Digitaltechnik & Rechnersysteme

Rechnerarchitektur I

Martin Kumm

Hochschule Fulda
University of Applied Sciences

AI | Angewandte Informatik

WiSe 2023/2024

Eigene Aufgabe in Klausur?

Nach Durchsicht der (leider recht wenigen) Abgabe der »Eigenen Aufgabe« in Moodle wird hiermit bekannt gegeben, dass eine der Aufgaben nahezu unverändert in der Klausur erscheinen wird.

Übernahme der Musterlöung(en) auf der eigenen Formelsammlung sind nicht zugelassen!

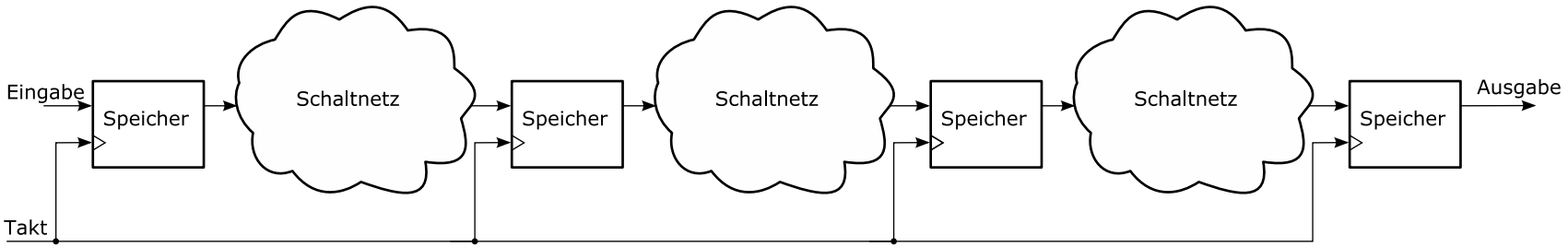
Was bisher geschah... I

- Zeitverhalten Gatter
  - Jedes Einzelgatter hat eine gewisse Laufzeit
  - In einem Schaltnetz lässt sich die (worst-case) Gesamtverzögerung für jeden Pfad ermitteln
  - Der Pfad mit der größten Verzögerung wird als längster Pfad oder kritischer Pfad bezeichnet
- Zeitverhalten synchroner Schaltungen
  - In einer synchronen Schaltung bestimmt das langsamste Schaltnetz die Taktperiode bzw. Taktfrequenz

Was bisher geschah... II

Zeitverhalten sync. Schaltungen

Synchrone Schaltung:



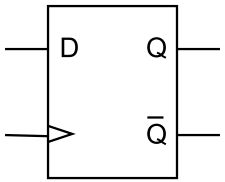
$$T_{\text{min}} = t_{\text{PD,FF}} + t_{\text{PD,max}} + t_{\text{SU}}$$

$\Rightarrow$ Die min. Taktperiode ergibt sich aus der Summe aus Setup- und Verzögerungszeit des Flipflops (t_{SU} und $t_{\text{PD,FF}}$) sowie der Verzögerungszeit des Schaltnetzes ($t_{\text{PD,max}}$)

$$f_{\text{max}} = \frac{1}{T_{\text{min}}}$$

D-Flipflops

Bisher haben wir ausschließlich 1-Bit Speicher wie z.B. das D-Flipflop betrachtet

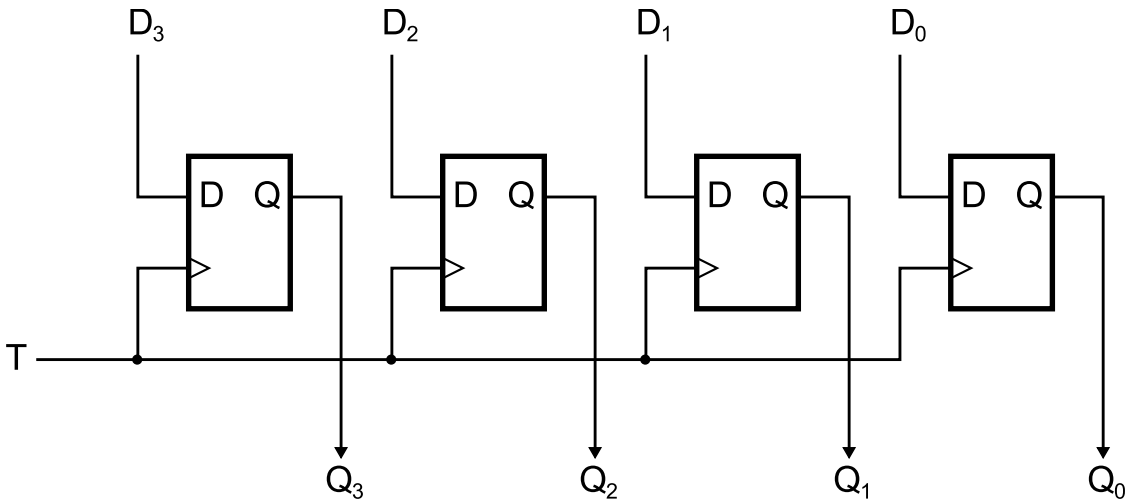


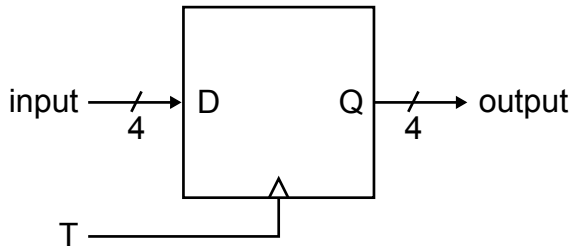
Wie können ganze Worte gespeichert werden?

Register

Angewandte Informatik

Mehrere D-Flipflops bilden ein Register:





Parallele Anordnung von D-Flipflops, gemeinsamer Arbeitstakt

Speichert eine neue Eingabe mit **jeder** steigenden Taktflanke

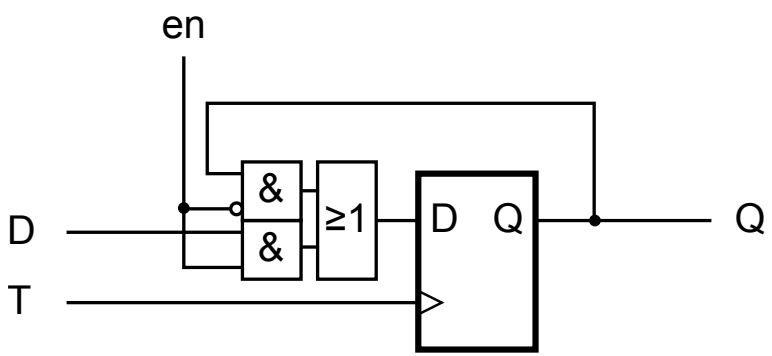
Register mit Enable

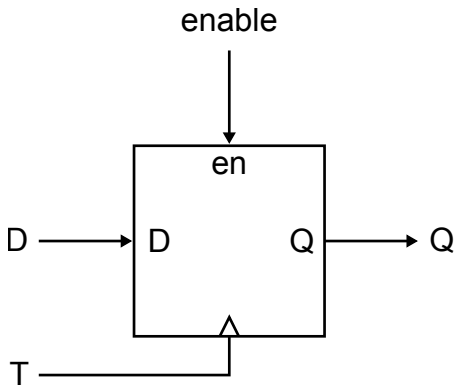
Angewandte Informatik

Oft ist es sinnvoll, ein Datum **über mehrere Taktzyklen** zu halten, bevor es durch einen neuen Wert ersetzt wird.

Dazu wird das Schreiben durch ein Kontrollsignal (**Load** oder **Enable**) gesteuert → paralleles Laden oder Wert behalten.

Der Aufbau erfolgt aus D-Flipflops mit Enable:





$$\text{en} = \begin{cases} 0 & \text{aktuelle Ausgabe } Q \text{ bleibt gespeichert} \\ 1 & \text{Eingang } D \text{ wird gespeichert} \end{cases}$$

Adressierbarer Speicher

Problem: Parallele Speicherung in Registern führt schnell zu einem I/O-Problem, da zu viele Leitungen benötigt werden

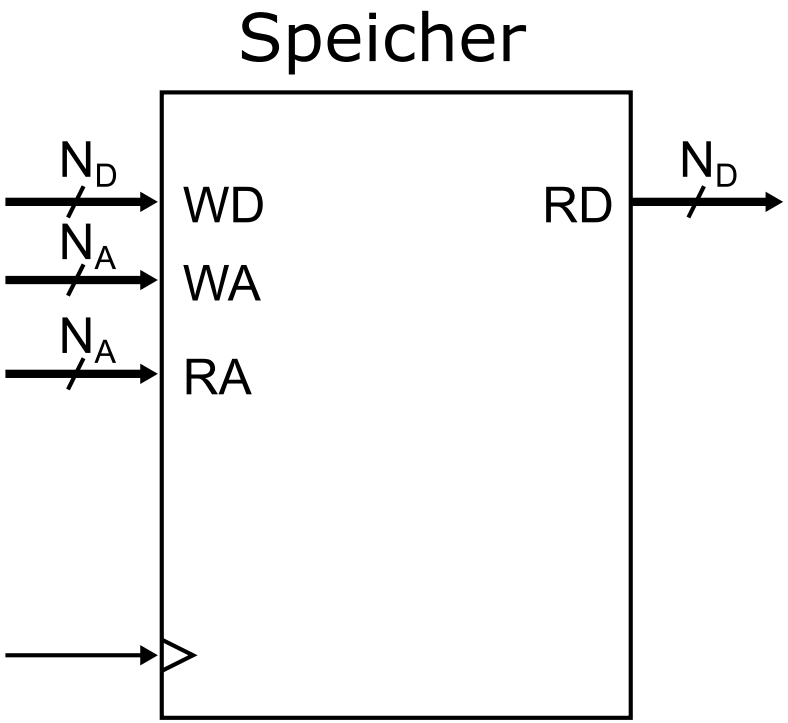
⇒ Lösung: Aufteilung in Daten- und Adressbits

Die Datenbits repräsentieren nur einen kleinen Teil des Speichers der zu einem Zeitpunkt gelesen oder geschrieben werden kann

Die Adressbits repräsentieren den Ort an dem die Datenbits geschrieben werden.

Speicher dieser Art lässt sich beliebig ansprechen, er wird daher auch Random Access Memory (RAM) bezeichnet.

Funktionsweise Speicher



Addresse (N_A Bits)	Daten (N_D Bits)
000...000	Speicherzelle 1
000...001	Speicherzelle 2
000...010	Speicherzelle 3
⋮	
111...111	Speicherzelle 2^{N_A}

WD und **RD**: Write- und Read-Data (Schreib-/Lese-Daten)

WA und **RA**: Write- und Read-Address (Schreib-/Lese-Adresse)

Mit N_A Adress-Bits lassen sich 2^{N_A} Datenworte adressieren.

Aufbau eines Speichers

Im RAM werden Speicherzellen als zweidimensionales **Array** mit 2^{N_A} Zeilen zu je N_D Speicherzellen angeordnet.

Es entsteht ein $2^{N_A} \times N_D$ Bit Speicher.

Jede Zeile speichert ein Datenwort mit N_D Bit

Werden Flipflops als Speicherstellen verwendet spricht man von einem **Registerfile**.

Aufbau eines Speichers

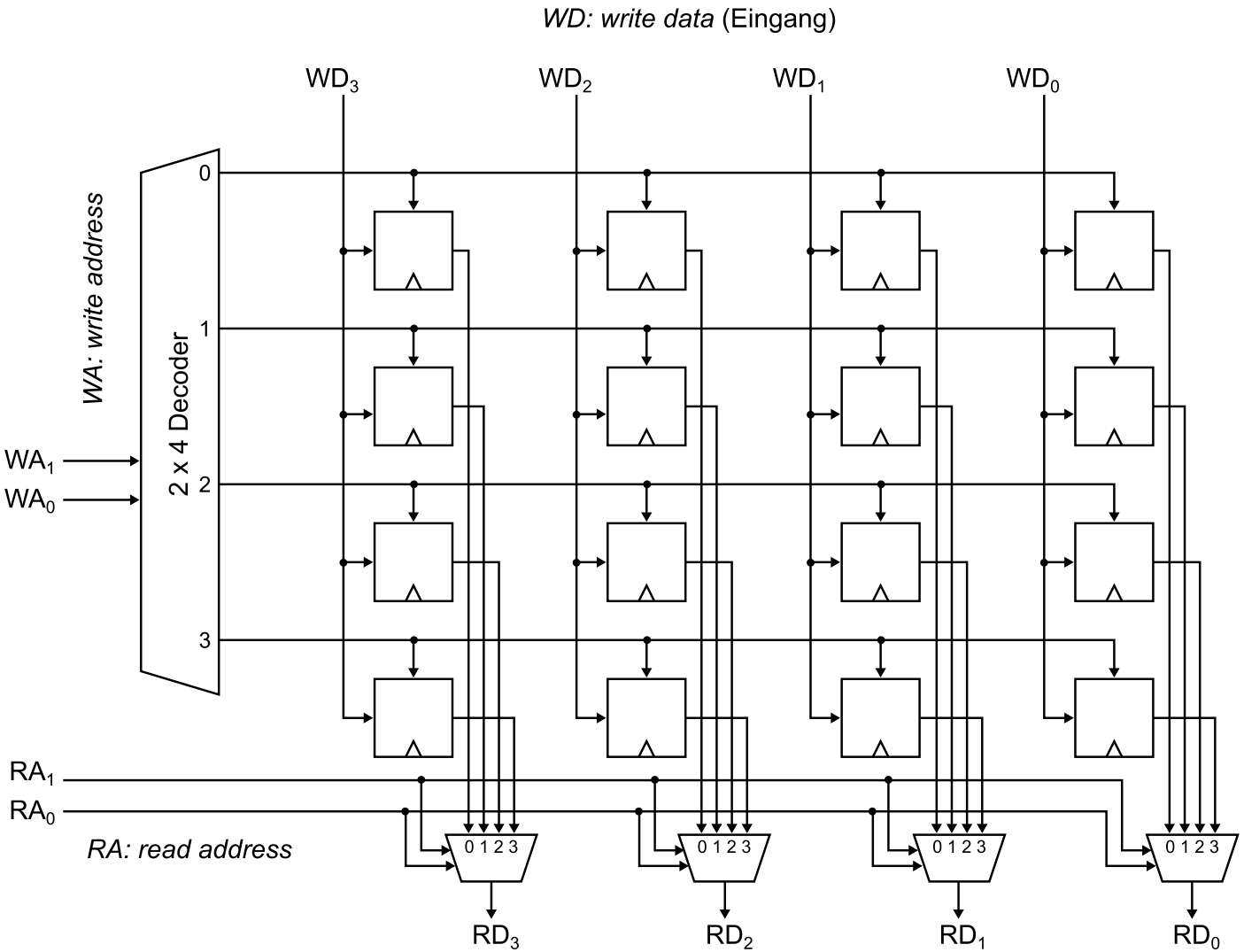
Im RAM werden Speicherzellen als zweidimensionales **Array** mit 2^{N_A} Zeilen zu je N_D Speicherzellen angeordnet.

Jede Zeile speichert ein Datenwort

Es entsteht ein $2^{N_A} \times N_D$ Bit Speicher.

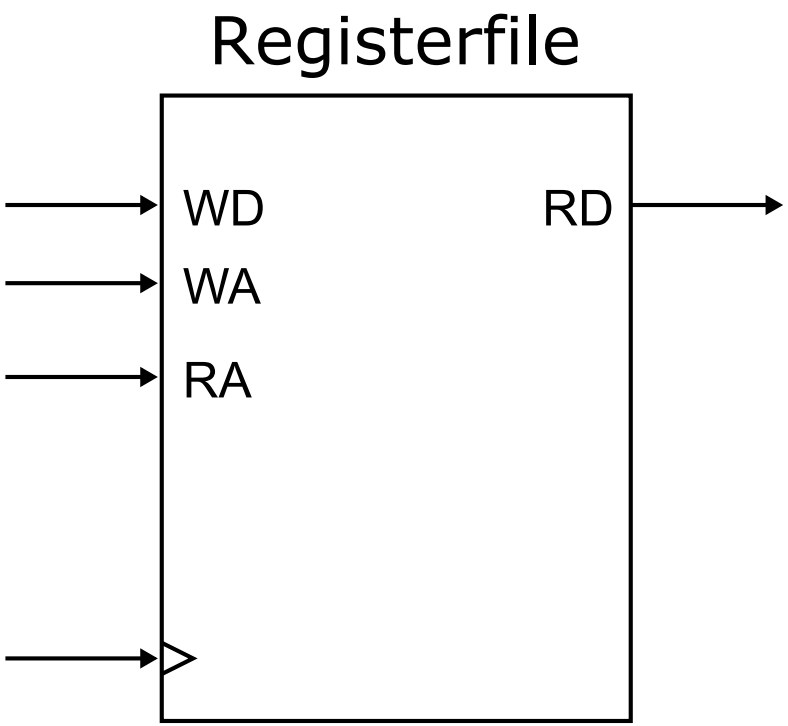
Werden Flipflops als Speicherstellen verwendet spricht man von einem **Registerfile**.

4×4 Bit Registerfile



Alle FFs arbeiten mit dem gleichen Takt (nicht eingezeichnet)

Zugriff Registerfile



Adresse für Lesen und Schreiben sind unabhängig

D.h. in jedem Takt kann in Register-Adresse *WA* geschrieben und von Register-Adresse *RA* gelesen werden

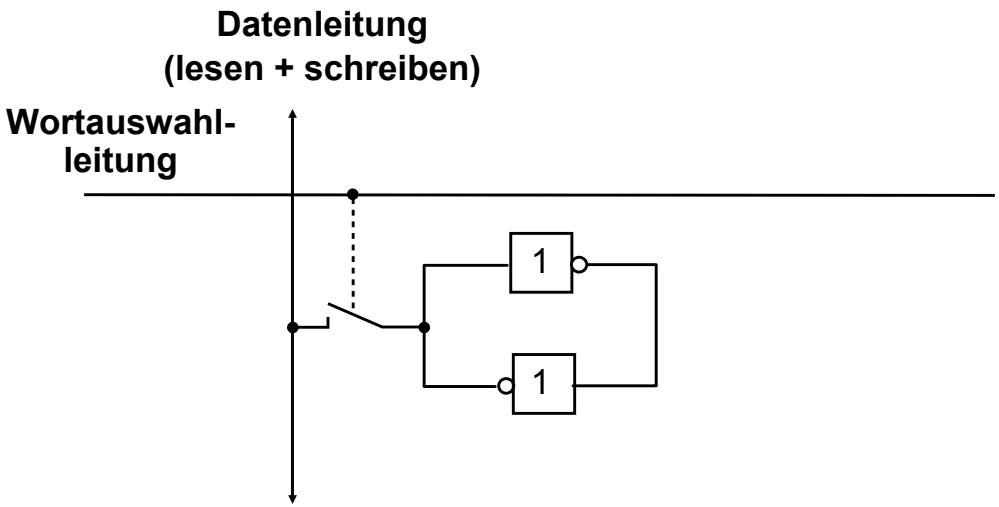
Optimierte Speicherzelle (SRAM-Zelle)

Statt D-Flipflop als Speicherzelle wird eine stark vereinfachte Schaltung verwendet

Zwei Rückgekoppelte Inverter bilden den Speicher

Diese können über Transistoren ausgelesen/gesetzt werden

Synchronisierung mit Takt erfolgt außerhalb



Statischer Speicher

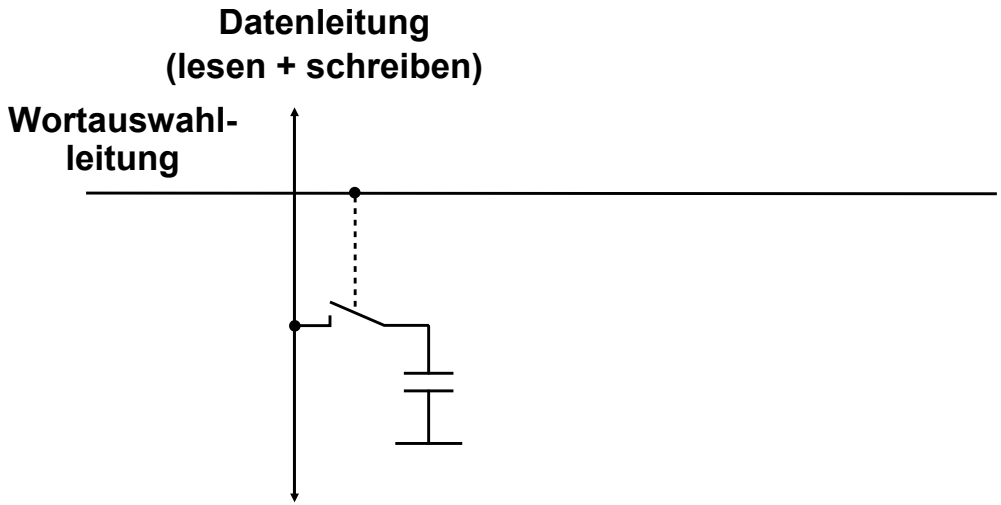
Dieser Speicher wird statischer Speicher (*static RAM*, SRAM) genannt

Bei Registerfile ist i.d.R. schneller als SRAM

Speicherung bleibt bei beiden beliebig lange solange Betriebsspannung erhalten bleibt

Dynamischer Speicher

Bei dynamischen Speicher (*dynamic RAM*, DRAM) wird jedes Bit in einem Kondensator gespeichert:



Dadurch sehr kompakt

Durch Leckströme muss dieser durch regelmäßiges Lesen und Schreiben aufgefrischt werden ⇒ Eigener Controller

Überblick (Haupt-)Speicher

Allen Speicherkonzepten gemein ist die Anordnung der Speicherzellen als Array

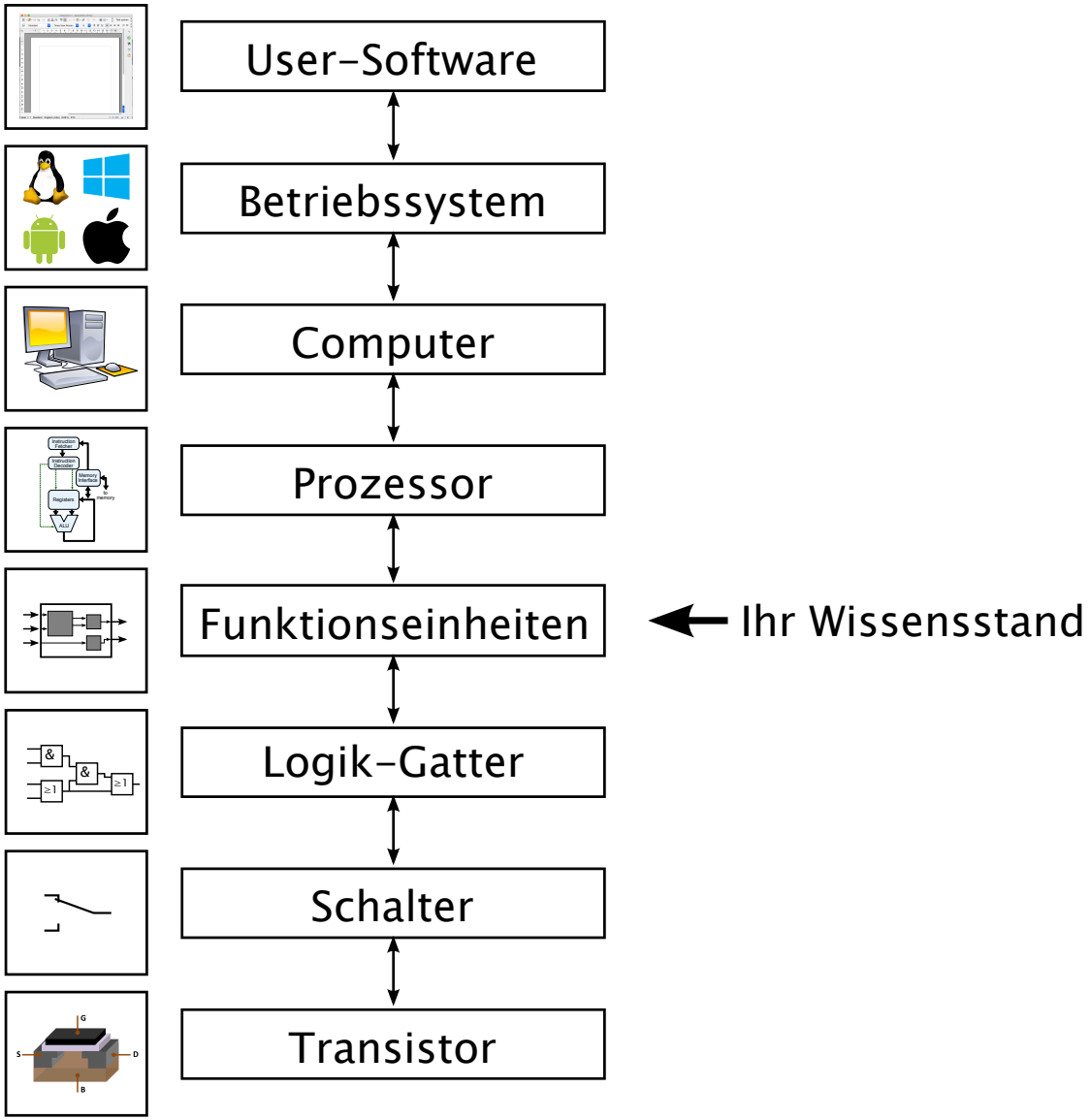
Unterschiede bestehen in der Art der Speicherzelle:

Speicherzelle	Bezeichnung	Eigenschaften
Register	Registerfile	Sehr schnell, sehr aufwendig, $56..83\,\mu m^2$ (i)
Rückgek. Inv.	SRAM	Schnell, weniger aufwendig, 4-6 Transistoren, $10..11\,\mu m^2$ (ii)
Kondensator	DRAM	Langsam, wenig aufwendig, 1 Transistor, $0,7..1\,\mu m^2$ (ii)

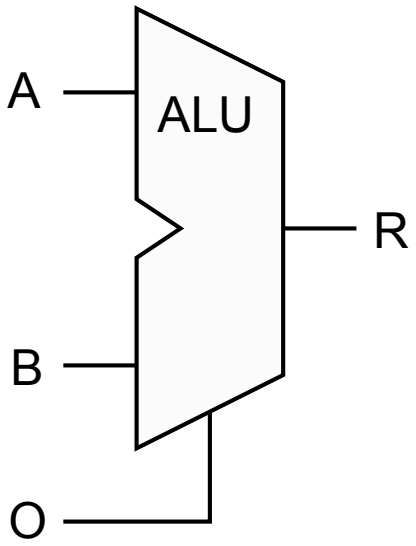
(i) Technologie: TSMC 180nm

(ii) Technologie: Infineon 200 nm,
aktuelle SRAM Zelle bei 3 nm: $0{,}02\,\mu m^2$ (`https://heise.de/-7396426`)

Die Macht der Abstraktion



Wiederholung: ALU – Der Rechenkern

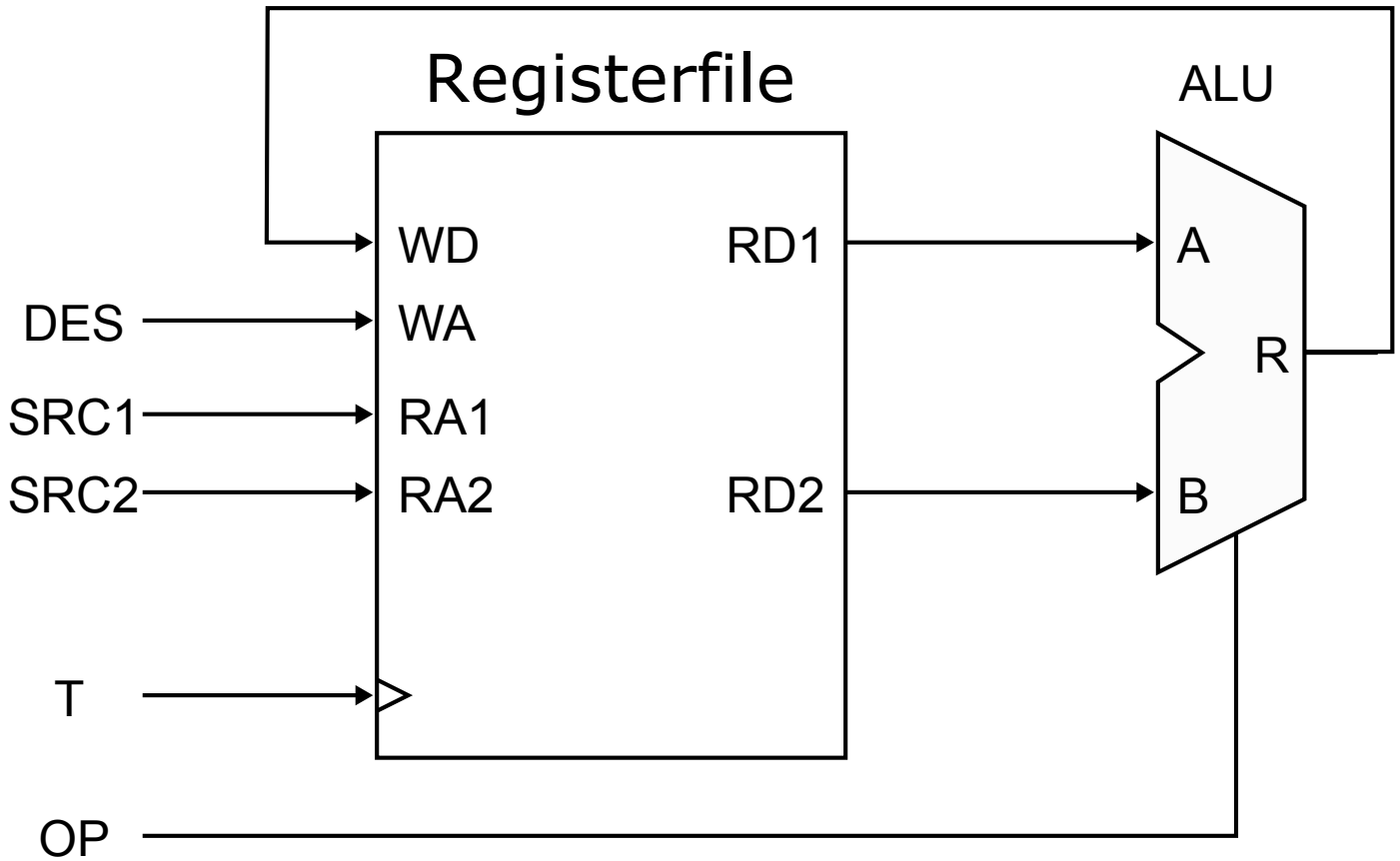


o_2	o_1	o_0	Operation	R
0	0	0	+	$A + B$
0	0	1	−	$A - B$
0	1	0	×	$A \times B$
0	1	1	UND	$A \wedge B$
1	0	0	ODER	$A \vee B$
1	0	1	XOR	$A \oplus B$
1	1	–	keine	–

🤔 Ok, das Teil kann Rechnen aber wohin mit den Ergebnissen?

⇒ Speicher

Registerfile mit ALU



DES: Zielregister (Destination) des Ergebnisses

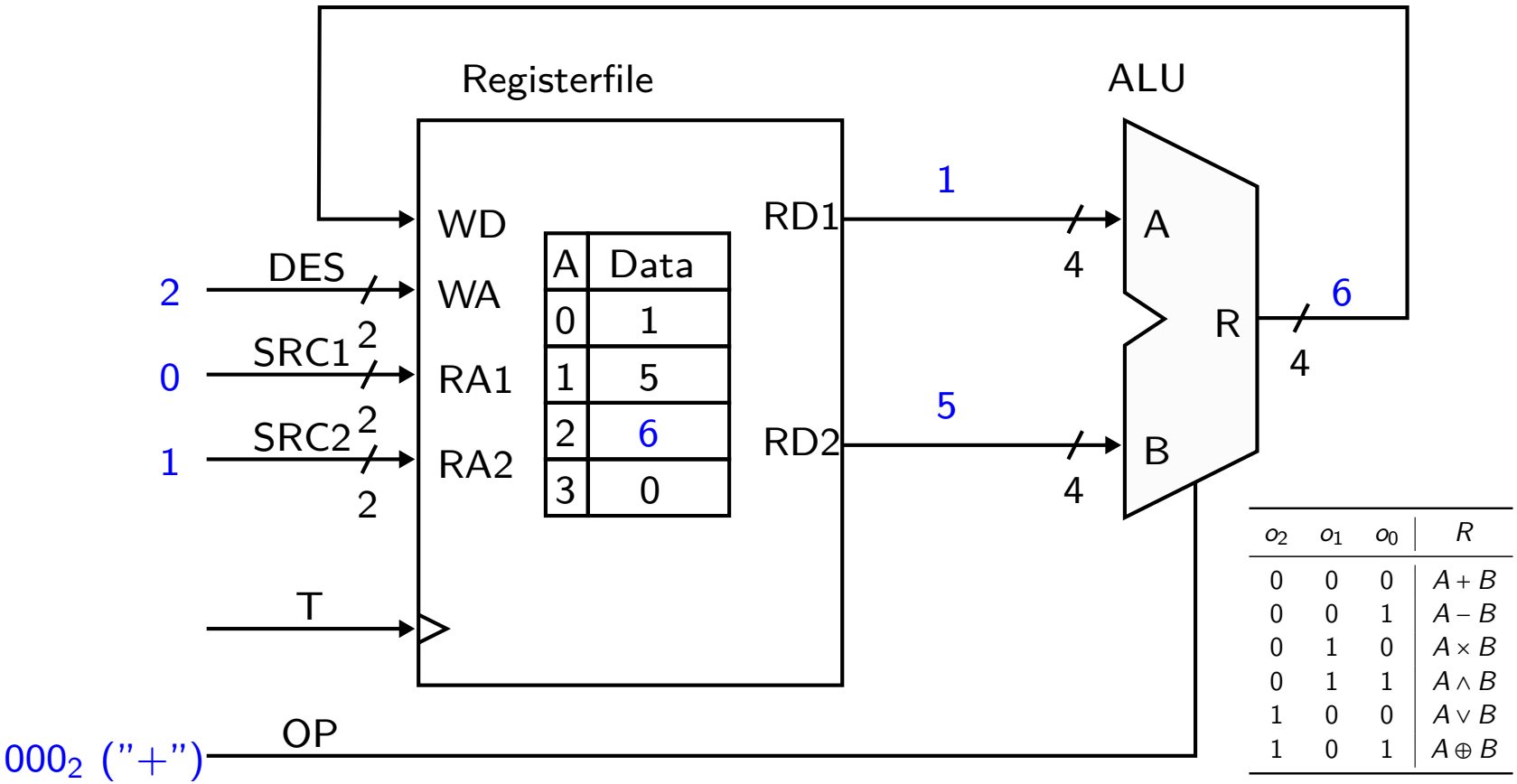
SRC1/2: Quellregister (Source) der beiden Argumente

OP: Operation (+, −, ×, etc.)

Beispiel 1

Angewandte Informatik

Beispiel mit 4×4 Bit Registerfile und 4 Bit ALU:

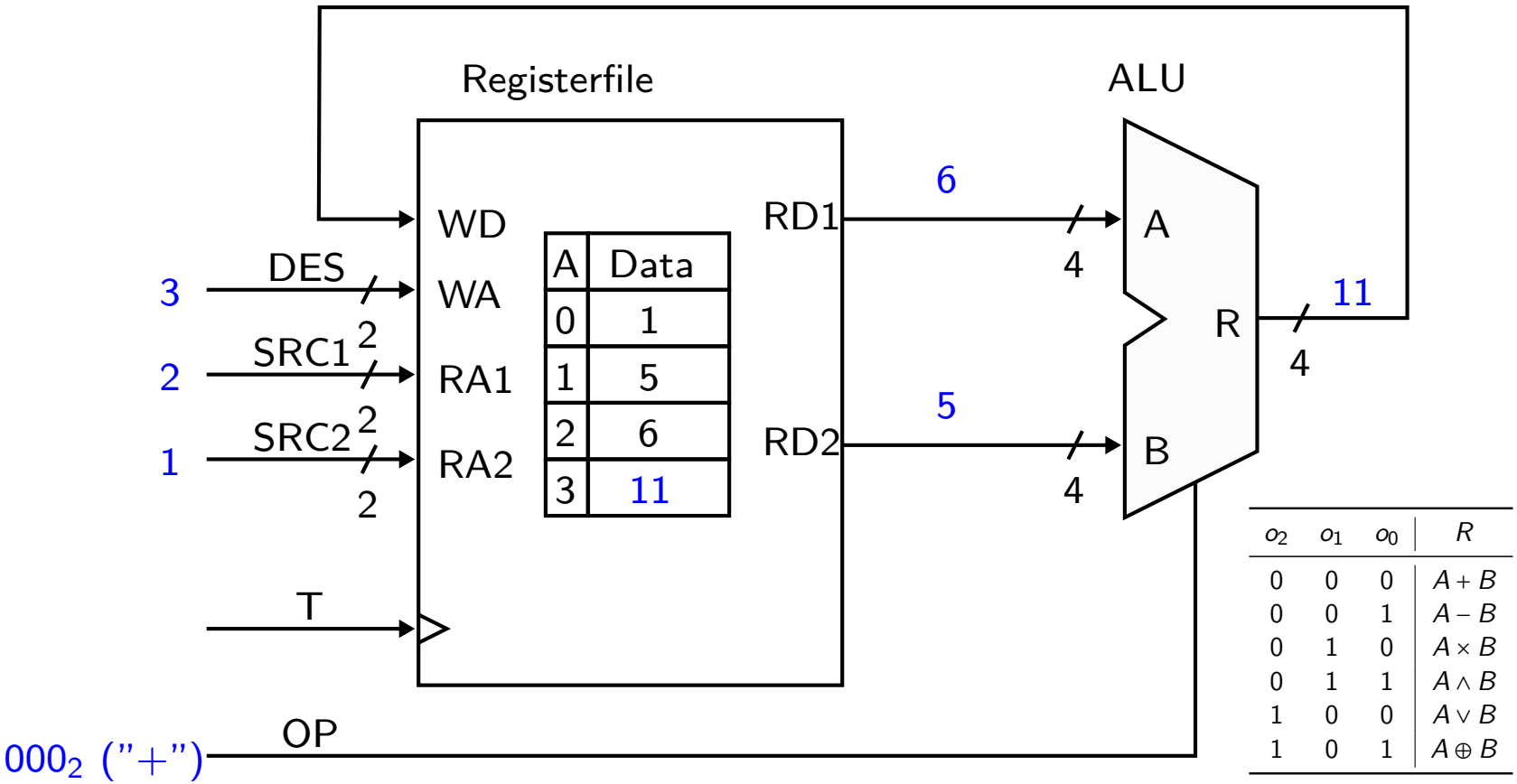


o_2	o_1	o_0	R
0	0	0	$A+B$
0	0	1	$A-B$
0	1	0	$A \times B$
0	1	1	$A \wedge B$
1	0	0	$A \vee B$
1	0	1	$A \oplus B$

Beispiel 2

Angewandte Informatik

Beispiel mit 4×4 Bit Registerfile und 4 Bit ALU:

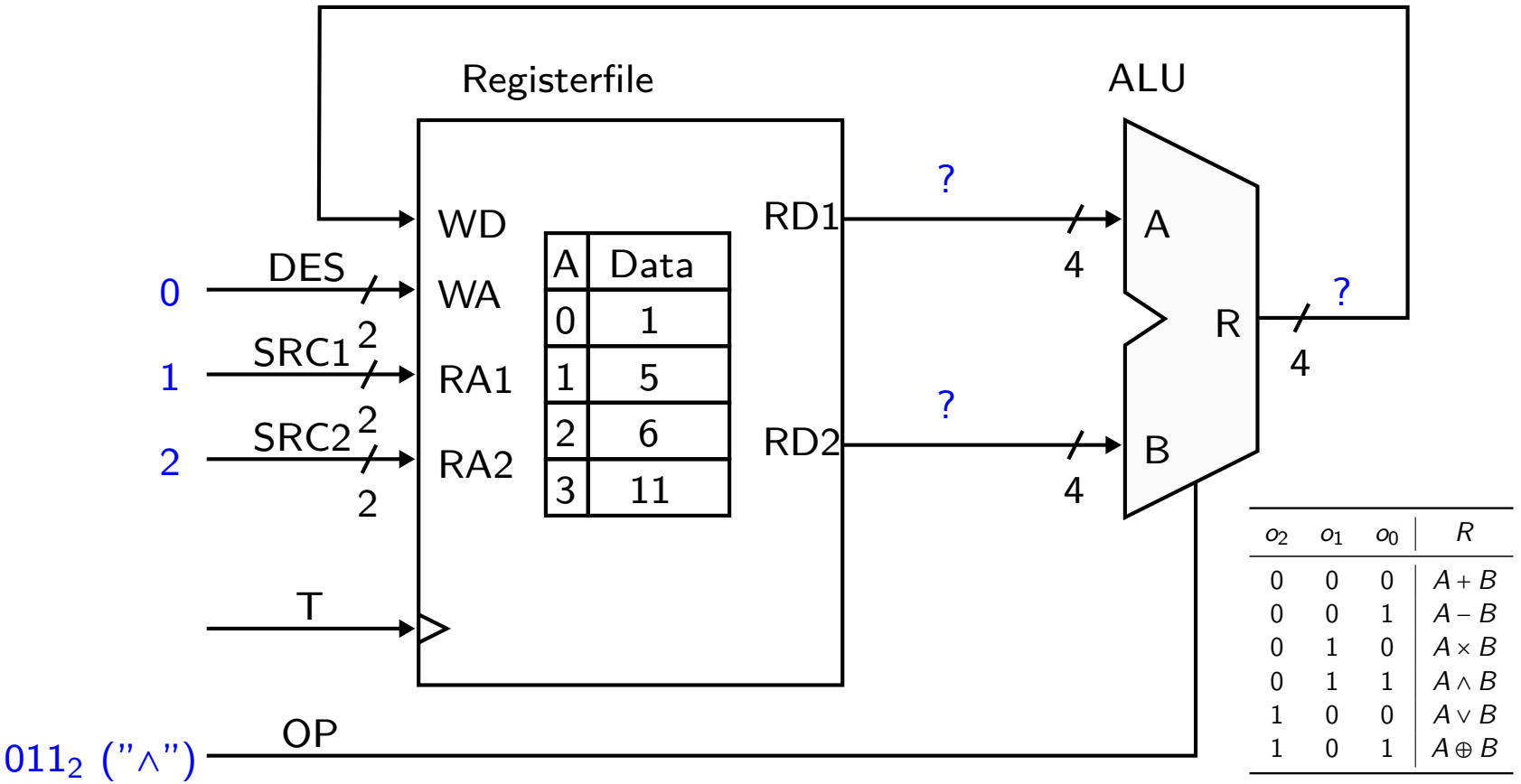


o_2	o_1	o_0	R
0	0	0	$A+B$
0	0	1	$A-B$
0	1	0	$A \times B$
0	1	1	$A \wedge B$
1	0	0	$A \vee B$
1	0	1	$A \oplus B$

Vorlesungsaufgabe

Angewandte Informatik

Wie lautet das Ergebnis R bei folgender Eingabe?
Welches Register wird hierbei überschrieben?

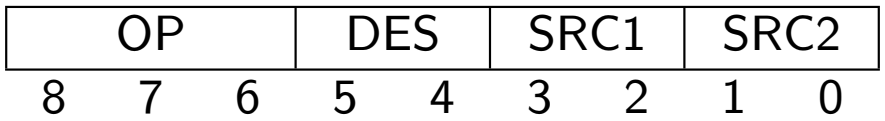


o_2	o_1	o_0	R
0	0	0	$A+B$
0	0	1	$A-B$
0	1	0	$A \times B$
0	1	1	$A \wedge B$
1	0	0	$A \vee B$
1	0	1	$A \oplus B$

Maschinencode (OPCode)

Die Eingabe bestehend aus Befehl (OP), Quellregistern (SRC1, SRC2) und Zielregister (DES) bilden einen Prozessorbefehl

Dieser muss entsprechend codiert werden, z.B.:



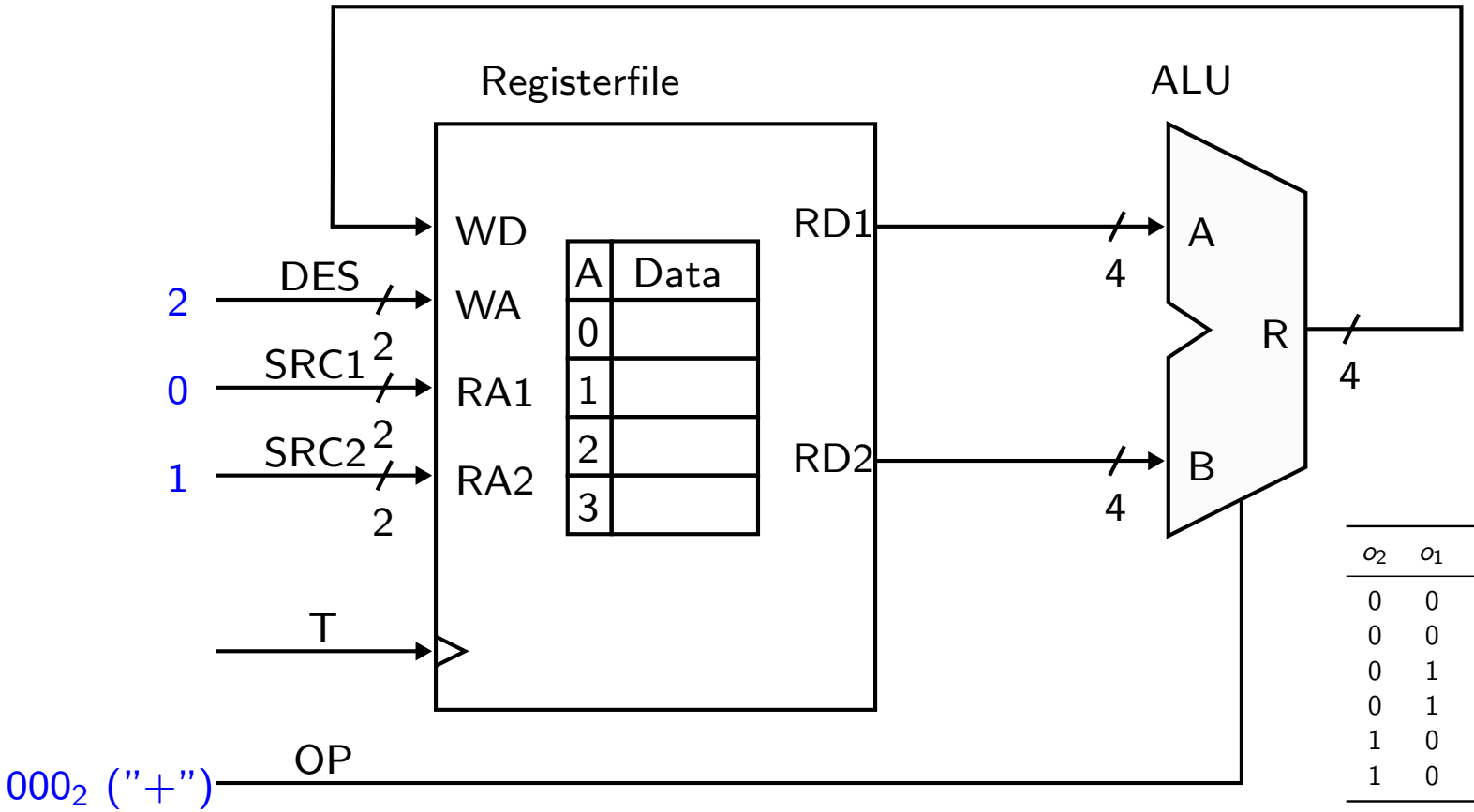
Zum Beispiel wird die Addition »Addiere Register 0 mit Register 1 und speichere in Register 2« abgekürzt zu:

```
add r2,r0,r1
```

Der *Maschinencode* dieses Beispiels lautet:

$$\underbrace{\texttt{add}}_{=000_2}\ \underbrace{\texttt{r2}}_{=10_2},\underbrace{\texttt{r0}}_{=00_2},\underbrace{\texttt{r1}}_{=01_2} \Rightarrow 000100001_2$$

Befehl aus Beispiel 1

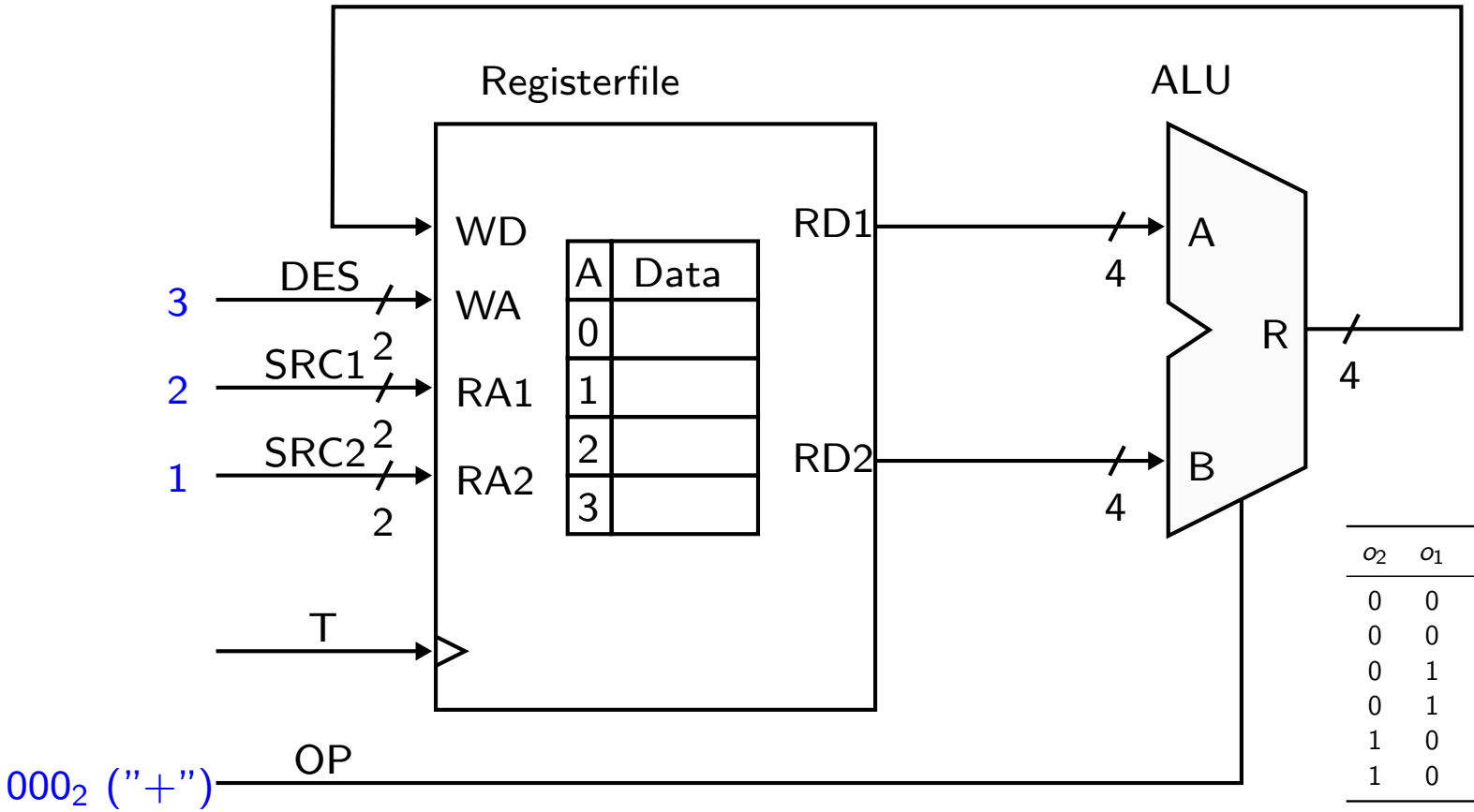


o_2	o_1	o_0	Befehl
0	0	0	add
0	0	1	sub
0	1	0	mul
0	1	1	and
1	0	0	or
1	0	1	xor

Der zugehörige Befehl lautet:

$$\underbrace{\texttt{add}}_{=000_2}\ \underbrace{\texttt{r2}}_{=10_2}, \underbrace{\texttt{r0}}_{=00_2}, \underbrace{\texttt{r1}}_{=01_2} \Rightarrow 000100001_2$$

Befehl aus Beispiel 2

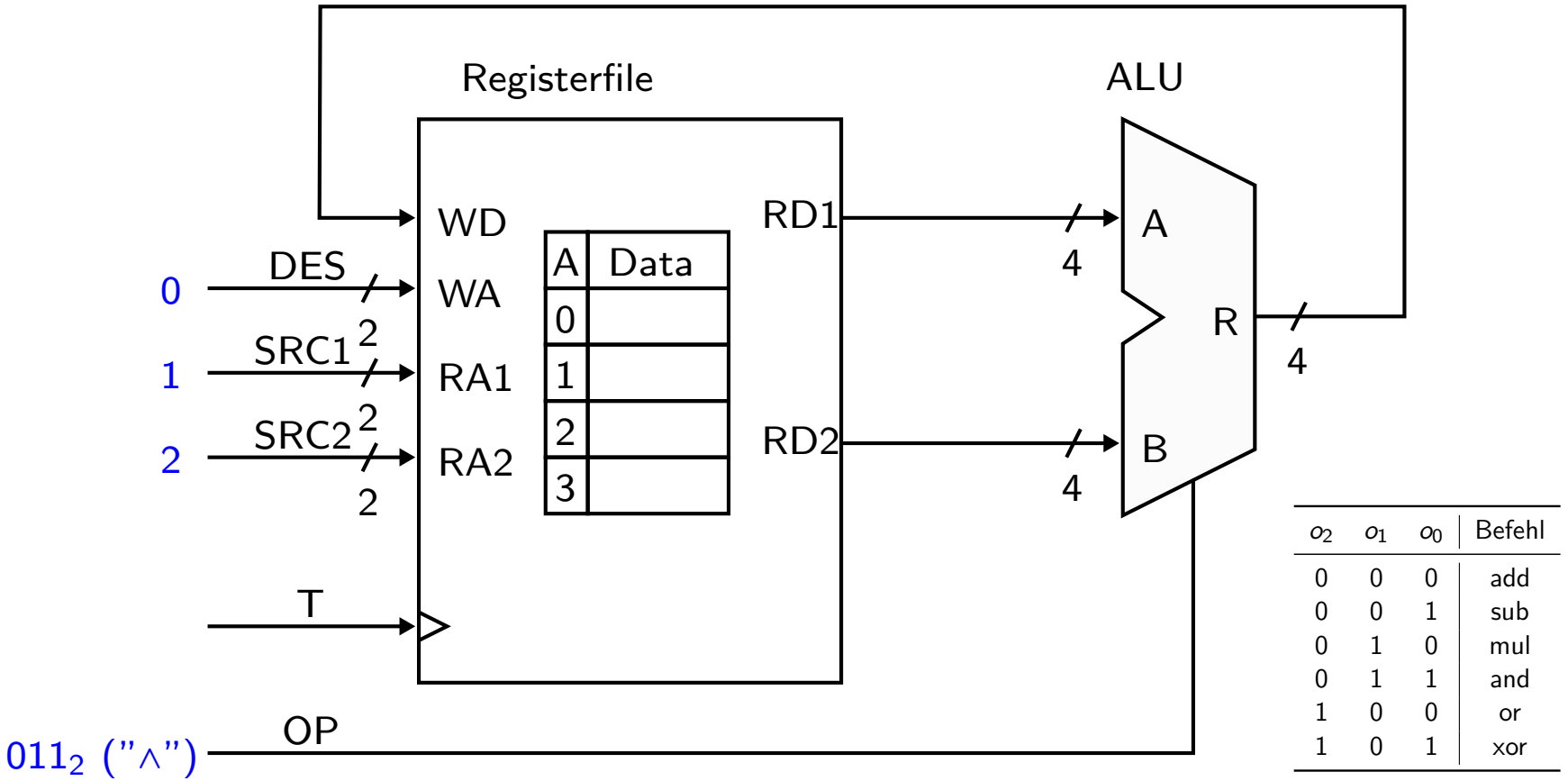


o_2	o_1	o_0	Befehl
0	0	0	add
0	0	1	sub
0	1	0	mul
0	1	1	and
1	0	0	or
1	0	1	xor

Der zugehörige Befehl lautet:

$$\underbrace{\texttt{add}}_{=000_2}\ \underbrace{\texttt{r3}}_{=11_2},\underbrace{\texttt{r2}}_{=10_2},\underbrace{\texttt{r1}}_{=01_2} \Rightarrow 000111001_2$$

Befehl aus Vorlesungsaufgabe



o_2	o_1	o_0	Befehl
0	0	0	add
0	0	1	sub
0	1	0	mul
0	1	1	and
1	0	0	or
1	0	1	xor

Der zugehörige Befehl lautet:

Registerfile mit ALU

Die 3 Beispiel-Befehle ergeben ein erstes Programm:

```
add r2,r0,r1
add r3,r2,r1
and r0,r1,r2
```

🤔 Ok, funktioniert, wenn ich jeden Takt einen neuen Befehl anlege.

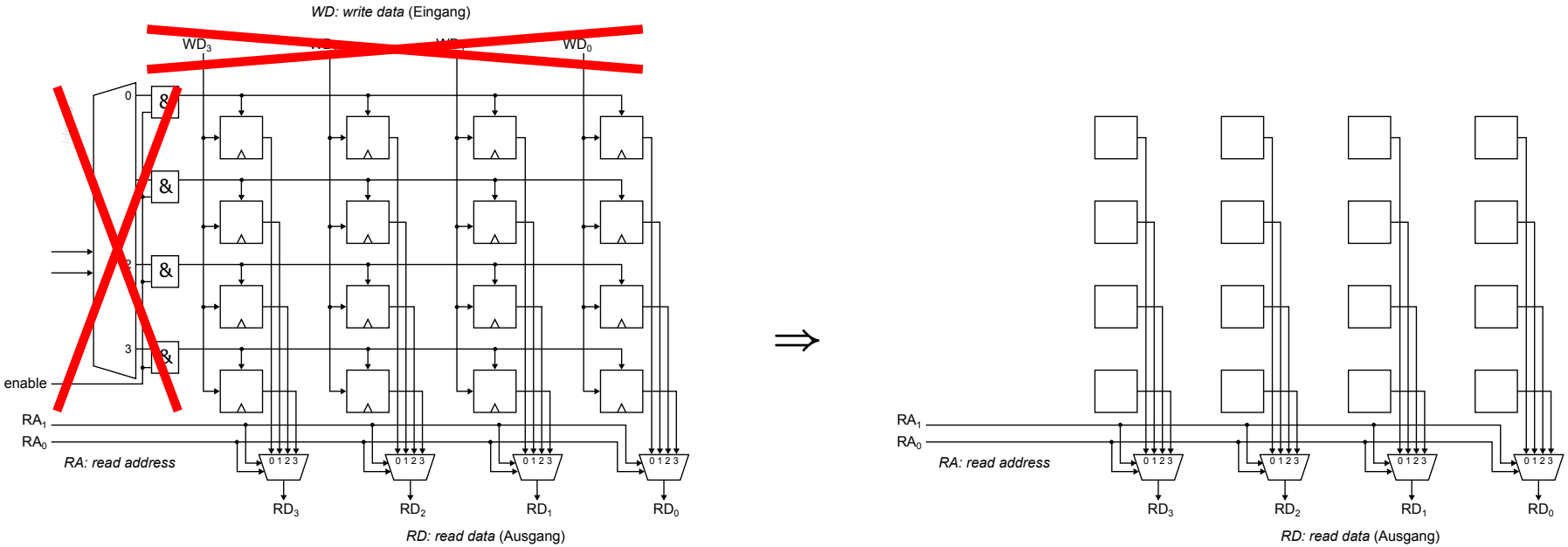
🤔 Aber lässt sich das nicht automatisieren?

⇒ Speicher für Programm ⇒ Programmspeicher

Programmspeicher

Angewandte Informatik

Programmspeicher ist im einfachsten Fall ein Nur-Lesespeicher (*read-only memory*, ROM):



Die »Speicherzellen« sind in diesem Fall Konstanten (0 oder 1).

Programmspeicher

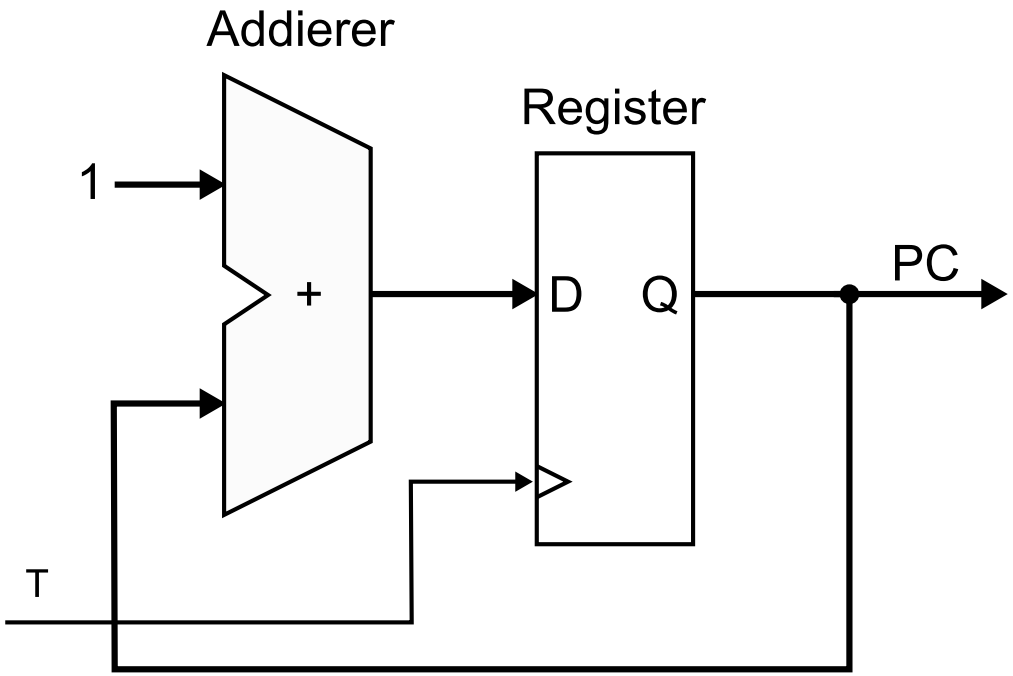
Angewandte Informatik

🤔 Wie lässt sich dieses Programm abrufen?

⇒ Zum Zeitpunkt 0 muss Adresse 0 abgefragt werden, zum Zeitpunkt 1 Adresse 1, zum Zeitpunkt 2 Adresse 2, ...

⇒ Wir brauchen einen Zähler!

Programmzähler



Moore-Automat mir Addierer als Zustandsübergangsfunktion!

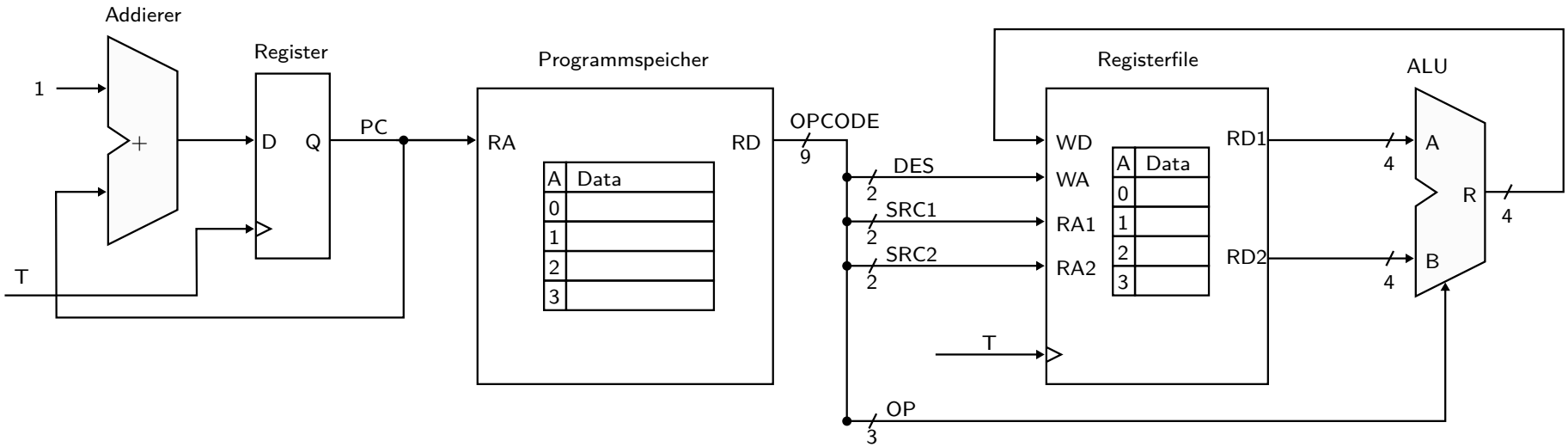
Der Zähler erhöht den Programmzähler (*program counter*, PC) mit jedem Takt um 1.

Minimal-Prozessor

Angewandte Informatik

Programmzähler wird verwendet um den Programmspeicher zu adressieren

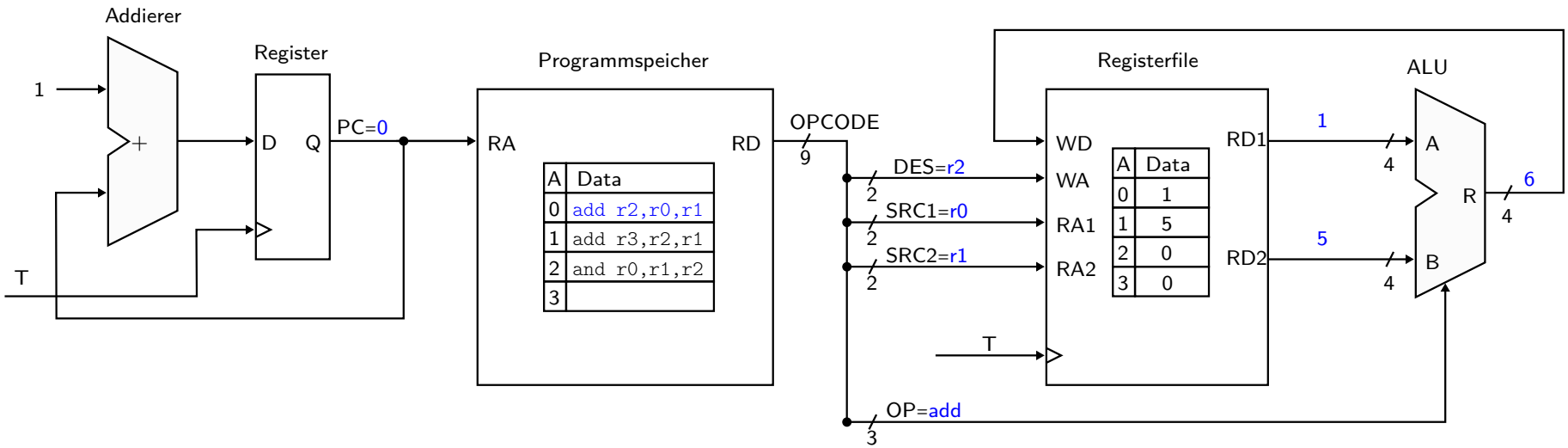
Ausgabe vom Programmspeicher wird als `OPCODE` interpretiert und adressiert Registerfile und bestimmt ALU-Operation



Minimal-Prozessor

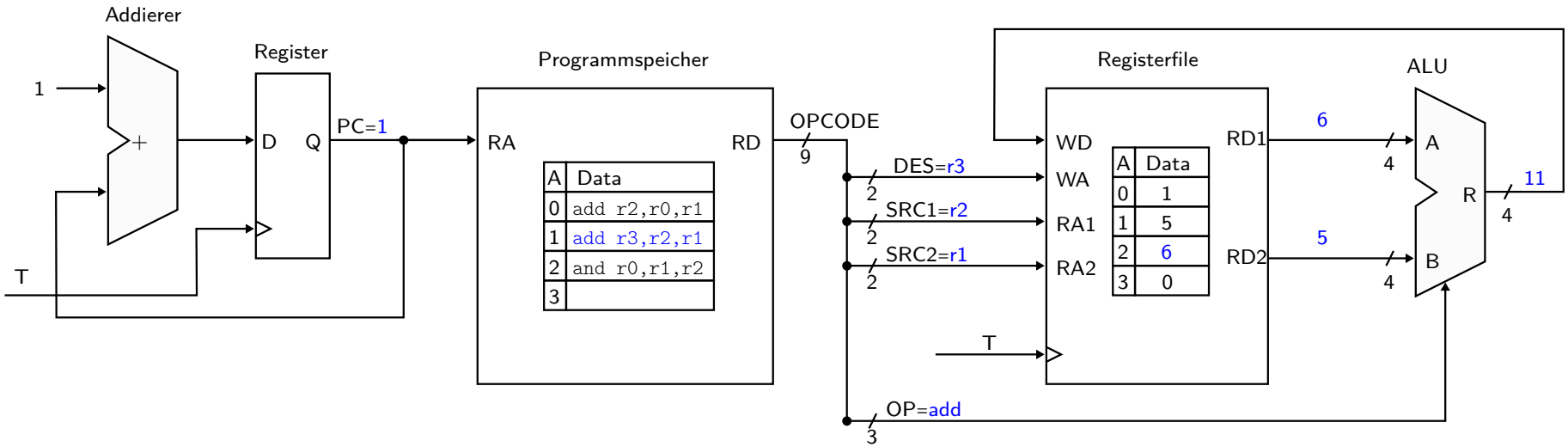
Angewandte Informatik

Simulation: Takt 0 (vor der 1. Taktflanke)



Minimal-Prozessor

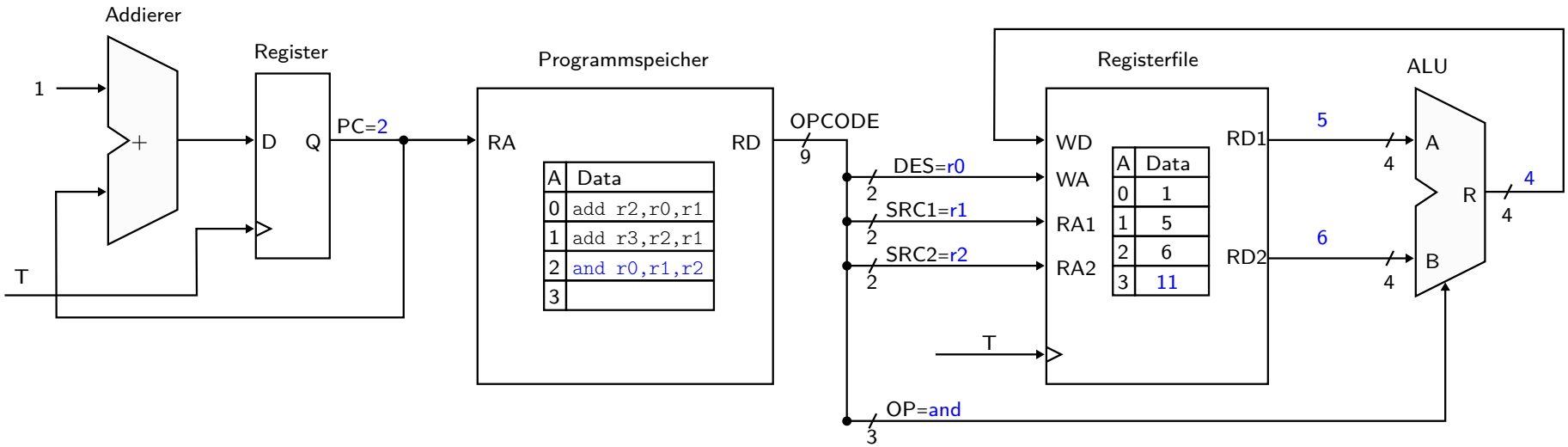
Simulation: Takt 1



Minimal-Prozessor

Angewandte Informatik

Simulation: Takt 2



Minimal-Prozessor

Was unser Minimal-Prozessor bisher kann:

- Ein lineares Programm »abspulen«
- Register können arithmetisch und logisch kombiniert werden

Was unser Minimal-Prozessor bisher nicht kann:

- Register können noch nicht initialisiert werden
- Sprünge oder Schleifen im Programmablauf
- Speicher limitiert auf Größe des Registerfiles

⇒ Wir müssen unseren Rechner um neue Befehle erweitern!