Universität Bamberg

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG · UNIVERSITY OF BAMBERG

Literatur:
Harris [2], Kap. 7

GRABS: Grundlagen der Rechnerarchitektur und Betriebssysteme

Vorlesung 7:
Von Pipelines und Kontrollpfaden

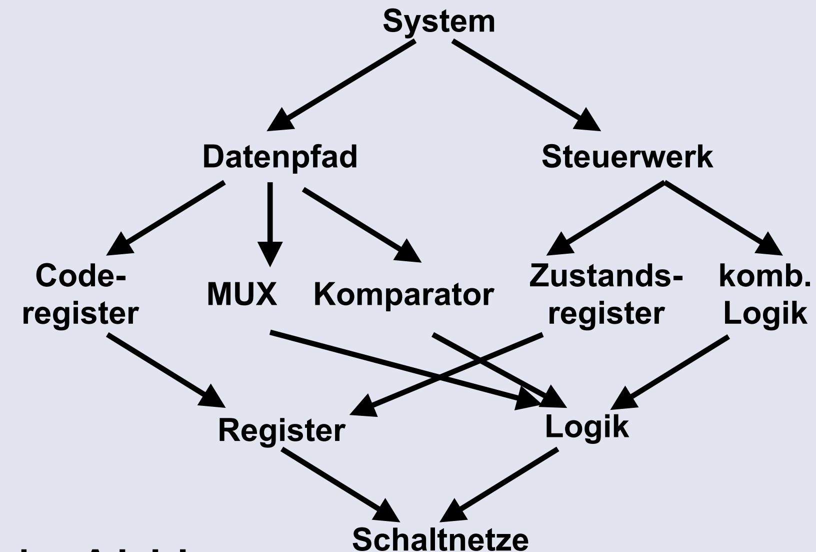
Michael Engel (michael.engel@uni-bamberg.de)
Lehrstuhl für Praktische Informatik, insbes. Systemnahe Programmierung
https://www.uni-bamberg.de/sysnap

Licensed under CC BY-SA 4.0
unless noted otherwise

CC BY SA

Prozessorstruktur

Universität Bamberg



Bei RISC-Prozessoren:

- Befehlssatzarchitektur (Instruction Set Architecture, ISA) direkt in Hardware implementiert
- **Datenpfad:** Teil des Prozessors, der die Hardware zur Durchführung der Instruktionen enthält
  - Sammlung aus Rechenwerken, wie ALU oder Multiplizierer, die Datenverarbeitungsoperationen, Register und Busse
- **Steuerwerk:** Teil des Prozessors, der den Datenpfad für die jeweilige Instruktion konfiguriert
  - Erzeugt Steuersignale zur Konfiguration der Datenpfadkomponenten und des Routings im Datenpfad durch Multiplexer

Prozessorzustand des 32 Bit RISC-V

Jede Instruktion liest und aktualisiert diesen Zustand bei der Ausführung:

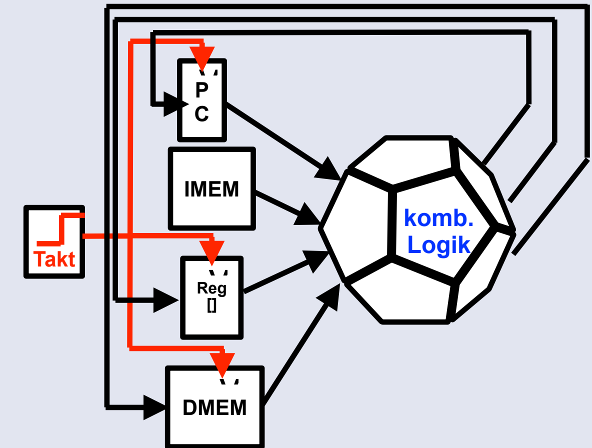
- **Register** (x0..x31)
  - *Register file* (or regfile) R hat 32 registers zu 32 Bit: R[0]…R[31]
    - Erstes gelesenes Register (*source*) in rs1-Feld der Instruktion
    - Zweites gelesenes Register in rs2-Feld der Instruktion
    - Schreibregister (Ziel, *destination*) in rd-Feld der Instruktion
    - Register x0 ist **immer** 0 (Schreibzugriffe zu R[0] ignoriert)
- **Programmzähler** (*program counter*, PC)
  - Speicher Adresse der aktuellen Instruktion
- **Speicher** (MEM)
  - Enthält Instruktionen und Daten in einem gemeinsamem 32-Bit Adressraum
  - Wir nutzen separate Speicher für Instruktionen (IMEM) und Daten (DMEM)
  - Load/store-Befehle greifen auf den Datenspeicher zu

Instruktionsausführung

Universität Bamberg

Welche Schritte müssen ausgeführt werden, um das Ergebnis einer RISC-V-Instruktion zu erhalten?

- …und was ist die korrekte Reihenfolge?



1. Hole die Instruktion: `add x10, x11, x12`
    - Aus Instruktionsspeicher IMEM
        - Adresse PC an IMEM, Ausgabe ist Instruktion
2. Welche Instruktion is es? `add`
3. Hole die Daten: lies Register `R[11]`, `R[12]`
    - Aus Register (hier) oder Instruktion (*immediate*)
4. Führe die Operation aus: berechne `R[11]+R[12]`
5. Speichere Ergebnis: schreibe in `R[10]`

Was ist eine Instruktion?

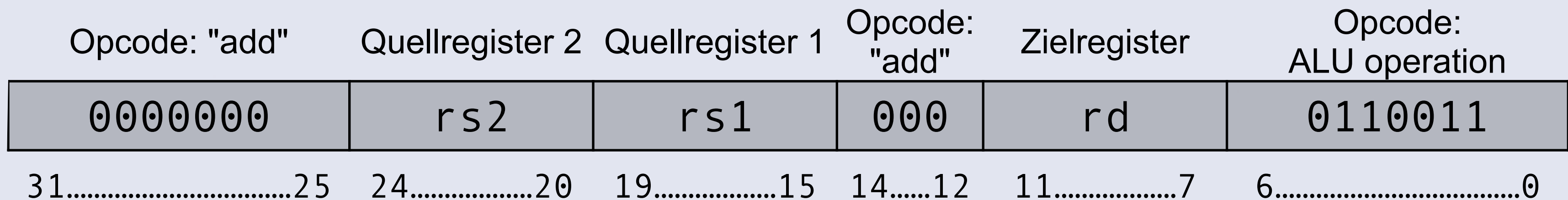
Universität Bamberg

Wie beim EA: Eingabe, codiert durch eine Menge von Bits

- RISC-V-Instruktionen (oder Befehle) sind 32 Bit breit und enthalten:
  - **Opcode** – gibt auszuführende Operation an
  - **Quellregister** – Eingabewerte für Operationen
  - **Zielregister** – Ausgabeziel für Ergebnis
  - **Direkte Parameter** – in Instruktion enthaltene Konstante

Beispiel: Addition – `add rd, rs1, rs2`

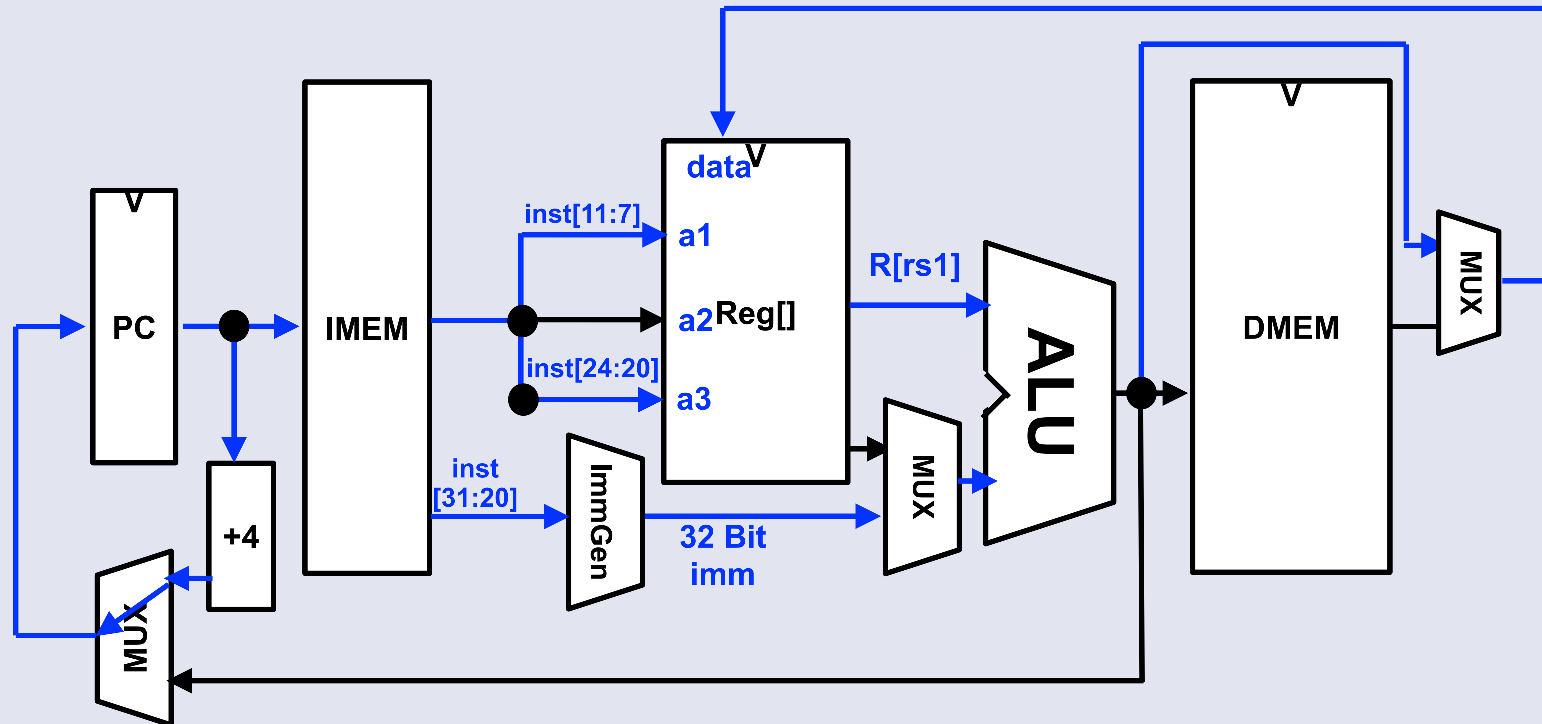
- Die Instruktion ändert Zustand des Prozessors an zwei Stellen:
  - `Reg[rd] = Reg[rs1] + Reg[rs2]`
  - `PC = PC + 4`



Opcode: "add"	Quellregister 2	Quellregister 1	Opcode: "add"	Zielregister	Opcode: ALU operation
0000000	rs2	rs1	000	rd	0110011
31……25	24……20	19……15	14……12	11……7	6……0

Binäre Codierung der `add`-Instruktion bei RISC-V

Datenpfad eines einfachen RISC-V



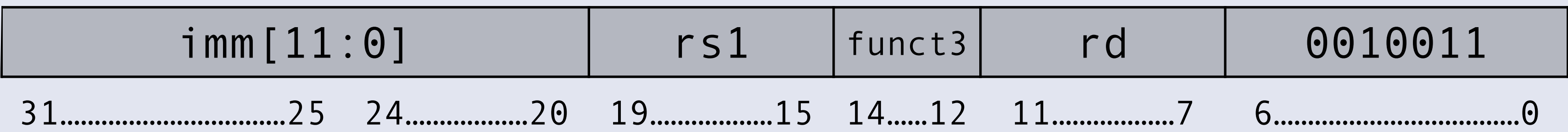
1. Instruktion holen
2. Decodieren/ Register lesen
3. Ausführen
4. Speicher
5. Register schreiben

Immediate-Instruktionen

```
addi x15, x1, -50
```

Datenpfad

PC
+4
MUX
IMEM
inst[11:7]
inst[24:20]
inst
[31:20]
ImmGen
data
a1
a2
a3
Reg[]
32 Bit
imm
R[rs1]
MUX
+
DMEM
MUX



imm[11:0]	rs1	funct3	rd	0010011
31.........25 24.........20	19.........15	14.....12	11.........7	6.........0

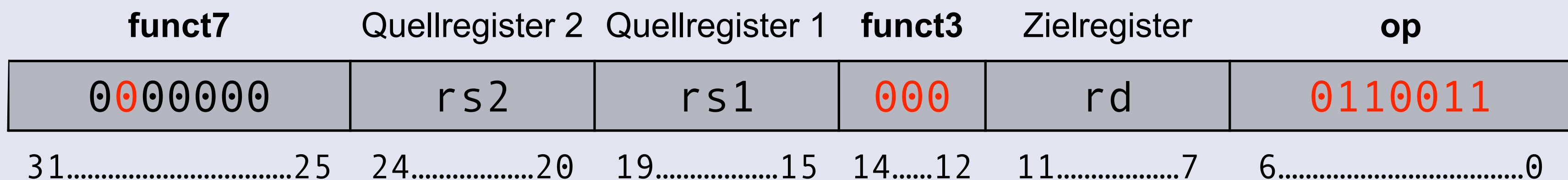
Binäre Codierung der addi-Instruktion bei RISC-V

Steuerwerks-Synthese

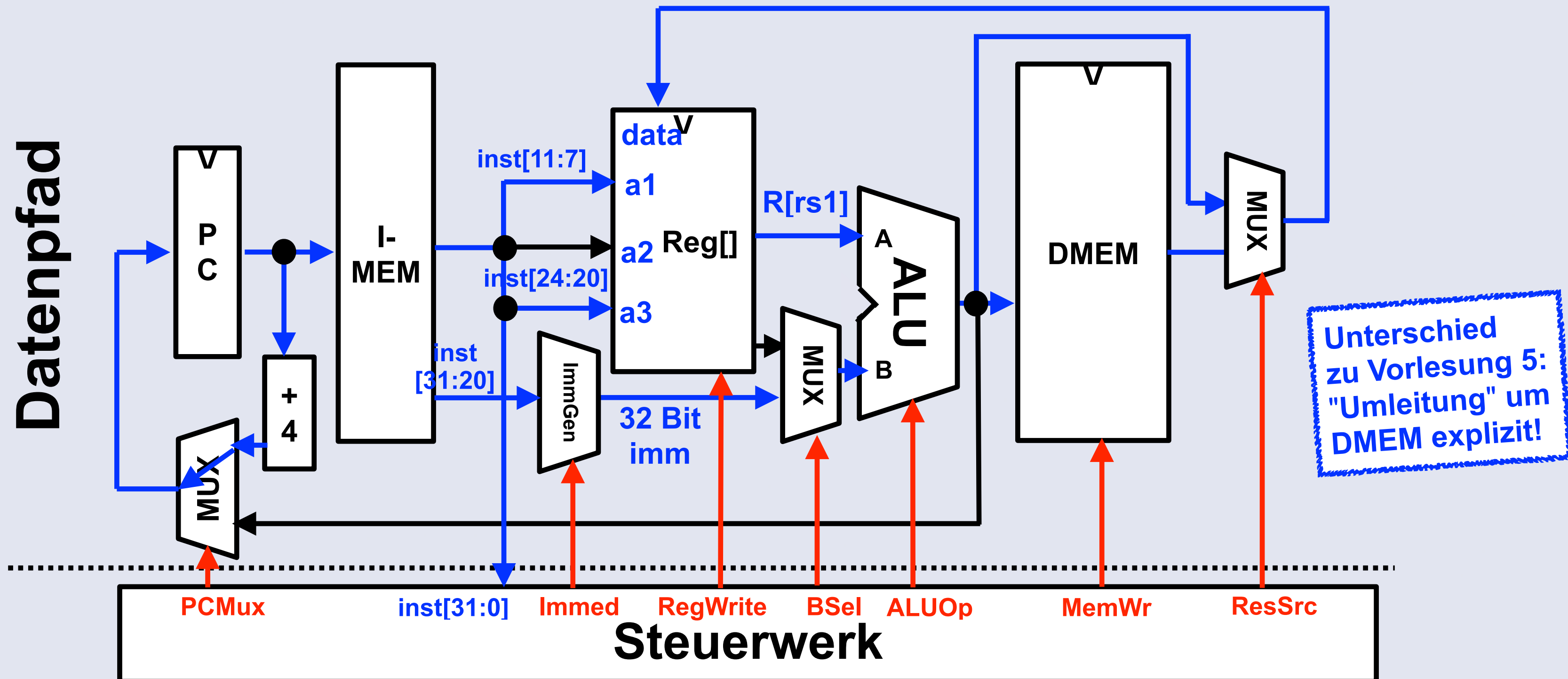
Wie entscheidet das Steuerwerk, welche Multiplexer im Datenpfad wie gesetzt werden müssen?

- Anhand der Opcode-Felder der Instruktionen: **op**, **funct3**, **funct7**

- Beim RV32I-Befehlssatz wird nur Bit 5 von funct7 genutzt
  - also müssen wir nur berücksichtigen:
    - **op** (Instruktionsbits 6:0),
    - **funct3** (Instruktionsbits 14:12),
    - und **funct7 Bit 5** (Instruktionsbit 30)



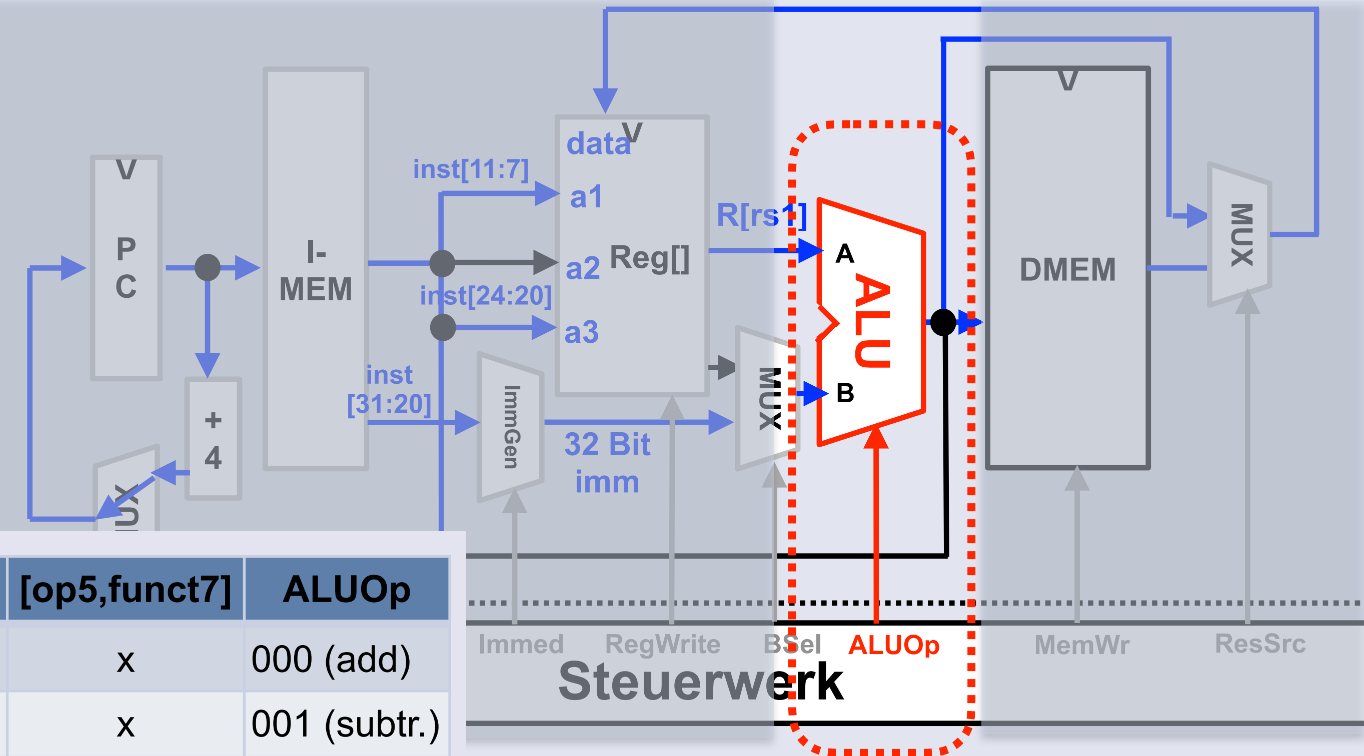
Steuerwerk: Multiplexer- und Schreibsteuerung



Instruktion	Op	RegWrite	Immed	BSel	MemWr	ResSrc	PCMux	ALUOp
lw	0000011	1	00	1	0	1	0	00
sw	0100011	0	01	1	1	x	0	00
R-Typ	0110011	1	xx	0	0	0	0	10
beq	1100011	0	10	0	0	x	1	01

Steuerwerk: ALU-Steuerung

Universität Bamberg



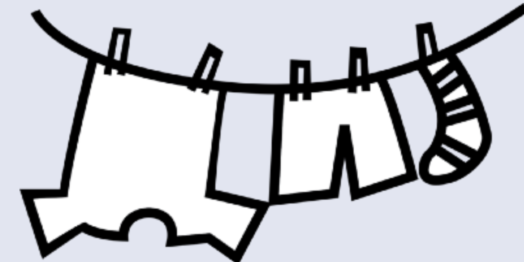
Instruktion	funct3	[op5,funct7]	ALUOp
lw, sw	x	x	000 (add)
beq	x	x	001 (subtr.)
add	000	00,01,10	000 (add)
sub	000	11	001 (subtr.)
slt	010	x	101 (slt)
or	110	x	011 (or)
and	111	x	010 (and)

…aus dem echten Leben…

Wäsche waschen besteht aus drei Schritten:

- **Waschen**
  30 Minuten/Waschladung
- **Trocknen**
  40 Minuten/Waschladung
- **Zusammenlegen**
  20 Minuten/Waschladung
- Einmal waschen, trocknen, zusammenlegen dauert 90 Minuten!
  - **Wie lange benötigen vier Wäscheladungen?**

EPA
Public Domain



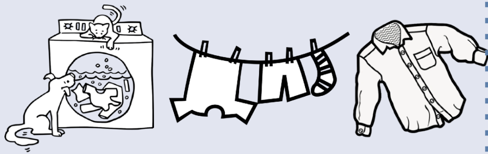
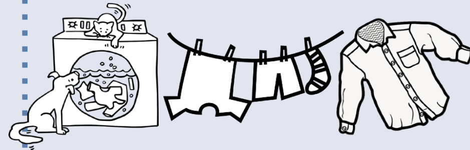
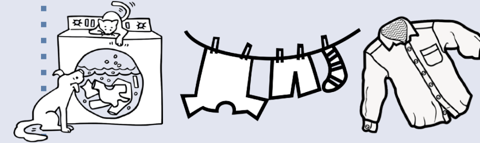
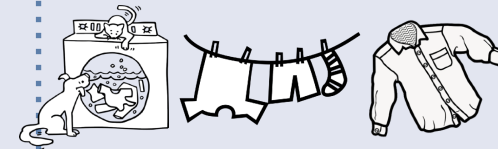
CC by
Wannapik.com

CC0

Langsam…

Universität Bamberg

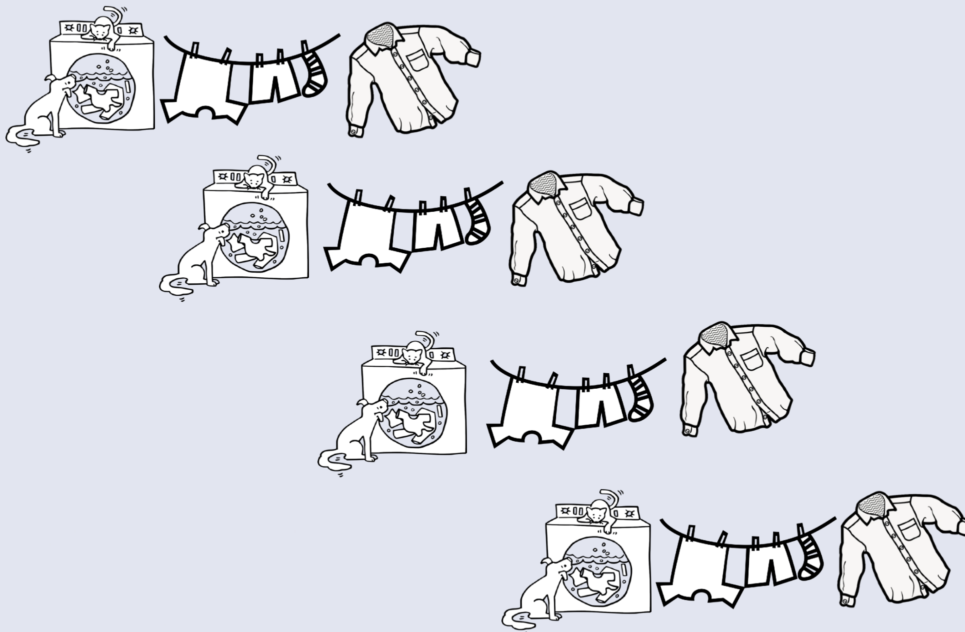
18:00	19:00		20:00		21:00		22:00		23:00		0:00
30	40	20	30	40	20	30	40	20	30	40	20

- **Sequentiell ausgeführt benötigen vier Waschladungen sechs Stunden!**

Ein Wäschefließband (Pipeline)

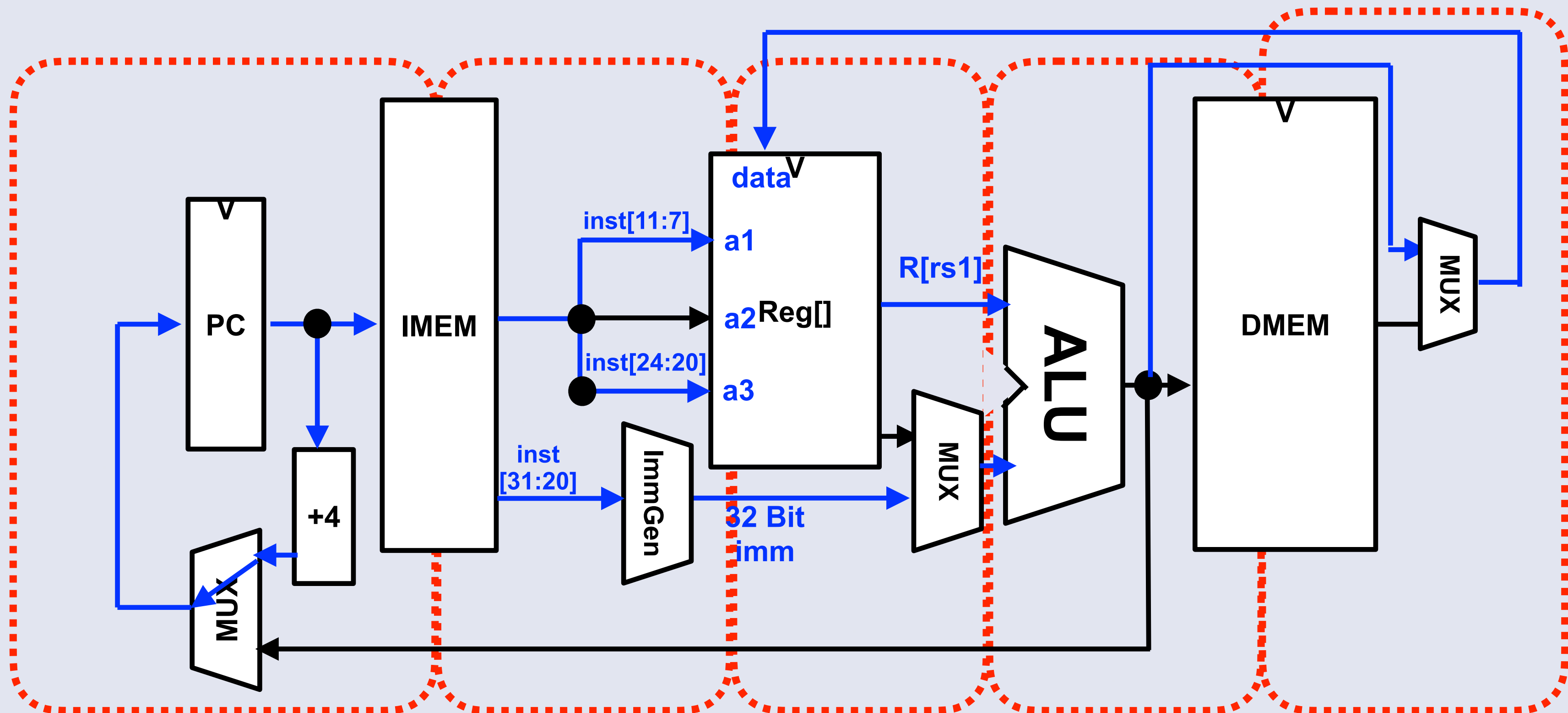
18:00	19:00		20:00	21:00	
30	40	40			
	40	40	40		
		40	40	40	
			40	40	20



- **Beobachtung:**
  Waschmaschine und Trockner waren jeweils eine Zeit lang ungenutzt
- **Idee:**
  Jede Waschladung so früh wie möglich beginnen – **Waschaufgaben überlappen!**
- Waschen mit **Pipeline** dauert **nur 3,5 Stunden**!

Pipelining im Datenpfad

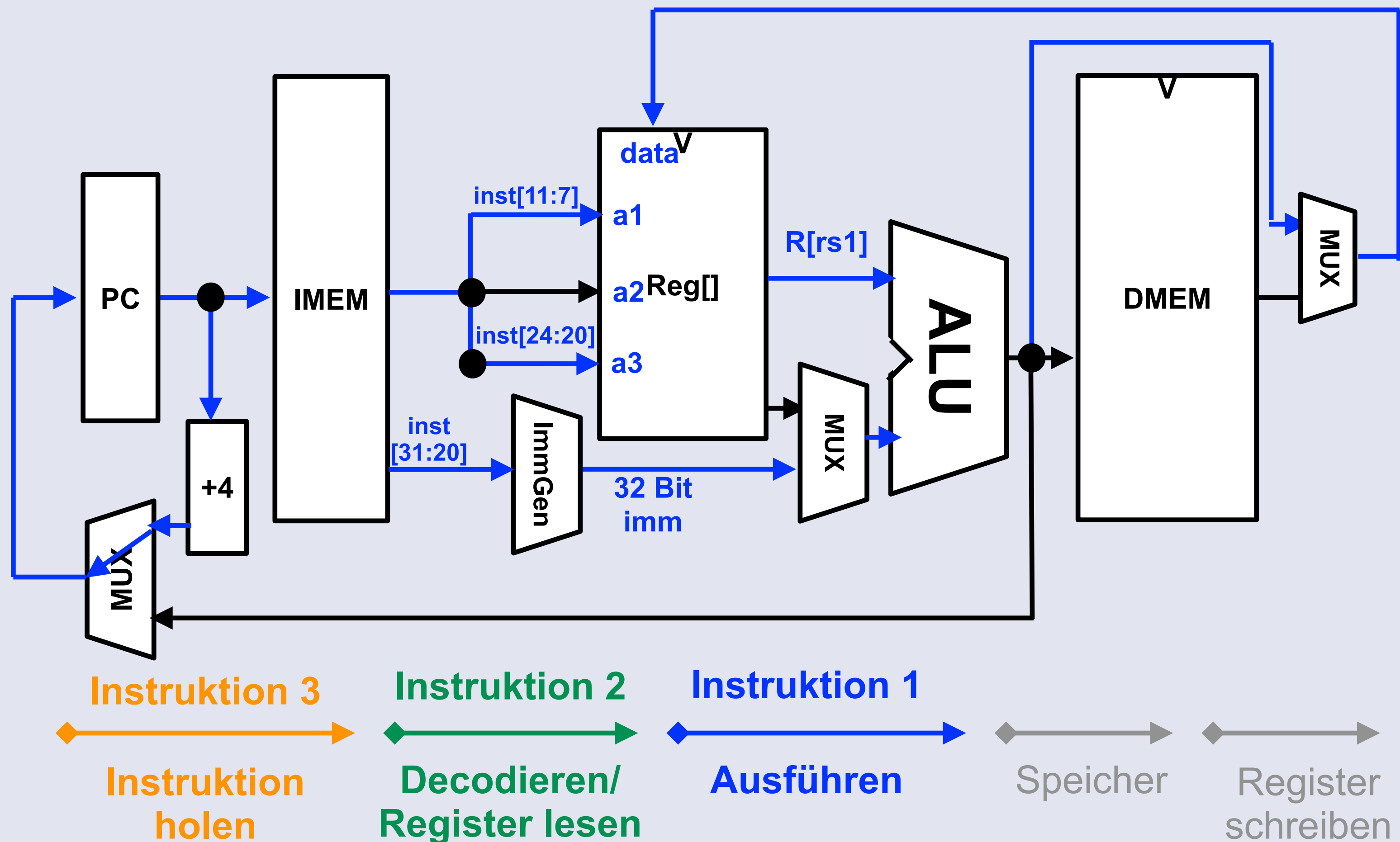
Gibt es **ungenutzte Teile** in unserem Prozessor-**Datenpfad**?



1. Instruktion holen
2. Decodieren/ Register lesen
3. Ausführen
4. Speicher
5. Register schreiben

Pipelining im Datenpfad

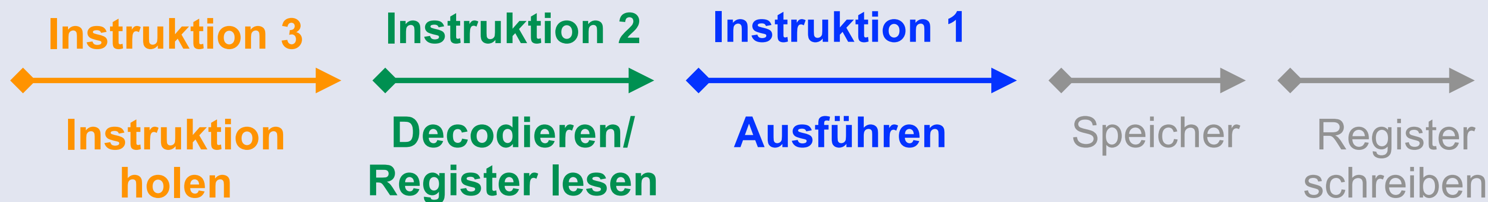
Idee: In einer Phase ungenutzte Teile des Datenpfads bereits für nachfolgende Instruktionen verwenden!



Pipelining im Datenpfad

Universität Bamberg

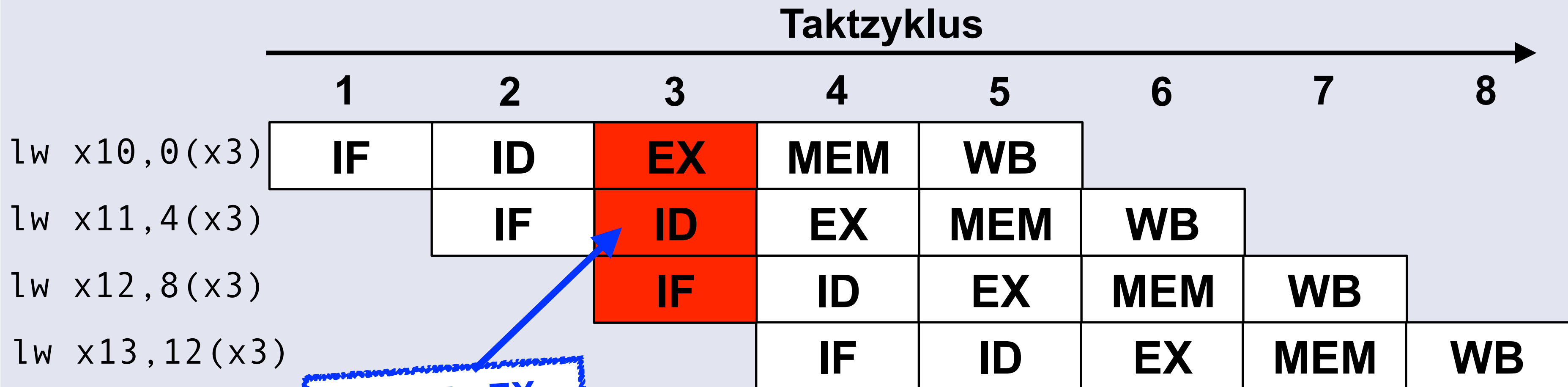
Problem: Phasen müssen **getrennt** werden: **Pipelineregister!**



Pipelining in RISC-V

Universität Bamberg

Übersichtlichere Darstellung:



1. **IF – Instruktion holen** *(Instruction Fetch)*
2. **ID – Dekodieren** *(Instruction Decode)*
3. **EX – Ausführen** *(Execute)*
4. **MEM – Speicherzugriff** *(Memory)*
5. **WB – Zurückschreiben** *(Write Back)*

Mit *Pipelinetiefe* bezeichnen wir die **Anzahl der Stufen**

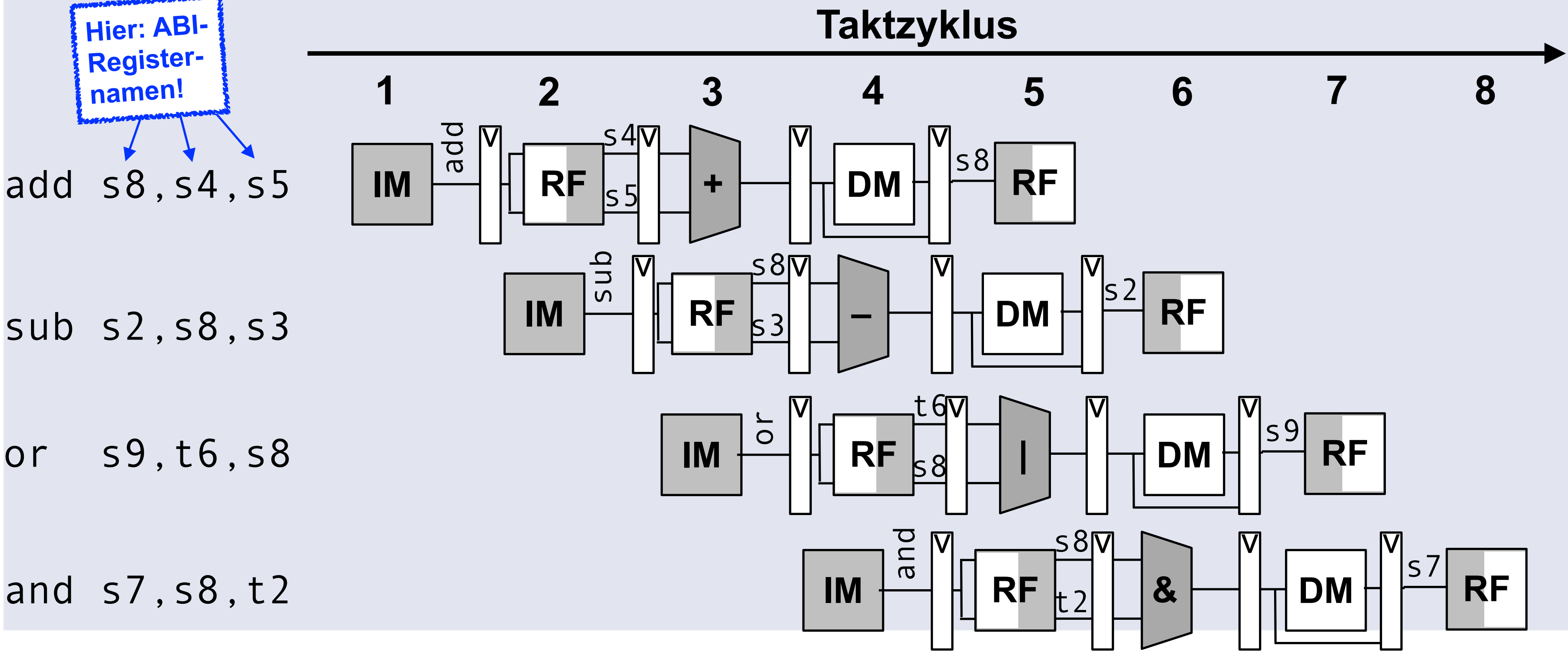
- hier: fünf Stufen

Pipelining in RISC-V

Universität Bamberg

Noch eine **alternative Darstellung**:

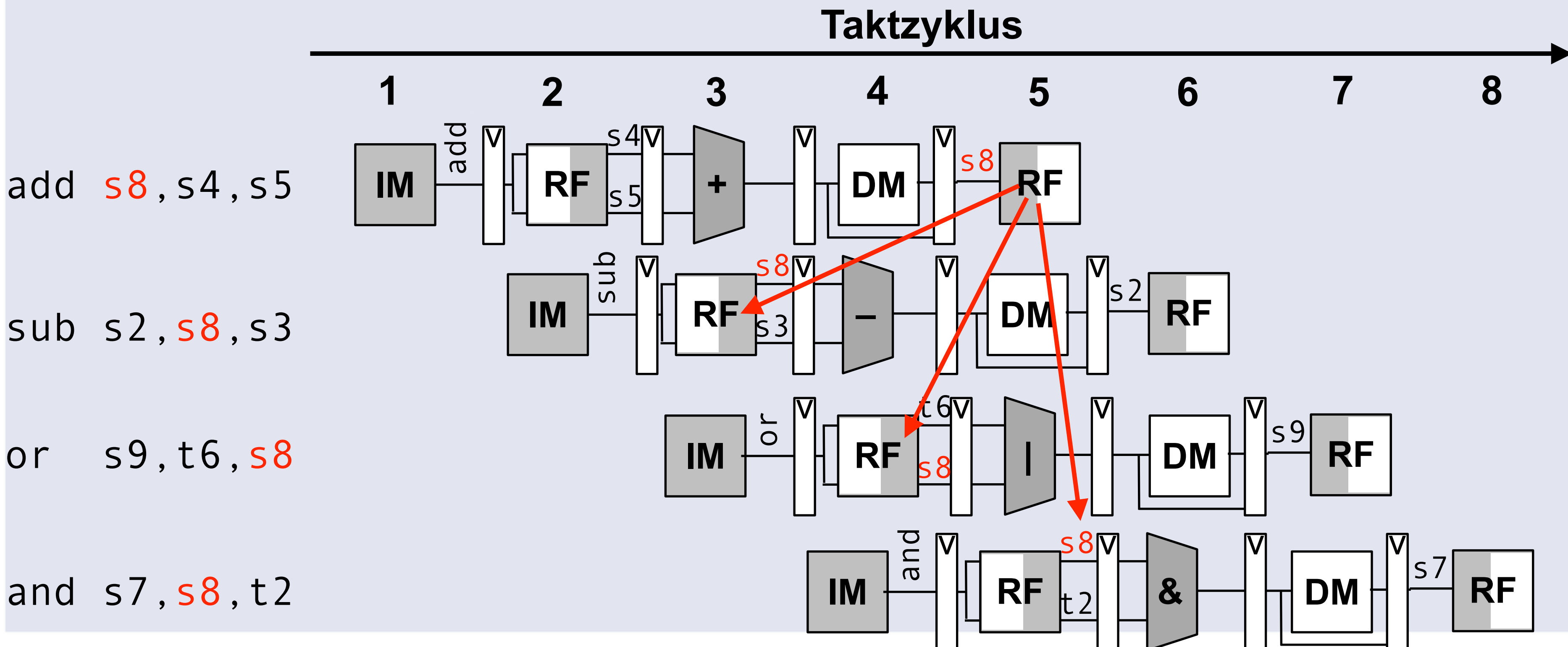
- Kombiniert vereinfachte Sicht des Datenpfads mit abstrakter Sicht auf die Prozessorpipeline
- Bei Harris & Harris [2] verwendet



Pipeline-Datenkonflikte

Diese Instruktionsfolge hat **Probleme**:

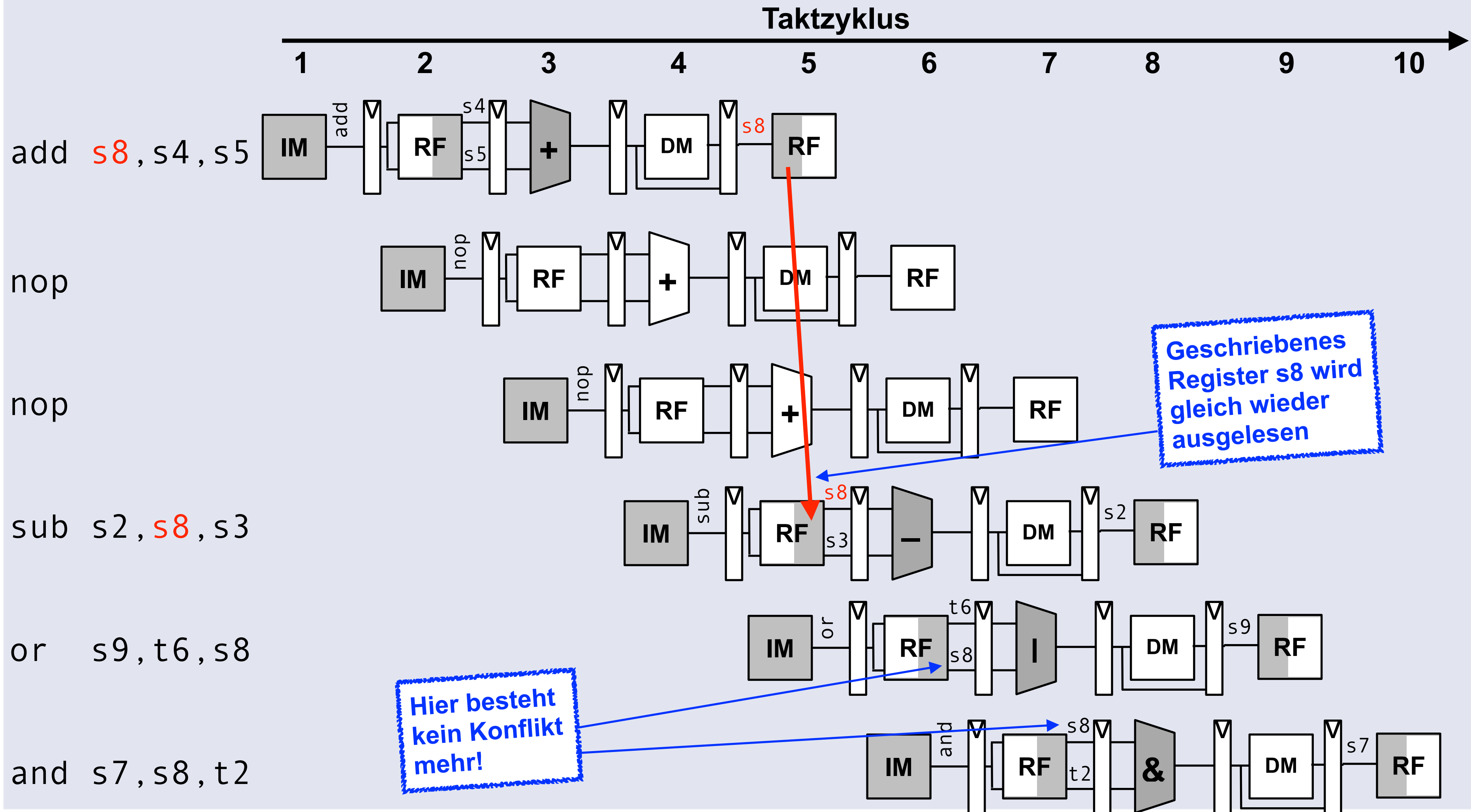
- Instruktionen verwenden Ergebnisse vorheriger Instruktionen
- Diese sind **zum Zeitpunkt, zu dem sie benötigt werden, noch nicht berechnet worden!** (hier: s8): **Konflikt!** *(hazard)*



Pipeline-Datenkonflikte lösen

Universität Bamberg

Idee: **Folgeinstruktionen verzögern**

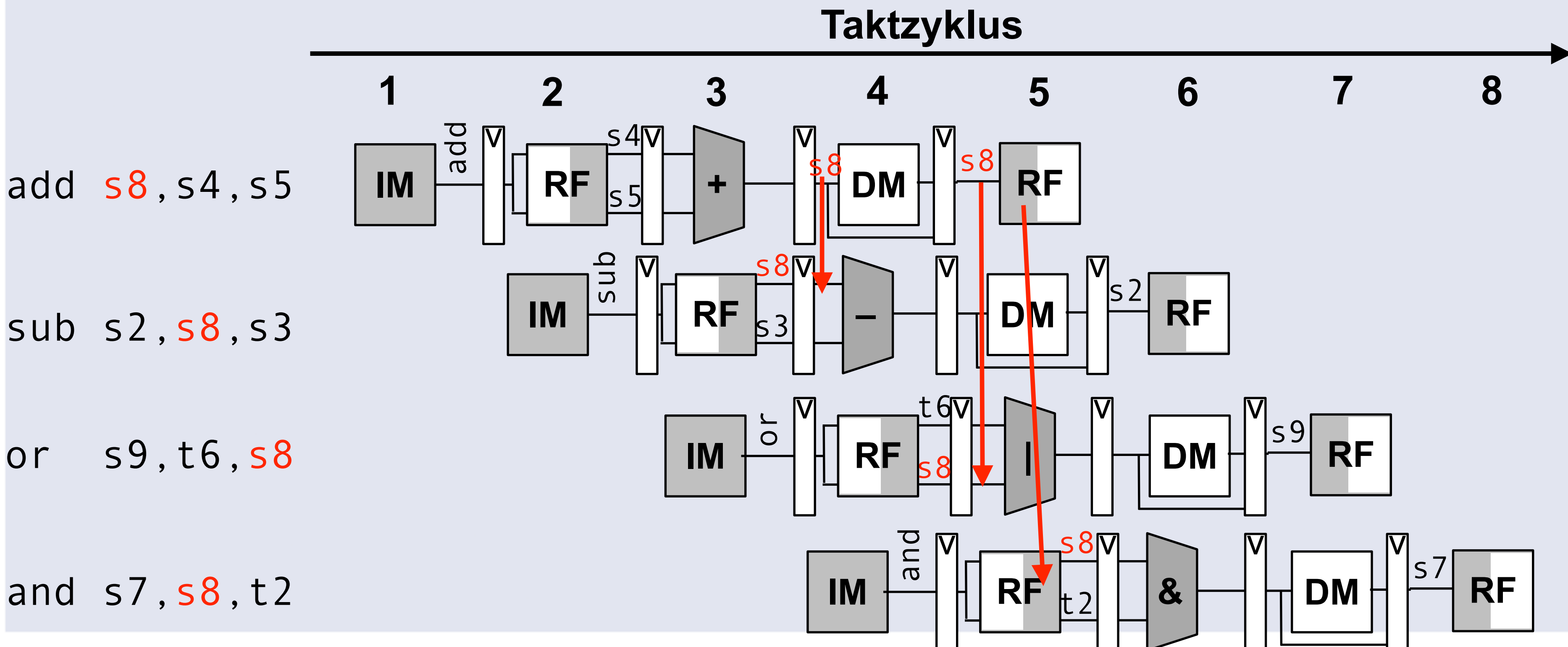


Pipeline-Datenkonflikte lösen

Universität Bamberg

Alternative Idee: "**Abkürzungen**" im Datenpfad bauen: ***forwarding***

- Direktes Weiterleiten von Registerinhalten an eine Folgeinstruktion in der Pipeline – vor Rückschreiben in Registerfile
  - Aufwändigere Hardware erforderlich

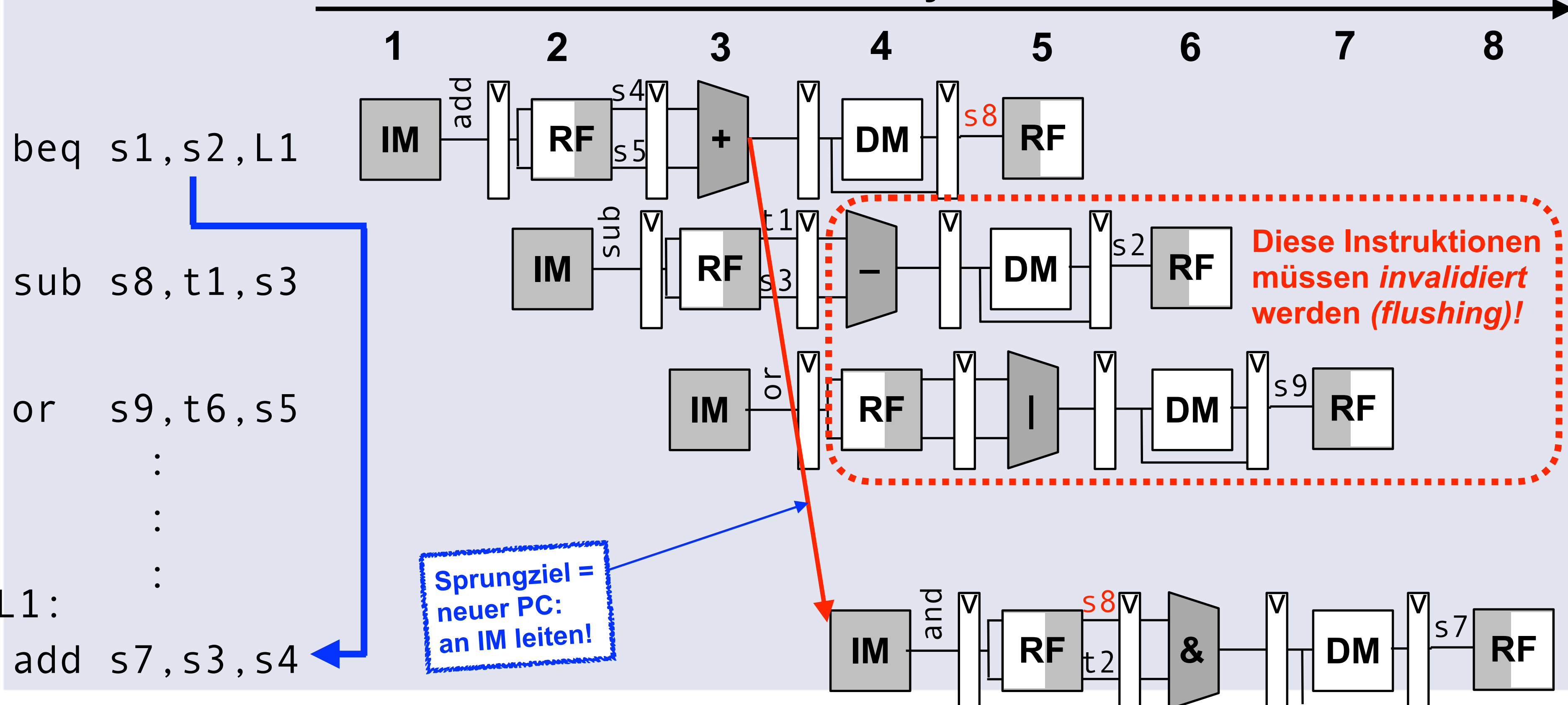


Pipeline-Kontrollkonflikte

Universität Bamberg

Sprünge bringen die Pipeline durcheinander!

- Instruktionen nach einem (genommenen) Sprung, die schon in der Pipeline sind, dürfen nicht ausgeführt werden!



Prozessor-Performance

Universität Bamberg

Latenz

- Zeit von Start bis Ende der Ausführung einer Instruktion
- wird durch Pipelining nicht verbessert!

Durchsatz

- Anzahl der pro Zeiteinheit zu Ende ausgeführter Instruktionen
- *Wird durch Pipelining verbessert!*

Maß für Durchsatz: **Zyklen pro Instruktion** (Cycles per Instr., **CPI**)

- Beim pipelined RISC-V – Ziel: **CPI = 1**!
  - Pipeline-Konflikte erhöhen die CPI
    - Flushes kosten 1 oder zwei Zyklen

Referenzen

Universität Bamberg

[1] Frank Slomka, Michael Glaß
Grundlagen der Rechnerarchitektur
Von der Schaltung zum Prozessor

[2] David Harris, Sarah Harris
Digital Design and Computer Architecture, RISC-V Edition