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

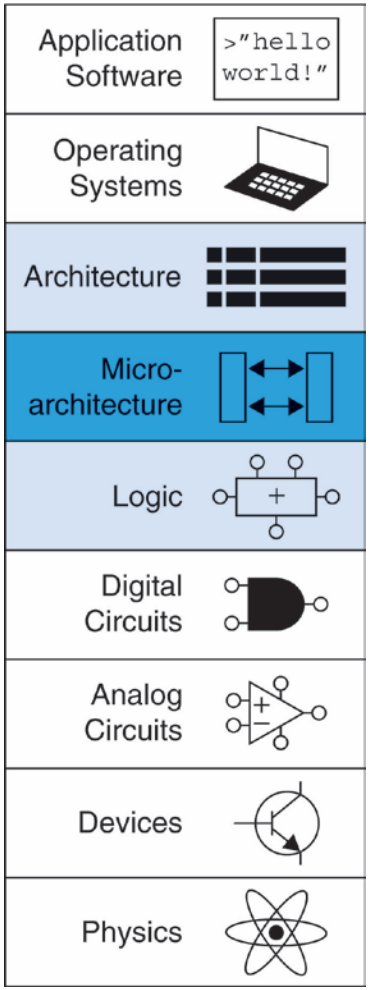
- Mikroarchitektur

  Hardware-Implementierung einer Architektur

- Prozessor:

  Datenpfad: funktionale Blöcke

  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

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

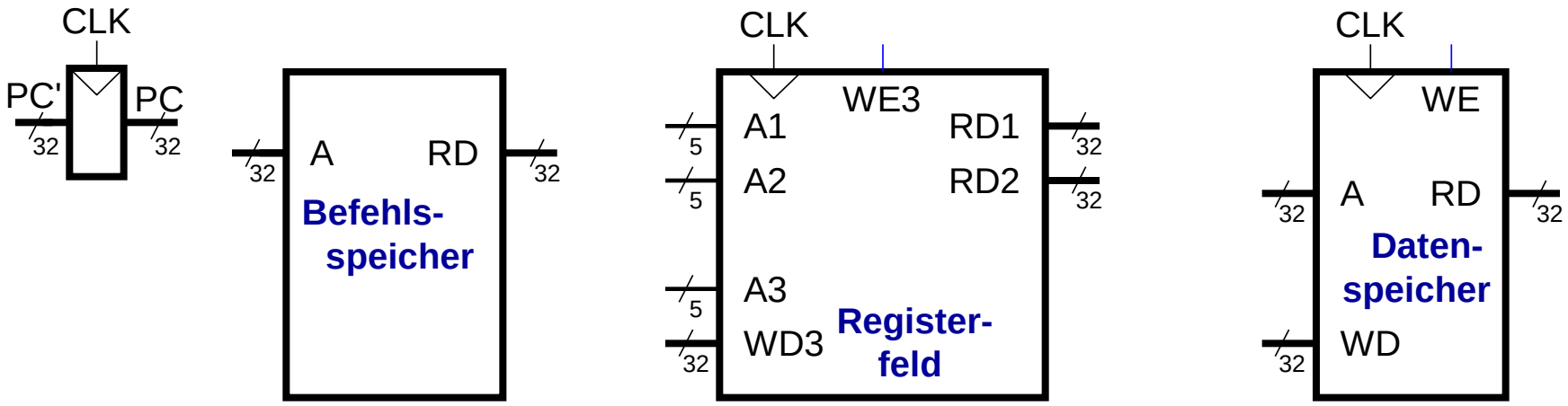
- Für den Programmierer zugänglich

Bestimmen vollständigen Zustand der Architektur

- PC
- 32 Register
- Speicher

Embedded Systems & Applications

Elemente des MIPS Architekturzustands

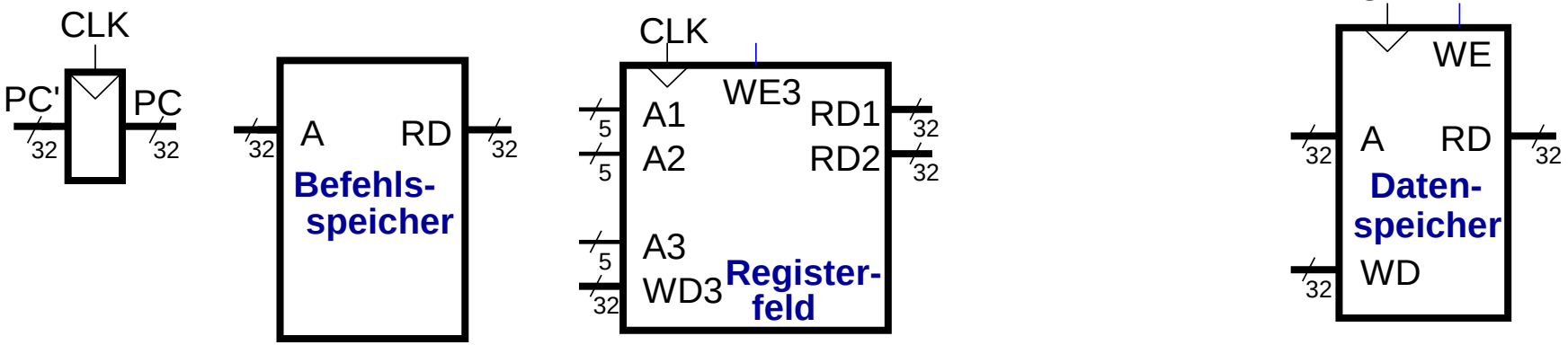


Ein-Takt MIPS Prozessor

TECHNISCHE UNIVERSITÄT DARMSTADT

- Datenpfad
- Steuerwerk

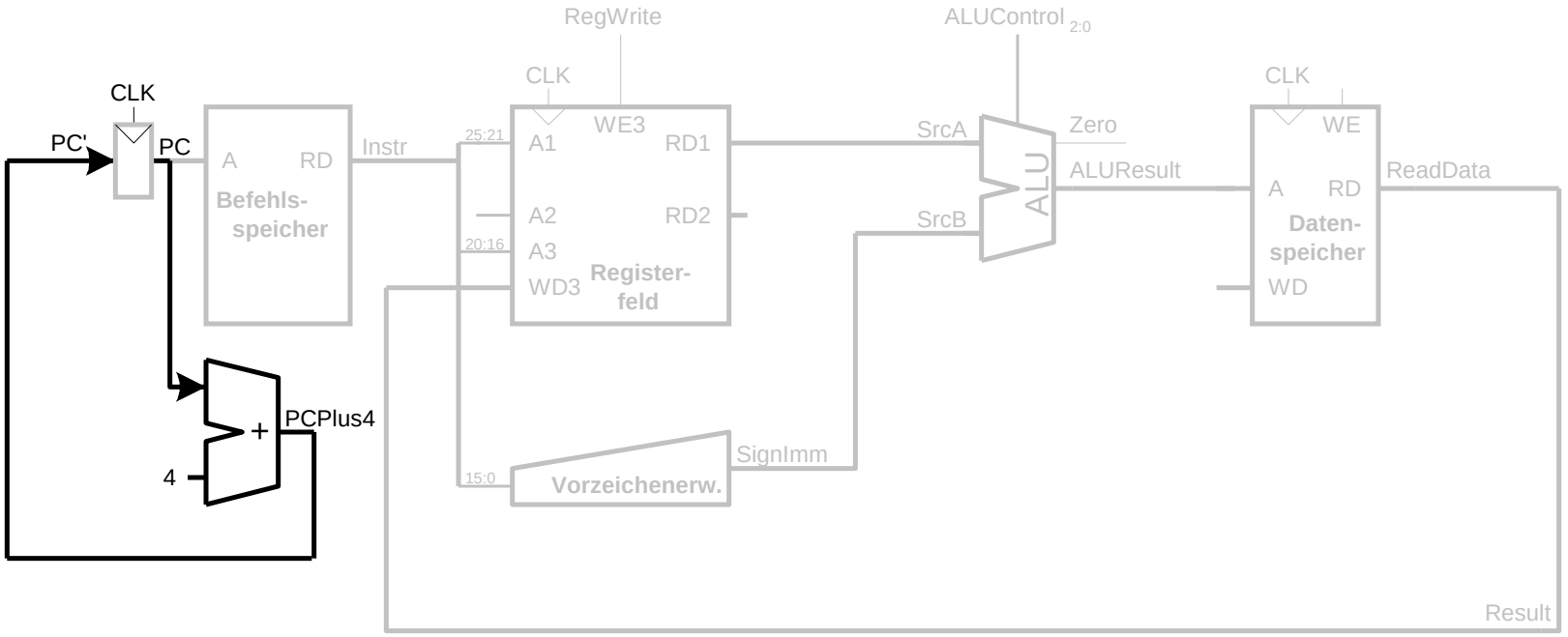
Datenpfad: `lw`




```
lw $t0, 16($s0)
lw  rt, imm(rs)
```

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

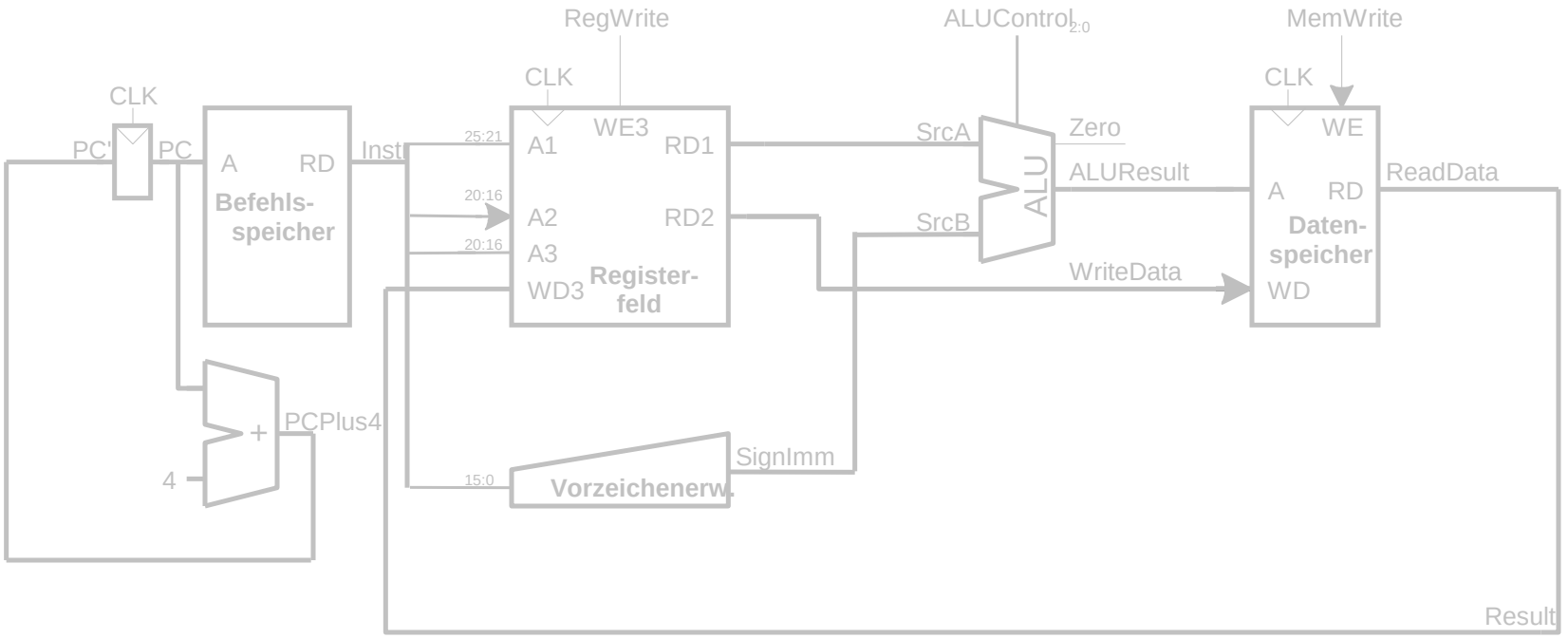
Ein-Takt Datenpfad : sw




```
sw $s2, 16($t7)
sw  rt, imm(rs)
```

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

Ein-Takt Datenpfad: R-Typ

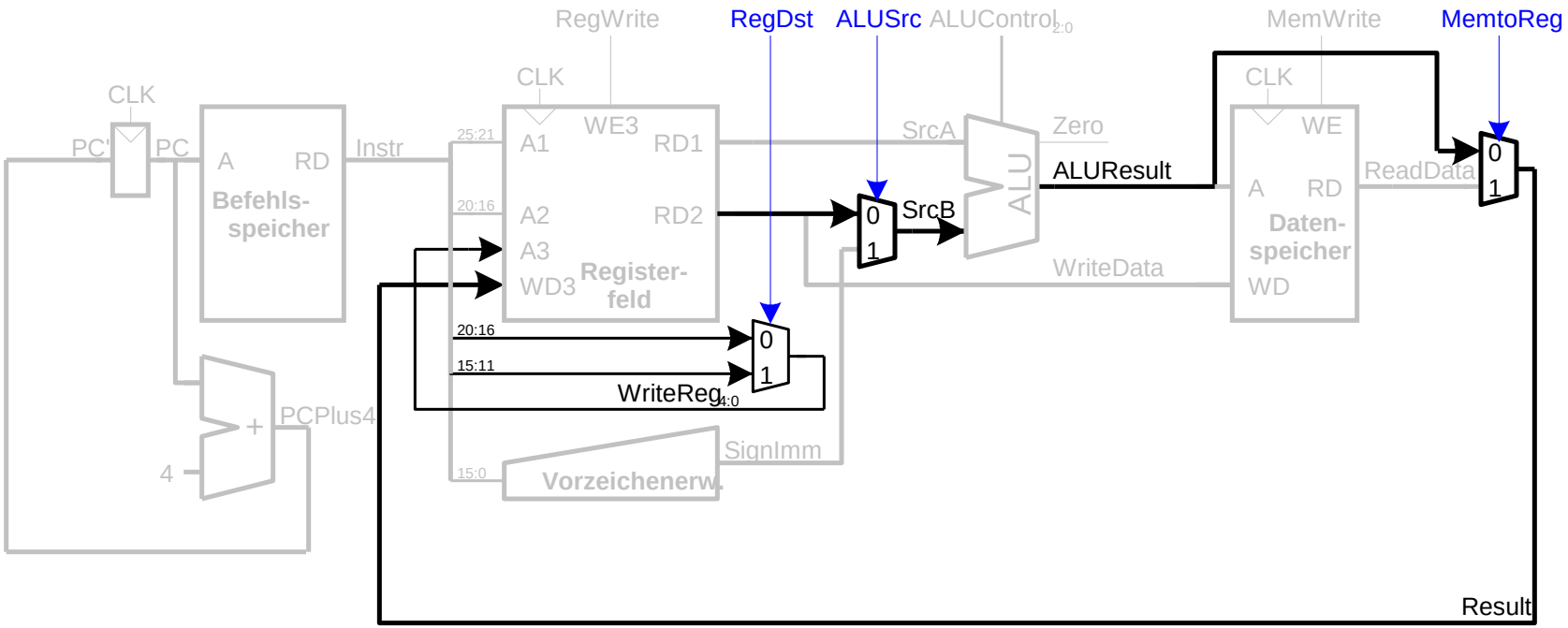


`add $s1,$t2,$t3`

`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: beq



`beq $s3,$s4,Loop`

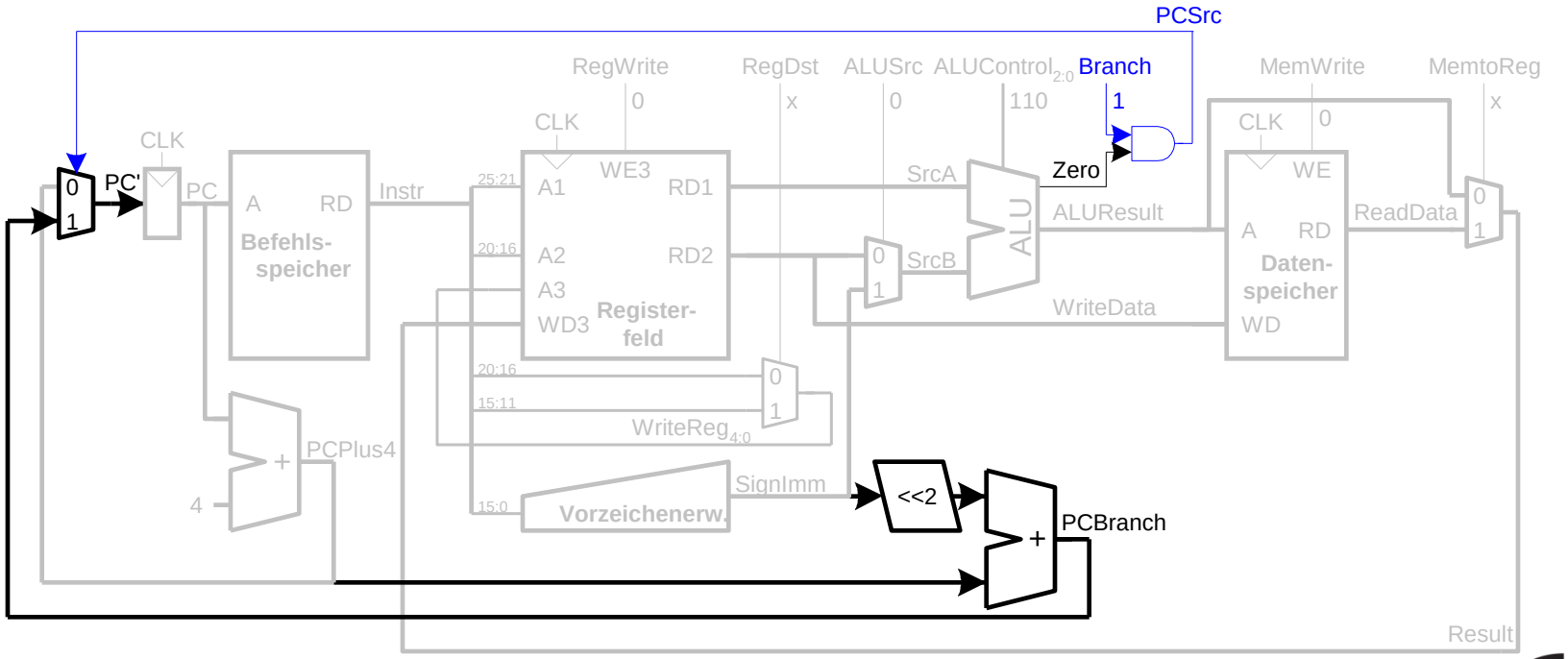
`beq rs, rt, imm`

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

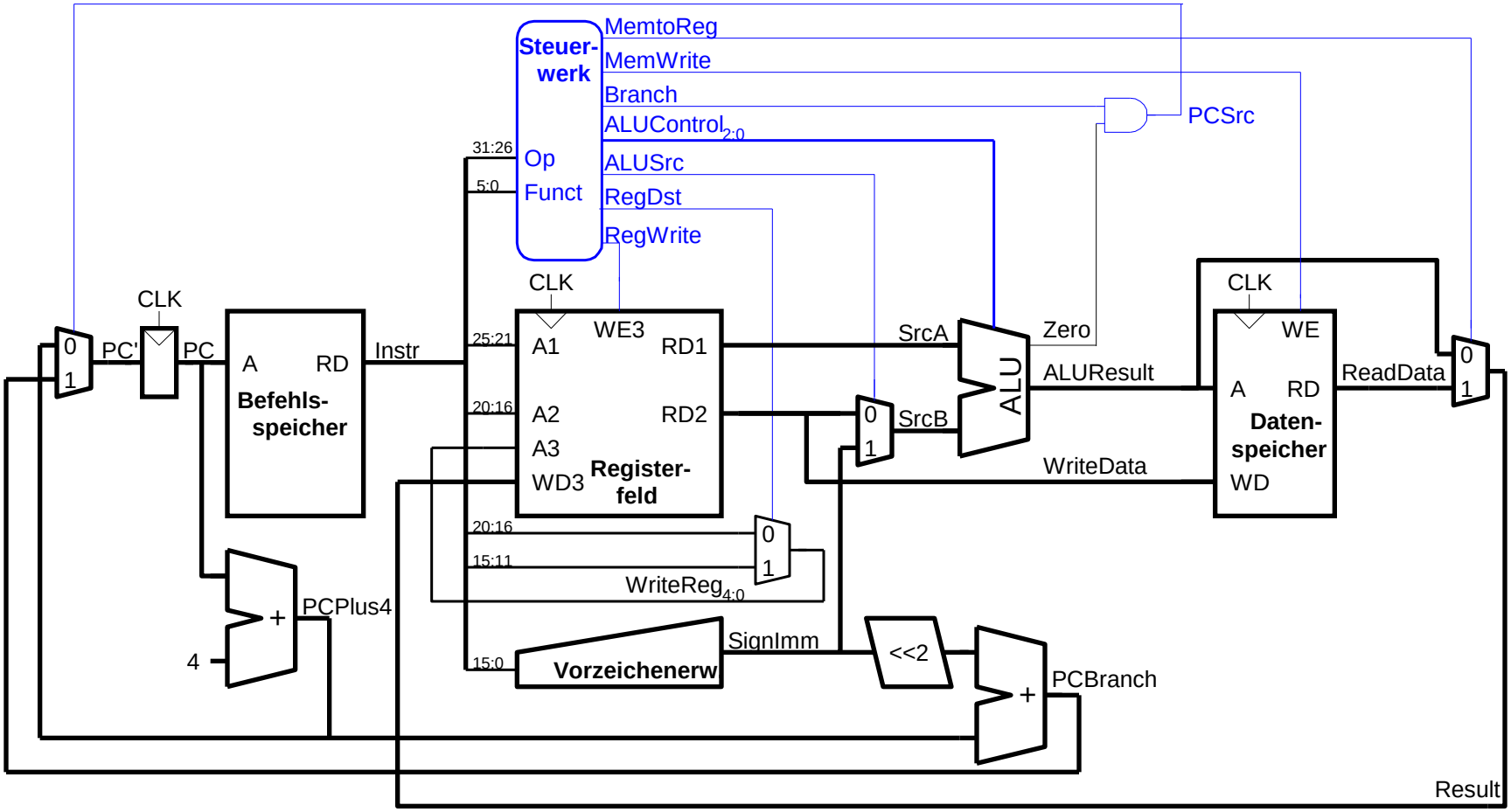
Ein-Takt Datenpfad: beq

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

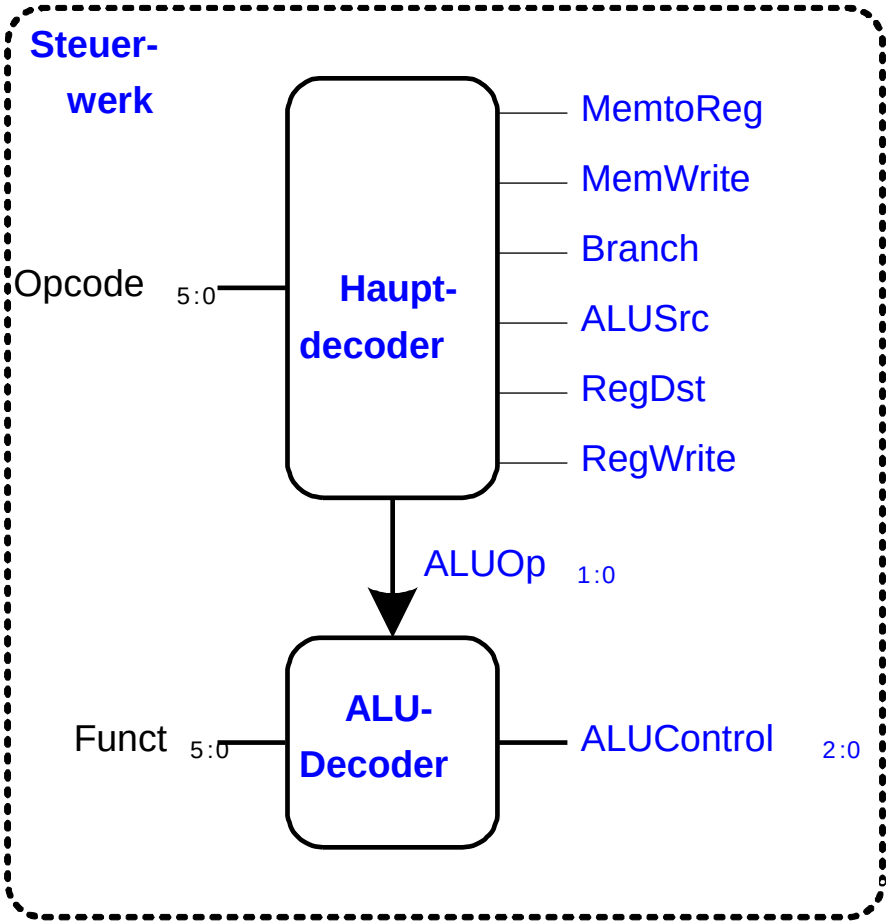
  BTA = (vorzeichenerweiterter Direktwert << 2) + (PC+4)



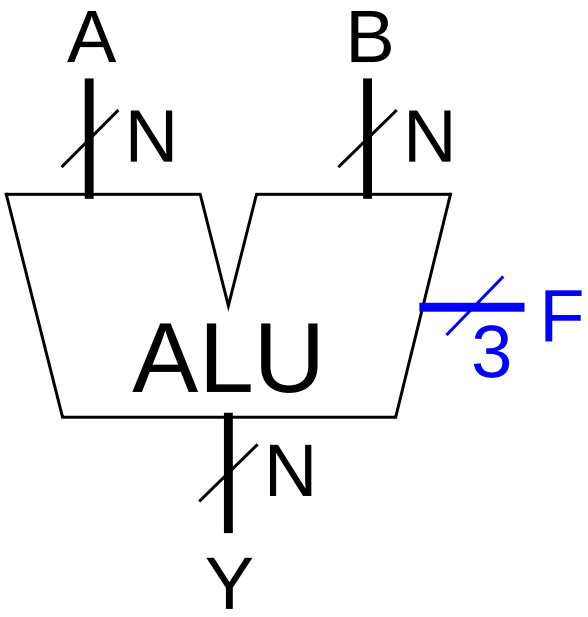
Vollständiger Ein-Takt-Prozessor



Steuerwerk

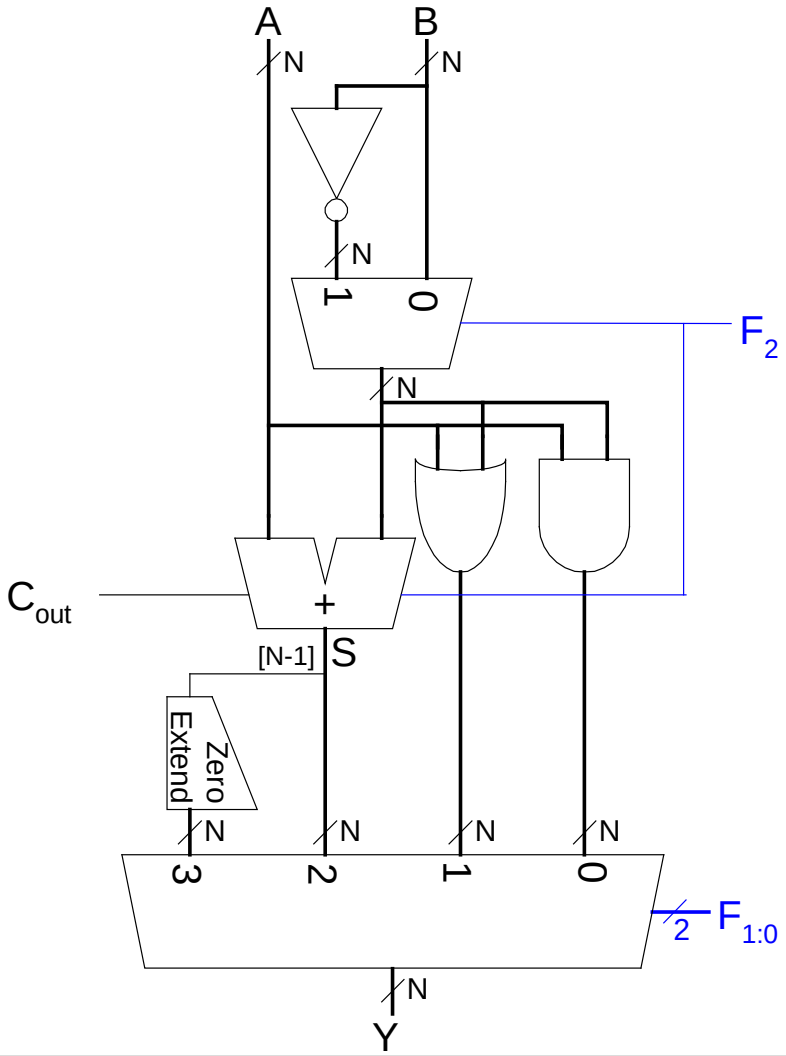

Steuer-
werk
Opcode
5:0
Haupt-
decoder
MemtoReg
MemWrite
Branch
ALUSrc
RegDst
RegWrite
ALUOp
1:0
Funct
5:0
ALU-
Decoder
ALUControl
2:0

Zur Erinnerung: ALU



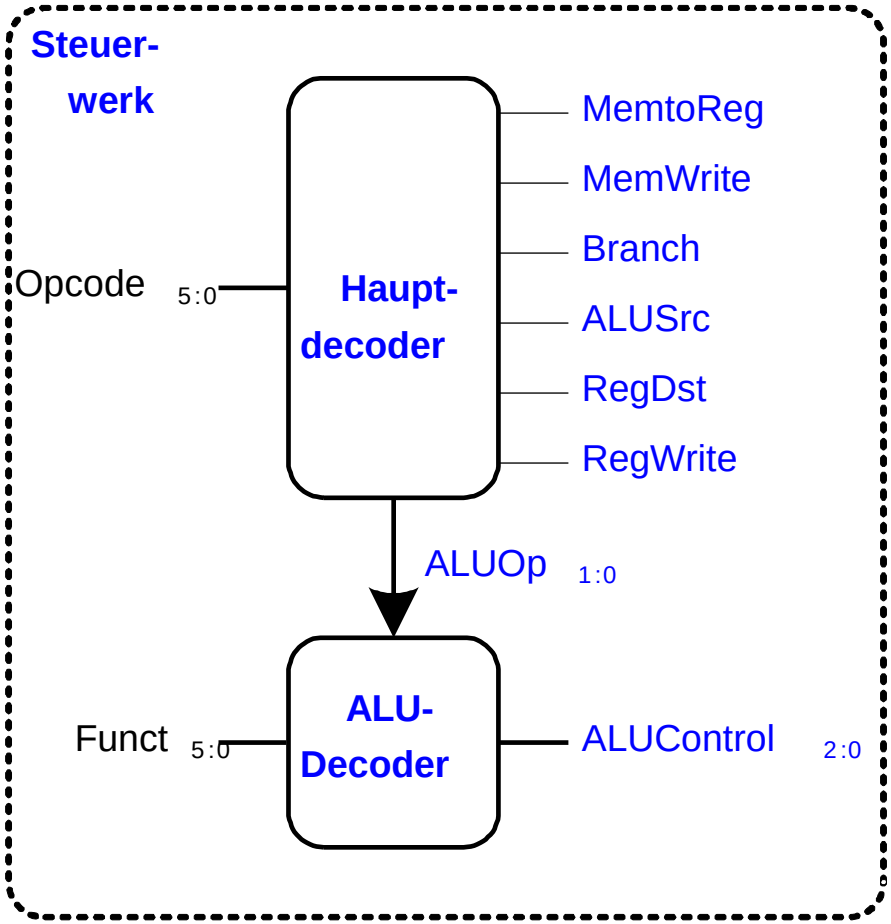
$F_{2:0}$	Funktion
000	A & B
001	A | B
010	A + B
011	unbenutzt
100	A & ~B
101	A | ~B
110	A - B
111	SLT

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

Steuerwerk: ALU-Decoder

TECHNISCHE UNIVERSITÄT DARMSTADT

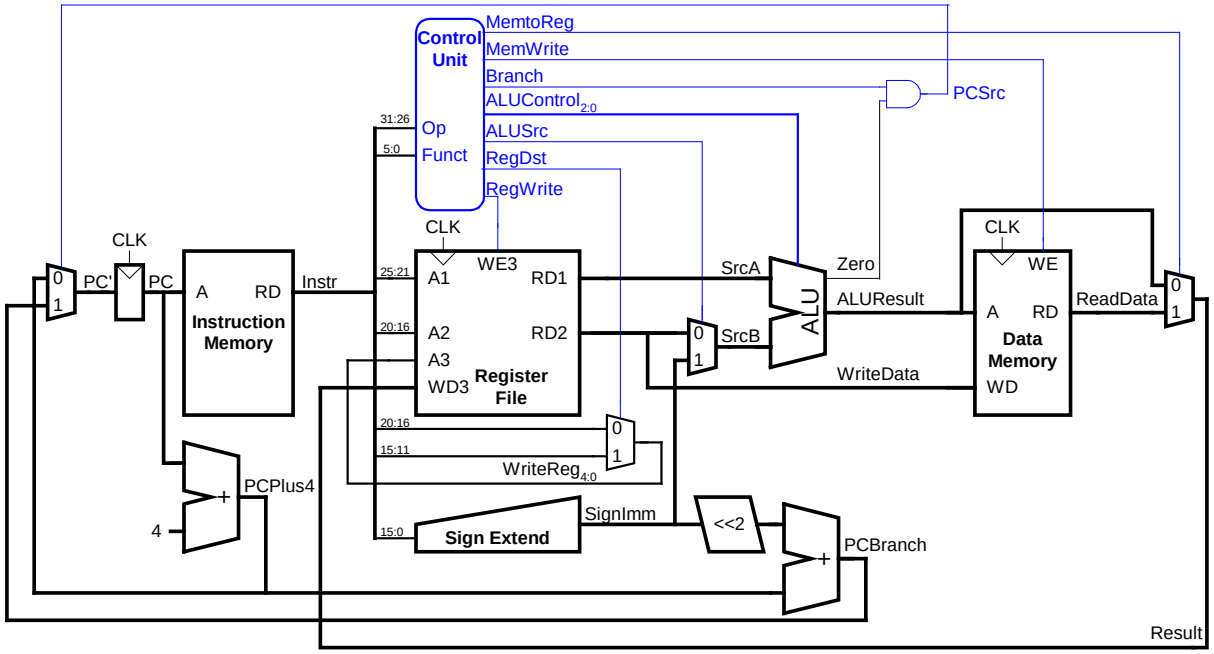
$ALUOp_{1:0}$	Bedeutung
00	Addiere
01	Subtrahiere
10	Werte Funct-Feld aus
11	unbenutzt

$ALUOp_{1:0}$	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)

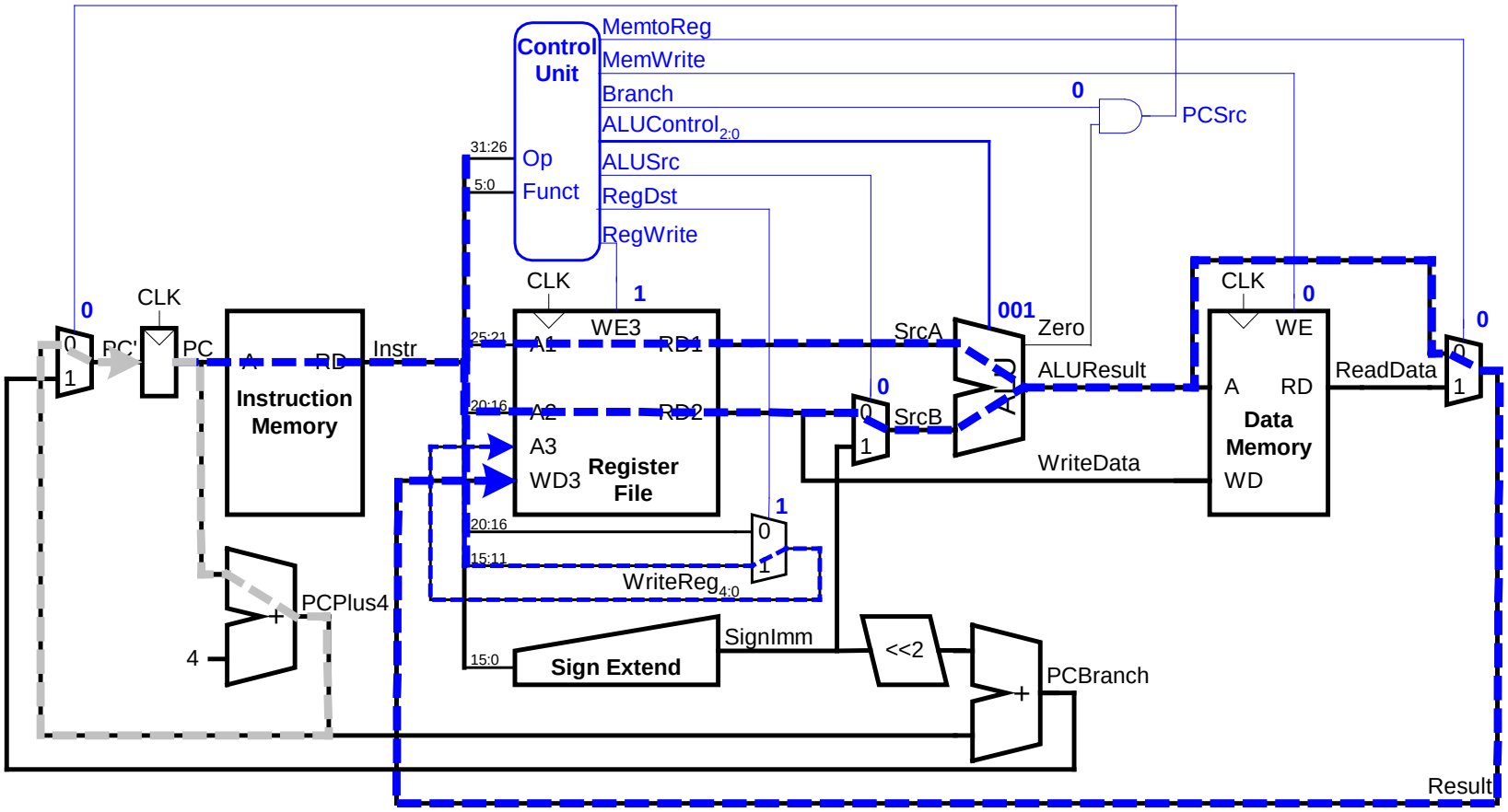
Embedded Systems & Applications

Steuerwerk: Hauptdecoder

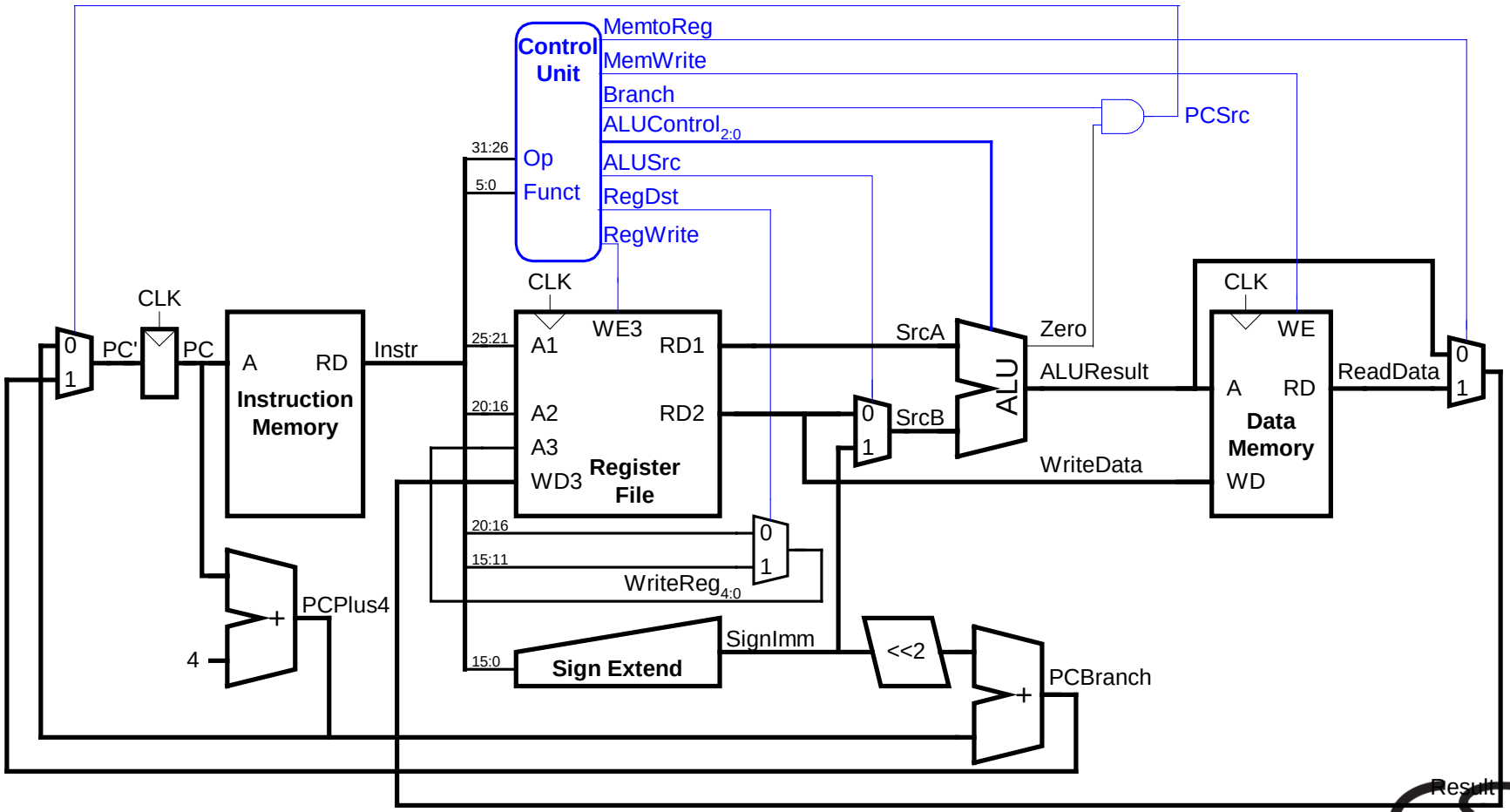
Instruktion	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-Typ	000000							
`lw`	100011							
`sw`	101011							
`beq`	000100							



Beispiel im Ein-Takt Datenpfad: or



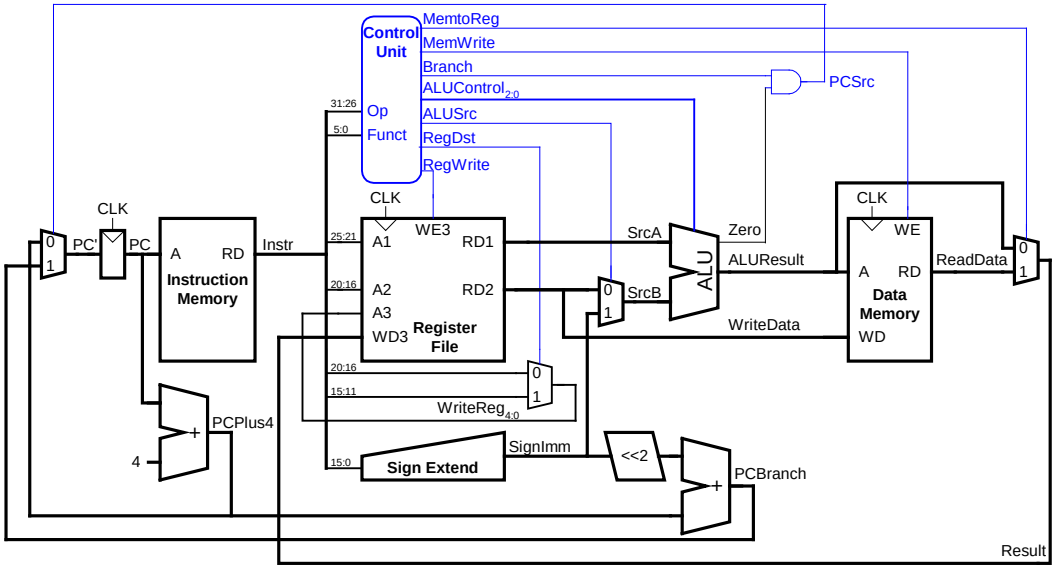
Erweitere Funktionalität: `addi`



Erweitere Steuerwerk: `addi`

TECHNISCHE UNIVERSITÄT DARMSTADT

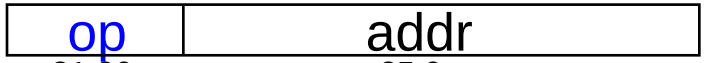
Instruktion	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-Typ	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01
`addi`	**001000**							

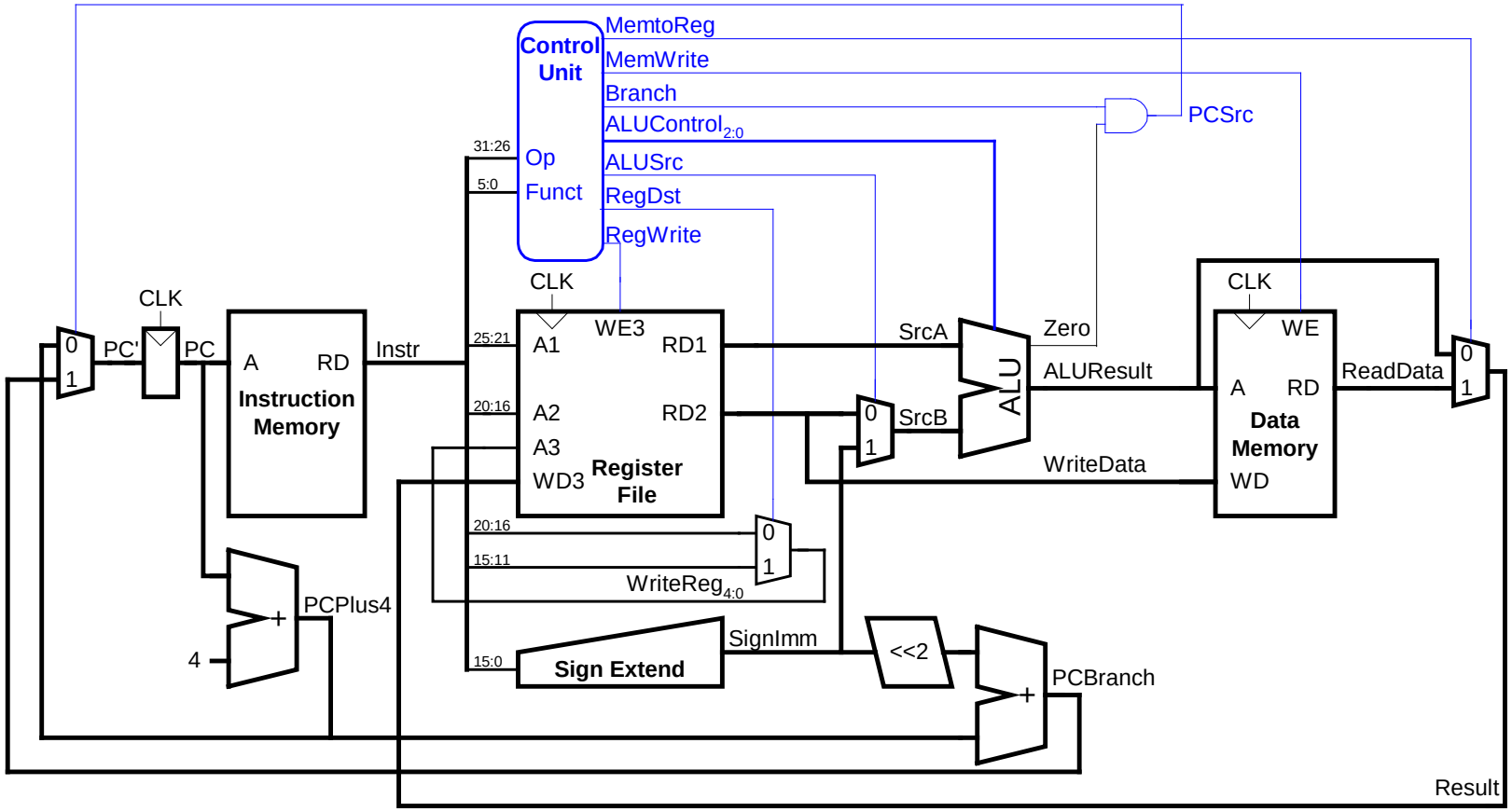


Embedded Systems & Applications

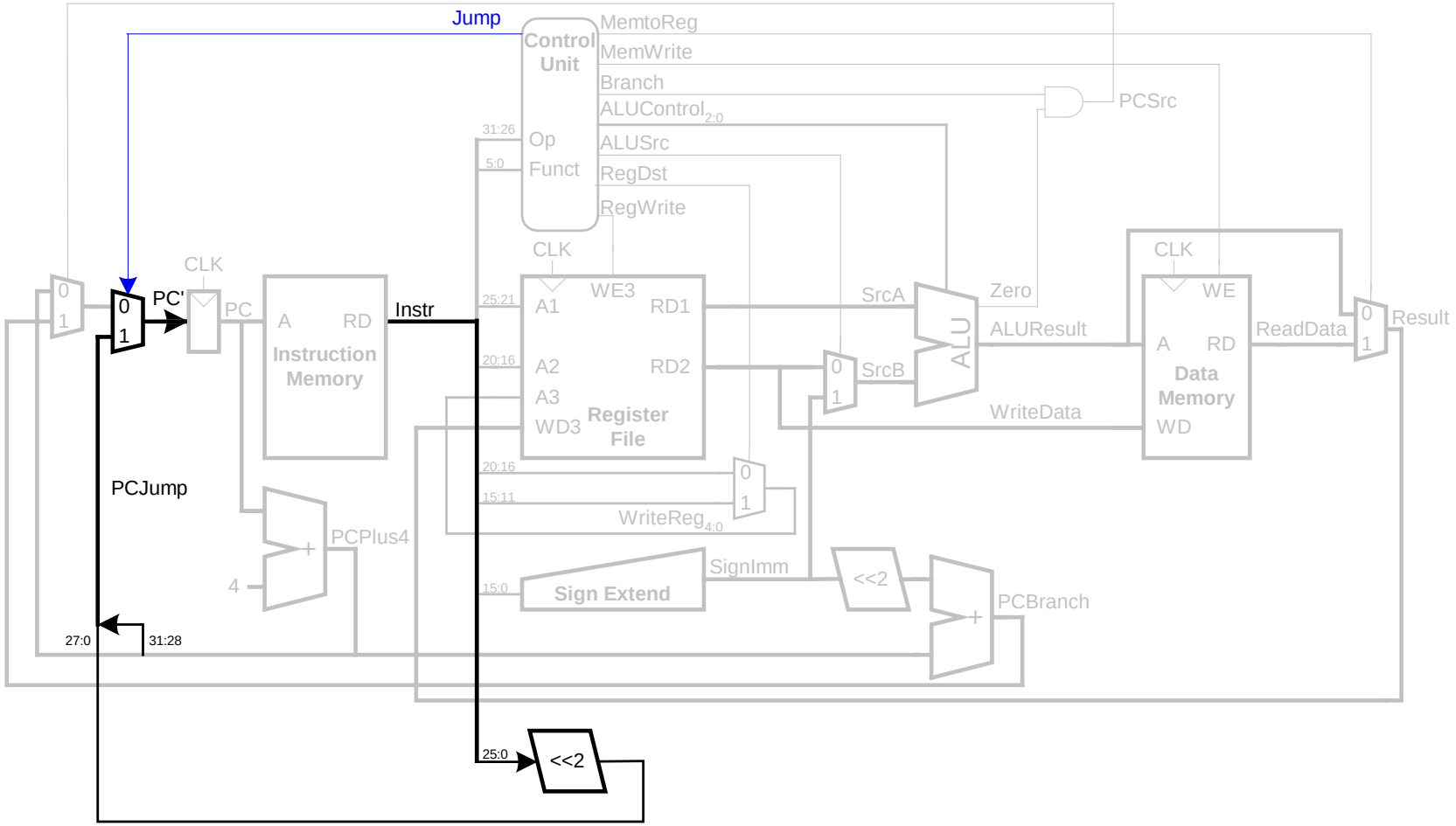
Erweitere Funktionalität: j

j Loop



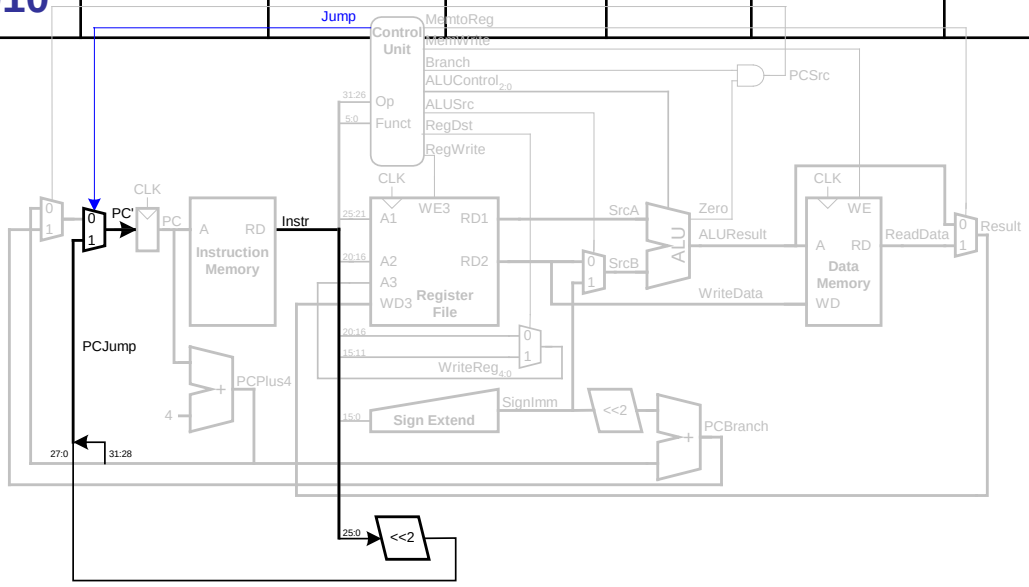


Erweitere Funktionalität: `j`



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**								



Wiederholung: Rechenleistung des Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Programmausführungszeit

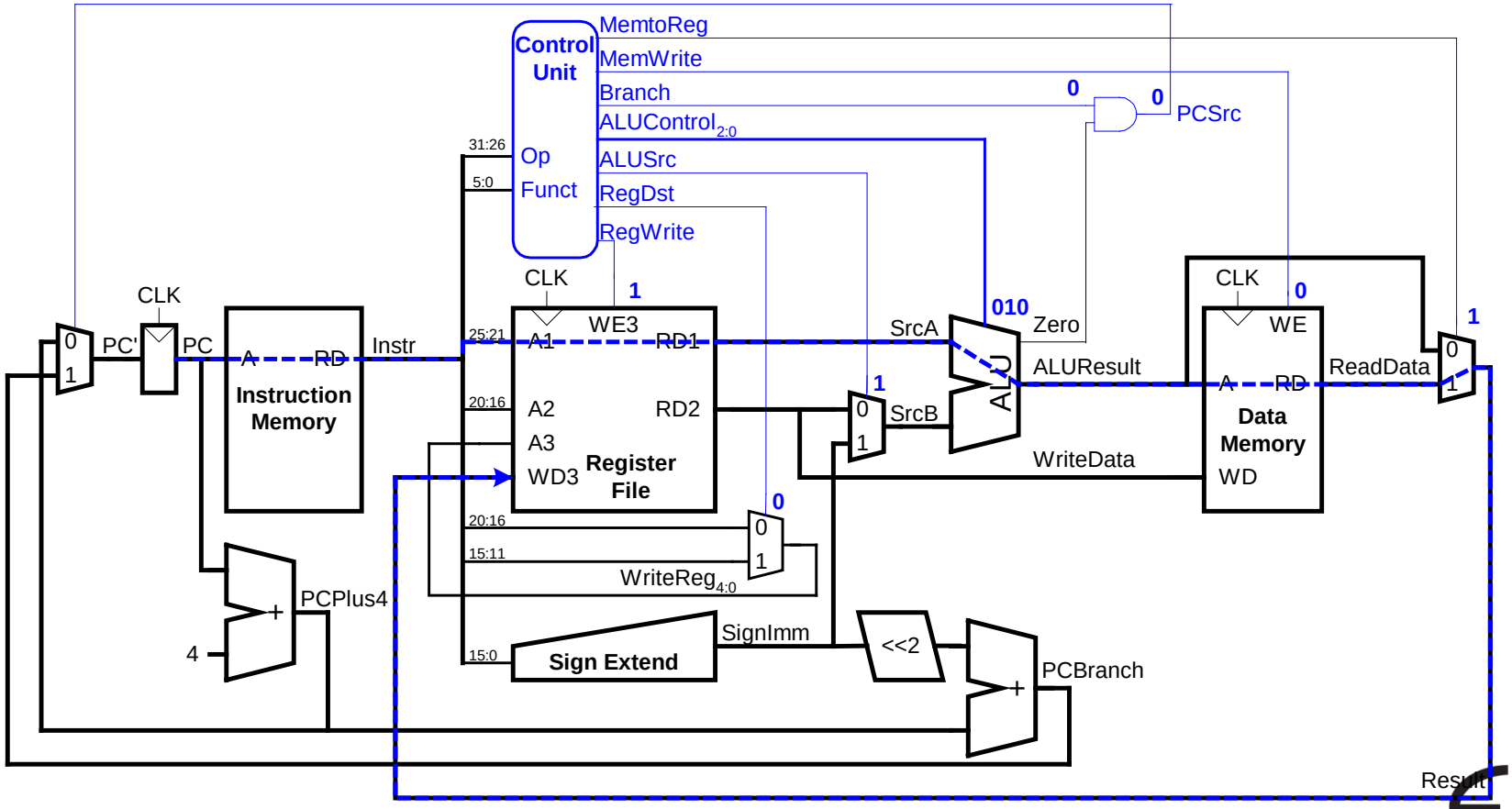
= (# Instruktionen) (Takte/Instruktion)(Sekunden/Takt)

= # Instruktionen * CPI * T_C

Embedded Systems & Applications

Rechenleistung des Ein-Takt-Prozessors

- T_C wird durch längsten Pfad bestimmt (`lw`)



Rechenleistung des Ein-Takt-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

- **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:

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

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 =

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

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

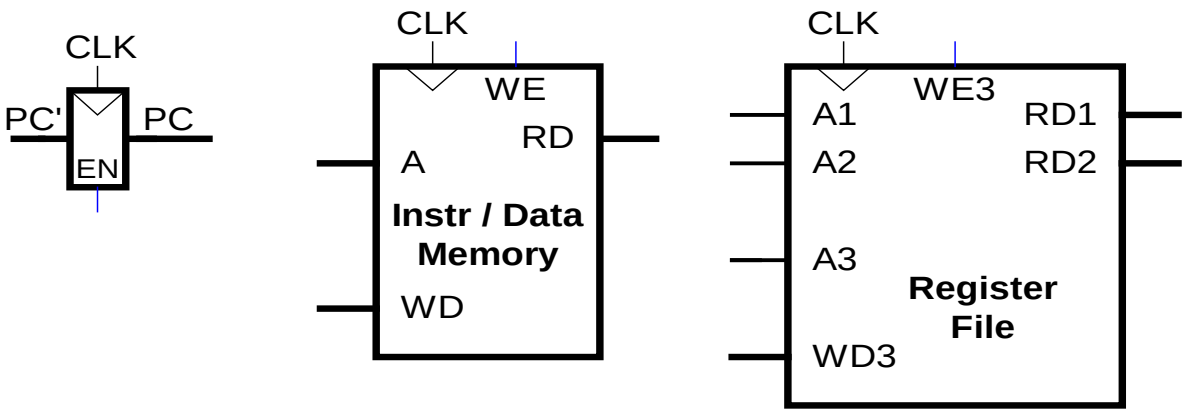
Gleiche Grundkomponenten

- Datenpfad
- Steuerwerk

Embedded Systems & Applications

Zustandselemente im Mehrtaktprozessor

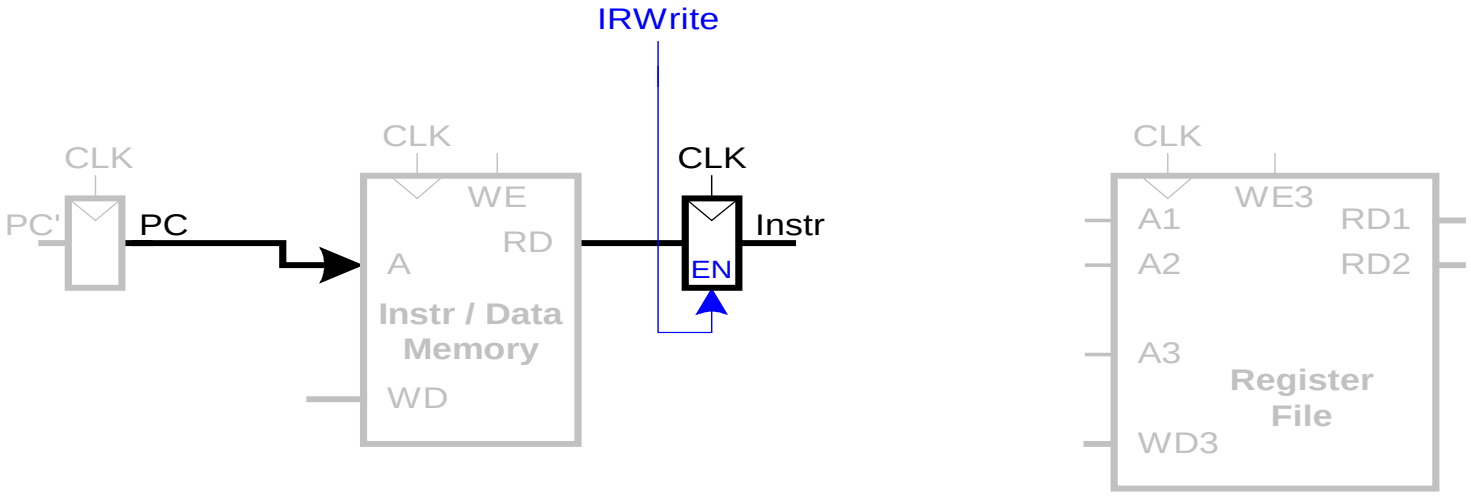
- Ersetze getrennte Instruktions- und Datenspeicher
  - Harvard-Architektur
- Durch einen gemeinsamen Speicher
  - Von Neumann-Architektur
  - Heute weiter verbreitet



Mehrtaktdatenpfad: Instruktionen holen (*fetch*)

Beispiel: Ausführung von `lw`

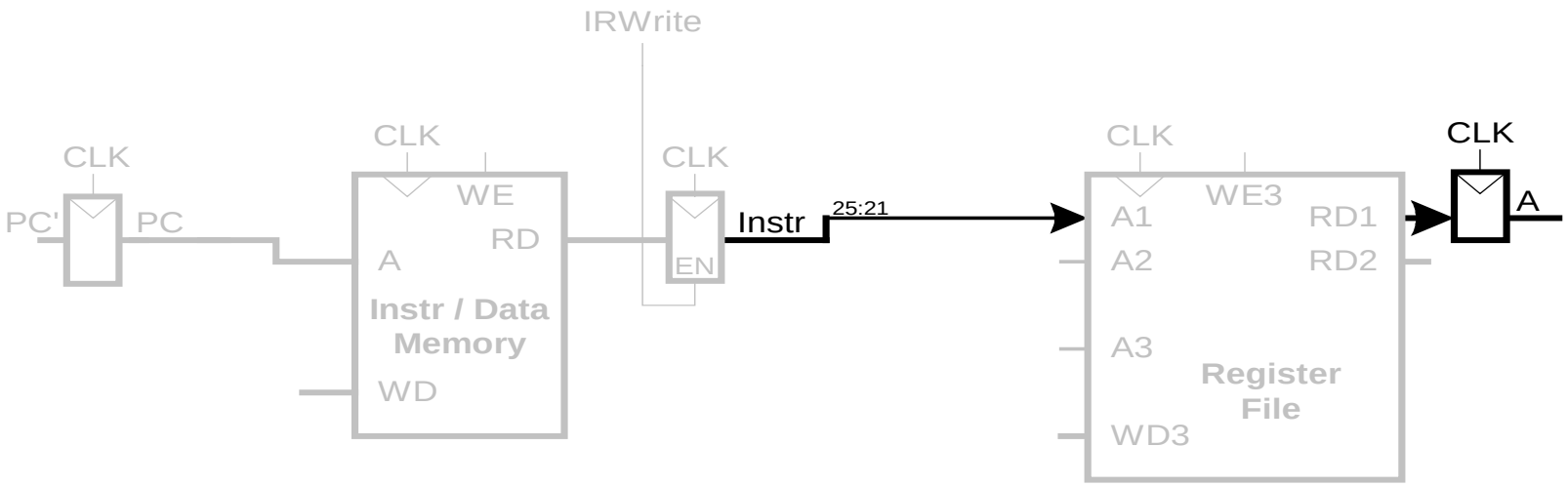
Schritt 1: Hole Instruktion



Mehrtaktdatenpfad: Lese Register für `lw`

TECHNISCHE UNIVERSITÄT DARMSTADT

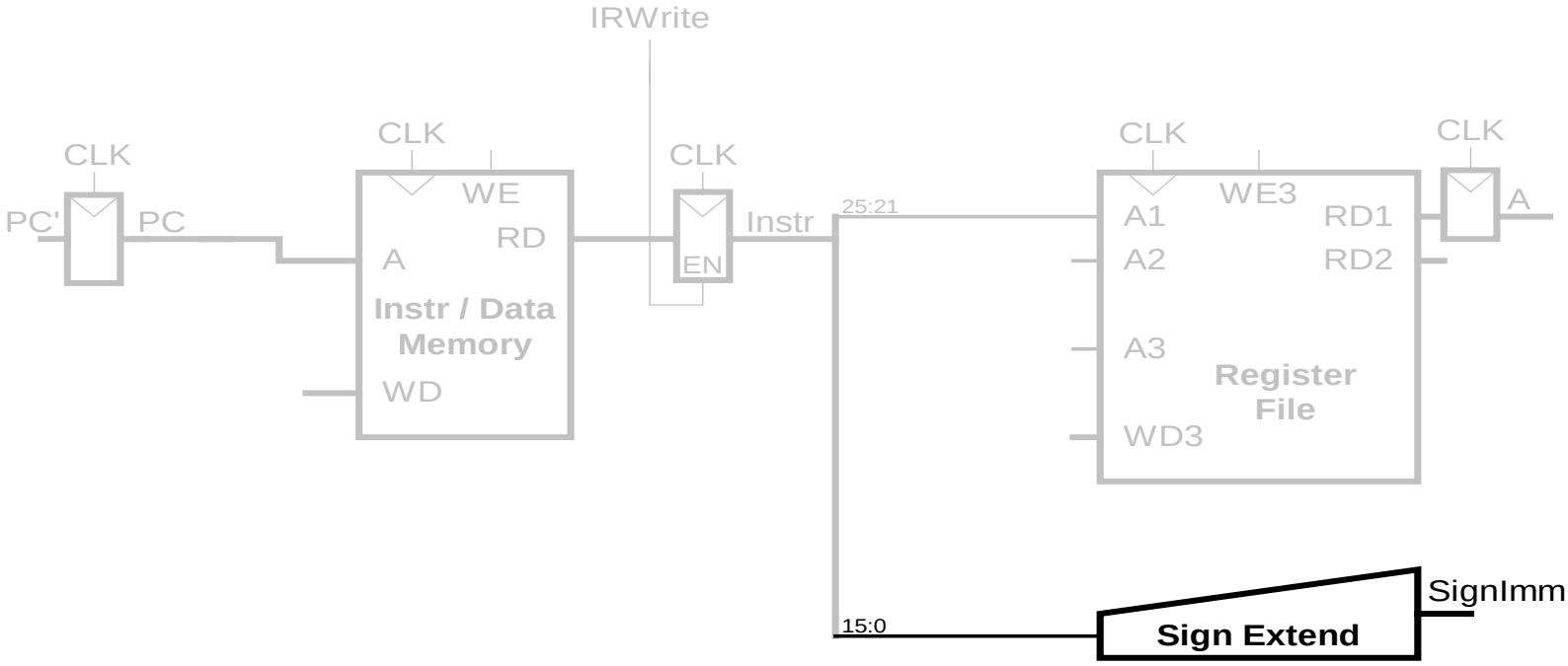
Schritt 2a: Lese Quelloperand aus Registerfeld



Embedded Systems & Applications

Mehrtaktdatenpfad: Werte `lw` Direktwert aus

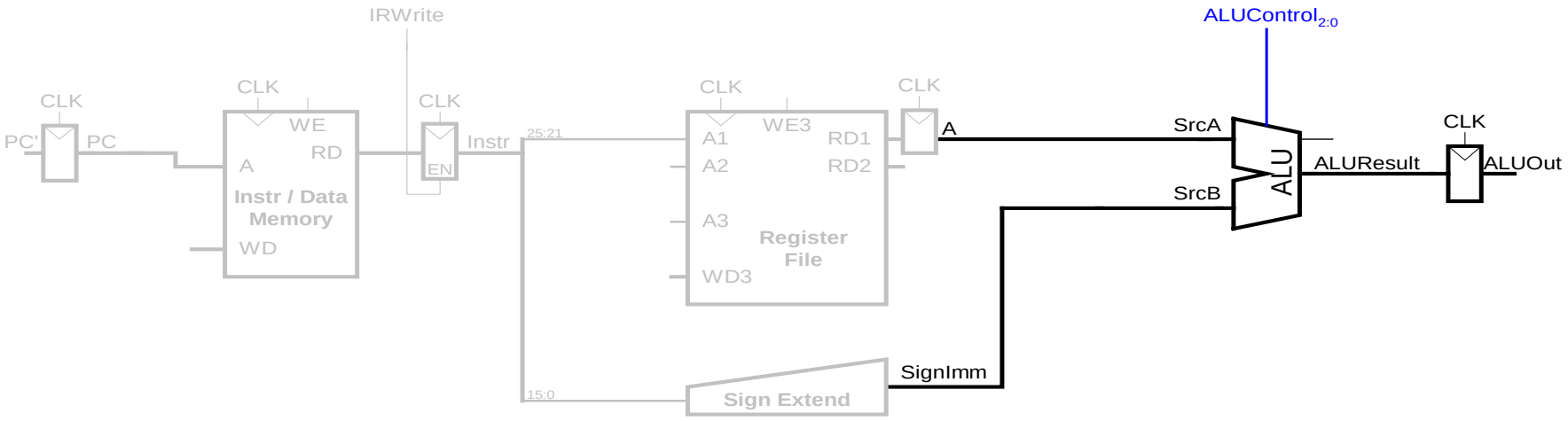
Schritt 2b: Vorzeichenerweitere den 16b Direktwert auf 32b Signal `SignImm`



Mehrtaktdatenpfad: Bestimme effektive Adresse für `lw`

TECHNISCHE UNIVERSITÄT DARMSTADT

Schritt 3: Berechne die effektive Speicheradresse

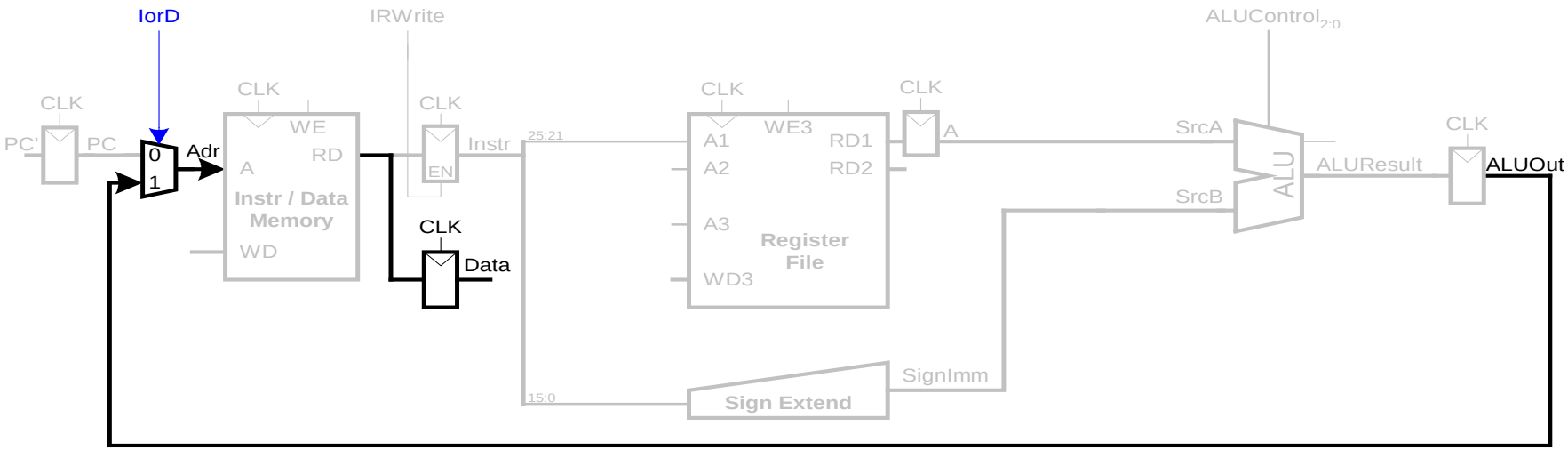


Embedded Systems & Applications

Mehrtaktdatenpfad: Lesezugriff von `lw`

TECHNISCHE UNIVERSITÄT DARMSTADT

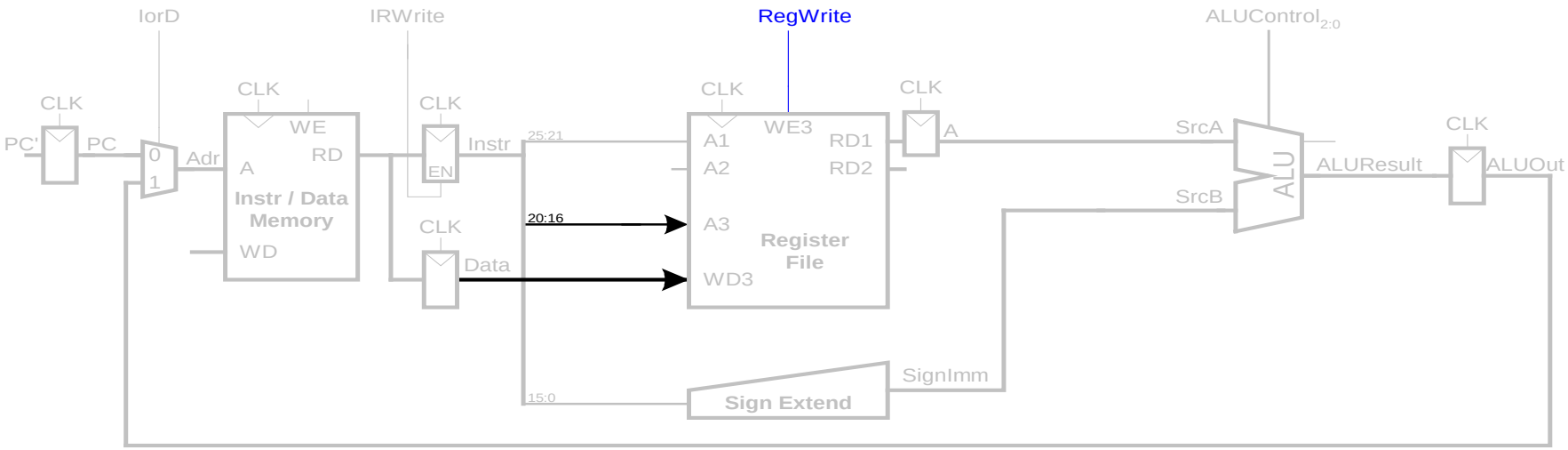
Schritt 4: Lese Daten aus Speicher



Embedded Systems & Applications

Mehrtaktdatenpfad: Schreibe Register in `lw`

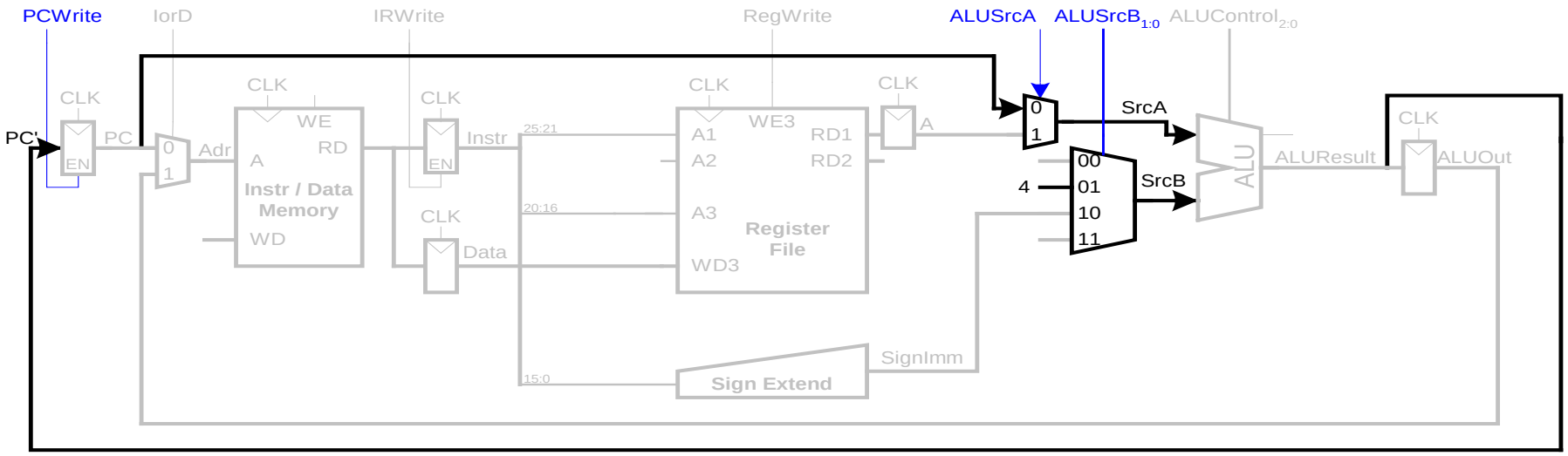
Schritt 5: Schreibe die Daten ins passende Register



Mehrtaktdatenpfad: Erhöhe PC

TECHNISCHE UNIVERSITÄT DARMSTADT

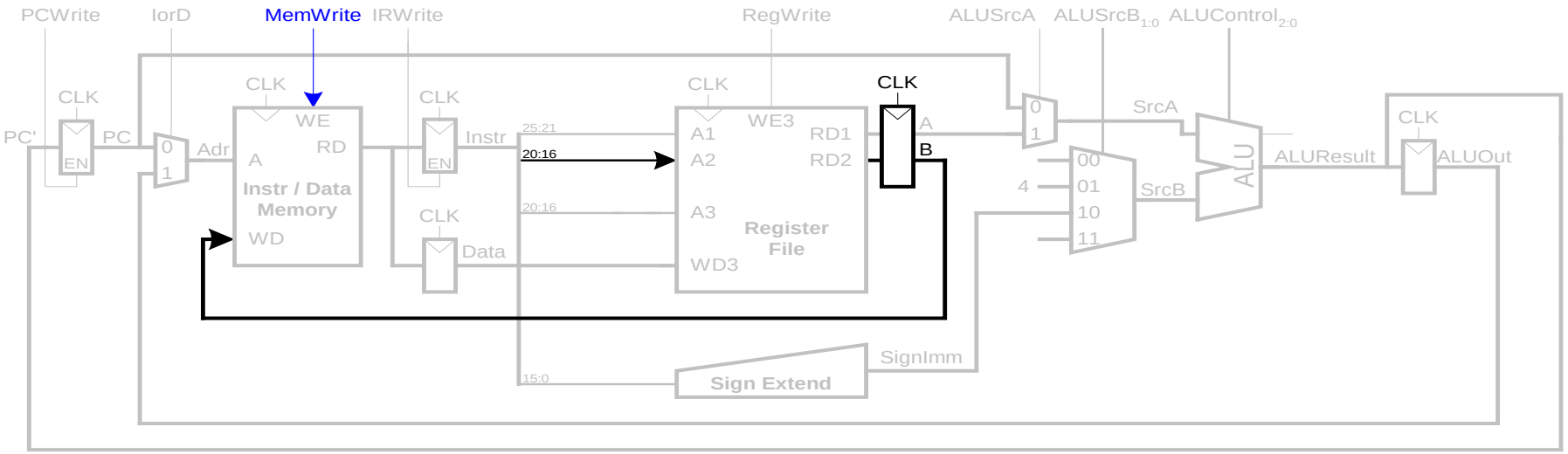
Schritt 6: Bestimme Adresse des nächsten Befehls



Embedded Systems & Applications

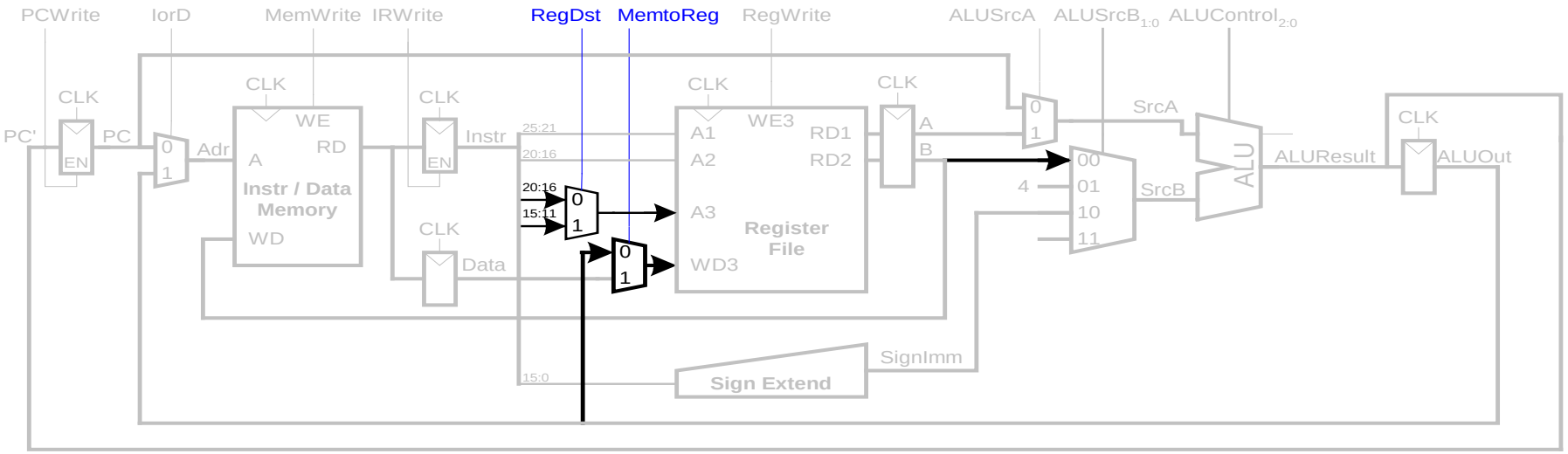
Mehrtaktdatenpfad: Nun Ausführung von sw

- Schreibe Daten aus `rt` in Speicher



Mehrtaktdatenpfad: Instruktion vom R-Typ

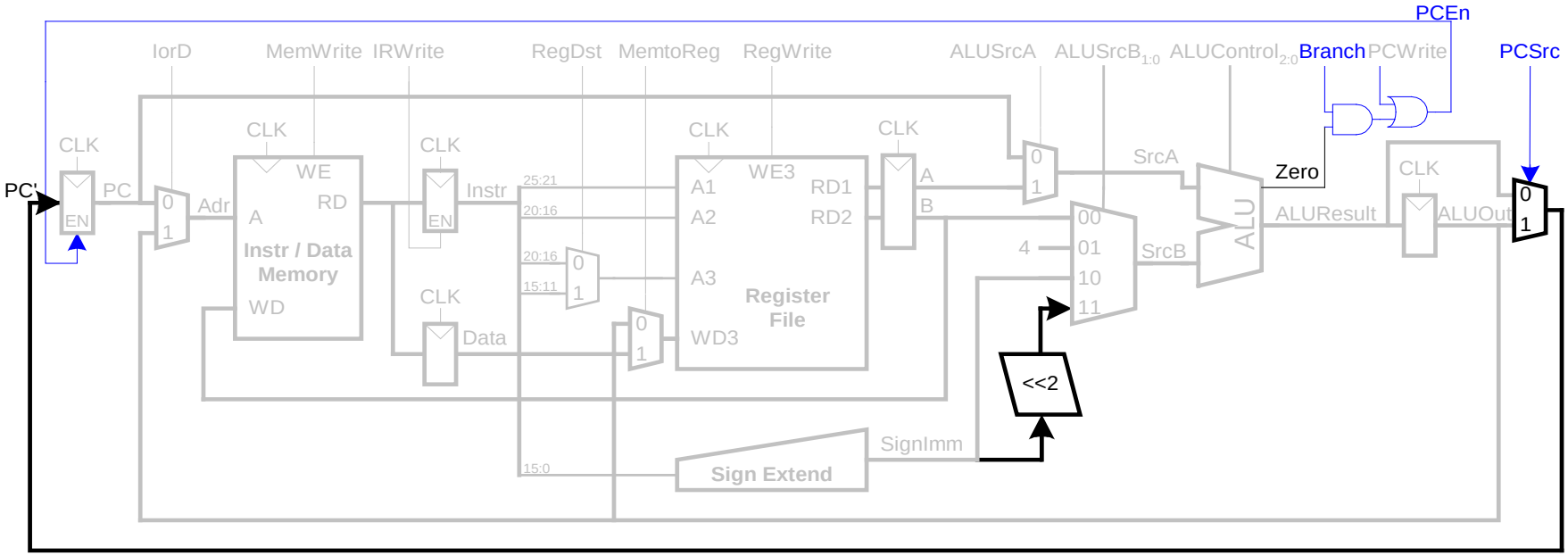
- Lese Werte aus `rs` und `rt`
- Schreibe *ALUResult* ins Registerfeld
- Schreibe Wert nach `rd` (statt nach `rt`)



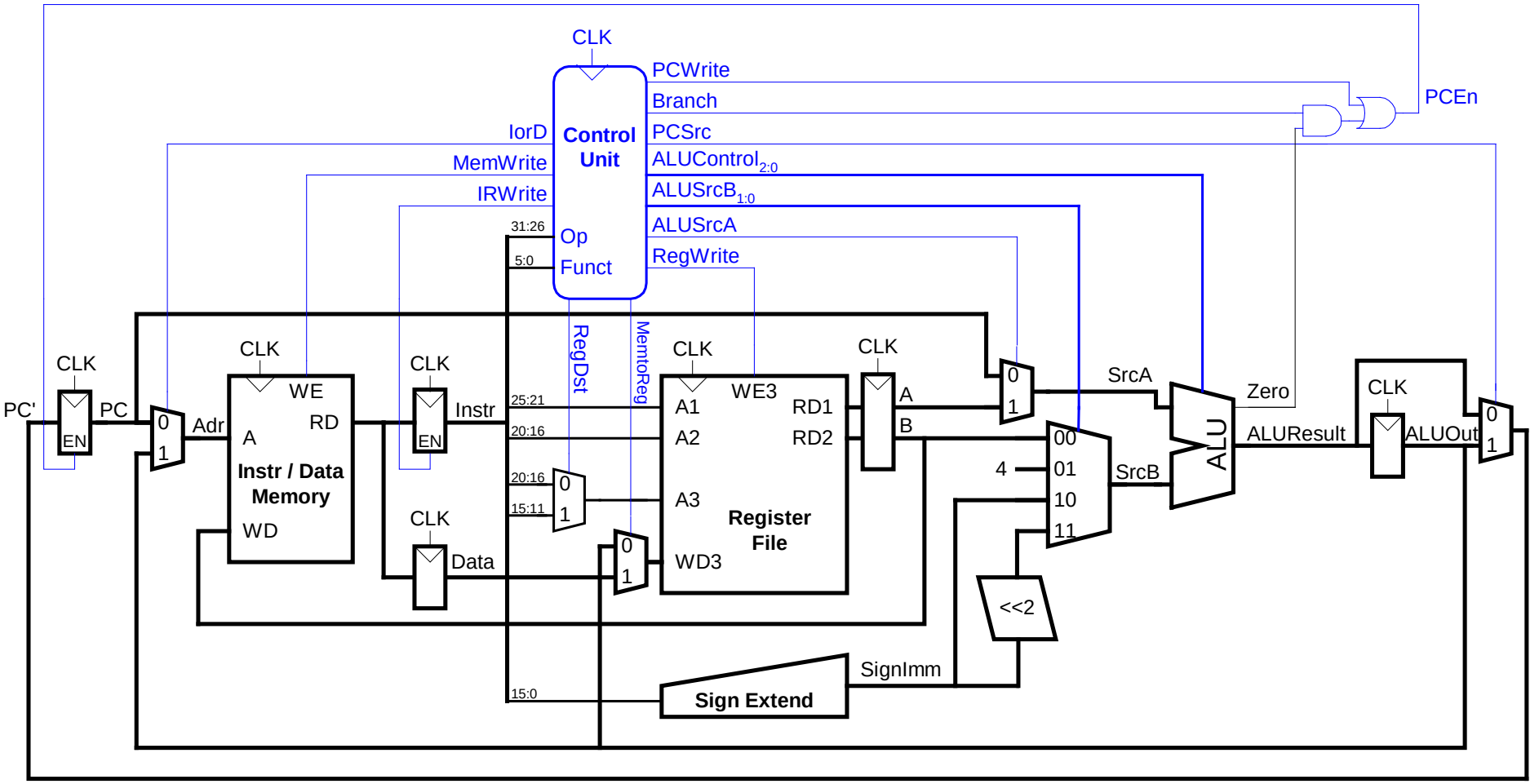
Mehrtaktdatenpfad: beq

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

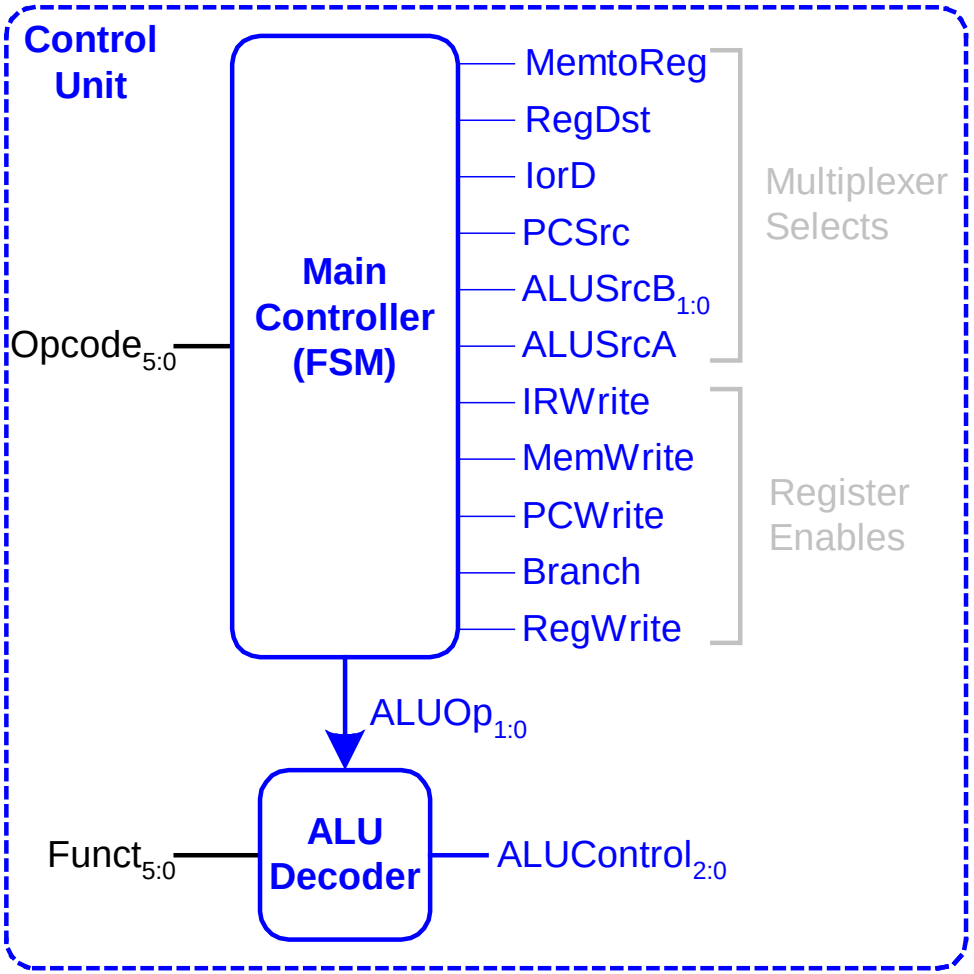
  BTA = (vorzeichenerweiterter Direktwert << 2) + (PC+4)



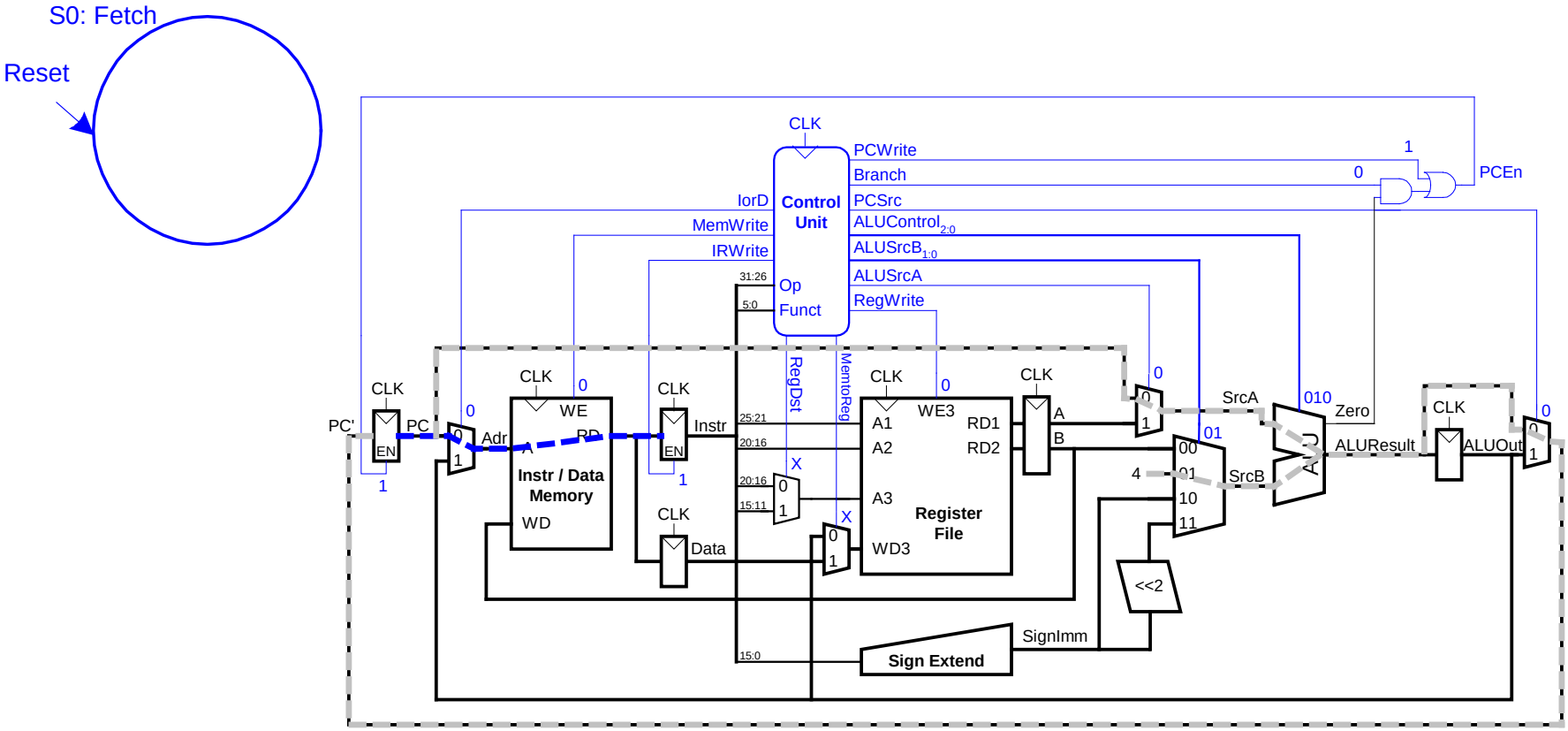
Vollständiger Mehrtaktprozessor



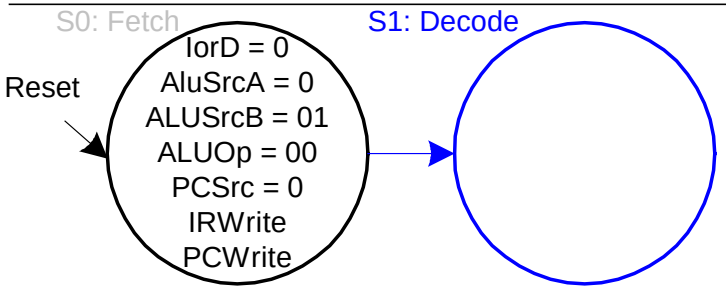
Steuerwerk

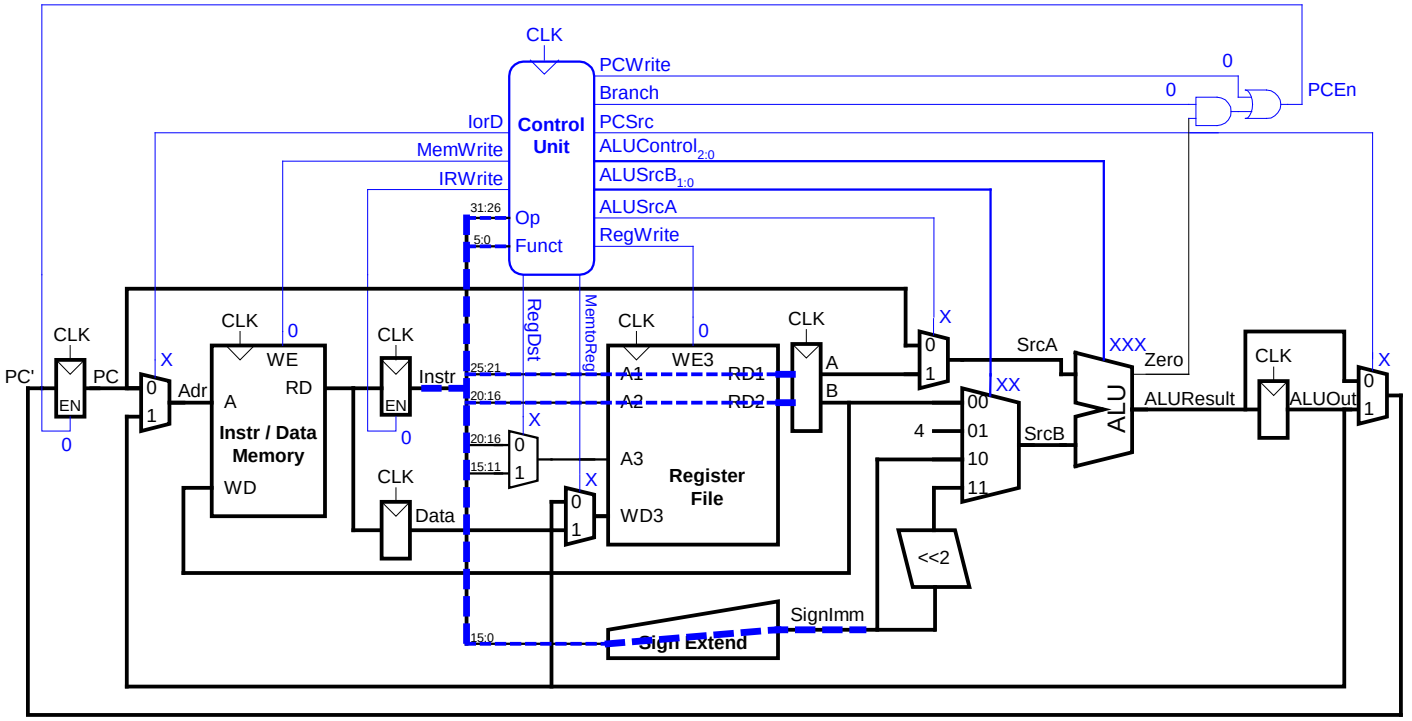

Control Unit
Main Controller (FSM)
Opcode5:0
MemtoReg
RegDst
IorD
PCSrc
ALUSrcB1:0
ALUSrcA
IRWrite
MemWrite
PCWrite
Branch
RegWrite
Multiplexer Selects
Register Enables
ALUOp1:0
ALU Decoder
Funct5:0
ALUControl2:0

Hauptsteuerwerk: Holen eines Befehls

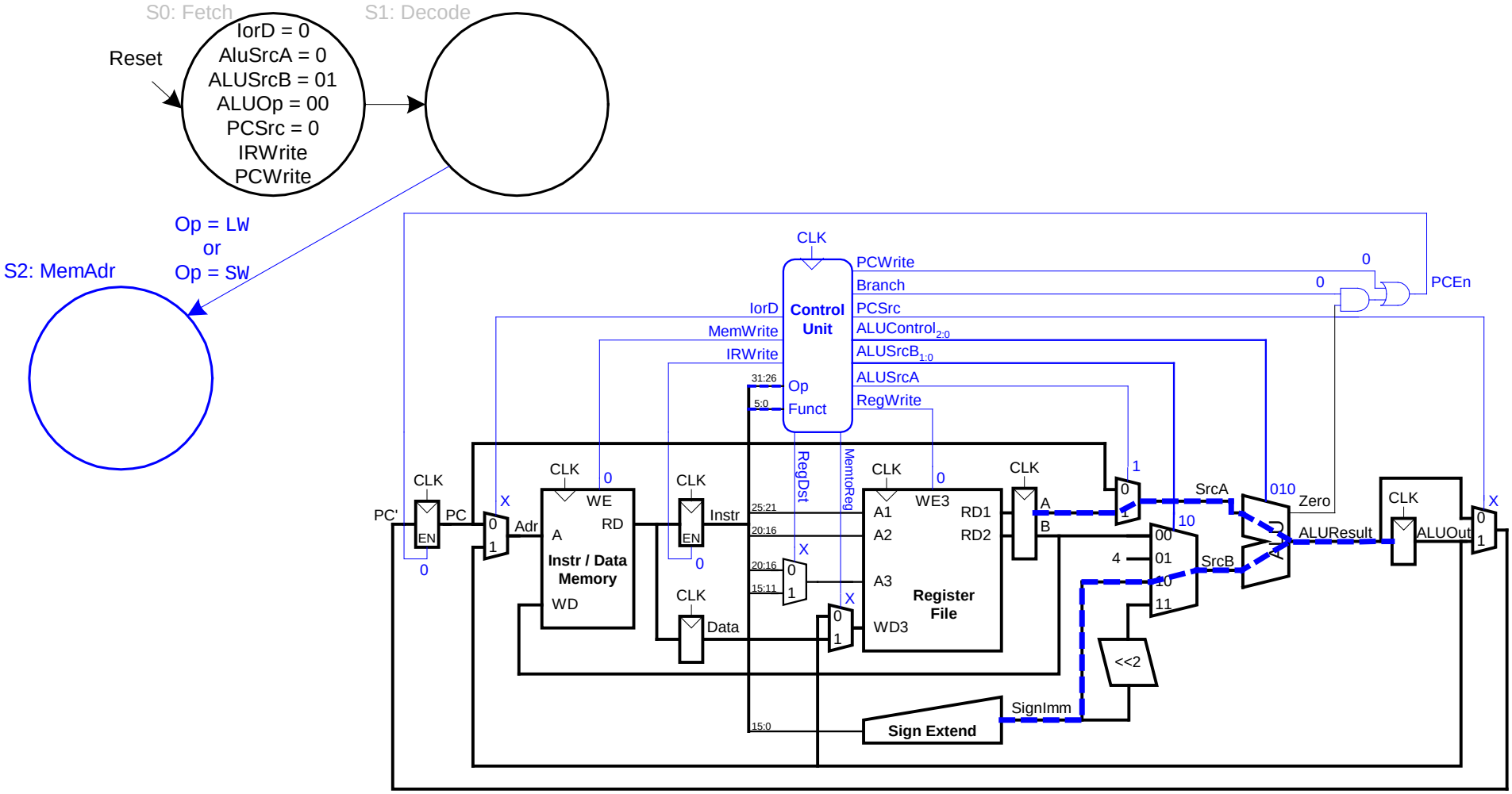


Hauptsteuerwerk: Dekodieren eines Befehls

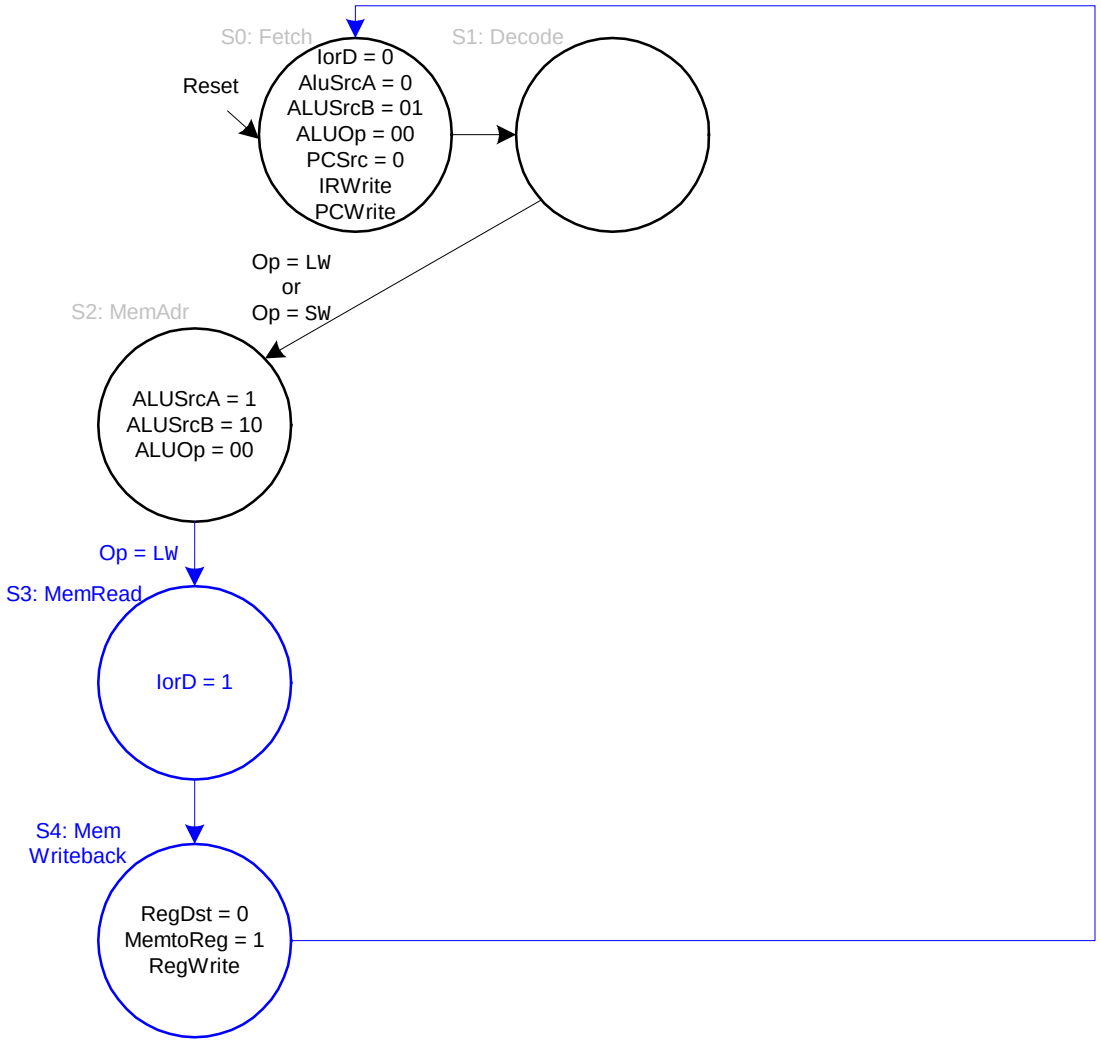




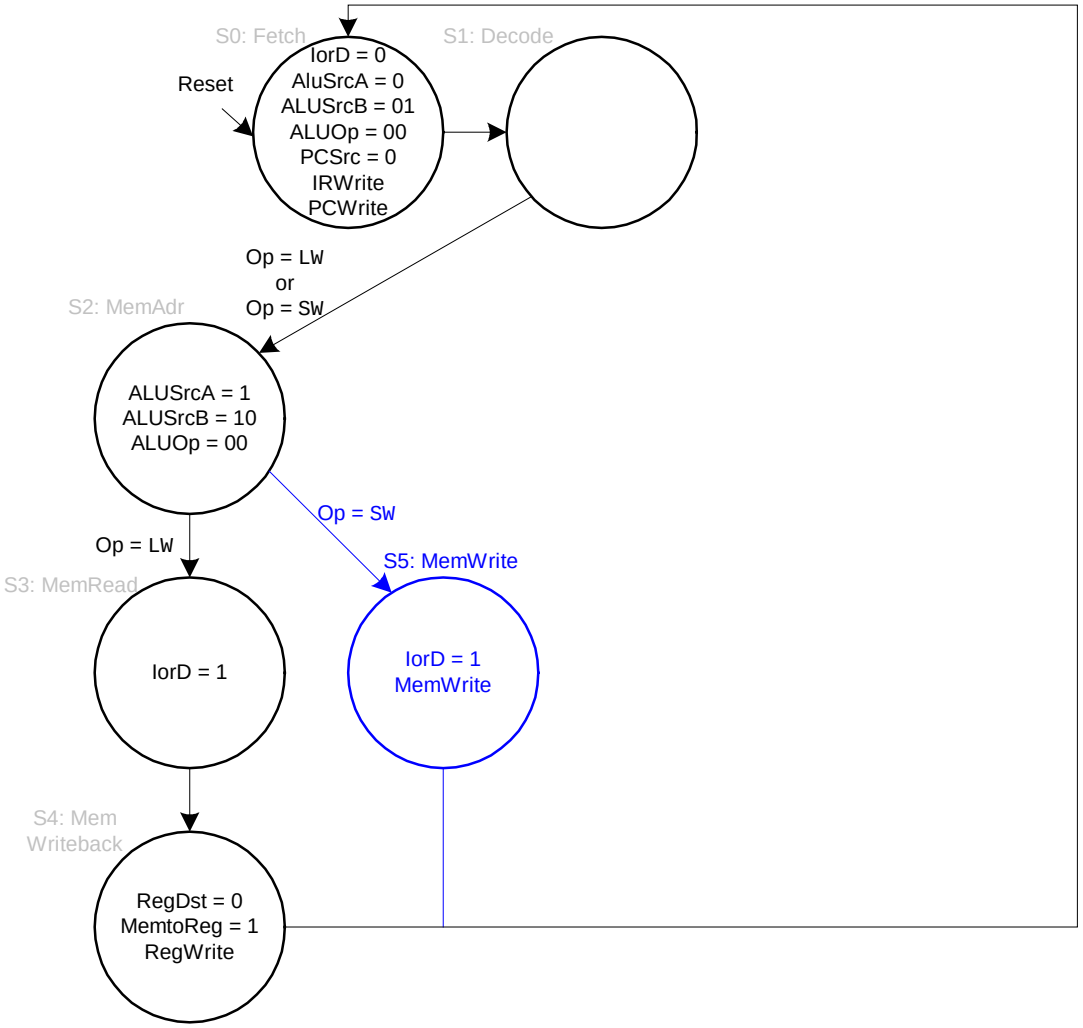
Hauptsteuerwerk: Adressberechnung



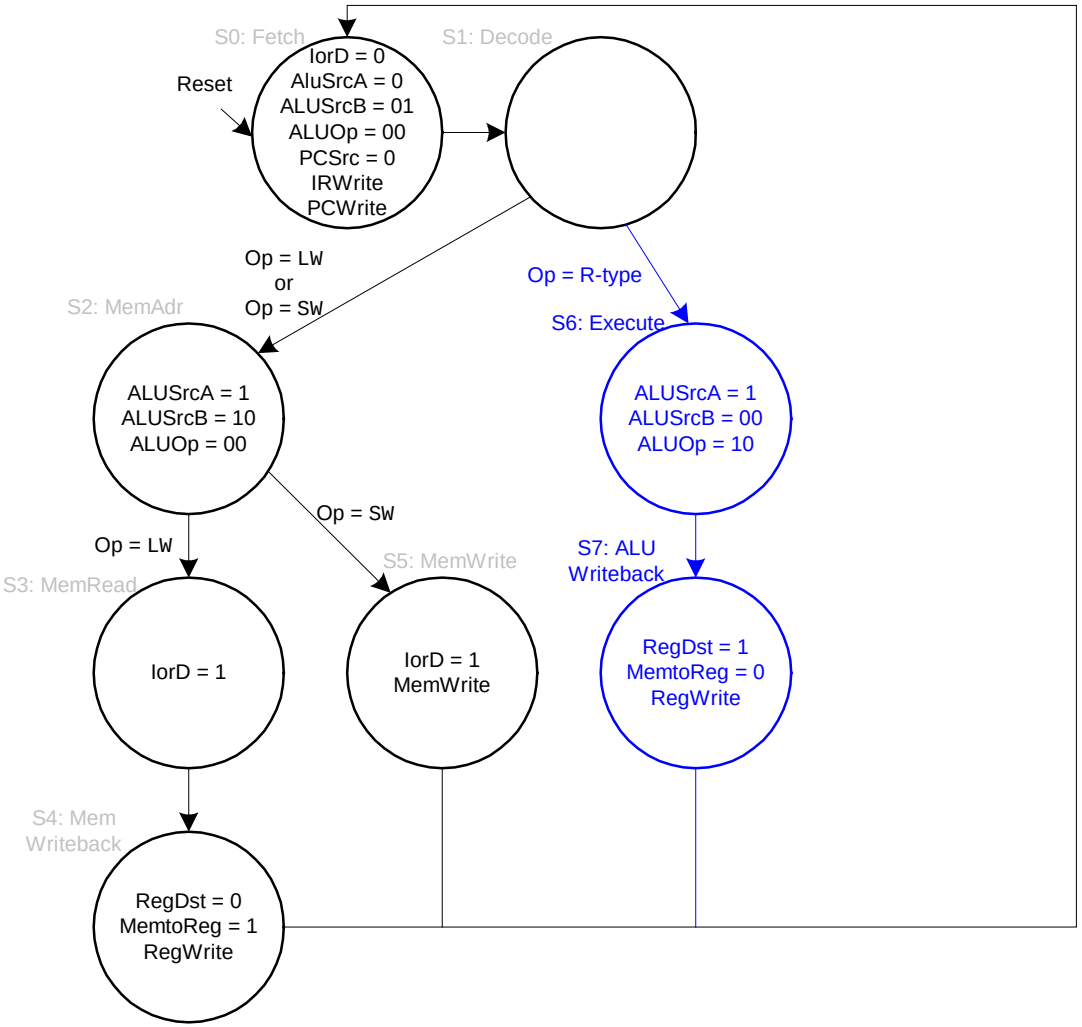
Hauptsteuerwerk: FSM für `lw`



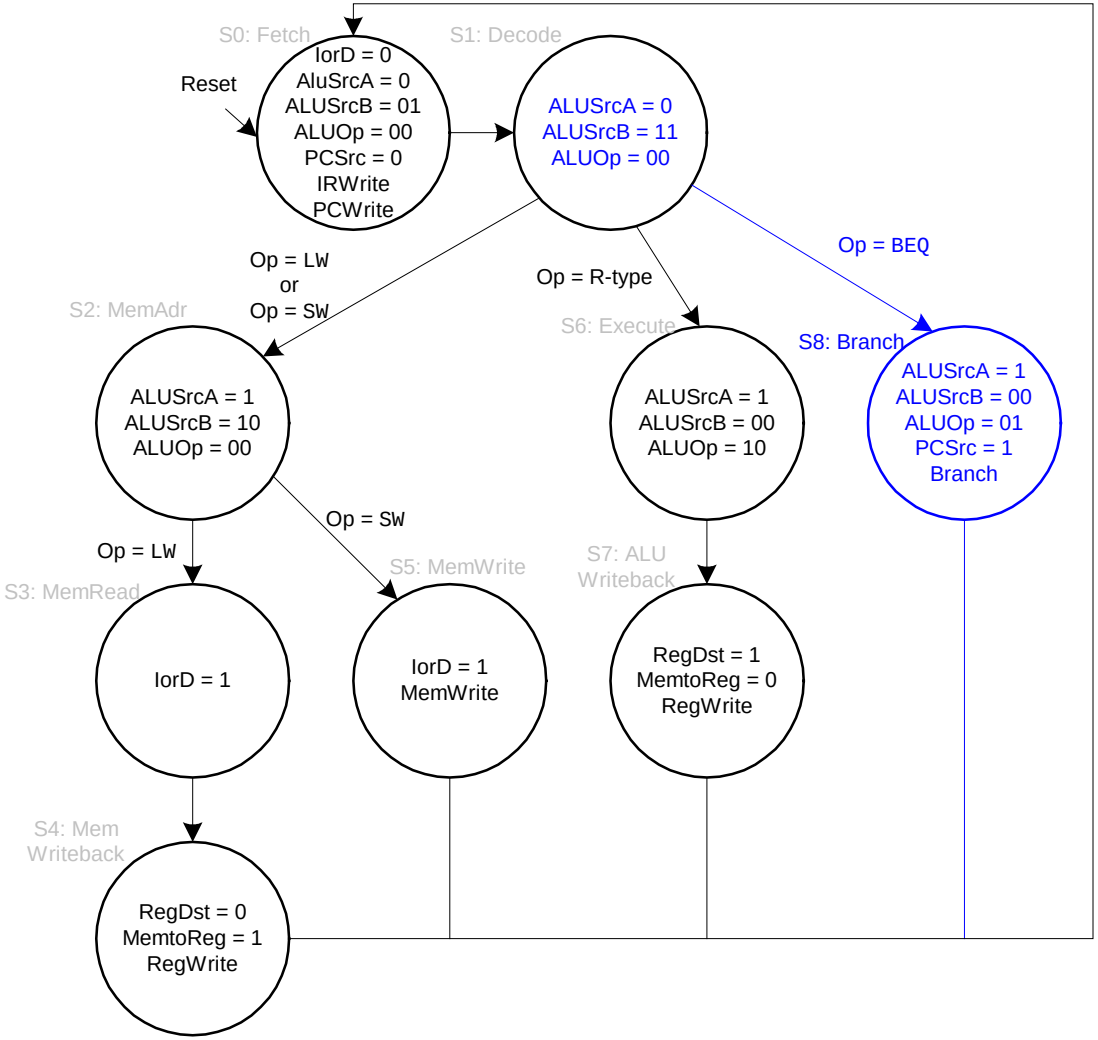
Hauptsteuerwerk: FSM für sw



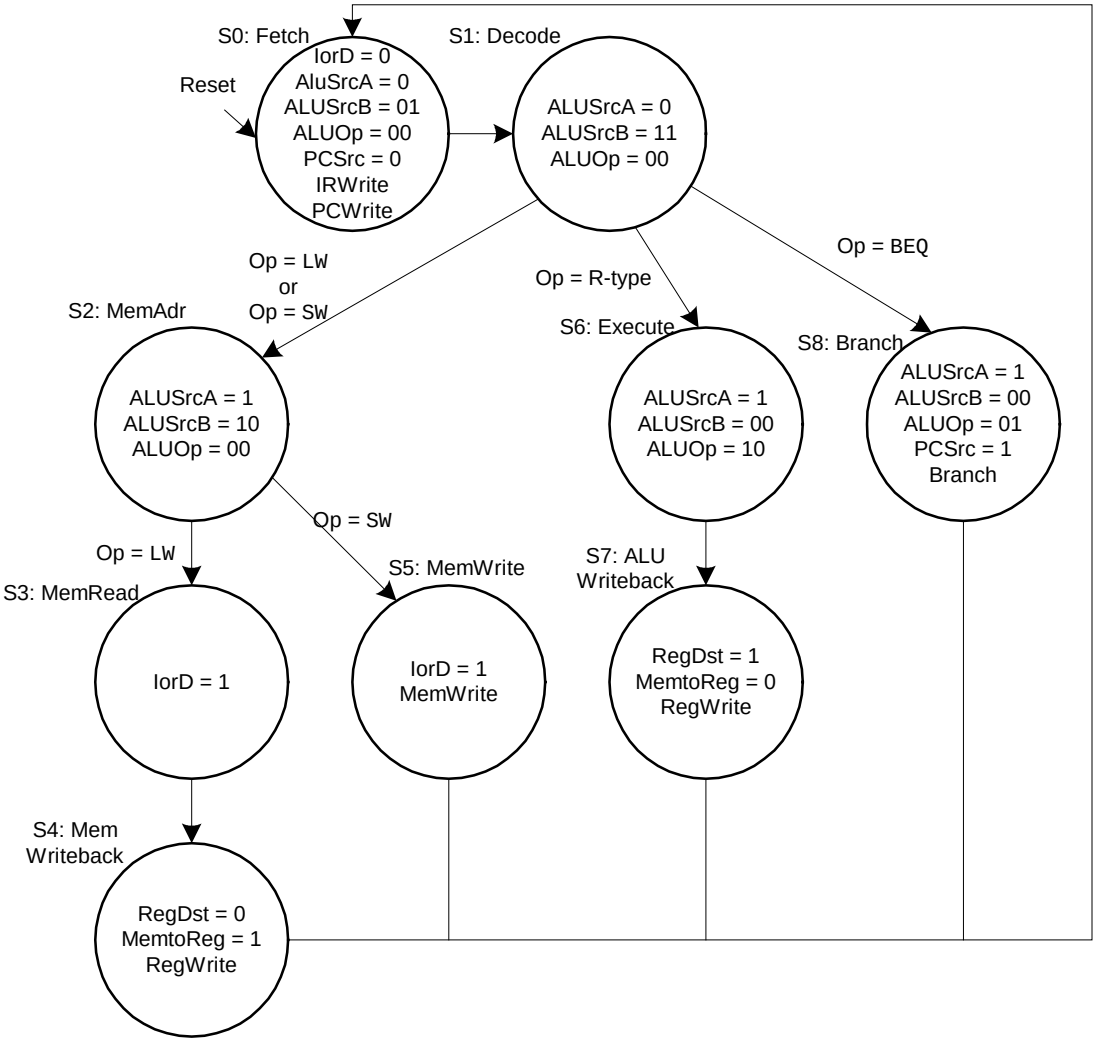
Hauptsteuerwerk: FSM für R-Typ



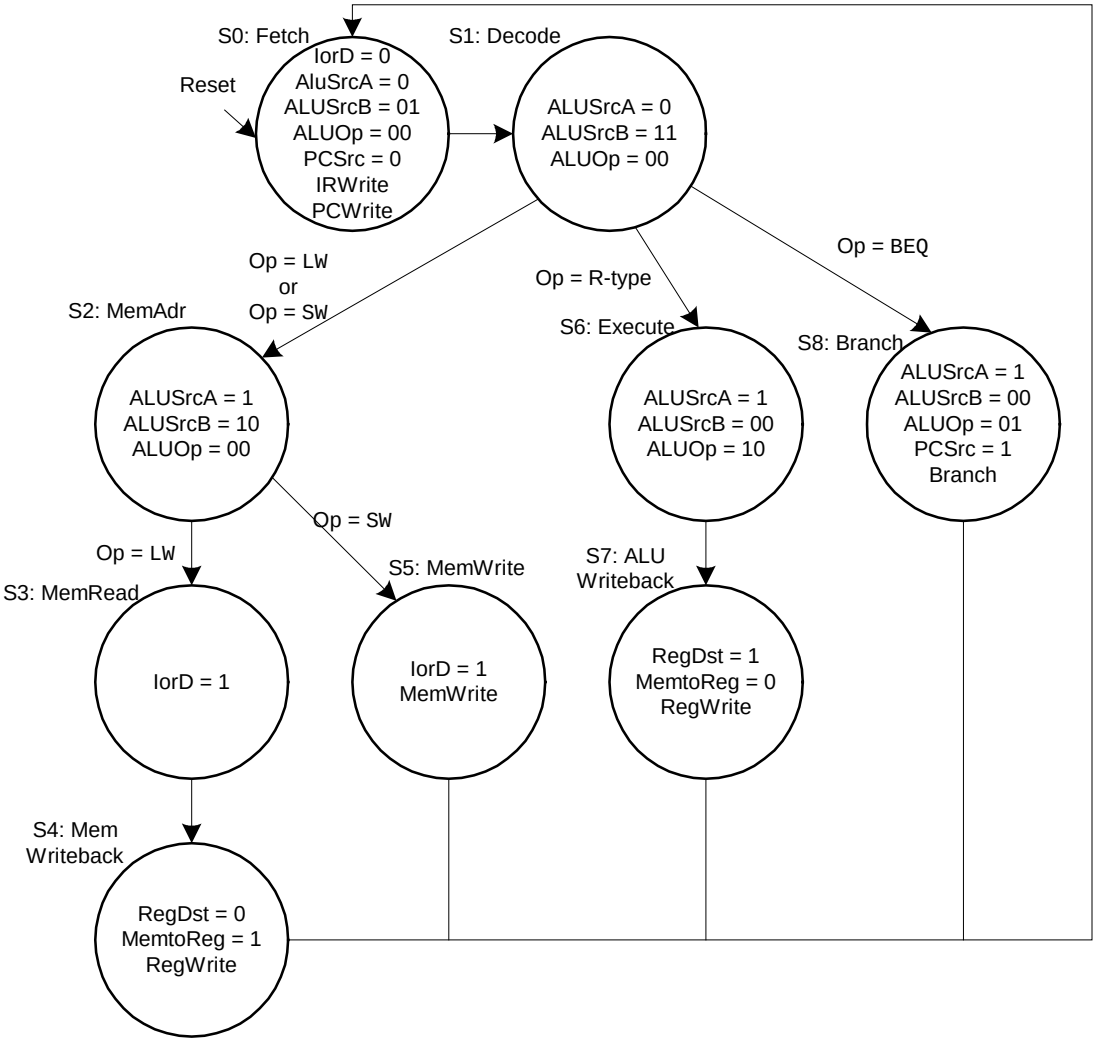
Hauptsteuerwerk: FSM für beq



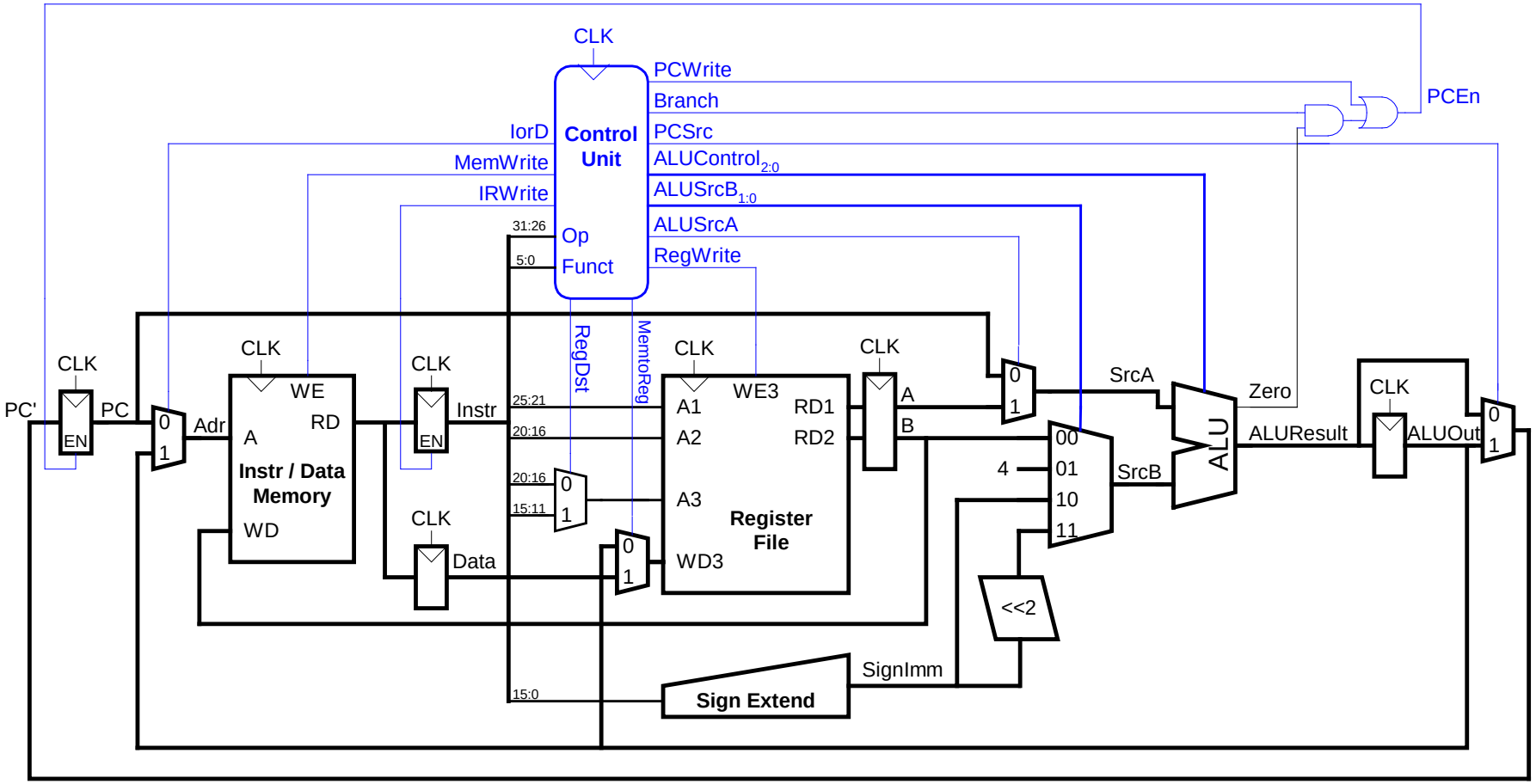
Vollständiges Hauptsteuerwerk für Mehrtakt-CPU



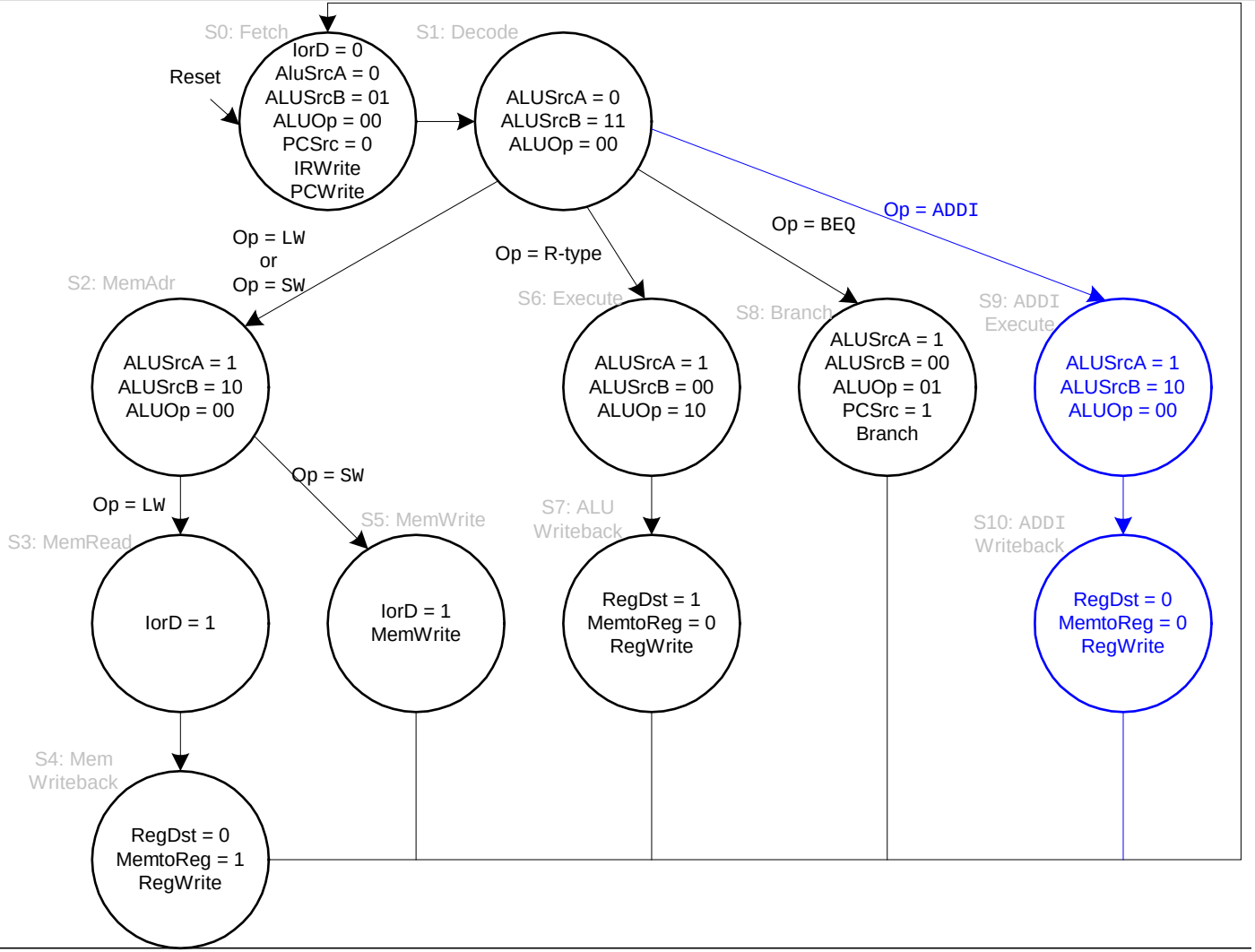
Erweiterung des Hauptsteuerwerks: `addi`



Erweiterung des Datenpfads für `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 =

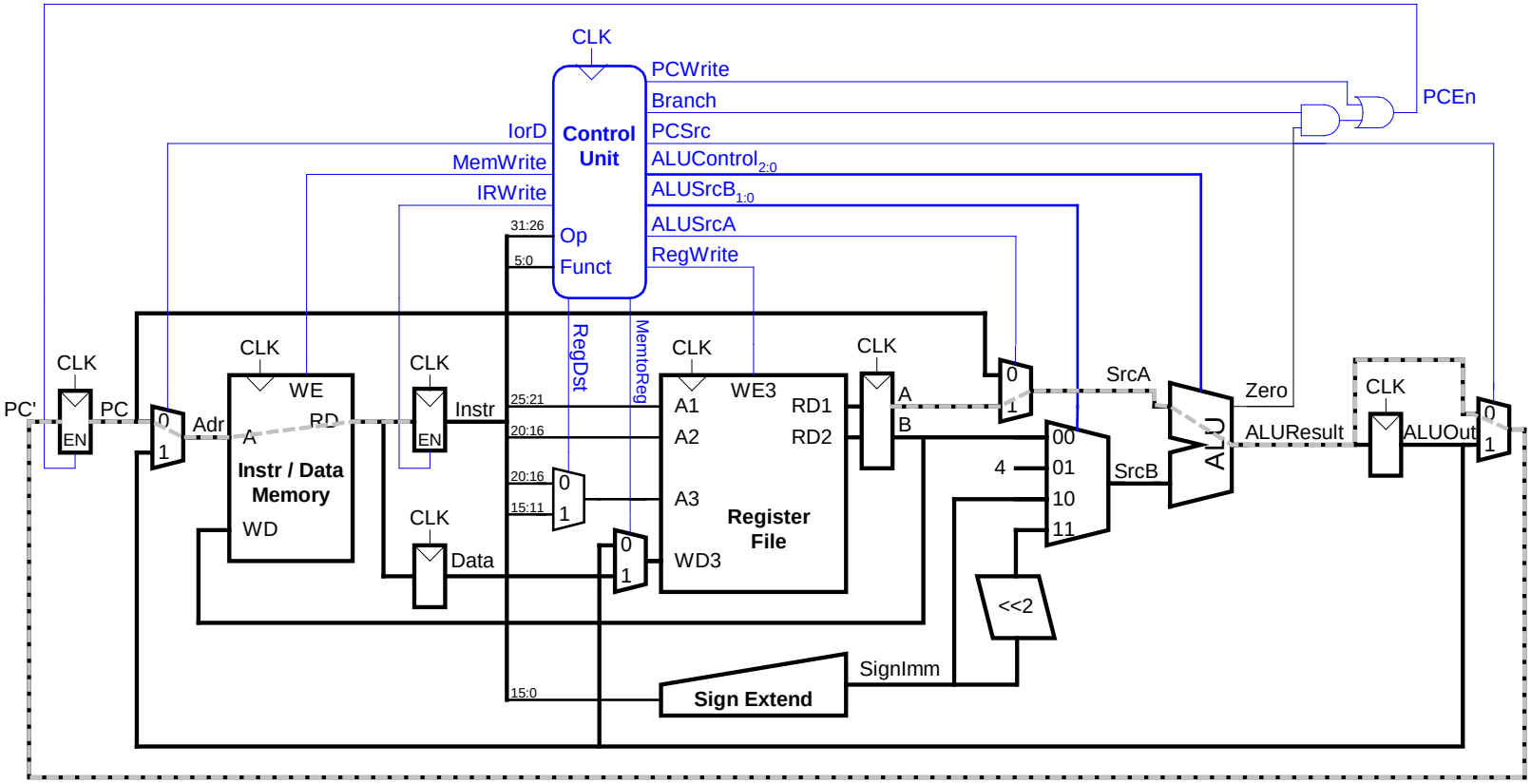
Embedded Systems & Applications

Rechenleistung des Mehrtaktprozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Führe Programm mit 100 Milliarden Instruktionen auf Mehrtaktprozessor aus

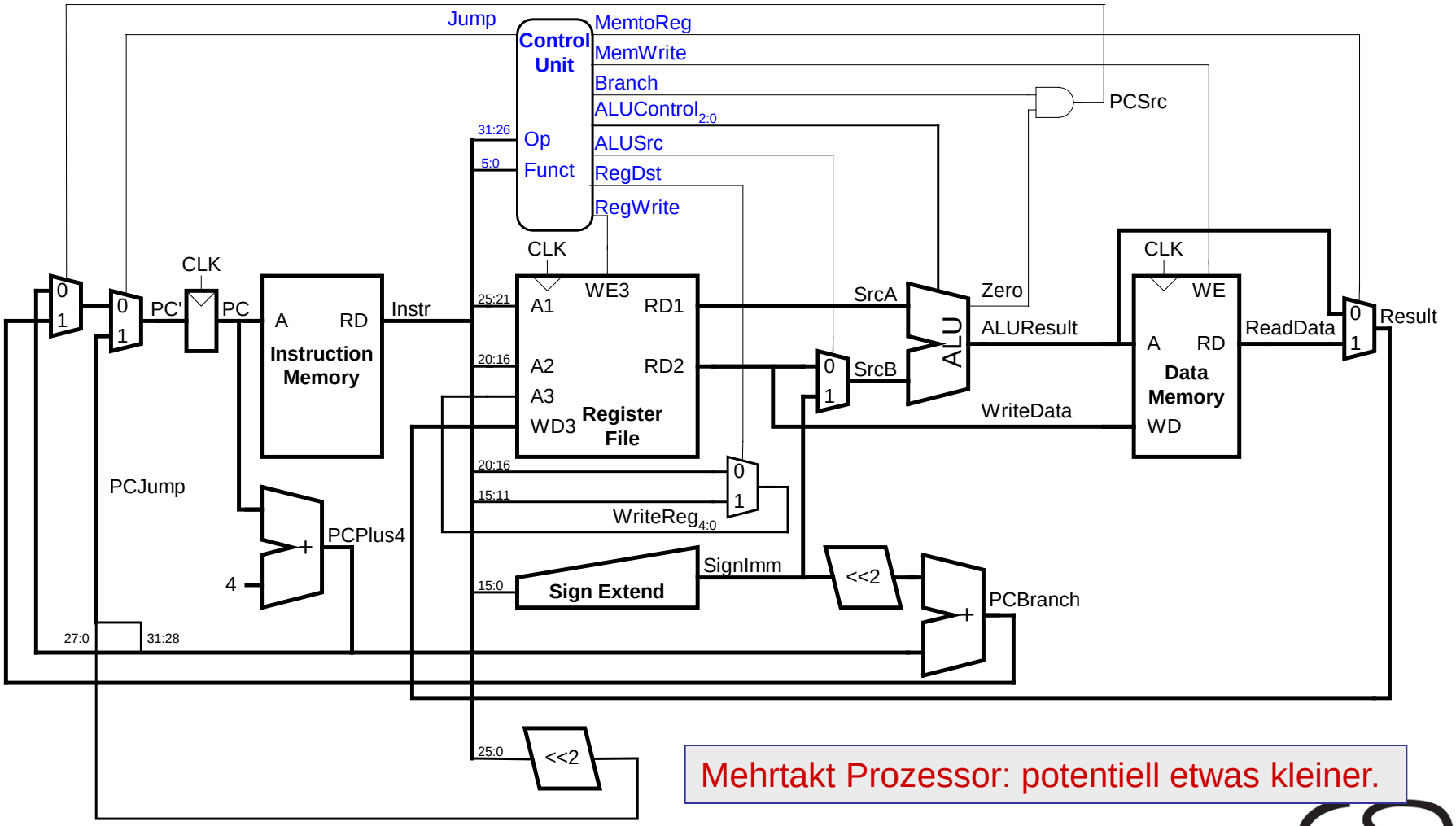
- CPI = 4,12
- T_c = 325 ps

Ausführungszeit = ?

Embedded Systems & Applications

Rückblick: Ein-Takt MIPS Prozessor

TECHNISCHE UNIVERSITÄT DARMSTADT



Mehrtakt Prozessor: potentiell etwas kleiner.

Embedded Systems & Applications

MIPS Prozessor mit Pipelining

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zeitliche Parallelität

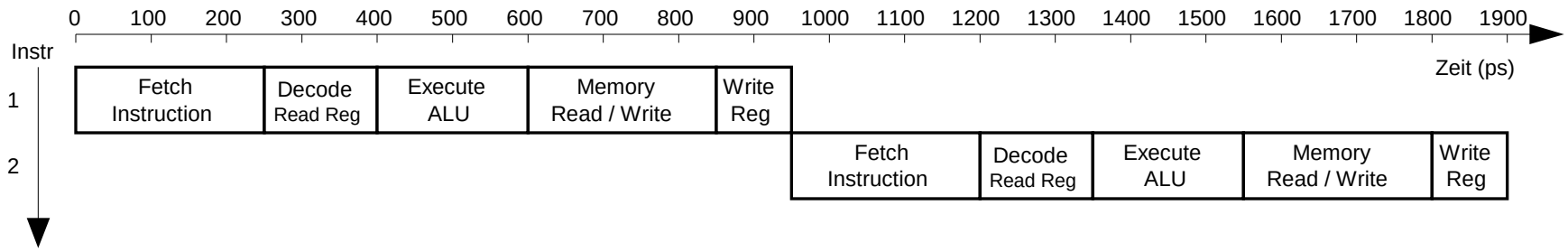
- Teile Ablauf im Ein-Takt-Prozessor in fünf Stufen:
  - Hole Instruktion (*Fetch*)
  - Dekodiere Bedeutung von Instruktion (*Decode*)
  - Führe Instruktion aus (*Execute*)
  - Greife auf Speicher zu (*Memory*)
  - Schreibe Ergebnisse zurück (*Writeback*)

- Füge Pipeline-Register zwischen den Stufen ein

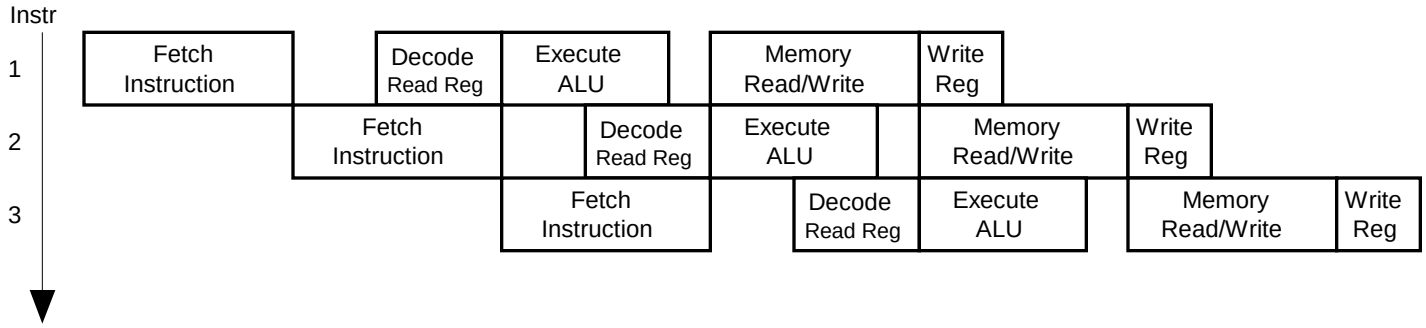
Rechenleistung: Ein-Takt und Pipelined

TECHNISCHE UNIVERSITÄT DARMSTADT

Ein-Takt

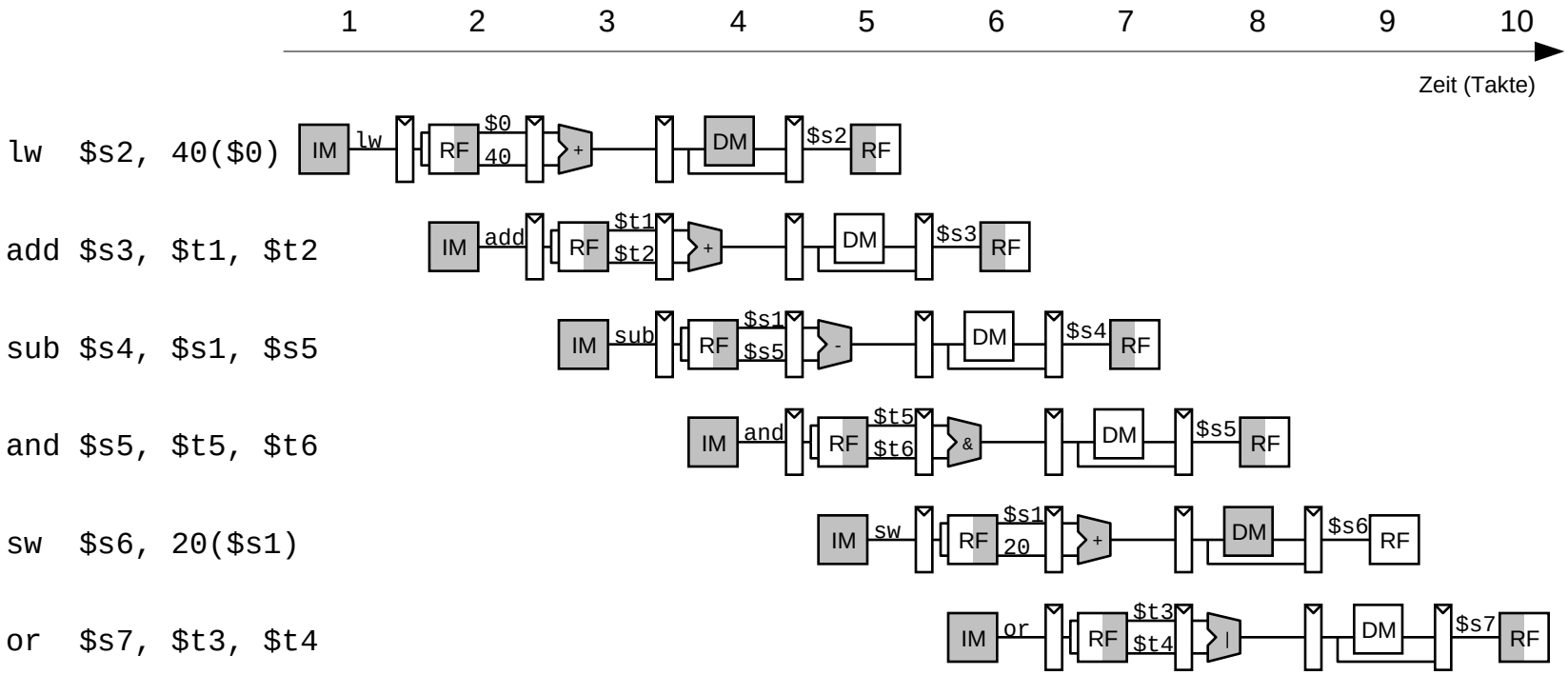


Pipelined

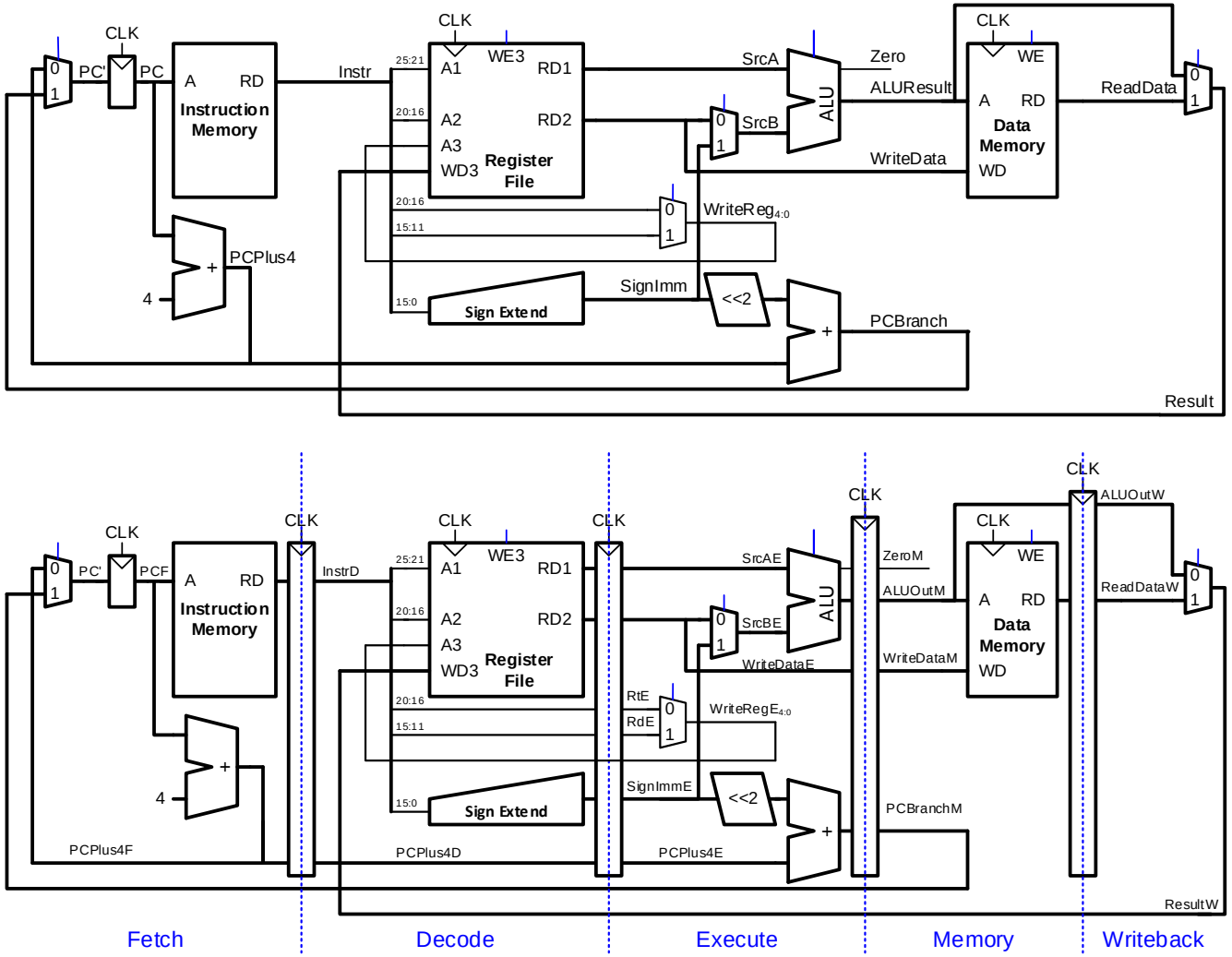


Embedded Systems & Applications

Abstraktere Darstellung des Pipelinings

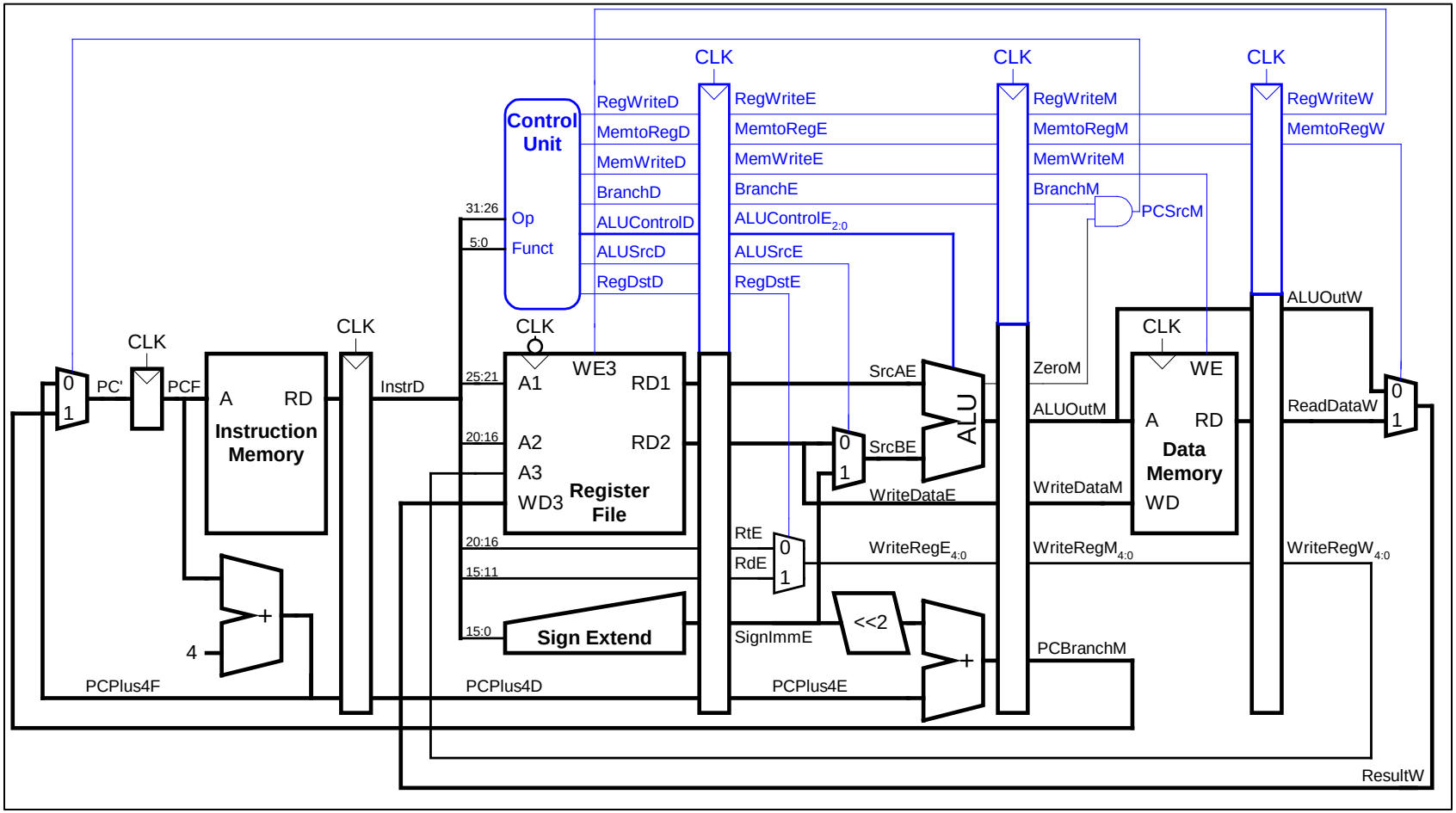


Ein-Takt- und Pipelined-Datenpfad



Aber, gibt es Fehler hier?

Steuersignale für Pipelined-Datenpfad



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

Abhängigkeiten zwischen Pipeline-Stufen (*hazards*)

TECHNISCHE UNIVERSITÄT DARMSTADT

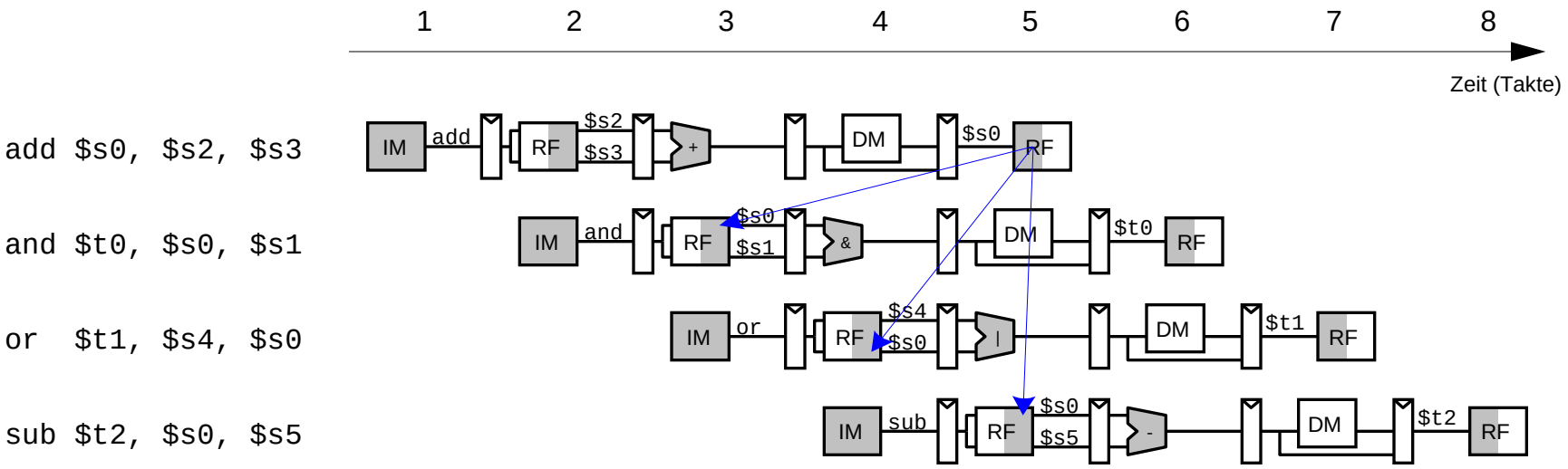
Treten auf wenn eine

- 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

Data Hazard



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

Umgang mit Data Hazards

TECHNISCHE UNIVERSITÄT DARMSTADT

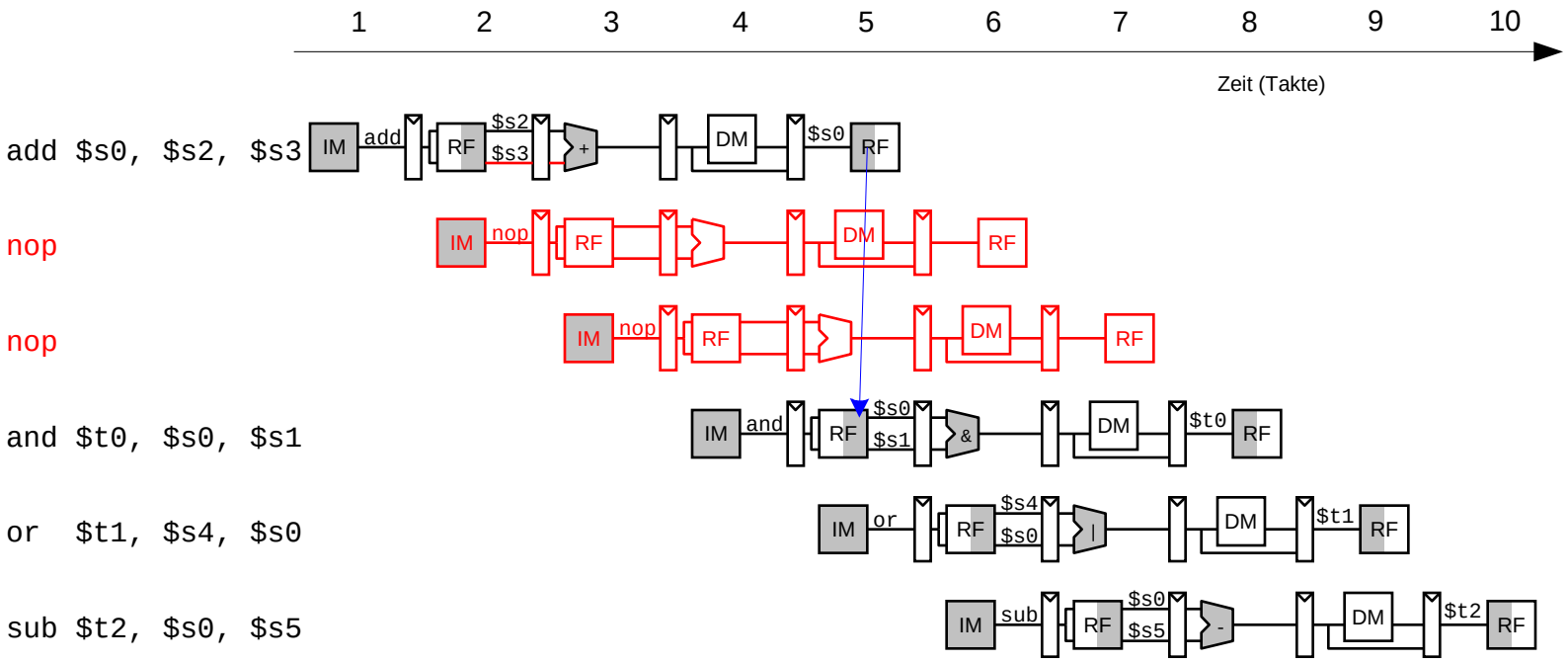
Möglichkeiten:

1. Plane Wartezeiten von Anfang an ein
   - Füge nops zur Compile-Zeit ein
   - *scheduling*
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

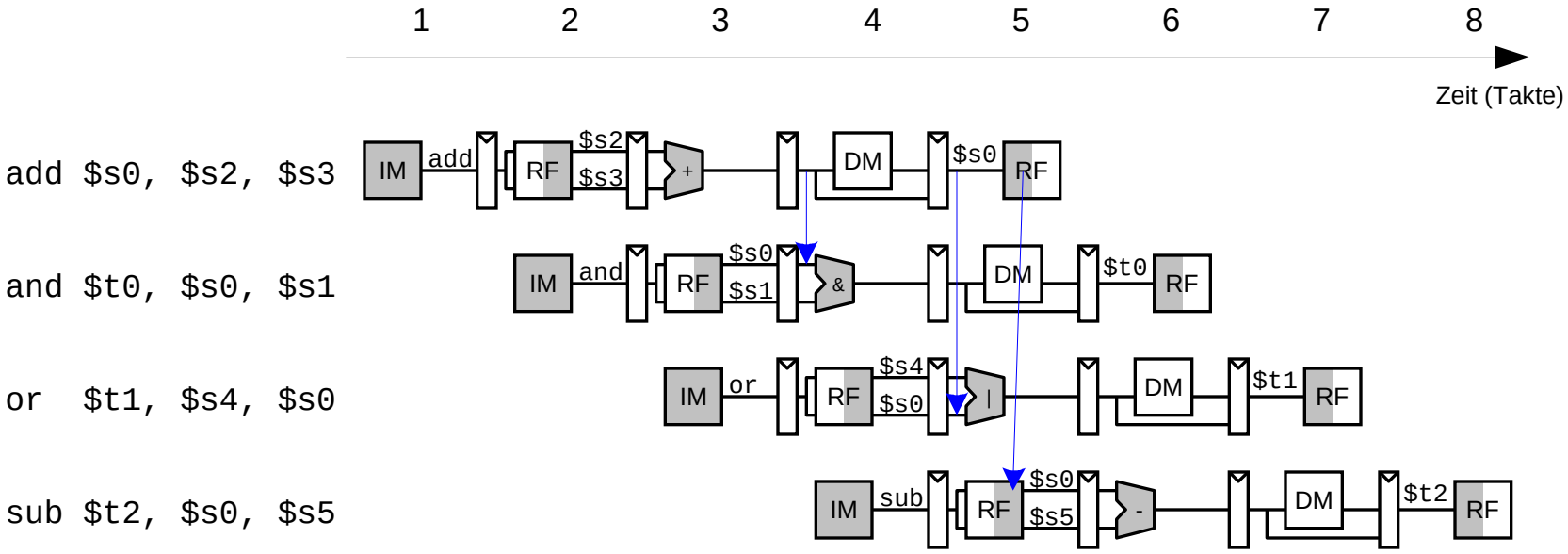
TECHNISCHE UNIVERSITÄT DARMSTADT

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

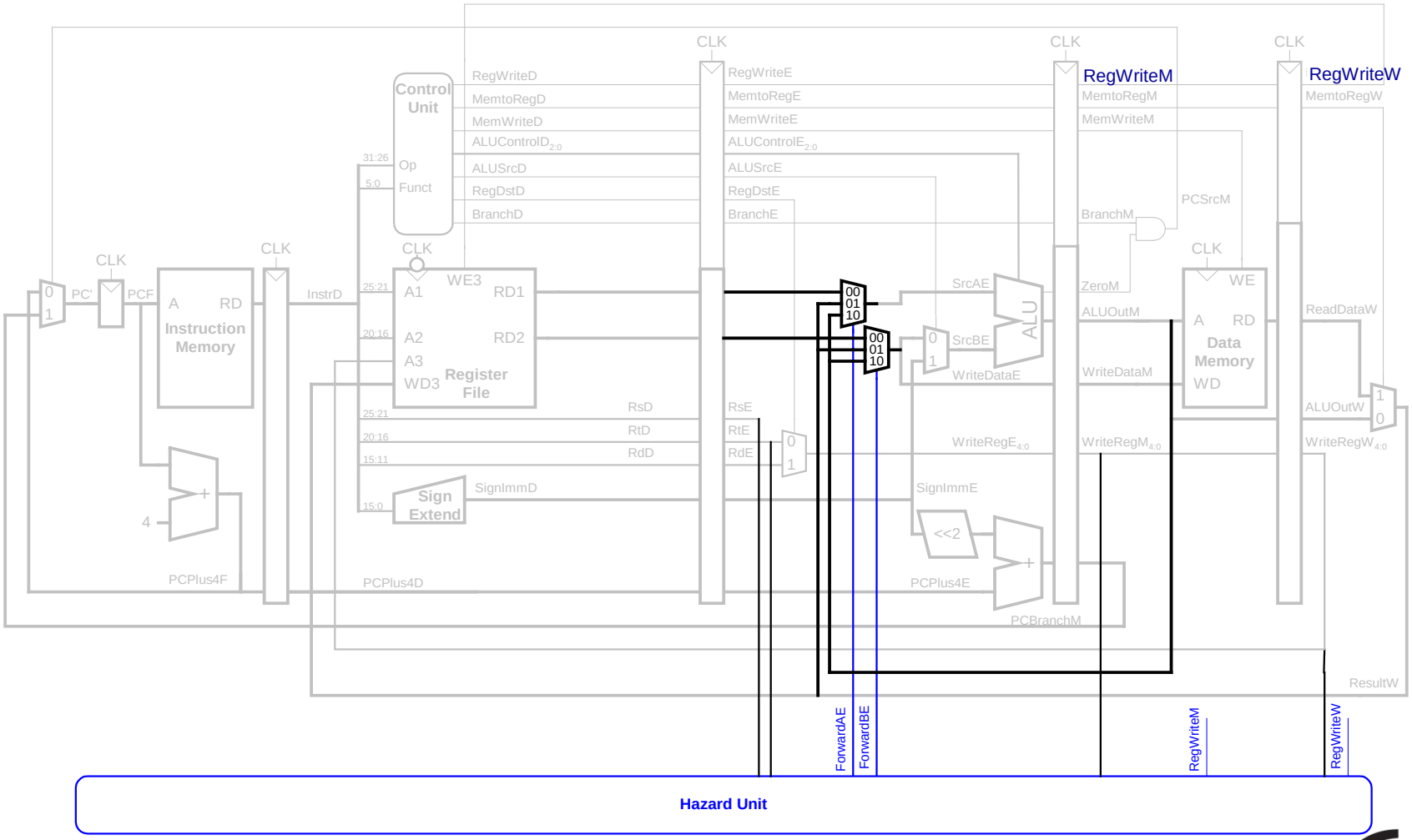


Embedded Systems & Applications

Data Forwarding: “Abkürzungen” einbauen



Data Forwarding: “Abkürzungen” einbauen



Data Forwarding: “Abkürzungen” einbauen

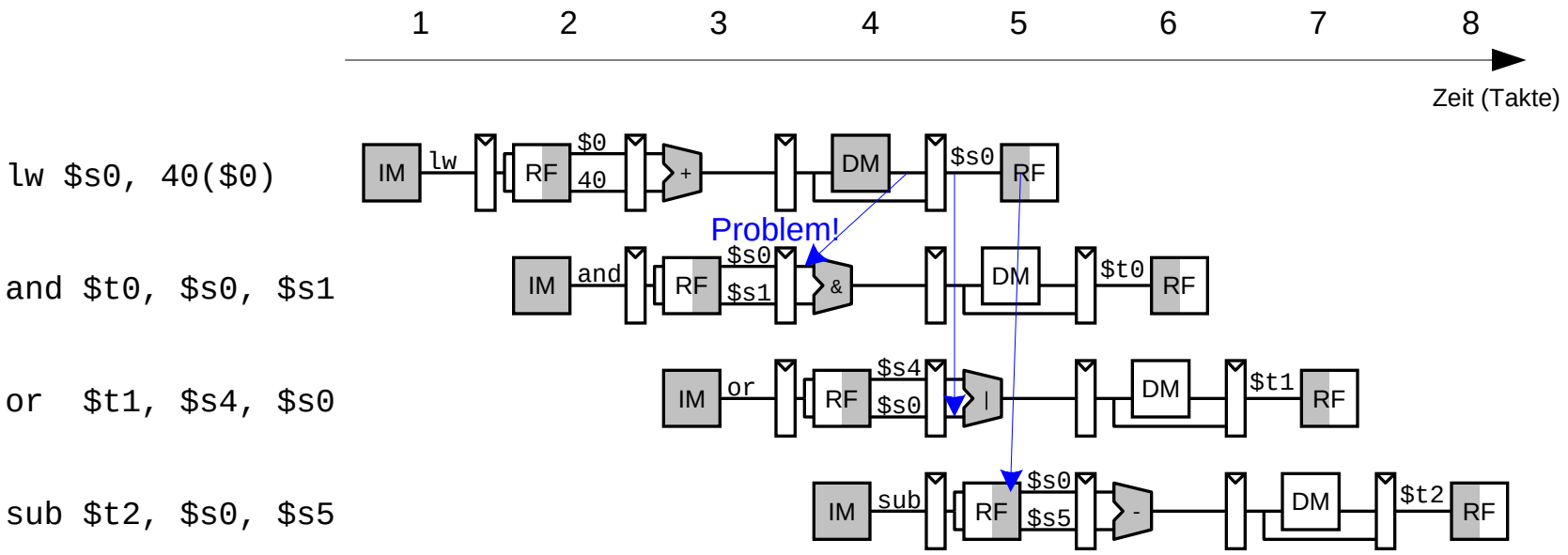
TECHNISCHE UNIVERSITÄT DARMSTADT

“Abkürzung” zur Execute-Stufe von

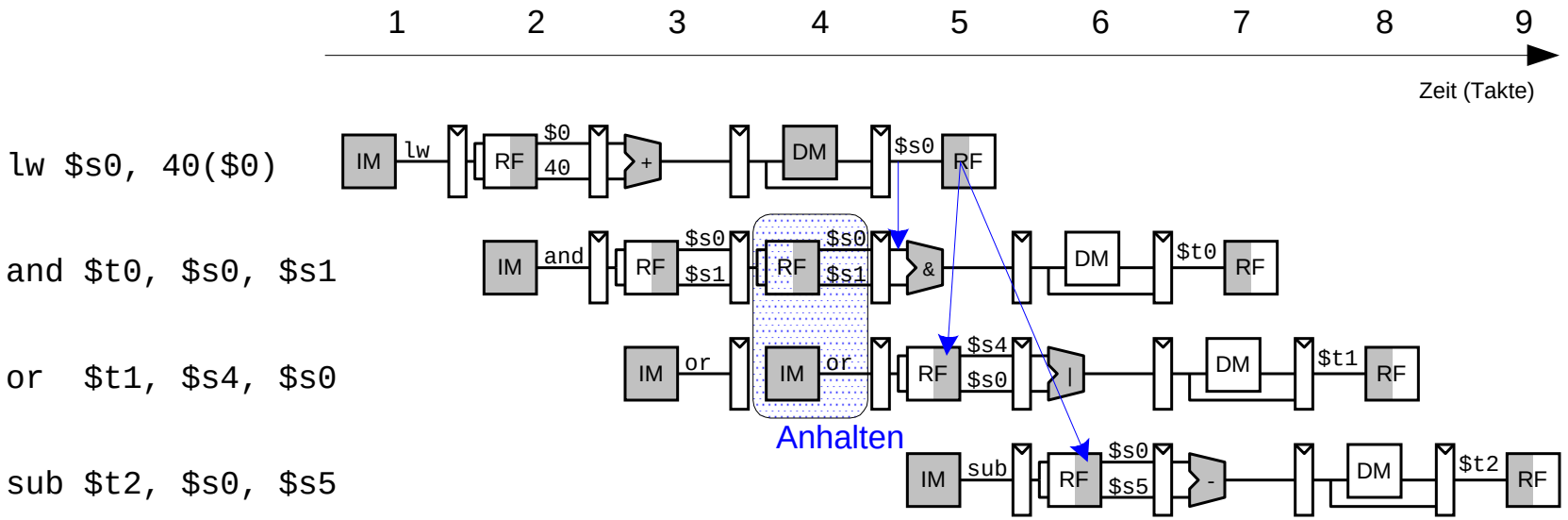
- Memory-Stufe oder
- Writeback-Stufe

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

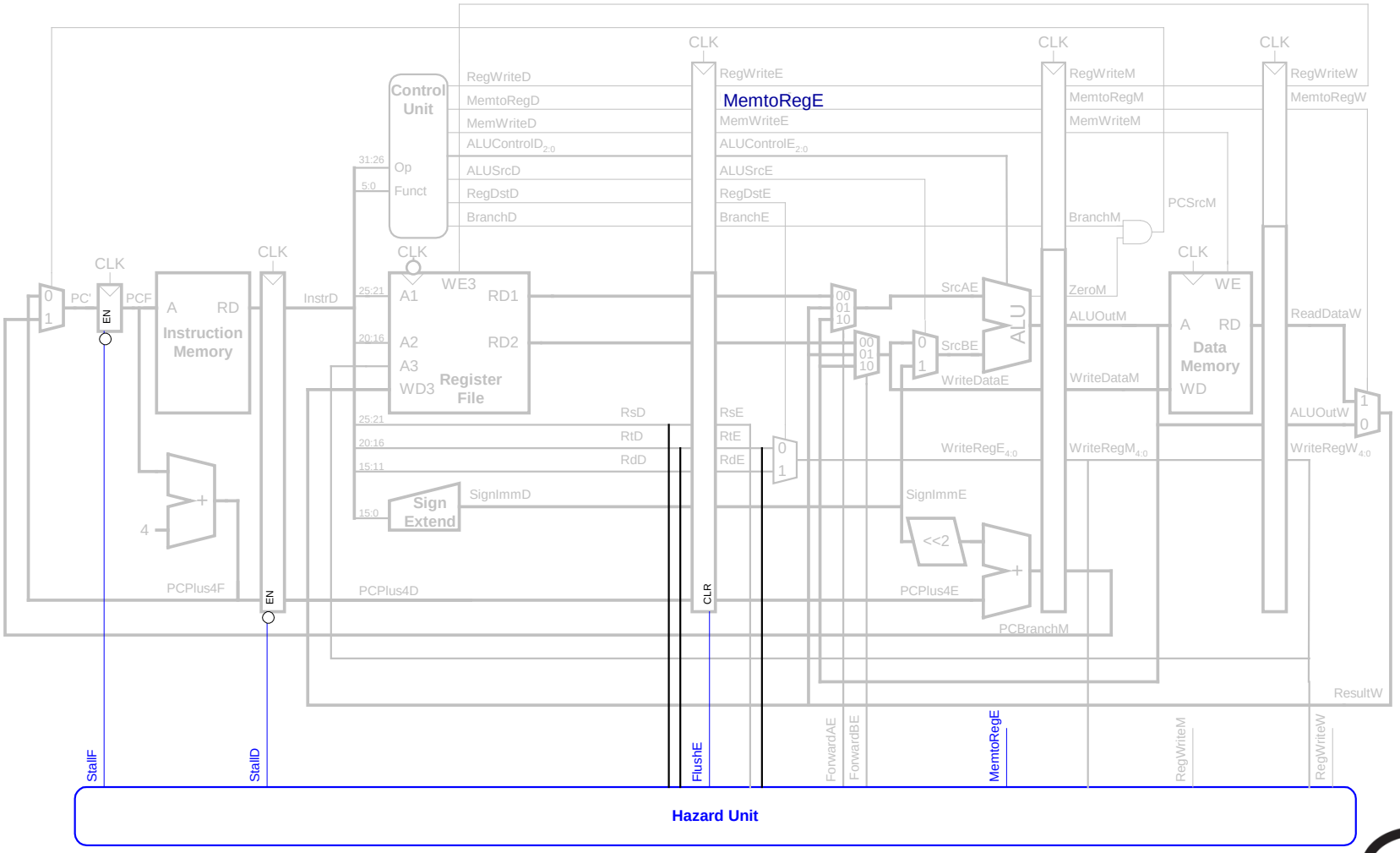
Anhalten des Prozessors (*stalling*)



Anhalten des Prozessors (*stalling*)



Erweiterung der Hazard-Einheit für Stalling



Behandlung von Stalling in Hazard-Einheit

TECHNISCHE UNIVERSITÄT DARMSTADT

Stalling-Logik:

```
lwstall =
```


Embedded Systems & Applications

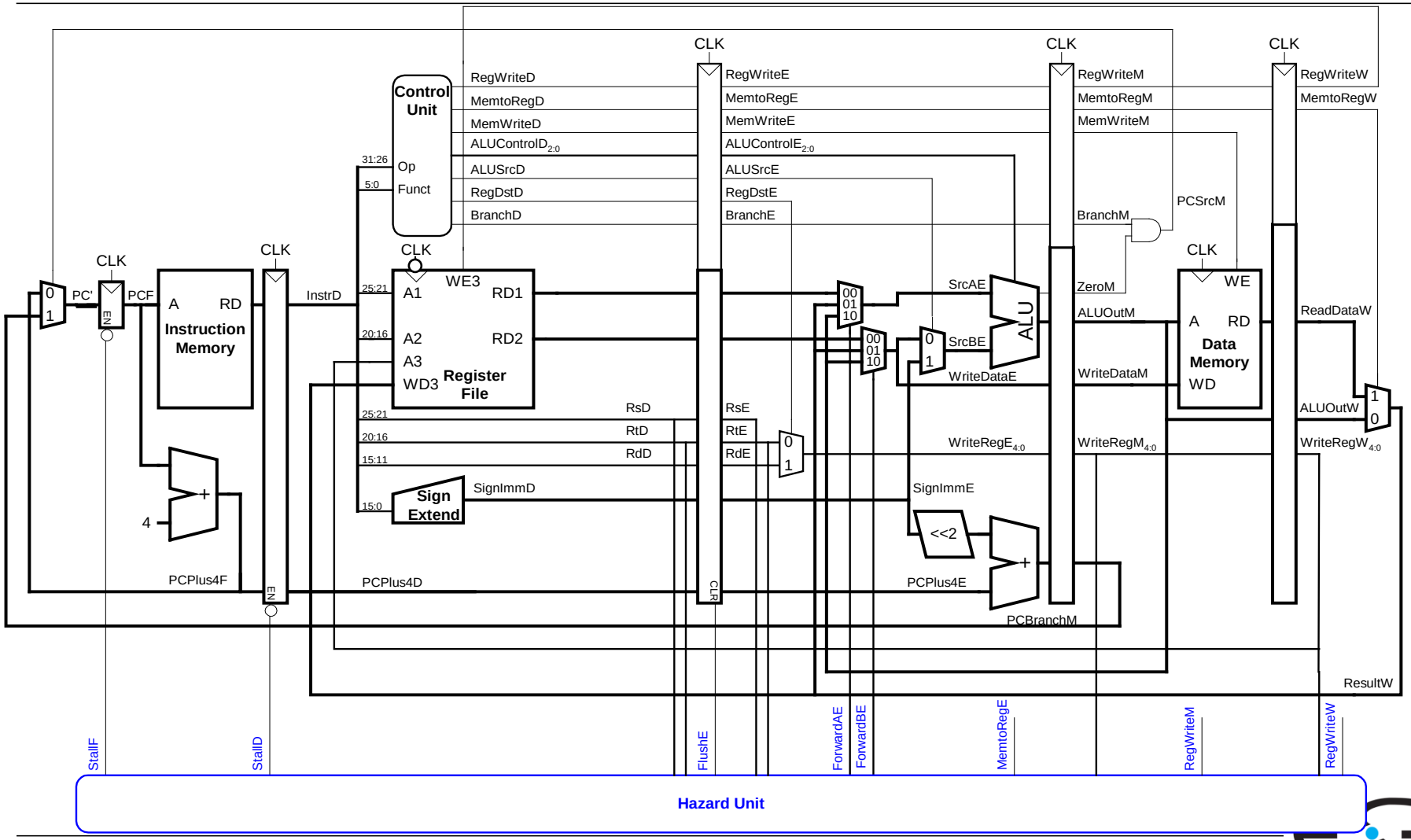
Control Hazards

TECHNISCHE UNIVERSITÄT DARMSTADT

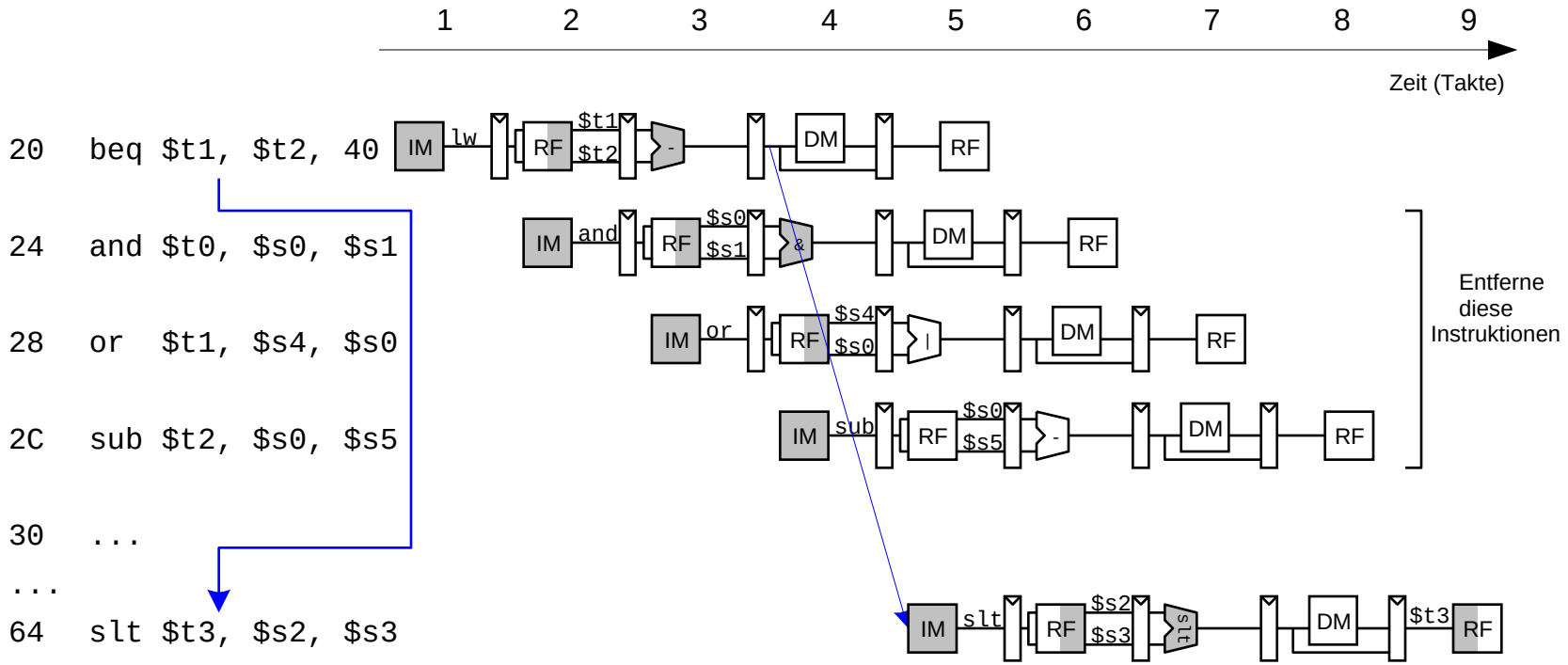
beq:

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

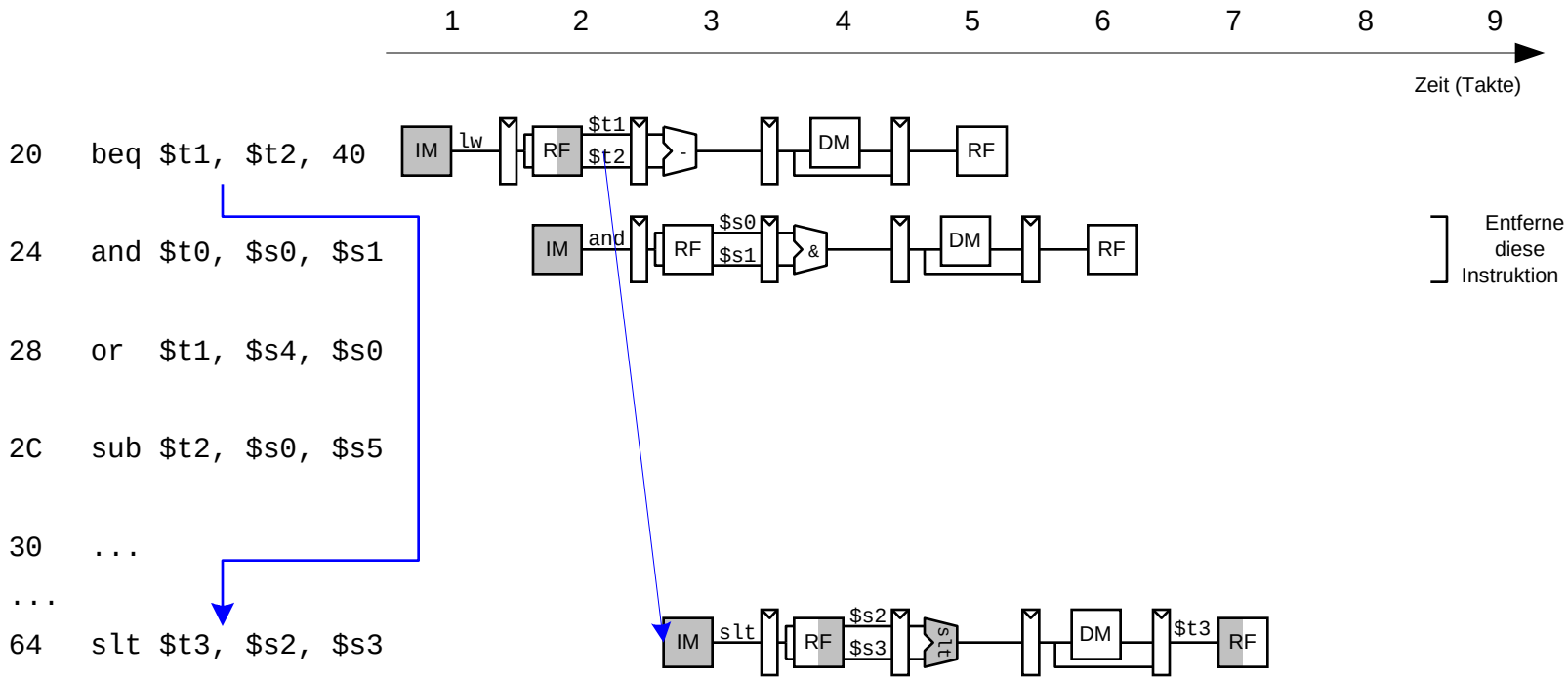
Control Hazards: Ursprüngliche Pipeline



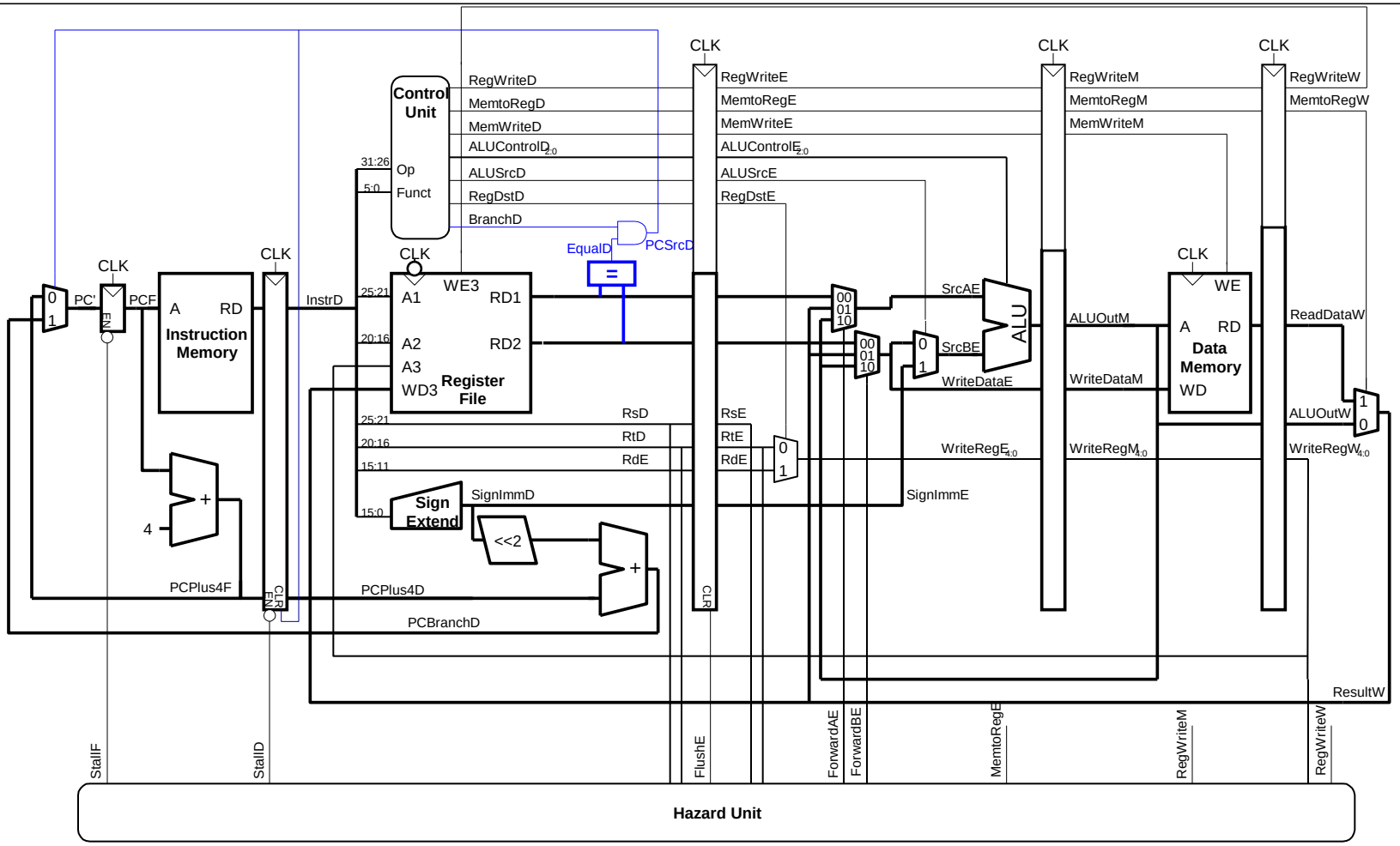
Beispiel: Control Hazards



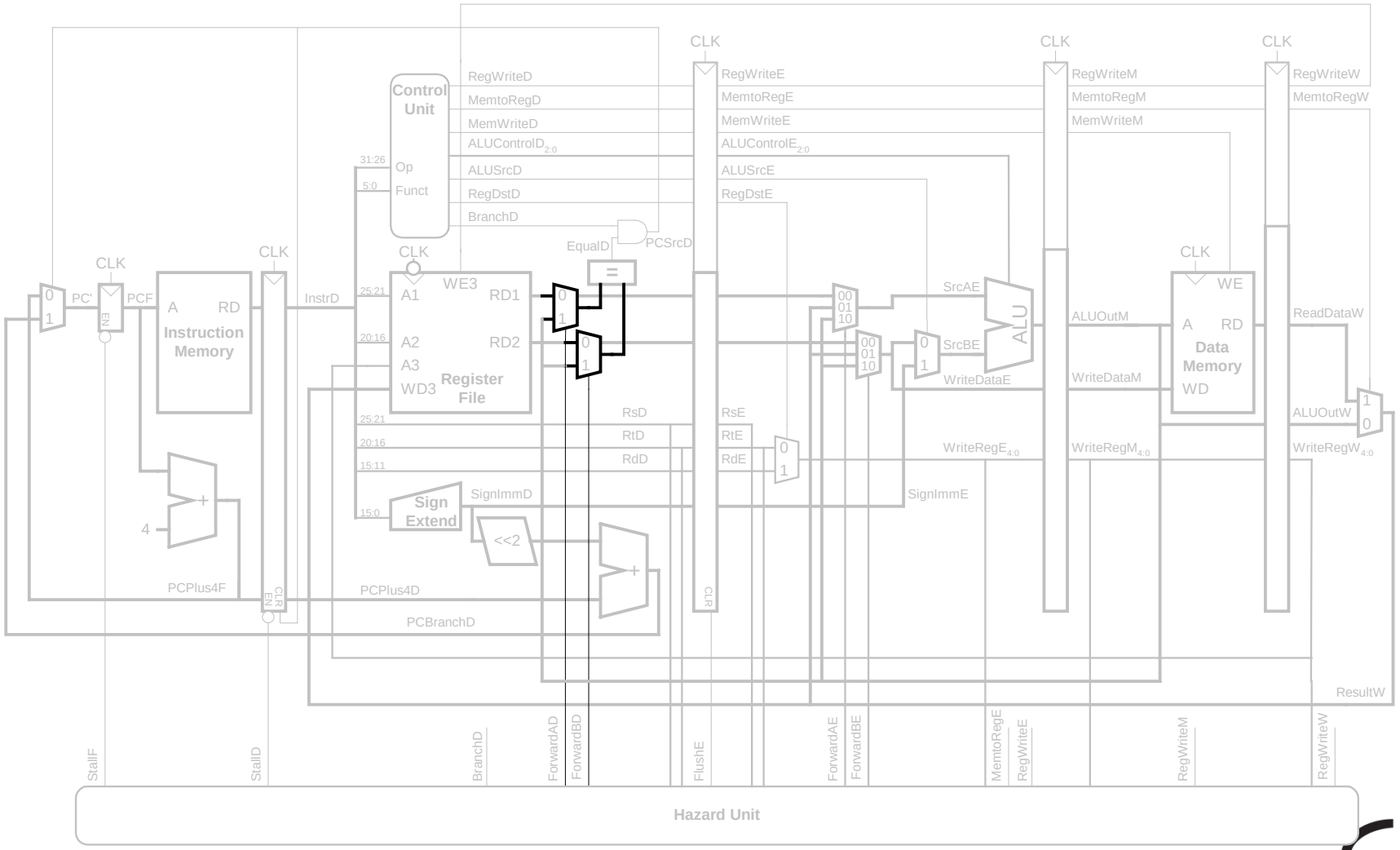
Auflösen von Control Hazards durch frühere Sprungentscheidung



Control Hazards: Ansatz “Frühere Sprungentscheidung”



Berücksichtige neue Data Hazards



Frühe Sprungentscheidung: Benötigte Logik für Forwarding und Stalling

TECHNISCHE UNIVERSITÄT DARMSTADT

- Forwarding-Logik:

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

- Stalling-Logik:

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

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

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

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

Kritischer Pfad des Pipelined-Prozessors:

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

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

Wiederholung: Ausnahmebehandlung (*exceptions*)

TECHNISCHE UNIVERSITÄT DARMSTADT

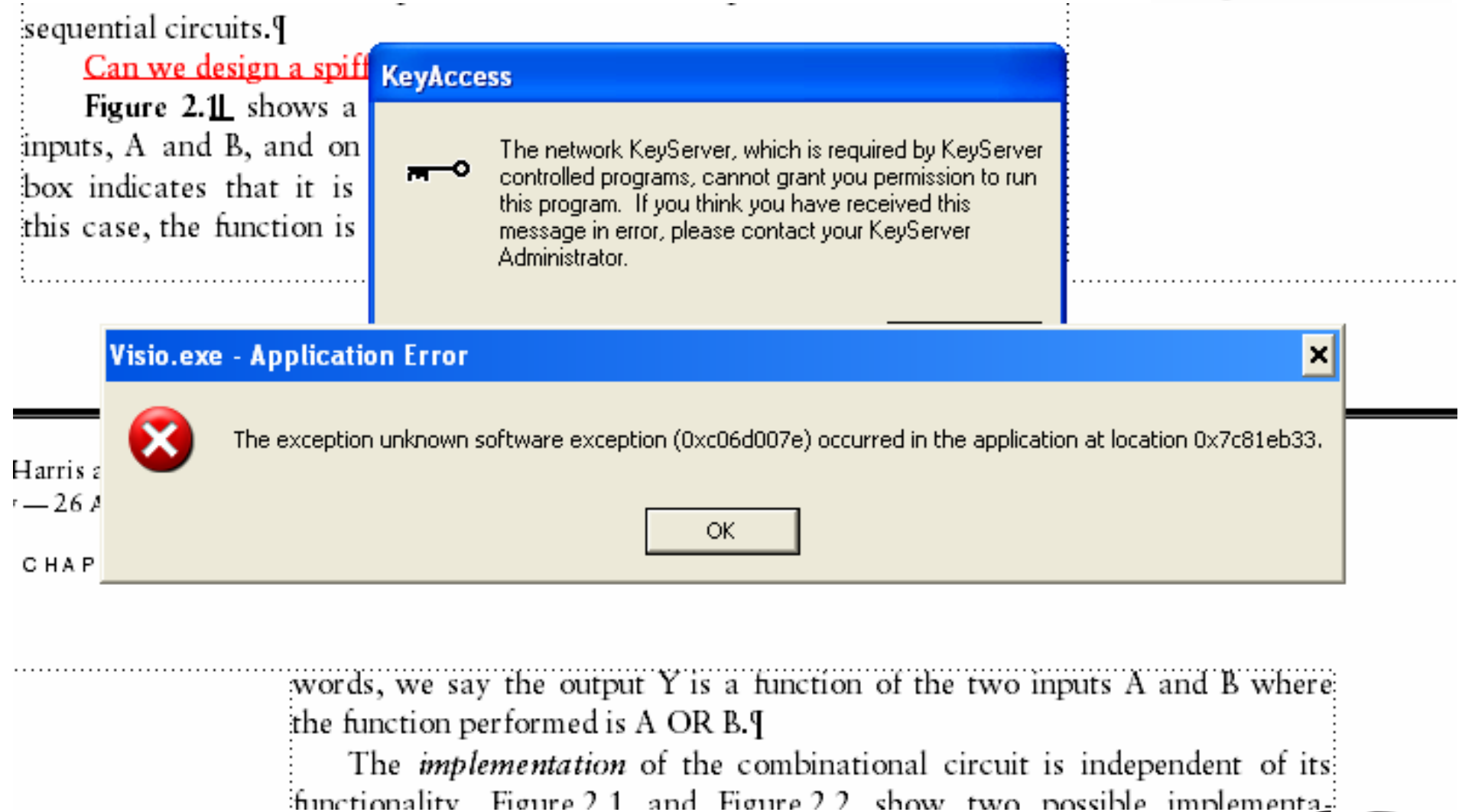
Außerplanmäßiger Aufruf der Ausnahmebehandlungsroutine

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

Beim Auftreten einer Ausnahme:

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

Beispiel für Ausnahme



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

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

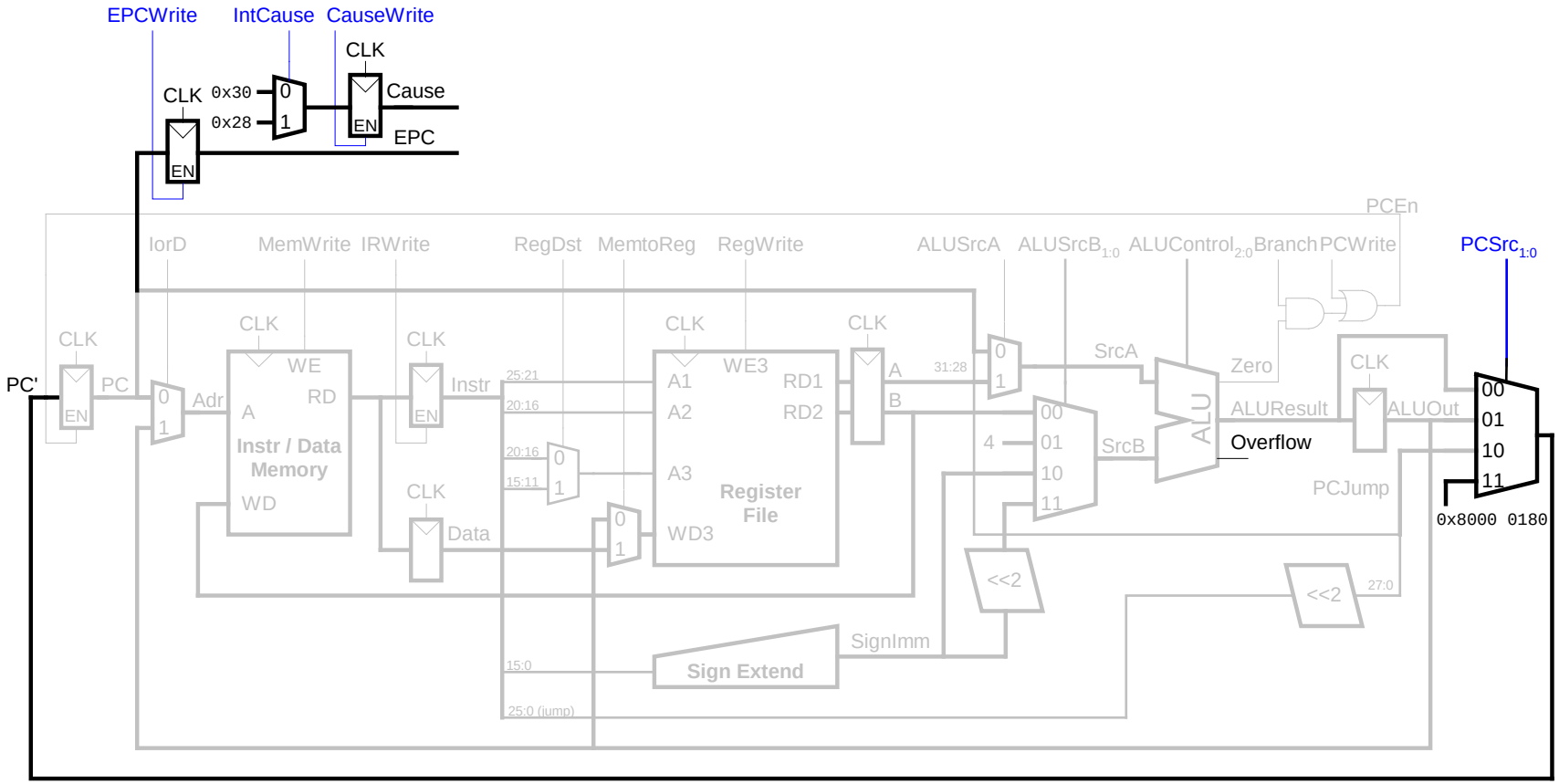
Auswahl von Ausnahmeursachen

TECHNISCHE UNIVERSITÄT DARMSTADT

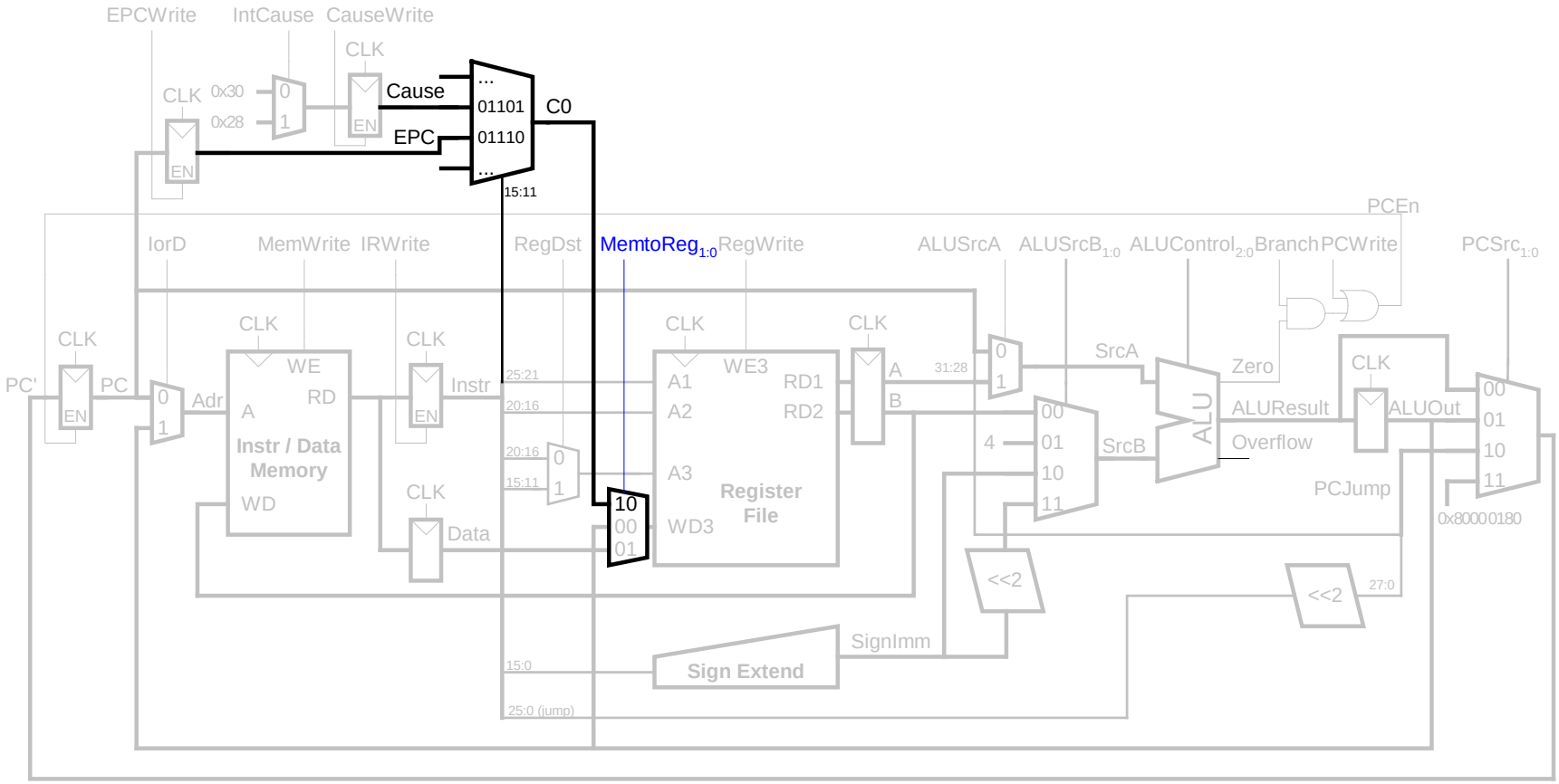
Ausnahme	Cause (Ursache)
Hardware Interrupt	0x00000000
System Call	0x00000020
Breakpoint / Division durch 0	0x00000024
Unbekannte Instruktion	0x00000028
Arithmetischer Überlauf	0x00000030

Ziel: Erweitere den Mehrtaktprozessor um Behandlung der letzten beiden Ausnahmen

Hardware für Ausnahmebehandlung: EPC und Cause



Hardware für Ausnahmebehandlung: `mfc0`



Steuerwerk-FSM erweitert um Ausnahmen