Technische Hochschule Ingolstadt
Fakultät Informatik

Einführung in die Informatik 1

Studiengang Künstliche Intelligenz

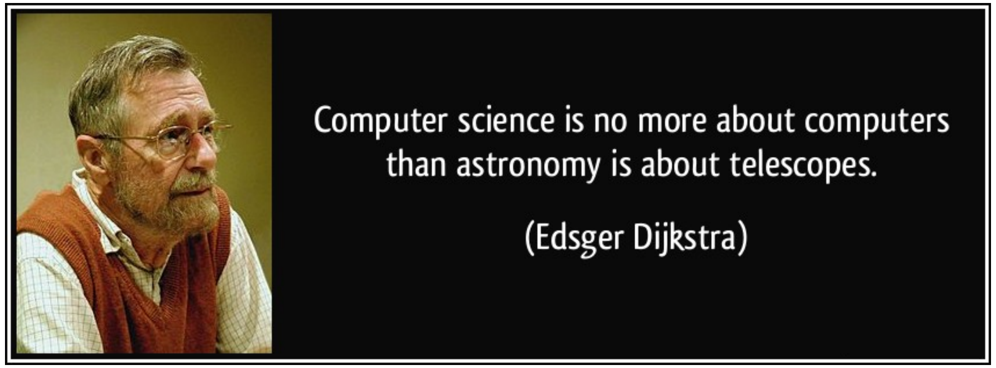
Prof. Dr. Wolf-Dieter Tiedemann

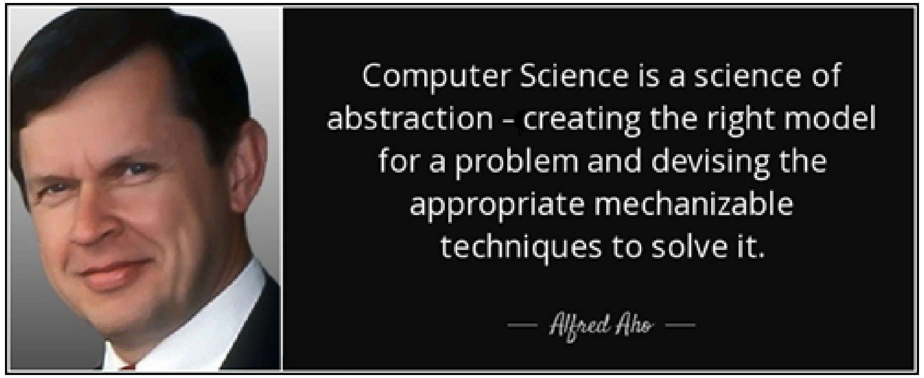
WS 2019/20

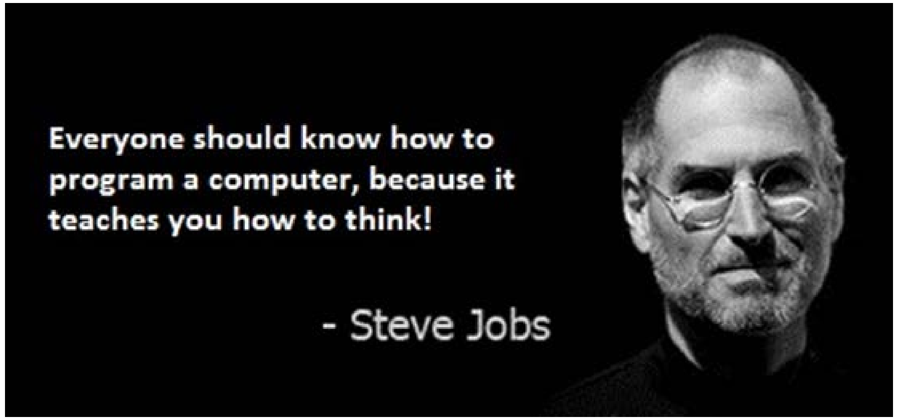
- **Was ist Informatik?**

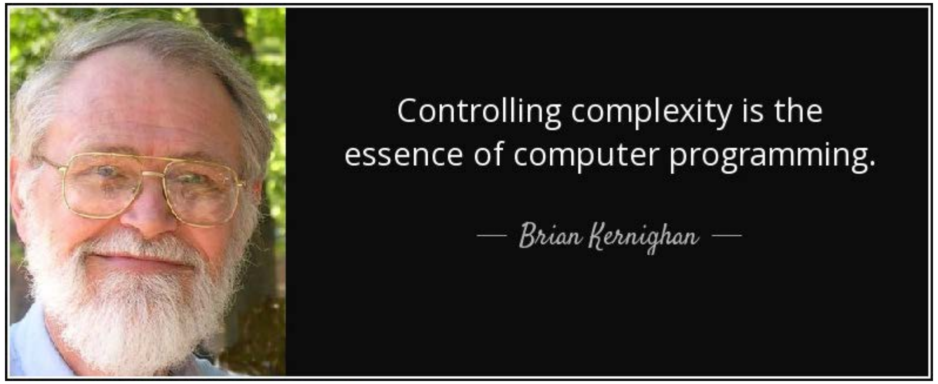
The European synonym for computer science — informatics — more clearly suggests the field is about information processes, not computers.

PETER J. DENNING









Computer Science

is the systematic study of **algorithmic processes** that describe and transform information: their theory, analysis, design, efficiency, implementation and application.

Association of Computing Machinery, 1989

Zentrale Fragestellungen dieser Vorlesung

- **Welche Probleme können grundsätzlich durch einen Algorithmus gelöst werden und darüber hinaus effizient gelöst werden?**
- **Wie muss eine Maschine beschaffen sein, um einen als Programm formulierten Algorithmus auszuführen?**

Algorithmus

- Der Begriff leitet sich ab aus dem Namen des persischen Mathematikers und Astronomen *Muhammad Ibn Musa Al-Chwarizmi*
  - um 825, *Haus der Weisheit* in Bagdad
  - Buch mit Rechenverfahren zur Lösung linearer und quadratischer Gleichungssysteme

- Ein Algorithmus ist eine **Handlungsvorschrift**, die in einer Folge von Einzelschritten beschreibt, wie aus gegebener Information (Eingabe) gesuchte Information (Ausgabe) ermittelt wird.
- Ein Algorithmus löst eine **Klasse** von Problemen
  - unterschiedliche Probleme derselben Klasse sind durch unterschiedliche Eingabedaten gekennzeichnet

Eigenschaften eines Algorithmus

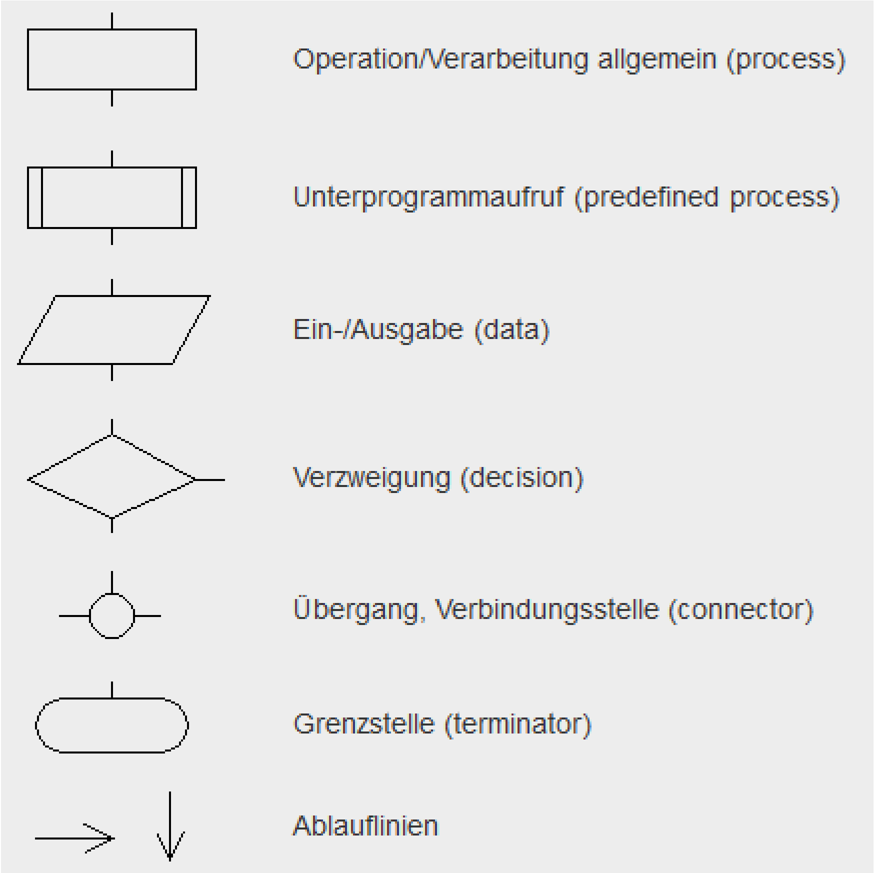
- **Allgemeinheit**
- **Eindeutigkeit**
- **Finitheit**
- **Ausführbarkeit**
- **Determinismus**
- **Determiniertheit**
- **Terminierung**
- **Korrektheit**
- **Effizienz**
- **Portabilität**
- **Wiederverwendbarkeit**
- **Erweiterbarkeit**

Darstellung von Algorithmen

- **Natürliche Sprache**
- **Visuelle Methoden**
  - Flussdiagramm
  - Struktogramm (NASSI-SHNEIDERMAN-Diagramm)
- **Programmiersprache**
- **Pseudocode**
- **Hardwareentwurf**

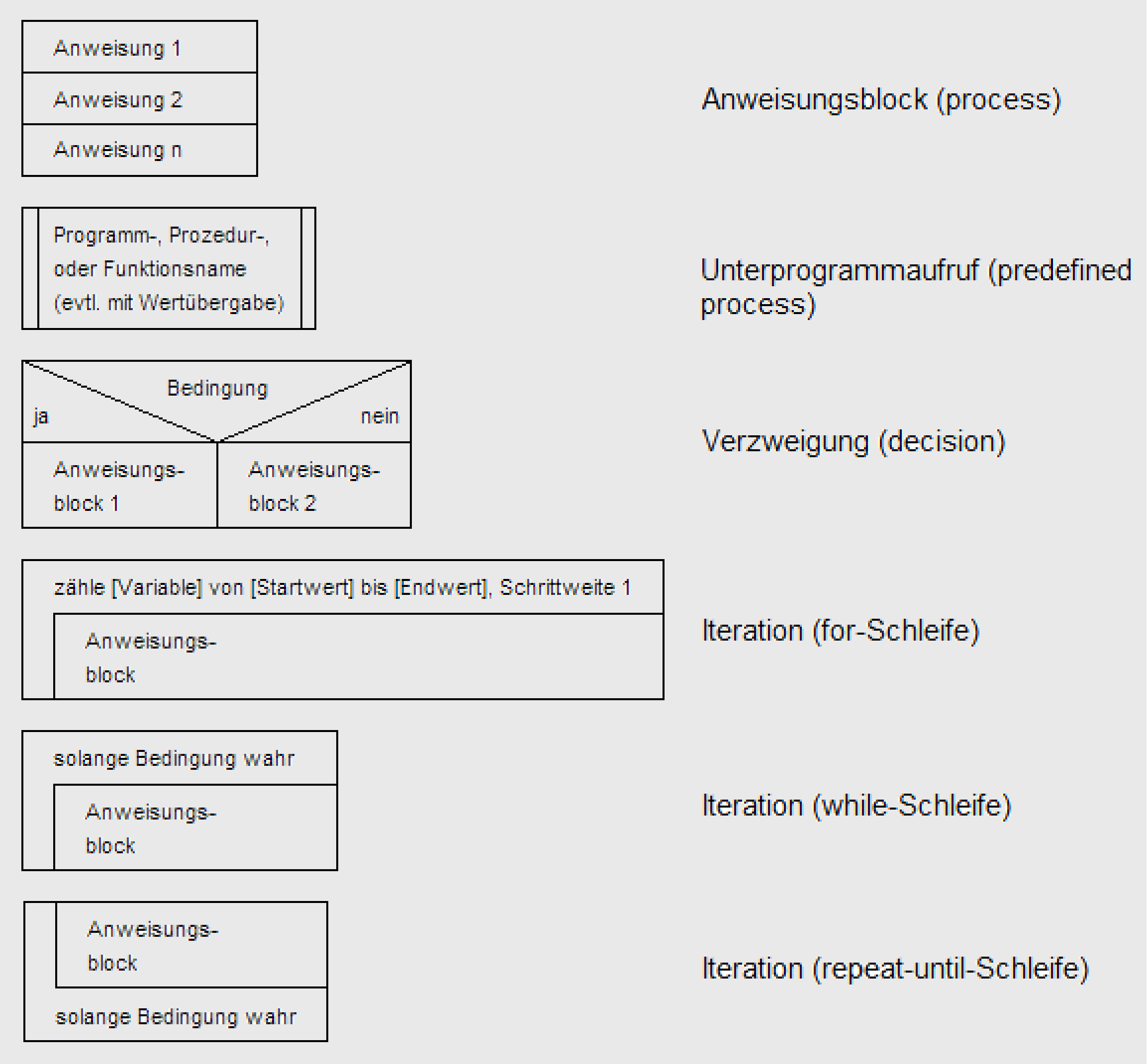
Flussdiagramme

- gehen zurück auf FRANK GILBRETH, 1921, Darstellung von Geschäftsprozessen und Arbeitsabläufen
- **Wesentliche Grafikelemente**
  - DIN 66001



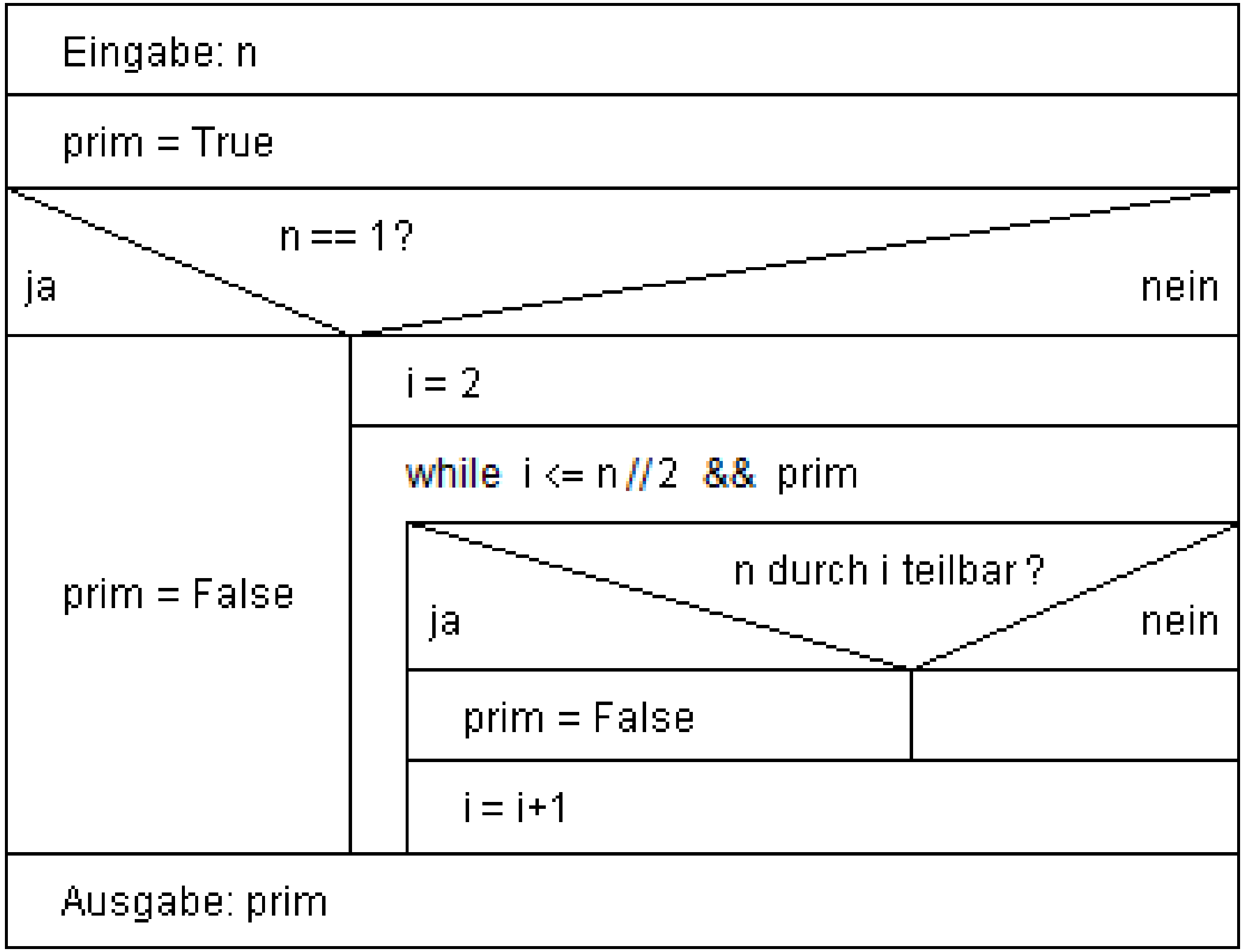
Struktogramme

- entwickelt von Isaac Nassi und Ben Shneiderman, 1972
- **Wesentliche Grafikelemente**
  - DIN 66261



Struktogramme

Beispiel: Primzahltest



Beschreibung durch Pseudocode

- Pseudocode ist ein Mittelding zwischen natürlicher Sprache und formaler Programmiersprache
- Schlüsselwörter, die an echte Programmiersprachen angelehnt sind, werden durch natürlich-sprachliche Formulierungen ergänzt
- Pseudocode ist nicht standardisiert, sondern wird intuitiv verwendet
- *Beispiel:* Berechnung des größten gemeinsamen Teilers zweier gegebener Zahlen *a* und *b* durch den *Euklidischen Algorithmus*

if *b* größer als *a* **then** vertausche beide
repeat
    teile *a* ganzzahlig durch *b*
    if Rest gleich 0 **then** *b* ist das Ergebnis und
        der Algorithmus endet
    else
        verwende *b* als neuen Dividenden
        verwende den Rest als neuen Divisor

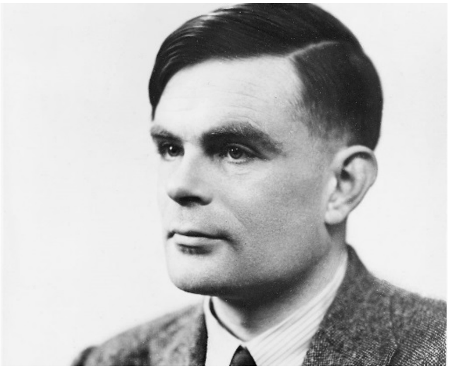
Es ist der ggT von *a*=544 und *b*=391 gesucht

544 : 391 = 1 Rest 153
391 : 153 = 2 Rest 85
153 : 85 = 1 Rest 68
85 : 68 = 1 Rest 17
68 : 17 = 4 Rest 0

Der ggT von 544 und 391 ist 17

Berechenbarkeit

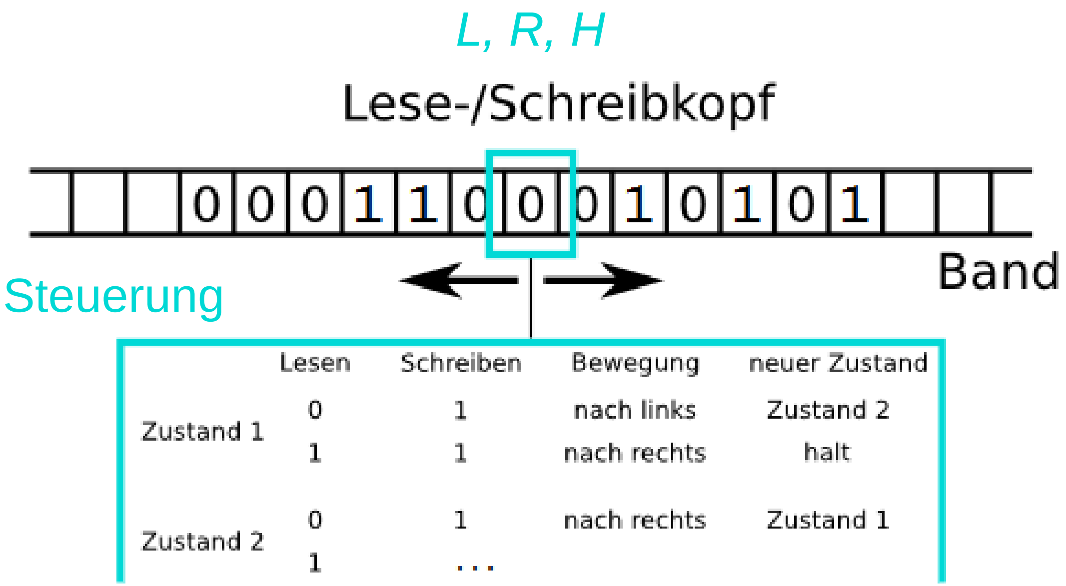
- Ein Problem heißt **berechenbar**, wenn zu seiner Lösung ein Algorithmus formuliert werden kann
- Um den Berechenbarkeitsbegriffs formal zu fassen, wurden (<u>vor</u> der Erfindung des Digitalrechners!) in der ersten Hälfte des 20. Jahrhunderts u.a. abstrakte Maschinenmodelle entwickelt wie die *Registermaschine* oder die ***TURING-Maschine***
- $\Rightarrow$ ***TURING-Berechenbarkeit***
  - Idee: ein Algorithmus ist (TURING-) berechenbar, wenn eine TURING-Maschine existiert, die die mit dem Algorithmus assoziierte Funktion berechnet
  - Der Ansatz wurde 1936 entwickelt vom englischen Mathematiker ALAN M. TURING, 1912-1954



Idee: kariertes Rechenpapier

5	6	7	8	·	4	3	2	1
	2	2	7	1	2			
		1	7	0	3	4		
			1	1	3	5	6	
					5	6	7	8
	2	4	5	3	4	6	3	8

Umsetzung: Band-bearbeitende Maschine



TURING-Maschine
Beispiel: Verdopplungsmaschine

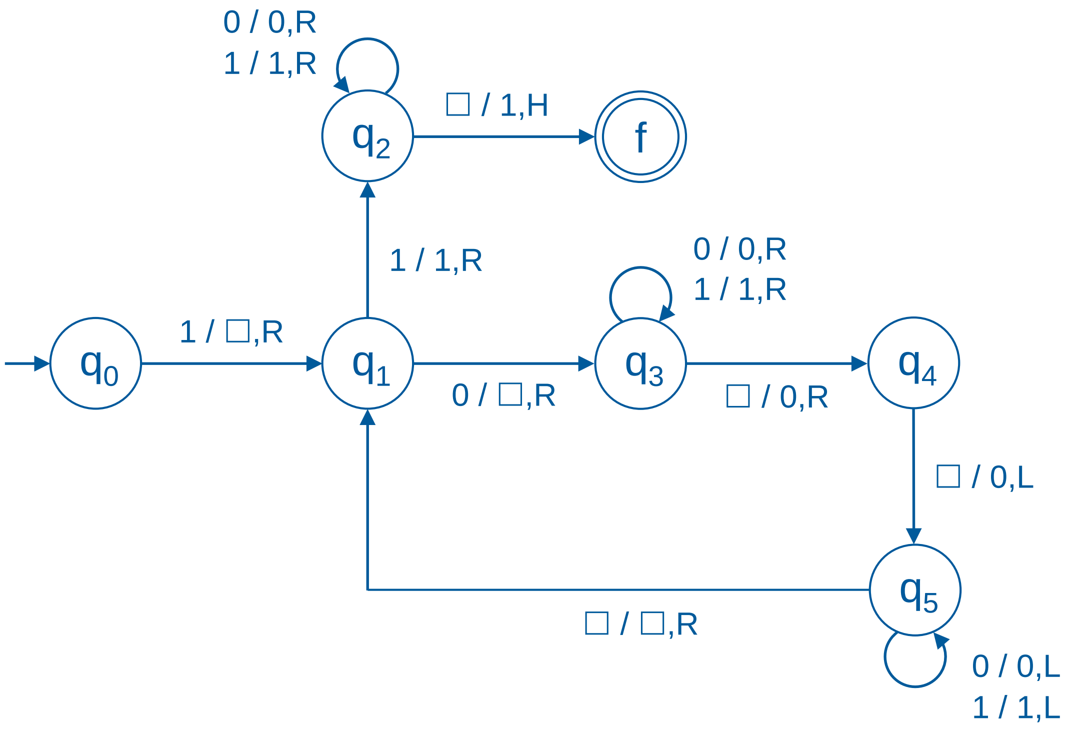
$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, f\}$$

$$\Sigma = \{0, 1\}$$

$$\Gamma = \{0, 1, \square\}$$

$$F = \{f\}$$

δ	0	1	$\square$
q_0	–	$\square, R, q_1$	–
q_1	$\square, R, q_3$	$1, R, q_2$	–
q_2	$0, R, q_2$	$1, R, q_2$	$1, H, f$
q_3	$0, R, q_3$	$1, R, q_3$	$0, R, q_4$
q_4	–	–	$0, L, q_5$
q_5	$0, L, q_5$	$1, L, q_5$	$\square, R, q_1$
f	–	–	–



TURING-Maschine
Beispiel: Inkrementiermaschine

- bei einer Eingabe „29“ ergeben sich folgende Momentaufnahmen

Schritt 1

		q_0			
...	□	2	9	□	...

Schritt 2

			q_0		
...	□	2	9	□	...

Schritt 3

				q_0	
...	□	2	9	□	...

Schritt 4

			q_1		
...	□	2	9	□	...

Schritt 5

		q_1			
...	□	2	0	□	...

Schritt 6

		f			
...	□	3	0	□	...

Die einzelne Momentaufnahme heißt **Konfiguration** der TURING-Maschine.

TURING-Berechenbarkeit

- Wenn ein Algorithmus Eingaben aus einer Menge X akzeptiert und Ausgaben aus einer Menge Y erzeugt, dann berechnet er eine (evtl. partielle) Funktion $f: X \to Y$

- Eine Funktion $f: \Sigma^* \to \Gamma^*$ heißt **TURING-berechenbar**, falls es eine TURING-Maschine TM gibt, so dass für alle $x \in \Sigma^*$ und $y \in \Gamma^*$ gilt:

  $$f(x) = y \qquad \text{genau dann, wenn} \qquad q_0\, x \;\vdash^*\; q_t\, y$$

  wobei $q_t \in F$.

- Mit anderen Worten: f ist TURING-berechenbar, wenn es eine TM gibt, die f **realisiert**,

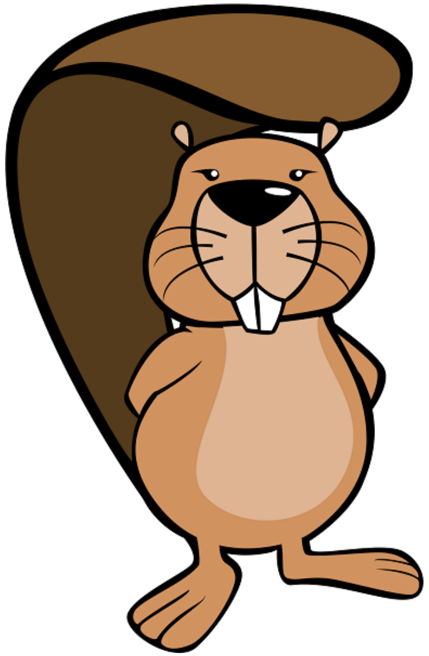
  d.h. bei Eingabe von $x \in \Sigma^*$ eine erfolgreiche Berechnung des Funktionswerts $f(x) \in \Gamma^*$ durchführt und stoppt,

  oder, falls $f(x)$ *undefiniert* ist, auch in eine unendliche Schleife gehen kann.

Fleißige Biber

TIBOR RADÓ, Ohio State University, 1962

- Eine TURING-Maschine die
  - aus genau einem Endzustand und *n* weiteren Zuständen besteht,
  - als nicht-leeres Bandsymbol nur den Strich besitzt ($\Gamma = \{ |, \square \}$),
  - mit dem Schreib-/Lesekopf nur R- und L-Bewegungen durchführt,
  - mit einem leeren Band beginnt und
  - irgendwann im Endzustand anhält,
- heißt **Biber**.
- Schreibt der Biber mit *n* inneren Zuständen die maximale Anzahl von Strichen aufs Band, heißt er **fleißiger Biber** (*busy beaver*).
- Die RADÓ-Funktion **bb(*n*)** nennt die Anzahl der Striche, die ein fleißiger Biber mit *n* inneren Zuständen aufs Band schreibt.



RADÓ-Funktion

n	Anzahl der TM	$\mathbf{bb}(n)$
1	64	1
2	20.736	4
3	16.777.216	6
4	$2,56 \cdot 10^{10}$	13
5	$\approx 6,34 \cdot 10^{13}$	≥ 4098
6	$\approx 2,32 \cdot 10^{17}$	$\geq 1,29 \cdot 10^{865}$
7	$\approx 1,18 \cdot 10^{21}$	?

Eine Funktion $f: I\!N^k \to I\!N$ heißt **WHILE-berechenbar**, falls es ein WHILE-Programm P gibt, das f berechnet, d.h. das mit den Eingabewerten $n_1, n_2, \dots, n_k$ in den Variablen $x_1, x_2, \dots, x_k$ (und dem Wert 0 in allen anderen Variablen) gestartet wird und, falls $f(n_1, n_2, \dots, n_k)$ *definiert* ist, mit diesem Ergebnis in der Variablen x_0 stoppt oder, falls $f(n_1, n_2, \dots, n_k)$ *undefiniert* ist, niemals anhält.

ACKERMANN-Funktion

Wertetabelle

	$n=0$	$n=1$	$n=2$	$n=3$	$n=k$
$A(0,n)$	1	2	3	4	$k+1$
$A(1,n)$	2	3	4	5	$k+2$
$A(2,n)$	3	5	7	9	$2k+3$
$A(3,n)$	5	13	29	61	$2^{k+3}-3$
$A(4,n)$	13	$2^{16}-3$	$2^{65536}-3$	$2^{2^{65536}}-3$	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{k+2 \text{ viele Potenzen}} -3$
$A(5,n)$	$2^{16}-3$	$2^{65536}-3$	$\ldots$		

ACKERMANN-Funktion
Definition

(1) $A(0, n) = n + 1$

(2) $A(m + 1,0) = A(m, 1)$

(3) $A(m + 1, n + 1) \; = \; A(m, A(m + 1, n))$

ACKERMANN-Funktion
Monotonie-Eigenschaften

a. $n < A(m,n)$

b. $A(m,n) < A(m,n+1)$

c. $A(m,n+1) \leq A(m+1,n)$

d. $A(m,n) < A(m+1,n)$

e. $m \leq m', n \leq n' \Rightarrow A(m,n) \leq A(m',n')$ allgemeine Monotonie-Eigenschaft

Fortsetzung des Beweises „es gibt ein k, sodass $f_P(n) < A(k, n)$ für alle n“

- Betrachtung von $P \equiv$ LOOP x_i DO Q END
- $m \leq n$ sei derjenige Variablenwert für x_i, bei dem $f_P(n)$ maximal wird

Fälle:

- $m = 0$: *(Schleife wird gar nicht durchlaufen)*
  - Alle Variablen in P behalten ihren ursprünglichen Wert, d.h. ihre Summe bleibt unverändert:
    Es gilt $f_P(n) \leq n$ und wegen Monotonie-Eigenschaft (a) auch $f_P(n) \leq n < A(k, n)$ für beliebige k.
    Wähle $k = 0$.
- $m = 1$: *(Schleife wird einmal durchlaufen)*
  - Es gilt $f_P(n) = f_Q(n-1) + 1$ da x_i in Q nicht vorkommt.
  - Nach Induktionsvoraussetzung gibt es ein k_1 für das gilt: $f_Q(n-1) < A(k_1, n-1)$.
  - Somit: $f_P(n) < A(k_1, n-1) + 1$
  - bzw. $f_P(n) \leq A(k_1, n-1)$ wegen der Ganzzahligkeit der Werte
  - $< A(k_1, n)$ wegen Monotonie-Eigenschaft (b).
  - Wähle $k = k_1$.

Fortsetzung des Beweises „es gibt ein k, sodass $f_P(n) < A(k, n)$ für alle n"

<u>Fälle:</u>

- $m > 1$: *(Schleife wird mehrmals durchlaufen)*
  - Es gilt $f_P(n) = f_Q(f_Q(\ldots\ (f_Q(n-m)\ldots) + m$ (f_Q m-mal geschachtelt)
  - Nach Induktionsvoraussetzung gibt es ein k_1 für das gilt: $f_Q(\cdot) < A(k_1, \cdot)$.
  - Somit: $f_P(n) < A(k_1, f_Q(f_Q(\ldots\ (f_Q(n-m)\ldots) + m$ (f_Q $m-1$-mal geschachtelt)
  - bzw. $f_P(n) \leq A(k_1, f_Q(f_Q(\ldots\ (f_Q(n-m)\ldots) + m - 1$ (wegen der Ganzzahligkeit der Werte)
  - usw...
  - bis $f_P(n) \leq A(k_1, A(k_1, (\ldots A(k_1, A(k_1, n-m)\ldots)$ (A m-mal geschachtelt)
  - $< A(k_1, A(k_1, (\ldots A(k_1, A(k_1+1, n-m)\ldots)$ (wegen Monotonie-Eigenschaft (d))
  - $= A(k_1, A(k_1, (\ldots A(k_1+1, n-m+1)\ldots)$ (wegen Definition (3), A $m-1$-mal geschachtelt)
  - usw...
  - $= A(k_1+1, n-1)$
  - $< A(k_1+1, n)$ (wegen Monotonie-Eigenschaft (b))
  - Wähle $k = k_1+1$.

Wiederholung Beispiel

$$
\begin{aligned}
A(1,3) &= A(0,A(1,2)) \\
&= A(0,A(0,A(1,1))) \\
&= A(0,A(0,A(0,A(1,0)))) \\
&= A(0,A(0,A(0,A(0,1)))) \\
&= A(0,A(0,A(0,2))) \\
&= A(0,A(0,3)) \\
&= A(0,4) \\
&= 5
\end{aligned}
$$

$$
\begin{array}{lll}
(1) & A(0,\, n) & = n+1 \\
(2) & A(m+1,\, 0) & = A(m,\, 1) \\
(3) & A(m+1,\, n+1) & = A(m,\, A(m+1,\, n))
\end{array}
$$

Abbildung auf Stapelverfahren

			0	1				
		1	1	0	2			
	2	1	0	0	0	3		
3	1	0	0	0	0	0	4	
1	0	0	0	0	0	0	0	5

- Ersetze $0, n$ durch $n+1$ (Stapel wird niedriger)
- Für $m > 0$ ersetze $m, 0$ durch $m-1, 1$
- Für $m, n > 0$ ersetze m, n durch $m-1, m, n-1$ (Stapel wird höher)

- **WHILE-Programm mit Stack-Operationen**

$$
\begin{array}{l}
\mathrm{INIT}(stack);\\
\mathrm{PUSH}(x_1,\ stack);\\
\mathrm{PUSH}(x_2,\ stack);\\
\mathrm{WHILE\ SIZE}(stack) \neq 1 \mathrm{\ DO}\\
\quad \mathrm{POP}(x_2,\ stack);\\
\quad \mathrm{POP}(x_1,\ stack);\\
\quad \mathrm{IF}\ x_1 = 0 \mathrm{\ THEN\ PUSH}(x_2 + 1,\ stack);\\
\qquad \mathrm{ELSIF}\ x_2 = 0 \mathrm{\ THEN\ PUSH}(x_1 - 1,\ stack);\ \mathrm{PUSH}(1,\ stack)\\
\qquad \mathrm{ELSE\ PUSH}(x_1 - 1,\ stack);\ \mathrm{PUSH}(x_1,\ stack);\ \mathrm{PUSH}(x_2 - 1,\ stack)\\
\quad \mathrm{END};\\
\mathrm{END};\\
\mathrm{POP}(x_0,\ stack)
\end{array}
$$

CHURCH-TURING-These

- **Gleichmächtige Berechnungsmodelle**
  - TURING-Maschinen
  - WHILE- und GOTO-Programme
  - λ-Kalkül, CHURCH, 1936
  - μ-rekursive Funktionen
  - MARKOV-Algorithmen, 1960
  - Registermaschinen (Random Access Machines), SHEPHERDSON & STURGIS, 1963
  - ...

- *These:*

  Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der im intuitiven Sinne berechenbaren Funktionen überein.

- **Entscheidungsproblem**
  - Frage, ob ein beliebiges Element x aus einer Grundmenge M eine bestimmte Eigenschaft P hat
  - Antwort: „Ja“ oder „Nein“

- **Sprache des Entscheidungsproblems**
  - $L_P = \{ x \mid x \in M \text{ und } x \text{ hat die Eigenschaft } P \}$

- **Definition der Entscheidbarkeit**
  - Eine Sprache (bzw. Menge) $L_P \subseteq M$ heißt **entscheidbar**, wenn es einen Algorithmus gibt, der zu jedem $x \in M$ nach endlich vielen Schritten die Antwort „Ja“ oder „Nein“ auf die Frage liefert, ob $x \in L_P$ ist.

  alternativ:
  - ..., wenn die charakteristische Funktion $\chi_{L_P} : M \to \{0, 1\}$ berechenbar ist.

Es ist nicht entscheidbar, ob eine gegebene TURING-Maschine für eine gegebene Eingabe anhält.

- **Beweis:**
  - Codiere eine TM als Wort über {0, 1}*
  - Definiere „Diagonalsprache“ *D*
    - $D = \{ \langle M \rangle \mid M$ ist eine TM, die bei Eingabe von $\langle M \rangle$ anhält $\}$
    - *D* ist entscheidbar, falls das Halteproblem entscheidbar wäre
    - Dann wäre χ_D durch eine TM M_D berechenbar
  - Modifiziere M_D zu M'_D gemäß:

$$\text{start} \longrightarrow M_D \longrightarrow \text{„Band} = 0\,?\text{“} \xrightarrow{ja} \text{stop}$$

$$\text{„Band} = 0\,?\text{“} \xrightarrow{nein} \text{„Band} = 0\,?\text{“}$$

	0	1	2	3	4	5	6	
S_0	0	1	1	0	1	0	1	...
S_1	1	1	1	0	1	0	1	...
S_2	0	0	1	0	1	0	1	...
S_3	0	1	1	0	0	0	1	...
S_4	0	1	0	0	1	0	1	...
S_5	0	1	1	0	1	0	0	...
S_6	1	1	1	0	1	0	1	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Halteproblem

Widerspruchsbeweis fortgesetzt...

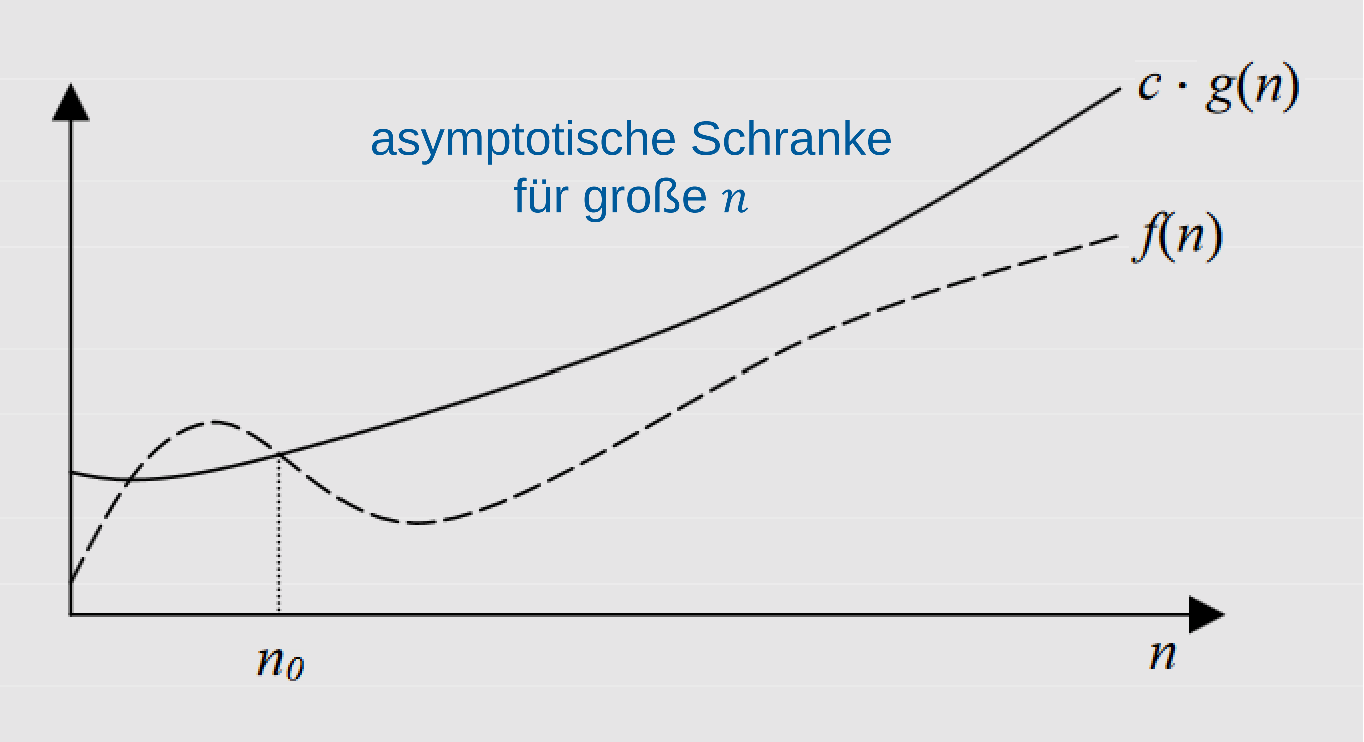
- Also: M'_D stoppt genau dann, wenn M_D den Wert 0 ausgeben würde. Falls M_D den Wert 1 ausgibt, geht M'_D in eine Endlosschleife.

- **nun starte M'_D mit Eingabe $\langle M'_D \rangle$**
- **Falls M'_D mit dieser Eingabe <u>anhält</u>:** $M'_D \in D$
  - M_D gibt bei dieser Eingabe 0 aus (vgl. Def. von M'_D),
  - d.h. $\chi_D(\langle M'_D \rangle) = 0$,
  - d.h. $M'_D \notin D$. Widerspruch!
- **Falls M'_D mit dieser Eingabe <u>nicht</u> anhält:** $M'_D \notin D$
  - M_D gibt bei dieser Eingabe 1 aus (vgl. Def. von M'_D),
  - d.h. $\chi_D(\langle M'_D \rangle) = 1$,
  - d.h. $M'_D \in D$. Widerspruch!
- Also ist die Annahme falsch: **das Halteproblem ist nicht entscheidbar!**

$\mathcal{O}$-Notation

- Eine Funktion $f(n)$ wächst *mit der Ordnung* $O(g(n))$, wenn eine positive Konstante c existiert, so dass

$$|f(n)| \leq c \cdot |g(n)| \text{ für alle } n > n_0$$

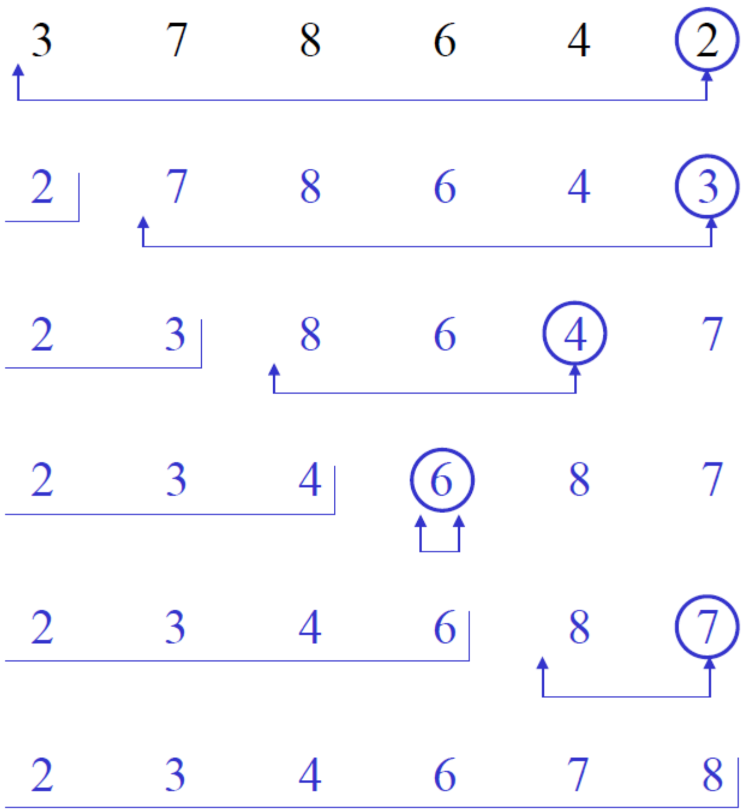


Sortierproblem

Algorithmus „SelectionSort“ (auch „MinSort“)

- **Prinzip:**
  - Suche kleinstes Element
  - Vertausche es mit dem Element an der ersten Stelle
  - Wende denselben Algorithmus auf die restlichen n-1 Elemente an

Beispiel:



Implementierung

die zu sortierenden Elemente befinden sich in der Liste **`M[0..n-1]`**

```
i = 0
n = len(M)
while i < n:
   min = i
   j = i + 1
   while j < n:
      if M[j] < M[min]:
         min = j
      j = j + 1
   M = swap(M,i,min)
   i = i+1
```

Komplexitätsanalyse

- zum Sortieren der gesamten Folge werden $n-1$ Durchläufe benötigt
- im i-ten Durchlauf werden $n-i$ Vergleiche und eine Vertauschung durchgeführt
- in Summe sind das
  $(n-1)+(n-2)+\ \ldots\ +2+1\ =\frac{(n-1)\cdot n}{2}=$
  $\frac{n^2}{2}-\frac{n}{2}$ Vergleiche
  und $n-1$ Vertauschungen
- die Komplexitätsklasse von SelectionSort ist somit $O(n^2)$

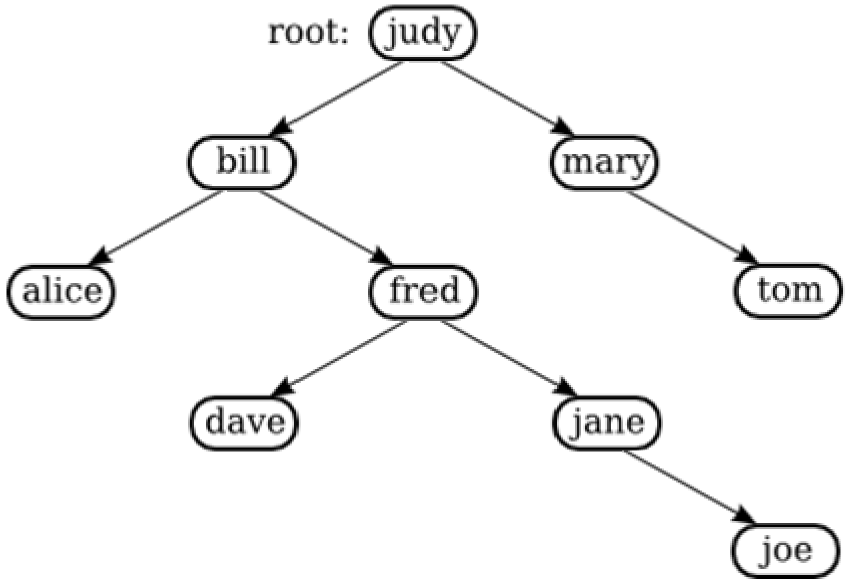
Sortierproblem
Algorithmus „TreeSort“

- **Prinzip:**
  - Die Elementmenge wird in einen binären Baum umsortiert
  - Der Baum ist mit jeder Einfügung automatisch sortiert:

    Ein *in-order-Durchlauf* durch einen binären Suchbaum ist äquivalent zum Durchlauf durch eine sortierte Liste (bei im Wesentlichen gleichem Laufzeitverhalten)

    in-order-Durchlauf: linker Teilbaum – Wurzel – rechter Teilbaum

Beispiel: Liste = [judy, mary, bill, fred, jane, tom, alice, joe, dave]



Implementierung

ein **tree** wird dargestellt als Liste

```
[<left_subtree>,root,<right_subtree>]
```

wobei beide Teilbäume vom Typ **tree** sind

```
def insert(t, item):
  if len(t) == 0:
     newt = [[],item,[]]
  else:
     if item < t[1]: newt =
            [insert(t[0],item),t[1],t[2]]
     else: newt =
            [t[0],t[1],insert(t[2],item)]
  return newt
```

Komplexitätsanalyse

- zum Sortieren der gesamten Folge werden n Durchläufe benötigt
- pro Durchlauf wird ein Element in den Baum eingefügt
- das Einfügen erfordert einen Suchaufwand, der höchstens der maximalen Höhe des Baums entspricht
  - bei einem balancierten Baum: $\mathrm{ld}(n)$
  - bei einem nicht-balancierten Baum: n
- die Komplexitätsklasse von TreeSort ist somit bestenfalls $O(n \cdot \log(n))$ bzw. schlechtestenfalls $O(n^2)$

$\mathrm{ld}(x) = \mathrm{ld}(10) \cdot \log(x)$

Komplexität

Beispiele für Laufzeiten von Algorithmen unterschiedlicher Komplexität

complexity	n=10	n=20	n=50	n=100	n=1000	n=10,000	n=100,000
$O(\log(n))$							1 ns
$O(\mathrm{n})$							6 µs
$O(n \cdot \log(n))$						8 µs	0.1 ms
$O(n^2)$					60 µs	6 ms	0.6 s
$O(2^n)$	600 µs	0.6 s	18.7 h	hangs	hangs	hangs	hangs
$O(n!)$	22 ms	111 y	hangs	hangs	hangs	hangs	hangs

ermittelt auf einem 16.700 Dhrystone-MIPS-Rechner (i5-4690 Quadcore, 3,8 GHz)

Travelling Salesman Problem

NP-vollständig

- Ein Handlungsreisender soll auf einer Rundreise n vorgegebene Stationen besuchen und schließlich zu seinem Ausgangspunkt zurückkehren. Die Entfernungen (Kosten) zwischen allen Paaren von Stationen sind gegeben. Die Gesamtlänge der Rundreise soll minimal sein.
- Alle bisher vorgeschlagenen Algorithmen laufen im Prinzip auf dasselbe Schema hinaus:
  - Bilde alle Permutationen der n Stationen $\in O(n!)$
  - Ignoriere diejenigen, die keine Rundreise darstellen
  - Von den verbliebenen Permutationen wähle diejenige mit minimalen Kosten

Städte	mögliche Rundreisen	Laufzeit	
3	1	1	msec
4	3	3	msec
5	12	6	msec
6	60	60	msec
7	360	360	msec
8	2.520	2,5	sec
9	20.160	20	sec
10	181.440	3	min
11	1.814.400	0,5	Stunden
12	19.958.400	5,5	Stunden
13	239.500.800	2,8	Tage
14	3.113.510.400	36	Tage
15	43.589.145.600	1,3	Jahre
16	653.837.184.000	**20**	**Jahre**

Rechner

Begriffserläuterung

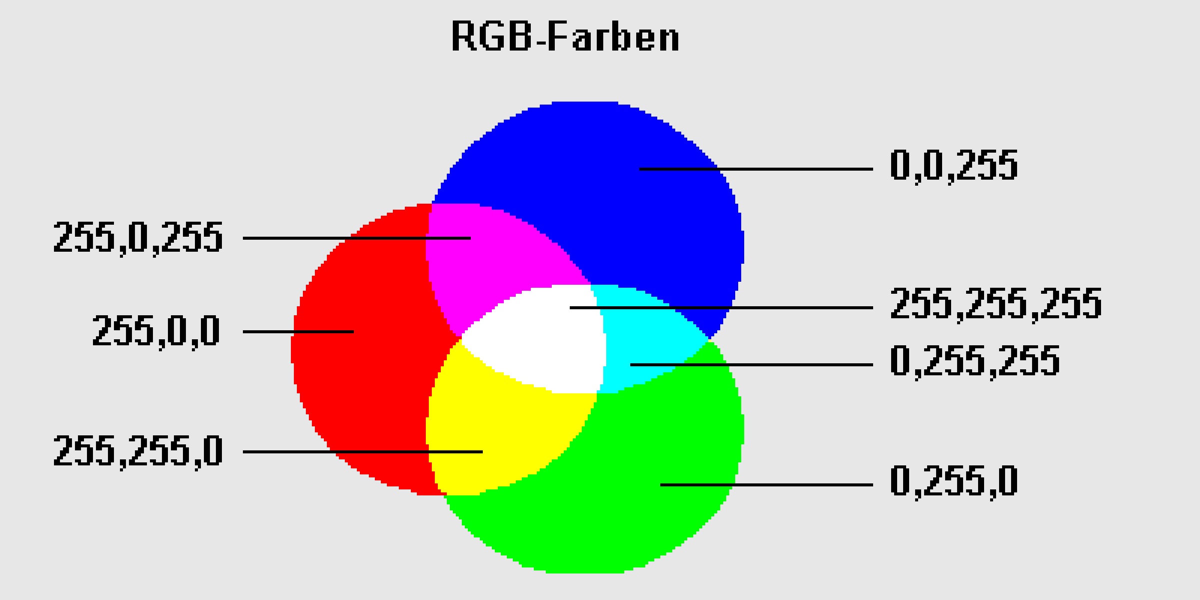
- Rechner = **programmgesteuertes Informationsverarbeitungs**system
- Informationsverarbeitung: das Erfassen, Eingeben, Sortieren, Filtern, Strukturieren, Konvertieren, Manipulieren, Verknüpfen, Speichern, Archivieren, Übertragen, Ausgeben und Löschen von Information
- Information $\underset{\text{Interpretation}}{\overset{\text{Repräsentation}}{\rightleftarrows}}$ Daten
- Informationsarten = Wahrheitswert, Zahlen, Text, Bild, Audio, Video, Befehle, Adressen, ...
- Programm = Verarbeitungsvorschrift (*Algorithmus*), Folge von Befehlen
- programmierbar = Programm ist (austauschbar) gespeichert

  $\Rightarrow$ **Universalrechner**

ASCII Zeichensatz

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@, §	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[, Ä	k	{, ä
1100	FF	FS	,	<	L	\, Ö	l	|, ö
1101	CR	GS	-	=	M	], Ü	m	}, ü
1110	SO	RS	.	>	N	^	n	~, ß
1111	SI	US	/	?	O	_	o	DEL

RGB-Farbcodierung

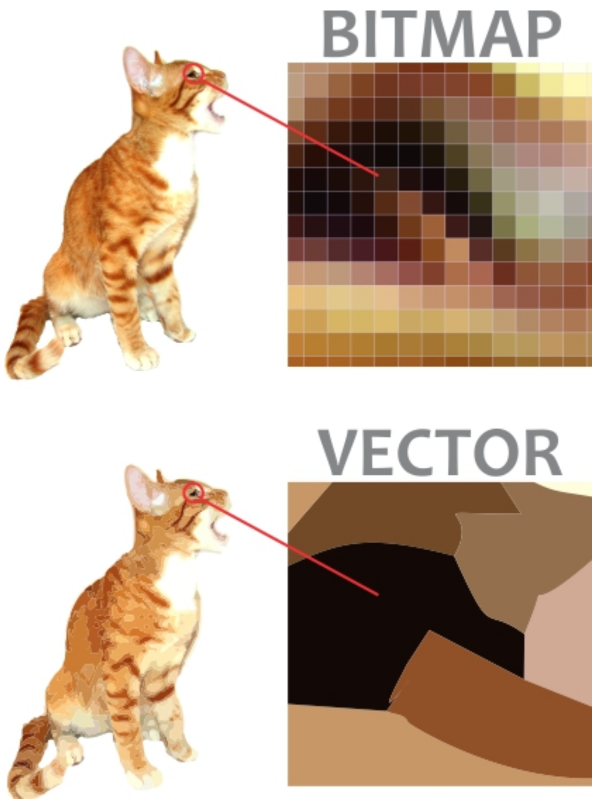


	R	G	B
schwarz	0	0	0
rot	255	0	0
grün	0	255	0
blau	0	0	255
cyan	0	255	255
magenta	255	0	255
gelb	255	255	0
weiß	255	255	255

True Color: 256 Intensitätstufen je Farbanteil

Vektorgrafik

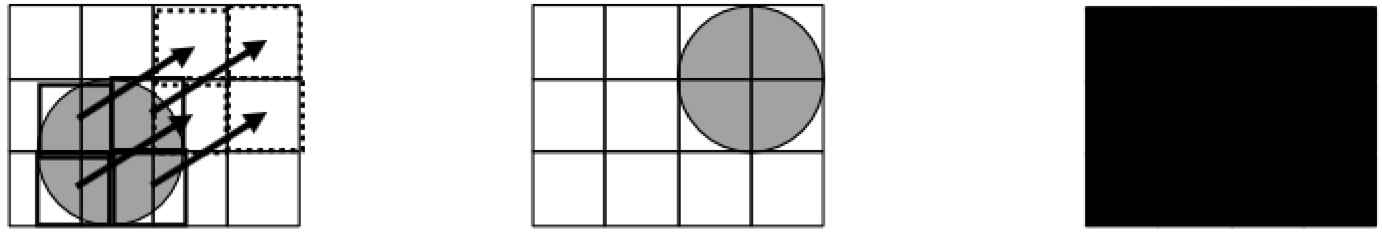
- **für geografische Karten, CAD-Zeichnungen oder virtuelle 3D-Bilder**
  - die Umrisse grafischer Objekte werden dargestellt durch Primitive wie Linien, Linienzüge, Bézierkurven, Polygone, Ellipsen, usw.
  - Farben, Farbverläufe, Schraffuren usw. werden als Attribute zugeordnet
  - Primitive werden üblicherweise als Text in einer Markup-Sprache gespeichert



Video (Bewegtbilder)

- **Bewegungseindruck entsteht durch Betrachten von Bildfolgen**
  - untere Schwelle fürs menschliche Auge: 16 – 18 Bilder/Sekunde (Hz)
  - Kino, Fernsehen: 24 – 48 Hz
  - Monitore: 60 Hz, für „weiche“ Bewegungsabläufe beim Gaming auch 120, 144 oder 240 Hz

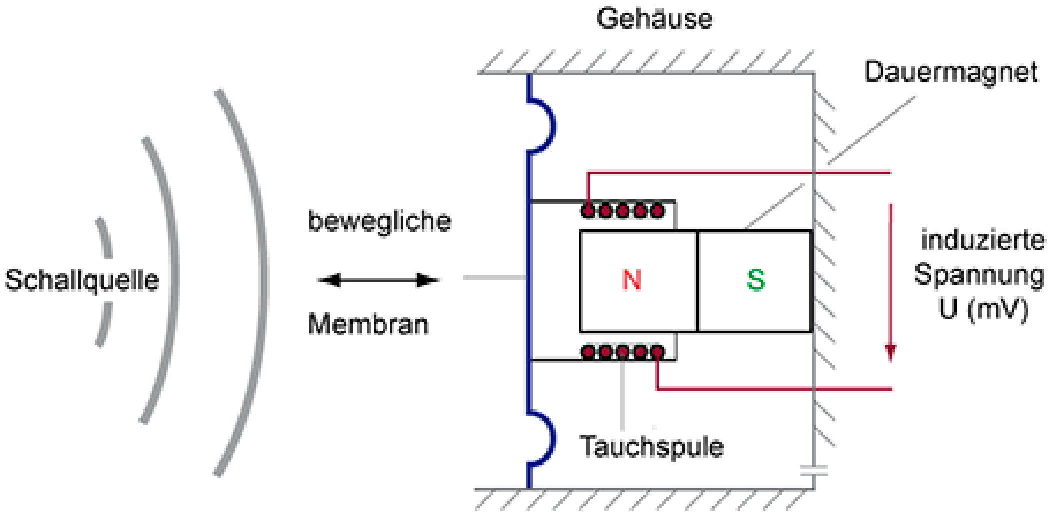
- **Kompressionsansätze**
  - Differenzcodierung (Unterschiede aufeinanderfolgender Bilder)
  - plus Verschiebungsvektor

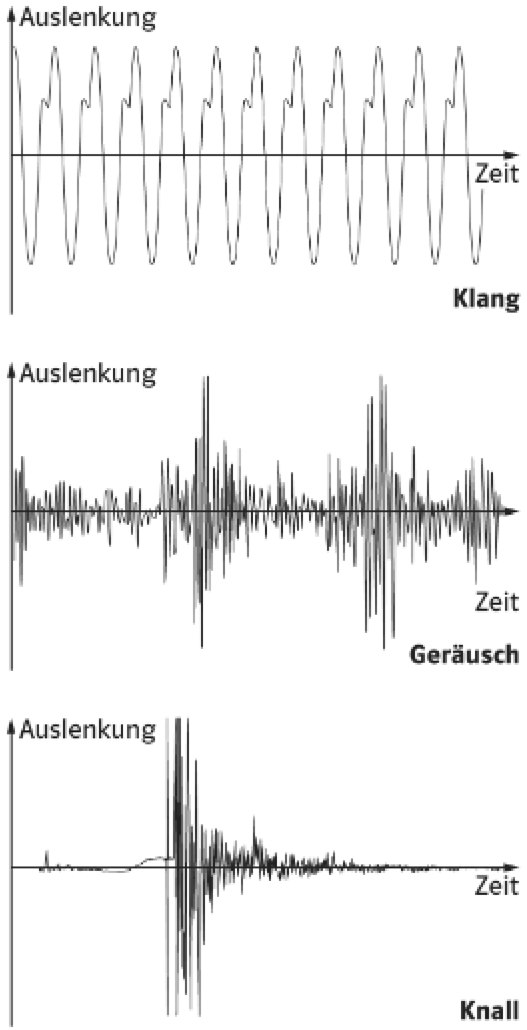


Referenzframe *N* — Zielframe *N*+1 — Differenzframe

Audiosignale

- Ton wird durch Luftdruckänderungen im Raum transportiert, die sich als Longitudinalwelle ausbreiten
- ein Mikrofon konvertiert Ton in ein (analoges) elektrisches Signal, ein Lautsprecher umgekehrt





Negative (ganze) Zahlen
Zweierkomplement

- Negative Zahlen sollten so codiert werden, dass die übliche Addition von Binärzahlen zum richtigen Ergebnis führt ($\rightarrow$ Zurückführung der Subtraktion auf die Addition)

- **Beispiel:** Was wäre eine geeignete Binärdarstellung für -10_{10}?

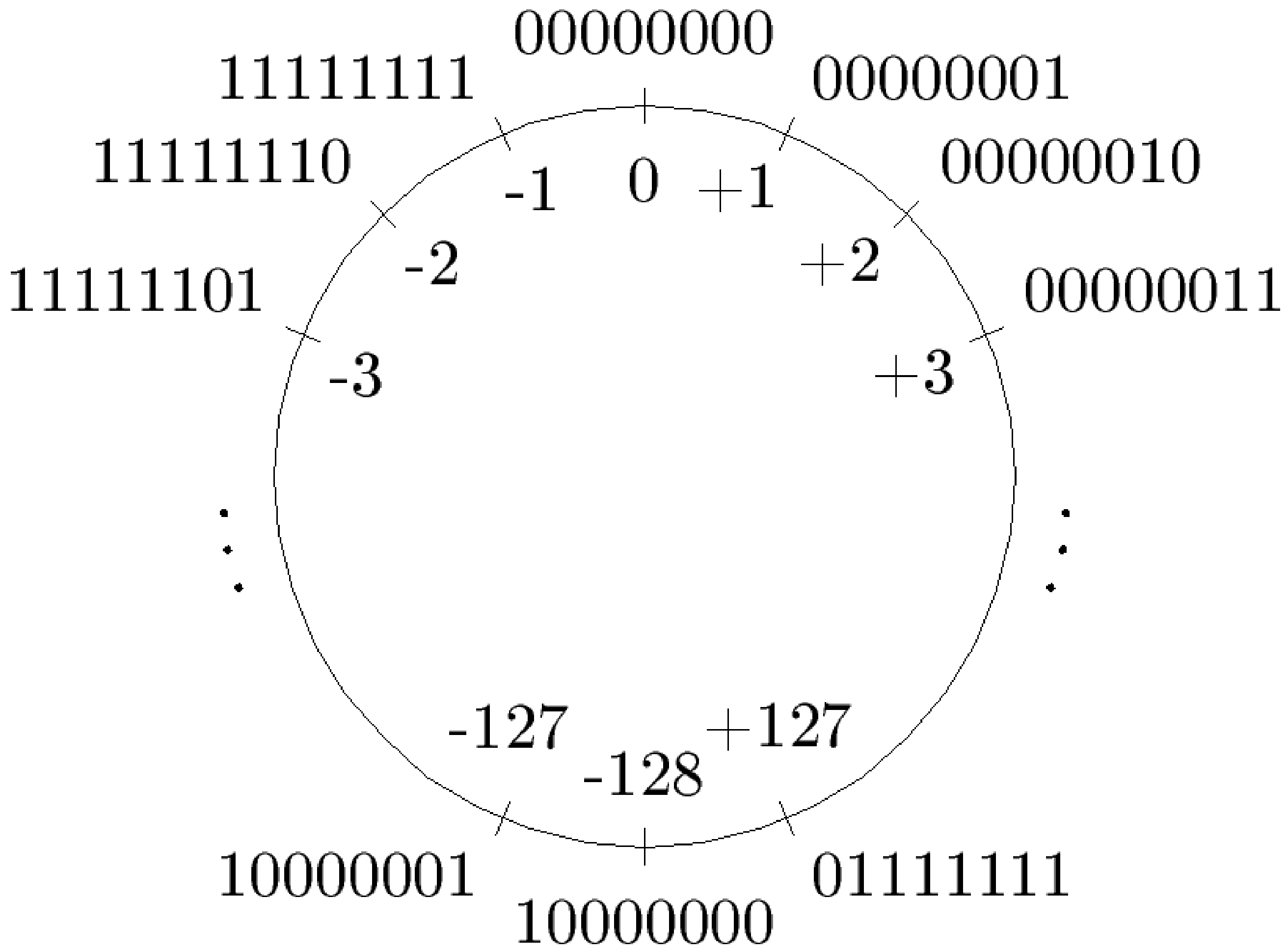
	0 0 0 0 1 0 1 0
+	? ? ? ? ? ? ? ?
=	0 0 0 0 0 0 0 0

	0 0 0 0 1 0 1 0
+	1 1 1 1 0 1 0 1
+	1
=	1 0 0 0 0 0 0 0 0

Antwort: Invertiere jede Stelle und addiere 1
(ignoriere den Übertrag in die N-te Stelle)

- Diese Darstellungsform negativer ganzer Zahlen heißt **Zweierkomplement**-Darstellung
  - die höchstwertige Stelle zeigt an, ob die Zahl positiv (0) oder negativ (1) ist
  - bei einer N-stelligen Darstellungsbreite ist der gültige Zahlenbereich das Intervall $[-2^{N-1}, +2^{N-1}-1]$

Zahlenring modulo 2^8 im Zweierkomplement



Formate nach IEEE 754-2008

Typ	Gesamtanzahl Stellen	Stellen c	Stellen f
half precision	16	5	10
single precision	32	8	23
double precision	64	11	52
quadruple precision	128	15	112

Präfixe für Dateneinheiten

- die gewohnten SI-Präfixe sind **Dezimalpräfixe**
- sie wurden jedoch häufig an Zweierpotenzen angepasst interpretiert (insbesondere mit der Maßeinheit Byte)
- um Mehrdeutigkeit zu vermeiden: **Binärpräfixe** (IEC 60027-2, 1998), ihre Akzeptanz ist allerdings gering

SI-Präfix	Zehner-potenz	an Binärzahlen angepasste Interpretation	Zweier-potenz	**Binärpräfix**
k (Kilo)	10^3	1.024	2^{10}	Ki (Kibi)
M (Mega)	10^6	1.048.576	2^{20}	Mi (Mebi)
G (Giga)	10^9	1.073.741.824	2^{30}	Gi (Gibi)
T (Tera)	10^{12}	1.099.511.627.776	2^{40}	Ti (Tebi)
P (Peta)	10^{15}	1.125.899.906.842.624	2^{50}	Pi (Pebi)
E (Exa)	10^{18}	1.152.921.504.606.846.976	2^{60}	Ei (Exbi)

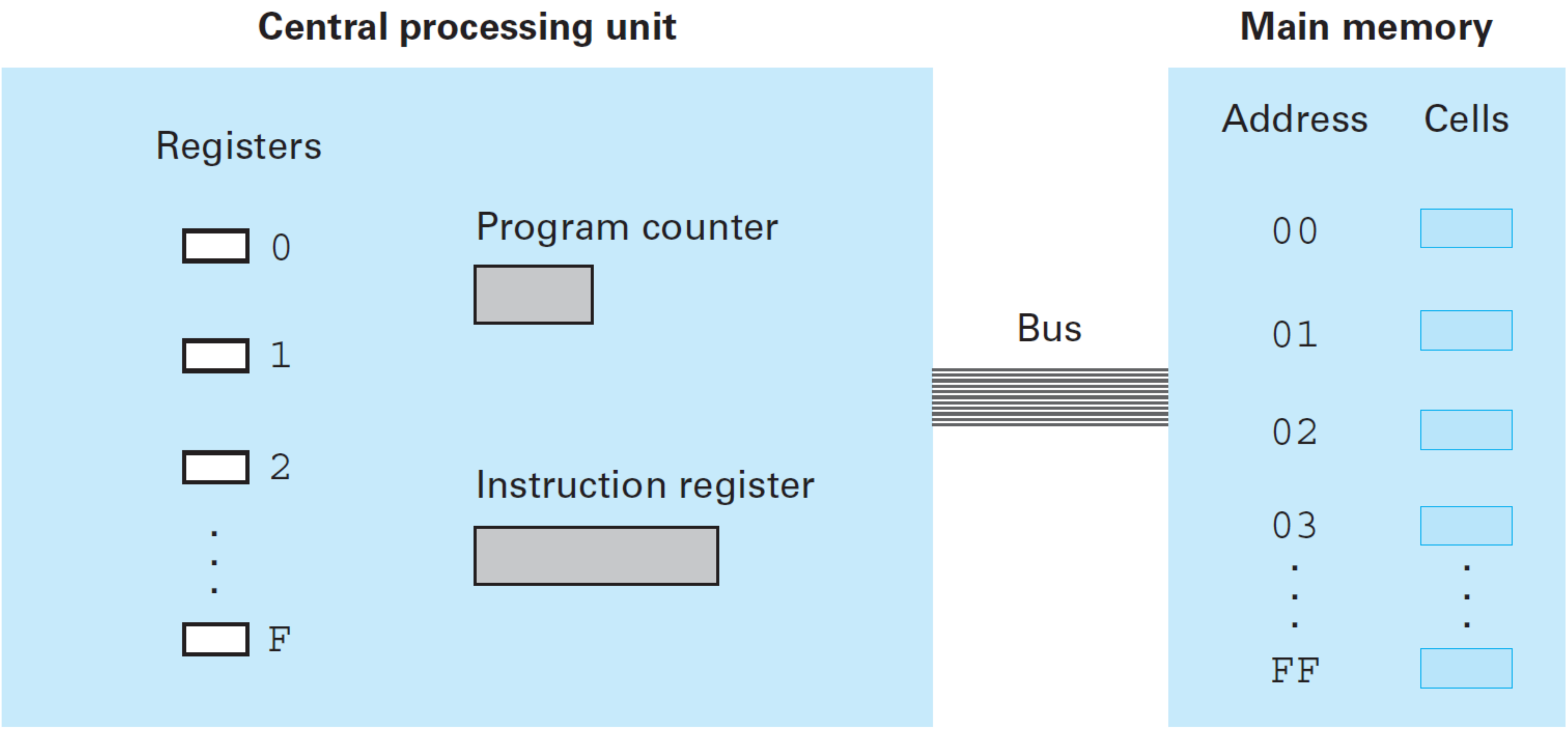
Befehlssatz

Befehlsgruppen

- **arithmetische Befehle**: Addition, Subtraktion, Multiplikation, Division, ...
- **logische Befehle**: Und-, Oder-, XOR-Verknüpfung, Negation, ...
- **Transportbefehle**: Laden bzw. Verschieben von Werten in Register oder in den Speicher
- **Schiebe- und Rotationsbefehle** (bezogen auf Registerinhalte)
- **Befehle zur Programmablaufsteuerung**: Test- und Vergleichsbefehle, Sprungbefehle, Unterprogrammaufruf und -rücksprung, ...
- **Systembefehle**: Ein-/Ausgabebefehle für Peripheriegeräte, Befehle die den Zustand des Rechners in besonderer Weise verändern wie HLT und SYSCALL (Zugriff auf privilegierte Funktionen des Betriebssystems)

Rechnermodell für eine einfache hypothetische Maschinensprache

J. Glenn Brookshear, Computer Science



Befehlssatz einer einfachen hypothetischen Maschinensprache

J. Glenn Brookshear, Computer Science, Appendix C

Opcode	Operanden	Beschreibung
1	RXY	Lade Speicherwort aus Adresse XY in Register R
2	RXY	Lade Wert XY in Register R
3	RXY	Speichere Inhalt von Register R in Speicheradresse XY
4	0RS	Verschiebe Inhalt von Register R in Register S
5	RST	Addiere Inhalte der Register S und T und lege Ergebnis in R ab (alle Werte im Zweierkomplement)
6	RST	Addiere Inhalte der Register S und T und lege Ergebnis in R ab (alle Werte im an IEEE 754 angelehnten Gleitkommaformat 1+3+5 bit)
7	RST	OR-verknüpfe Inhalte der Register S und T und lege Ergebnis in R ab
8	RST	AND-verknüpfe Inhalte der Register S und T und lege Ergebnis in R ab
9	RST	XOR-verknüpfe Inhalte der Register S und T und lege Ergebnis in R ab
A	R0X	Rotiere den Inhalt von Register R um X Stellen nach rechts
B	RXY	Springe zum Befehl in Speicheradresse XY wenn der Inhalt von Register R gleich dem Inhalt von Register R0 ist
C	000	Halte die weitere Befehlsausführung an

Befehlssatz einer einfachen hypothetischen Maschinensprache

Assembler-Notation

Opcode	Operanden	Assemblernotation
1	RXY	LOAD R, XY
2	RXY	LOADI R, XY
3	RXY	STORE XY, R
4	0RS	MOVE S, R
5	RST	ADD R, S, T
6	RST	ADD-FLOAT R, S, T
7	RST	OR R, S ,T
8	RST	AND R, S, T
9	RST	XOR R, S, T
A	R0X	ROTATE-RIGHT R, X
B	RXY	JUMP XY, R
C	000	HALT

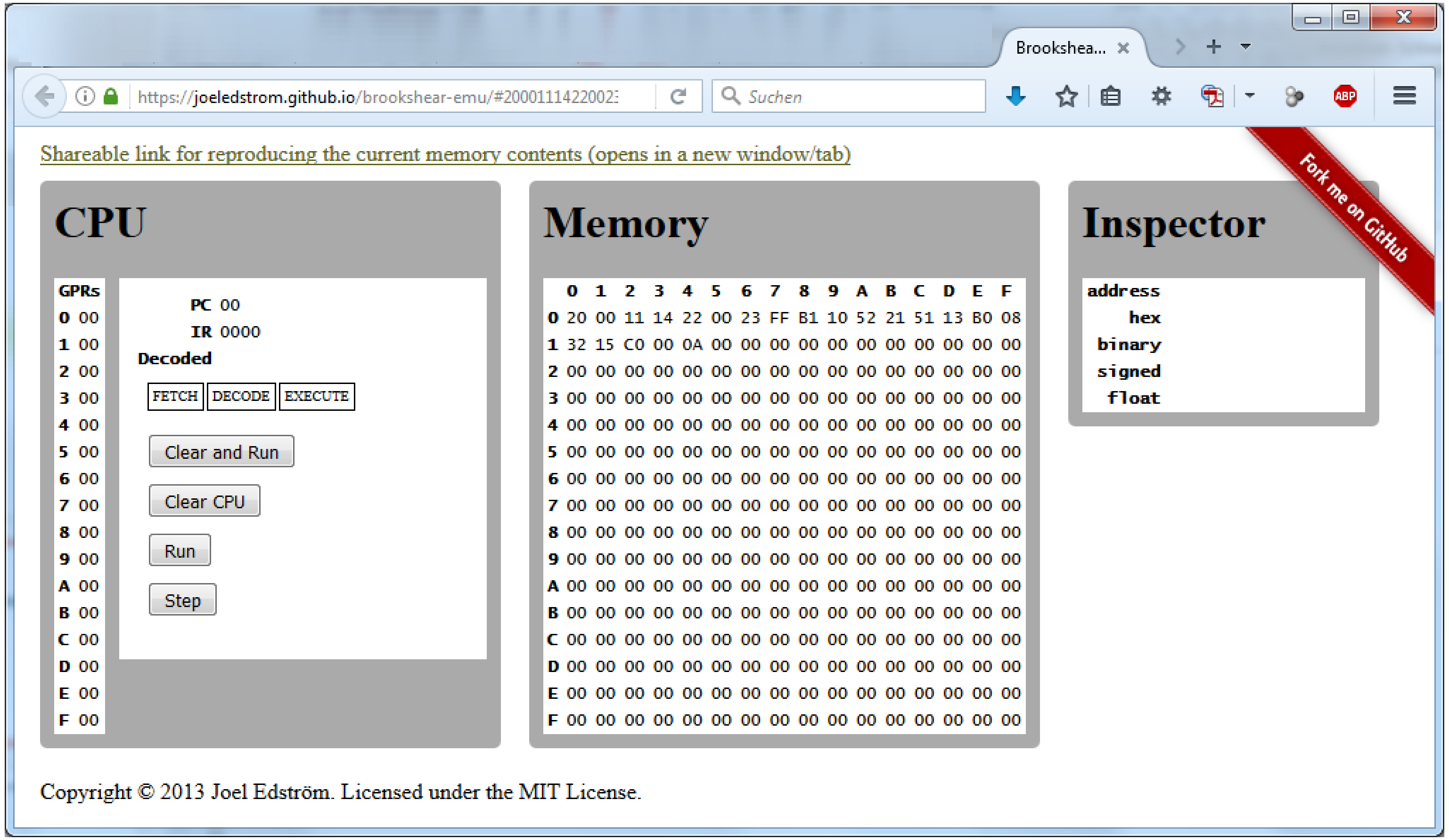
Beispielprogramm: GAUßsche Summe

```
        ; Initialisierung
00:     LOADI 0,00
02:     LOAD  1,14
04:     LOADI 2,00
06:     LOADI 3,FF

        ; Laufschleife
08:     JUMP  10,1
0A:     ADD   2,2,1
0C:     ADD   1,1,3
0E:     JUMP  08,0

        ; Abschluss
10:     STORE 15,2
12:     HALT
```

0	20
1	00
2	11
3	14
4	22
5	00
6	23
7	FF
8	B1
9	10
A	52
B	21
C	51
D	13
E	B0
F	08
10	32
11	15
12	C0
13	00
14	n
15	Ergebnis

Simulator

- https://joeledstrom.github.io/brookshear-emu/#20001114220023FFB11052215113B0083215C0000A



Gesetze der Schaltalgebra

- Axiome

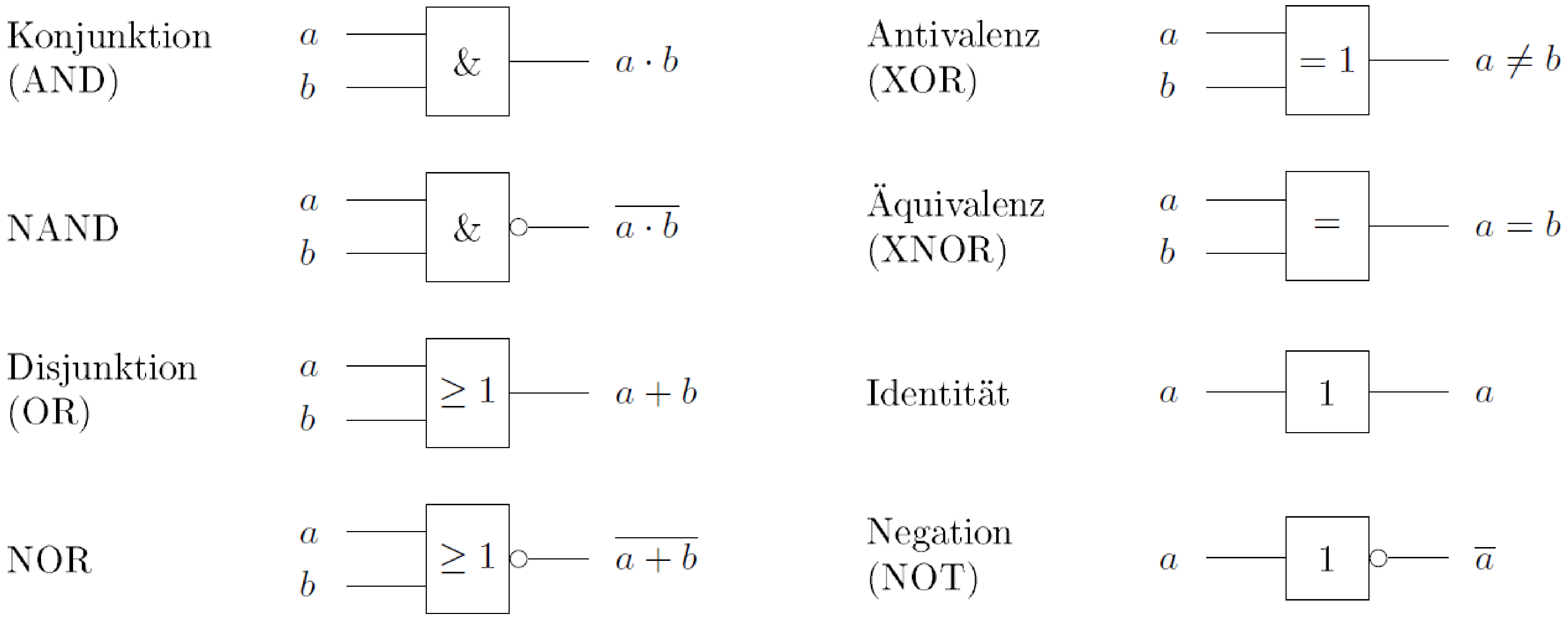
$a \cdot b = b \cdot a$	$a + b = b + a$	Kommutativität
$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$	$a + (b \cdot c) = (a + b) \cdot (a + c)$	Distributivität
$a \cdot 1 = a$	$a + 0 = a$	Identität
$a \cdot \bar{a} = 0$	$a + \bar{a} = 1$	Komplementierung

- Theoreme

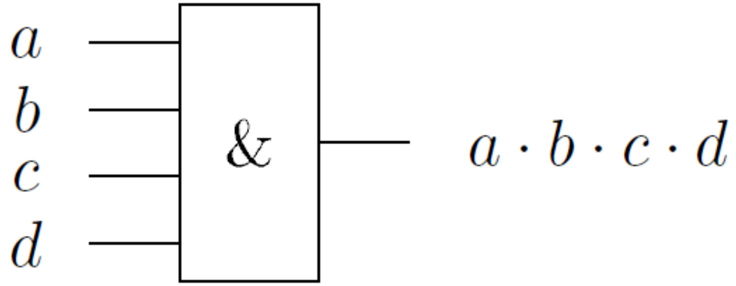
$a \cdot (b \cdot c) = (a \cdot b) \cdot c$	$a + (b + c) = (a + b) + c$	Assoziativität
$a \cdot a = a$	$a + a = a$	Idempotenz
$a + (a \cdot b) = a$	$a \cdot (a + b) = a$	Absorption
$\overline{a + b} = \bar{a} \cdot \bar{b}$	$\overline{a \cdot b} = \bar{a} + \bar{b}$	DeMorgan
$\bar{\bar{a}} = a$		Involution

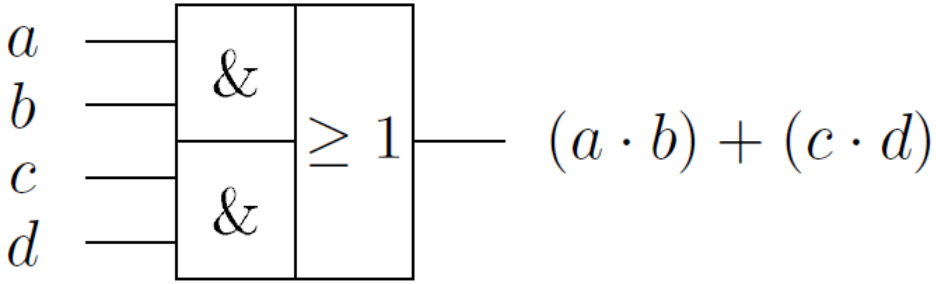
Man beachte die **Dualität**:
Tausch von · /+ und 0/1 ergibt jeweils das duale Gesetz

Verknüpfungsglieder
Schaltsymbole



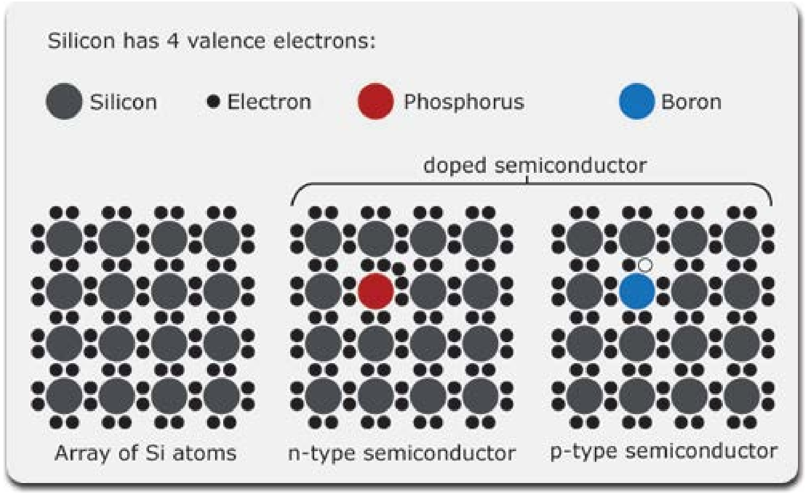
- Verknüpfungsglieder mit mehr als zwei Eingängen





Halbleiter
n- und p-Dotierung

- **Kristallgitter mit Störatomen**

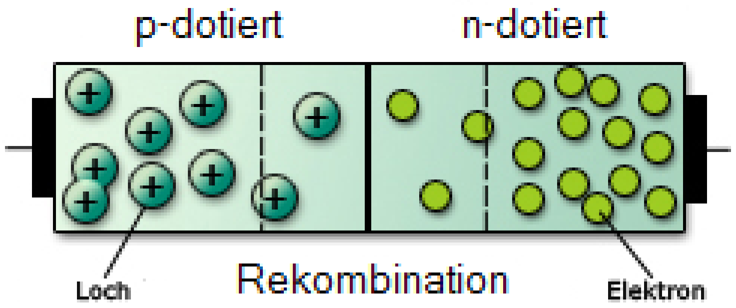


Silizium (Si)
Germanium (Ge)
Galliumarsenid (GaAs)

Phosphor (P)
Arsen (As)
Antimon (Sb)

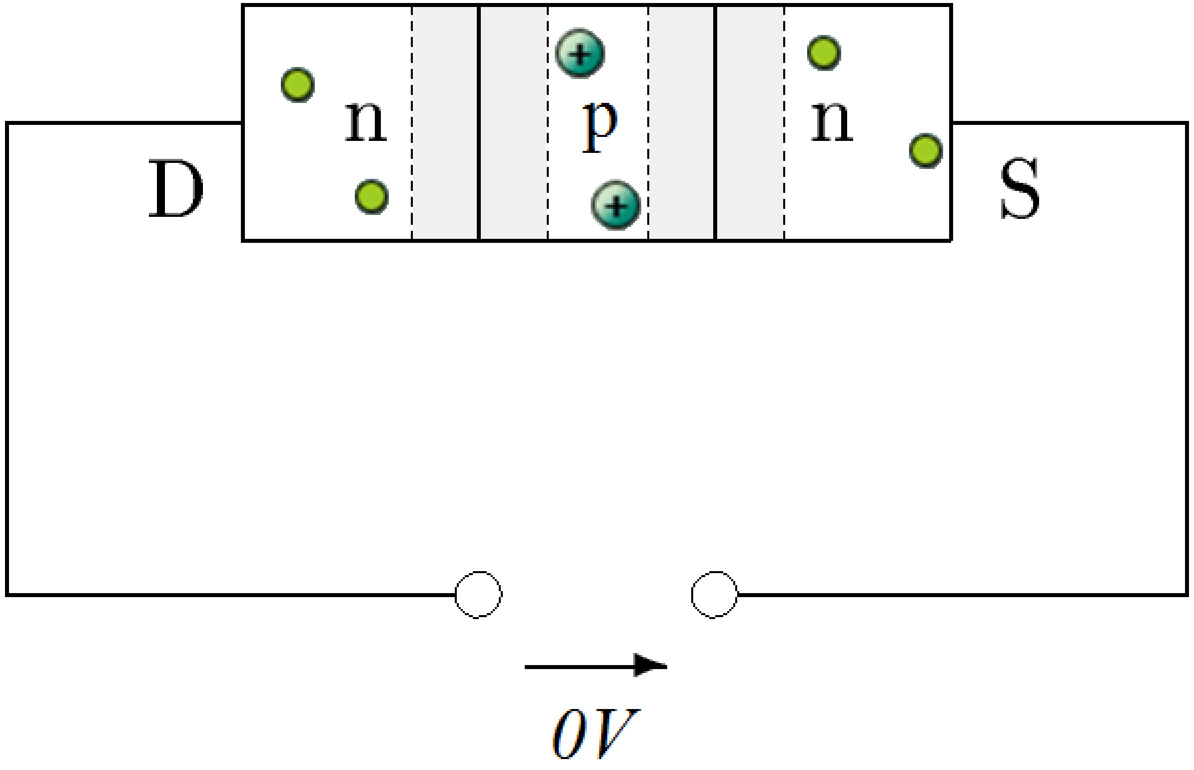
Bor (B)
Gallium (Ga)
Indium (In)

- **pn-Übergang**



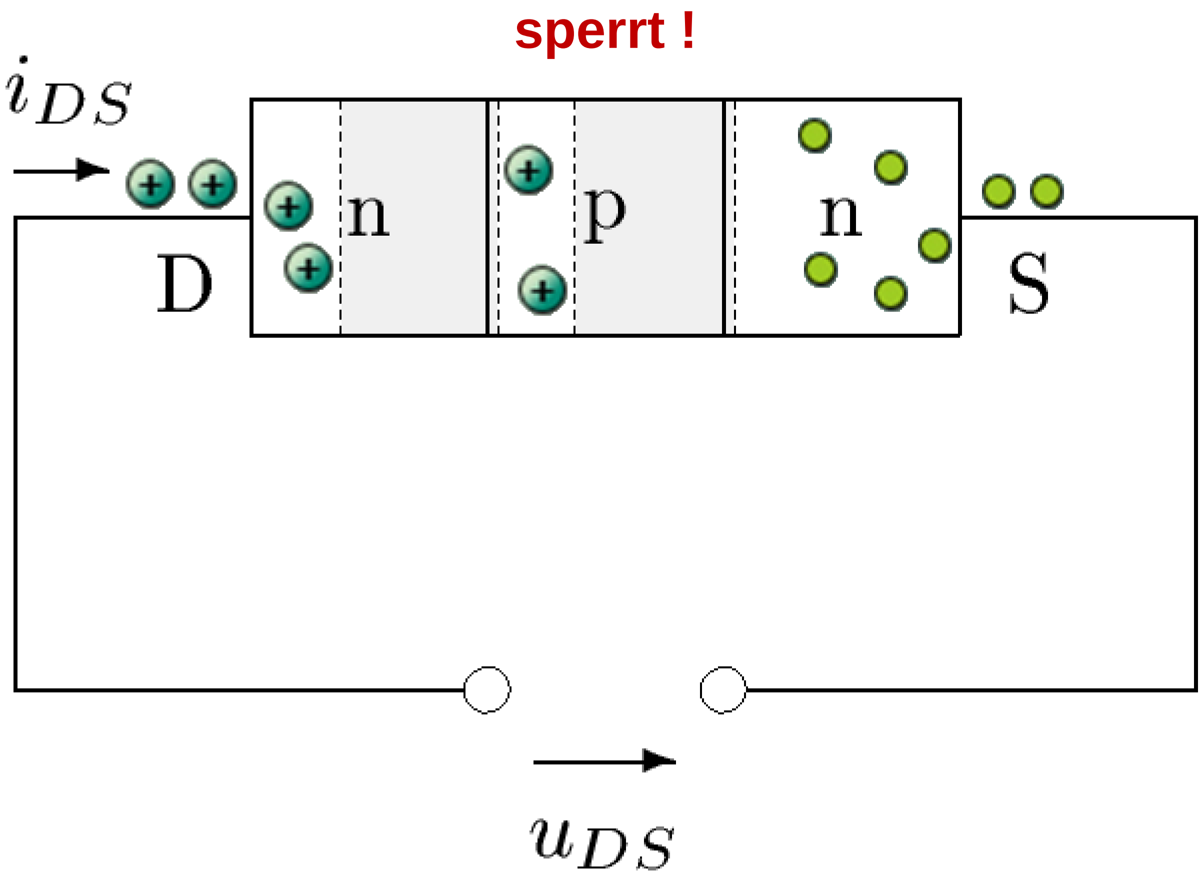
- **keine Spannung liegt an**

sperrt !



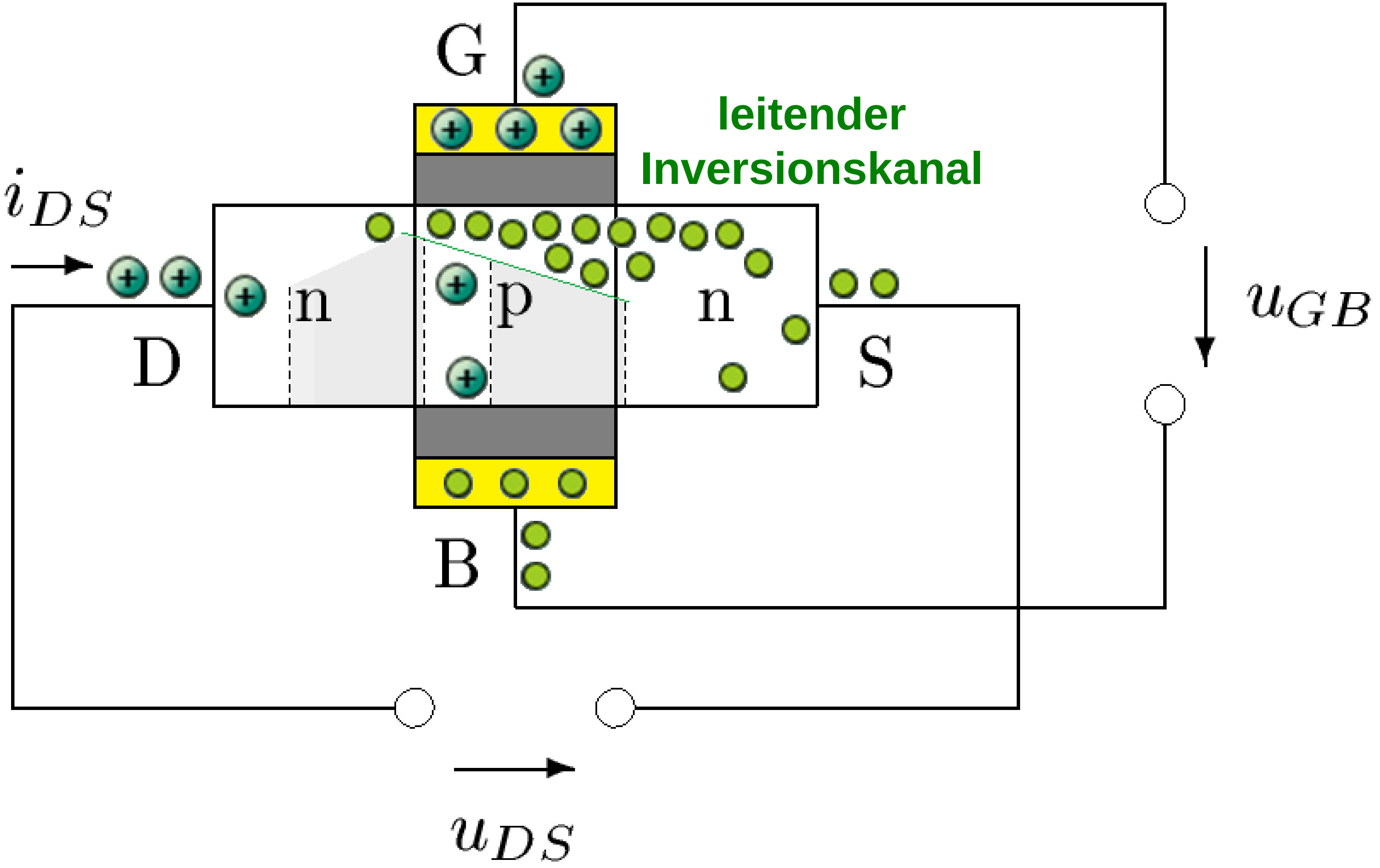
Feldeffekt-Transistor
npn-Zonenfolge

- **Spannung über der Drain-Source-Strecke**



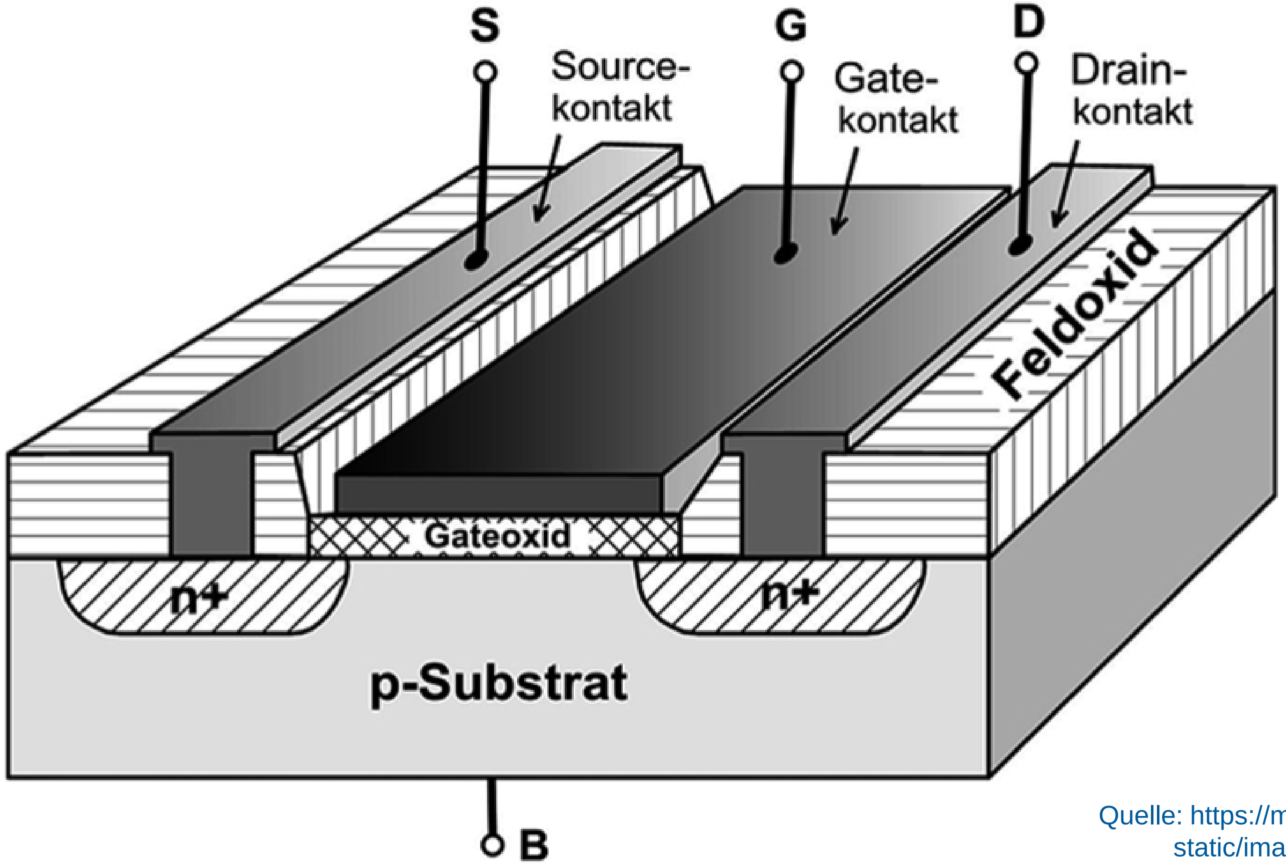
Feldeffekt-Transistor

Anlegen eines elektrischen Felds über der p-Zone



Feldeffekt-Transistor
Aufbau in Planartechnik

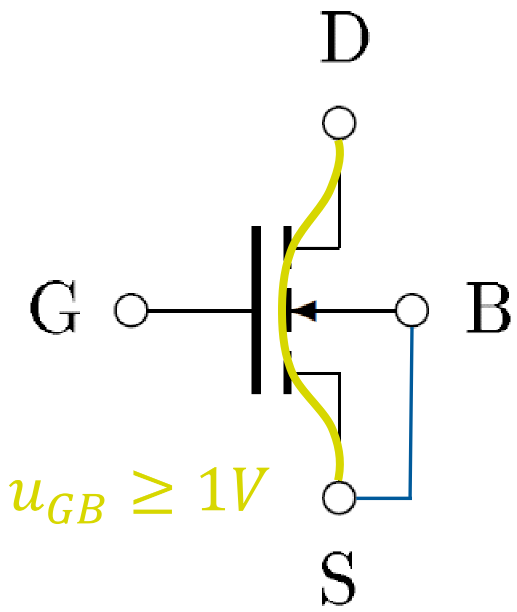
- **Schichtfolge: Metal – Oxide – Semiconductor (MOS)**



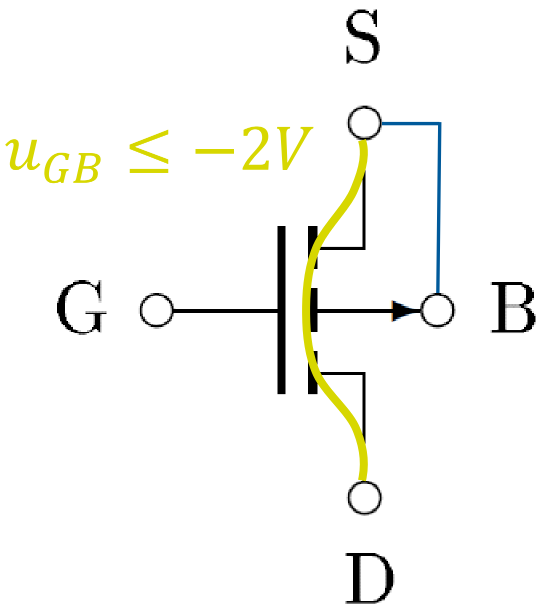
Quelle: https://media.springernature.com/original/springer-static/image/chp%3A10.1007%2F978-3-658-24752-2_6/MediaObjects/331316_4_De_6_Fig23_HTML.png

MOS-Feldeffekt-Transistoren

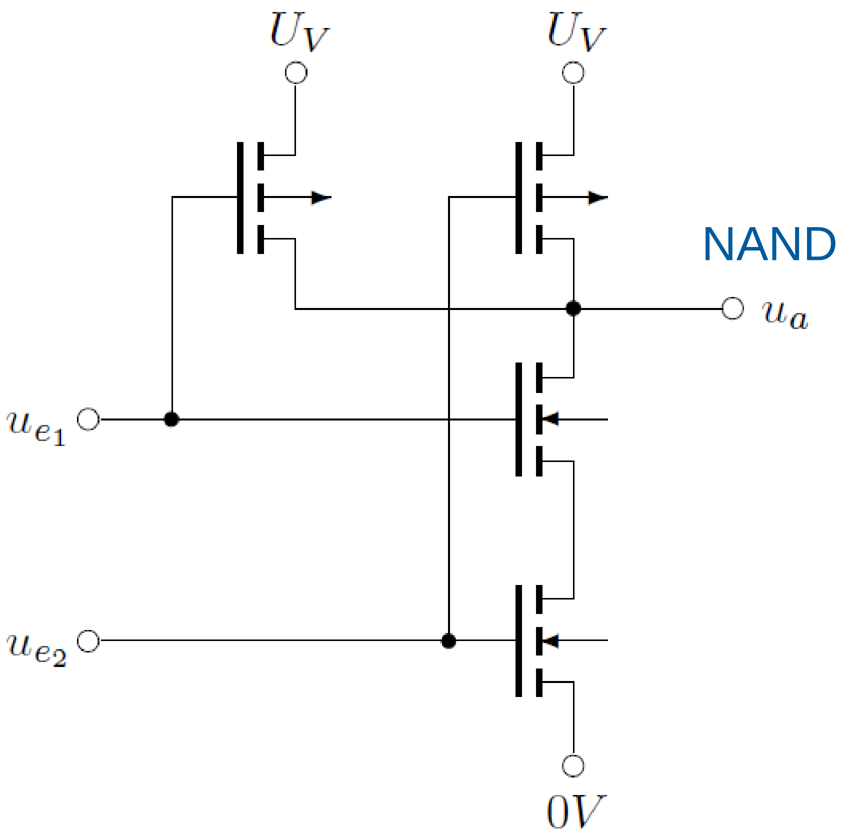
nMOS

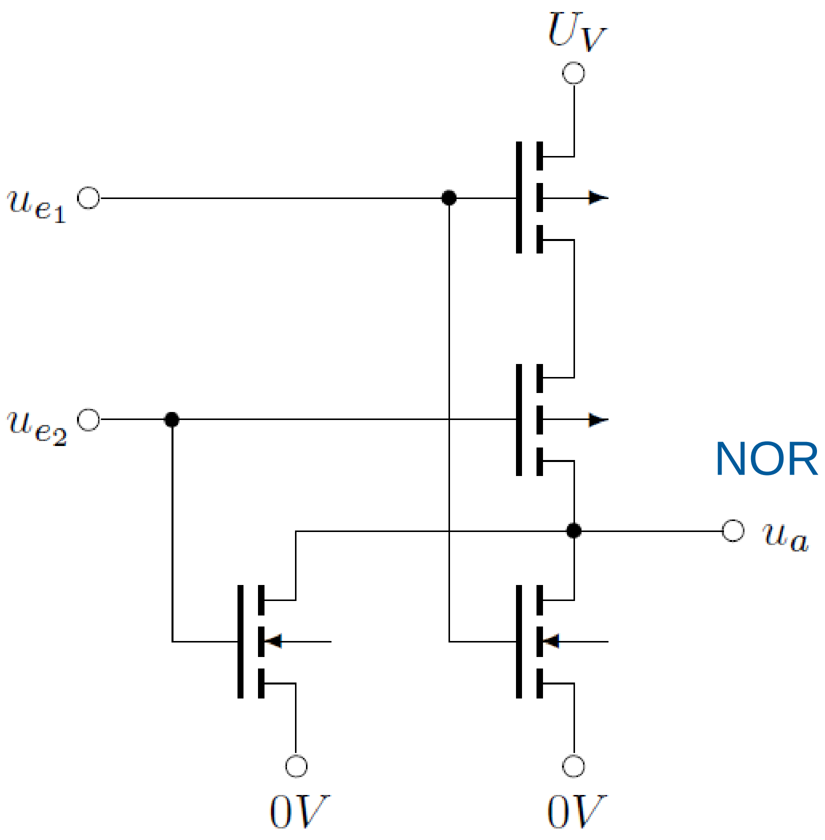


pMOS

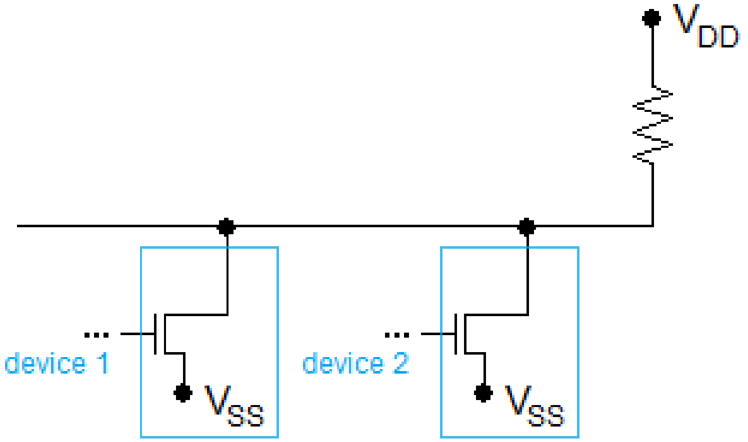


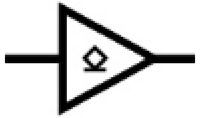
CMOS-Schaltungen
NAND und NOR



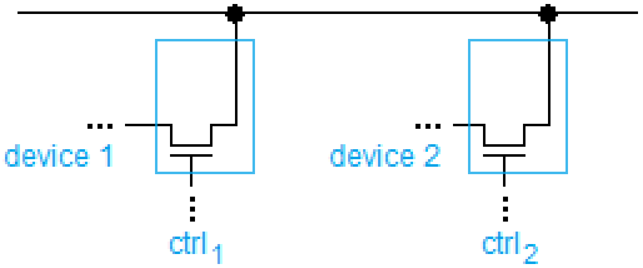


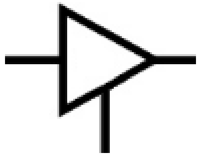
- **Wired-OR (Open Drain)**
  - low-aktive Signale
  - aktiviert durch <u>mindestens</u> einen Busteilnehmer



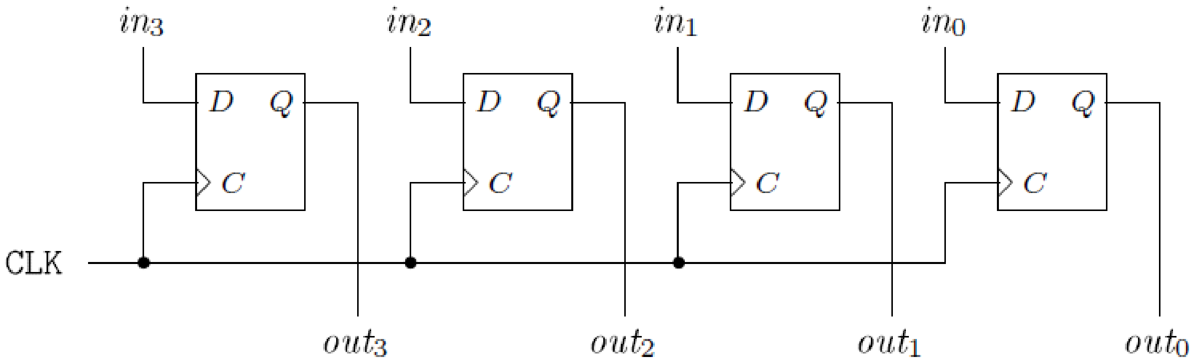


- **Tristate**
  - Z (hochohmig)
  - trennt einen Busteilnehmer vom Bus
  - <u>höchstens</u> ein Busteilnehmer gleichzeitig darf schreiben

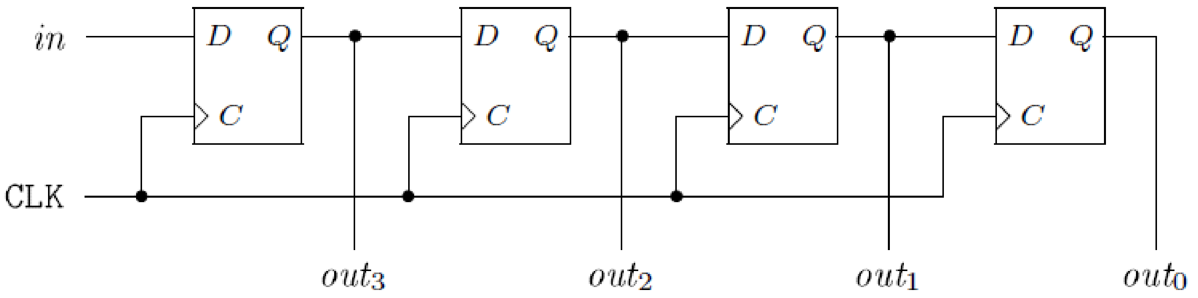




- **Grundschaltung eines 4-bit-Registers**



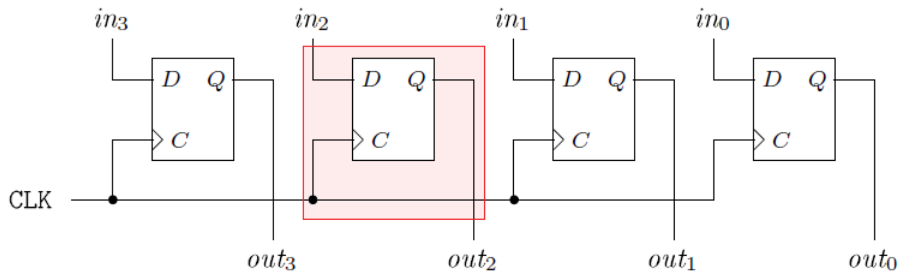
- **Grundschaltung eines 4-bit-Rechtsschieberegisters**

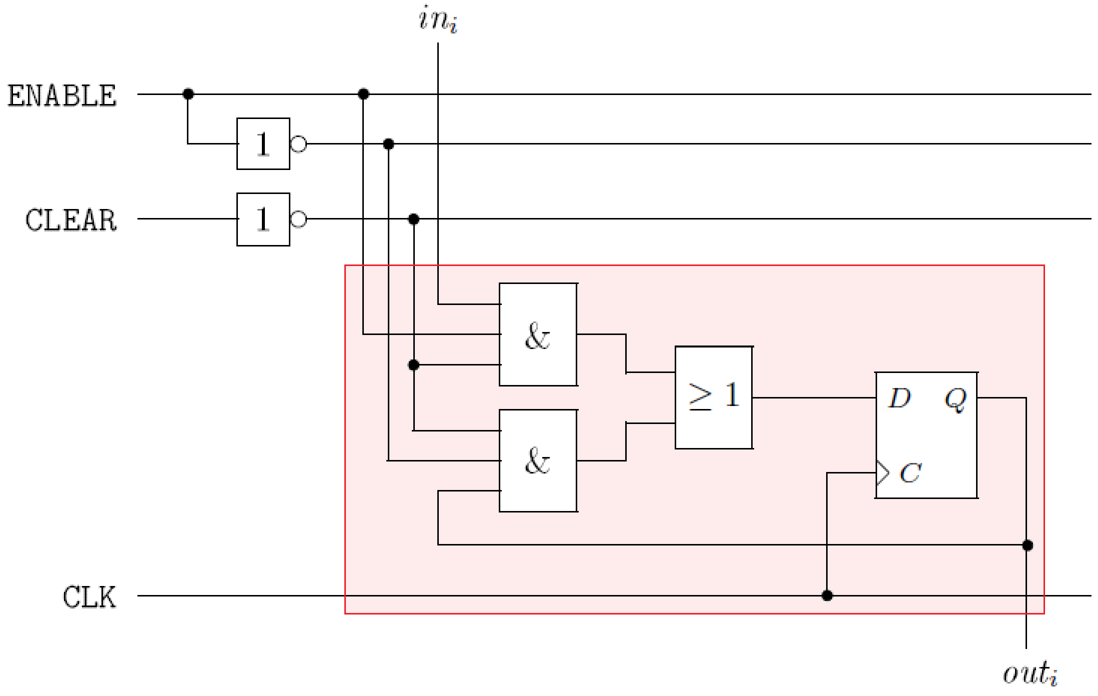


Erweiterung der Register-Grundschaltung

ENABLE und CLEAR-Steuersignale

CLEAR	ENABLE	D
0	0	out_i
0	1	in_i
1	0	0
1	1	0

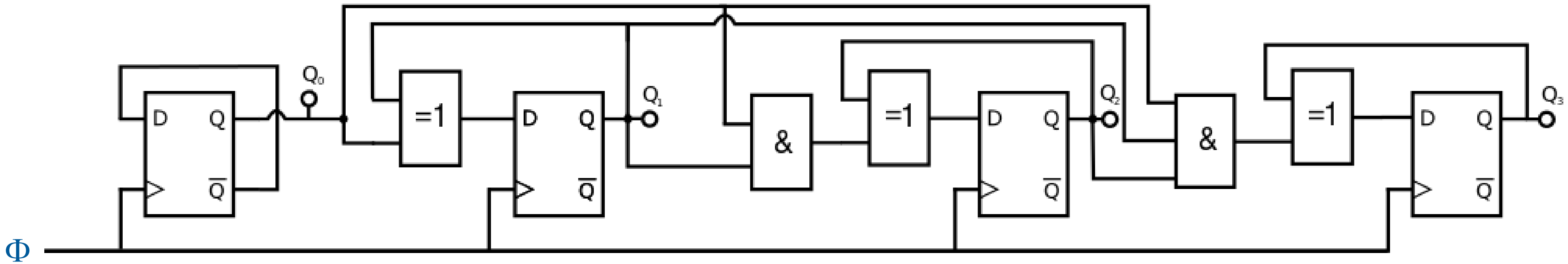




Erweiterung der Register-Grundschaltung

Zählfunktion

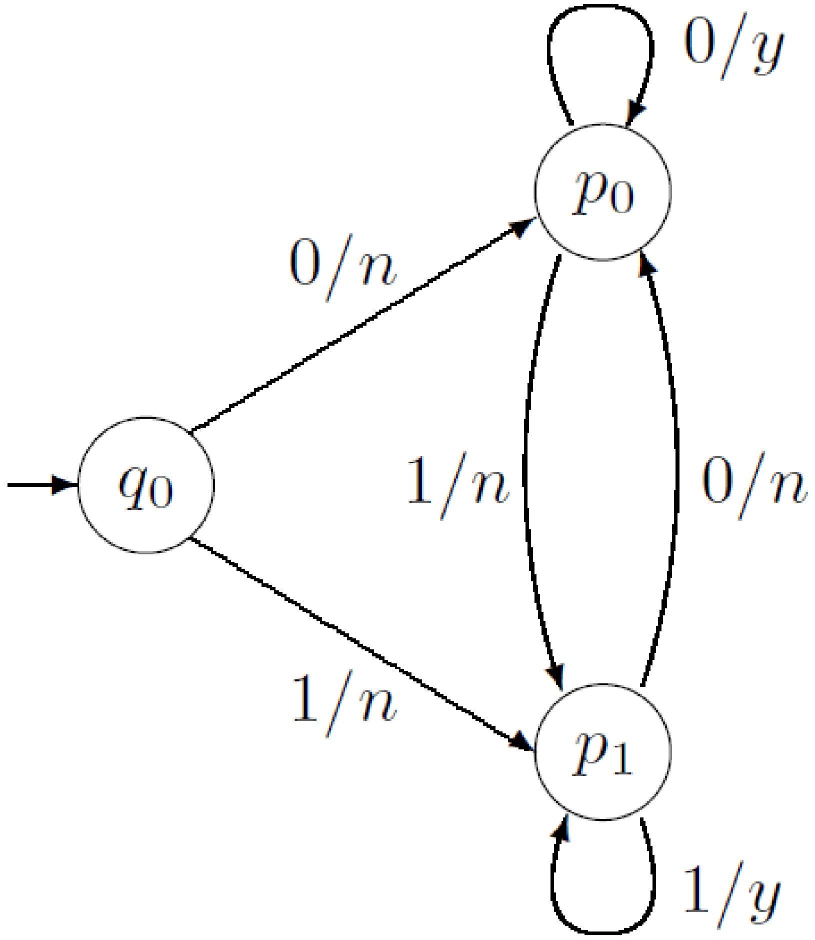
- **Synchroner 4-Bit-Vorwärtszähler**



niederwertigste Stelle

höchstwertige Stelle

Endliche Automaten
Beispiel



	0	1	1	0	0	0	1	0 …
	n	n	y	n	y	y	n	$n\ldots$
q_0	p_0	p_1	p_1	p_0	p_0	p_0	p_1	p_0 …

Endliche Automaten

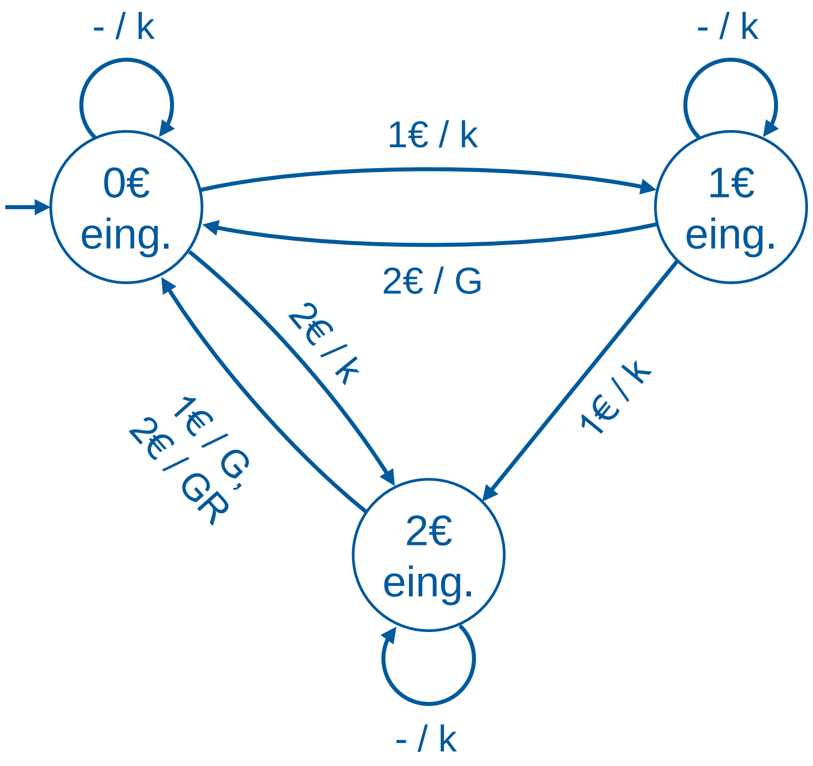
Beispiel: Steuerung eines Getränke-Münzautomaten

- ein Becher Getränk kostet 3€
- Münzeingabe: 1€ oder 2€
- sobald der Verkaufspreis erreicht oder überschritten ist, wird ein Becher Getränk ausgegeben
- bei Überzahlung wird das Restgeld ebenfalls ausgegeben

Spezifikation mittels Automatengraph

X = { -, 1€, 2€ }
Münzeinwurf

Y = { k, G, GR }
kein Getränk, **G**etränk,
Getränk und **R**ückgeld



Technische Realisierung

- Schritt 1: Codierungen festlegen

X	x_1	x_0
-	0	0
1€	0	1
2€	1	0

Y	y_1	y_0
k	0	0
G	1	0
GR	1	1

Z	z_1	z_0
0€ eing.	0	0
1€ eing.	0	1
2€ eing.	1	0

- Schritt 2: Automatentabelle aufstellen

z_1	z_0	x_1	x_0	y_1	y_0	z_1	z_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	D	D	D	D
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	1	0	0	0
0	1	1	1	D	D	D	D
1	0	0	0	0	0	1	0
1	0	0	1	1	0	0	0
1	0	1	0	1	1	0	0
1	0	1	1	D	D	D	D
1	1	0	0	D	D	D	D
1	1	0	1	D	D	D	D
1	1	1	0	D	D	D	D
1	1	1	1	D	D	D	D

Technische Realisierung

- Schritt 3: pro Ausgabespalte minimale DNF finden

y_1

z_1z_0 \ x_1x_0	00	01	11	10
00			D	
01			D	1
11	D	D	D	D
10		1	D	1

y_0

z_1z_0 \ x_1x_0	00	01	11	10
00			D	
01			D	
11	D	D	D	D
10			D	1

z'_1

z_1z_0 \ x_1x_0	00	01	11	10
00			D	1
01		1	D	
11	D	D	D	D
10	1		D	

z'_0

z_1z_0 \ x_1x_0	00	01	11	10
00		1	D	
01	1		D	
11	D	D	D	D
10			D	

$$y_1 = x_0 z_1 + x_1 z_0 + x_1 z_1$$

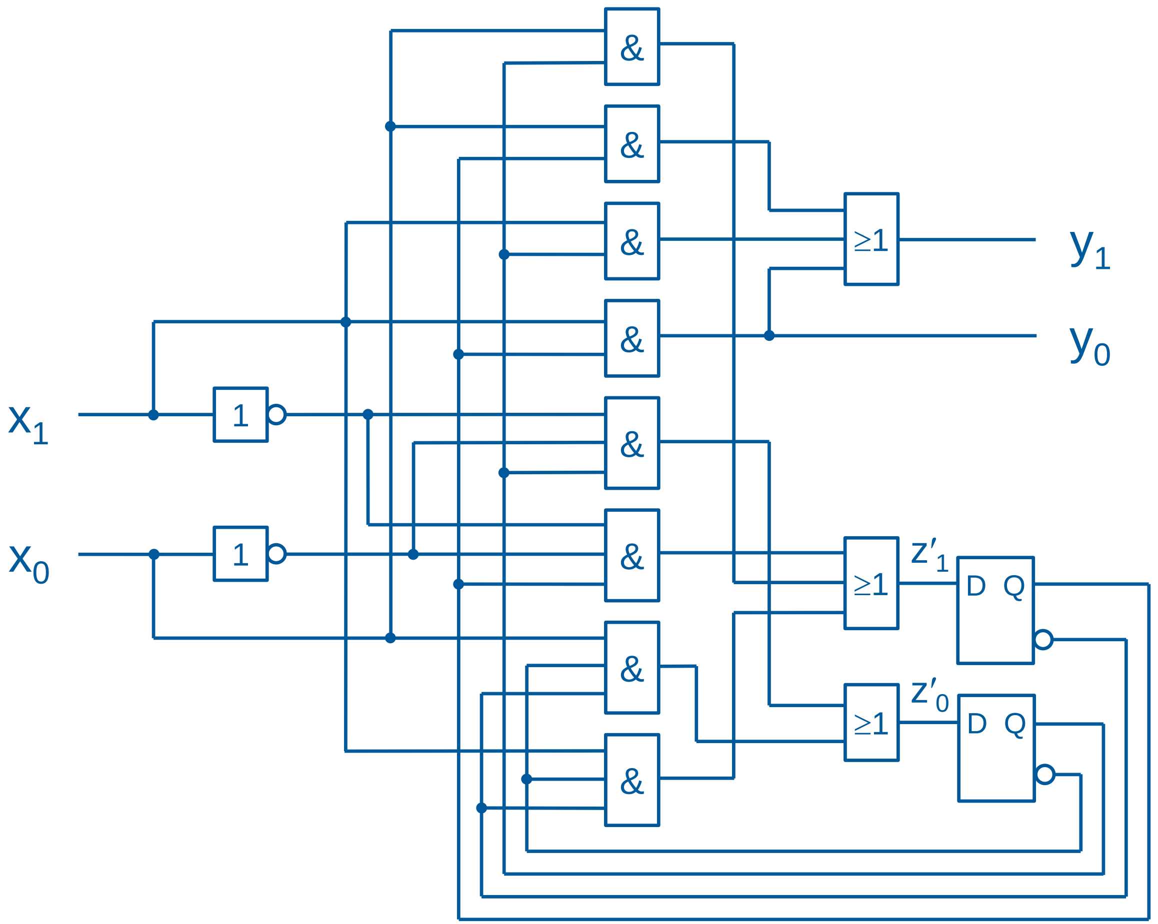
$$z_1' = x_0 z_0 + \bar{x}_0 \bar{x}_1 z_1 + x_1 \bar{z}_0 \bar{z}_1$$

$$y_0 = x_1 z_1$$

$$z_0' = \bar{x}_0 \bar{x}_1 z_0 + x_0 \bar{z}_0 \bar{z}_1$$

Technische Realisierung

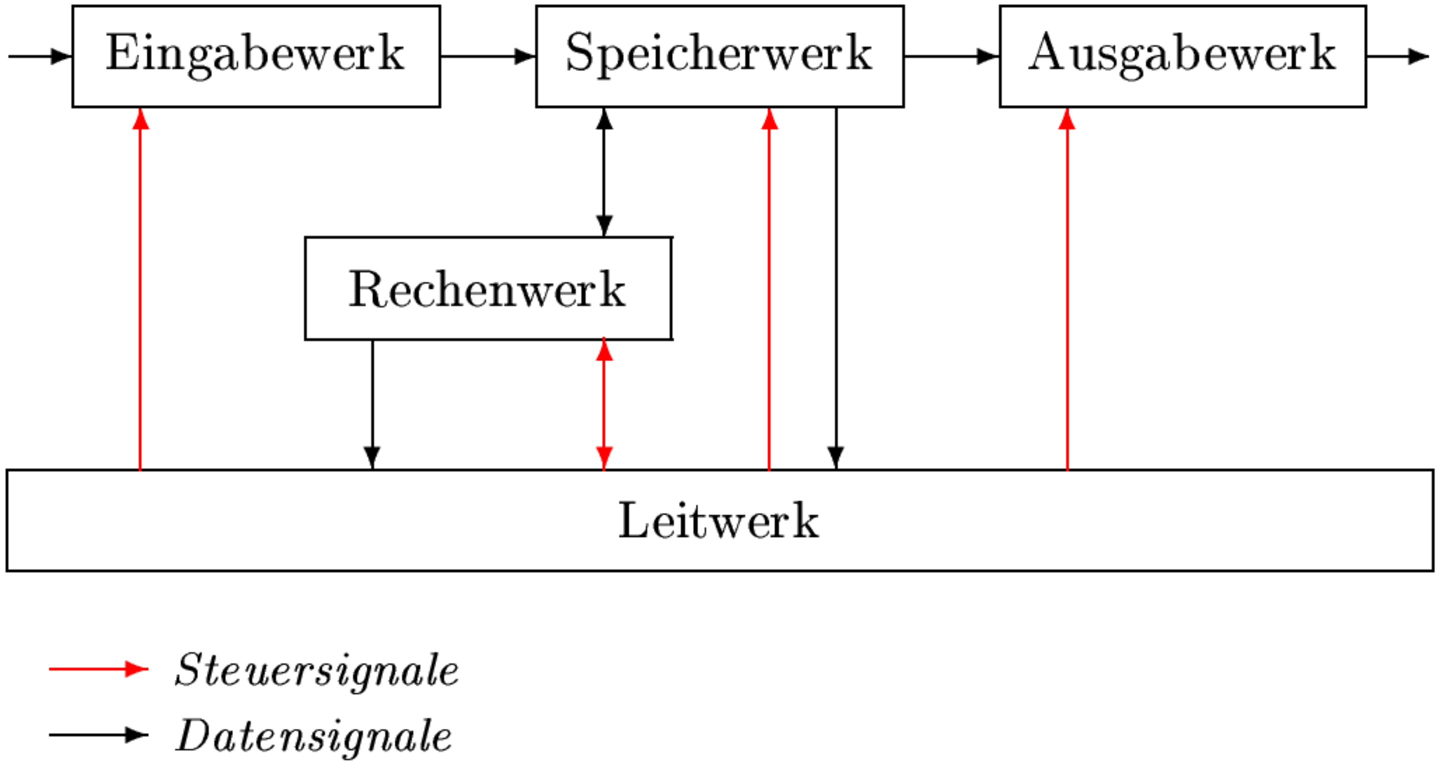
- Schritt 4: Schaltbild erzeugen



Princeton-Rechner

John von Neumann, Arthur W. Burks, Herman H. Goldstine, 1946/47

- **„Klassischer Universalrechenautomat“**



von Neumann-Architektur

- Die Struktur des Rechners ist **unabhängig** von speziellen, zu bearbeitenden Problemen.
- Vielmehr wird für jedes Problem eine Bearbeitungsvorschrift, das **Programm**, von außen eingegeben und im Speicher abgelegt. Erst dieses Programm macht den Rechner arbeitsfähig.
- Programme und von diesen benötigte Daten sowie Zwischen- und Endergebnisse werden in **einem einheitlichen** Speicher abgelegt.
- Befehle eines Programms werden im allgemeinen aus aufeinanderfolgenden Speicherplätzen geholt. Diese **sequentielle** Verarbeitung kann jedoch durch Sprungbefehle unterbrochen werden.

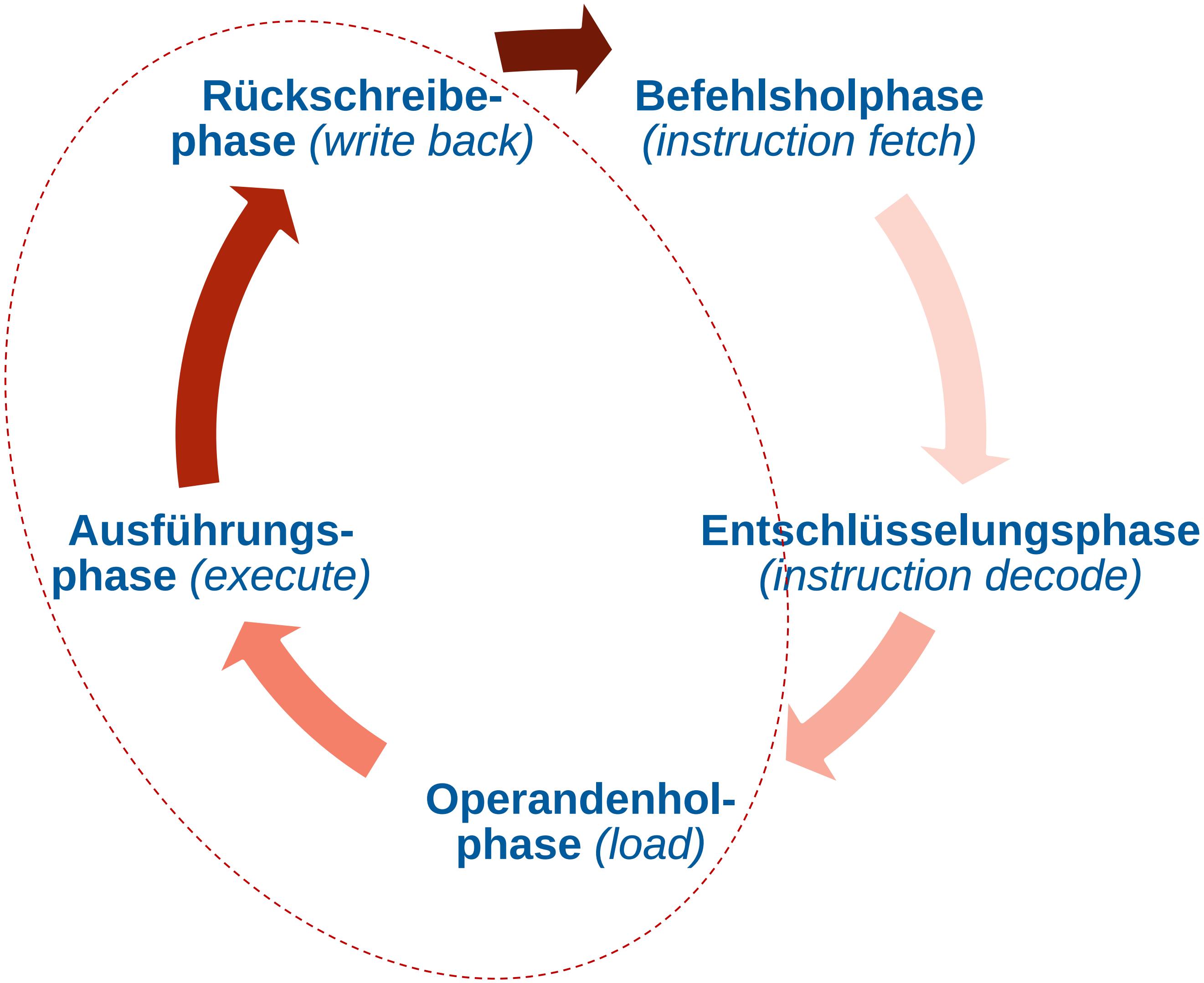
von Neumann-Architektur

Erweiterungen

- Die Werke werden nicht mehr paarweise miteinander verbunden, sondern durch eine gemeinsame Übertragungsschiene (**Bus**).
- Statt eines einziges Rechenregisters (Akkumulator, AC) werden im Rechenwerk mehrere Universalregister (**Registersatz**) verwendet.
- Leit- und Rechenwerk werden gemeinsam als **Zentraleinheit** bezeichnet (CPU, central processing unit)
- Eingabe- und Ausgabewerk werden zu einem **E/A-Werk** zusammengefasst. „Das“ E/A-Werk ist Stellvertreter für viele im Grundsatz gleichartige E/A-Werke (Tastatur und Maus, Monitor, HDD, optische Laufwerke, LAN-Schnittstelle, USB-Schnittstelle, usw.)

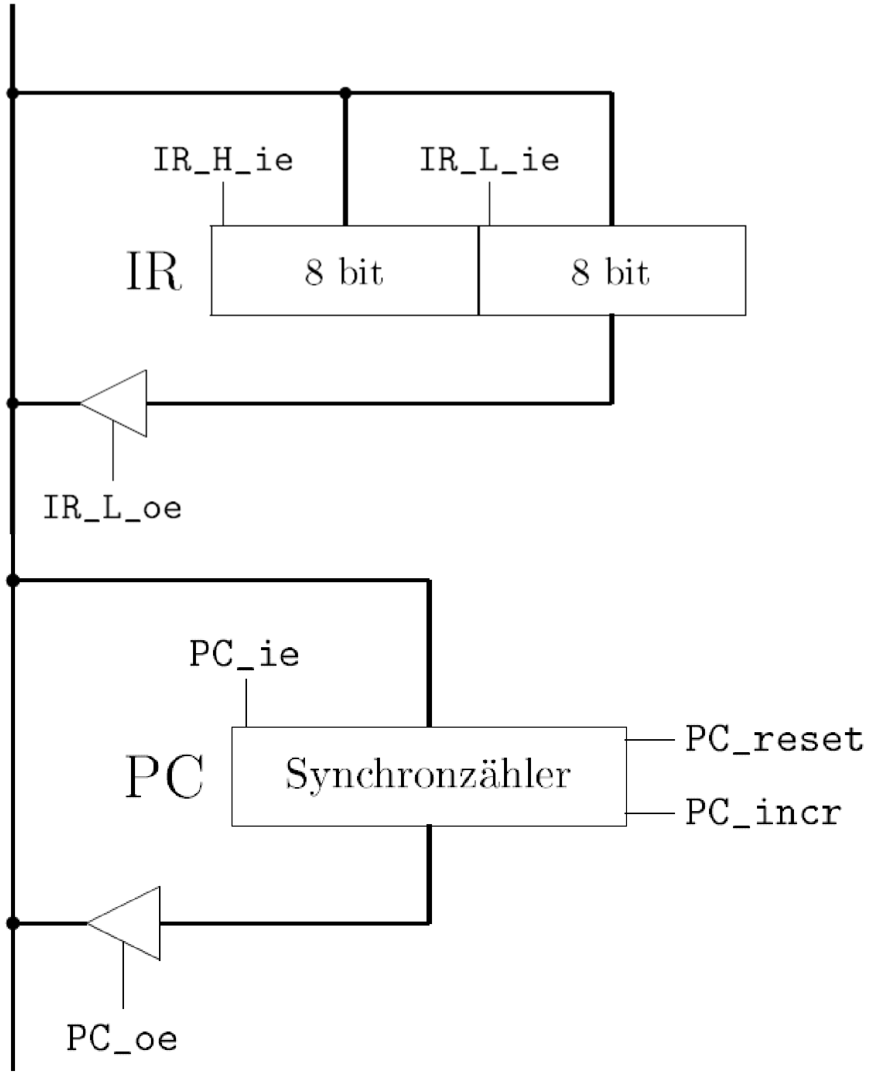
Befehlszyklus

3..5 Phasen



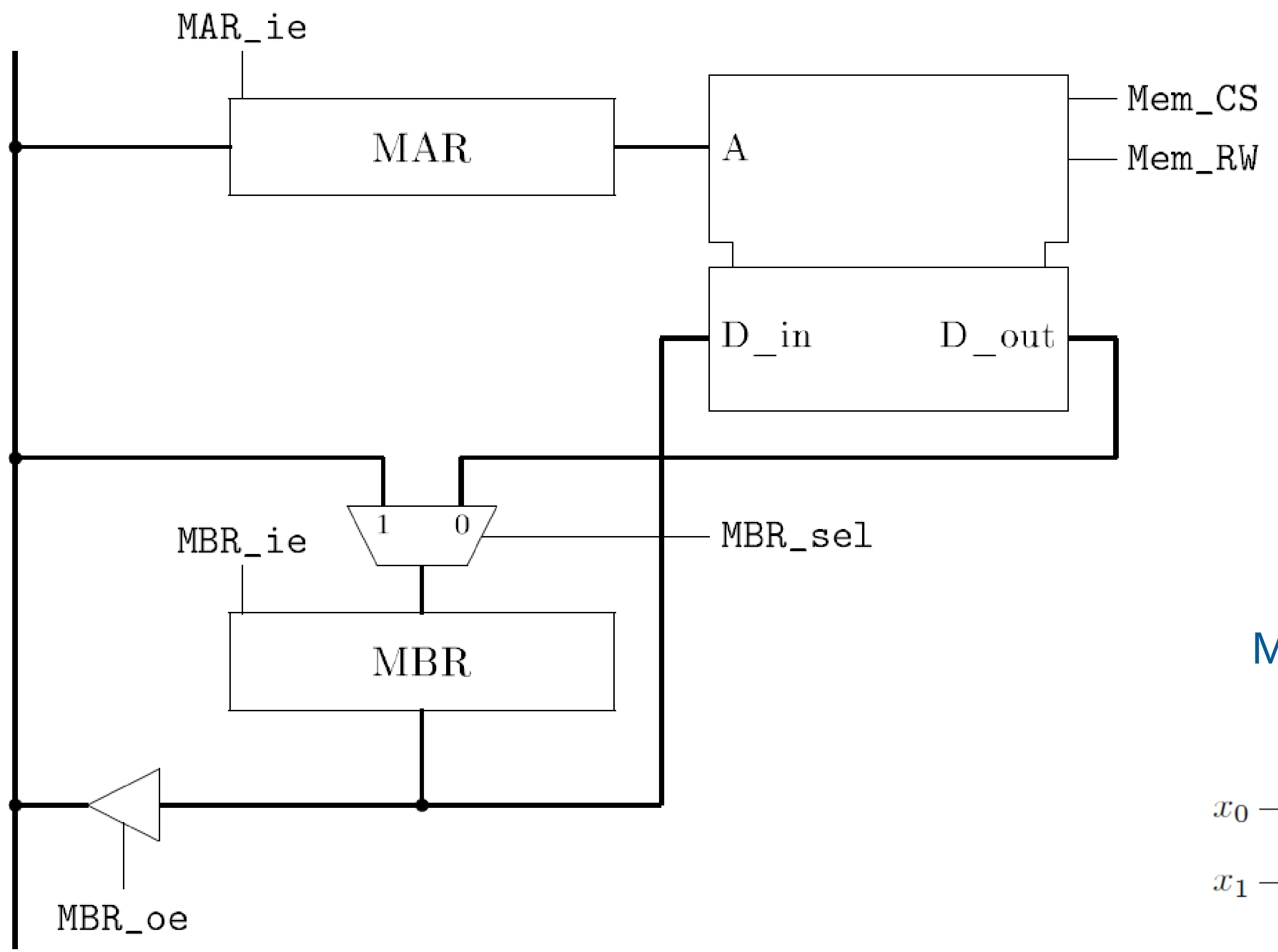
Realisierung eines einfachen Rechners mit BROOKSHEARS Befehlssatz

IR und PC

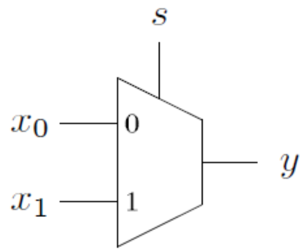


_ie input enable

_oe output enable

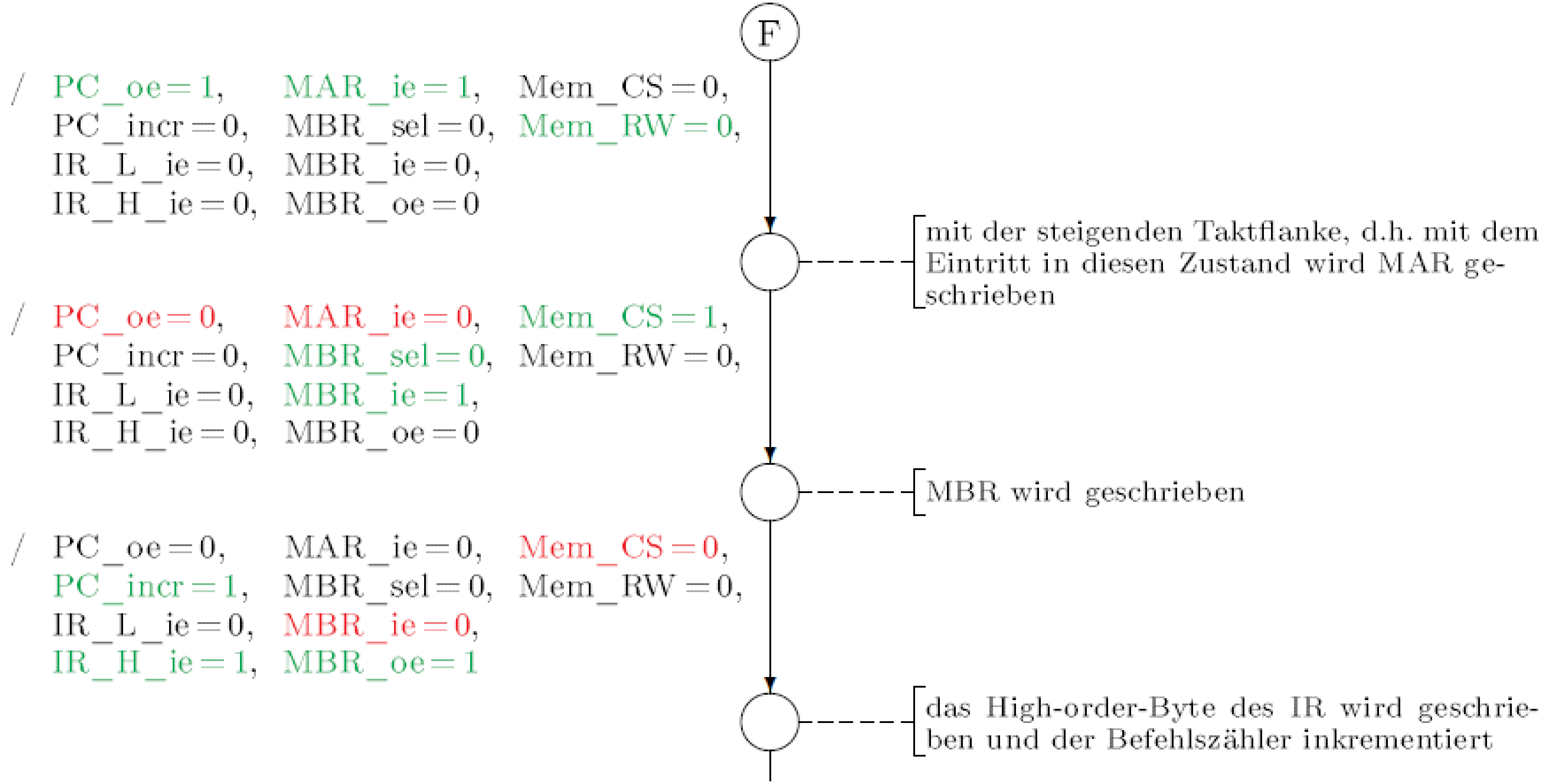


Multiplexer (1-MUX)



s	y
0	x_0
1	x_1

Befehlsholphase (1)



Befehlsholphase (2)

das High-order-Byte des IR wird geschrieben und der Befehlszähler inkrementiert

/ PC_oe = 1, MAR_ie = 1, Mem_CS = 0,
PC_incr = 0, MBR_sel = 0, Mem_RW = 0,
IR_L_ie = 0, MBR_ie = 0,
IR_H_ie = 0, MBR_oe = 0

MAR wird geschrieben

/ PC_oe = 0, MAR_ie = 0, Mem_CS = 1,
PC_incr = 0, MBR_sel = 0, Mem_RW = 0,
IR_L_ie = 0, MBR_ie = 1,
IR_H_ie = 0, MBR_oe = 0

MBR wird geschrieben

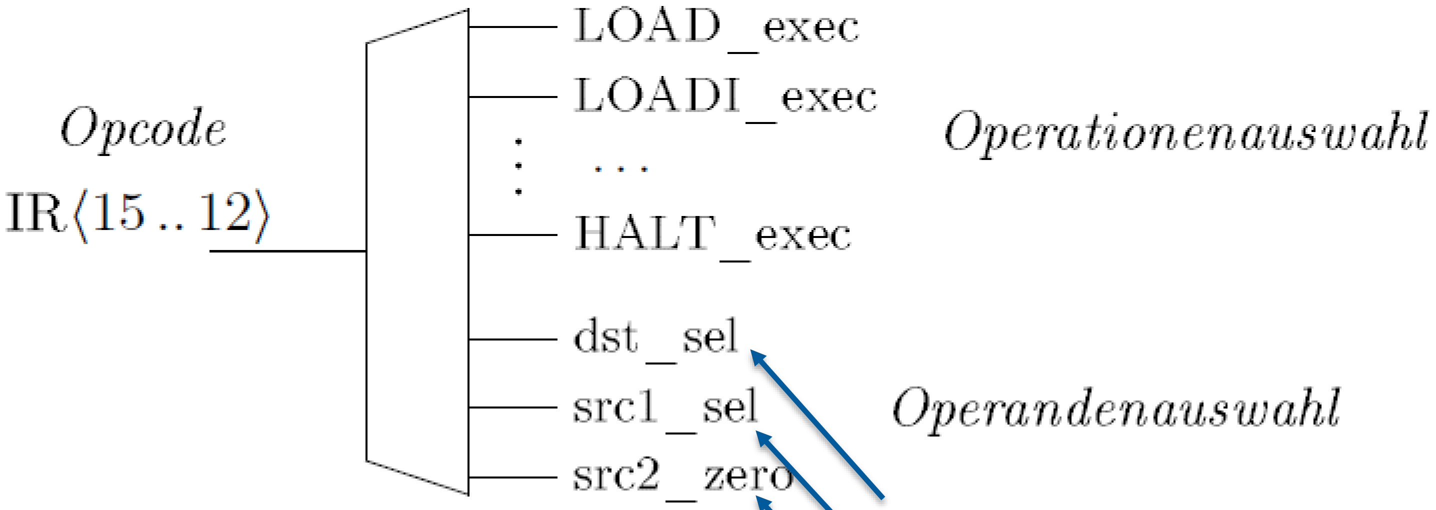
/ PC_oe = 0, MAR_ie = 0, Mem_CS = 0,
PC_incr = 1, MBR_sel = 0, Mem_RW = 0,
IR_L_ie = 1, MBR_ie = 0,
IR_H_ie = 0, MBR_oe = 1

X

das Low-order-Byte des IR wird geschrieben und der Befehlszähler inkrementiert

Realisierung eines einfachen Rechners mit Brookshears Befehlssatz

Befehlsentschlüsselung



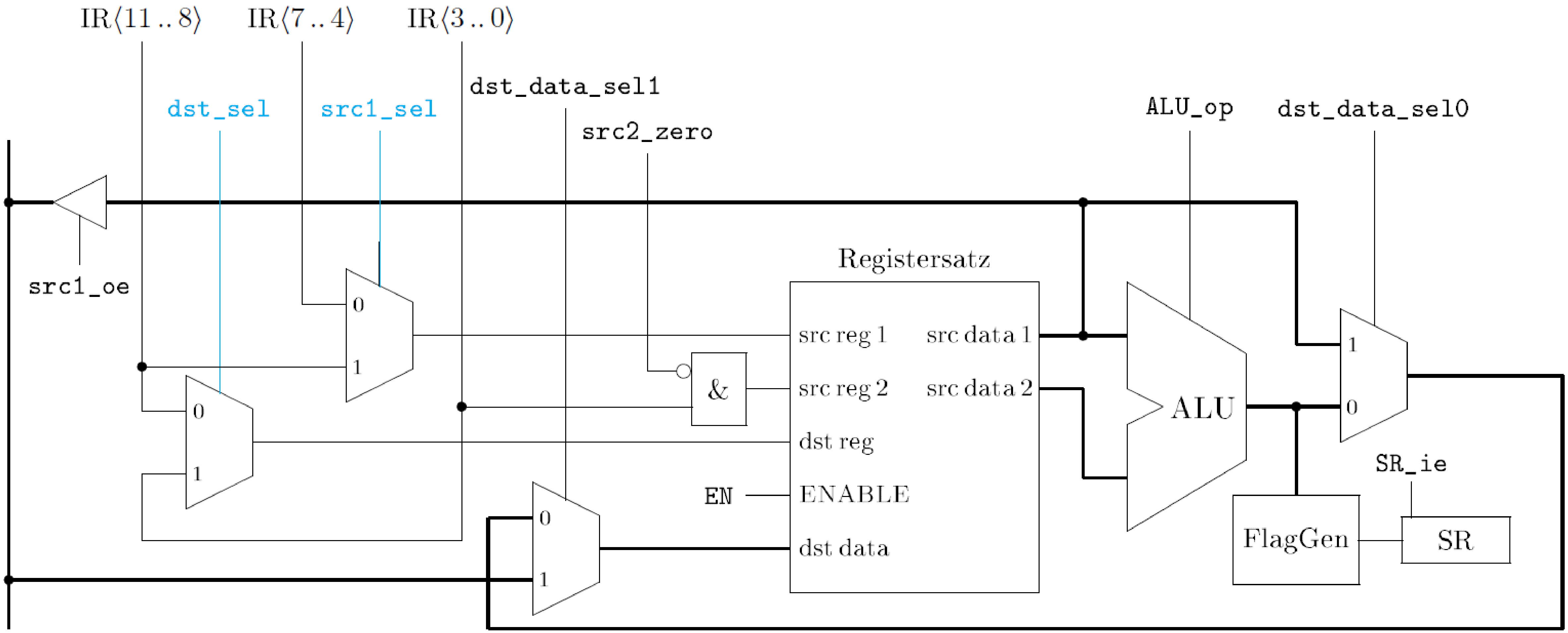
identisch mit MOVE_exec

kennz. STORE, JUMP und ROTATE-RIGHT

identisch mit JUMP_exec

Realisierung eines einfachen Rechners mit BROOKSHEARS Befehlssatz

Rechenwerk

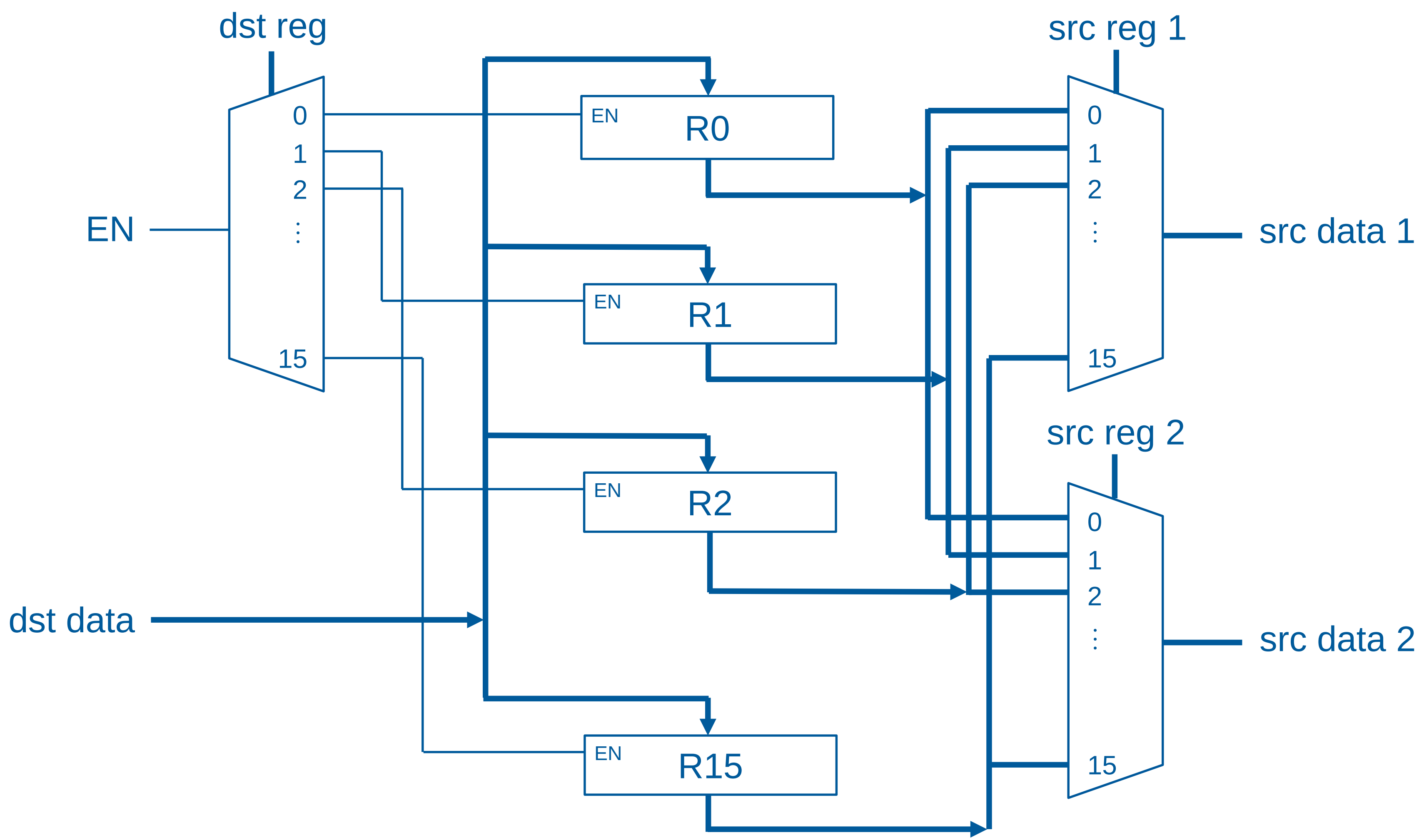


8-fach NOR

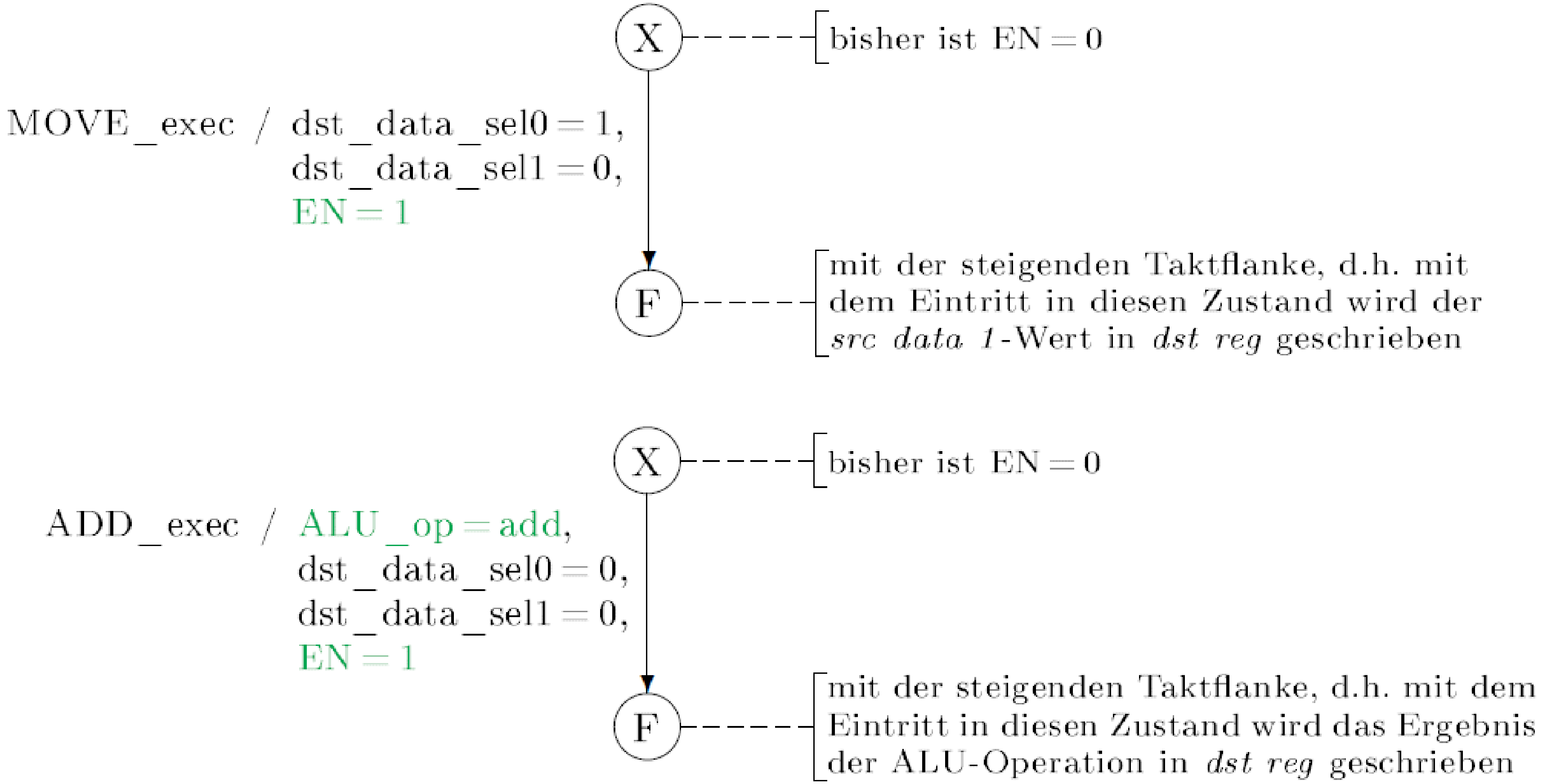
nur ZF benötigt

Realisierung eines einfachen Rechners mit BROOKSHEARS Befehlssatz

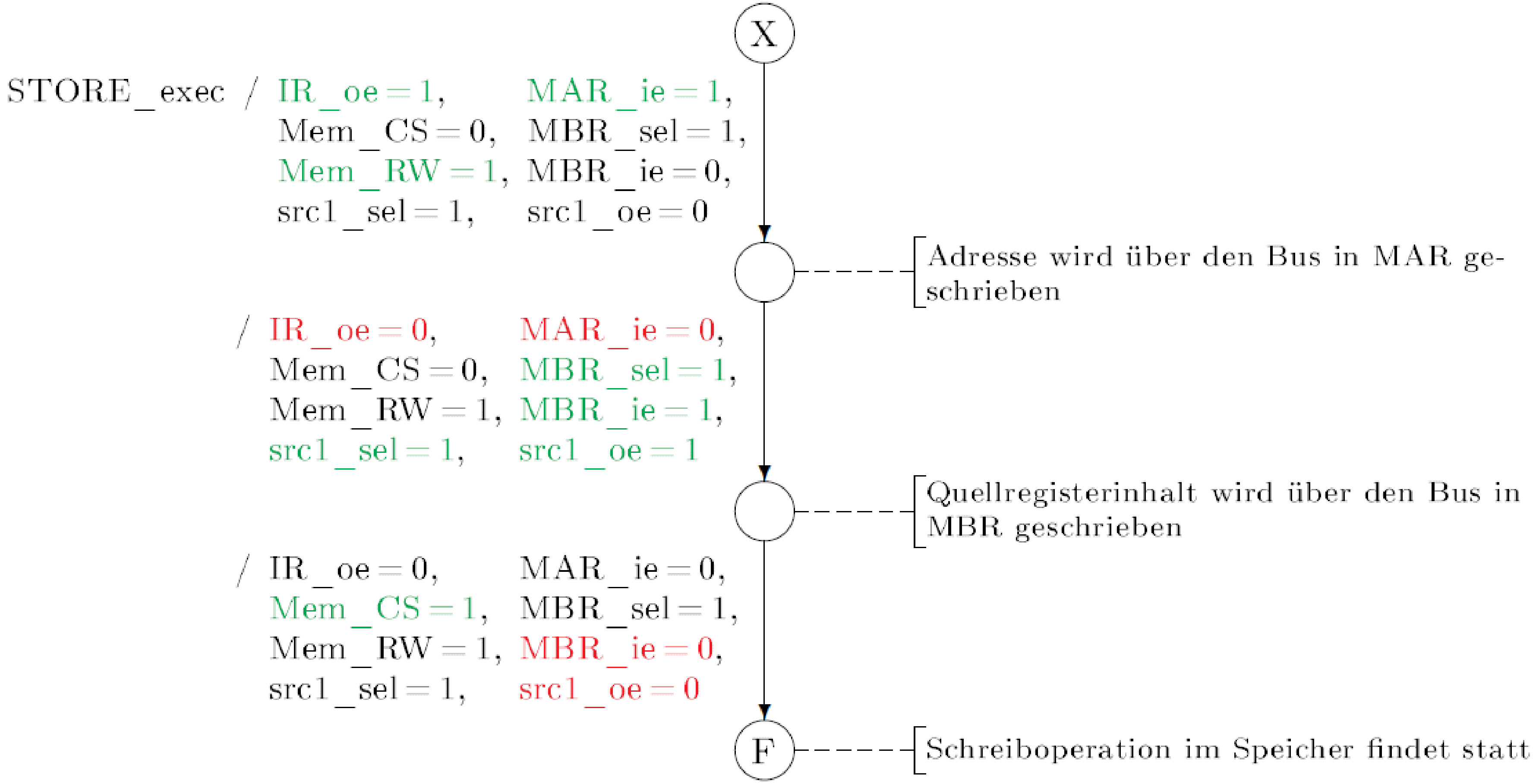
Registersatz



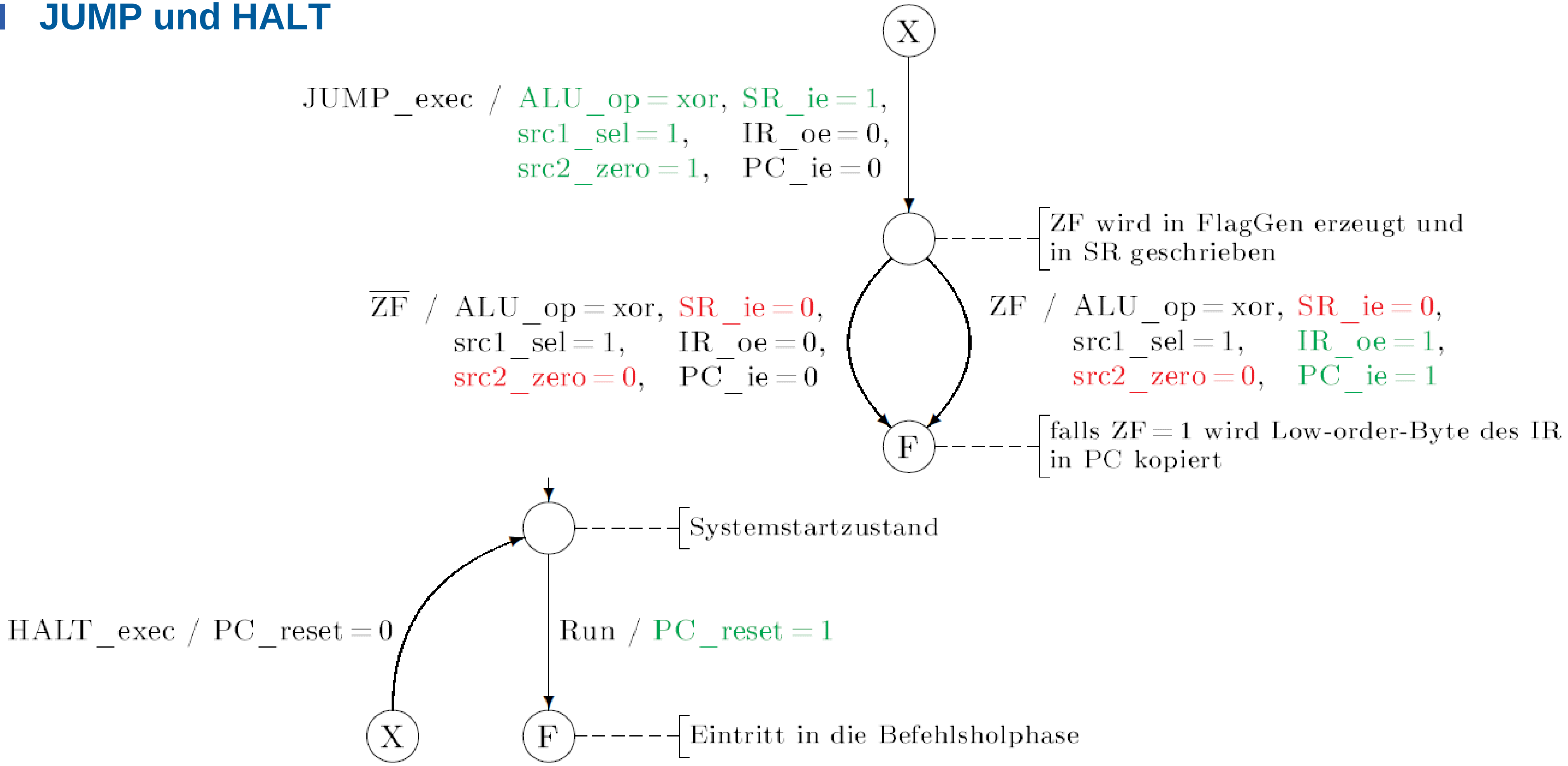
MOVE und ADD


X
bisher ist EN = 0
MOVE_exec / dst_data_sel0 = 1,
dst_data_sel1 = 0,
EN = 1
F
mit der steigenden Taktflanke, d.h. mit dem Eintritt in diesen Zustand wird der *src data 1*-Wert in *dst reg* geschrieben
X
bisher ist EN = 0
ADD_exec / ALU_op = add,
dst_data_sel0 = 0,
dst_data_sel1 = 0,
EN = 1
F
mit der steigenden Taktflanke, d.h. mit dem Eintritt in diesen Zustand wird das Ergebnis der ALU-Operation in *dst reg* geschrieben

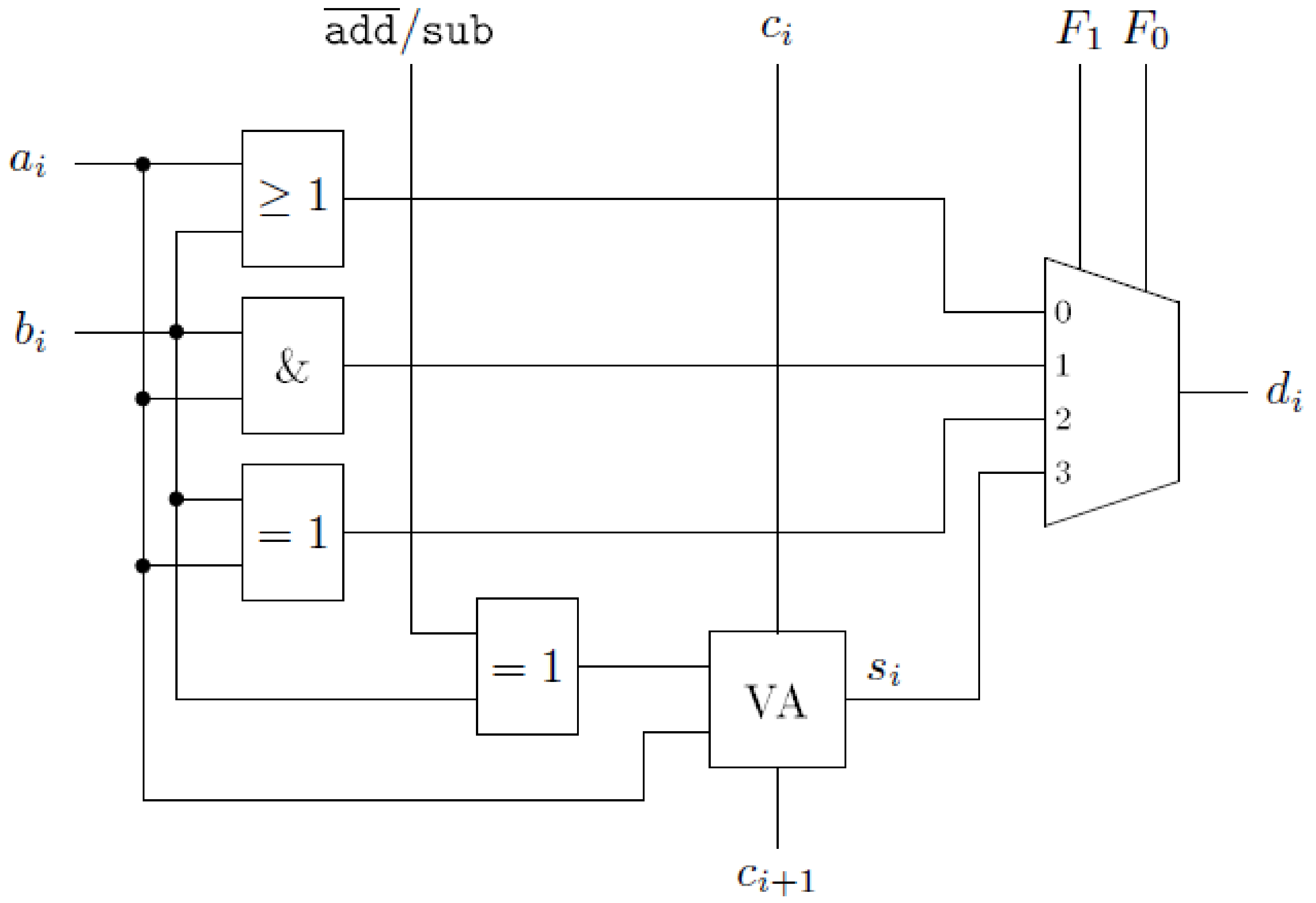
STORE


X
STORE_exec / IR_oe = 1, MAR_ie = 1,
Mem_CS = 0, MBR_sel = 1,
Mem_RW = 1, MBR_ie = 0,
src1_sel = 1, src1_oe = 0
Adresse wird über den Bus in MAR geschrieben
/ IR_oe = 0, MAR_ie = 0,
Mem_CS = 0, MBR_sel = 1,
Mem_RW = 1, MBR_ie = 1,
src1_sel = 1, src1_oe = 1
Quellregisterinhalt wird über den Bus in MBR geschrieben
/ IR_oe = 0, MAR_ie = 0,
Mem_CS = 1, MBR_sel = 1,
Mem_RW = 1, MBR_ie = 0,
src1_sel = 1, src1_oe = 0
F
Schreiboperation im Speicher findet statt

- **JUMP und HALT**

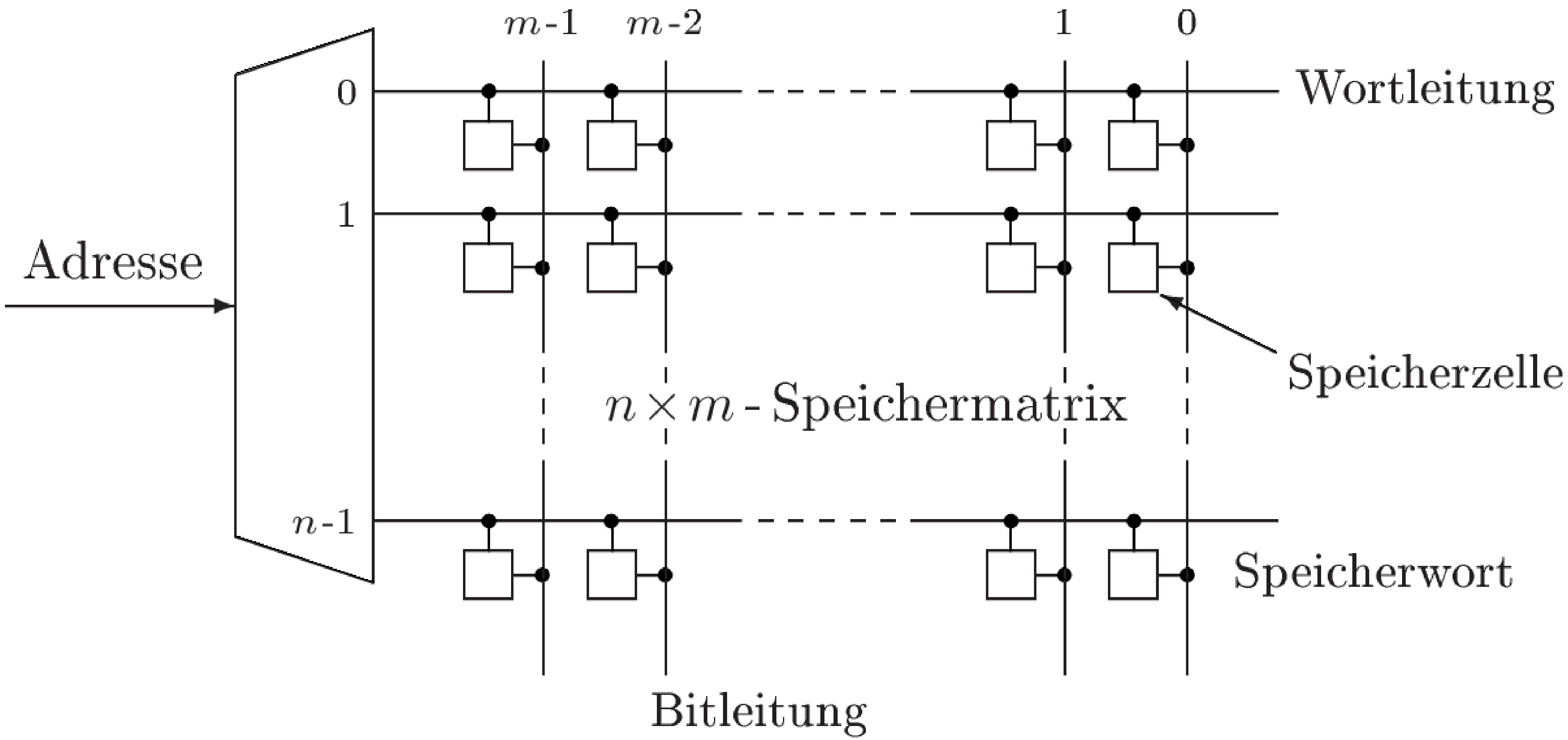


1-bit-ALU

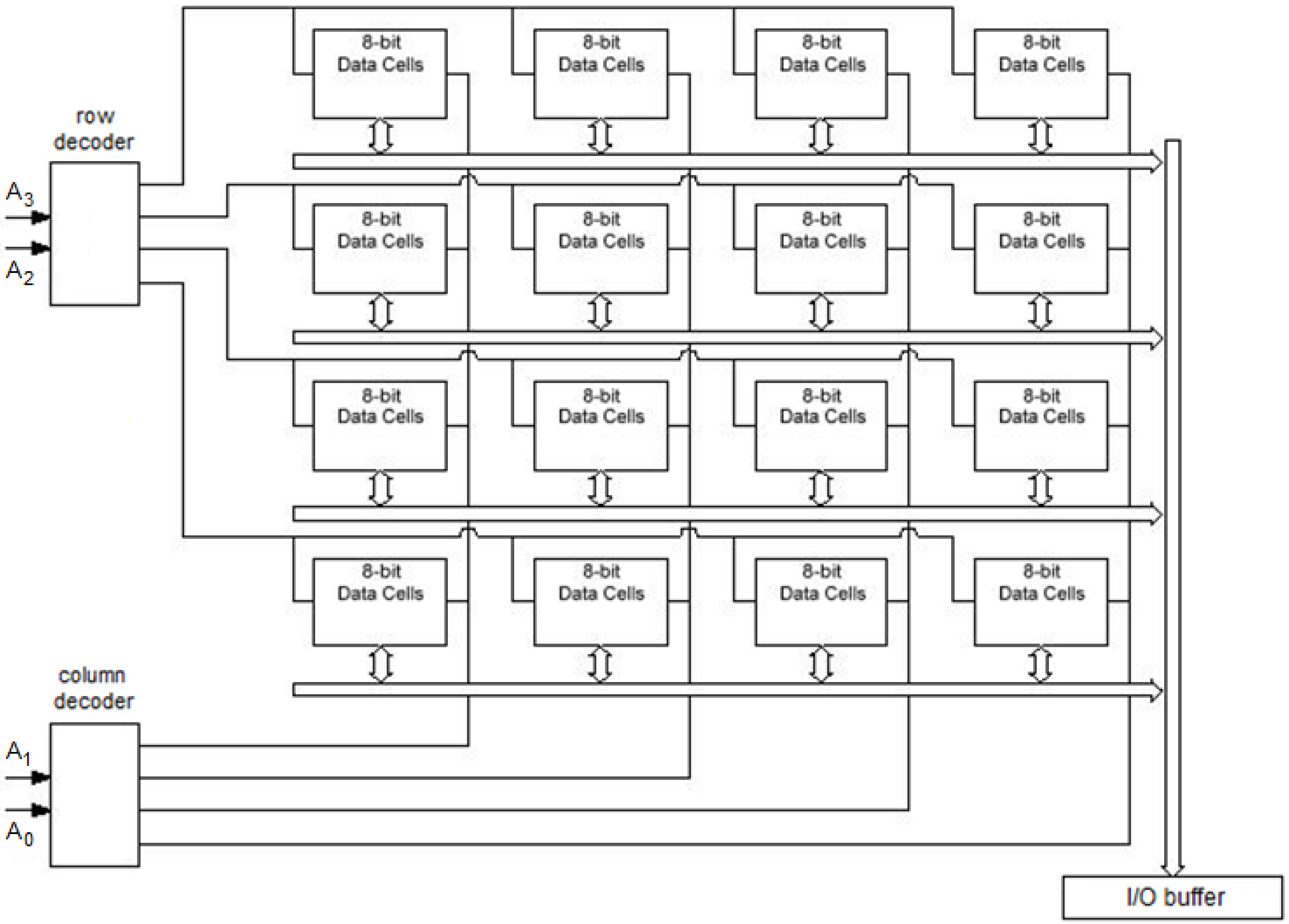


$\overline{\mathtt{add}}/\mathtt{sub}$
c_i
F_1 F_0
a_i
b_i
≥ 1
&
$= 1$
0
1
2
3
d_i
$= 1$
VA
s_i
c_{i+1}

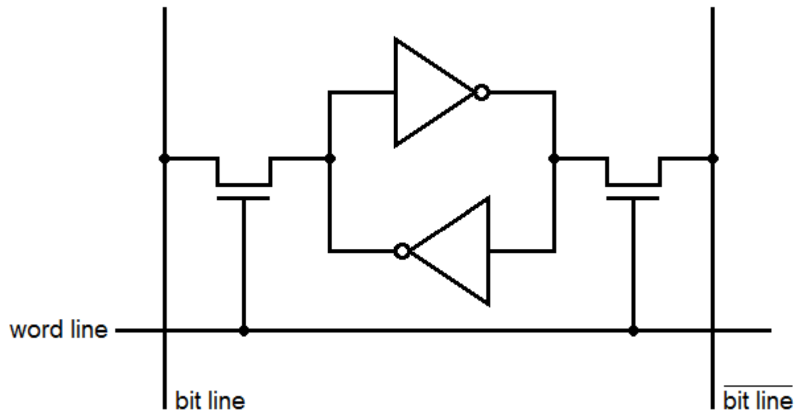
Modell eines RAM



Speicherorganisation

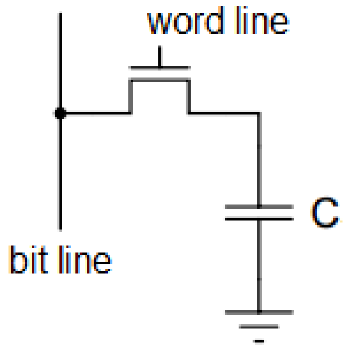


- **Statischer RAM (SRAM)**



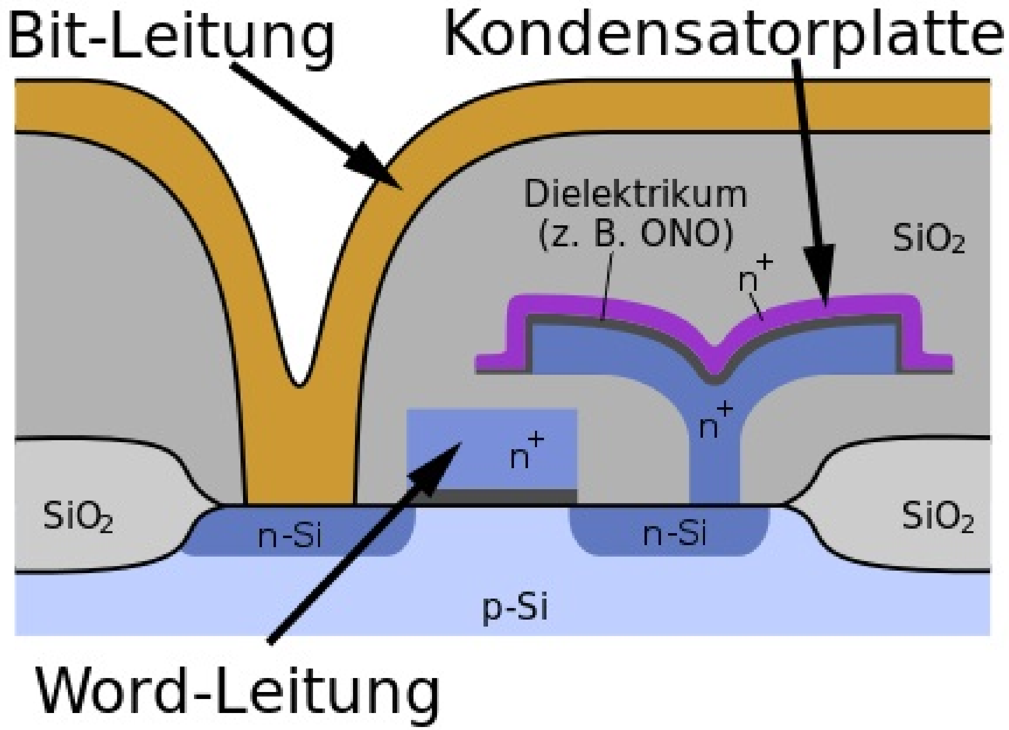
  - 6 Transistoren
  - schnell
  - für Cache-Speicher

- **Dynamischer RAM (DRAM)**



  - 1 Transistor, 1 Kapazität
  - langsam, hoher Energieverbrauch, zerstörendes Lesen, Refreshing erforderlich
  - für Hauptspeicher

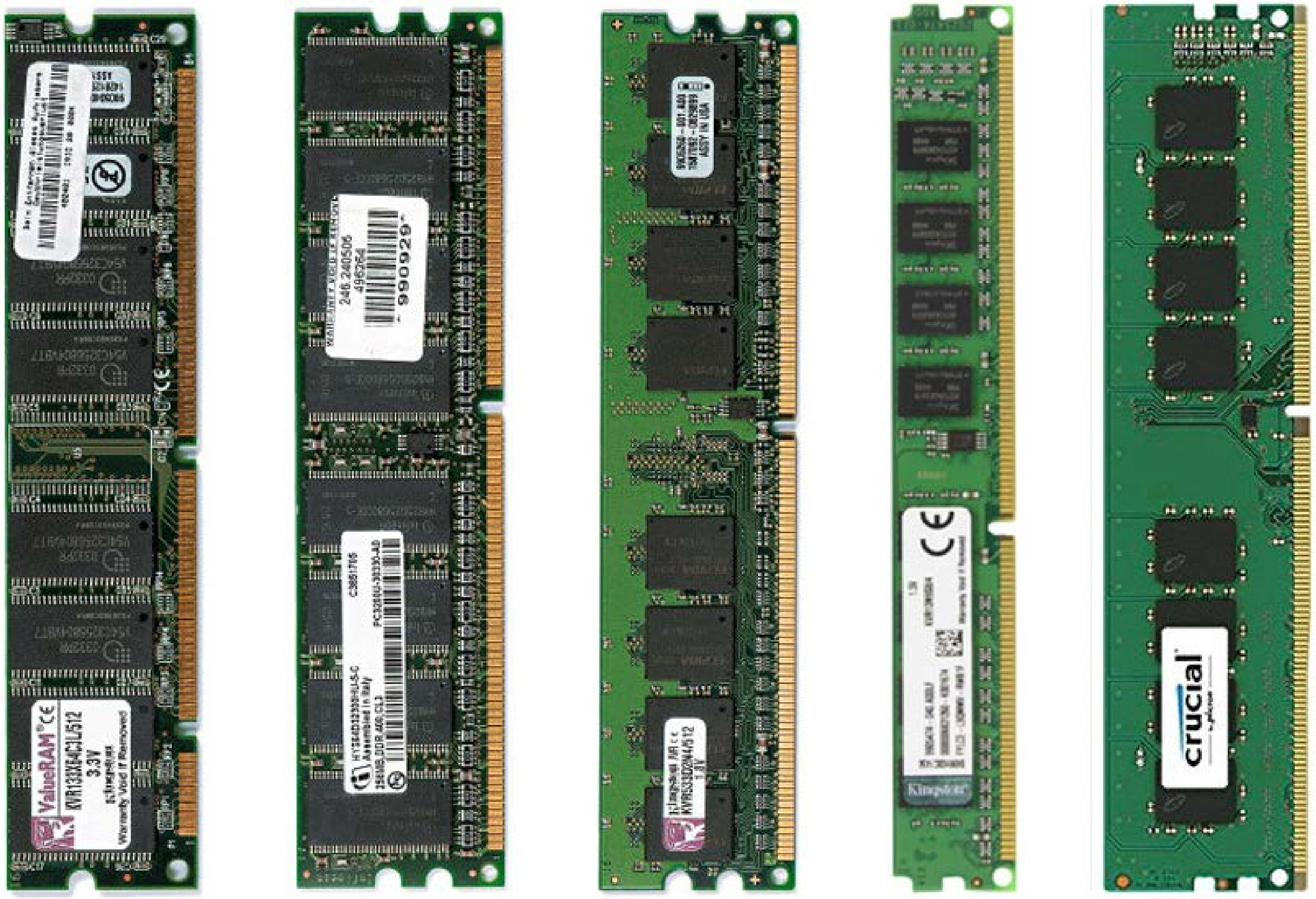
1-Transistor-DRAM-Zelle
„Stacked Capacitor“-Technik



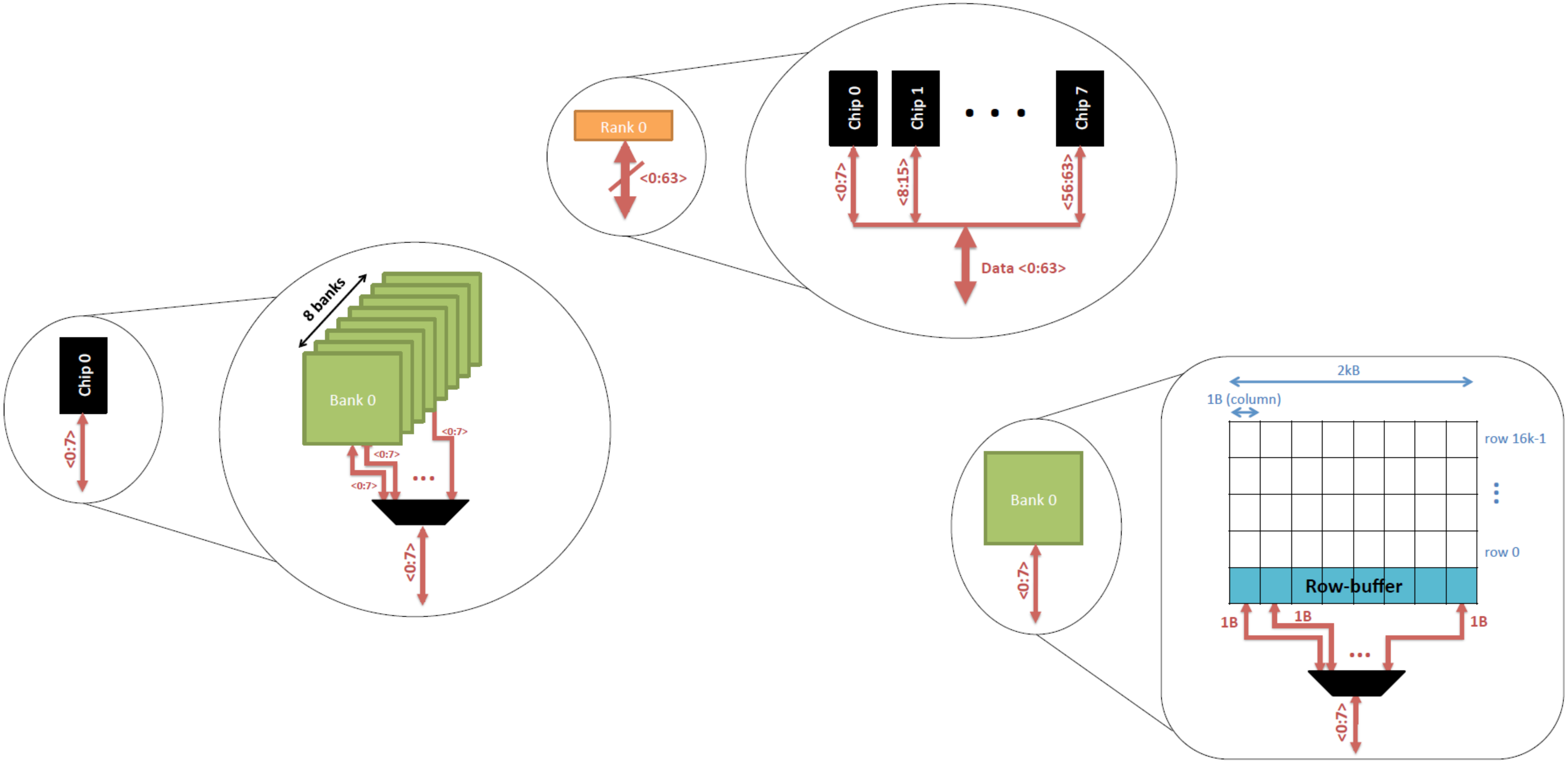
RAM-Varianten

DIMM – Dual In-line Memory Module

SDRAM DDR DDR2 DDR3 DDR4



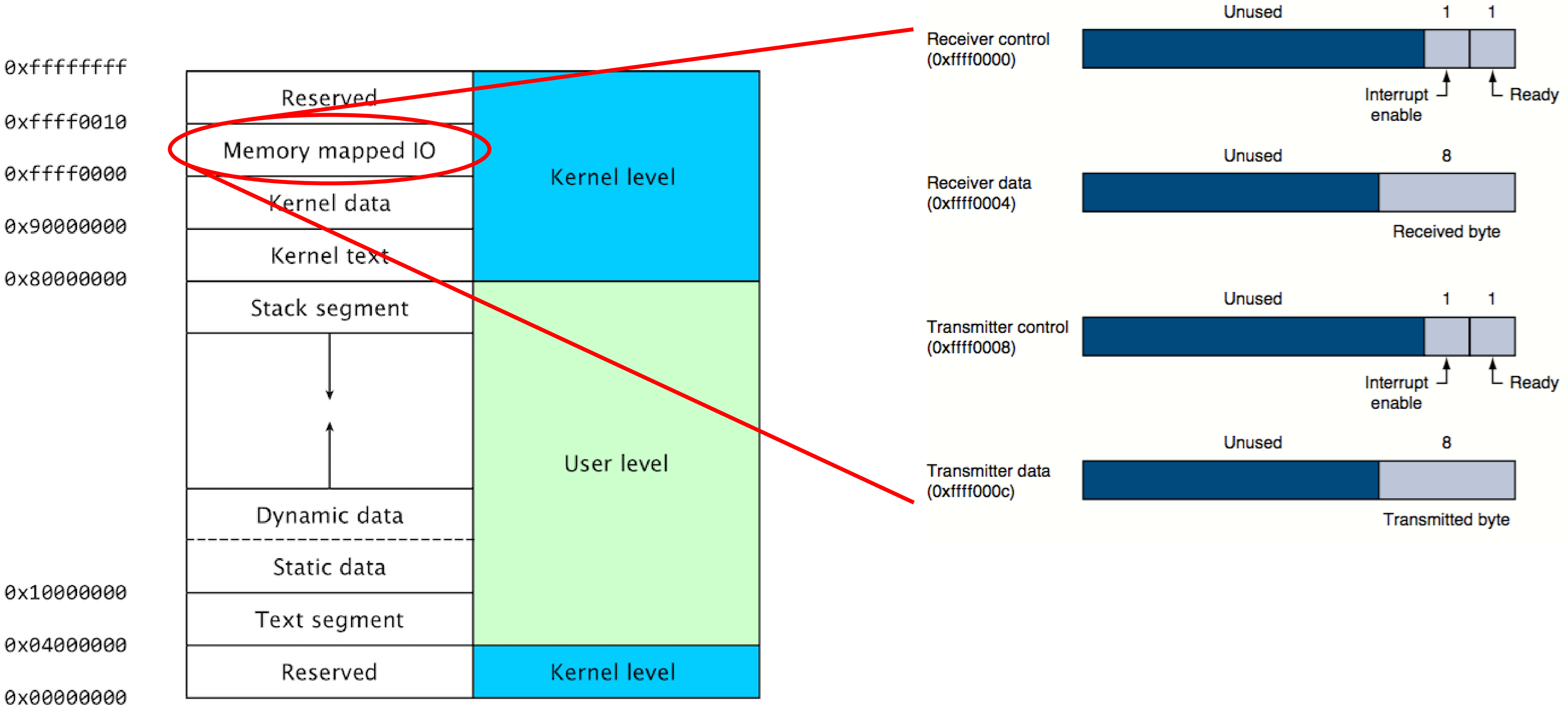
Aufgliederung eines DIMM-Speichermoduls



http://www.ece.cmu.edu/~ece740/f10/doku.php?id=lectures

Memory-mapped I/O

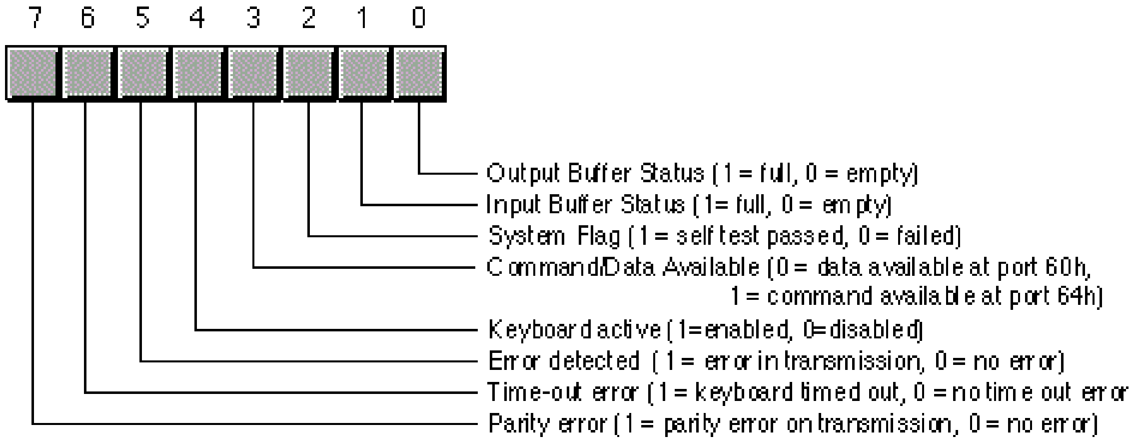
speicherbezogene E/A-Register-Adressierung



Programmgesteuerte, Port-basierte Ein-/Ausgabe

Intel 8042 Tastaturcontroller für serielle PS/2-Schnittstelle

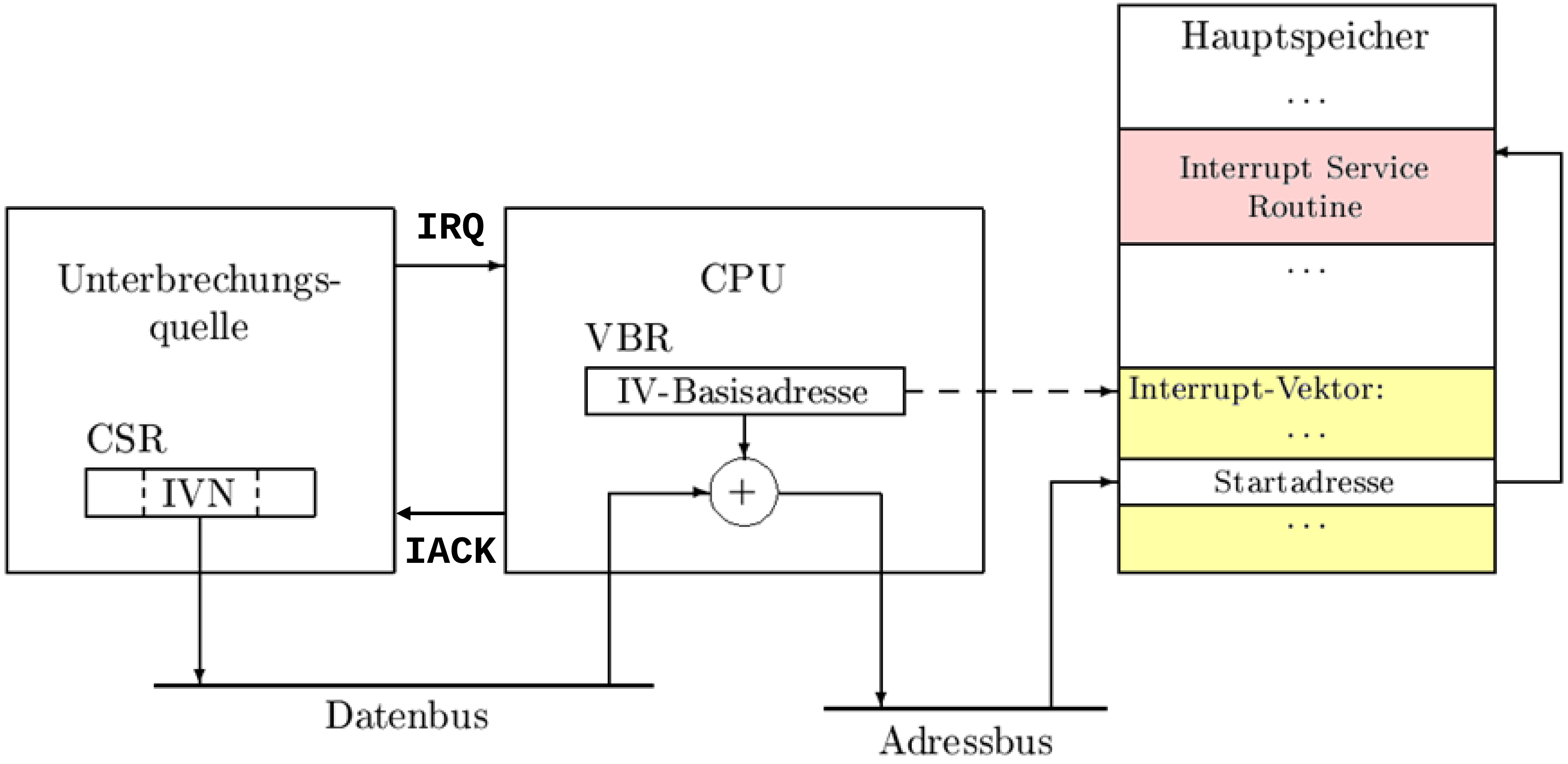
- **Beispiel:** Abfrage des Tastatur-Controllers
  - DR: Port 60h
  - CSR: Port 64h



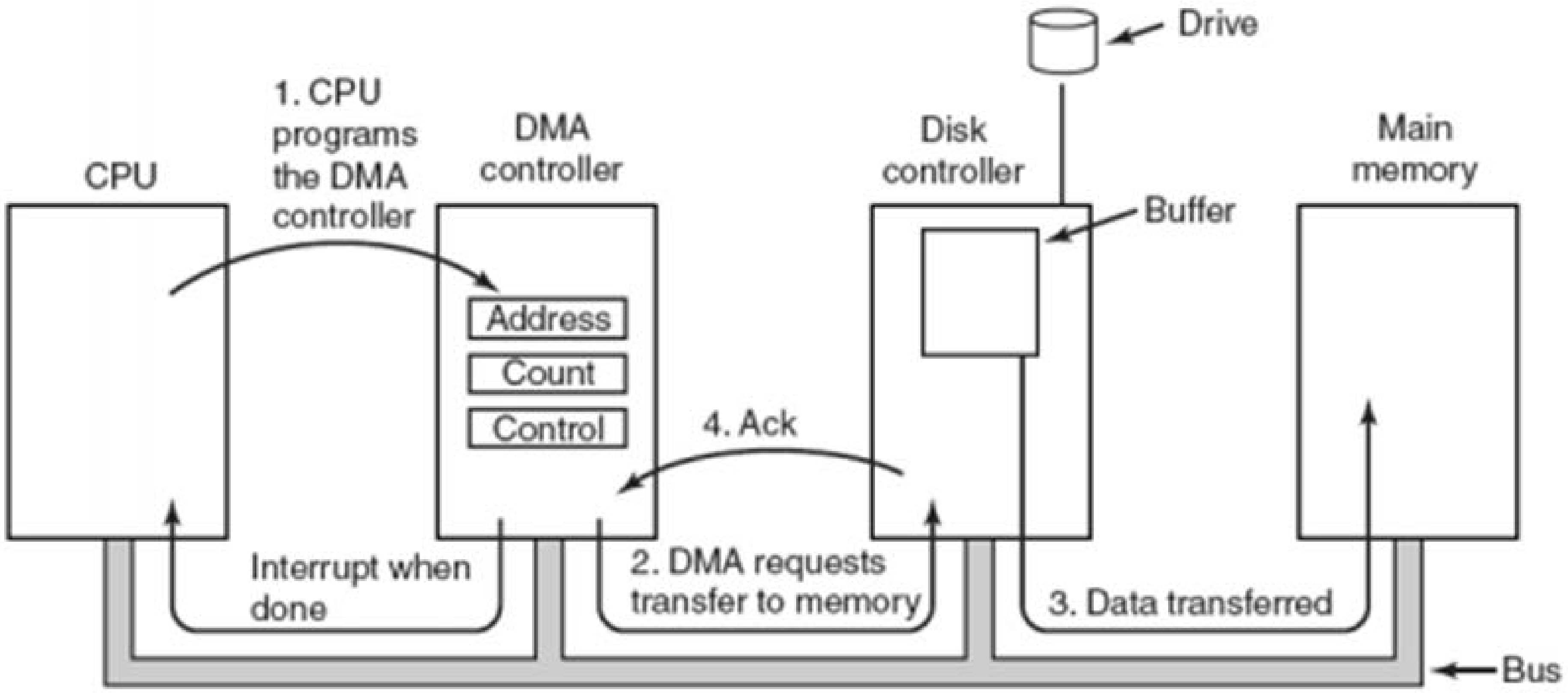

```
kbRead:   in    al, 64h     ; read status byte
          test  al, 1       ; test OUTB flag
          jz    kbRead      ; wait for OUTB = 1
          in    al, 60h     ; read data byte
          ...
```

Unterbrechungen

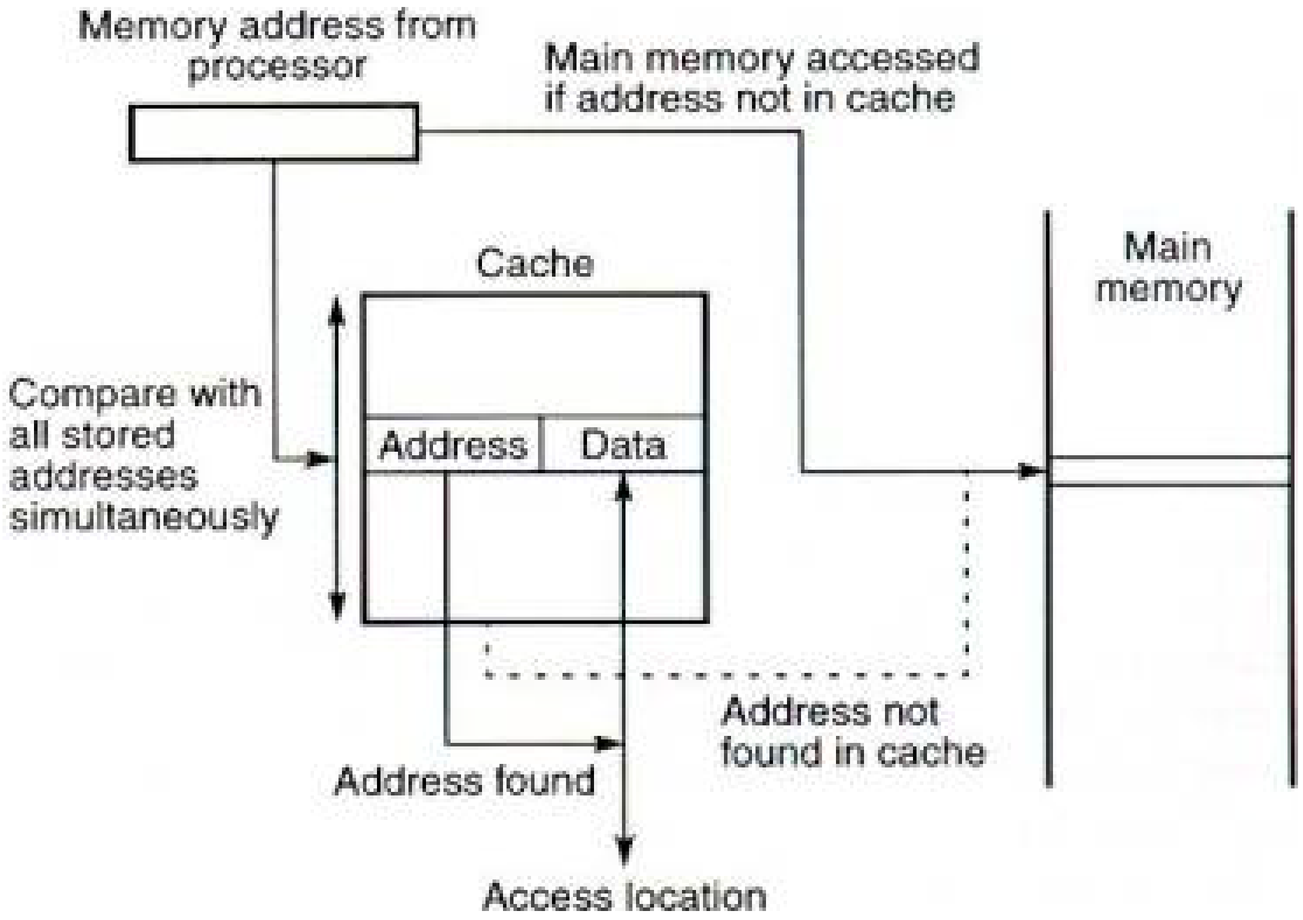
Berechnung der Einsprungadresse von Unterbrechungsroutinen



DMA – Direct Memory Access
Ablauf eines DMA-Transfers



Cache-Speicher
Prinzipieller Aufbau

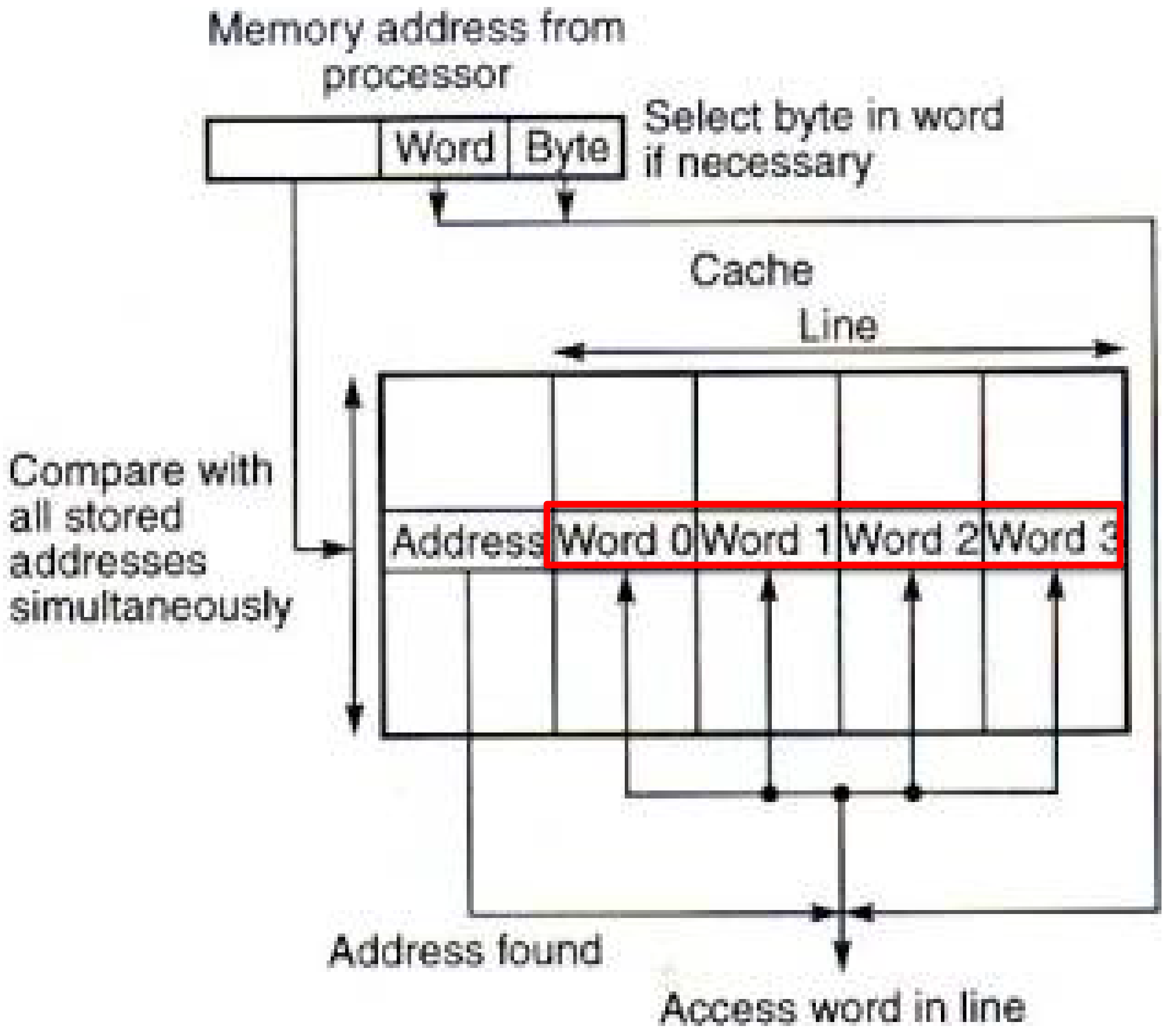


- **Lesezugriff** versucht zuerst im Cache zu lesen
  - *hit* → ✓
  - *miss* → Zugriff auf Hauptspeicher, Kopie in Cache
- **Schreibzugriff**
  - *miss* →
    - ***write-around*** (auch *write-no-allocate*): Schreiben *nur* in den Hauptspeicher
    - ***write-allocate***: Schreiben in Hauptspeicher, Kopie in Cache
  - *hit* → Kohärenzproblem
    - **Durchschreibeverfahren** (engl. *write-through*): jeder Schreibvorgang wird auf beiden Speichern durchgeführt
    - **Rückschreibeverfahren** (engl. *write-back*): Daten werden zunächst nur in den Cache geschrieben und durch ein *Dirty-Bit* als geändert gekennzeichnet

      Rückschreiben erfolgt erst bei Verdrängung oder (falls auch andere Komponenten auf Hauptspeicher zugreifen dürfen) gemäß eines Cache-Kohärenz-Protokolls
  - **Praxiserfahrung**: die beste Effizienz bietet die Kombination *write-allocate/write-back*

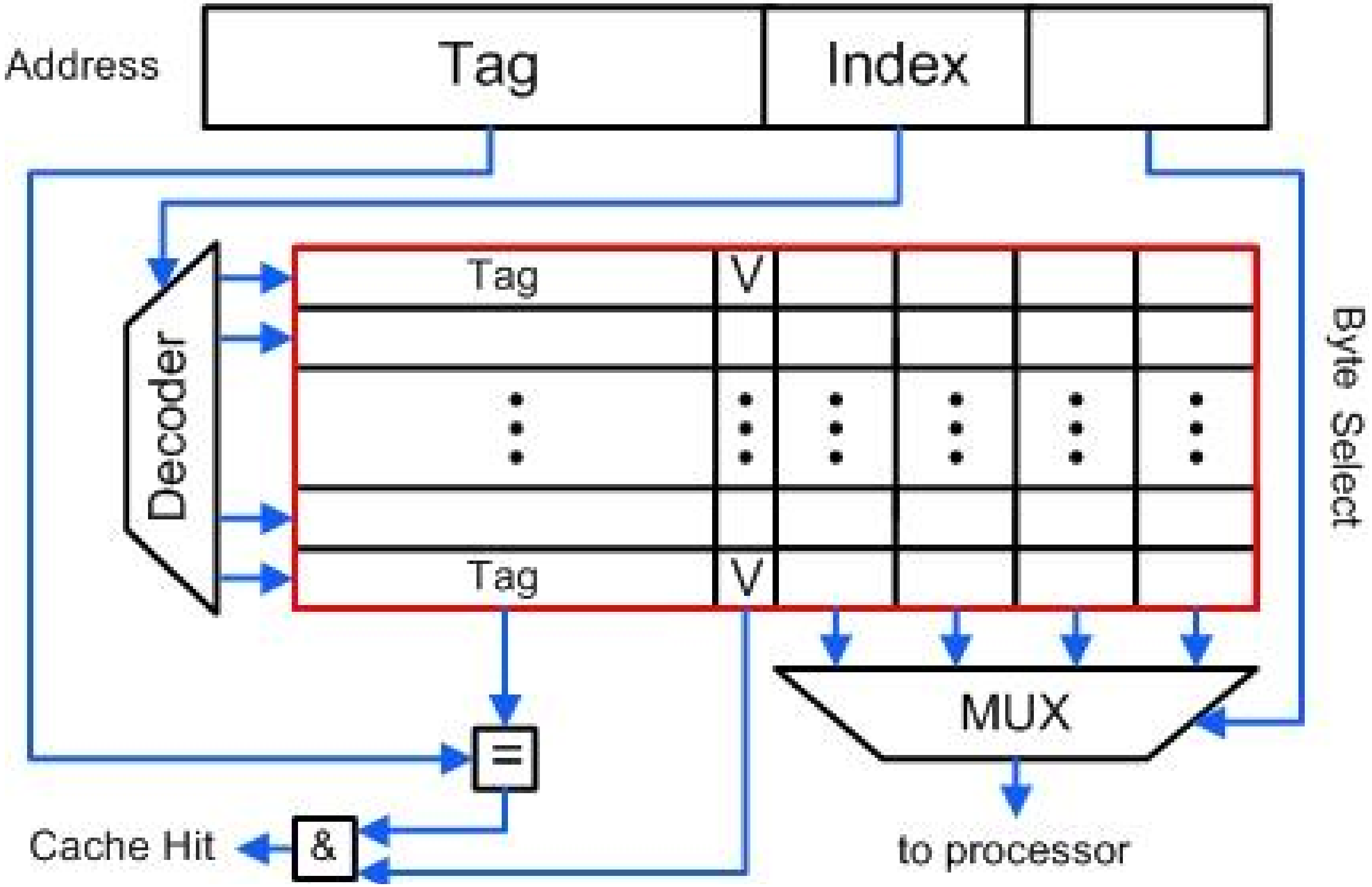
Cache-Speicher

Cache Line



Organisationsformen für Cache-Speicher

Direct mapped cache / direkt abbildender Cache

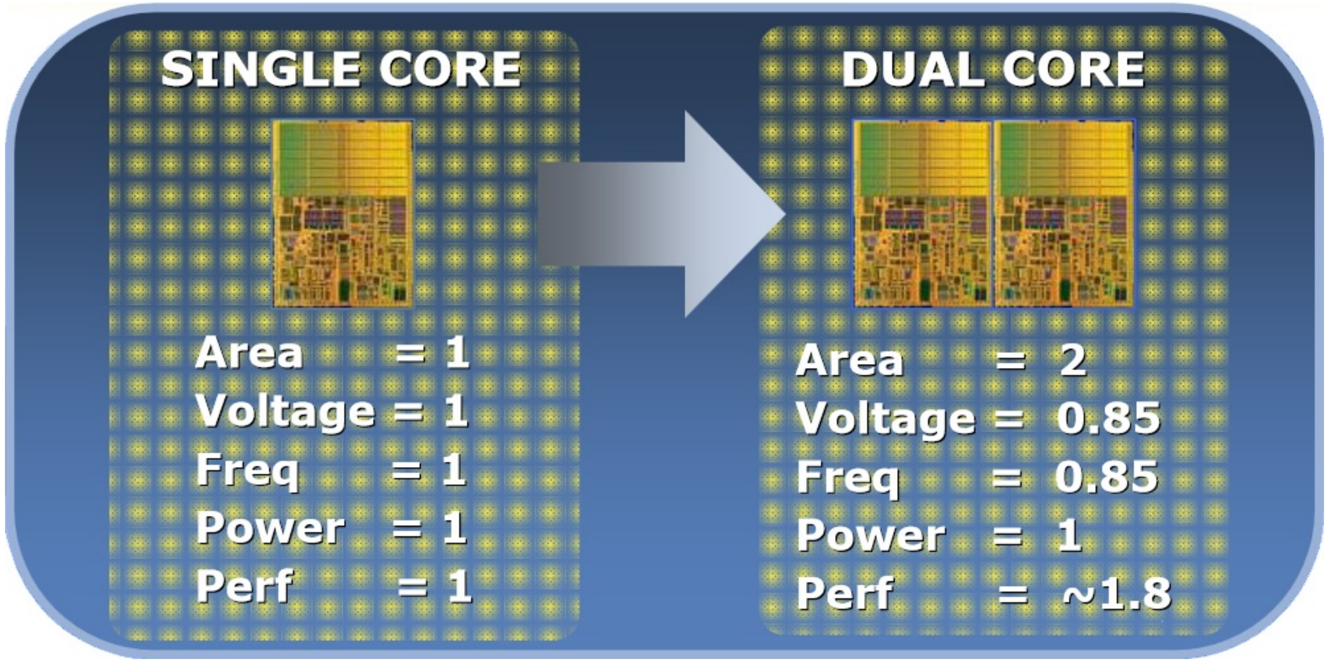


Speicherhierarchie

100 B	Register	0.4 ns
32 – 128 KB	L1-Cache	1.2 – 2 ns
256 KB – 2 MB	L2-Cache	3 – 5 ns
2 – 20 MB	L3-Cache	12 – 40 ns
1 – 100 GB	Hauptspeicher	10 – 100 ns
1 – 4 TB	Sekundärspeicher (HDD, SSD)	100 μs – 10 ms
viele TB	Archivspeicher, Cloud	100 ms – min

Vergleich Einkern-/Zweikernprozessor

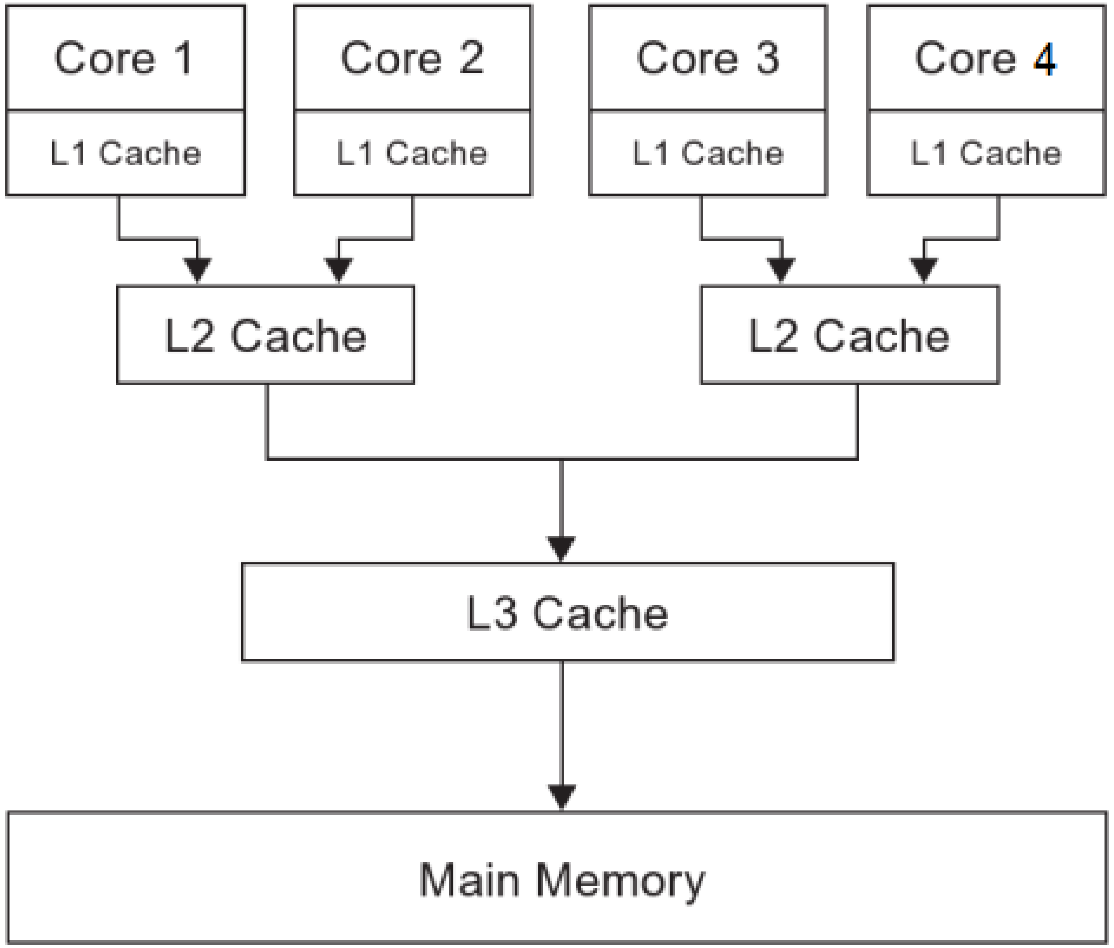
- **Elektr. Leistungsaufnahme** $P \sim C \cdot U_V^2 \cdot f$
- **Zwei Kerne, jeweils um 15% reduzierte Taktrate und Betriebsspannung**



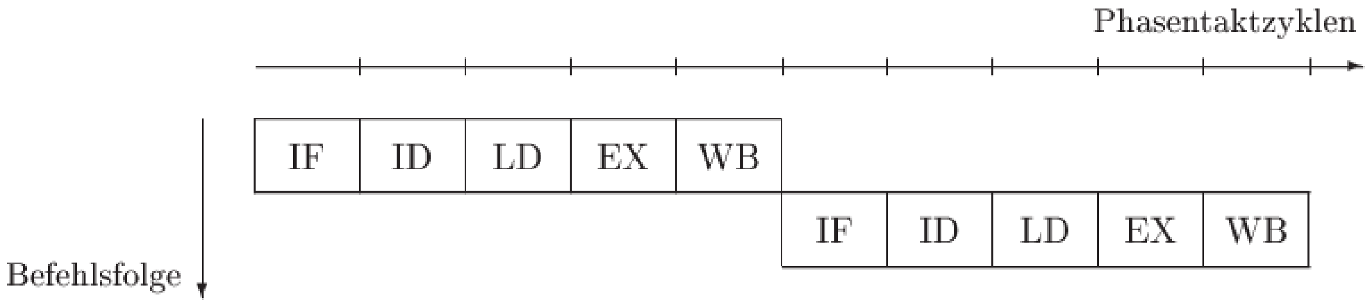
Quelle: intel.com

fast doppelte Performanz

Cache-Struktur in Mehrkernprozessoren

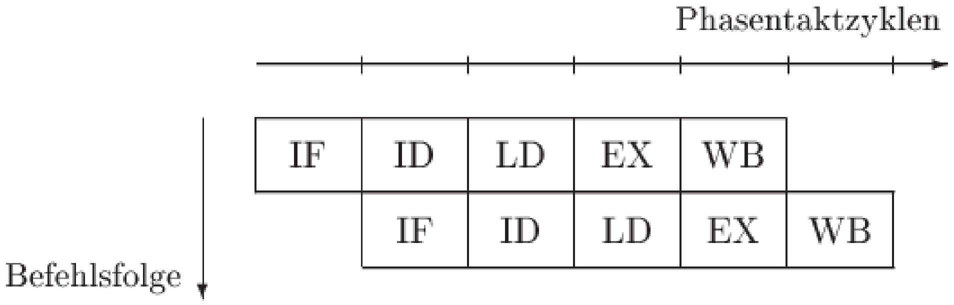


ohne Pipelining



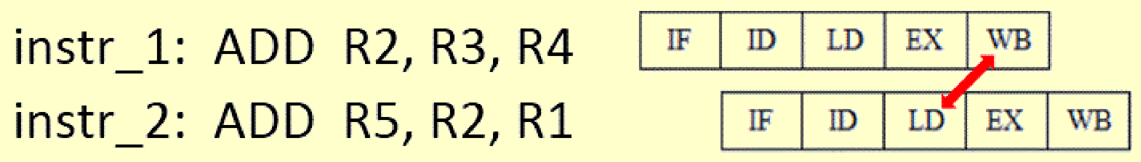
$$T_{oP} = (k \cdot n) \cdot \tau$$

mit Pipelining

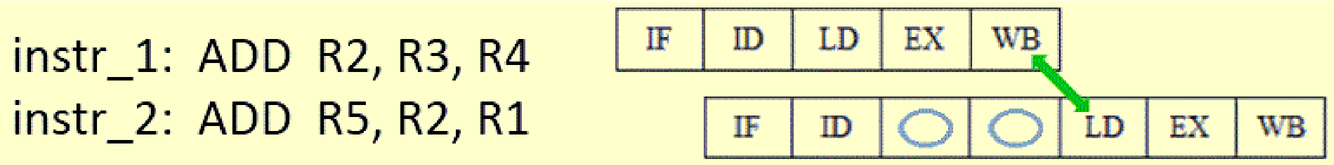


$$T_{mP} = (k + (n - 1)) \cdot \tau$$

- **RAW (Read-After-Write)-Konflikt**



- **Lösung**
  - Einfügen von Wartezyklen (*pipeline stalls*)



- **bei Sprungbefehlen**
  - **unbedingter** Sprungbefehl
    - *Delayed-branch*-Methode: Einfügen von no-ops
  - **bedingter** Sprungbefehl
    - Sprungvorhersage (*branch prediction*)