

# 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



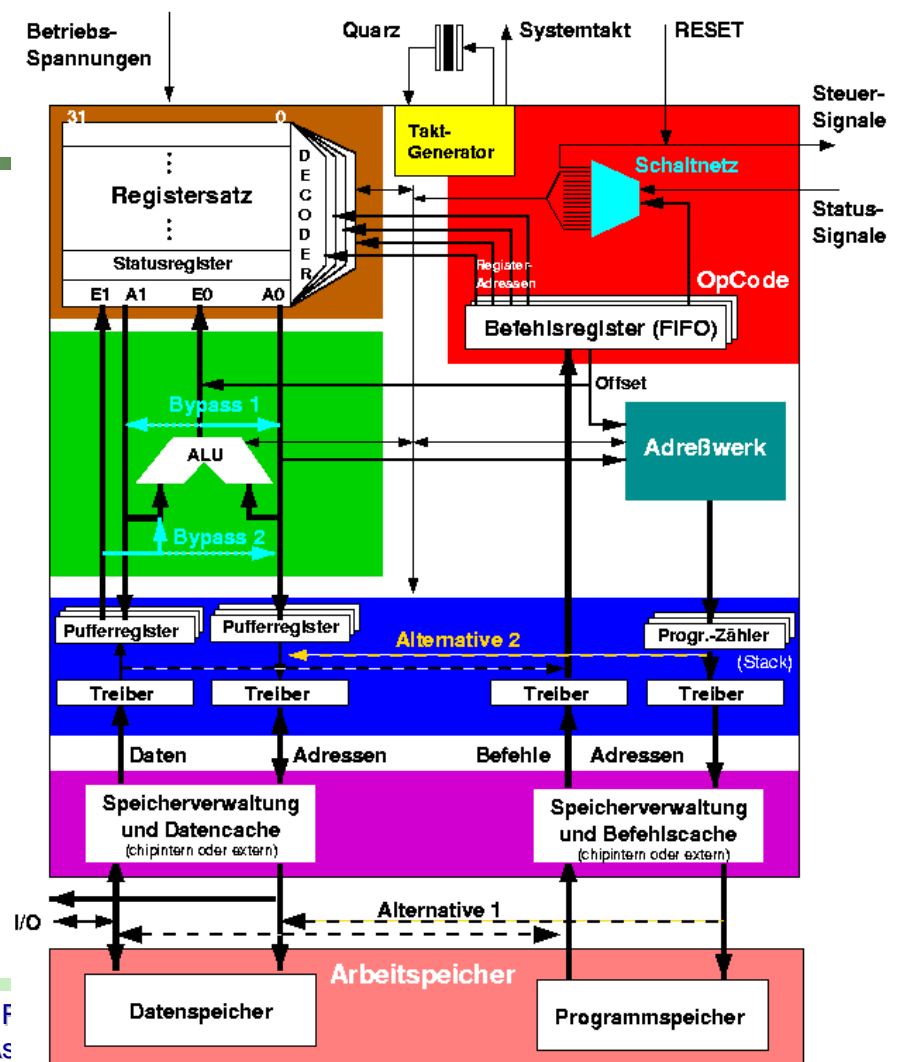
## Vergleich CISC & RISC

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

	IBM 370/168	CISC VAX 11/780	Xerox Dorado	IBM 801	RISC 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



Institut für Rechnerentwurf und F  
Prof. Dr. W. Karl & Dr.-Ing. T. As

## 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

---

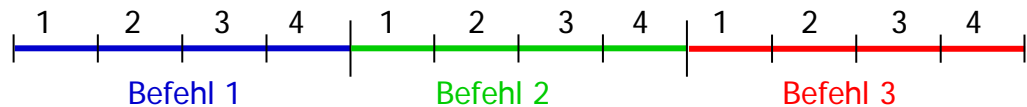
# Pipeline-Verarbeitung



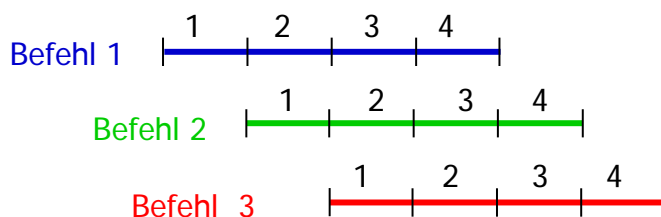
# 5. 1 Pipeline-Verarbeitung

Ausführung von 3 gleichartigen Verarbeitungsaufträgen  
in 4 Teilverarbeitungsschritten:

## Serielle Verarbeitung:



## Pipeline-Verarbeitung:



## Pipelining „Fließband-Verarbeitung“

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

(Peter Wayner 1992)



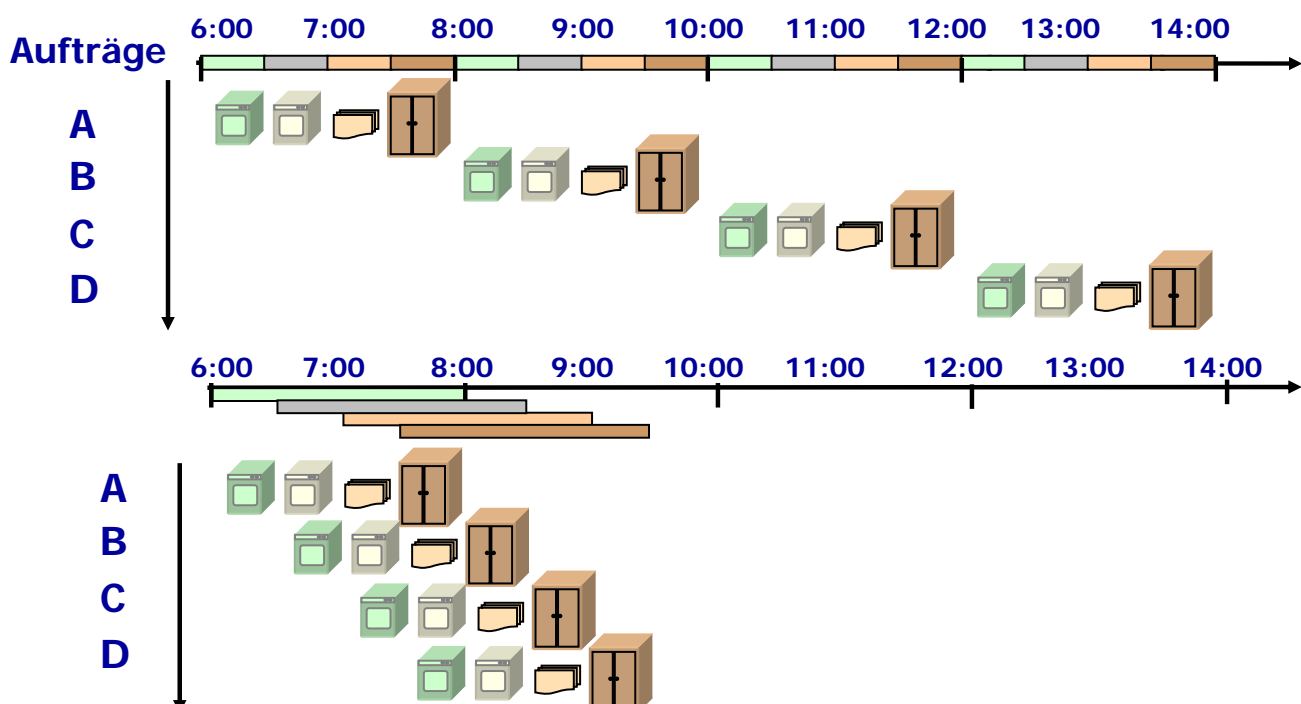
# Wäsche Pipelining

□ 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



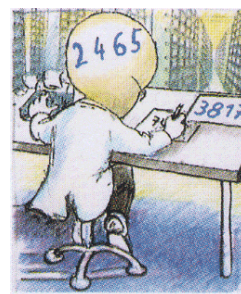
**Befehl  
bereitstellen**



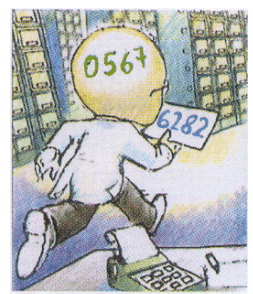
**Befehl  
dekodieren**



**Operanden  
holen**



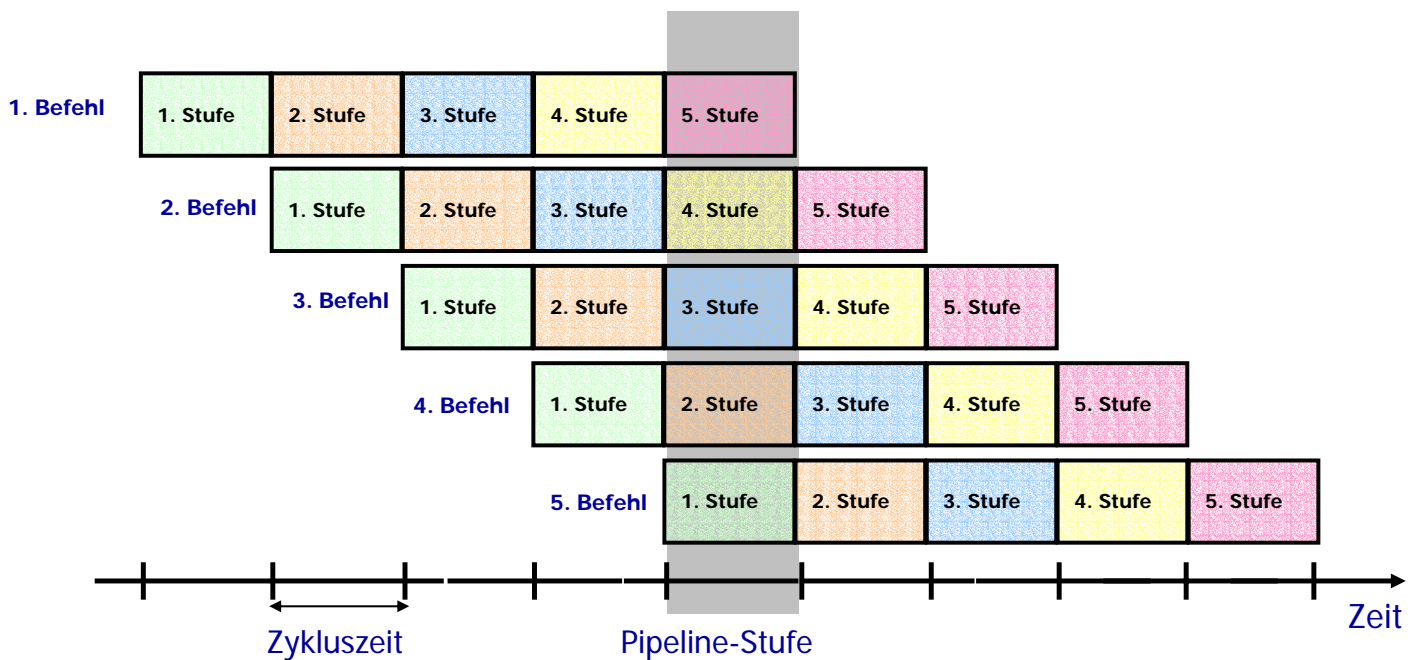
**Operation  
ausführen**



**Ergebnis  
speichern**

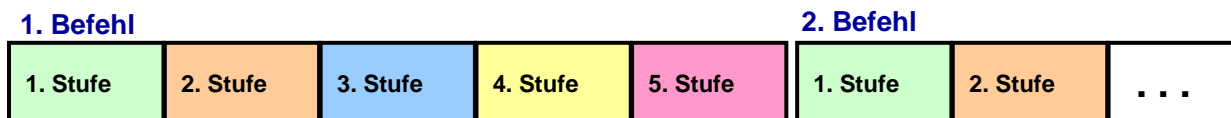


# Einfache fünfstufige Befehlspipeline

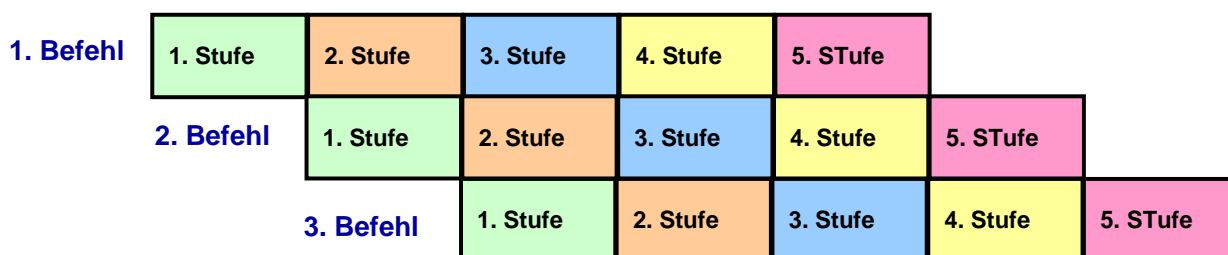


## Pipelining

### Sequentielle Ausführung:



### Pipelining:

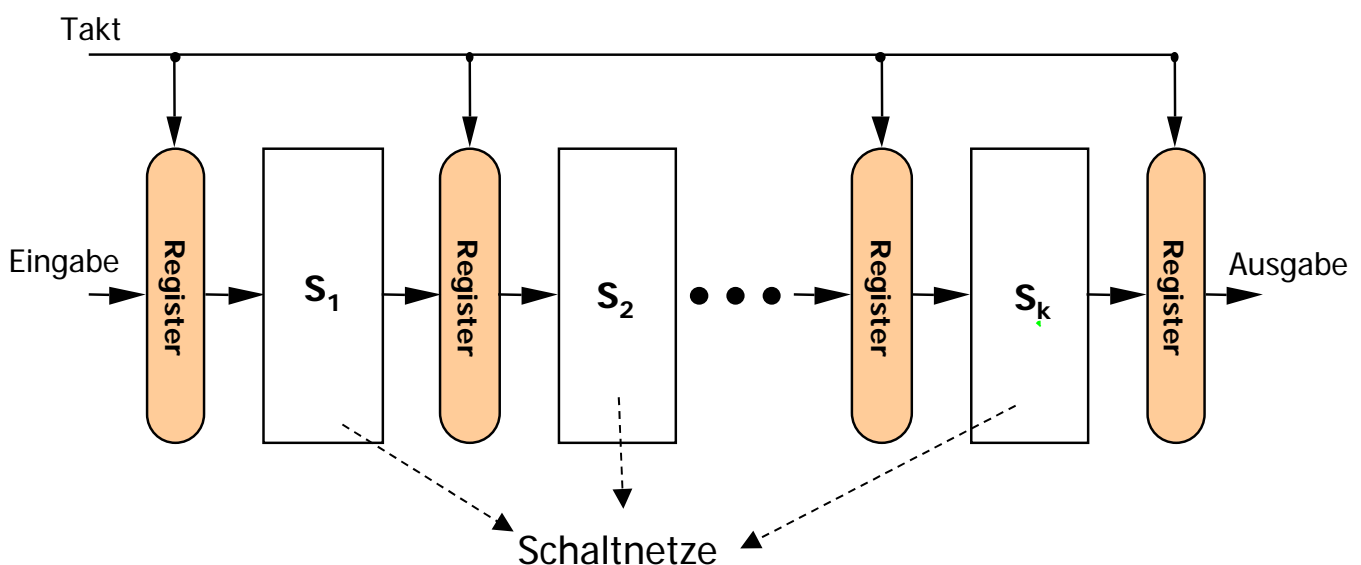


# Definitionen

- ❑ **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



## 5.2 Pipeline-Stufen und Pipeline-Register



Verzögerungszeiten:

➤ der Schaltnetze:  $\tau_i$  ( $i = 1, \dots, k$ )

➤ der Pipeline-Register:  $\tau_{reg}$

Länge eines Taktzyklus:

$$\tau = \max\{\tau_1, \tau_2, \dots, \tau_k\} + \tau_{reg}$$



# Definitionen

---

- ❑ Jede Stufe der Pipeline heißt **Pipeline-Stufe** oder **Pipeline-Segment**.
- ❑ Pipeline-Stufen werden durch getaktete **Pipeline-Register** (auch *latches* genannt) getrennt.
- ❑ Ein Pipeline-Maschinentakt ist die Zeit, die benötigt wird, um einen Befehl eine Stufe weiter durch die Pipeline zu schieben.
- ❑ Idealerweise wird ein Befehl in einer **k-stufigen Pipeline** in **k** Takten von **k** Stufen ausgeführt.
- ❑ Wird in jedem Takt ein neuer Befehl geladen, dann werden zu jedem Zeitpunkt unter idealen Bedingungen **k** Befehle gleichzeitig behandelt und jeder Befehl benötigt **k** Takte, bis zum Verlassen der Pipeline.



# Definitionen

---

- ❑ **Latenz:** die Zeit, die ein Befehl benötigt, um alle **k** Pipeline-Stufen zu durchlaufen.

## Ideale Verhältnisse:

- Ausführung eines Befehls in **k** Takten.
- Es werden gleichzeitig **k** Befehle bearbeitet.

- ❑ **Durchsatz einer Pipeline:** Anzahl der Befehle, die eine Pipeline pro Takt verlassen können. Dieser Wert spiegelt die Rechenleistung einer Pipeline wider.



# Leistungssteigerung durch Pipelining

## $n$ Befehle in einer Pipeline mit $k$ Stufen

- Hypothetischen Prozessor ohne Pipeline:  
 $n * k$  Taktzyklen
- Pipeline-Prozessor mit einer  $k$ -stufigen Pipeline:  
 $k + (n-1)$  Taktzyklen (unter der Annahme idealer Bedingungen mit einer Latenz von  $k$  Takten und einem Durchsatz von 1)
  - $k$  Taktzyklen, um die Pipeline zu füllen
  - $(n-1)$  Taktzyklen, um die restlichen  $(n-1)$  Befehle auszuführen.

➔ Leistungssteigerung  $S$  (*speedup*):

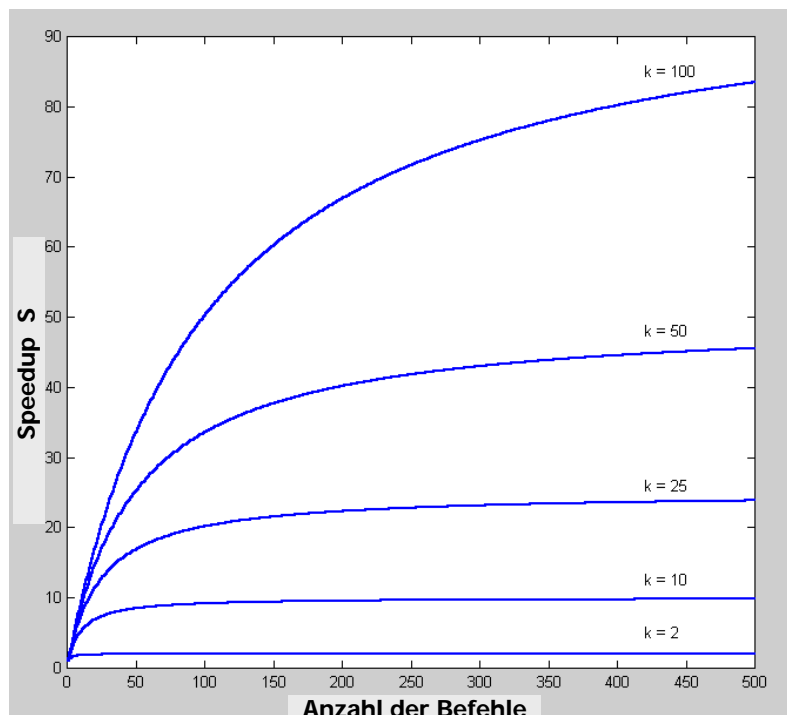
$$S = \frac{n * k}{k + (n-1)}$$



# Leistungssteigerung durch Pipelining

$$S = \frac{n * k}{k + (n-1)}$$

$$\lim_{n \rightarrow \infty} S = k$$

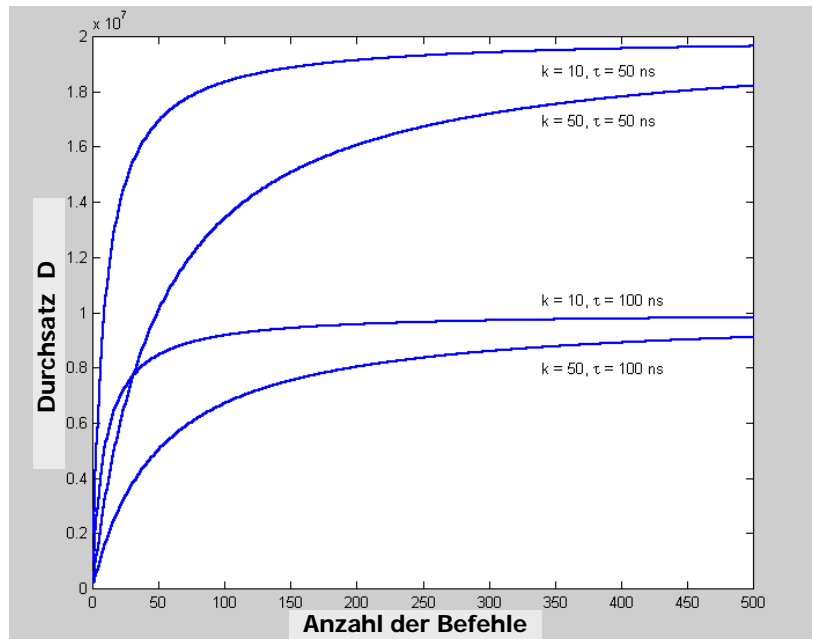


# Durchsatz

- Der **Durchsatz** gibt an, wie viele Befehle in einem Zeitraum  $T_k * \tau$  ausgeführt werden.

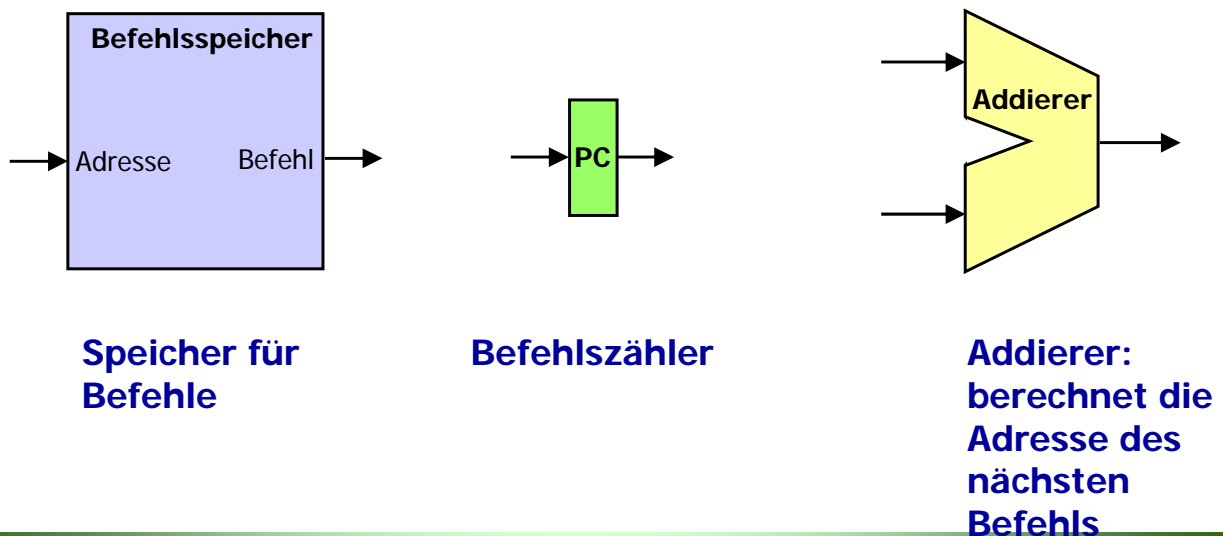
$$D = \frac{n}{T_k * \tau}$$
$$= \frac{n}{(k + (n-1)) * \tau}$$

$$\lim_{n \rightarrow \infty} D = \frac{1}{\tau} = D_{max}$$



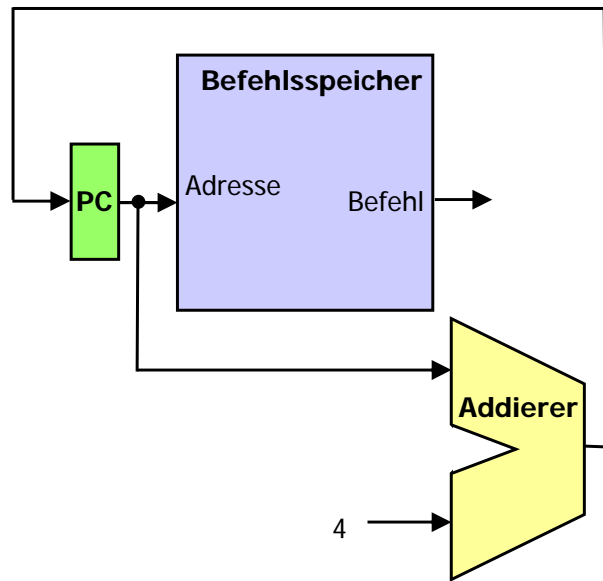
## 5. 3 Befehlsabarbeitung und Datenpfade der MIPS-Befehle

- Welche Hardware-Komponenten sind zur Ausführung der **MIPS-Befehle** notwendig?
  - Für alle Befehlsklassen werden folgende Komponenten benötigt:



## 5. 3 Befehlsabarbeitung und Datenpfade

- Befehl aus dem Befehlsspeicher holen
  - Befehl im Befehlsspeicher adressieren
  - Befehlszähler um 4 inkrementieren



Einheit für das Holen von Befehlen



## 5. 3 Befehlsabarbeitung und Datenpfade

- Befehle vom R-Typ: **opcode  $r_z$ ,  $r_m$ ,  $r_n$**

- Arithmetisch logische Befehle: **add, sub, and or**
- Vergleichsbefehle: **slt**

	6	5	5	5	5	6
Typ R	op	rs	rt	rd	shamt	funct

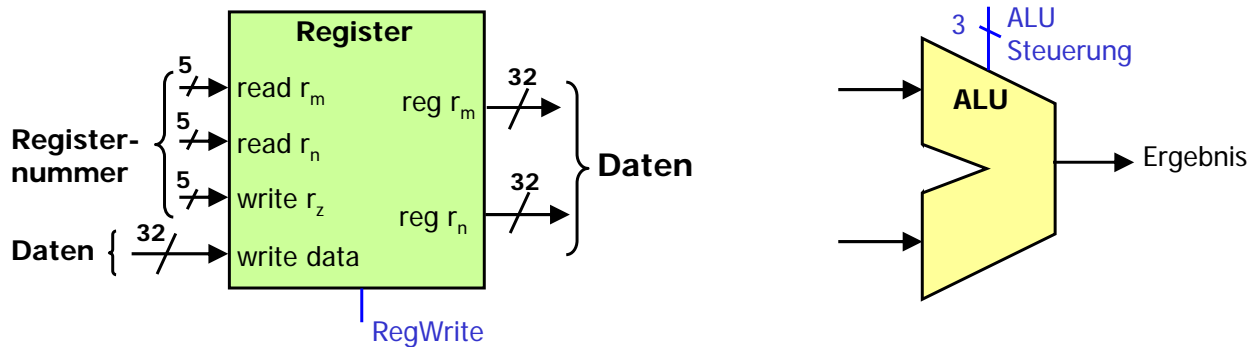
- Befehle haben 3 Operanden
- Operanden stehen in Registern
- Lesezugriff auf beiden Operanden Register und
- ein Schreibzugriff auf das Zielregister



## 5. 3 Befehlsabarbeitung und Datenpfade

### □ Befehle vom R-Typ:

**opcode**  $r_z, r_m, r_n$

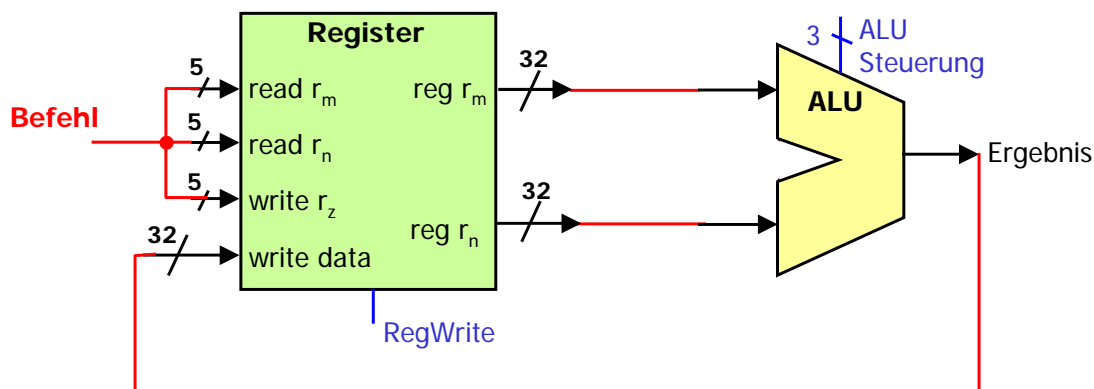


- Lesezugriff auf beiden Operanden Register und
- ein Schreibzugriff auf das Zielregister



## 5. 3 Befehlsabarbeitung und Datenpfade

### □ Befehle vom R-Typ:



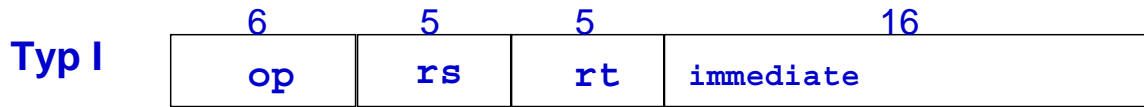
- Lesezugriff auf beiden Operanden Register und
- ein Schreibzugriff auf das Zielregister



## 5. 3 Befehlsabarbeitung und Datenpfade

### □ Lade- und Speicherbefehle (*load and store*)

**lw  $r_z$ , offset( $r_b$ )**      **sw  $r_z$ , offset( $r_b$ )**



- Speicheradresse wird durch die Addition einer Basisadresse im Register  $r_b$  zu einem vorzeichenbehafteten 16 Bit offset
- Bei Speicherbefehle wird das zu speichernde Wort aus dem Register  $r_z$  gelesen. Bei Ladebefehlen wird das geladene Wort ins Register  $r_b$  geladen

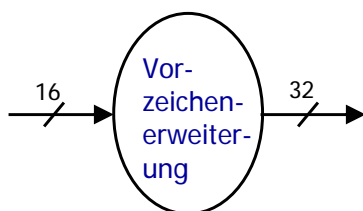


## 5. 3 Befehlsabarbeitung und Datenpfade

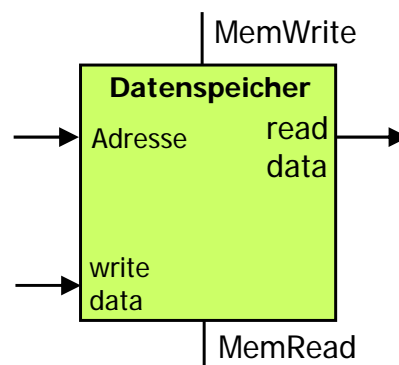
### □ Lade- und Speicherbefehle (*load and store*)

**lw  $r_z$ , offset( $r_b$ )**      **sw  $r_z$ , offset( $r_b$ )**

- Weitere benötigte Komponenten:



**Vorzeichen-  
erweiterungs-  
einheit**

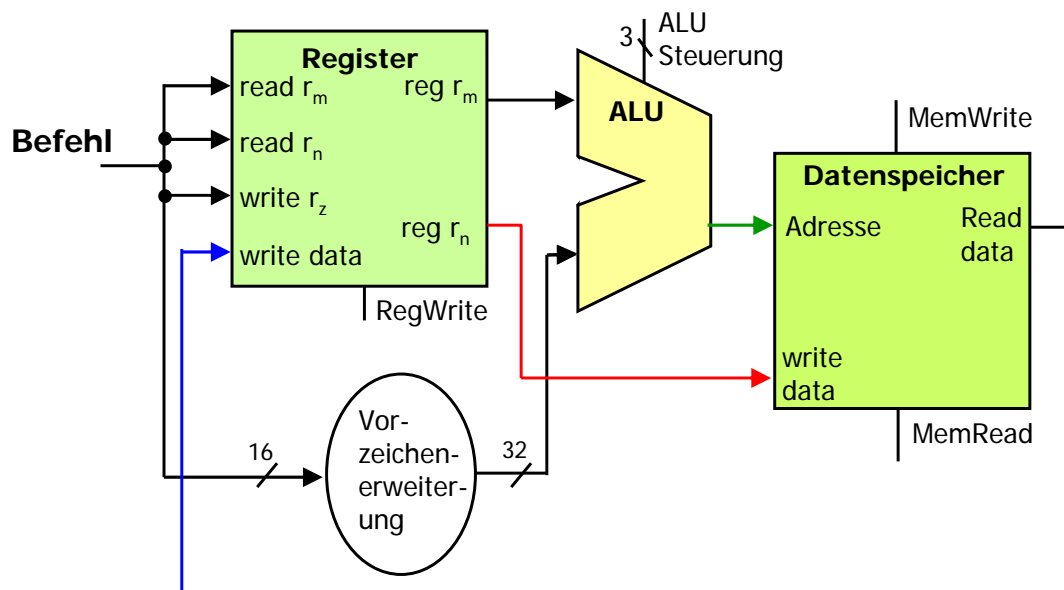


**Datenspeicher:**  
Steuersignale für Lese- (read data)  
und Schreibzugriffe (write data)



## 5. 3 Befehlsabarbeitung und Datenpfade

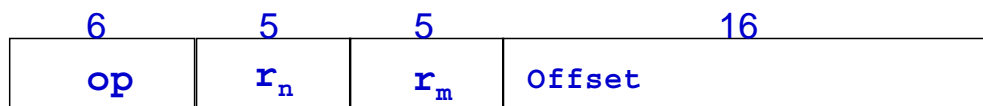
### □ Lade- und Speicherbefehle (*load and store*)



## 5. 3 Befehlsabarbeitung und Datenpfade

### □ Verzweigungsbefehle

**beq  $r_n$ ,  $r_m$ , offset**

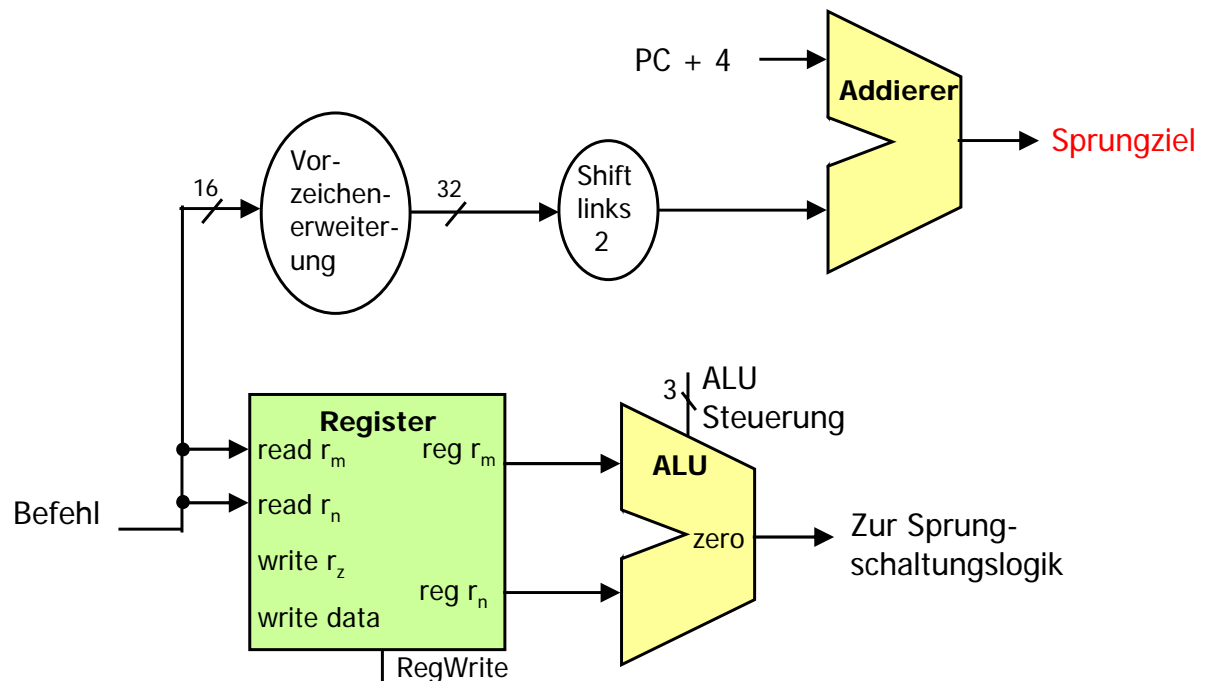


- 16-bit vorzeichenbehaftetes Offset  
➔  $2^{15}-1$  Befehle vorwärts und  $2^{15}$  rückwärts
- Basisadresse zur Berechnung der Sprungadresse ist die Adresse des Befehls hinter dem Verzweigungsbefehl, d. h. (PC+4)
  - Offset: um 2 Bits nach links verschieben, um Wörter zu adressieren
- Ergebnis des Vergleichs von  $r_n$  und  $r_m$  entscheidet, ob der Sprung „genommen“ bzw. „nicht genommen“ wird



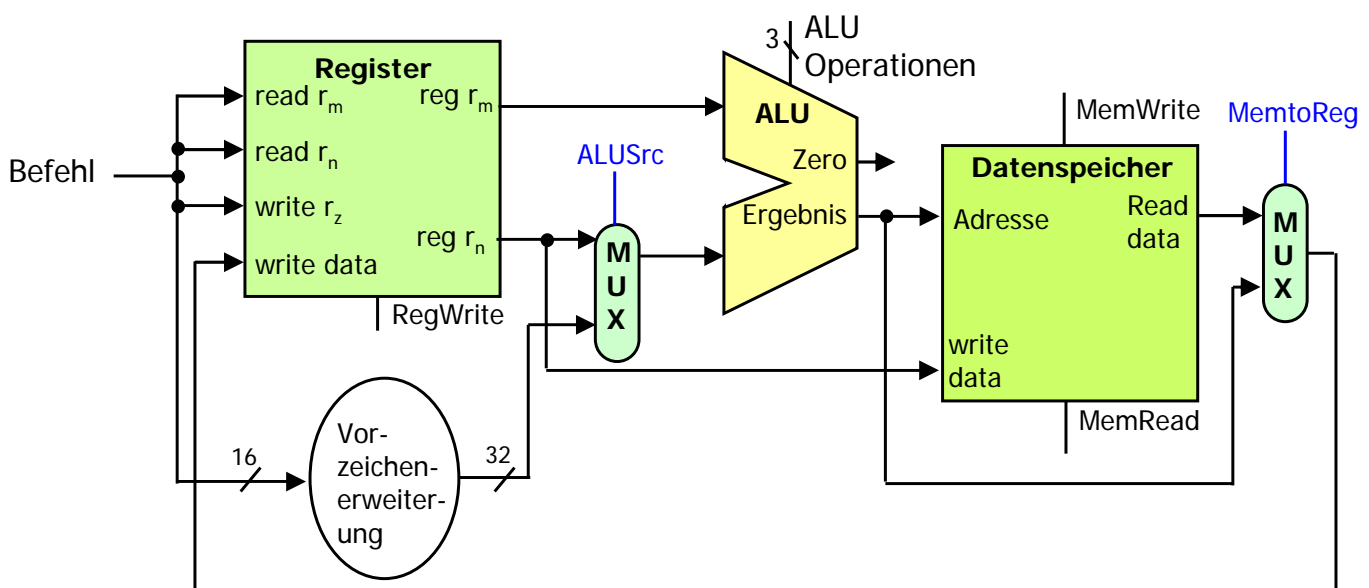
## 5. 3 Befehlsabarbeitung und Datenpfade

## ❑ Verzweigungsbefehle

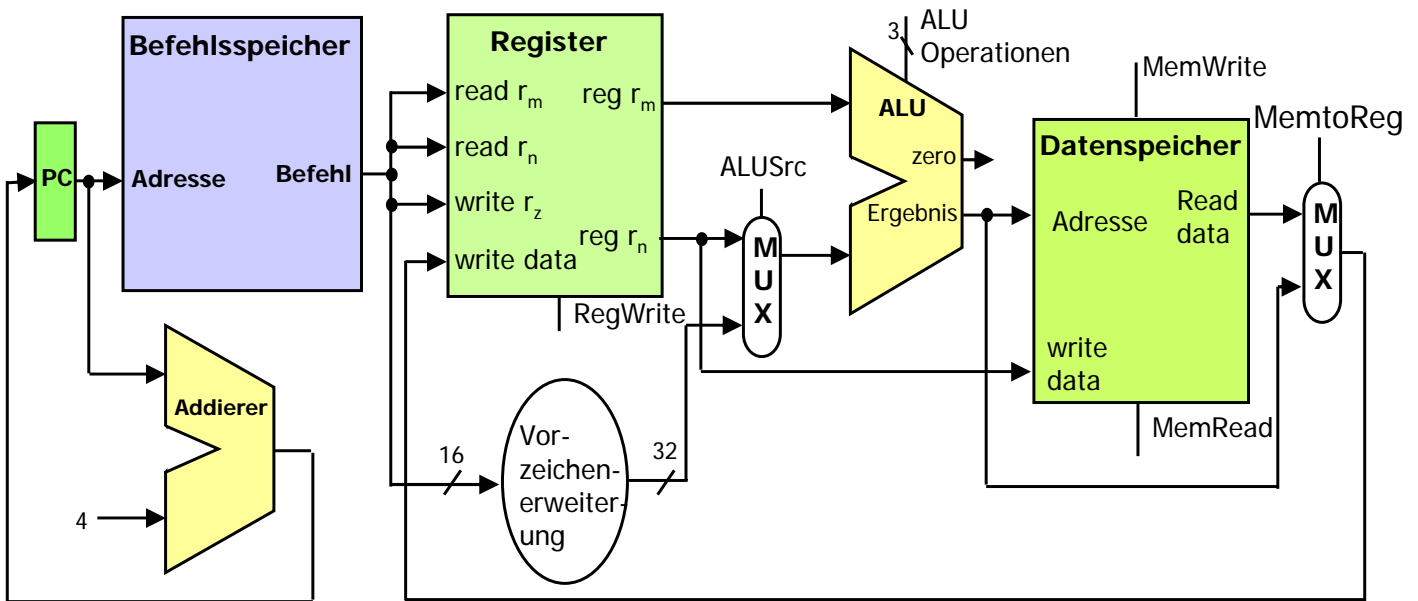


## 5. 3 Befehlsabarbeitung und Datenpfade

- ❑ **Datenpfad für Lade-Speicherbefehle und Befehle vom R-Typ**



## 5. 3 Befehlsabarbeitung und Datenpfade

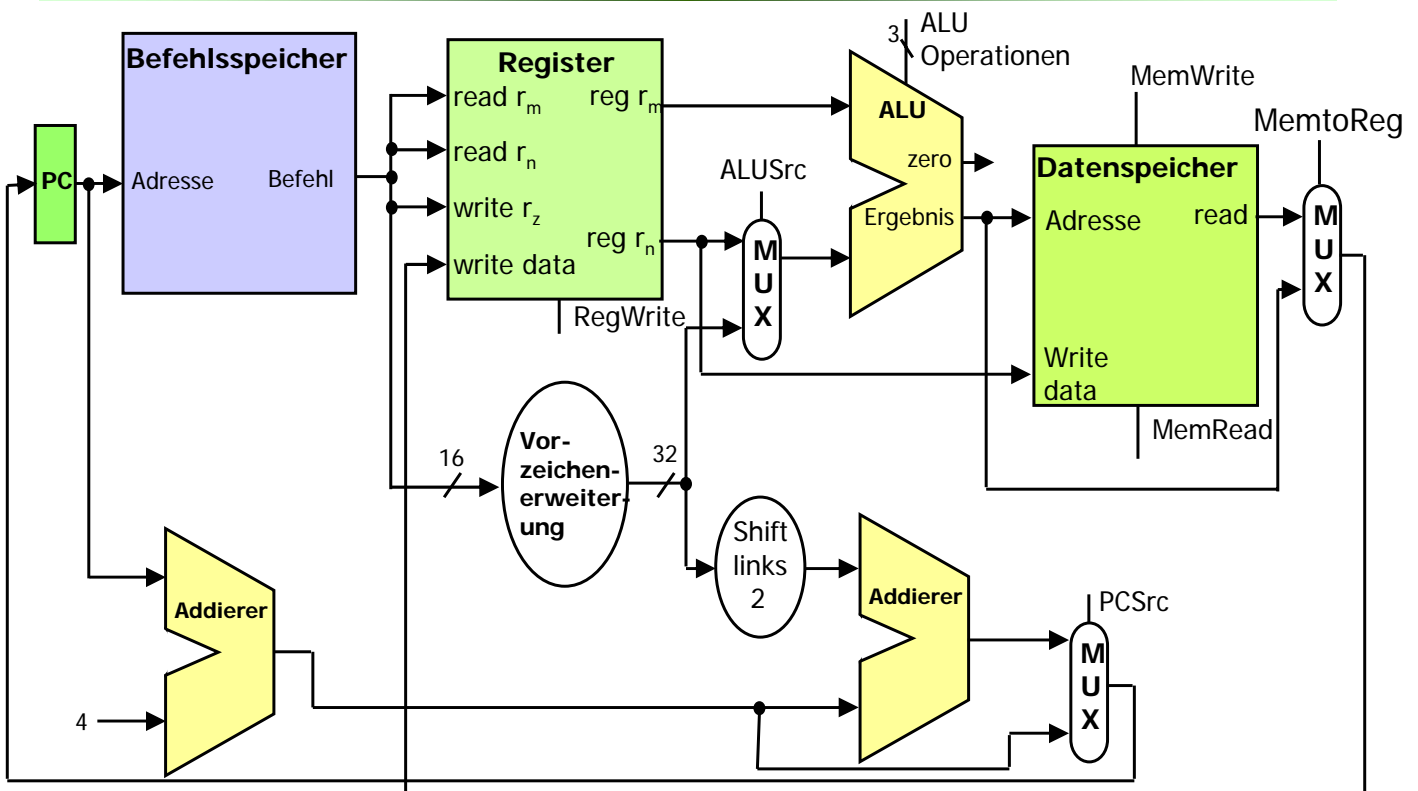


Einheit für das  
Holen von Befehlen

Datenpfad für Lade-Speicherbefehle  
und Befehle vom R-Typ

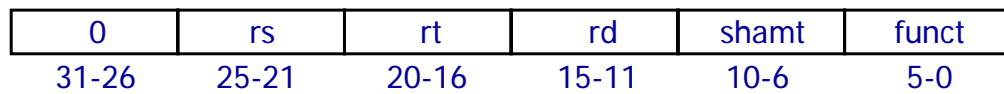


## Datenpfad für die MIPS-Architektur

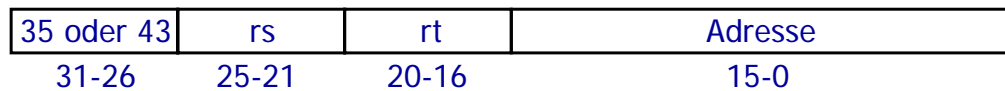


# Erinnerung: MIPS-Befehlsformate

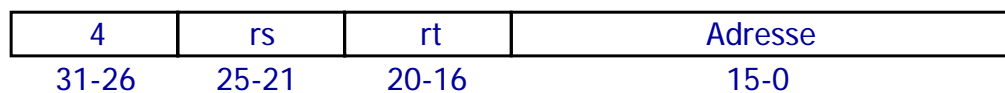
## ➤ R-Typ Befehl



## ➤ Lade/Speicher Befehl



## ➤ Sprung Befehl



# Datenpfad für die MIPS-Architektur

