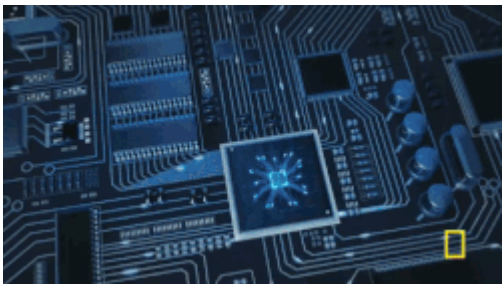


# Von mathematischen Operationen zur CPU

Parameter	Kursinformationen
Veranstaltung:	<a href="#">Digitale Systeme / Eingebettete Systeme</a>
Semester	<a href="#">Wintersemester 2023/24</a>
Hochschule:	<a href="#">Technische Universität Freiberg</a>
Inhalte:	<a href="#">Realisierung grundlegender arithmetischer Operationen im Rechner</a>
Link auf GitHub:	<a href="https://github.com/TUBAF-lfl-LiaScript/VL_EingebetteteSysteme/blob/master/10_CPU_Basis.md">https://github.com/TUBAF-lfl-LiaScript/VL_EingebetteteSysteme/blob/master/10_CPU_Basis.md</a>
Autoren	Sebastian Zug & André Dietrich & Fabian Bär

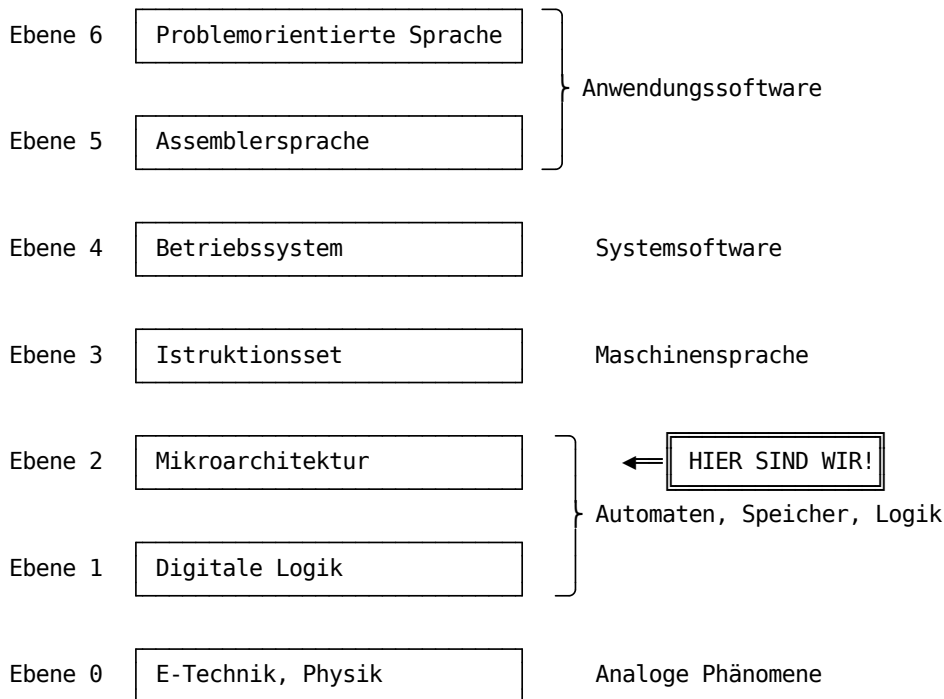


---

## Fragen an die Veranstaltung

- Welche Funktionalität sollte eine ALU bereitstellen?
  - Wie werden Programmierbefehle umgesetzt und im Speicher abgebildet?
  - Welche Aufgaben hat das Steuerwerk?
-

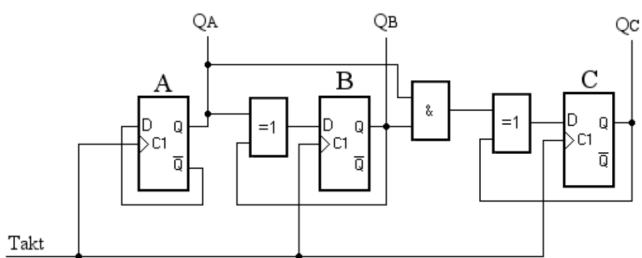
## Abstraktionsebenen



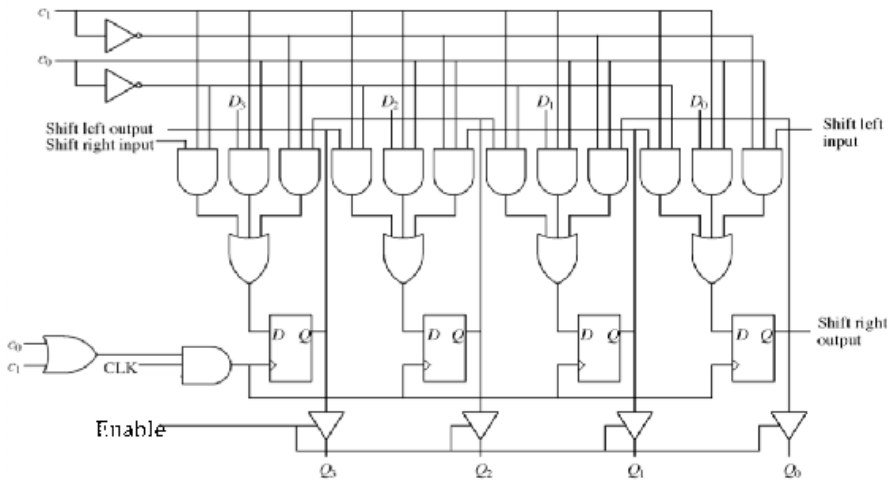
## Ausgangspunkt

**Aufgabe:** Welche Funktionalität verbirgt sich hinter folgenden Schaltwerken/ Schaltnetze?

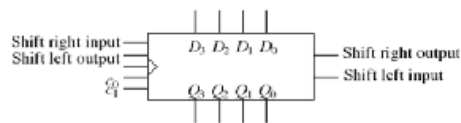
Synchroner 3-Bit-Zähler auf der Basis von JK Flip-Flops



4-Bit PIPO Schieberegister (Links-Rechts)

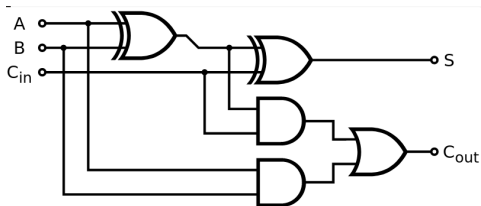


Control	Function
$c_1$ $c_0$	
0 0	No change
0 1	Shift left
1 0	Shift right
1 1	Parallel load



[Alnoaman]

## 1 Bit Volladdierer



[Alnoaman] Ali Alnoaman, Foreintrag Electrical Engineering, [Link](#)

## Schritt 1 - Operationsauswahl

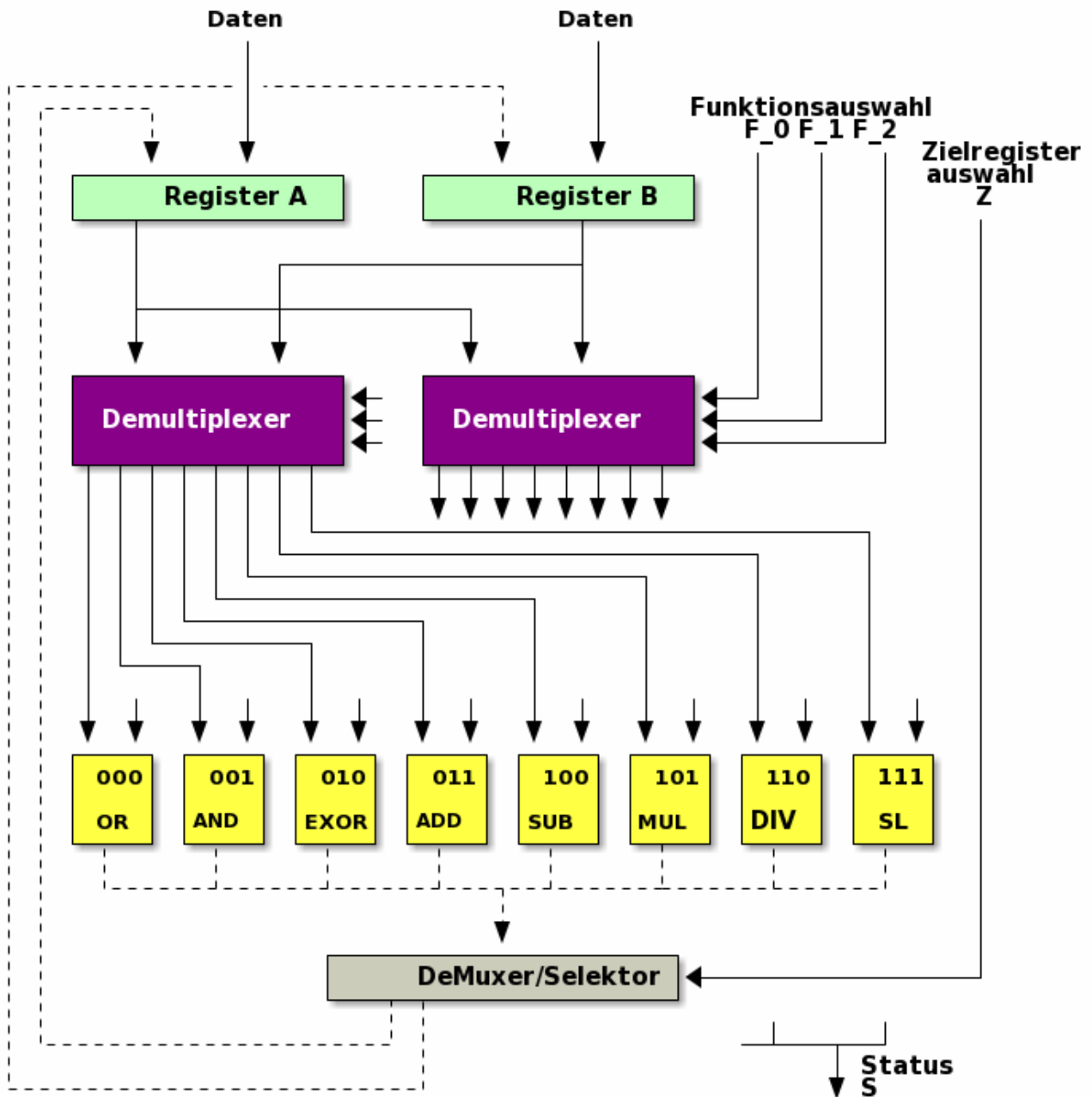
Zwischenstand

Stufe	Funktionalität	Wunschzettel
0	Addition/Subtraktion von einzelnen Werten ✓	
1		Flexibles Handling mehrerer Operationen ?

Logische Funktionen: NOT, AND, NOR, XOR

Arithmetische Funktionen: ADD, SUB, (MUL), (DIV)

Sonstige: SHIFT LEFT (arithmetisch, logisch), SHIFT RIGHT (arithmetisch, logisch)



Der Status S umfasst eine Zusammenstellung der Resultate der Operationen codiert als 1-Bit Werte:

- **Carry** - Flag für die Addition / Multiplikation
- **Overflow** - Flag
- **Error** - Flag für die Division durch Null
- ...

Wie "programmieren" wir unser System?

F_0	F_1	F_2	Z	Bezeichnung
0	0	0	0	OR_A
0	0	0	1	OR_B
...				
0	1	1	0	ADD_A
0	1	1	1	ADD_B
...				

### Beispielanwendungen

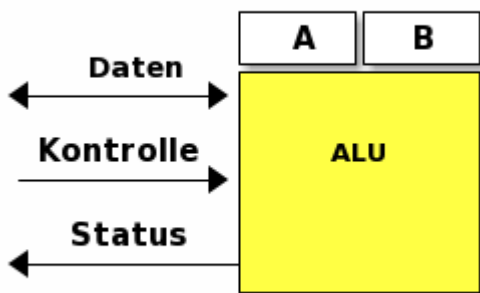
	Inkrementieren	Togglern
Vorbereitung	$A \leftarrow 1, B \leftarrow 0$	$A \leftarrow 1$
"Programm"	ADD_B (0111)	EOR_A (0101)
	ADD_B (0111)	EOR_A (0101)
	ADD_B (0111)	EOR_A (0101)

## Schritt 2 - Sequenz von Berechnungen

Zwischenstand

Stufe	Funktionalität	Wunschzettel
0	Addition/Subtraktion von einzelnen Werten ✓	
1	Arithmetische Einheit mit mehreren Funktionen und wählbarem Ergebnisregister ✓	
2		Sequenz von Berechnungsfolgen ?  $Reg\_A \leftarrow 3$ $Reg\_B \leftarrow 2$ $ADD\_B$ $Reg\_A \leftarrow -3$ $MUL\_B$ ...

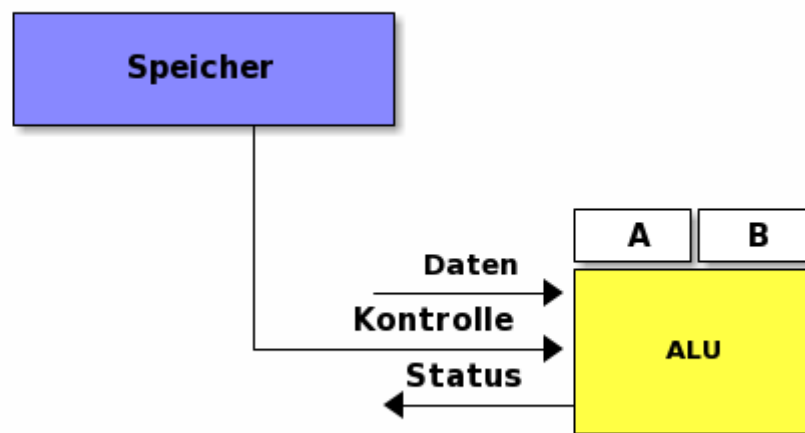
Für diesen Schritt fassen wir das obige Schaltbild unserer hypothetischen ALU mit 8 Funktionen in einem abstrakteren Schaubild zusammen.



Bezeichnung	Bedeutung
A, B	Datenregister
Daten	Zugriff auf die Datenregister
Kontrolle	Steuerleitungen $F_0$ bis $F_2$ und $Z$
Status	Carry oder Fehlerflags

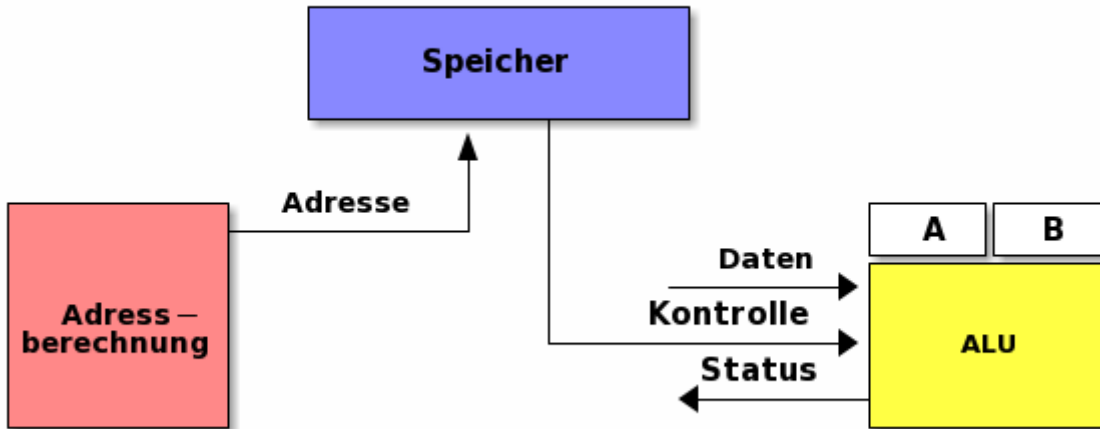
Wir erweitern diese ALU-Komponenten nun um zwei weitere Module - einen Speicher und ein Steuerwerk. Der Speicher umfasst unsere Programmbestandteile `AND_B` usw. in jeweils einem 4 Bit breiten Register.

Analog zur Diskussion um die abstraktere Darstellung der ALU fassen wir auch den Speicher in einem Block-Symbol zusammen.

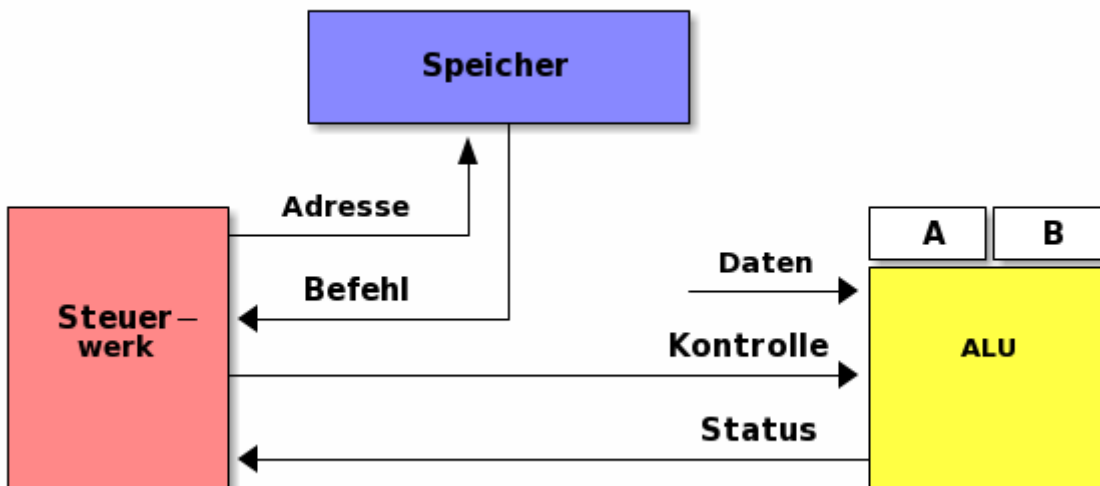


Wie allerdings setzen wir den Fortschritt im Programm um? Nach welcher Methodik werden die nachfolgenden Befehle aufgerufen?

Eine weitere Komponente, das Steuerwerk übernimmt diese Aufgabe. Anstatt nun eine Folge von Kontrollflags vorzugeben, erzeugen wir intern eine Folge von Adressen, die auf Speicherbereiche verweisen, in denen die Konfigurationen der ALU hinterlegt sind.



Allerdings bleibt bei dieser Konfiguration unser Status auf der Strecke! Im Grund müssen wir die Information dazu aber Operationsspezifisch auswerten. Es genügt also nicht allein eine Adressberechnung zu realisieren, vielmehr bedarf es einer generellen Steuerungskomponente, die die Ausführung von Befehlen initiiert und überwacht.



Das Steuerwerk ist nun dafür verantwortlich:

- Adressen zu berechnen
- Befehle zu interpretieren
- die ALU über entsprechende Flags zu konfigurieren
- die Statusinformationen entsprechend dem aktuellen Befehl auszuwerten

Wir lösen uns von dem Zugriff auf die Kontrollbits und etablieren abstrakte Befehle.



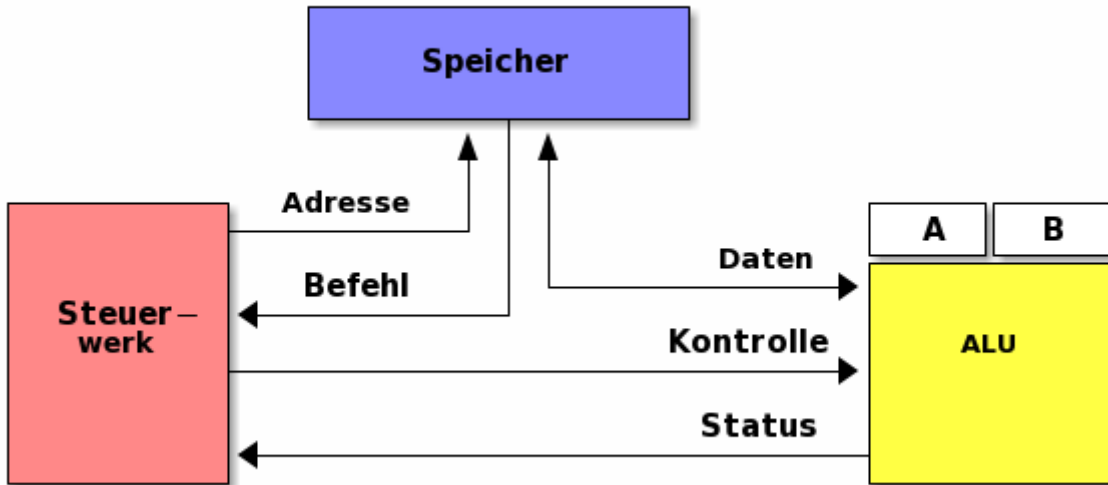
**Merke:** Das Steuerwerk entkoppelt die Konfiguration einzelner Komponenten und die Befehlsdarstellung.

## Schritt 3 - Handhabung der Daten

Zwischenstand

Stufe	Funktionalität	Wunschzettel
0	Addition/Subtraktion von einzelnen Werten ✓	
1	Arithmetische Einheit mit mehreren Funktionen und wählbarem Ergebnisregister ✓	
2	Sequenz von Berechnungsfolgen ✓	
3		Freie Definition von Operanden ?  <i>Reg_A ← 3</i> <i>Reg_B ← 2</i> <i>ADD_B</i> <i>Reg_A ← -3</i> <i>MUL_B</i>  ...

Wo kommen aber die Daten her? Bislang haben wir uns damit begnügt anzunehmen, dass diese auf "magische" Art und Weise in unseren Registern stehen.



Im Speicher stehen nun nicht nur Befehle für die Ausführung unserer ALU-Funktionen, sondern auch die Daten für unsere Berechnungen. Auf diese verweisen wir mit separaten Befehlen.

Befehle	Codierung	Bedeutung
OR_A	0000 bisher(!)	Logisches Oder
	....	
LDA Adresse	10000	Laden der Daten von der Adresse X in das Register A
LDB Adresse	10001	Laden der Daten von der Adresse X in das Register B
STA Adresse	10010	Speichern der Daten aus Register A an der Adresse X
STB Adresse	10011	Speichern der Daten aus Register B an der Adresse X

Damit nimmt der Aufwand im Steuerwerk nochmals signifikant zu! Neben dem Adressbus besteht nun ein Datenbus als weiterer Kommunikationspfad.

Schritt	Vorgang
0	Lesen eines Befehls aus dem Speicher
1	Erkennen, dass es sich um ein <code>LDA</code> handelt - Eine Berechnung ist nicht erforderlich.
2	Verschieben des Adresszählers auf die nächste Speicherstelle
3	Aktivieren eines schreibenden Zugriffs auf das Register A der ALU
4	Umleiten der Inhalte aus dem Speicher an die ALU (anstatt an das Steuerwerk)

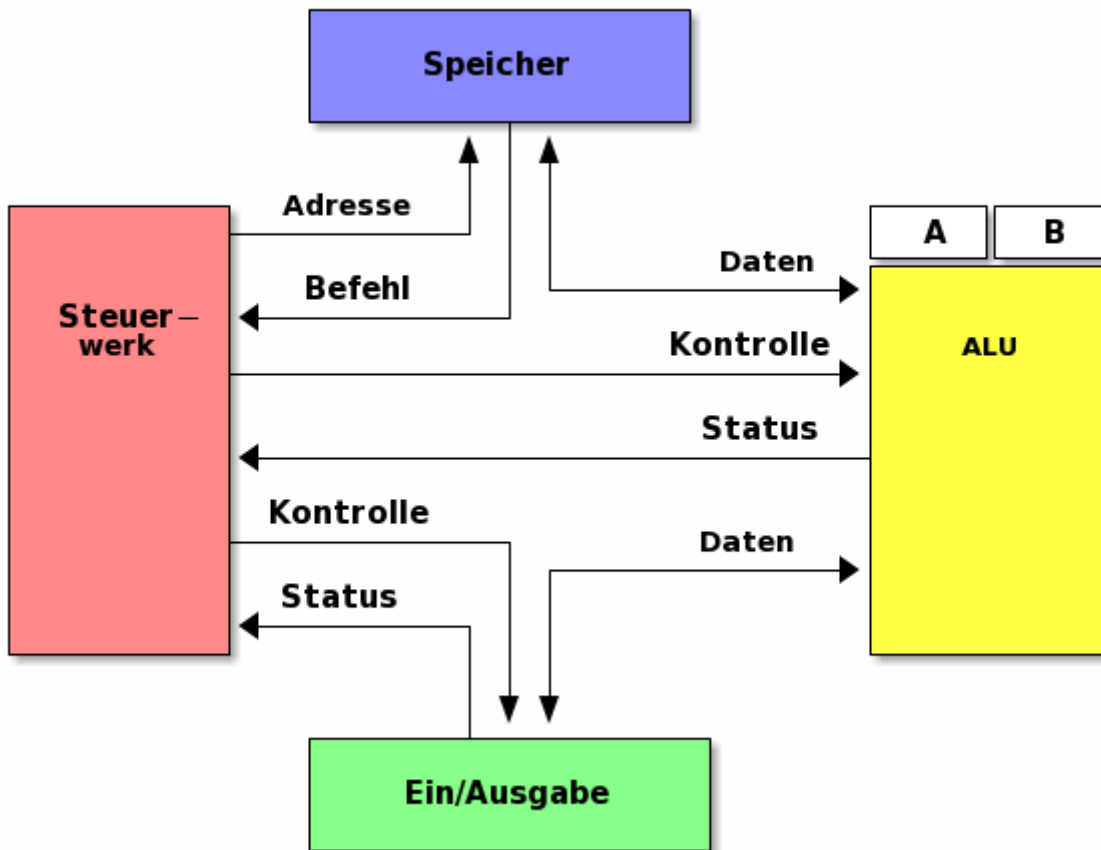
**Achtung:** Offenbar haben wir jetzt 2 Kategorien von Befehlen! `ADD_A` oder `OR_B` werden in einem Zyklus ausgeführt. Die `LDA` oder `STB` Befehle brauchen ein Nachladen der zusätzlichen Parameter (hier Daten).

## Schritt 4 - Ein- und Ausgaben

Zwischenstand

Stufe	Funktionalität	Wunschzettel
0	Addition/Subtraktion von einzelnen Werten ✓	
1	Arithmetische Einheit mit mehreren Funktionen und wählbarem Ergebnisregister ✓	
2	Sequenz von Berechnungsfolgen ✓	
	Darstellung von Programmen als Sequenzen abstrakter Befehle ✓	
3	Flexibler Zugriff auf Daten und Programme im Speicher ✓	
		Ein- und Ausgabe von Daten wäre schön ?

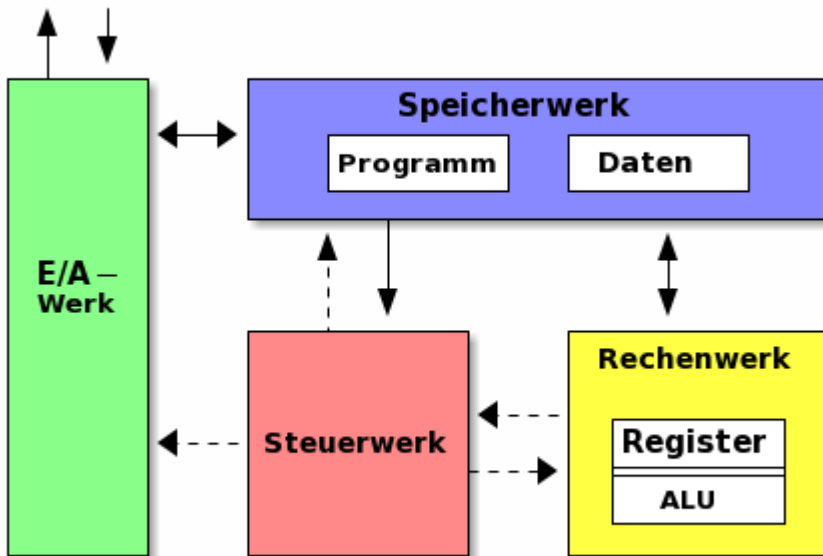
Das Steuerwerk koordiniert neben der ALU die Ein- und Ausgabeschnittstelle.



## 1945: Von-Neumann Architektur

John von Neumann beschrieb 1945 in seinem Aufsatz "First Draft of a Report on the EDVAC" ein strukturelles Konzept, wonach Computerprogramme und die zu verarbeitenden Daten zusammen im gleichen Speicher abgelegt werden sollten. [Link](#)

Einige der Ideen des Konzepts wurden bereits von Konrad Zuse erkannt und teilweise in der Z1 und der Z3 realisiert.



## Hausaufgaben

1. Vergleichen Sie die von-Neumann-Architektur und die Harvard Architektur. Welche Vor- und Nachteile finden sich?
2. Welche Schritte sind notwendig, um einen oder sehr viele neue(n) Befehl(e) zu integrieren? Woran müssen wir arbeiten?