Rechnerorganisation – Kapitel 7

Prof. Sarah Harris
Fachgebiet Eingebettete Systeme und ihre Anwendungen (ESA)
Fachbereich Informatik

TECHNISCHE UNIVERSITÄT DARMSTADT

SS 16

Kapitel 7: Themen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Einführung in die Mikroarchitektur
- Analyse der Rechenleistung
- Ein-Takt-Prozessor
- Mehrtakt-Prozessor
- Pipeline-Prozessor
- Ausnahmebehandlung
- Weiterführende Themen

Embedded Systems & Applications

Einleitung

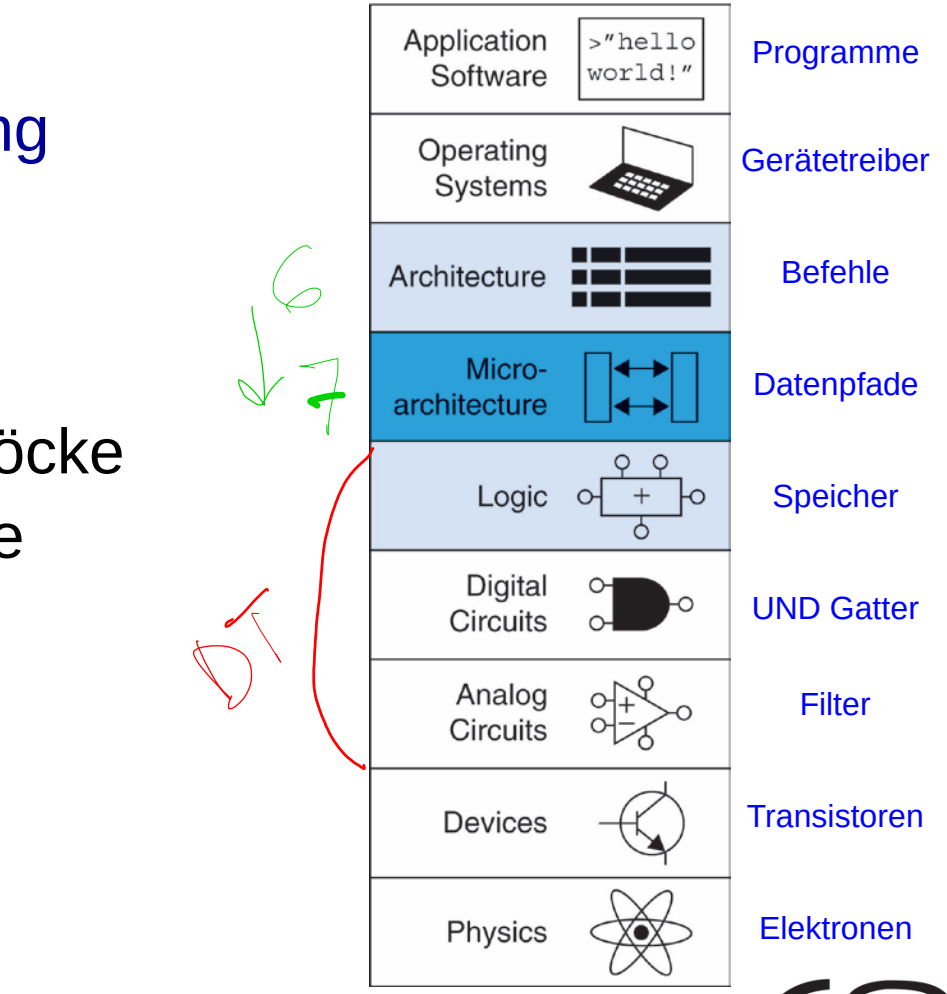
TECHNISCHE UNIVERSITÄT DARMSTADT

- Mikroarchitektur

  Hardware-Implementierung einer Architektur

- Prozessor:

1. Datenpfad: funktionale Blöcke
2. Steuerwerk: Steuersignale



Embedded Systems & Applications

Mikroarchitektur

TECHNISCHE UNIVERSITÄT DARMSTADT

Mehrere Implementierungen für eine Architektur

- **Ein-Takt**

  Jede Instruktion wird in einem Takt ausgeführt

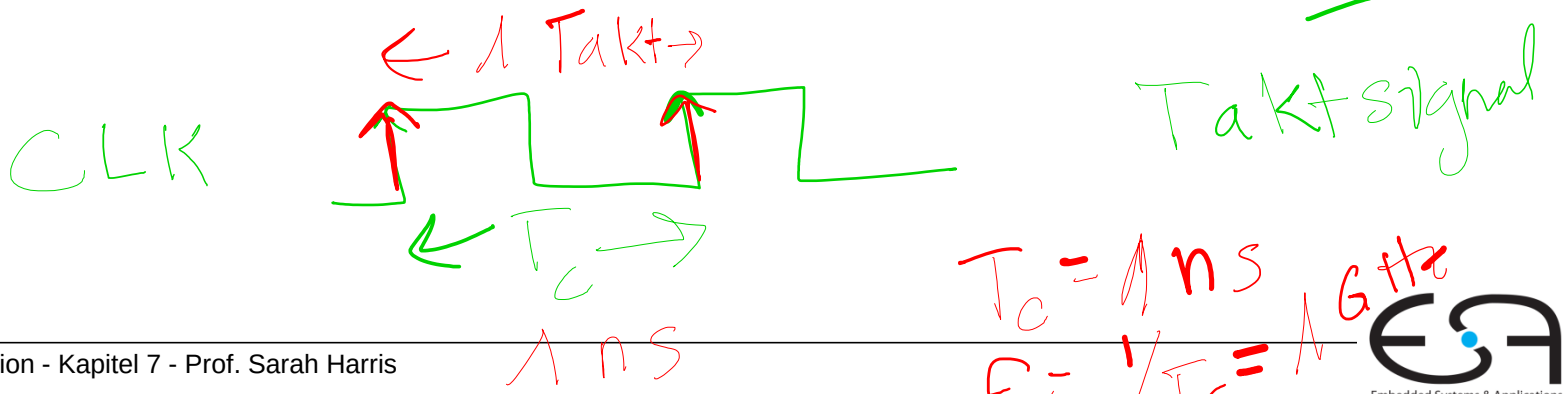
- **Mehrtakt**

  Jede Instruktion wird in Teilschritte zerlegt

- **Pipelined**

  Jede Instruktion wird in Teilschritte zerlegt

  Mehrere Instruktionen werden gleichzeitig ausgeführt



Embedded Systems & Applications

Rechenleistung eines Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Ausführungszeit eines Programms

Ausführungszeit = (#Instruktionen)(Takte/Instruktion)(Sekunden/Takt)

Definitionen:

- Takte/Instruktion = **CPI** (*cycles per instruction*)
- Sekunden/Takt = **Taktperiode**
- 1/CPI = Instruktionen/Takt = **IPC** (*instructions per cycle*)

Herausforderung: Einhalten zusätzlicher Anforderungen

- Kosten
- Energiebedarf
- Rechenleistung

Embedded Systems & Applications

Unser erster MIPS Prozessor

TECHNISCHE UNIVERSITÄT DARMSTADT

Zunächst Untermenge des MIPS Befehlssatzes:

- R-Typ Befehle: `and`, `or`, `add`, `sub`, `slt`
- Speicherbefehle: `lw`, `sw`
- Bedingte Verzweigungen: `beq`

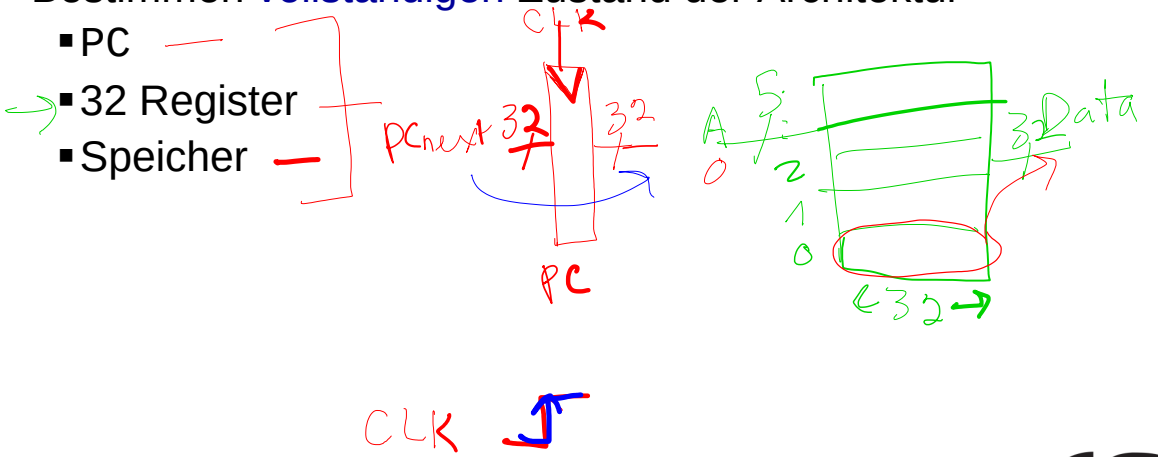
Später hinzunehmen: `addi` und `j`

Architekturzustand

TECHNISCHE UNIVERSITÄT DARMSTADT

Auf Ebene der Architektur sichtbare Daten
- Für den Programmierer zugänglich

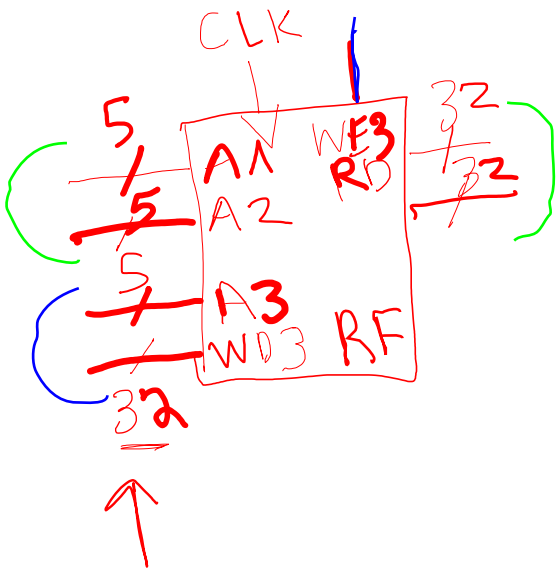
Bestimmen vollständigen Zustand der Architektur
- PC
- 32 Register
- Speicher



CLK

Embedded Systems & Applications

Read data
Lese data

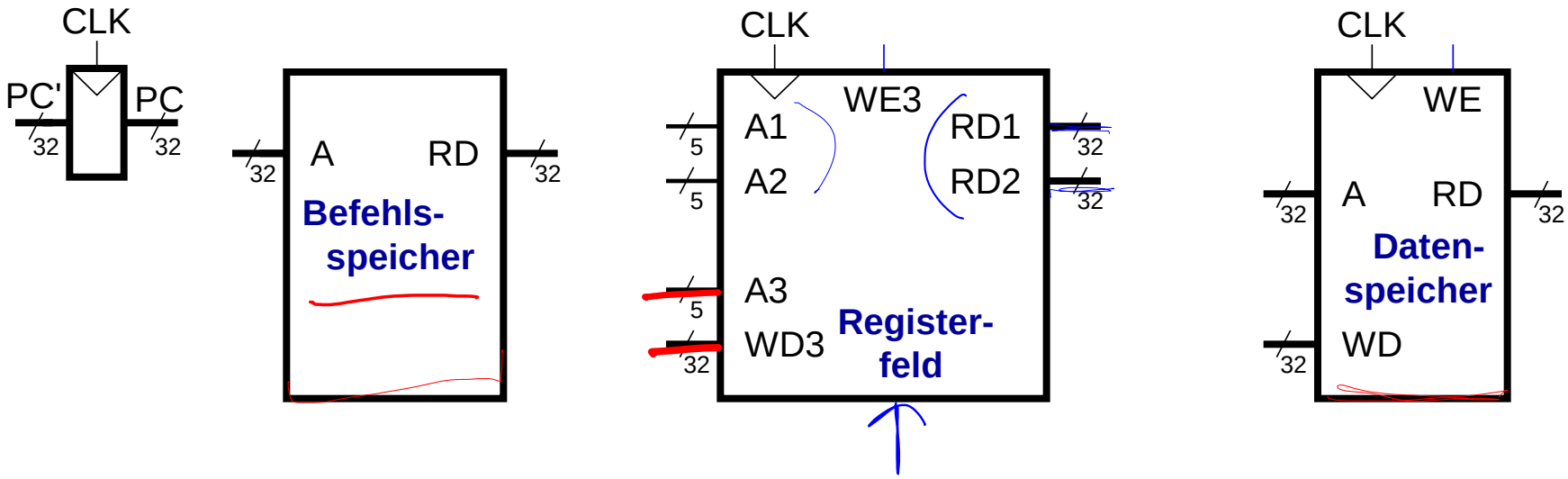


CLK

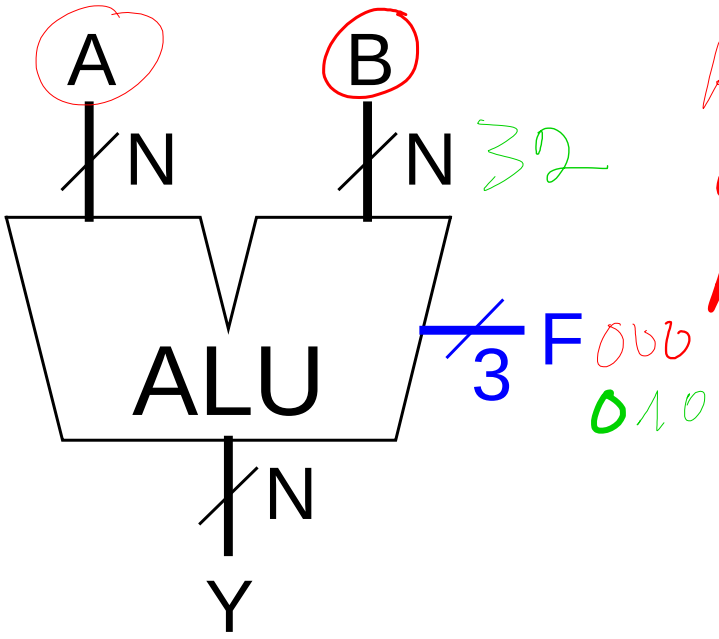
Enable = Freigabe-Signal

WE

Elemente des MIPS Architekturzustands



Zur Erinnerung: ALU



$F_{2:0}$	Funktion
000	A & B
001	A | B
010	A + B
011	unbenutzt
100	A & ~B
101	A | ~B
110	A - B
111	SLT

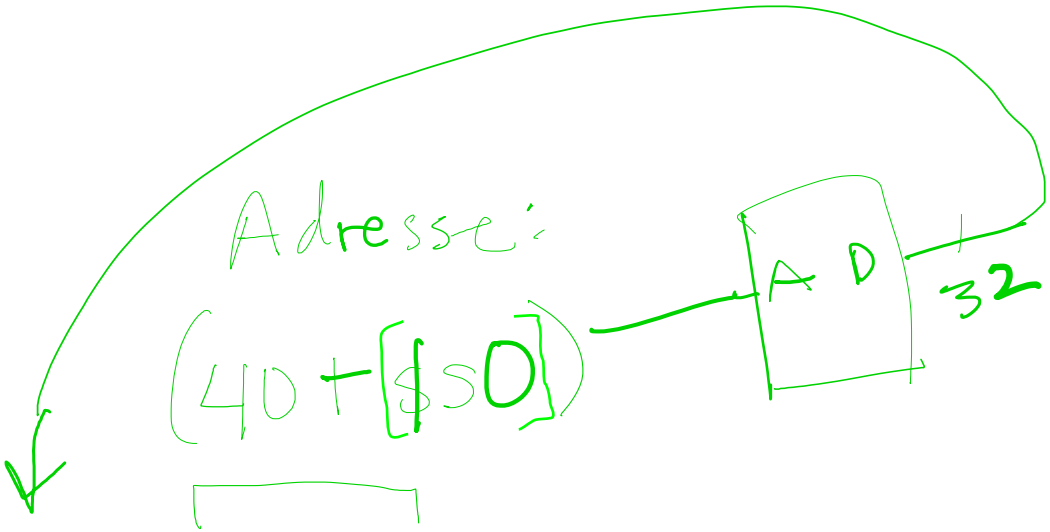
Ein-Takt MIPS Prozessor

TECHNISCHE UNIVERSITÄT DARMSTADT

- Datenpfad
- Steuerwerk

Embedded Systems & Applications

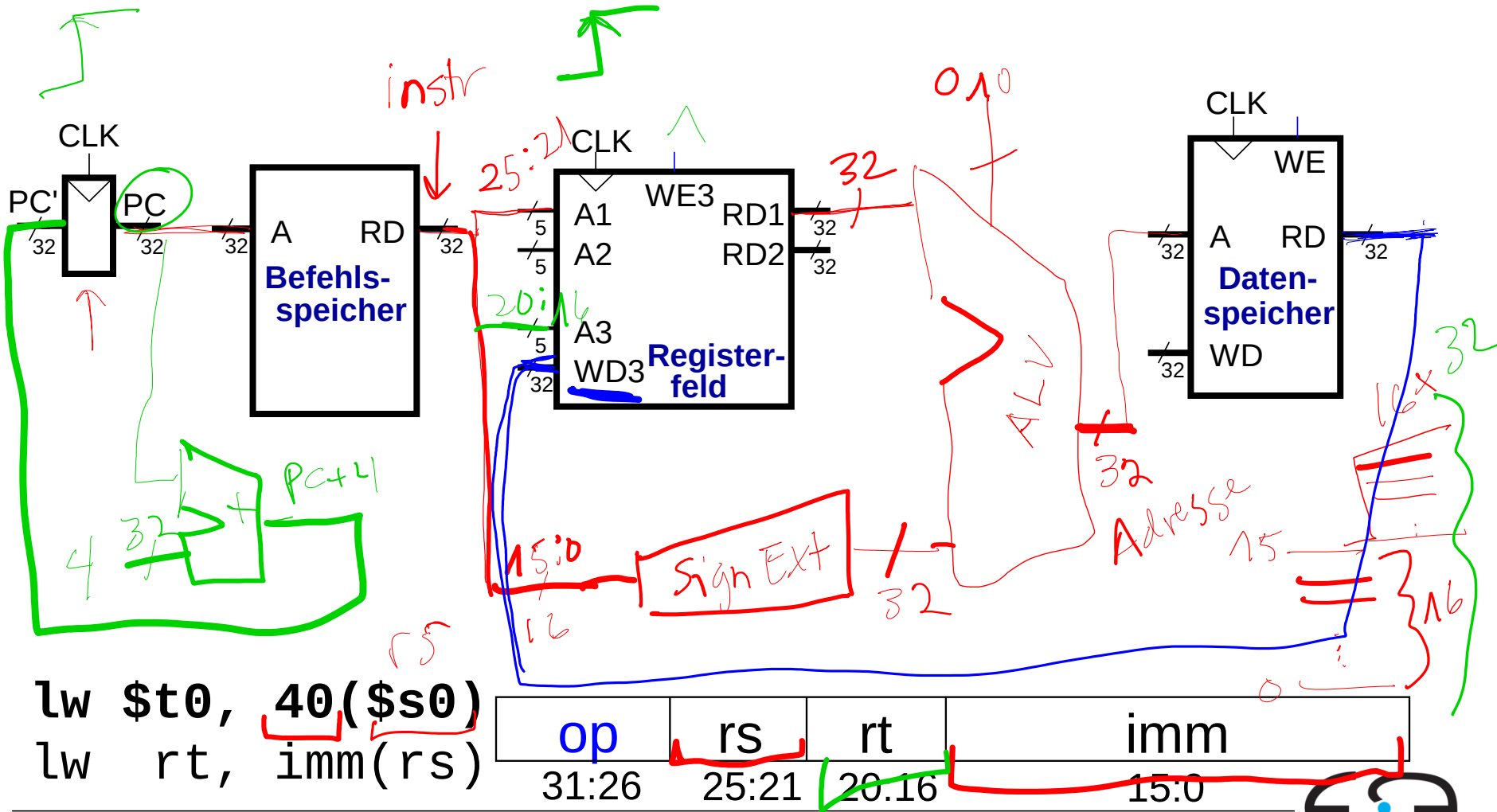
Datenpfad: lw




```
lw $t0, 40($s0)
lw  rt, imm(rs)
```

op	rs	rt	imm
31:26	25:21	20:16	15:0

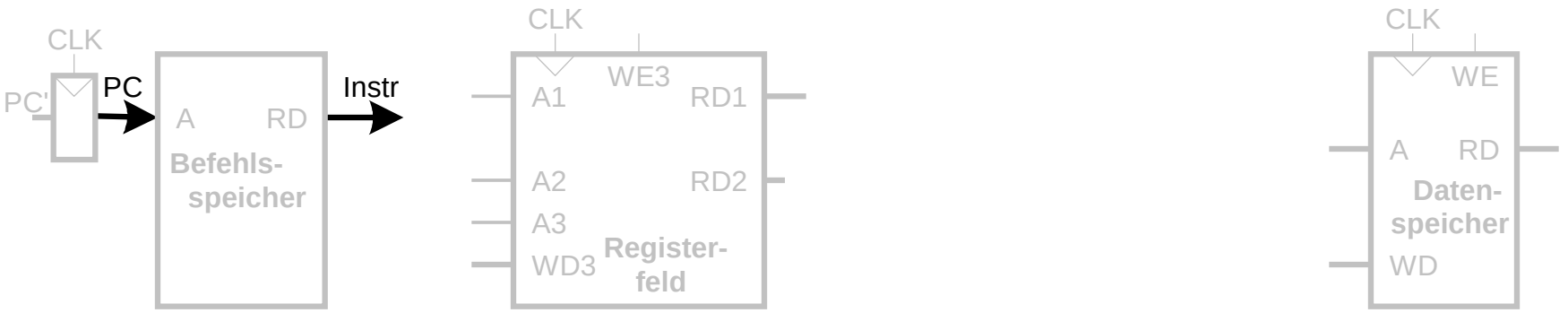
Datenpfad: lw



Ein-Takt Datenpfad: Holen eines `lw` Befehls

Ein *load word* Befehl (`lw`) soll ausgeführt werden

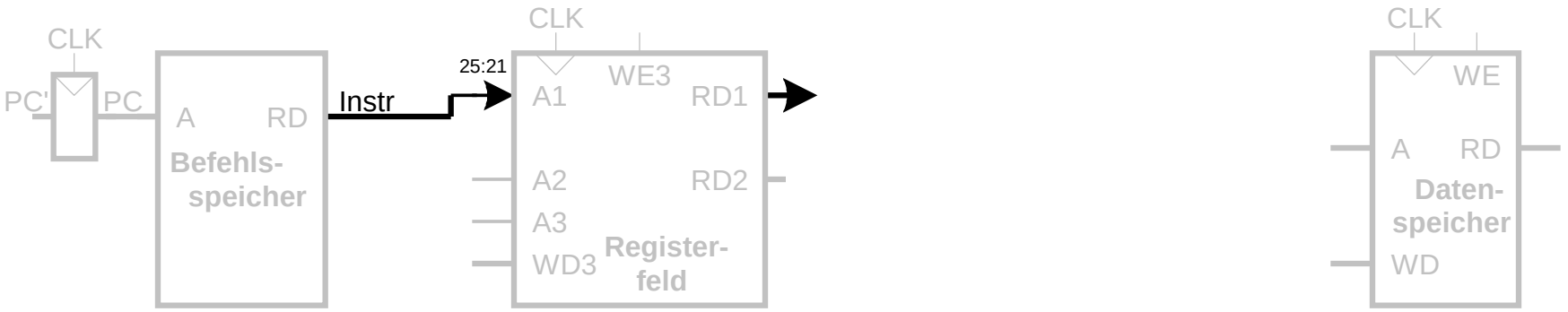
Schritt 1: Hole Instruktion



Ein-Takt Datenpfad: Lesen des Registers für `lw`

TECHNISCHE UNIVERSITÄT DARMSTADT

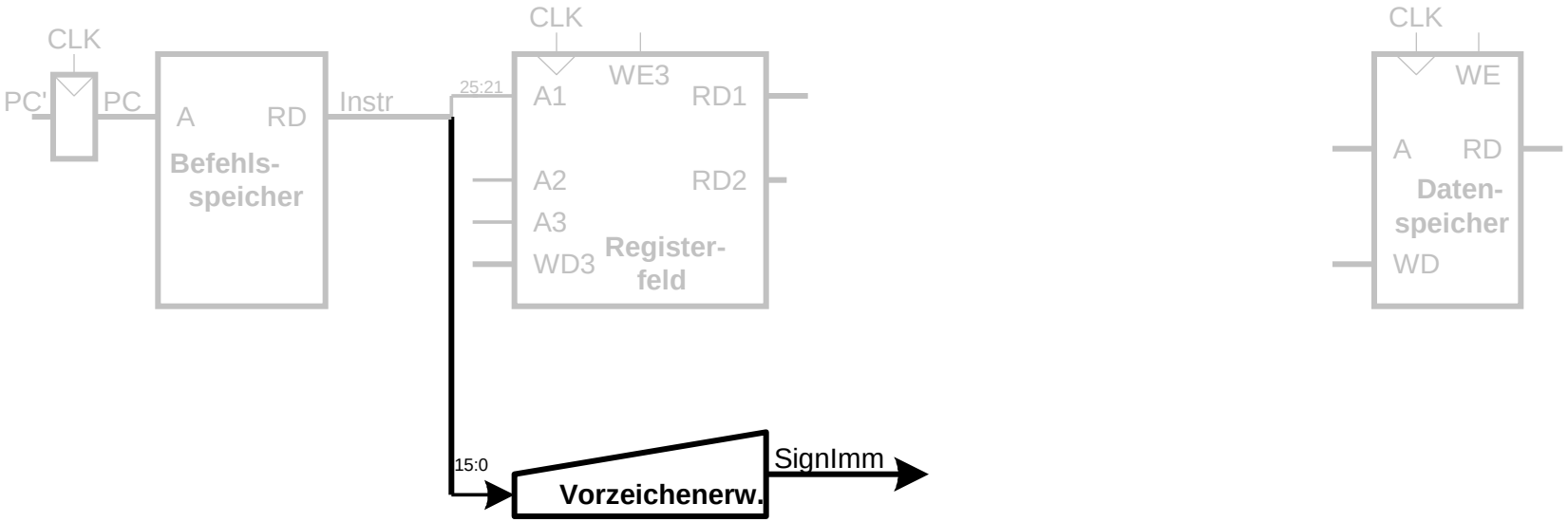
Schritt 2: Lese Quelloperand aus Registerfeld



Embedded Systems & Applications

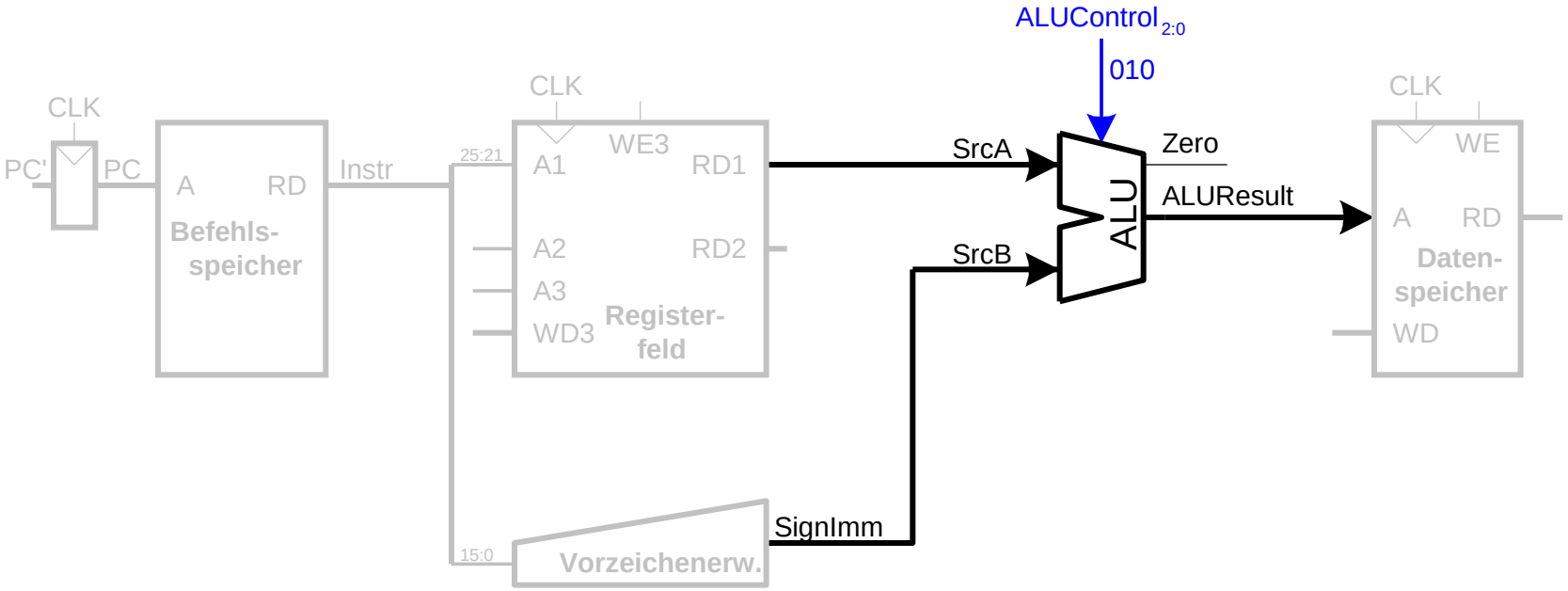
Ein-Takt Datenpfad: Behandle `lw` Direktwert

Schritt 3: Vorzeichenerweitere den 16b Direktwert auf 32b Signal `SignImm`



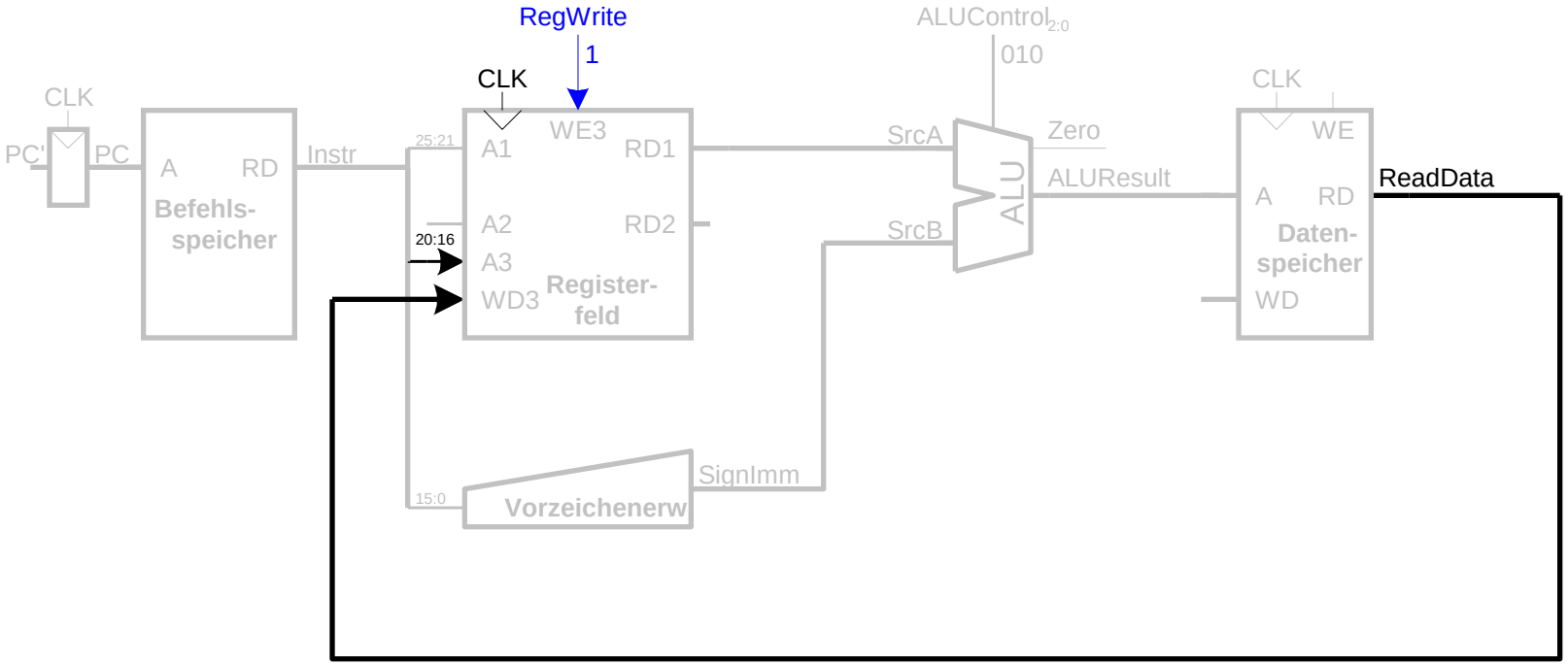
Ein-Takt Datenpfad: Berechne `lw` Zieladdresse

Schritt 4: Berechne die effektive Speicheradresse



Ein-Takt Datenpfad: Lese Speicher mit `lw`

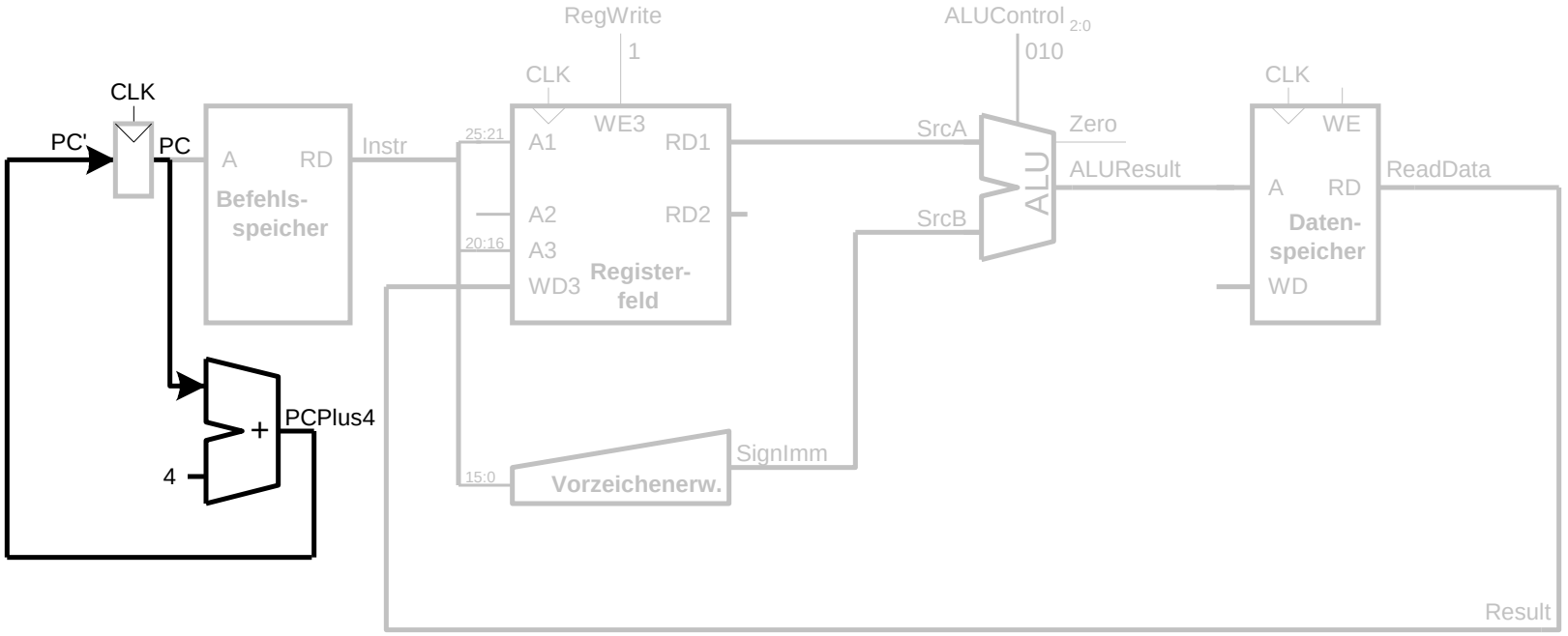
Schritt 5: Lese Daten aus Speicher und schreibe sie ins passende Register



Ein-Takt Datenpfad : Erhöhe PC nach `lw`

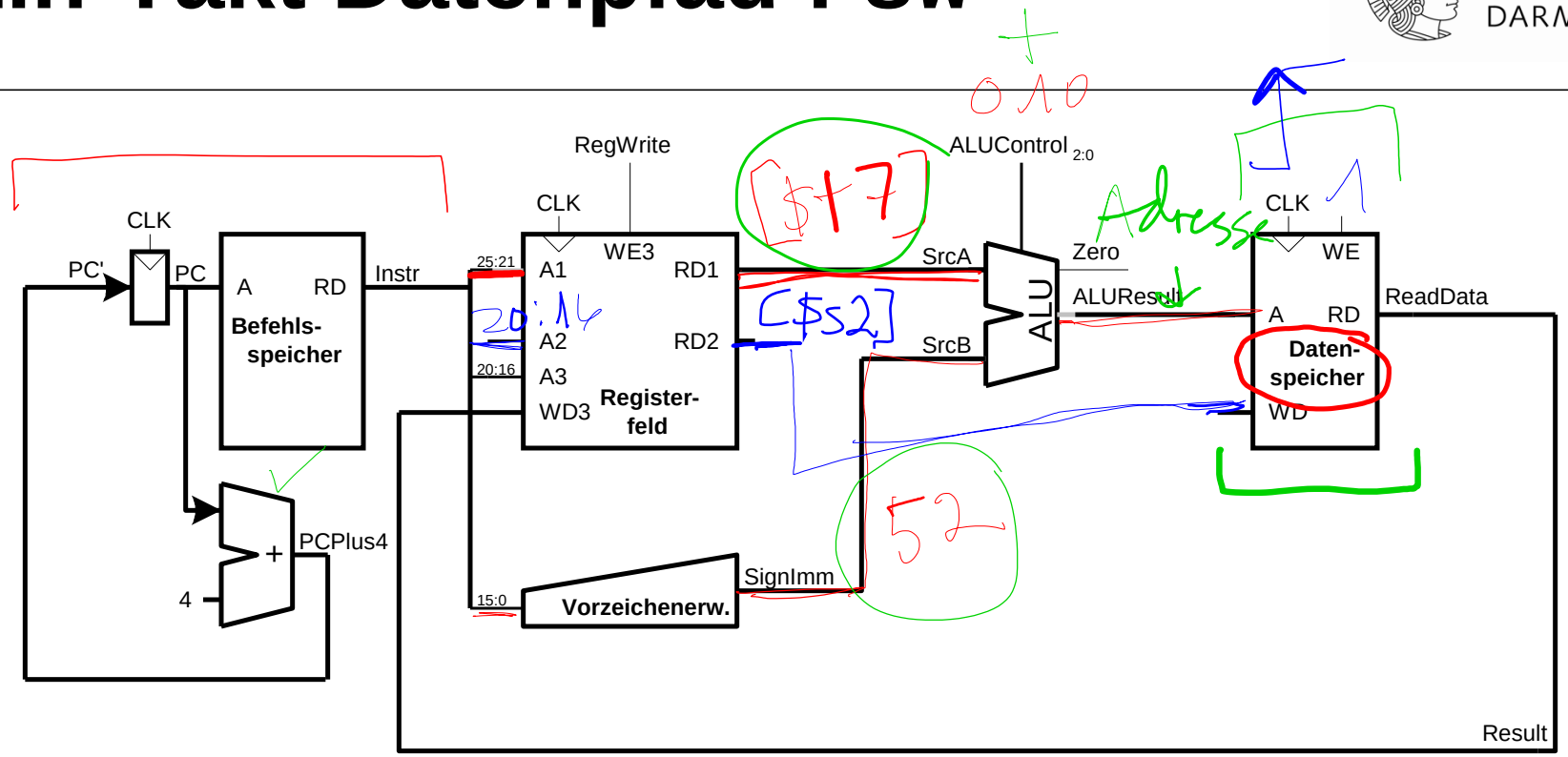
TECHNISCHE UNIVERSITÄT DARMSTADT

Schritt 6: Bestimme Adresse des nächsten Befehls



Embedded Systems & Applications

Ein-Takt Datenpfad : sw



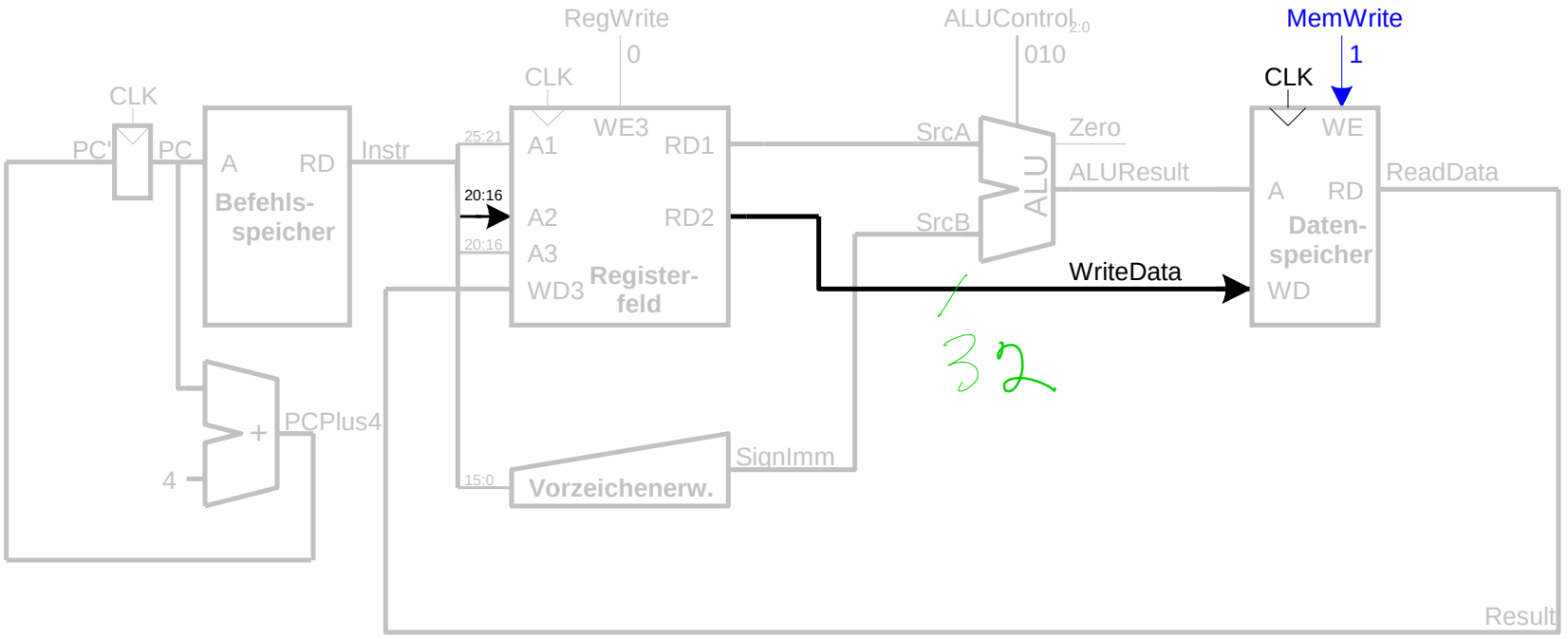

```
sw $s2, 52($t7)
sw  rt, imm(rs)
```

op	rs	rt	imm
31:26	25:21	20:16	15:0

Ein-Takt Datenpfad: sw

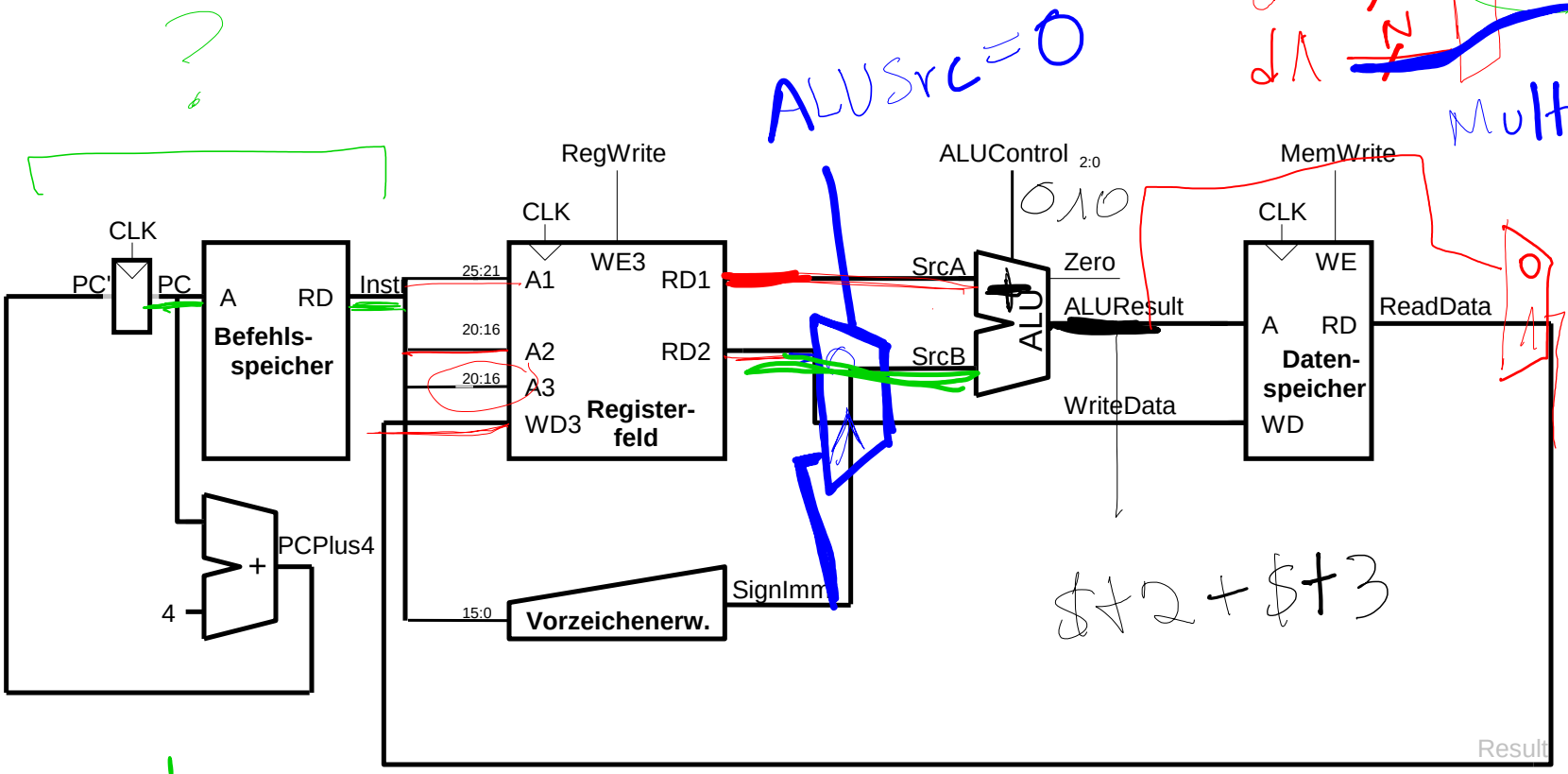
TECHNISCHE UNIVERSITÄT DARMSTADT

Schreibe Daten aus `rt` in den Speicher



Embedded Systems & Applications

Ein-Takt Datenpfad: R-Typ

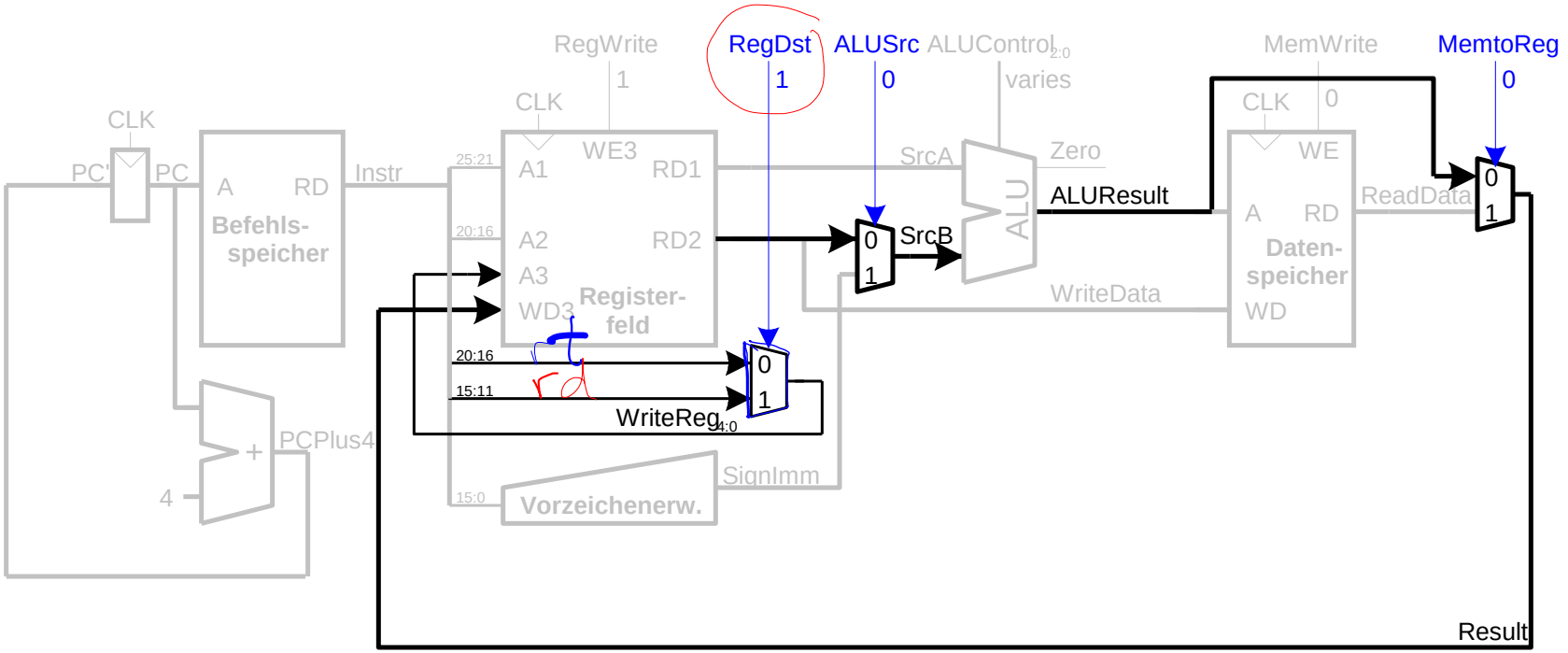



```
add $s1,$t2,$t3
add rd,rs,rt
```

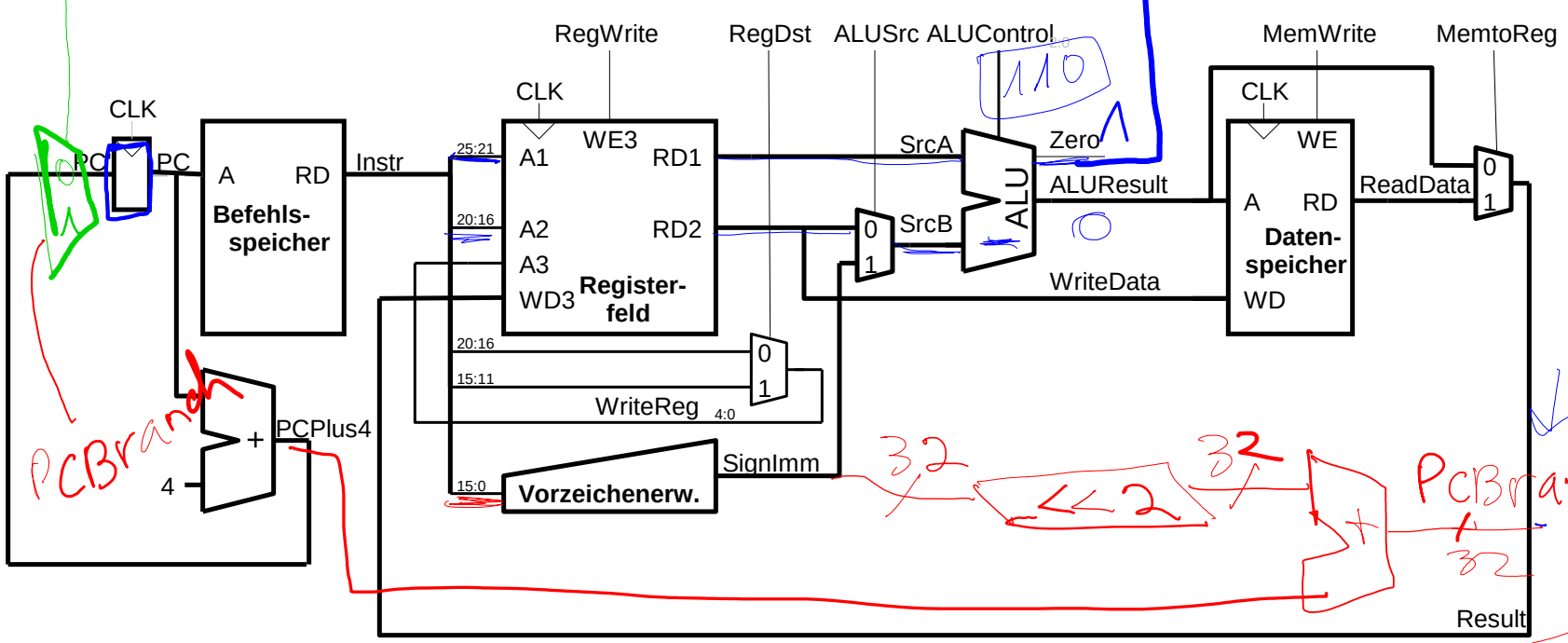
op	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

Ein-Takt Datenpfad: Instruktionen vom R-Typ

- Lese aus `rs` und `rt`
- Schreibe *ALUResult* ins Registerfeld
- Schreibe nach `rd` (statt nach `rt` wie bei `sw`)



Ein-Takt Datenpfad: beq



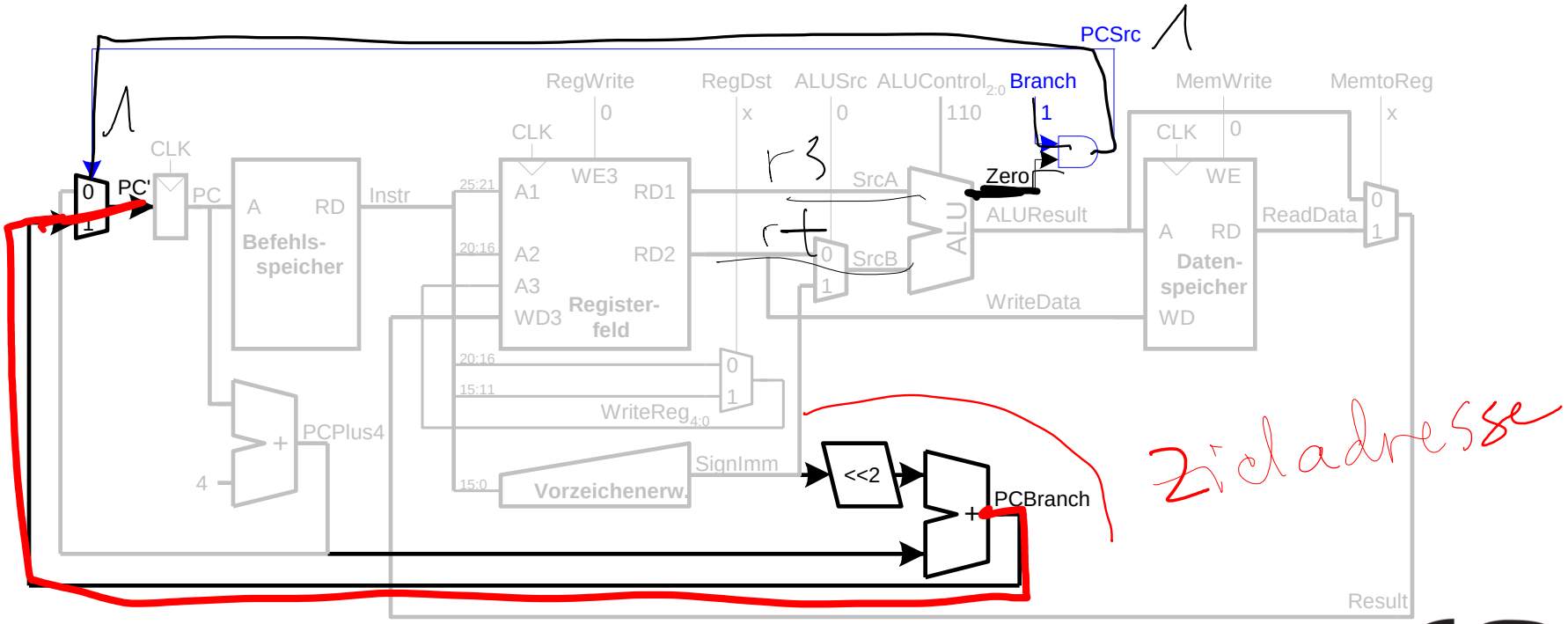

```
beq $s3,$s4,Loop
beq rs, rt, imm
```

op	rs	rt	imm
31:26	25:21	20:16	15:0

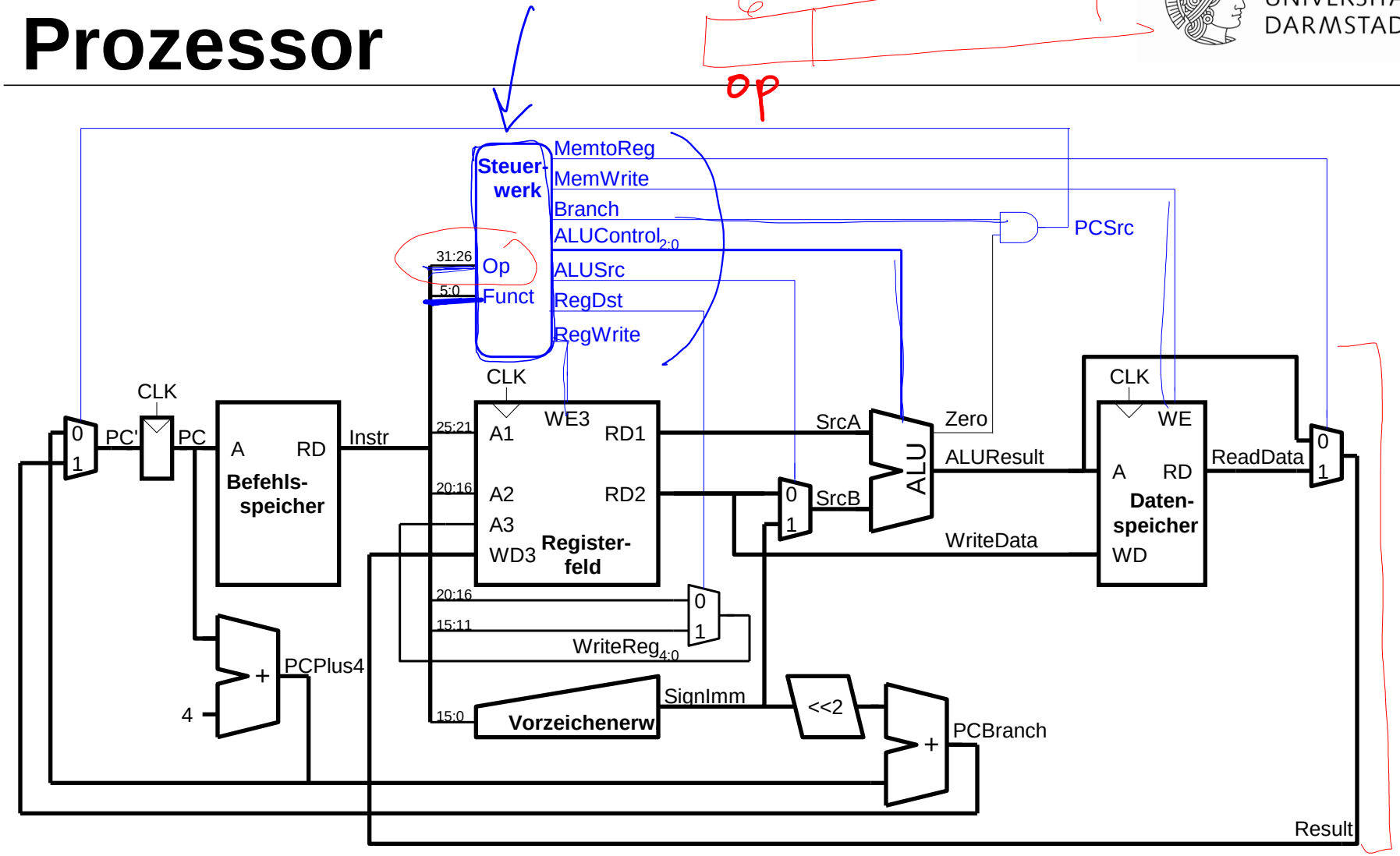
Ein-Takt Datenpfad: beq

- Prüfe ob Werte in `rs` und `rt` gleich sind
- Bestimme Adresse von Sprungziel *(branch target adress, BTA):*

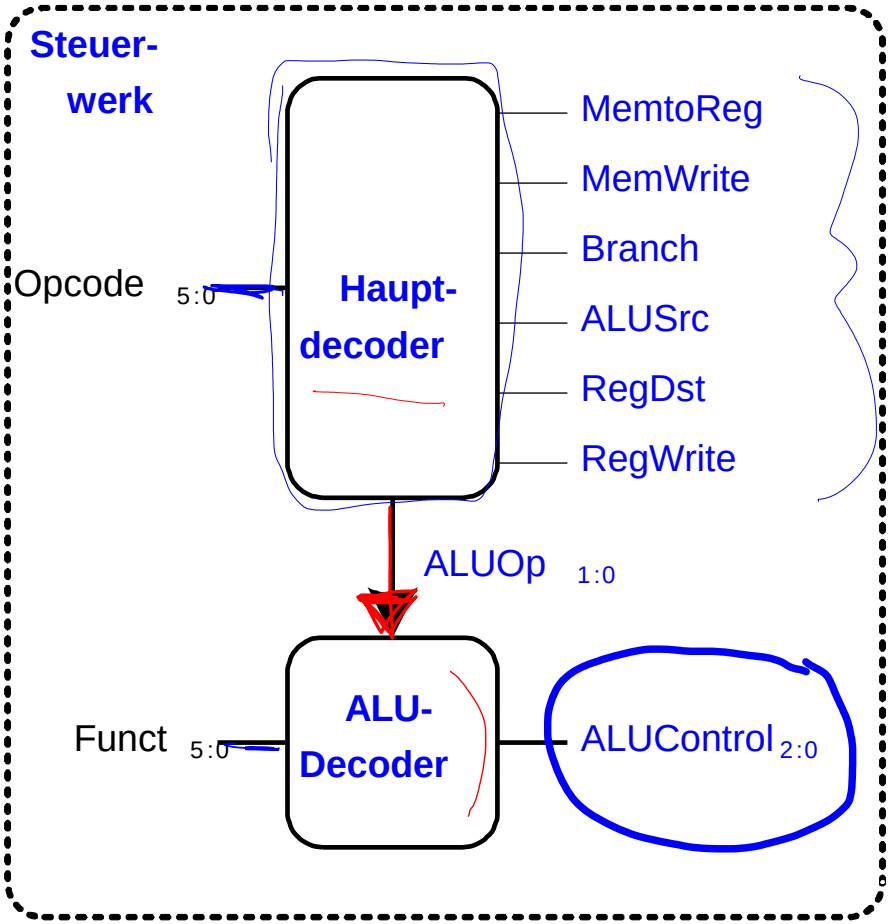
  BTA = (vorzeichenerweiterter Direktwert << 2) + (PC+4)

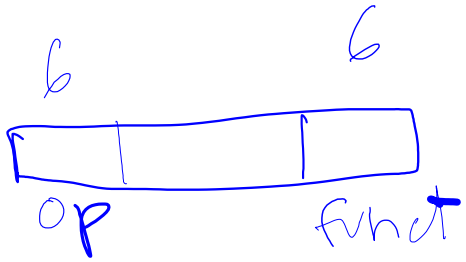


Vollständiger Ein-Takt-Prozessor

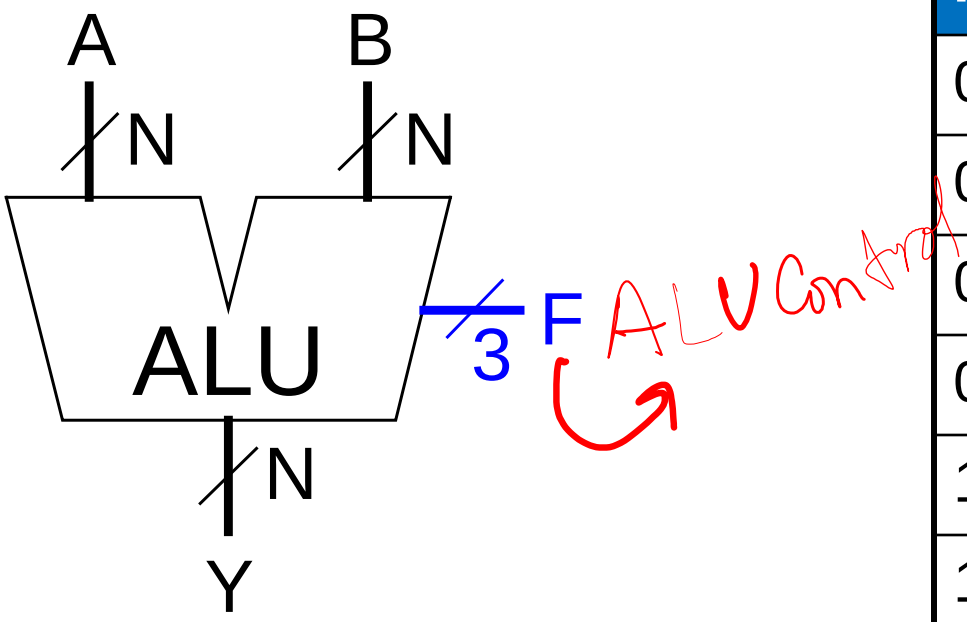


Steuerwerk


Steuer-
werk
Opcode
5:0
Haupt-
decoder
MemtoReg
MemWrite
Branch
ALUSrc
RegDst
RegWrite
ALUOp
1:0
Funct
5:0
ALU-
Decoder
ALUControl
2:0

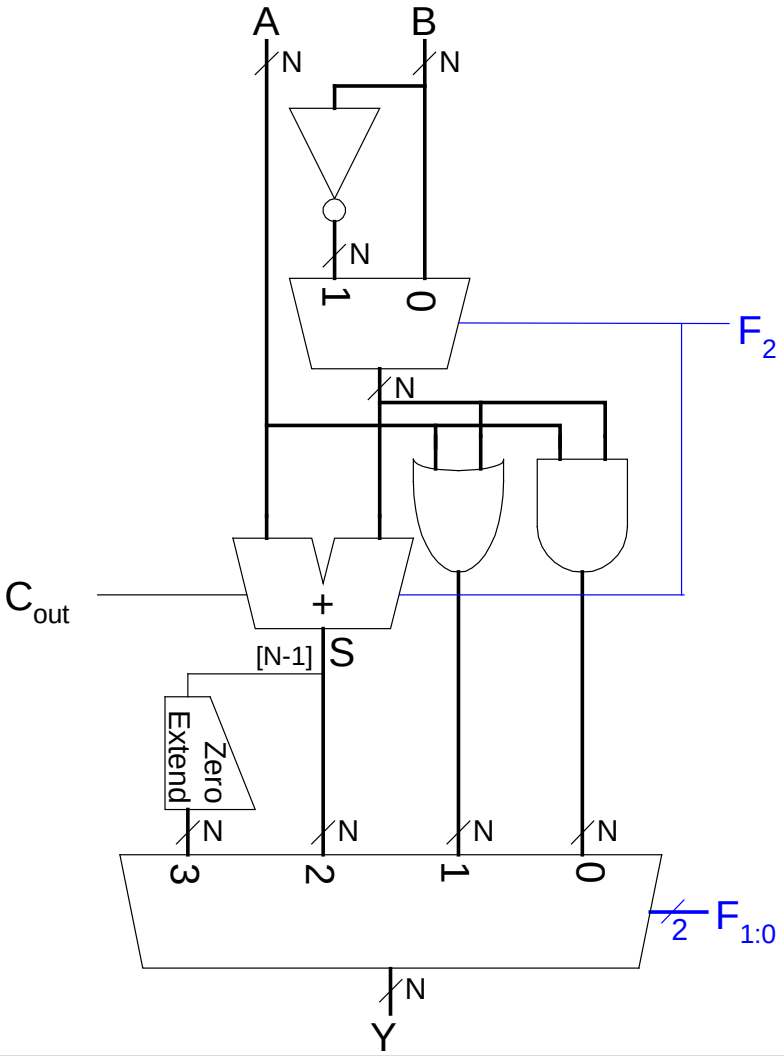

6
6
op
funct

Zur Erinnerung: ALU



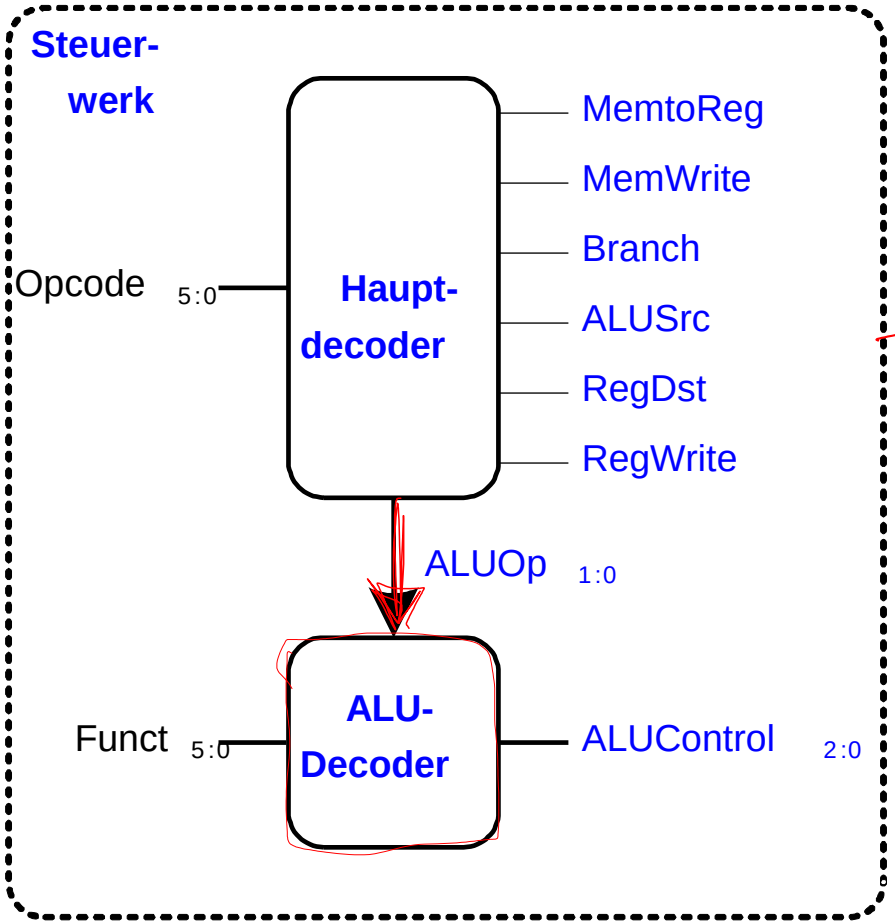
$F_{2:0}$	Funktion
000	A & B
001	A | B
010	A + B
011	unbenutzt
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Zur Erinnerung: ALU



$F_{2:0}$	Funktion
000	A & B
001	A | B
010	A + B
011	unbenutzt
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Steuerwerk: ALU Decoder



$ALUOp_{1:0}$	Bedeutung
00	Addiere
01	Subtrahiere
10	Werte Funct-Feld aus
11	unbenutzt

R-Type

Steuerwerk: ALU-Decoder

TECHNISCHE UNIVERSITÄT DARMSTADT

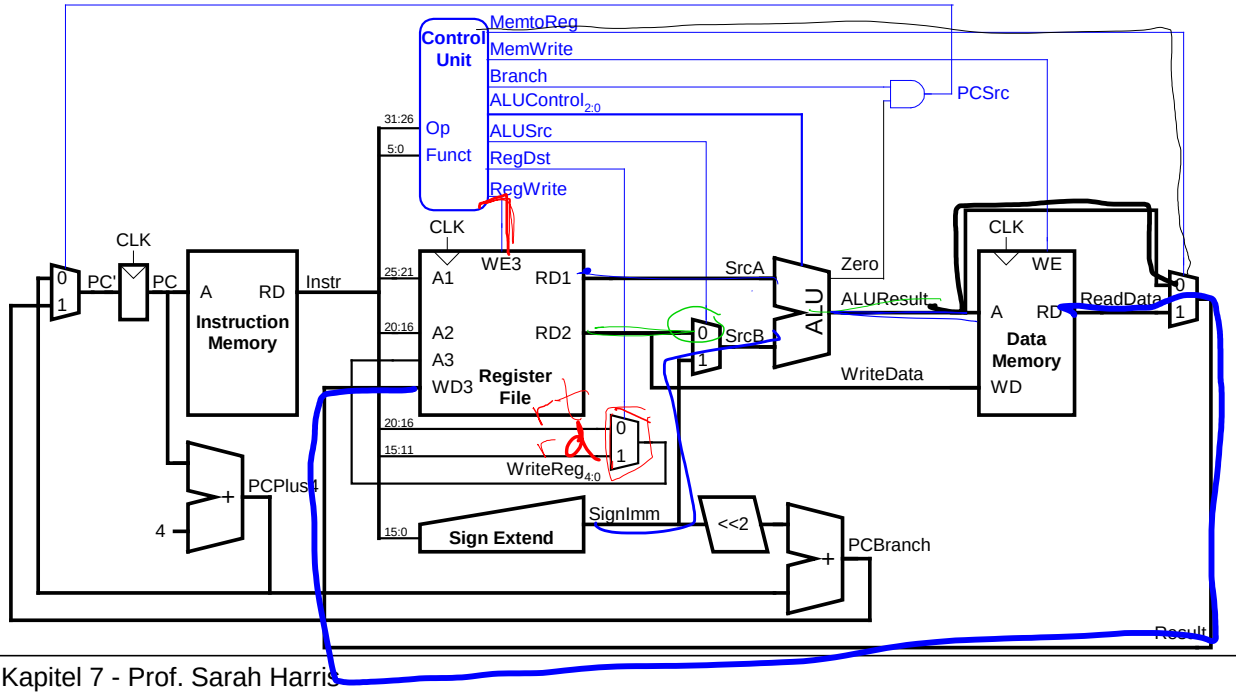
$ALUOp_{1:0}$	Bedeutung
00	Addiere
01	Subtrahiere
10	Werte Funct-Feld aus
11	unbenutzt

$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$
00	X	010 (Add)
X1	X	110 (Subtract)
1X	100000 (add)	010 (Add)
1X	100010 (sub)	110 (Subtract)
1X	100100 (and)	000 (And)
1X	100101 (or)	001 (Or)
1X	101010 (slt)	111 (SLT)

Embedded Systems & Applications

Steuerwerk: Hauptdecoder

Instruktion	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-Typ	000000							
lw	100011							
sw	101011							
beq	000100							



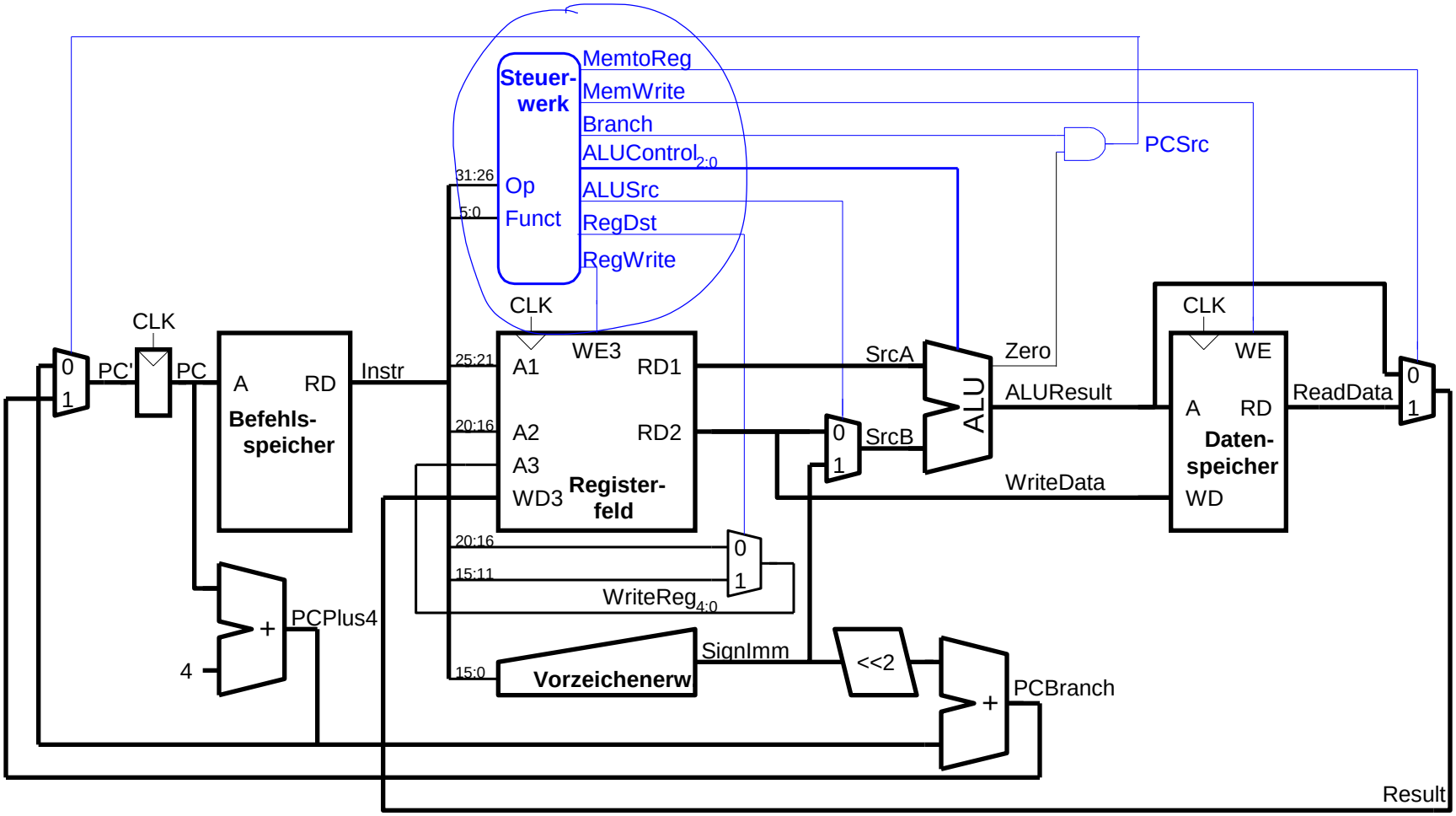
Steuerwerk: Hauptdecoder

TECHNISCHE UNIVERSITÄT DARMSTADT

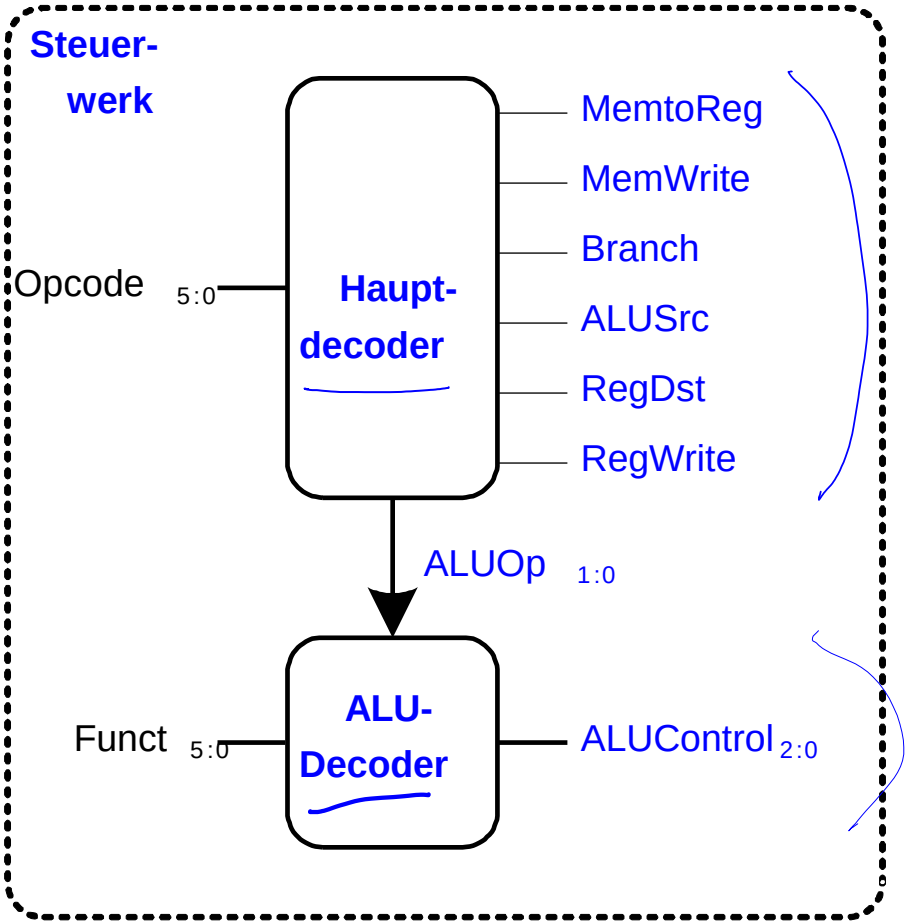
Instruktion	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-Typ	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01

Embedded Systems & Applications

Vollständiger Ein-Takt-Prozessor



Steuerwerk


Steuer-
werk
Opcode
5:0
Haupt-
decoder
MemtoReg
MemWrite
Branch
ALUSrc
RegDst
RegWrite
ALUOp
1:0
Funct
5:0
ALU-
Decoder
ALUControl
2:0

Steuerwerk: Hauptdecoder

TECHNISCHE UNIVERSITÄT DARMSTADT

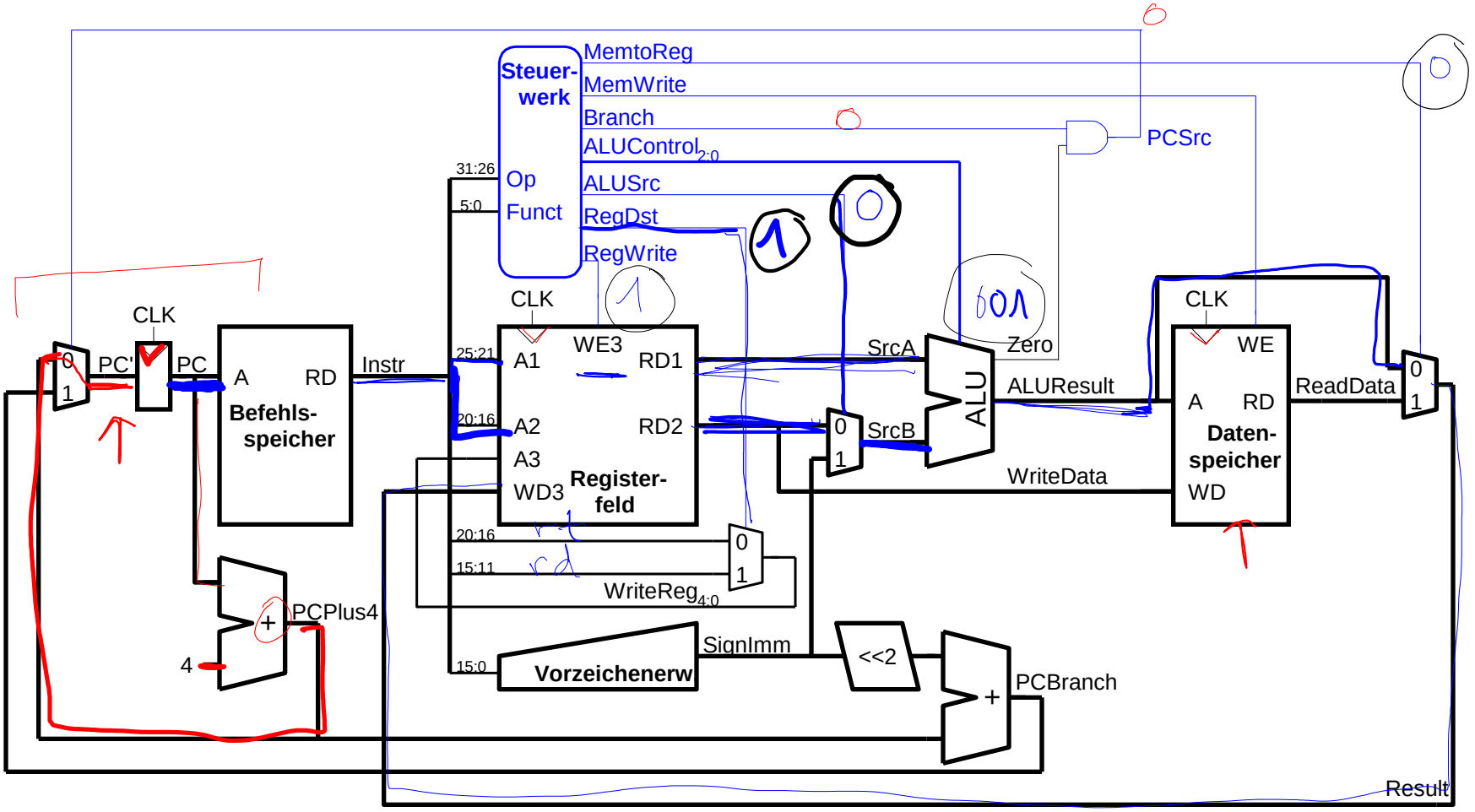
Instruktion	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-Typ	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01

$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$
00	X	010 (Add)
X1	X	110 (Subtract)
1X	100000 (`add`)	010 (Add)
1X	100010 (`sub`)	110 (Subtract)
1X	100100 (`and`)	000 (And)
1X	100101 (`or`)	001 (Or)
1X	101010 (`slt`)	111 (SLT)

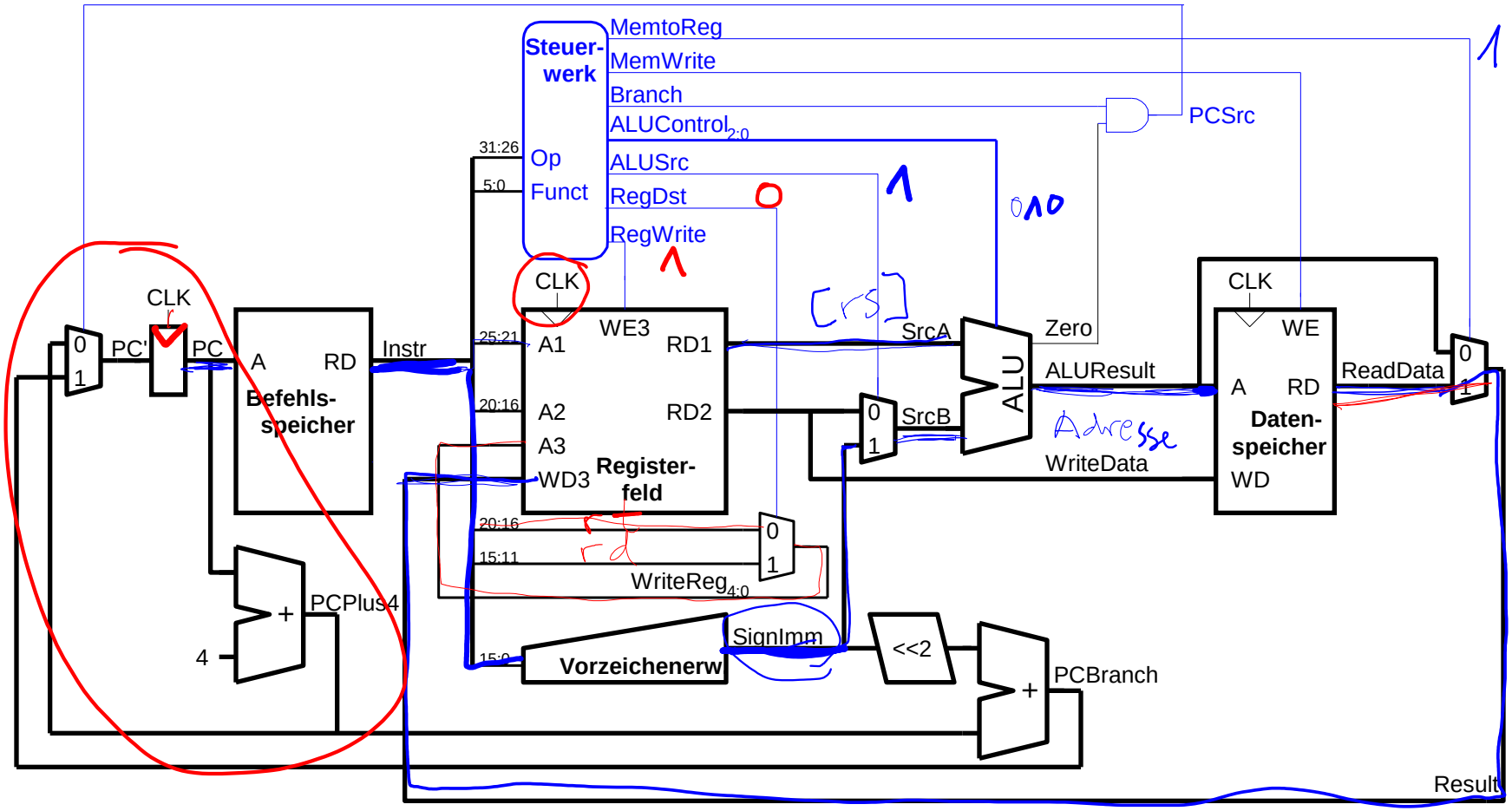
$ALUOp_{1:0}$	Bedeutung
00	Addiere
01	Subtrahiere
10	Werte Funct-Feld aus
11	unbenutzt

Embedded Systems & Applications

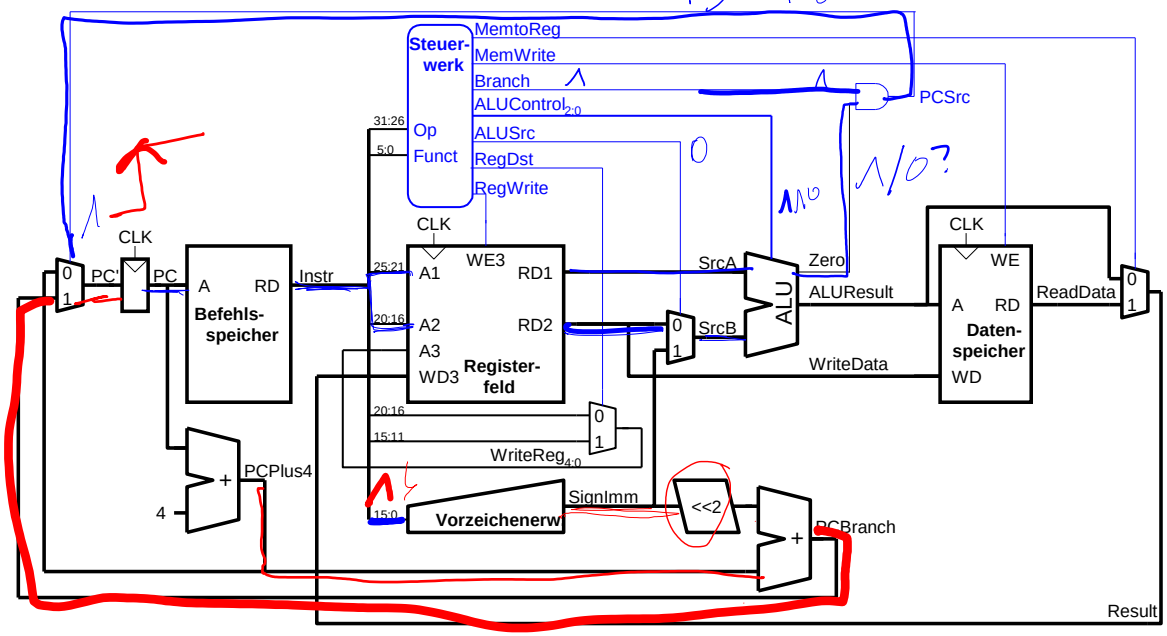
Vollständiger Ein-Takt-Prozessor



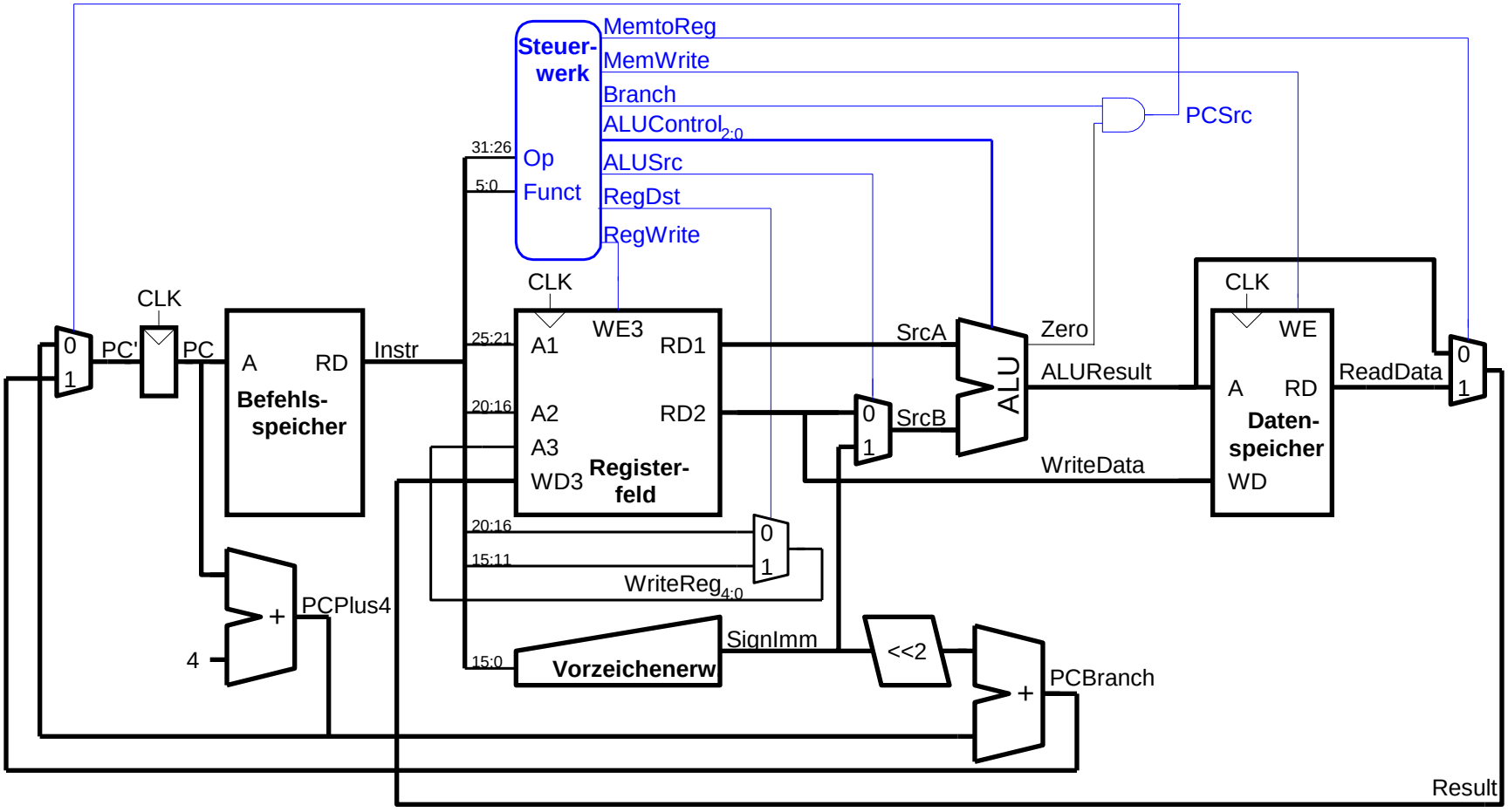
Vollständiger Ein-Takt-Prozessor



Vollständiger Ein-Takt-Prozessor

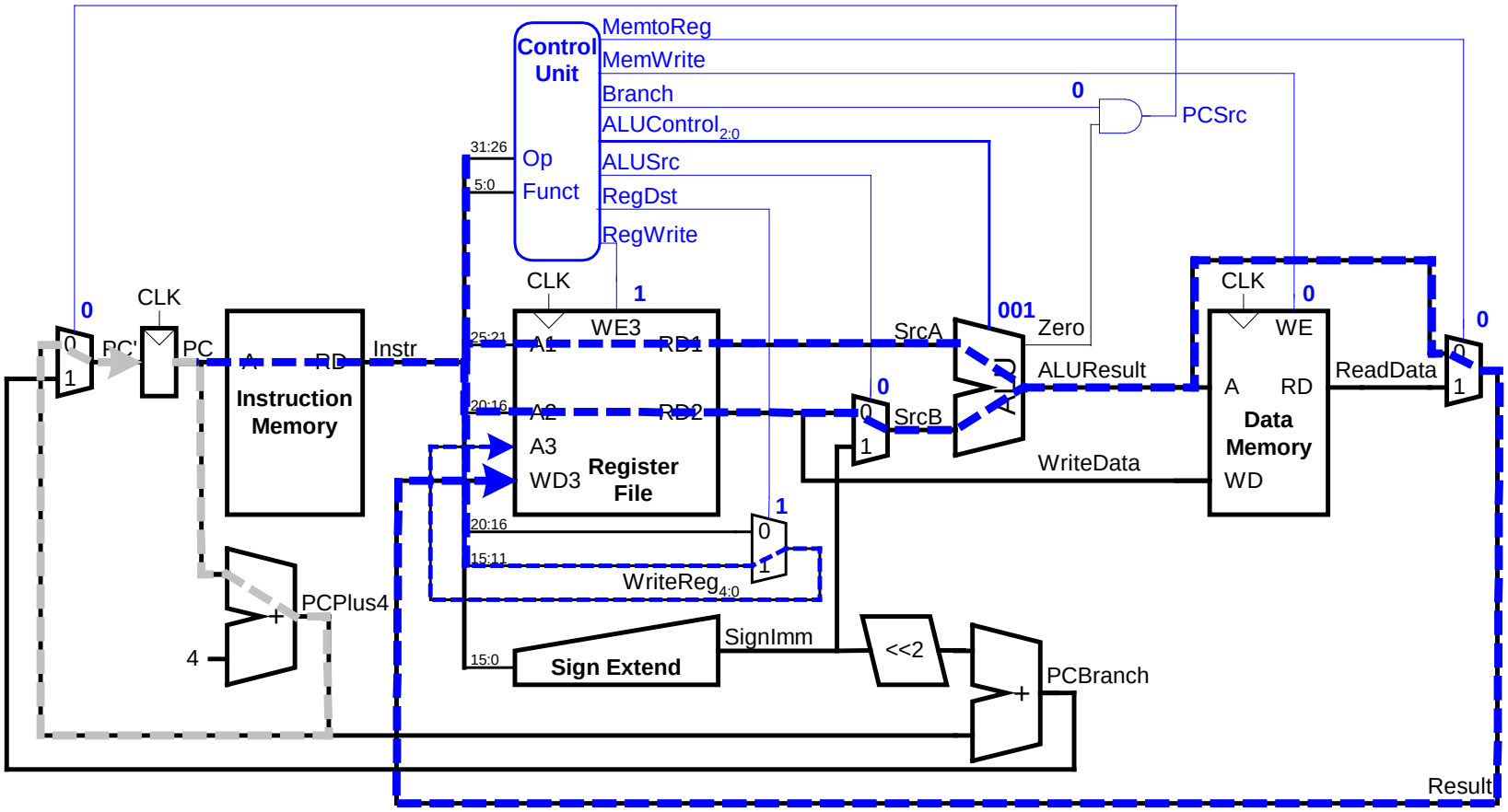


Vollständiger Ein-Takt-Prozessor



Beispiel im Ein-Takt Datenpfad:

or



Unser erster MIPS Prozessor

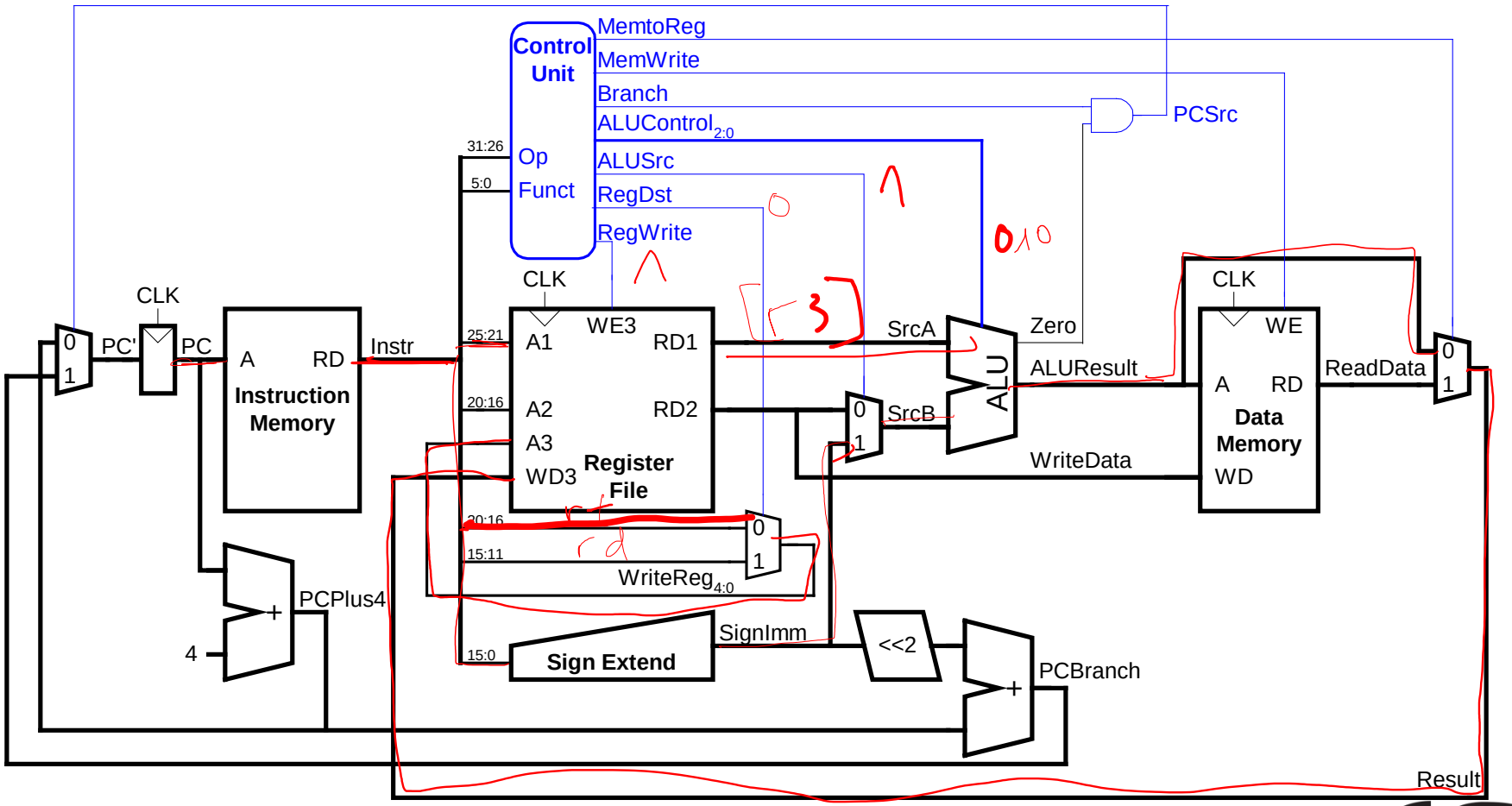
TECHNISCHE UNIVERSITÄT DARMSTADT

Zunächst Untermenge des MIPS Befehlssatzes:

- R-Typ Befehle: `and`, `or`, `add`, `sub`, `slt`
- Speicherbefehle: `lw`, `sw`
- Bedingte Verzweigungen: `beq`

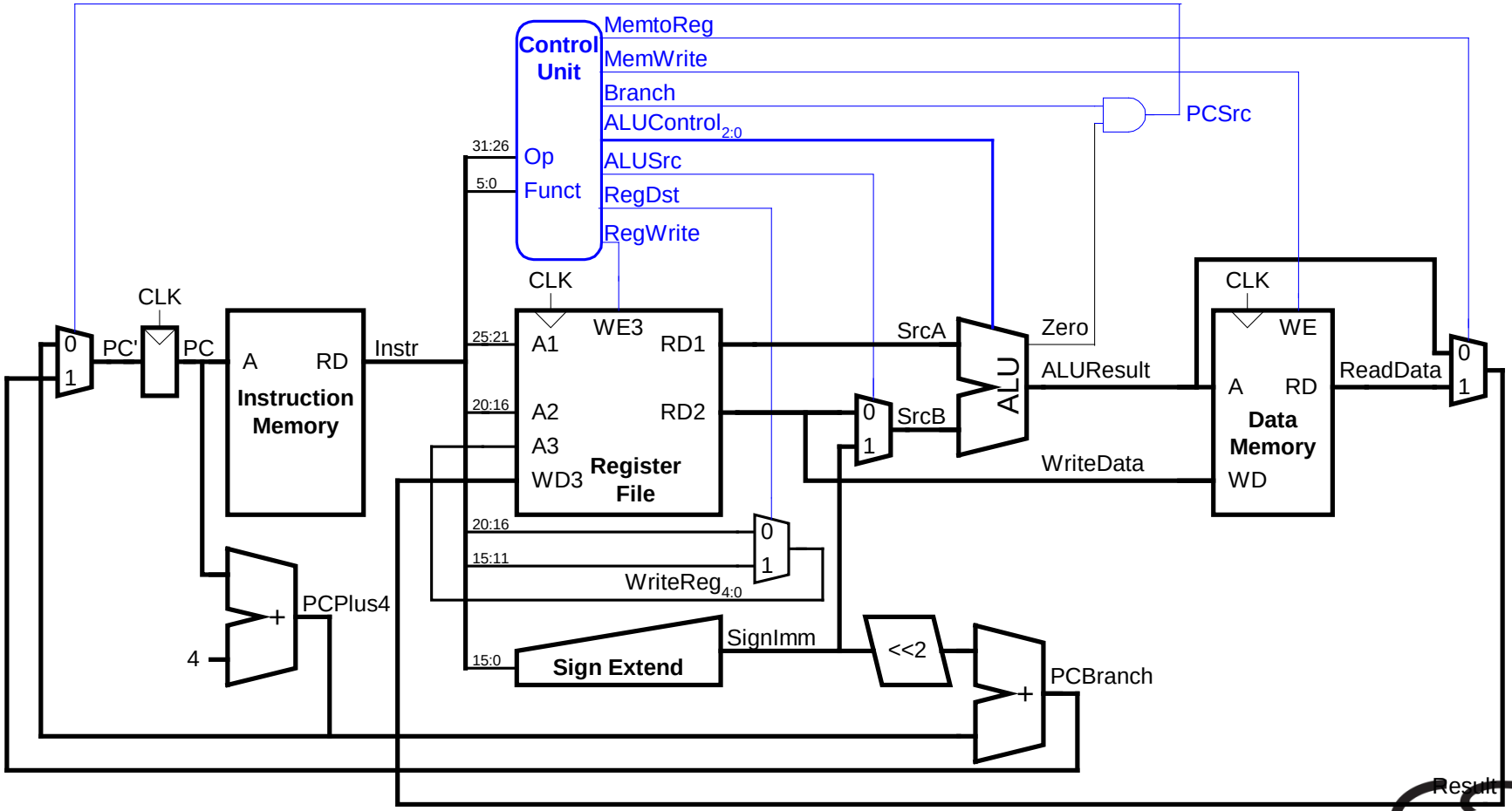
Später hinzunehmen: `addi` und `j`

Erweitere Funktionalität: addi



Erweitere Funktionalität: `addi`

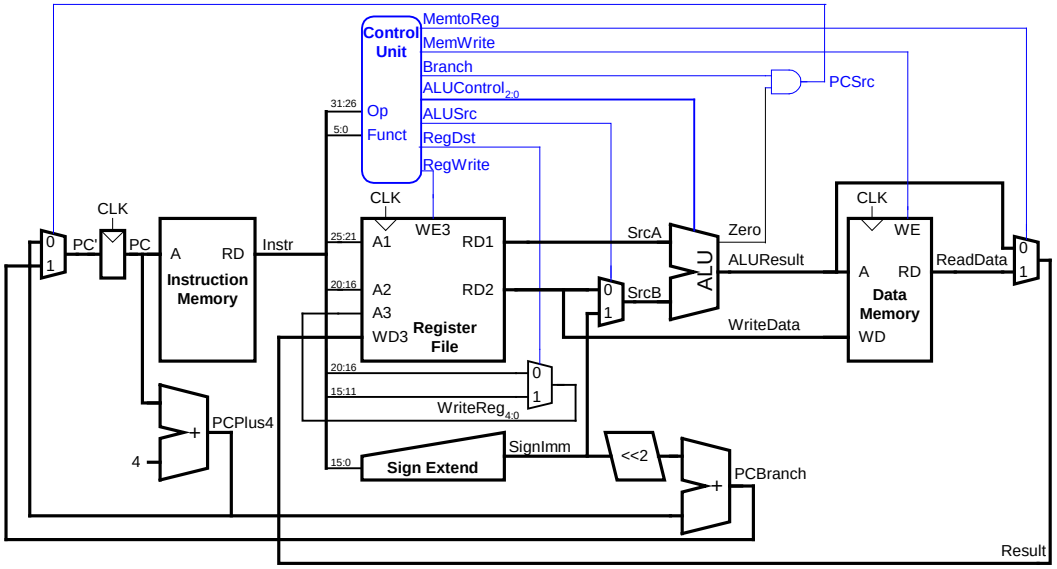
- Keine Änderung am Datenpfad nötig



Erweitere Steuerwerk: `addi`

TECHNISCHE UNIVERSITÄT DARMSTADT

Instruktion	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-Typ	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01
`addi`	**001000**							



Embedded Systems & Applications

Erweitere Steuerwerk: `addi`

TECHNISCHE UNIVERSITÄT DARMSTADT

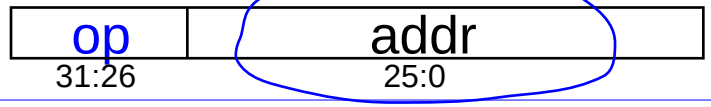
Instruktion	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-Typ	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01
`addi`	**001000**	**1**	**0**	**1**	**0**	**0**	**0**	**00**

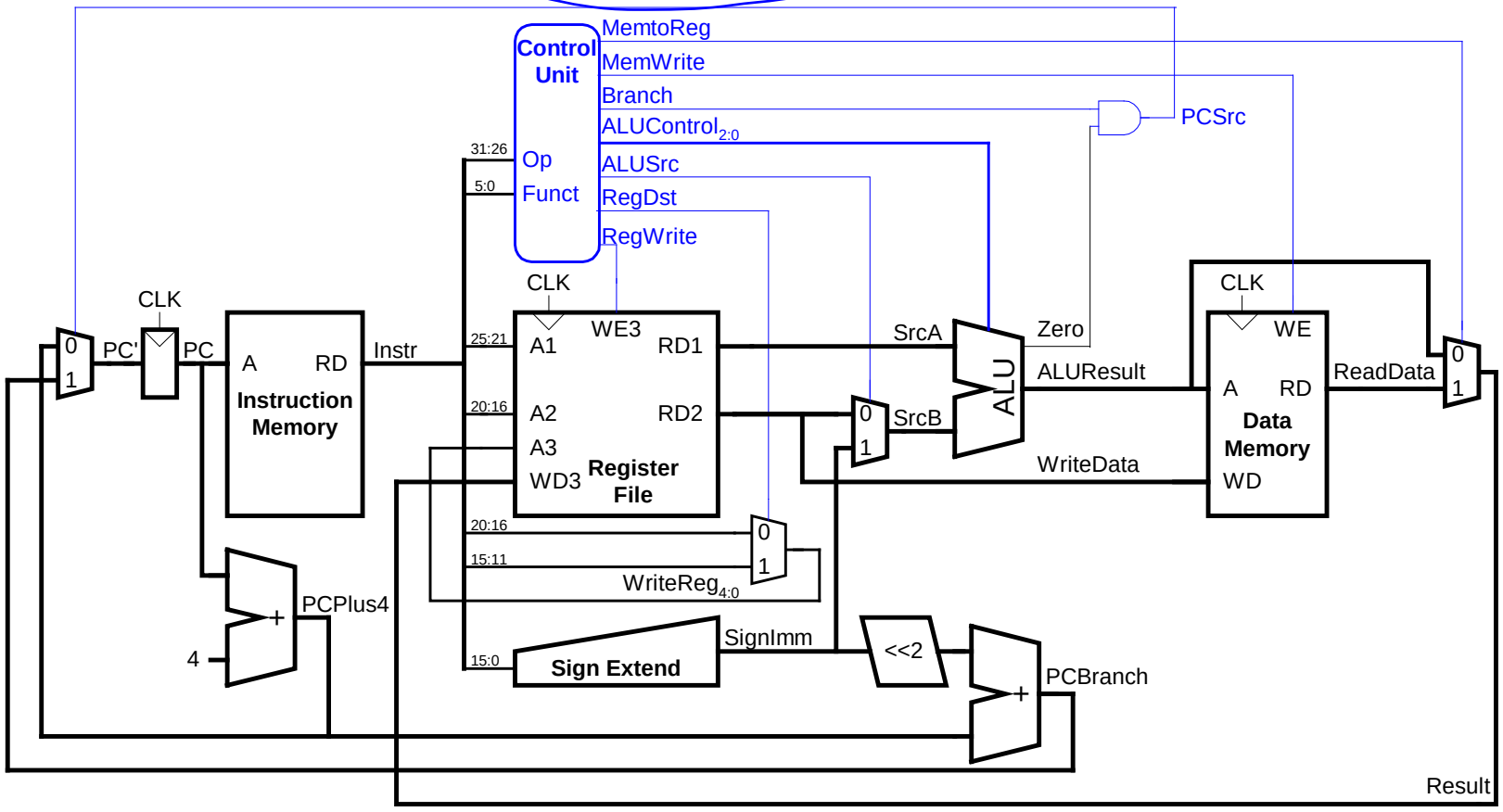
Embedded Systems & Applications

Erweitere Funktionalität: j

TECHNISCHE UNIVERSITÄT DARMSTADT

j Loop





Embedded Systems & Applications

Wiederholung: J-Typ Instruktion

TECHNISCHE UNIVERSITÄT DARMSTADT

Pseudodirekte Operanden

Auffüllen von entfallenen Bits (mit Nullen und (PC+4)[31:28])

```
0x0040005C      jal   sum
...
0x004000A0 sum: add   $v0, $a0, $a1
```

32b Sprungzieladresse 0000 0000 0100 0000 0000 0000 1010 0000 (0x004000A0)

26b Feld in J-Instruktion 0000 0000 0100 0000 0000 0000 1010 00 00 (0x0100028)

0 1 0 0 0 2 8

Bitfelder in Instruktion

op	imm
3	0x0100028
6 bits	26 bits

Maschinencode

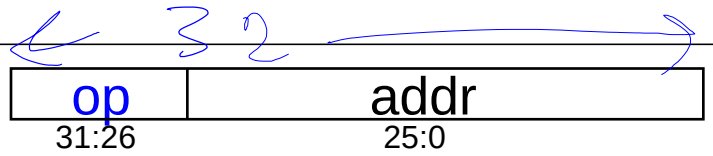
op	addr	
000011	00 0001 0000 0000 0000 0010 1000	(0x0C100028)
6 bits	26 bits	

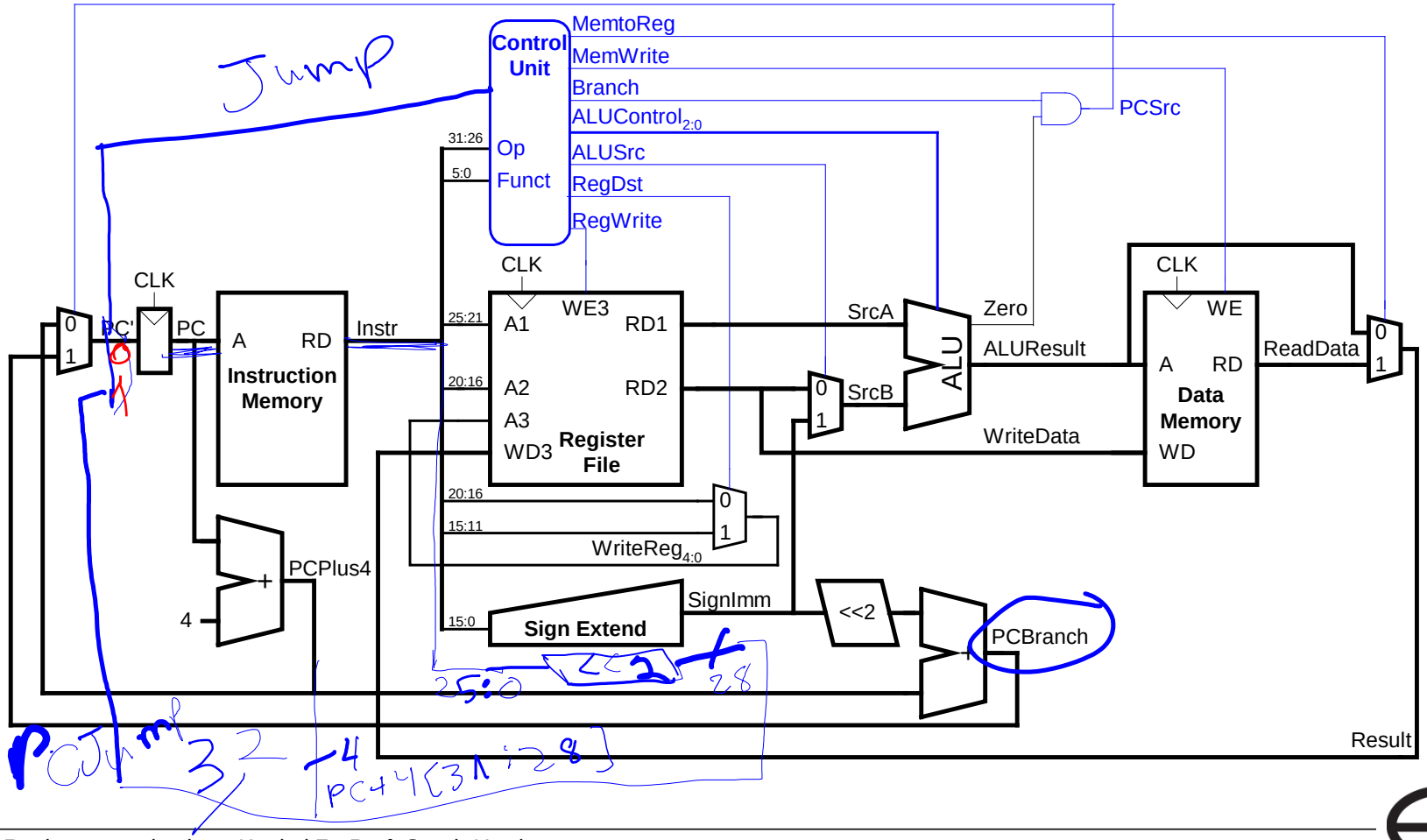
Embedded Systems & Applications

Erweitere Funktionalität: j

TECHNISCHE UNIVERSITÄT DARMSTADT

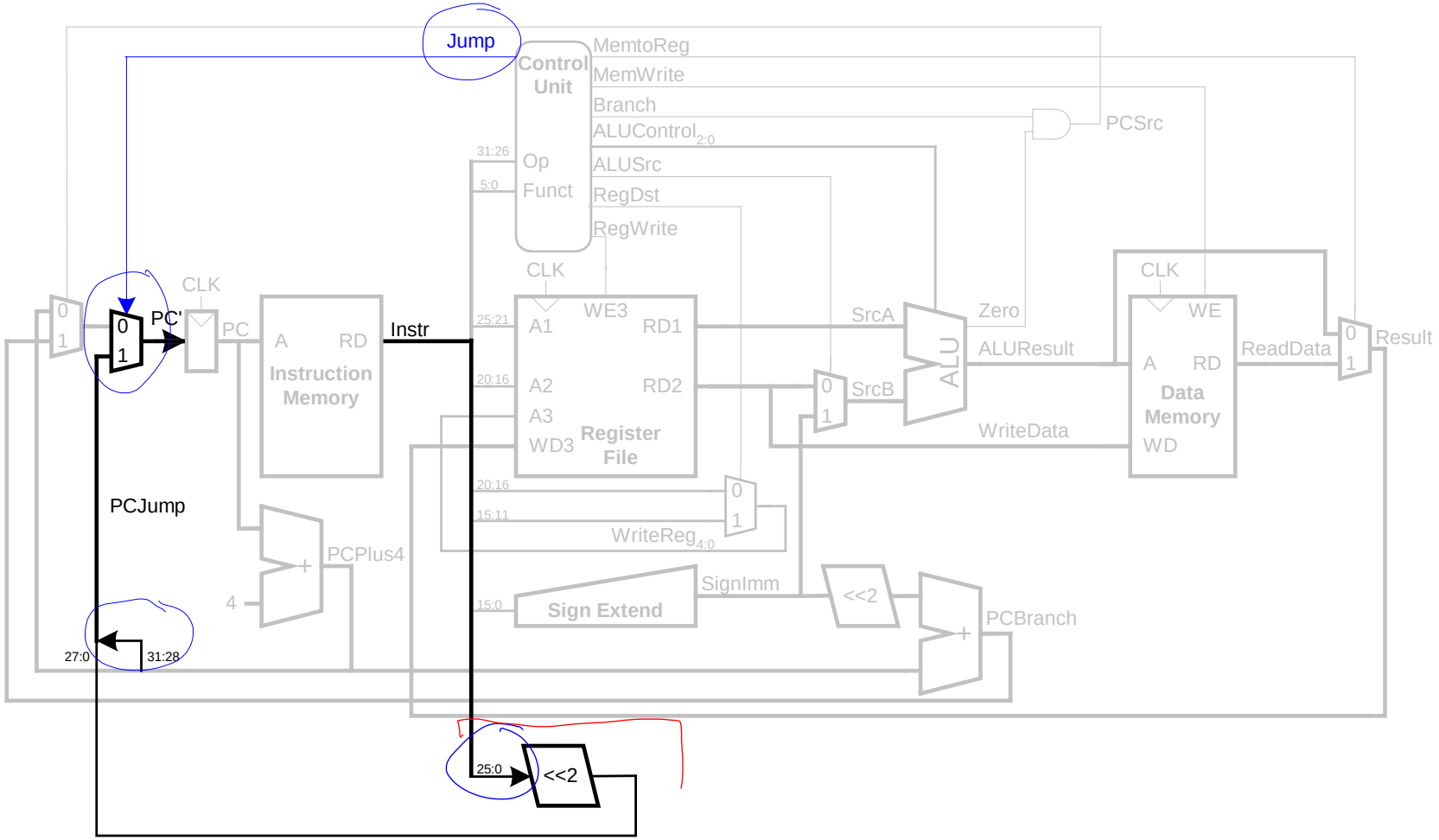
j Loop





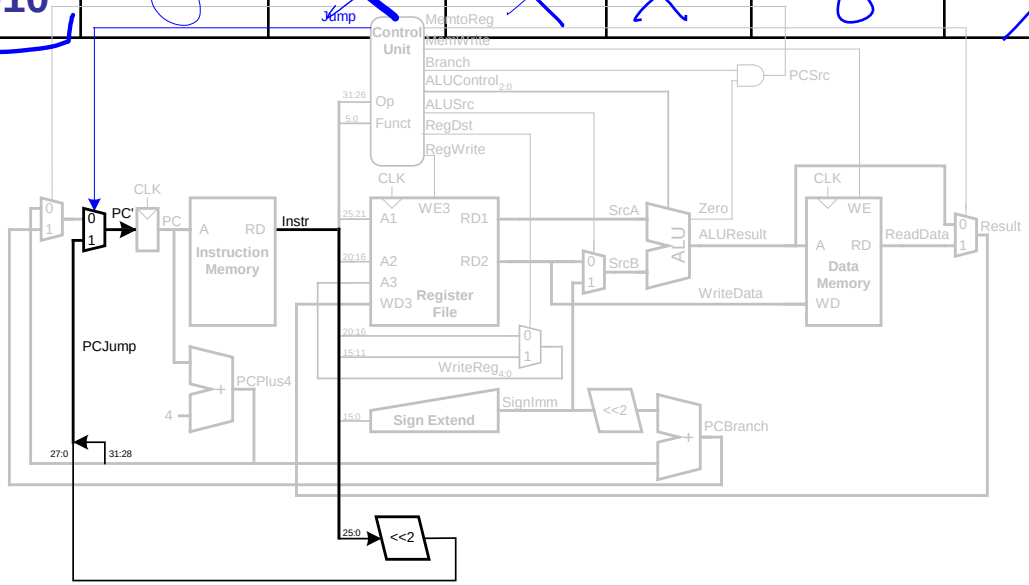
Embedded Systems & Applications

Erweitere Funktionalität: j



Steuerwerk: Hauptdecoder

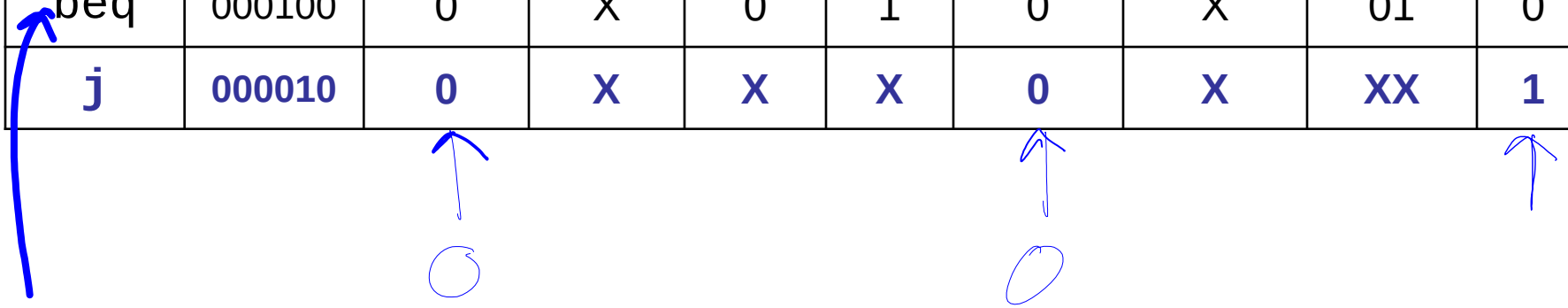
Instruktion	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-Typ	000000	1	1	0	0	0	0	10	0
lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	X	1	0	1	X	00	0
beq	000100	0	X	0	1	0	X	01	0
j	000010	0	X	X	X	0	X	XX	1



Steuerwerk: Hauptdecoder

TECHNISCHE UNIVERSITÄT DARMSTADT

Instruktion	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-Typ	000000	1	1	0	0	0	0	10	0
lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	X	1	0	1	X	00	0
beq	000100	0	X	0	1	0	X	01	0
j	**000010**	**0**	**X**	**X**	**X**	**0**	**X**	**XX**	**1**



Embedded Systems & Applications

Wiederholung: Rechenleistung des Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Programmausführungszeit

= (# Instruktionen) (Takte/Instruktion)(Sekunden/Takt)

= # Instruktionen * CPI * T_C

vom Programm

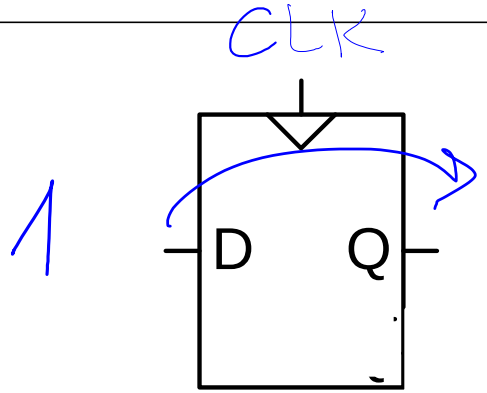
CLK

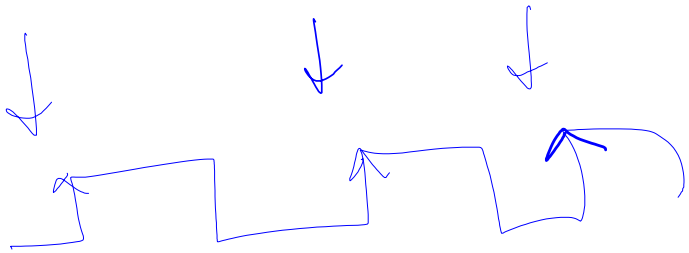
T_C

Taktperiode

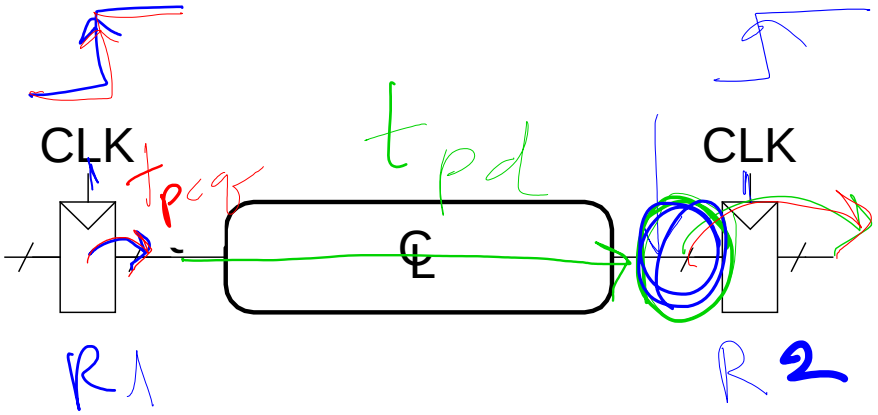
Embedded Systems & Applications

Wiederholung: Verhalten eines Flip-Flops





Taktperiode (Clock Cycle Time)

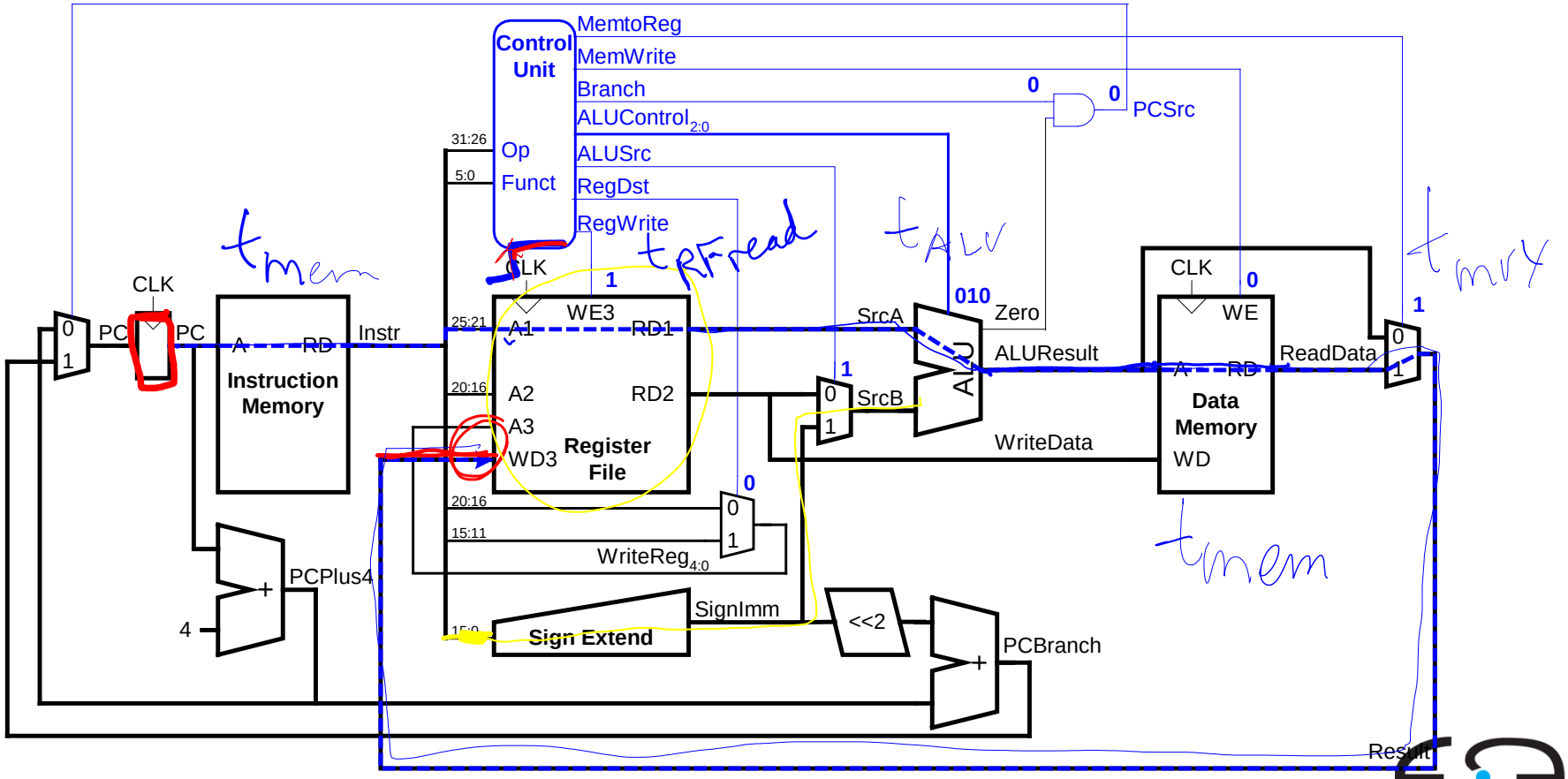


$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$

Rechenleistung des Ein-Takt-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

- T_C wird durch längsten Pfad bestimmt (lw)



Embedded Systems & Applications

Rechenleistung des Ein-Takt-Prozessors

- **Kritischer Pfad:**

$$T_c = t_{pcq\_PC} + t_{mem} + \max(t_{RFread}, t_{sext} + t_{mux}) + t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup}$$

In vielen Implementierungen: Kritischer Pfad durch

- Speicher, ALU, Registerfeld

Damit:

- $\boldsymbol{T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup}}$

Ein-Takt Prozessor Rechenleistung: Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Element	Parameter	Verzögerung (ps)
Register Clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Speicher lesen	t_{mem}	250
Registerfeld lesen	t_{RFread}	150
Registerfeld setup	$t_{RFsetup}$	20

$$T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup}$$

$$= 30 + 2(250) + 150 + 25 + 200 + 20$$

Embedded Systems & Applications

Ein-Takt Prozessor Rechenleistung: Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Element	Parameter	Verzögerung (ps)
Register Clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Speicher lesen	t_{mem}	250
Registerfeld lesen	t_{RFread}	150
Registerfeld setup	$t_{RFsetup}$	20

$$T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup}$$
$$= [30 + 2(250) + 150 + 25 + 200 + 20]\ ps$$
$$= 925\ ps$$

$f_c = \frac{1}{T_c} \approx 1\ GHz$

Embedded Systems & Applications

Ein-Takt Prozessor Rechenleistung: Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auszuführen: Programm mit 100 Milliarden Instruktionen auf Ein-Takt MIPS Prozessor

Ausführungszeit = ?

Embedded Systems & Applications

Ein-Takt Prozessor Rechenleistung: Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auszuführen: Programm mit 100 Milliarden Instruktionen auf Ein-Takt MIPS Prozessor

Ausführungszeit = # Instruktionen * CPI * T_C

$100 \times 10^9 \times 1 \times 925ps$

Embedded Systems & Applications

Ein-Takt Prozessor Rechenleistung: Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auszuführen: Programm mit 100 Milliarden Instruktionen auf Ein-Takt MIPS Prozessor

Ausführungszeit = # Instruktionen * CPI * T_C

$= (100 \times 10^9)\ (1)\ (925 \times 10^{-12}\ s)$

= 92,5 Sekunden

Embedded Systems & Applications

Mikroarchitektur

TECHNISCHE UNIVERSITÄT DARMSTADT

Mehrere Implementierungen für eine Architektur

- **Ein-Takt**

  Jede Instruktion wird in einem Takt ausgeführt

- **Mehrtakt**

  Jede Instruktion wird in Teilschritte zerlegt

- **Pipelined**

  Jede Instruktion wird in Teilschritte zerlegt

  Mehrere Instruktionen werden gleichzeitig ausgeführt

Embedded Systems & Applications

Mehrtakt-MIPS-Prozessor

TECHNISCHE UNIVERSITÄT DARMSTADT

Ein-Takt-Mikroarchitektur:

+ einfach
- Taktfrequenz wird durch langsamste Instruktion bestimmt (`lw`)
- Drei Addierer / ALUs und zwei Speicher

Mehrtaktmikroarchitektur:

+ höhere Taktfrequenz
+ einfachere Instruktionen laufen schneller
+ bessere Wiederverwendung von Hardware in verschiedenen Takten
- aufwendigere Ablaufsteuerung

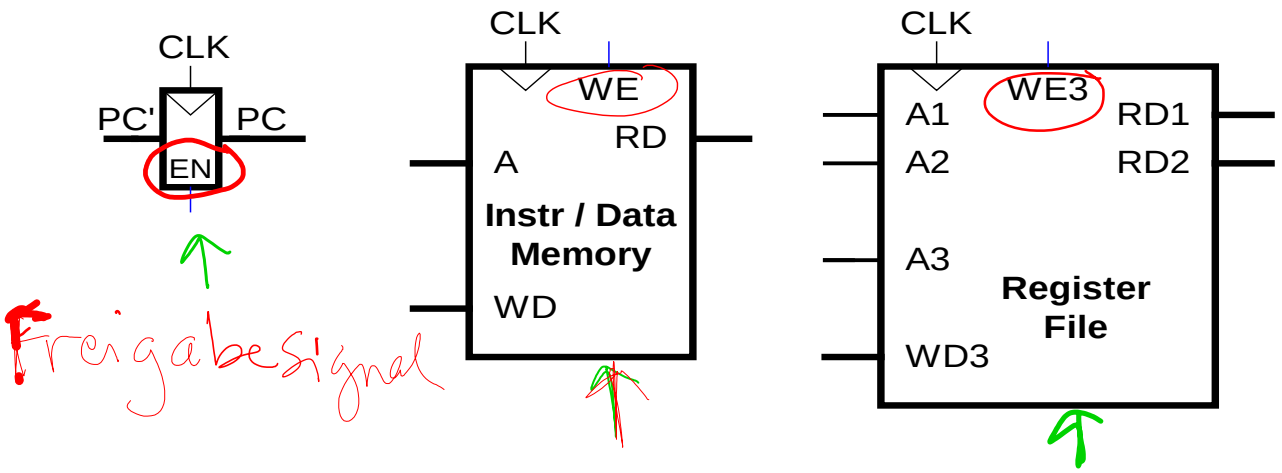
Gleiche Grundkomponenten

- Datenpfad
- Steuerwerk

Embedded Systems & Applications

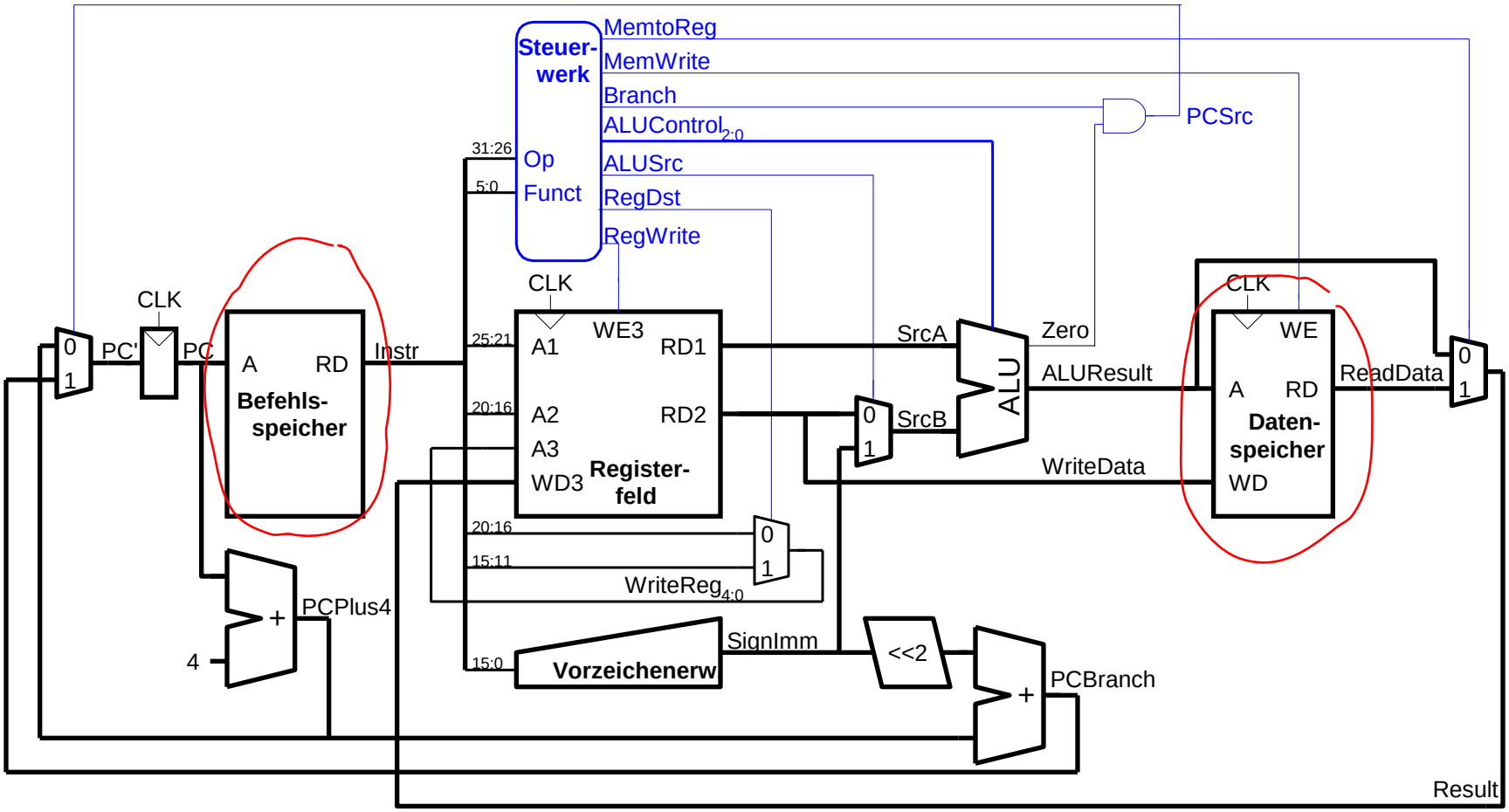
Zustandselemente im Mehrtaktprozessor

- Ersetze getrennte Instruktions- und Datenspeicher
  - Harvard-Architektur
- Durch einen gemeinsamen Speicher
  - Von Neumann-Architektur
  - Heute weiter verbreitet



Vollständiger Ein-Takt-Prozessor

2 Speicherelemente



Mehrtaktprozessor: Arbeit in einem Takt

TECHNISCHE UNIVERSITÄT DARMSTADT

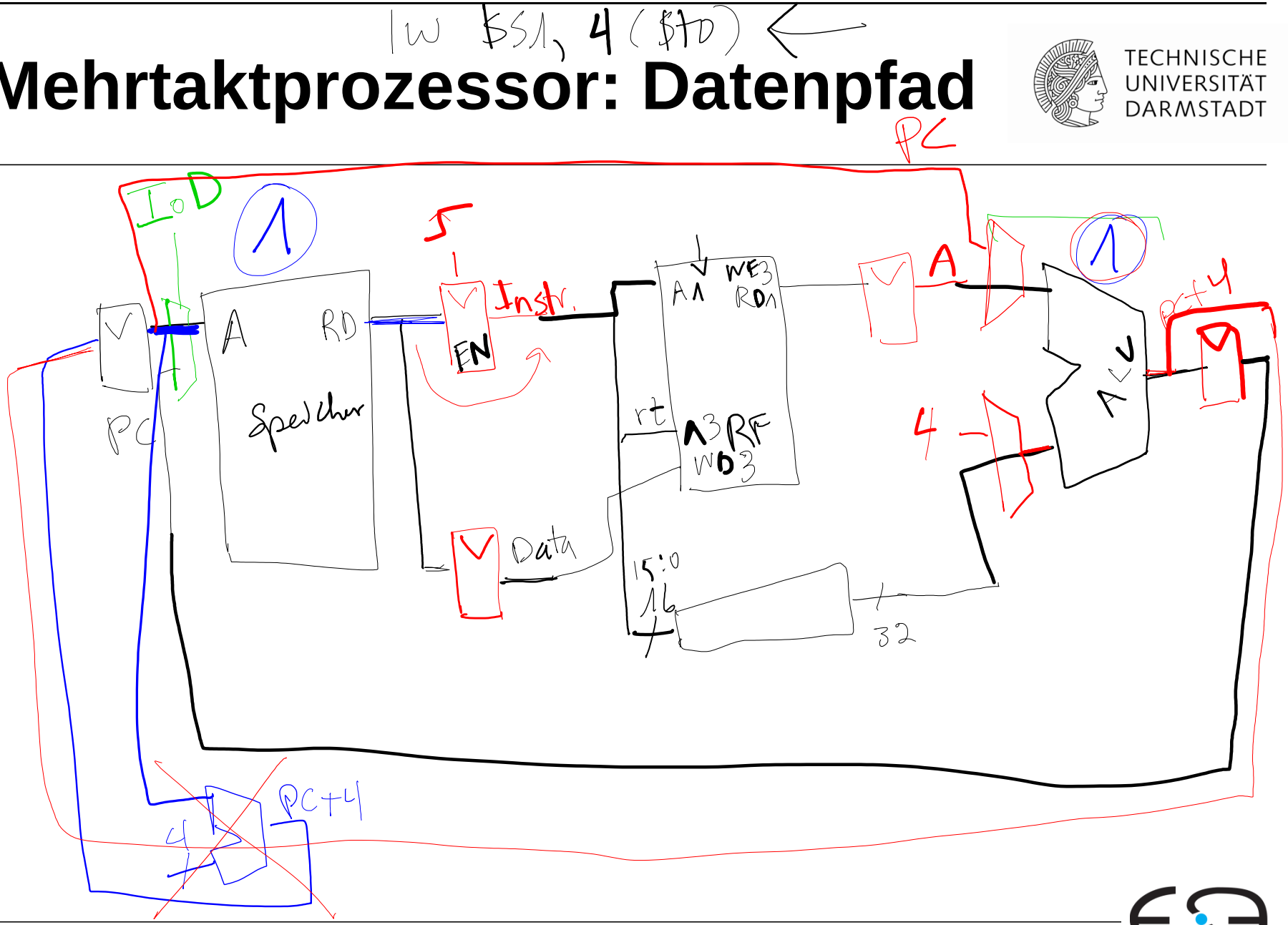
Darf nur einen benutzen in einem Takt:

ALU, Speicher, Registerfeld

...damit die Takte kurz bleiben

Embedded Systems & Applications

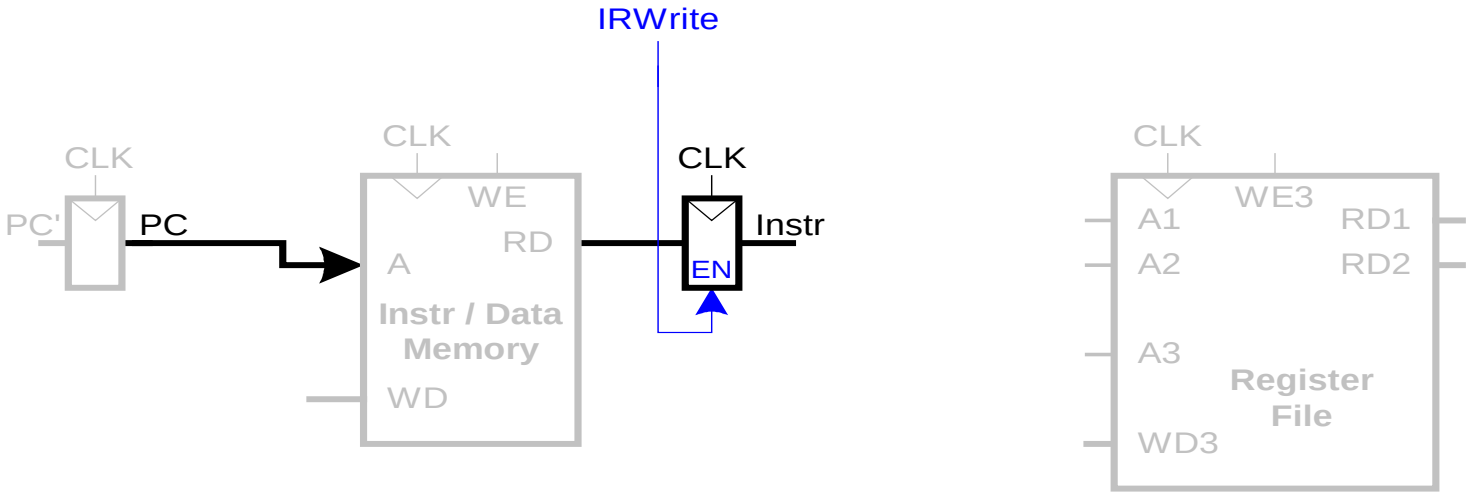
Mehrtaktprozessor: Datenpfad



Mehrtaktdatenpfad: Instruktionen holen (*fetch*)

Beispiel: Ausführung von `lw`

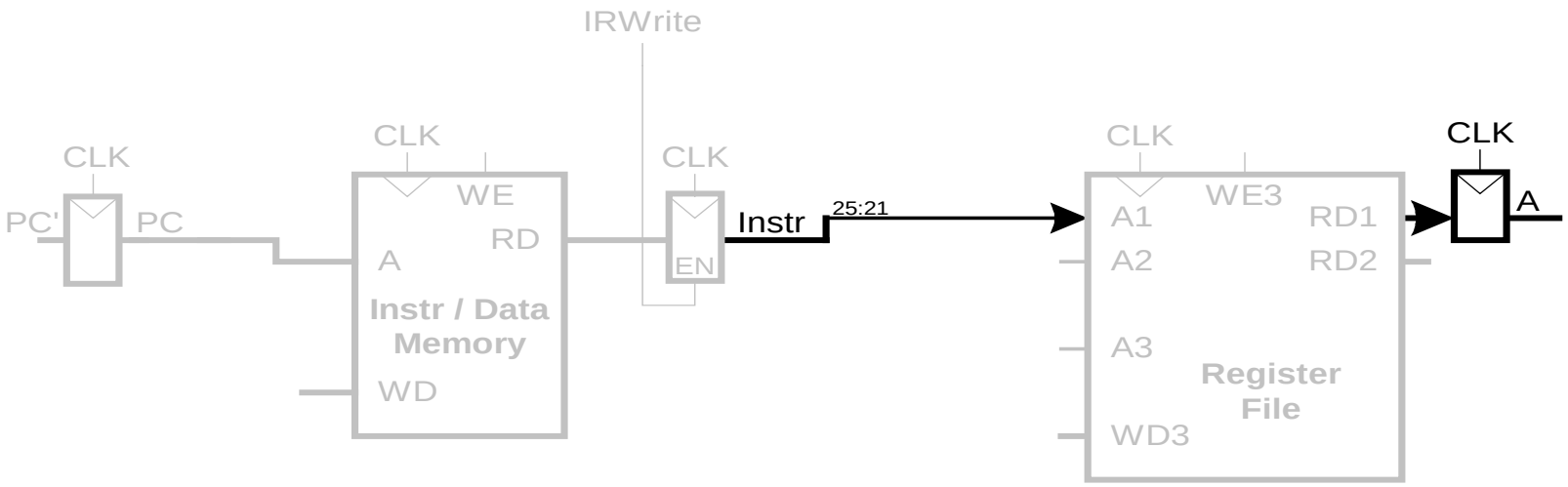
Schritt 1: Hole Instruktion



Mehrtaktdatenpfad: Lese Register für `lw`

TECHNISCHE UNIVERSITÄT DARMSTADT

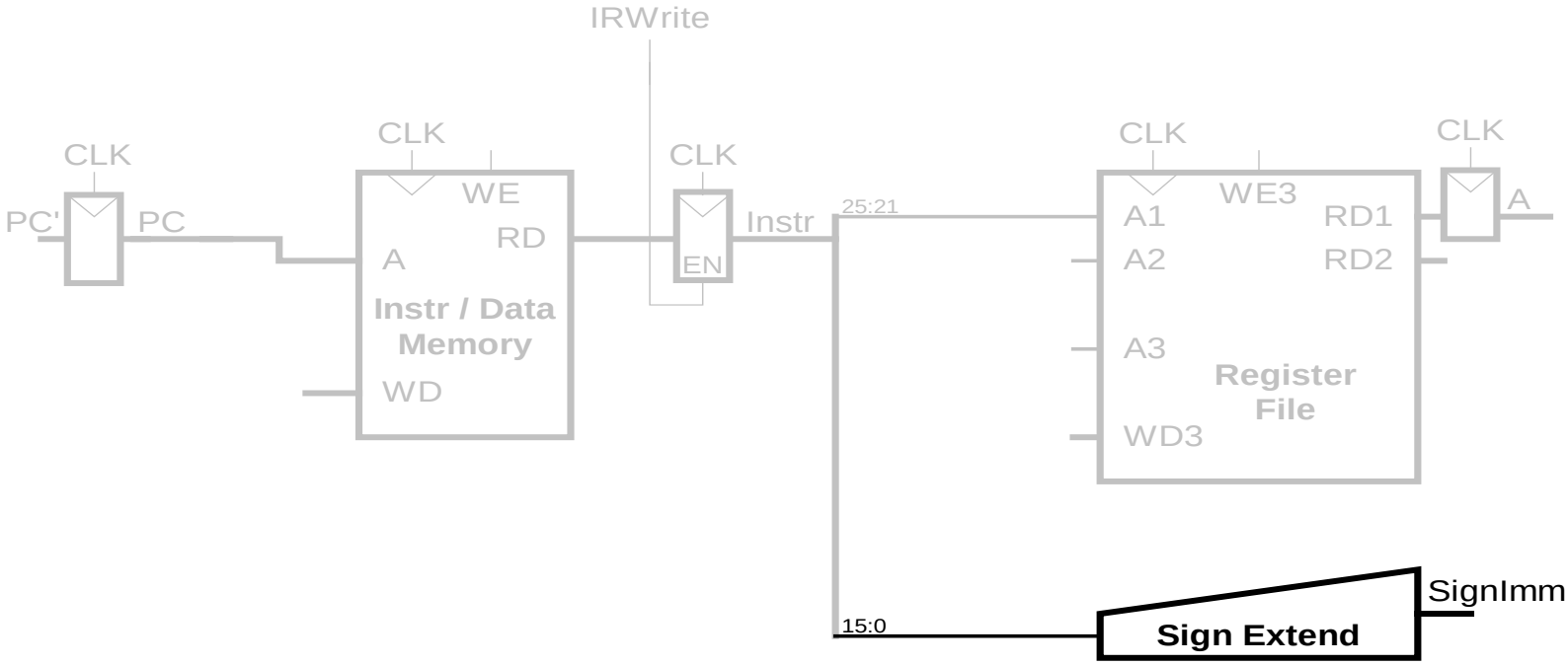
Schritt 2a: Lese Quelloperand aus Registerfeld



Embedded Systems & Applications

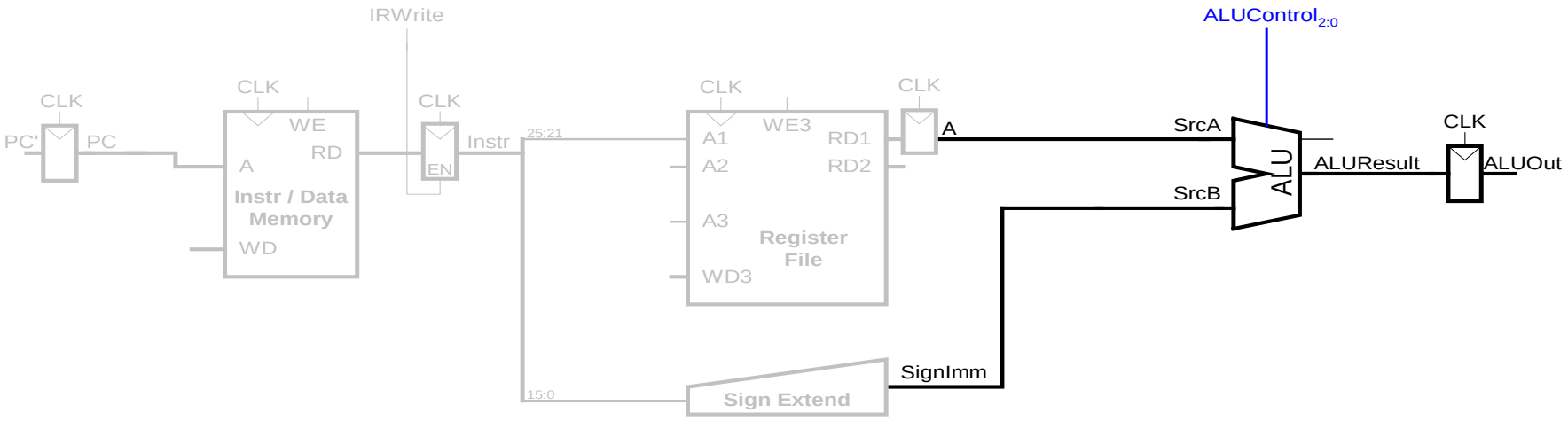
Mehrtaktdatenpfad: Werte `lw` Direktwert aus

Schritt 2b: Vorzeichenerweitere den 16b Direktwert auf 32b Signal `SignImm`



Mehrtaktdatenpfad: Bestimme effektive Adresse für `lw`

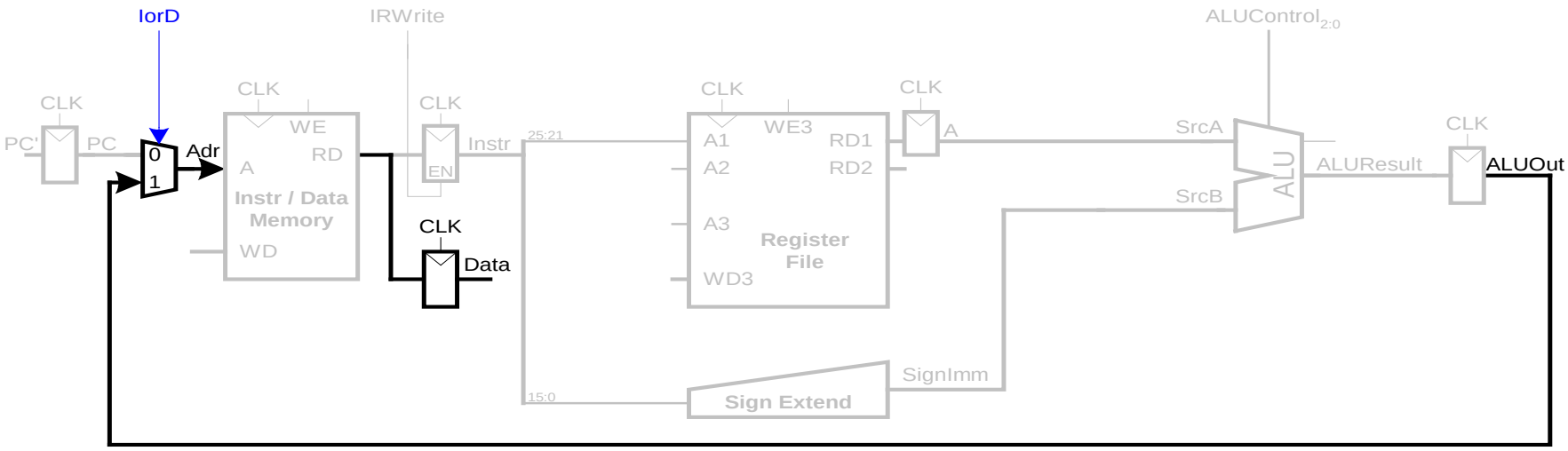
Schritt 3: Berechne die effektive Speicheradresse



Mehrtaktdatenpfad: Lesezugriff von `lw`

TECHNISCHE UNIVERSITÄT DARMSTADT

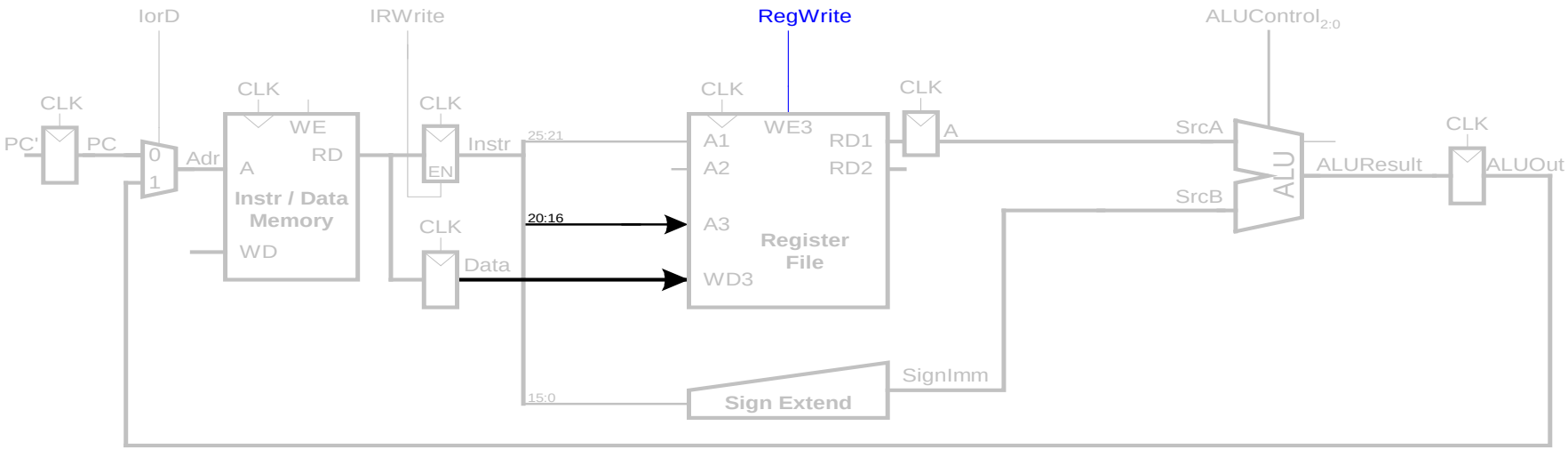
Schritt 4: Lese Daten aus Speicher



Embedded Systems & Applications

Mehrtaktdatenpfad: Schreibe Register in `lw`

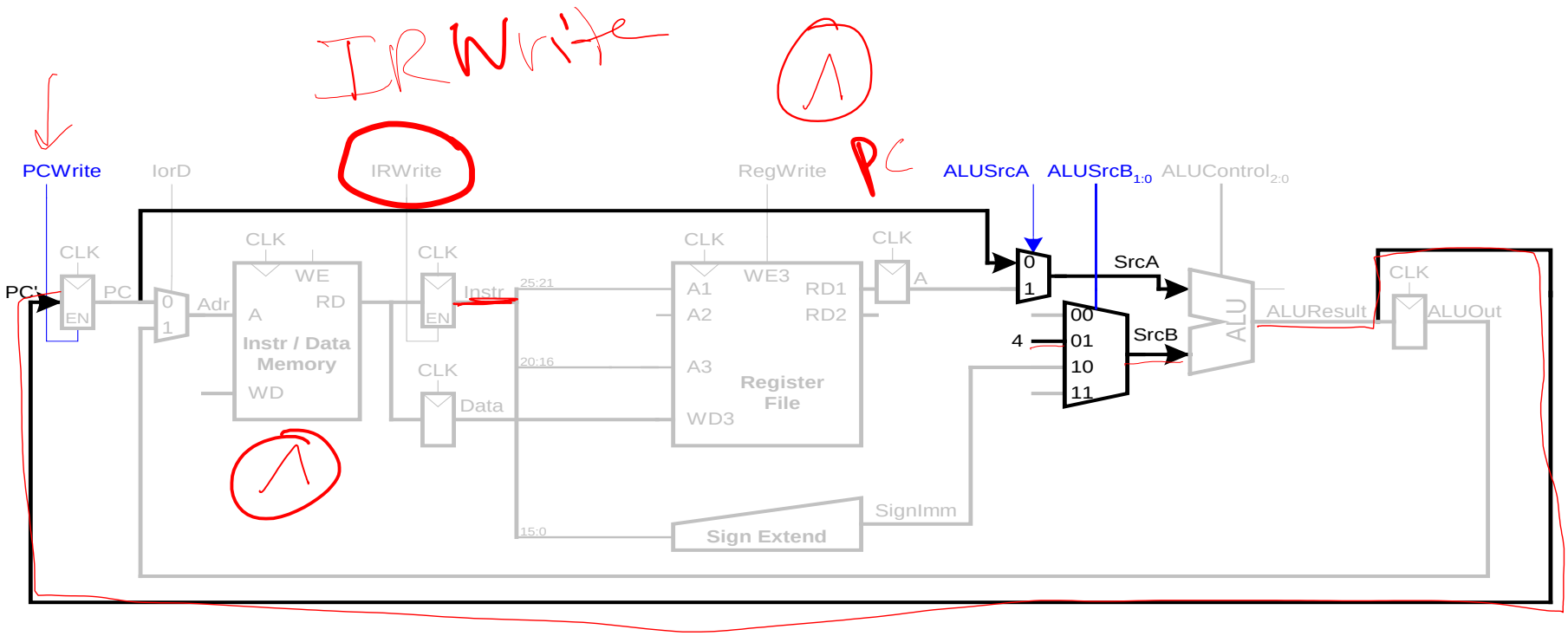
Schritt 5: Schreibe die Daten ins passende Register



Mehrtaktdatenpfad: Erhöhe PC

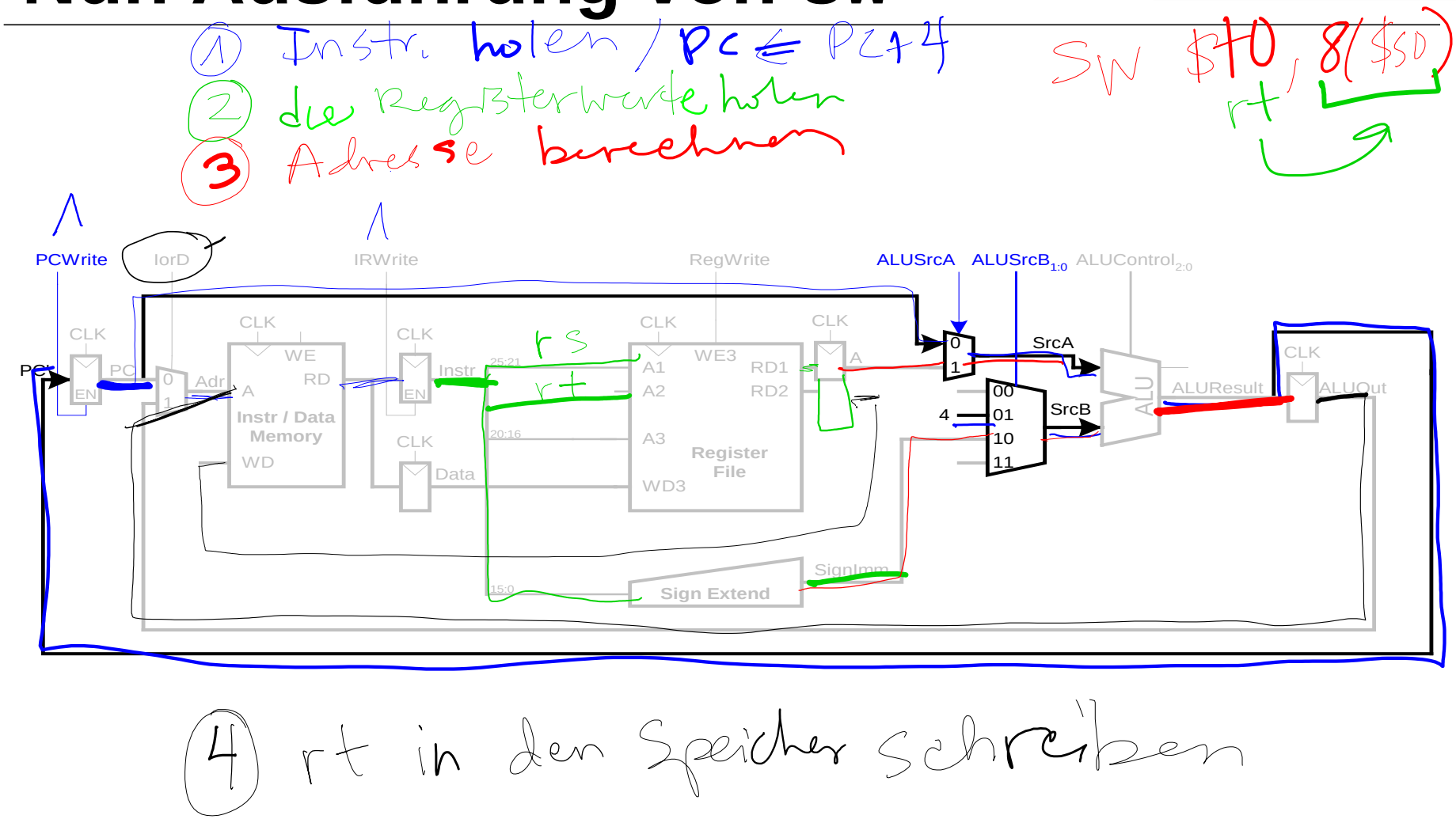
TECHNISCHE UNIVERSITÄT DARMSTADT

Schritt 6: Bestimme Adresse des nächsten Befehls



Embedded Systems & Applications

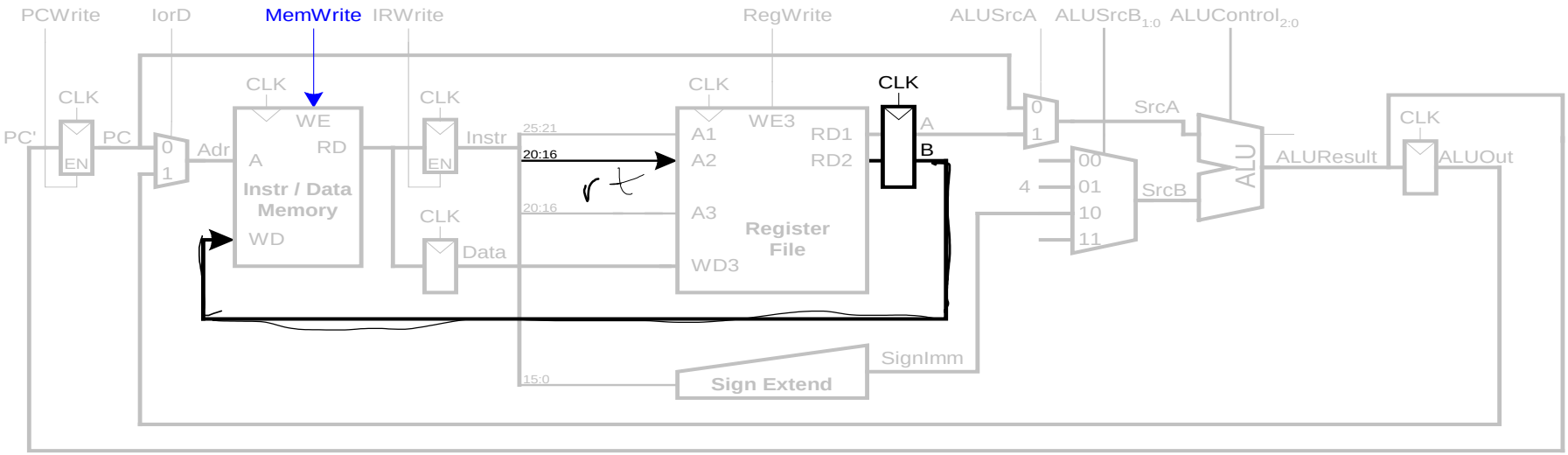
Mehrtaktdatenpfad: Nun Ausführung von sw



Mehrtaktdatenpfad: Nun Ausführung von sw

TECHNISCHE UNIVERSITÄT DARMSTADT

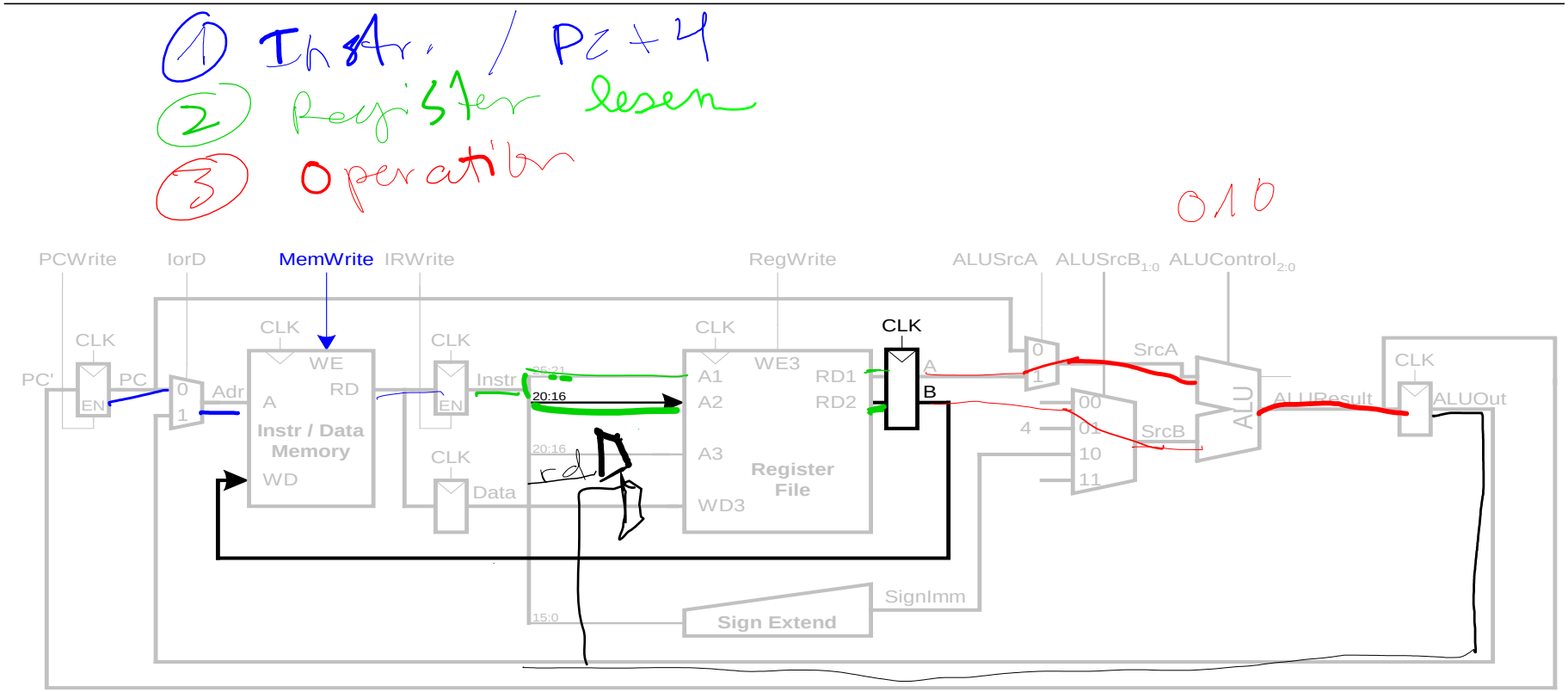
- Schreibe Daten aus `rt` in Speicher



Embedded Systems & Applications

Mehrtaktdatenpfad: Instruktion vom R-Typ

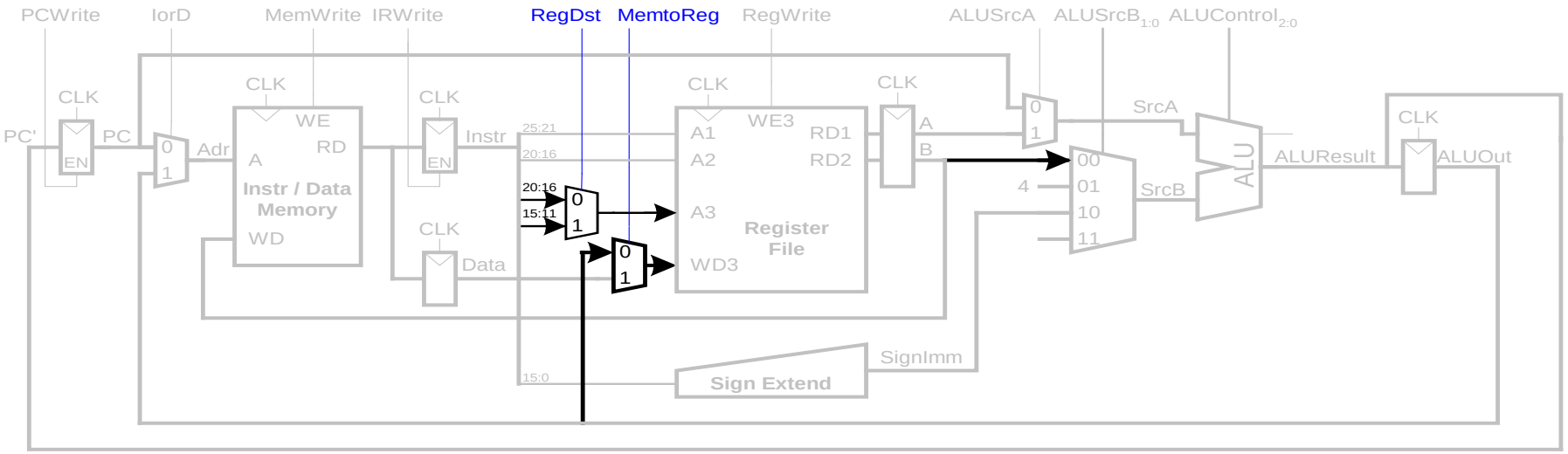
TECHNISCHE UNIVERSITÄT DARMSTADT



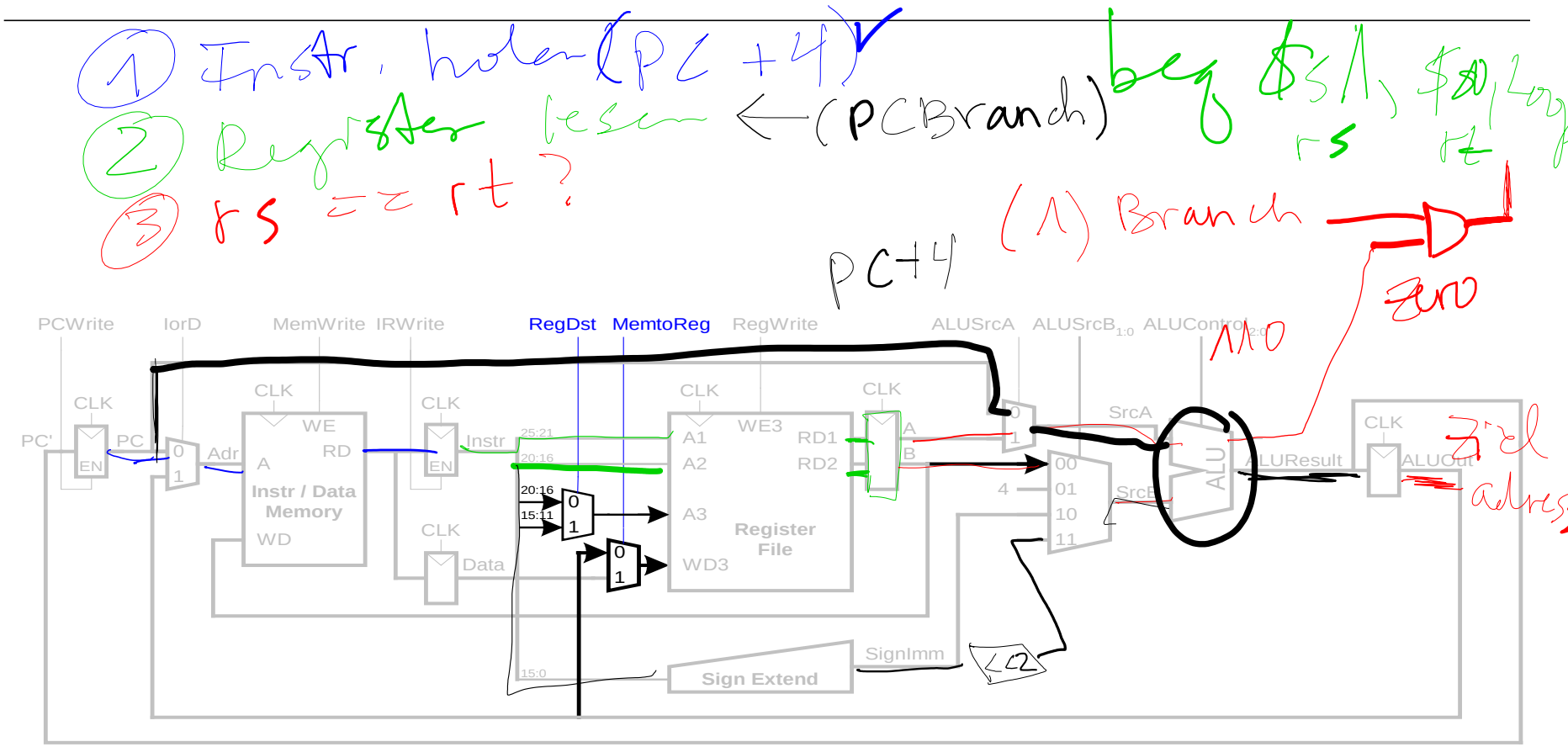
Embedded Systems & Applications

Mehrtaktdatenpfad: Instruktion vom R-Typ

- Lese Werte aus `rs` und `rt`
- Schreibe *ALUResult* ins Registerfeld
- Schreibe Wert nach `rd` (statt nach `rt`)



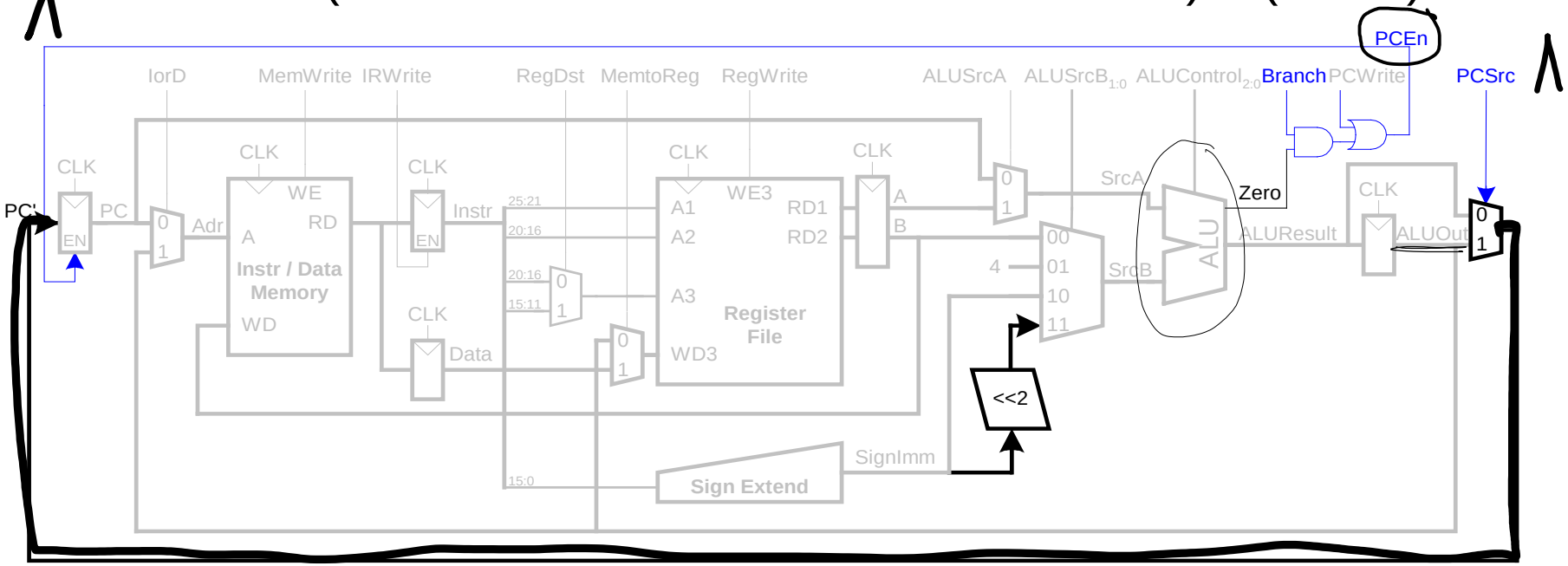
Mehrtaktdatenpfad: beq



Mehrtaktdatenpfad: beq

- Prüfe, ob Werte in `rs` und `rt` gleich sind
- Bestimme Adresse des Sprungziels *(branch target address):*

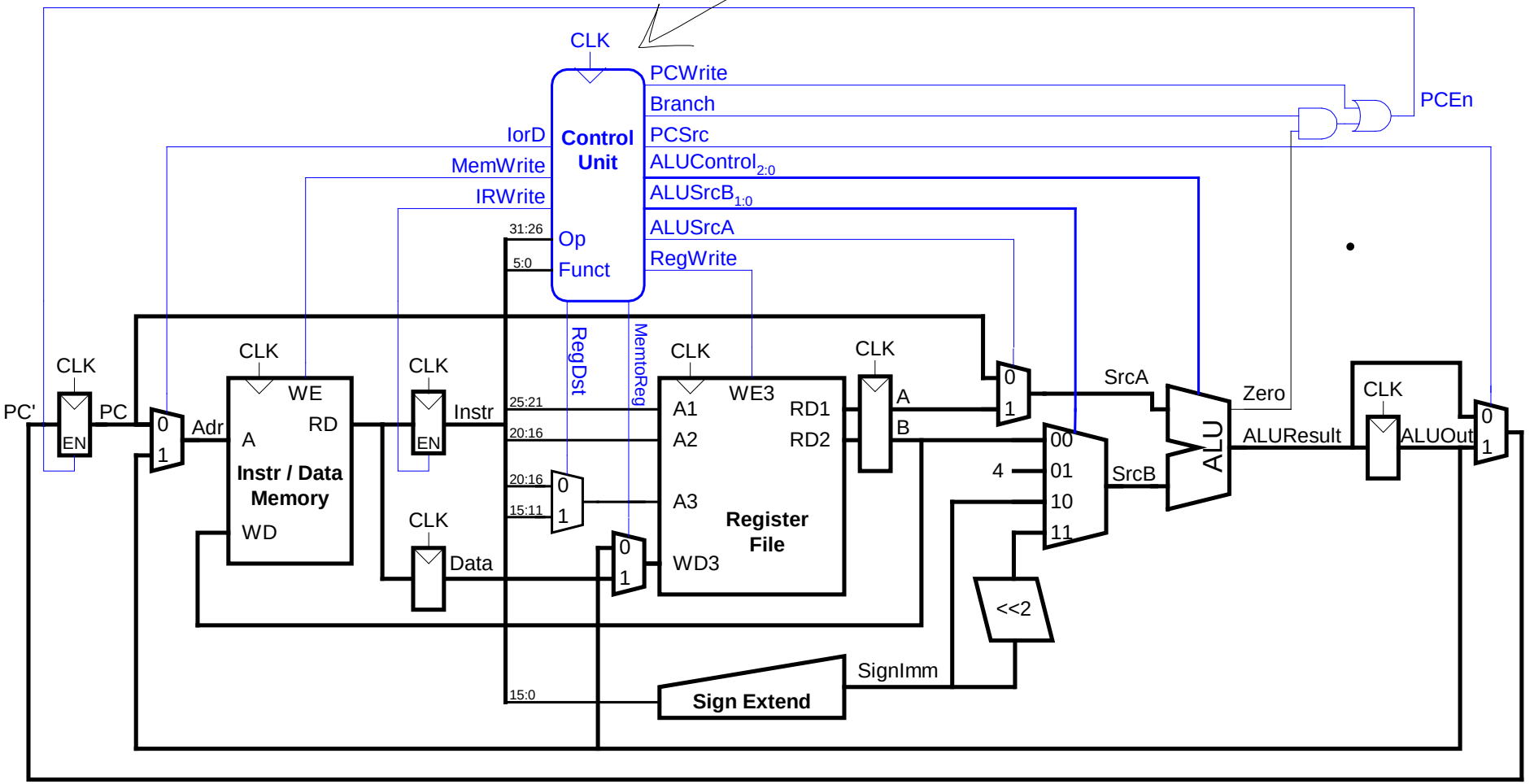
BTA = (vorzeichenerweiterter Direktwert << 2) + (PC+4)



Vollständiger Mehrtaktprozessor

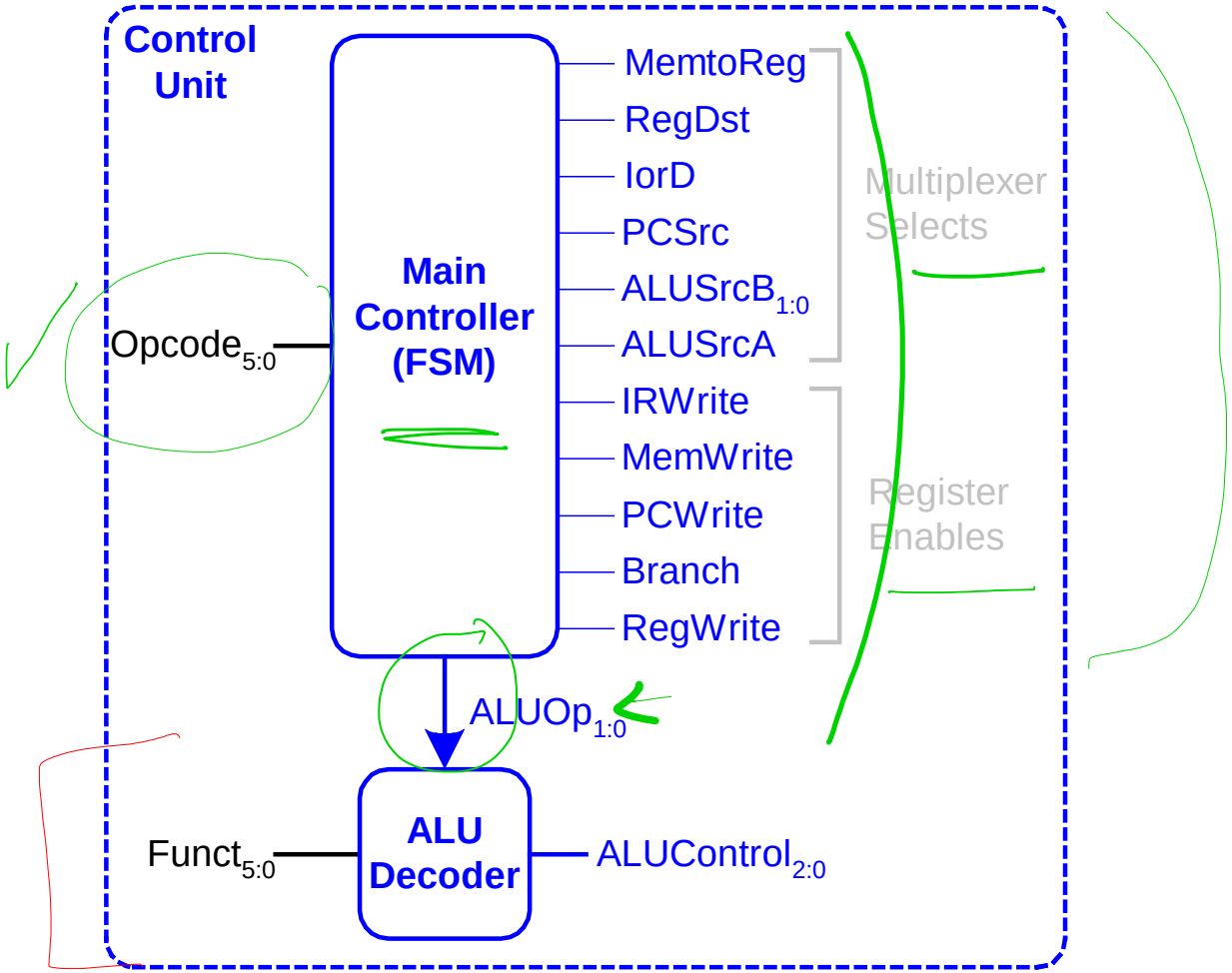
nächstes Mal

TECHNISCHE UNIVERSITÄT DARMSTADT



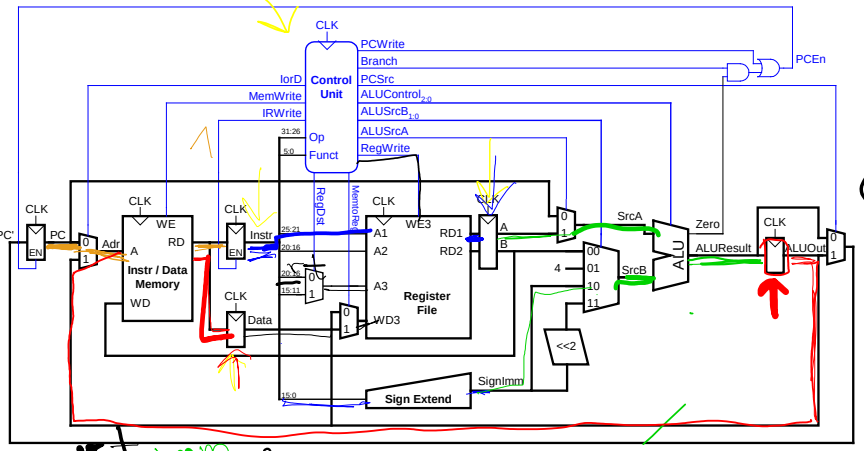
Embedded Systems & Applications

Steuerwerk


Control Unit
Main Controller (FSM)
Opcode5:0
MemtoReg
RegDst
IorD
PCSrc
ALUSrcB1:0
ALUSrcA
IRWrite
MemWrite
PCWrite
Branch
RegWrite
Multiplexer Selects
Register Enables
ALUOp1:0
ALU Decoder
Funct5:0
ALUControl2:0

Steuerwerk: Main Controller
Endlicher Zustandsautomat (FSM)

TECHNISCHE UNIVERSITÄT DARMSTADT



lw $s1, 5($t1)

1. Instr. holen
2. Register holen
3. Adresse berechnen
4. Daten holen
5. Daten ins Register schreiben

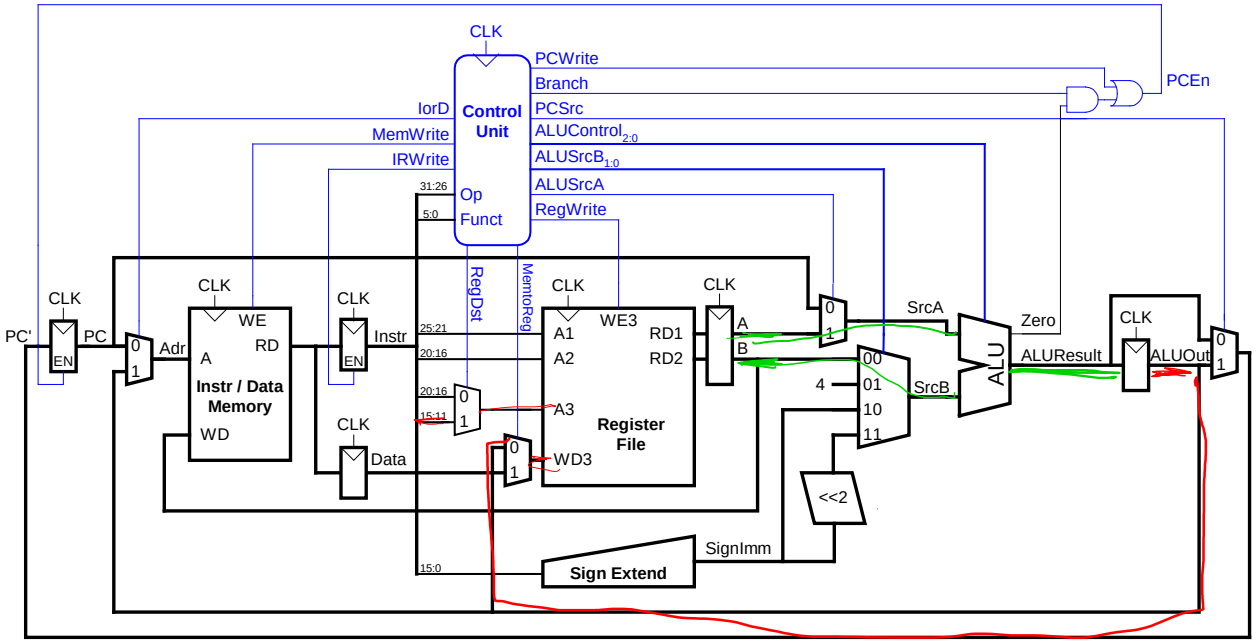
Embedded Systems & Applications

sw $s7, 16($t0)
rt imm rs

2 3 4 Daten schreiben

Vollständiger Mehrtaktprozessor

TECHNISCHE UNIVERSITÄT DARMSTADT



R-Typ

1

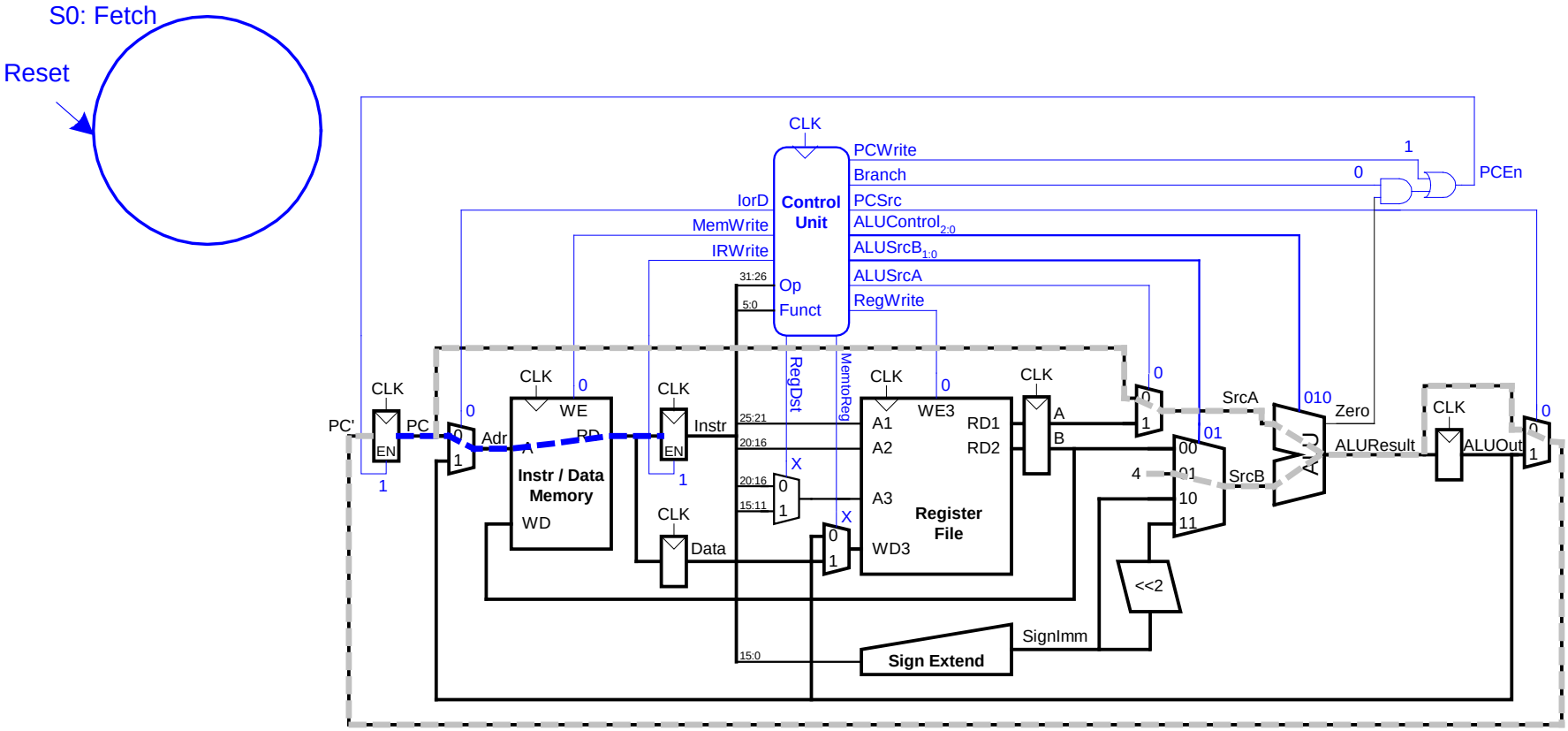
2

3 Operation ausführen

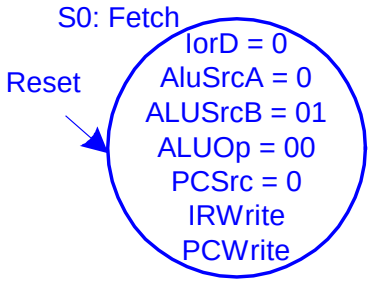
4 Das Ergebnis ins Register Feld schreiben

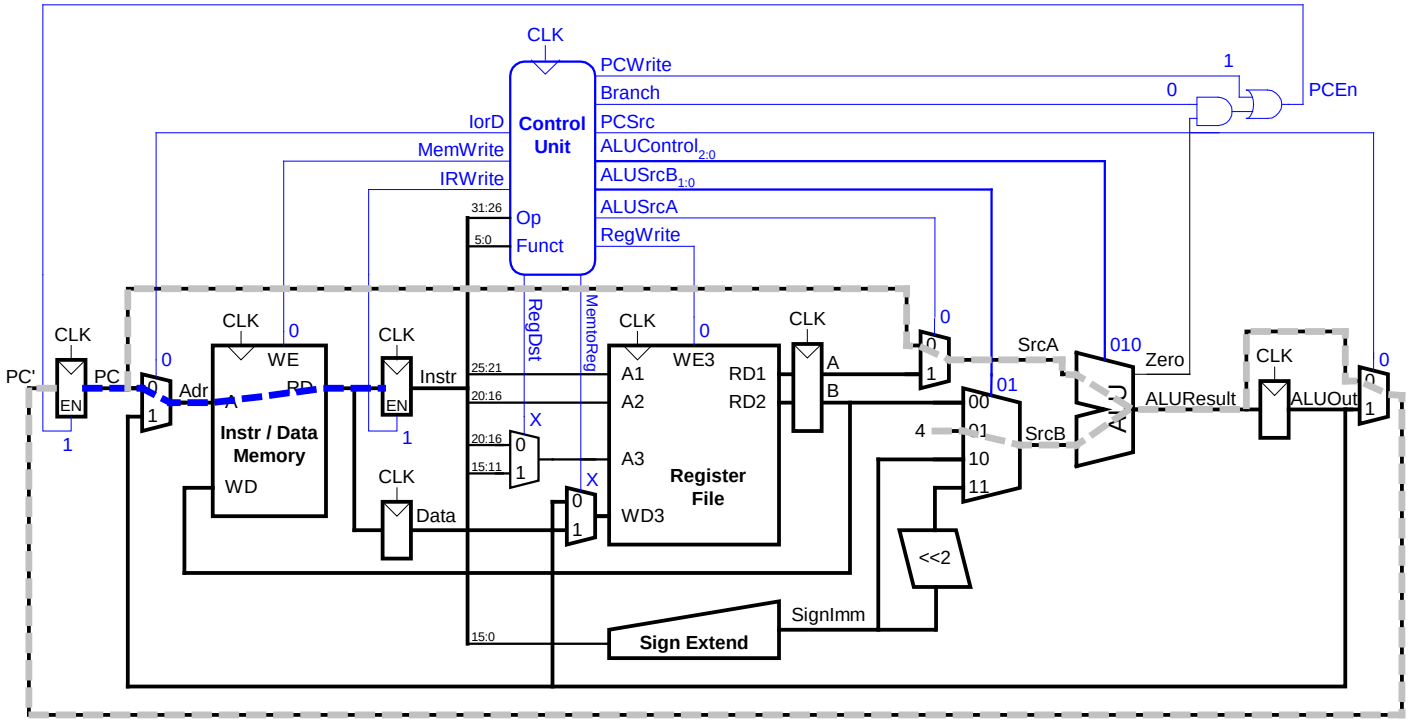
Embedded Systems & Applications

Hauptsteuerwerk: Holen eines Befehls

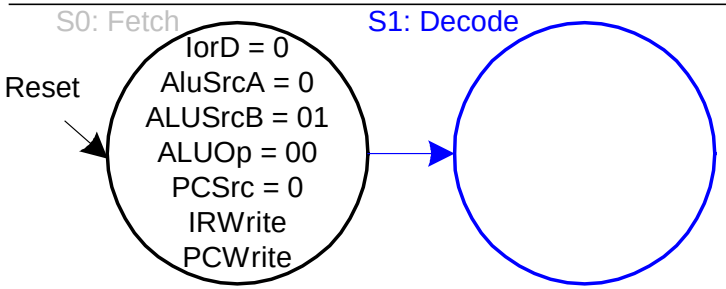


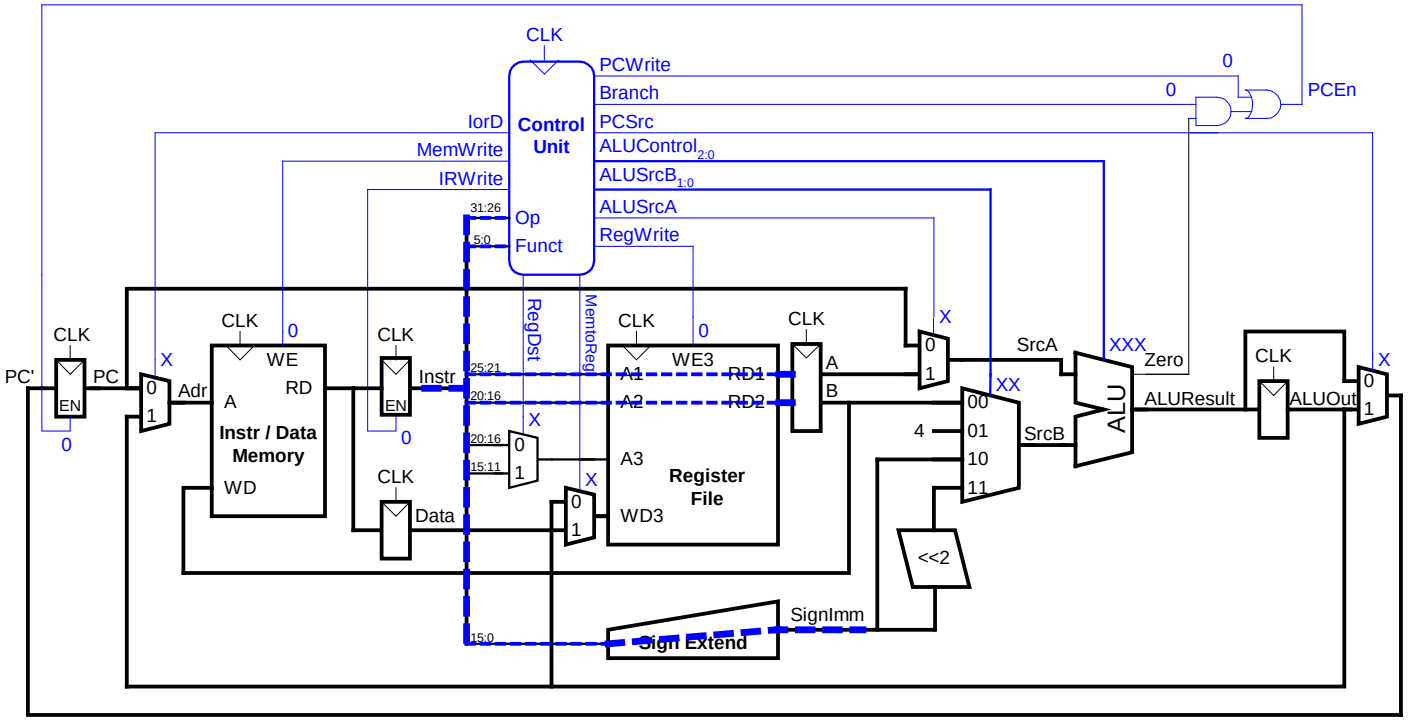
Hauptsteuerwerk: Holen eines Befehls



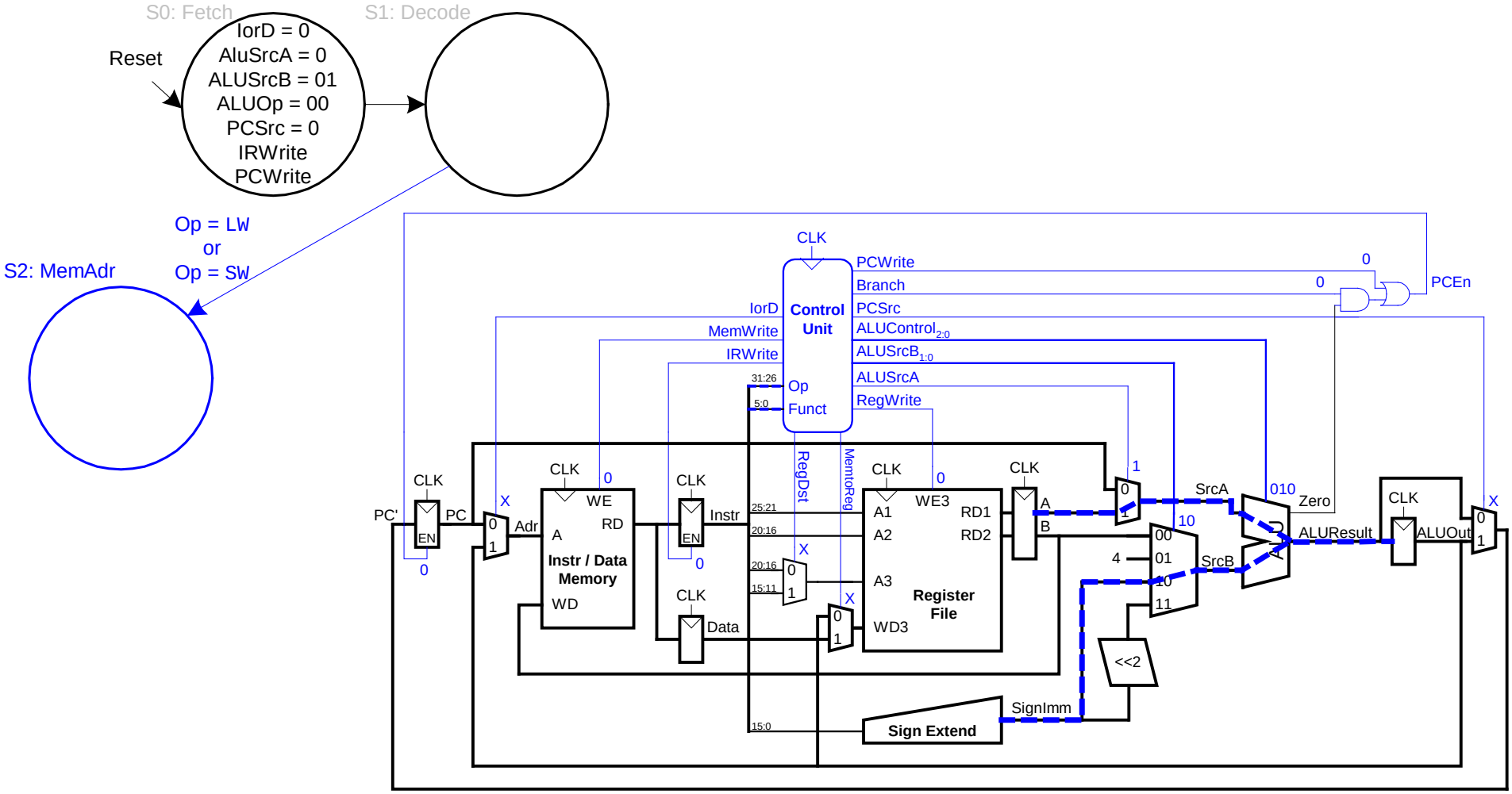


Hauptsteuerwerk: Dekodieren eines Befehls

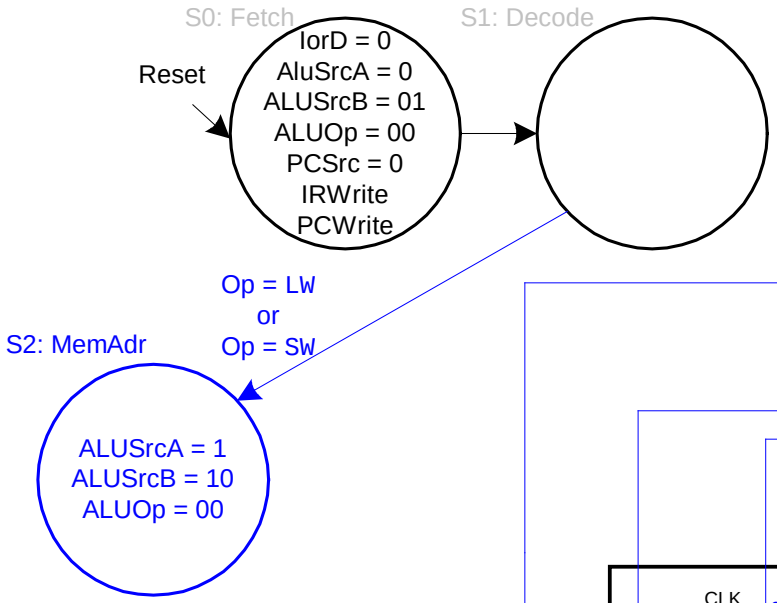


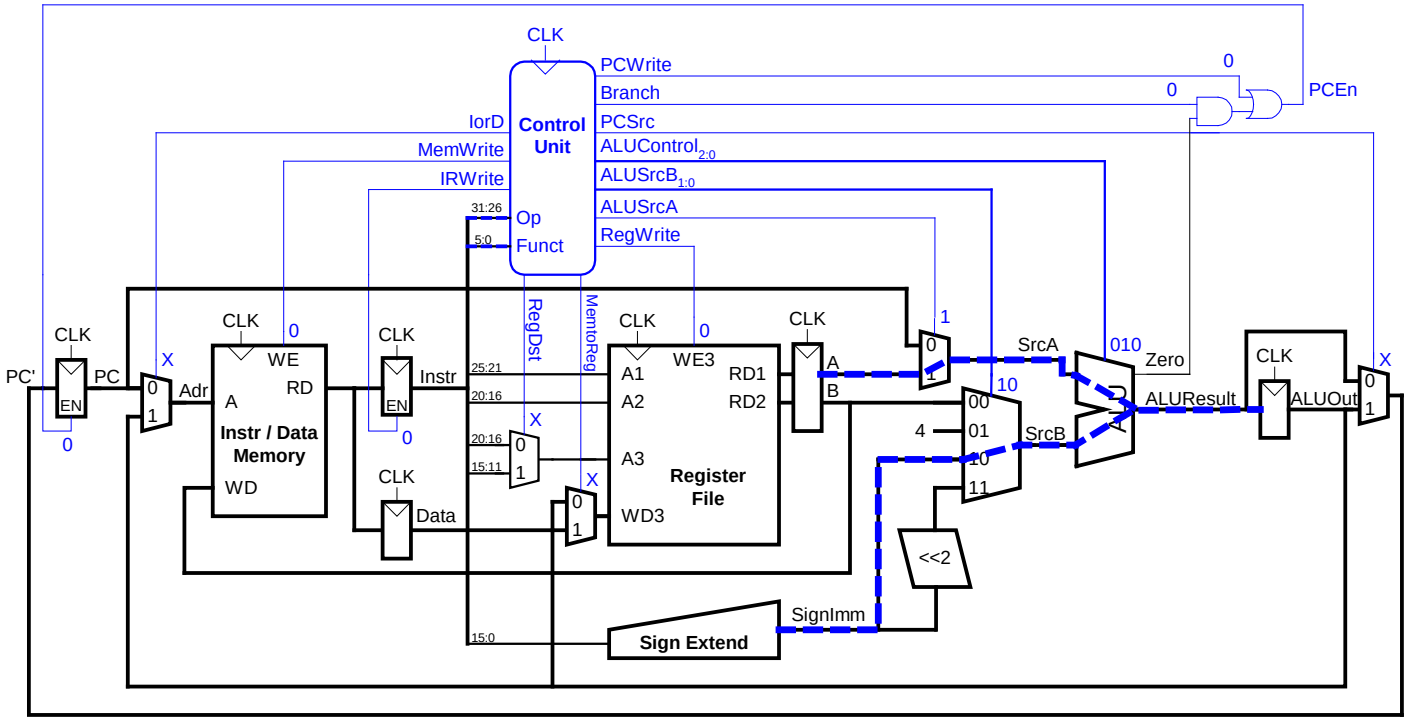


Hauptsteuerwerk: Adressberechnung

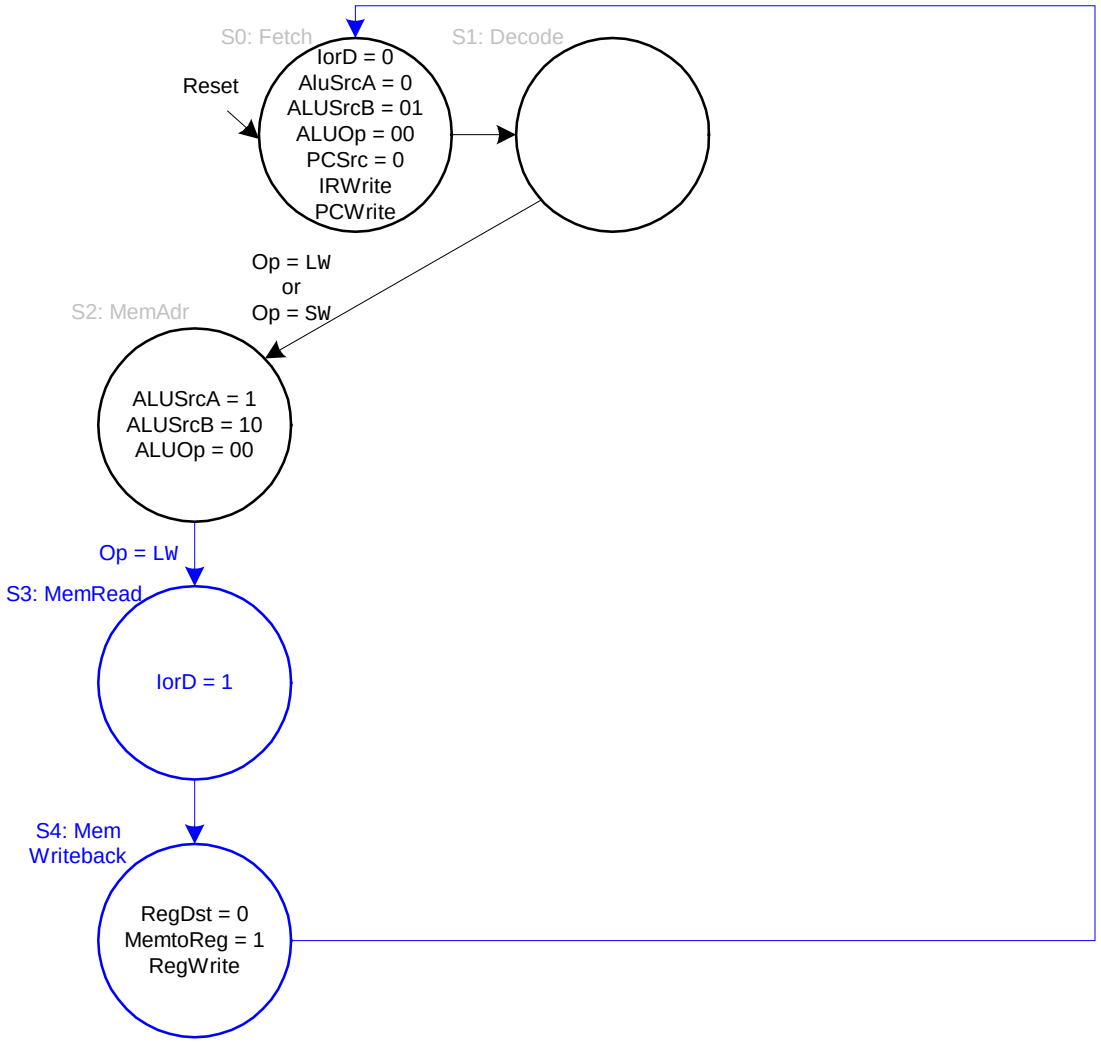


Hauptsteuerwerk: Adressberechnung

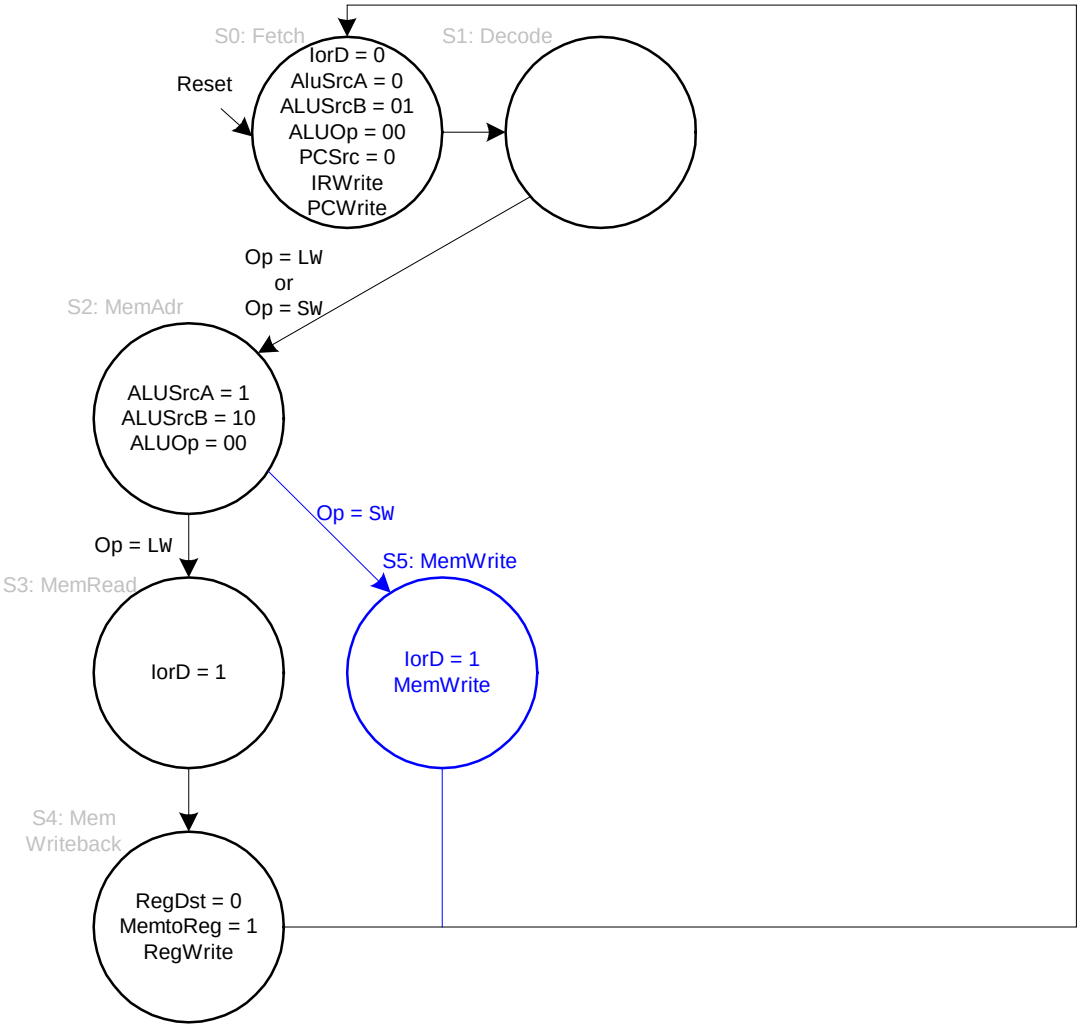




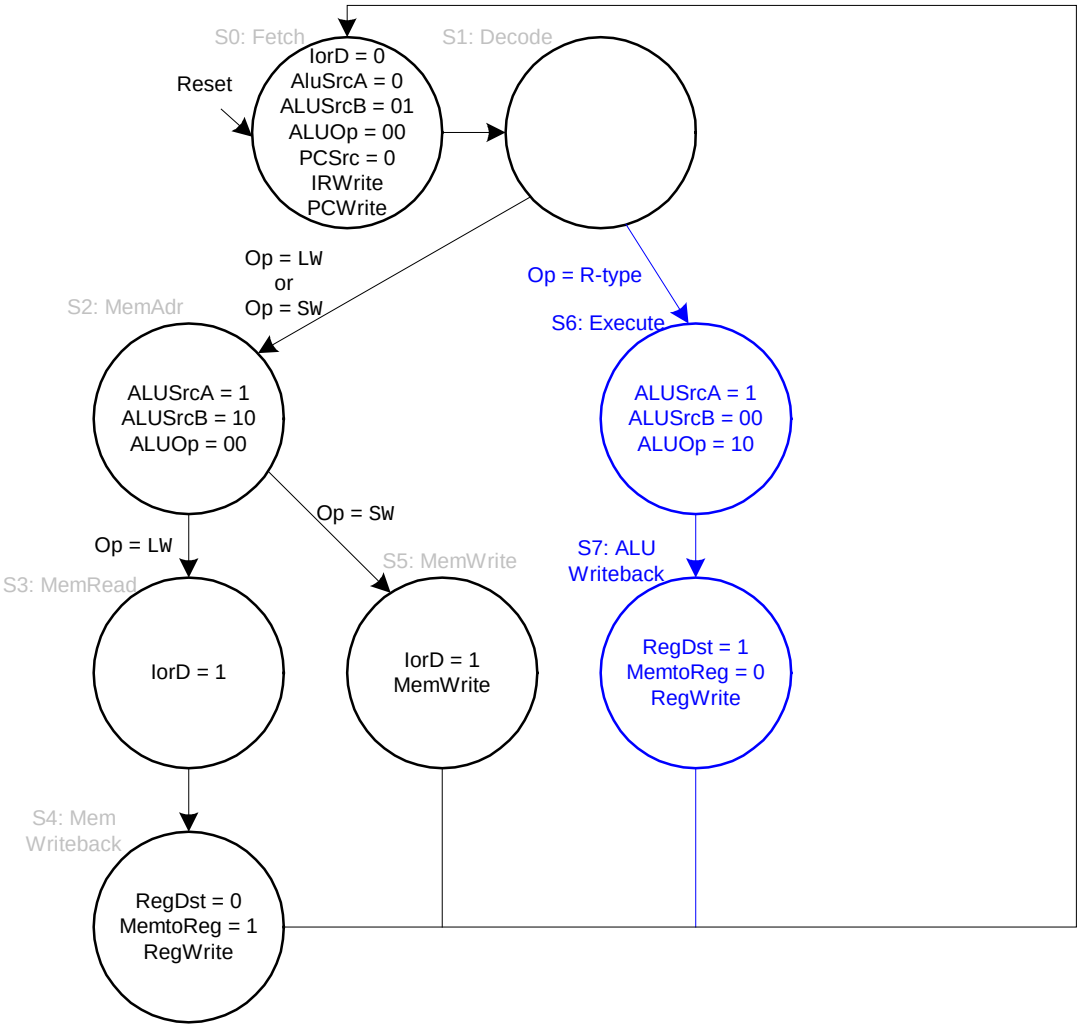
Hauptsteuerwerk: FSM für `lw`



Hauptsteuerwerk: FSM für sw

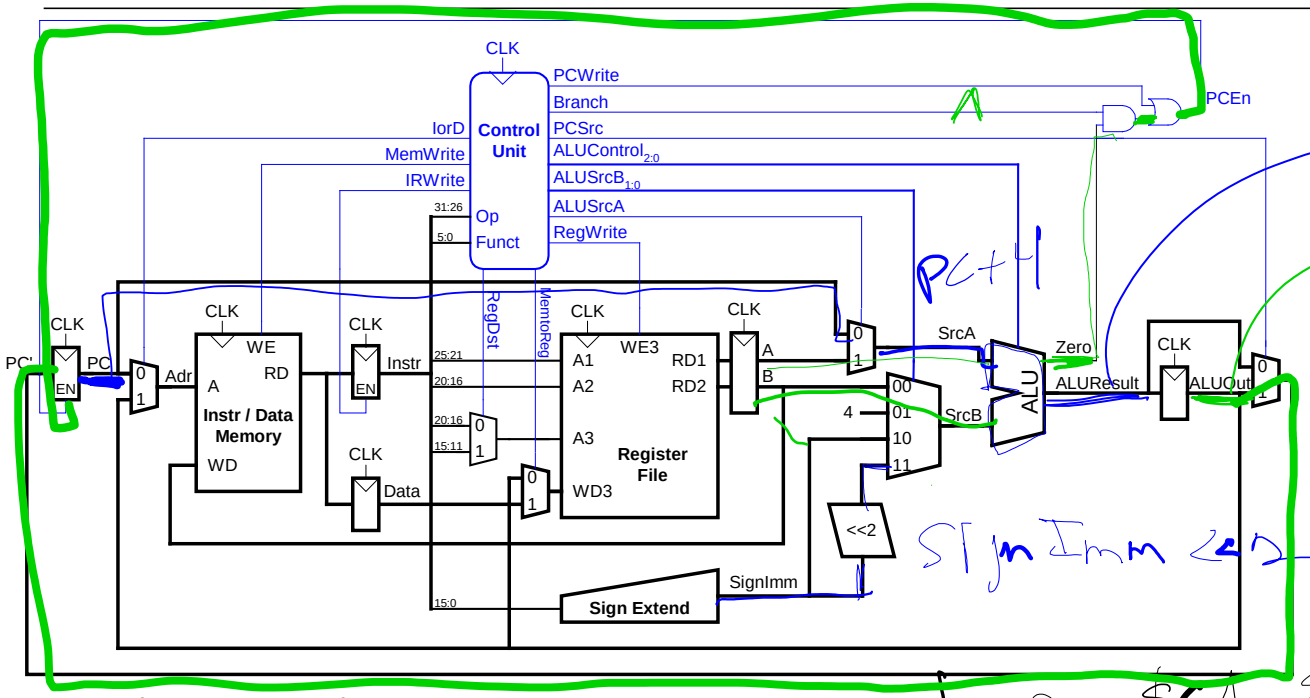


Hauptsteuerwerk: FSM für R-Typ



Steuerwerk: Main Controller
Endlicher Zustandsautomat (FSM)

TECHNISCHE UNIVERSITÄT DARMSTADT



PC+4

Sprungadresse

SignImm ×4

lw $s1, 5($t1)

1. Instr. holen
2. Register holen
3. Adresse berechnen
4. Daten holen
5. Daten ins Register schreiben

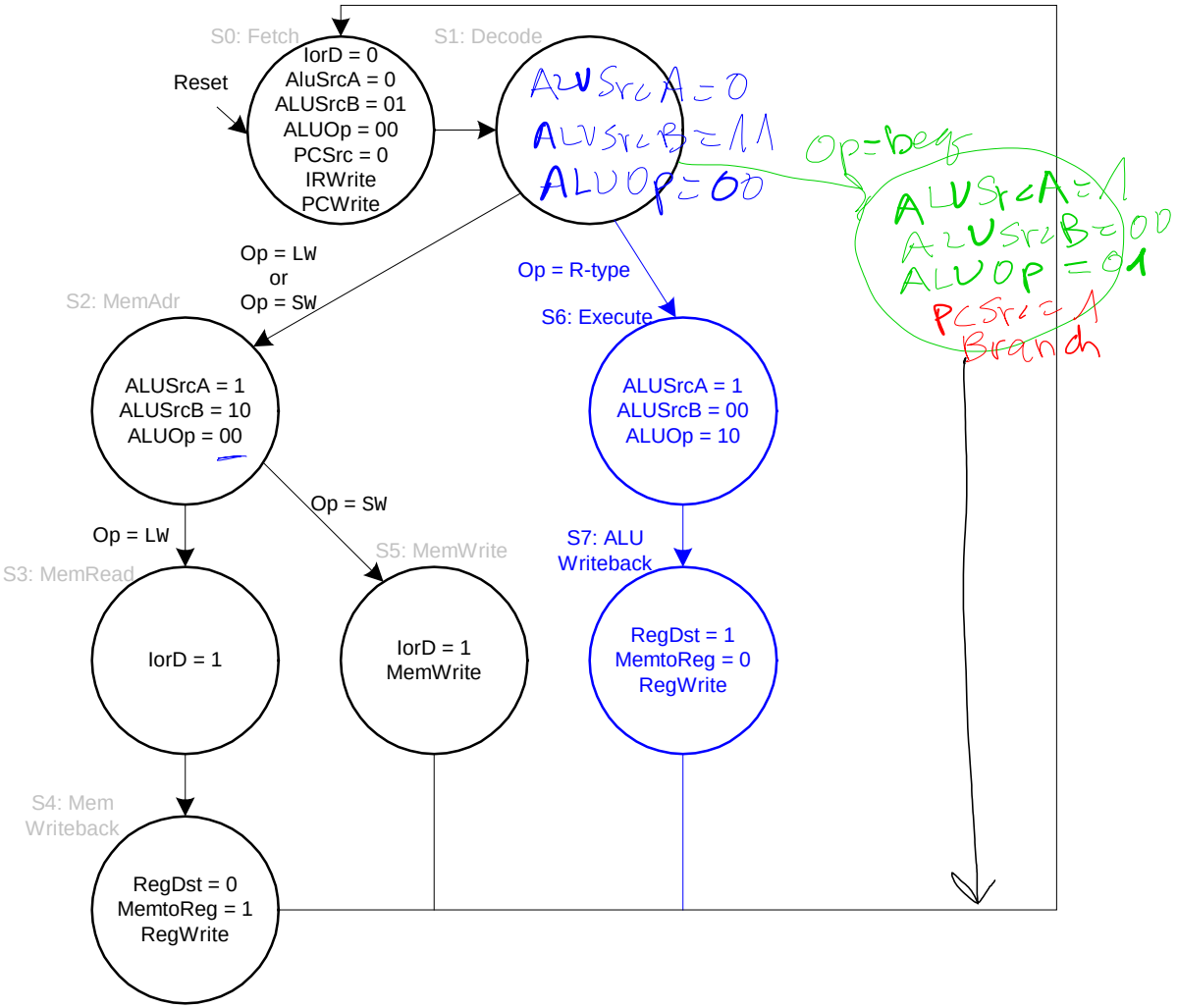
beq $s1, $t7, Loop

1 → 2 → Sprungadresse bestimmen

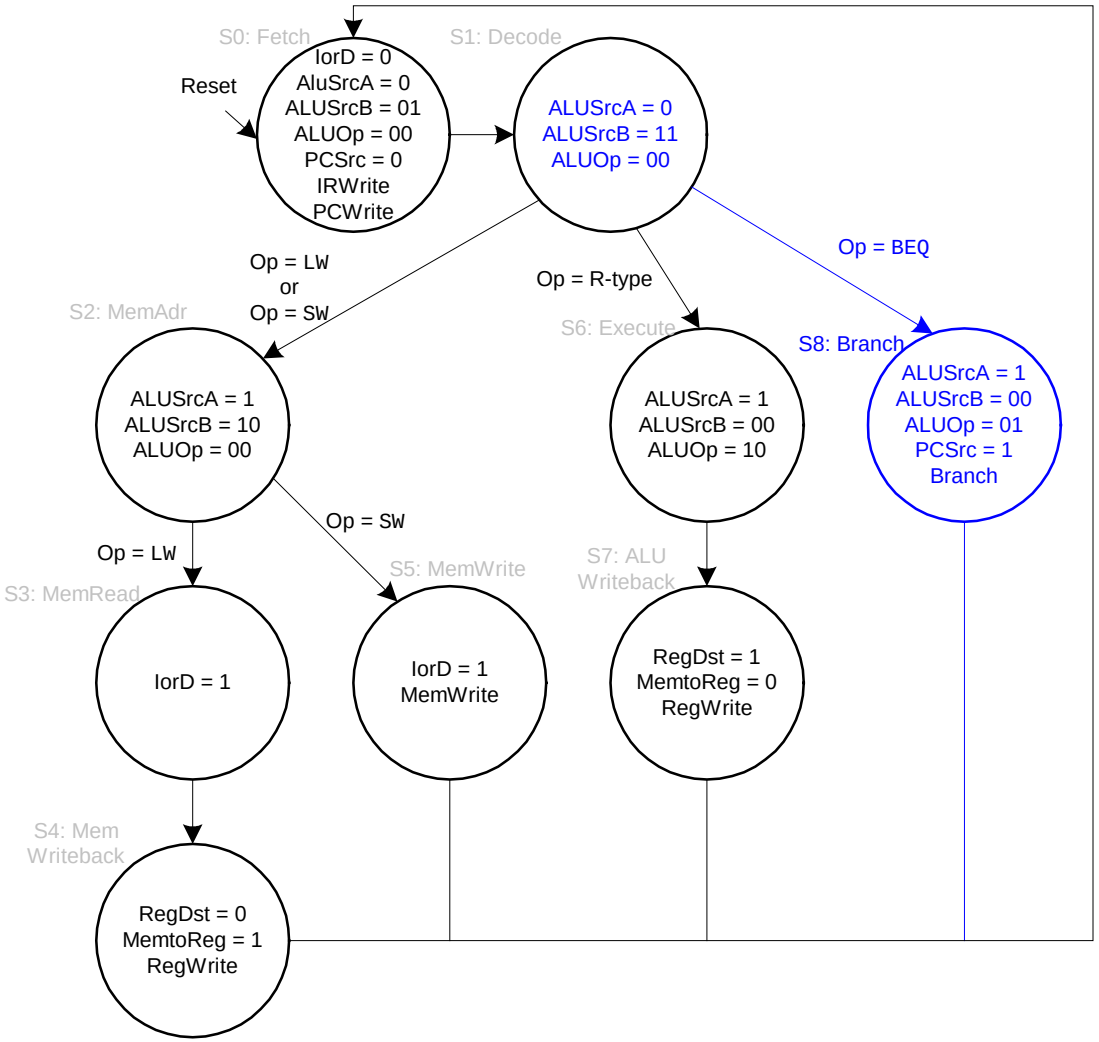
3 Vergleich

Embedded Systems & Applications

Hauptsteuerwerk: FSM für R-Typ

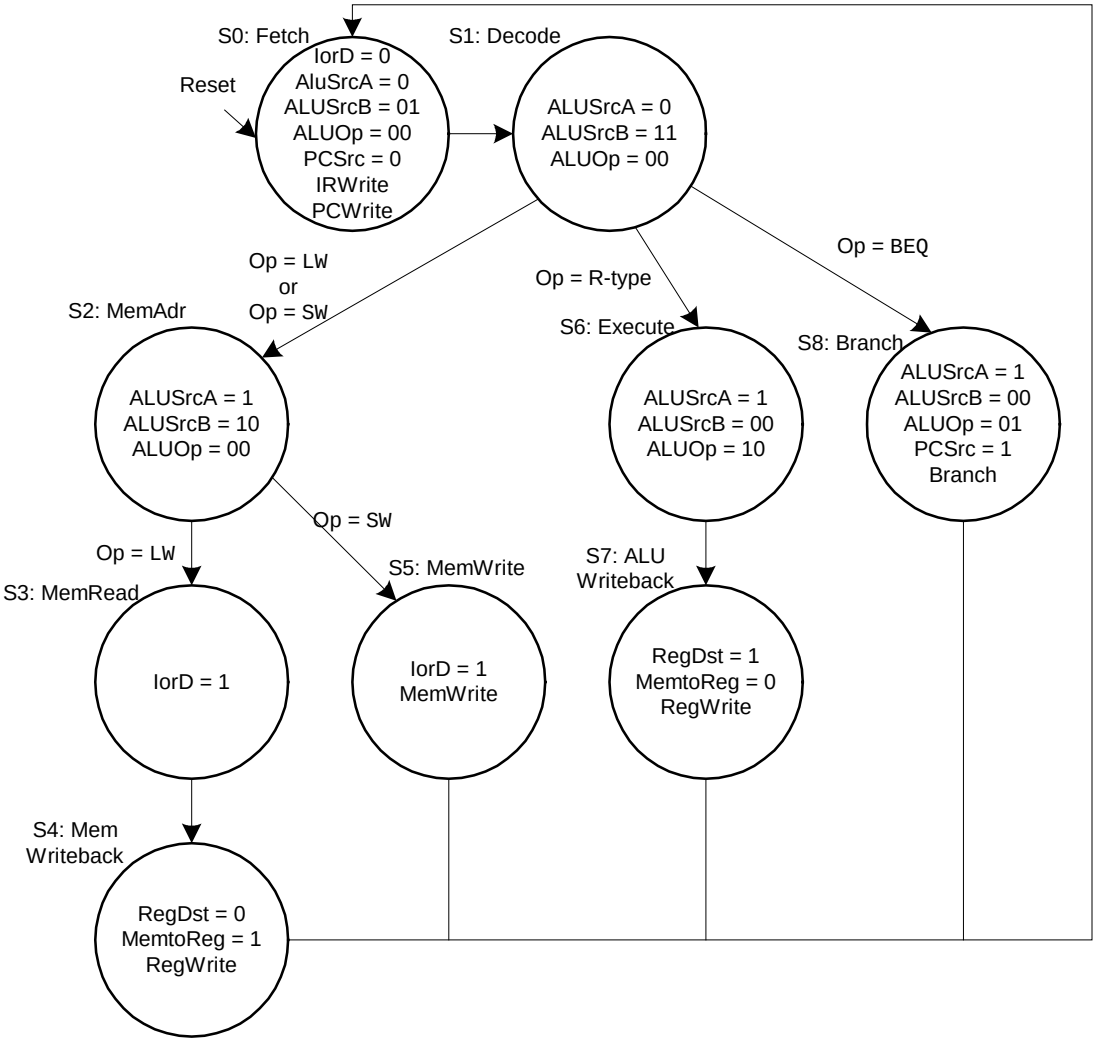


Hauptsteuerwerk: FSM für beq



Vollständiges Hauptsteuerwerk für Mehrtakt-CPU

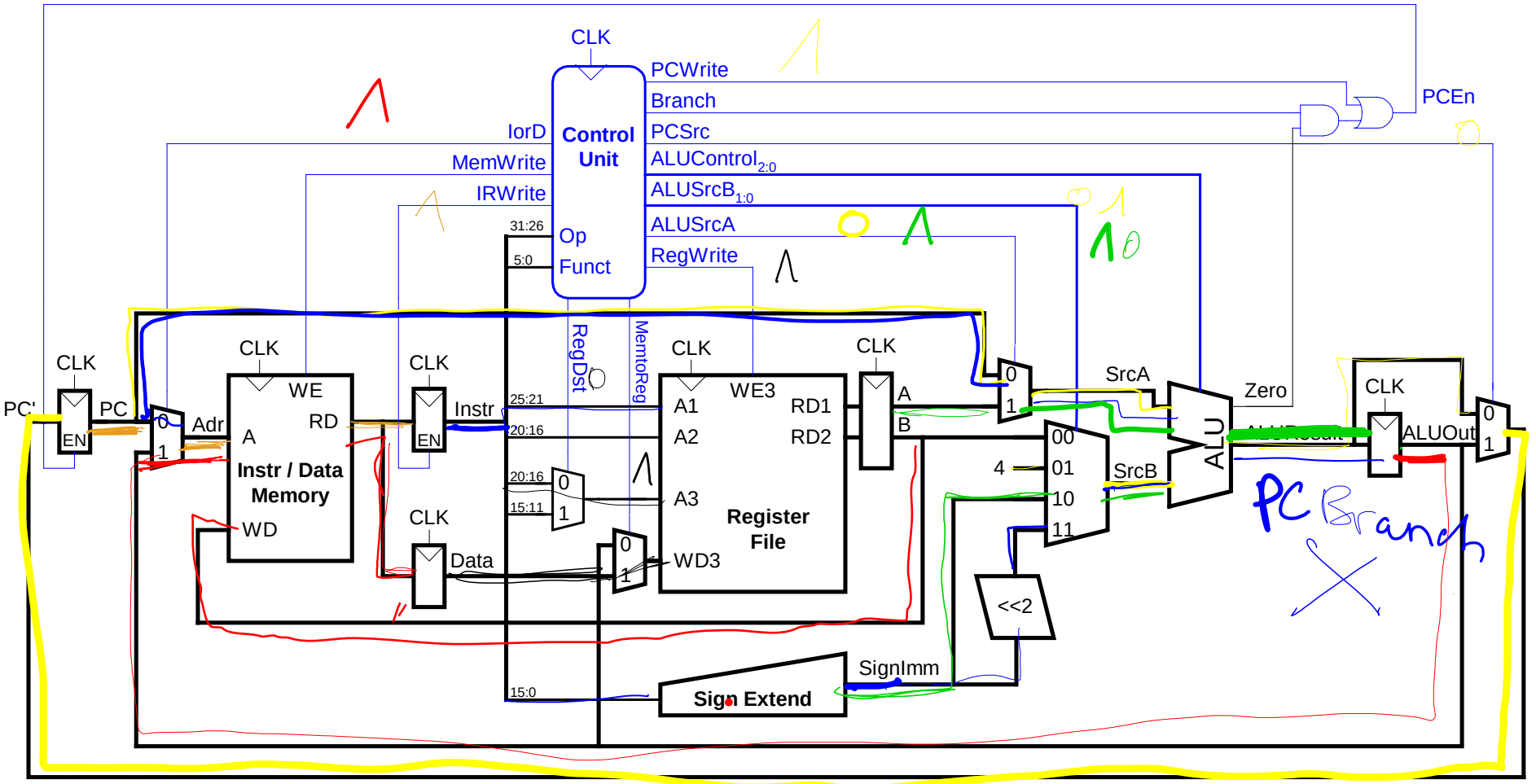
- Die Signale die **nicht** im Kreis stehen sind 0.
- Die Signale die im Kreis stehen ohne Wert sind 1 (Enable Signale).



Vollständiger Mehrtaktprozessor

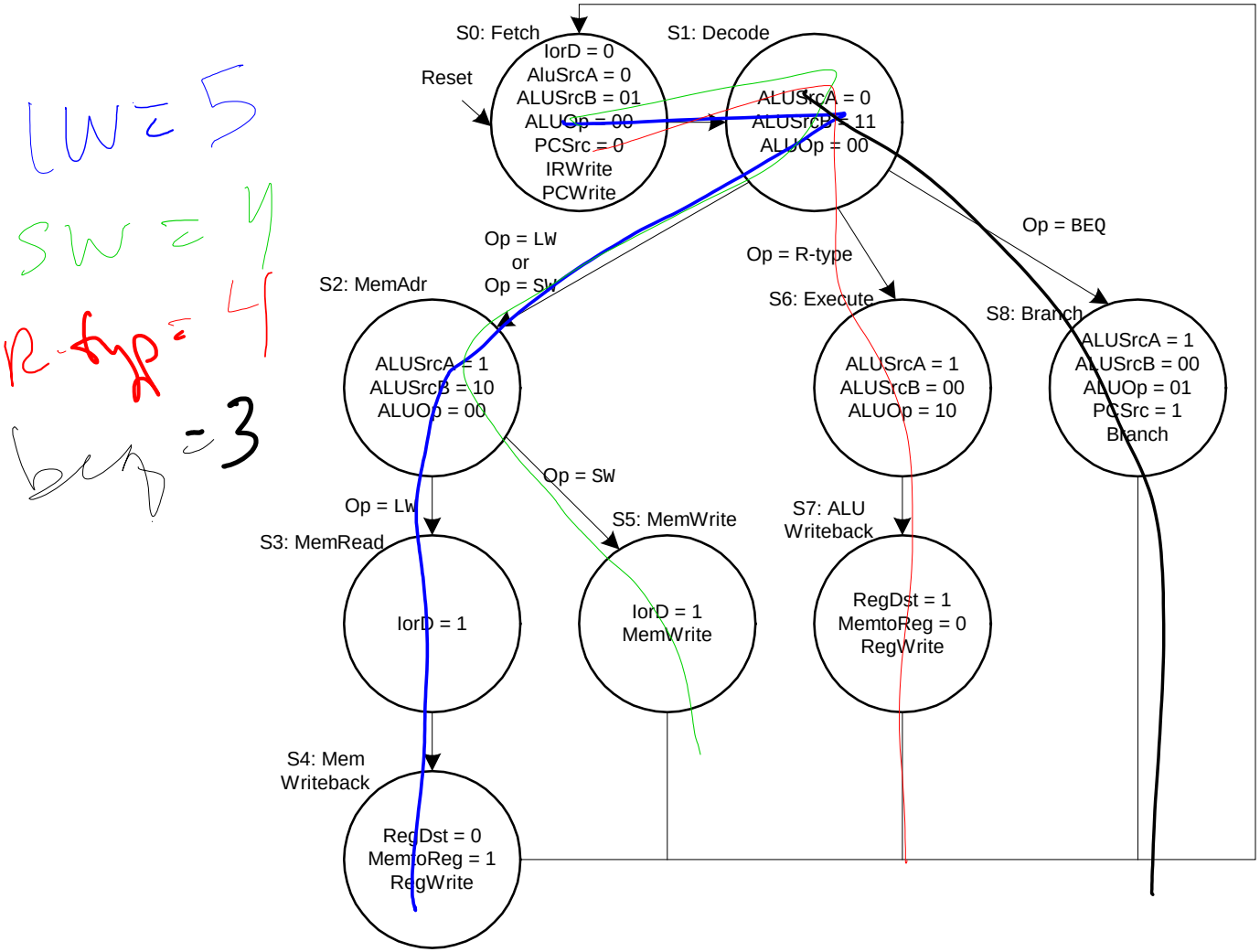
SW Befehl

TECHNISCHE UNIVERSITÄT DARMSTADT



Embedded Systems & Applications

Vollständiges Hauptsteuerwerk für Mehrtakt-CPU



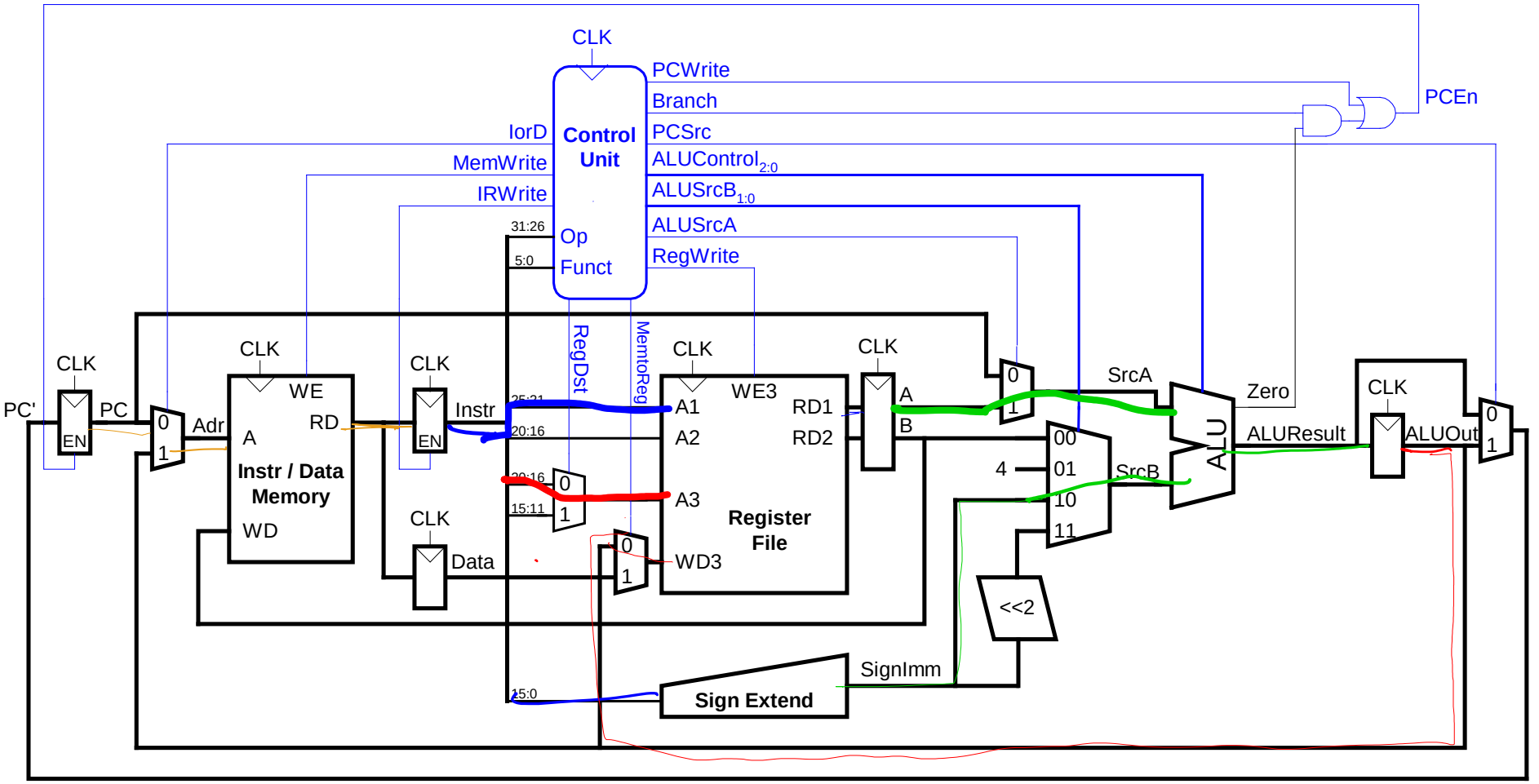
LW = 5

SW = 4

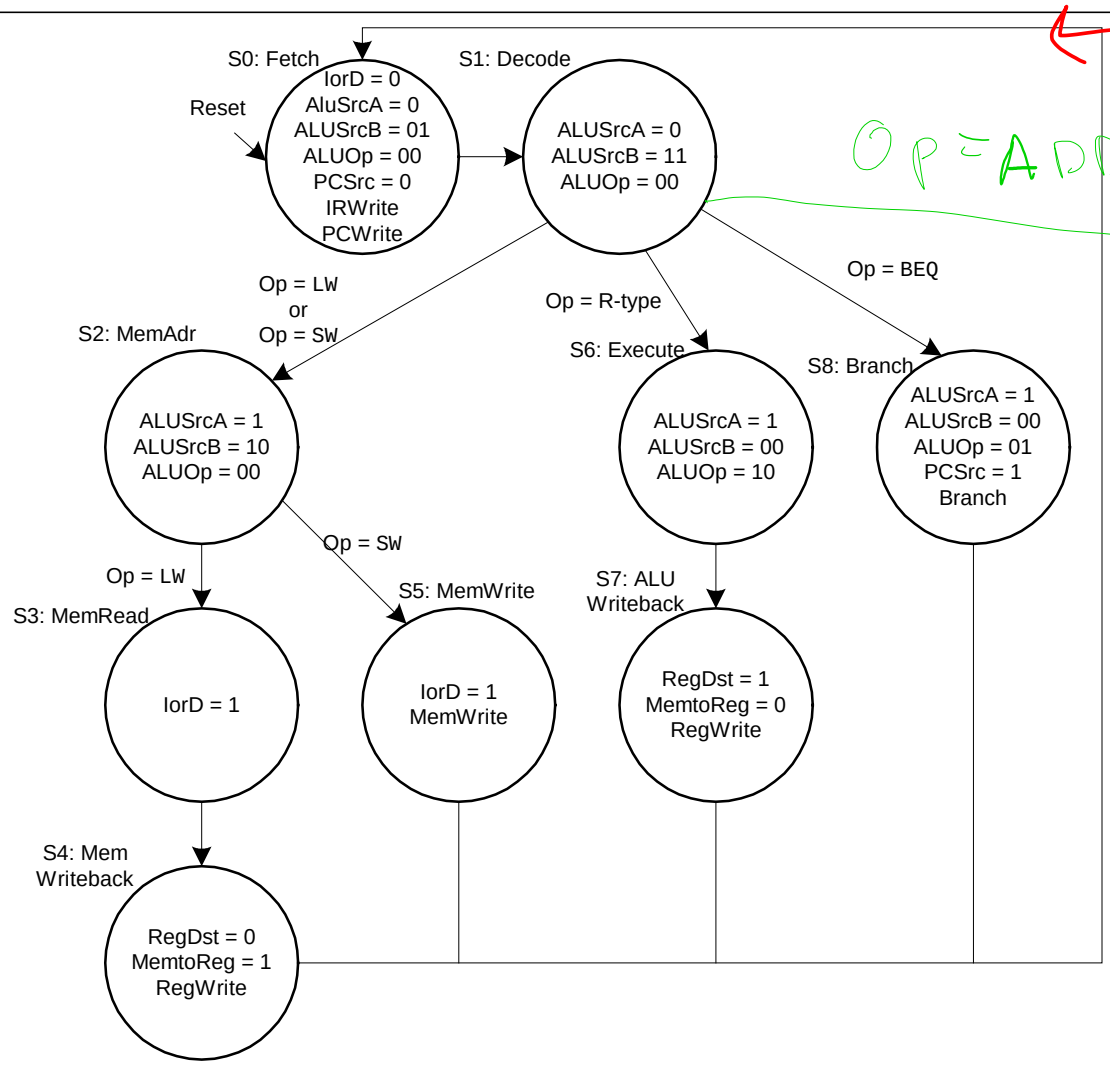
R-typ = 4

beq = 3

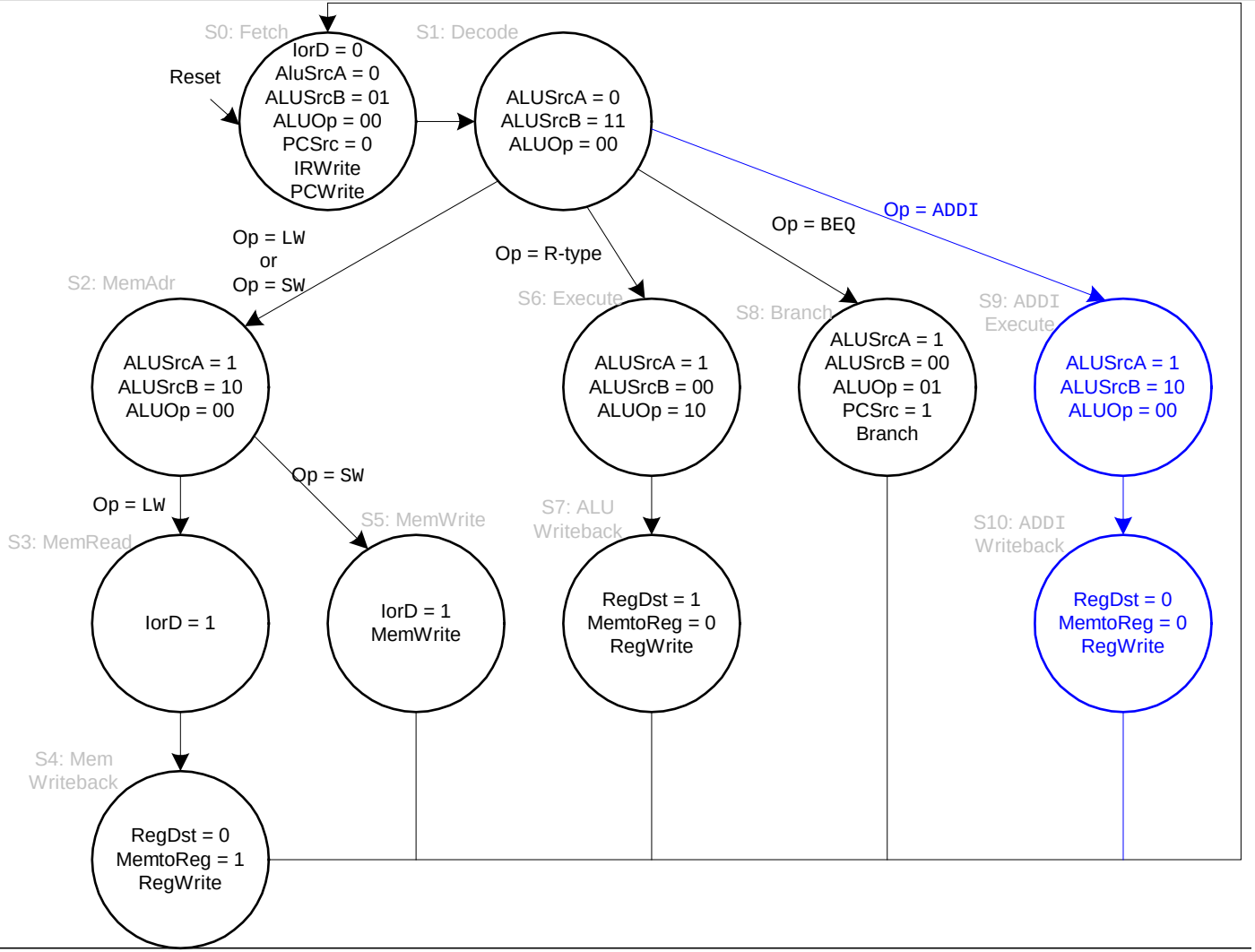
Vollständiger Mehrtaktprozessor: addi



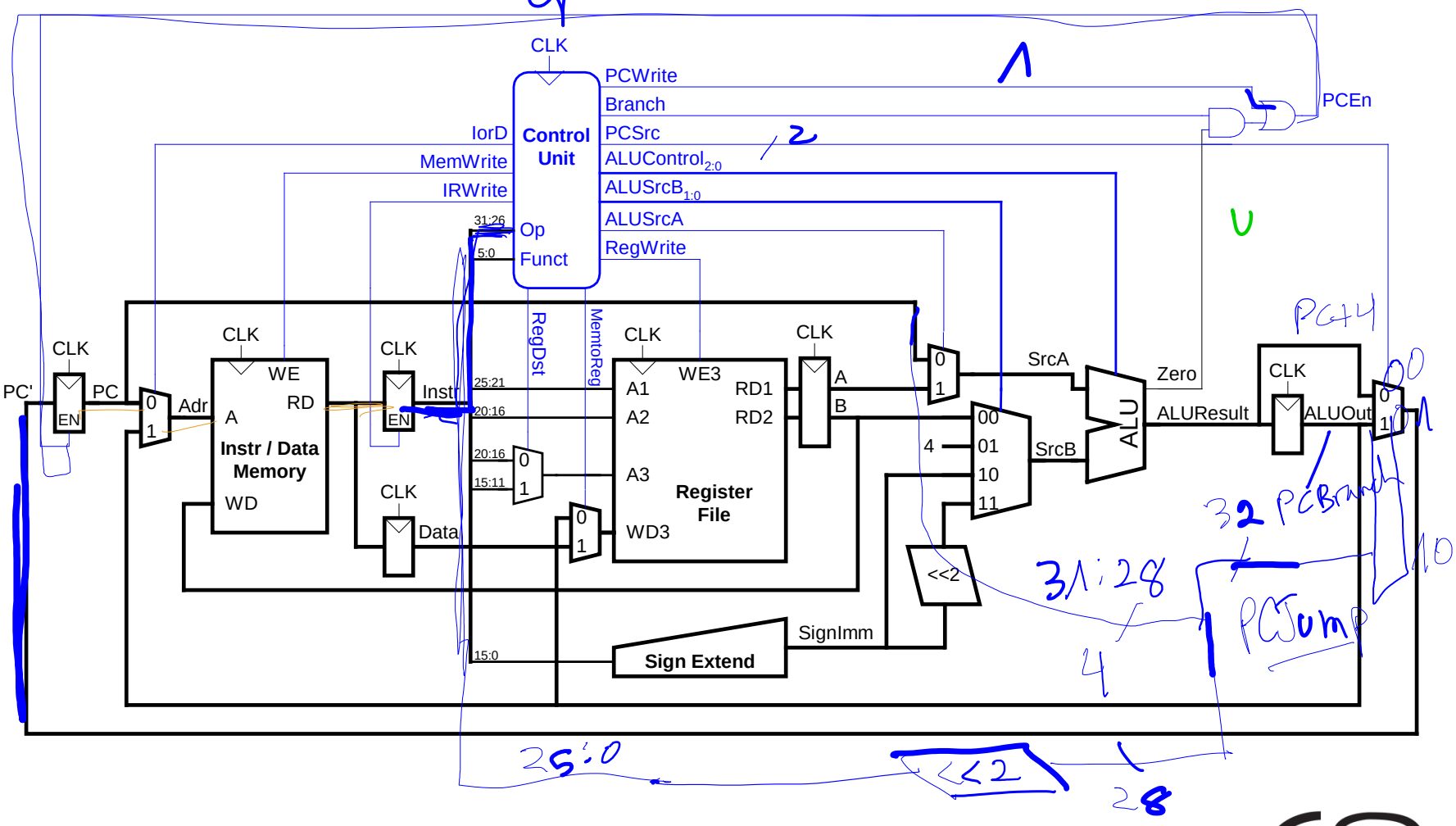
Erweiterung des Hauptsteuerwerks: `addi`



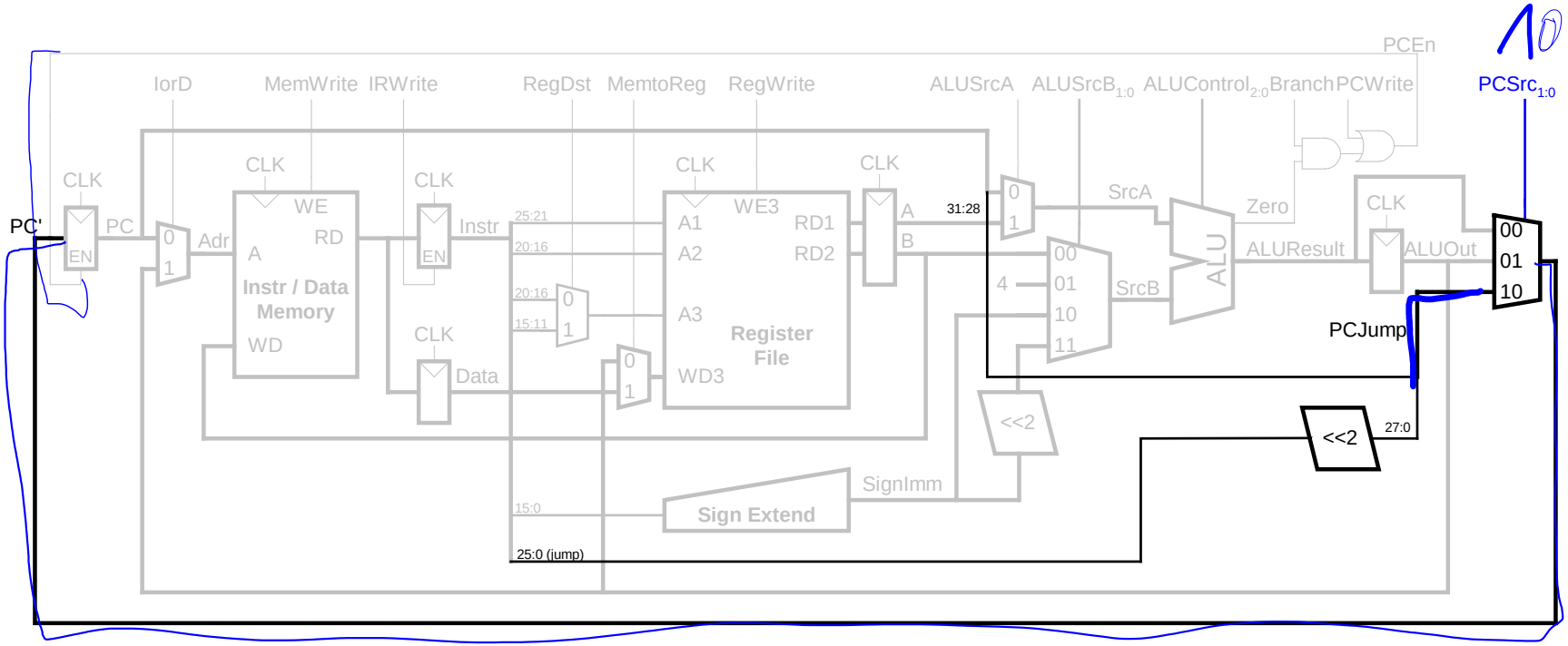
Erweiterung des Hauptsteuerwerks: `addi`



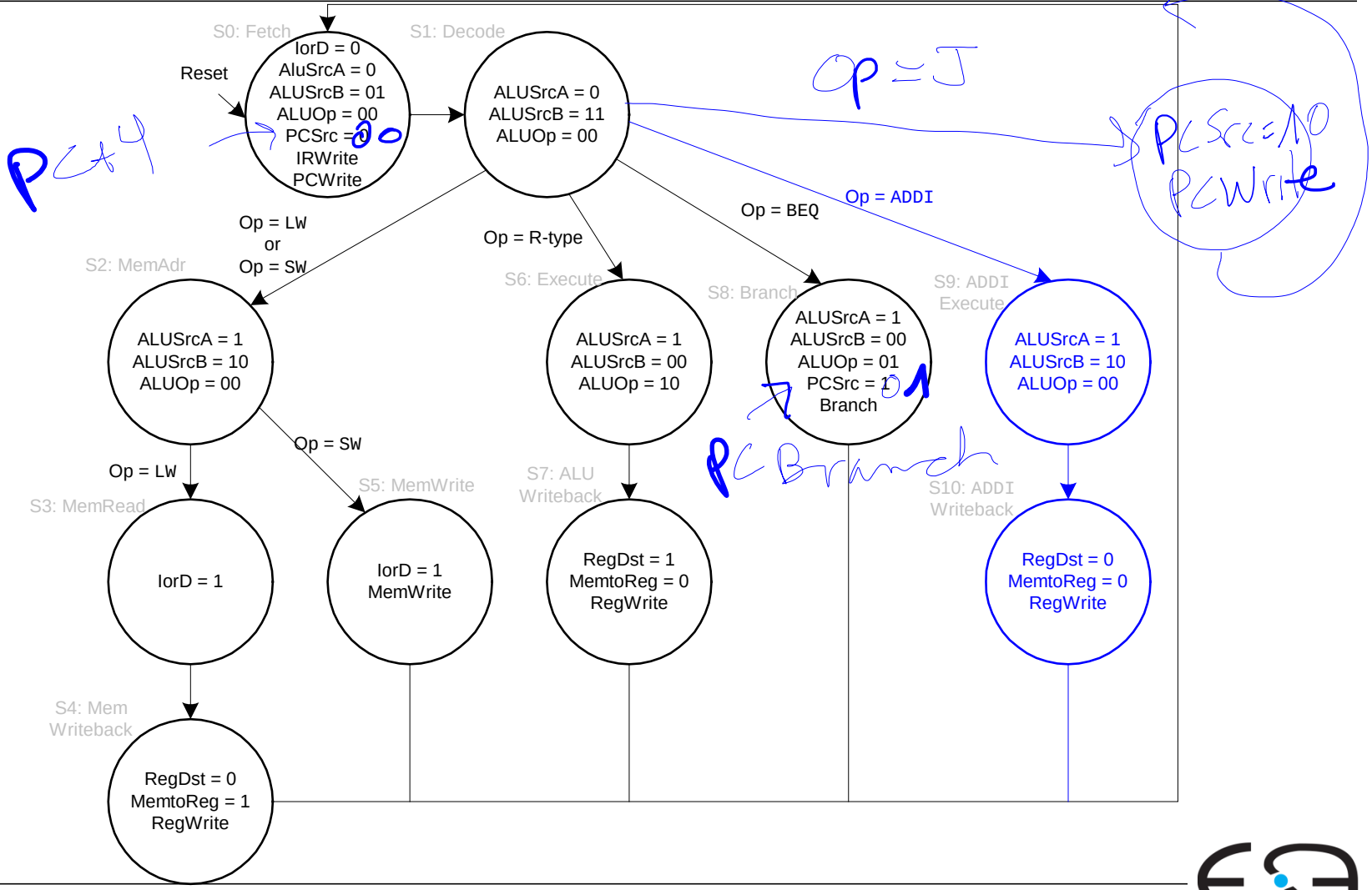
Erweiterung des Datenpfads für j



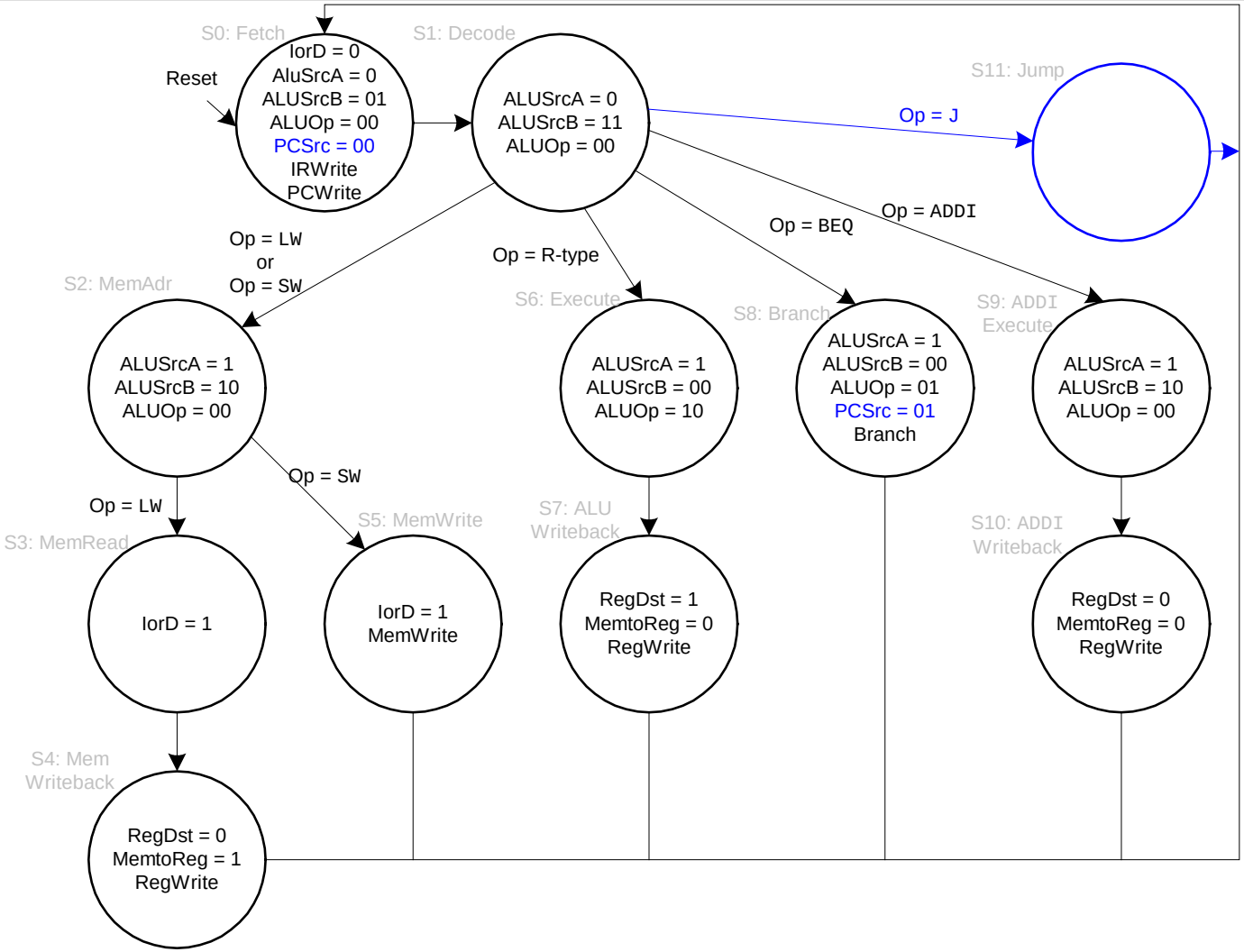
Erweiterung des Datenpfads für `j`



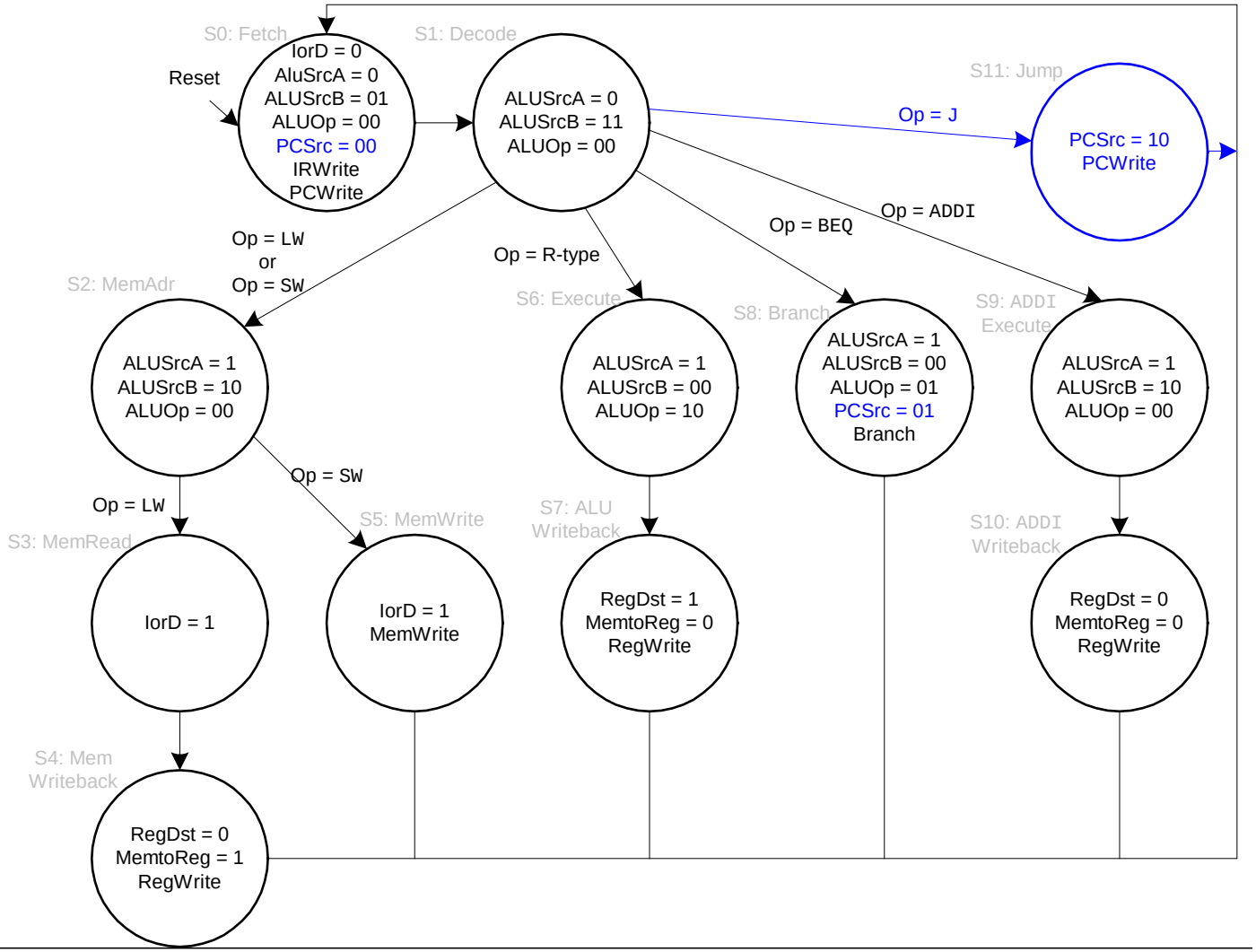
Erweiterung des Hauptsteuerwerks um `j`



Erweiterung des Hauptsteuerwerks um `j`



Erweiterung des Hauptsteuerwerks um `j`



Rechenleistung des Mehrtaktprozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Instruktionen benötigen unterschiedliche viele Takte:

- 3 Takte : `beq`, `j`
- 4 Takte : R-Typ, `sw`, `addi`
- 5 Takte : `lw`

CPI wird bestimmt als gewichteter Durchschnitt

SPECint 2000 Benchmark:

- 25% Laden
- 10% Speichern
- 11% Verzweigungen
- 2% Sprünge
- 52% R-Typ

Durchschnittliche CPI =

Embedded Systems & Applications

Rechenleistung des Mehrtaktprozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Instruktionen benötigen unterschiedliche viele Takte:

- 3 Takte : `beq`, `j`
- 4 Takte : R-Typ, `sw`, `addi`
- 5 Takte : `lw`

CPI wird bestimmt als gewichteter Durchschnitt

SPECint 2000 Benchmark:

- 25% Laden
- 10% Speichern
- 11% Verzweigungen
- 2% Sprünge
- 52% R-Typ

Durchschnittliche CPI = (0,11 + 0,02)(3) + (0,52 + 0,10)(4) + (0,25)(5) = 4,12

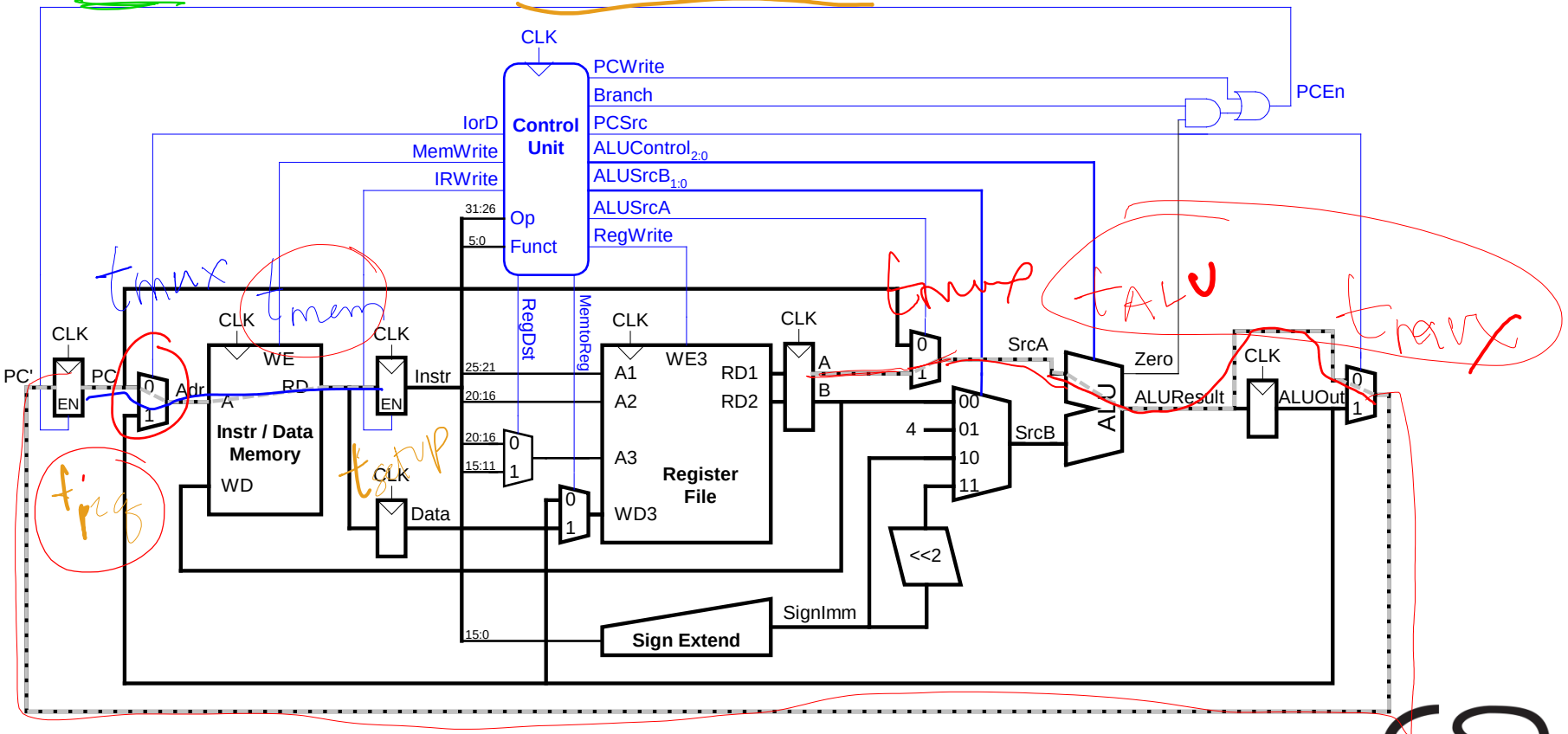
Embedded Systems & Applications

Rechenleistung des Mehrtaktprozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Kritischer Pfad :

$$T_c = t_{pcq} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup}$$



Embedded Systems & Applications

Beispiel: Rechenleistung Mehrtaktprozessor

TECHNISCHE UNIVERSITÄT DARMSTADT

Element	Parameter	Verzögerung (ps)
Register Clock-to-Q	t_{pcq}	30
Register Setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Speicher Lesen	t_{mem}	250
Registerfeld Lesen	t_{RFread}	150
Registerfeld Setup	$t_{RFsetup}$	20

T_c =

Embedded Systems & Applications

Beispiel: Rechenleistung Mehrtaktprozessor

TECHNISCHE UNIVERSITÄT DARMSTADT

Element	Parameter	Verzögerung (ps)
Register Clock-to-Q	t_{pcq}	30
Register Setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Speicher Lesen	t_{mem}	250
Registerfeld Lesen	t_{RFread}	150
Registerfeld Setup	$t_{RFsetup}$	20

$$T_c = t_{pcq\_PC} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup}$$
$$= t_{pcq\_PC} + t_{mux} + t_{mem} + t_{setup}$$
$$= [30 + 25 + 250 + 20] \text{ ps}$$
$$= 325 \text{ ps}$$

Embedded Systems & Applications

Beispiel: Rechenleistung Mehrtaktprozessor

TECHNISCHE UNIVERSITÄT DARMSTADT

Führe Programm mit 100 Milliarden Instruktionen auf Mehrtaktprozessor aus

- CPI = 4,12
- T_c = 325 ps

Embedded Systems & Applications

Beispiel: Rechenleistung Mehrtaktprozessor

TECHNISCHE UNIVERSITÄT DARMSTADT

Führe Programm mit 100 Milliarden Instruktionen auf Mehrtaktprozessor aus

- CPI = 4,12
- T_c = 325 ps

Ausführungszeit = (# Instruktionen) × CPI × T_c

= (100×10^9) (4,12) (325×10^{-12})

= 133,9 Sekunden

- **Langsamer als Ein-Takt-Prozessor (brauchte 92,5 Sekunden).**

Embedded Systems & Applications

Beispiel: Rechenleistung Mehrtaktprozessor

TECHNISCHE UNIVERSITÄT DARMSTADT

Führe Programm mit 100 Milliarden Instruktionen auf Mehrtaktprozessor aus

- CPI = 4,12
- T_c = 325 ps

Ausführungszeit = (# Instruktionen) × CPI × T_c

= (100×10^9) (4,12) (325×10^{-12})

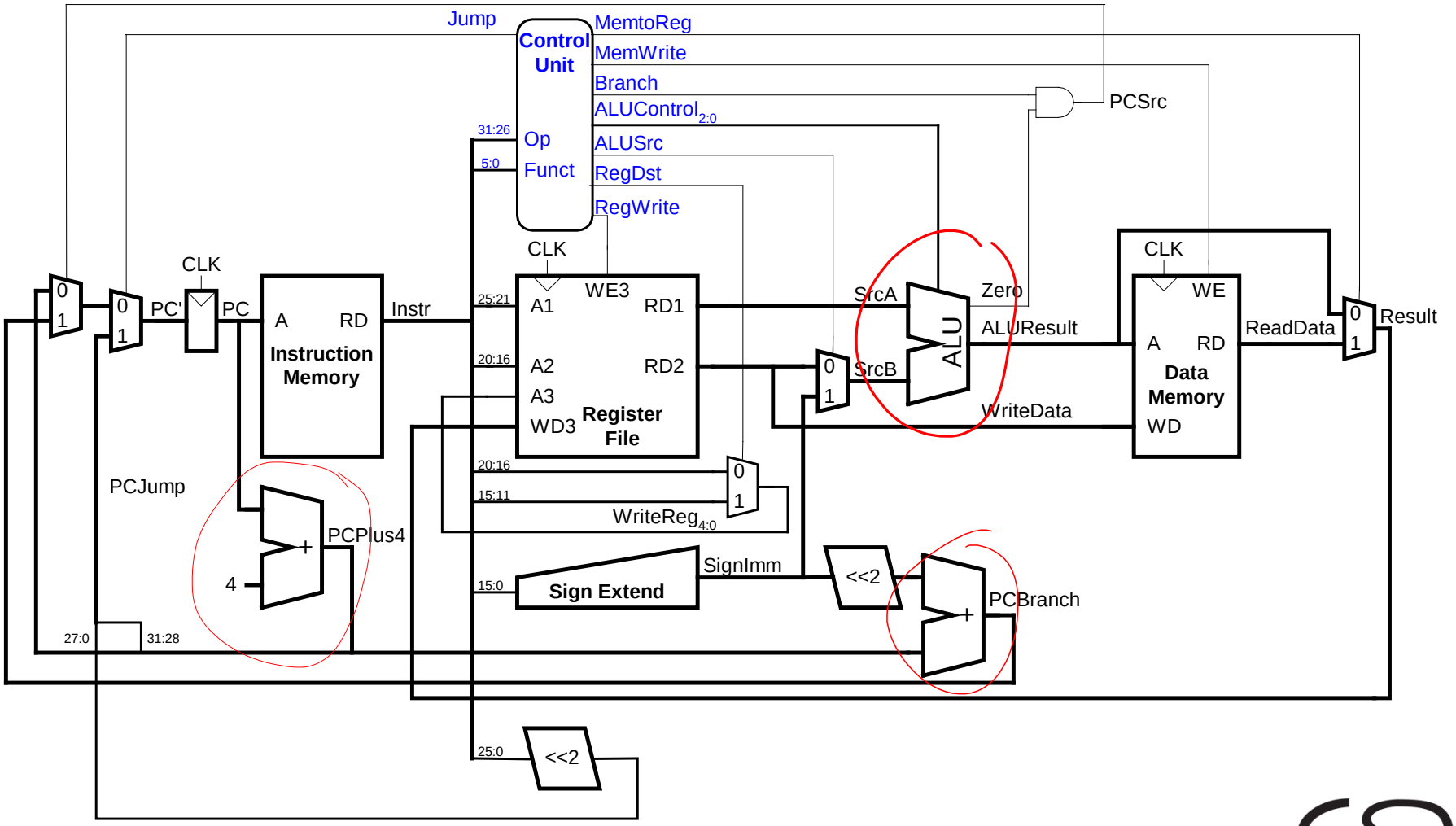
= 133,9 Sekunden

- **Langsamer als Ein-Takt-Prozessor (brauchte 92,5 Sekunden).**
  - Unterschiedlich lange Anzahl von Ausführungstakten (bis zu 5 für `lw`)
    - Aber nicht 5x schnellere Taktfrequenz
  - Nun zusätzliche Verzögerungen für sequentielle Logik mehrfach je Befehl

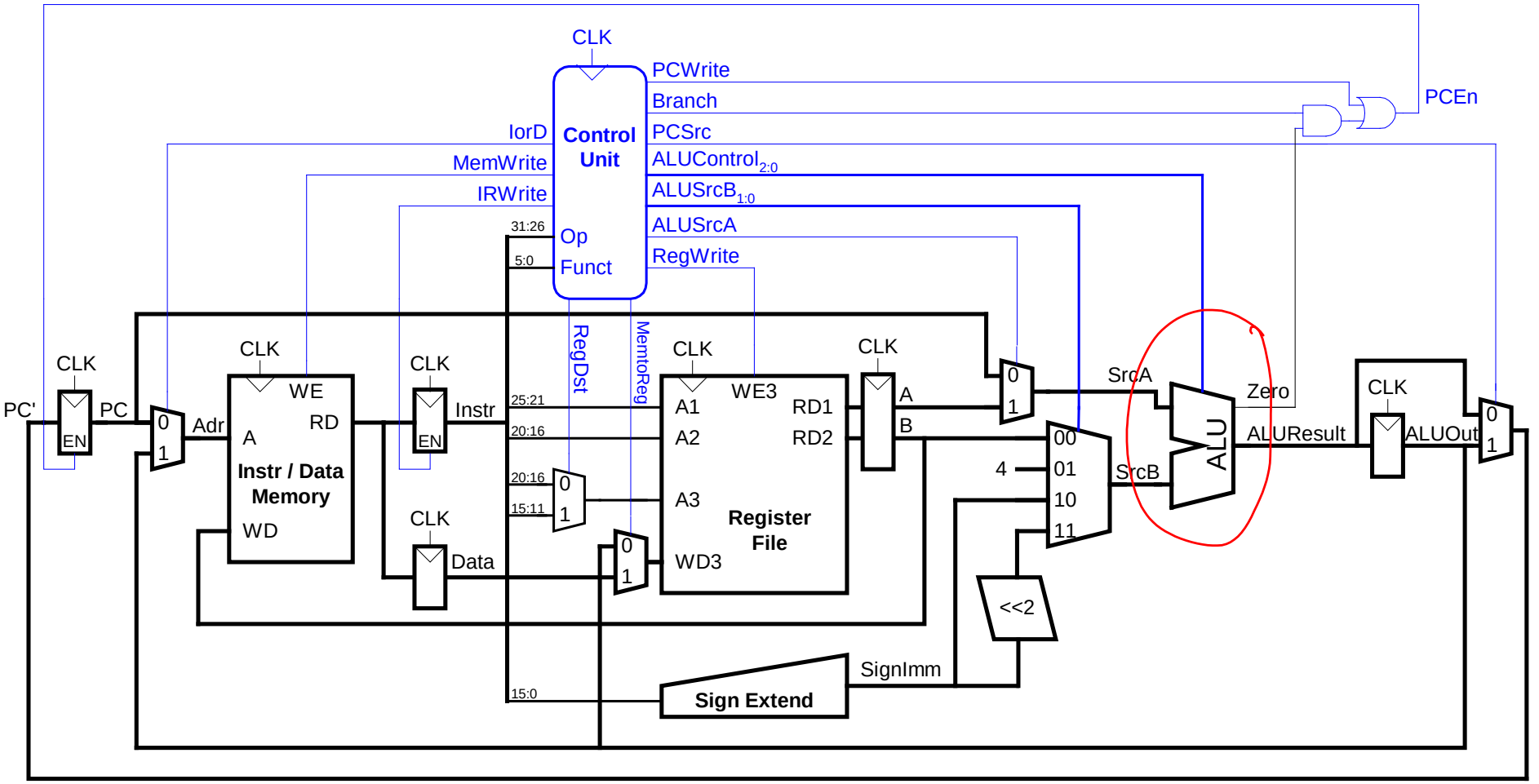
$t_{pcq} + t_{setup} = 50$ ps ✓

Embedded Systems & Applications

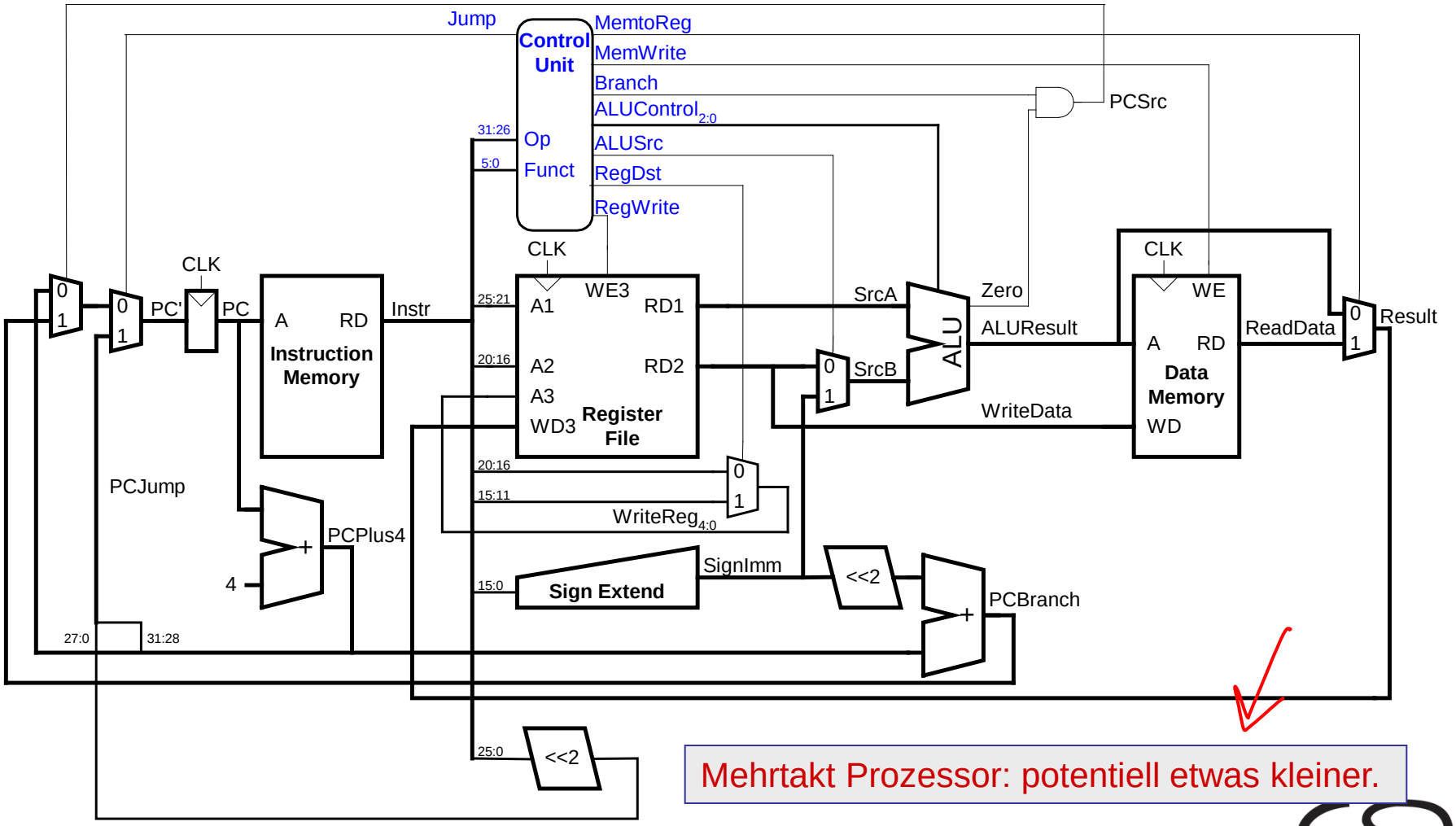
Rückblick: Ein-Takt MIPS Prozessor



Mehrtaktprozessor



Rückblick: Ein-Takt MIPS Prozessor



Mehrtakt Prozessor: potentiell etwas kleiner.

Mikroarchitektur

TECHNISCHE UNIVERSITÄT DARMSTADT

Mehrere Implementierungen für eine Architektur

- **Ein-Takt**

  Jede Instruktion wird in einem Takt ausgeführt

- **Mehrtakt**

  Jede Instruktion wird in Teilschritte zerlegt

- **Pipelined**

  Jede Instruktion wird in Teilschritte zerlegt

  Mehrere Instruktionen werden gleichzeitig ausgeführt

MIPS Prozessor mit Pipelining

TECHNISCHE UNIVERSITÄT DARMSTADT

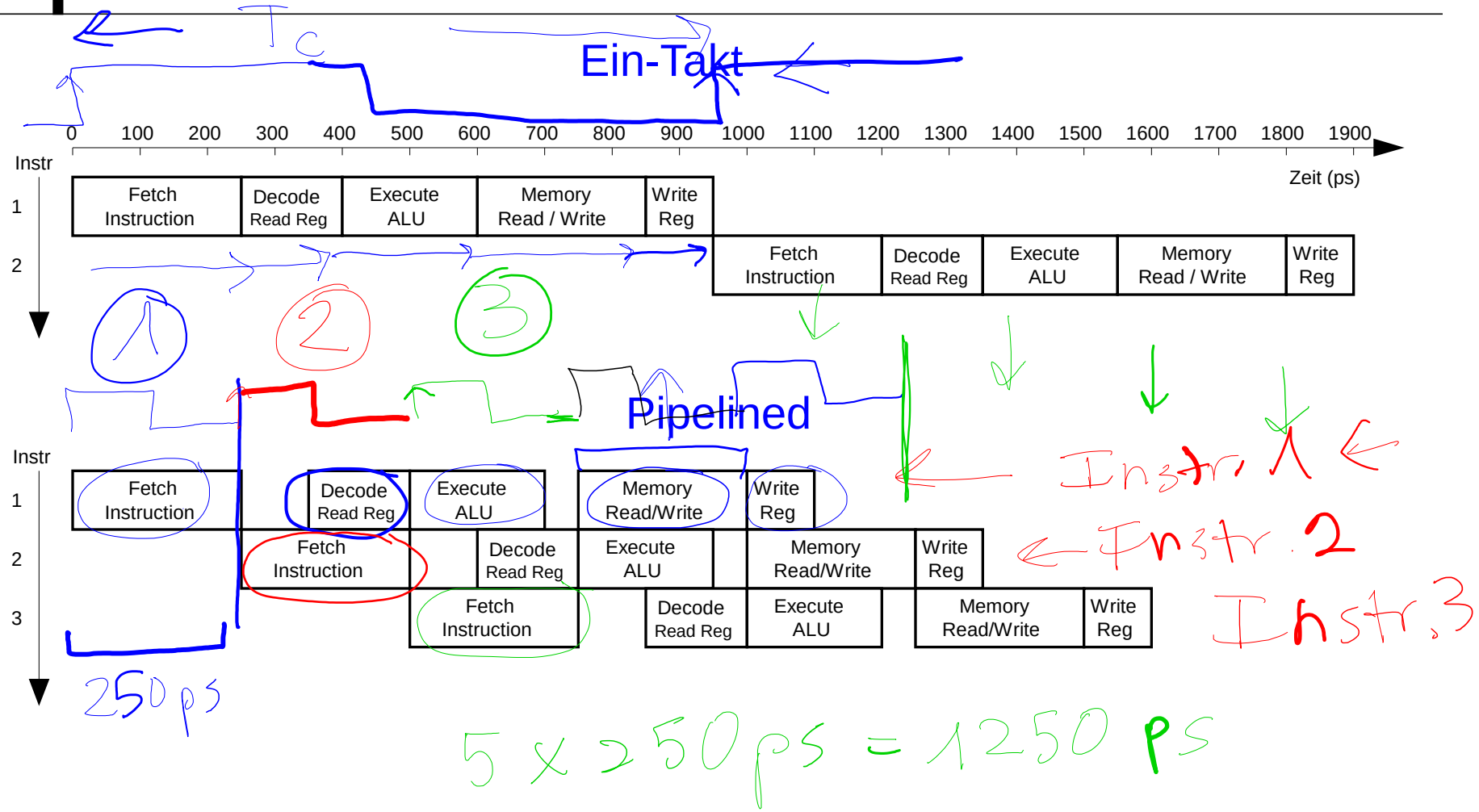
Zeitliche Parallelität

Teile Ablauf im Ein-Takt-Prozessor in fünf Stufen:

- Hole Instruktion (*Fetch*)
- Dekodiere Bedeutung von Instruktion (*Decode*)
- Führe Instruktion aus (*Execute*)
- Greife auf Speicher zu (*Memory*)
- Schreibe Ergebnisse zurück (*Writeback*)

Füge Pipeline-Register zwischen den Stufen ein

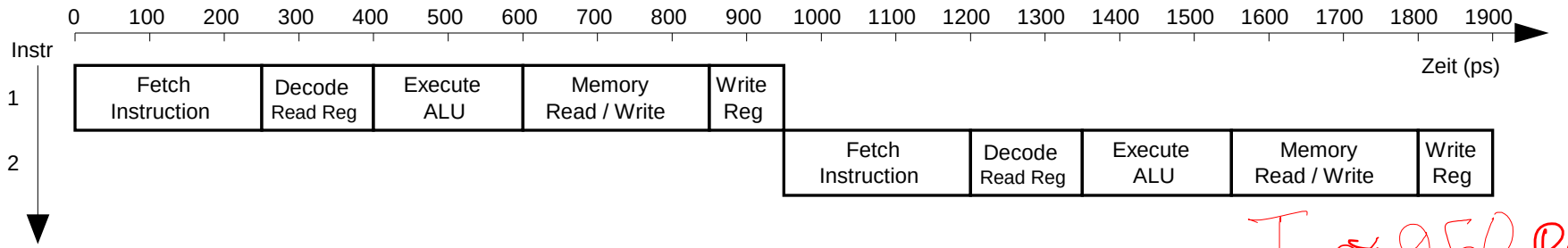
Rechenleistung: Ein-Takt und Pipelined



Rechenleistung: Ein-Takt und Pipelined

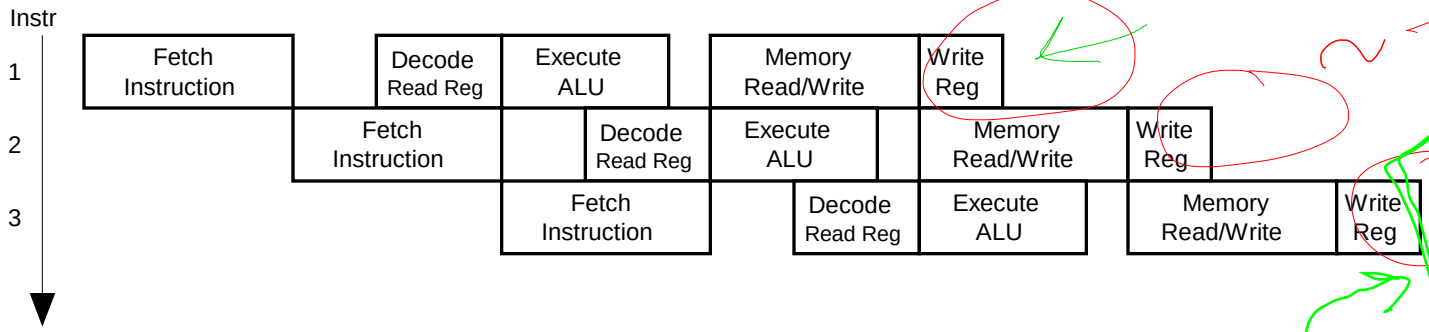
TECHNISCHE UNIVERSITÄT DARMSTADT

Ein-Takt



~ Tc ≥ 950 ps

Pipelined



~ Tc ≥ 250 ps

Eintakt = 1 Takt 950 ps

Pipelined = 5 Takte à 250 ps

Latenz: Zeit von der Eingabe einer Instruktion bis sie fertig ist

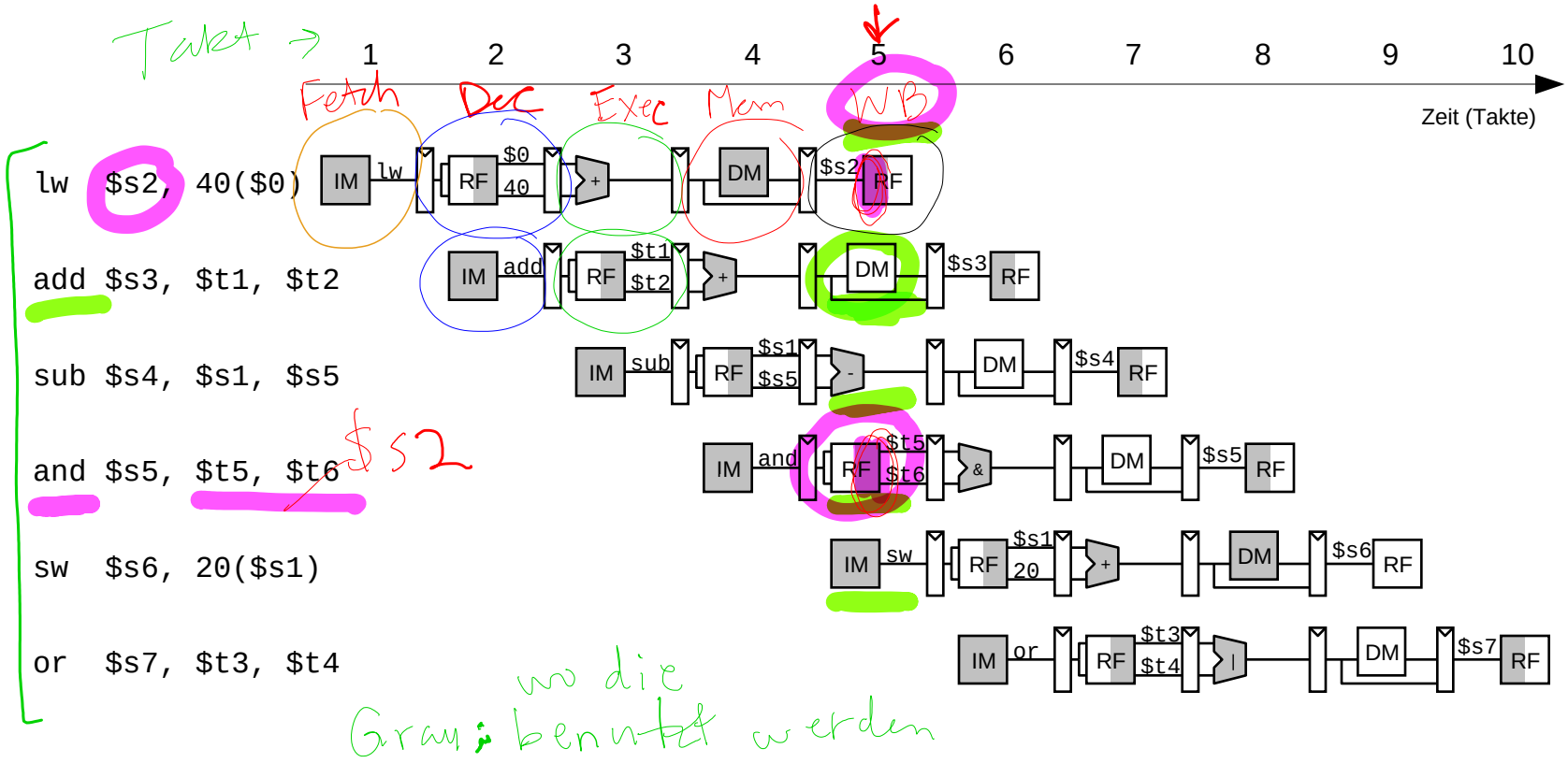
Durchsatz: Die Anzahl von Instruktionen die pro Zeiteinheit bearbeitet werden können

Eintakt: 1 Instr/Takt

Pipelined: ~ 1 Instr/Takt

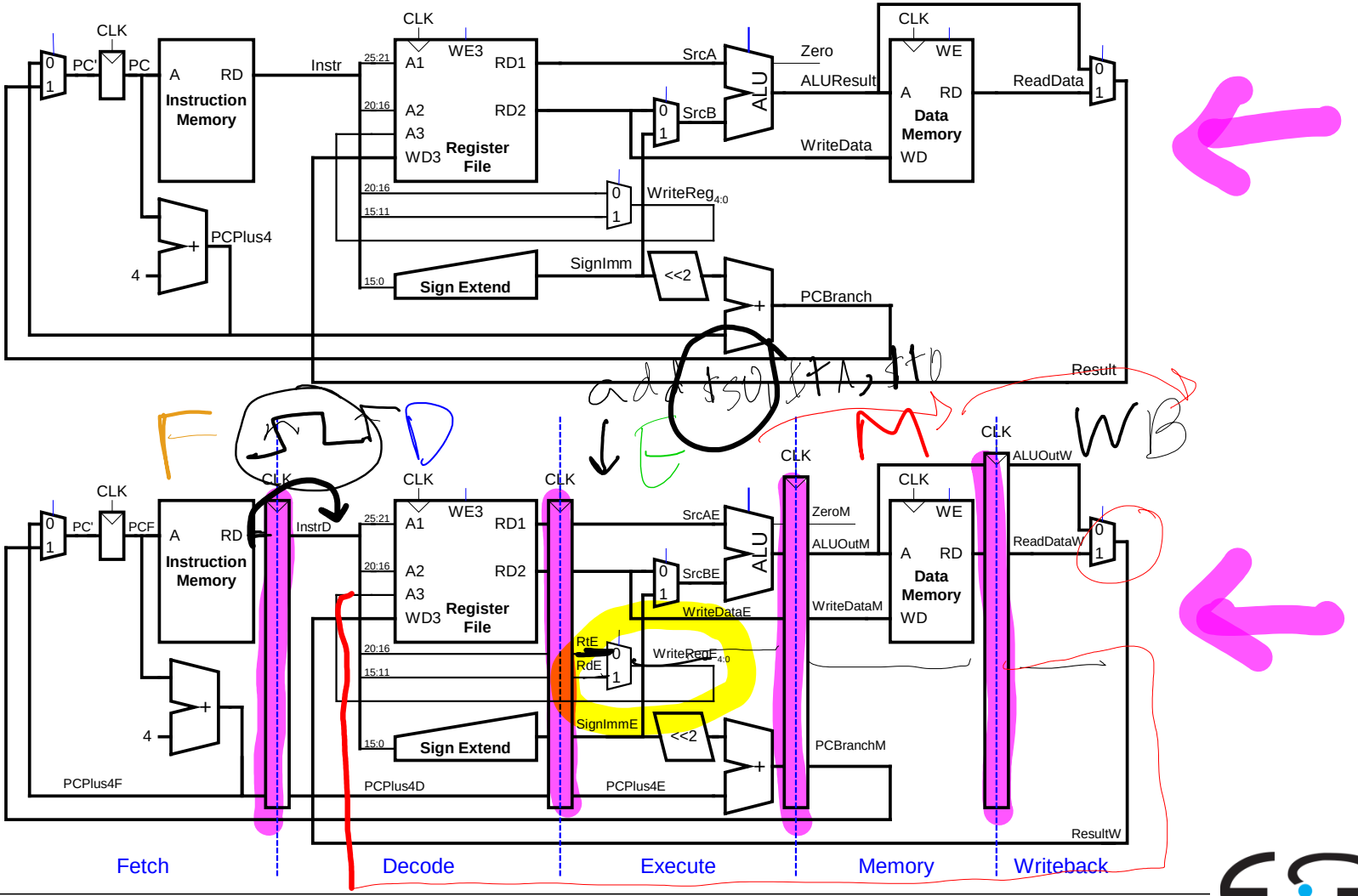
Embedded Systems & Applications

Abstraktere Darstellung des Pipelinings



Ein-Takt- und Pipelined-Datenpfad

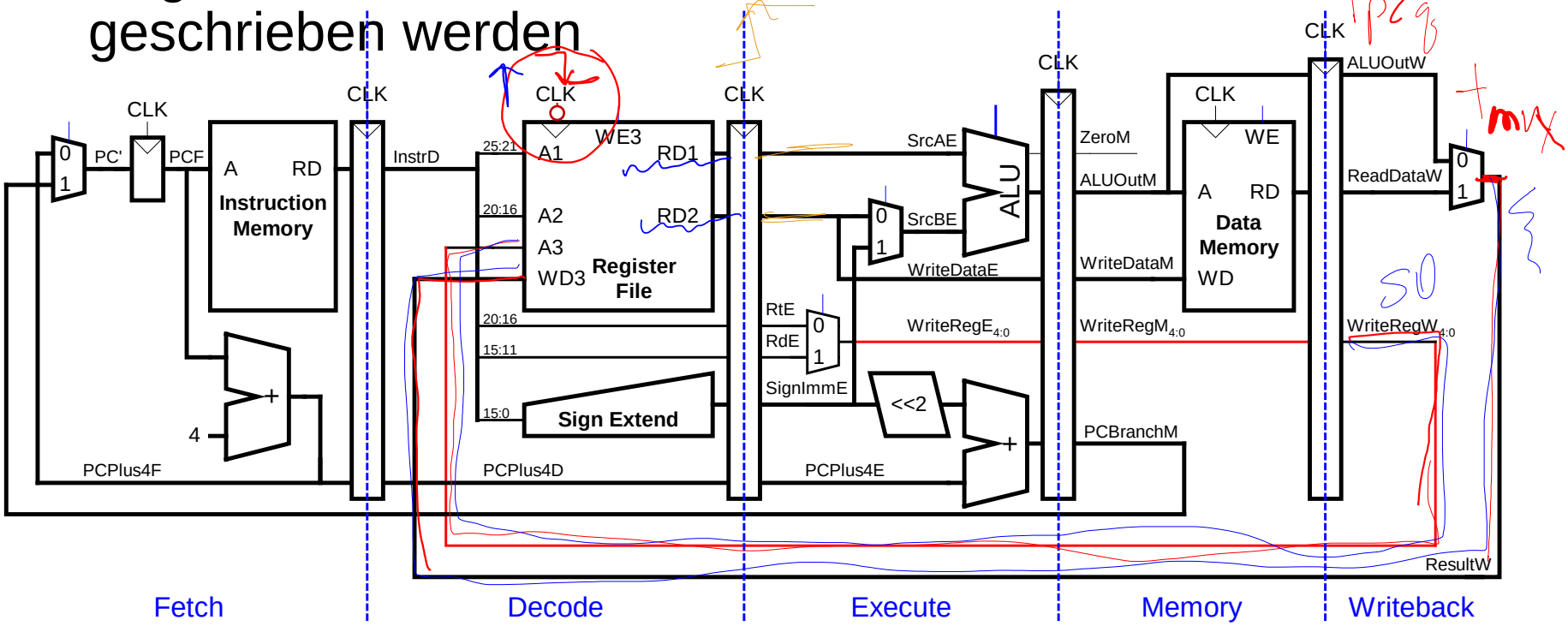
TECHNISCHE UNIVERSITÄT DARMSTADT



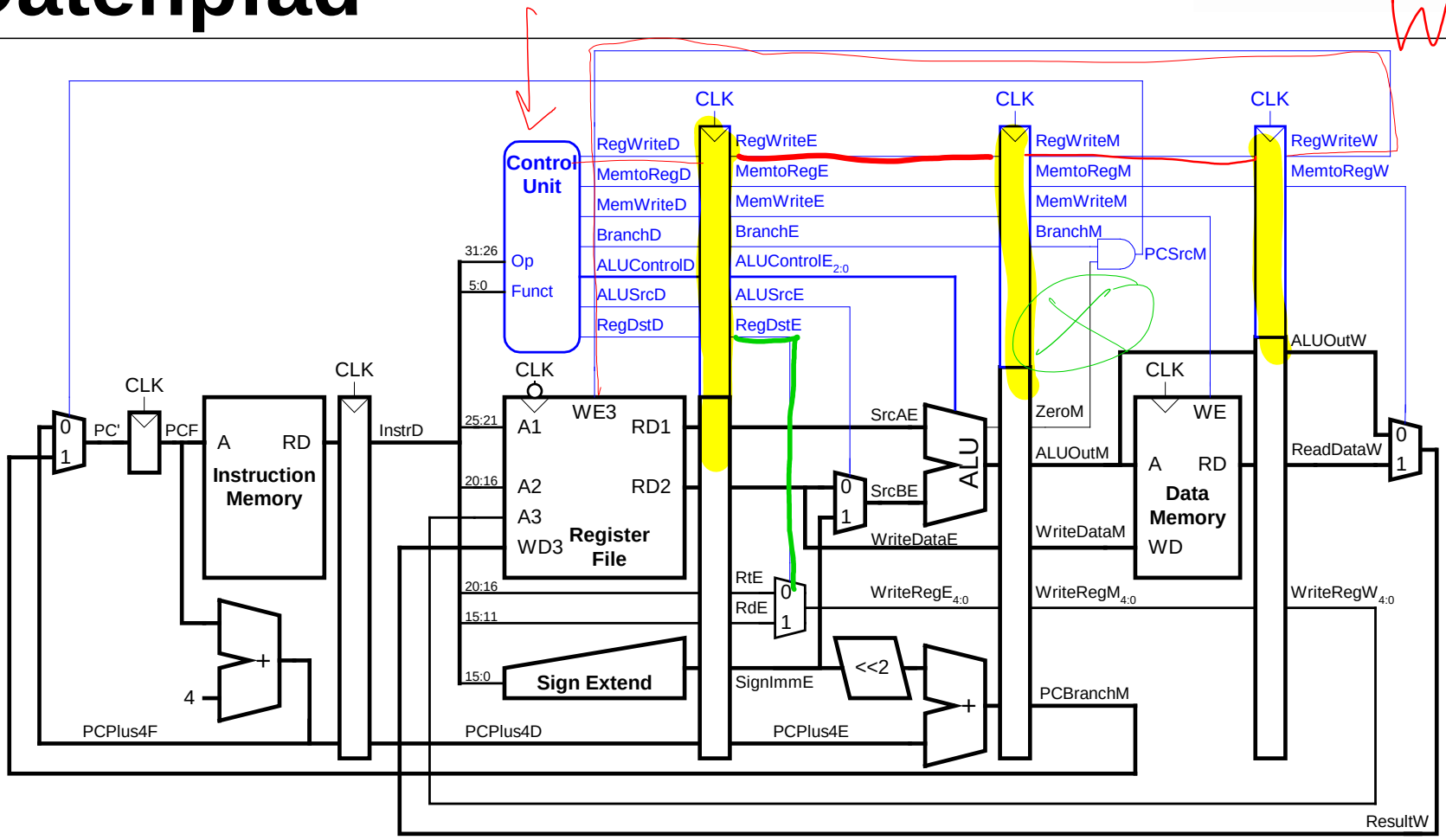
Embedded Systems & Applications

Korrigierter Pipelined-Datenpfad

- **WriteReg** muss zur gleichen Zeit am Registerfeld ankommen wie **Result**
- Registerfeld muss bei der **fallenden Taktflanke** geschrieben werden



Steuersignale für Pipelined-Datenpfad



Identisch zu Ein-Takt-Steuerwerk, aber Signale verzögert über Pipeline-Stufen

Abhängigkeiten zwischen Pipeline-Stufen (*hazards*)

TECHNISCHE UNIVERSITÄT DARMSTADT

Treten auf wenn eine

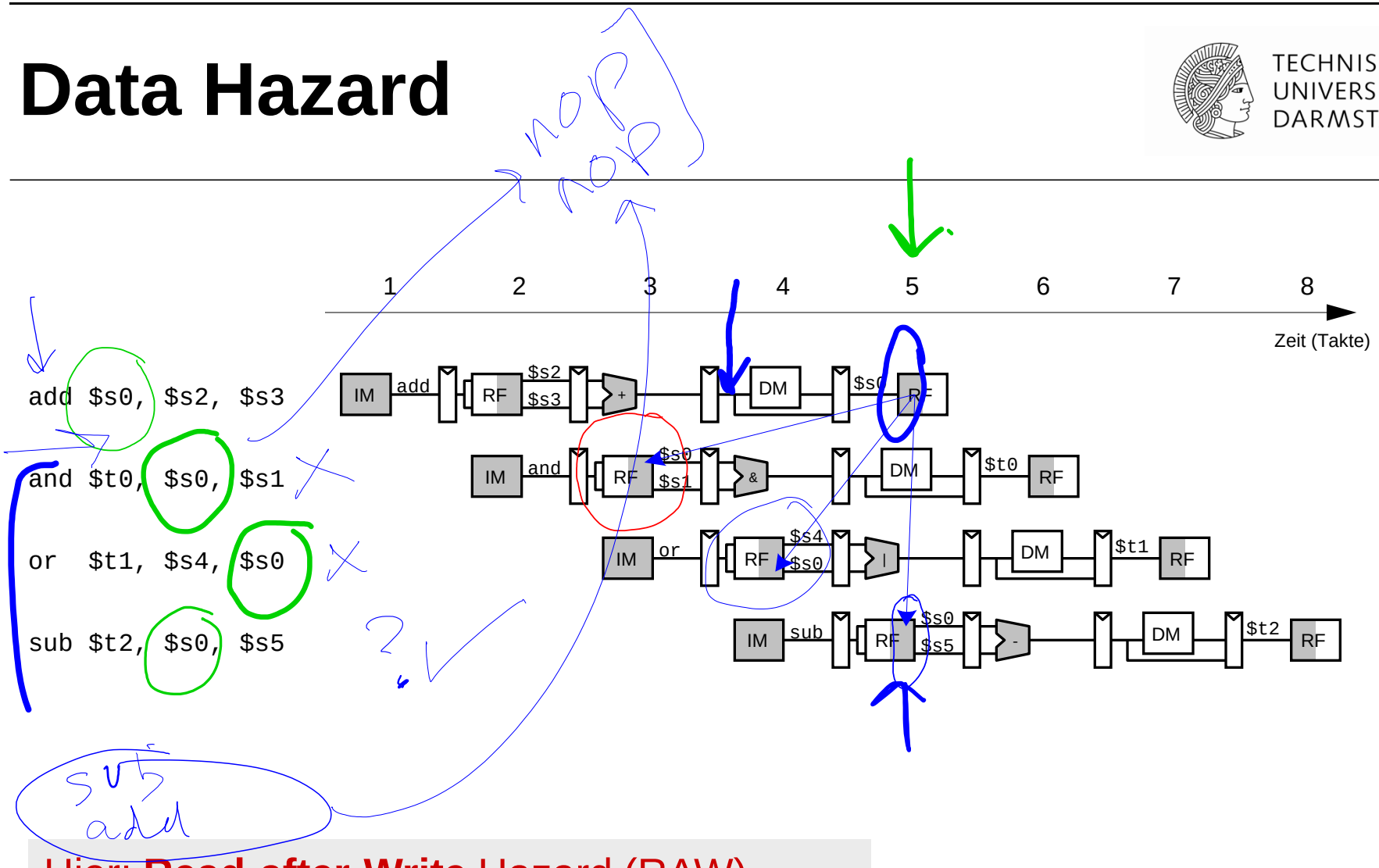
- Instruktion vom Ergebnis einer vorhergehenden abhängt
- … diese aber noch kein Ergebnis geliefert hat

Arten von Hazards

- **Data Hazard:** z.B. Neuer Wert von Register noch nicht in Registerfeld eingetragen
- **Control Hazard:** Unklar welche Instruktion als nächstes ausgeführt werden muss
  - Tritt bei Verzweigungen auf

Embedded Systems & Applications

Data Hazard



Hier: **Read-after-Write** Hazard (RAW)
- $s0 „muss vor Lesen geschrieben werden“

Umgang mit Data Hazards

TECHNISCHE UNIVERSITÄT DARMSTADT

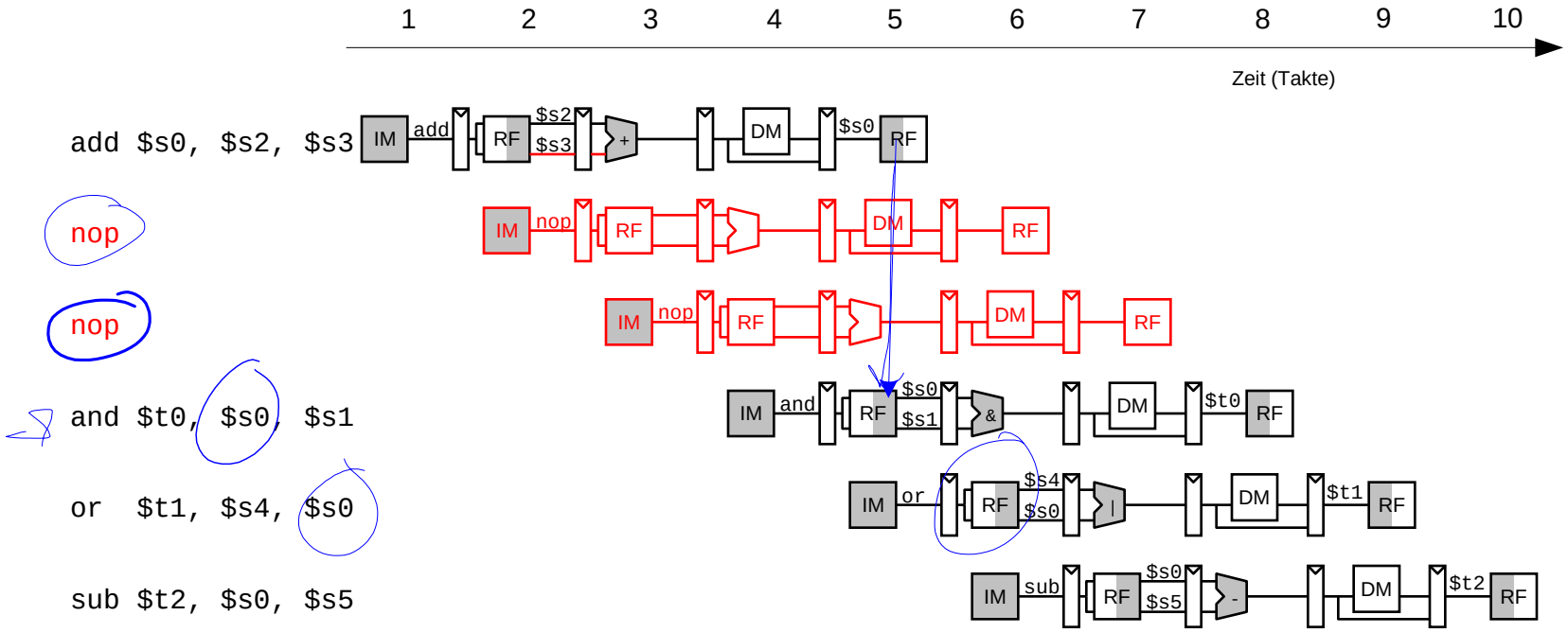
Möglichkeiten:

1. Plane Wartezeiten von Anfang an ein
   - Füge nops zur Compile-Zeit ein
   - *scheduling (Ablaufplanung)*
2. Stelle Maschinencode zur Compile-Zeit um
   - *scheduling / reordering*
3. Leite Daten zur Laufzeit schneller über Abkürzungen weiter
   - *bypassing / forwarding*
4. Halte Prozessor zur Laufzeit an bis Daten da sind
   - *stalling*

Beseitigung von Data Hazards zur Compile-Zeit

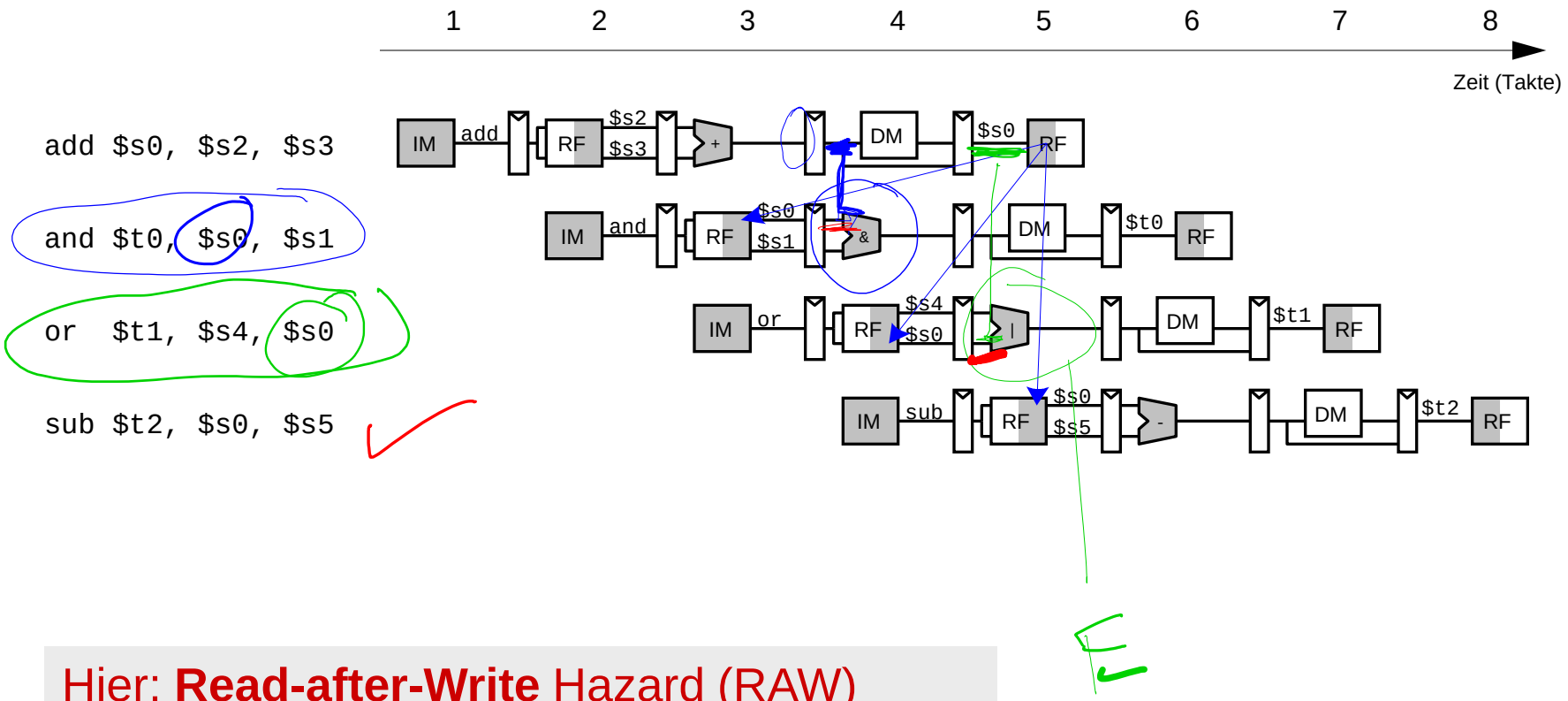
TECHNISCHE UNIVERSITÄT DARMSTADT

- Füge ausreichend viele nops ein bis Ergebnis bereitsteht
- Oder schiebe unabhängige Instruktionen nach vorne (statt nops)



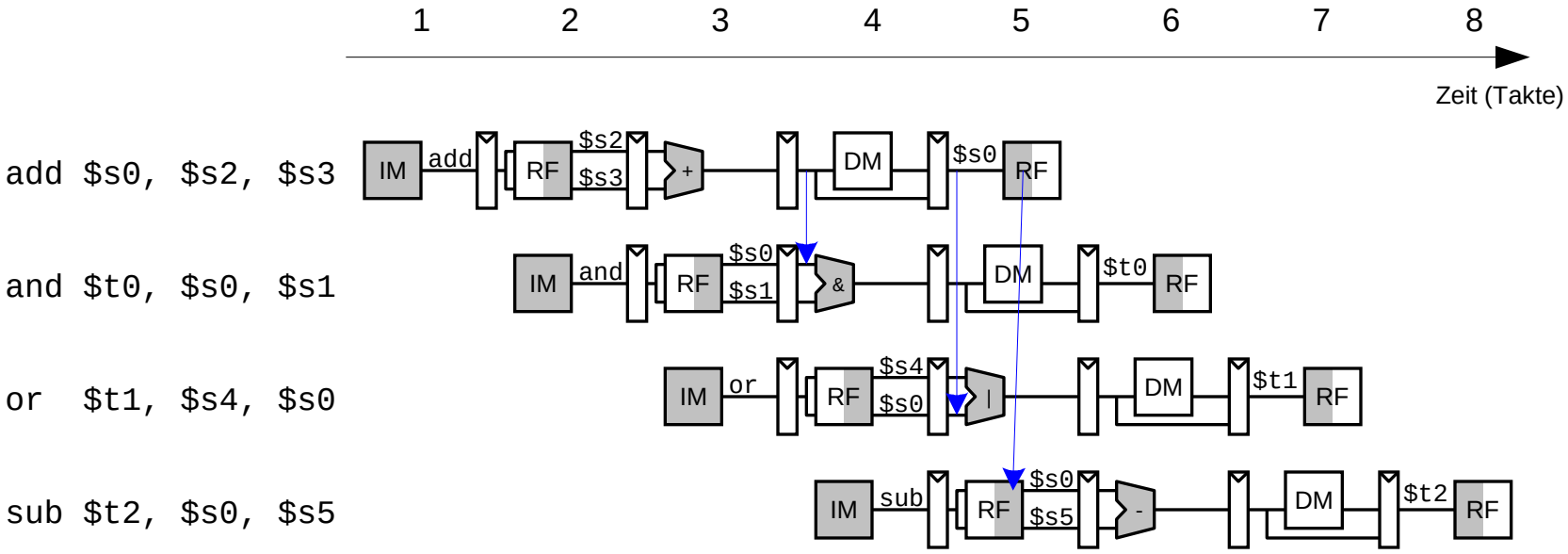
Embedded Systems & Applications

Data Hazard



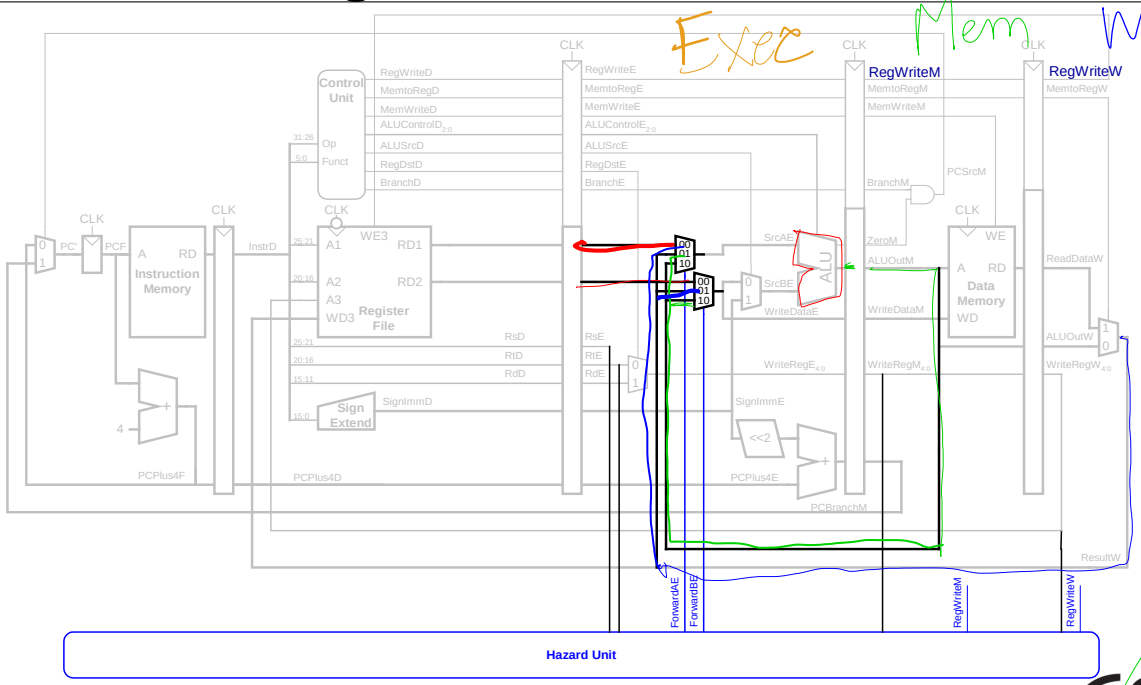
Hier: **Read-after-Write** Hazard (RAW)
- $s0 „muss vor Lesen geschrieben werden“

Data Forwarding: “Abkürzungen” einbauen



Data Forwarding: “Abkürzungen” einbauen

TECHNISCHE UNIVERSITÄT DARMSTADT



add $0, $s1, $s2

$0

$0

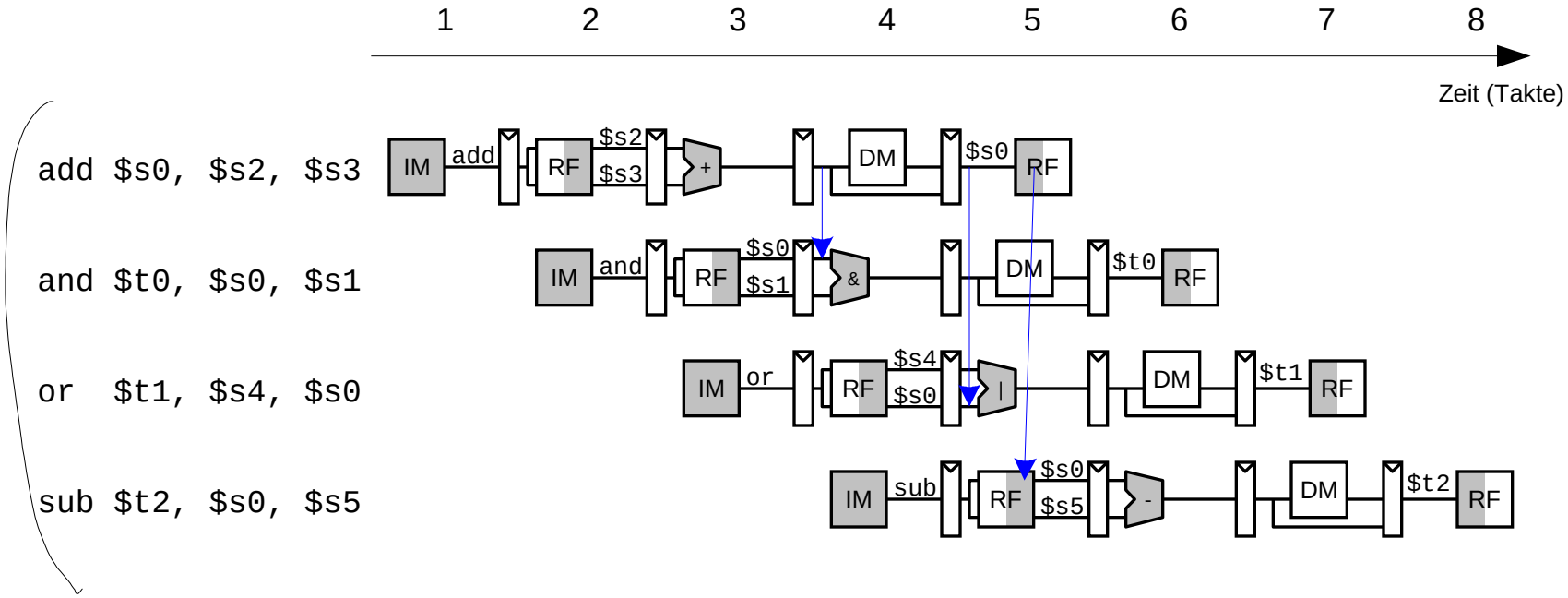
if RegWriteM AND (rsE == WriteRegM) AND (rsE != 0)
ForwardAE = 10

else if RegWriteW AND (rsE == WriteRegW) AND (rsE != 0)
ForwardAE = 01

else ForwardAE = 00 ← vom RF

Embedded Systems & Applications

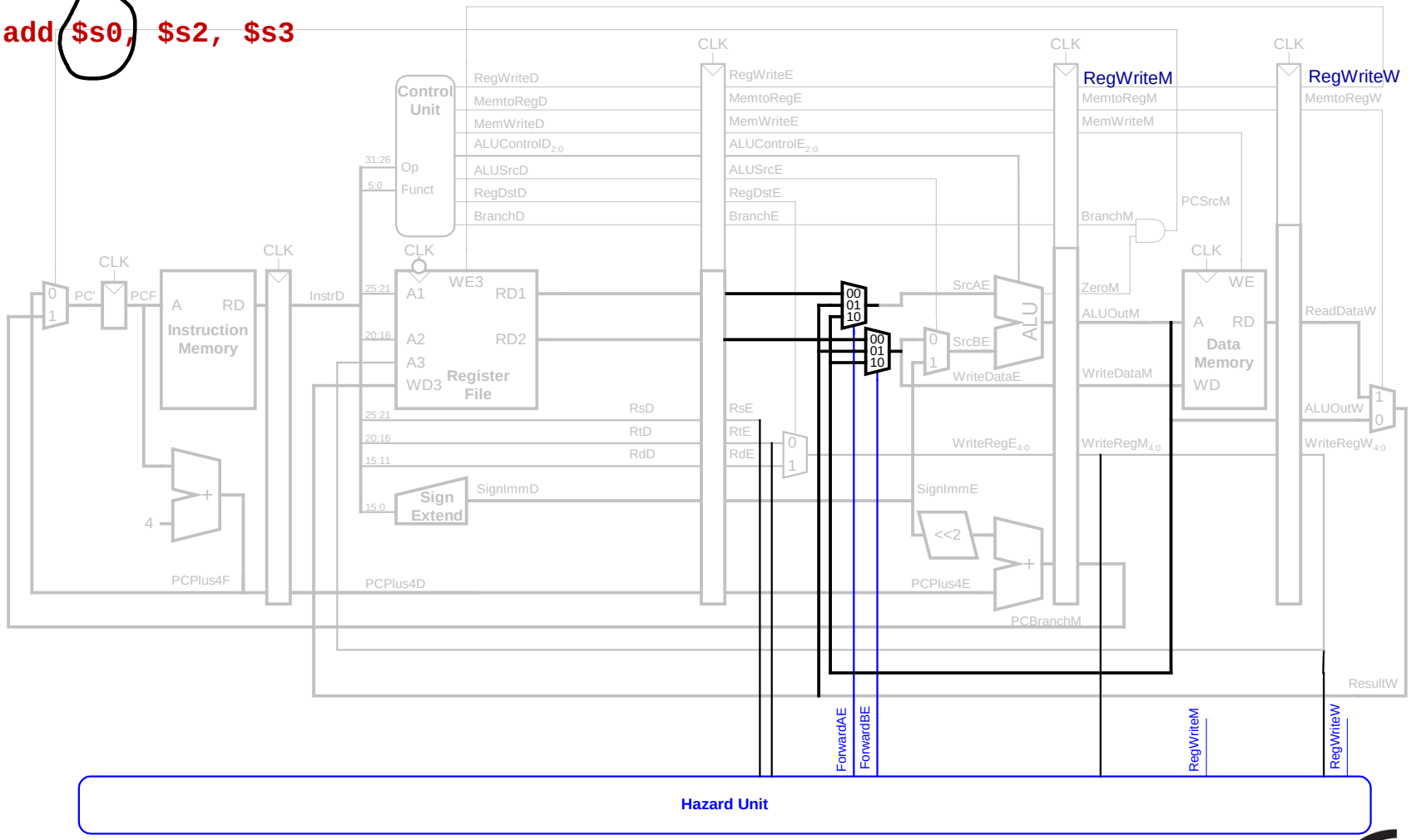
Data Forwarding: “Abkürzungen” einbauen



Data Forwarding: “Abkürzungen” einbauen: Takt 1

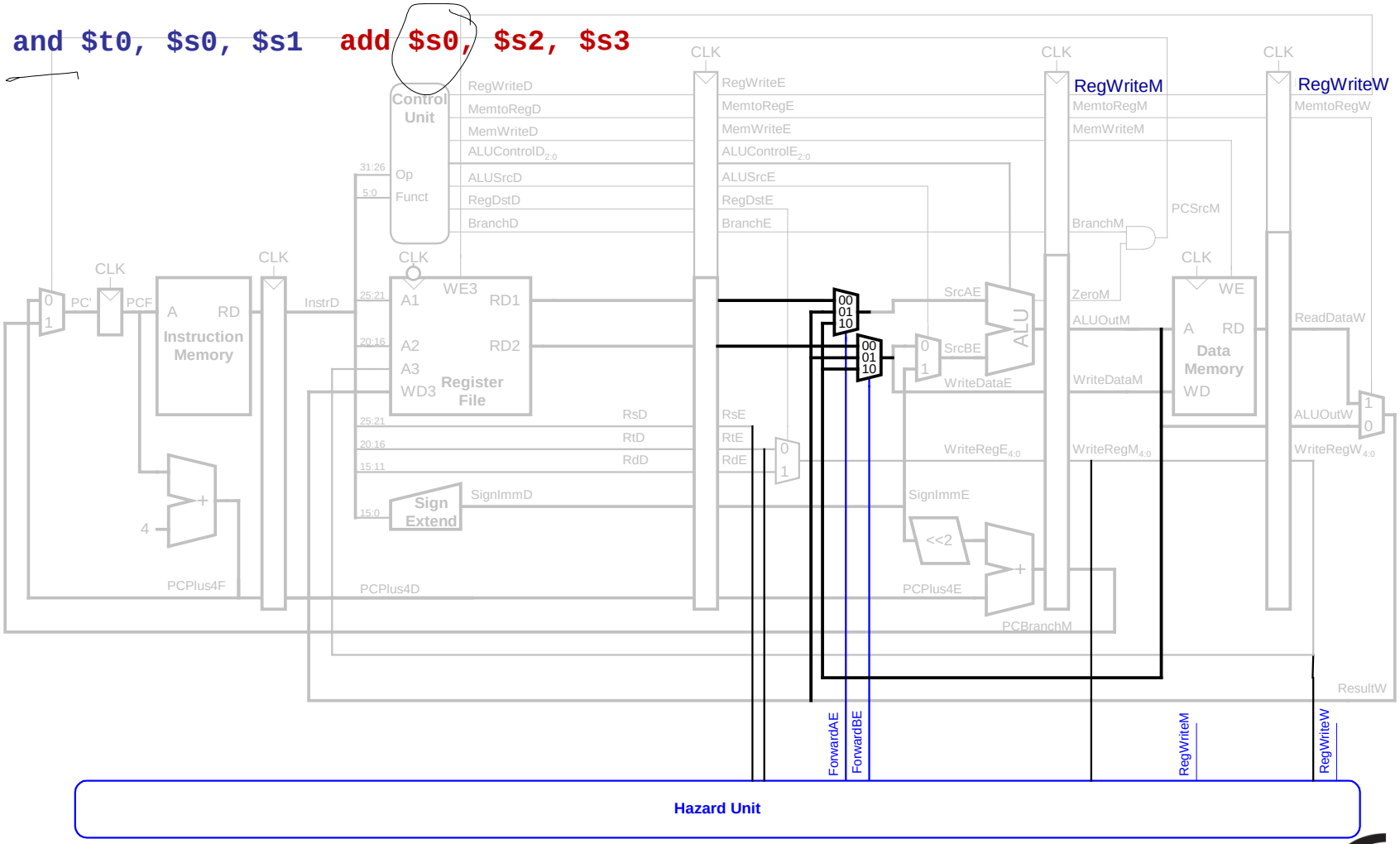
TECHNISCHE UNIVERSITÄT DARMSTADT

```
add $s0, $s2, $s3
```



Embedded Systems & Applications

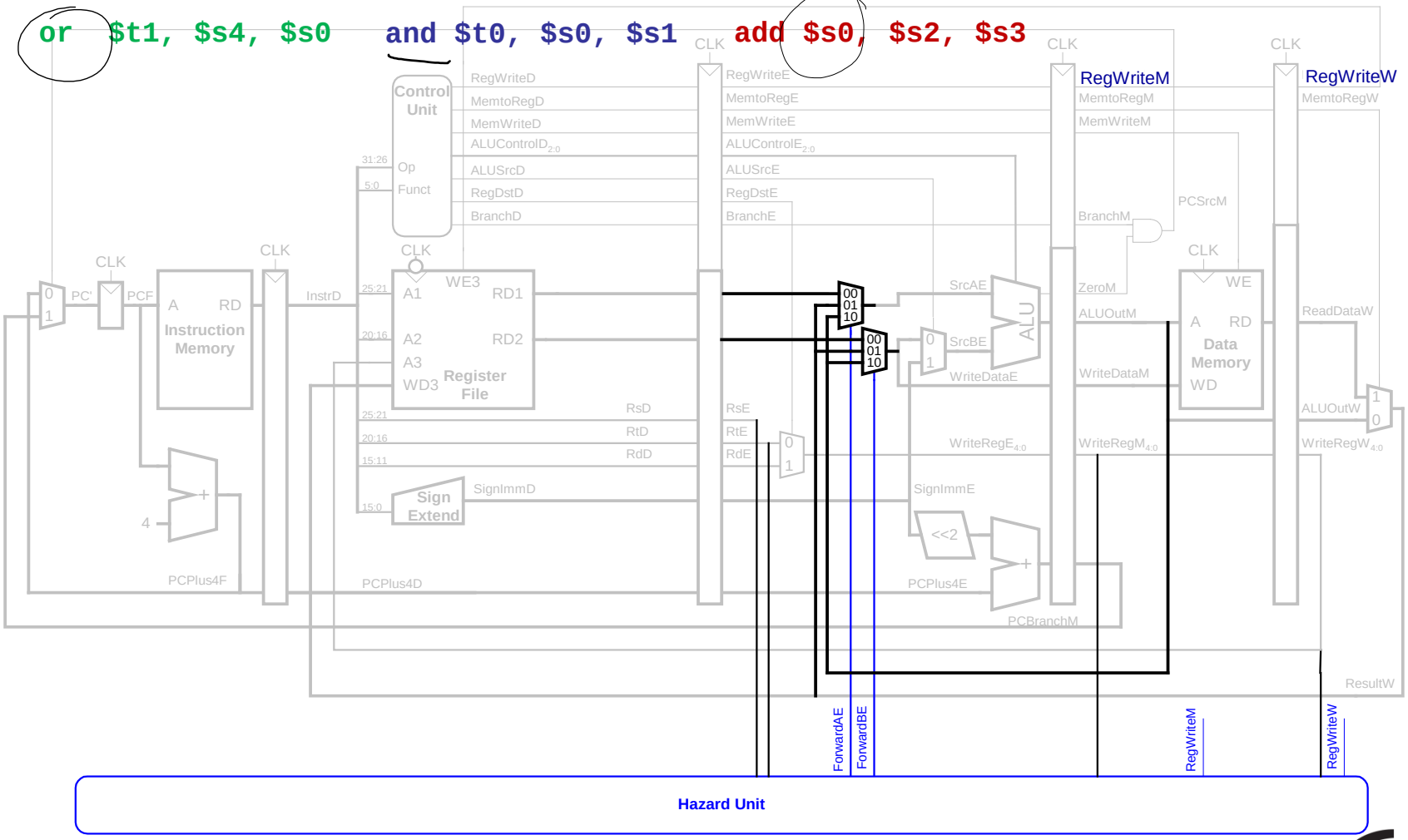
Data Forwarding: "Abkürzungen" einbauen: Takt 2



Data Forwarding: “Abkürzungen” einbauen: Takt 3

TECHNISCHE UNIVERSITÄT DARMSTADT

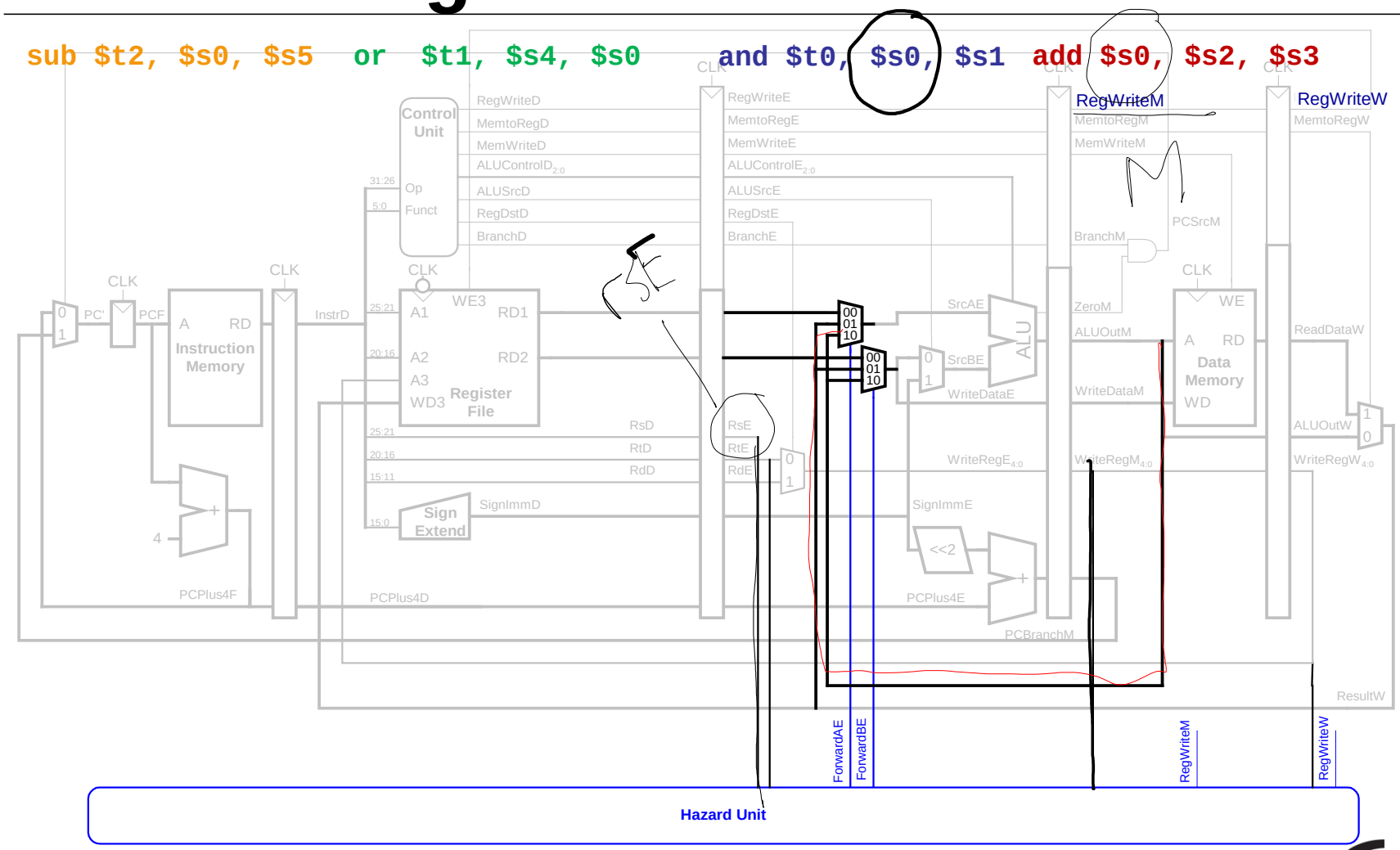
```
or  $t1, $s4, $s0    and $t0, $s0, $s1    add $s0, $s2, $s3
```



Embedded Systems & Applications

Data Forwarding: “Abkürzungen” einbauen: Takt 4

TECHNISCHE UNIVERSITÄT DARMSTADT

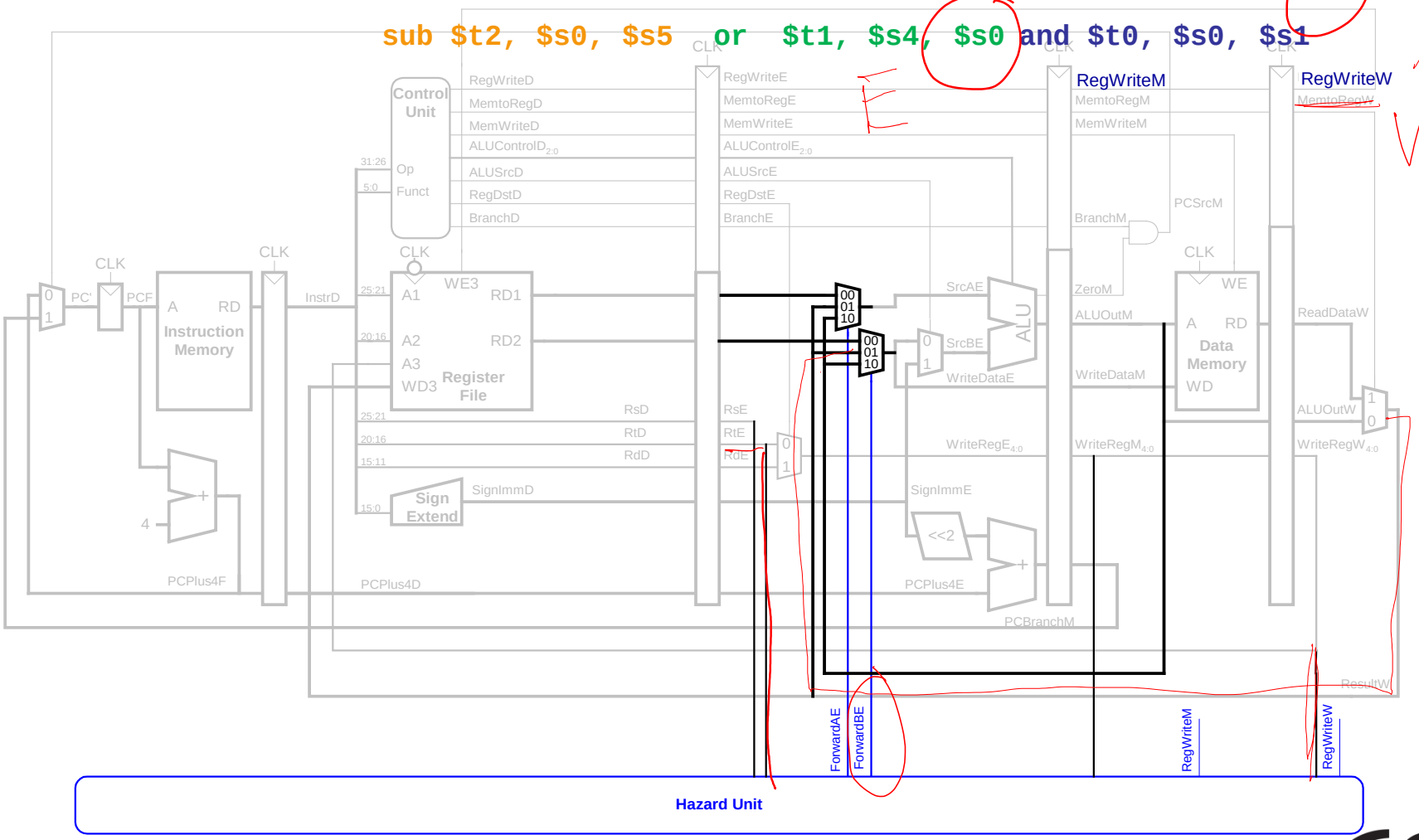


Embedded Systems & Applications

Data Forwarding: “Abkürzungen” einbauen: Takt 5

TECHNISCHE UNIVERSITÄT DARMSTADT

```
sub $t2, $s0, $s5   or  $t1, $s4, $s0   and $t0, $s0, $s1   add $s0, $s2, $s3
```



Embedded Systems & Applications

Data Forwarding: “Abkürzungen” einbauen

TECHNISCHE UNIVERSITÄT DARMSTADT

“Abkürzung” zur Execute-Stufe von

- Memory-Stufe oder
- Writeback-Stufe

Forwarding-Logik für Signal *ForwardAE* (Weiterleiten von Operand A):

Embedded Systems & Applications

Data Forwarding: “Abkürzungen” einbauen

TECHNISCHE UNIVERSITÄT DARMSTADT

“Abkürzung” zur Execute-Stufe von

- Memory-Stufe oder
- Writeback-Stufe

Forwarding-Logik für Signal *ForwardAE* (Weiterleiten von Operand A):

```
if      ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM)
then     ForwardAE = 10
else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)
then    ForwardAE = 01
else     ForwardAE = 00
```

Forwarding-Logik für Signal *ForwardBE* (Weiterleiten von Operand B) analog: ersetze *rsE* durch *rtE*

Embedded Systems & Applications

Data Forwarding: “Abkürzungen” einbauen

TECHNISCHE UNIVERSITÄT DARMSTADT

“Abkürzung” zur Execute-Stufe von

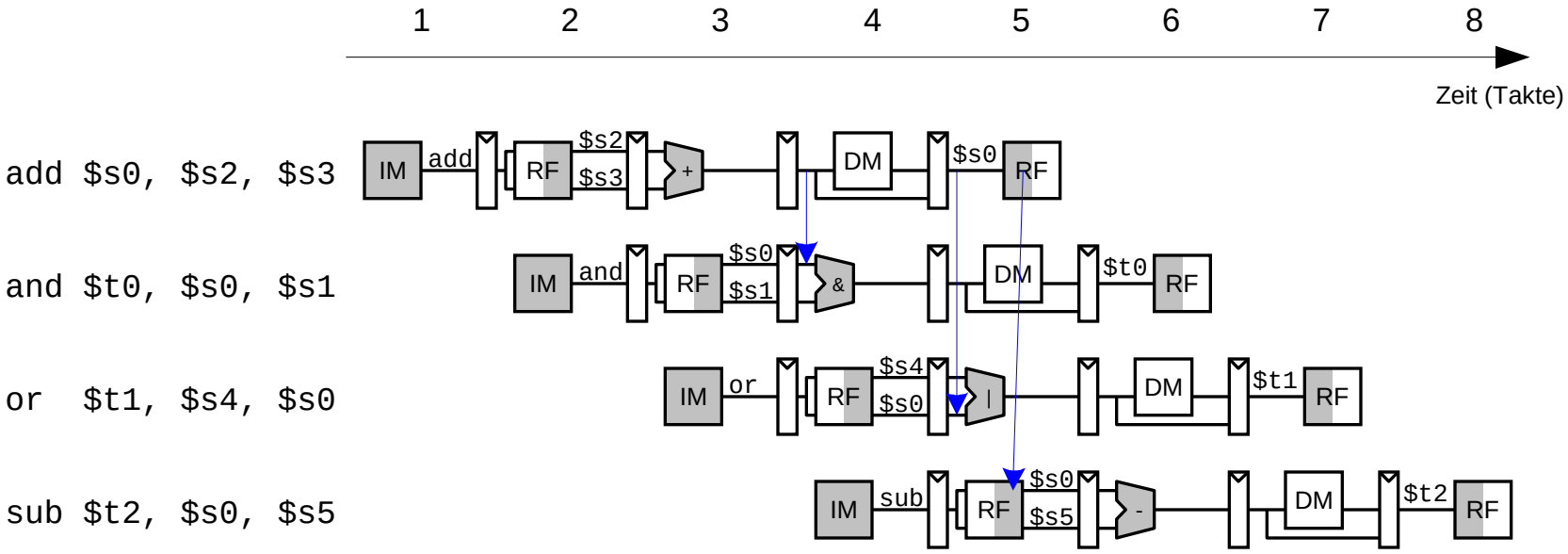
- Memory-Stufe oder
- Writeback-Stufe

Forwarding-Logik für Signal *ForwardBE* (Weiterleiten von Operand B):

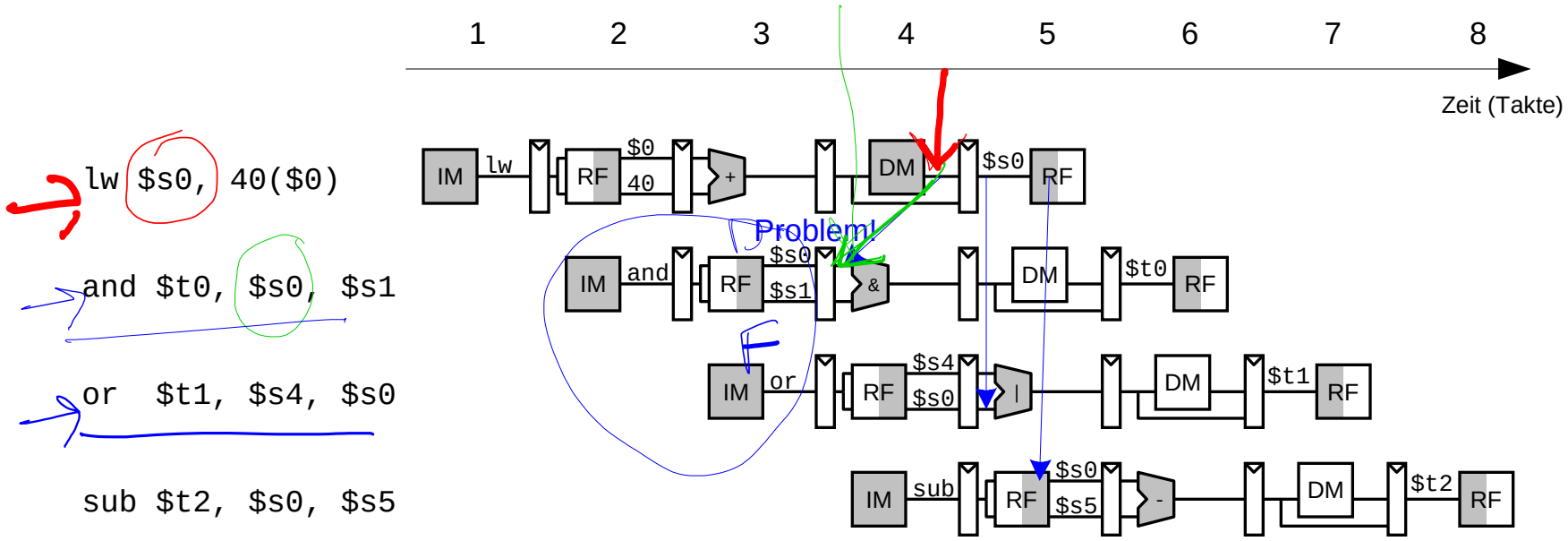
```
if      ((rtE != 0) AND (rtE == WriteRegM) AND RegWriteM)
then     ForwardBE = 10
else if ((rtE != 0) AND (rtE == WriteRegW) AND RegWriteW)
then    ForwardBE = 01
else     ForwardBE = 00
```


Embedded Systems & Applications

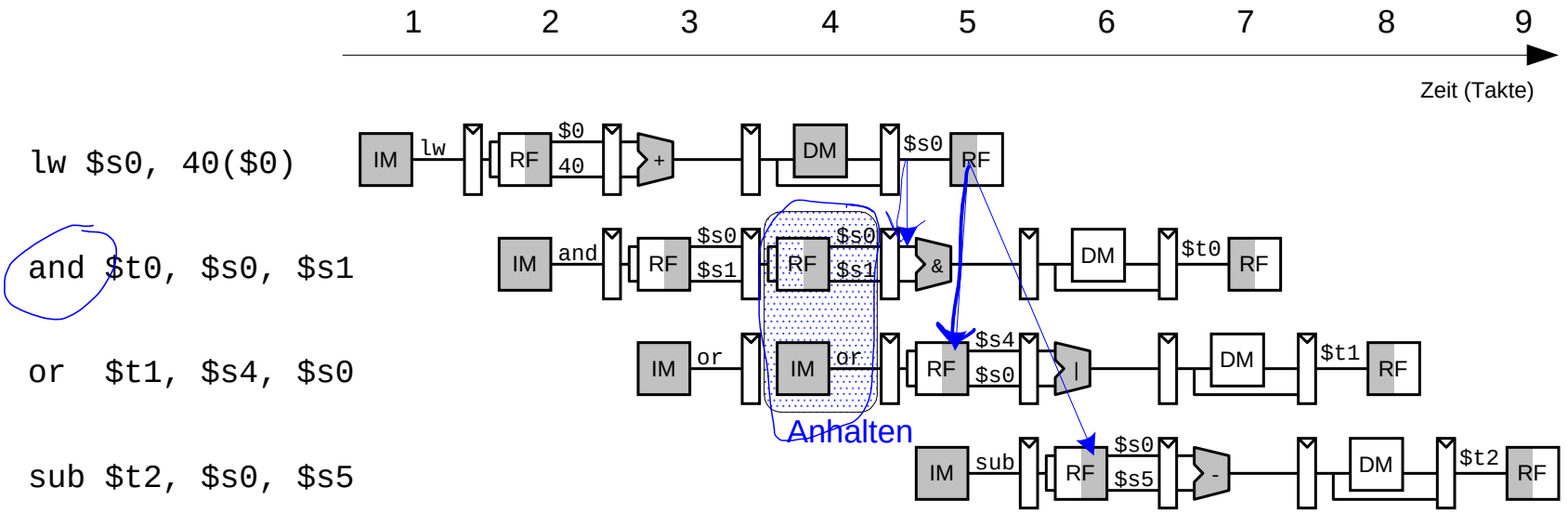
Data Forwarding: “Abkürzungen” einbauen



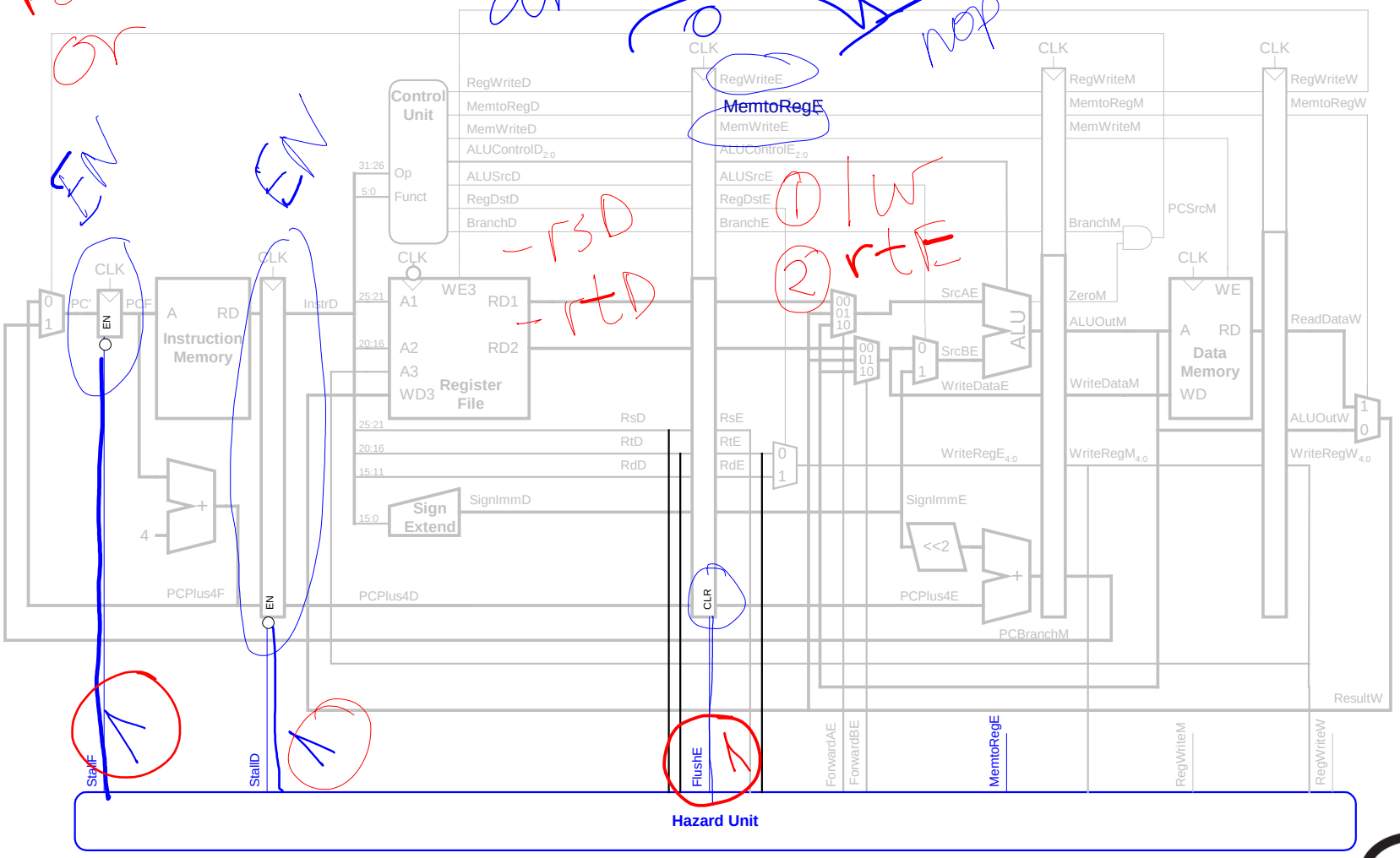
Anhalten des Prozessors (*stalling*)



Anhalten des Prozessors (*stalling*)



Erweiterung der Hazard-Einheit für Stalling



Behandlung von Stalling in Hazard-Einheit

lw in Exec

Stalling-Logik:

lwstall = MemtoRegE AND [(rsD != 0) AND (rsD == rtE) OR (rtD != 0) AND (rtD == rtE)]

lw $0, 5($t0)

Behandlung von Stalling in Hazard-Einheit

TECHNISCHE UNIVERSITÄT DARMSTADT

Stalling-Logik:

```
lwstall = MemtoRegE AND((rsD==rtE) OR
                        (rtD==rtE))

StallF = StallD = FlushE = lwstall
```


Embedded Systems & Applications

Behandlung von Stalling in Hazard-Einheit

TECHNISCHE UNIVERSITÄT DARMSTADT

Stalling-Logik:

```
lwstall = MemtoRegE AND((rsD!=0)AND(rsD==rtE) OR
                        (rtD!=0)AND(rtD==rtE))
```

```
StallF = StallD = FlushE = lwstall
```


Embedded Systems & Applications

Control Hazards

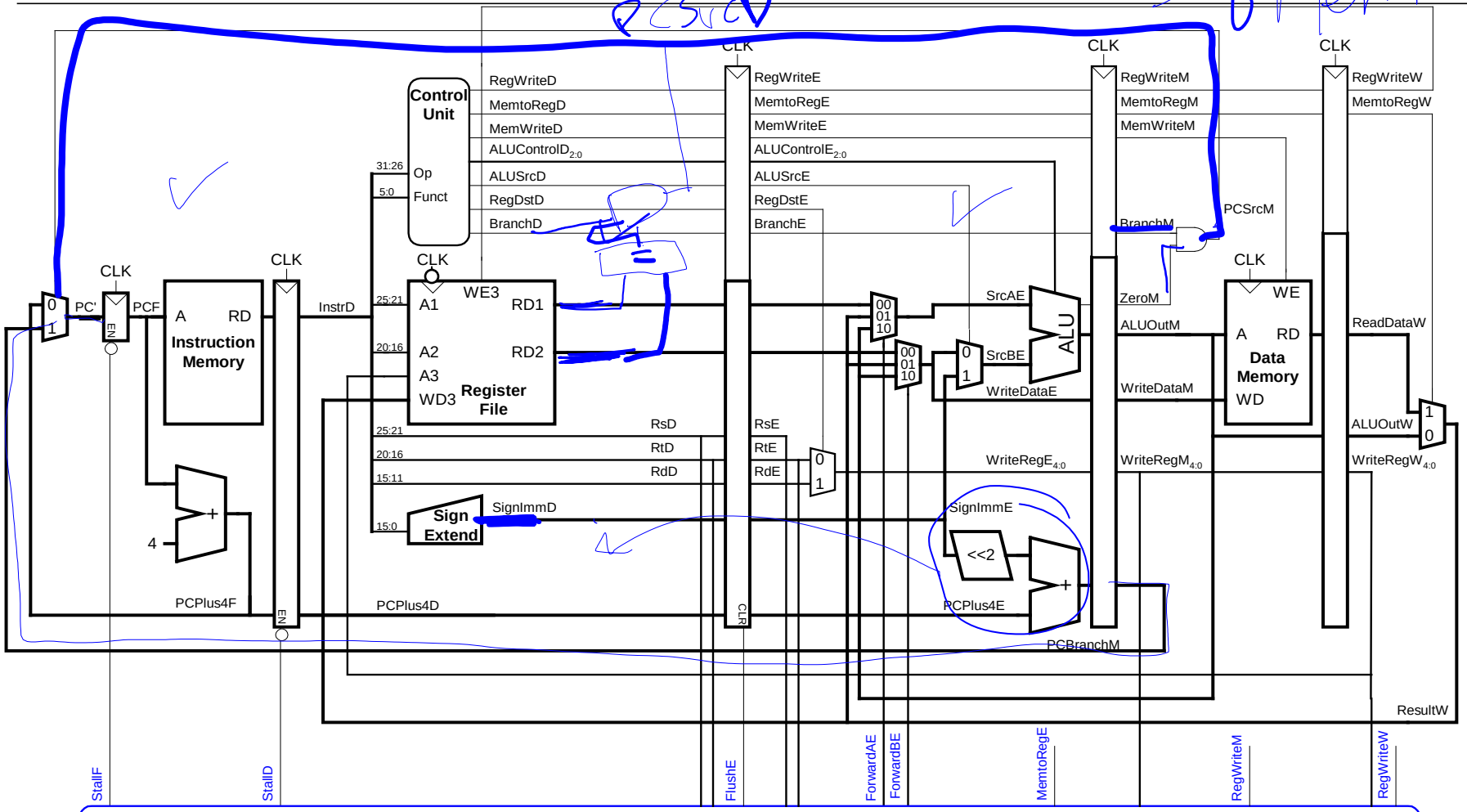
TECHNISCHE UNIVERSITÄT DARMSTADT

beq:

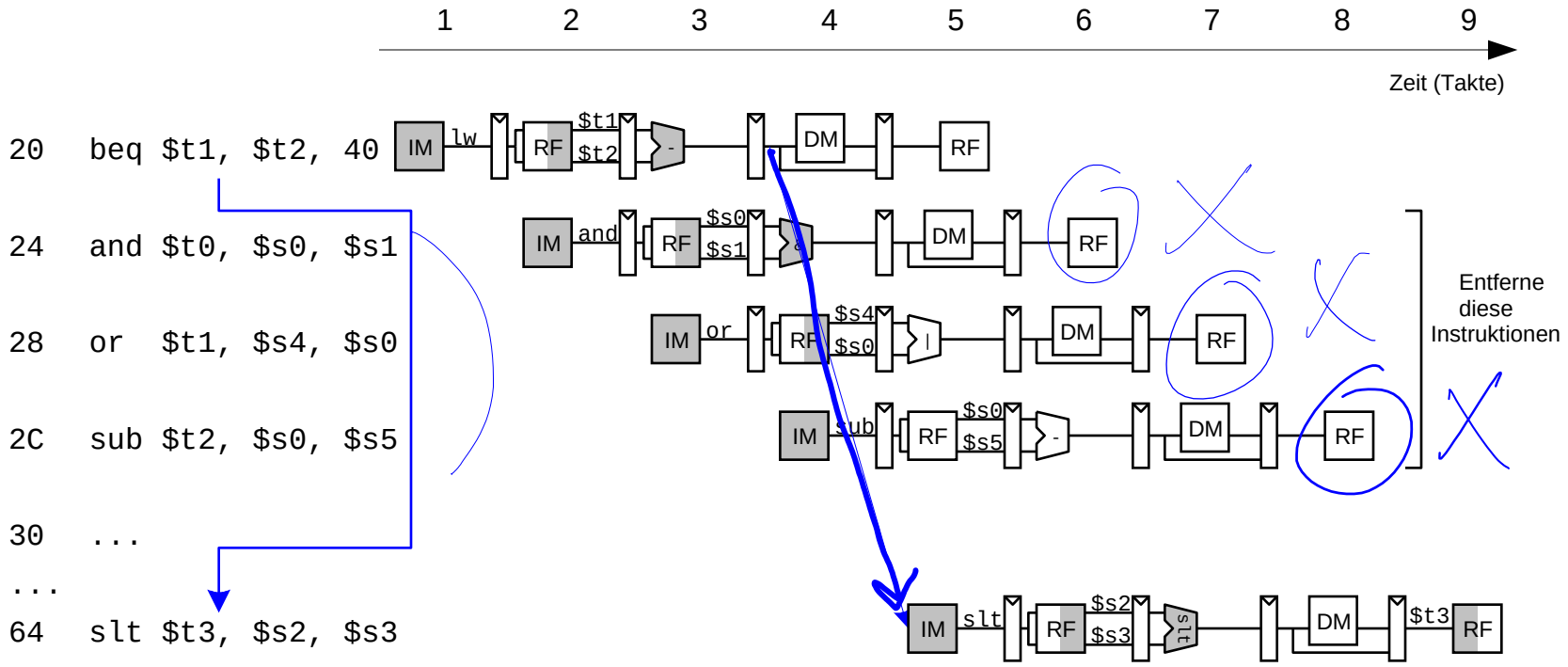
- Entscheidung zu Springen wird erst in vierter Stufe der Pipeline (M) getroffen
- Neue Instruktionen werden aber bereits geholt

Embedded Systems & Applications

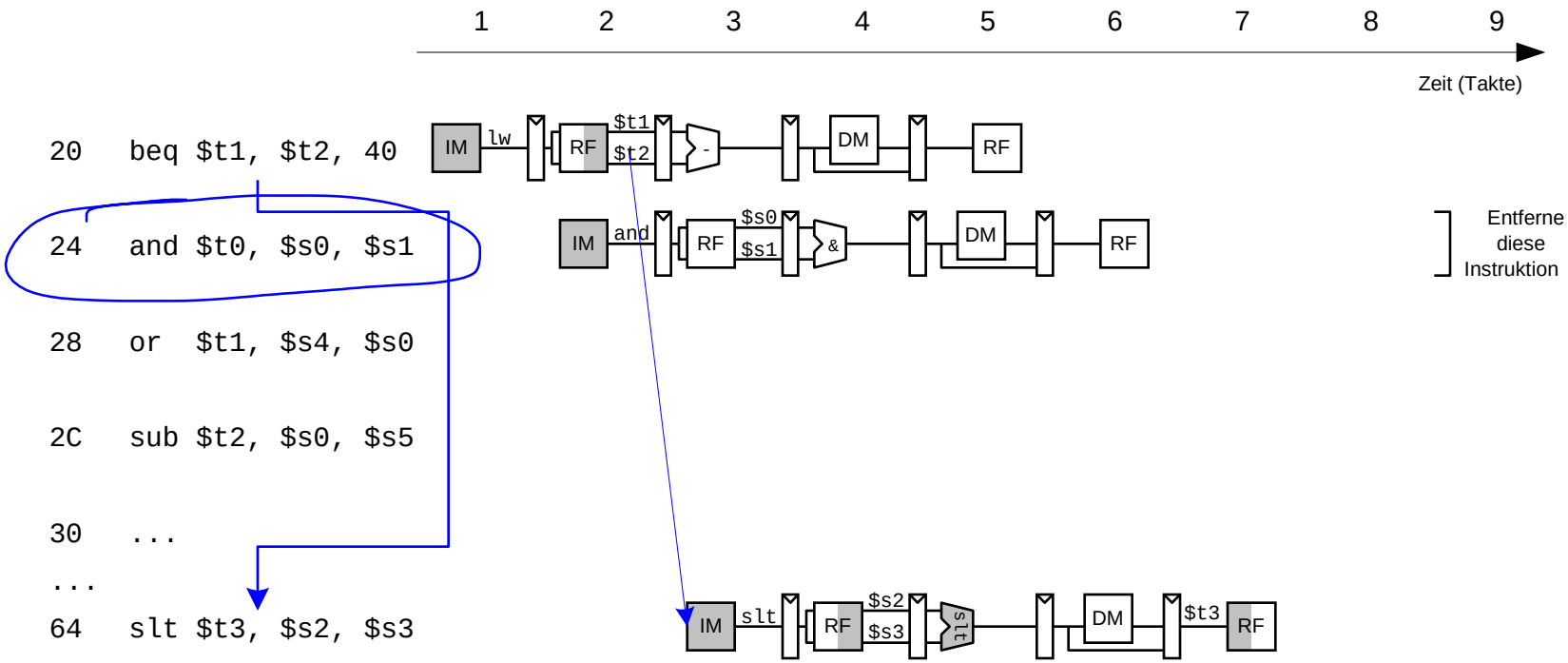
Control Hazards: Ursprüngliche Pipeline



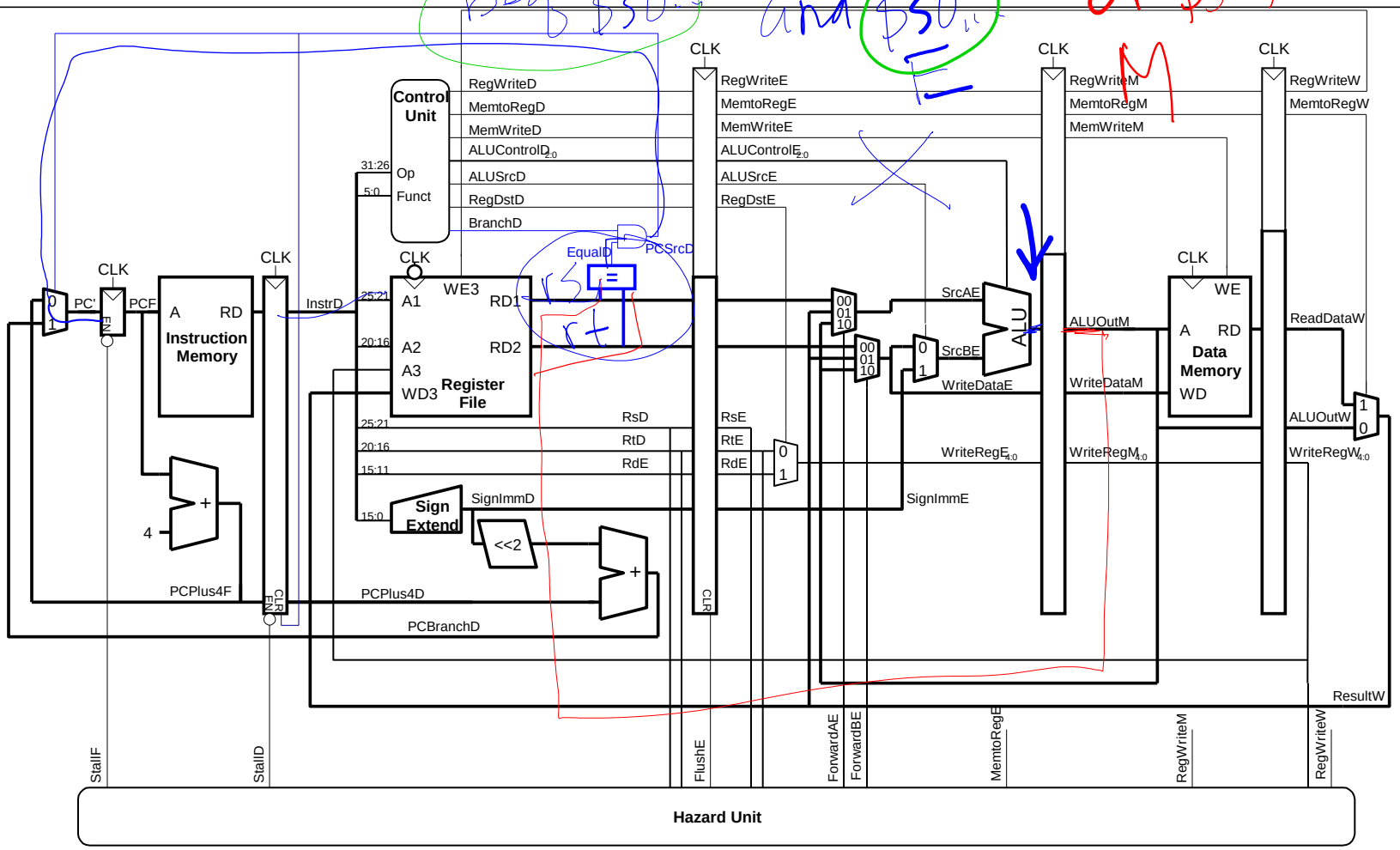
Beispiel: Control Hazards



Auflösen von Control Hazards durch frühere Sprungentscheidung



Control Hazards: Ansatz “Frühere Sprungentscheidung”



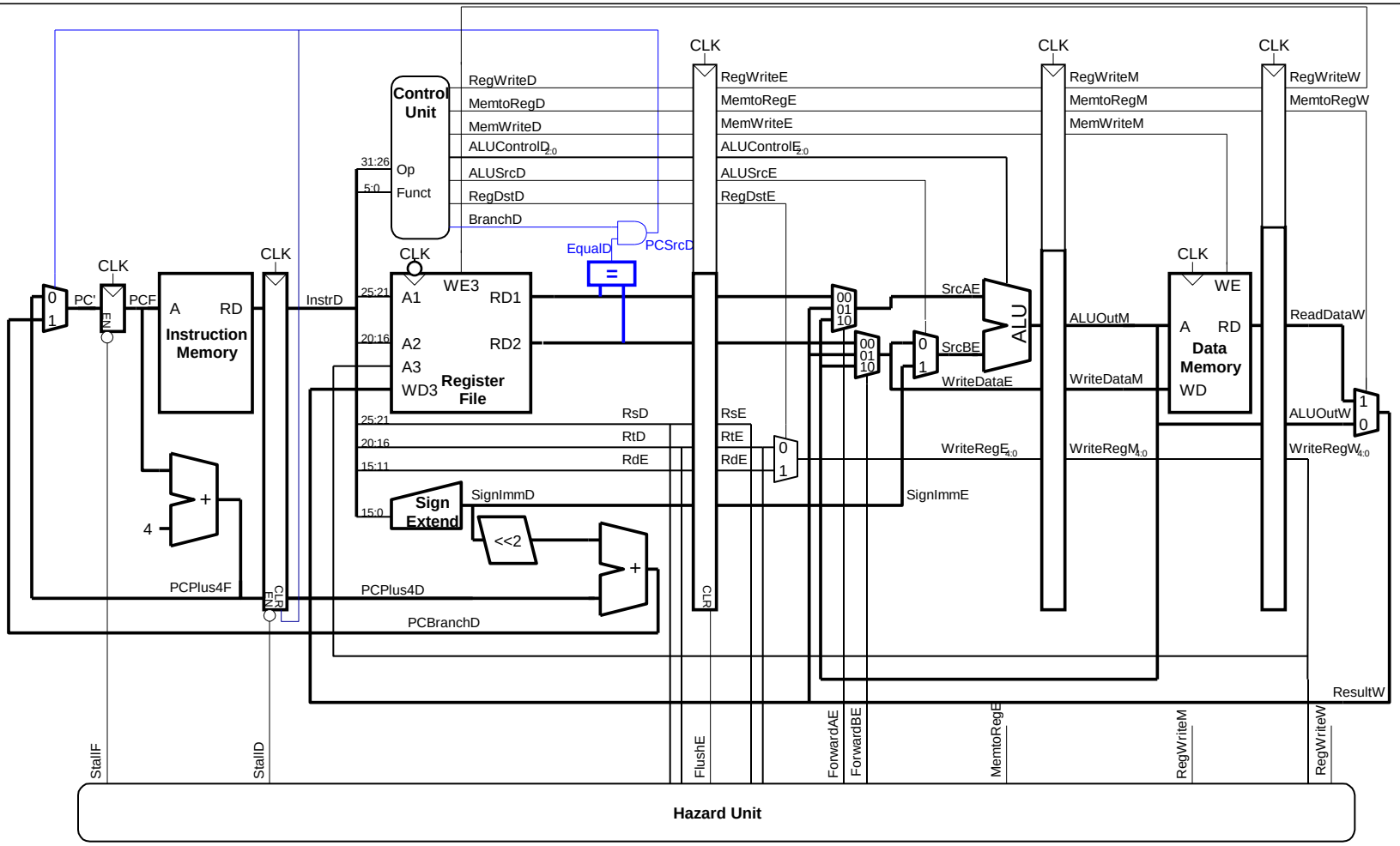
Control Hazards

TECHNISCHE UNIVERSITÄT DARMSTADT

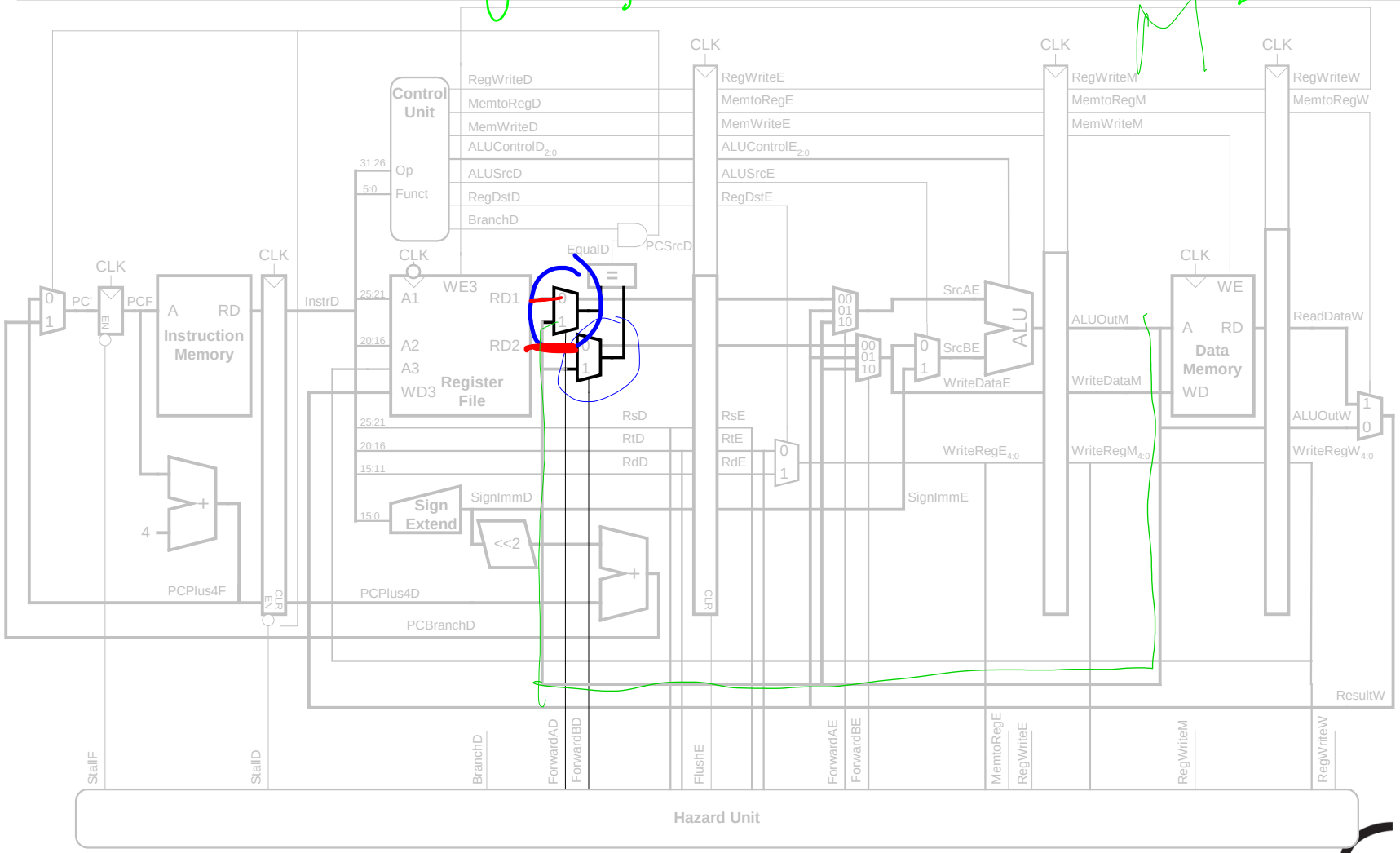
beq:

- Entscheidung zu Springen wird erst in vierter Stufe der Pipeline (M) getroffen
- Neue Instruktionen werden aber bereits geholt
  - Im einfachsten Fall: Von PC+4, +8, +12, …
- Falls zu springen ist, müssen diese Instruktionen aus der Pipeline entfernt werden
  - … das Programm wäre ja woanders (am Sprungziel) weitergegangen
  - "Spülen" (*flush*)
- Kosten eines solchen falsch vorhergesagten Sprunges:
  - Anzahl von zu entfernenden Instruktion falls Sprung genommen
  - Könnte reduziert werden, wenn Sprung in früherer Pipeline-Stufe entschieden würde

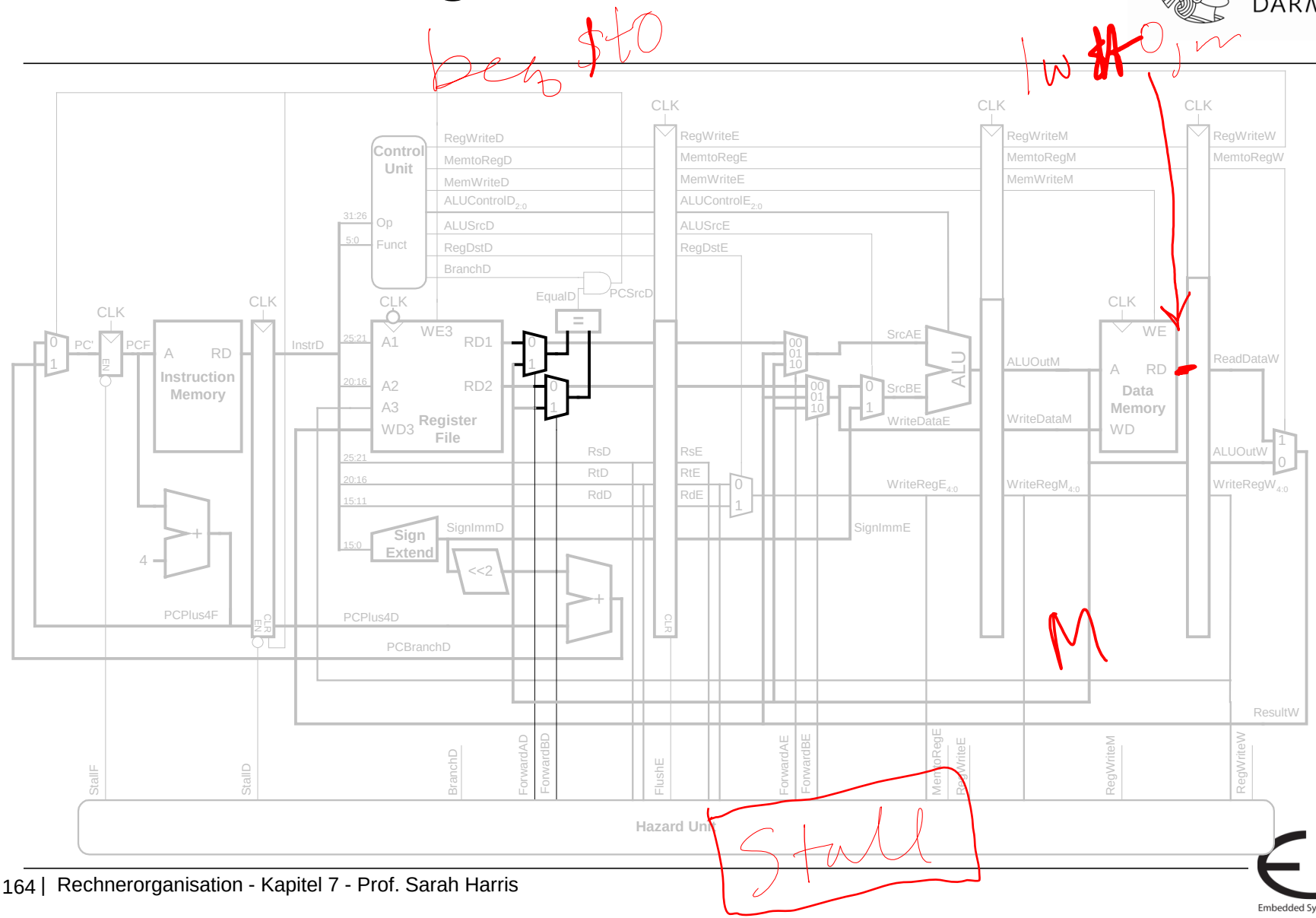
Control Hazards: Ansatz “Frühere Sprungentscheidung”



Berücksichtige neue Data Hazards



Berücksichtige neue Data Hazards



Frühe Sprungentscheidung: Benötigte Logik für Forwarding und Stalling

TECHNISCHE UNIVERSITÄT DARMSTADT

- Forwarding-Logik:

```
ForwardAD = RegWriteM AND (rsD==WriteRegM) AND (rsD !=0)
ForwardBD = RegWriteM AND (rtD==WriteRegM) AND (rtD !=0)
```

- Stalling-Logik:

```
branchstall =
        BranchD AND (
          RegWriteE AND(rsD==WriteRegE OR rtD==WriteRegE)
                        OR
          MemtoRegM AND(rsD==WriteRegM OR rtD==WriteRegM)
        )

StallF = StallD = FlushE = lwstall OR branchstall
```


Embedded Systems & Applications

Orthogonaler Ansatz: Sprungvorhersage

TECHNISCHE UNIVERSITÄT DARMSTADT

Versuche vorherzusagen, ob ein Sprung genommen wird

- Dann können Instruktionen von der richtigen Stelle geholt werden
- Rückwärtssprünge werden üblicherweise genommen (Schleifen!)
- Genauer: Für jeden Sprung Historie führen, ob er die letzten Male genommen wurde
  - … dann wird jetzt vermutlich auch wieder genommen

Eine gute Vorhersage reduziert die Zahl der Sprünge, die ein Flush der Pipeline erforderlich machen

Embedded Systems & Applications

Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

- Idealerweise wäre CPI = 1
- Manchmal treten aber Stalls auf (wegen Lade- und Verzweigungsbefehlen)
- SPECint 2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type
- Annahmen:
  - 40% der geladenen Daten werden gleich in der nächsten Instruktion gebraucht
  - 25% aller Verzweigungen werden falsch vorhergesagt
  - Alle Sprünge erzeugen eine zu entfernende (*flush*) Instruktion
- **Wie hoch ist der durchschnittliche CPI-Wert?**

Embedded Systems & Applications

Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

- SPECint 2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type

- Annahmen:
  - 40% der geladenen Daten werden gleich in der nächsten Instruktion gebraucht
  - 25% aller Verzweigungen werden falsch vorhergesagt
  - Alle Sprünge erzeugen eine zu entfernende (*flush*) Instruktion

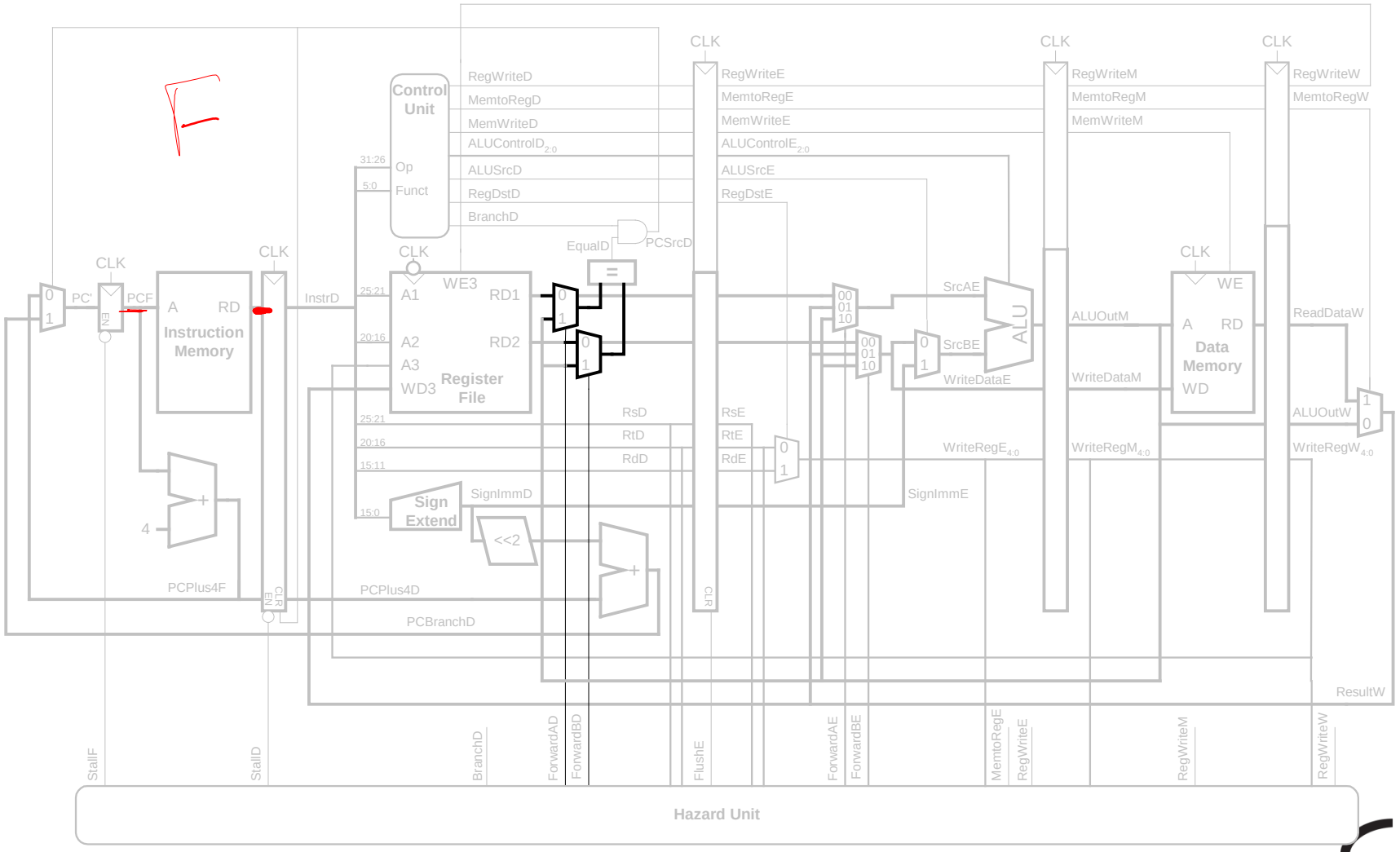
- **Wie hoch ist der durchschnittliche CPI-Wert?**
  - Lade/Verzweigungsinstruktionen haben CPI = 1 ohne Stall, = 2 mit Stall. Daher:
  - CPI_{lw} = 1 (0,6) + 2 (0,4) = 1,4
  - CPI_{beq} = 1 (0,75) + 2 (0,25) = 1,25
  - Also:

Durchschnittliche CPI = (0,25) (1,4) + (0,1) (1,0) + (0,11)(1,25) + (0,02) (2,0) + (0,52)(1,0)

= 1,15

Embedded Systems & Applications

MIPS Pipeline Prozessor



Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Kritischer Pfad des Pipelined-Prozessors:

$$
\begin{aligned}
T_c = \max \{ & \\
& t_{pcq} + t_{mem} + t_{setup}, && \text{Fetch} \\
& 2\,(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup}), && \text{Decode} \\
& t_{pcq} + t_{mux} + t_{mux} + t_{mux} + t_{ALU} + t_{setup}, && \text{Execute} \\
& t_{pcq} + t_{memwrite} + t_{setup}, && \text{Memory} \\
& 2\,(t_{pcq} + t_{mux} + t_{RFwrite}) \} && \text{Writeback}
\end{aligned}
$$

$T_c/2 = [$ ... $]$

$\therefore T_c = 2(\ldots)$

Embedded Systems & Applications

Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Element	Parameter	Verzögerung (ps)
Register Clock-to-Q	t_{pcq_PC}	30
Register Setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Speicher Lesen	t_{mem}	250
Registerfeld Lesen	t_{RFread}	150
Registerfeld Setup	$t_{RFsetup}$	20
Vergleich auf Gleichheit	t_{eq}	40
AND Gatter	t_{AND}	15
Speicher Schreiben	$T_{memwrite}$	220
Registerfeld Schreiben	$t_{RFwrite}$	100

$$T_c = 2\,(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup})$$
$$= 2\,[150 + 25 + 40 + 15 + 25 + 20]\ \text{ps} = 550\ \text{ps}$$

Embedded Systems & Applications

Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Führe Programm mit 100 Milliarden Instruktionen auf Pipelined-MIPS-Prozessor aus

- CPI = 1,15
- T_c = 550 ps

Ausführungszeit = (# Instruktionen) × CPI × T_c

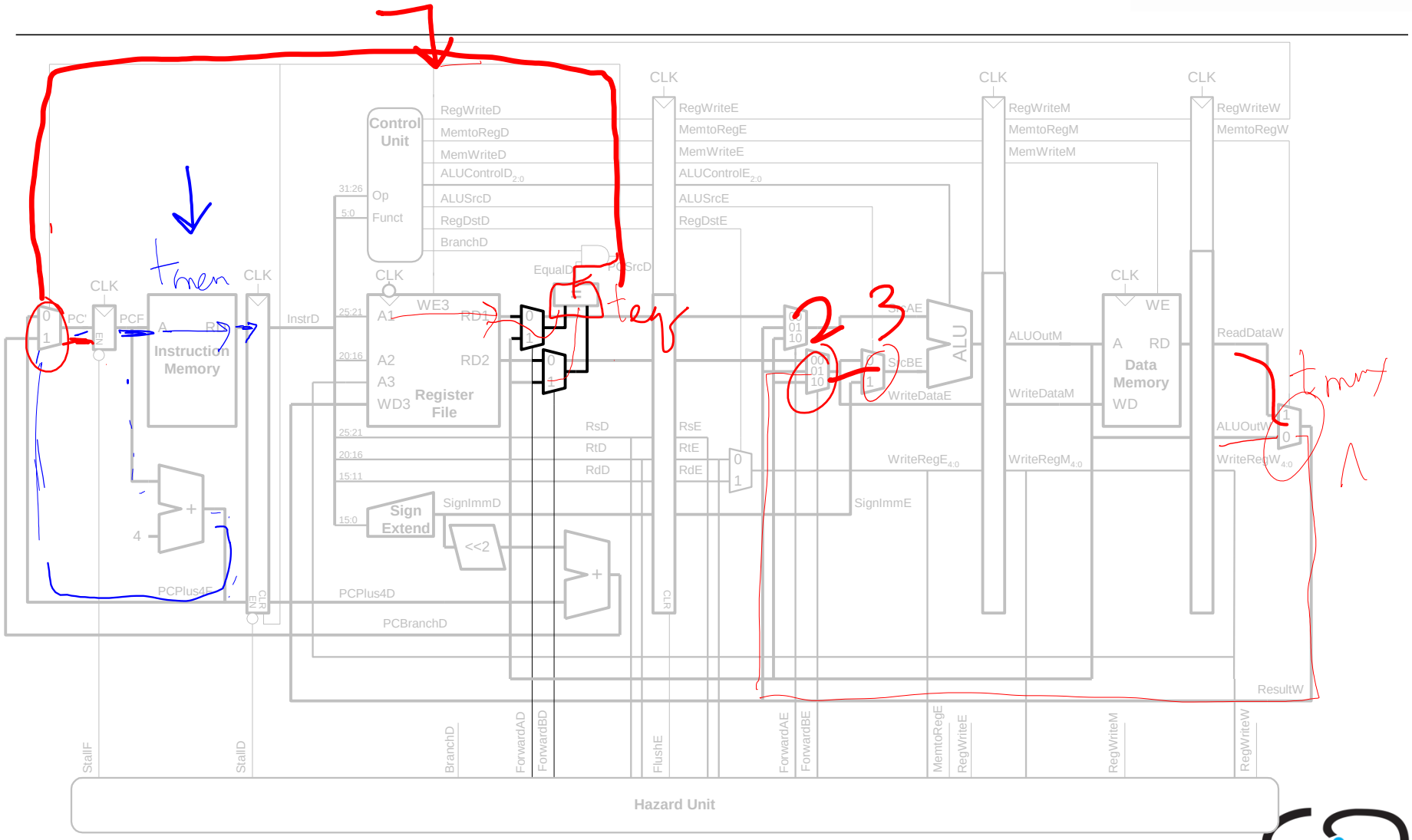
= (100×10^9) (1,15) (550×10^{-12})

= 63 Sekunden

Prozessor	Ausführungszeit (Sekunden)	Beschleunigungsfaktor (im Vergleich zu Ein-Takt-CPU)
Ein-Takt	95	1,00
Mehrtakt	133	0,71
Pipelined	63	1,51

Embedded Systems & Applications

Berücksichtige neue Data Hazards



Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

- SPECint 2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type

- Annahmen:
  - 40% der geladenen Daten werden gleich in der nächsten Instruktion gebraucht
  - 25% aller Verzweigungen werden falsch vorhergesagt
  - Alle Sprünge erzeugen eine zu entfernende (*flush*) Instruktion

- **Wie hoch ist der durchschnittliche CPI-Wert?**
  - Lade/Verzweigungsinstruktionen haben CPI = 1 ohne Stall, = 2 mit Stall. Daher:
  - CPI_{lw} = 1 (0,6) + 2 (0,4) = 1,4
  - CPI_{beq} = 1 (0,75) + 2 (0,25) = 1,25
  - Also:

Durchschnittliche CPI = (0,25) (1,4) + (0,1) (1,0) + (0,11)(1,25) + (0,02) (2,0) + (0,52)(1,0)

= 1,15

Embedded Systems & Applications

Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Kritischer Pfad des Pipelined-Prozessors:

$$
\begin{aligned}
T_c = \max \{ & \\
& t_{pcq} + t_{mem} + t_{setup}, && \text{Fetch} \\
& 2\,(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup}), && \text{Decode} \\
& t_{pcq} + t_{mux} + t_{mux} + t_{mux} + t_{ALU} + t_{setup}, && \text{Execute} \\
& t_{pcq} + t_{memwrite} + t_{setup}, && \text{Memory} \\
& 2\,(t_{pcq} + t_{mux} + t_{RFwrite}) \} && \text{Writeback}
\end{aligned}
$$

Fehlt im Buch

Embedded Systems & Applications

MIPS Pipeline Prozessor

TECHNISCHE UNIVERSITÄT DARMSTADT

T_c = max {

$t_{pcq} + t_{mem} + t_{setup}$,

$2\,(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup})$,

$t_{pcq} + t_{mux} + t_{mux} + t_{mux} + t_{ALU} + t_{setup}$,

$t_{pcq} + t_{memwrite} + t_{setup}$,

$2\,(t_{pcq} + t_{mux} + t_{RFwrite})$ }

Fetch

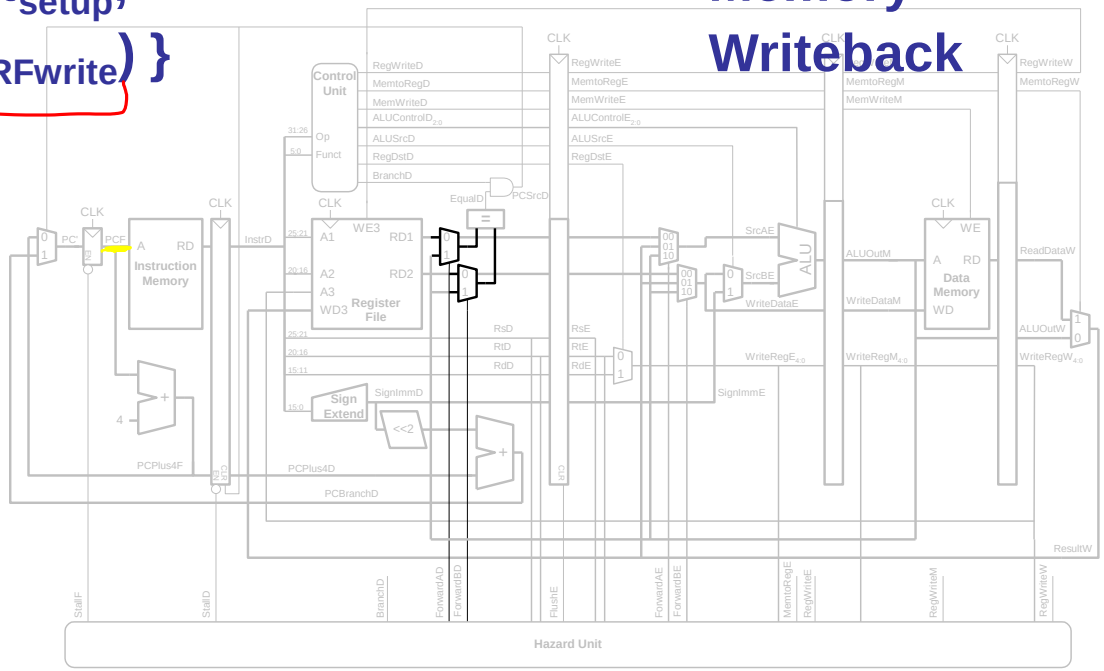
Decode

Execute

Memory

Writeback

t_{setup}



Embedded Systems & Applications

Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Element	Parameter	Verzögerung (ps)
Register Clock-to-Q	t_{pcq_PC}	30
Register Setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Speicher Lesen	t_{mem}	250
Registerfeld Lesen	t_{RFread}	150
Registerfeld Setup	$t_{RFsetup}$	20
Vergleich auf Gleichheit	t_{eq}	40
AND Gatter	t_{AND}	15
Speicher Schreiben	$T_{memwrite}$	220
Registerfeld Schreiben	$t_{RFwrite}$	100

$$T_c = 2\,(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup})$$
$$= 2\,[150 + 25 + 40 + 15 + 25 + 20]\ \text{ps} = 550\ \text{ps}$$

Decode

Embedded Systems & Applications

Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Führe Programm mit 100 Milliarden Instruktionen auf Pipelined-MIPS-Prozessor aus

- CPI = 1,15
- T_c = 550 ps

Ausführungszeit = (# Instruktionen) × CPI × T_c

= (100×10^9) (1,15) (550×10^{-12})

= 63 Sekunden

Prozessor	Ausführungszeit (Sekunden)	Beschleunigungsfaktor (im Vergleich zu Ein-Takt-CPU)
Ein-Takt	95	1,00
Mehrtakt	133	0,71
Pipelined	63	1,51

Embedded Systems & Applications

Wiederholung: Ausnahmebehandlung (*exceptions*)

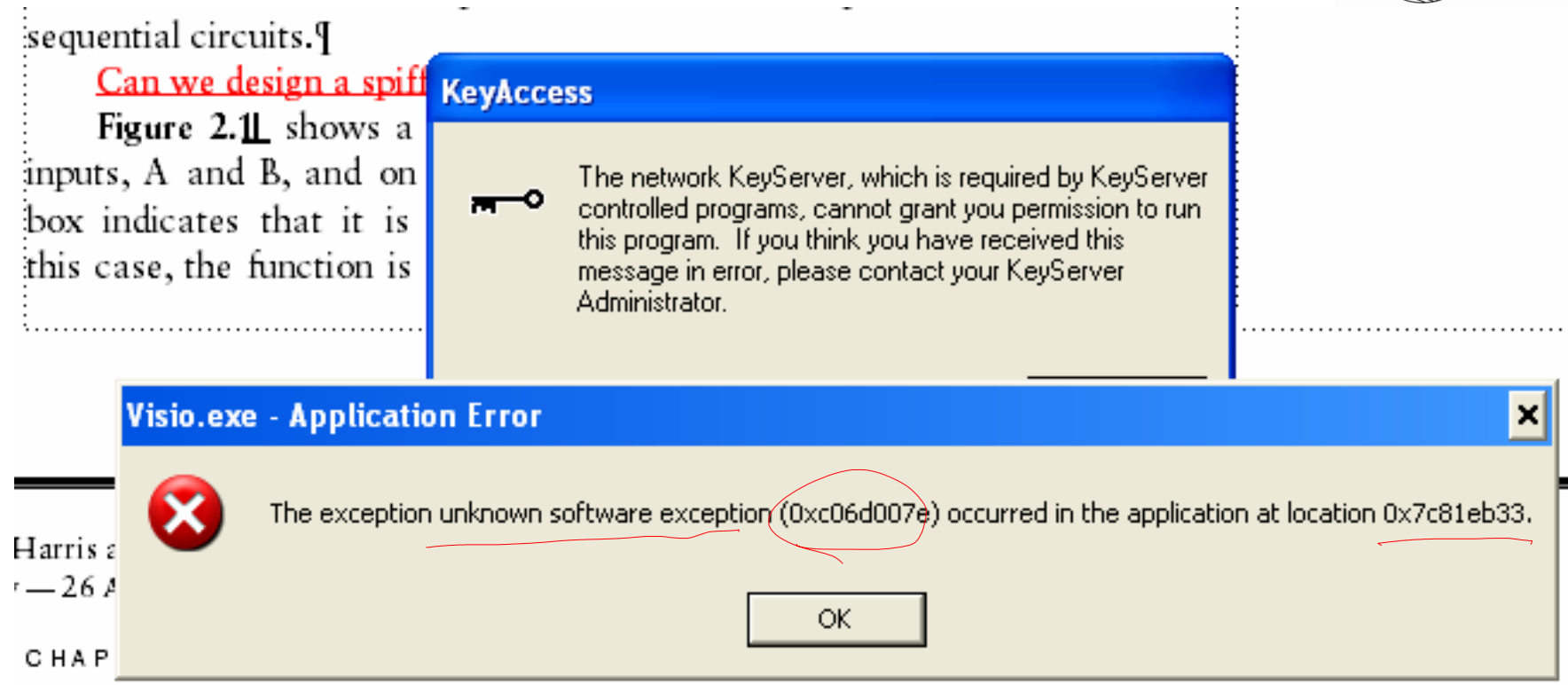
TECHNISCHE UNIVERSITÄT DARMSTADT

Außerplanmäßiger Aufruf der Ausnahmebehandlungsroutine

- Verursacht durch:
  - Hardware, auch genannt *Interrupt*, z.B. Tastatur, Netzwerk, …
  - Software, auch genannt *Traps*, z.B. unbekannte Instruktion, Überlauf, Teilen-durch-Null, …

Embedded Systems & Applications

Beispiel für Ausnahme



Wiederholung: Ausnahmebehandlung (*exceptions*)

TECHNISCHE UNIVERSITÄT DARMSTADT

Beim Auftreten einer Ausnahme:

- Abspeichern der Ursache für Ausnahme im **`Cause`** Register
- Sprung zu Ausnahmebehandlungsroutine bei 0x80000180
- Rückkehr zum Programm (über **`EPC`** Register)

Register für Ausnahmebehandlung

TECHNISCHE UNIVERSITÄT DARMSTADT

Nicht Teil des regulären MIPS Registersfelds

- **`Cause`**
  - Speichert die Ursache der Ausnahme
  - Koprozessor 0, Register 13
- **`EPC`** (Exception PC)
  - Speichert den PC-Stand, an dem die Aufnahme auftrat
  - Koprozessor 0, Register 14

PC+4

Register für Ausnahmebehandlung

TECHNISCHE UNIVERSITÄT DARMSTADT

Wie man **`Cause`** und **`EPC`** im Prozessor liest:

- Befehl: “Move from Coprocessor 0”
  - `mfc0 $t0, Cause`
  - Überträgt aktuellen Wert von `Cause` nach `$t0`

mfc0

010000	00000	`$t0` (8)	`Cause` (13)	00000000000
31:26	25:21	20:16	15:11	10:0

Embedded Systems & Applications

Auswahl von Ausnahmeursachen

TECHNISCHE UNIVERSITÄT DARMSTADT

Ausnahme	Cause (Ursache)
Hardware Interrupt	0x00000000
System Call	0x00000020
Breakpoint / Division durch 0	0x00000024
Unbekannte Instruktion	0x00000028
Arithmetischer Überlauf	0x00000030

Ziel: Erweitere den Mehrtaktprozessor um Behandlung der letzten beiden Ausnahmen

Embedded Systems & Applications

Ausnahmebehandlung (*exceptions*)

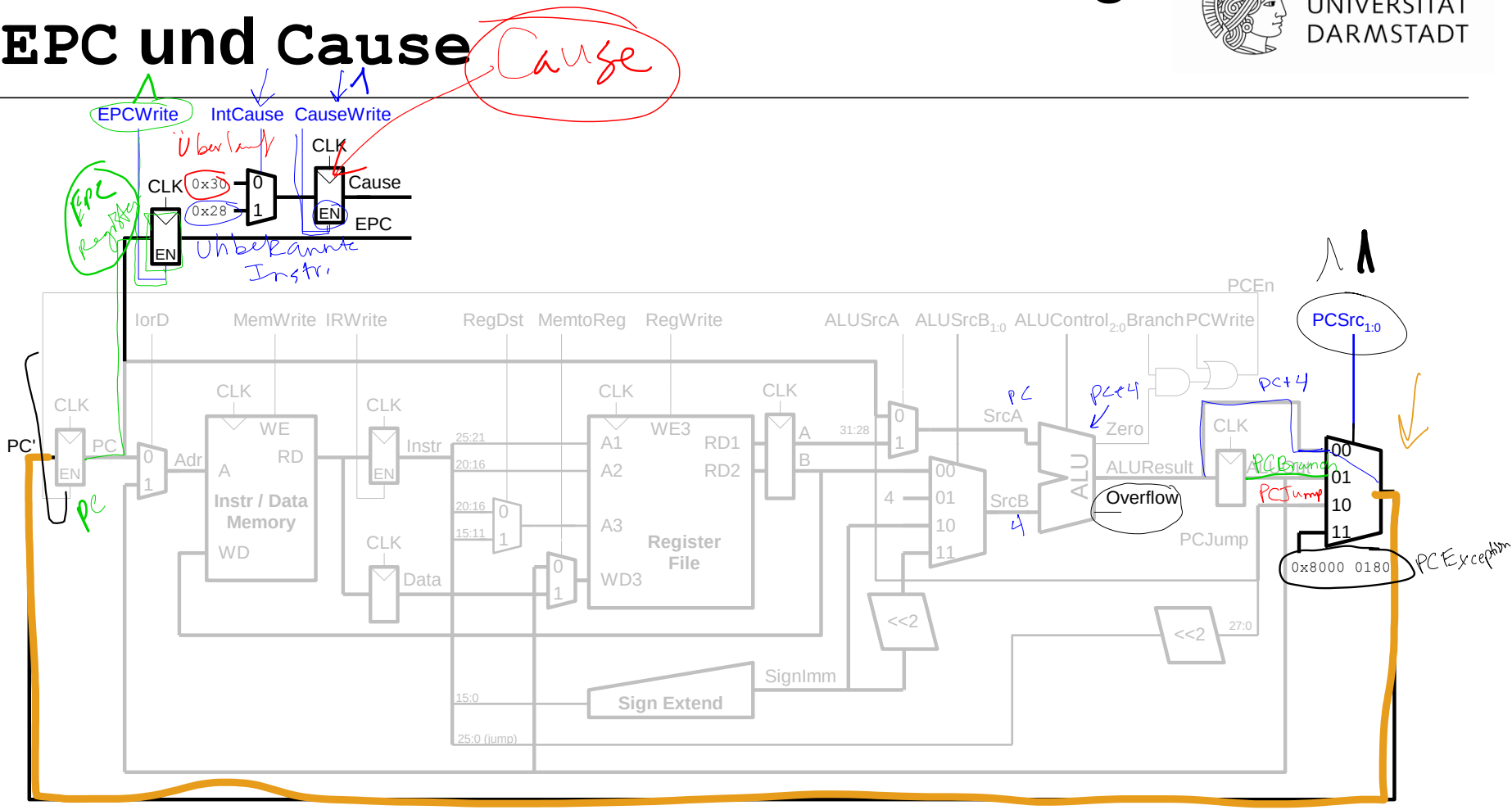
TECHNISCHE UNIVERSITÄT DARMSTADT

Beim Auftreten einer Ausnahme:

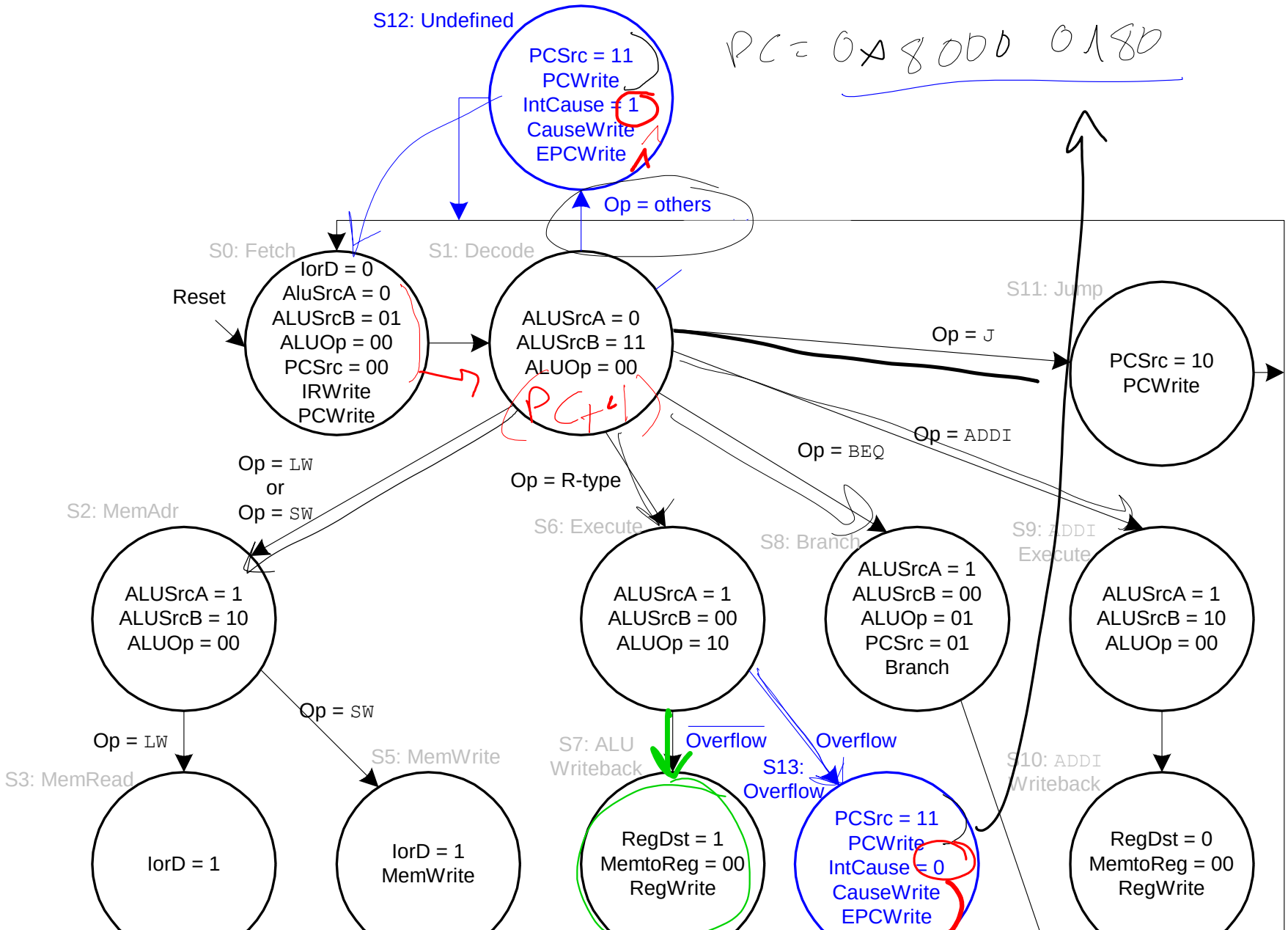
- **Abspeichern der Ursache für Ausnahme im `Cause` Register**
- **Sprung zu Ausnahmebehandlungsroutine bei 0x80000180**
- Rückkehr zum Programm (über `EPC` Register)

Embedded Systems & Applications

Hardware für Ausnahmebehandlung: EPC und Cause



snahmen


S12: Undefined
PCSrc = 11
PCWrite
IntCause = 1
CauseWrite
EPCWrite
PC = 0x8000 0180
Op = others
S0: Fetch
IorD = 0
AluSrcA = 0
ALUSrcB = 01
ALUOp = 00
PCSrc = 00
IRWrite
PCWrite
Reset
S1: Decode
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(PC+4)
S11: Jump
Op = J
PCSrc = 10
PCWrite
Op = ADDI
Op = BEQ
Op = LW
or
Op = SW
Op = R-type
S2: MemAdr
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
S6: Execute
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
S8: Branch
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCSrc = 01
Branch
S9: ADDI
Execute
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Op = SW
Op = LW
S5: MemWrite
S3: MemRead
IorD = 1
IorD = 1
MemWrite
S7: ALU
Writeback
Overflow
Overflow
S13:
Overflow
RegDst = 1
MemtoReg = 00
RegWrite
PCSrc = 11
PCWrite
IntCause = 0
CauseWrite
EPCWrite
S10: ADDI
Writeback
RegDst = 0
MemtoReg = 00
RegWrite

Ausnahmebehandlung (*exceptions*)

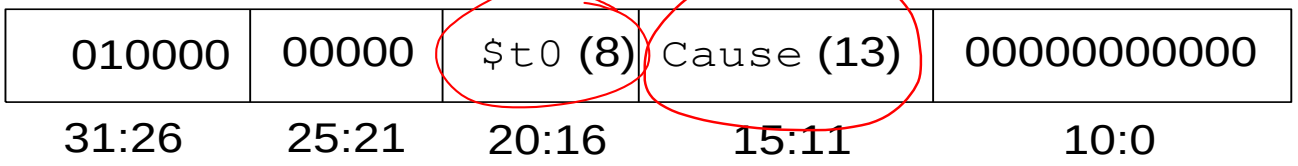
TECHNISCHE UNIVERSITÄT DARMSTADT

Beim Auftreten einer Ausnahme:

- Abspeichern der Ursache für Ausnahme im `Cause` Register
- Sprung zu Ausnahmebehandlungsroutine bei 0x80000180
- **Rückkehr zum Programm (über `EPC` Register):**

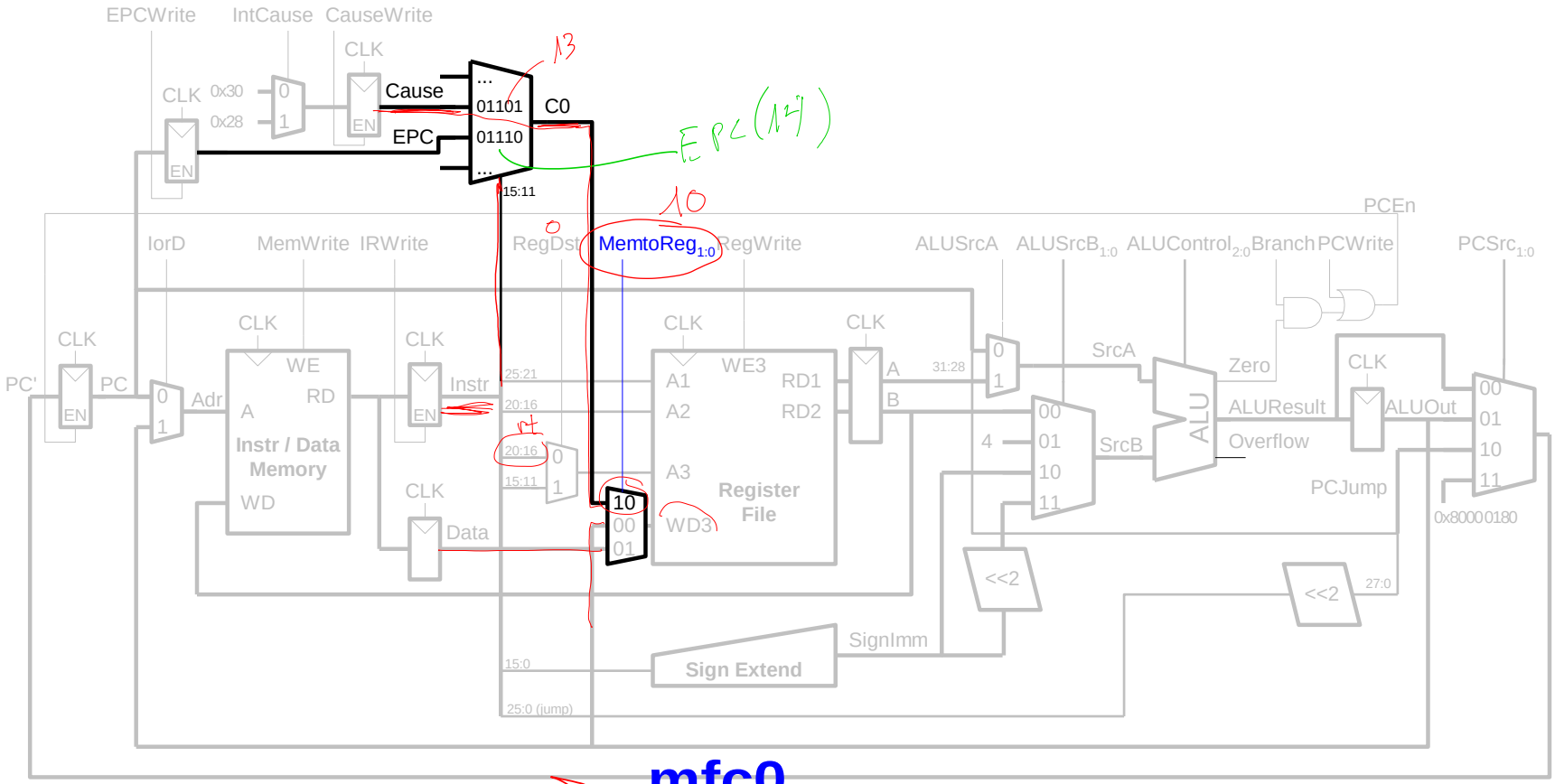
```
mfc0 $t0, Cause
...
mfc0 $k0, EPC
jr $k0
```

mfc0



Embedded Systems & Applications

Hardware für Ausnahmebehandlung: `mfc0`



mfc0

010000	00000	`$t0` (8)	`Cause` (13)	00000000000
31:26	25:21	20:16	15:11	10:0

Steuerwerk-FSM erweitert um Ausnahmen