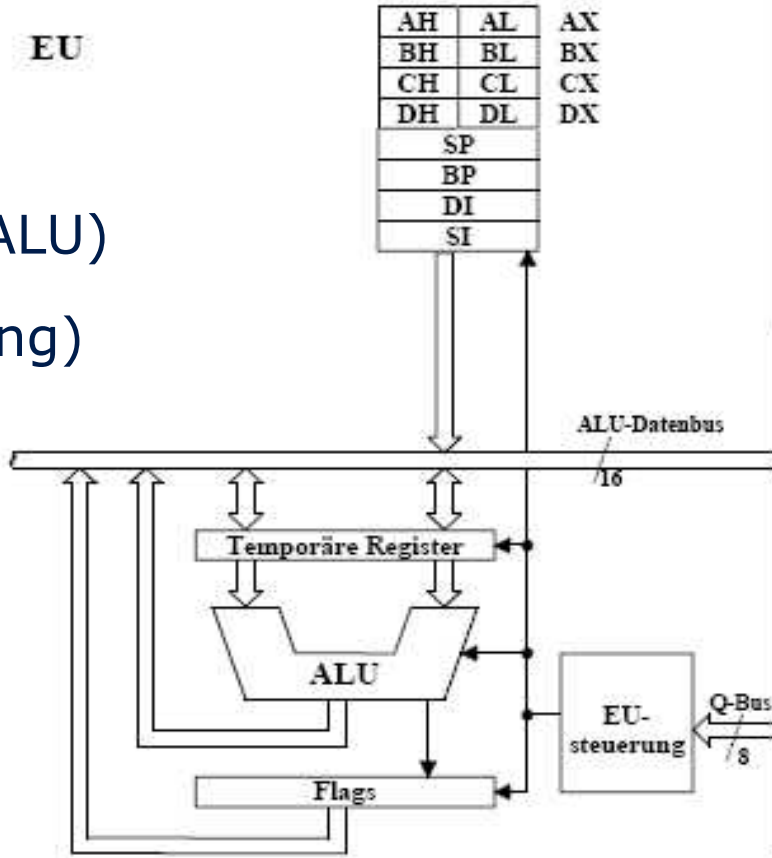
- Execution Unit (EU)
  - Befehlskodierung
  - Ausführung
- Bus Interface Unit (BIU)
  - Datenhandling
  - Befehlsspeicher
  - Adressberechnung



(aus Vierhaus & Galke 2000)

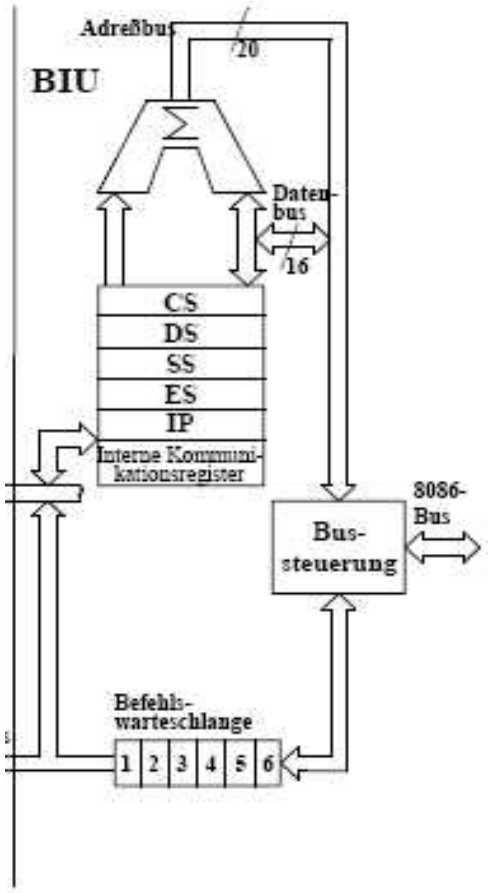
8086 Execution Unit

- Arithmetische logische Einheit (ALU)
- Steuereinheit (Befehlsdekodierung)
- Statusregister (Flags)
- 4 Mehrzweckregister
  - 16 Bit AX, BX, CX, DX
  - Teilbar für 8-Bit Zugriffe
- 4 Zeiger/Indexregister
  - 16 Bit SP, BP, DI, SI



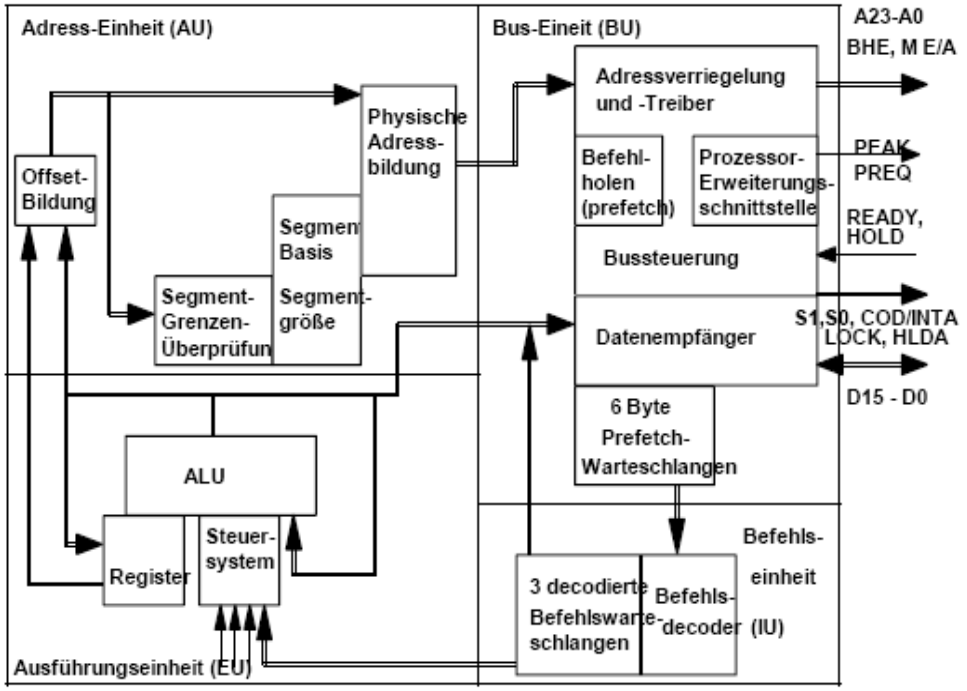
8086 Bus Interface Unit

- Addressaddierwerk mit 20 Bit Addressbreite
- Bussteuerung
- Befehlswarteschlange
- 4 Register für die Adressberechnung
  - 16 Bit CS, DS, SS, ES
- Befehlszeigerregister
  - 16 Bit IP



Exkurs 80286

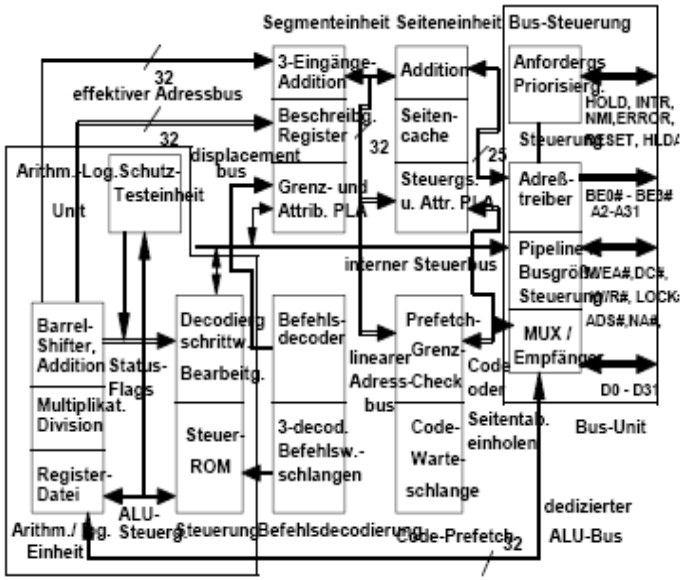
- Aufwärtskompatible Weiterentwicklung
  - 16 Bit Datenbus
  - 24 Bit Addressbus
- Vier nebenläufig arbeitende Einheiten:
  - AU, EU, BU, IU



(aus Vierhaus & Galke 2000)

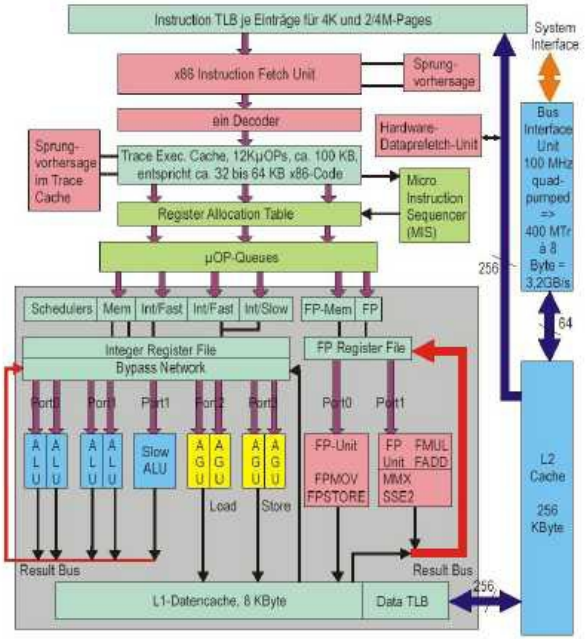
Exkurs 80386

- 34 32-bit Register
- Pipeline



Pentium

- Superskalare Architektur



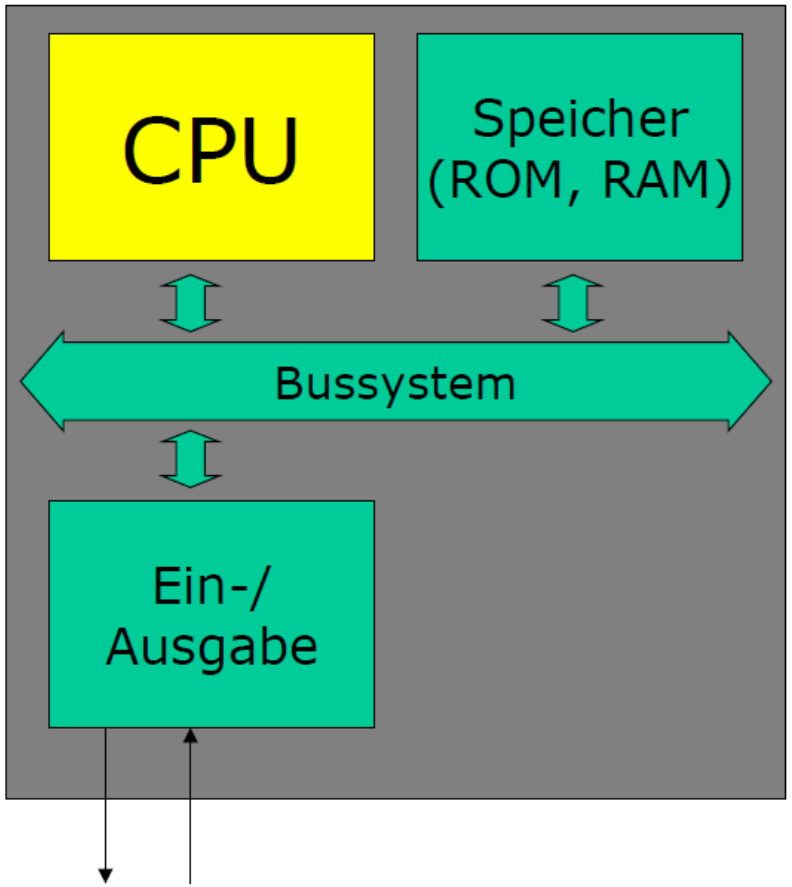
Aufwärtskompatibilität?

- Drastische Änderungen in der Architektur:
  - 16bit -> 32bit -> 64bit
  - EU/BIU -> Superskalare Architektur
  - CISC -> RISC + Microcode
- Wieso läuft ein maschinennahes Programm für 8086 auch auf allen späteren Prozessoren?
- Verschiedene Betriebsarten mit leicht unterschiedlicher aber syntaxkompatibler Befehlssatzarchitektur:
  - 16-bit Real Mode (8086, z.B. MSDOS)
  - 16-bit Protected Mode (ab 286 → Win3.1)
  - 32-bit Protected Mode (ab 386 → NT/2K/XP, Linux)

Befehlssatzarchitektur des 80x86-Prozessors (im 16-Bit Real Modus)

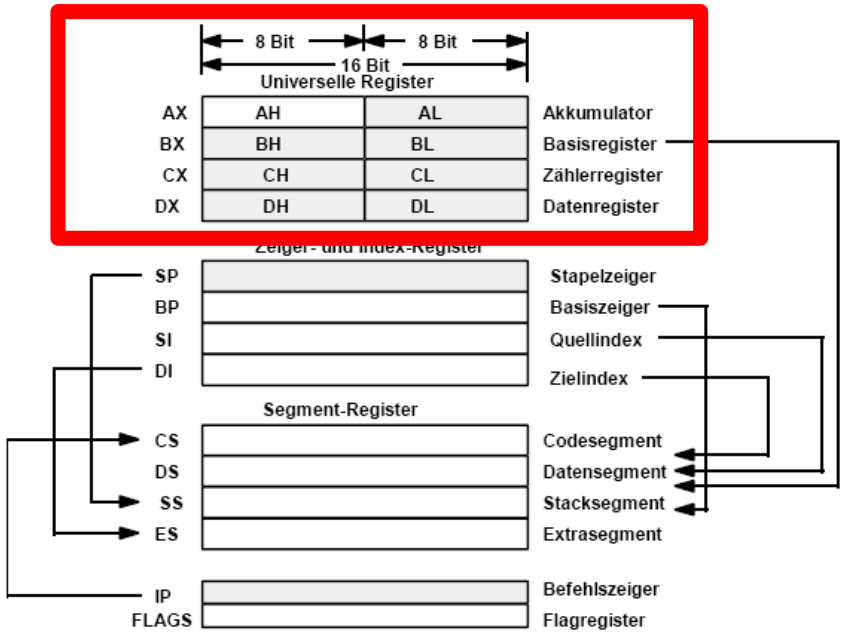
8086 Befehlssatzarchitektur

- CPU
  - Zwitter Akkumulator- / Register-Speicher-Architektur
  - 14 **16 Bit** Register
    - 4 x Universelle Register
    - 4 x Zeiger/Indexregister
    - 4 x Segmentregister
    - + Flagregister, + Befehlszeiger
- Speicher (Memory)
  - **20 Bit** Adressraum (1 Mbyte)
- Ein-/Ausgabe (I/O)
  - Spezielle Ein/Ausgabebefehle



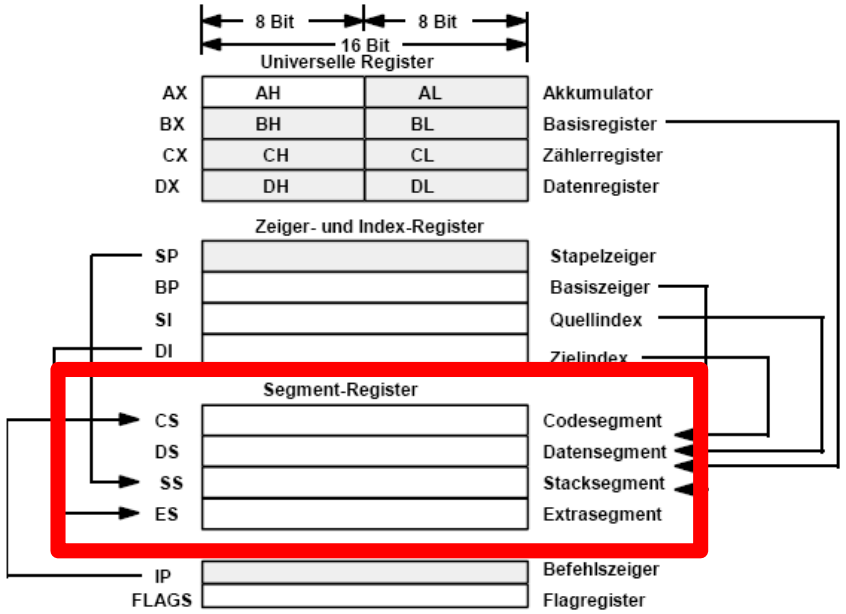
Universelle Register (mit Spezialfunktionen)

- **AX** – Akkumulator
  - Multiplikation, Division
- **BX** – Basisregister
  - indirekte Adressierung.
- **CX** - Zählerregister
  - Schleifenoperationen.
- **DX** – Datenregister
  - Mult, Division von 16-Bit-Worten
  - Kanalnummer bei I/O-Befehlen



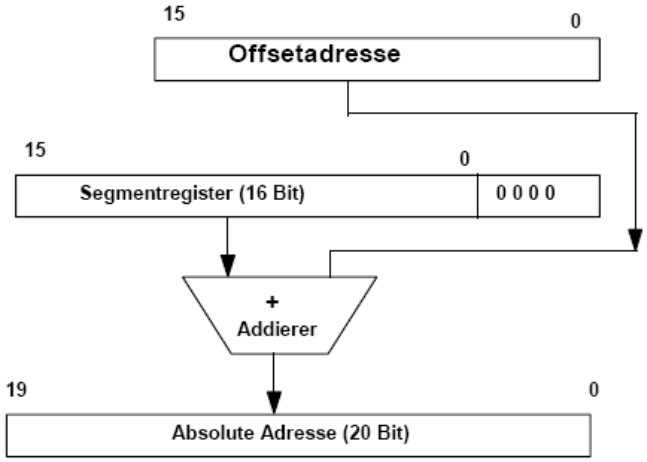
Register zur Speicherverwaltung

- **CS** - Code-Segment
  - Segment für Befehle
- **DS** – Daten-Segment
  - Segment für Daten
- **SS** - Stack-Segment
  - Segment für den Stapelspeicher
- **ES** - Extrasegment
  - für Zeichenkettenoperationen



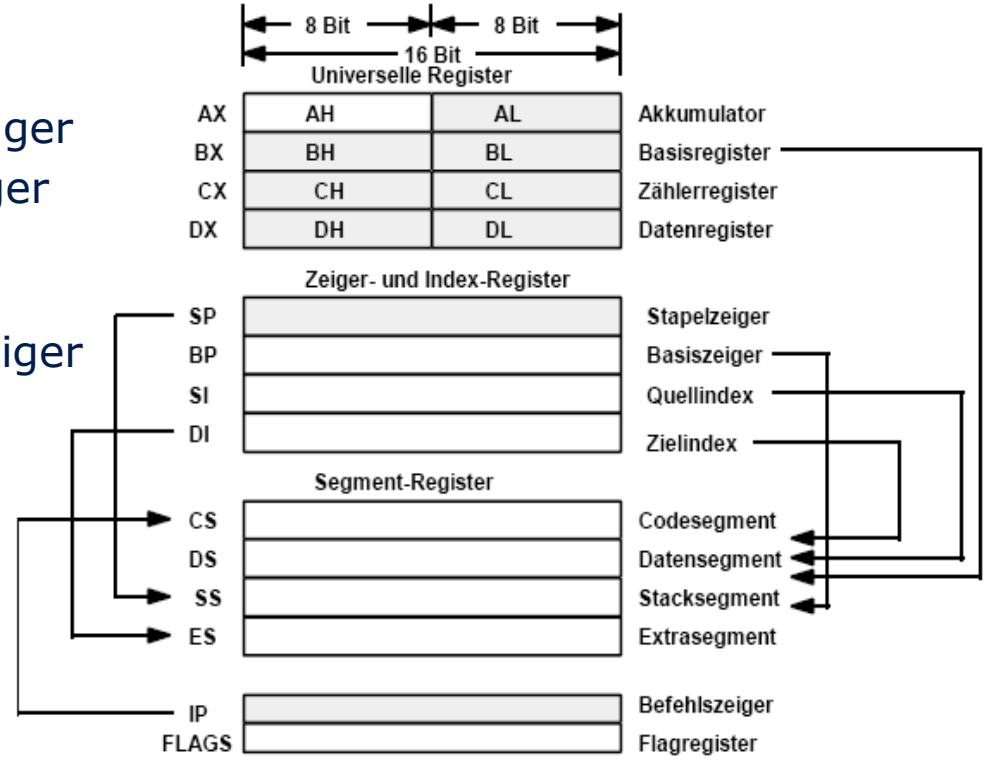
8086 Adressaddierwerk

- 16 bit Register, aber 1 MB Adressraum → Zerlegung in Segmente (16 bit = 65 kByte)
- Adresse = Segmentadresse * 16 + Offsetadresse
- Berechnung automatisch in BIU
- Segmente überlappen!
- Registerkombinationen nicht frei wählbar



Registerpaarung für die Adressberechnung

- Stapelspeicher
  - Stacksegment + Stapelzeiger
  - Stacksegment + Basiszeiger
- Befehle
  - Codesegment + Befehlszeiger
- Daten
  - Datensegment + diverse Zeiger und Indexregister
  - Extrasegment + Zielindex



Angabe von Operanden – Konstanten, Register, Speicher

Angabe von Operanden

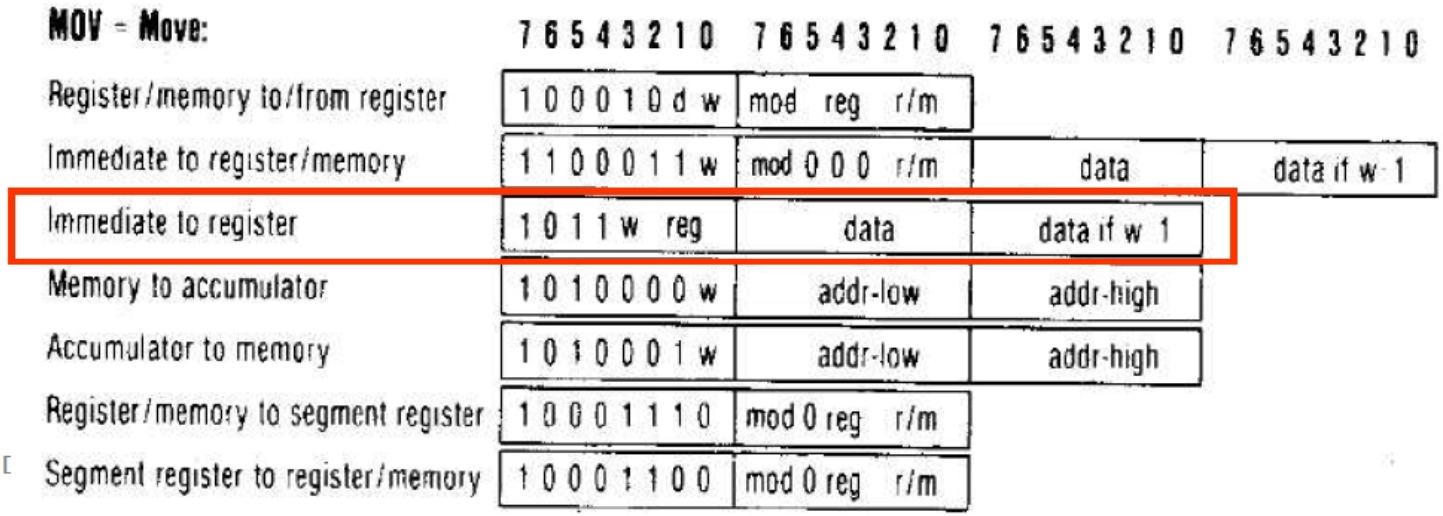
- Ohne Zugriff auf Speicher
  - Immediate (Konstante)
  - Register Immediate (Inhalt eines Registers)
- Mit Zugriff auf Speicher
  - Direct Mode (feste Speicherstelle)
  - Register Indirect Mode (Speicherstelle ist in Register angegeben)
- Mit indiziertem Zugriff auf Speicher
  - Base Mode
  - Base-Indexed Mode
  - Base-Indexed Mode with Displacement

Immediate (Direktwert)

- Belege ein Register oder eine Speicherstelle mit einer Konstanten
- Adressierungsarten (Ziel/Quelle) und Direktwert (8/16 Bit) werden im Maschinenbefehl kodiert
- Beispiel: Belege das Register AH mit dem Wert 76 (= 4Ch)
  - Assembler: MOV AH, 4Ch
  - Maschinencode: B4 4C

Kurzer Exkurs: Maschinencode

- Zur Erinnerung: Programme sind auch nur Daten, die vom Steuerwerk interpretiert werden
- Belege Register AH (reg=0) direkt mit Bytewert 4Ch (w=0, data=4C) to --> 10110000 01001100



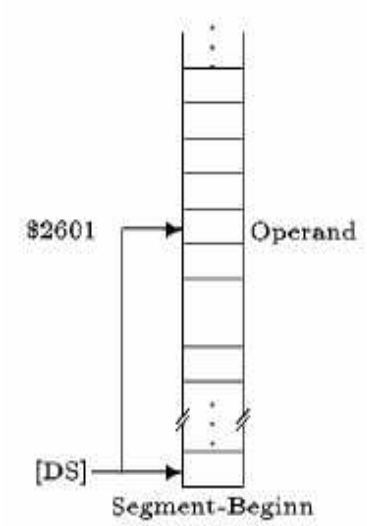
MOV = Move:	76543210	76543210	76543210	76543210
Register/memory to/from register	100010dw	mod reg r/m		
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w=1
Immediate to register	1011w reg	data	data if w=1	
Memory to accumulator	1010000w	addr-low	addr-high	
Accumulator to memory	1010001w	addr-low	addr-high	
Register/memory to segment register	10001110	mod 0 reg r/m		
Segment register to register/memory	10001100	mod 0 reg r/m		

Register Immediate (Direktbelegung Register)

- Zugriff auf den Inhalt eines Registers
- Art der Adressierung (register-register) und Auswahl der Register wird im Befehl kodiert
- Beispiel: Belege das Register BX mit dem in Register AX gespeicherten Wert
  - Assembler: MOV BX, AX
  - Maschinencode: 8E D8

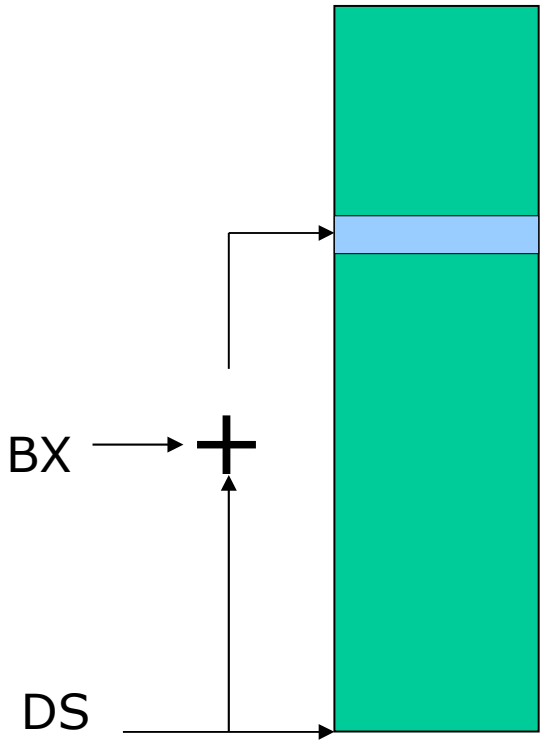
Direct Mode (Feste Speicheradresse)

- Zugriff auf den Inhalt einer festen Speicherstelle
- Wenn nicht anders angegeben, wird das Datensegmentregister (DS) zur Adressberechnung verwendet
- Beispiel: Belege Register AX mit dem Wert an Speicherstelle 2601h
  - Assembler: MOV AX, [2601h]
  - Maschinencode: 8B 1E 26 01



Base Mode (Indirekte Registeradressierung)

- Zugriff auf den Inhalt einer Speicherstelle, deren Adresse in einem Register liegt
  - Default für Segmentregister abhängig von Register
- Beispiel: Belege AX mit dem Wert der an der Speicherstelle liegt, die in BX angegeben ist
  - Assembler: MOV AX, [BX]

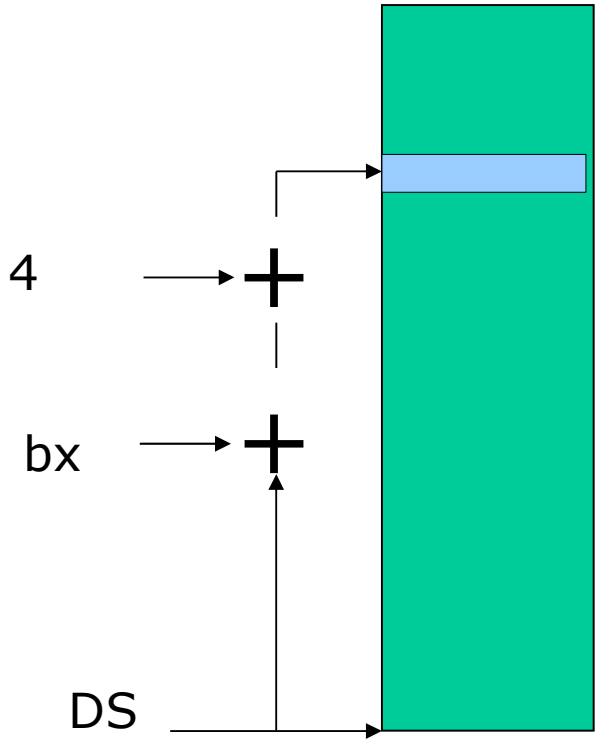


Base-Indexed-Mode with Displacement

Zugriff auf die Adresse, deren Offset sich aus dem Inhalt des Basisregisters plus einer Konstanten ergibt.

Beispiel

```
mov ax, [bx+4]
```



16-bit protected mode (ab 286)

- Virtuelle Speicherverwaltung
  - Segment = Selektor (Index in eine Deskriptortabelle)
  - **Keine festgelegte Position im Speicher mehr**
  - Effiziente Auslagerung von Speicher auf die Festplatte
  - Vergabe von Zugriffsrechten (read-only)
- Problem
  - Register nach wie vor 16 bit, d.h. Segmentgröße immer noch auf 64 kByte beschränkt

32-bit protected Mode (ab 386)

- Viele Register sind nun 32 Bit breit
  - EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
  - Segmentregister nach wie vor 16 Bit, neu FS, GS
- 32 bit Adressregister → Segmentgröße 4 Gbyte
  - **Flaches Programmiermodell, Segment und Segmentgröße muss normalerweise nicht beachtet werden**
- Aber: Unterteilung von Segmenten in Seiten (Page, 4 Kbyte) möglich
  - Nur Teile eines Segments müssen im Speicher sein.
  - Notwendige Voraussetzung für Mehraufgabensysteme wie Windows NT/2000/XP, OS/2, MacOS und Linux

32-Bit Register 386 ff.

E = Extendend = 32 Bit

- EAX, EBX, ECX, EDX
  - Allgemeine 32 bit Register, aber:
  - EAX: Berechnungen
  - EBX: Zeiger (Pointer)
  - ECX: Schleifen
  - EDX: Mult/Div, 64 bit mit EAX
  - 16 and 8 bit Teile (286, 8088)
- ESI, EDI: String Operationen
- EBP, ESP: Base/Stack Pointer
- CS-GS: Segmentierter Speicher
- EIP: Instruction Counter (=PC)
- EFLAGS: Prozessorstatuswort