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

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

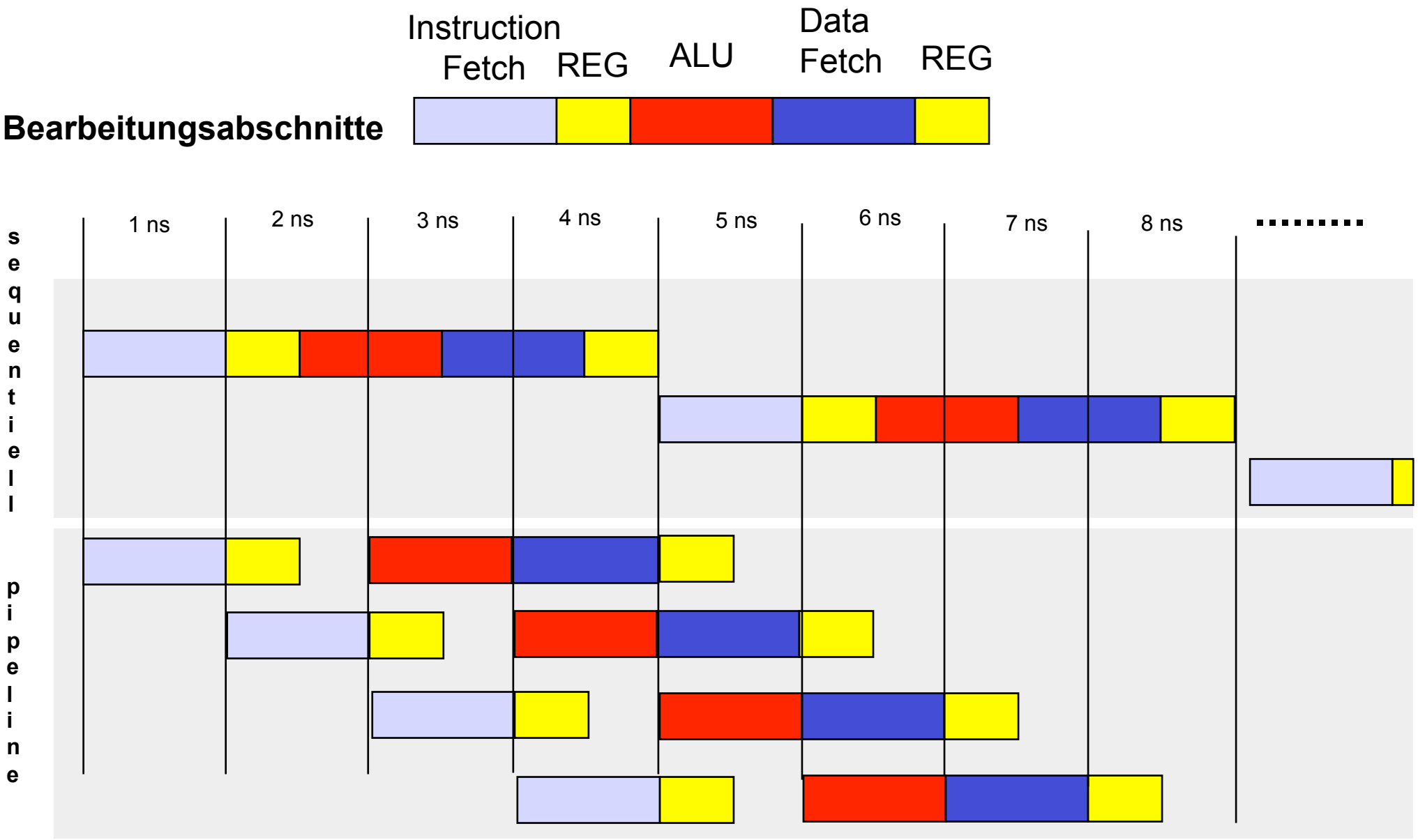
Grundeigenschaften einer Pipeline:

- **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 entprechenden 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
1 ns
2 ns
3 ns
4 ns
5 ns
6 ns
7 ns
8 ns
sequentiell
pipeline

Struktur des Datenpfads

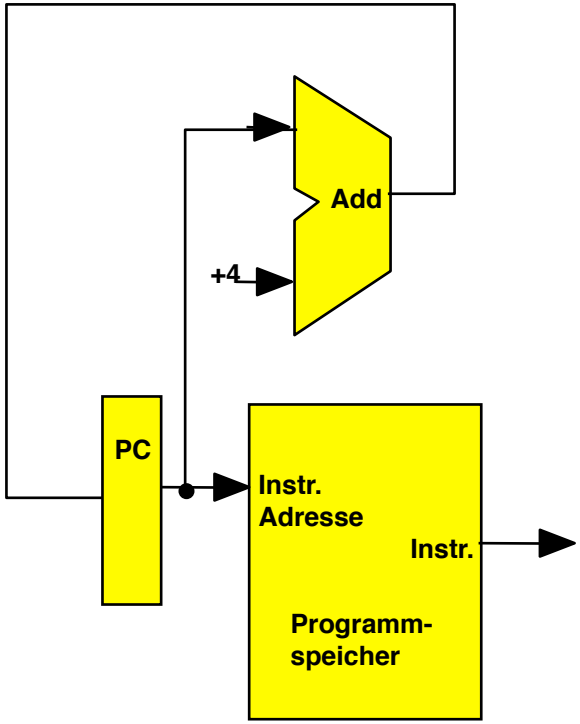
die folgende Architektur der Pipeline ist an die MIPS-Architektur angelehnt, die im Buch:

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

beschrieben ist.

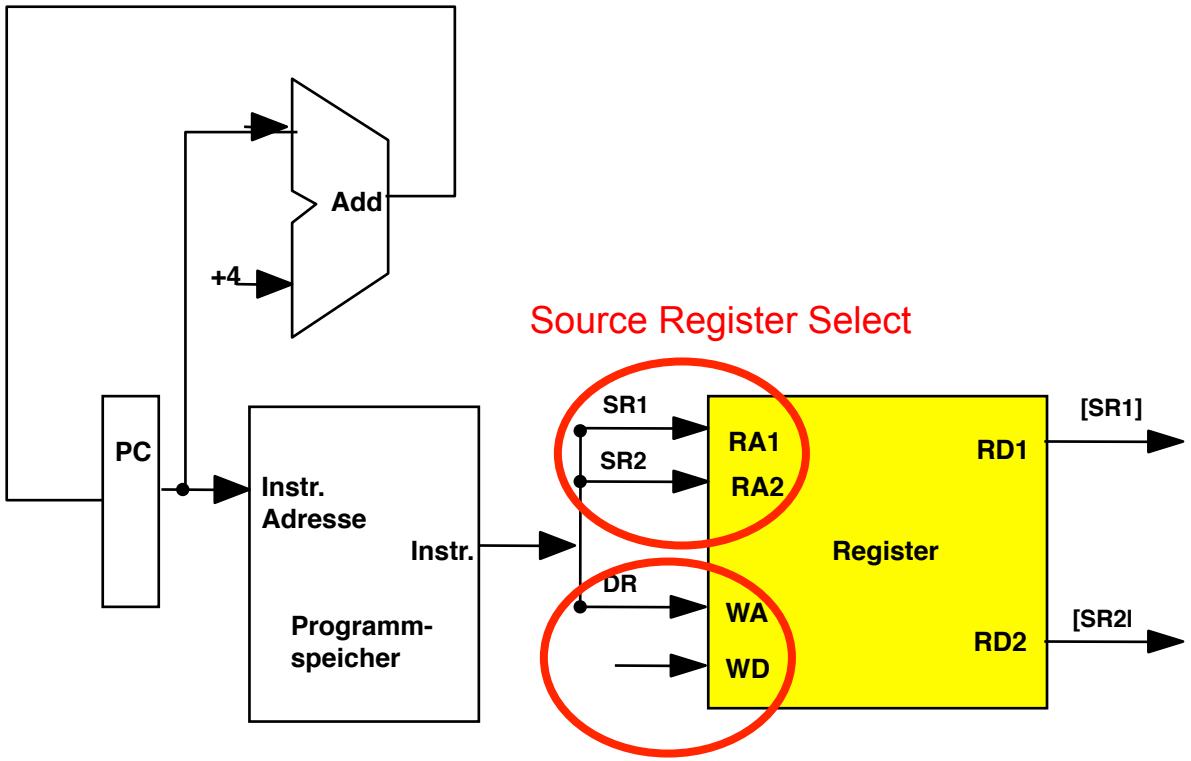
Instruction Fetch (IF)

- Adressierung des Programmspeichers
- Holen der Instruktion



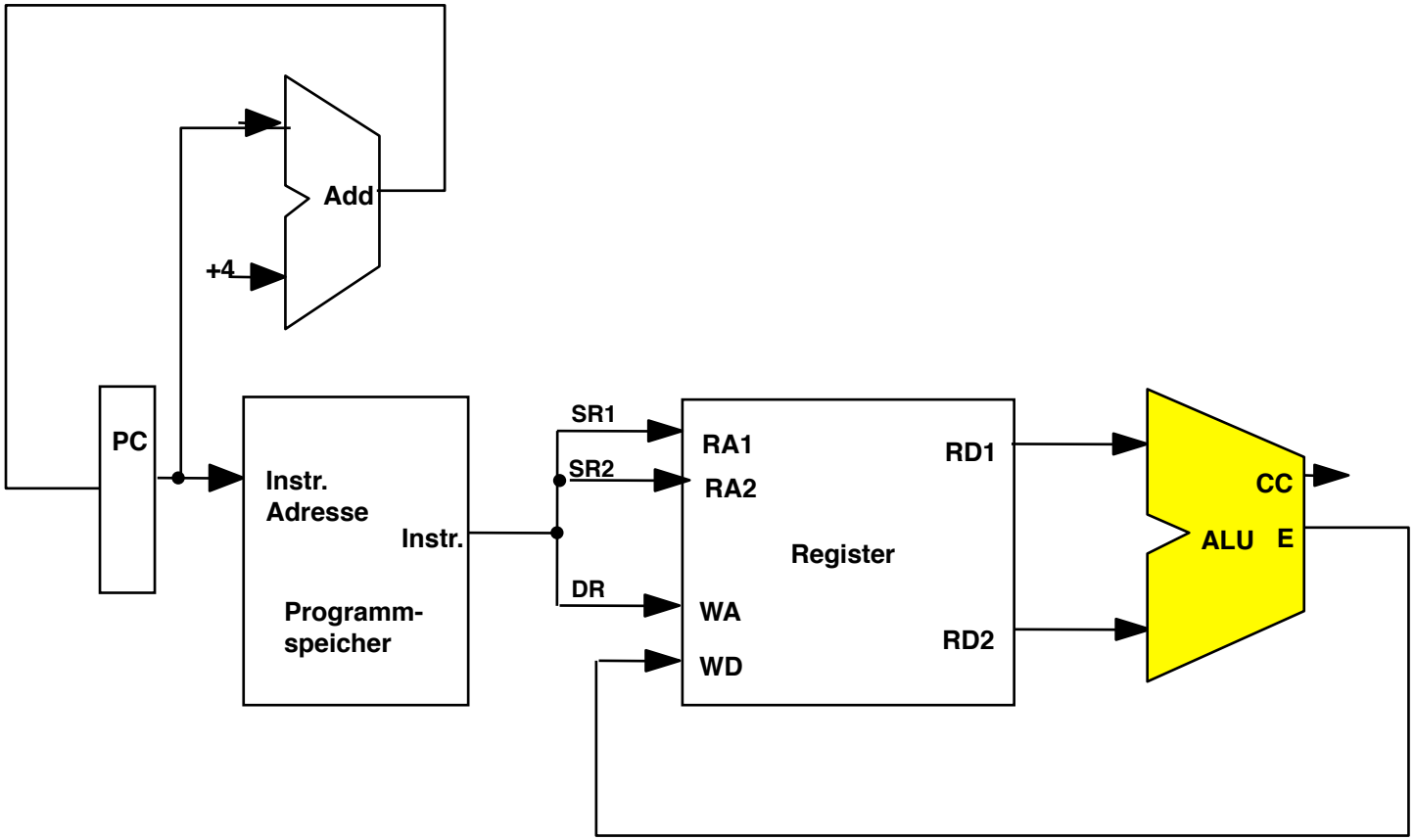
Dekodierungsphase (ID):

Auswahl der Register

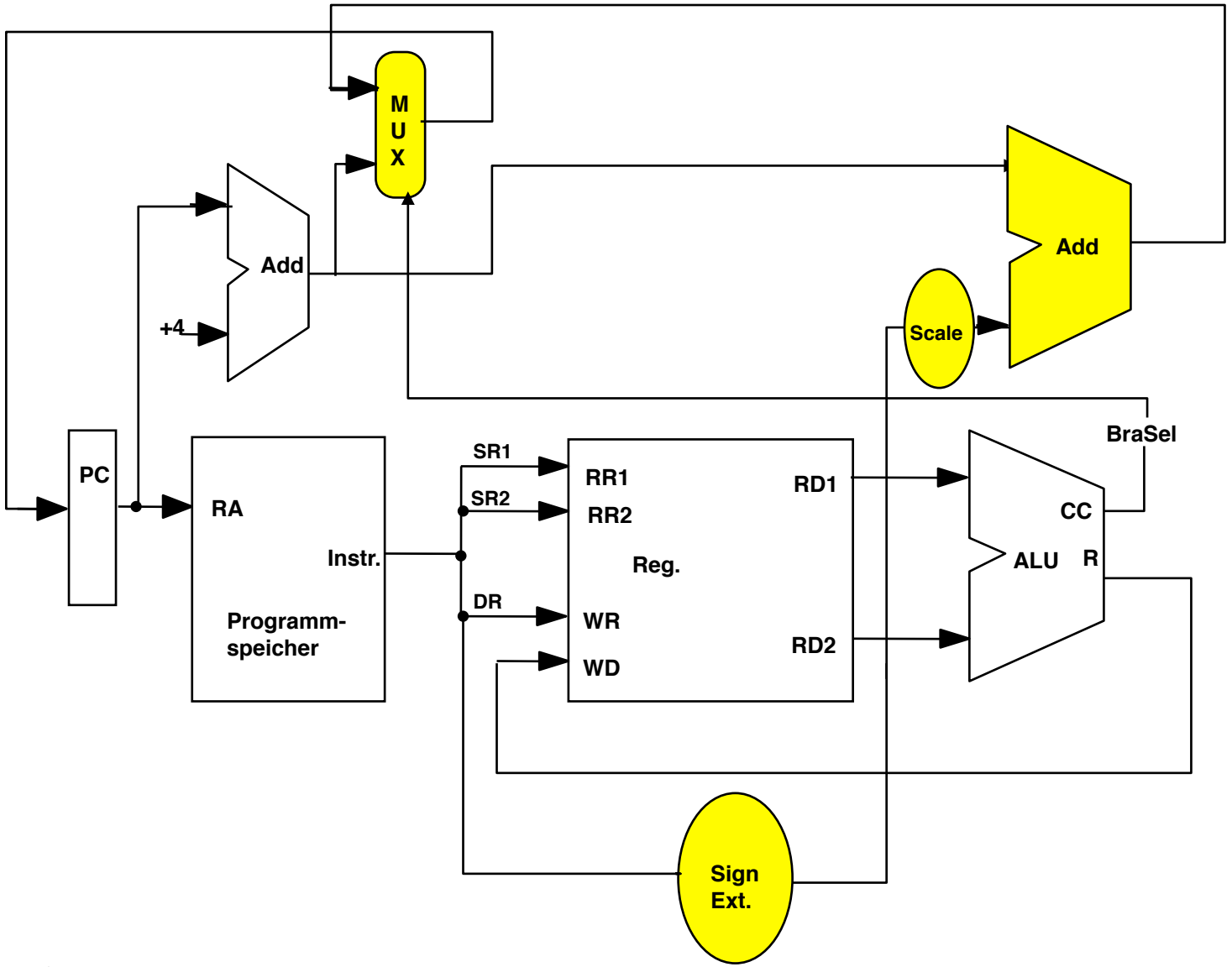


Ausführungsphase (EX)

- Auswahl der ALU-Funktion
- Ausführung

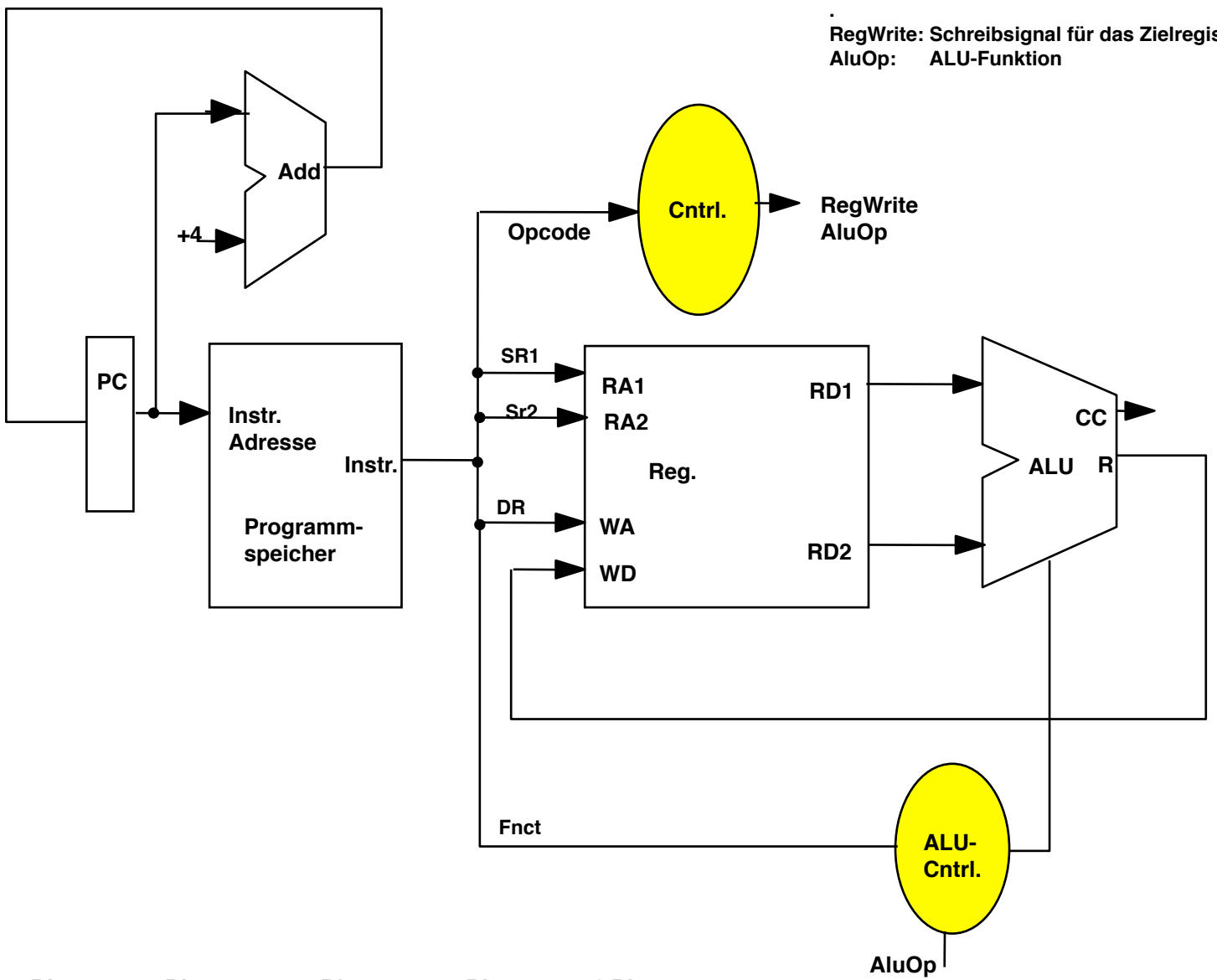


Einbeziehung der Sprungbefehle



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

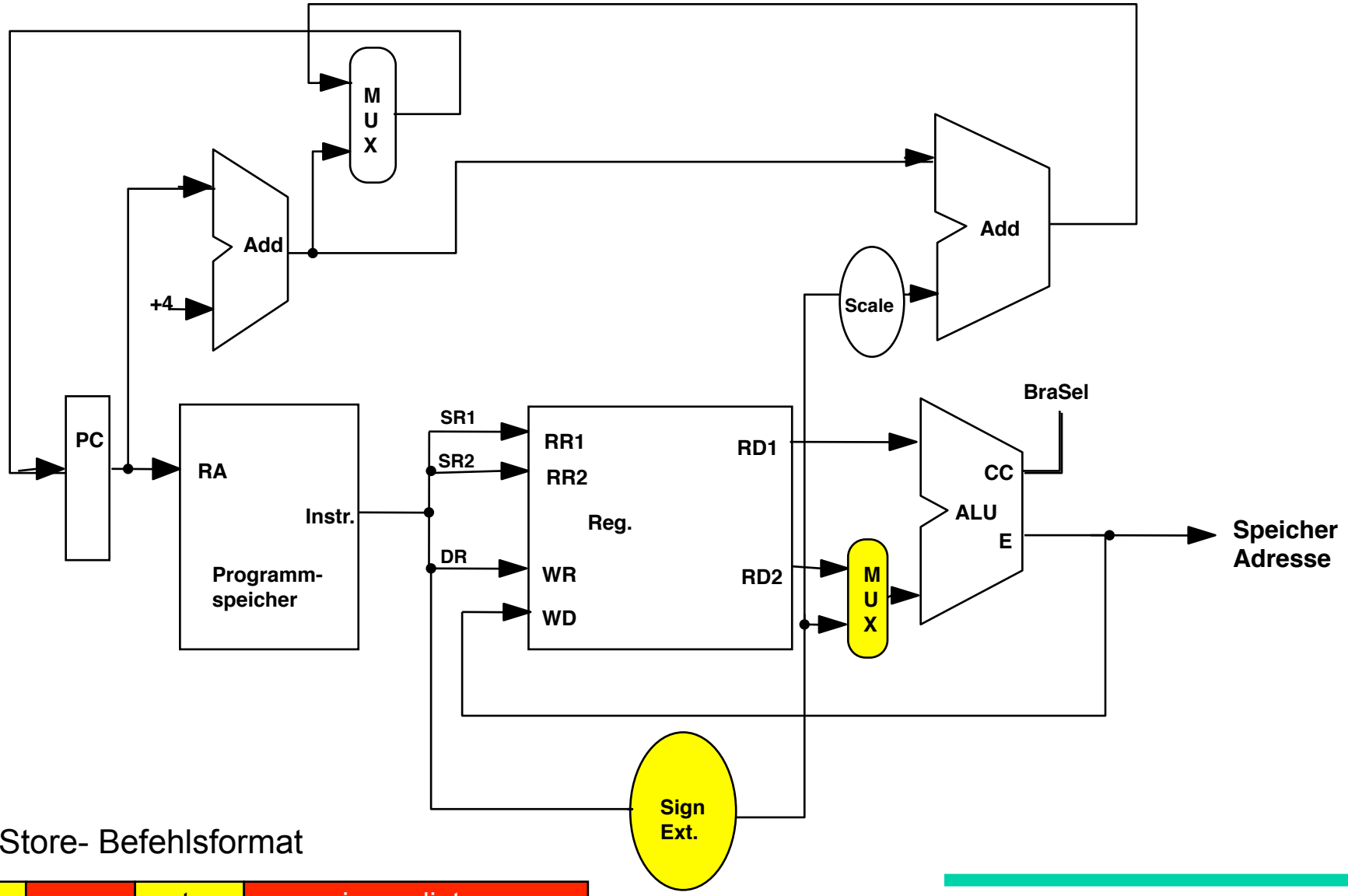
Einbeziehung der Kontrolle



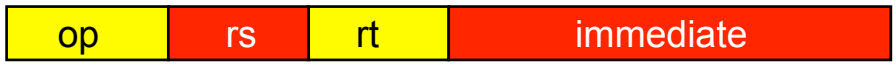
Befehlsformat:

6 Bit	5 Bit	5 Bit	5 Bit	5 Bit	6 Bit
op	rs	rt	rd	shamt	func

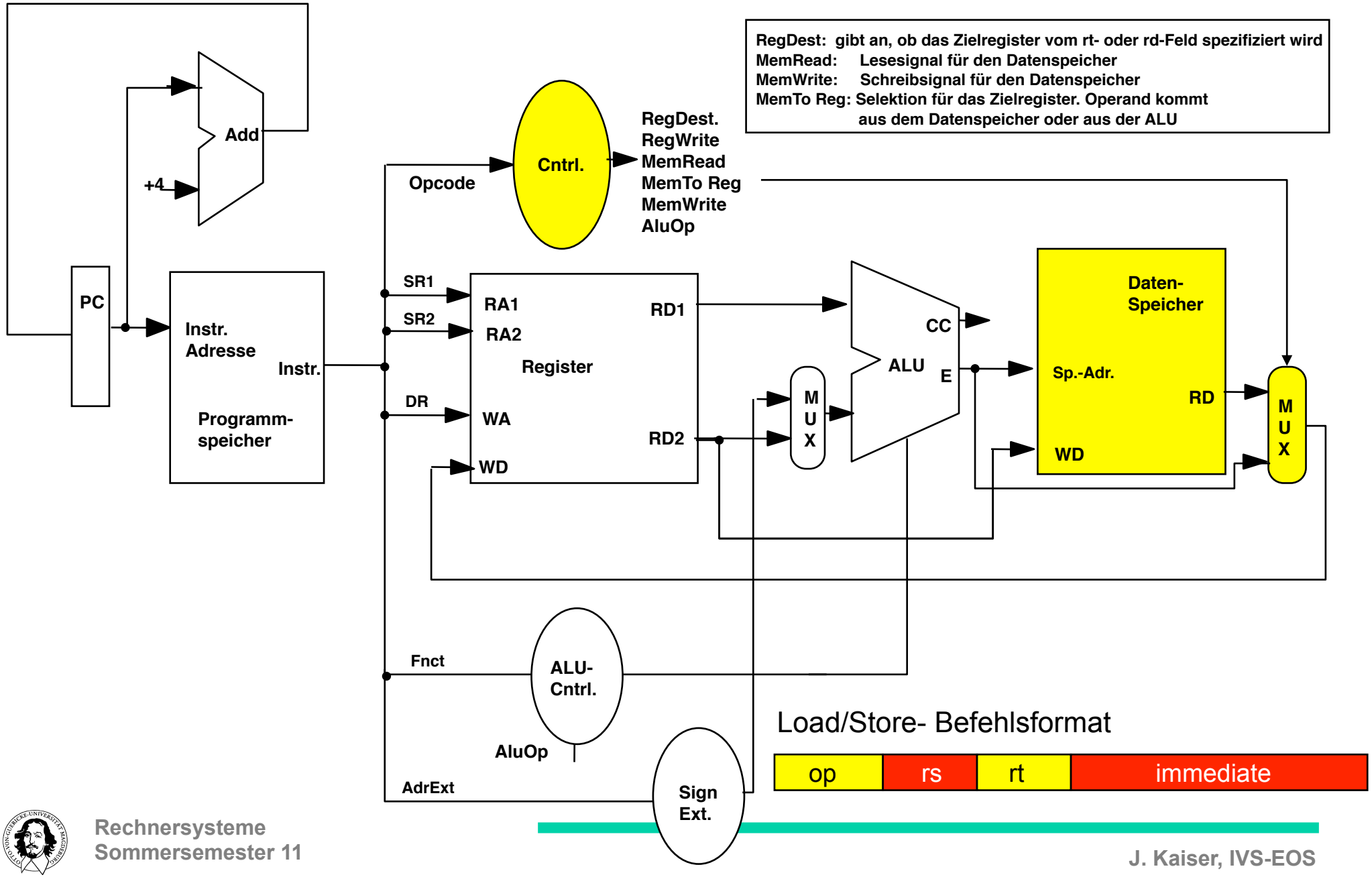
Einbeziehung der Adreßberechnung für die Datenphase

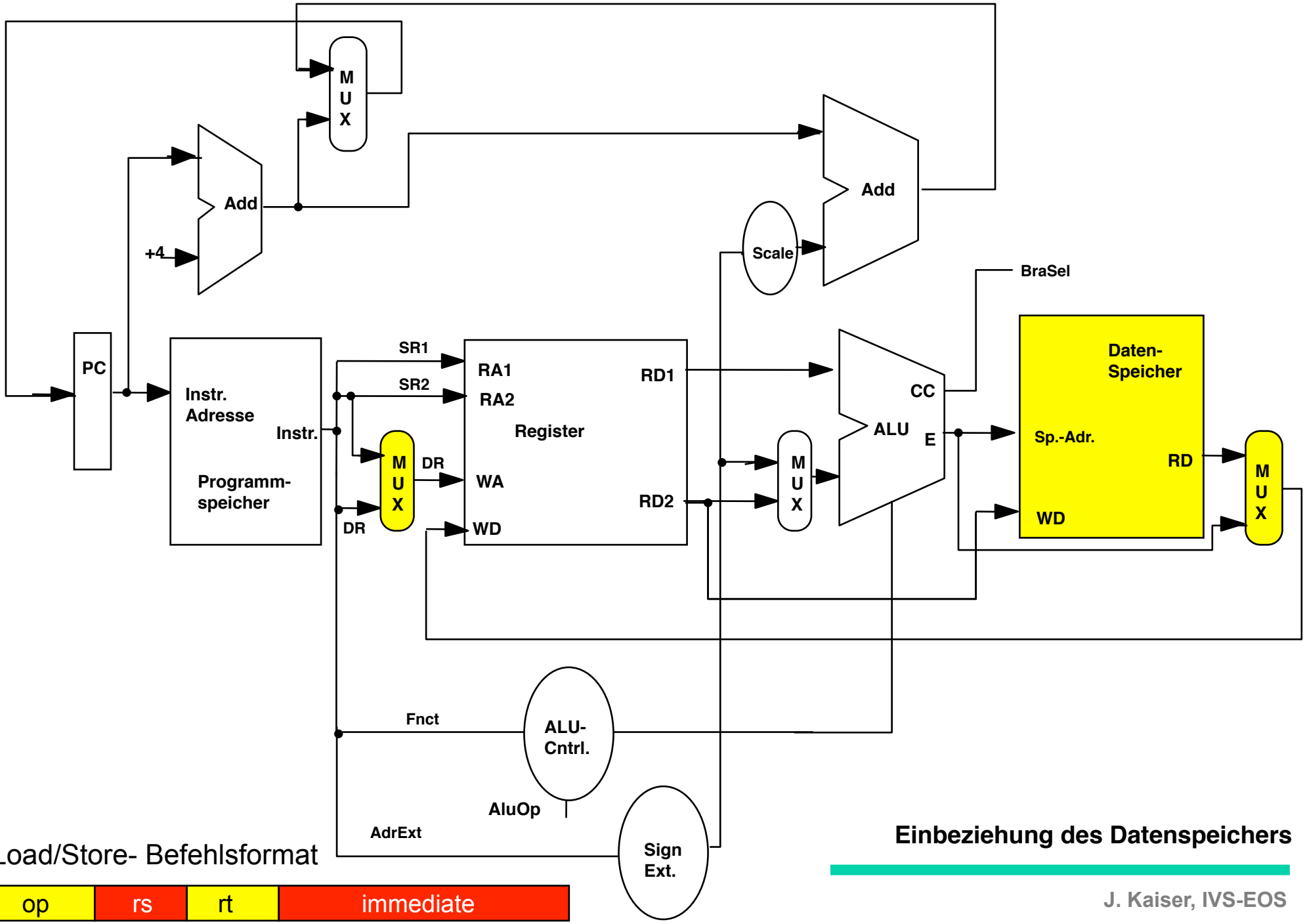


Load/Store- Befehlsformat



Einbeziehung des Datenspeichers



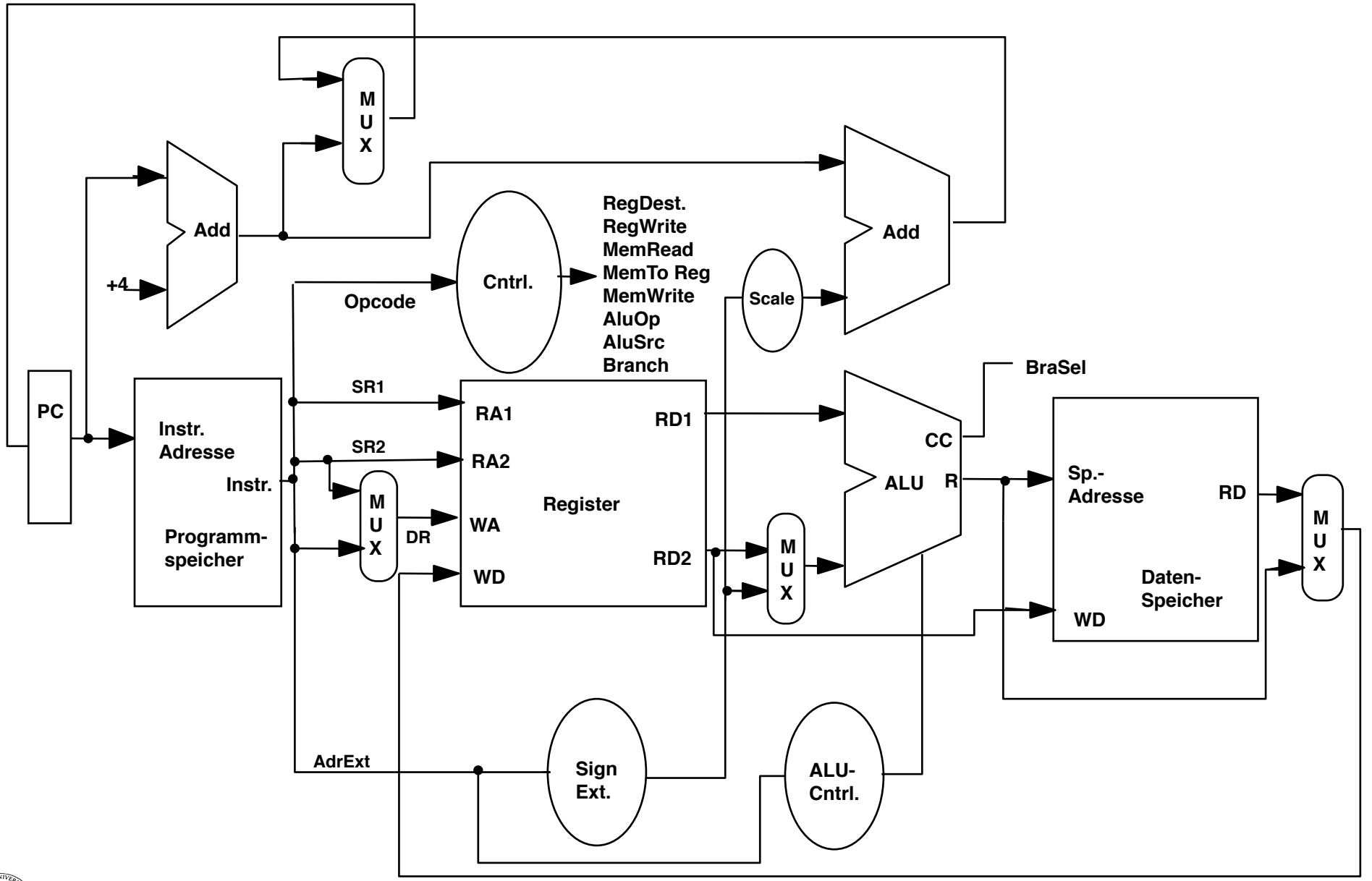


Load/Store- Befehlsformat

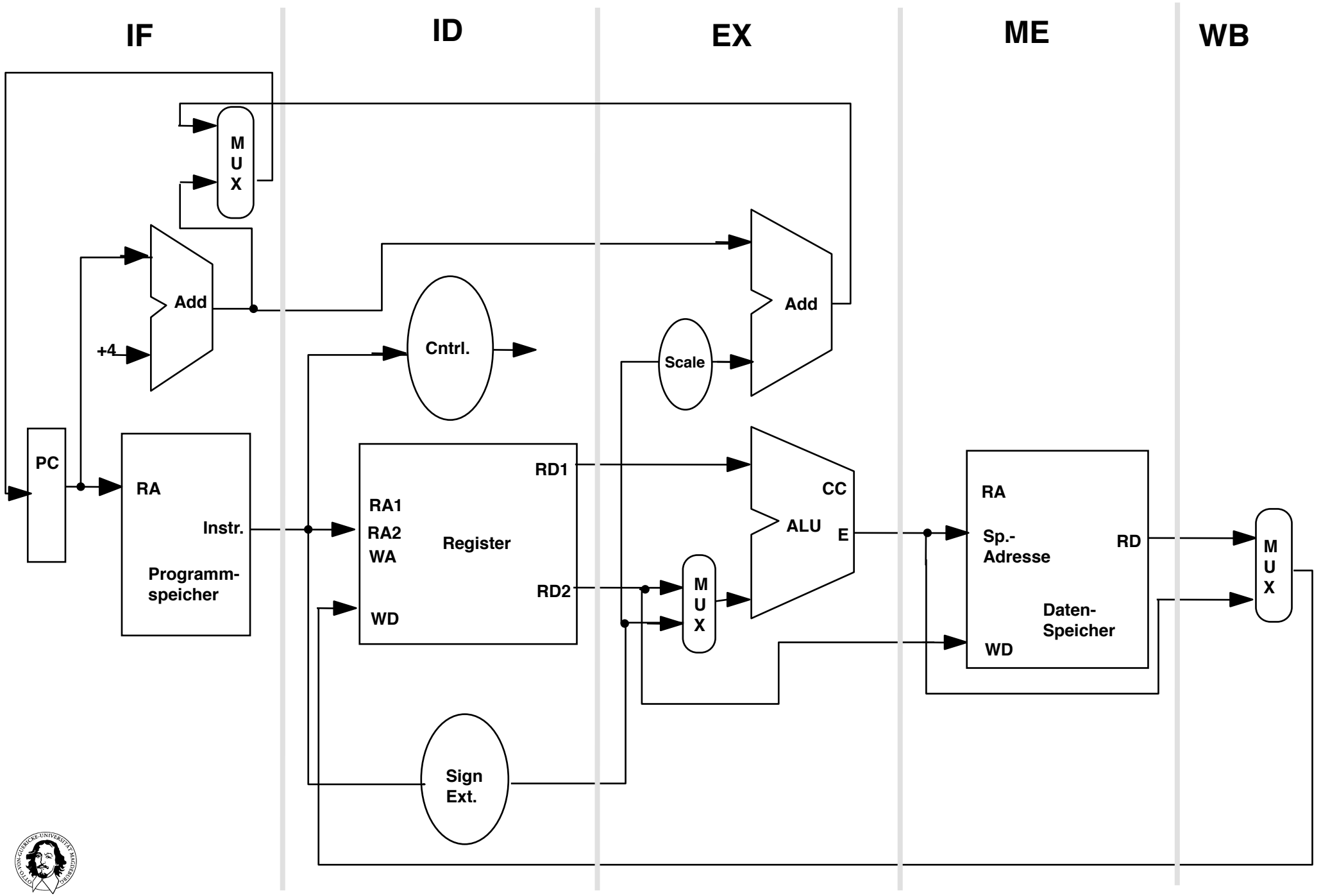
op	rs	rt	immediate

Einbeziehung des Datenspeichers

Vollständiger Datenpfad



Aufbau der Pipeline-Stufen


IF
ID
EX
ME
WB
M
U
X
Add
+4
Cntrl.
Scale
Add
PC
RA
Instr.
Programm-
speicher
RA1
RA2
WA
WD
Register
RD1
RD2
CC
ALU
E
M
U
X
RA
Sp.-
Adresse
RD
Daten-
Speicher
WD
M
U
X
Sign
Ext.

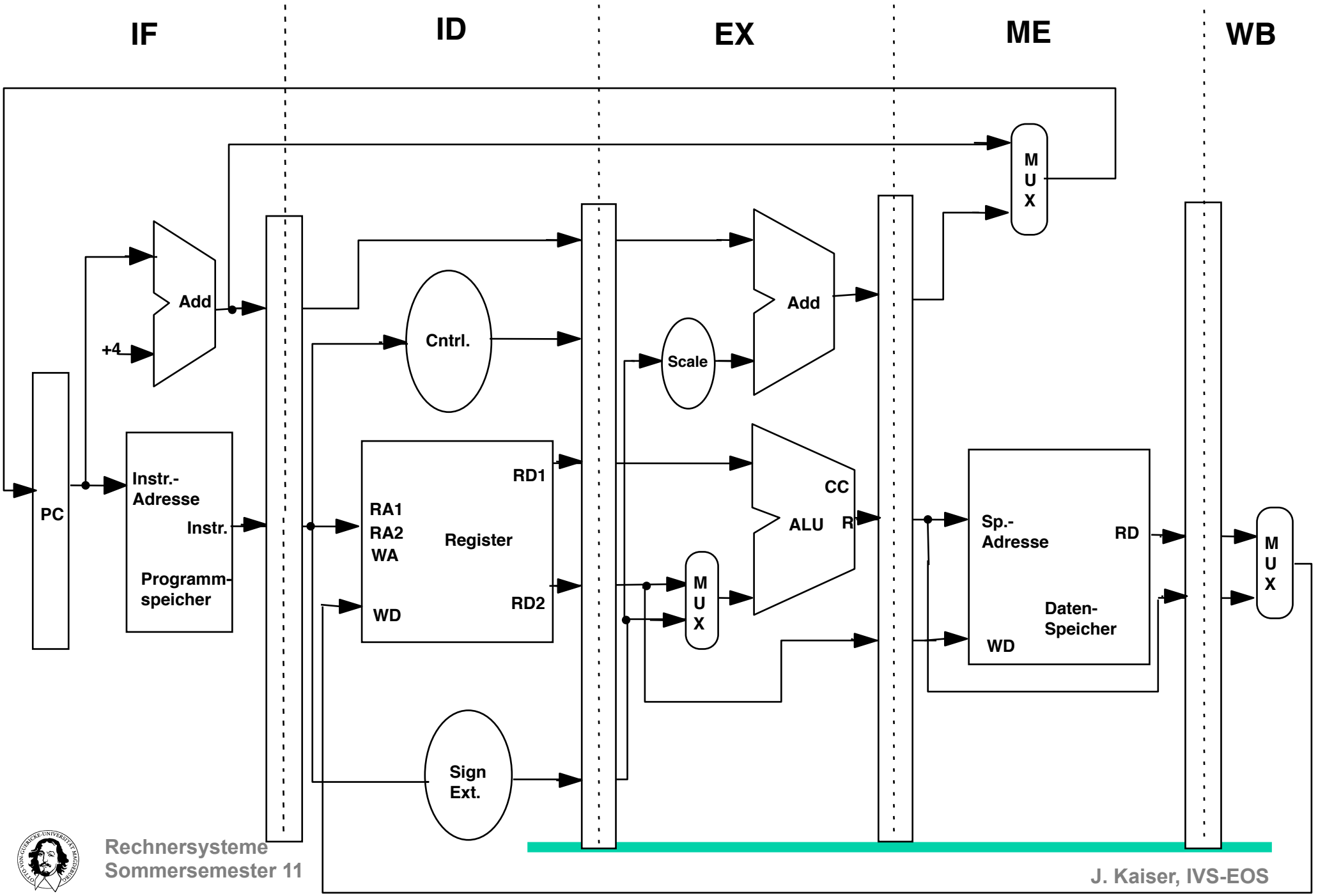
Pipeline-Register

Während der Bearbeitung eines Einzelschritts in Stufe S der Pipeline müssen die Eingangssignale $\{E_S\}$ in diese Stufe stabil sein. Die Eingangs-Signale $\{E_S\}$ 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 Zwischespeicherung werden sogenannte *Pipeline-Register* eingesetzt.


IF
ID
EX
ME
WB
Add
+4
PC
Instr.-
Adresse
Instr.
Programm-
speicher
Cntrl.
RA1
RA2
WA
WD
RD1
RD2
Register
Sign
Ext.
Scale
Add
M
U
X
CC
ALU
R
Sp.-
Adresse
RD
Daten-
Speicher
WD
M
U
X
M
U
X

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:

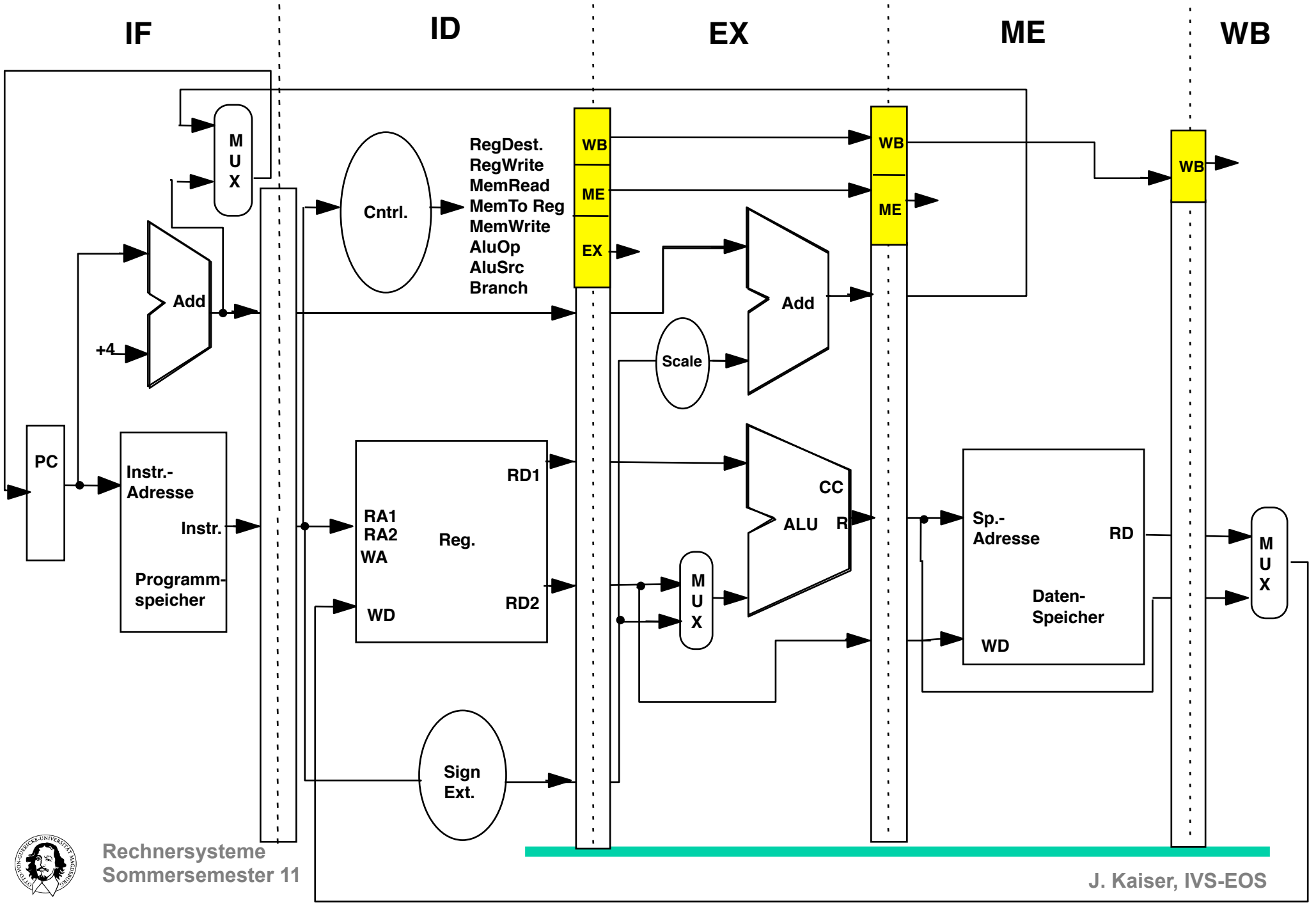
- **MemToReg:** Selektion für das Zielregister - Operand wird aus dem Speicher gelesen oder ist Resultat einer ALU-Operation

Steuersignale für die ALU:

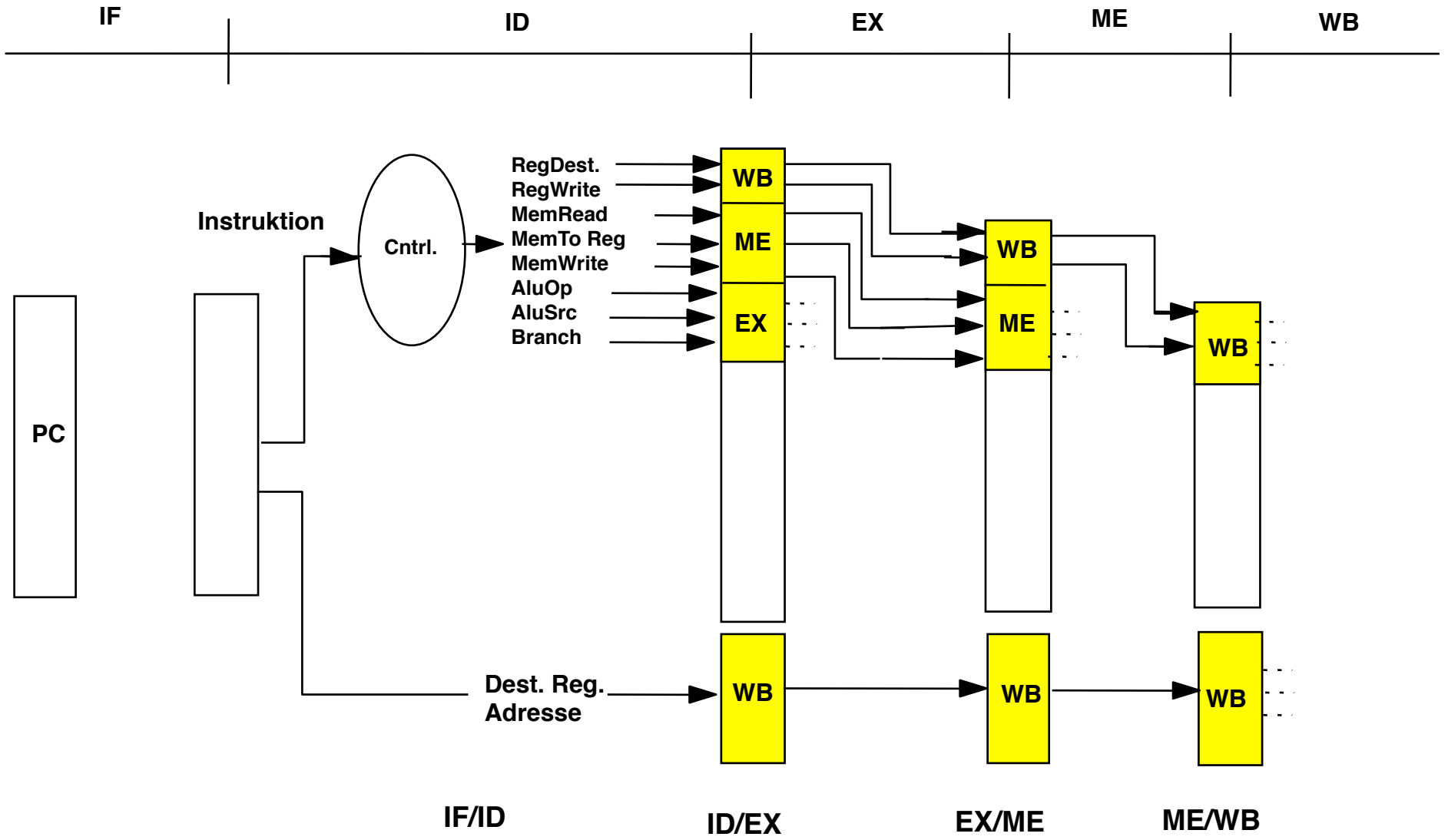
- **AluOp:** Auswahl einer ALU-Funktion
- **AluSrc:** Selektion des 2ten Operanden. Op kommt aus rt oder ist Sign ext. Immediate Op

Steuersignale für Sprünge:

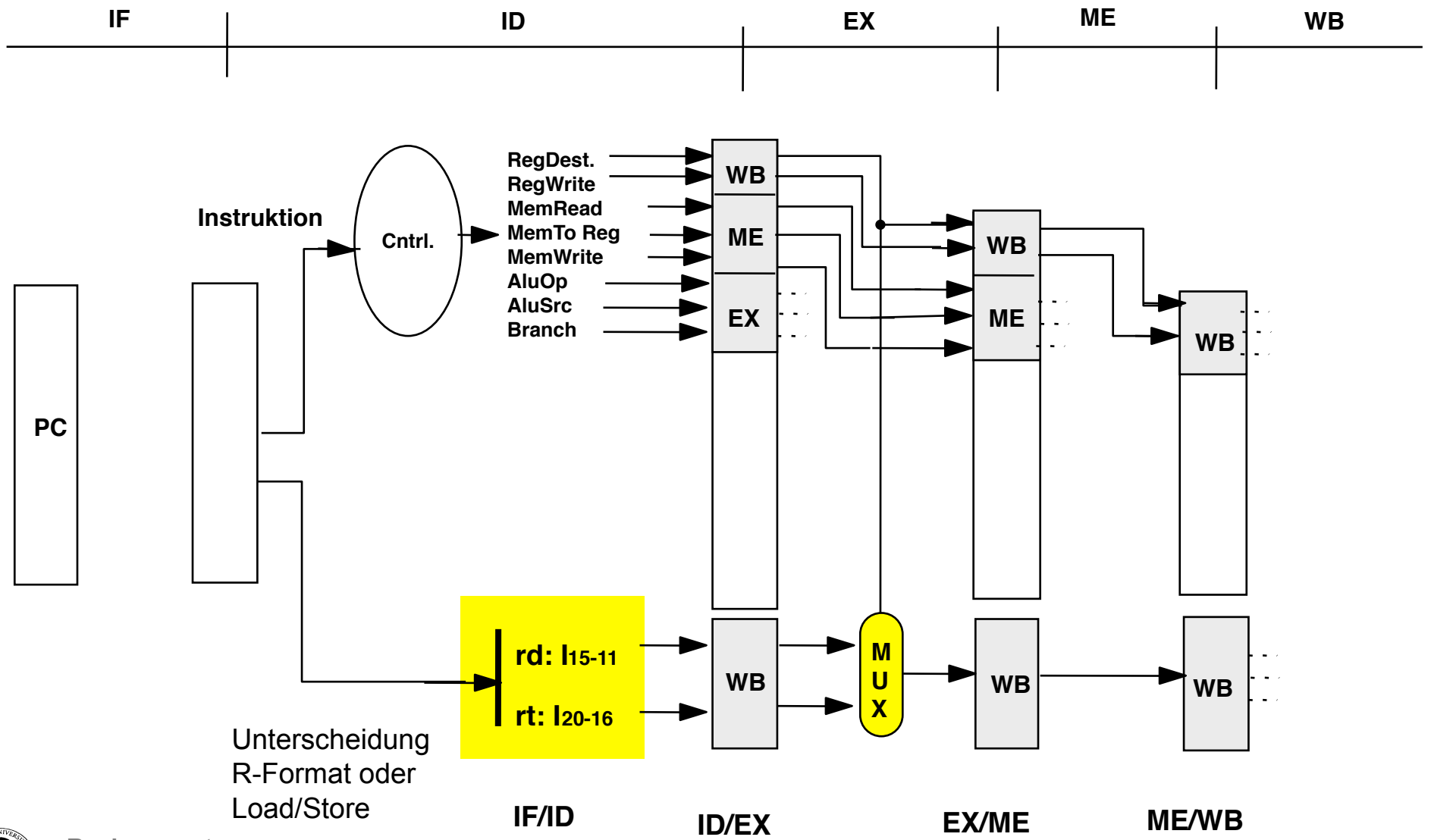
- **Branch:** OPCODE spezifiziert einen Sprung, d.h. es wird eine (Sprung) Adreßberechnung durchgeführt.

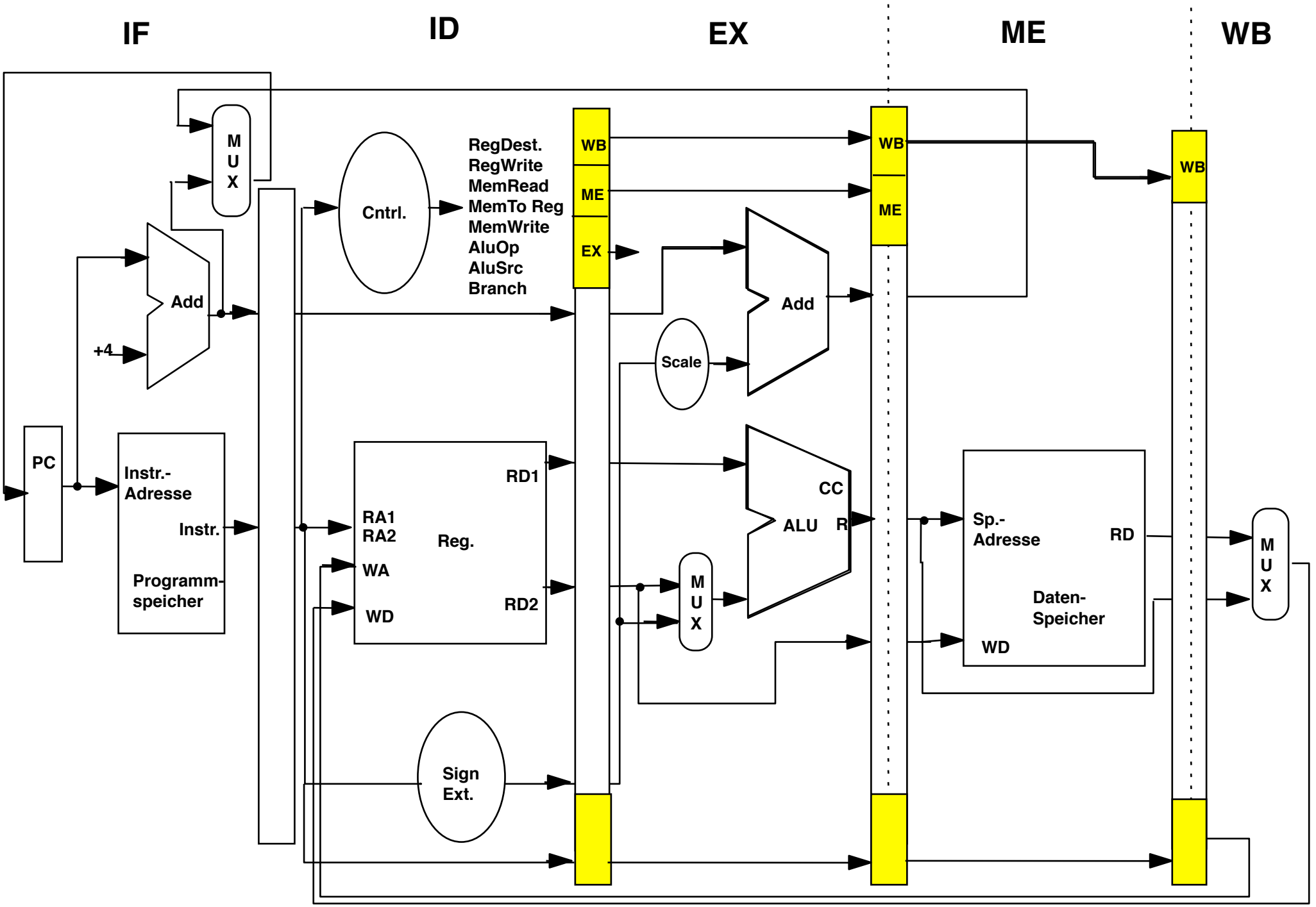

IF
ID
EX
ME
WB
M
U
X
Cntrl.
RegDest.
RegWrite
MemRead
MemTo Reg
MemWrite
AluOp
AluSrc
Branch
WB
ME
EX
Add
+4
Scale
PC
Instr.-
Adresse
Instr.
Programm-
speicher
RA1
RA2
WA
WD
Reg.
RD1
RD2
CC
ALU
R
Sp.-
Adresse
RD
Daten-
Speicher
Sign
Ext.

Puffern der Kontrollsignale und der Adresse des Zielregisters

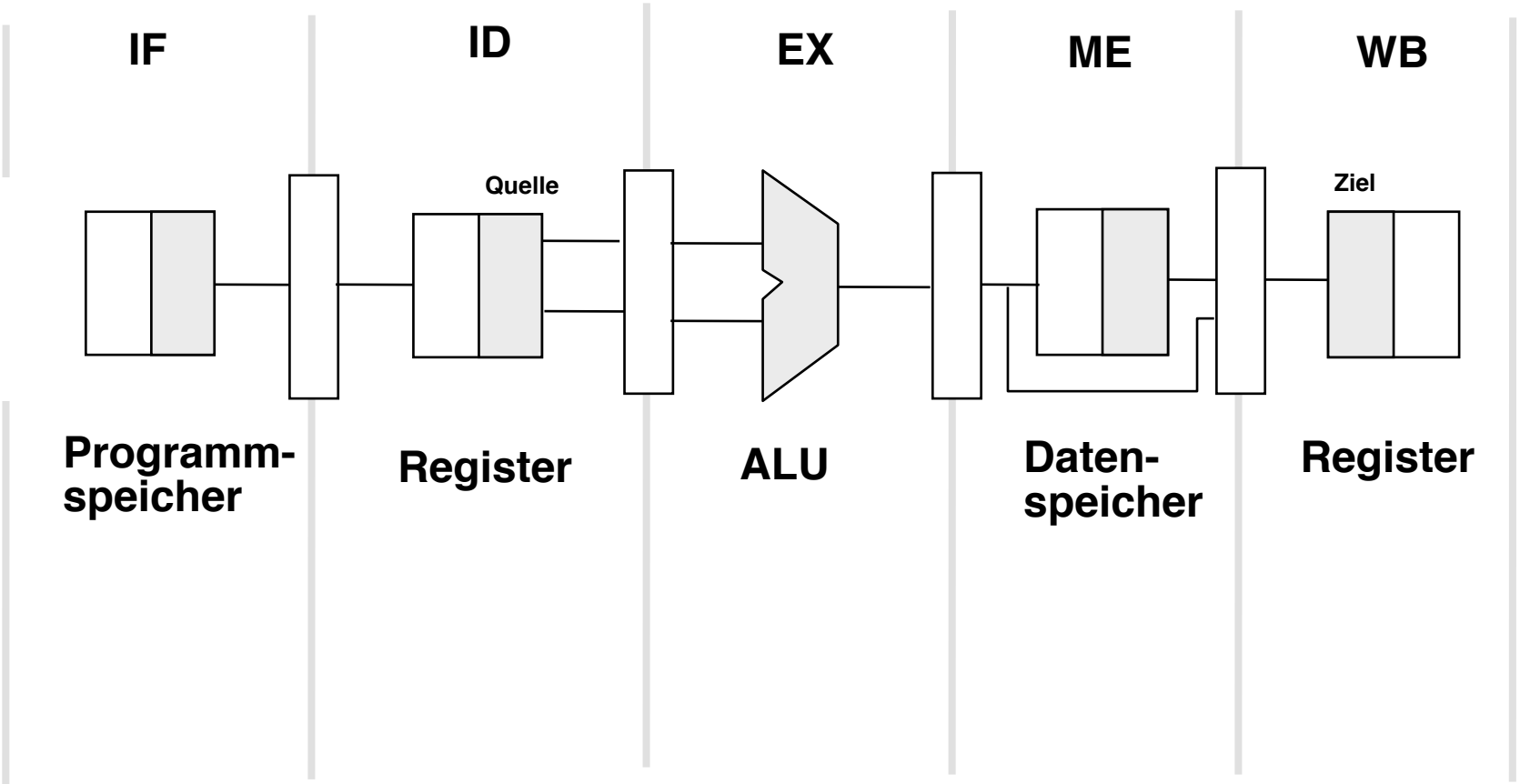


Puffern der Kontrollsignale und der Adresse des Zielregisters



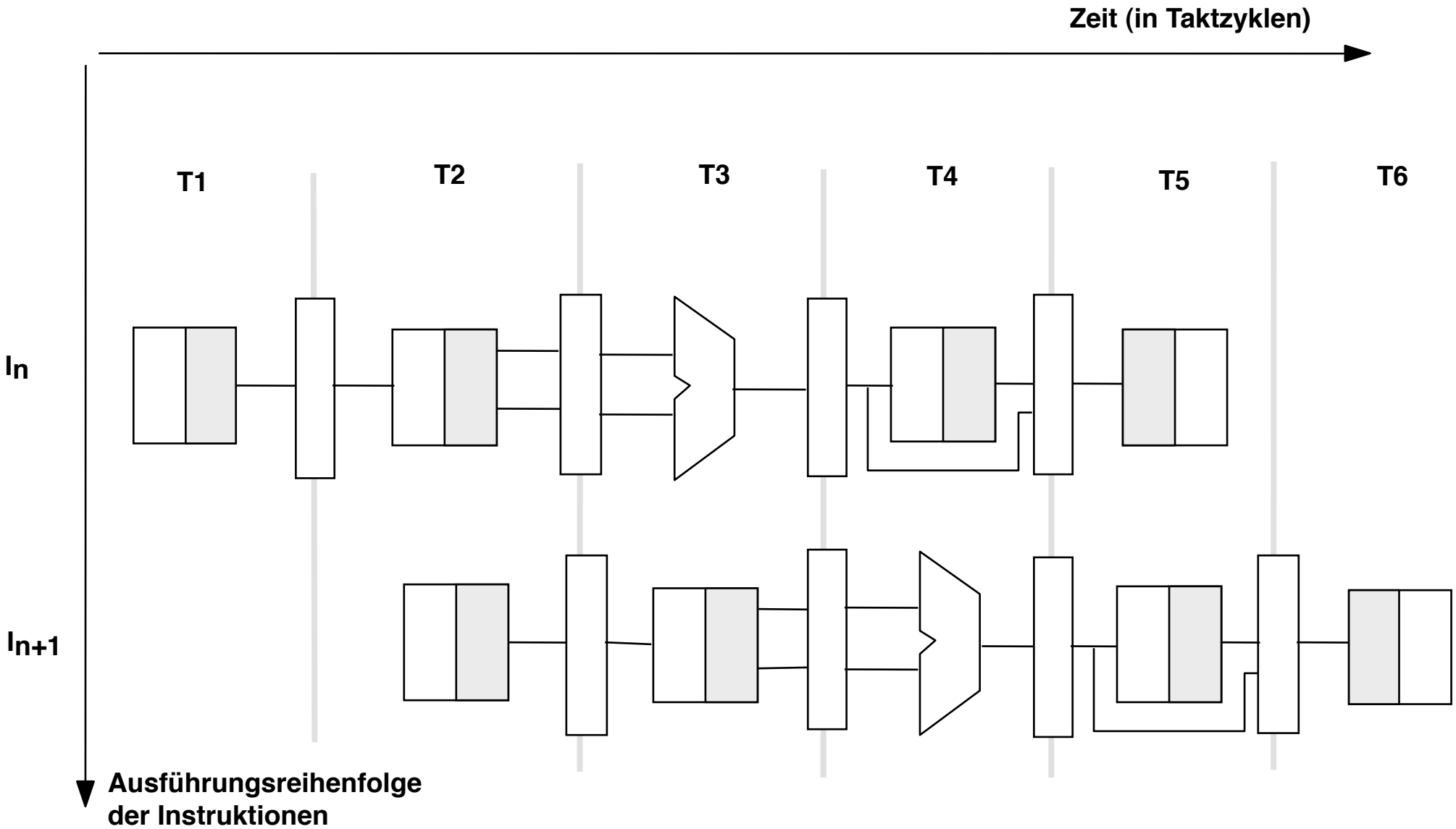

IF
ID
EX
ME
WB
M
U
X
Cntrl.
RegDest.
RegWrite
MemRead
MemTo Reg
MemWrite
AluOp
AluSrc
Branch
WB
ME
EX
Add
+4
Scale
Add
PC
Instr.-
Adresse
Instr.
Programm-
speicher
RA1
RA2
WA
WD
Reg.
RD1
RD2
CC
ALU
R
M
U
X
Sp.-
Adresse
RD
Daten-
Speicher
WD
Sign
Ext.
M
U
X

Vereinfachung der Darstellung einer Pipeline



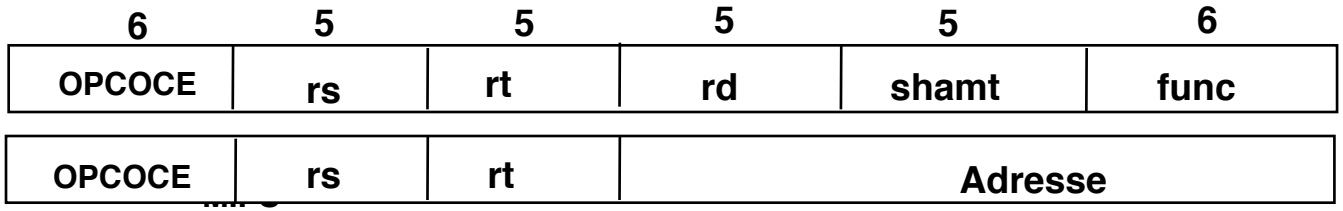
Pipeline-Hindernisse: Datenabhängigkeiten (Data Hazards)

Darstellung einer Pipeline in einem Multi-Zyklus Diagramm



MIPS

Befehls-format

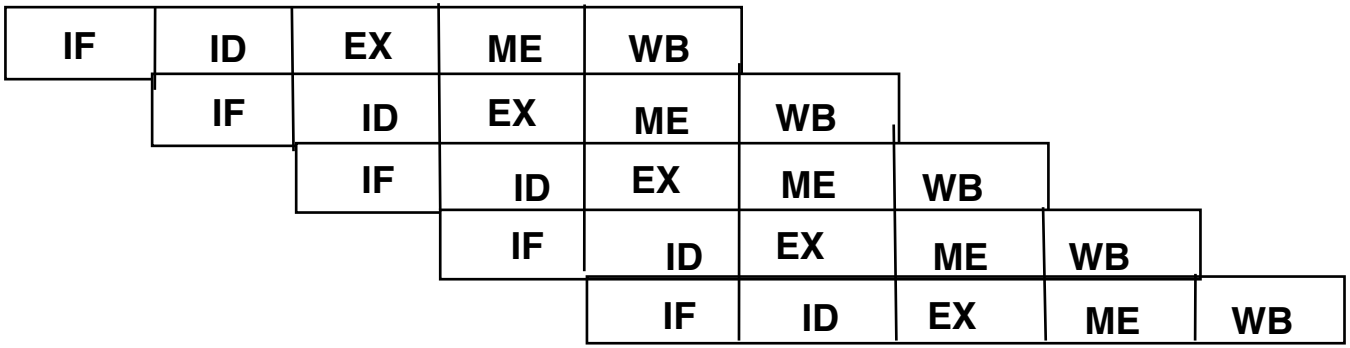


rs: 1.Op, rt: 2. Op, rd: Zielreg.

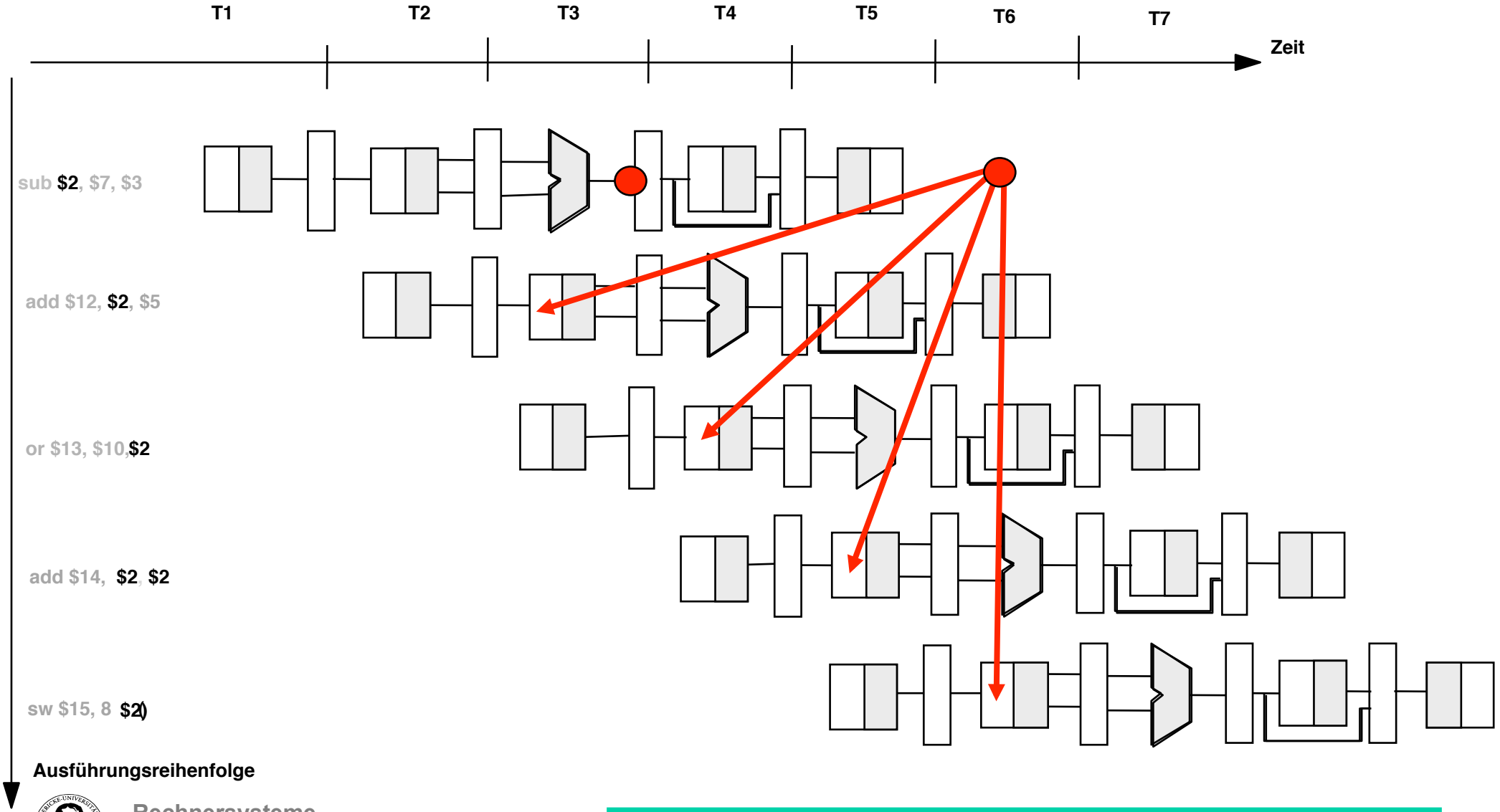
Beisp.:

sub $2, $7, $3	Subtrahiere Inhalt von Reg 3 von Reg.7 und speichere das Resultat in Reg 2
and $12, $2, $5	Logisches UND von Reg 5 und Reg 2; Resultat nach Reg 12
or $13, $10, $2	Logisches ODER von Reg 2 und Reg 10; Resultat nach Reg 13
add $14, $2, $2	Addiere den Inhalt von Reg 2 zum Inhalt von Reg 2; Resultat nach Reg 14
sw $15, 8($2)	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)

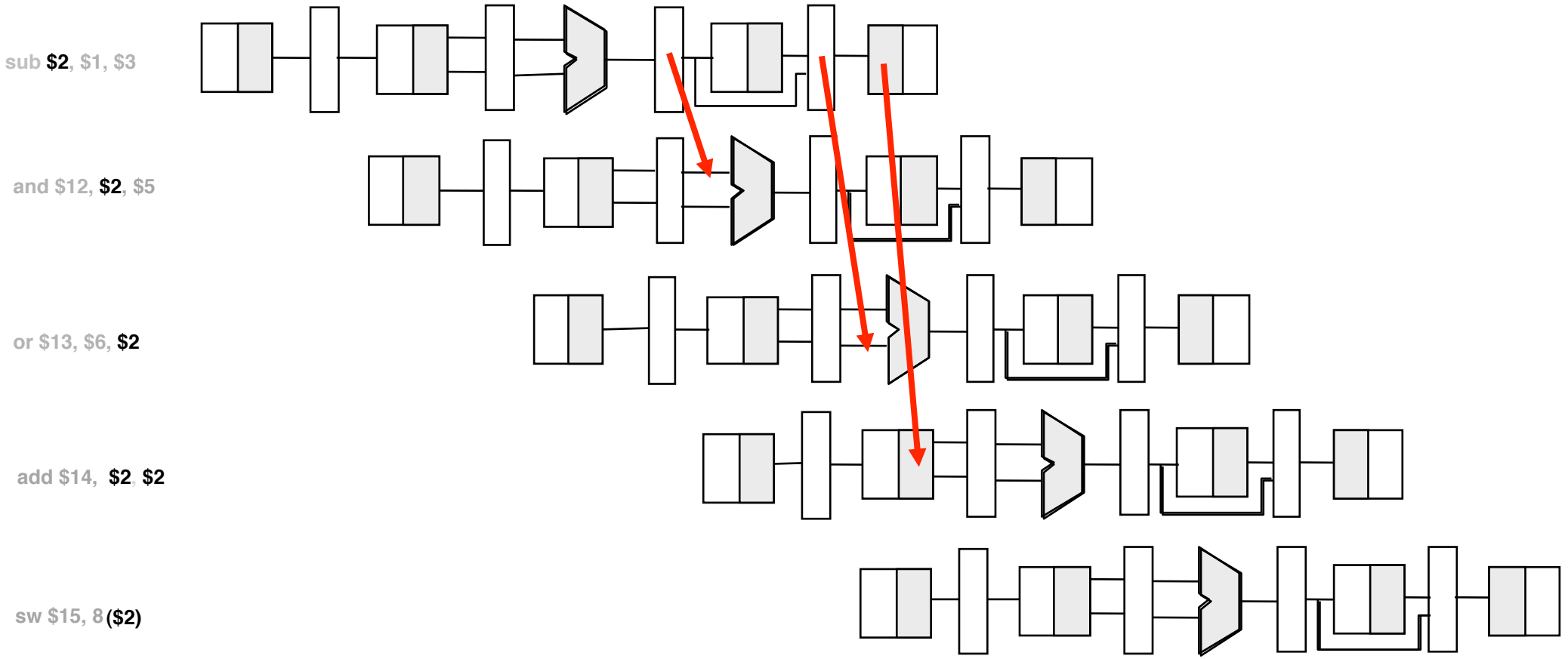


Pipeline - Hindernisse: Datenabhängigkeiten



Datenabhängigkeiten zwischen den Stufen der Pipeline

	T1	T2	T3	T4	T5	T6	T7	T8	T9
Inhalt Reg. $2	10	10	10	10	10	-25	-25	-25	-25
Wert in EX Stufe	x	x	x	-25	x	x	x	x	x
Wert in ME Stufe	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)

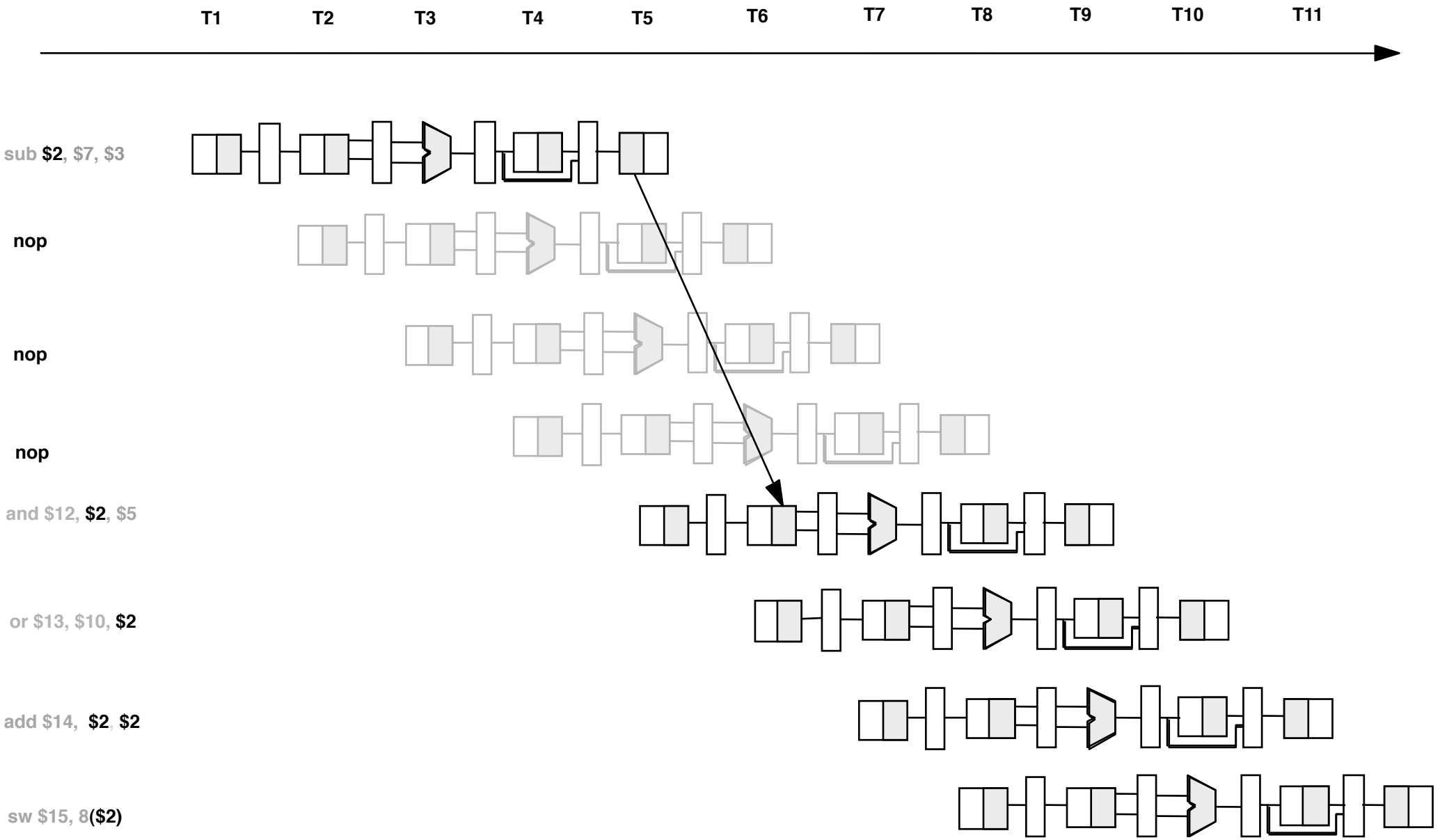
Hardware: Ausnutzung der Bedingungen für die Datenabhängigkeit, um eine automatische Erkennung der Abhängigkeiten während der Laufzeit zu ermöglichen.

Beispielprogramm:

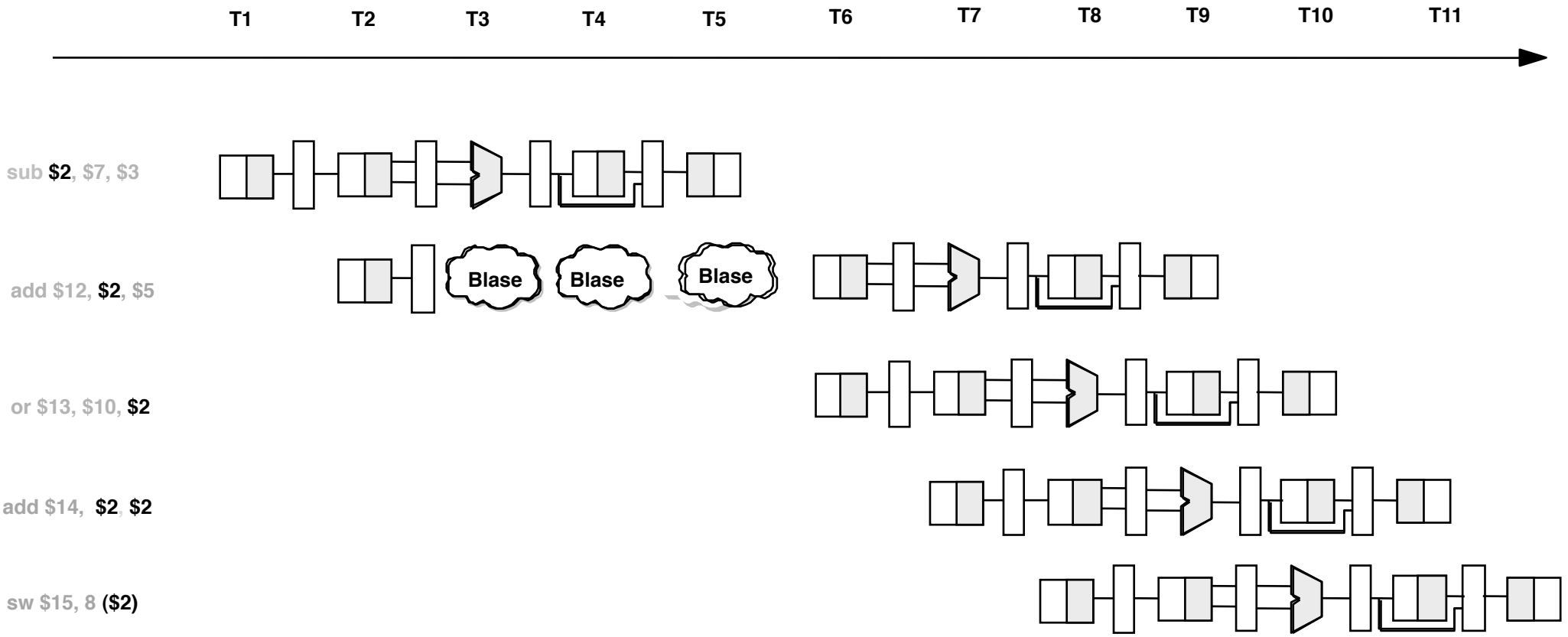
```
sub $2, $7, $3
and $12, $2, $5
or $13, $10, $2
add $14, $2, $2
sw  $15, 8($2)
```

```
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



Pipeline - Hindernisse: Datenabhängigkeiten



Bedingungen für Datenabhängigkeit:

EX

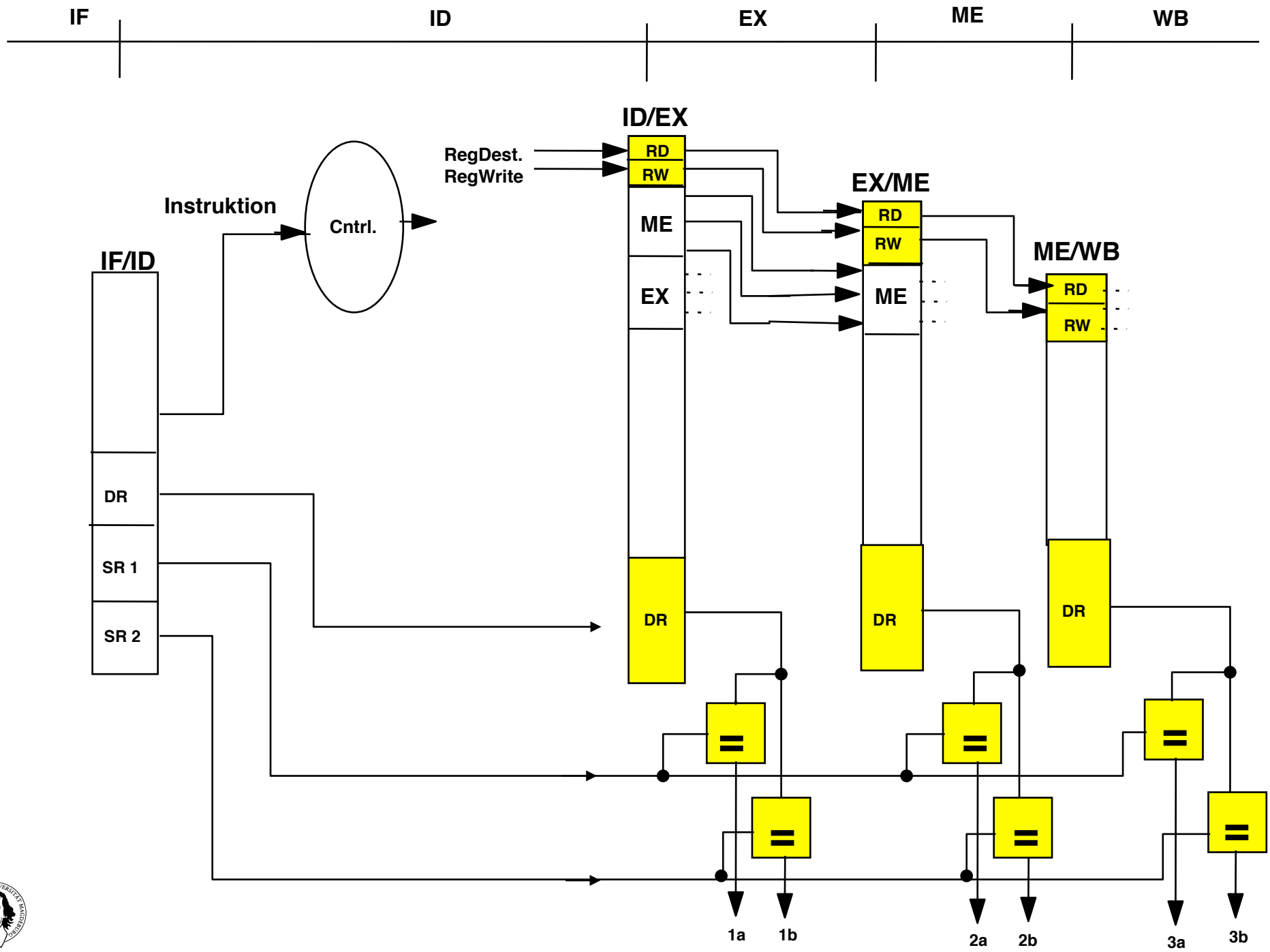
In der EX-Stufe befindet sich eine Instruktion, die ein Resultat in ein Zielregister schreibt, das von einer nachfolgenden Instruktion als 1. oder 2. Quellregister definiert wird.

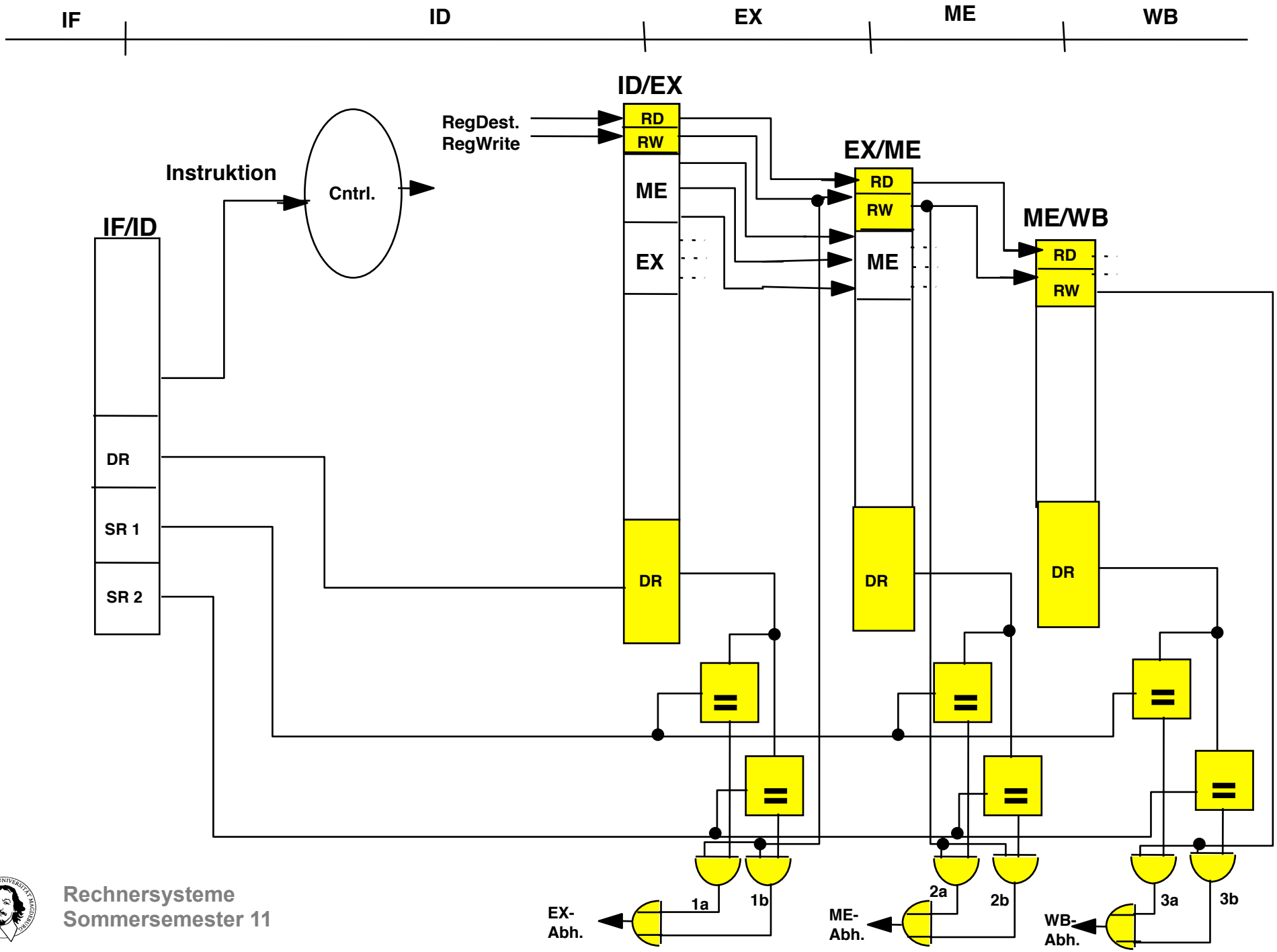
ME

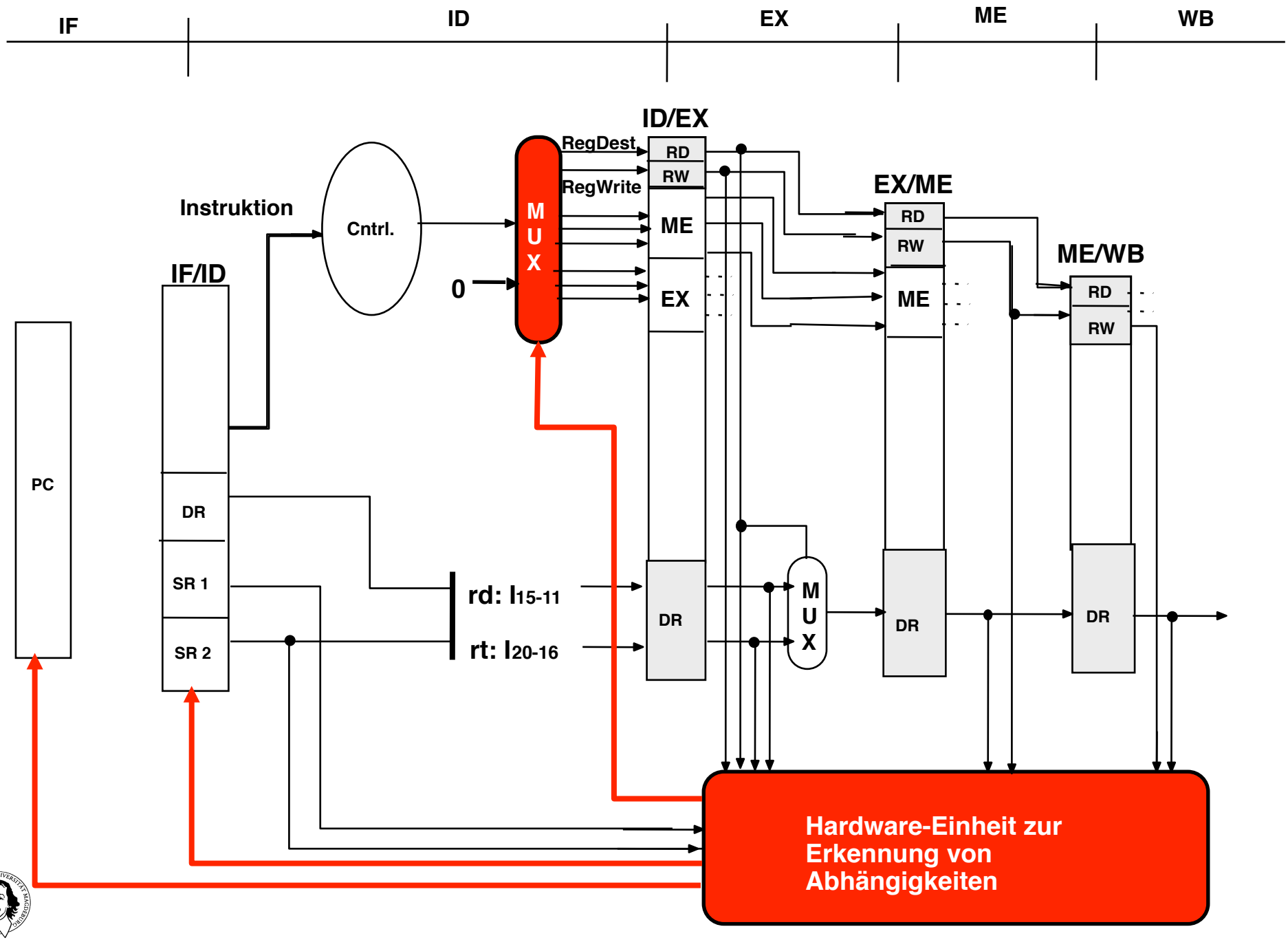
In der ME-Stufe befindet sich eine Instruktion, die ein Resultat in ein Zielregister schreibt, das von einer nachfolgenden Instruktion als 1. oder 2. Quellregister definiert wird.

WB

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


IF
ID
EX
ME
WB
ID/EX
RegDest.
RegWrite
RD
RW
ME
EX
DR
EX/ME
RD
RW
ME
DR
ME/WB
RD
RW
DR
Instruktion
Cntrl.
IF/ID
DR
SR 1
SR 2
=
1a
1b
2a
2b
3a
3b

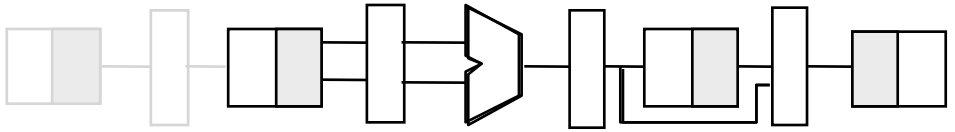

IF
ID
EX
ME
WB
ID/EX
EX/ME
ME/WB
IF/ID
Instruktion
Cntrl.
RegDest.
RegWrite
RD
RW
ME
EX
DR
SR 1
SR 2
1a
1b
2a
2b
3a
3b
EX-
Abh.
ME-
Abh.
WB-
Abh.

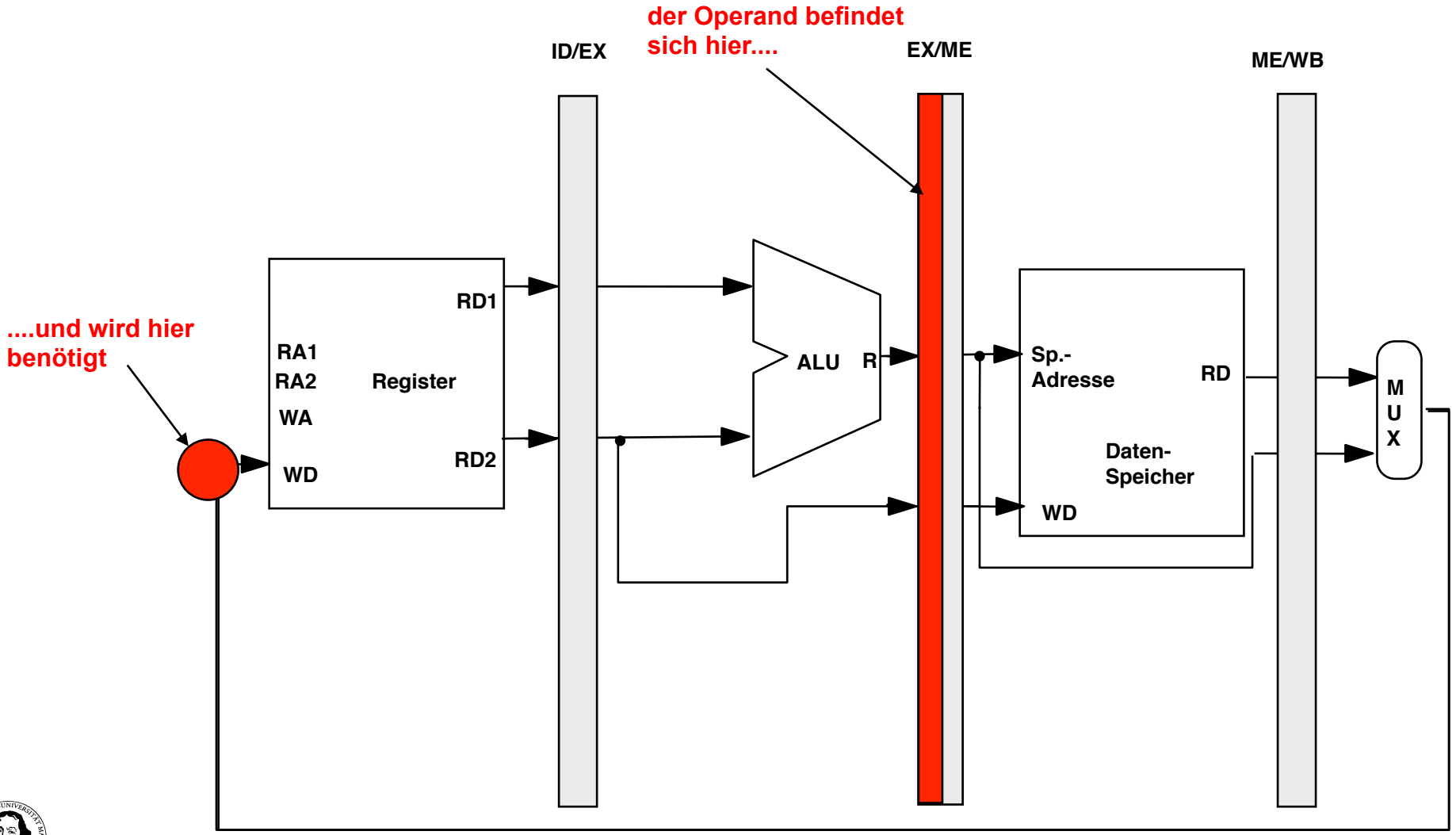

IF
ID
EX
ME
WB
ID/EX
RegDest
RegWrite
RD
RW
ME
EX
EX/ME
RD
RW
ME
ME/WB
RD
RW
Instruktion
Cntrl.
M
U
X
0
IF/ID
PC
DR
SR 1
SR 2
rd: I15-11
rt: I20-16
DR
M
U
X
DR
DR
Hardware-Einheit zur
Erkennung von
Abhängigkeiten

Eine Einheit zur Umgehung von Datenabhängigkeiten (Bypass)

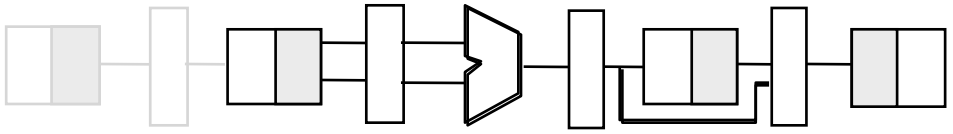
Forwarding

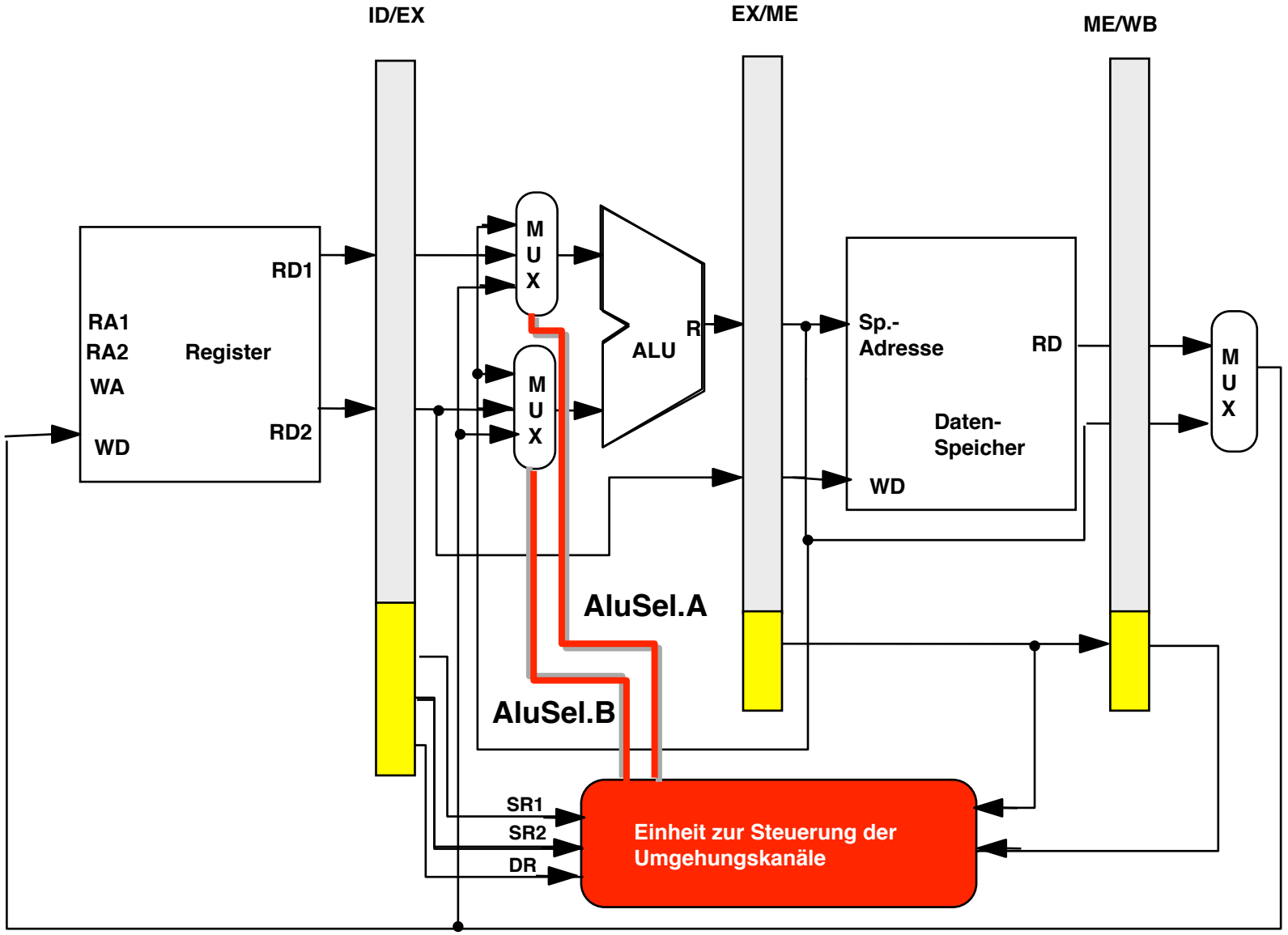
Relevante Stufen der Pipeline ohne Kontrollsignale





Pipeline mit “Umgehungskanälen“ (Bypass) zur Weiteleitung (Forwarding) von Resultaten





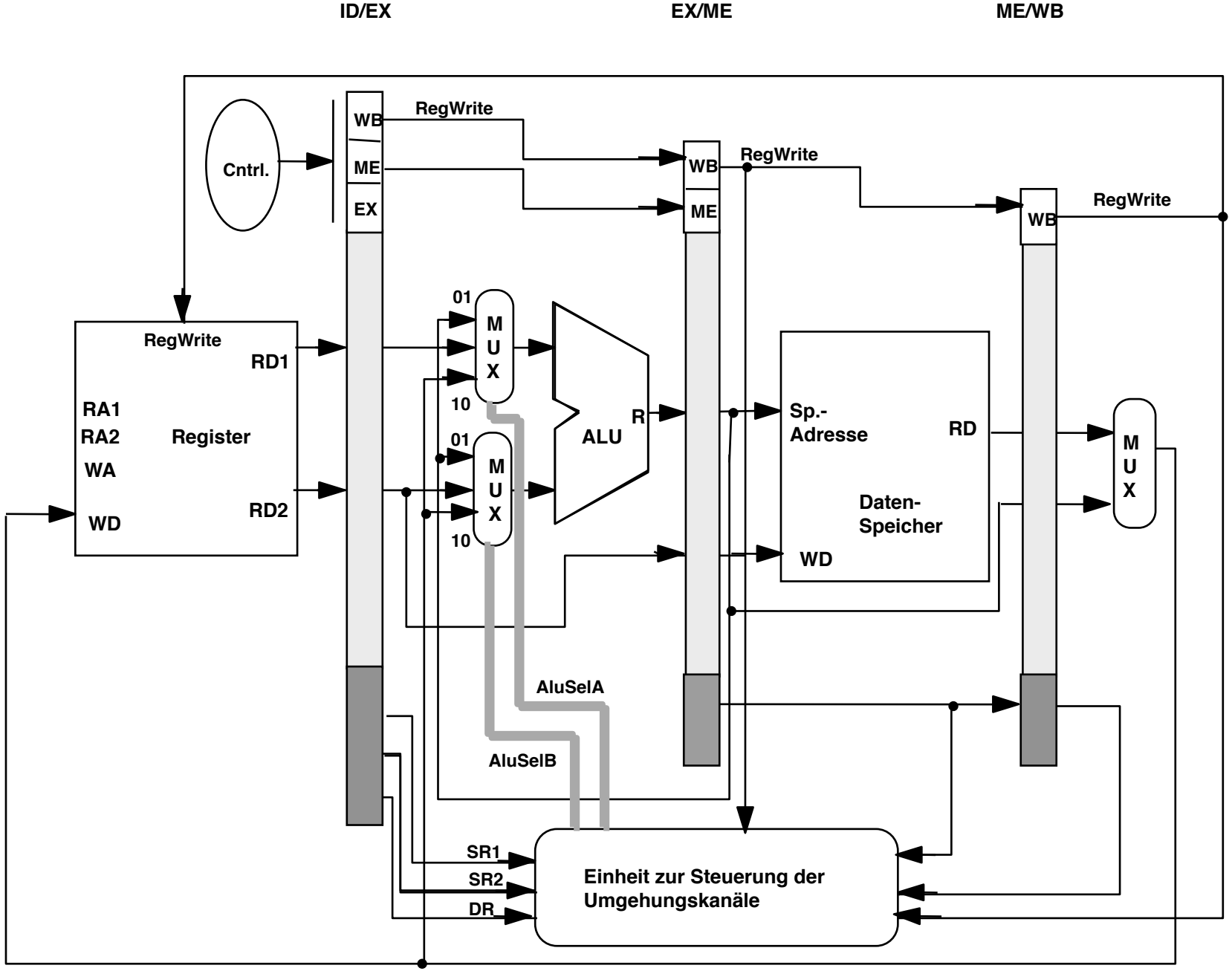
Bedingungen zur Steuerung der Umleitungskanäle

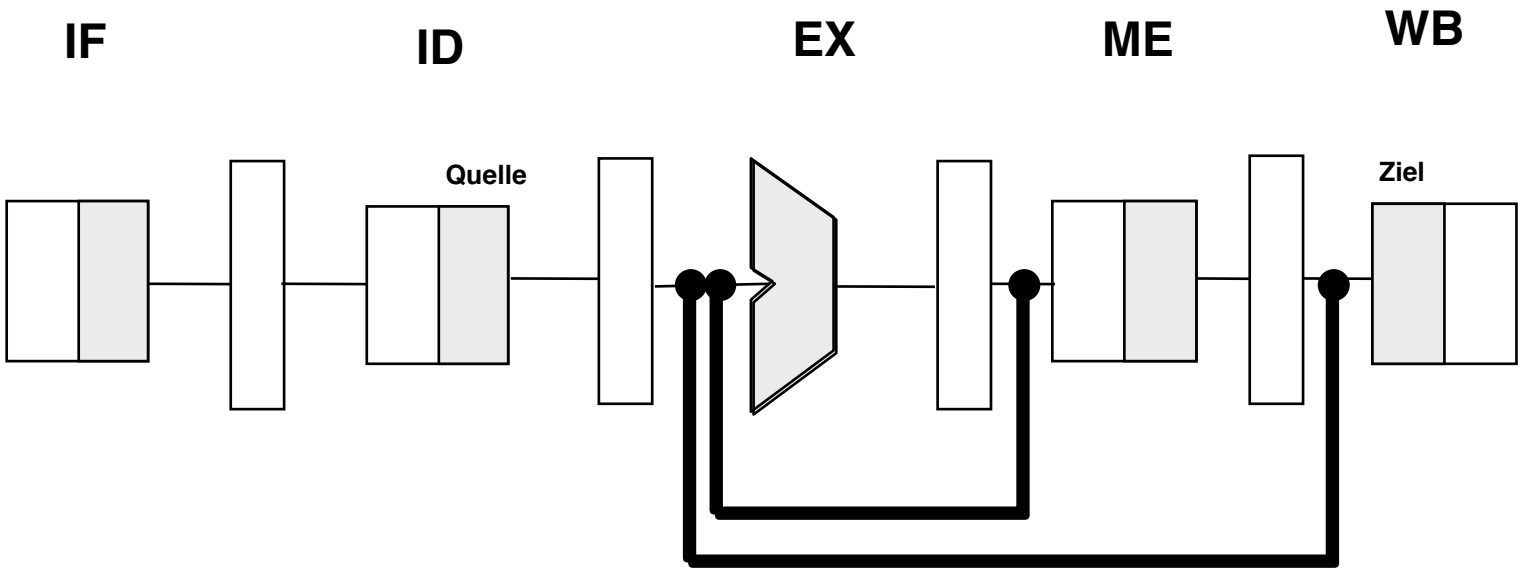
EX-Abhängigkeit:

(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:

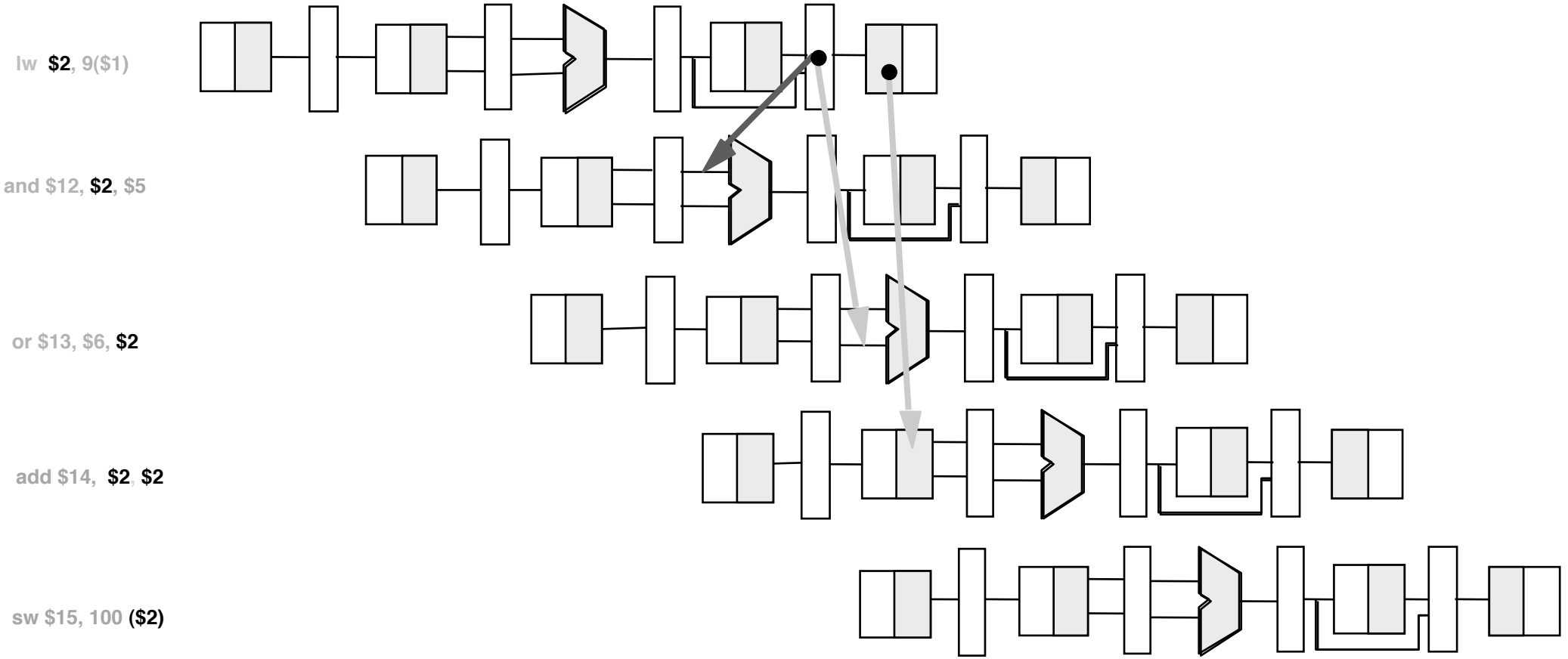
(ME/WB.RegWrite • (ME/WB.WriteReg = ID/EX.ReadReg1) : AluSel.A ← 10
(ME/WB.RegWrite • (ME/WB.WriteReg = ID/EX.ReadReg2) : AluSel.B ← 10


ID/EX
EX/ME
ME/WB
Cntrl.
WB
ME
EX
RegWrite
WB
ME
RegWrite
WB
RegWrite
RegWrite
RD1
RA1
RA2
Register
WA
WD
RD2
01
10
M
U
X
01
10
M
U
X
ALU
R
Sp.-
Adresse
RD
Daten-
Speicher
WD
M
U
X
AluSelA
AluSelB
SR1
SR2
DR
Einheit zur Steuerung der
Umgehungskanäle

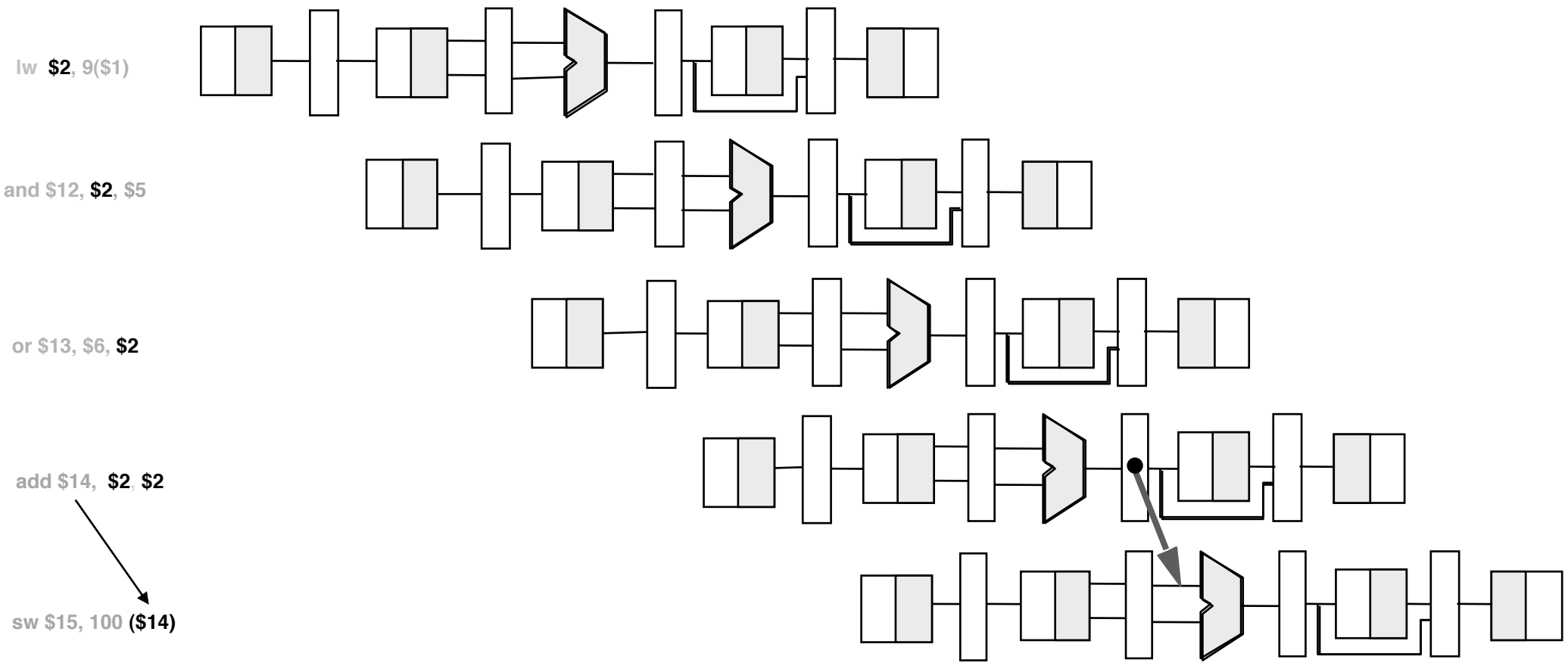

IF
ID
EX
ME
WB
Quelle
Ziel

Datenabhängigkeiten zwischen den Stufen der Pipeline bei Load- Befehlen.

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

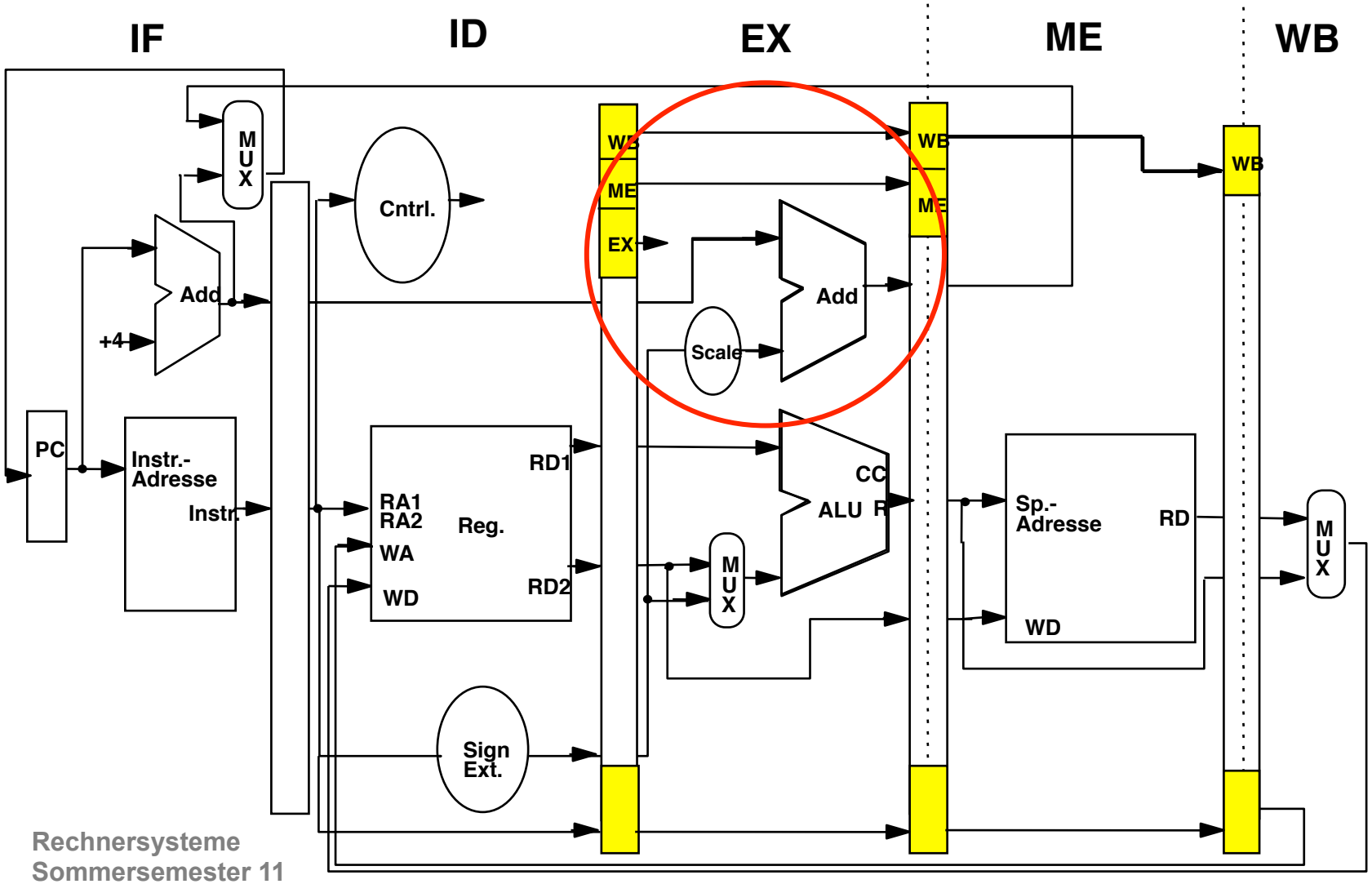


Datenabhängigkeiten zwischen den Stufen der Pipeline bei Store-Befehlen

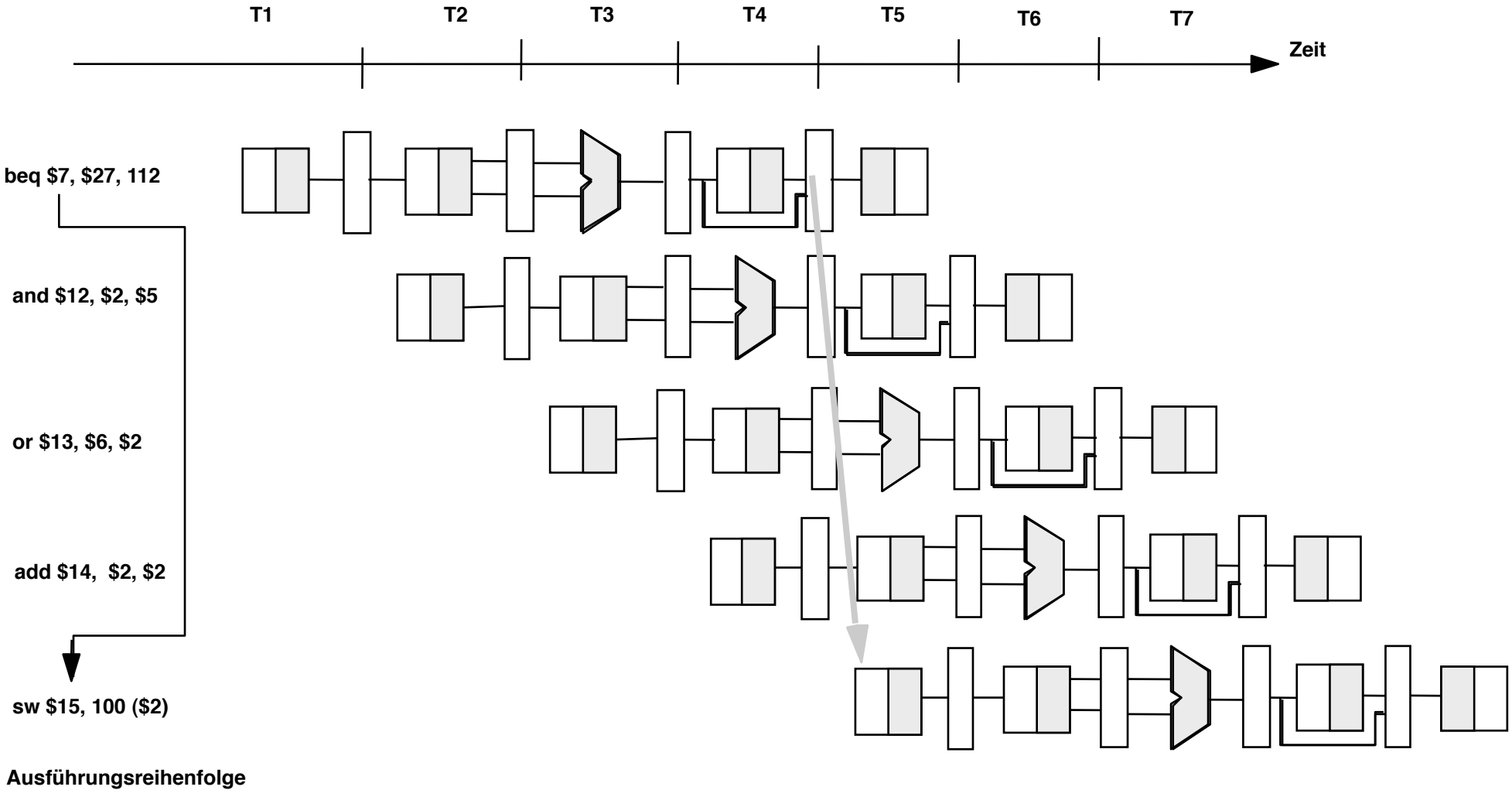


Kontroll Abhängigkeiten
Branch Hazards

Kontrollabhängigkeiten (Control Hazards):

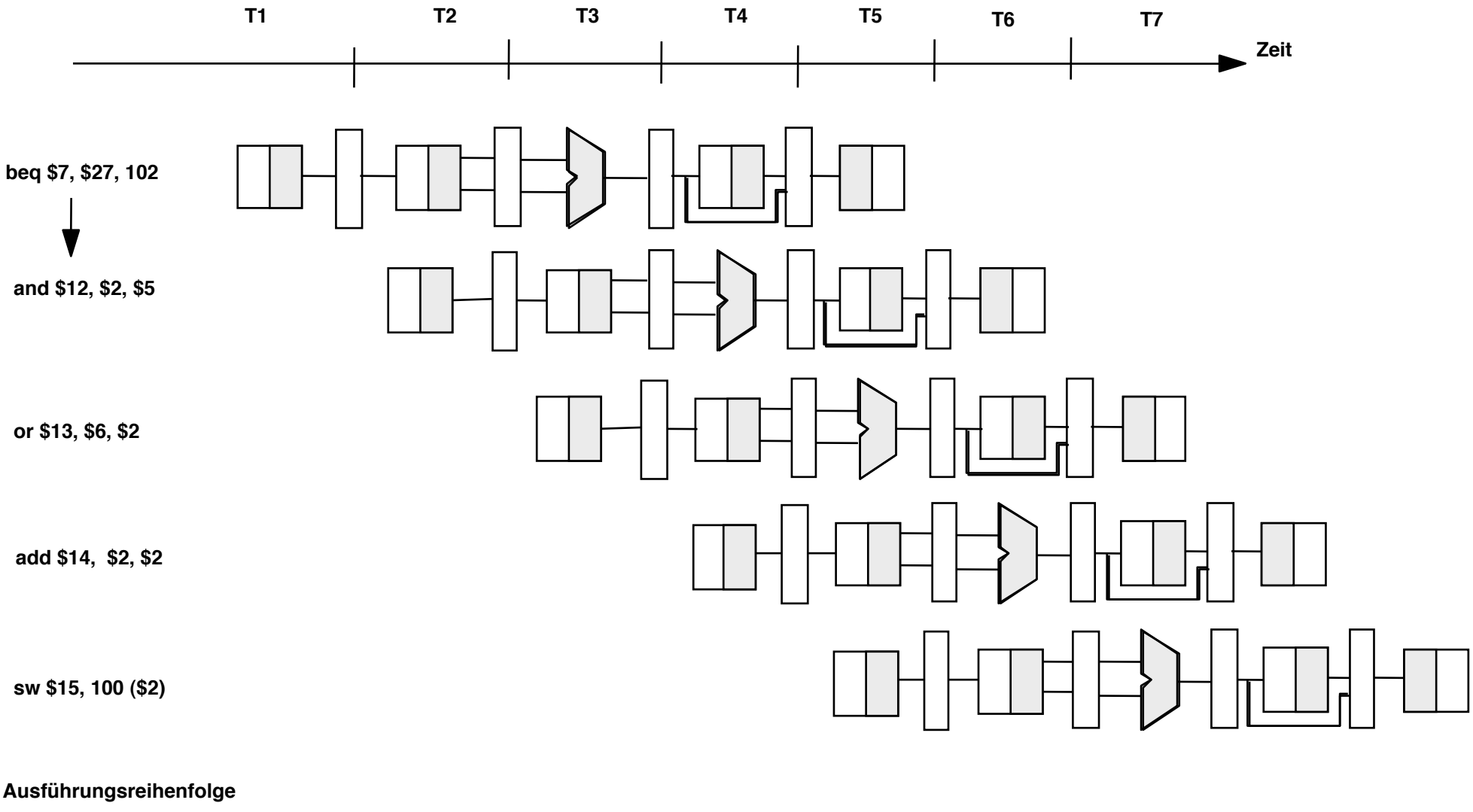


Pipeline - Hindernisse: Kontrollabhängigkeiten



Pipeline - Hindernisse: Kontrollabhängigkeiten

Der Sprung wird nicht ausgeführt



Pipeline - Hindernisse: Kontrollabhängigkeiten

Auflösung von Kontrollabhängigkeiten:

- **Verzögern**

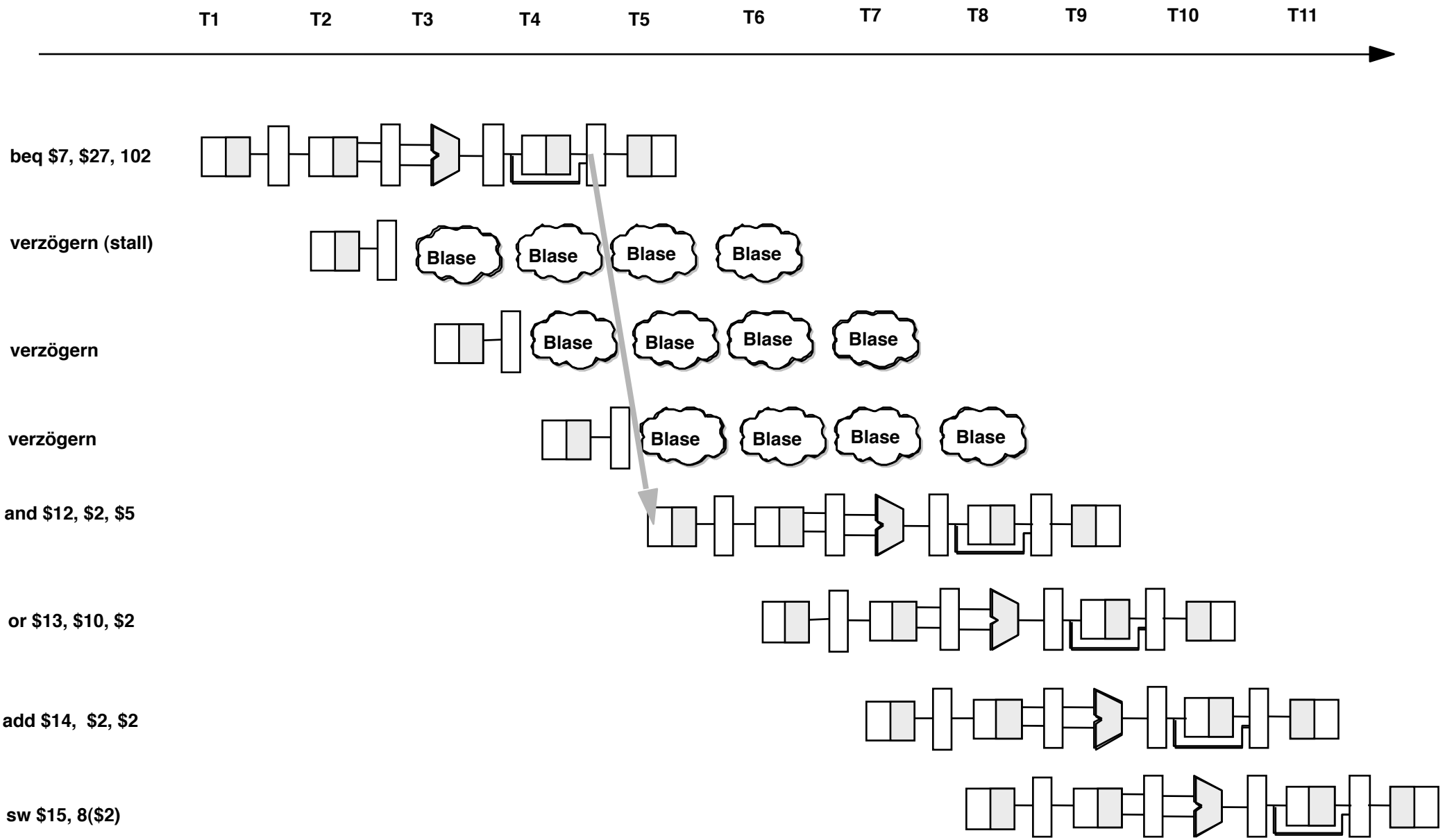
  Zur automatischen Erkennung von Sprüngen muß lediglich der OPCODE ausgewertet werden. Andere Abhängigkeiten zu Instruktionen in der Pipeline bestehen NICHT !

- **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

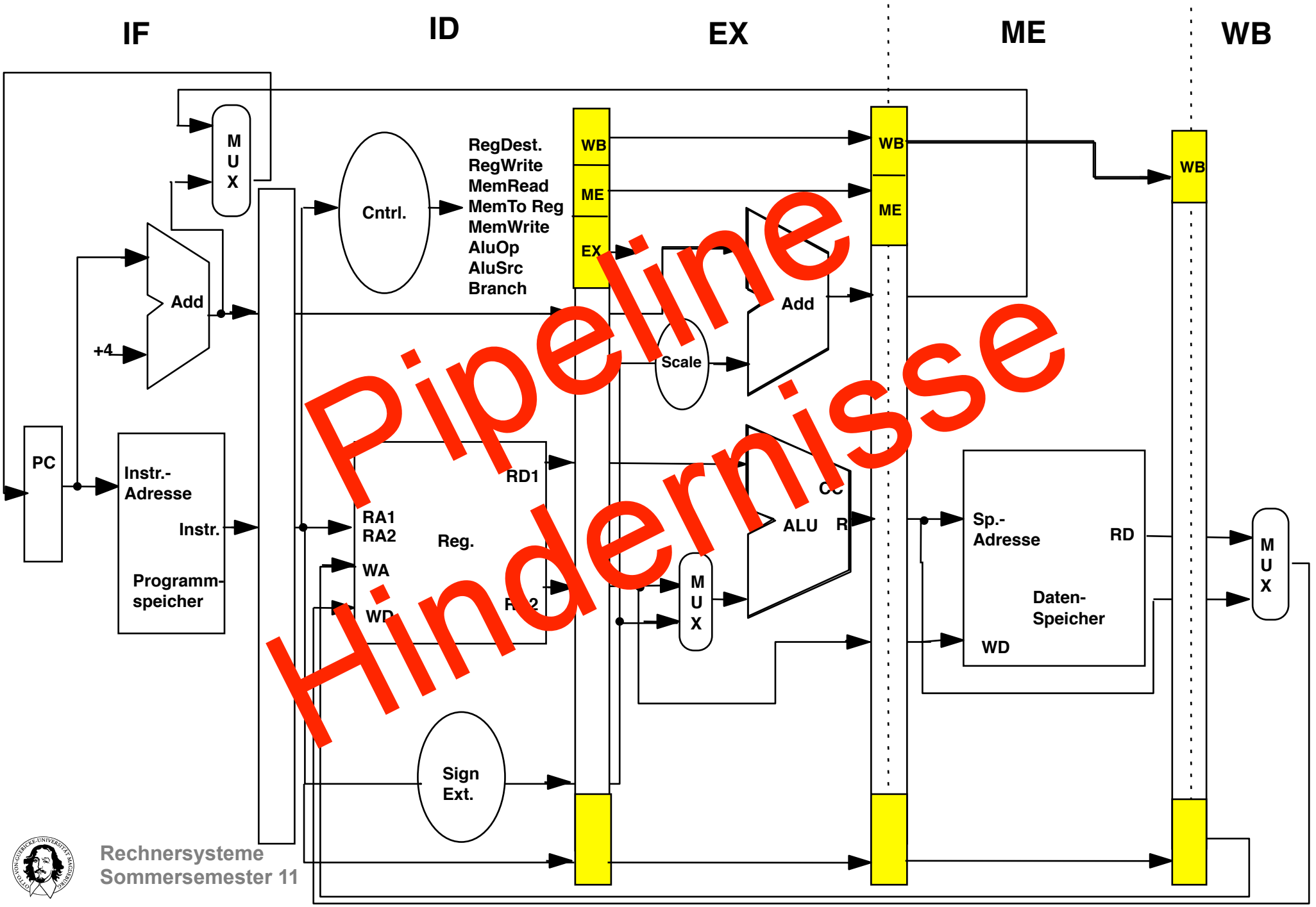
→ **Unterbrechung der aktuellen Instruktion**

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

→ **die in der Pipeline befindlichen Instruktionen müssen gelöscht werden**

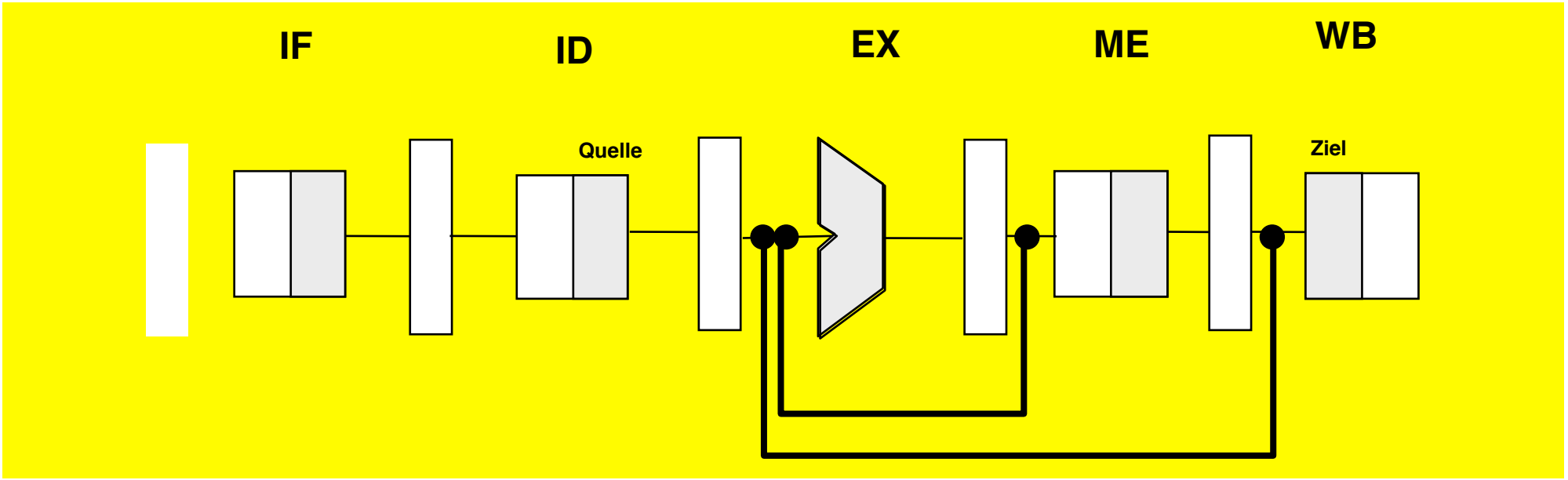
Zusammenfassung: Pipelining

- 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

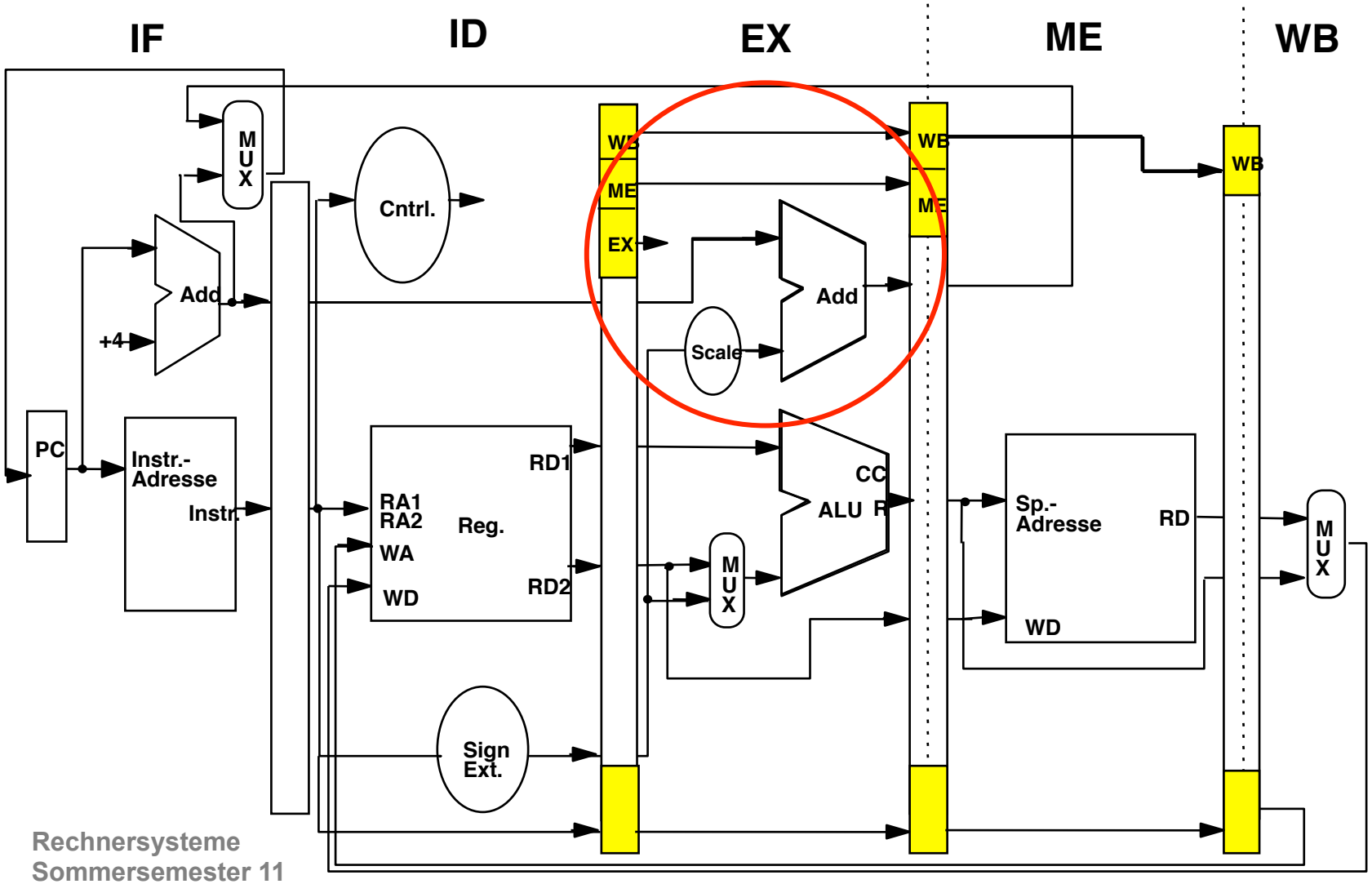

IF
ID
EX
ME
WB
M U X
Cntrl.
RegDest.
RegWrite
MemRead
MemTo Reg
MemWrite
AluOp
AluSrc
Branch
WB
ME
EX
Add
+4
Scale
PC
Instr.-
Adresse
Instr.
Programm-
speicher
RD1
RA1
RA2
Reg.
WA
WD
ALU
CC
R
Sp.-
Adresse
RD
Daten-
Speicher
WD
Sign
Ext.
Pipeline
Hindernisse

Datenabhängigkeiten (Data Hazards):

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



Kontrollabhängigkeiten (Control Hazards):



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 (Brach 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 kann 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:

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

Instruction-Level-Parallelism (ILP):

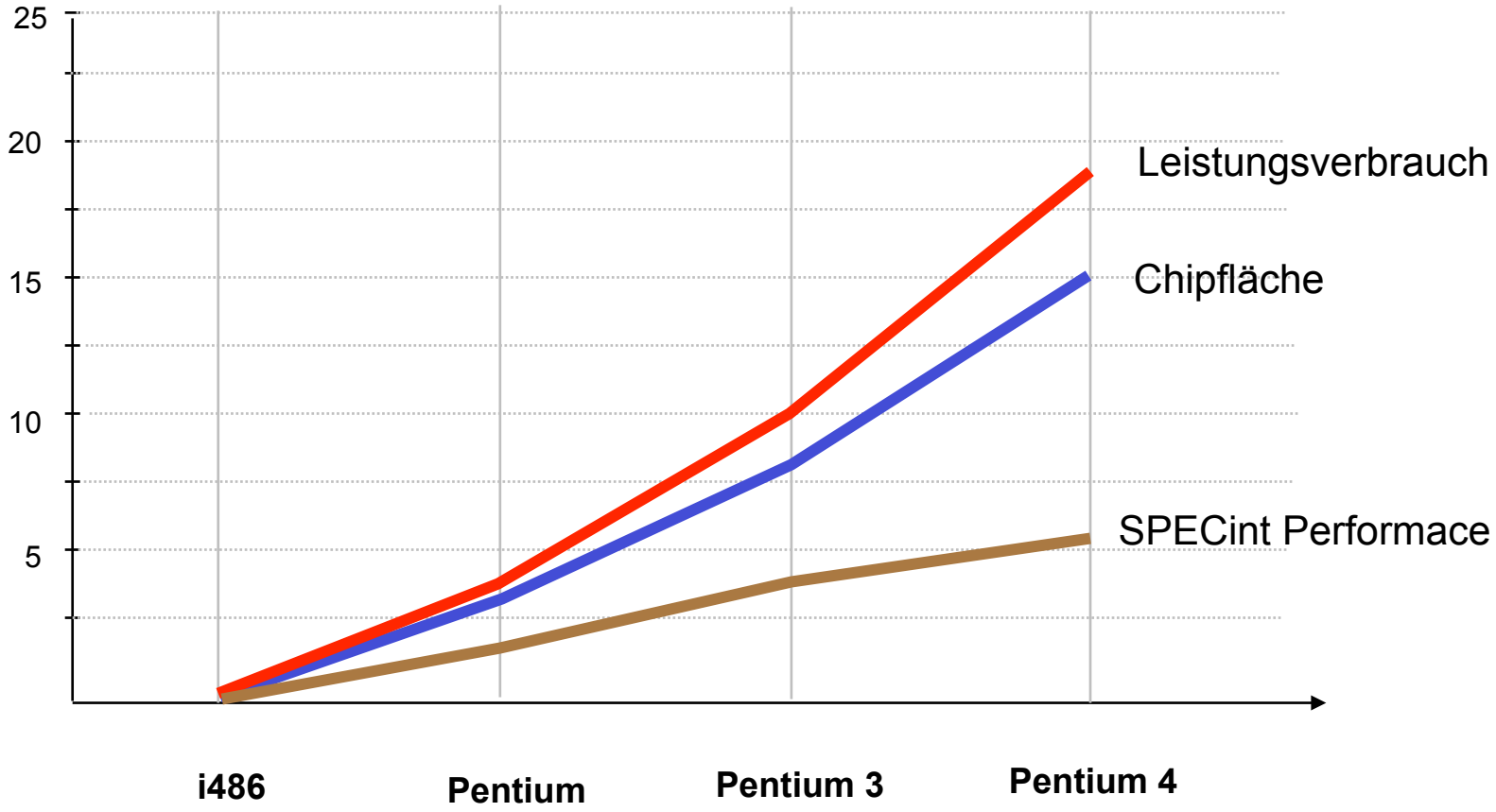
+ In mehreren Pipelines können Befehle parallel ausgeführt werden
- Es müssen genügend unabhängige Befehle gefunden werden
  → Out-of –order-execution wertet ein großes „Befehls-Fenster“ aus . Die Reihenfolge der Befehle richtet sich nach ihren Abhängigkeiten, nicht nach ihrer Reihenfolge im Programm.
- 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.

Vorteile und Probleme

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



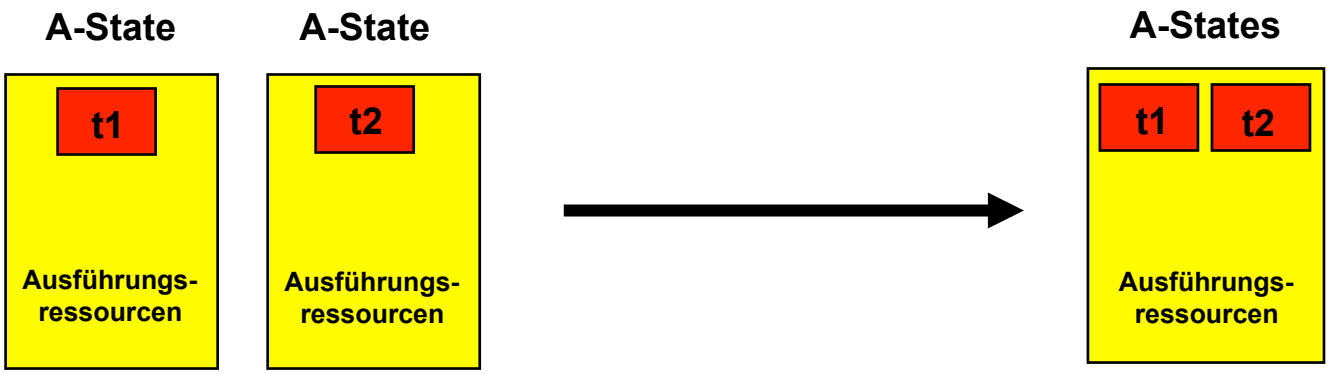
Thread-Level Parallelismus

Ausführliche Information unter: http://www.intel.com/technology/itj/2002/volume06issue01/vol6iss1_hyper_threading_technology.pdf

Annahme: Zwei unabhängige Threads haben keine abhängigen Instruktionen.

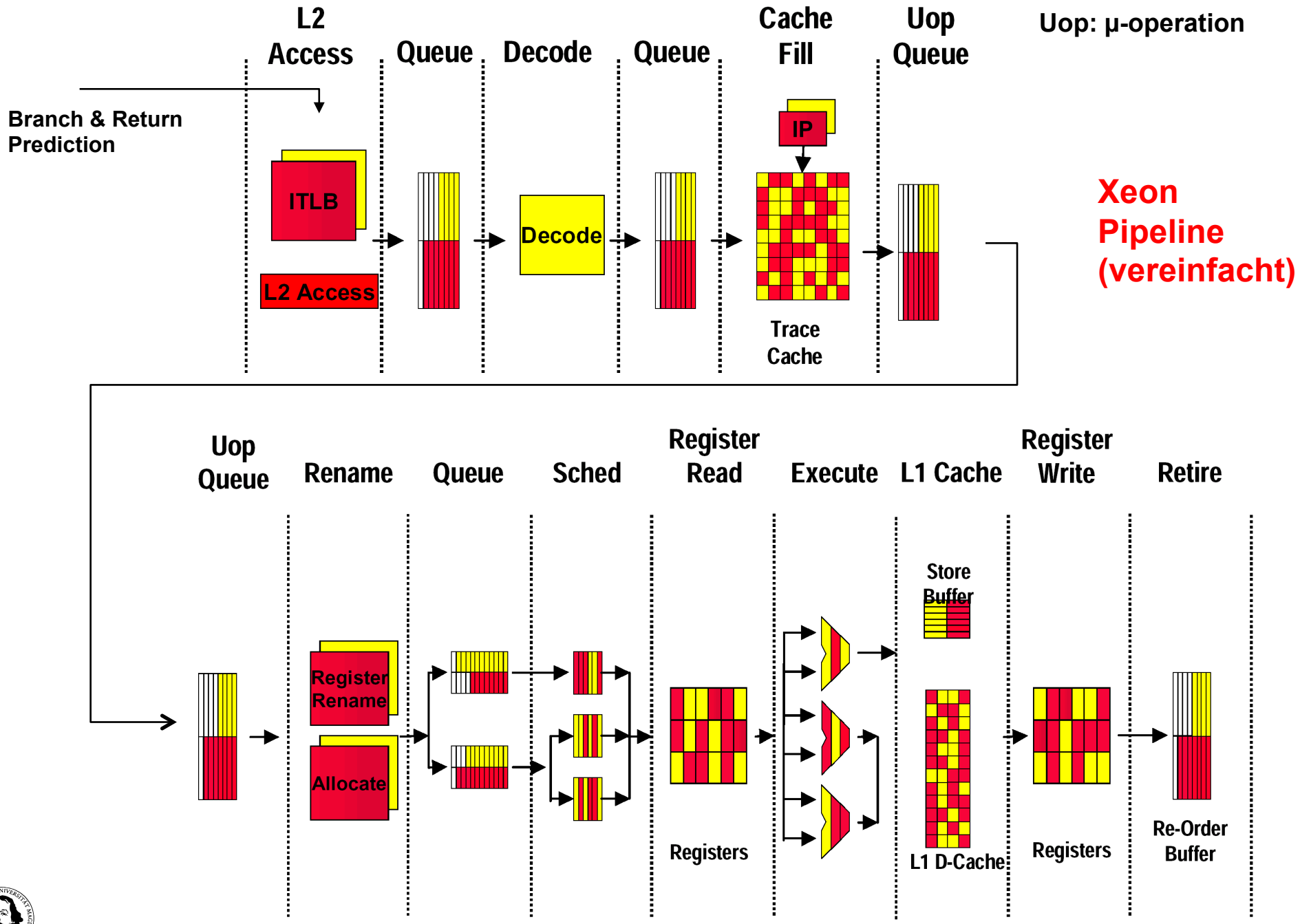
Beobachtung: Die Speicherung des Zustands eines Thread braucht weniger Transistoren/Chipfläche als die zugehörigen Ausführungseinheiten.

Folgerung: Multiprocessing auf EINEM Chip



Architecture State umfasst:
- **General purpose Register**
- **Interrupt Contr. Register**
- **Machine State Register**

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



Ausführliche Information unter: http://www.intel.com/technology/itj/2002/volume06issue01/vol6iss1_hyper_threading_technology.pdf

Lernziele

Verständnis ....

des Prinzips des Pipelining als Form der Leistungssteigerung durch Parallelverarbeitung

warum Pipelining "einfache" Instruktionen voraussetzt

der Pipelinehindernisse durch
- Datenabhängigkeiten
- Kontrollabhängigkeiten

wie Konsistenz der Berechnungen durch Erkennen und Umgehen der Hindenisse erhalten wird.