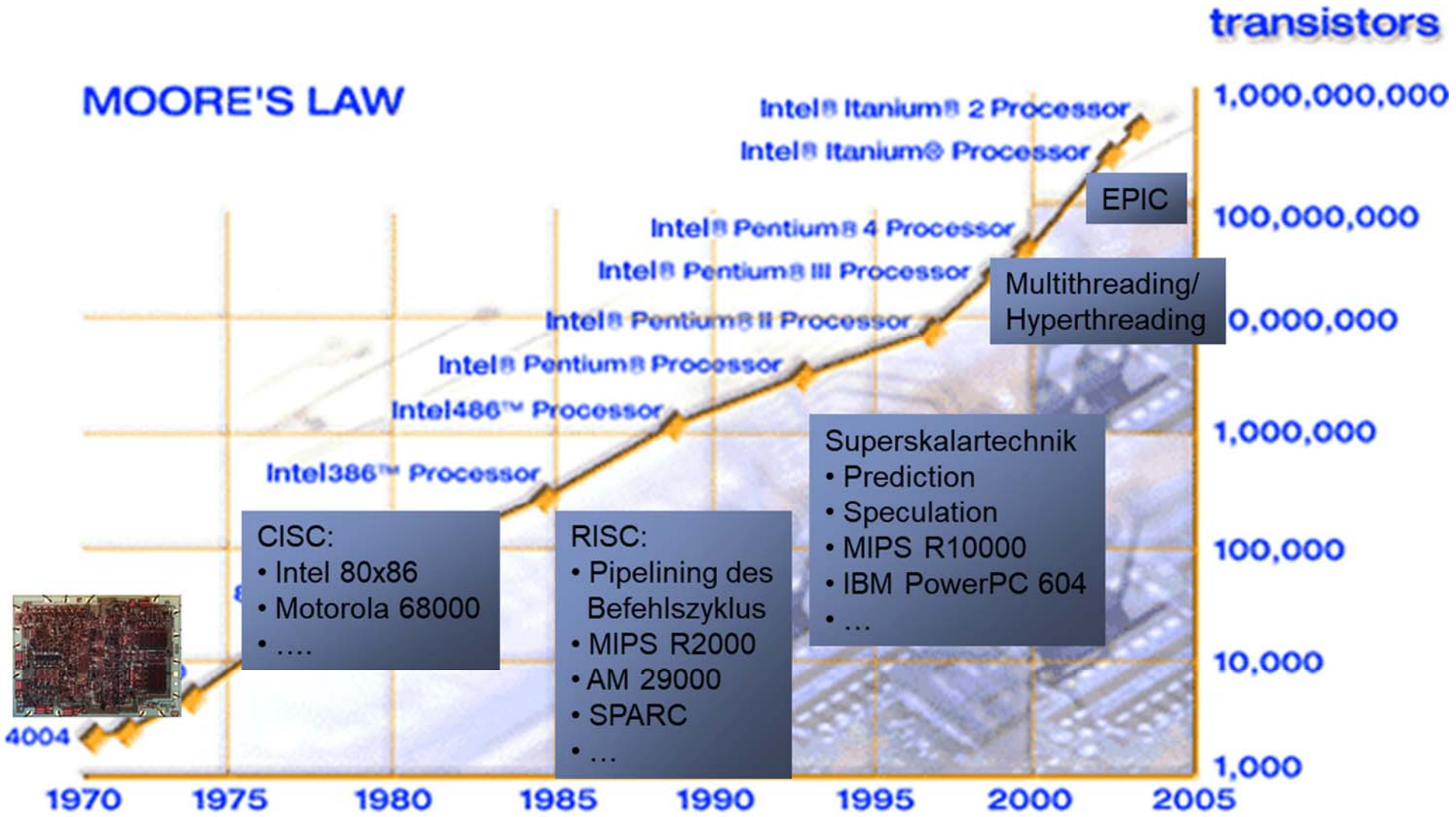
4.6 CISC vs. RISC

- **CISC Befehlssatz-Architektur (ISA)**
  - Complex Instruction Set Computer
- **RISC Befehlssatz-Architektur (ISA)**
  - Reduced Instruction Set Computer

CISC vs. RISC

- **Entwicklung der Mikroprozessoren**



4.6 CISC vs. RISC

- **CPU-Zeit**
  - Ausführungszeit eines Programms auf der CPU (gemessen in Taktzyklen oder in Zeiteinheiten)

$$CPU - Zeit = IC * CPI * Taktdauer$$

$$CPU - Zeit = \frac{IC * CPI}{Taktfrequenz}$$

  - Mit
    - IC: Anzahl der ausgeführten Befehle (Instruction Count)
    - CPI: Anzahl der Taktzyklen pro Befehl

$$CPI = \frac{Anzahl\ der\ Taktzyklen}{IC}$$

4.6 CISC vs. RISC

- **CISC ISA**
  - Befehlssatz mit komplexen Maschinenbefehlen
    - Entwurf der CISC Befehlssatz-Architekturen in einer Zeit, in der viel in Assembler programmiert worden ist
    - Tendenz komplexe Funktionalität mit den Instruktionen bereitzustellen, um Assembler-Programmierer zu unterstützen
    - Operationen einer Instruktion möglichst nahe an Anweisungen einer höheren Programmiersprache
      - Beispiel: Intel iAPX32 (~Mitte der 1980er Jahre):
        - Befehlssatz mit objektorientierter Verarbeitung in HW
        - Neben grundlegenden Datentypen auch Bitstreams, Arrays und Objects
  - Mikroprogrammierte Implementierung des Steuerwerks
  - Variables Befehlsformat und Befehlslänge

4.6 CISC vs. RISC

- **CISC ISA**
  - Ziel: $CPU - Zeit = IC * CPI * Taktdauer$
    - Reduzierung IC, also die Anzahl der auszuführenden Befehle für ein auszuführendes Programm
    - Je kompakter der Befehlsstrom, desto besser die Nutzung des Haupt- / Cache-Speichers, desto weniger Befehle müssen geholt werden, um ein Programm auszuführen
  - Beispiele:
    - IBM System 360, 30 (Familienkonzept)
    - DEC VAX-11
    - Intel x86 ISA (IA-32)
    - Motorola 68000 Familie
    - …

4.6 CISC vs. RISC

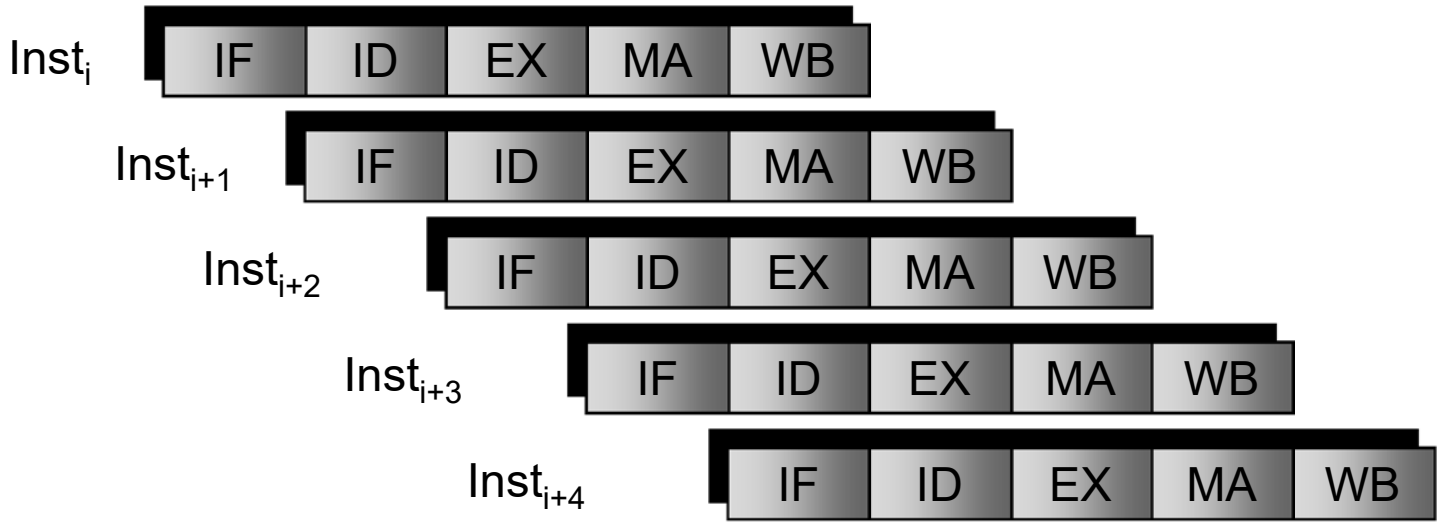
- **RISC ISA**
  - Ziel: $CPU - Zeit = IC * CPI * Taktdauer$
    - Reduzierung CPI, also die Anzahl der Zyklen pro Instruktion (CPI ~ 1)
  - Beispiele:
    - Forschung bei IBM, Standford (MIPS) Berkeley (RISC)
    - AMD 29000
    - MIPS R2000, Nachfolger
    - SPARC
    - ARM
    - IBM POWER

4.6 CISC vs. RISC

- **RISC ISA**
  - Einfache Maschinenbefehle
    - Einheitliches und festes Befehlsformat
  - Load/Store Architektur
    - Befehle arbeiten auf Registeroperanden
    - Lade- und Speicherbefehle greifen auf Speicher zu
  - Einzyklus-Maschinenbefehle
    - Effizientes Pipelining des Maschinenbefehlszyklus
    - Einheitliches Zeitverhalten der Maschinenbefehle, wovon nur Lade- und Speicherbefehle sowie die Verzweigungsbefehle abweichen
  - Optimierende Compiler
    - Reduzierung der Befehle im Programm

4.6 CISC vs. RISC

- **RISC ISA**
  - Effizientes Pipelining des Maschinenbefehlszyklus



IF: Befehl holen
ID: Befehl dekodieren
EX: Befehl ausführen
MA: Speicherzugriff
WB: Zurückschreiben

Kapitel 5
Prozessor-Organisation und Pipelining des Maschinenbefehlszyklus

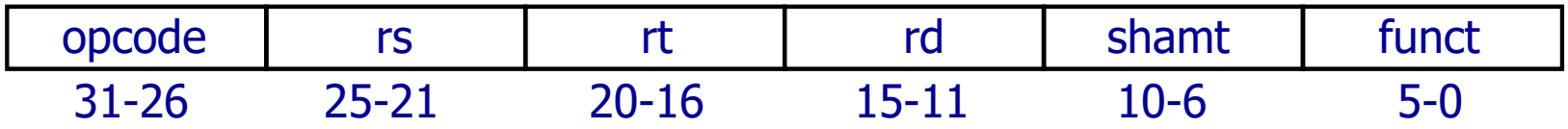
5. Prozessor-Organisation

- 5.1 Aufbau des DLX- (MIPS-) Prozessors
  - Datenpfade und Befehlsabarbeitung

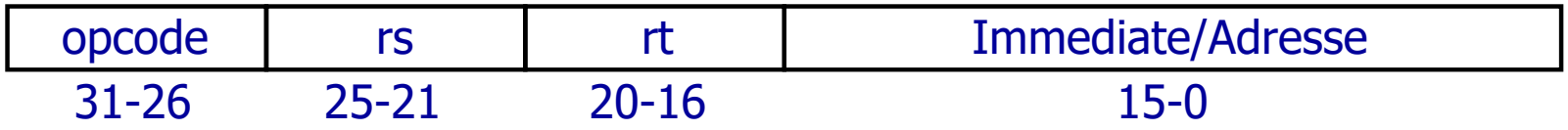
5.1 Aufbau des DLX- (MIPS-) Prozessors

- **Datenpfade und Befehlsabarbeitung**
- **Befehlsformate MIPS ISA**

  - Typ R: Register-Register-Befehle (z.B. add, sub, …)



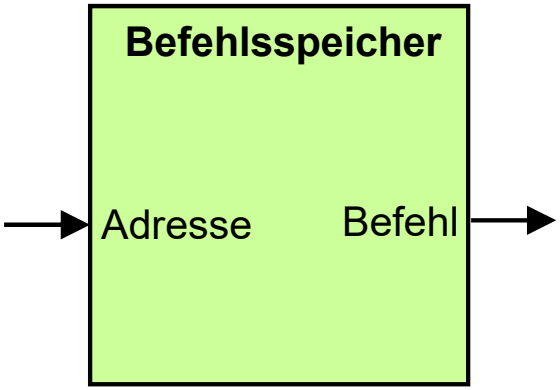
  - Typ I: Immediate-Register Befehle (z.B. addi, lw, beq, …)



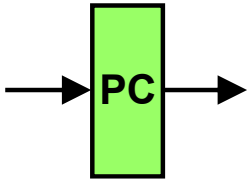
  - Typ J: Jump (z.B. j, jal, …)

5.1 Aufbau des DLX- (MIPS-) Prozessors

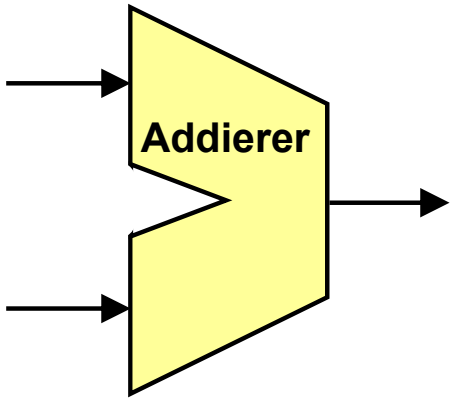
- **Datenpfade und Befehlsabarbeitung**
  - Welche Hardware-Komponenten sind zur Ausführung der MIPS-Befehle notwendig?
  - Für alle Befehlsklassen werden folgende Komponenten benötigt:



Speicher für Befehle



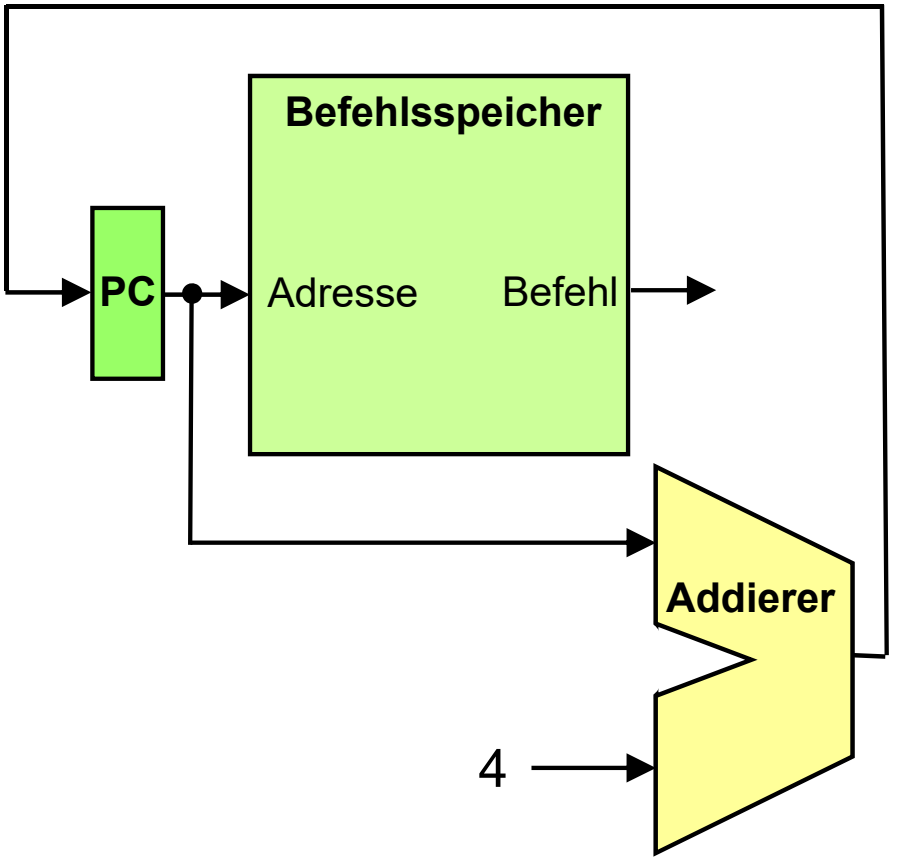
Befehlszähler



Addierer: berechnet die Adresse des nächsten Befehls

5.1 Aufbau des DLX- (MIPS-) Prozessors

- **Datenpfade und Befehlsabarbeitung**
  - Befehl aus dem Befehlsspeicher holen
    - Befehl im Befehlsspeicher adressieren
    - Befehlszähler um 4 inkrementieren

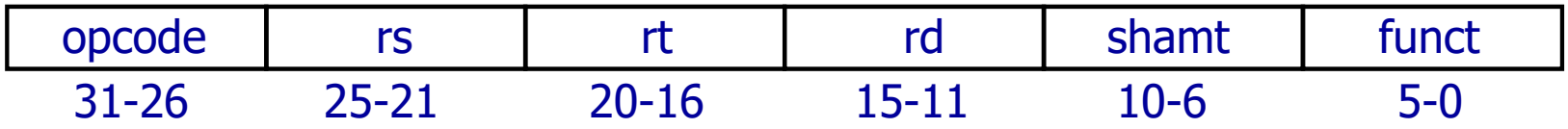


Einheit für das Holen von Befehlen

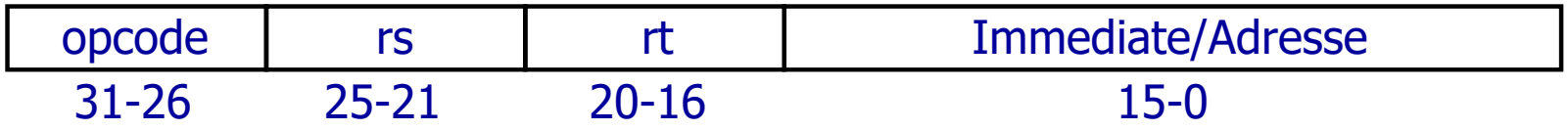
5.1 Aufbau des DLX- (MIPS-) Prozessors

- **Datenpfade und Befehlsabarbeitung**
- **Befehlsformate MIPS ISA**

  - Typ R: Register-Register-Befehle (z.B. add, sub, …)



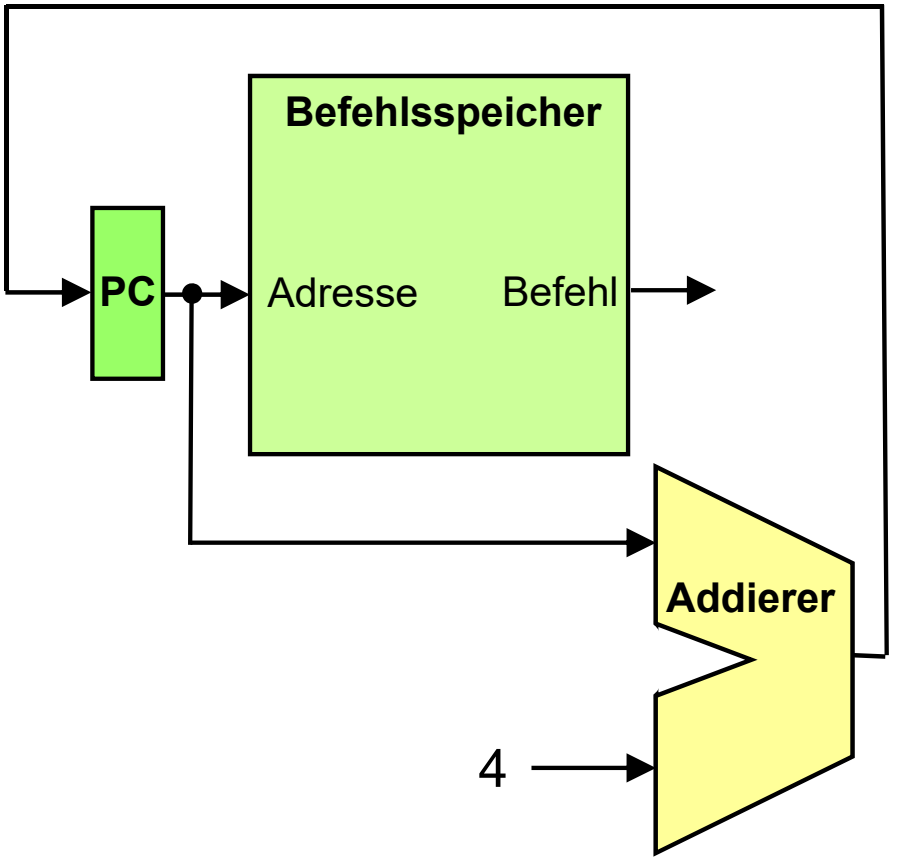
  - Typ I: Immediate-Register Befehle (z.B. addi, lw, beq, …)



  - Typ J: Jump (z.B. j, jal, …)

5.1 Aufbau des DLX- (MIPS-) Prozessors

- **Datenpfade und Befehlsabarbeitung**
  - Befehl aus dem Befehlsspeicher holen
    - Befehl im Befehlsspeicher adressieren
    - Befehlszähler um 4 inkrementieren



Einheit für das Holen von Befehlen

5.1 Aufbau des DLX- (MIPS-) Prozessors

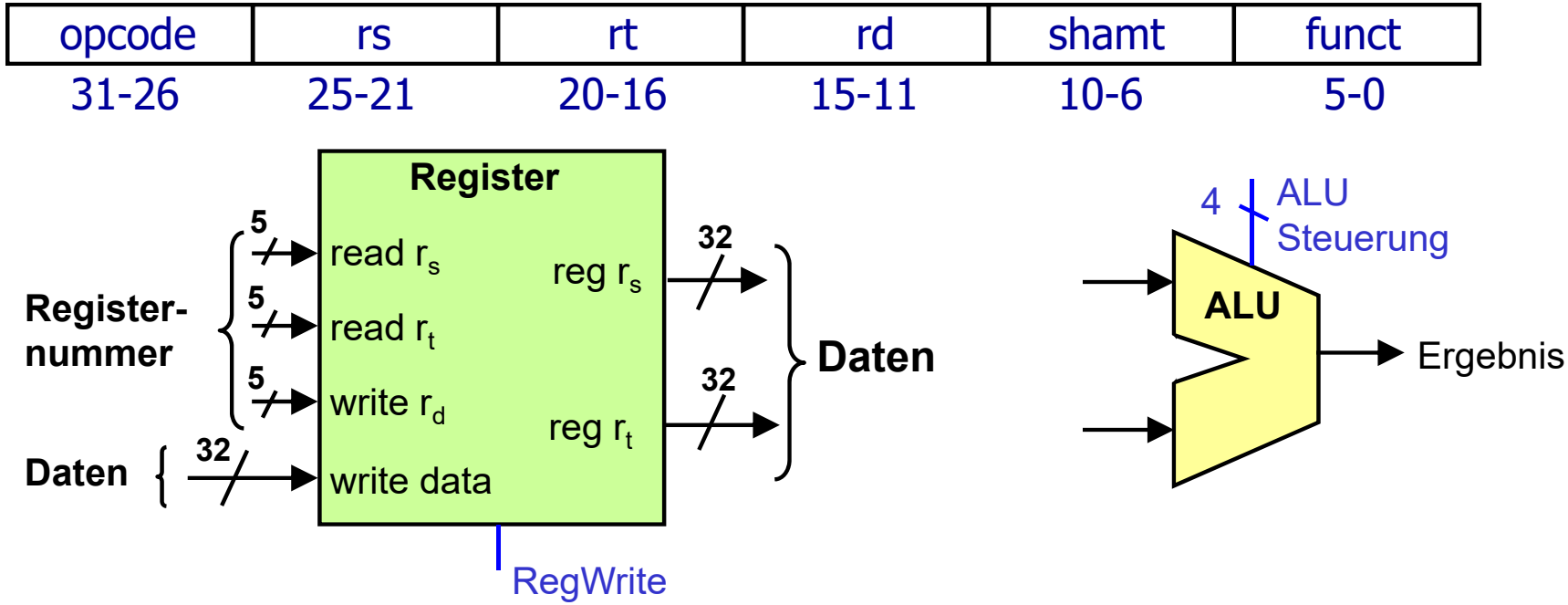
- **Datenpfade und Befehlsabarbeitung**
- **Befehle vom R-Typ: `opcode rd,rs,rt`**

opcode	rs	rt	rd	shamt	funct
31-26	25-21	20-16	15-11	10-6	5-0

  - Haben alle den Opcode 0 (Bitfeld 31-26)
    - Arithmetisch logische Befehle: **`add`**, **`sub`**, **`and`**, **`or`**
    - Vergleichsbefehle: **`slt`**
    - ALU-Funktion steht im Bitfeld 5-0 (funct)
  - Befehle haben 3 Registeroperanden
    - Felder **`rt`**, und **`rs`** bezeichnen Quellregister
    - Feld **`rd`** bezeichnet das Zielregister

5.1 Aufbau des DLX- (MIPS-) Prozessors

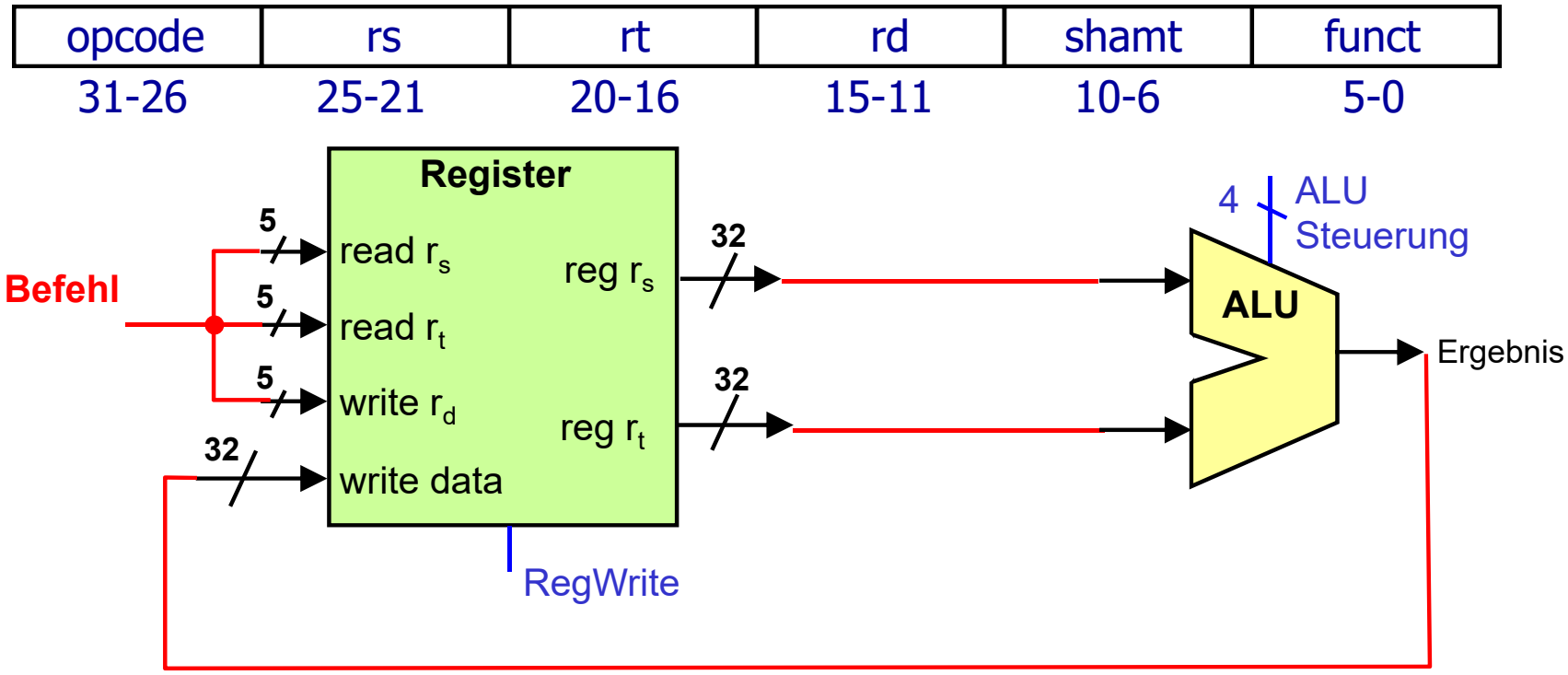
- **Datenpfade und Befehlsabarbeitung**
- **Befehle vom R-Typ:** `opcode rd,rs,rt`



- 2 Lesezugriffe auf die Operanden-Register und
- 1 Schreibzugriff auf das Zielregister

5.1 Aufbau des DLX- (MIPS-) Prozessors

- **Datenpfade und Befehlsabarbeitung**
- **Befehle vom R-Typ:** `opcode rd,rs,rt`

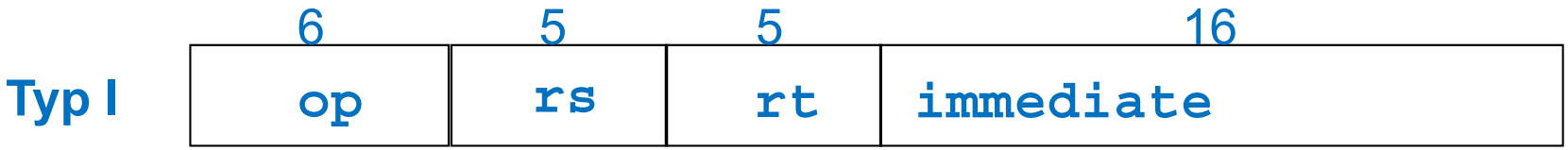


- 2 Lesezugriffe auf die Operanden-Register und
- 1 Schreibzugriff auf das Zielregister

5.1 Aufbau des DLX- (MIPS-) Prozessors

- **Datenpfade und Befehlsabarbeitung**
- **Lade- und Speicherbefehle (load, store)**

```
lw rt, offset(rs)          sw rt, offset(rs)
```



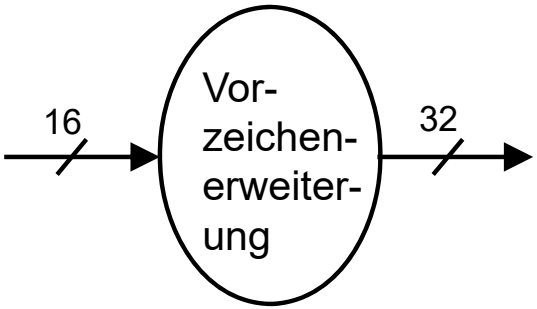
- Opcode für Lade- (35_D) und Speicherbefehl (43_D)
- Das `rs`-Register ist das Basisregister, das mit dem Wert im Feld immediate aufsummiert
- Bei Ladebefehlen ist das `rt`-Register das Zielregister, in der der Wert geladen wird.
- Bei Speicherbefehlen ist `rt` das Quellregister, dessen Wert im Speicher gespeichert werden soll

5.1 Aufbau des DLX- (MIPS-) Prozessors

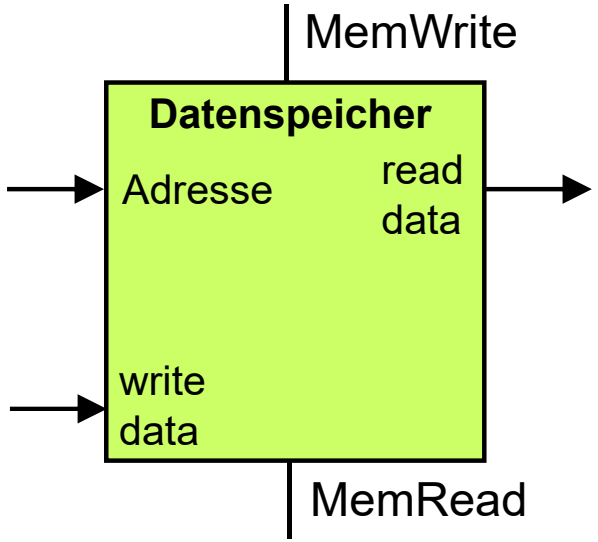
- **Datenpfade und Befehlsabarbeitung**
- **Lade- und Speicherbefehle (load, store)**

```
lw rt, offset(rs)        sw rt, offset(rs)
```

  - Weitere Komponenten



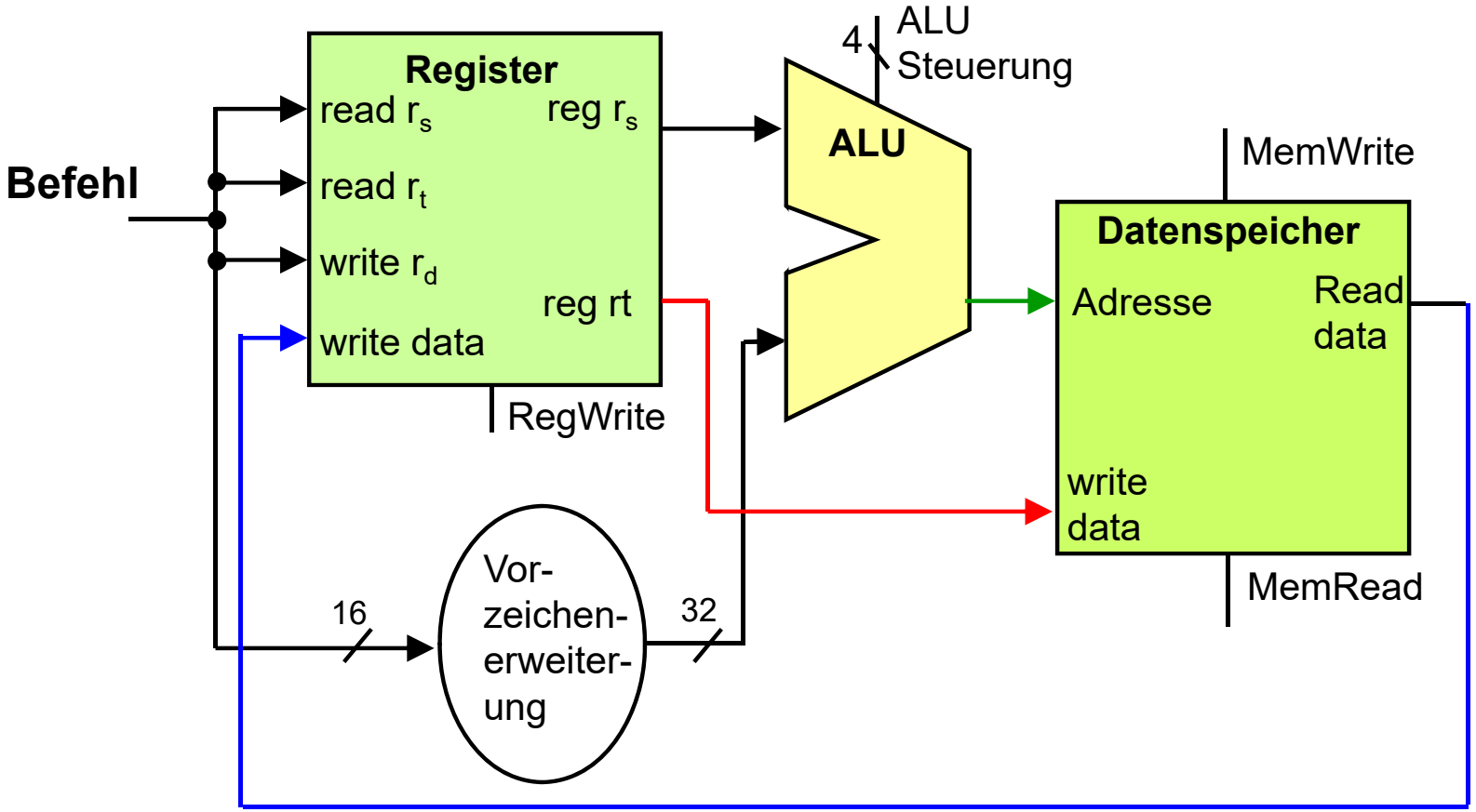
Vorzeichen-
erweiterungs-
einheit

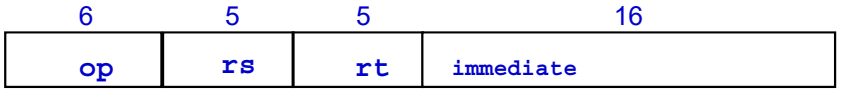


Datenspeicher:
Steuersignale für Lese- (read data) und Schreibzugriffe (write data)

5.1 Aufbau des DLX- (MIPS-) Prozessors

- Datenpfade und Befehlsabarbeitung
- Lade- und Speicherbefehle (load, store)

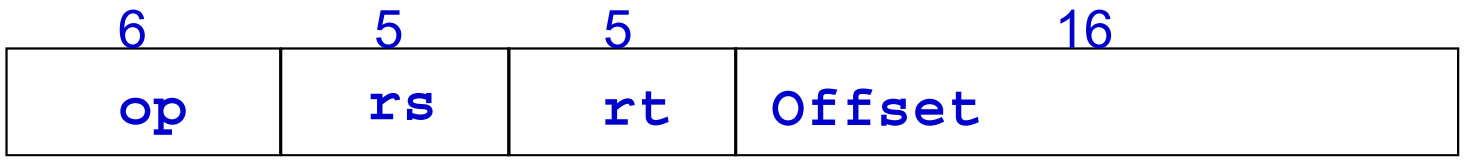




5.1 Aufbau des DLX- (MIPS-) Prozessors

- **Datenpfade und Befehlsabarbeitung**
- **Verzweigungsbefehl (`branch on equal`, Opcode 4_D)**

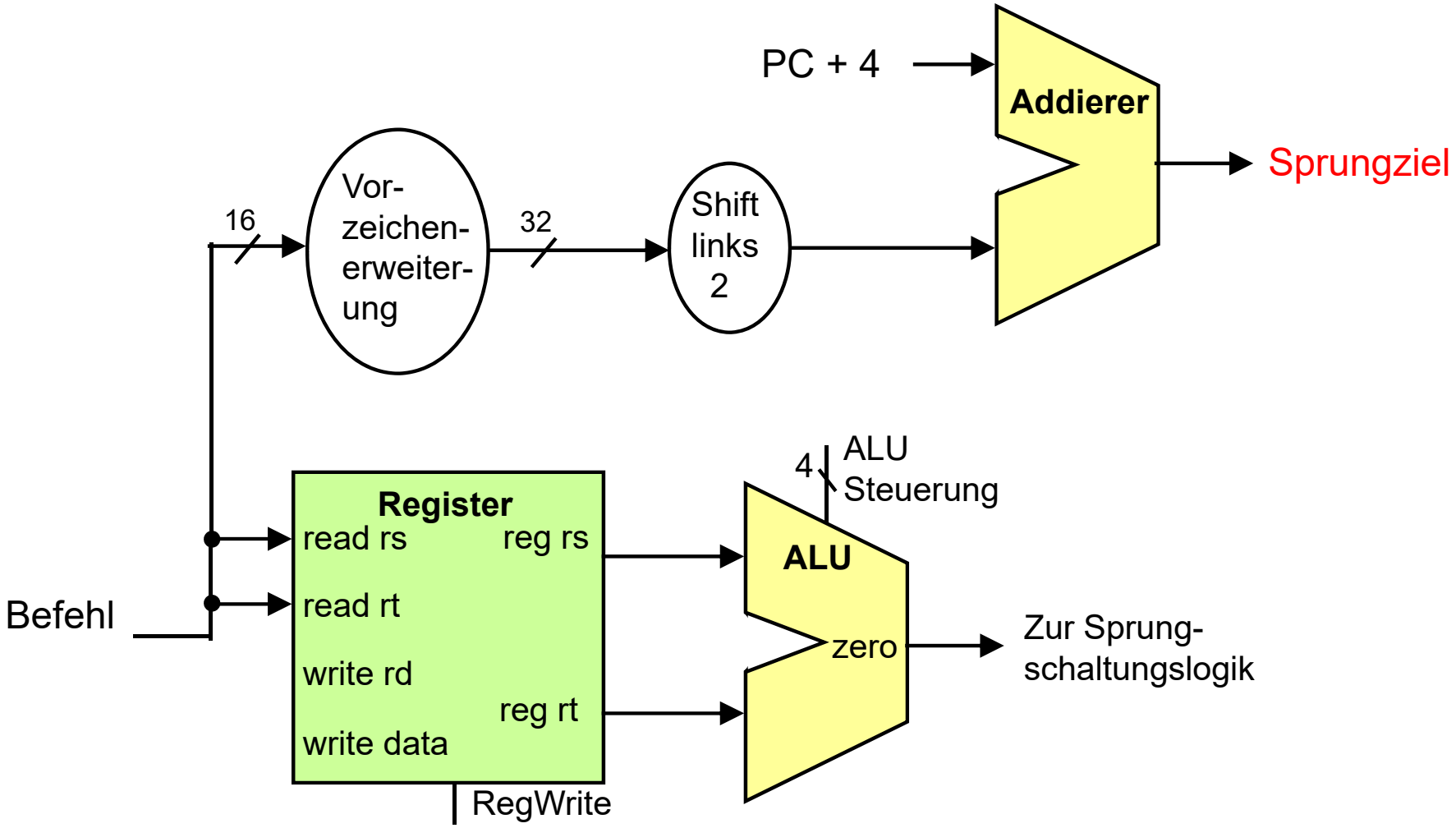
```
beq rs,rt,offset
```



  - Die Register `rs` und `rt` sind die Quellregister, die auf Gleichheit geprüft werden
    - Ergebnis entscheidet, ob der Sprung genommen wird oder nicht
  - Das Adressfeld wird vorzeichenerweitert, geschoben und zum Befehlszähler addiert und ergibt so die Sprungadresse
    - 16-bit vorzeichenbehaftetes Offset →bis zu 2^{15}-1 Befehle vorwärts, bzw. 2^{15} rückwärts
    - Offset: um 2 Bits nach links verschieben, um Wörter zu adressieren

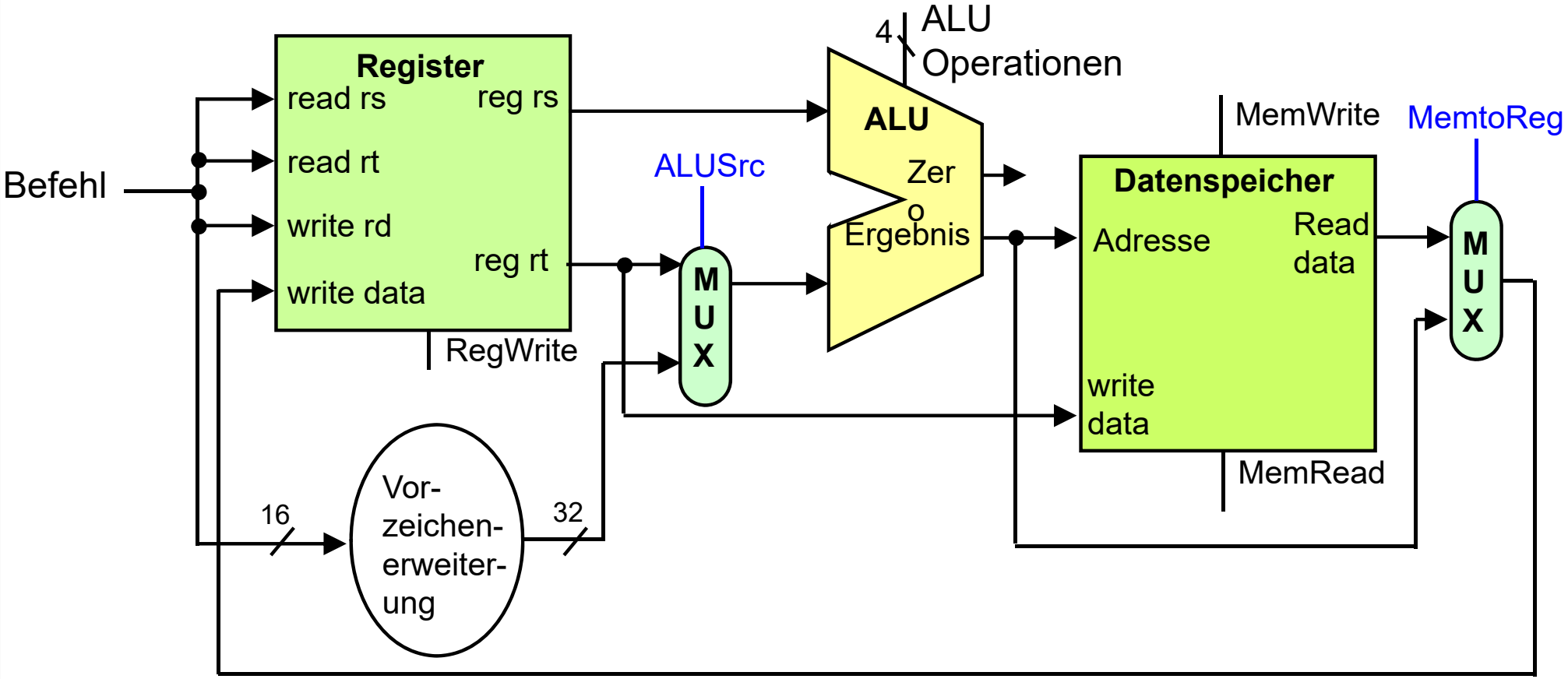
5.1 Aufbau des DLX- (MIPS-) Prozessors

- **Datenpfade und Befehlsabarbeitung**
- **Verzweigungsbefehl**



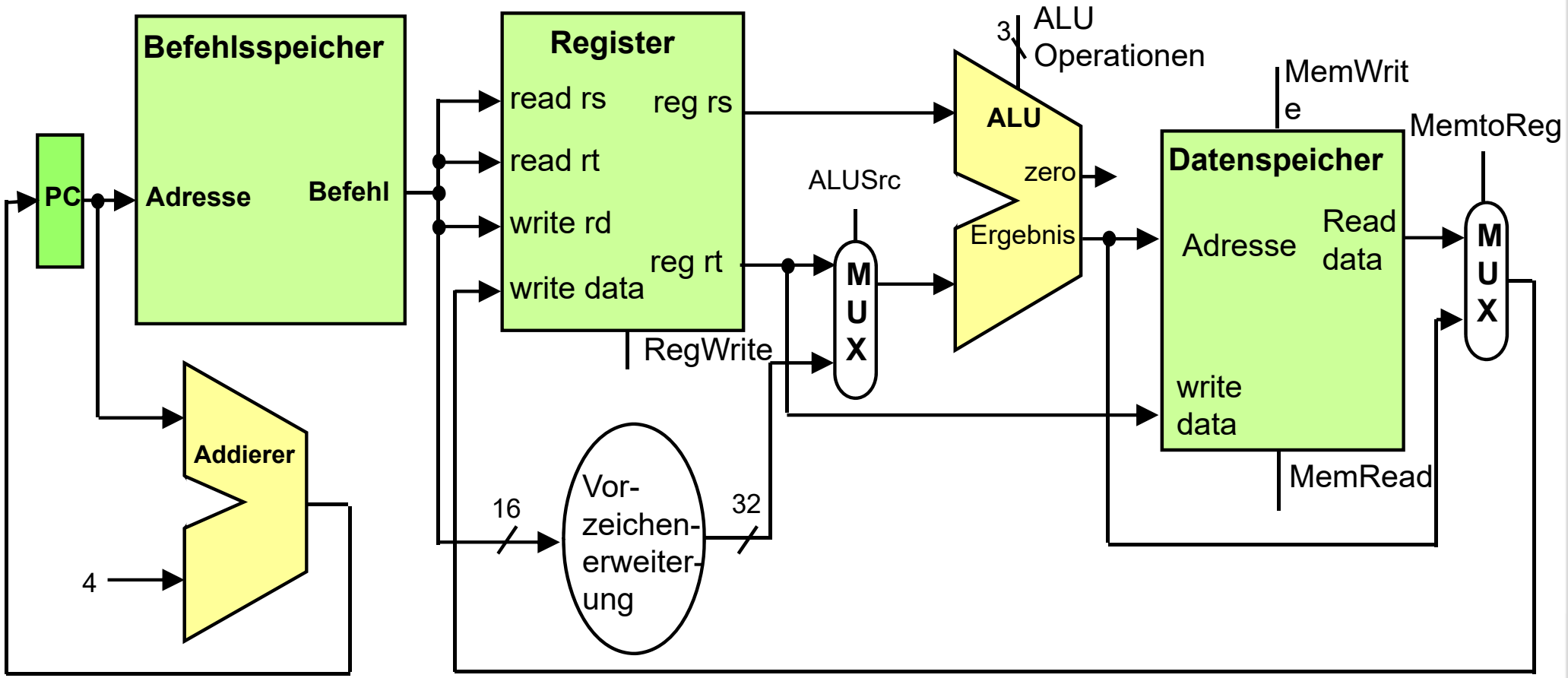
5.1 Aufbau des DLX- (MIPS-) Prozessors

- Datenpfade und Befehlsabarbeitung
- Datenpfade für Lade-/Speicher-Befehle und R-Typ-Befehle



5.1 Aufbau des DLX- (MIPS-) Prozessors

Datenpfade und Befehlsabarbeitung

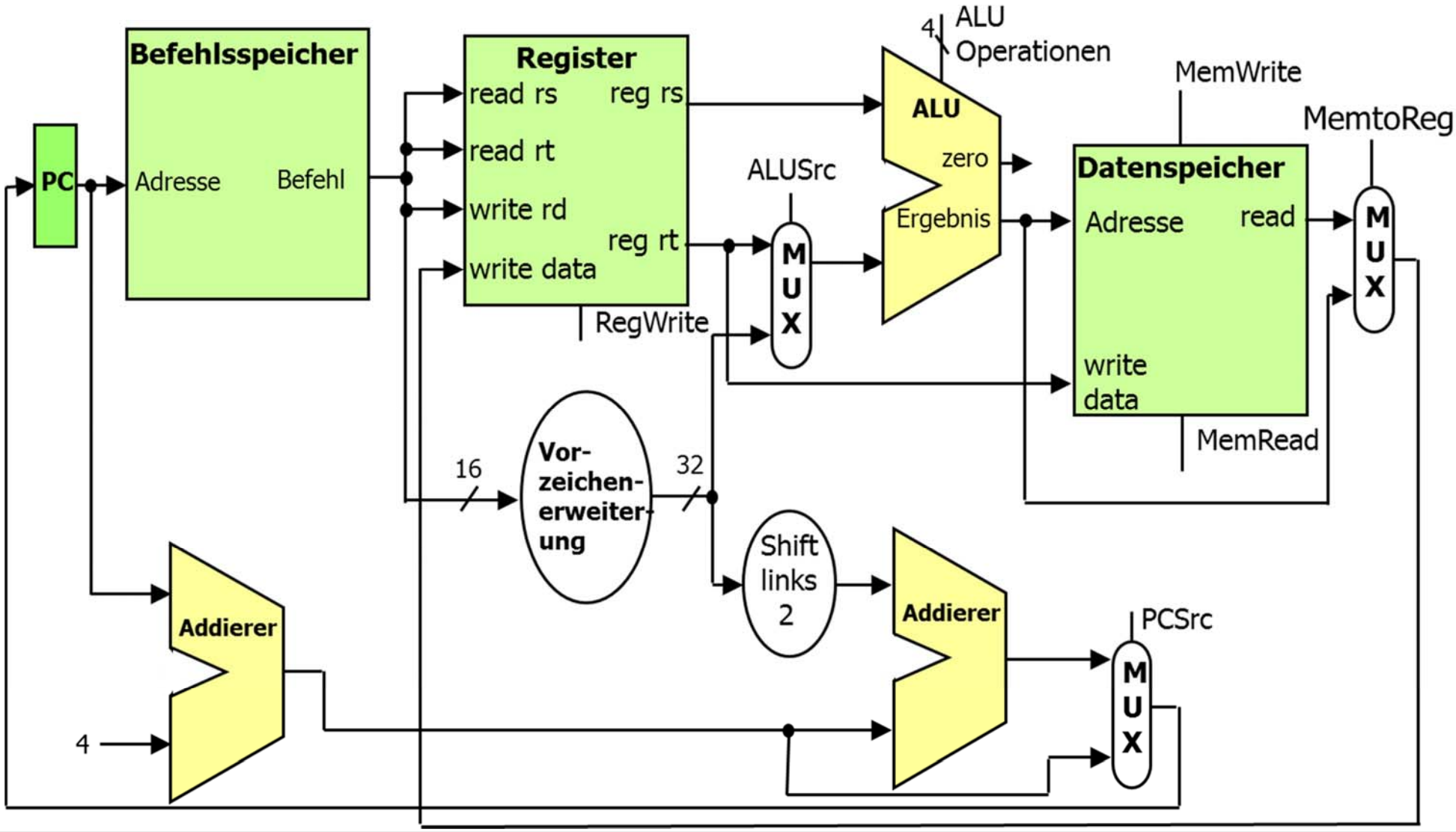


Einheit für das Holen von Befehlen

Datenpfade für Lade-Speicherbefehle und Befehle vom R-Typ

5.1 Aufbau des DLX- (MIPS-) Prozessors

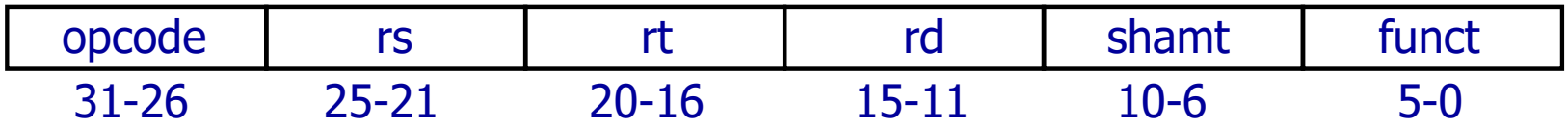
Datenpfade für DLX-Prozessor (MIPS)



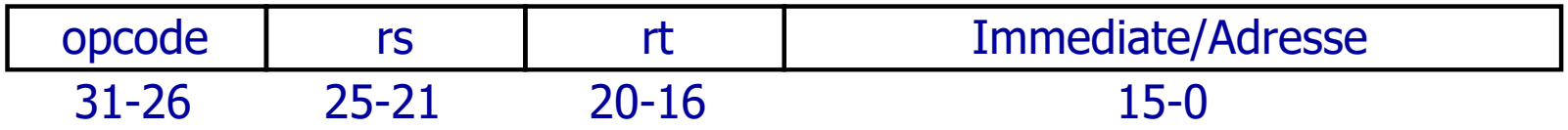
5.1 Aufbau des DLX- (MIPS-) Prozessors

- **Datenpfade und Befehlsabarbeitung**
- **Befehlsformate MIPS ISA**

  - Typ R: Register-Register-Befehle (z.B. add, sub, …)



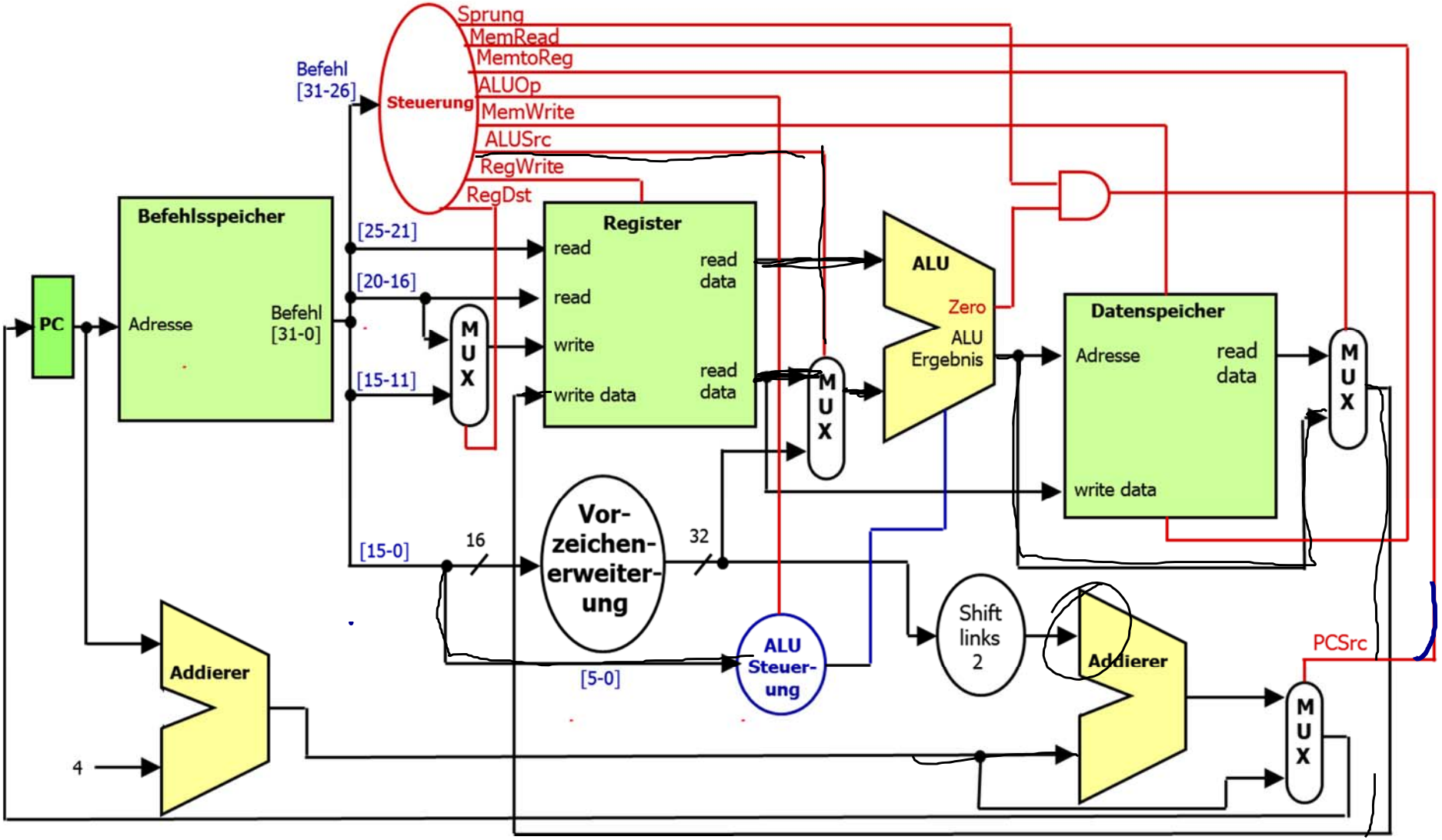
  - Typ I: Immediate-Register Befehle (z.B. addi, lw, beq, …)



  - Typ J: Jump (z.B. j, jal, …)

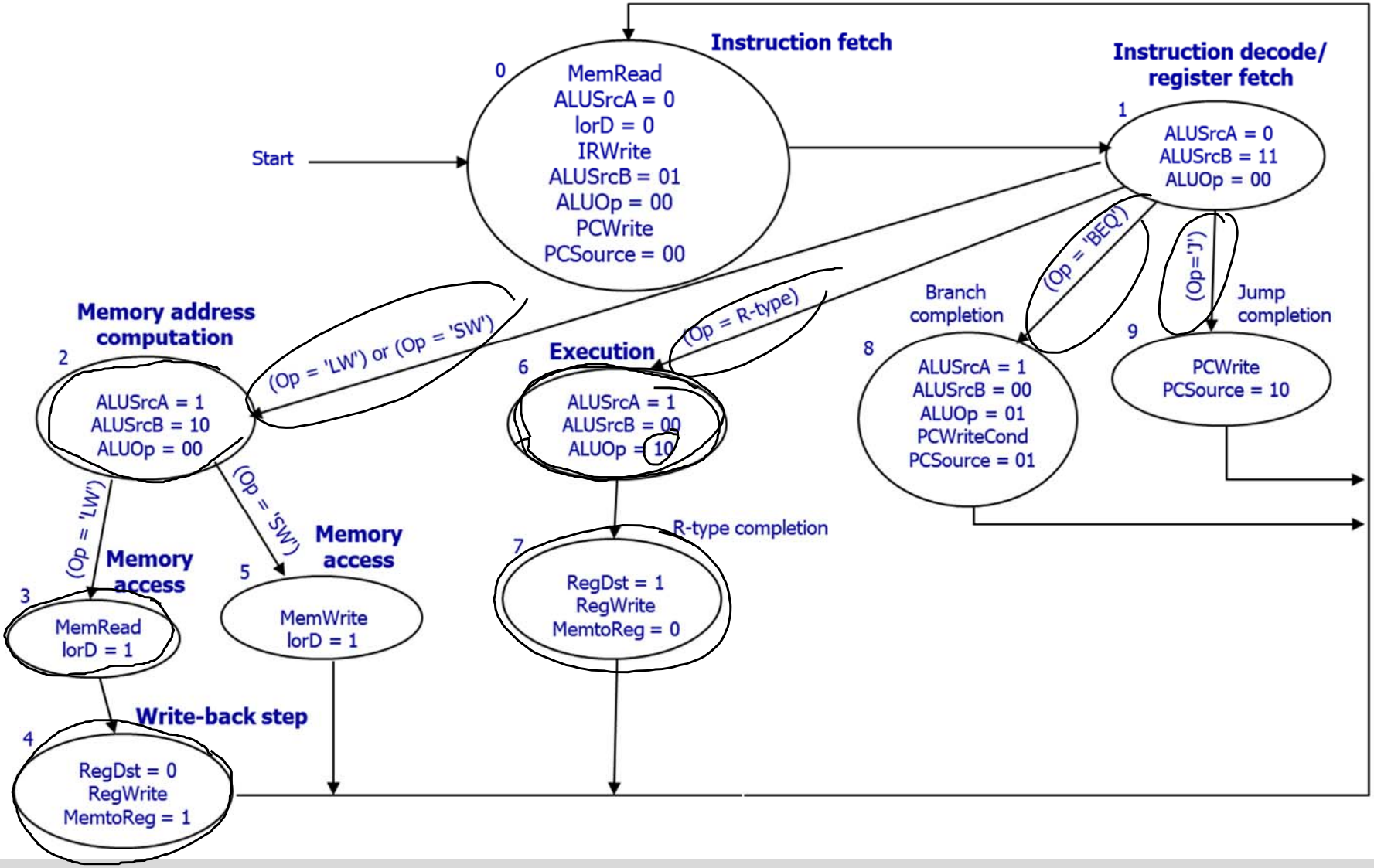
5.1 Aufbau des DLX- (MIPS-) Prozessors

Datenpfade für DLX-Prozessor (MIPS): Steuerung



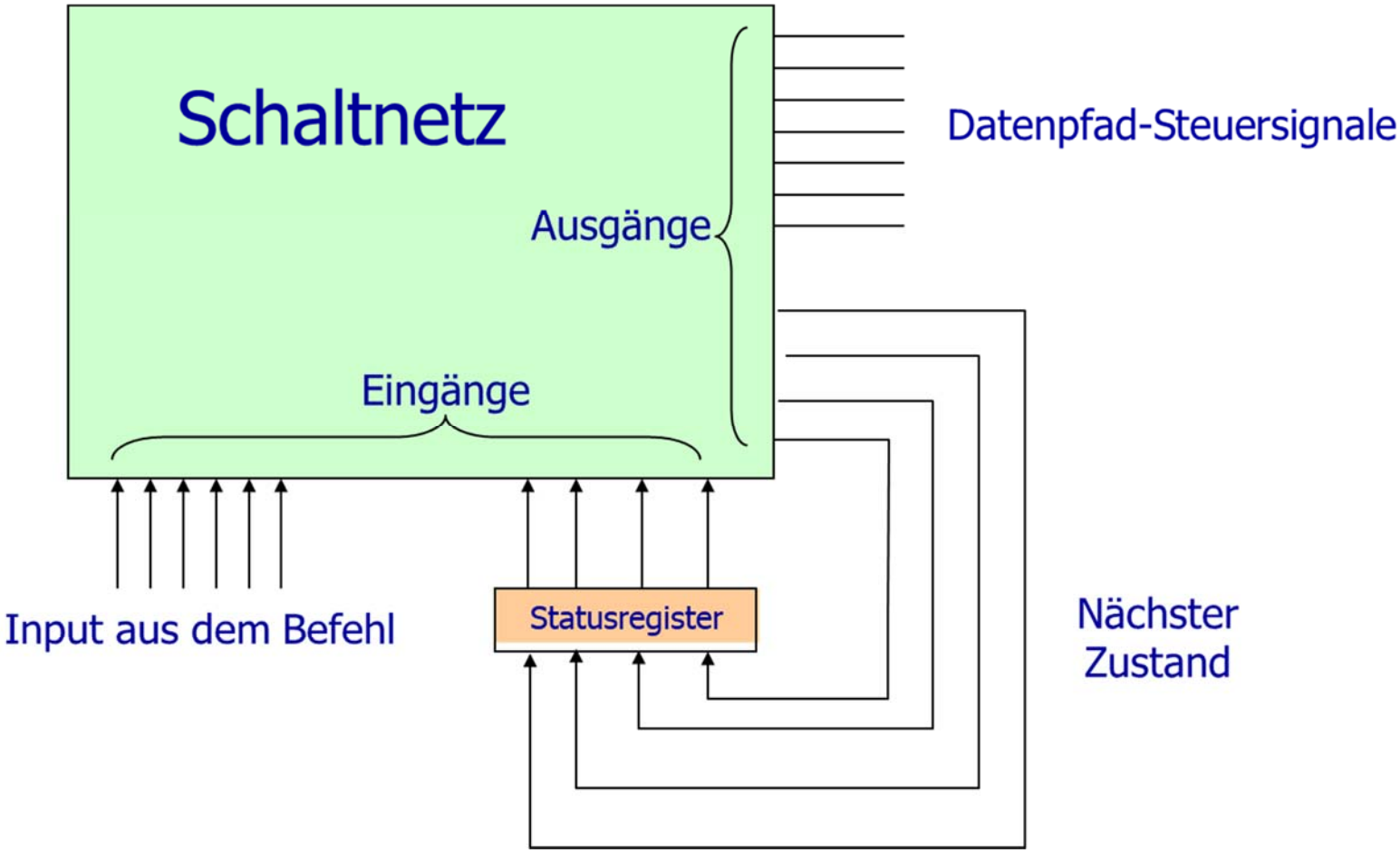
5.1 Aufbau des DLX- (MIPS-) Prozessors

DLX-Prozessor (MIPS): Zustandsautomat



5.1 Aufbau des DLX- (MIPS-) Prozessors

- **DLX-Prozessor (MIPS): Implementierung**

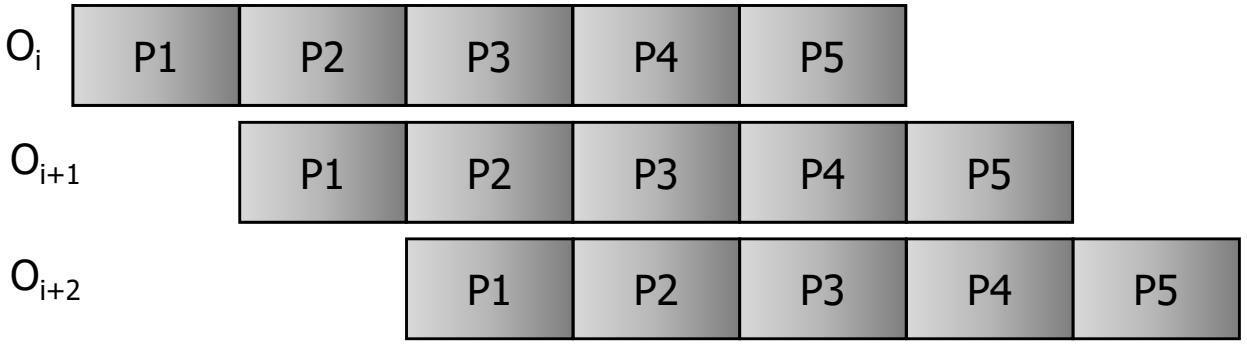


5. Prozessor-Organisation

- 5.1 Aufbau des DLX- (MIPS-) Prozessors
  - Datenpfade und Befehlsabarbeitung
- 5.2 Pipelining des Maschinenbefehlszyklus
  - Pipeline-Organisation des DLX- (MIPS) Prozessors

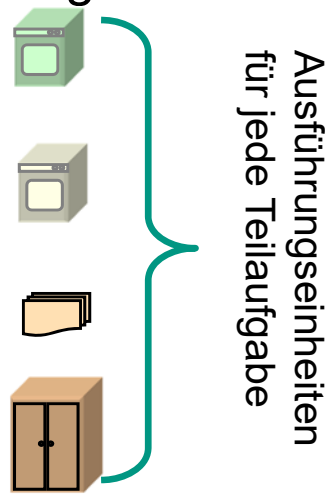
5.2 Pipelining des Maschinenbefehlszyklus

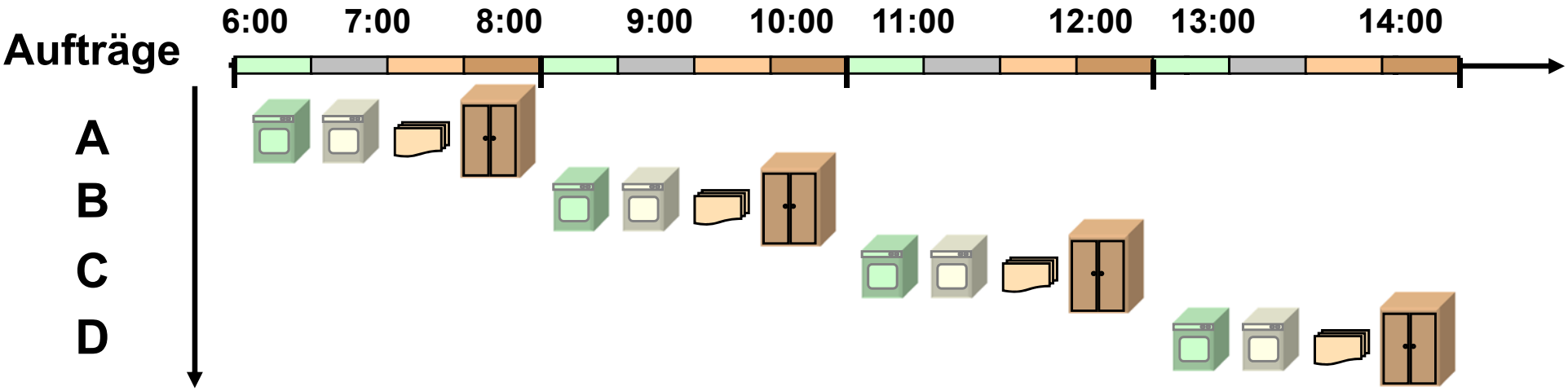
- **Pipelining – Grundprinzip**
  - **Definition**
    - Pipelining auf einer Maschine liegt dann vor, wenn die Bearbeitung eines Objektes in Teilschritte zerlegt und diese in einer sequentiellen Folge (Phasen der Pipeline) ausgeführt werden. Die Phasen der Pipeline können für verschiedene Objekte überlappt abgearbeitet werden. (Bode 95)



5.2 Pipelining des Maschinenbefehlszyklus

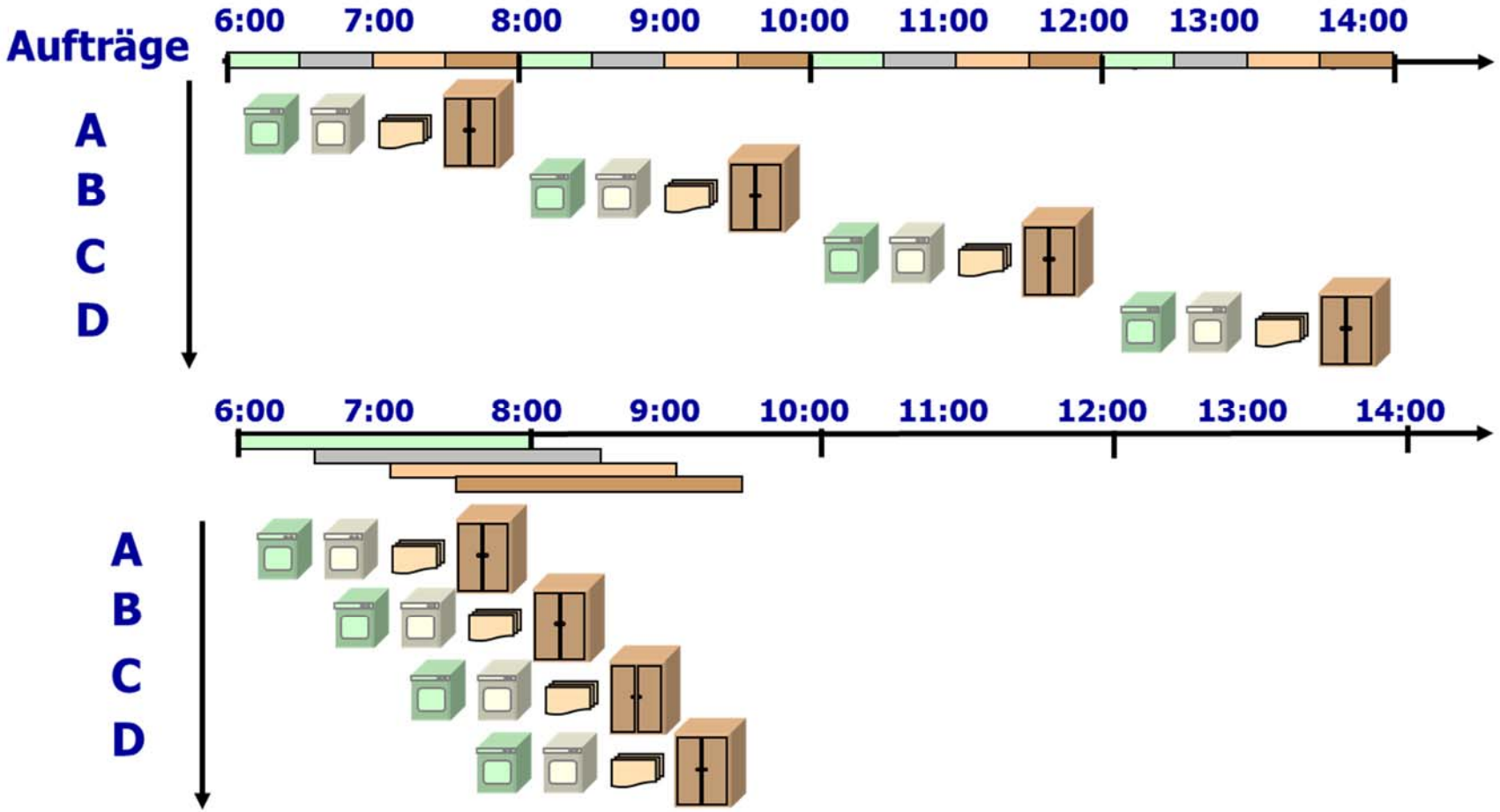
- **Pipelining – Grundprinzip**
  - **Beispiel Waschvorgang:** umfasst folgende 4 Teilaufgaben:
    - Schmutzige Wäsche in die Waschmaschine
    - Nasse Wäsche in den Trockner
    - Falten, Bügeln, ...
    - Kleider in den Schrank





5.2 Pipelining des Maschinenbefehlszyklus

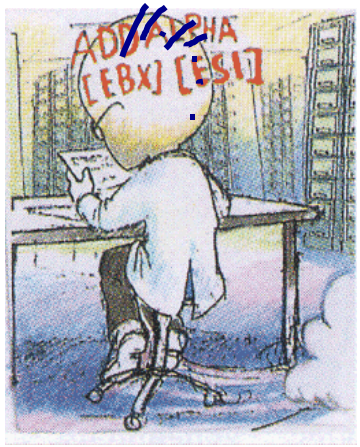
- **Pipelining – Grundprinzip**
  - **Beispiel Waschvorgang:** Vergleich sequentielle vs. Pipeline-Verarbeitung



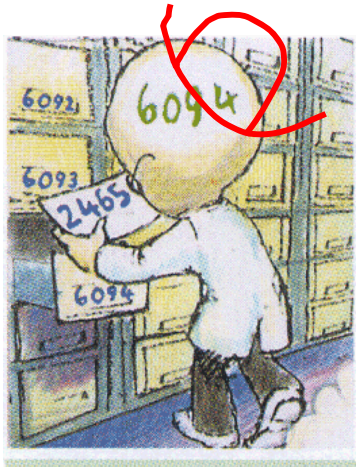
5.2 Pipelining des Maschinenbefehlszyklus

- **Pipelining – Grundprinzip**
  - **Beispiel Befehlsabarbeitung**

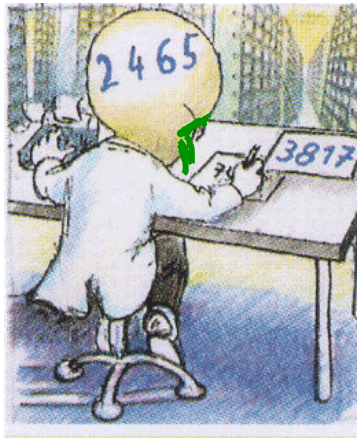
Befehl holen



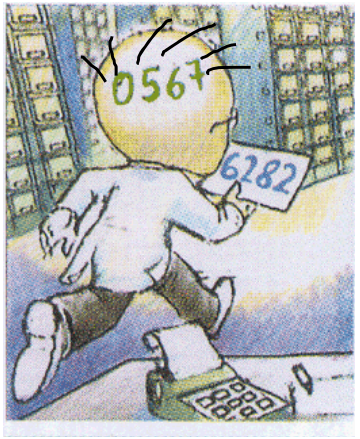
Befehl dekodieren



Operanden holen



Operation ausführen



Ergebnis speichern

5.2 Pipelining des Maschinenbefehlszyklus

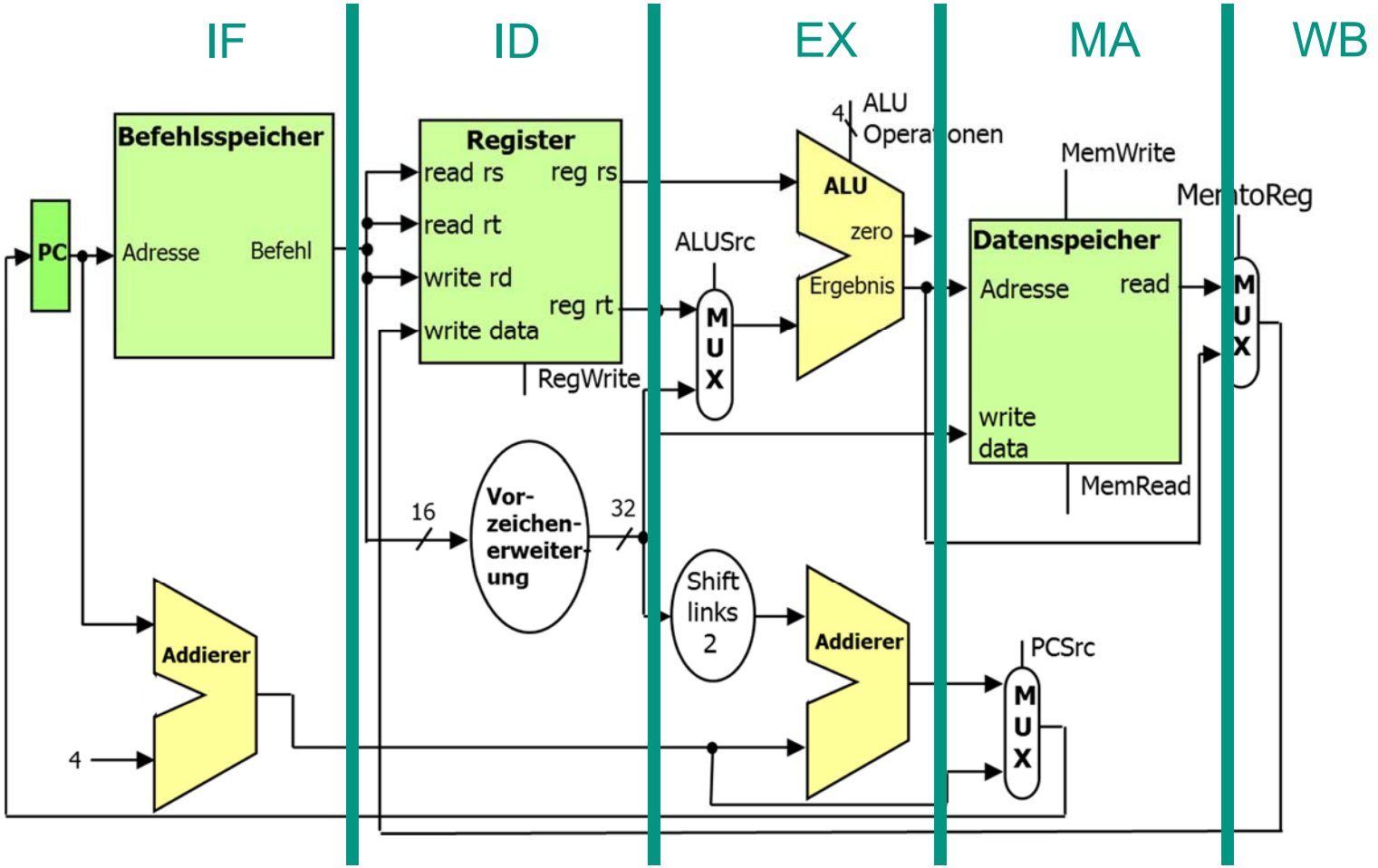
- **Pipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
  - Zerlegung der Ausführung einer Maschinenoperation in Teilphasen, die dann von hintereinander geschalteten Verarbeitungseinheiten taktsynchron bearbeitet werden, wobei jede Einheit genau eine spezielle Teiloperation ausführt.

  - Pipeline:
    - Gesamtheit der Verarbeitungseinheiten

Teiloperationen:
IF: Befehl holen
ID: Befehl dekodieren / Operanden bereitstellen
EX: Befehl ausführen
MA: Speicherzugriff
WB: Zurückschreiben

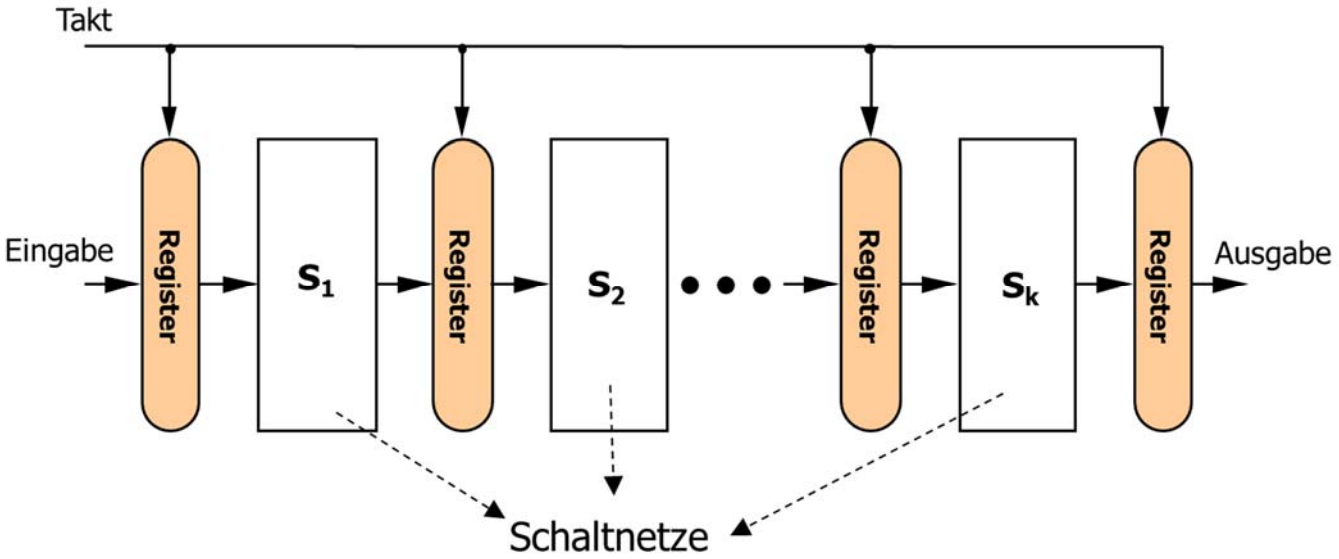
5.2 Pipelining des Maschinenbefehlszyklus

- **Pipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
  - Pipeline: Gesamtheit der Verarbeitungseinheiten



5.2 Pipelining des Maschinenbefehlszyklus

- **Pipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
  - k-stufige Befehlspipeline



  - Pipeline-Stufen durch Pipelineregister getrennt: Taktsynchrone Abarbeitung
  - Verzögerungszeiten:
    - der Schaltnetze: $t_i \quad (i = 1,2, \dots k)$
    - der Pipeline-Register: t_{reg}
  - Länge des Taktzyklus: $t = \max\{t_1, t_2, \dots, t_k\} + t_{reg}$