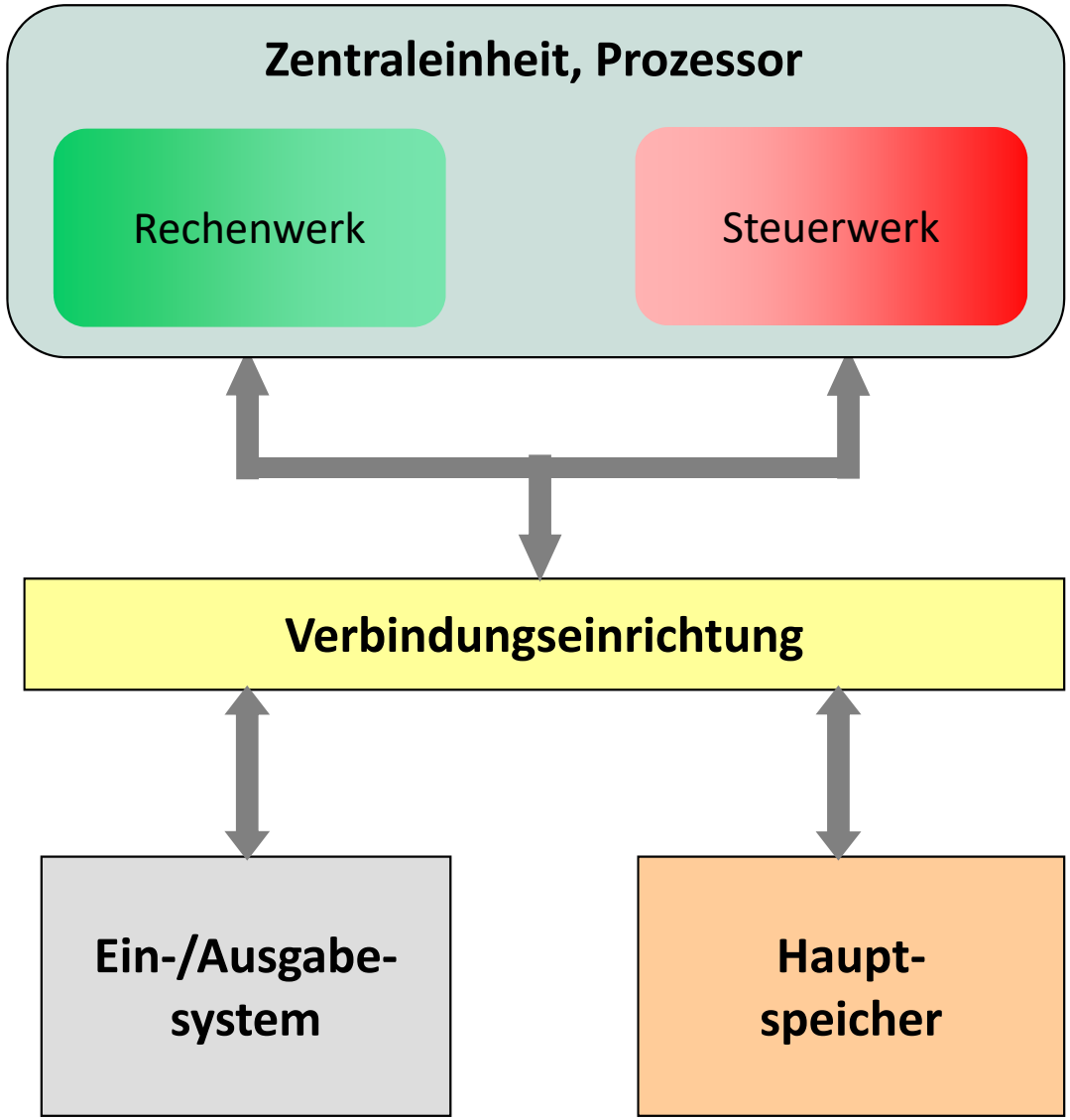
Übung 2

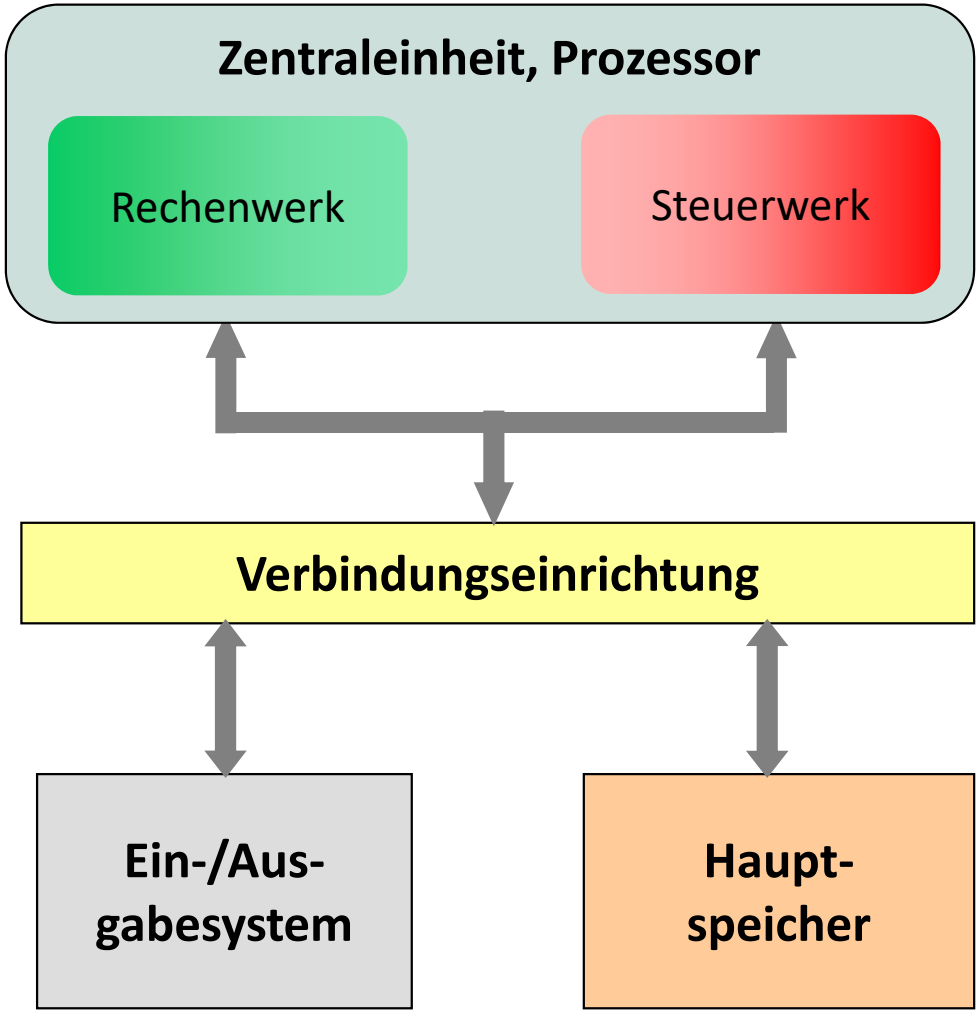
- Mikroprogrammierung
- MIMA-Architektur
- Einführung in die Assembler-Programmierung

Organisationsprinzip des von Neumann Rechners



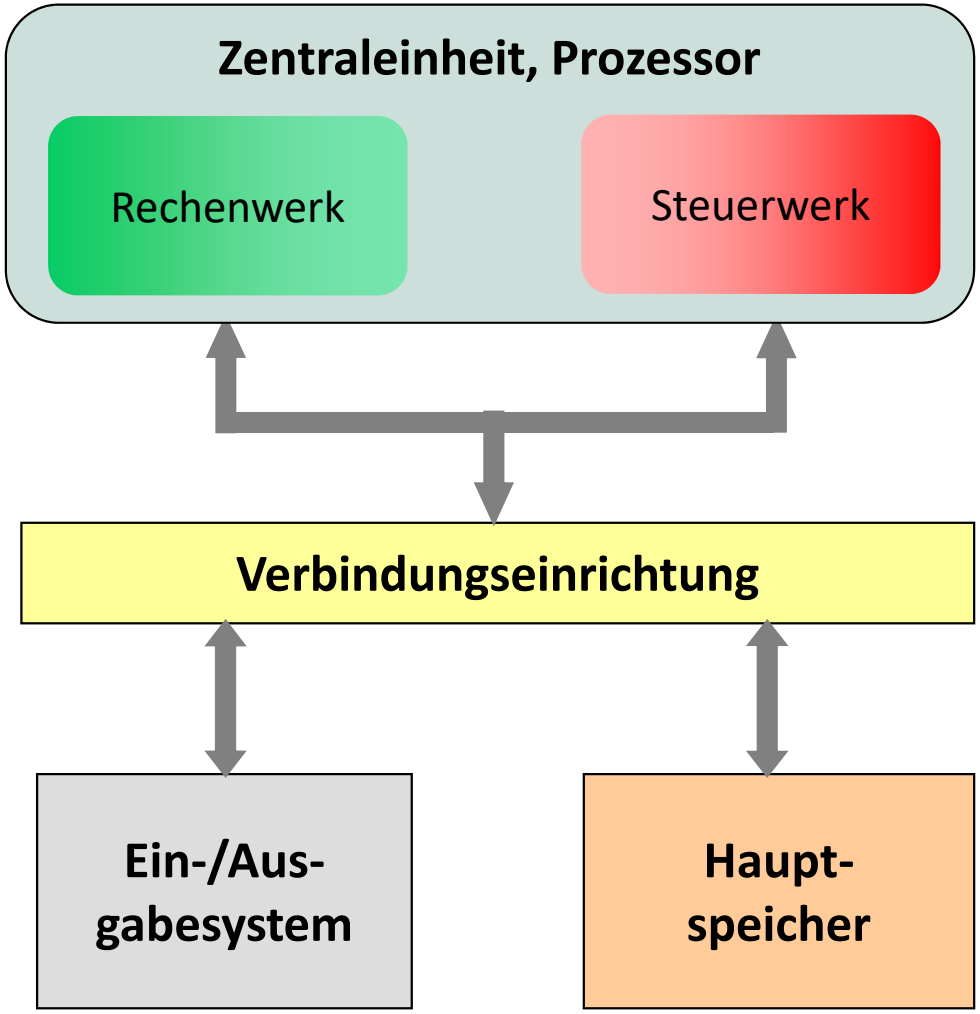
Zentraleinheit

- **Zentraleinheit** (central processing unit, CPU, Prozessor)
  - Verarbeitet Daten gemäß eines Programms.
  - Sie besteht aus Steuerwerk und Rechenwerk.



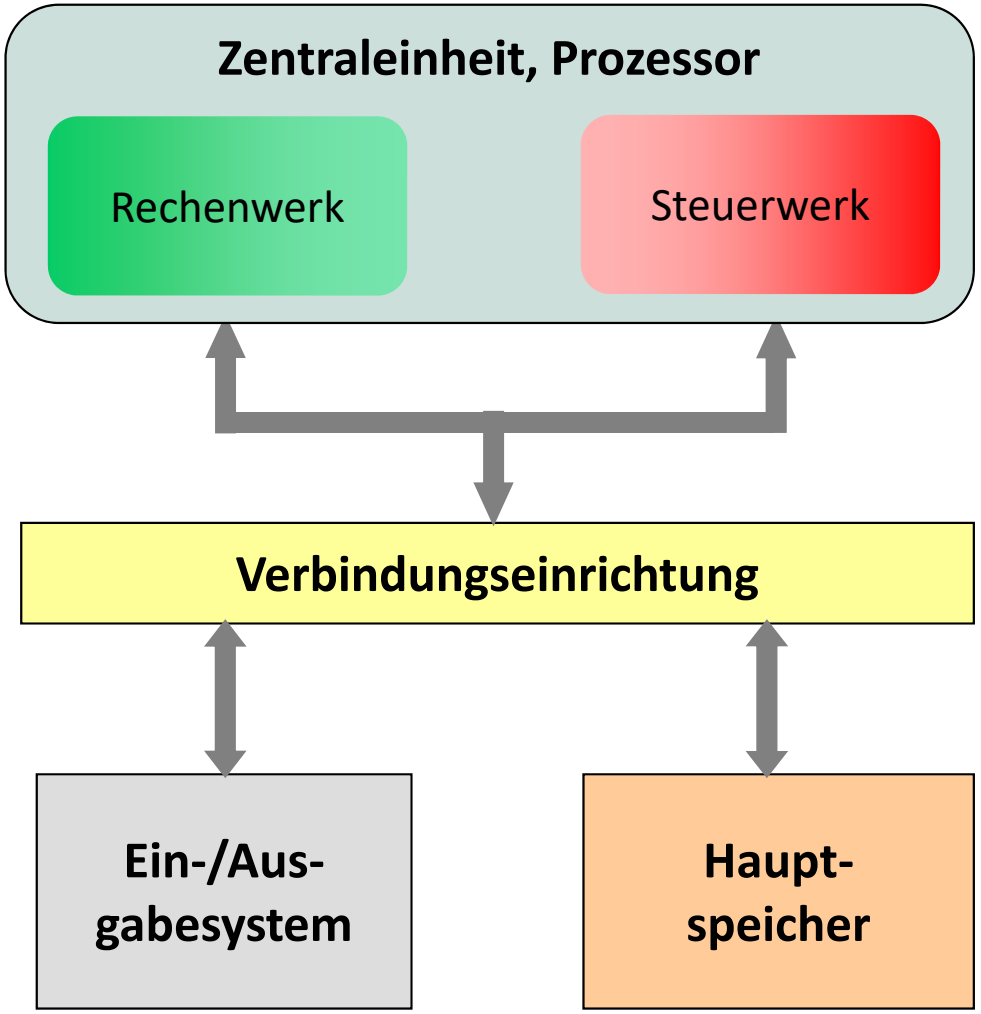
Zentraleinheit: Steuerwerk

- **Steuerwerk** (Leitwerk, control unit, CU)
  - Holt die Befehle eines Programms aus dem Speicher,
  - entschlüsselt sie und
  - steuert ihre Ausführung in der verlangten Reihenfolge durch Steuer- und Synchronisier-Signale.



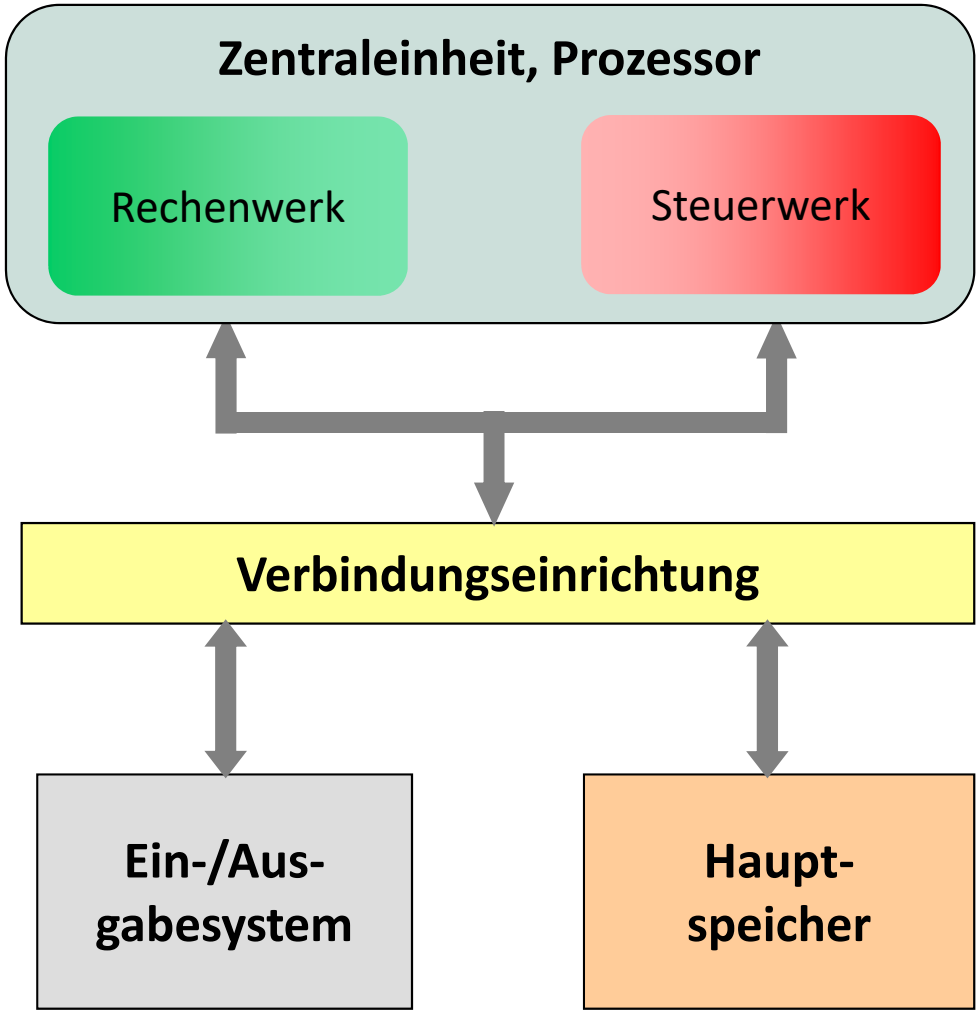
Zentraleinheit: Rechenwerk

- **Rechenwerk (Operationswerk, Ausführungseinheit, Arithmetic Logic Unit: ALU)**
  - Führt arithmetisch/logische Operationen aus.
  - Wird durch Steuersignale des Steuerwerks beeinflußt und
  - liefert Meldesignale an das Steuerwerk zurück.



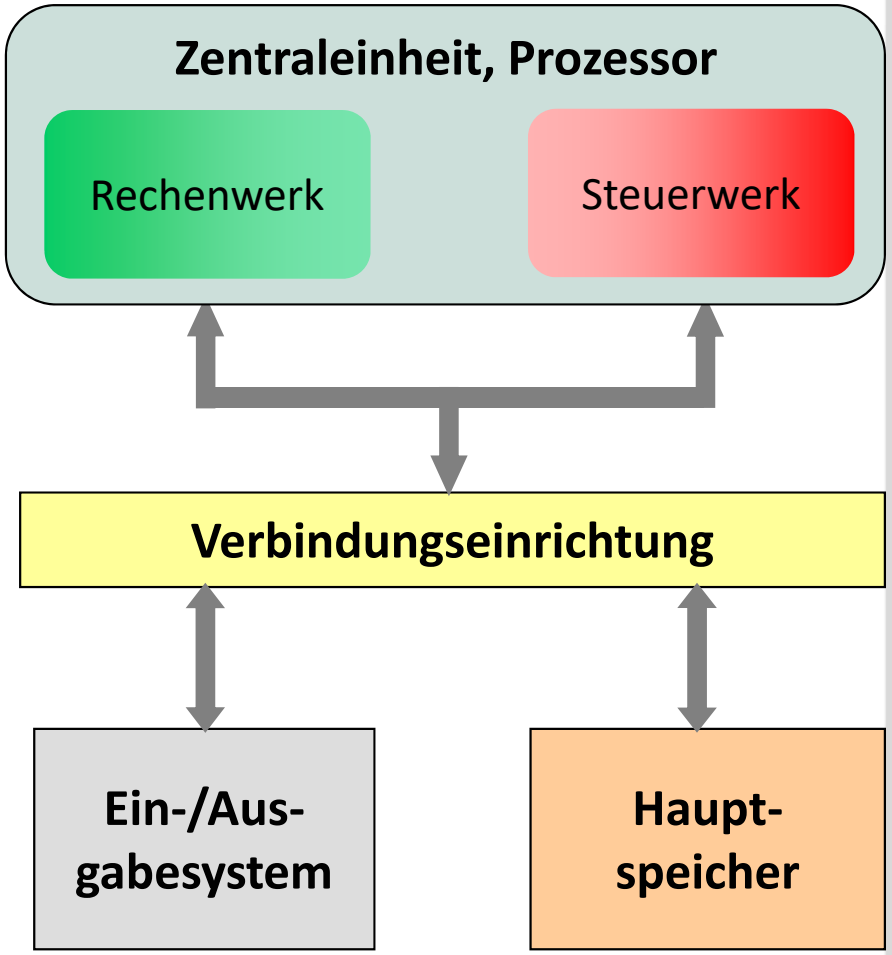
Hauptspeicher

- Jede Speicherzelle ist eindeutig durch ihre Nummer (Adresse) identifizierbar.
- Dort werden Programme und Daten aufbewahrt
- Bei der von-Neumann-Architektur liegen Daten und Befehle in einem gemeinsamen Speicher → Den einzelnen Speicherzellen ist nicht anzusehen, welchen Typ von Information sie enthalten.
  - Alternativ: **Harvard-Architektur** mit getrenntem Programm- und Datenspeicher.
- Inhalt nach Abschaltung des Rechners flüchtig.



Verbindungstruktur (BUS)

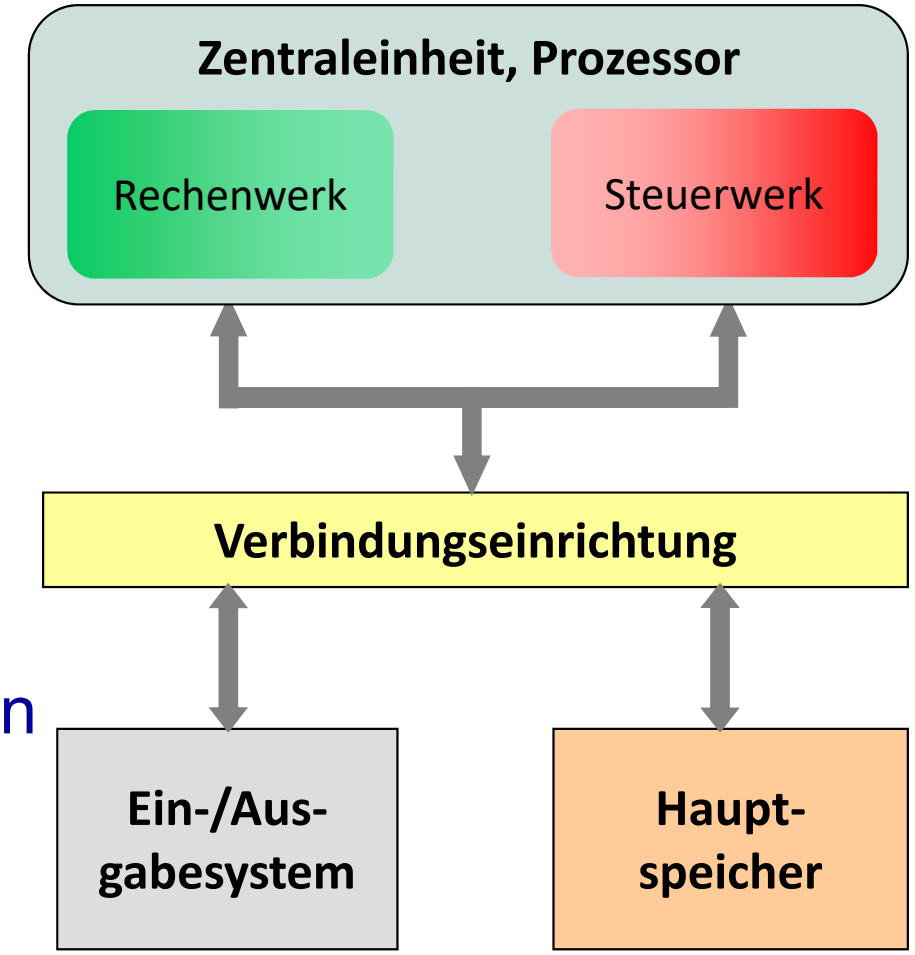
- **Verbindungstruktur (BUS)**
  - **Adreßleitungen:** Leitungen, auf denen die Adreßinformation transportiert wird (unidirektional).
  - **Datenleitungen:** Transportieren Daten und Befehle von/zum Prozessor (bidirektional).
  - **Steuerleitungen**: Geben Steuer-informationen von/zum Prozessor (uni- oder bidirektional).



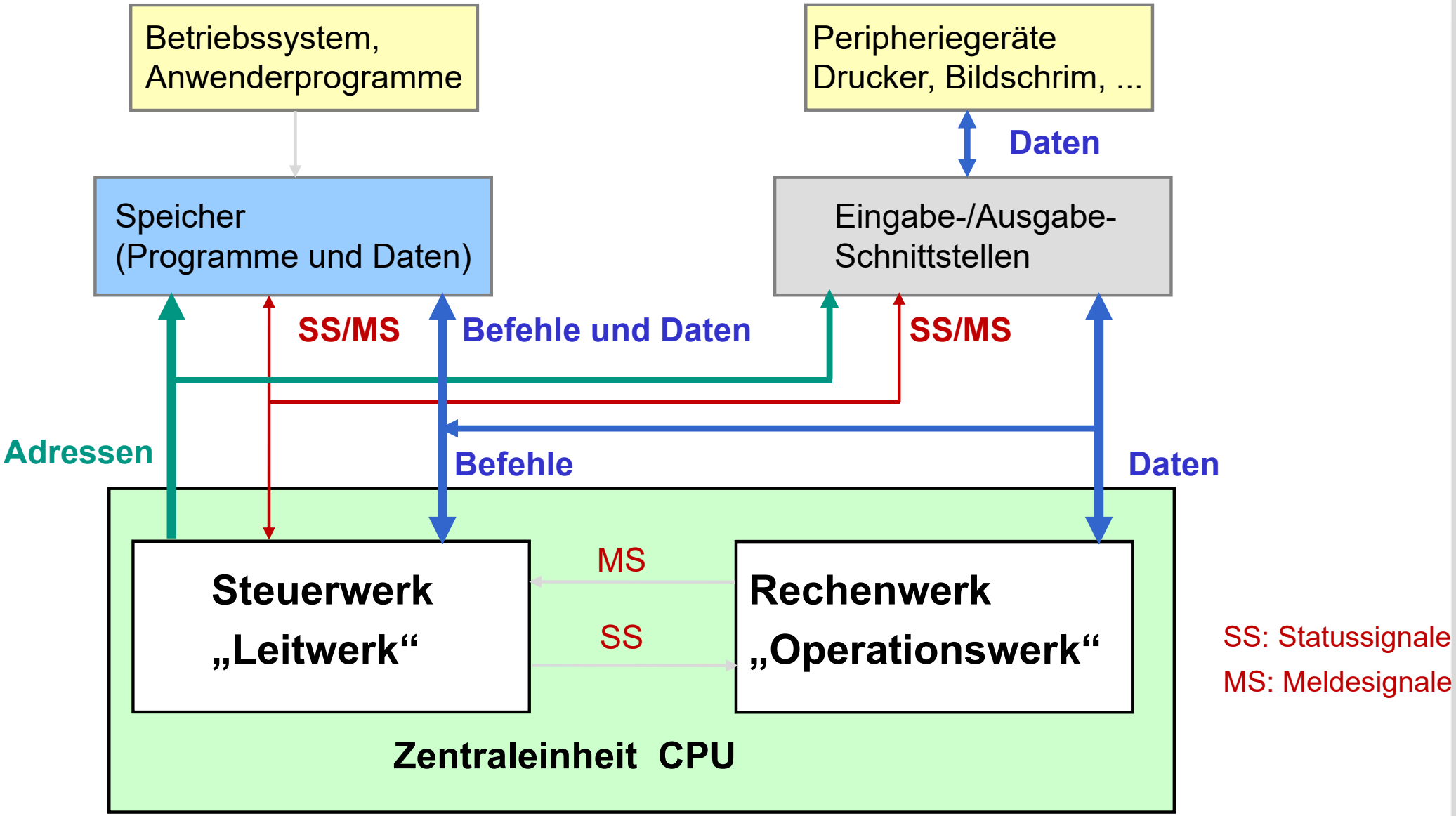
- **Bus (Sammelschiene)**
  Systembus = Adreßbus + Datenbus + Steuerbus

Ein-/Ausgabesystem (Peripheriegeräte)

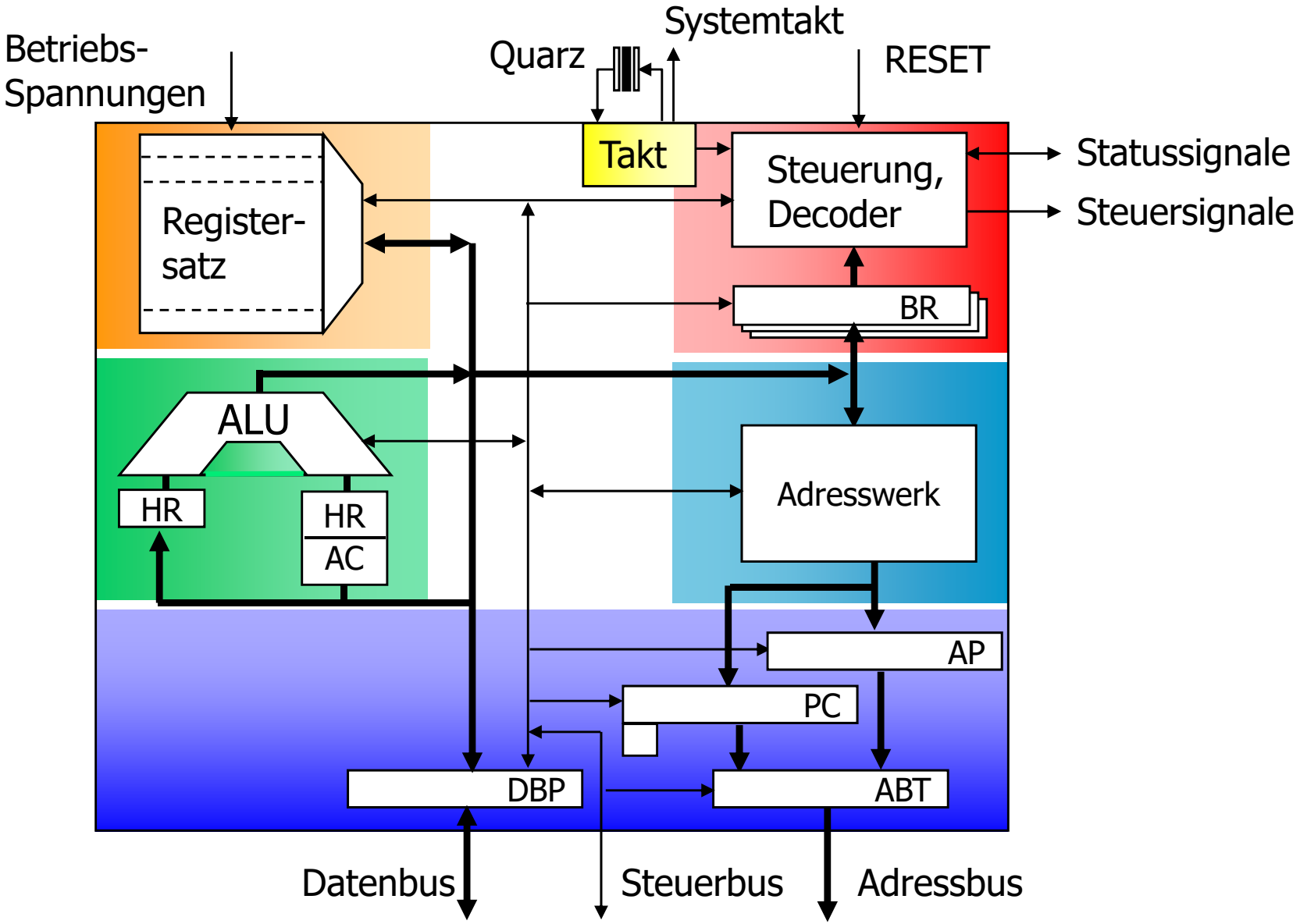
- Geräte zur Eingabe von Daten und Programmen und zur Ausgabe der verarbeiteten Daten (Bildschirme, Drucker, Terminals, ...)
- Diese Geräte sind über Ein-/Ausgabe-Schnittstellen mit dem Rechner verbunden.
- Die Verbindung der Schnittstellen mit dem Prozessor (und zu den Peripheriegeräten) geschieht durch Adreß-, Daten- und Steuerleitungen.



Komponenten eines von-Neumann Rechners



Aufbau eines einfachen µP



Karlsruher Institut für Technologie

MIKROPROGRAMMIERUNG

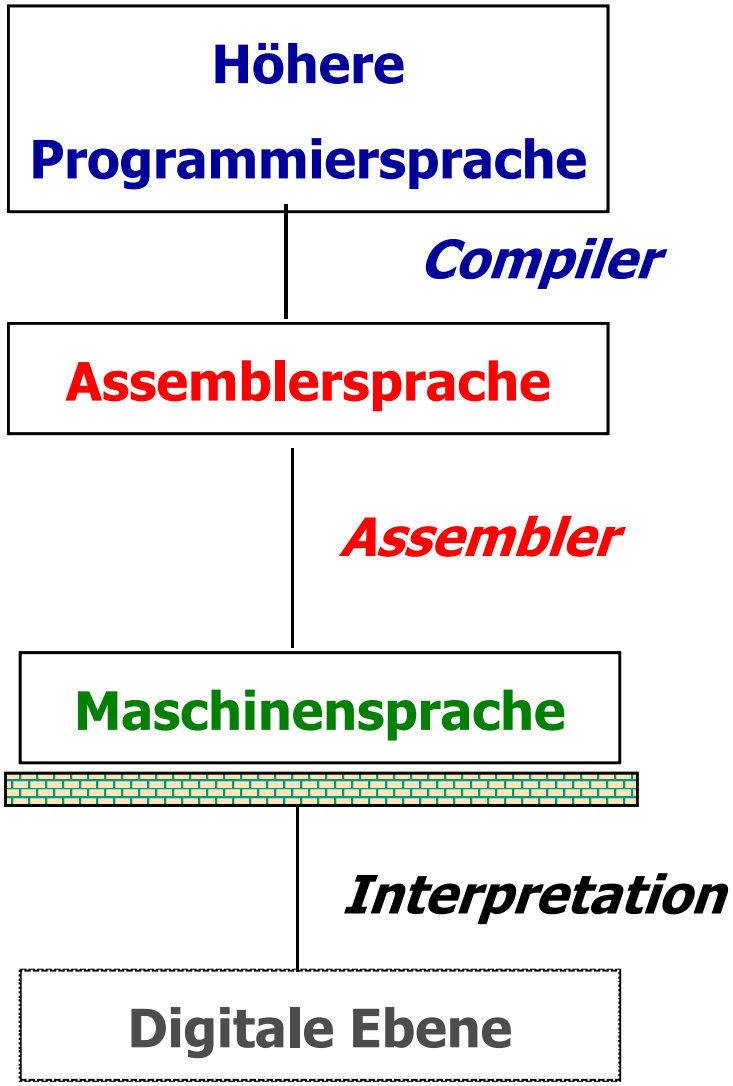
Mikroprogrammierung

- Zwei Techniken zur Implementierung von Maschinenbefehlen im Rechner
  - Die Ausführung eines Maschinenbefehls wird direkt in Hardware implementiert → festverdrahtetes Steuerwerk
  - Die Ausführung eines Maschinenbefehls wird durch ein Mikroprogramm implementiert → mikroprogrammiertes Steuerwerk

Mikroprogramm

- Ein Mikroprogram eines Maschinenbefehls besteht aus Mikrobefehlen
- Ein Mikrobefehl besteht aus einer Bitfolge, die unmittelbar Steuersignale für verschiedene Komponenten des Rechners darstellen

Hierarchie




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw   $15,  0($2)
lw   $16,  4($2)
sw   $16,  0($2)
sw   $15,  4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

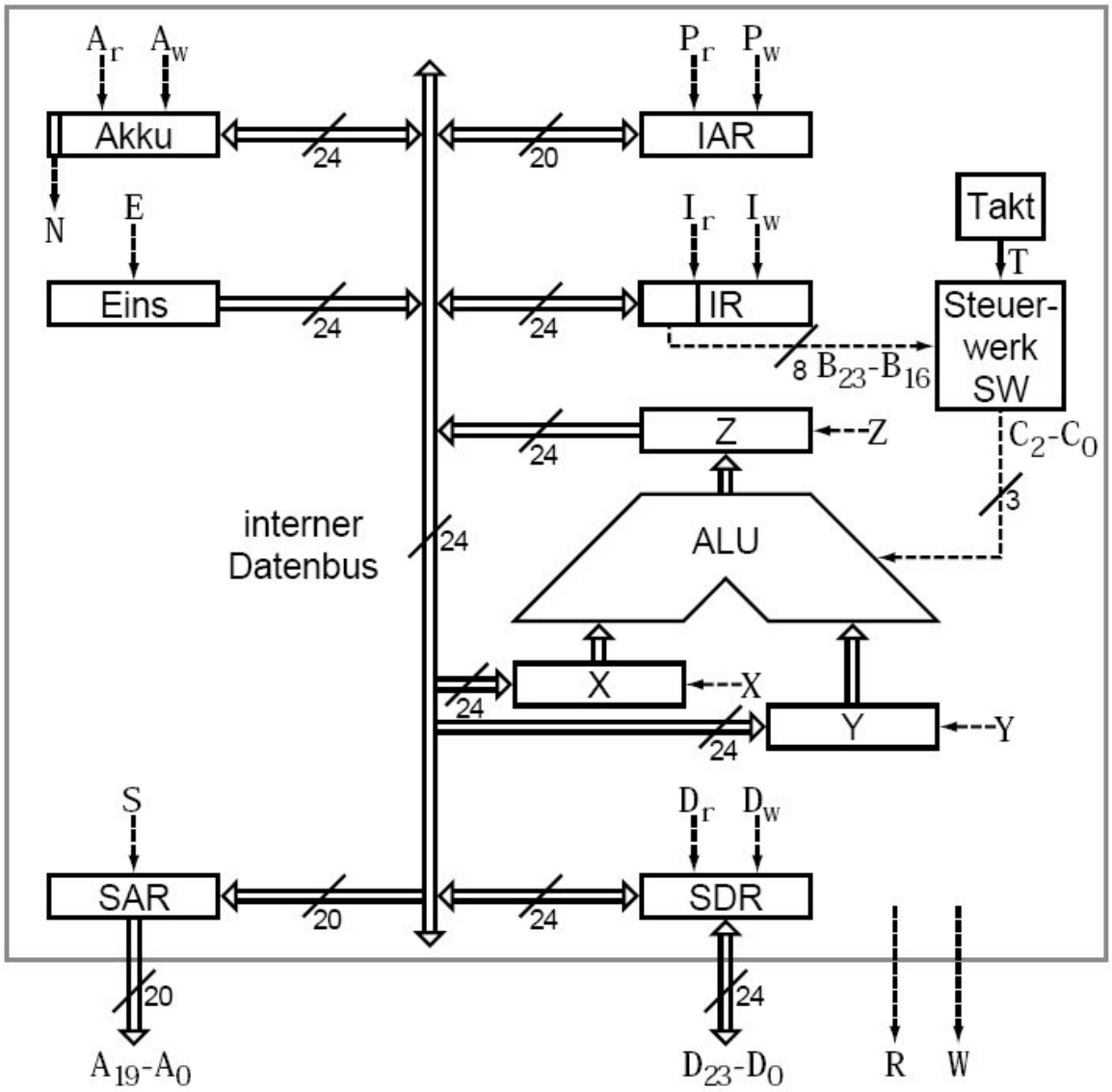
```
ALUOP[0:3] ← InstReg[9:11] & MASK
```

MIMA-Architektur

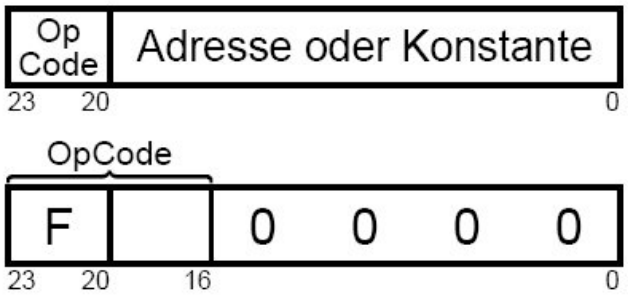
Mikroprogrammierte Minimalmaschine

MIMA-Architektur

- Mikroprogrammierte Minimalmaschine (von-Neumann-Prinzip)
- SW mit 10 Meldesignalen, 18 Steuersignale und Mikroprogrammspeicher für maximal 256 Mikrobefehle
- Befehlsabarbeitung:
  - Lese-Phase
  - Dekodierphase
  - Ausführungsphase
- 3 Taktzyklen für Lese- und Schreibzugriffe



Befehlsformate, ALU-Operationen, ...



$C_2C_1C_0$	*ALU Operation*
0 0 0	tue nichts (d.h. Z -> Z)
0 0 1	X + Y -> Z
0 1 0	rotiere X nach rechts -> Z
0 1 1	X AND Y -> Z
1 0 0	X OR Y -> Z
1 0 1	X XOR Y -> Z
1 1 0	Eins-Komplement von X -> Z
1 1 1	falls X = Y, -1 -> Z, sonst 0 -> Z

OpCode	*Mnemonik*	*Beschreibung*
0	LDC c	c -> Akku
1	LDV a	<a> -> Akku
2	STV a	Akku -> <a>
3	ADD a	Akku + <a> -> Akku
4	AND a	Akku AND <a> -> Akku
5	OR a	Akku OR <a> -> Akku
6	XOR a	Akku XOR <a> -> Akku
7	EQL a	falls Akku = <a>:-1 -> Akku
		sonst : 0 -> Akku
8	JMP a	a -> IAR
9	JMN a	falls Akku < 0 : a -> IAR
F0	HALT	stoppt die MIMA
F1	NOT	bilde Eins-Komplement von Akku -> Akku
F2	RAR	rotiere Akku eins nach rechts -> Akku

MIMA-Architektur

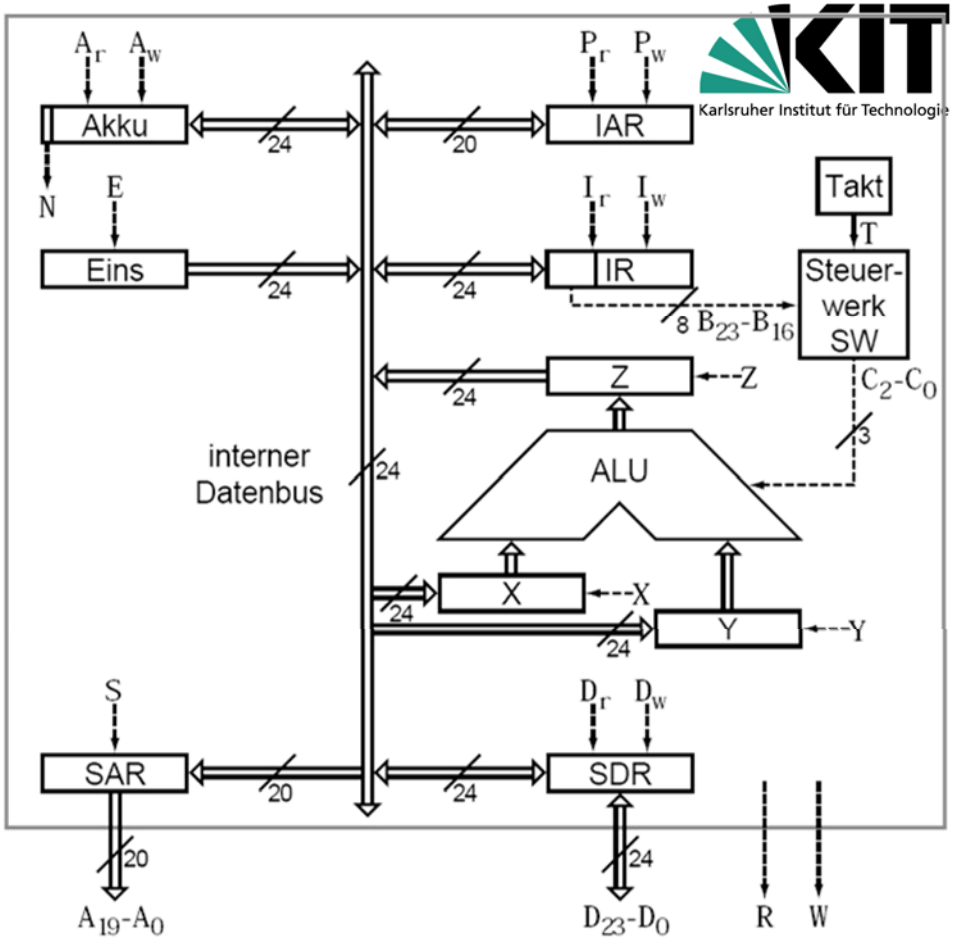
1.Takt: `IAR → SAR; IAR → X; R = 1`

2.Takt: `Eins → Y; R = 1`

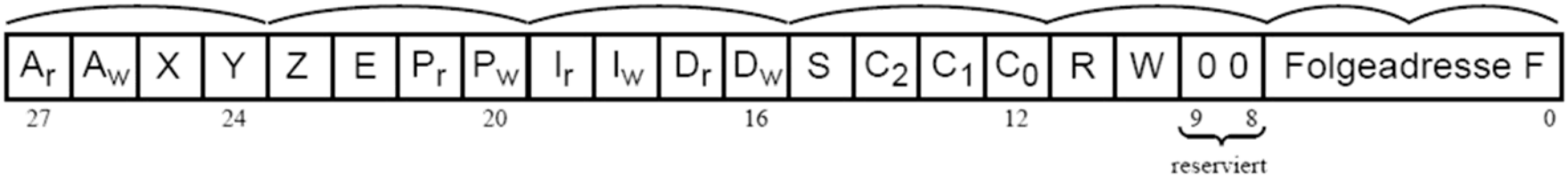
3.Takt: `ALU auf Addieren; R = 1`

4.Takt: `Z → IAR`

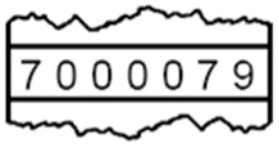
5.Takt: `SDR → IR`



Mikrobefehlsformat



Beispiel:



$A_w = X = Y = 1$ (Akku -> X; Akku -> Y)

Adresse des nächsten Befehls ist 0x79

MIMA-Architektur

- Mikroprogramm für die Lesephase aller Befehle besteht aus 5 Mikrobefehlen:

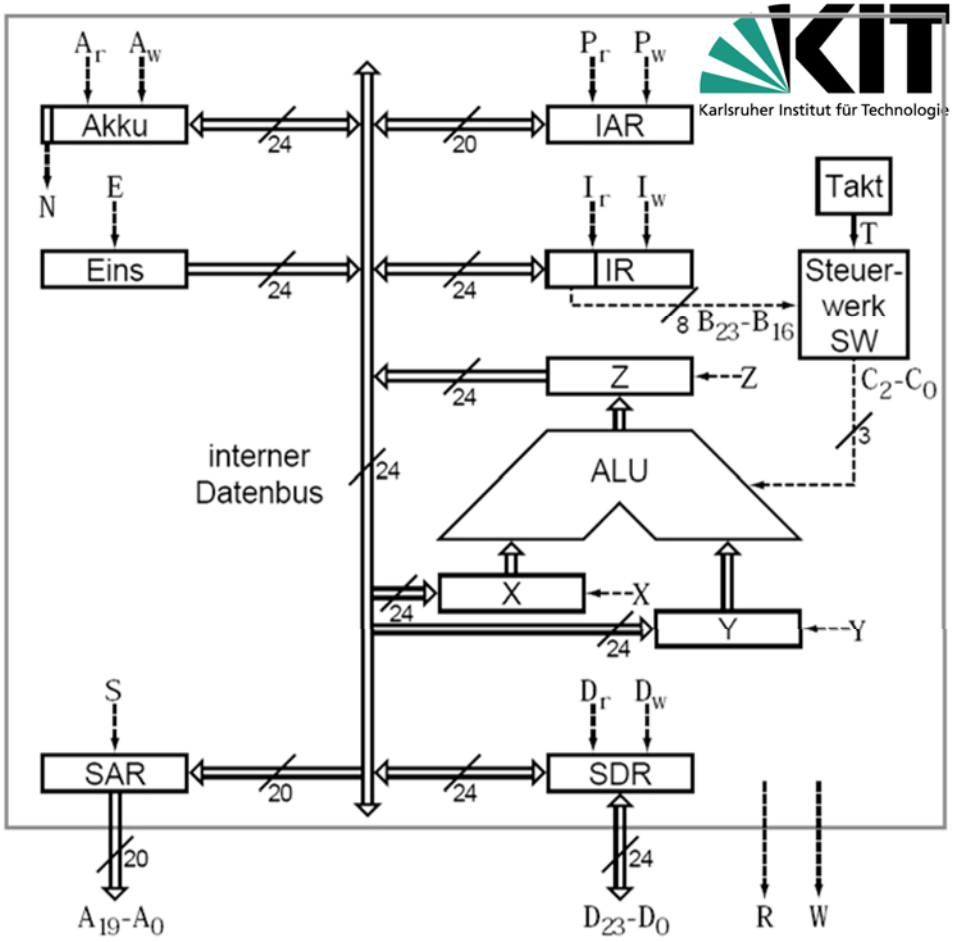
1.Takt: `IAR → SAR; IAR → X; R=1`

2.Takt: `Eins → Y; R=1`

3.Takt: `ALU auf Addieren; R=1`

4.Takt: `Z → IAR`

5.Takt: `SDR → IR`

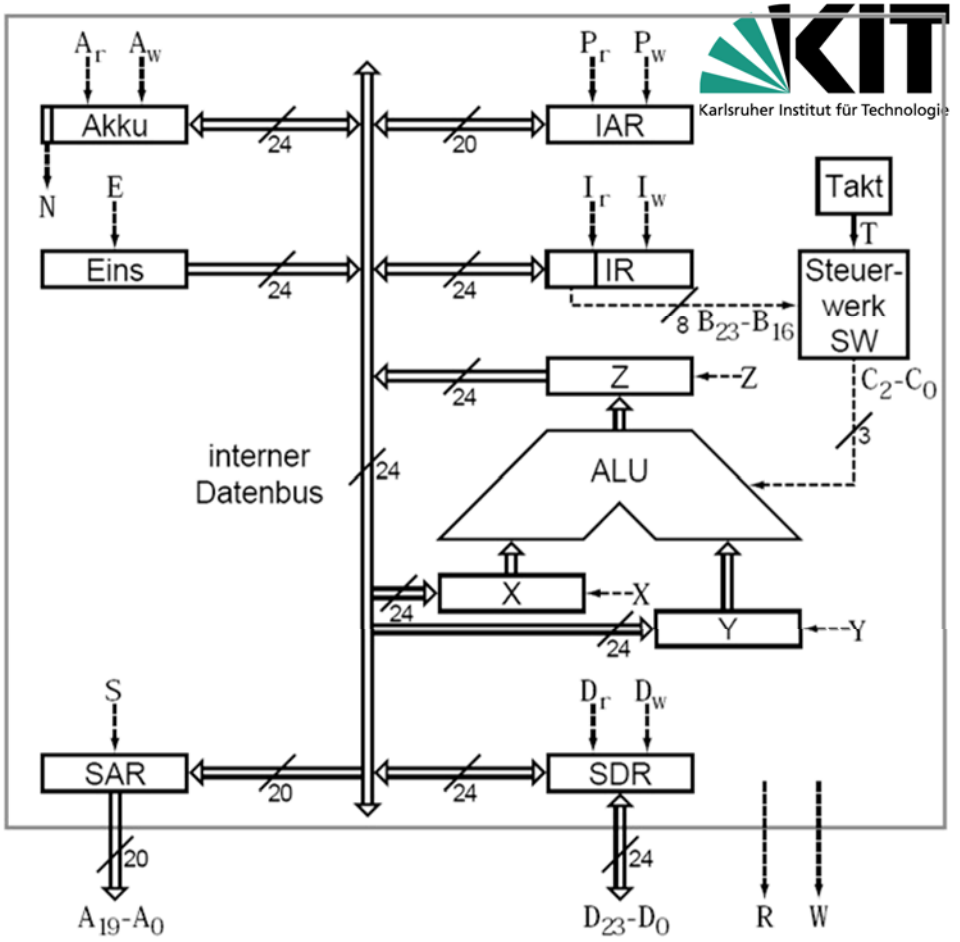


0x00	2 1 0 8 8 0 1	$X = P_w = S = 1; R = 1$
0x01	1 4 0 0 8 0 2	$Y = E = 1; R = 1$
0x02	0 0 0 1 8 0 3	$C_2\text{-}C_0 = 001; R = 1$
0x03	0 A 0 0 0 0 4	$Z = P_r = 1$
0x04	0 0 9 0 0 0 5	$I_r = D_w = 1$

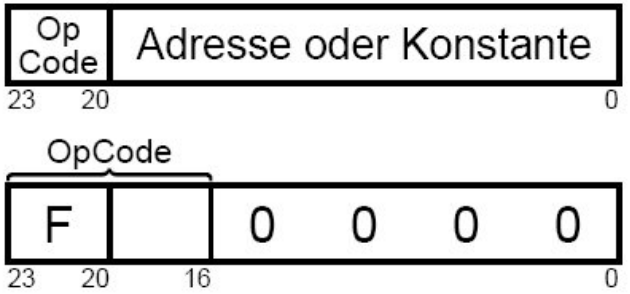
Beispiel

Mikroprogramm für:

`OR a  Akku OR <a> → Akku`



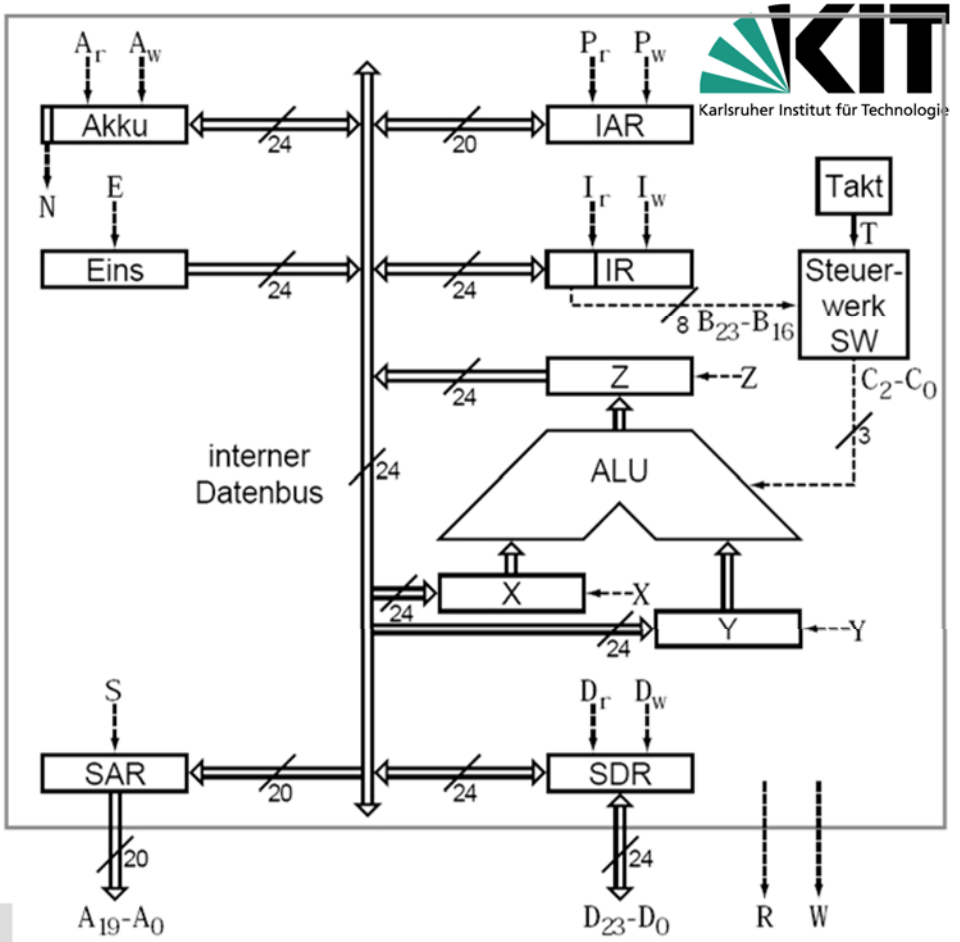
7. Takt: **`IR → SAR; R = 1`**
8. Takt: **`Akku → X; R = 1`**
9. Takt: **`R = 1`**
10. Takt: **`SDR → Y`**
11. Takt: **`ALU auf OR (C2C1C0 = 100)`**
12. Takt: **`Z → Akku`**



Beispiel

Mikroprogramm für:

```
EQL a     falls Akku = <a>:   -1  → Akku
          sonst               0 → Akku
```



7. Takt: `IR → SAR; R = 1`

8. Takt: `Akku → X; R = 1`

9. Takt: `R = 1`

10. Takt: `SDR → Y`

11. Takt: `ALU auf Vergleich` ($C_2C_1C_0$ = 111)

12. Takt: `Z → Akku`

MIMA-Architektur

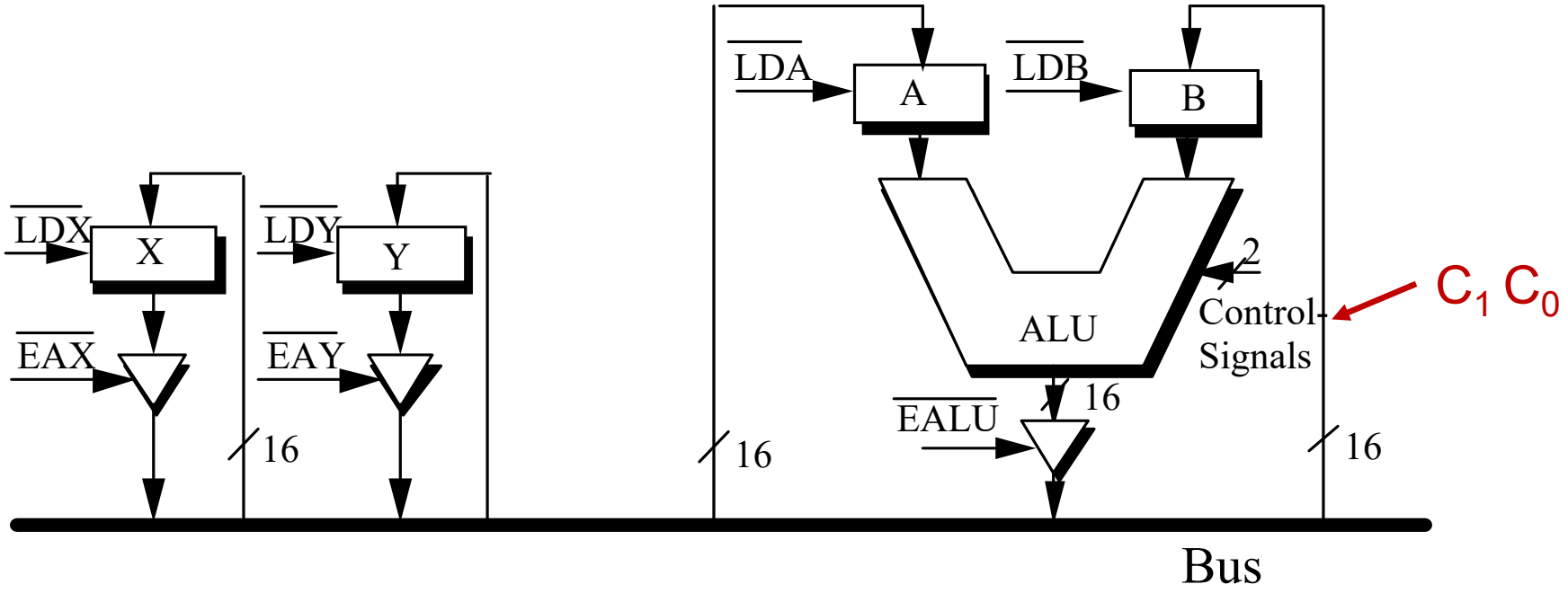
- JAVA-Simulation der MIMA, JavaScript Visualisierung, Beispielprogramme und ein c-Interpreter auf der TI-Homepage:

  http://ti.ira.uka.de/Visualisierungen/Mima/

Aufgabe

Gegeben:

Eine mikroprogrammierbare Schaltung besteht aus einem Bus, 4 Registern, einer ALU und einigen Tristate-Treibern



Aufgabe

Gesucht:

Mikroprogramme für die folgenden Operationen 1-5 durch Programmierung eines Datenpfades:

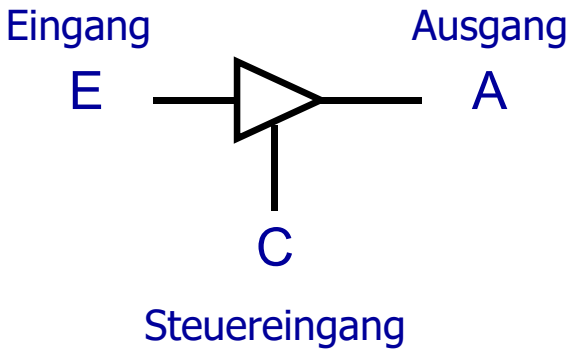
		Kontrollsignale	Operation
		C_1 C_0	
1.	x = x + y	0 0	A + B
2.	x = x - y	0 1	A - B
3.	x = x and y	1 0	A und B
4.	x = x or y	1 1	A oder B
5.	y = x (move x to y)		

Tri-State-Treiber

Gatter, die neben den Pegelzuständen **H** (high) und **L** (low) einen **dritten hochohmigen Zustand** besitzen.

- In diesem Zustand ist der Ausgang hochohmig gegen Betriebsspannungen beider Polaritäten.
- Diese Gatter ermöglichen es, mehrere Gatter-ausgänge auf eine gemeinsame Leitung zusammenzuschalten (Bussystem)
- Sie dienen auch durch Ausgangstreiber zur elektrischen Anpassung der Prozessorsignale an die Signalspezifikationen, die von anderen Systemkomponenten verlangt werden.

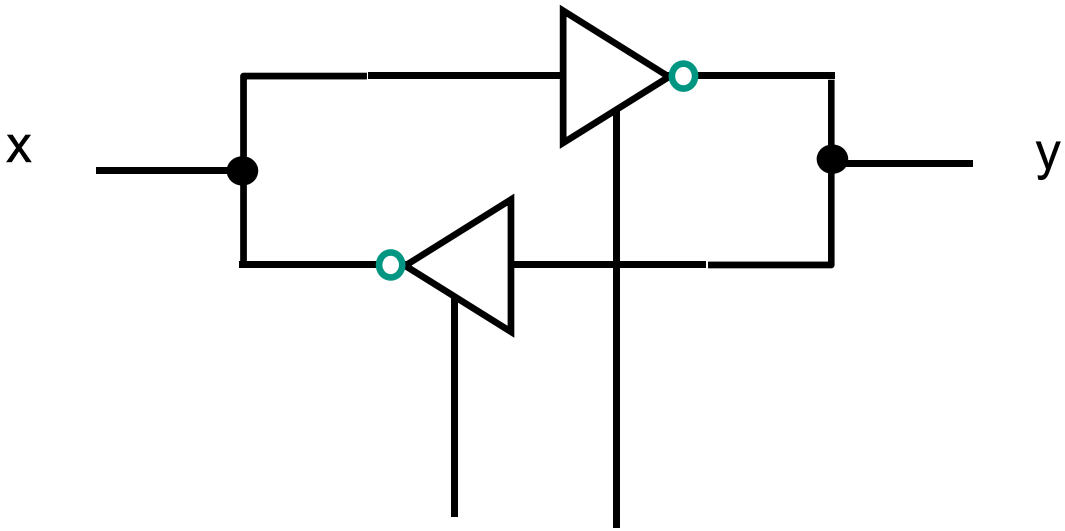
Tri-State-Treiber



Funktionstabelle:

C	E	A
H	L	L
H	H	H
L	-	Z hochohmig

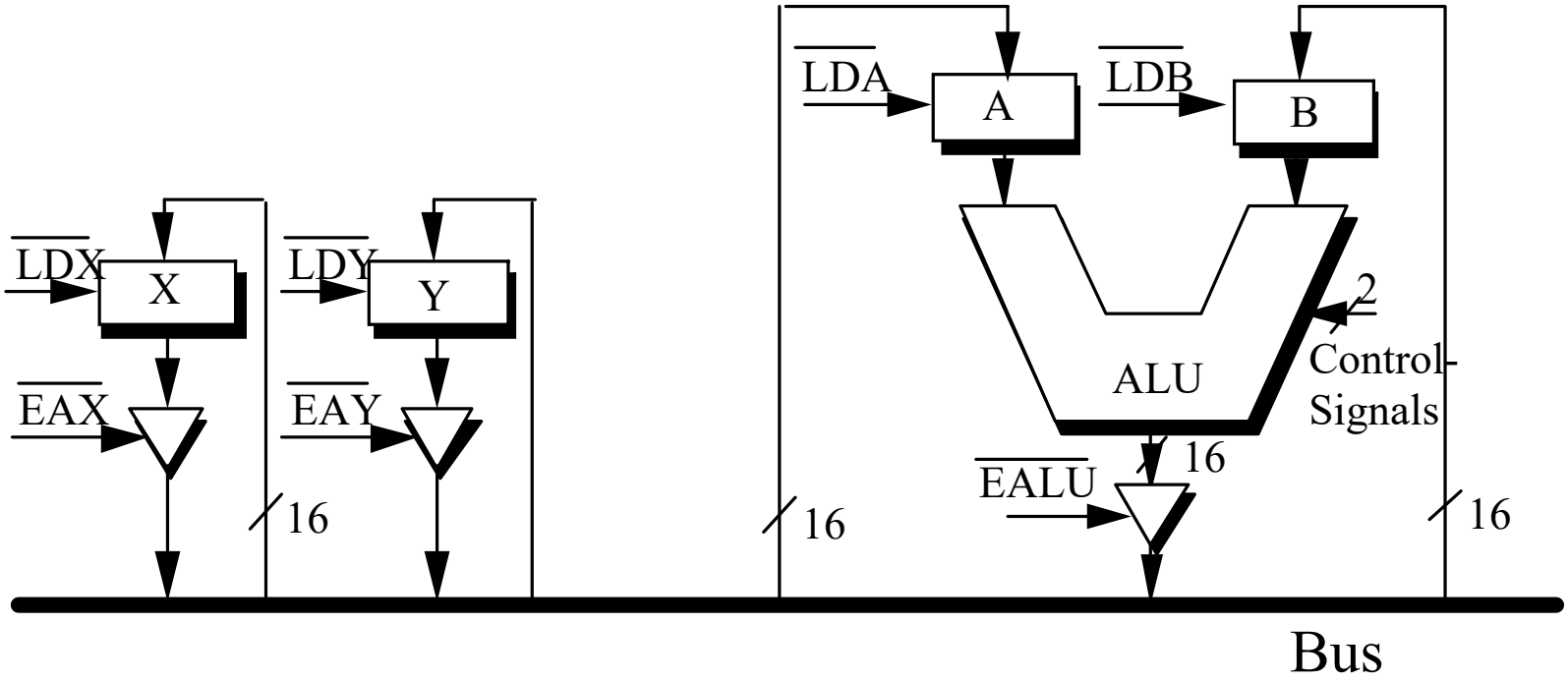
Bidirektionale Tristate-Gatter



Funktionstabelle:

C	DIR	
L	L	x → y
L	H	y → x
H	-	Z hochohmig

Aufgabe



$\overline{LDX} = 0$ → Register X laden

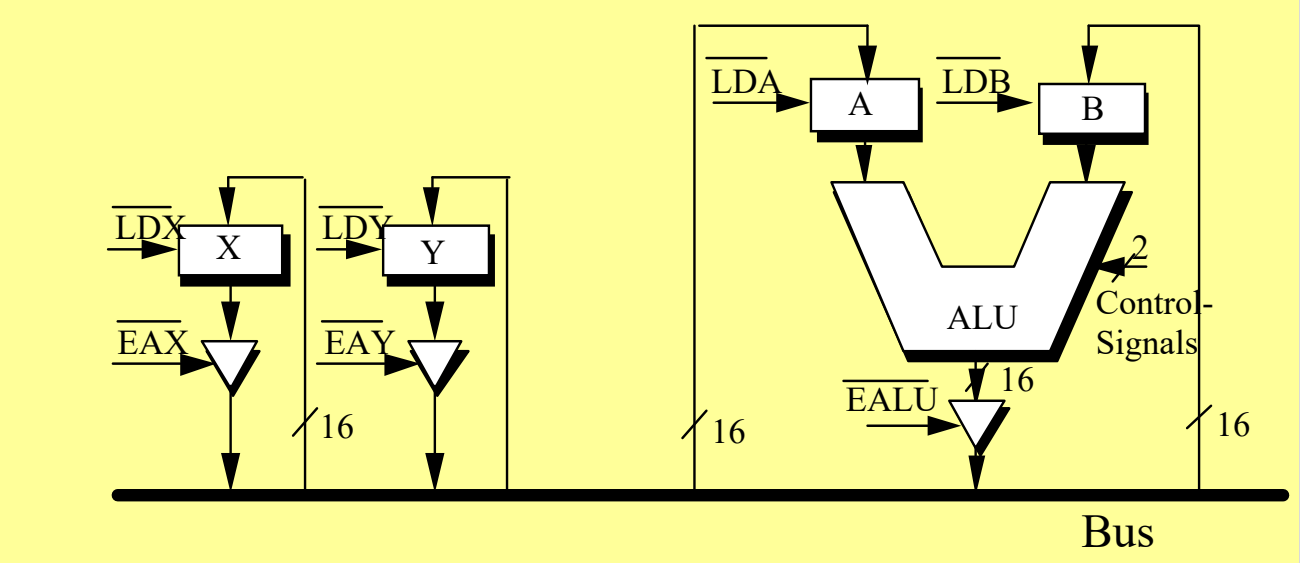
$\overline{EAX} = 0$ → Register X auf den Bus

$C_1 C_0 = 00$ → ALU addiert

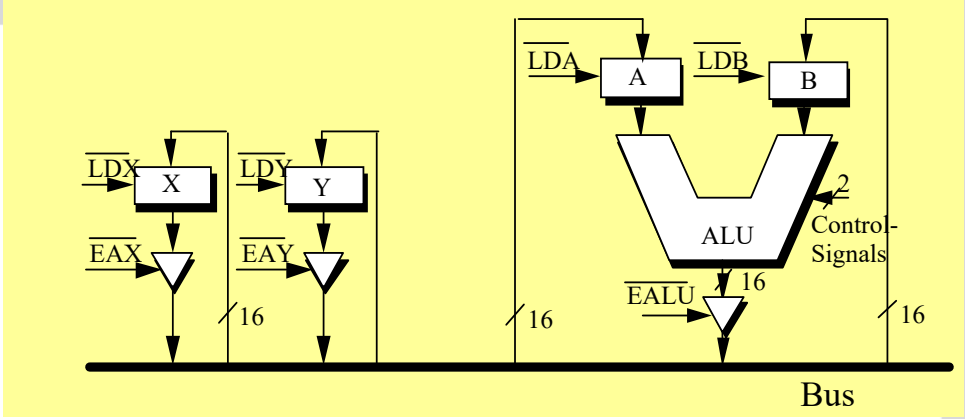
Lösung

Mikroprogramme für die Operationen 1-4:

- X auf dem Bus legen. Warten bis die Daten stabil anliegen.
- X ins Register A laden.
- Y auf dem Bus legen. Warten bis die Daten stabil anliegen.
- Y ins Register B laden.
- Ergebnis auf den Bus. Warten bis es stabil anliegt.
- Ergebnis ins Register X laden.



Lösung

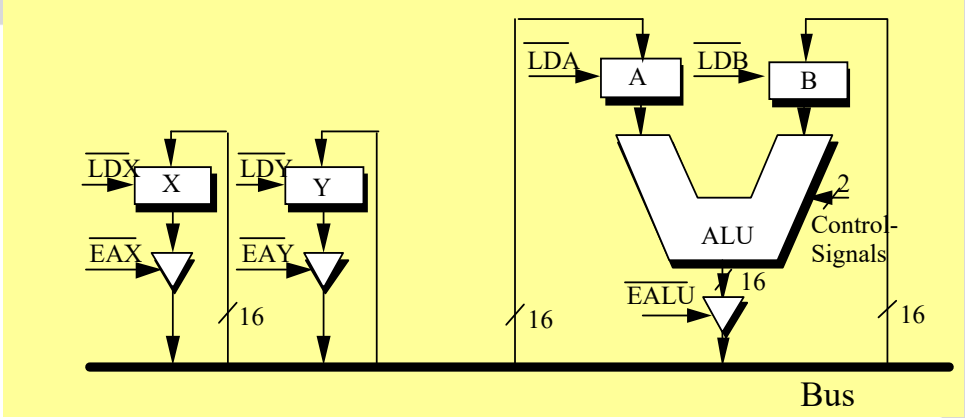


$\overline{EAX}$	$\overline{LDX}$	$\overline{EAY}$	$\overline{LDY}$	$\overline{LDA}$	$\overline{LDB}$	$\overline{EALU}$	C_1	C_0	
0	1	1	1	1	1	1	-	-	
0	1	1	1	0	1	1	-	-	
1	1	0	1	1	1	1	-	-	
1	1	0	1	1	0	1	0	0	
1	1	1	1	1	1	0	0	0	x = x+y
1	0	1	1	1	1	0	0	0	

0 1	x = x-y
1 0	x = x and y
1 1	x = x or y

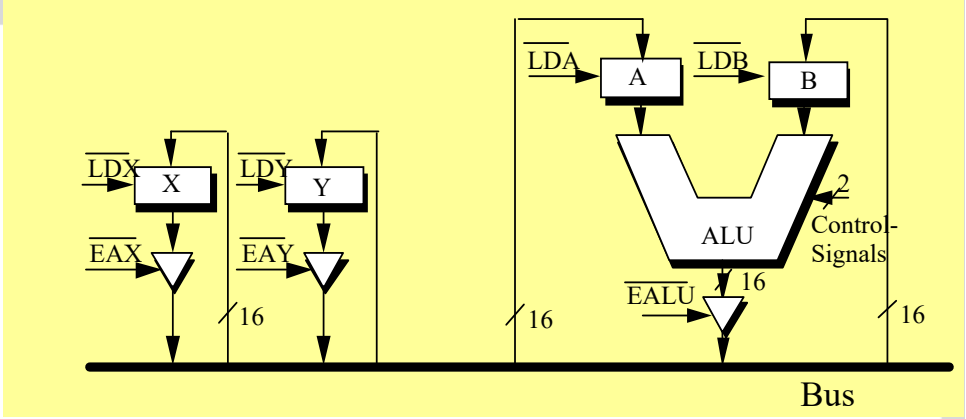
Lösung

$\overline{EAX}$	0	0	1	1	1	1
$\overline{LDX}$	1	1	1	1	1	0
$\overline{EAY}$	1	1	0	0	1	1
$\overline{LDY}$	1	1	1	1	1	1
$\overline{LDA}$	1	0	1	1	1	1
$\overline{LDB}$	1	1	1	0	1	1
	1	1	1	1	0	0
	-	-	-	0	0	0
$\overline{EALU}$	-	-	-	0	0	0
C_1						
C_0						



0	1	1	x = x+y
1	0	1	x = x-y
			x = x and y
			x = x or y

Lösung



Mikroprogramm für die Operation 5 (x ➔ y):

- X auf dem Bus legen. Warten bis die Daten stabil anliegen.
- Y laden.

$\overline{EAX}$	$\overline{LDX}$	$\overline{EAY}$	$\overline{LDY}$	$\overline{LDA}$	$\overline{LDB}$	$\overline{EALU}$
0	1	1	1	1	1	1
0	1	1	0	1	1	1