

## 4.4 RISC & CISC

---

**CISC**

**Complex Instruction Set Computers**

**RISC**

**Reduced Instruction Set Computers**



# Programmiermodell der Intel 80x86

Die Allzweckregister können als ..  
16-Bit oder 8-Bit Register verwendet werden. Sie nennen sich dann

16-Bit AX - DX  
8-Bit AH - DH und AL - DL

Die Adreßregister besitzen immer eine Länge von 16-Bit

## Adreß-Segmentregister

15 8 7 0

**DS** Datensegment-Register

**ES** Extrasegment-Register

**SS** Stcksegment-Register

**CS** Codesegment-Register

## Allzweckregister

Bit 15 8 7 0

Akkumulator

**AX**

AH

AL

Basisregister

**BX**

BH

BL

Counter - Zähler

**CX**

CH

CL

Daten- und Torregister

**DX**

DH

DL

## Adreß- Offsetregister

Bit 15 8 7 0

**SI** Sourceindex - Quellenregister

**DI** Destinationindex - Zielregister

**BP** Basepointer - Basiszeiger

**SP** Stackpointer - Stapelzeiger

**IP** Instructionptr - Befehlszeiger



# CISC & RISC

---

Zwei Techniken zur Implementierung von Befehlen im Rechner

➤ **Direkt durch Hardware**

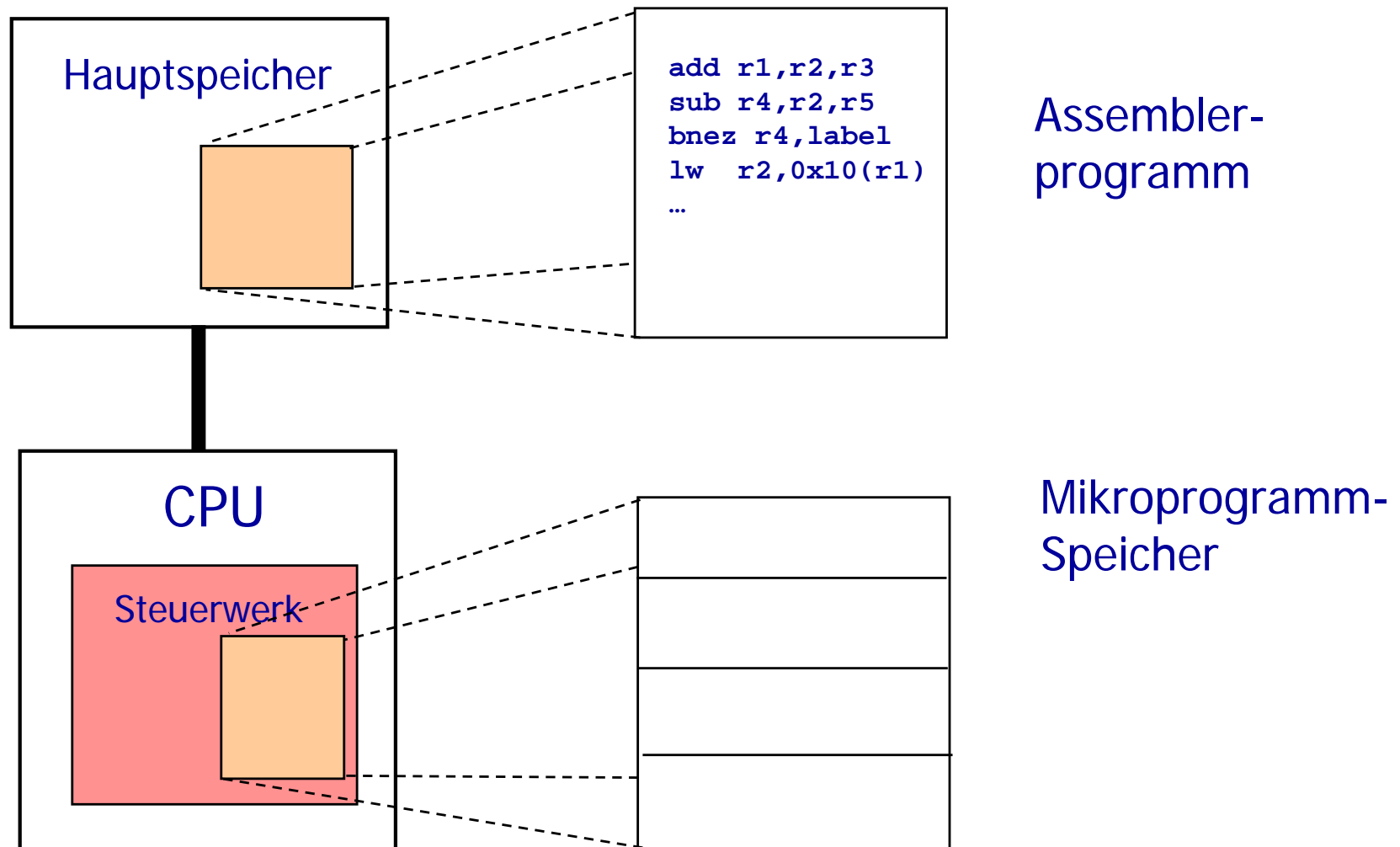
- aufwendige Schaltnetze und Schaltwerke bei großer Anzahl von Befehlen (200-300)

➤ **Mikroprogramme als Befehlsinterpreter**

- Mikroprogrammspeicher im Steuerwerk, der neu geladen werden kann
- verschiedene Befehlssätze können implementiert werden



# Prinzip der Mikroprogrammierung



# Vorteile und Nachteile

---

## ➤ Vorteile der Mikroprogrammierung:

- Mehrere Befehlssätze auf einem Rechner → Anpassung des Befehlssatz an der Anwendung
- Mehrere Rechnertypen mit dem gleichen Befehlssatz

## ➤ Nachteile der Mikroprogrammierung:

- Aufwendig
- Langsam



# CISC (complex instruction set computers)

---

## Gründe dafür:

- Ausführung komplizierter Befehle ist immer noch schneller als die Ausführung von Programmen gleicher Funktion
- Mikroprogrammierung begünstigt komplizierte Befehle
- komplizierte Befehle führen zu kurzen Programmen
- Umfang des Befehlssatzes wird oft als Werbeargument verwendet
- Unterstützung höherer Programmiersprachen durch komplizierte Befehle (Direkte Abbildung: *Sprachkonstrukt* ➔ *Befehl*)



# CISC (complex instruction set computers)

---

## Gründe dafür:

- Unterstützung von Compilern durch entsprechende Befehle
- Unterstützung spezieller Einsatzgebiete

## Fazit:

**Entwicklung von Hardware,  
Programmiersprachen und Einsatzgebieten  
begünstigt „komplizierte“ Befehle.**



# CISC (complex instruction set computers)

---

## Gründe dagegen:

- Schnellere Hauptspeicher und die Verwendung von Cache-Speichern beschleunigen die Programmausführung
- Mikroprogramme wurden immer umfangreicher; Verlängerte Entwurfszeit, Komplexe Steuerwerke (> 50% der Chipfläche)
- Nur relativ kleine Teile des großen Befehlssatzes werden häufig benutzt
- Größere Fehlerhäufigkeit auf der Mikroprogrammebene
- Schwieriger Compilerbau



# CISC (complex instruction set computers)

---

- **Systemprogramme in XPL auf IBM/360:**

- 90 % aller ausgeführten Befehle: 10 verschiedene Befehle

- 95 % aller ausgeführten Befehle: 21 verschiedene Befehle

- 99 % aller ausgeführten Befehle: 30 verschiedene Befehle

- **COBOL-Programme auf IBM/370:**

- 90,28 % aller ausgeführten Befehle: 26 verschiedene Befehle

- 99,08 % aller ausgeführten Befehle: 48 verschiedene Befehle

- (nur 84 verschiedene Befehle wurden überhaupt benutzt)*



# Limitationen der CISC Architekturen

---

- **Befehlsausnutzung (80/20 Regel):**

viele mächtige Befehle, komplexes Befehlsformat, Mikroprogrammierung, nur 20 % der Befehle werden überwiegend benutzt

- **Kritisches Problem: Anzahl der Zyklen pro Instruktion (CPI)**

bei allen heutigen CISC Architekturen ist  $CPI \gg 2$



# Prozentualer Anteil von Anweisungen in Hochsprachenprogrammen

Großteil der in Hochsprachen verwendeten Anweisungen ist sehr einfach:

Anweisung	Mittlerer zeitlicher Anteil
Zuweisung	47 %
if	23%
call	15 %
loop	6 %
goto	3 %
Andere	7 %



# RISC (reduced instruction set computers)

---

## Grundprinzipien:

- Viel benutzte einfache Befehle so schnell wie möglich machen (Ausführung möglichst in einer Taktphase. Keine Mikroprogrammierung mehr, Befehls-Pipeline)
- Der größte Teil der Arbeit soll durch optimierende Compiler zur Übersetzungszeit erledigt werden
- Operanden werden nach Möglichkeit in großen Registersätzen gehalten → schneller Zugriff → schnelle Verarbeitung
- Einheitliche Befehlsformate → schnelle Decodierung
- Pipelining anwenden, so gut es geht



# RISC (reduced instruction set computers)

---

## Entwurfsziele:

- Ausführung jedes Befehls in einem Taktzyklus  
(*Befehl  $\approx$  bisheriger Mikrobefehl bei CISC*)
- Alle Befehle gleich lang:  
Decodierschaltung wird einfacher  
Programme länger, aber Ausführungszeit kürzer
- Nur Load-Store und Register-Register-Befehle:  
weniger Adressierungsarten → schnelle Ausführung
- Koprozessorarchitektur für komplexe Befehle



# RISC (reduced instruction set computers)

---

## **Keine Entwurfsziele sind z. B.:**

- Unterstützung von Gleitkomma-Arithmetik
- Unterstützung von Betriebssystemfunktionen



# Zielvorstellungen für RISC Rechner

---

- Ein-Zyklus-Befehle
- Einheitliches Befehlsformat
- Wenige Maschinenbefehle
- Load/Store-Architektur
- Großer Registersatz
- Verzicht auf Mikroprogrammierung
- 32-Bit-Architektur
- Pipeline-gerechter Maschinenbefehlssatz (gleiche Befehlsausführzeiten)
- Keine Unterstützung für Betriebssystem und Gleitkomma-Arithmetik



# Forderungen an RISC-Systeme

---

- Mindestens 75% aller Befehle sind Ein-Zyklus-Befehle
- Einheitliche Länge aller Befehle entsprechend der Datenbusbreite
- Nicht mehr als 128 Befehle
- Nicht mehr als 4 Befehlsformate
- Nicht mehr als 4 Adressierungsarten
- Load/Store-Architektur
- Festverdrahtete Steuereinheit, keine Mikroprogrammierung
- Mindestens 32 allgemein verwendbare Register



# RISC Rechner aus heutiger Sicht

---

**Geblieden ist von der RISC-Idee im wesentlichen:**

- das Befehlspipelining
- die Load/Store-Architektur
- ein großer Registersatz: 32 allgemeine und 32 Gleitpunkt-Register
- ein einheitliches Befehlsformat
- die Verwendung weniger Adressierungsarten
- der Verzicht auf Mikroprogrammierung



# RISC & CISC

CISC	RISC
Komplexe Befehle, Ausführung in mehreren Taktzyklen	Einfache Befehle, Ausführung in einem Taktzyklen
Jeder Befehl kann auf den Speicher zugreifen	Nur Lade- und Speicherbefehle greifen auf den Speicher zu
Wenig Pipelining	Intensives Pipelining
Befehle werden von einem Mikroprogramm interpretiert	Befehle werden durch festverdrahtete Hardware ausgeführt
Befehlsformat variabler Länge	Alle Befehle mit fester Länge
Die Komplexität liegt im Mikroprogramm	Die Komplexität liegt im Compiler
Einfacher Registersatz	Mehrere Registersätze



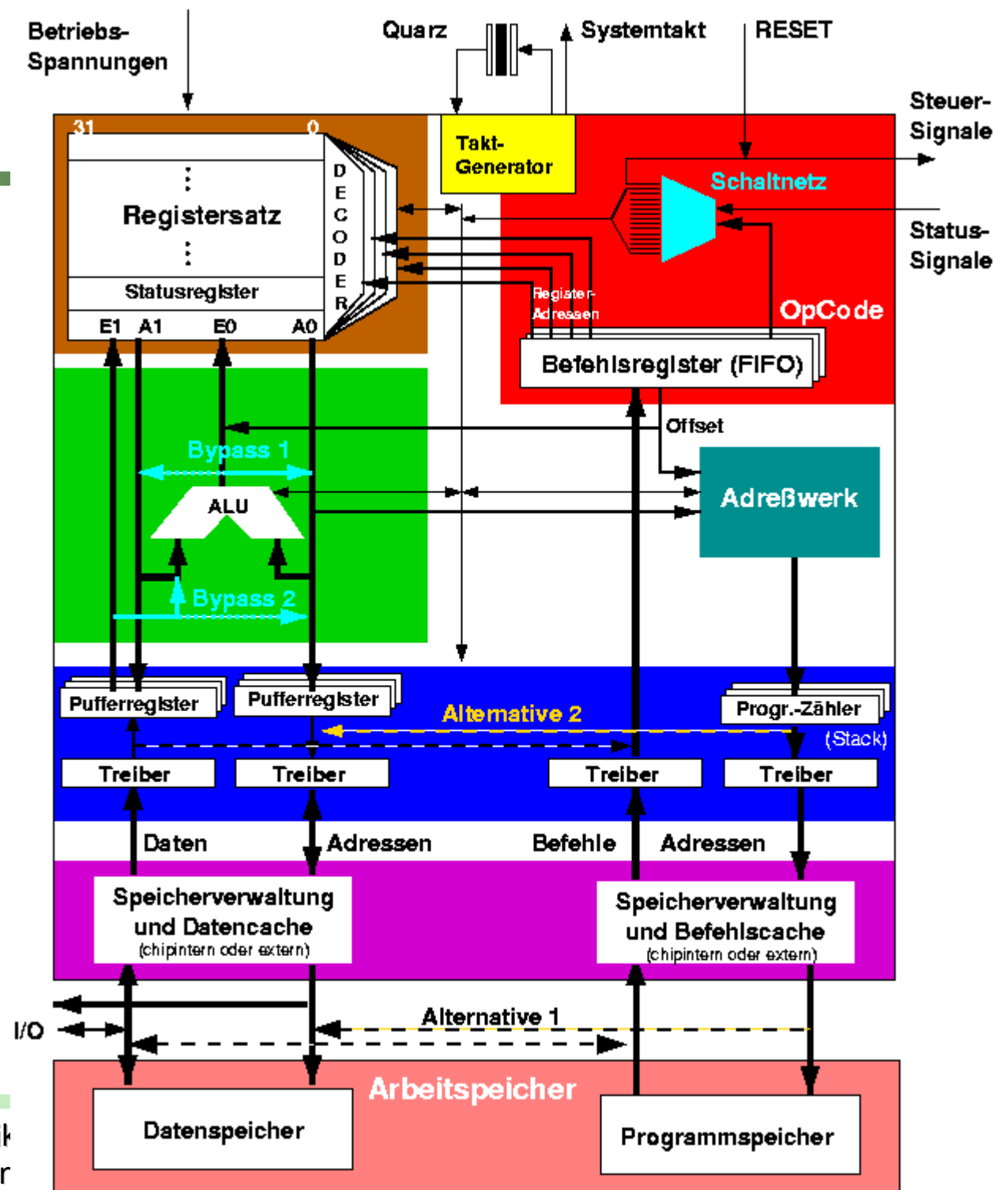
# Vergleich CISC & RISC

Vergleich von drei typischen CISC-Rechnern mit den ersten drei RISC-Rechnern [Tanenbaum]:

	CISC			RISC		
	IBM 370/168	VAX 11/780	Xerox Dorado	IBM 801	Berkeley RISC I	Stanford MIPS
Fertigstellungsjahr	1973	1978	1978	1980	1981	1983
Instruktionen	208	303	270	120	<b>31</b>	55
Mikrocodegröße	54k	61k	17k	0	0	0
Instruktionsgröße	2-6 Bytes	2-57 Bytes	1-3 Bytes	4 Bytes	4 Bytes	4 Bytes
Operationsmodell	Reg-Reg Reg-Mem Mem-Mem	Reg-Reg Reg-Mem Mem-Mem	Stack	Reg-Reg	Reg-Reg	Reg-Reg



# Aufbau eines RISC Prozessors



# Aufbau eines RISC-Prozessors

---

## Havard Architektur:

getrennter Programm- und Datenspeicher, deshalb  
zwei Adress- und Datenbusse

→ paralleles Holen von Operanden und Instruktionen

## Vereinfachende Varianten:

1. zwei getrennte Bussysteme bis zu den Cache-Speichern, jedoch nur ein Arbeitsspeicher (niedrigere Kosten)
2. nur ein Bussystem wie bei Standard-Mikroprozessoren



# Aufbau eines RISC-Prozessors

---

## **Systembusschnittstelle:**

enthält Registerblocks sowohl für Daten als auch für Adressen (gleichzeitiges Lesen eines Datums und Zwischenspeichern eines Ergebnisses)

## **Befehlszähler:**

ist manchmal als Hardware-Stack ausgebildet (beschleunigt Unterprogrammaufrufe)



# Aufbau eines RISC-Prozessors

---

## Steuerwerk:

- festverdrahtet
- Das Befehlsregister als Warteschlange (FIFO) realisiert
- Für jede Pipeline-Stufe ist dort ein Register vorhanden
- Die OpCodes jeder Stufe können vom Schaltnetz des Steuerwerks ausgewertet werden

## Registersatz:

- besteht aus einer großen Zahl von Registern
- erlaubt gleichzeitige Auswahl von 3 bis 4 Registern  
(z. B. 4 Port Registersatz, gleichzeitiges Schreiben (E0, E1) und Lesen (A0, A1) von jeweils 2 Registern)



# Aufbau eines RISC-Prozessors

---

## Rechenwerk:

- Besitzt eine Load/Store-Architektur.
- Die Operanden werden über 2 Operandenbusse aus dem Registersatz herbeigeführt, das Ergebnis (noch im selben Taktzyklus) über den Ergebnisbus in den Registersatz geschrieben.
- Normalerweise gibt es keine direkte Verbindung zwischen ALU und Systemdatenbus
- Datentransfer läuft über die Register (Load/Store-Architektur)



# Befehlsverarbeitung in RISC-Prozessoren

---

## Einfacher Befehlssatz von RISC-Prozessoren

→ Maschinenprogramme sind länger als bei CISC Prozessoren

*(komplexe Befehle und Adressierungsarten müssen aus den einfachen RISC Befehlen zusammengesetzt werden)*

Trotzdem arbeitet ein RISC-Prozessor meist schneller als ein CISC-Prozessor. Der Grund liegt in der nahezu **vollständigen Parallelarbeit aller Komponenten** eines RISC-Prozessors (extreme Pipeline-Verarbeitung)

Es wird mit großer Wahrscheinlichkeit **in jedem Taktzyklus ein Befehl** beendet



# RISC - superskalar

---

- ❑ RISC-Prozessoren, die das Entwurfsziel von durchschnittlich einer Befehlsausführung pro Takt (CPI – *cycles per instruction* oder IPC – *instructions per cycle* von eins) erreichen, werden als **skalare RISC-Prozessoren** bezeichnet.
- ❑ Die Superskalar-Technik ermöglicht es, pro Takt mehrere Befehle den Ausführungseinheiten zuzuordnen und eine gleiche Anzahl von Befehlsausführungen pro Takt zu beenden.
- ❑ Solche Prozessoren werden als **superskalare (RISC)-Prozessoren** bezeichnet, da die oben definierten RISC-Charakteristika auch heute noch weitgehend beibehalten werden.
- ❑ Heutige Mikroprozessoren nutzen Befehlsebenenparallelität durch die Pipelining- und Superskalartechnik.



# Kapitel 5

---

## Pipeline-Verarbeitung



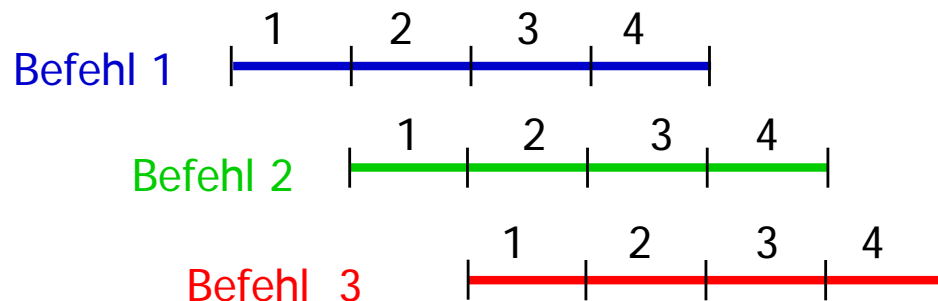
# 5. 1 Pipeline-Verarbeitung

Ausführung von 3 gleichartigen Verarbeitungsaufträgen in 4 Teilverarbeitungsschritten:

## Serielle Verarbeitung:



## Pipeline-Verarbeitung:



# Pipelining „Fließband-~~B~~earbeitung“

---

„Pipelines beschleunigen die Ausführungsgeschwindigkeit eines Rechners in gleicher Weise wie Henry Ford die Autoproduktion mit der Einführung des Fließbandes revolutionierte.“

(Peter Wayner 1992)



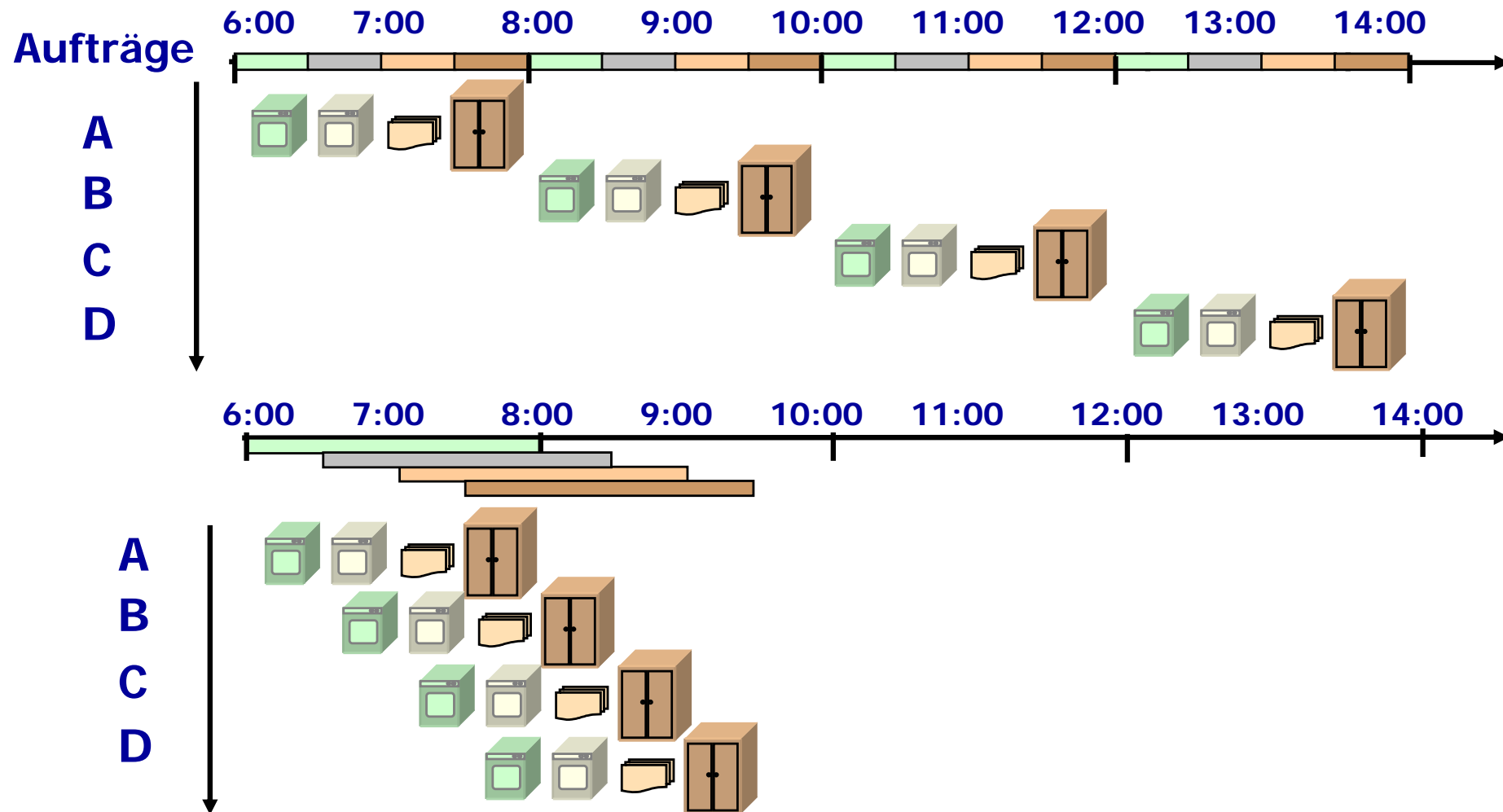
# Wäsche Pipelining

---

- Ein Wäsche-Vorgang kann in 4 Teilvorgänge unterteilt werden:
  - Schmutzige Wäsche in die Waschmaschine
  - Nasse Wäsche in den Trockner
  - Falten, Bügeln, ...
  - Kleider in den Schrank



# Wäsche Pipelining



# Pipeline Verarbeitung

---

Oft ablaufende Operationen werden in eine Folge von Teilprozessen zerlegt. Für jeden Teilprozess wird ein spezieller Prozessor (spezielle Ausführungseinheit) vorgesehen.

**Beispiel:** Befehlsverarbeitung wird aufgeteilt in:

- Befehl holen
- Befehl decodieren (interpretieren) und
- Operand(en) holen
- Operation ausführen
- Ergebnis speichern



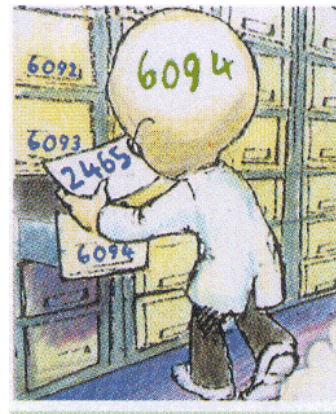
# Beispiel



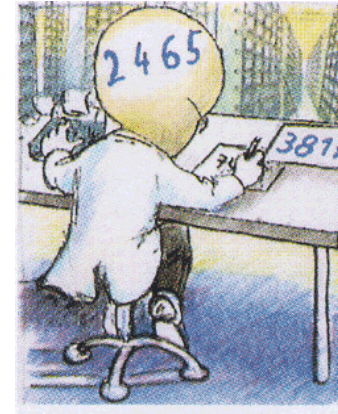
**Befehl  
bereitstellen**



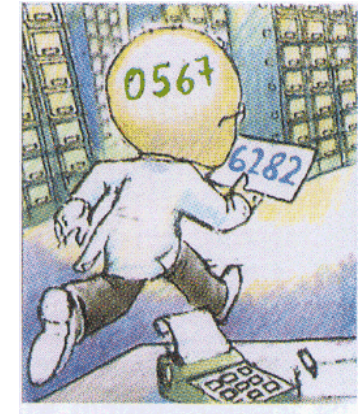
**Befehl  
dekodieren**



**Operanden  
holen**

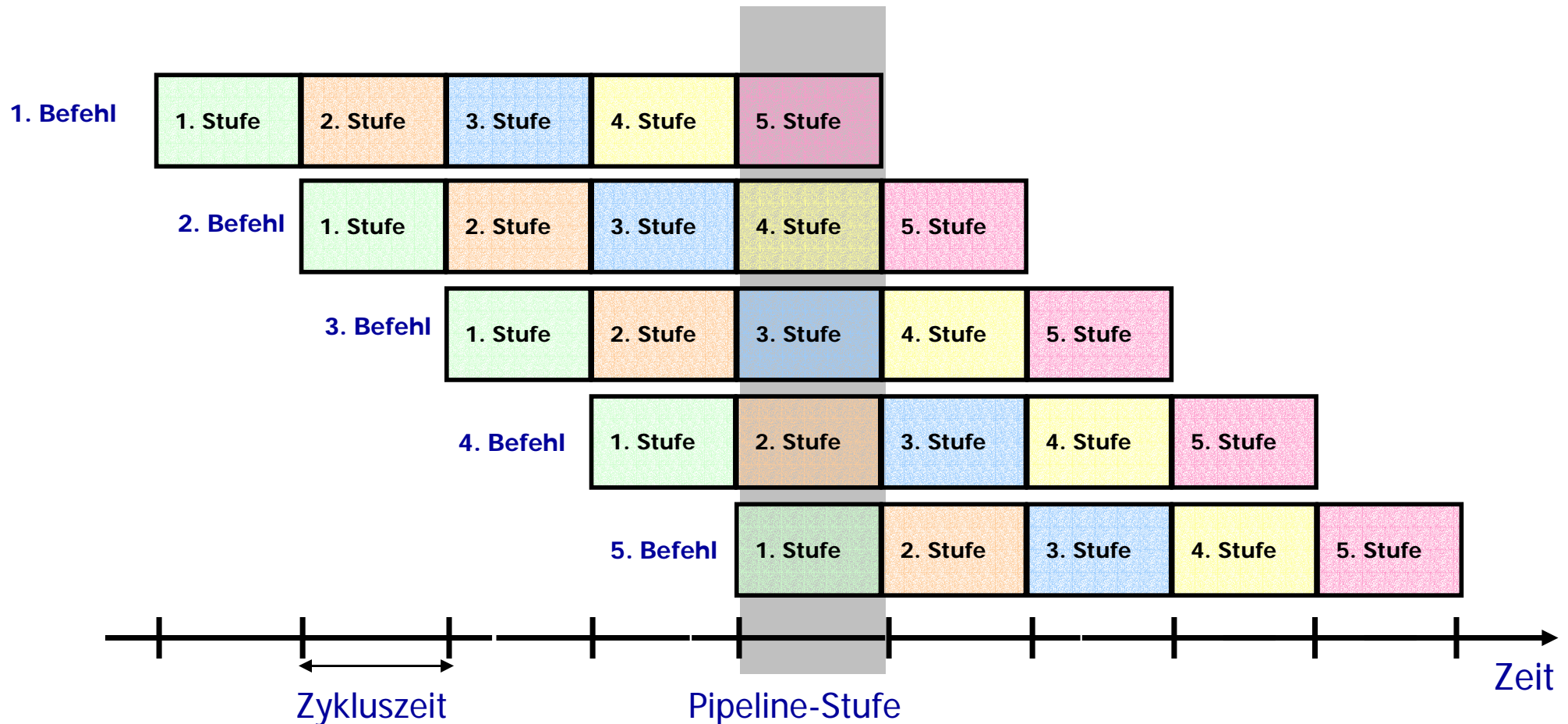


**Operation  
ausführen**



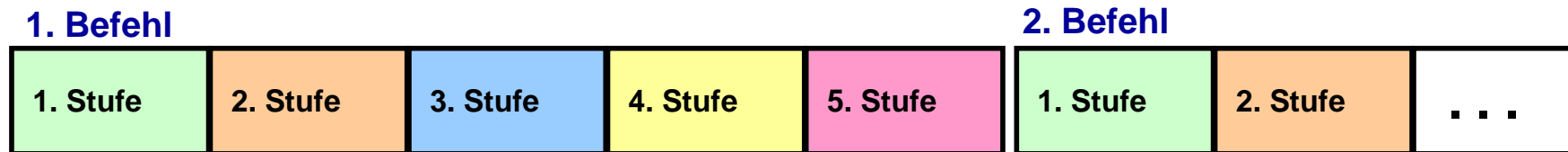
**Ergebnis  
speichern**

# Einfache fünfstufige Befehlspipeline

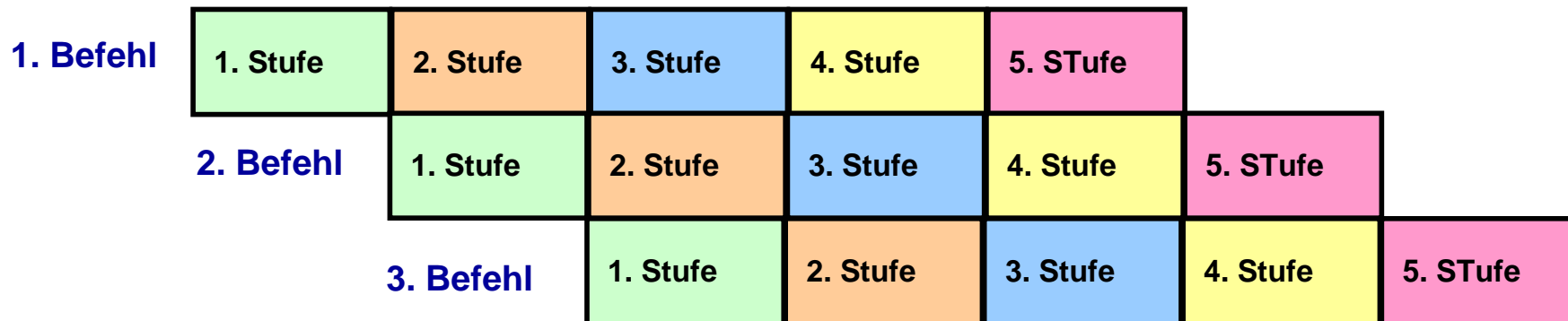


# Pipelining

## Sequentielle Ausführung:



## Pipelining:



# Definitionen

---

- ❑ **Pipelining:** Zerlegung einer Maschinenoperation in mehrere Phasen oder Suboperationen, die dann von hintereinander geschalteten Verarbeitungseinheiten **taktsynchron** bearbeitet werden, wobei jede Verarbeitungseinheit genau eine spezielle Teiloperation ausführt
- ❑ Die Gesamtheit dieser Verarbeitungseinheiten nennt man eine **Pipeline**.
- ❑ Bei einer **Befehlspipeline** (Instruction Pipeline) wird die Ausführung eines Maschinenbefehls in verschiedene Phasen unterteilt, aufeinanderfolgende Maschinenbefehle werden jeweils um einen Taktzyklus versetzt ausgeführt

