

# Pipelining: Fließbandverarbeitung

**Pipelining has become the accepted implementation method for computers of virtually every class, while the serial one-at-a-time execution model is fading into history - existing only in beginning computer architecture textbooks.**

**Today's computer architect thinks in terms of architectures that map well onto a pipeline, and implementors begin a design by determining the overall pipeline structure.**

**Shlomo Weiss, James E. Smith : POWER and PowerPC, 1994**



**Möglichkeiten der Parallelarbeit**

<p><b>Jeder Arbeiter baut ein vollständiges Auto</b></p>	<p><b>Organisation von Gruppen, die parallel vollständige Autos bauen</b></p>	<p><b>Organisation eines Fließbands, an dem in jedem Abschnitt eine spezielle Arbeit ausgeführt wird. An allen Abschnitten wird gleichzeitig gearbeitet</b></p>	<p><b>Organisations-Struktur</b></p>
<p><b>Hoher Grad an unterschiedlichen Fertigkeiten erforderlich</b></p> <p><b>geringe Spezialisierung</b></p>	<p><b>Mittlerer Grad an Fertigkeiten erforderlich</b></p> <p><b>mittlere Spezialisierung</b></p>	<p><b>Geringer Grad an unterschiedlichen Fähigkeiten erforderlich</b></p> <p><b>hohe Spezialisierung</b></p>	<p><b>Spezialisierungs-Grad</b></p>
<p><b>Robust gegen Ausfälle und unterschiedlich Arbeitsgeschwindigkeiten</b></p> <p><b>lokale Koordination/ globale Versorgung</b></p>	<p><b>Robust gegen Ausfälle und unterschiedlich Arbeitsgeschwindigkeiten</b></p> <p><b>lokale Koordination/ globale Versorgung</b></p>	<p><b>Empfindlich gegen Ausfälle und untersch. Arbeitsgeschw.</b></p> <p><b>globale Koordination/ lokale Versorgung</b></p>	<p><b>Robustheit und Aufwand zur Koordinierung der Arbeit</b></p>
<p><b>Erhöhung der Arbeitsleistung des Einzelnen</b></p>	<p><b>Erhöhung der Arbeitsleistung des Einzelnen</b></p> <p><b>Erhöhung der Anzahl der Mitarbeiter</b></p>	<p><b>Erhöhung der Arbeitsleistung des Einzelnen</b></p>	<p>↑ <b>Verminderung der Verzögerung</b></p>
<p><b>Erhöhung der Anzahl der Mitarbeiter</b></p>	<p><b>Erhöhung der Anzahl der Teams</b></p>	<p><b>Erhöhung der Anzahl der Fließbänder</b></p> <p><b>Erhöhung der Anzahl der Mitarbeiter durch die Einführung neuer Fließbandabschnitte</b></p>	<p>↑ <b>Erhöhung der Arbeitsleistung/ Zeit</b></p> <p>↑ <b>Erhöhung des Durchsatzes</b></p>

## Grundeigenschaften einer Pipeline:

- **Gesamtarbeitsablauf wird in kleinere Einzelschritte unterteilt**
- **Jeder Einzelschritt erledigt eine einzige spezialisierte Aufgabe**
- **Jeder Einzelschritt braucht dieselbe Zeit**
- **Jeder Einzelschritt hängt von der erfolgreichen Bearbeitung des vorhergehenden Schrittes ab**
  
- **Die Anzahl der Abschnitte in der Pipeline bestimmt den Grad der Parallelität**
- **Die Durchlaufzeit durch eine Pipeline ist weitgehend unabhängig von der Anzahl der Abschnitte**
- **Der Durchsatz der Pipeline wächst mit der Anzahl der Abschnitte**



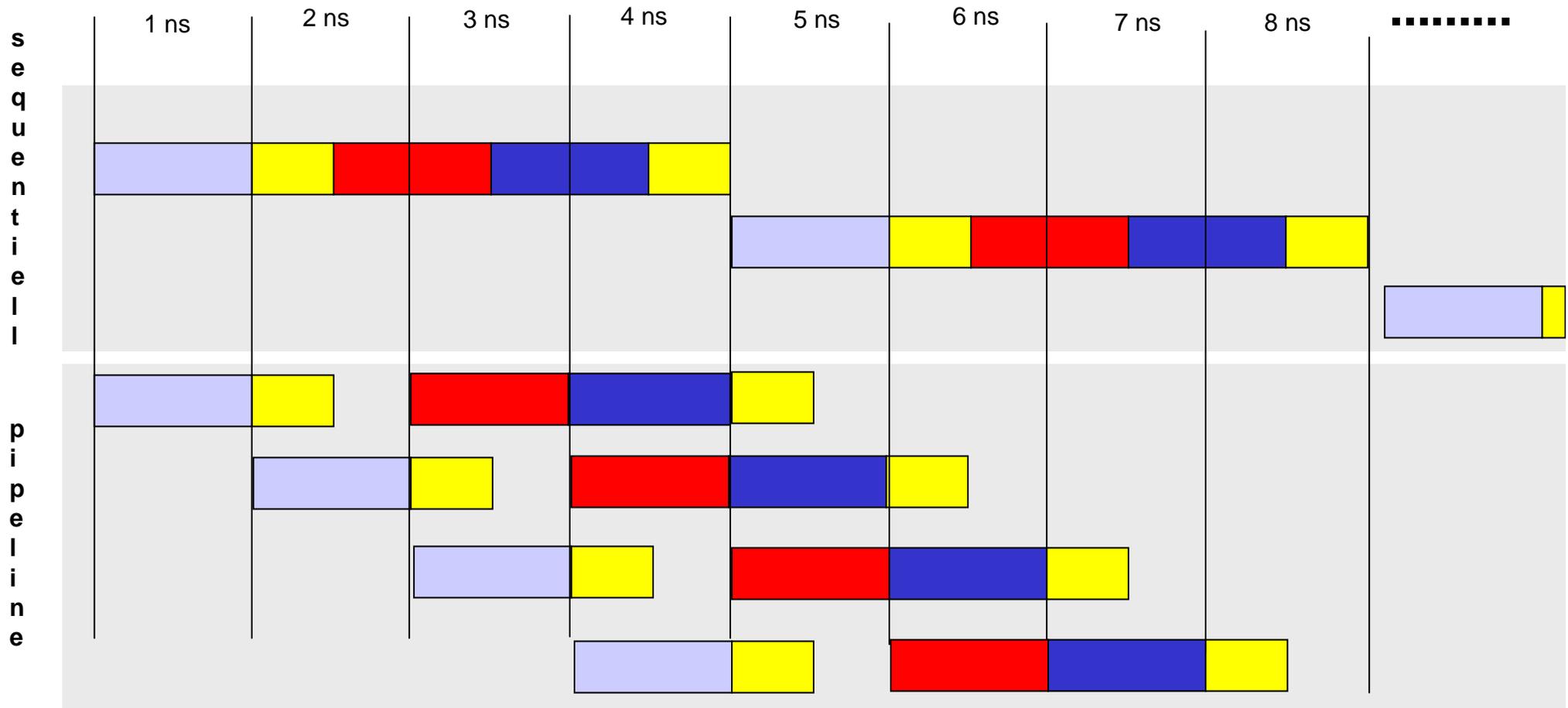
## Hauptphasen der Befehlsausführung

- **IF (Instruction Fetch) Befehlsholphase** : Die nächste Instruktion wird unter Benutzung des Programmzählers (PC) geladen.
- **ID (Instruction Decode and Operand Fetching) Decodierungs - und Operanden Auswertungsphase** : Der **OPCODE** und die Operanden in der Instruktion werden ausgewertet und die entsprechenden Kontrollsignale erzeugt. Register werden selektiert und die entsprechenden Operanden gelesen.
- **EX (Instruction Execution) Ausführungsphase** : Die durch den **OPCODE** spezifizierte Operation wird ausgeführt. Bei Instruktionen, die einen Speicherzugriff erfordern, wird in dieser Phase die effektive Adresse berechnet.
- **ME (Memory Access) Speicherzugriffsphase** : Daten werden aus dem Speicher geladen oder in den Speicher geschrieben. Bei RISCs ist meist ein Datencache vorhanden.
- **WB (Write Back) Zurückschreiben der Ergebnisse in das Dest. Reg.** : Das Resultat einer Berechnung wird in ein Zielregister geschrieben.



Instruction  
Fetch REG ALU Data  
Fetch REG

**Bearbeitungsabschnitte**



# Struktur des Datenpfads

die folgende Architektur der Pipeline ist an die MIPS-Architektur angelehnt, die im Buch:

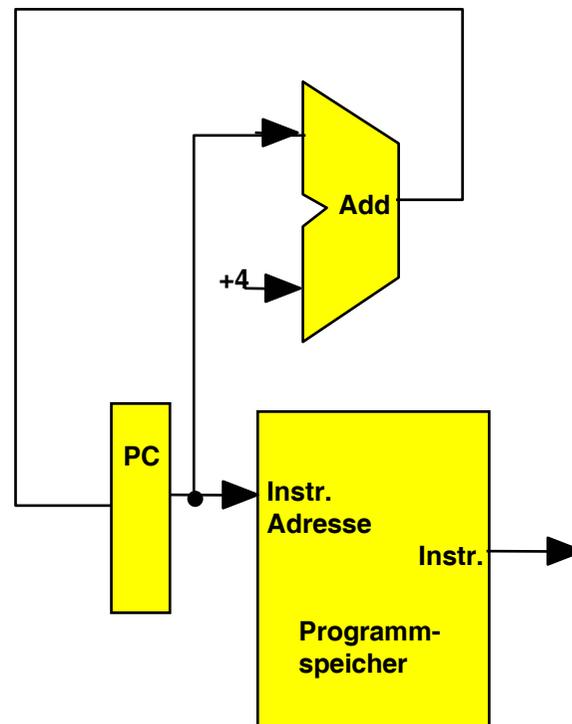
**David A. Patterson, John L. Hennessy**  
**Computer Organization & Design – The Hardware/Software Interface**  
**Morgan Kaufmann Publishers, San Mateo, CA, 1994**

beschrieben ist.



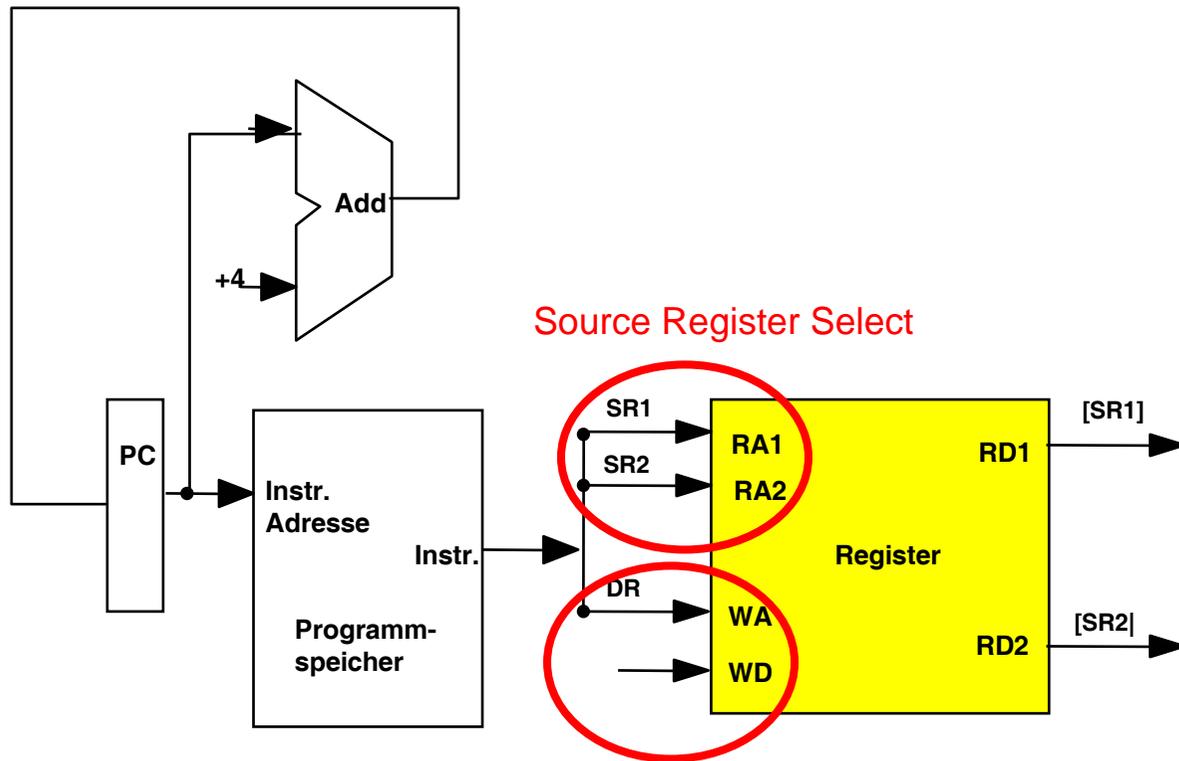
## Instruction Fetch (IF)

- Adressierung des Programmspeichers
- Holen der Instruktion



# Dekodierungsphase (ID):

## Auswahl der Register



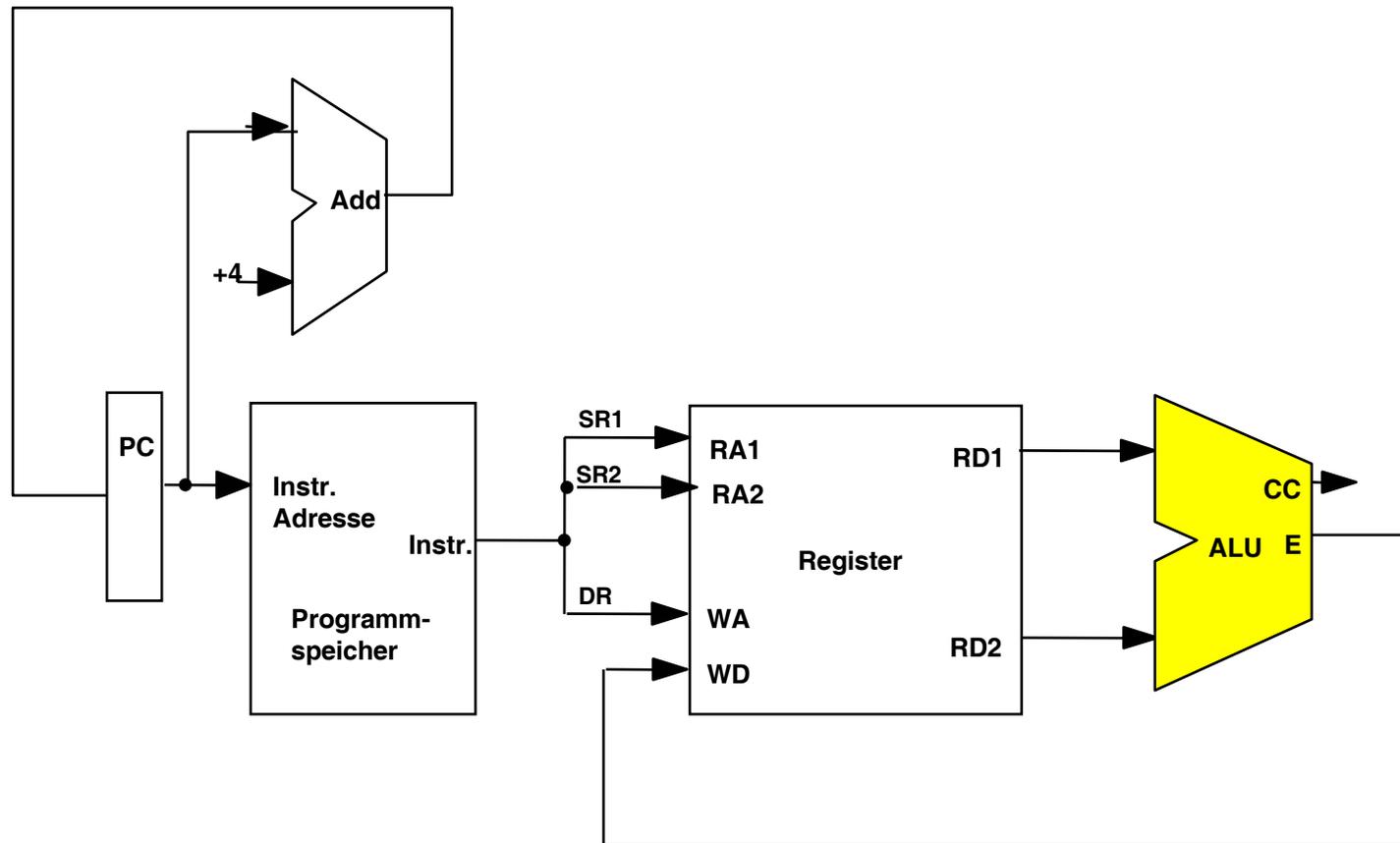
Source Register Select

Destination Register Select (WA)  
& Write Back (WD)

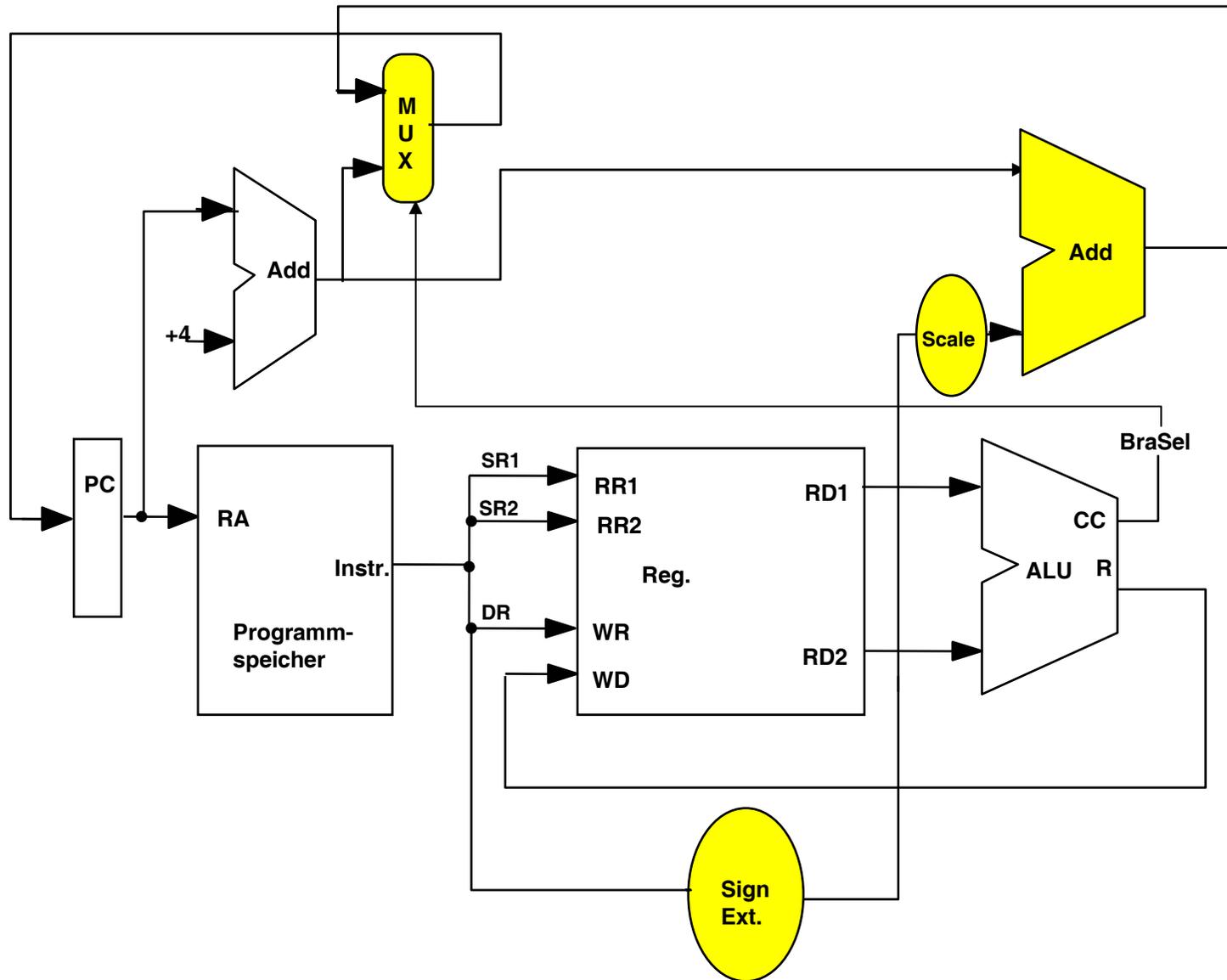


## Ausführungsphase (EX)

- Auswahl der ALU-Funktion
- Ausführung

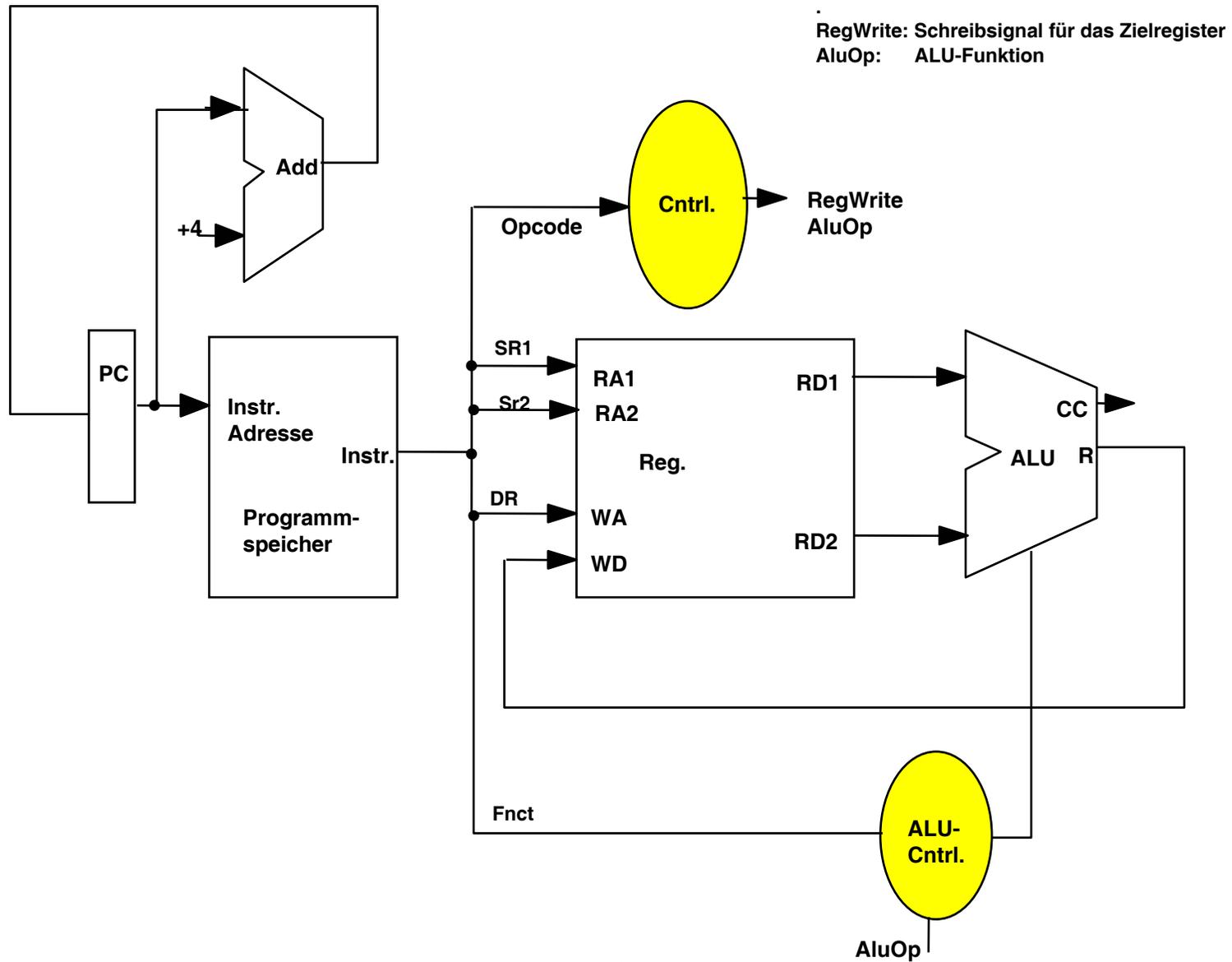


# Einbeziehung der Sprungbefehle



branch on equal	beq \$1,\$2,100	if (\$1 == \$2) go to PC+4+100	Equal test; PC relative branch
branch on not eq.	bne \$1,\$2,100	if (\$1 != \$2) go to PC+4+100	Not equal test; PC relative

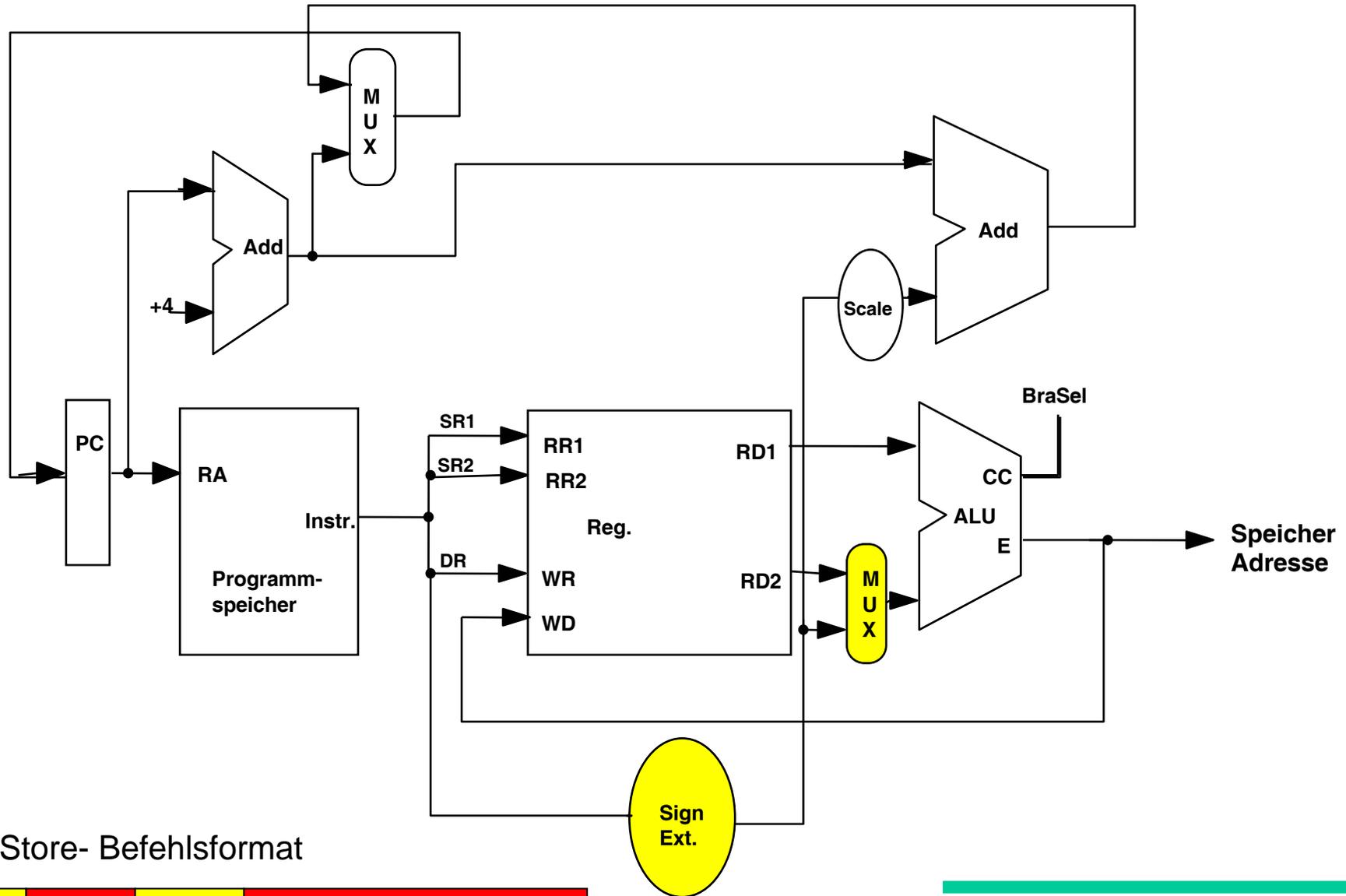
# Einbeziehung der Kontrolle



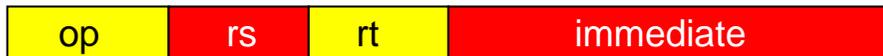
Befehlsformat:



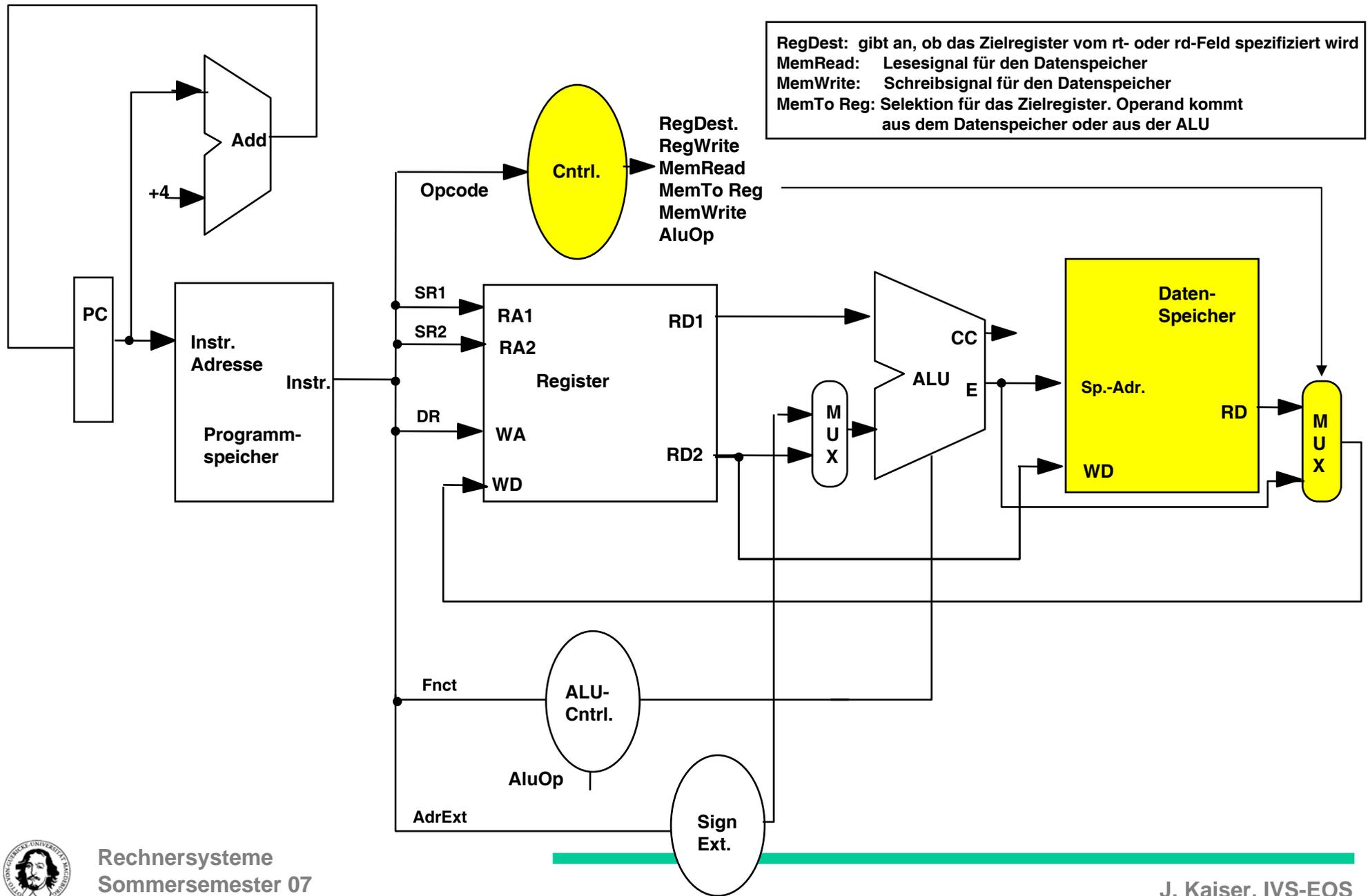
# Einbeziehung der Adreßberechnung für die Datenphase

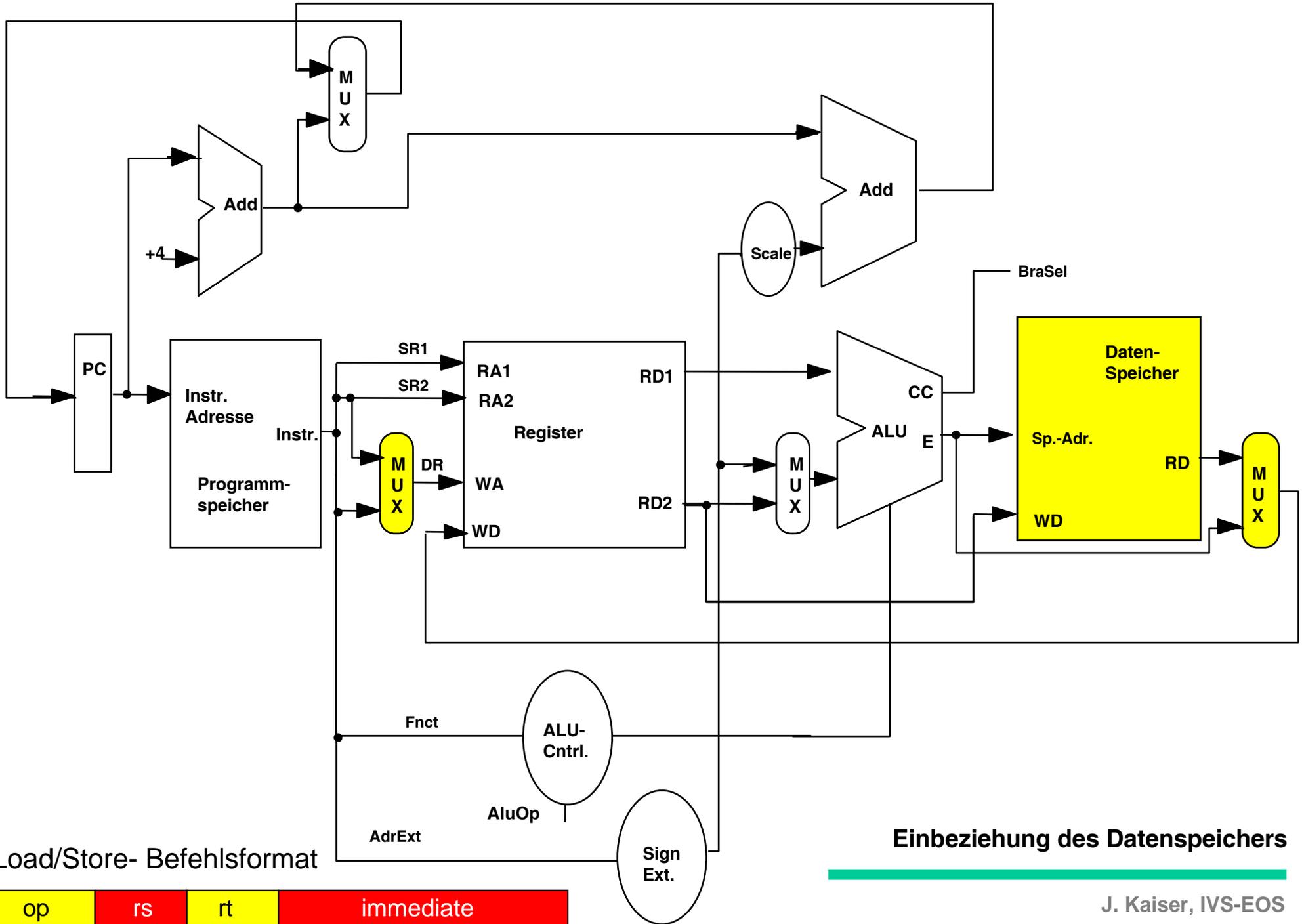


Load/Store- Befehlsformat

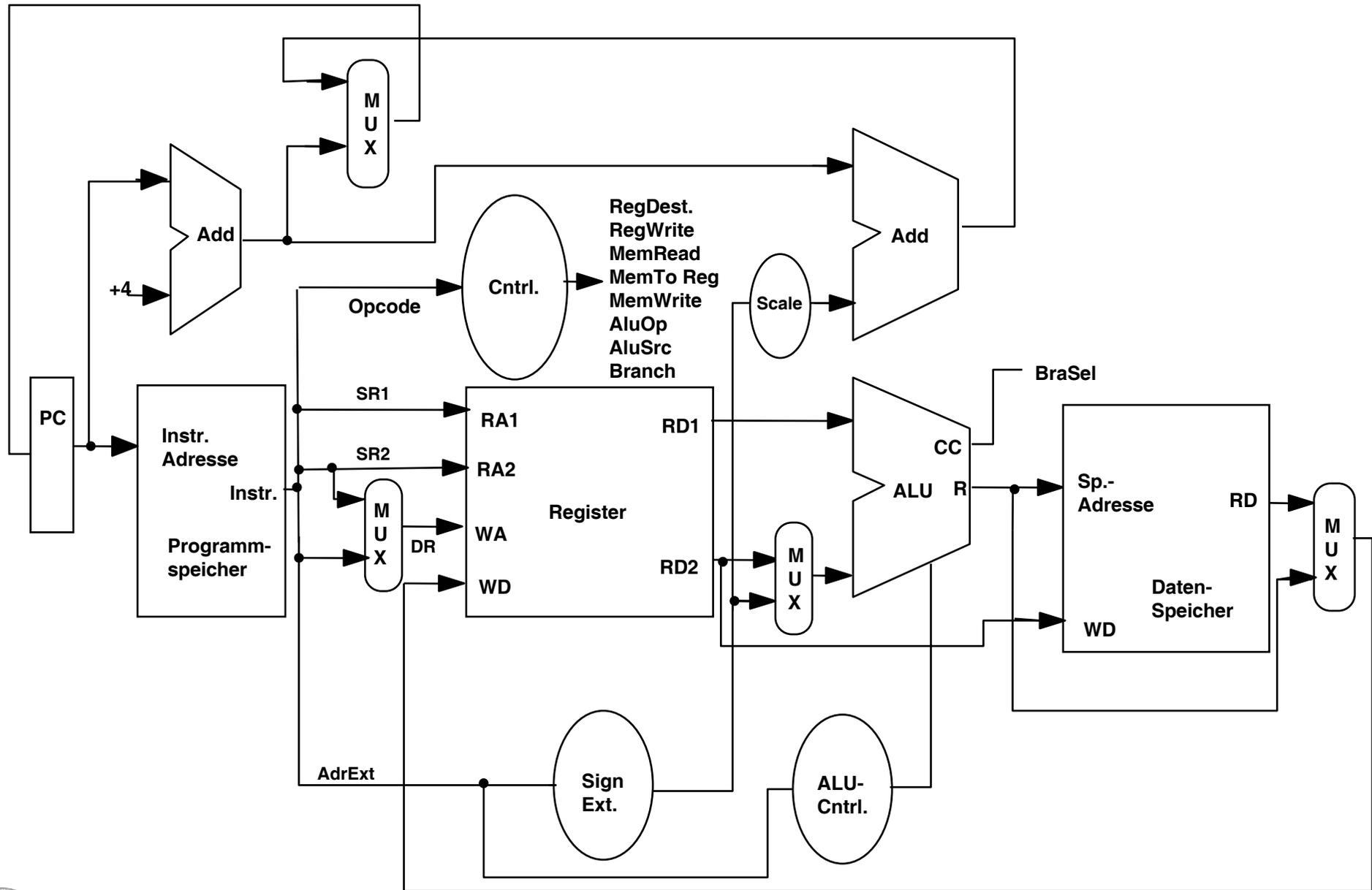


## Einbeziehung des Datenspeichers



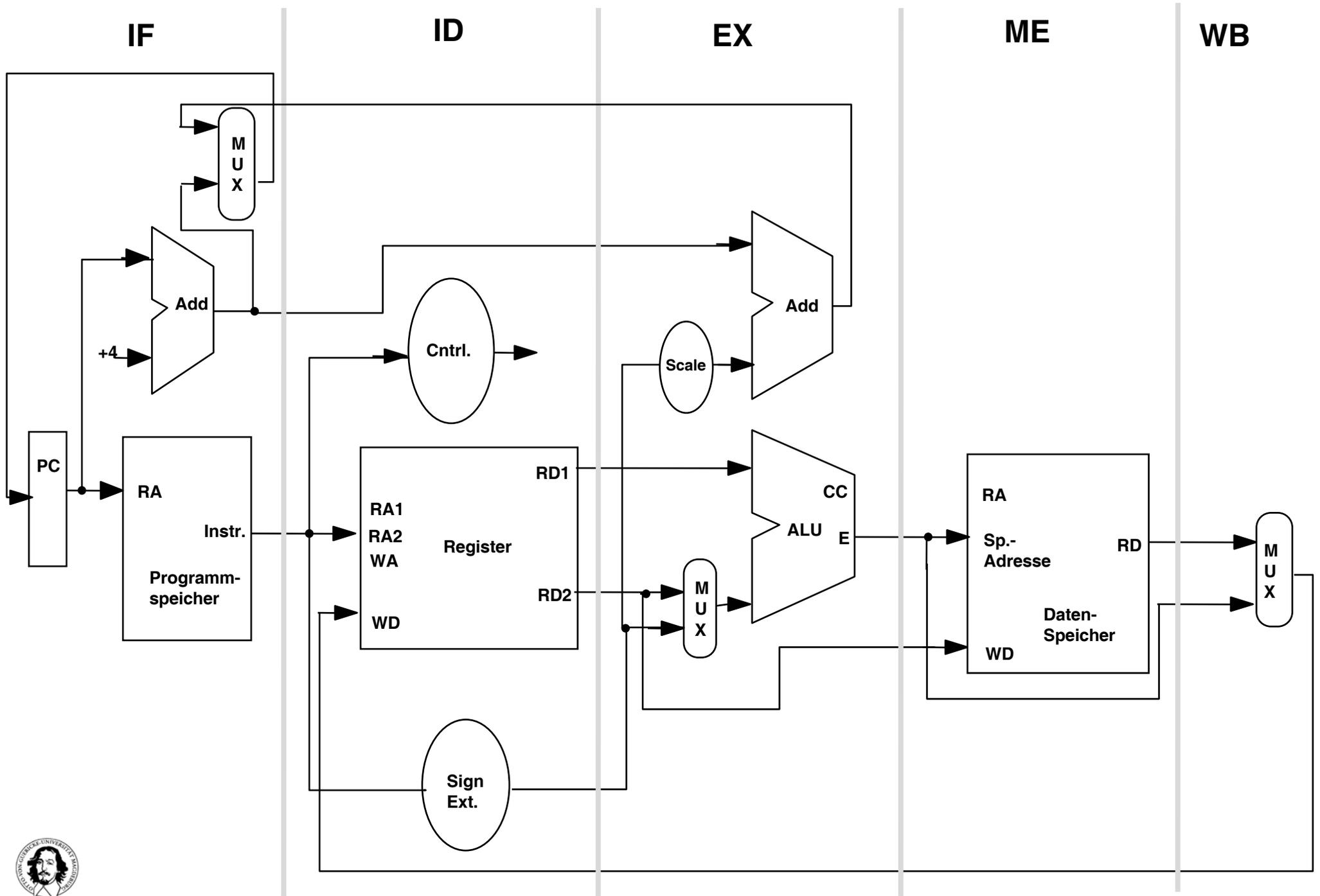


# Vollständiger Datenpfad



# Aufbau der Pipeline-Stufen





# Pipeline-Register

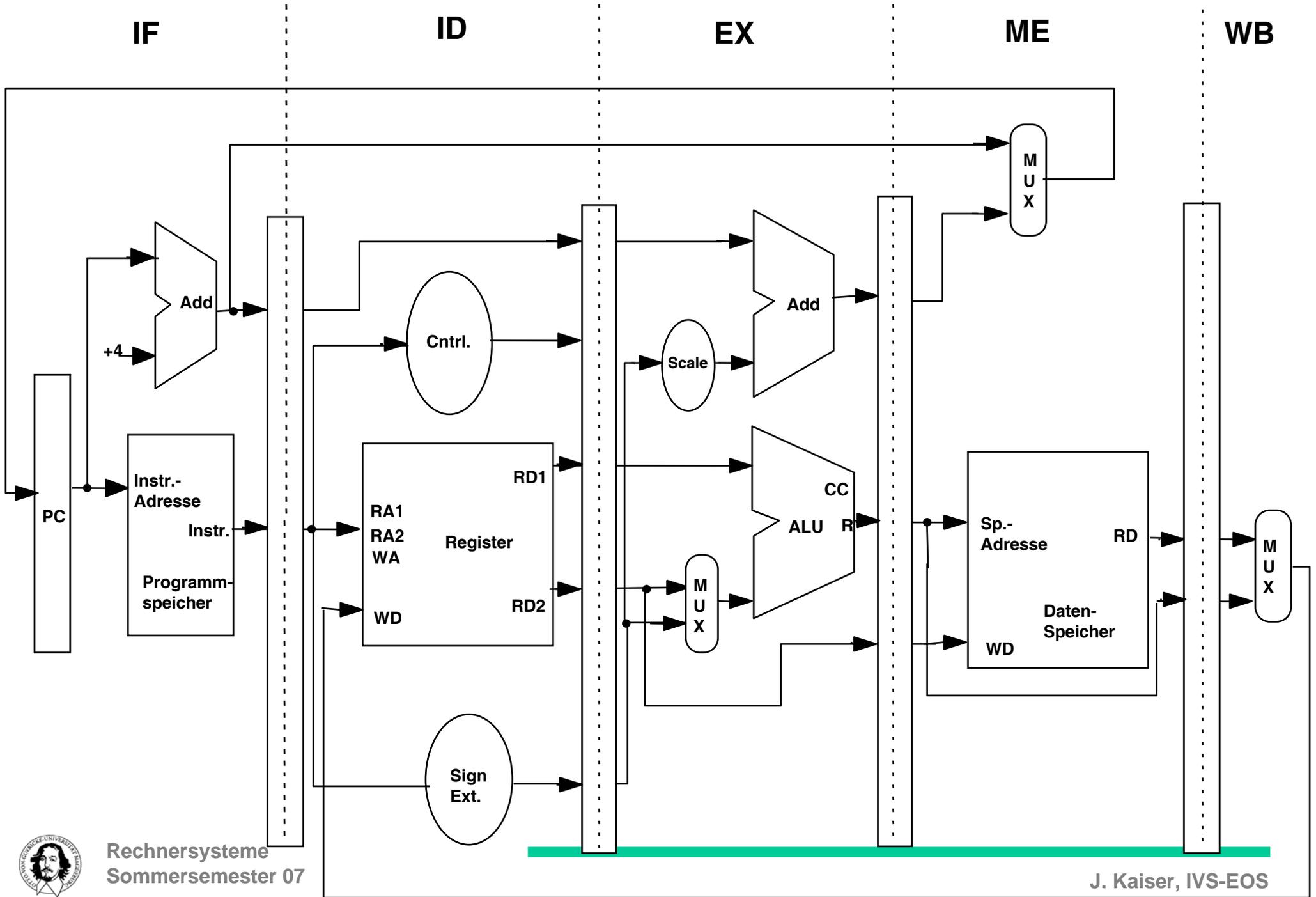
Während der Bearbeitung eines Einzelschritts in Stufe  $S$  der Pipeline müssen die Eingangssignale  $\{E_S\}$  in diese Stufe stabil sein. Die Eingangssignale  $\{E_S\}$  für Stufe  $S$  sind die Ausgangssignale  $\{A_{S-1}\}$  der Stufe  $S-1$ .

Da die vorhergehende Stufe  $S-1$  aber bereits mit der Bearbeitung des nächsten Auftrags beschäftigt ist, müssen alle Signale  $\{E_S\}$  zwischengespeichert werden.

Die Zwischenspeicherung entkoppelt zwei aufeinanderfolgende Pipeline-Stufen.

Zur Zwischenspeicherung werden sogenannte *Pipeline-Register* eingesetzt.





### Steuersignale für den Registersatz:

- **RegDst:** gibt an, ob das Zielregister durch das rd-Feld (bei R-Format Instr.) oder durch das rt-Feld (bei Load/Store-Instr.) spezifiziert wird
- **RegWrite:** Schreibsignal für das Zielregister

### Steuersignale für den Datenspeicher:

- **MemRead:** Lesesignal für den Datenspeicher
- **MemWrite:** Schreibsignal für den Datenspeicher

### Operandenselektion:

- **MemToReg:** Selektion für das Zielregister - Operand wird aus dem Speicher gelesen oder ist Resultat einer ALU-Operation

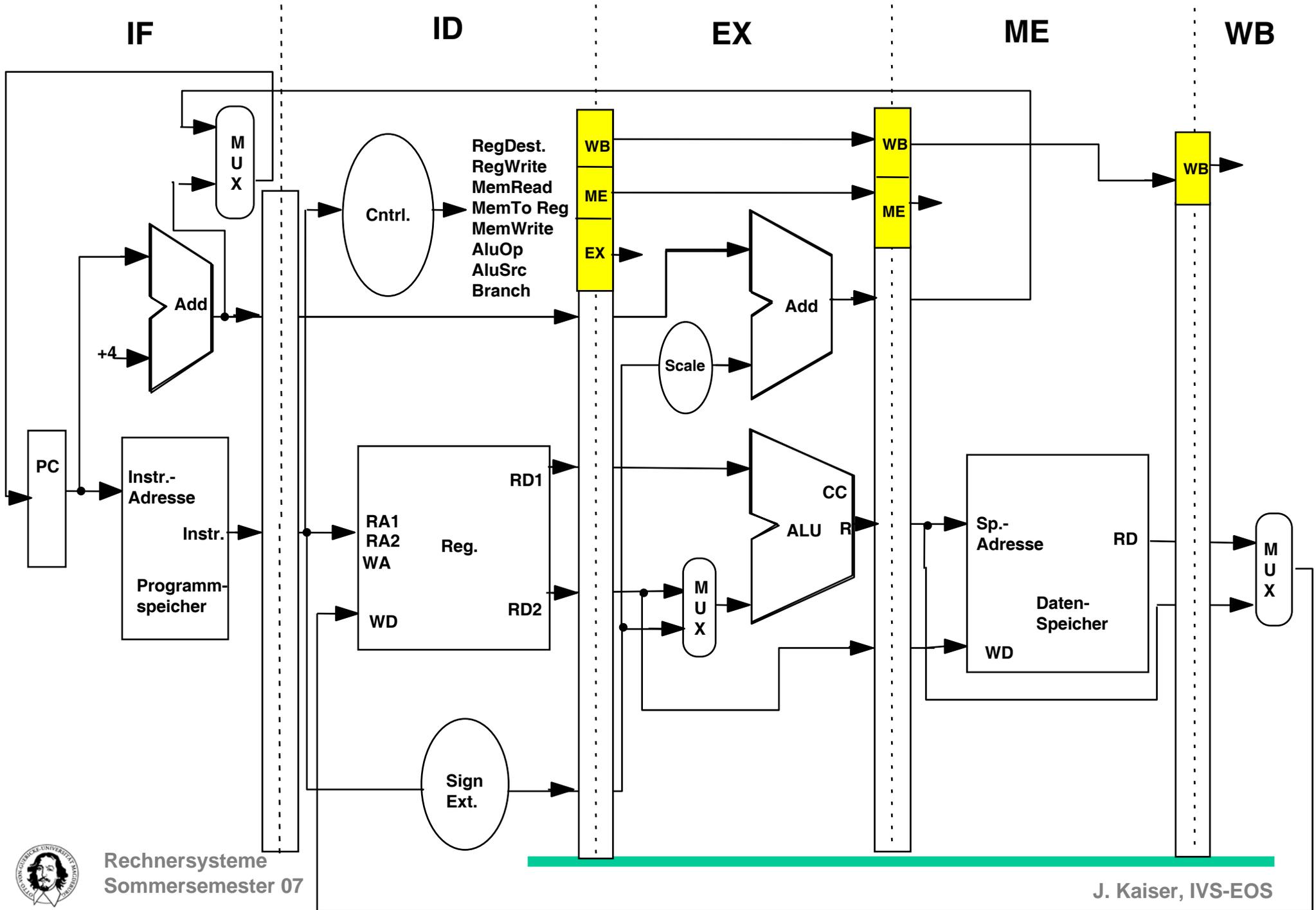
### Steuersignale für die ALU:

- **AluOp:** Auswahl einer ALU-Funktion
- **AluSrc:** Selektion des 2ten Operanden. Op kommt aus rt oder ist Sign ext. Immediate Op

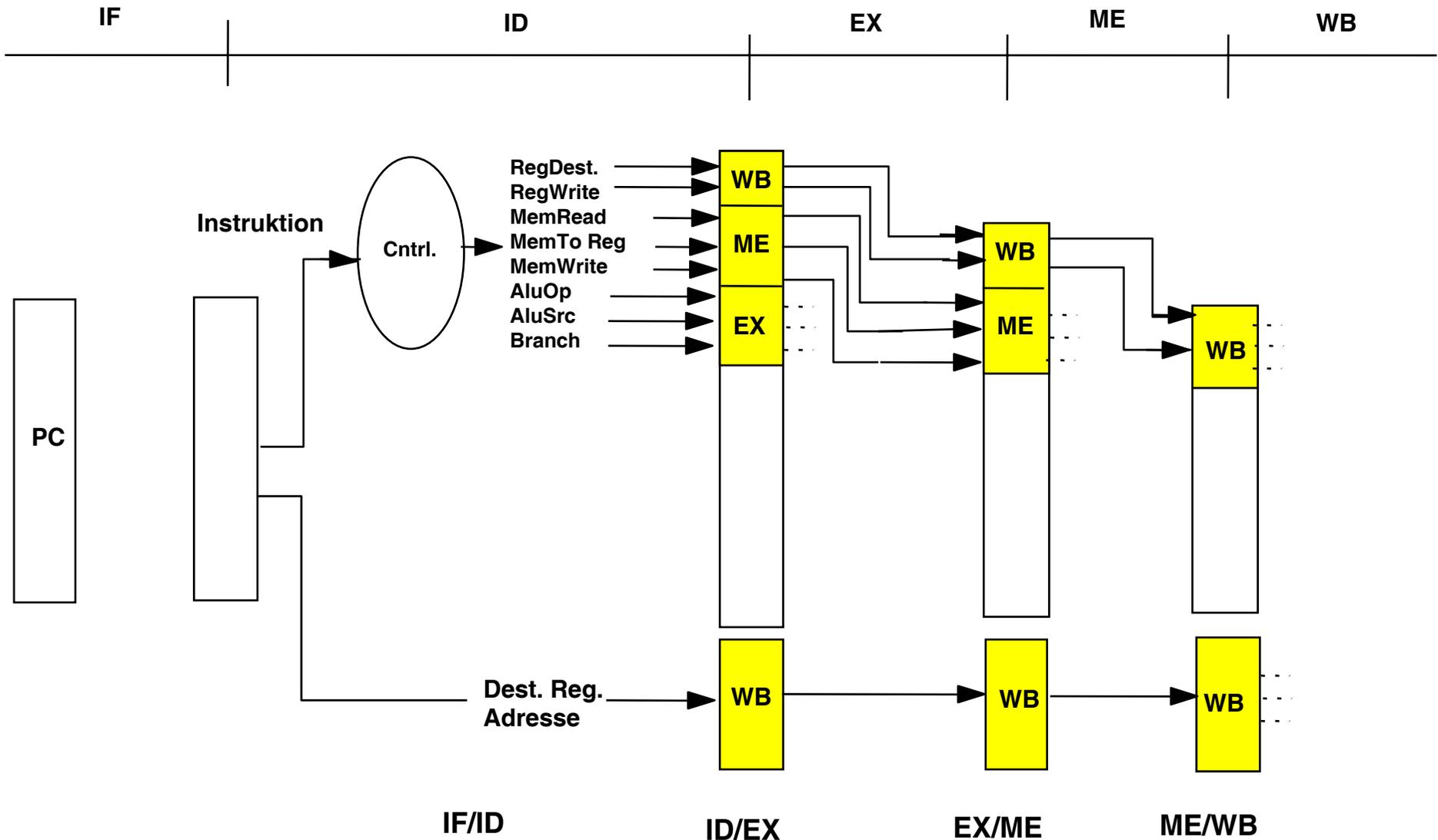
### Steuersignale für Sprünge:

- **Branch:** OPCODE spezifiziert einen Sprung, d.h. es wird eine (Sprung) Adreßberechnung durchgeführt.

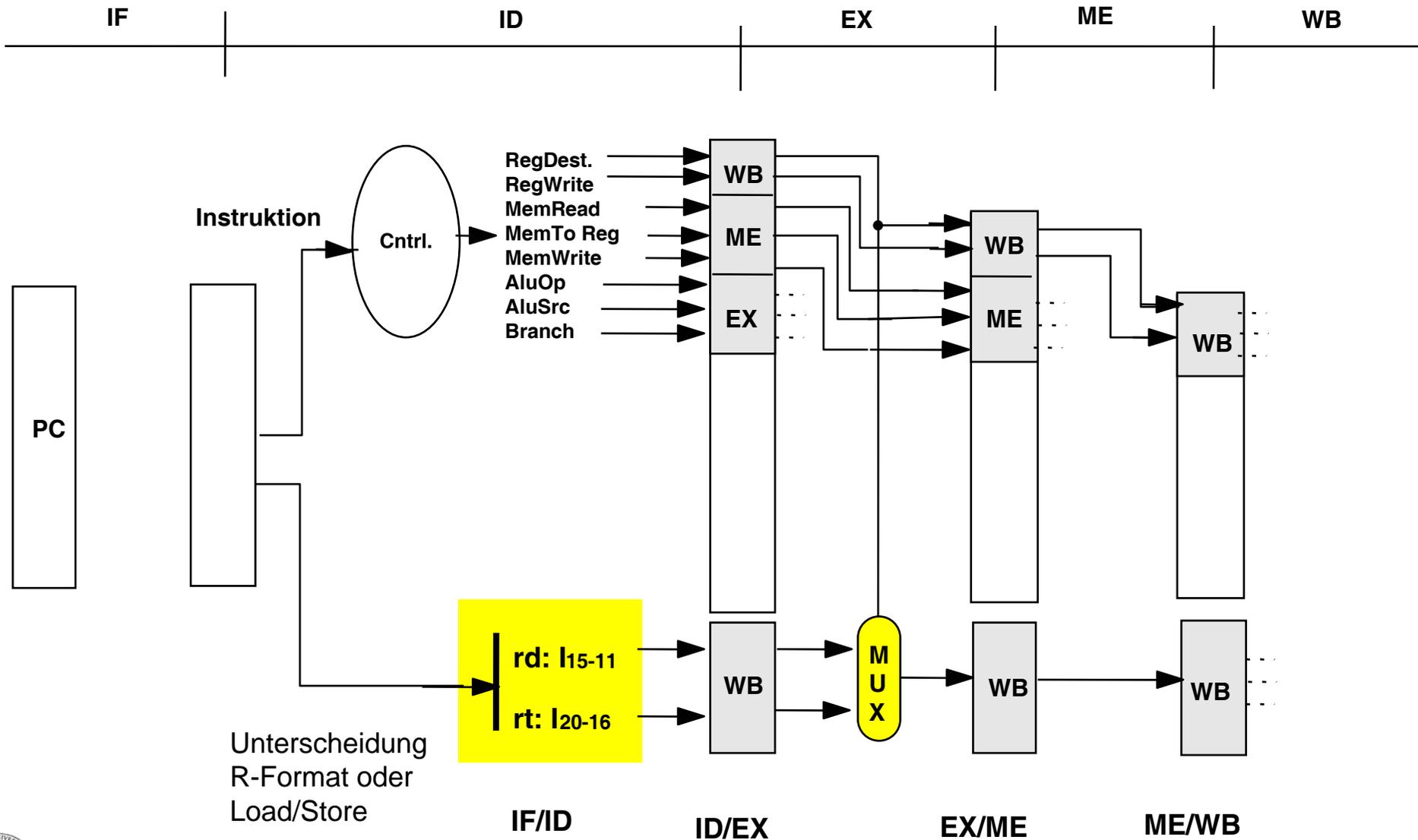




# Puffern der Kontrollsignale und der Adresse des Zielregisters

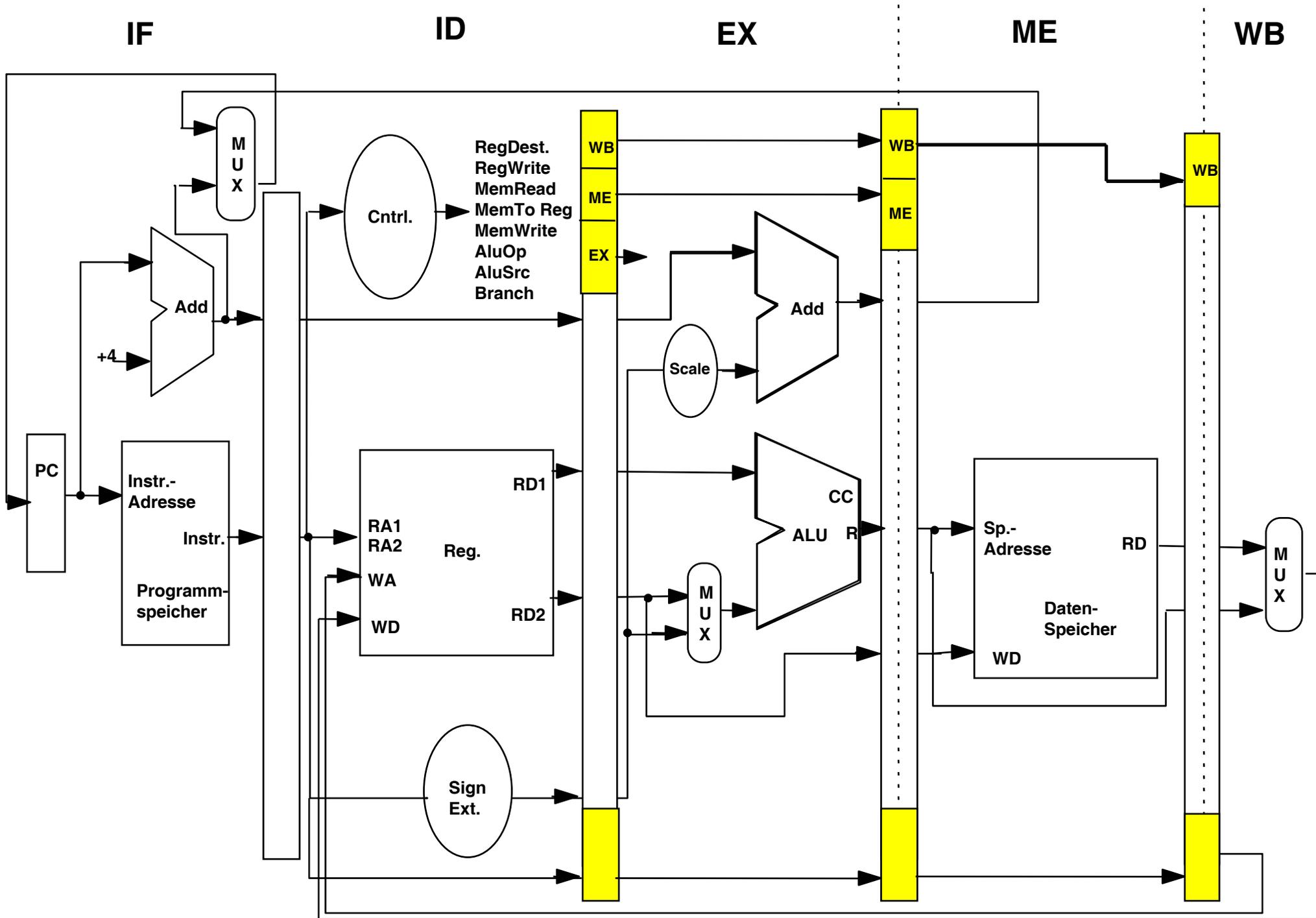


# Puffern der Kontrollsignale und der Adresse des Zielregisters



Unterscheidung  
R-Format oder  
Load/Store

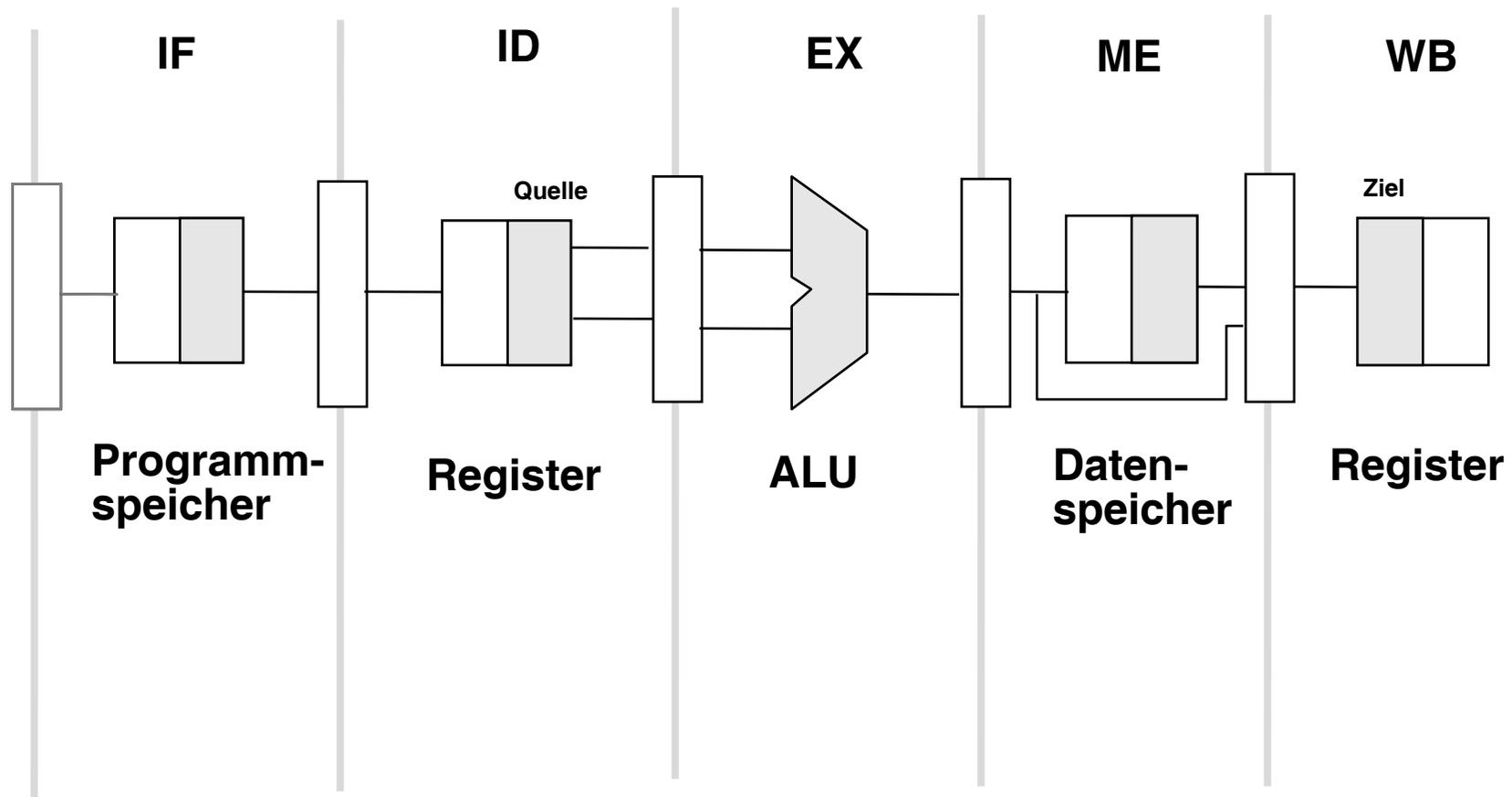


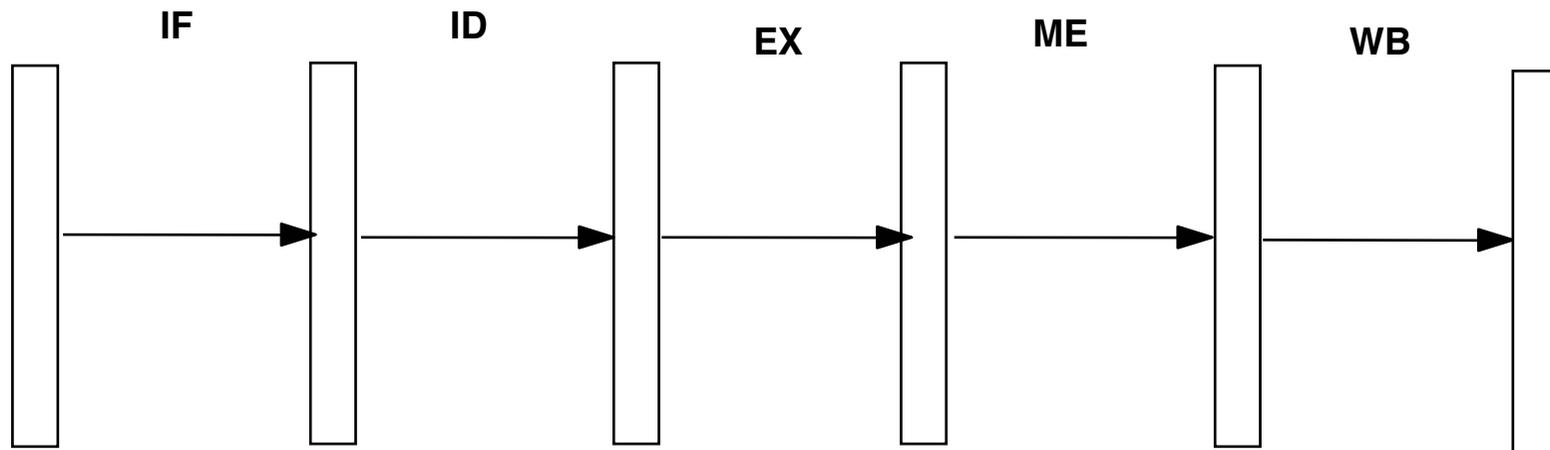


# Darstellung der Pipeline

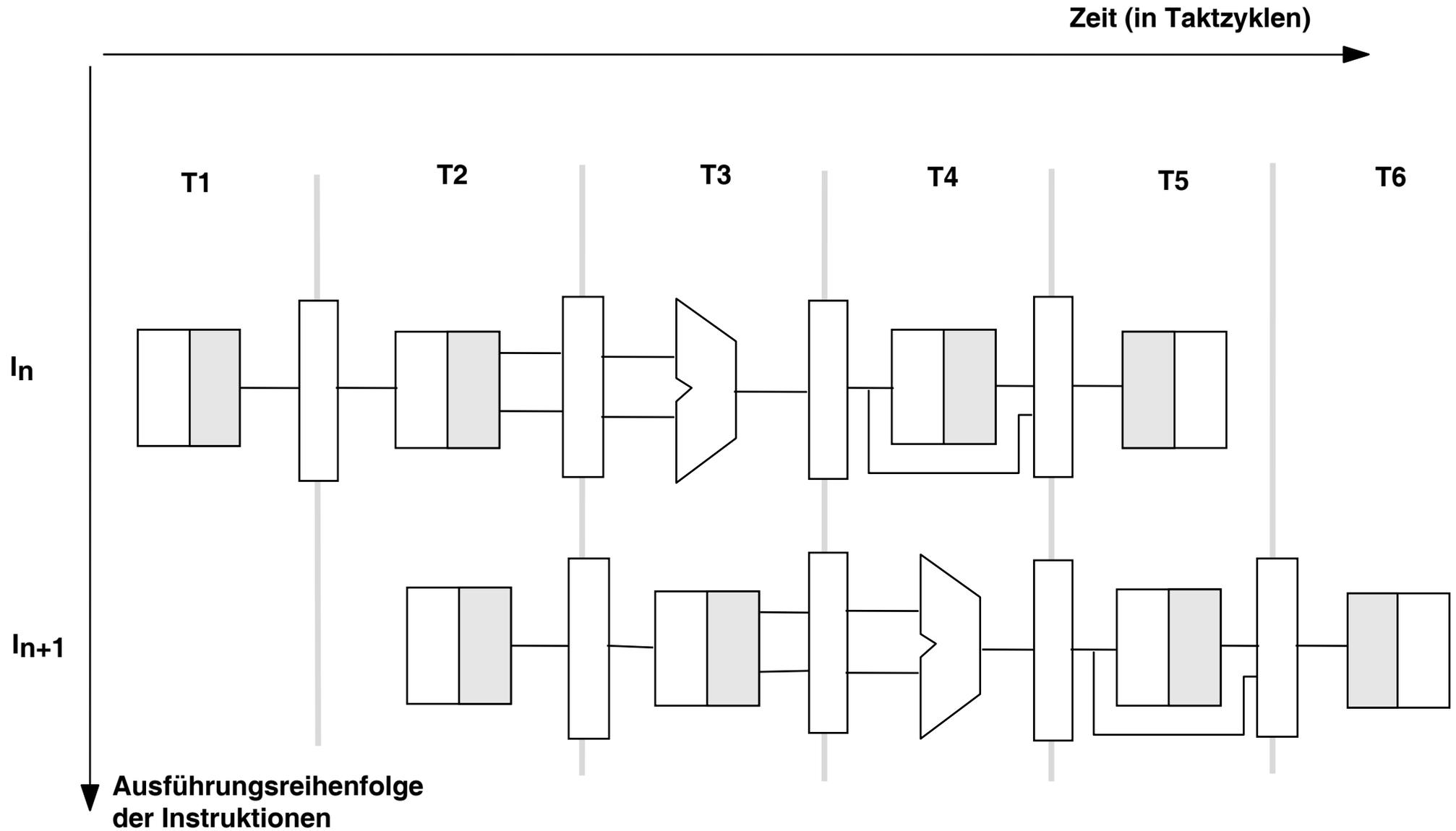


# Vereinfachung der Darstellung einer Pipeline

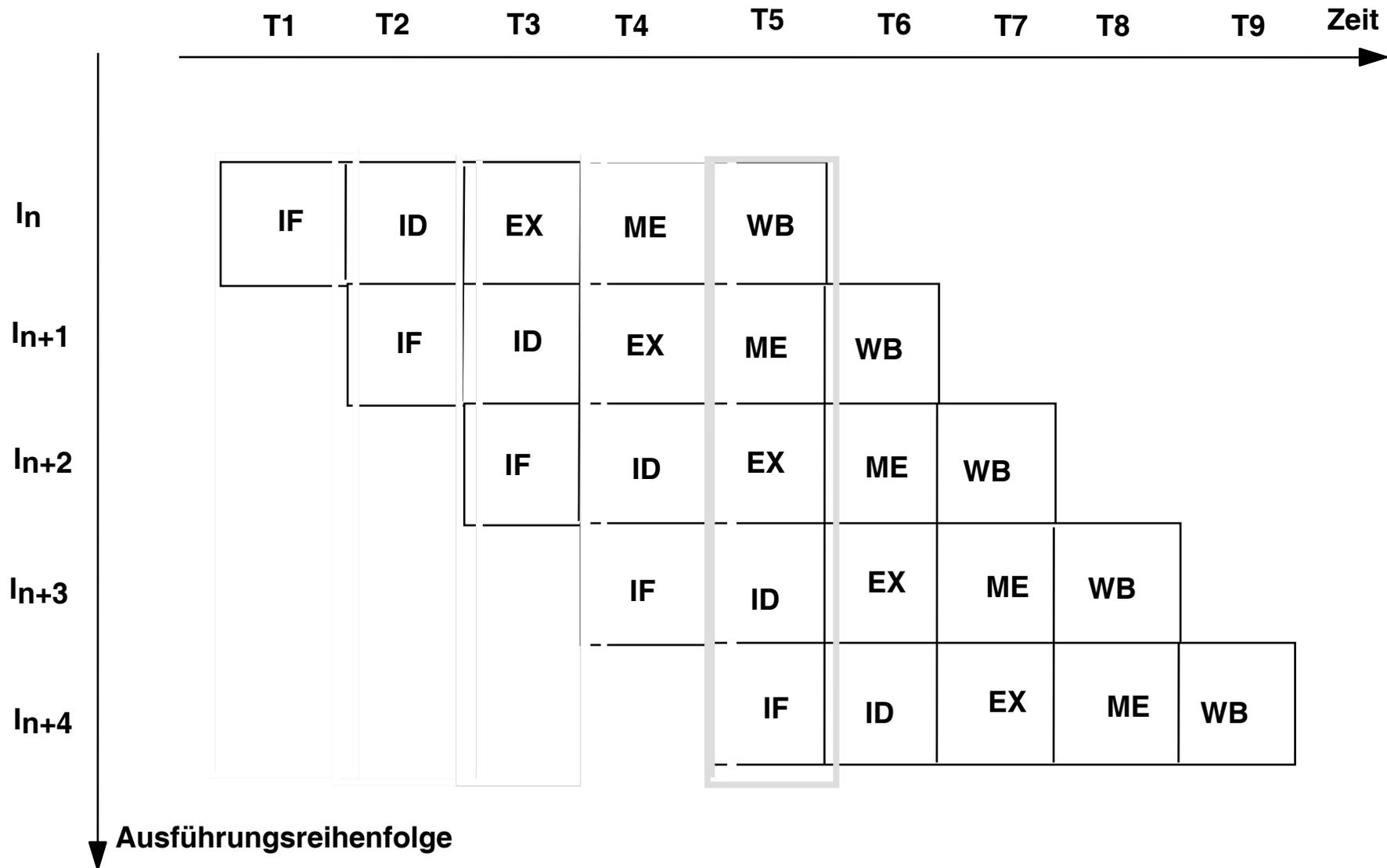




# Darstellung einer Pipeline in einem Multi-Zyklus Diagramm



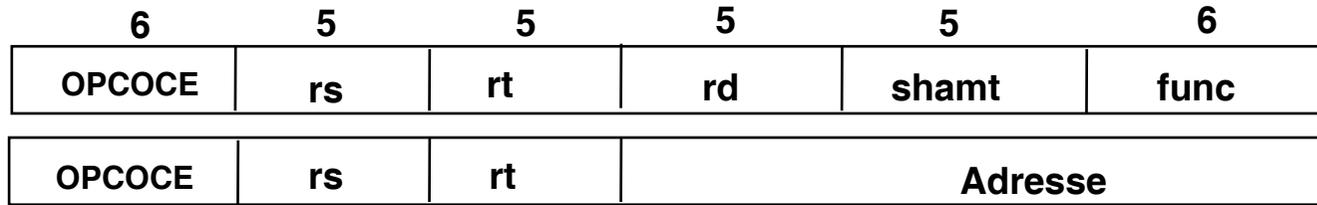
# Pipeline - Graphische Darstellung



# Pipeline-Hindernisse: Datenabhängigkeiten (Data Hazards)



**MIPS**  
**Befehlsformat**



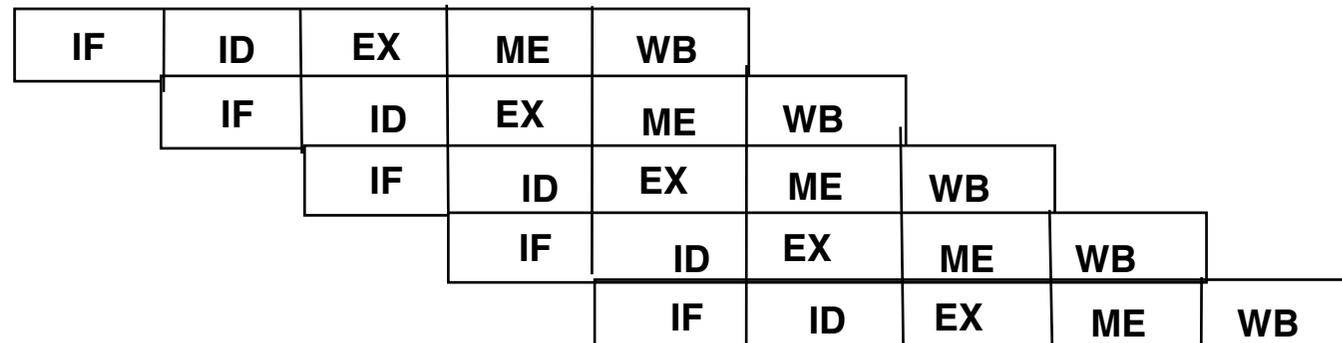
rs: 1.Op, rt: 2. Op, rd: Zielreg.

**Beisp.:**

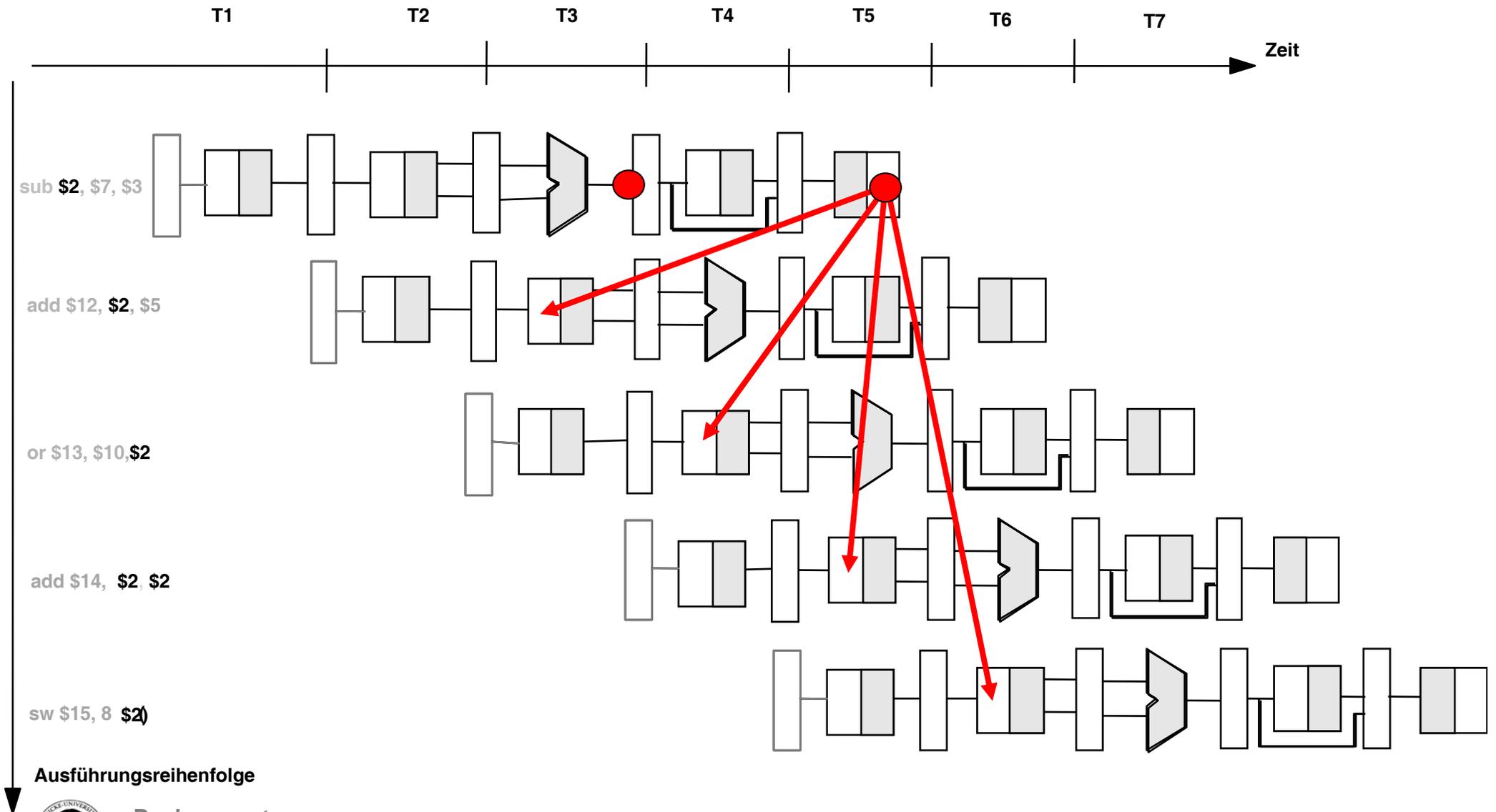
sub \$2, \$7, \$3  
and \$12, \$2, \$5  
or \$13, \$10, \$2  
add \$14, \$2, \$2  
sw \$15, 8(\$2)

Subtrahiere Inhalt von Reg 3 von Reg.7 und speichere das Resultat in Reg 2  
Logisches UND von Reg 5 und Reg 2; Resultat nach Reg 12  
Logisches ODER von Reg 2 und Reg 10; Resultat nach Reg 13  
Addiere den Inhalt von Reg 2 zum Inhalt von Reg 2; Resultat nach Reg 14  
Speichere Wort aus Register 15. Die EA wird gebildet aus dem Inhalt des (Adreß) Registers 2 mit dem Offset 8

sub **\$2**, \$7, \$3  
add \$12, **\$2**, \$5  
or \$13, \$10, **\$2**  
add \$14, **\$2**, **\$2**  
sw \$15, 8(**\$2**)



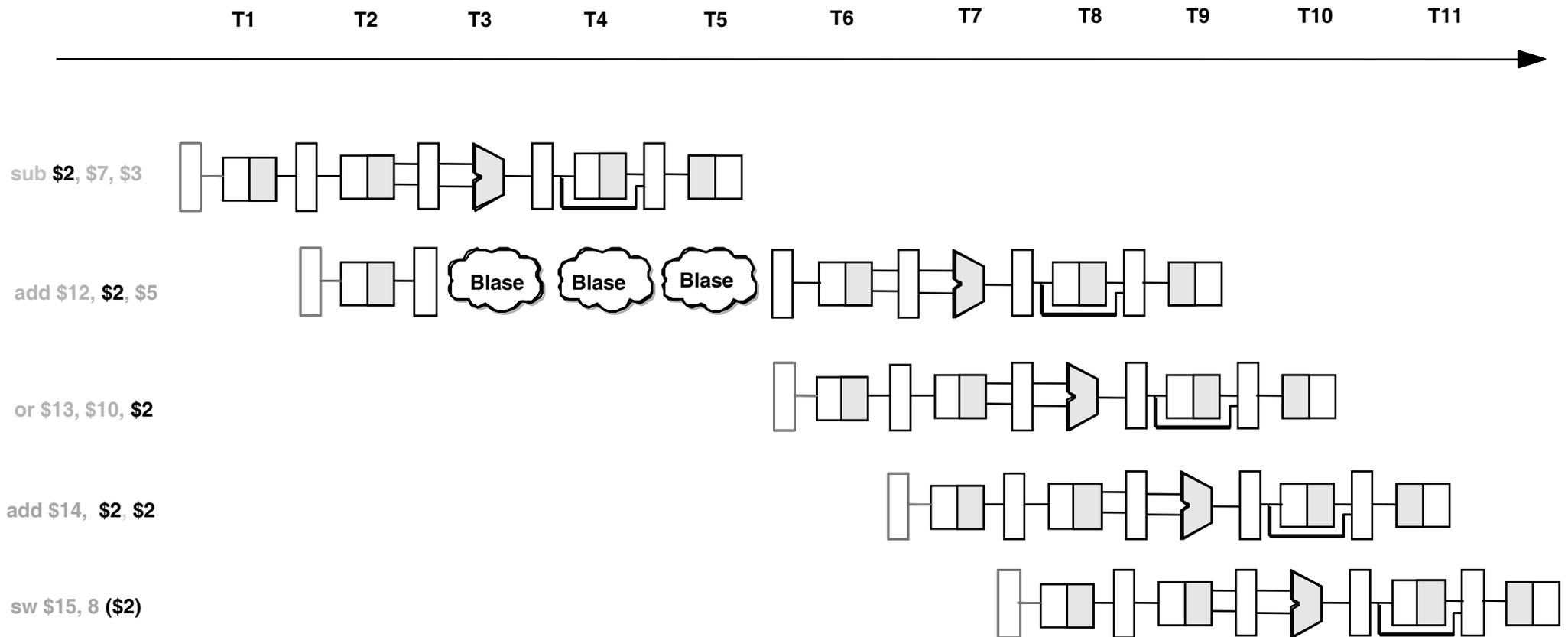
# Pipeline - Hindernisse: Datenabhängigkeiten



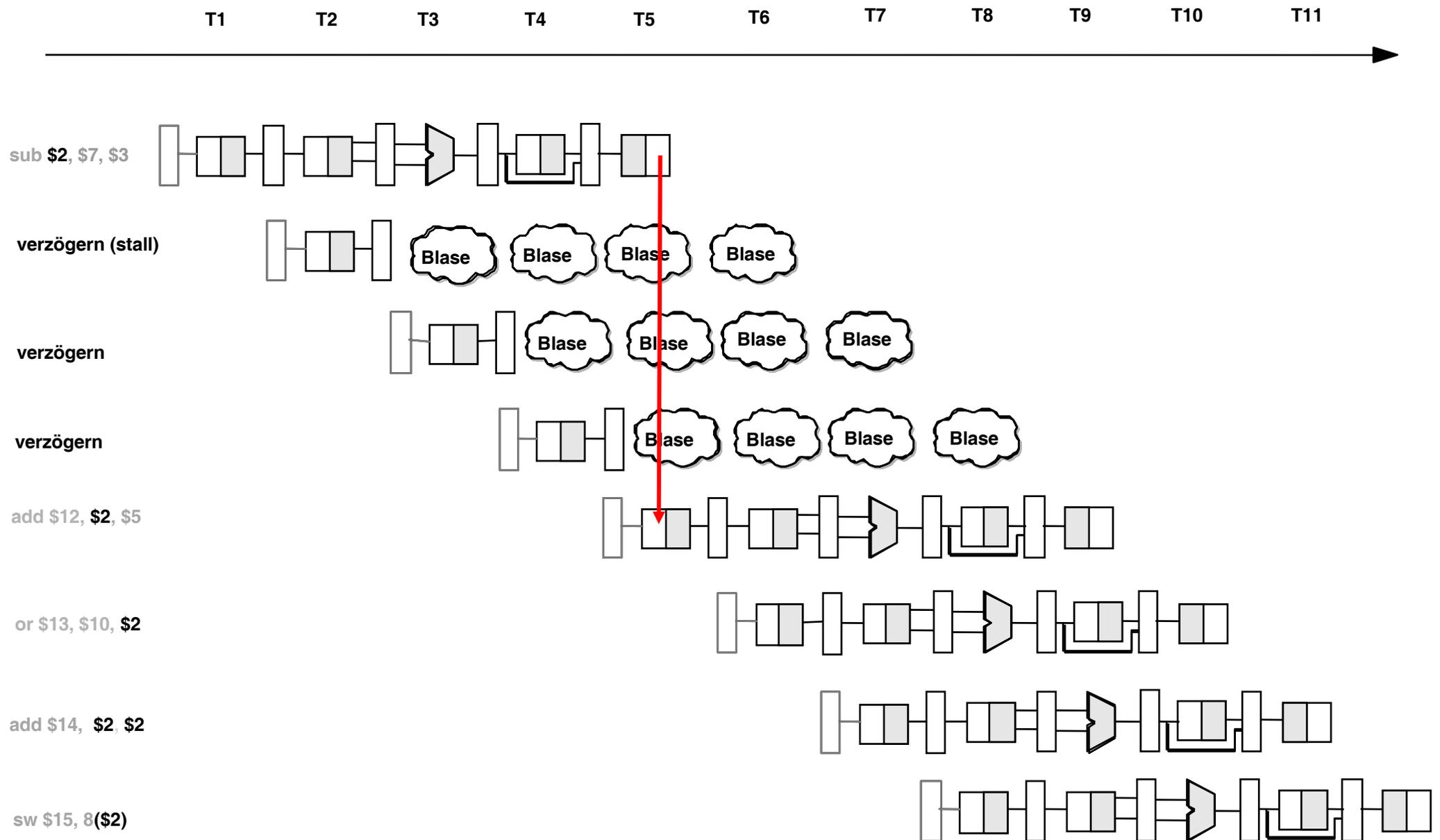
Ausführungsreihenfolge



# Pipeline - Hindernisse: Datenabhängigkeiten



# Pipeline - Hindernisse: Datenabhängigkeiten



### Bedingungen für Datenabhängigkeit:

- 1a. ID/EX.WriteReg = IF/ID.ReadReg1
- 1b. ID/EX.WriteReg = IF/ID.ReadReg2
- 2a. EX/ME.WriteReg = IF/ID.ReadReg1
- 2b. EX/ME.WriteReg = IF/ID.ReadReg2
- 3a. ME/WB.WriteReg = IF/ID.ReadReg1
- 3b. ME/WB.WriteReg = IF/ID.ReadReg2

- 1a. In der EX-Stufe befindet sich eine Instruktion, die ein Resultat in ein Zielregister schreibt, das von einer nachfolgenden Instruktion als 1. Quellregister definiert wird.
- 1b. In der EX-Stufe befindet sich eine Instruktion, die ein Resultat in ein Zielregister schreibt, das von einer nachfolgenden Instruktion als 2. Quellregister definiert wird.
- 2a. In der ME-Stufe befindet sich eine Instruktion, die ein Resultat in ein Zielregister schreibt, das von einer nachfolgenden Instruktion als 1. Quellregister definiert wird.
- 2b. In der ME-Stufe befindet sich eine Instruktion, die ein Resultat in ein Zielregister schreibt, das von einer nachfolgenden Instruktion als 1. Quellregister definiert wird.
- 3a. In der WB-Stufe befindet sich eine Instruktion, die ein Resultat in ein Zielregister schreibt, das von einer nachfolgenden Instruktion als 1. Quellregister definiert wird.
- 3b. In der WB-Stufe befindet sich eine Instruktion, die ein Resultat in ein Zielregister schreibt, das von einer nachfolgenden Instruktion als 1. Quellregister definiert wird.



# Pipeline - Hindernisse: Datenabhängigkeiten

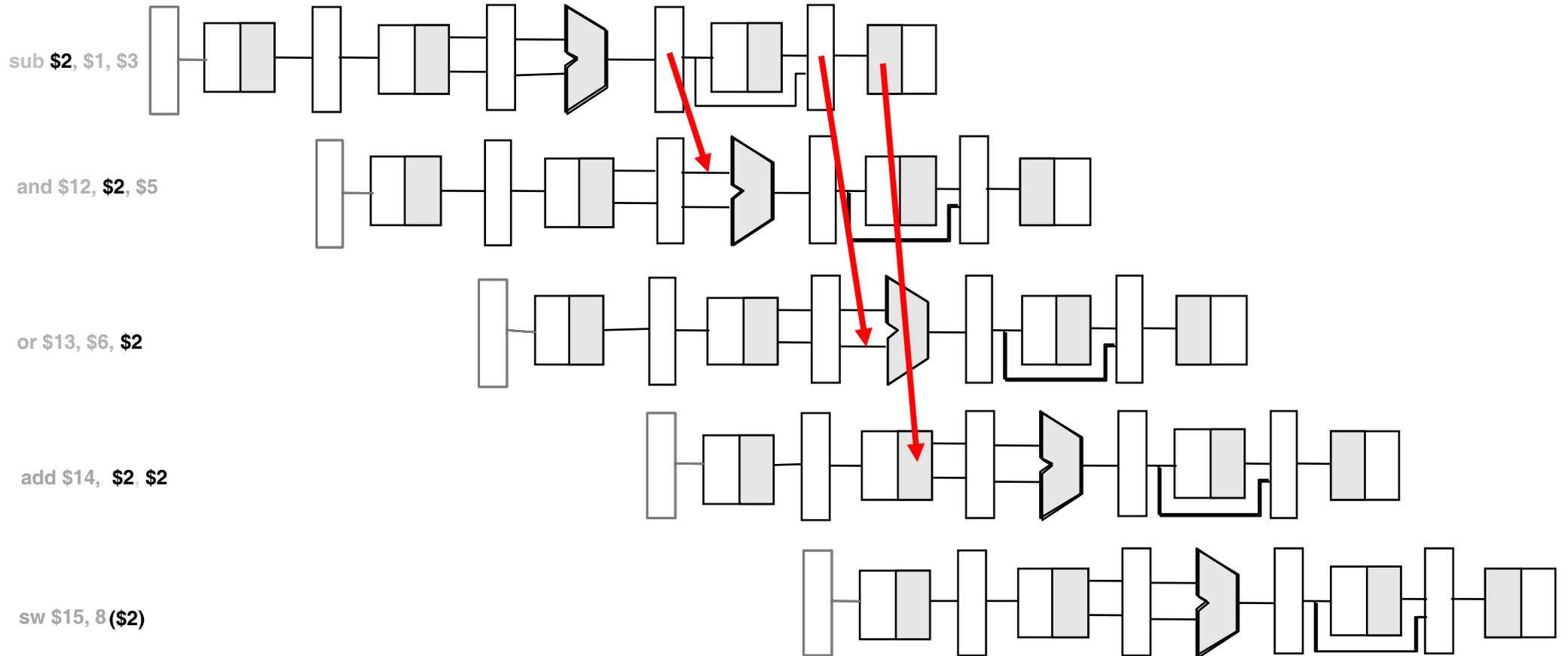
## Klassifizierung der Datenabhängigkeiten

sub \$2, \$7, \$3	_____	keine Datenabhängigkeit
and \$12, \$2, \$5	_____ sub - add	1a
or \$13, \$10, \$2	_____ sub - or	2b
add \$14, \$2, \$2	_____ sub - add 1, sub - add 2	3a, 3b
sw \$15, 8 (\$2)	_____	keine Datenabhängigkeit - \$2 steht hier bereits im Zielregister von "sub \$2, \$7, \$3"



## Datenabhängigkeiten zwischen den Stufen der Pipeline

	T1	T2	T3	T4	T5	T6	T7	T8	T9
<b>Inhalt Reg. \$2</b>	10	10	10	10	10	-25	-25	-25	-25
<b>Wert in EX Stufe</b>	x	x	x	-25	x	x	x	x	x
<b>Wert in ME Stufe</b>	x	x	x	x	-25	x	x	x	x



## Pipeline - Hindernisse: Datenabhängigkeiten

**Datenabhängigkeiten, die nicht erkannt und behandelt werden, führen zur fehlerhaften Ausführung von Programmen !**

**Möglichkeiten, mit Datenabhängigkeiten umzugehen?**

**Software:** Der Compiler erzeugt keine abhängigen Instruktionen, d.h. er fügt so viele NOPs ein, daß die Abhängigkeiten nicht auftreten. Er muß also bei jeder Instruktion, die er erzeugt, überprüfen, ob eine Datenabhängigkeit zu einer vorhergehenden Instruktion vorliegt. (eine frühere Version des MIPS-Prozessors nutzte diese Möglichkeit, um seine Hardwarekomplexität klein zu halten)

**Hardware:** Ausnutzung der Bedingungen für die Datenabhängigkeit, um eine automatische Erkennung der Abhängigkeiten während der Laufzeit zu ermöglichen.

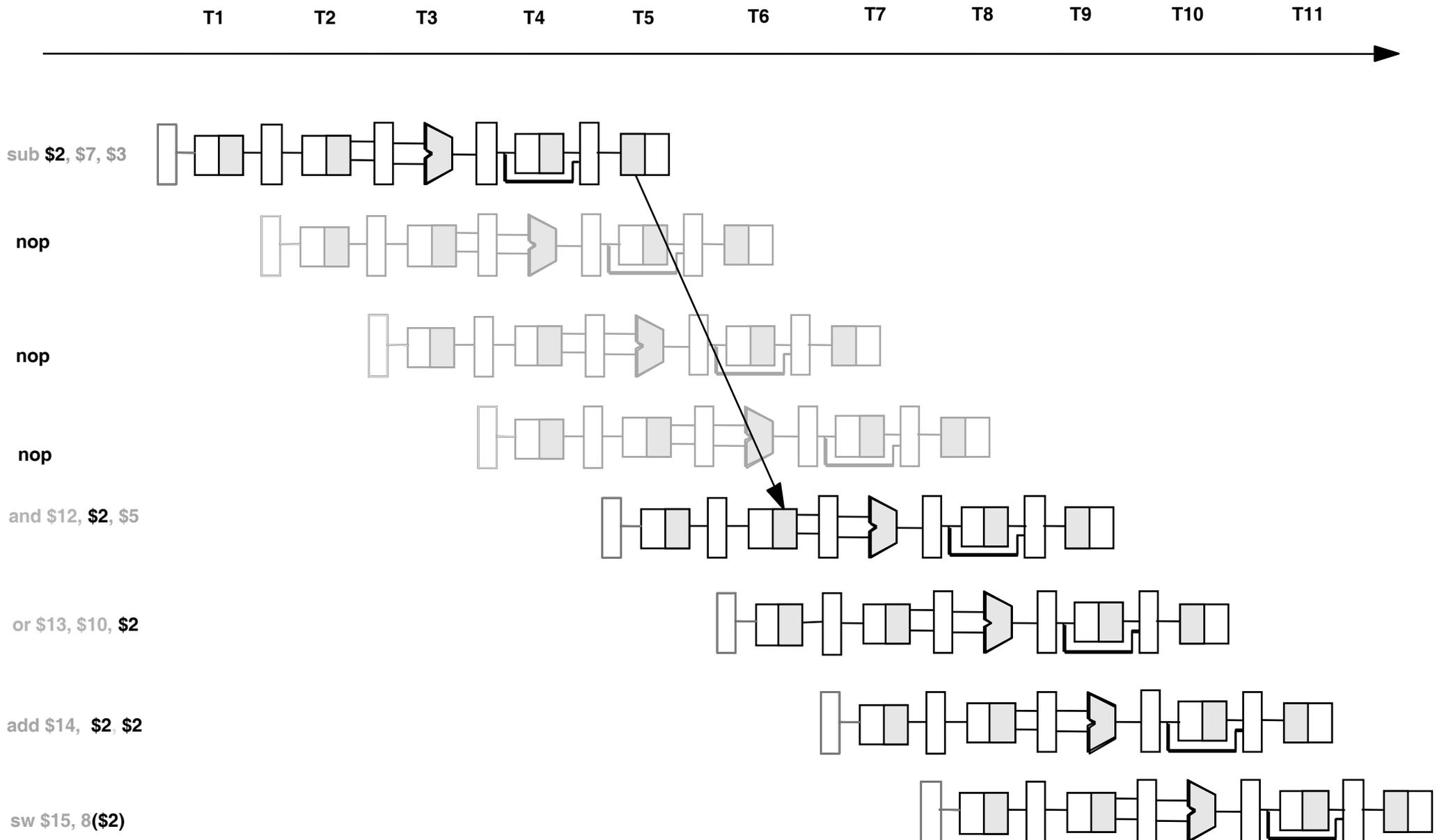
Beispielprogramm:

```
sub $2, $7, $3
and $12, $2, $5
or $13, $10, $2
add $14, $2, $2
sw $15, 8($2)
```



```
sub $2, $7, $3
nop
nop
nop
and $12, $2, $5
or $13, $10, $2
add $14, $2, $2
sw $15, 8($2)
```

# Pipeline - Hindernisse: Datenabhängigkeiten



# Eine Einheit zur Erkennung von Datenabhängigkeiten



# Pipeline - Hindernisse: Datenabhängigkeiten

## Bedingungen für Datenabhängigkeiten

### EX - Abhängigkeit

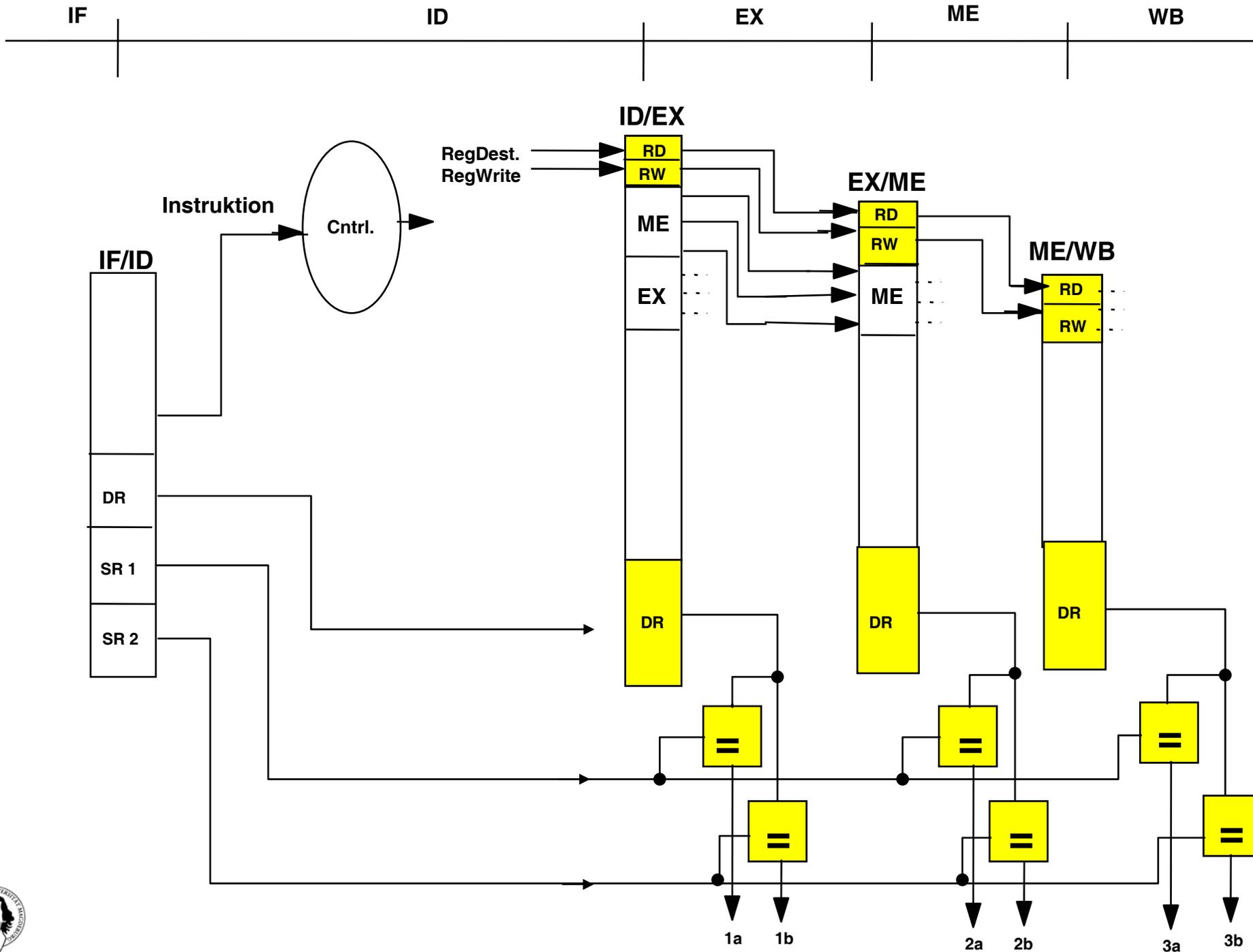
$$\text{ID/EX.RegWrite} \bullet ( (\text{ID/EX.WriteRegister} = \text{IF/ID.ReadRegister1}) \\ + (\text{ID/EX.WriteRegister} = \text{IF/ID.ReadRegister2}) )$$

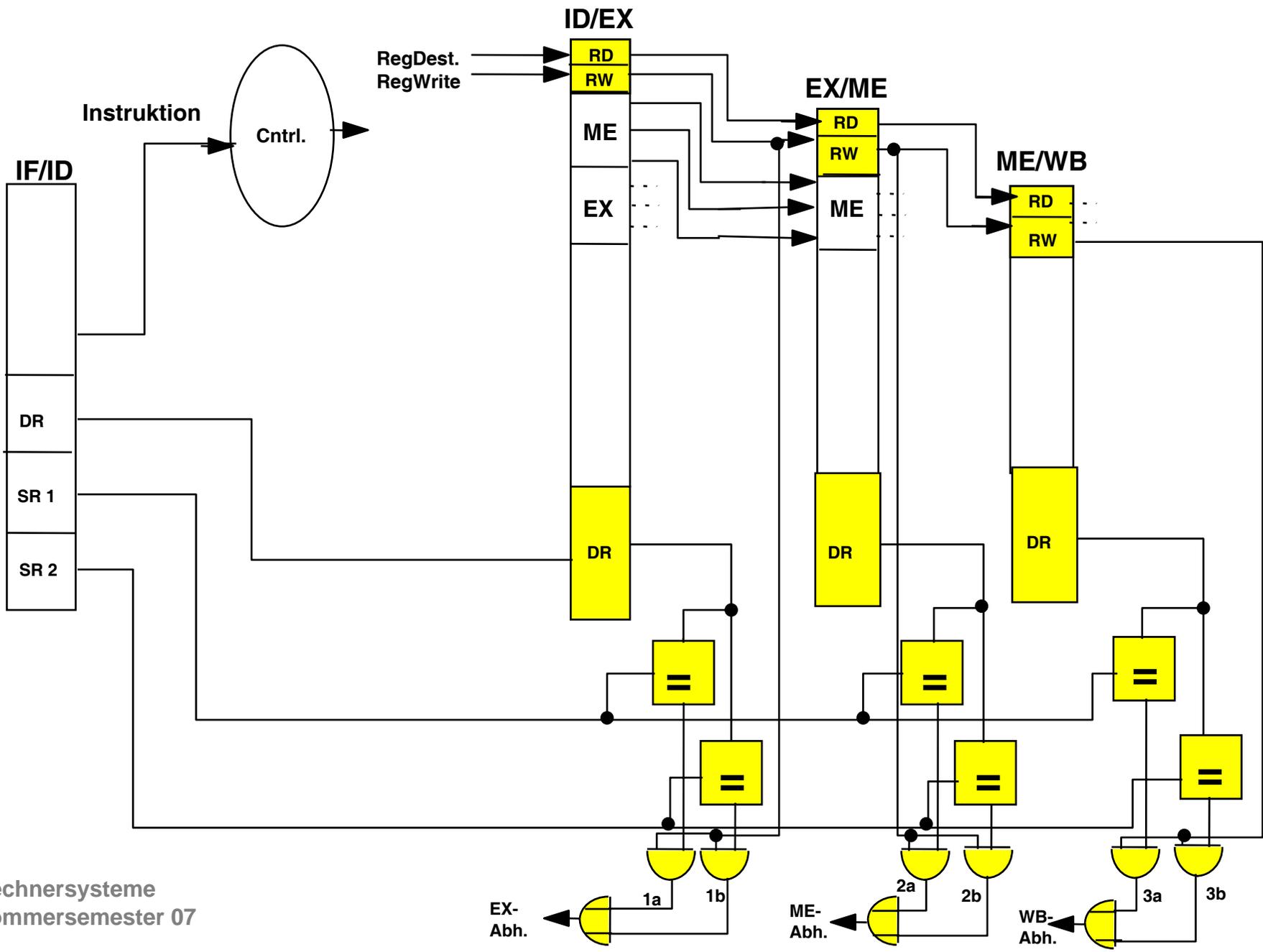
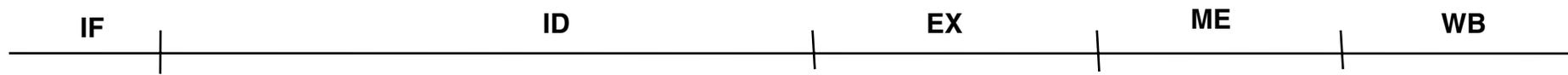
### ME - Abhängigkeit

$$\text{EX/ME.RegWrite} \bullet ( (\text{EX/ME.WriteRegister} = \text{IF/ID.ReadRegister 1}) \\ + (\text{EX/ME.WriteRegister} = \text{IF/ID.ReadRegister 2}) )$$

### WB - Abhängigkeit

$$\text{ME/WB.RegWrite} \bullet ( (\text{ME/WB.WriteRegister} = \text{IF/ID.ReadRegister 1}) \\ + (\text{ME/WB.WriteRegister} = \text{IF/ID.ReadRegister 2}) )$$





# Pipeline - Hindernisse: Datenabhängigkeiten

## Bedingungen für Datenabhängigkeiten als Funktion der Kontrollsignale

### EX - Abhängigkeit

$ID/EX.RegWrite \bullet ( (ID/EX.RegDst = 0 \bullet ID/EX.WriteRegisterRt = IF/ID.ReadRegister 1)$	<i>/*</i> Load: RegDest =0
$+ (ID/EX.RegDst = 1 \bullet ID/EX.WriteRegisterRd = IF/ID.ReadRegister 1)$	<i>/*</i> WriteRegisterRt
$+ (ID/EX.RegDst = 0 \bullet ID/EX.WriteRegisterRt = IF/ID.ReadRegister 2)$	<i>/*</i> R-Format: RegDest =1
$+ (ID/EX.RegDst = 1 \bullet ID/EX.WriteRegisterRd = IF/ID.ReadRegister2) )$	<i>/*</i> WriteRegisterRd

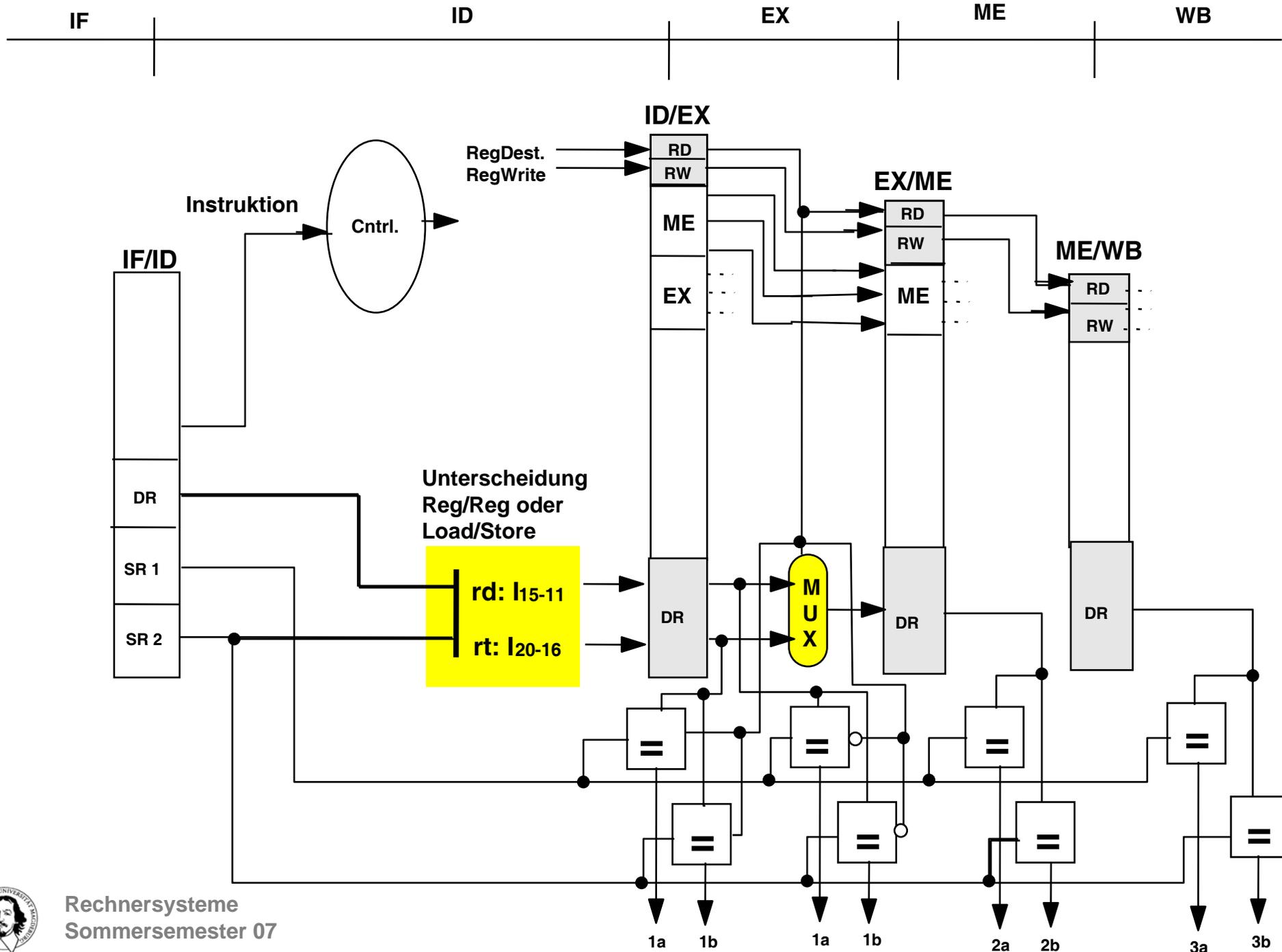
### ME - Abhängigkeit

$EX/ME.RegWrite \bullet ( (EX/ME.WriteRegister = IF/ID.ReadRegister 1)$   
 $+ (EX/ME.WriteRegister = IF/ID.ReadRegister 2)$

### WB - Abhängigkeit

$ME/WB.RegWrite \bullet ( (ME/WB.WriteRegister = IF/ID.ReadRegister 1)$   
 $+ (ME/WB.WriteRegister = IF/ID.ReadRegister 2)$





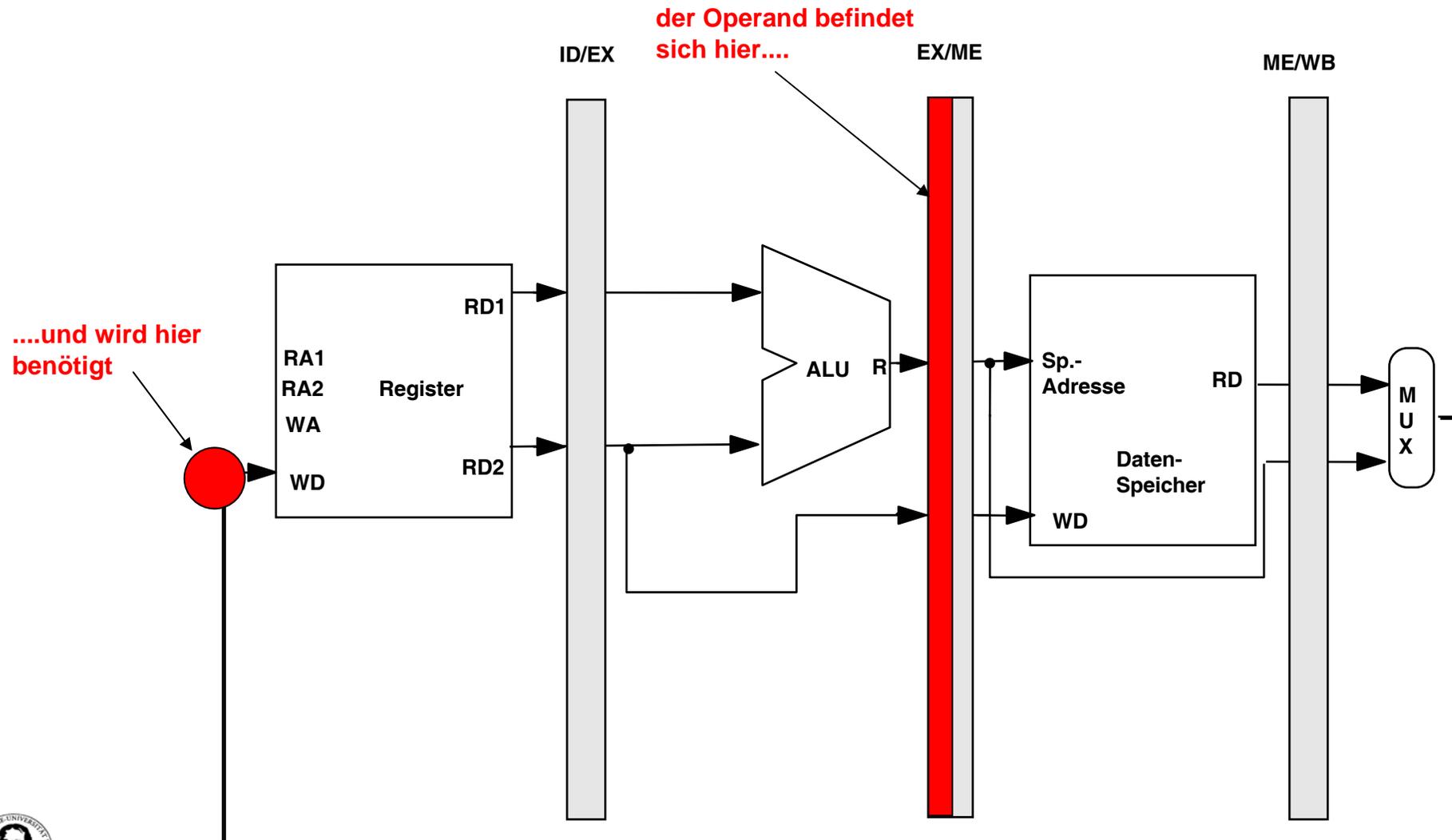
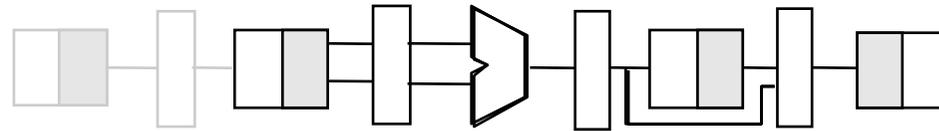


# Eine Einheit zur Umgehung von Datenabhängigkeiten (Bypass)

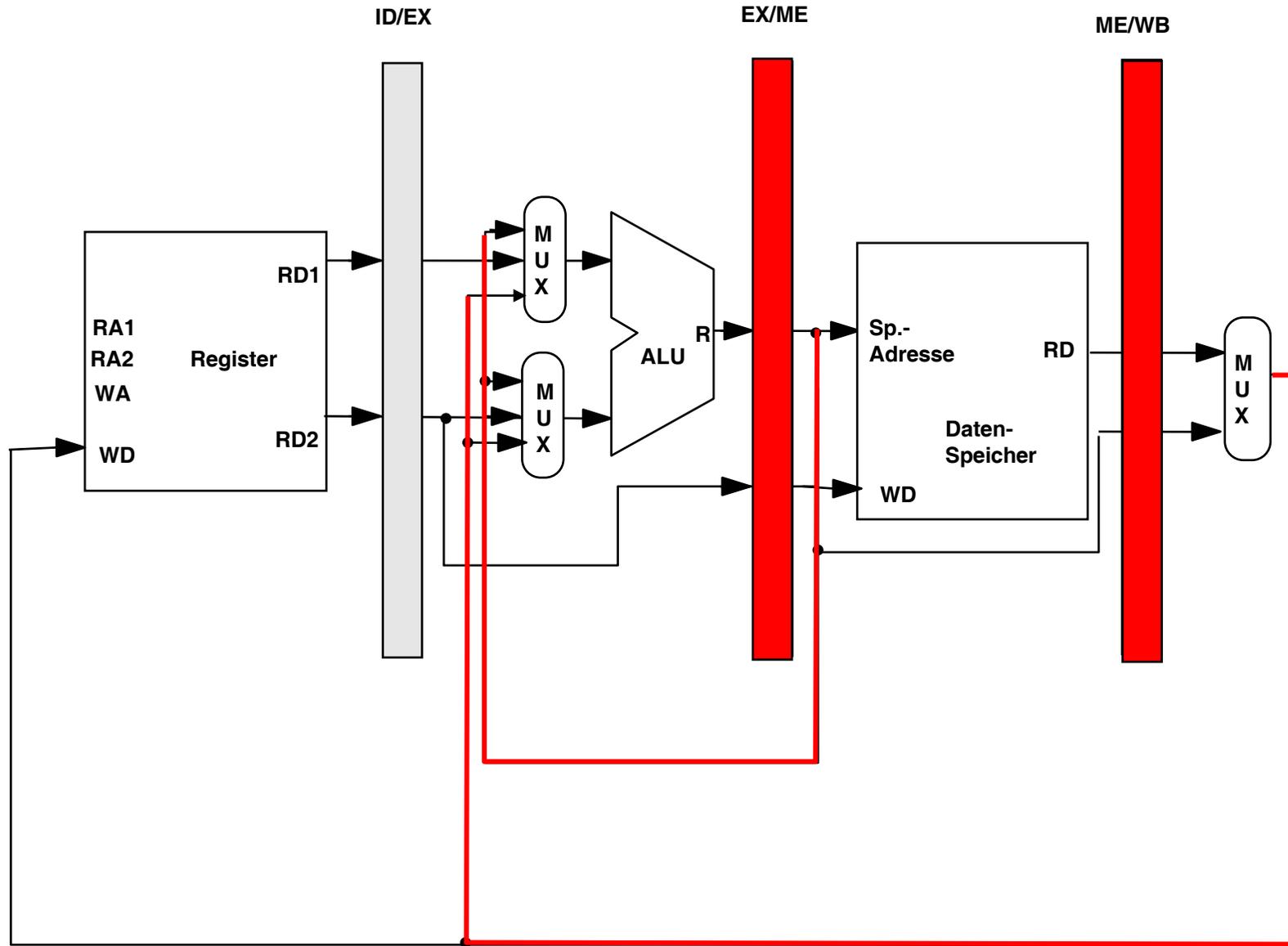
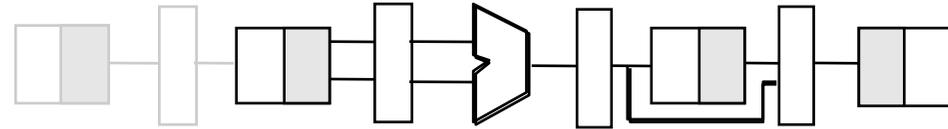
## Forwarding



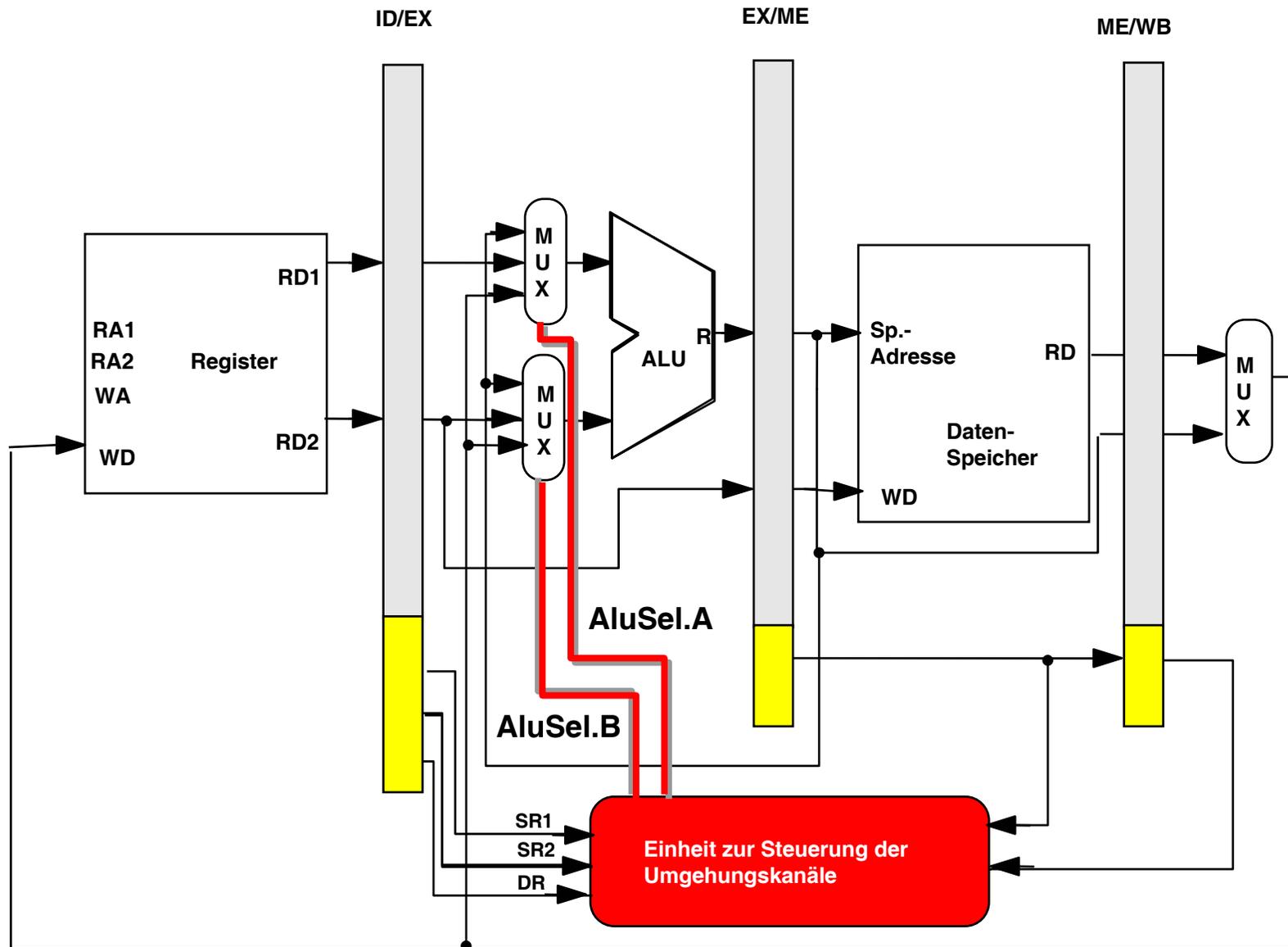
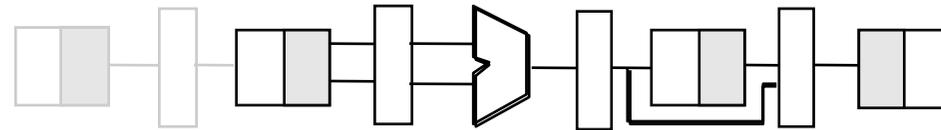
# Relevante Stufen der Pipeline ohne Kontrollsignale



**Pipeline mit "Umgehungskanälen"  
(Bypass) zur Weiteleitung  
(Forwarding) von Resultaten**



**Pipeline mit "Umgehungskanälen"  
(Bypass) zur Weiteleitung  
(Forwarding) von Resultaten**



## Bedingungen zur Steuerung der Umleitungskanäle

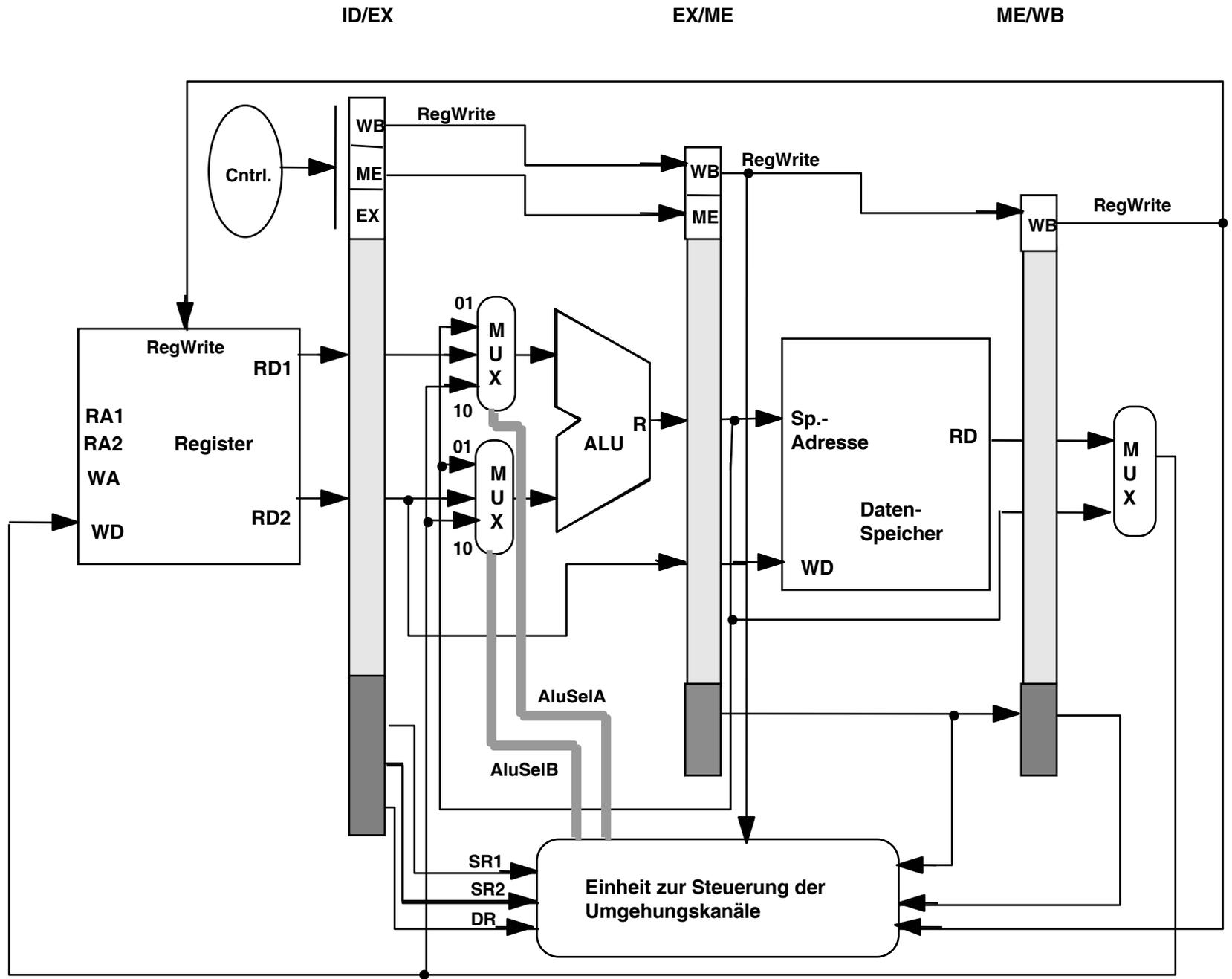
### EX-Abhängigkeit:

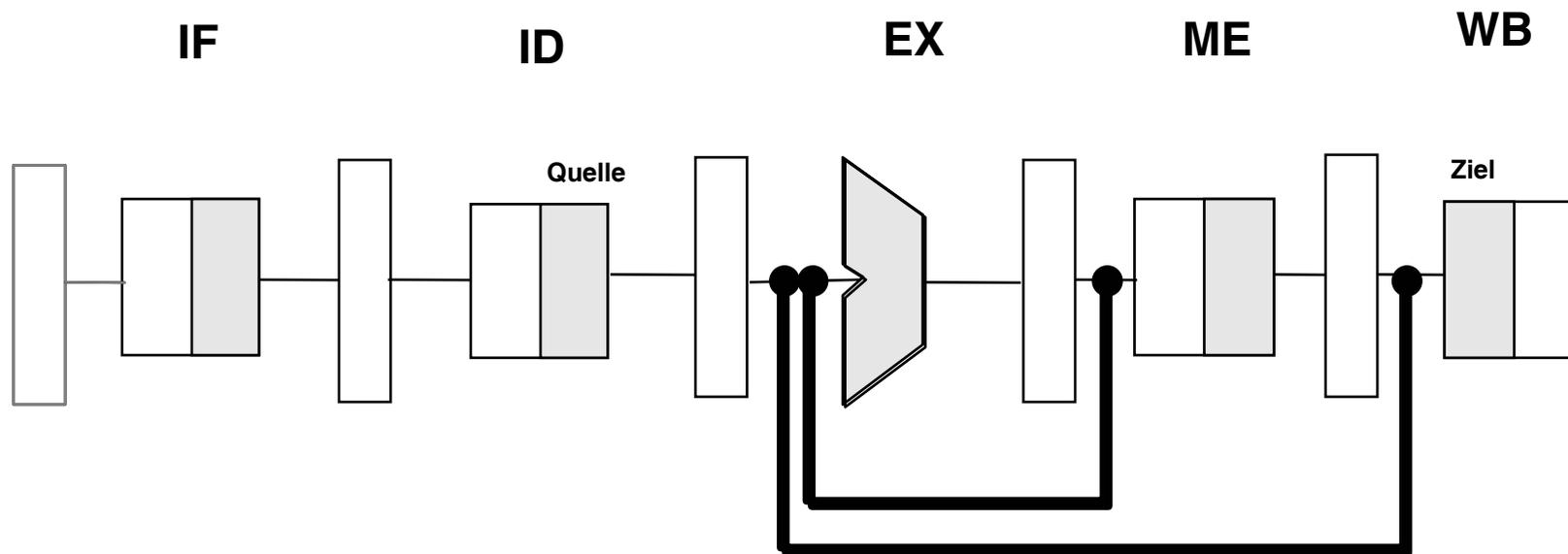
( EX/ME.RegWrite • (EX/ME.WriteReg = ID/EX.ReadReg1) ) : AluSel.A ← 01  
( EX/ME.RegWrite • (EX/ME.WriteReg = ID/EX.ReadReg2) ) : AluSel.B ← 01

### ME-Abhängigkeit:

( ME/WB.RegWrite • (ME/WB.WriteReg = ID/EX.ReadReg1) ) : AluSel.A ← 10  
( ME/WB.RegWrite • (ME/WB.WriteReg = ID/EX.ReadReg2) ) : AluSel.B ← 10

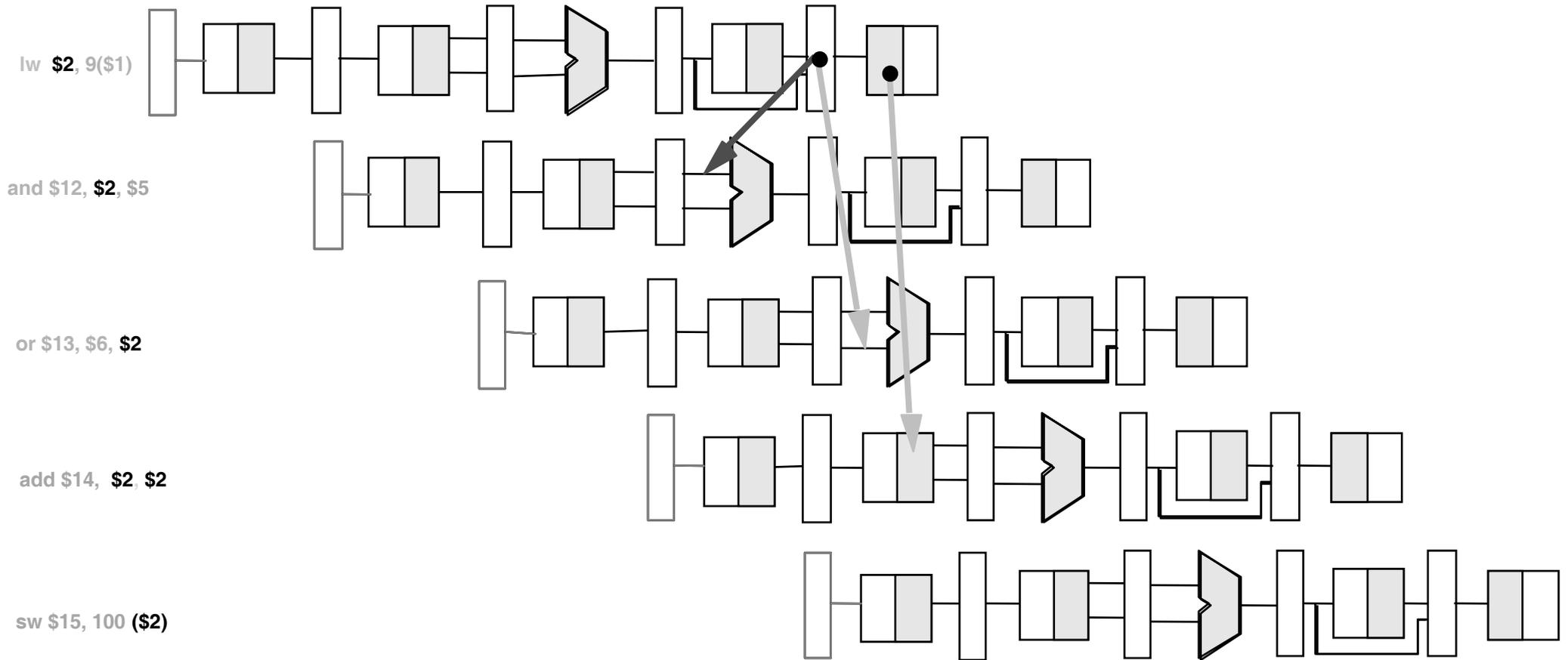






**Datenabhängigkeiten zwischen den Stufen der Pipeline bei Load- Befehlen.**

**Da der Operand sich noch nicht in der Pipeline befindet, bring Forwarding keinen Nutzen.**



## Auflösung rückwärtsgerichteter Datenabhängigkeiten zwischen den Stufen der Pipeline bei Load-Befehlen

1. Softwarelösung - Einfügen von NOPs
2. Modifikation der Einheit zur Erkennung von Datenabhängigkeiten

**Bedingung:**

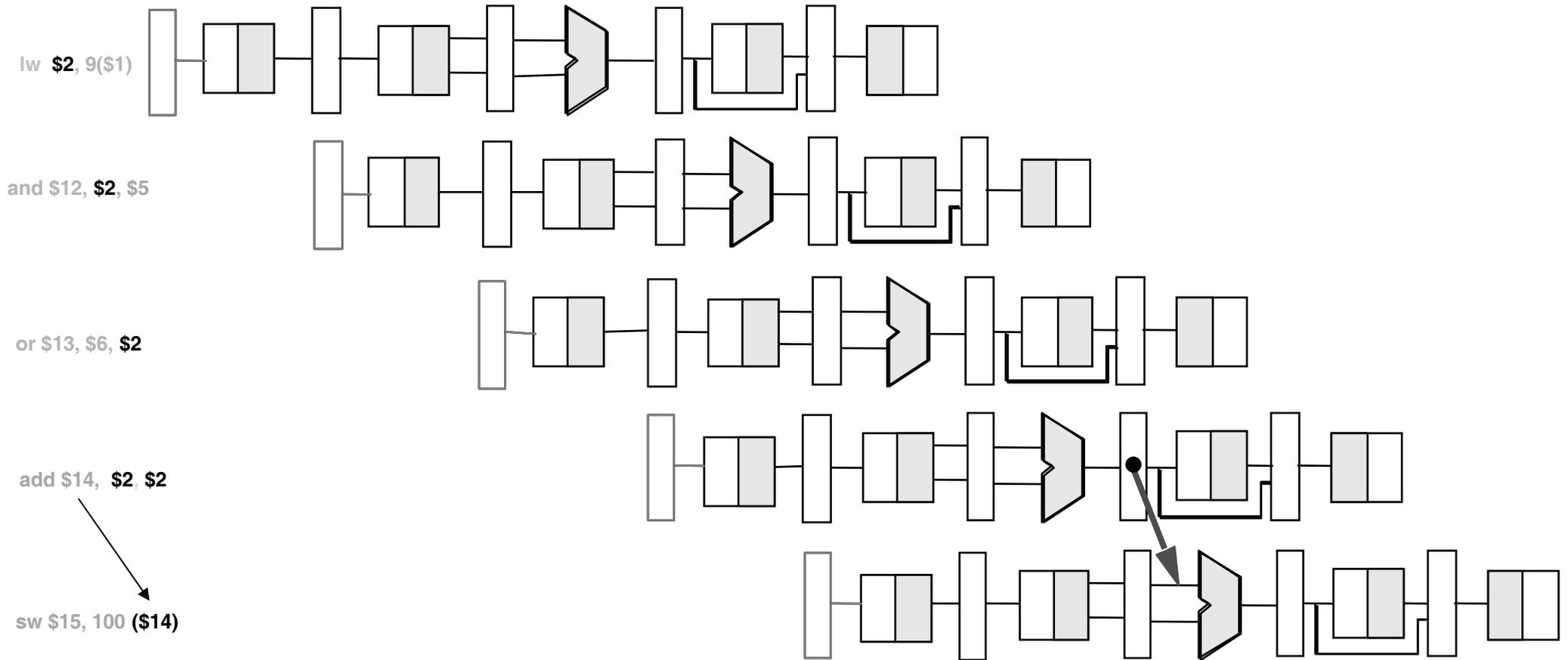
**Voraussetzung:** Bei einer LOAD-Operation gilt für das Kontrollsignal :  $ID/EX.RegDst = 0$ .

**Für die Erkennung der entsprechenden Abhängigkeit gilt deshalb die Bedingung:**

**IF**  
 $(ID/EX.RegWrite \cdot (ID/EX.RegDest = 0) \cdot ( (ID/EX.WriteRegRT = IF/ID.ReadRegister1) + (ID/EX.WriteRegRT = IF/ID.ReadRegister2) ) )$   
**THEN**  
(Verzögern der nächsten Instruktion)



# Datenabhängigkeiten zwischen den Stufen der Pipeline bei Store-Befehlen

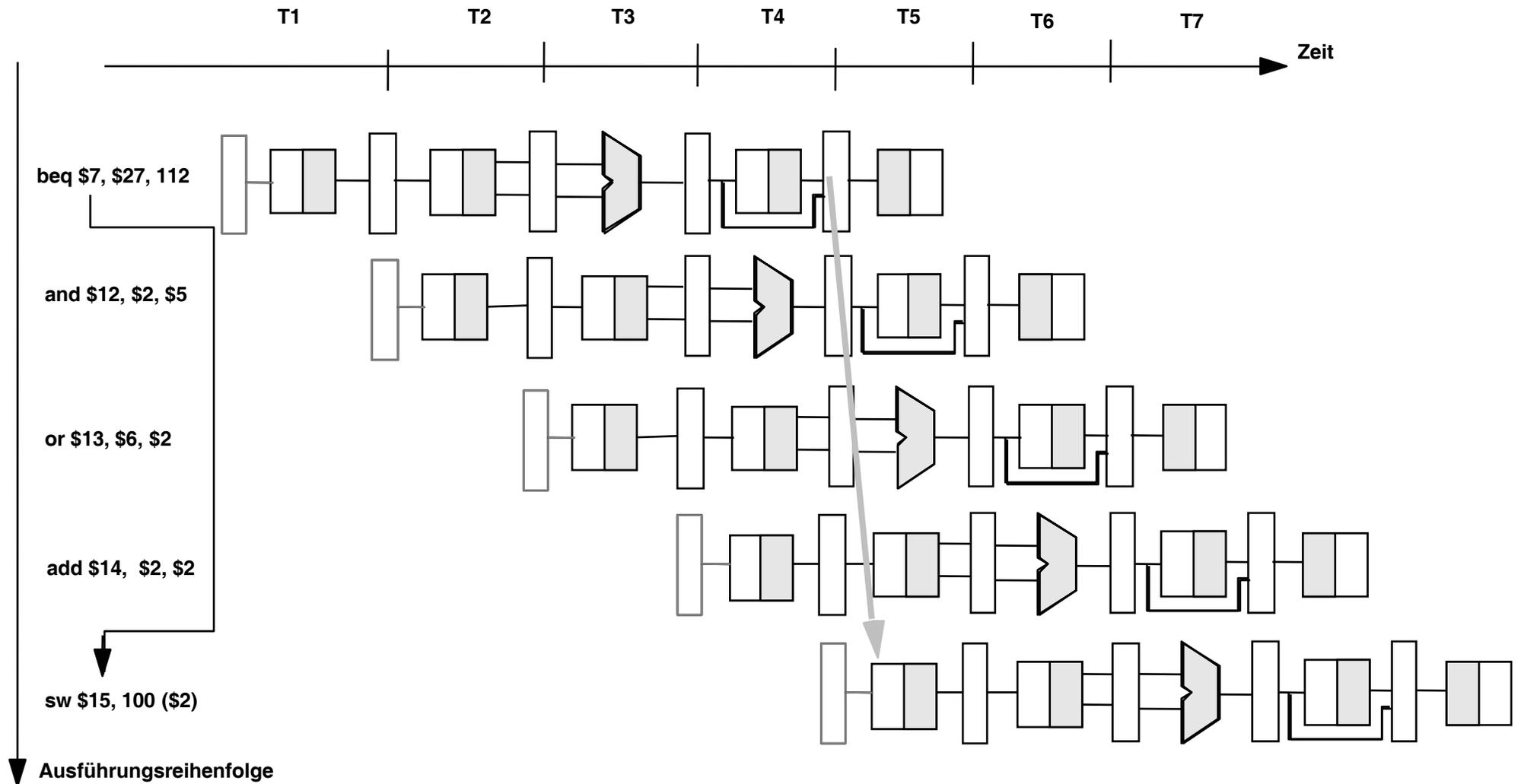


# Kontroll Abhängigkeiten

## Branch Hazards

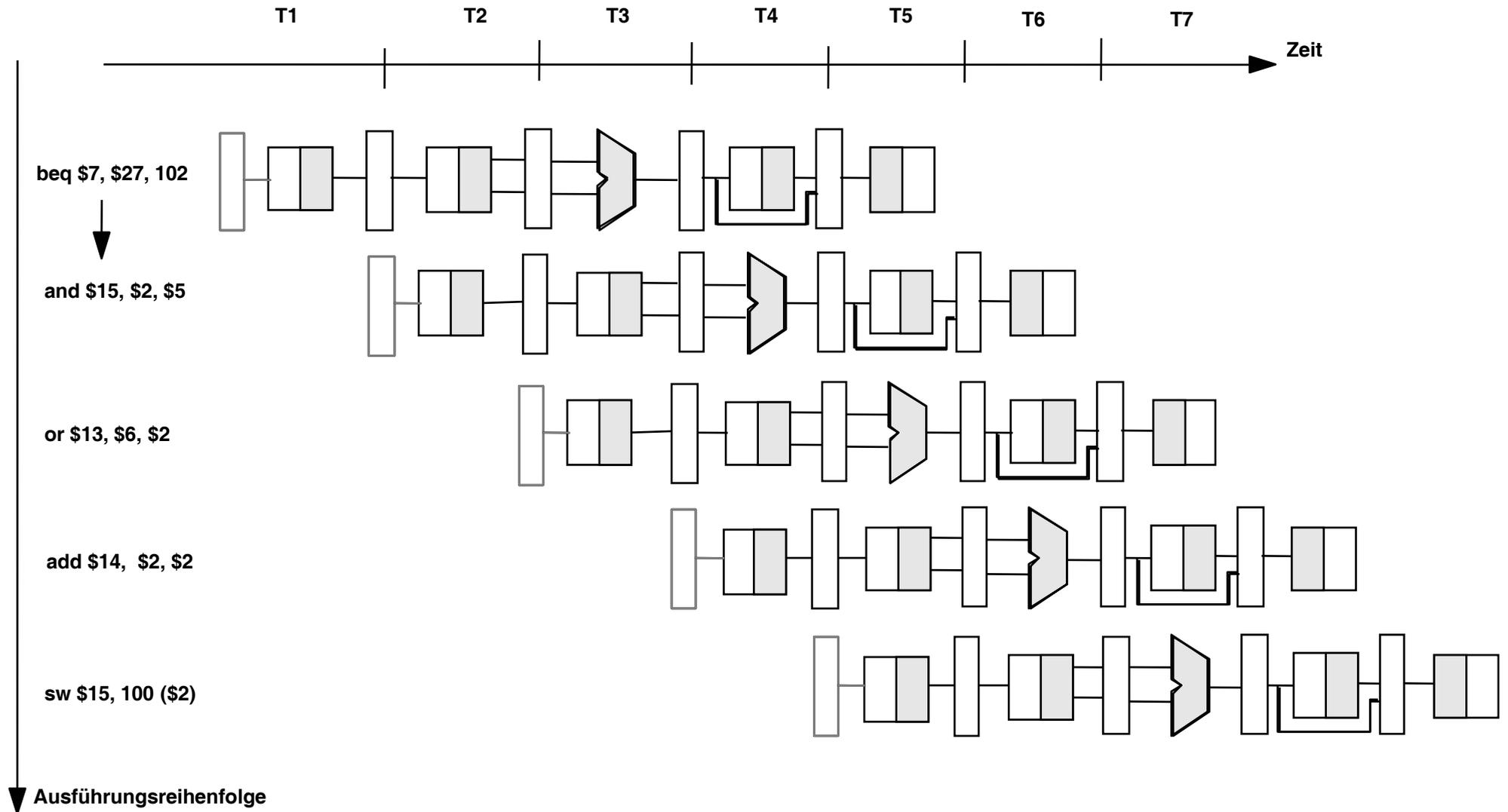


# Pipeline - Hindernisse: Kontrollabhängigkeiten



# Pipeline - Hindernisse: Kontrollabhängigkeiten

Der Sprung wird nicht ausgeführt



# Pipeline - Hindernisse: Kontrollabhängigkeiten

## Auflösung von Kontrollabhängigkeiten:

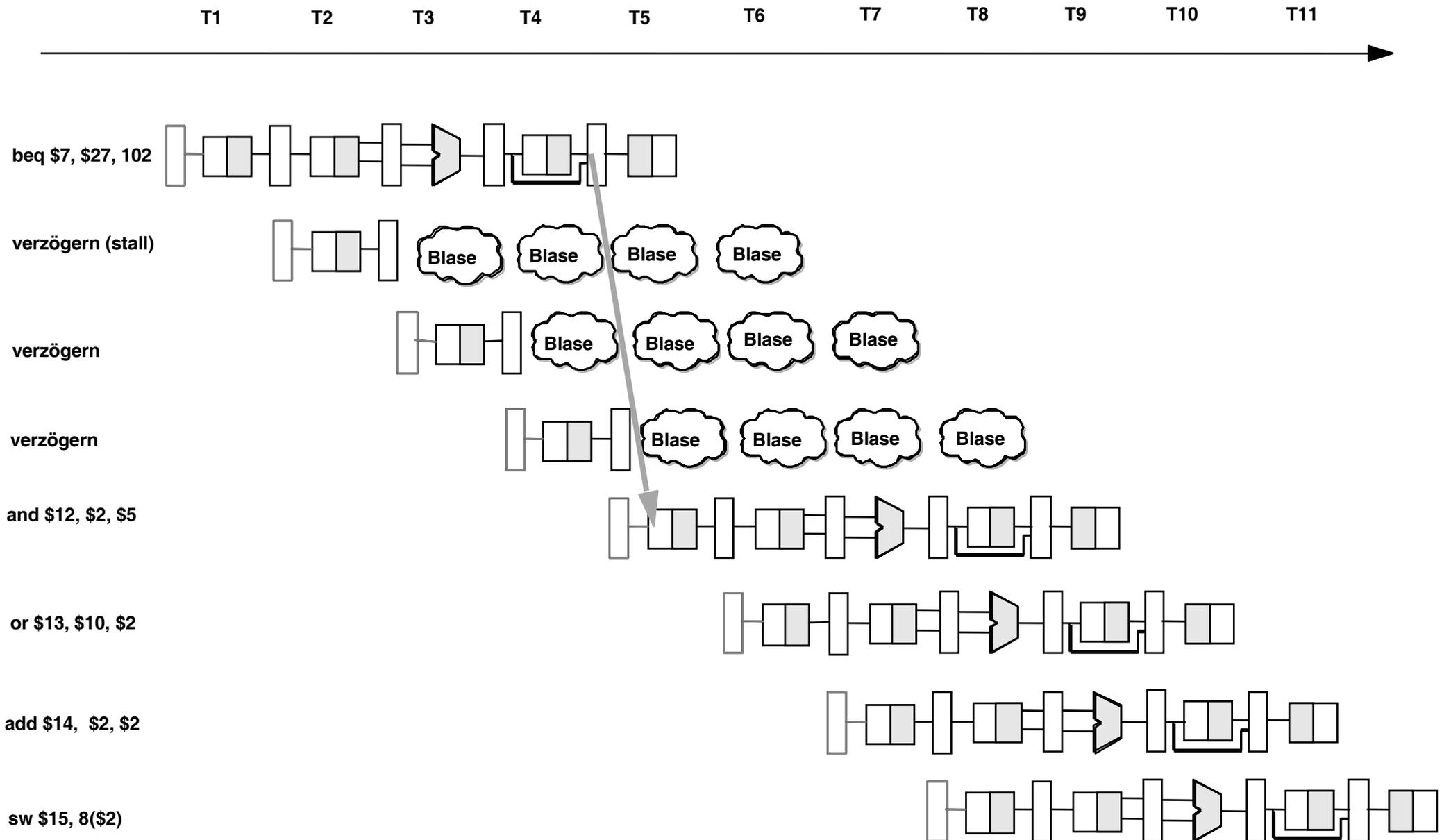
- Verzögern

Zur automatischen Erkennung von Sprüngen muß lediglich der OPCODE ausgewertet werden.  
Andere Abhängigkeiten zu Instruktionen in der Pipeline bestehen NICHT !

- Sprung-Voraussage (Branch Prediction)
- Verzögerter Sprung (Delayed Branch) : Die Operation unmittelbar hinter dem Sprung wird immer ausgeführt, unabhängig davon, ob der Sprung ausgeführt wird oder nicht.
- “Ausrollen“ von Schleifen (Loop Unrolling)



# Pipeline - Hindernisse: Kontrollabhängigkeiten



# Pipeline - Hindernisse: Kontrollabhängigkeiten

**Ausnahmebehandlung:**

**Beisp.**

sub \$7, \$27, \$23

and \$12, \$2, \$5

or \$13, \$6, \$2

add \$14, \$2, \$14

sw \$15, 100 (\$2)

1. es muß verhindert werden, daß das fehlerhafte Resultat zurückgeschrieben wird

—▶ Unterbrechung der aktuellen Instruktion

2. der Sprung zur Ausnahmebehandlungsroutine muß als nächste Instruktion ausgeführt werden

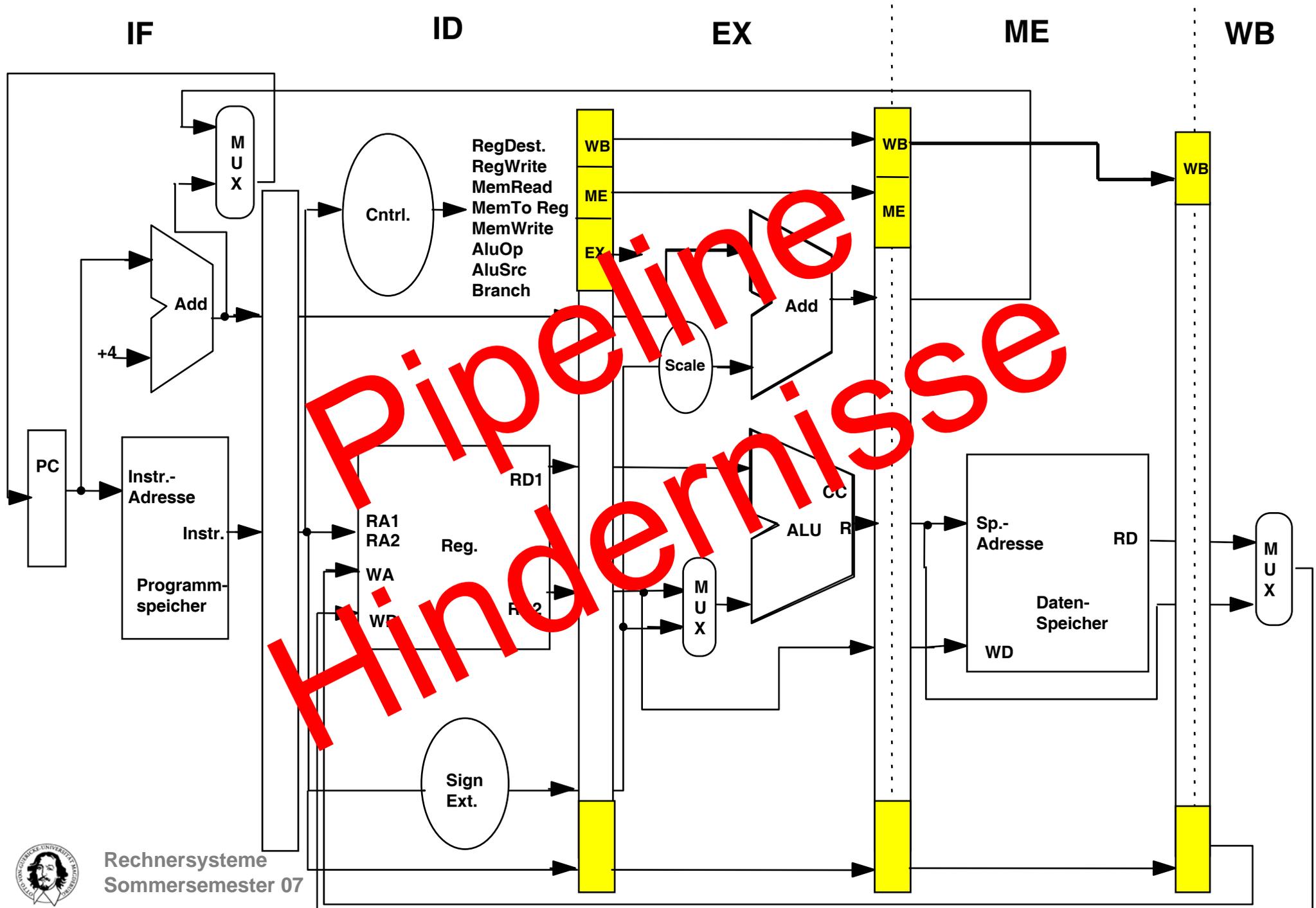
—▶ die in der Pipeline befindlichen Instruktionen müssen gelöscht werden



# Zusammenfassung: Pipelining

- **Pipelining ist der Schlüssel zur “Single Cycle“ Architektur**
- **Pipelining nutzt Parallelismus auf der Befehlsebene,**
  - **Voraussetzung: angepaßter Instruktionssatzes - einfache reguläre Instruktionen**
- **Pipelining für den Programmierer (weitgehend) transparent**



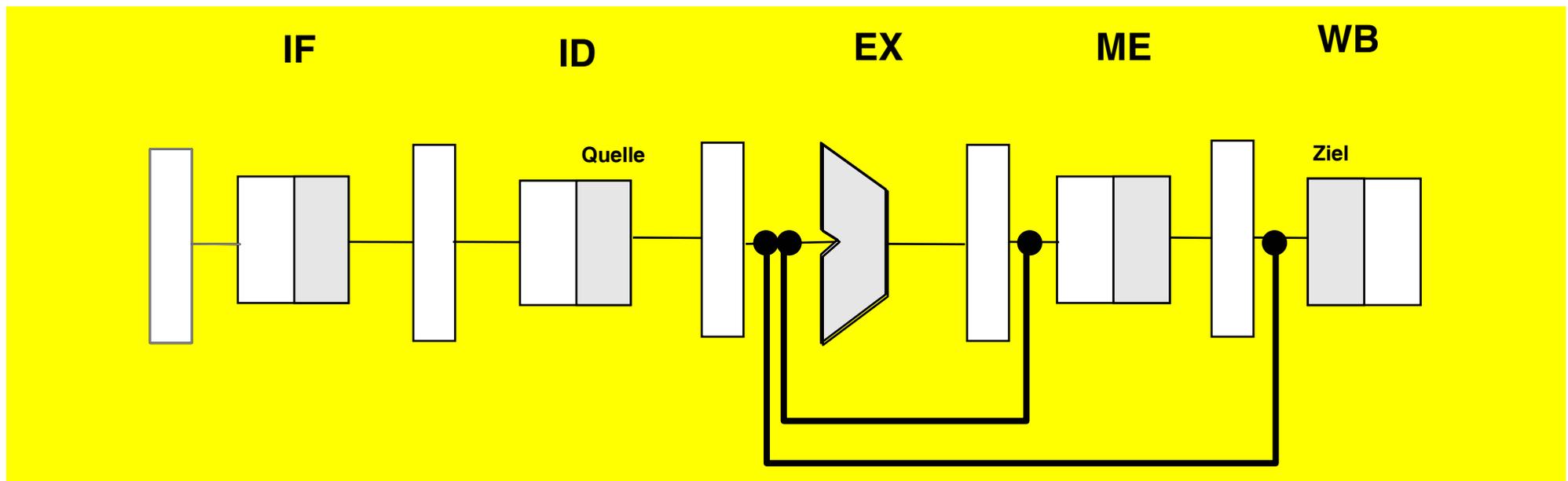


**Pipeline Hindernisse**



# Datenabhängigkeiten (Data Hazards):

- Datenabhängigkeiten können automatisch erkannt werden
- Datenabhängigkeiten können durch Umgehungen (Forwarding) aufgelöst werden
- **LOAD** ist ein Problem





# Kontrollabhängigkeiten (Control Hazards):

- Sprunabhängigkeiten können meist nur im Zusammenspiel zwischen SW und HW zufriedenstellend aufgelöst werden.
- Unbedingte Sprünge → spezielle Sprungeinheiten
- Bedingte Sprünge:
  - Verzögerter Sprung (Delayed Branching): Befehl nach Sprung wird immer ausgeführt
  - Sprungvorhersage (Branch Prediction)
    - Heuristik: Rücksprünge werden immer ausgeführt
    - Statistik : Auswerten der vergangenen Sprungentscheidungen
  - Ausrollen von Schleifen durch den Compiler
- Problem mit Ausnahmebehandlung



# Ausblick:

- Superpipelining:** Anzahl der Pipelinestufen wird erhöht (bis zu > 30 Stufen).  
Dadurch werden die Aufgaben der einzelnen Pipeline-Stufen einfacher.  
Damit kann auch der Takt erhöht werden.  
Große Probleme mit Pipelinehindernissen!
- Parallele Pipelines:** Anzahl der Pipelines wird erhöht. Erlaubt den Einsatz von Pipelines für spezielle Funktionen.
- Superskalare Pipelines:** Mehrere Instruktionen werden gleichzeitig geholt, decodiert und ausgeführt.  
Ebenfalls Probleme bei Abhängigkeiten.
- Out of Order Processing:** Unabhängige Befehle, die im Programmablauf weiter hinten stehen werden vorgezogen, um die Pipeline zu füllen. Allerdings muss Reihenfolge der Ergebnisse korrekt hergestellt werden → Instruction Retirement
- Probleme mit Interrupt und Exceptions.



# Vorteile und Probleme

Höhere Taktraten und mehr Pipelinestufen:

- + Vergrößern den Durchsatz (Instruktionen/Zeit)
- Mehr „Instructions in-flight“ → mehr Verlust bei Pipeline-Hindernissen

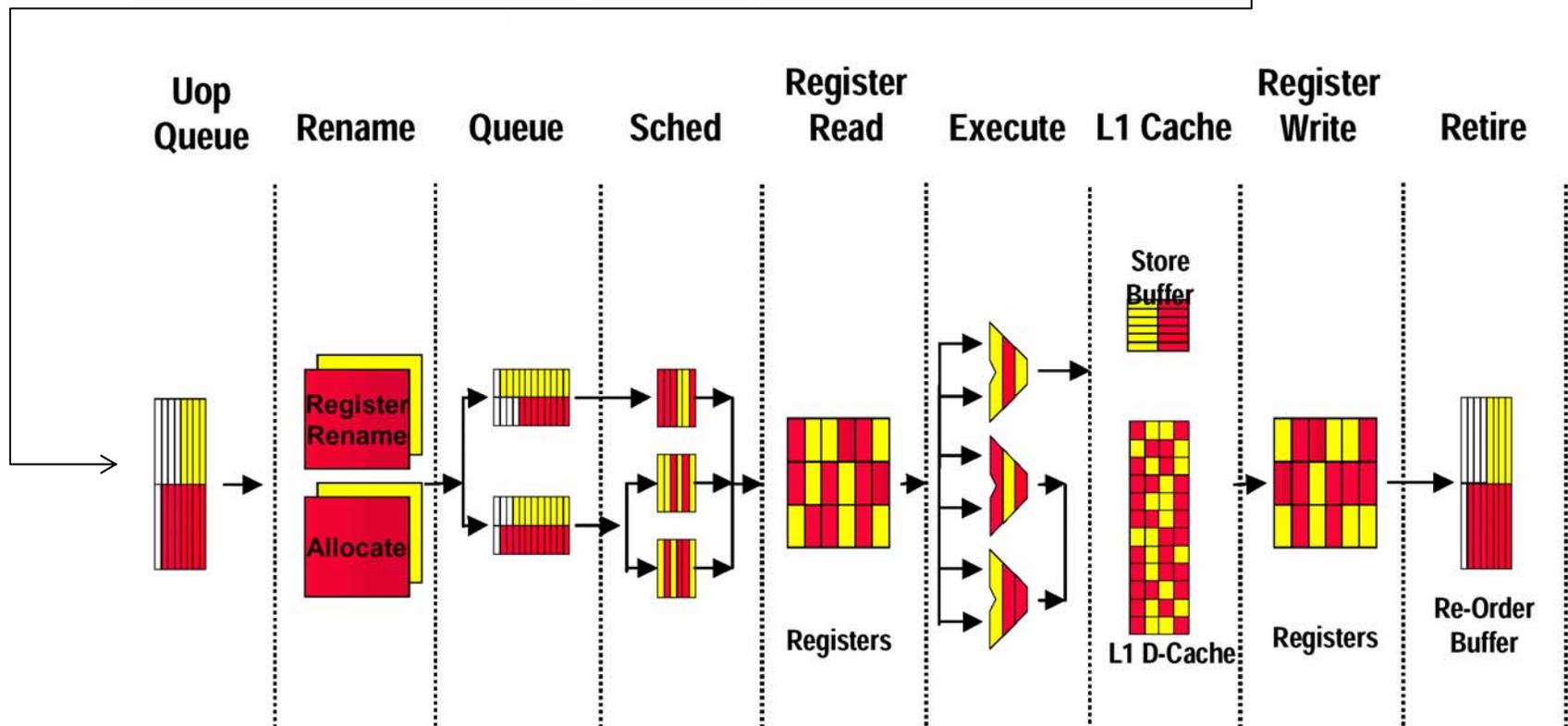
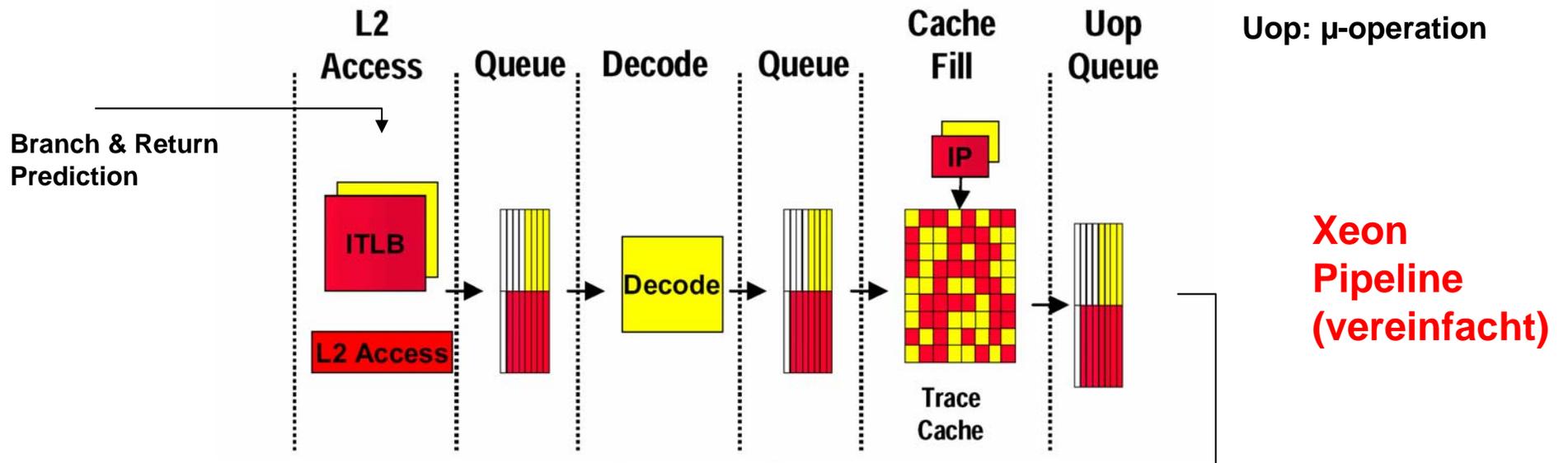
Instruction-Level-Parallelism (ILP):

- + In mehreren Pipelines können Befehle parallel ausgeführt werden
- Es müssen genügend unabhängige Befehle gefunden werden  
→ Out-of-order-execution wertet ein großes „Befehls-Fenster“ aus .  
Die Reihenfolge der Befehle richtet sich nach ihren Abhängigkeiten, nicht nach ihrer Reihenfolge im Programm.
- Befehle müssen durch „Retirement Unit“ wieder in eine konsistente Ordnung gebracht werden. Probleme mit Interrupts und Exceptions.

Caches:

- + schneller Zugriff auf Speicher
- Caches können nur schnell sein, wenn sie klein sind.  
→ Cache misses treten auf und die Pipelines laufen schneller leer als ein „miss“ behandelt werden kann.

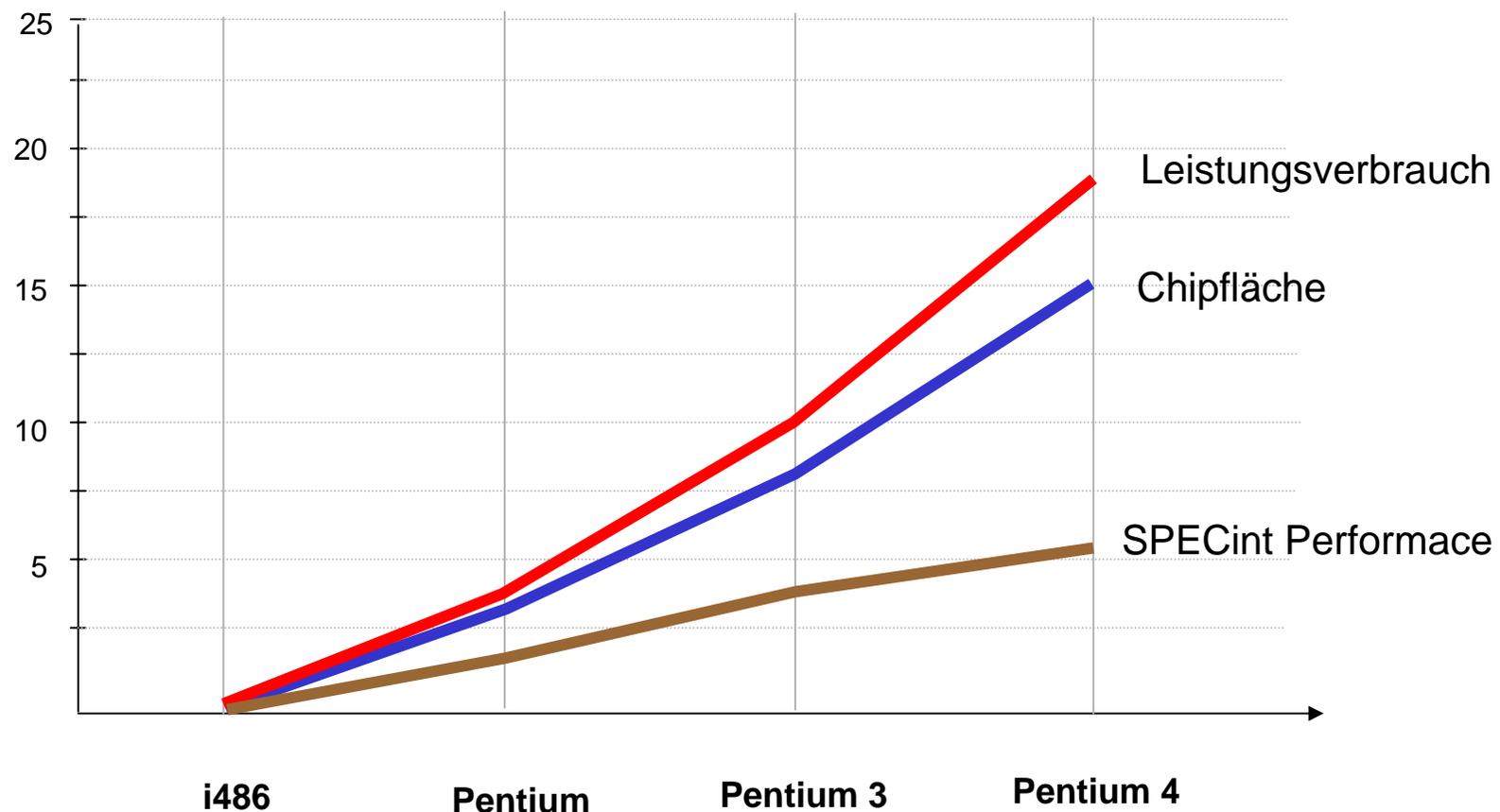




Ausführliche Information unter: [http://www.intel.com/technology/itj/2002/volume06issue01/vol6iss1\\_hyper\\_threading\\_technology.pdf](http://www.intel.com/technology/itj/2002/volume06issue01/vol6iss1_hyper_threading_technology.pdf)

# Vorteile und Probleme

hohe Taktraten und Super(skalar) Pipelines und Caches brauchen überproportional Energie und Chipfläche



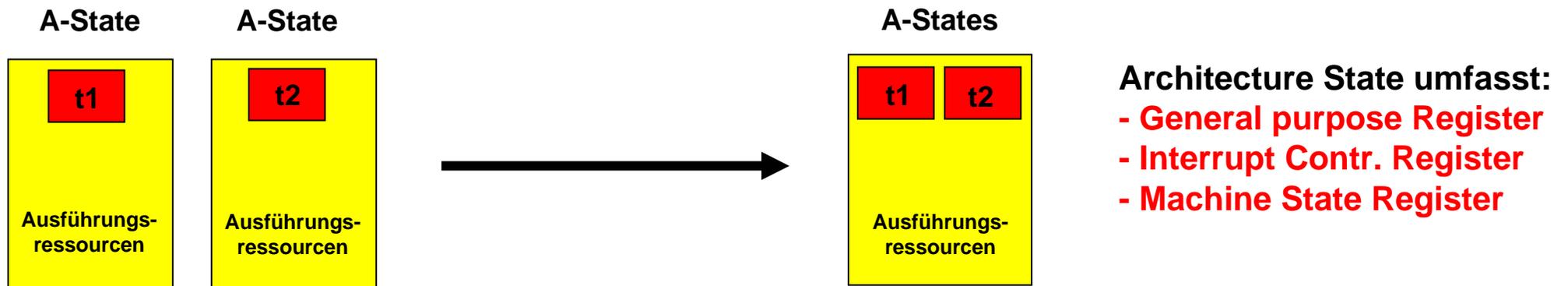
# Thread-Level Parallelismus

Ausführliche Information unter: [http://www.intel.com/technology/itj/2002/volume06issue01/vol6iss1\\_hyper\\_threading\\_technology.pdf](http://www.intel.com/technology/itj/2002/volume06issue01/vol6iss1_hyper_threading_technology.pdf)

**Annahme:** Zwei unabhängige Threads haben keine abhängigen Instruktionen.

**Beobachtung:** Die Speicherung des Zustands eines Thread braucht weniger Transistoren/Chipfläche als die zugehörigen Ausführungseinheiten.

Folgerung: Multiprocessing auf **EINEM** Chip



Alle Ausführungseinheiten wie Caches, Pipeline, Sprungvohersage, OOO-Einheit, Kontroll-Logik, Busse werden gemeinsam benutzt.