Grundlagen der Technischen Informatik 2

Sommersemester 2008

Digitaltechnik
Rechneraufbau

Prof. Dr. M. Bogdan

Technische Informatik

bogdan@informatik.uni-leipzig.de

Übersicht

- Einleitung
- Grundlagen
  - ⇨ aus TI 1
- Schaltnetze
  - ⇨ KV-Diagramme
  - ⇨ Minimierung nach Quine MC-Cluskey
- Speicherglieder
  - ⇨ RS-Flipflop
  - ⇨ D-Flipflop
  - ⇨ JK-Flipflop
  - ⇨ T-Flipflop

Übersicht

- Schaltwerke
  - ⇨ Darstellung endlicher Automaten
  - ⇨ Minimierung der Zustandszahl
  - ⇨ Einfluss der Zustandskodierung
- Spezielle Schaltnetze und Schaltwerke
  - ⇨ Multiplexer, Demultiplexer, Addierer
  - ⇨ Register, Schieberegister, Zähler
- Rechnerarithmetik
  - ⇨ Formale Grundlagen
  - ⇨ Addition und Subtraktion
  - ⇨ Multiplikation und Division
  - ⇨ Arithmetisch-Logische Einheit (ALU)

Übersicht

- Ein minimaler Rechner
  - ⇨ Befehlssatz
  - ⇨ Realisierung
  - ⇨ Arbeitsweise und Programmierung
- Aufbau von Rechnersystemen
  - ⇨ Komponenten eines Rechnersystems
  - ⇨ Prinzipieller Aufbau eines Mikroprozessors
  - ⇨ Steuerwerk und Mikroprogrammierung
  - ⇨ Rechenwerk
  - ⇨ Das Adresswerk

Übersicht

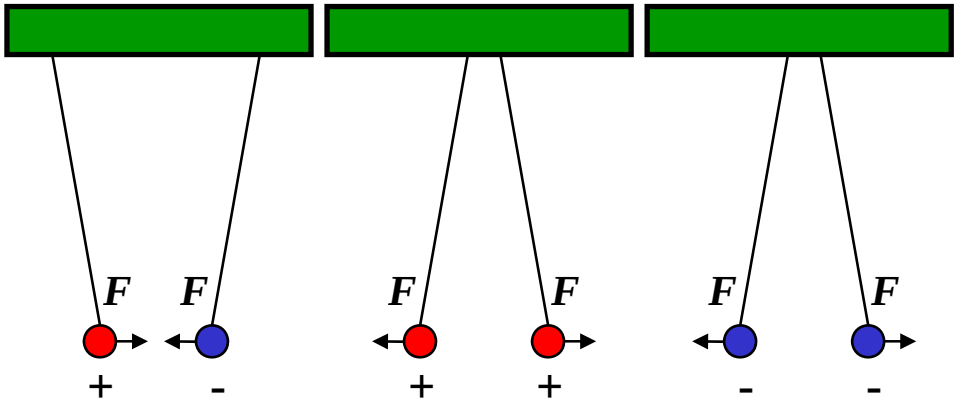
- Rechner- und Gerätebusse
  - ⇨ interne Busse
  - ⇨ externe Busse
- E/A-Steuerungen
  - ⇨ Prinzip der Datenein-und -ausgabe
  - ⇨ Parallele Schnittstellen
  - ⇨ Serielle Schnittstellen
  - ⇨ Analoge Ein- und Ausgabe
- Peripheriegeräte
  - ⇨ Tastatur
  - ⇨ Graphikadapter
  - ⇨ Festplatten- und Diskettenlaufwerke
  - ⇨ Sonstige E/A-Geräte

Literatur

Die Vorlesung basiert auf den Lehrbüchern:

- **U. Tietze, C. Schenk, E. Gamm: „Halbleiter Schaltungstechnik“, Springer –Verlag (2002)**
- **W. Schiffmann, R. Schmitz: „Technische Informatik 1**
  Grundlagen der digitalen Elektronik“
  Springer-Lehrbuch, Springer-Verlag (1992)
- **W. Schiffmann, R. Schmitz: „Technische Informatik 2**
  Grundlagen der Computertechnik“
  Springer-Lehrbuch, Springer-Verlag (1992)
- **H. Bähring: „Mikrorechnersysteme“**
  Springer Lehrbuch, Springer-Verlag (1994)

Elektrische Ladung und elektrisches Feld (Wdh.)



- **Elektrische Ladungen üben Kräfte aufeinander aus**

  ⇨ **ungleiche Ladungen ziehen sich an**

  ⇨ **gleiche Ladungen stoßen sich ab**

Elektrische Ladung und elektrisches Feld (Wdh.)

Elektrische Ladung

- Die Einheit der elektrischen Ladung ist

  1 C = 1 Asec

- Die elektrische Ladung eines Elektrons beträgt

$$e_0 = 1{,}602 \cdot 10^{-19} C$$

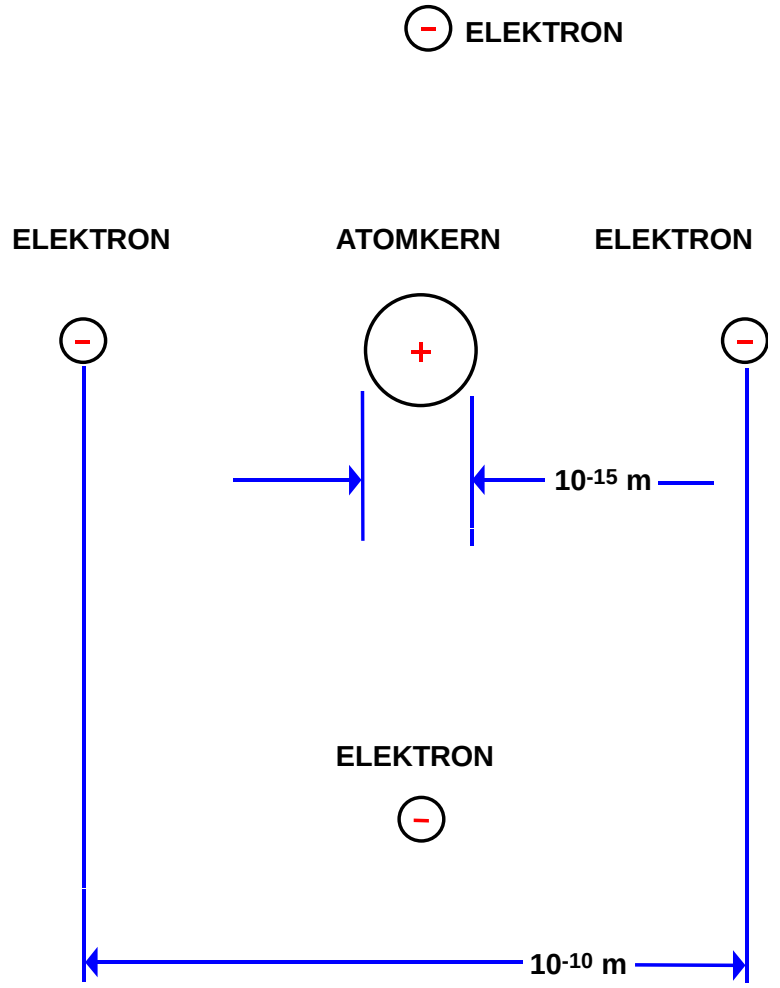
- Man benötigt

  6,242 * 10^{18} Elektronen

  um die Ladung 1 C zu erhalten

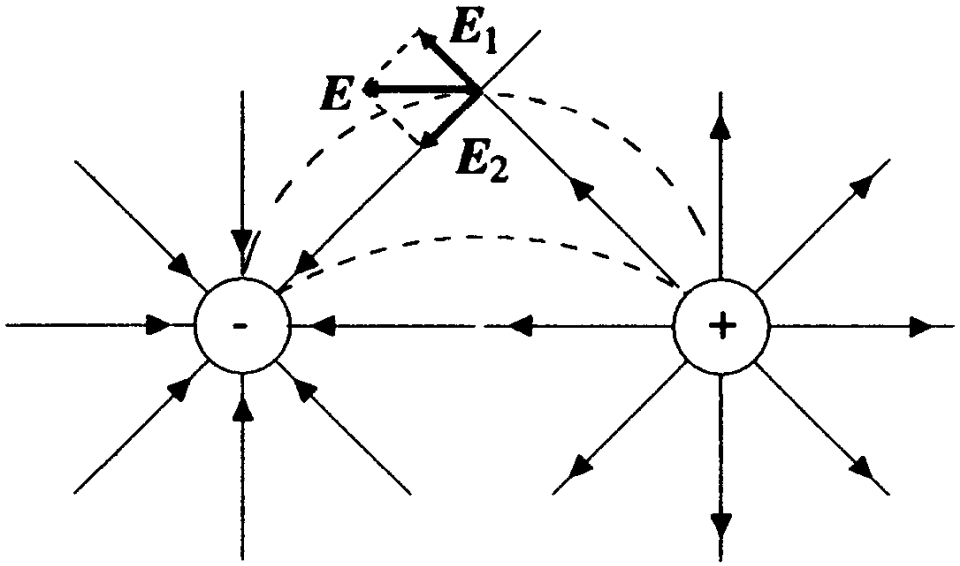
- Die Ladungsmenge Q ist das Vielfache der Elementarladung

$$Q = n \cdot e_0$$



Das elektrische Feld (Wdh.)

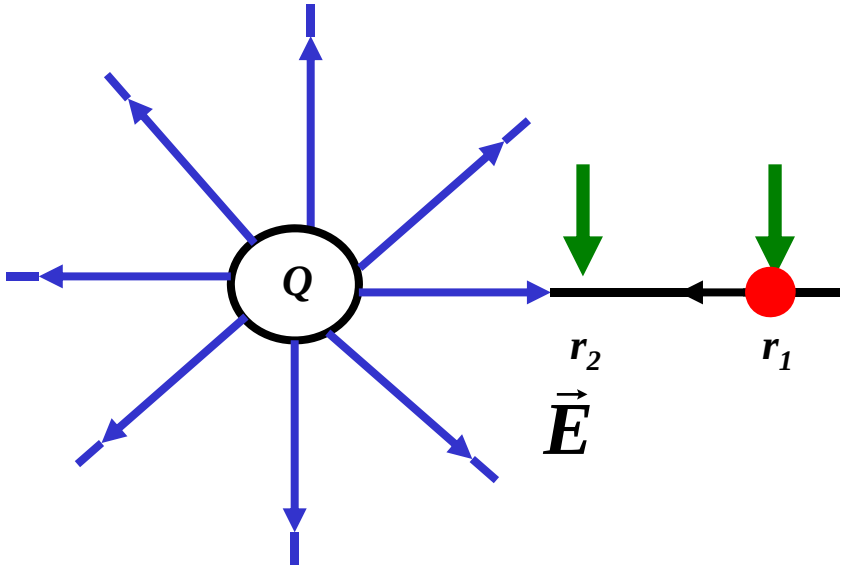
- **Feldlinien dienen zur Veranschaulichung des elektrischen Feldes**
  - ⇨ **sie zeigen immer in Richtung der wirkenden Kraft,**
  - ⇨ **sie erfüllen den Raum kontinuierlich,**
  - ⇨ **sie verlaufen von einer positiven zu einer negativen Ladung,**
  - ⇨ **sie sind nicht geschlossen.**



Elektrische Felder überlagern sich additiv

$$\vec{E} = \sum_{n=1}^{N} \vec{E}_n$$

Potential und Spannung (Wdh.)



- Das elektrische Potential ist eng verbunden mit dem Begriff Arbeit. Physikalische Arbeit ergibt sich als Kraft mal Weg

$$W = \vec{F} \cdot \Delta\vec{r}$$

- Im elektrischen Feld wirkt auf eine Ladung q die Kraft

$$\vec{F} = q \cdot \vec{E}$$

- Damit beträgt die Arbeit um eine Ladung q von r_1 nach r_2 zu bewegen

$$W_{1,2} = \int_{r_1}^{r_2} \vec{F} \cdot d\vec{r}$$

Die elektrische Spannung (Wdh.)

- **Verschiebt man eine Ladung von P 1 nach P 2, so muß die Arbeit $W_{1,2}$ aufgebracht werden**

$$W_{1,2} = \int_{r_1}^{r_2} \vec{F}\,dr = q\int_{r_1}^{r_2} \vec{E}\,dr$$

⇨ **Die Spannung ist**

$$\vec{U}_{1,2} = \int_{r_1}^{r_2} \vec{E}\,dr$$

⇨ **Es ergibt sich allgemein**

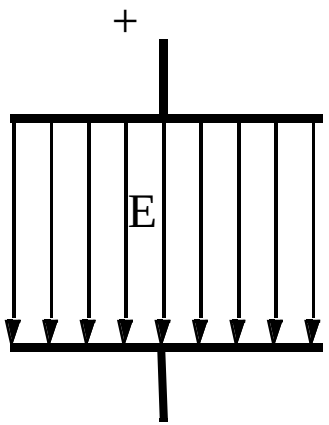
$$W = U \cdot q \Rightarrow U = \frac{W}{q} \qquad \text{Spannung} = \frac{\text{Arbeit}}{\text{Ladung}}$$

⇨ **Die Einheit der Spannung ist**

$$1\,\text{V} = 1\,\frac{\text{Nm}}{\text{C}}$$

Elektrische Ladung und Leiter (Wdh.)

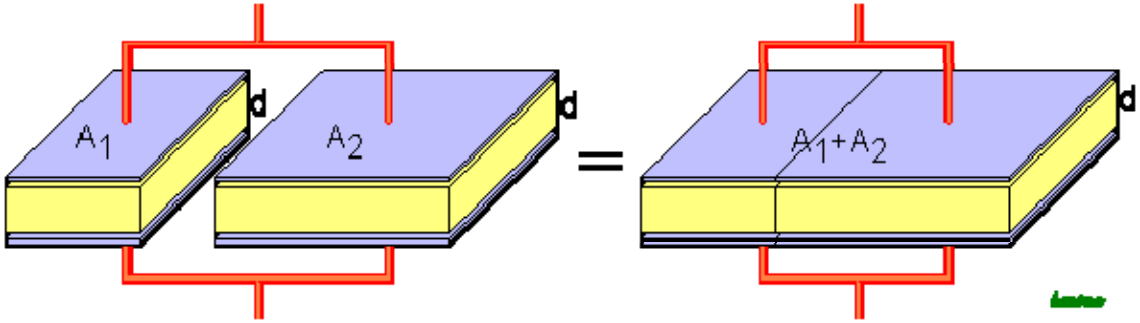
- **Auf metallischen Leitern sind elektrische Ladungen frei beweglich und verteilen sich aufgrund der Abstoßung gleichmäßig auf der gesamten Oberfläche.**
- **Feldlinien des elektrischen Feldes sind senkrecht zur Oberfläche gerichtet. Das Innere eines metallischen Hohlraumes ist ein feldfreier Raum (Faraday'scher Käfig).**
- **Eine parallele Anordnung zweier Metallflächen (Elektroden), von denen eine positiv, die andere negativ geladen ist, heißt Kondensator.**



Elektrisches Feld eines geladenen Plattenkondensators

Kondensator (Wdh.)

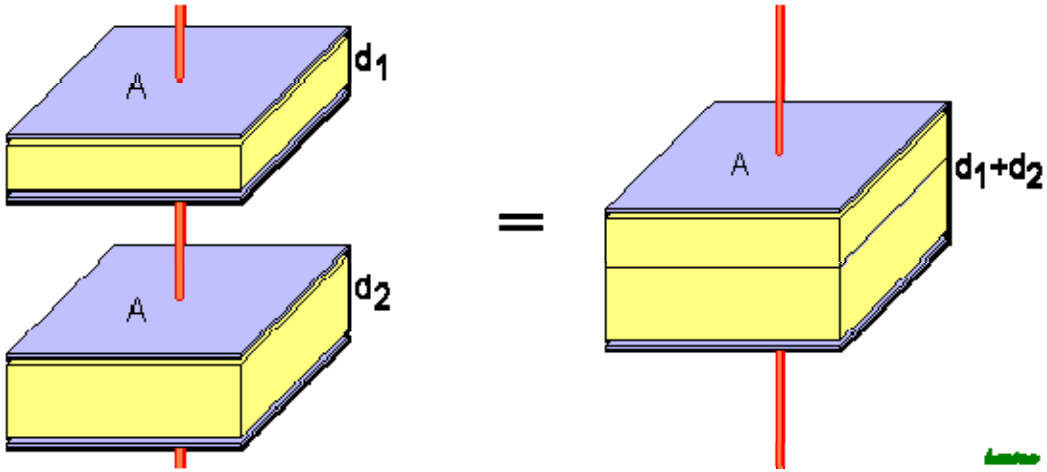
Parallelschaltung



$$C_{ges} = \sum_{n=1}^{N} C_n$$

Bilder: Wikipedia.de

Reihenschaltung



$$C_{ges} = \frac{1}{\sum_{n=1}^{N} \frac{1}{C_n}}$$

Der elektrische Strom (Wdh.)

- **Ist der Ladungstransport in eine Richtung und gleichmäßig, dann sprechen wir von Gleichstrom.**
- **Zusammenhang zwischen Stromstärke, Spannung und Widerstand wird durch das Ohmsche Gesetz und die Kirchhoffschen Gesetz beschrieben.**
- **Elektrischer Strom ist der Fluss von Elektronen**
- **Die Stromstärke I entspricht der bewegten Ladungsmenge pro Zeiteinheit**

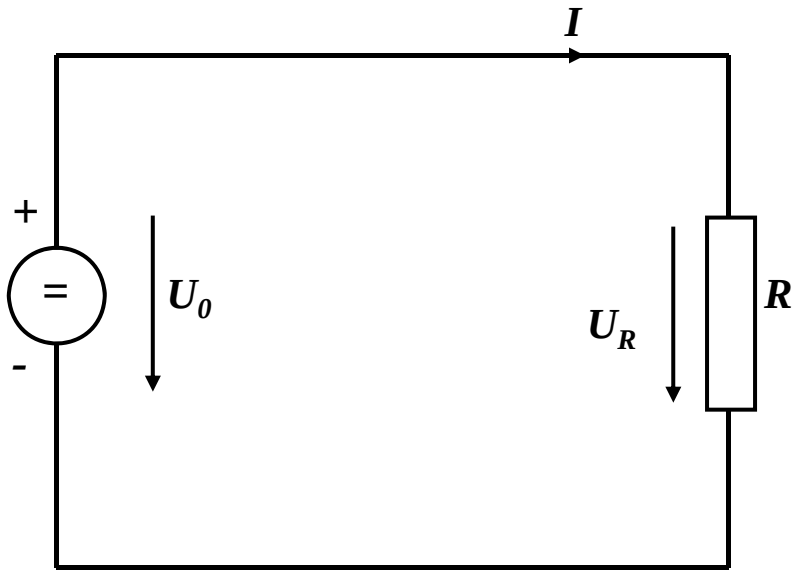
$$I = \frac{Q}{t}$$

- **Fließen durch einen Leiter pro Sekunde *n* Coulomb [C], so messen wir einen Strom von *n* Ampere [A]**

$$1\,\mathrm{A} = 1\frac{\mathrm{C}}{\mathrm{s}} = \frac{1}{1{,}602}\cdot 10^{19}\,\frac{\text{Elektronen}}{\mathrm{s}}$$

Elektrischer Stromkreis (Wdh.)

- **Ein elektrischer Stromkreis ist eine Anordnung aus**
  - ⇨ **Spannungsquelle (Stromquelle)**
  - ⇨ **Verbraucher *R***
  - ⇨ **Verbindungsleitungen**
- **In der Spannungsquelle wird Energie aufgewendet**
  - ⇨ **(W< 0)**
- **In R wird Energie verbraucht**
  - ⇨ **(W> 0)**
- **Der elektrische Strom fließt (per Definition) von Plus (+) nach Minus (-)**
- **Die Elektronen fließen von Minus (-) nach Plus (+)**
- **Die Spannungsquelle bewirkt im Verbraucher R einen Stromfluss von von Plus nach Minus (Pfeilrichtung)**



Ohmsches Gesetz (Wdh.)

- **Es gibt einen festen Zusammenhang zwischen dem Strom *I* und der Spannung *U***
  - ⇨ **Ohmsches Gesetz**

$$I = \frac{1}{R} \cdot U$$

$$U = R \cdot I$$

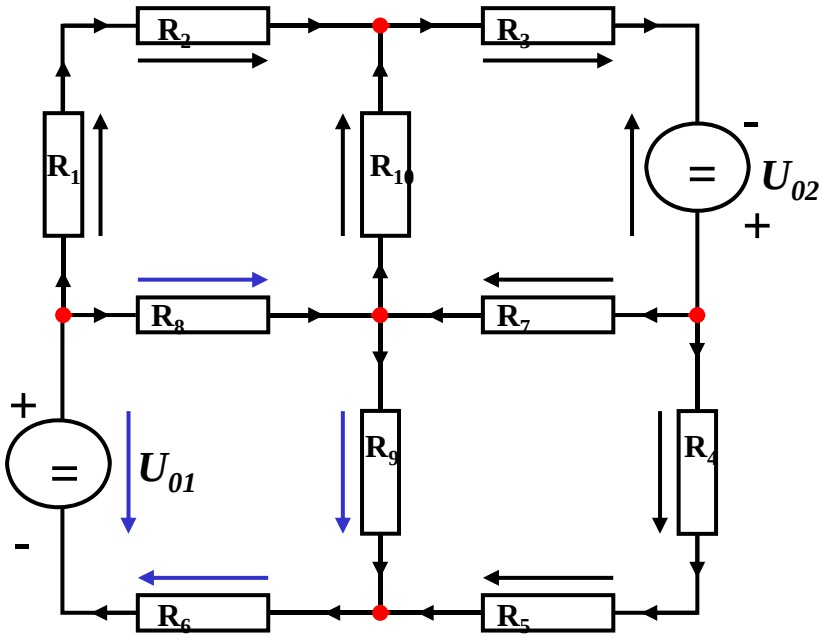
$$R = \frac{U}{I}$$

- **Die Einheit für den Widerstand ist Ohm Ω**

$$1\Omega = 1\frac{\mathrm{V}}{\mathrm{A}}$$

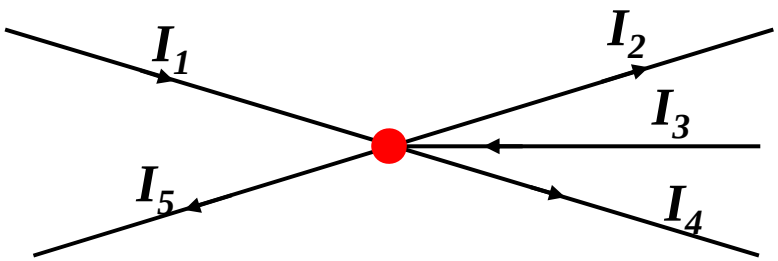
Die kirchhoffschen Sätze

- Nur selten wird an einer Spannungsquelle nur ein einzelner Verbraucher *R* angeschlossen
- Eine Anordnung aus Spannungsquellen und Verbrauchern heißt Netz
- Es besteht aus Knoten und Maschen
  - ⇨ **Knoten**: Verzweigungspunkte
  - ⇨ **Masche**: Pfad, bei dem kein Knoten mehrfach durchlaufen wird
- Richtung der Pfeile (Vorzeichen)
  - ⇨ Spannung ist von Plus nach Minus gerichtet
  - ⇨ Strom fließt von Plus nach Minus



Knotenregel (1. kirchhoffscher Satz)

- In einem **Knoten** ist die Summe aller Ströme Null
  - ⇨ An keiner Stelle des Netzes werden Ladungen angehäuft



- Definition der Stromrichtung für die mathematische Formulierung
  - ⇨ zufließende Ströme werden mit einem **positiven** Vorzeichen behaftet
  - ⇨ abfließende Ströme werden mit einem **negativen** Vorzeichen behaftet

$$0 = I_1 - I_2 + I_3 - I_4 - I_5$$

oder

$$I_2 + I_4 + I_5 = I_1 + I_3$$

allgemein

$$\sum_{i=0}^{n} I_i = 0$$

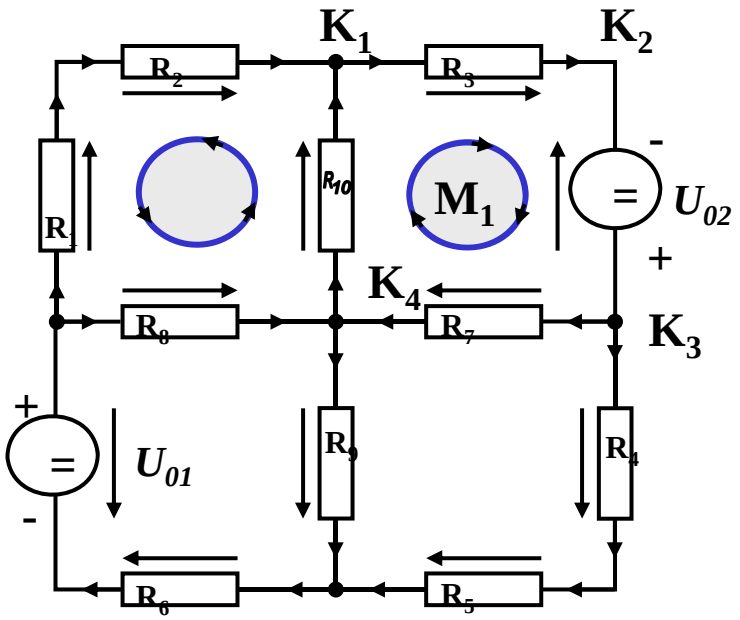
Maschenregel (2. kirchhoffscher Satz)

- **Bei einem geschlossenen Umlauf einer Masche ist die Summe aller Spannungen Null**
  - **die Spannungsquellen erzeugen die Spannungen U_{01} und U_{02}**
  - **durch die Widerstände fließt ein Strom**
  - **nach dem Ohmschen Gesetz gilt für die Spannung**

$$U = R \cdot I$$

  - **die Knotenpunkte K_1, K_2, K_3 und K_4 können deshalb ein unterschiedliches Potenzial besitzen**

$$\sum_{i=0}^{m} U_i = 0$$

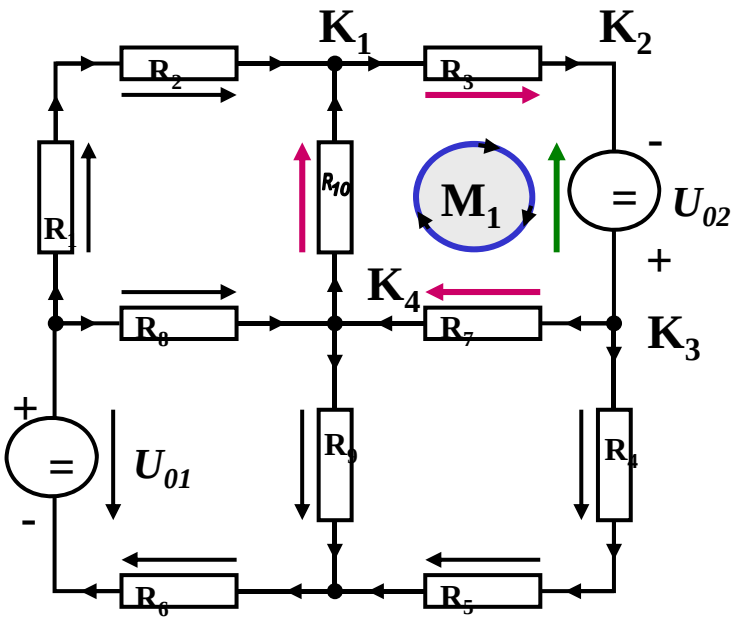


Maschenregel (2. kirchhoffscher Satz)

- **Werden die Knotenspannungen addiert, so folgt:**

$$U_{K12} + U_{K23} + U_{K34} + U_{K14} = 0$$

- **Vorzeichen der Spannung**
  - ⇨ **die Spannungsrichtung der Quellen ist vorgegeben (von + nach -)**
  - ⇨ **Umlaufrichtung der Masche wird festgelegt**
  - ⇨ **Spannungspfeile mit der Umlaufrichtung werden positiv gezählt**
  - ⇨ **Spannungspfeile gegen die Umlaufrichtung werden negativ gezählt**



$$U_{K12} - U_{02} + U_{K34} + U_{K14} = 0$$

$$U_{K12} + U_{K34} + U_{K14} = U_{02}$$

Anwendung Knotenregel

Sie haben einen neuen Personal Computer gekauft.

Sie benutzen ein Strommeßgerät (Ampere-Meter) und stellen damit fest, dass die 5 Volt Stromversorgung Ihres PC im eingeschalteten Zustand 4,0 A liefert. Versorgt wird damit die Hauptplatine, das Festplattenlaufwerk und das DVD-Laufwerk.

Sie messen, dass der Strom in die Hauptplatine 2,2 A beträgt und der Strom in die Festplatte 1,0 A.

An das DVD-Laufwerk kommen sie nicht heran.

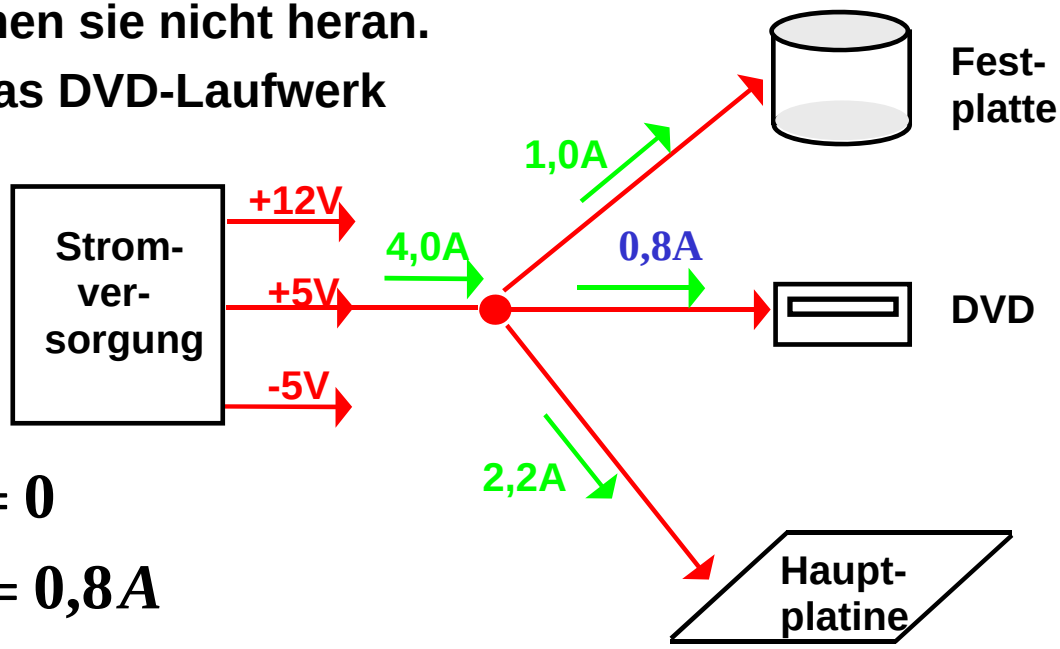
➔Wieviel Strom bekommt das DVD-Laufwerk bei der Spannung 5 V?

$$\sum_{n=1}^{4} I_n = 0$$

$$I_{ein} - I_{Fest} - I_{HP} - I_{DVD} = 0$$

$$4{,}0A - 1{,}0A - 2{,}2A - I_{DVD} = 0$$

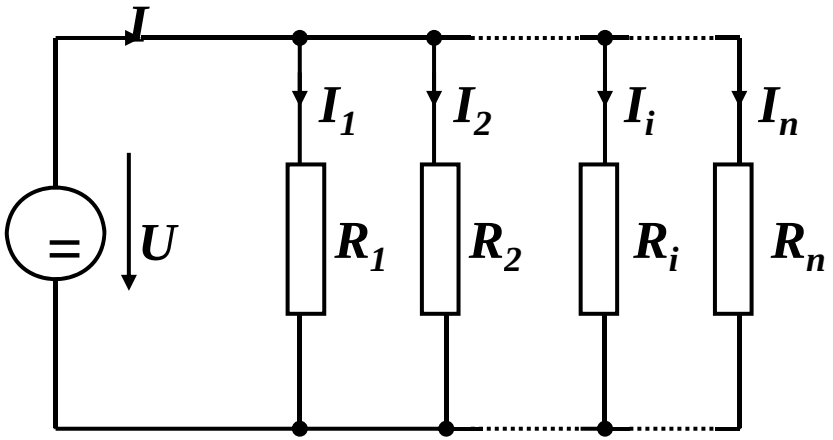
$$4{,}0A - 1{,}0A - 2{,}2A = I_{DVD} = 0{,}8A$$



Parallelschaltung von Widerständen (Wdh.)

- Für die Teilströme I_1, I_2,...,I_n gilt:

$$I_1 = \frac{U}{R_1}, I_2 = \frac{U}{R_2}, ..., I_n = \frac{U}{R_n}$$



- Nach Knotenregel:

$$I_{ges} = I_1 + I_2 + ... + I_n$$

$$= \frac{U}{R_1} + \frac{U}{R_2} + ... + \frac{U}{R_n}$$

$$= U \cdot \left(\frac{1}{R_1} + \frac{1}{R_2} + ... + \frac{1}{R_n} \right)$$

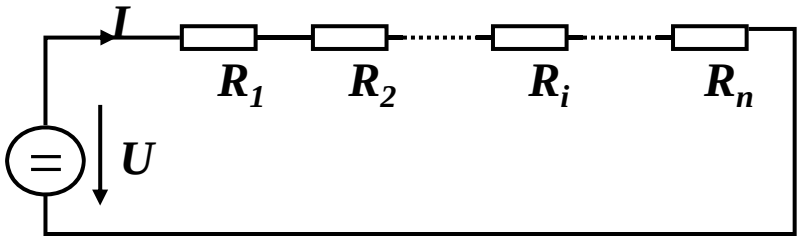
- Der Ersatzwiderstand der gesamten Schaltung berechnet sich durch:

$$\frac{1}{R_{gesamt}} = \frac{1}{R_1} + \frac{1}{R_2} + ... + \frac{1}{R_n} = \sum_{k=1}^{n} \frac{1}{R_k}$$

Reihenschaltung von Widerständen (Wdh.)

- Für die Spannungen U_1, U_2,...,U_n an den Widerständen gilt:

$$U_1 = I \cdot R_1, U_2 = I \cdot R_2, ..., U_n = I \cdot R_n$$



- Nach Maschenregel:

$$\begin{aligned} U &= U_1 + U_2 + ... + U_n \\ &= I \cdot R_1 + I \cdot R_2 + ... + I \cdot R_n \\ &= I \cdot (R_1 + R_2 + ... + R_n) \end{aligned}$$

- Der Ersatzwiderstand der gesamten Schaltung berechnet sich durch:

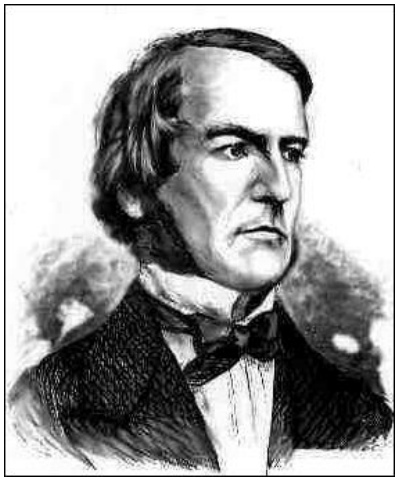
$$R_{gesamt} = R_1 + R_2 + ... + R_n = \sum_{k=1}^{n} R_k$$

Schaltnetze

- **Entwurf und Realisierung digitaler Schaltnetze**
  - ⇨ **Formale Grundlagen**
  - ⇨ **Realisierung**
  - ⇨ **Entwurf**
  - ⇨ **Laufzeiteffekte**

Formale Grundlagen

- **George Boole (1815-1864)**
  - ⇨ **Algebra der Logik (Boolesche Algebra)**



Def. 9.1: Eine Boolesche Algebra ist eine Menge *V={a,b,c,...}*, auf der zwei zweistellige Operationen ◊ und # so definiert sind, dass durch ihre Anwendung auf Elemente aus V wieder Elemente aus V entstehen (Abgeschlossenheit). Es müssen die Huntingtonschen Axiome gelten.

Huntingtonschen Axiome

- **Kommutativgesetze:**

$$a \lozenge b = b \lozenge a$$

$$a \# b = b \# a$$

- **Distributivgesetze:**

$$a \lozenge (b \# c) = (a \lozenge b) \# (a \lozenge c)$$

$$a \# (b \lozenge c) = (a \# b) \lozenge (a \# c)$$

- **Neutrale Elemente:**

  Es existieren zwei Elemente *e*, *n* $\in$ *V*, so dass gilt:

$a \lozenge e = a$ **(*e* wird Einselement genannt)**

$a \# n = a$ **(*n* wird Nullelement genannt)**

- **Inverse Elemente:**

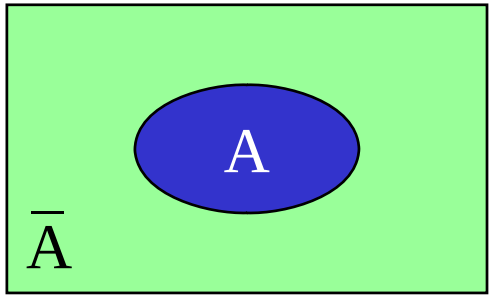
  Für alle *a* $\in$ *V* gibt es ein $\overline{a}$, so dass gilt:

$$a \lozenge \overline{a} = n$$

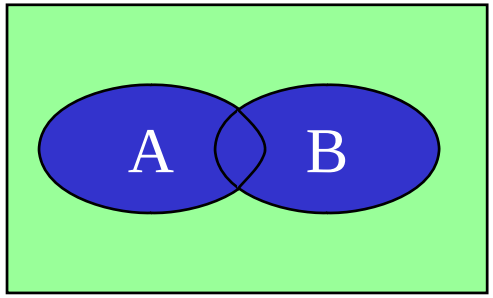
$$a \# \overline{a} = e$$

Beispiel: Mengenalgebra

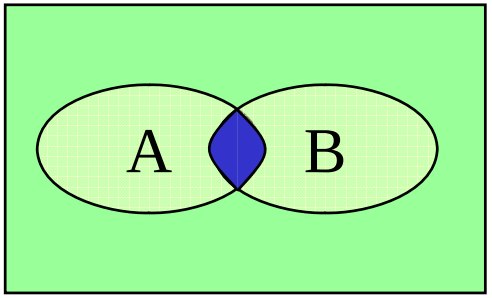
Boolesche Algebra	Mengenalgebra	
V	$P(T)$	**Potenzmenge einer Grundmenge T**
#	$\cup$	**Vereinigung**
◊	$\cap$	**Schnitt**
n	$\varnothing$	**Leere Menge**
e	T	**Grundmenge**
$\overline{a}$	$\overline{A}$	**Komplementmenge von A**



Komplement



$A \cup B$



$A \cap B$

Beispiel: Mengenalgebra

- **Grundmenge**

  T={⌨,🖱,💻}

- **Potenzmenge**

  P(T)= {∅,{⌨}, {🖱}, {💻}, {⌨,🖱}, {⌨,💻}, {🖱,💻}, {⌨,🖱,💻}}

- **Für alle A, B, C ∈ T gilt:**

  - **Abgeschlossenheit**

    $A \cup B \in P(T)$ $A \cap B \in P(T)$

  - **Kommutativgesetze**

    $A \cap B = B \cap A$ $A \cup B = B \cup A$

  - **Distributivgesetze**

    $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

  - **Neutrale Elemente**

    $A \cap T = A$ $A \cup \varnothing = A$

  - **Inverse Elemente**

    $A \cap \overline{A} = \varnothing$ $A \cup \overline{A} = T$

Schaltalgebra

- **Boolesche Algebra bei der die folgende Zuordnungstabelle gilt:**

Boolesche Algebra	Schaltalgebra	
V	$\boldsymbol{B}=\{0,1\}$	**Boolesche Grundmenge**
$\#$	$\vee$	**Oder**
$\lozenge$	$\wedge$	**Und**
n	0	**neutrales Element**
e	1	**Einselement**
$\overline{a}$	$\overline{x_i}$	**Negation**

- **Andere Schreibweisen**
  - ⇨ **Oder:** $x_1 + x_2$, $x_1 \mid x_2$
  - ⇨ **Und:** $x_1 \bullet x_2$, $x_1 * x_2$, $x_1 \,.\, x_2$, $x_1 \,\&\, x_2$, $x_1 x_2$
  - ⇨ **Negation:** $/x_1$, $`x_1$, $\neg x_1$

Funktionstabellen

- **Aus den Huntigtonschen Axiomen lassen sich bereits die Funktionstabellen der in der Algebra definierten Verknüpfungen ableiten**

Oder

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

Und

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Nicht

x_1	$\overline{x}_1$
0	1
1	0

Weitere Sätze

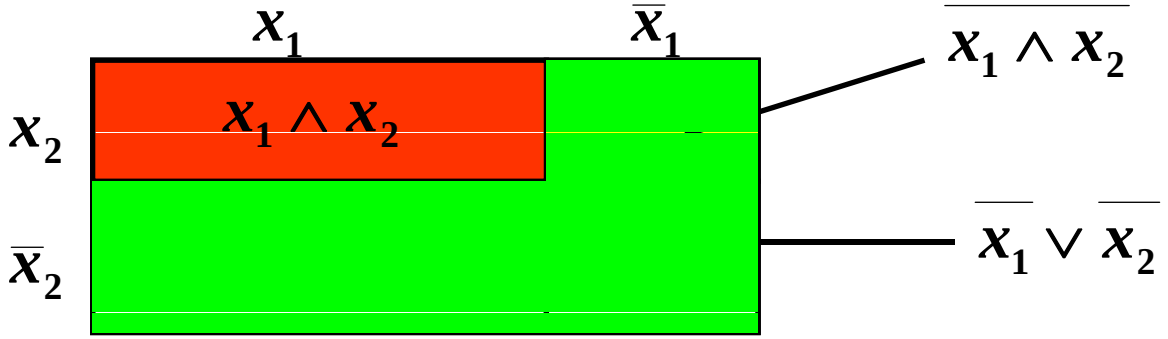
- **Aus den vier Huntigtonschen Axiomen lassen sich weitere Sätze ableiten**
  - **Assoziativgesetze**

    $(x_1 \wedge x_2) \wedge x_3 = x_1 \wedge (x_2 \wedge x_3)$ $(x_1 \vee x_2) \vee x_3 = x_1 \vee (x_2 \vee x_3)$
  - **Idempotenzgesetze**

    $(x_1 \wedge x_1) = x_1$ $(x_1 \vee x_1) = x_1$
  - **Absorptionsgesetze**

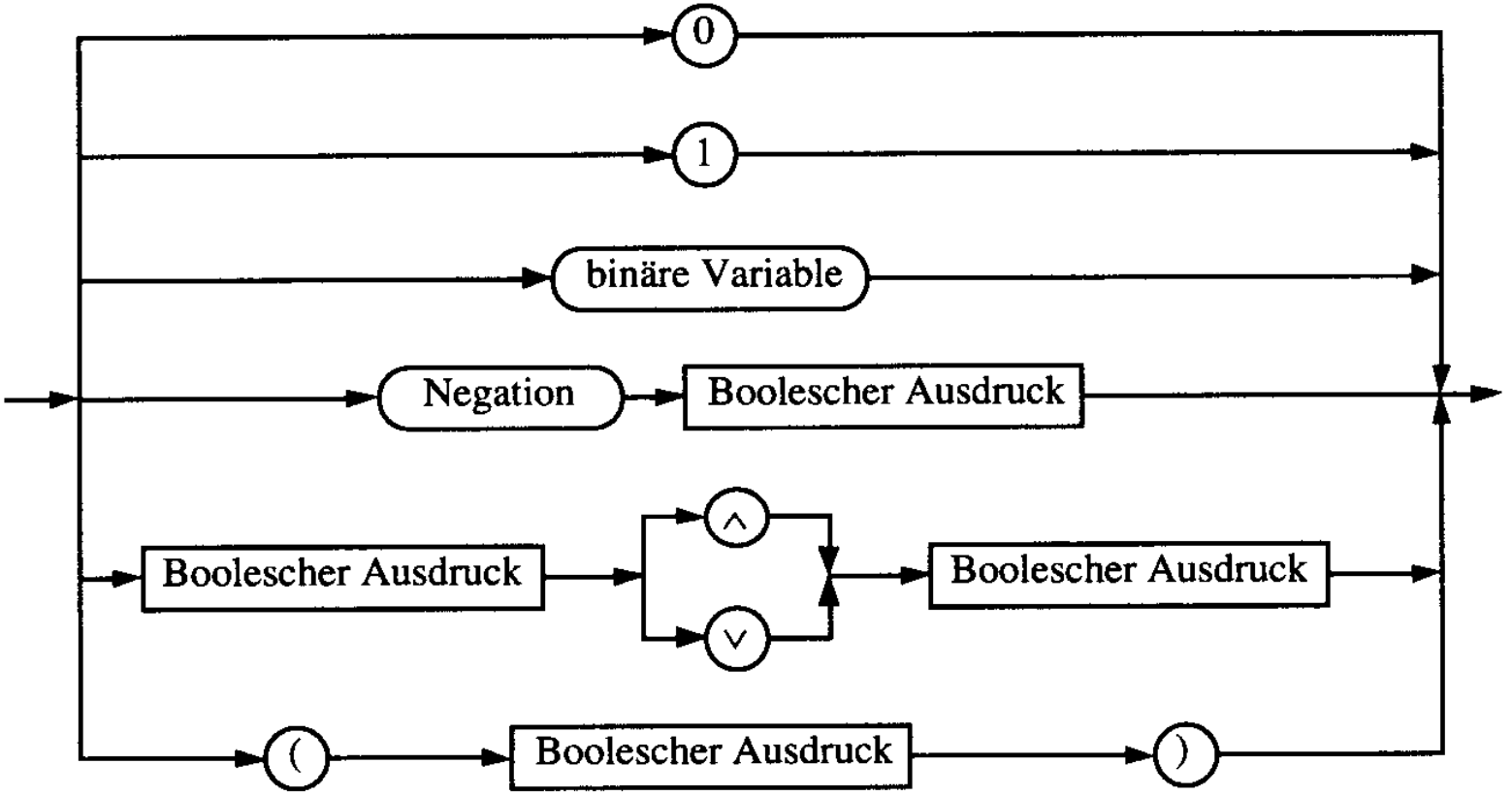
    $x_1 \wedge (x_1 \vee x_2) = x_1$ $x_1 \vee (x_1 \wedge x_2) = x_1$
  - **DeMorgan-Gesetze**

    $\overline{x_1 \wedge x_2} = \overline{x_1} \vee \overline{x_2}$ $\overline{x_1 \vee x_2} = \overline{x_1} \wedge \overline{x_2}$



Boolescher Ausdruck

- **Zeichenfolge, die aus binären Variablen, den Operatoren $\wedge$, $\vee$ und Klammern besteht und den folgenden syntaktischen Regeln folgt:**



Boolescher Ausdruck

- **Boolesche Ausdrücke sind nur eine syntaktische Konstruktion**
  - ⇨ **Bedeutung erhält ein Boolescher Ausdruck erst, wenn den Konstanten 0 und 1 die Wahrheitswerte „falsch“ oder „wahr“ zugeordnet wird**
- **Interpretation**
  - ⇨ **Belegung der binären Variablen eines Booleschen Ausdrucks mit Wahrheitswerten**
  - ⇨ **Liefert eine Aussage, die entweder „wahr“ oder „falsch“ sein kann**
  - ⇨ **Anwendung: Simulation**
- **Tautologie**
  - ⇨ **Boolescher Ausdruck, bei dem alle Belegungen der binären Variablen den Wahrheitswert „wahr“ liefern**
  - ⇨ $(x_1 \vee x_2) \vee (\overline{x}_1 \wedge \overline{x}_2)$
  - ⇨ **Anwendung: Verifikation von Schaltungen**

Boolesche Funktion

Def. 9.2: Es sei ein n-Tupel von binären Variablen $(x_1, x_2,...,x_n)$ **gegeben. Eine n-stellige Boolesche Funktion ordnet jeder Belegung der Variablen** $x_1, x_2,...,x_n$ **mit den Wahrheitswerten „wahr“ oder „falsch“ genau einen Wahrheitswert zu.**

$f : \{0,1\}^n \rightarrow \{0,1\}$ **oder** $f : B^n \rightarrow B$

Satz 9.1: Es gibt genau 2^n **verschiedene Belegungen der Variablen einer n-stelligen Booleschen Funktion. Die Anzahl verschiedener n-stelliger Boolescher Funktionen beträgt** $2^{(2^n)}$

Bew: Über Funktionstabelle

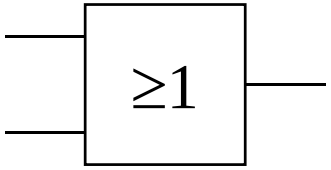
Übersicht der 2-stelligen Booleschen Funktionen

Benennung der Verknüpfung	Funktionswert $y = f(x_1, x_2)$					Schreibweise mit den Zeichen $\wedge \vee -$	Bemerkung
	x_1	=	0	1	0 1		
	x_2	=	0	0	1 1		
Null	y_0	=	0	0	0 0	0	Null
UND–Verknüpfung	y_1	=	0	0	0 1	$x_1 \wedge x_2$	x_1 UND x_2
Inhibition	y_2	=	0	0	1 0	$\overline{x}_1 \wedge x_2$	
Transfer	y_3	=	0	0	1 1	x_2	
Inhibition	y_4	=	0	1	0 0	$x_1 \wedge \overline{x}_2$	
Transfer	y_5	=	0	1	0 1	x_1	
Antivalenz	y_6	=	0	1	1 0	$(x_1 \wedge \overline{x}_2) \vee (\overline{x}_1 \wedge x_2)$	Exclusiv–ODER
ODER–Verknüpfung	y_7	=	0	1	1 1	$x_1 \vee x_2$	x_1 ODER x_2
NOR–Verknüpfung	y_8	=	1	0	0 0	$\overline{x_1 \vee x_2}$	NICHT–ODER
Äquivalenz	y_9	=	1	0	0 1	$(x_1 \wedge x_2) \vee (\overline{x}_1 \wedge \overline{x}_2)$	
Komplement	y_{10}	=	1	0	1 0	$\overline{x}_1$	
Implikation	y_{11}	=	1	0	1 1	$\overline{x}_1 \vee x_2$	
Komplement	y_{12}	=	1	1	0 0	$\overline{x}_2$	
Implikation	y_{13}	=	1	1	0 1	$x_1 \vee \overline{x}_2$	
NAND–Verknüpfung	y_{14}	=	1	1	1 0	$\overline{x_1 \wedge x_2}$	NICHT–UND
Eins	y_{15}	=	1	1	1 1	1	Eins

Darstellung einiger zweistelliger Funktionen

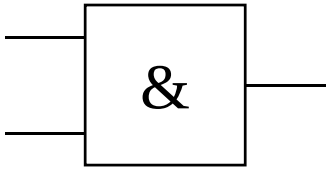
ODER

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1



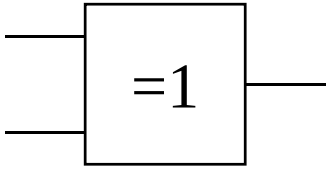
UND

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1



Exklusiv-Oder

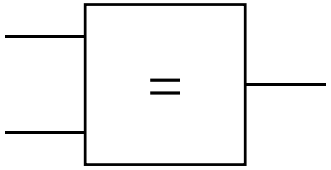
x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0



Darstellung einiger zweistelliger Funktionen

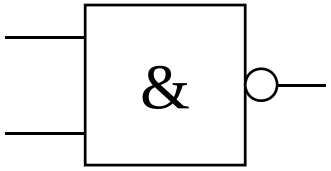
Äquivalenz

x_1	x_2	$x_1 \equiv x_2$
0	0	1
0	1	0
1	0	0
1	1	1



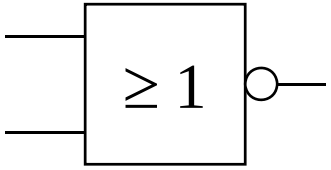
NAND

x_1	x_2	$x_1 \overline{\wedge} x_2$
0	0	1
0	1	1
1	0	1
1	1	0



NOR

x_1	x_2	$x_1 \overline{\vee} x_2$
0	0	1
0	1	0
1	0	0
1	1	0



Operatorensysteme

Def. 9.3: Eine Menge von Operatoren (Operatorensystem) heißt vollständig für eine Menge F von Funktionen, wenn sich jede Funktion in F durch die Operatoren darstellen lässt.

- **Beispiel für ein vollständiges Operatorensysteme:**

$$\{\wedge, \vee, \bar{}\}$$

Beweis:

Rückführung von $\{\wedge, \vee, \bar{}\}$ auf $\{\wedge, \bar{}\}$:

$$\vee: \quad a \vee b = \overline{(\overline{(a \vee b)})} = \overline{(\overline{a} \wedge \overline{b})}$$

Entsprechend zeigt man, dass $\{\vee, \bar{}\}$ ein vollständiges Operatorensystem ist.

Operatorensysteme

- **Beispiele für vollständige Operatorensysteme:**

Operatorensystem	Negation	Konjunktion	Disjunktion
$(\wedge,\vee,\bar{})$	$\bar{x}_1$	$x_1 \wedge x_2$	$\overline{x_1 \vee x_2}$
$(\wedge,\bar{})$	$\bar{x}_1$	$\overline{x_1 \wedge x_2}$	$\bar{x}_1 \wedge \bar{x}_2$
$(\vee,\bar{})$	$\bar{x}_1$	$\overline{\bar{x}_1 \vee \bar{x}_2}$	$x_1 \vee x_2$
$(\wedge)$	$x_1 \wedge x_1$	$(x_1 \wedge x_2) \wedge (x_1 \wedge x_2)$	$(x_1 \wedge x_1) \wedge (x_2 \wedge x_2)$
$(\vee)$	$x_1 \vee x_1$	$(x_1 \vee x_1) \vee (x_2 \vee x_2)$	$(x_1 \vee x_2) \vee (x_1 \vee x_2)$
$(\wedge,\oplus)$	$x_1 \oplus 1$	$x_1 \wedge x_2$	$x_1 \wedge x_2 \oplus x_1 \oplus x_2$

Auswertung

- **Zum Wahrheitswert einer Aussage gelangt man durch rekursives Auswerten der Booleschen Funktionen in einem Ausdruck**
  - ⇨ **Negation vor Konjunktion**
  - ⇨ **Konjunktion vor Disjunktion**
  - ⇨ **Klammerung beachten**
- **Beispiel: Ist die folgende Funktion eine Tautologie?**

$$f(x_1, x_2) = (x_1 \wedge x_2) \vee (\overline{x}_1 \wedge \overline{x}_2) \equiv (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee x_2)$$

Normalformen

- Eine Funktion kann durch verschiedene Boolesche Ausdrücke beschrieben werden
  - ⇨ Auch bei der Beschränkung auf ein vollständiges Operatorensystem ergeben sich noch mehrere Darstellungsmöglichkeiten
- Normalformen bilden eine Standarddarstellung in einem vollständigen Operatorensystem
  - ⇨ Disjunktive Normalform
  - ⇨ Konjunktive Normalform
- Es gibt weitere Normalformen, die in dieser Vorlesung nicht behandelt werden
  - ⇨ Reed-Muller-Form
  - ⇨ Äquivalenzpolynom

Literal und Produktterm

Def. 9.4: Ein Literal L_i ist entweder eine Variable x_i oder ihre Negation $\overline{x}_i$. $L_i \in \{x_i, \overline{x}_i\}$

Def. 9.5: Ein Produktterm $K(x_1,...,x_m)$ ist die Konjunktion von Literalen oder den Konstanten 0 oder 1:

$$\bigwedge_{i=1}^{m} L_i = L_1 \wedge ... \wedge L_m$$

- **Jeder Produktterm $K(x_1,...,x_m)$ kann so dargestellt werden, dass eine Variable x in höchstens einem Literal vorkommt.**
  - **Falls $Lj=x$ und $L_k=x$ ist, gilt** $L_j \wedge L_k = x$
  - **Falls $Lj=\overline{x}$ und $L_k=\overline{x}$ ist, gilt** $L_j \wedge L_k = \overline{x}$
  - **Falls $Lj=x$ und $L_k=\overline{x}$ ist, gilt** $L_j \wedge L_k = 0$

Implikant und Minterm

Def. 9.6: Ein Produktterm $K(x_1,...,x_m)$ heißt Implikant einer Booleschen Funktion $f(x_1,...,x_n)$, wenn aus $K(x_1,...,x_m)=1$ für eine Belegung $x_1,...,x_m \in B^n$ folgt, dass $f(x_1,...,x_n)=1$.

Def. 9.7: Ein Implikant $K(x_1,...,x_n)$ heißt Minterm (m), wenn ein Literal jeder Variablen x_i der Funktion $f(x_1,...,x_n)$ genau einmal in K vorkommt.

- **Implikanten haben eine oder mehrere 1-Stellen in der Funktion**
  - ⇨ **mehrere Implikanten können sich überdecken**
- **Ein Minterm ist genau bei einer Belegung der Variablen gleich 1**
  - ⇨ **Ein Minterm trägt zu genau einer 1-Stelle der Funktion bei**
  - ⇨ **Die Minterme einer Funktion können sich nicht überdecken**

Mintermtabelle

Satz 9.2: Zu einer Booleschen Funktion $f(x_1,\ldots,x_n)$ mit n Literalen gibt es maximal 2^n verschiedene Minterme m_i.

Bew: Durch Aufzählung aller Kombinationen und Induktion über n.

- **Man definiert eine Reihenfolge aller Minterme über den Index i**

i_{10}	i_2	**Minterm** m_i
0	000	$\overline{x}_2 \wedge \overline{x}_1 \wedge \overline{x}_0$
1	001	$\overline{x}_2 \wedge \overline{x}_1 \wedge x_0$
2	010	$\overline{x}_2 \wedge x_1 \wedge \overline{x}_0$
3	011	$\overline{x}_2 \wedge x_1 \wedge x_0$
4	100	$x_2 \wedge \overline{x}_1 \wedge \overline{x}_0$
5	101	$x_2 \wedge \overline{x}_1 \wedge x_0$
6	110	$x_2 \wedge x_1 \wedge \overline{x}_0$
7	111	$x_2 \wedge x_1 \wedge x_0$

Disjunktive Normalform

Def. 9.8: Es sei eine Boolesche Funktion $f(x_1,...,x_n):B^n \to B$ gegeben. Ein Boolescher Ausdruck heißt disjunktive Normalform (DNF) der Funktion f, wenn er aus einer disjunktiven Verknüpfung von Mintermen K_i besteht.

$$f(x_1,...,x_n) = K_0 \vee K_1 \vee ... \vee K_k \text{ mit } 0 \le k \le 2^n - 1$$

$$= \bigvee_{0}^{2^n-1} \alpha_i \wedge K_i \text{ mit } \alpha_i \in \{0,1\}$$

- α_i **heißt <u>Mintermkoeffizient</u>**
  - ⇨ $\alpha_i = 1$**, wenn der Minterm** m_i **zu** f **gehört,**
  - ⇨ $\alpha_i = 0$**, sonst**
- **Beispiele**

$f(x_2, x_1, x_0) = x_2 x_1 x_0 \vee x_2 \bar{x}_1 x_0 \vee \bar{x}_2 x_1 \bar{x}_0 \vee \bar{x}_2 \bar{x}_1 \bar{x}_0$ **ist eine DNF**

$f(x_2, x_1, x_0) = x_2 x_1 x_0 \vee x_2 \bar{x}_1 \vee x_1 (x_2 x_0 \vee \bar{x}_2 \bar{x}_0)$ **ist keine DNF**

Disjunktion und Maxterm

Def. 9.9: Es sei $D(x_1,...,x_m)$ eine Disjunktion von Literalen, wobei die Konstanten 0 und 1 auftreten dürfen. $D(x_1,...,x_m)$ heißt Implikat einer Booleschen Funktion $f(x_1,...,x_n)$, wenn aus $D(x_1,...,x_m)=0$ für eine Belegung $x_1,...,x_n \in B^n$ folgt, dass $f(x_1,...,x_n)=0$.

Def. 9.10: Ein Implikat $D(x_1,...,x_n)$ heißt Maxterm (M), wenn ein Literal jeder Variablen x_i der Funktion $f(x_1,...,x_n)$ genau einmal in D vorkommt.

- **Implikate haben eine oder mehrere Nullstellen in der Funktion**
  - ⇨ **mehrere Implikaten können sich überdecken**
- **Ein Maxterm ist genau bei einer Belegung der Variablen gleich 0**
  - ⇨ **Ein Maxterm trägt zu genau einer Nullstelle der Funktion bei**
  - ⇨ **Die Maxterme einer Funktion können sich in den 1-Stellen überdecken**

Min- und Maxtermtabelle

Satz 9.3: Zu einer Booleschen Funktion $f(x_1,\ldots,x_n)$ mit n Literalen gibt es maximal 2^n verschiedene Maxterme M_i.

Bew: Durch Aufzählung aller Kombinationen und Induktion über n.

- **Man definiert eine Reihenfolge aller Maxterme über den Index i analog zu den Mintermen**

i_{10}	i_2	**Minterm** m_i	**Maxterm** M_i
0	000	$\bar{x}_2 \wedge \bar{x}_1 \wedge \bar{x}_0$	$x_2 \vee x_1 \vee x_0$
1	001	$\bar{x}_2 \wedge \bar{x}_1 \wedge x_0$	$x_2 \vee x_1 \vee \bar{x}_0$
2	010	$\bar{x}_2 \wedge x_1 \wedge \bar{x}_0$	$x_2 \vee \bar{x}_1 \vee x_0$
3	011	$\bar{x}_2 \wedge x_1 \wedge x_0$	$x_2 \vee \bar{x}_1 \vee \bar{x}_0$
4	100	$x_2 \wedge \bar{x}_1 \wedge \bar{x}_0$	$\bar{x}_2 \vee x_1 \vee x_0$
5	101	$x_2 \wedge \bar{x}_1 \wedge x_0$	$\bar{x}_2 \vee x_1 \vee \bar{x}_0$
6	110	$x_2 \wedge x_1 \wedge \bar{x}_0$	$\bar{x}_2 \vee \bar{x}_1 \vee x_0$
7	111	$x_2 \wedge x_1 \wedge x_0$	$\bar{x}_2 \vee \bar{x}_1 \vee \bar{x}_0$

Konjunktive Normalform

Def. 9.11: Es sei eine Boolesche Funktion $f(x_1,...,x_n):B^n \rightarrow B$ gegeben. Ein Boolescher Ausdruck heißt Konjunktive Normalform (KNF) der Funktion f, wenn er aus einer konjunktiven Verknüpfung von Maxtermen D_i besteht.

$$f(x_1,...,x_n) = D_0 \wedge D_1 \wedge ... \wedge D_k \text{ mit } 0 \le k \le 2^n - 1$$

$$= \bigwedge_{0}^{2^n-1} (\beta_i \vee D_i) \text{ mit } \beta_i \in \{0,1\}$$

- β_i **heißt <u>Maxtermkoeffizient</u>**
  - ⇨ $\beta_i = 0$**, wenn der Maxterm** m_i **zu** f **gehört,**
  - ⇨ $\beta_i = 1$**, sonst**
- **Beispiel**

$f(x_2, x_1, x_0) = (x_2 \vee x_1 \vee x_0) \wedge (x_2 \vee \bar{x}_1 \vee x_0) \wedge (\bar{x}_2 \vee x_1 \vee \bar{x}_0)$ **ist eine KNF**

KNF-DNF Umwandlung

Satz 9.4: Für jede Boolesche Funktion $f(x_1,\ldots,x_n)$ gilt $\alpha_i = \beta_i$.

Bew: 2 Fälle

- **Fall 1:** $\alpha_i=1$
  - $\Rightarrow$ m_i **gehört zur DNF der Funktion** f
  - $\Rightarrow$ M_i **gehört nicht zur KNF der Funktion** f
  - $\Rightarrow$ $\beta_i=1$
- **Fall 2:** $\alpha_i=0$
  - $\Rightarrow$ m_i **gehört nicht zur DNF der Funktion** f
  - $\Rightarrow$ M_i **gehört zur KNF der Funktion** f
  - $\Rightarrow$ $\beta_i=0$

Zusammenfassung der wichtigsten Begriffe

- **Literal:** Boolesche Variable
- **Produktterm:** UND-Verknüpfung von Literalen
- **Implikant:** Produktterm, der eine oder mehrere „1“-Stellen einer booleschen Funktion beschreibt (impliziert)
- **Implikat:** Disjunktion (ODER-Verknüpfung) von Literalen
- **Minterm:** Implikant, der genau eine „1“-Stelle einer booleschen Funktion beschreibt
- **Maxterm:** Implikat, der genau eine „0“-Stelle einer booleschen Funktion beschreibt
- **Normalform:** Darstellung einer Booleschen Funktion durch Minterme (DNF) oder Maxterme (KNF)

Der Shannonsche Entwicklungssatz

- **DNF und KNF können durch einfache logische Umformungen in gewöhnliche disjunktive und konjunktive Formen gebracht werden**
  - ⇨ **DF und KF**
- **Zur Berechnung der Normalformen ist der shannonsche Entwicklungssatz hilfreich**

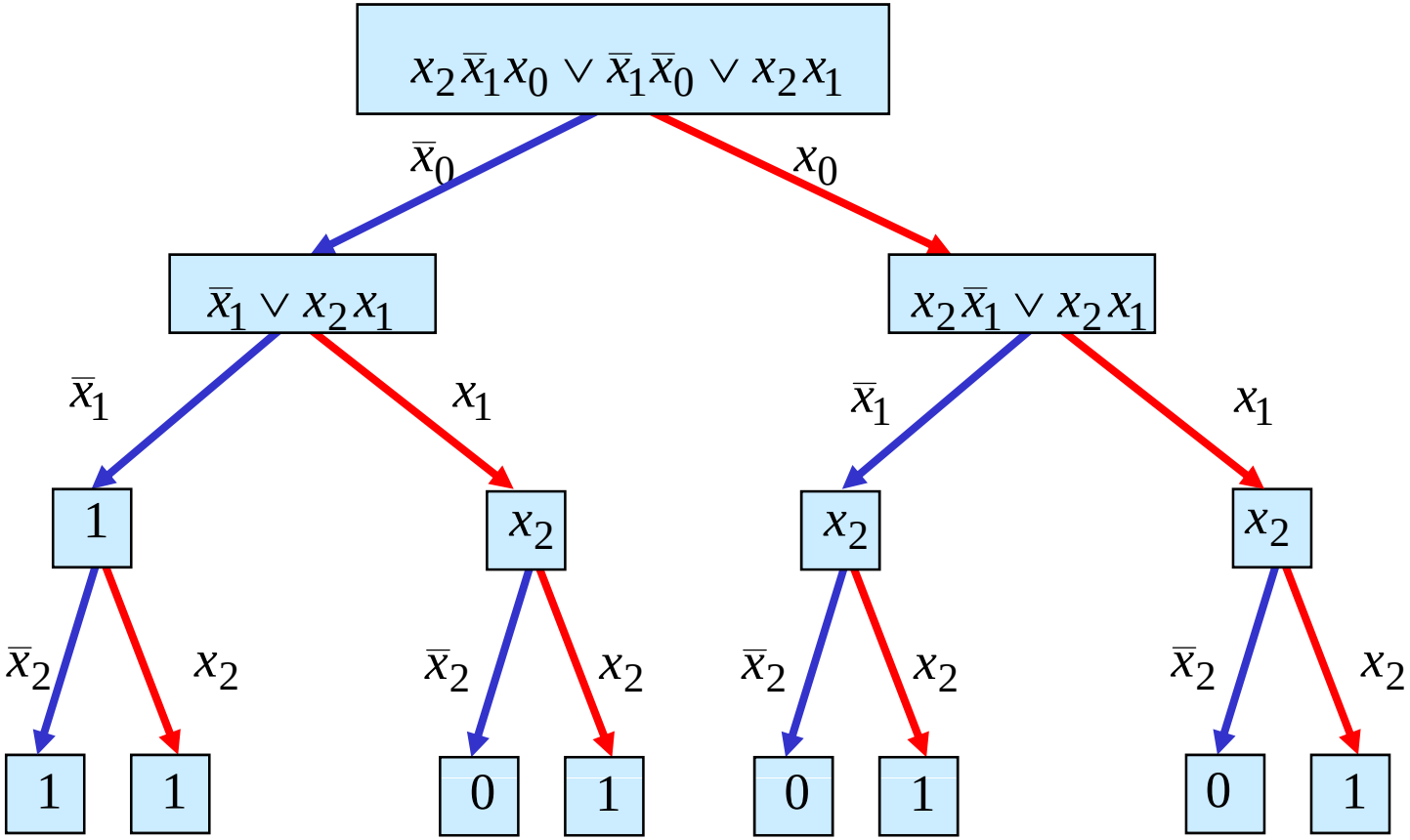
Satz 9.5: Für jede Boolesche Funktion $f(x_1,\ldots,x_n)$ gilt

$$f(x_1,\ldots,x_n) = \left(x_i \wedge f(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n)\right) \vee \left(\bar{x}_i \wedge f(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n)\right)$$

- **Beispiel:**

$$\begin{aligned} f(x_2,x_1,x_0) &= x_2\bar{x}_1x_0 \vee \bar{x}_1\bar{x}_0 \vee x_2x_1 \\ &= x_0(x_2\bar{x}_1 \vee x_2x_1) \vee \bar{x}_0(\bar{x}_1 \vee x_2x_1) \\ &= x_2\bar{x}_1x_0 \vee x_2x_1x_0 \vee \bar{x}_1\bar{x}_0 \vee x_2x_1\bar{x}_0 \\ &= x_2(\bar{x}_1x_0 \vee x_1x_0 \vee \bar{x}_1\bar{x}_0 \vee x_1\bar{x}_0) \vee \bar{x}_2(\bar{x}_1\bar{x}_0) \\ &= x_2x_1x_0 \vee x_2\bar{x}_1x_0 \vee x_2x_1\bar{x}_0 \vee x_2\bar{x}_1\bar{x}_0 \vee \bar{x}_2\bar{x}_1\bar{x}_0 \end{aligned}$$

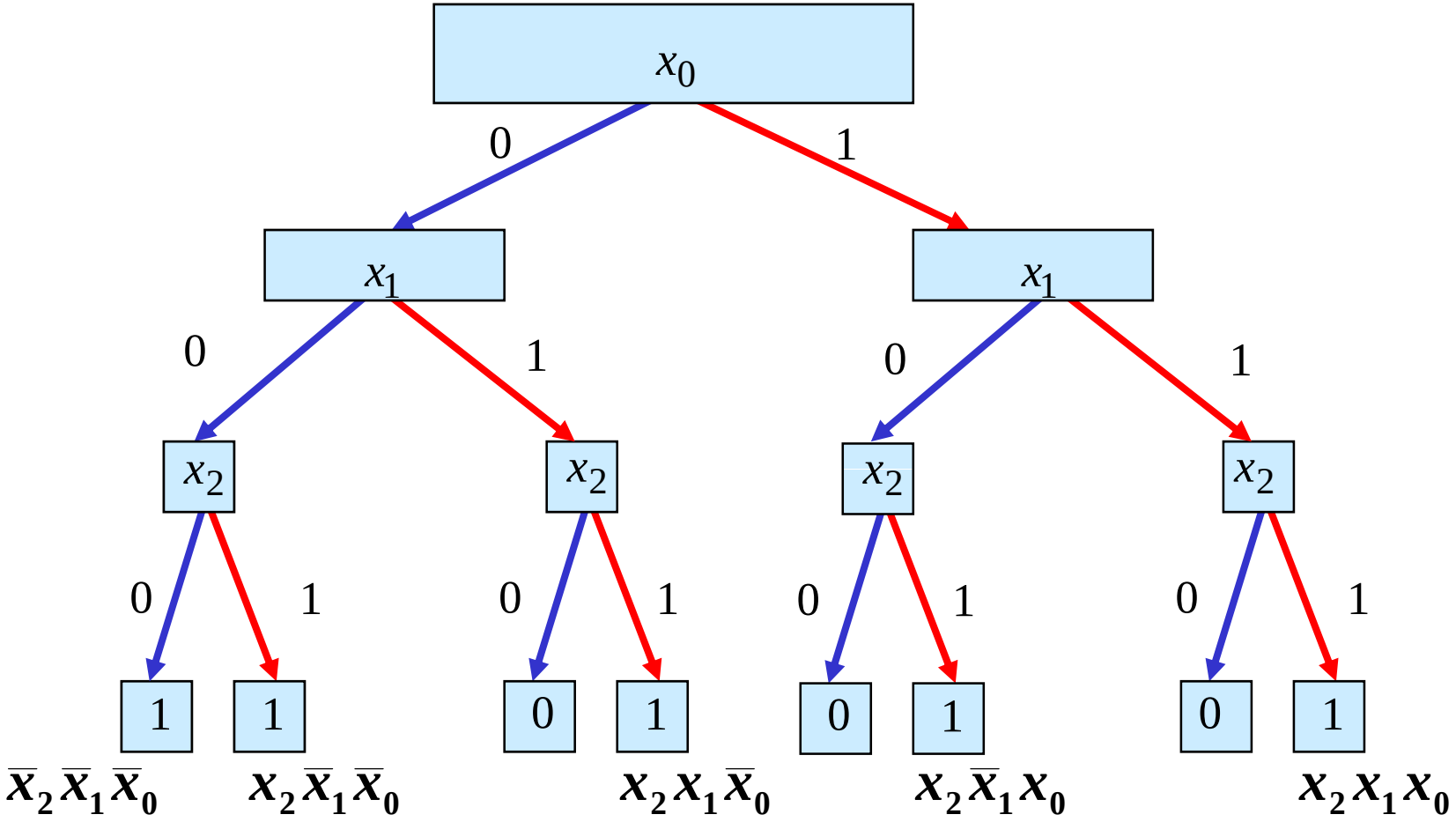
Baumdarstellung



Binary Decision Diagram (BDD)

- **Andere Interpretation der Shannon-Entwicklung**

$$f(x_2, x_1, x_0) = x_2 x_1 x_0 \vee x_2 \overline{x}_1 x_0 \vee x_2 x_1 \overline{x}_0 \vee x_2 \overline{x}_1 \overline{x}_0 \vee \overline{x}_2 \overline{x}_1 \overline{x}_0$$

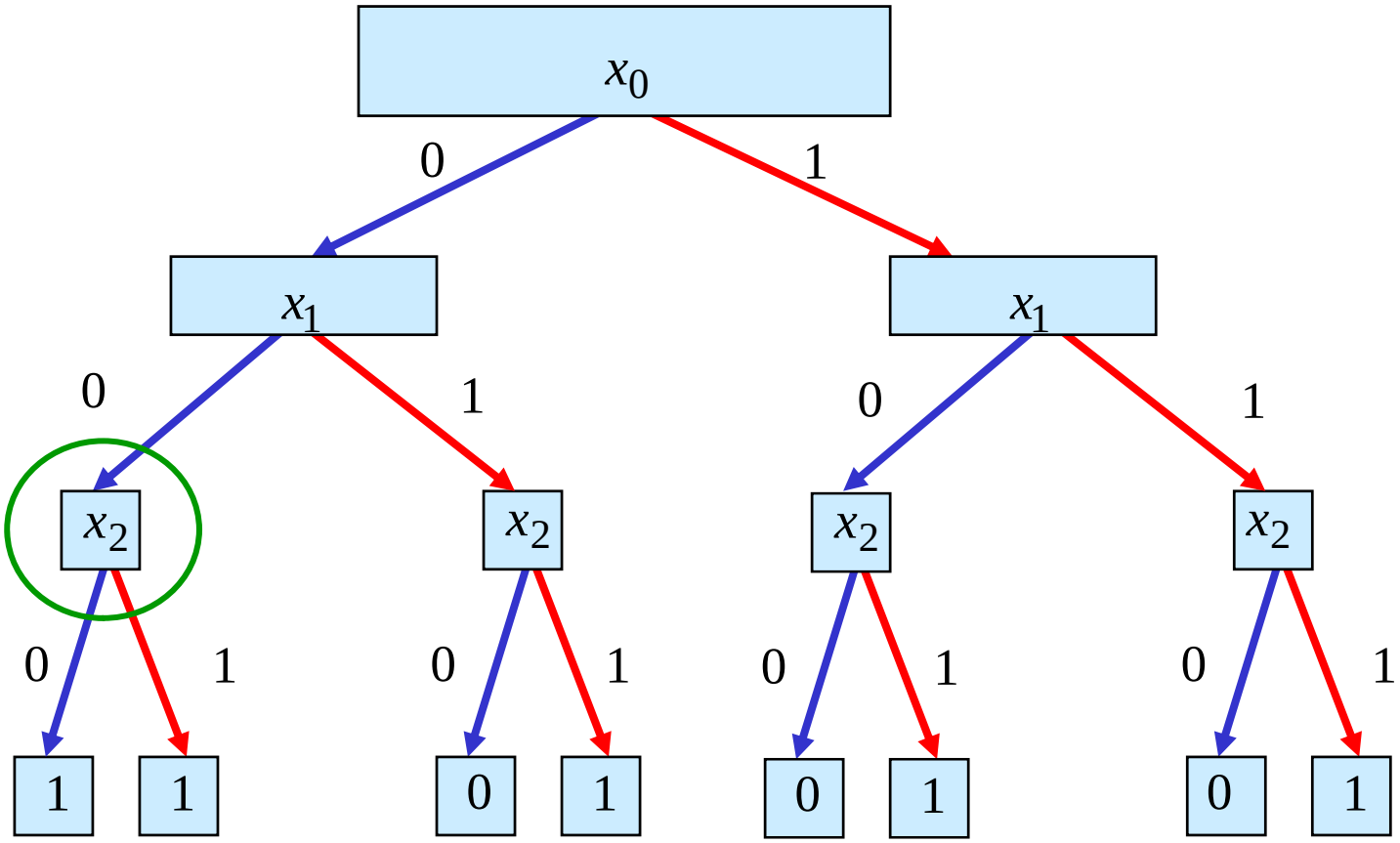


Reduzierte Baumdarstellungen

- **Da die Variablen in allen Pfaden in der gleichen Reihenfolge auftauchen, spricht man auch von einem ordered BDD (OBDD)**
- **Ein BDD benötigt 2^n Knoten bei n Variablen**
  - ⇨ **Für viele Anwendung ist die Speicherung aller Knoten nicht notwendig**
  - ⇨ **Regel 1: Knoten, deren Nachfolger gleich sind, können eliminiert werden**
  - ⇨ **Regel 2: Teile des Baumes, die genau so noch einmal vorkommen, können gemeinsam genutzt werden**
- **Es entsteht ein bezüglich einer Ordnung der Variablen eindeutiger reduzierter Baum**

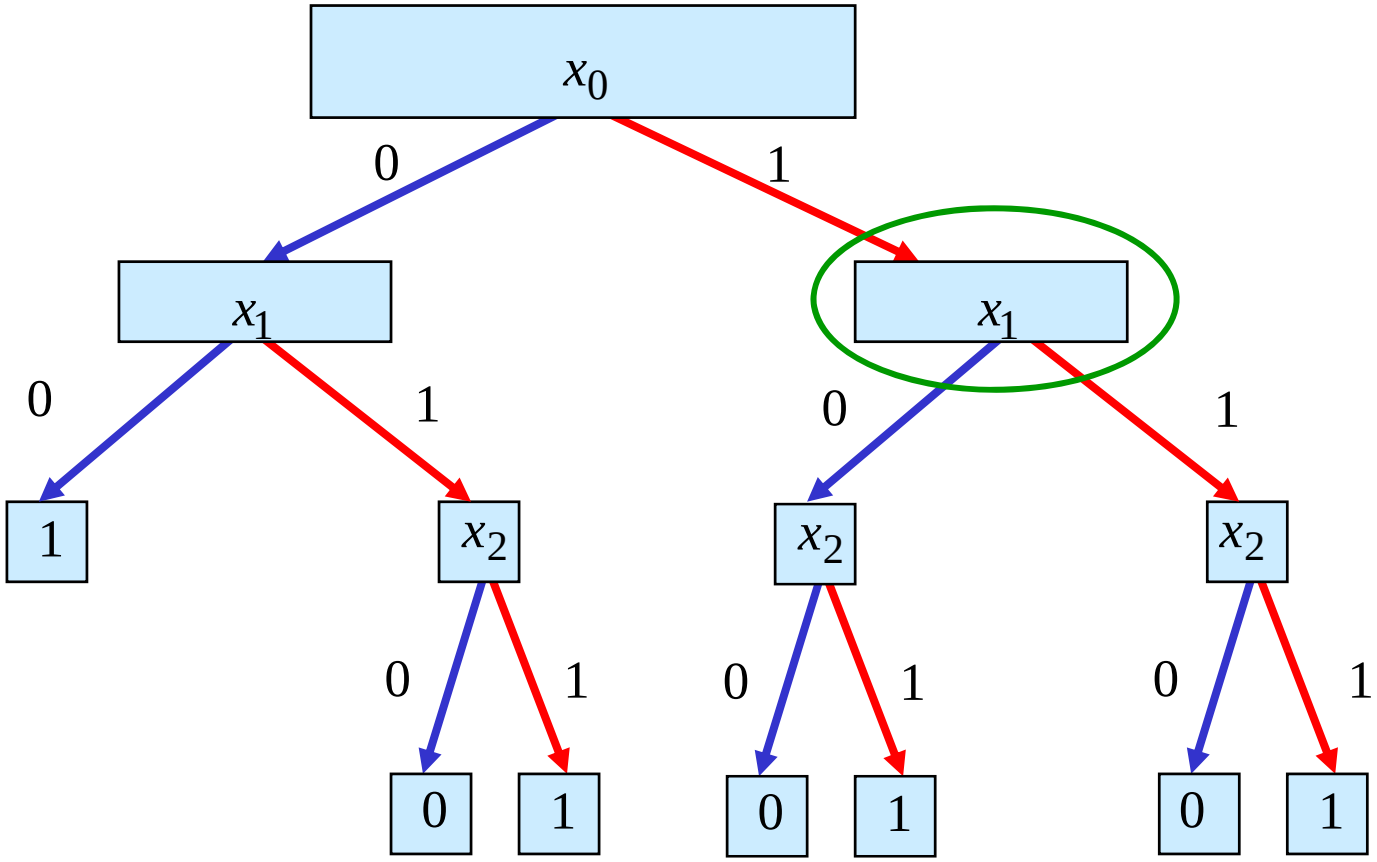
Reduced Ordered BDD (ROBDD)

Ausgangsgraph



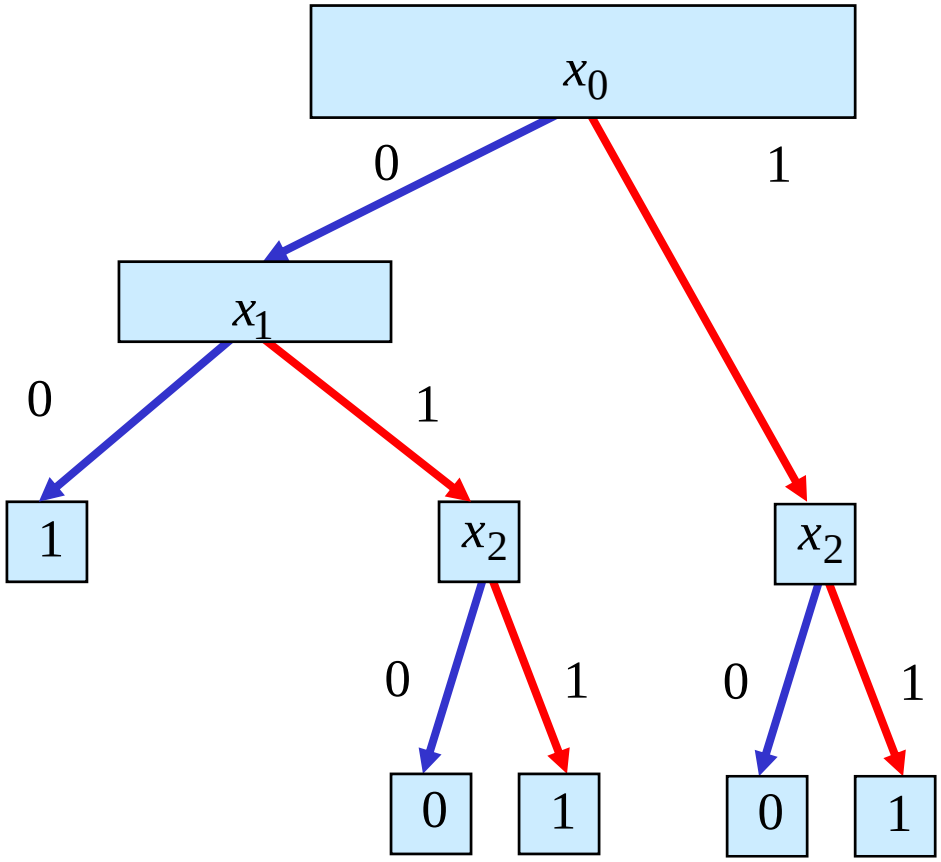
Reduced Ordered BDD (ROBDD)

Regel 1



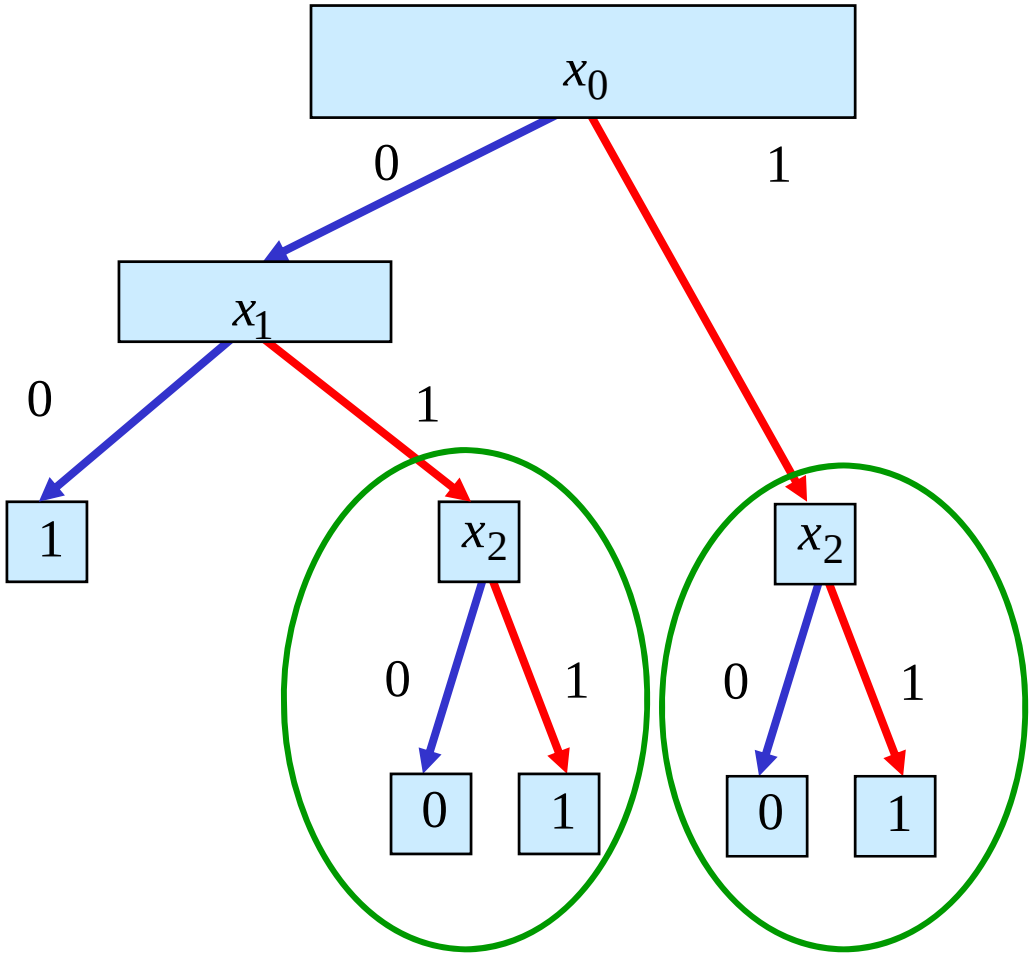
Reduced Ordered BDD (ROBDD)

Regel 1



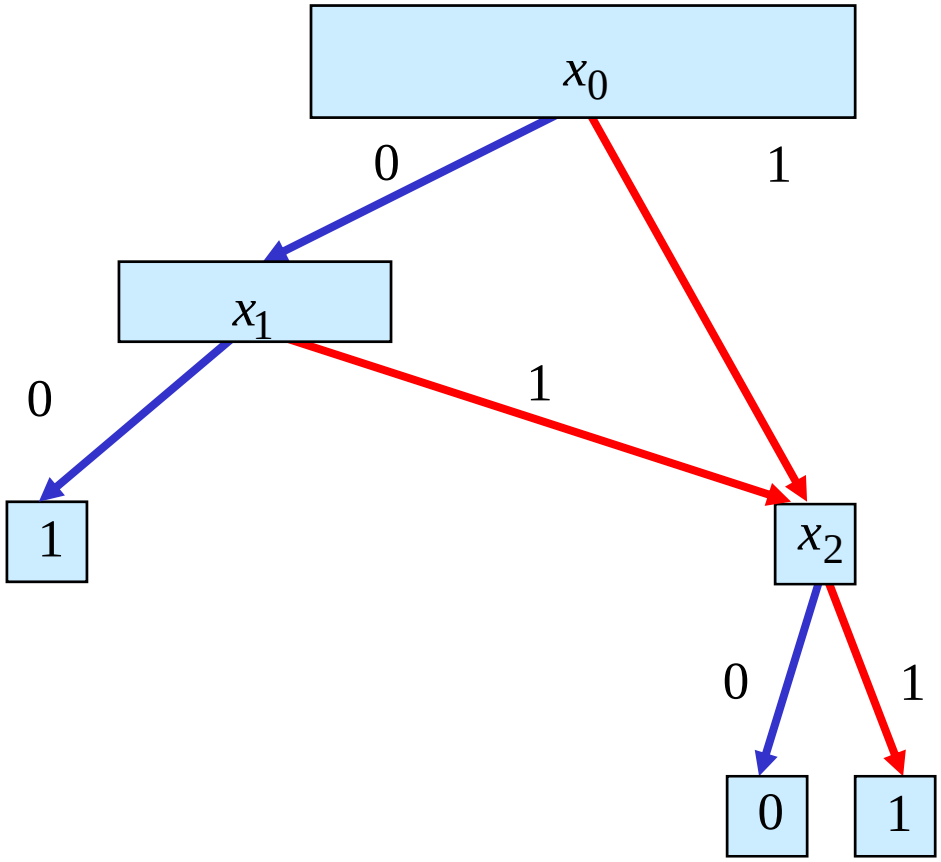
Reduced Ordered BDD (ROBDD)

Regel 2



Reduced Ordered BDD (ROBDD)

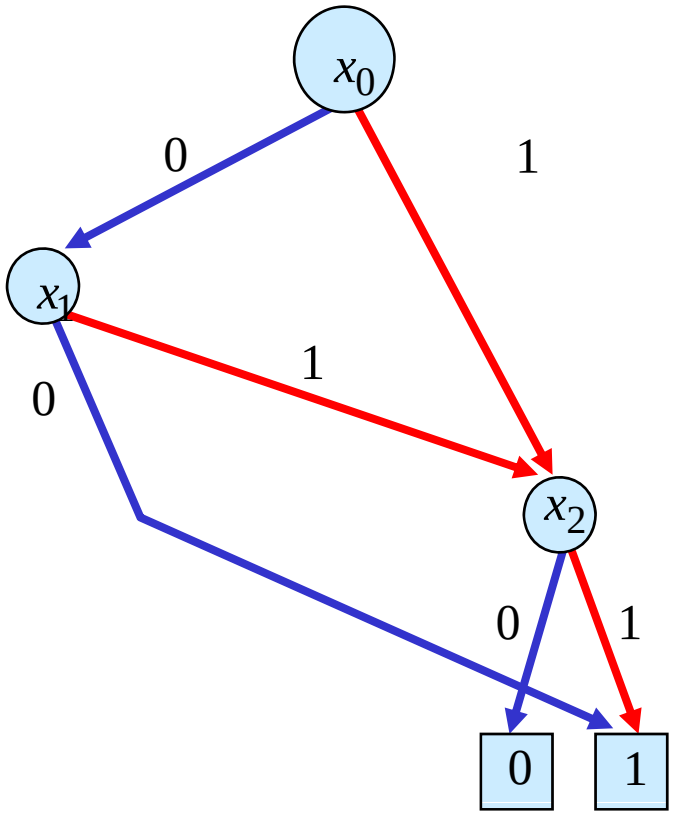
Regel 2



Anwendung: Simulation

- **Der Funktionswert wird ausgehend von der Wurzel ermittelt**

  ⇨ $f(x_2, x_1, x_0) = 0$ **für** $x_2 = 0, x_1 = 1, x_0 = 0$



Anwendung: Verifikation

- **Die Gleichung**

$$s \equiv t \Leftrightarrow N(M(s,t)) = 0$$

kann umgeschrieben werden zu

$$s \equiv t \Leftrightarrow BDD(s \oplus t) = 0$$

DNF/KNF-Konversion

- **Statt der Min- und Maxterme kann man auch deren Indizes angeben**
  - ⇨ f=MINt(0,3,4,7)
  - ⇨ f=MAXt(1,2,5,6)
- **Für die Umwandlung der DNF einer Funktion f in die entsprechende KNF folgt direkt aus Satz 9.4:**
  - ⇨ **Die Indizes der Minterme, die nicht in der Funktionsdarstellung der DNF der Funktion verwendet werden, sind Indizes der Maxterme der KNF der Funktion**

DNF/KNF-Konversion

i_{10}	$x_2 x_1 x_0$	**Minterme**	**Maxterme**
0	000	$\bar{x}_2 \wedge \bar{x}_1 \wedge \bar{x}_0$	
1	001		$x_2 \vee x_1 \vee \bar{x}_0$
2	010		$x_2 \vee \bar{x}_1 \vee x_0$
3	011	$\bar{x}_2 \wedge x_1 \wedge x_0$	
4	100	$x_2 \wedge \bar{x}_1 \wedge \bar{x}_0$	
5	101		$\bar{x}_2 \vee x_1 \vee \bar{x}_0$
6	110		$\bar{x}_2 \vee \bar{x}_1 \vee x_0$
7	111	$x_2 \wedge x_1 \wedge x_0$	

DNF : $f(x_2, x_1, x_0) = \bar{x}_2 \bar{x}_1 \bar{x}_0 \vee \bar{x}_2 x_1 x_0 \vee x_2 \bar{x}_1 \bar{x}_0 \vee x_2 x_1 x_0$

KNF : $f(x_2, x_1, x_0) = (x_2 \vee x_1 \vee \bar{x}_0) \wedge (x_2 \vee \bar{x}_1 \vee x_0) \wedge (\bar{x}_2 \vee x_1 \vee \bar{x}_0) \wedge (\bar{x}_2 \vee \bar{x}_1 \vee x_0)$

NAND-NOR-Konversion

- **Sowohl NAND- als auch NOR-System sind vollständige Operatorensysteme**
  - ⇨ **alle Booleschen Funktionen lassen sich mit mit diesen Operatoren darstellen**
  - ⇨ **da sowohl NAND- als auch NOR-Gatter besonders einfach realisiert werden, haben diese Darstellungen eine besondere Bedeutung im Schaltkreisentwurf**
- **NAND-Konversion aus der DNF:**

$$
\begin{aligned}
f(x_2, x_1, x_0) &= x_2 x_1 \bar{x}_0 \vee x_2 \bar{x}_1 x_0 \vee \bar{x}_2 x_1 x_0 \vee \bar{x}_2 \bar{x}_1 \bar{x}_0 \\
&= \overline{\overline{x_2 x_1 \bar{x}_0 \vee x_2 \bar{x}_1 x_0 \vee \bar{x}_2 x_1 x_0 \vee \bar{x}_2 \bar{x}_1 \bar{x}_0}} \\
&= \overline{\overline{x_2 x_1 \bar{x}_0} \wedge \overline{x_2 \bar{x}_1 x_0} \wedge \overline{\bar{x}_2 x_1 x_0} \wedge \overline{\bar{x}_2 \bar{x}_1 \bar{x}_0}} \\
&= \mathbf{NAND_4}(\mathbf{NAND_3}(x_2 x_1 \bar{x}_0), \mathbf{NAND_3}(x_2 \bar{x}_1 x_0), \\
&\quad \mathbf{NAND_3}(\bar{x}_2 x_1 x_0), \mathbf{NAND_3}(\bar{x}_2 \bar{x}_1 \bar{x}_0))
\end{aligned}
$$

NAND-NOR-Konversion

- **NOR-Konversion aus der KNF:**

$$f(x_2, x_1, x_0) = x_2 x_1 \bar{x}_0 \vee x_2 \bar{x}_1 x_0 \vee \bar{x}_2 x_1 x_0 \vee \bar{x}_2 \bar{x}_1 \bar{x}_0 \quad \textbf{(DNF)}$$

$$= (x_2 \vee x_1 \vee x_0)(x_2 \vee \bar{x}_1 \vee \bar{x}_0)(\bar{x}_2 \vee x_1 \vee \bar{x}_0)(\bar{x}_2 \vee \bar{x}_1 \vee x_0) \quad \textbf{(KNF)}$$

$$= \overline{\overline{(x_2 \vee x_1 \vee x_0)(x_2 \vee \bar{x}_1 \vee \bar{x}_0)(\bar{x}_2 \vee x_1 \vee \bar{x}_0)(\bar{x}_2 \vee \bar{x}_1 \vee x_0)}}$$

$$= \overline{(x_2 \vee x_1 \vee x_0)} \vee \overline{(x_2 \vee \bar{x}_1 \vee \bar{x}_0)} \vee \overline{(\bar{x}_2 \vee x_1 \vee \bar{x}_0)} \vee \overline{(\bar{x}_2 \vee \bar{x}_1 \vee x_0)}$$

$$= \mathbf{NOR_4}(\mathbf{NOR_3}(x_2, x_1, x_0), \mathbf{NOR_3}(x_2, \bar{x}_1, \bar{x}_0),$$

$$\mathbf{NOR_3}(\bar{x}_2, x_1, \bar{x}_0), \mathbf{NOR_3}(\bar{x}_2, \bar{x}_1, \bar{x}_0))$$

Minimalformen

- **Boolesche Ausdrücke für eine Boolesche Funktion *f* in einer kürzestmöglichen Darstellung**
  - ⇨ **technische Realisierung mit möglichst geringen Kosten**
- **Disjunktive und konjunktive Minimalformen**
  - ⇨ **Disjunktion von Implikanten (DMF)**
  - ⇨ **Konjunktion von Implikaten (KMF)**
- **Die DMF und KMF sind nicht eindeutig**

$f(x_1, x_0) = \bar{x}_1 x_0 \vee x_1 \bar{x}_0$	**DMF**
$g(x_1, x_0) = \bar{x}_1 x_0 \vee x_1 x_0$	**keine DMF**
$= x_0$	**DMF**
$h(x_2, x_1, x_0) = (x_1 \vee \bar{x}_0) \wedge (x_2 \vee x_0)$	**KMF**
$k(x_2, x_1, x_0) = \bar{x}_0 \wedge (x_2 \vee \bar{x}_0) \wedge (x_2 \vee x_1)$	**keine KMF**
$= \bar{x}_0 \wedge (x_2 \vee x_1)$	**KMF**

Minimalformen

- **Das Finden einer Minimalform ist nicht trivial**
  - ⇨ **besonders bei Funktionen mit vielen Variablen**
  - ⇨ **oft nur suboptimale Lösungen**
  - ⇨ **Einsatz von Heuristiken**
- **Allgemeines zweischrittiges Vorgehen:**
  - ⇨ **Finden einer Menge von Implikanten bzw. Implikate mit einer möglichst geringen Anzahl von Literalen**
  - ⇨ **Auswahl aus dieser Menge, so dass deren Disjunktion bzw. Konjunktion die gesuchte Funktion erhält**

Ökonomische Kriterien für den Entwurf von Schaltnetzen

- Geringe Kosten für den Entwurf (Entwurfsaufwand)
  - ⇨ Lohnkosten
  - ⇨ Rechnerbenutzung, Softwarelizenzen
- Geringe Kosten für die Realisierung (Realisierungsaufwand)
  - ⇨ Bauelemente, Gehäuseformen
  - ⇨ Kühlung
- Geringe Kosten für die Inbetriebnahme
  - ⇨ Kosten für den Test
  - ⇨ Fertigstellung programmierbarer Bauelemente
- Geringe Kosten für den Betrieb
  - ⇨ Wartung
  - ⇨ Energie

Entwurfsziele

- Manche Kriterien stehen im Widerspruch
  - ⇨ zuverlässigere Schaltungen erfordern einen höheren Realisierungsaufwand
  - ⇨ Verringerung des Realisierungsaufwand erfordert eine Erhöhung der Entwurfskosten
- Ziel des Entwurfs ist das Finden des günstigsten Kompromisses
  - ⇨ Korrektheit der Realisierung
  - ⇨ Einhaltung der technologischen Grenzen
  - ⇨ ökonomische Kriterien

☞ Wir betrachten in dieser Vorlesung nur die Minimierung des Realisierungsaufwands

Minimierungsverfahren

- **Finden von Minimalformen Boolescher Funktionen**
  - ⇨ **ohne Betrachtung der Zieltechnologie**
  - ⇨ **mit Betrachtung der Zieltechnologie**
- **Drei Minimierungsansätze**
  - ⇨ **algebraische Verfahren**
  - ⇨ **graphische Verfahren**
  - ⇨ **tabellarische Verfahren**
- **Man unterscheidet**
  - ⇨ **exakte Minimierungsverfahren (z.B. Quine McCluskey), deren Ergebnis das absolute Minimum einer Schaltungsdarstellung ist**
  - ⇨ **heuristische Minimierungsverfahren auf der Basis von iterativen Minimierungsschritten**

Darstellung Boolescher Funktionen durch Funktionstabellen

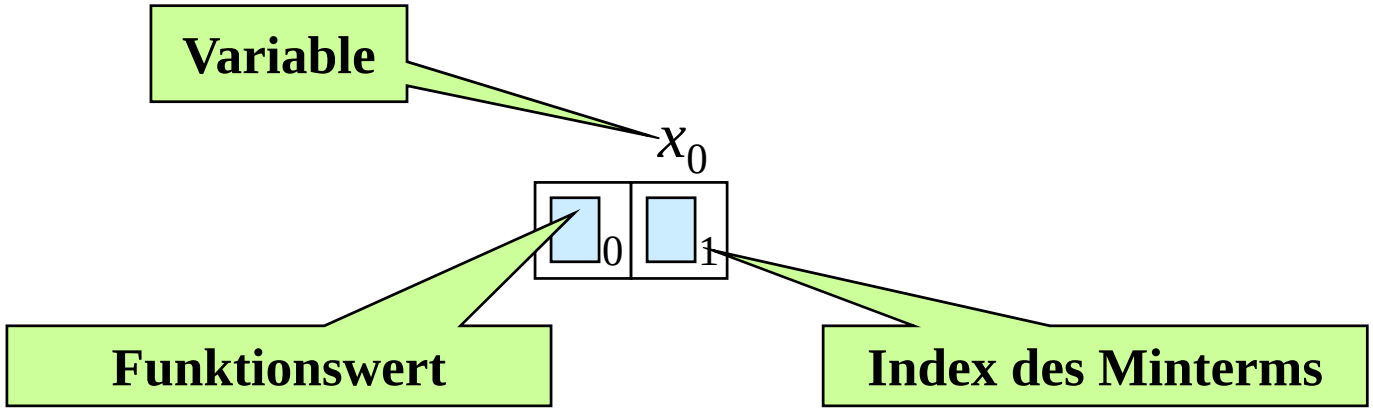
- **Darstellung des Verhaltens einer Booleschen Funktion mit Hilfe einer vollständigen Funktionstabelle**
  - ⇨ **Jeder Belegung der Booleschen Variablen wird ein Funktionswert zugeordnet**
  - ⇨ $f(x_2, x_1, x_0) \rightarrow y$, *mit* $x_i, y \in \{0,1\}$

Index	x_2	x_1	x_0	y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

$$f(x_2, x_1, x_0) = \overline{x}_2 x_1 \overline{x}_0 \vee x_2 \overline{x}_1 \overline{x}_0 \vee x_2 x_1 \overline{x}_0 \vee x_2 x_1 x_0$$

KV-Diagramme

- Nach Karnaugh und Veitch
- Möglichkeit, Boolesche Funktionen übersichtlich darzustellen
  - ⇨ bis 6 Variablen praktisch einsetzbar
- Ausgangspunkt ist ein Rechteck mit 2 Feldern



KV-Diagramme

- **Beispiele**

	x_0
	0_0 | 1_1

$f(x_0) = x_0$

	x_0
	1_0 | 0_1

$f(x_0) = \bar{x}_0$

- **Erweiterung durch Spiegelung**
  - ⇨ **für jede zusätzliche Variable verdoppelt sich die Zahl der Felder**

	x_0
0	1
2	3

x_1

	$\overline{x_0}$		
0	1	5	4
2	3	7	6

x_1

$\overline{x_2}$

	$\overline{x_0}$		
0	1	5	4
2	3	7	6
10	11	15	14
8	9	13	12

x_3 | x_1

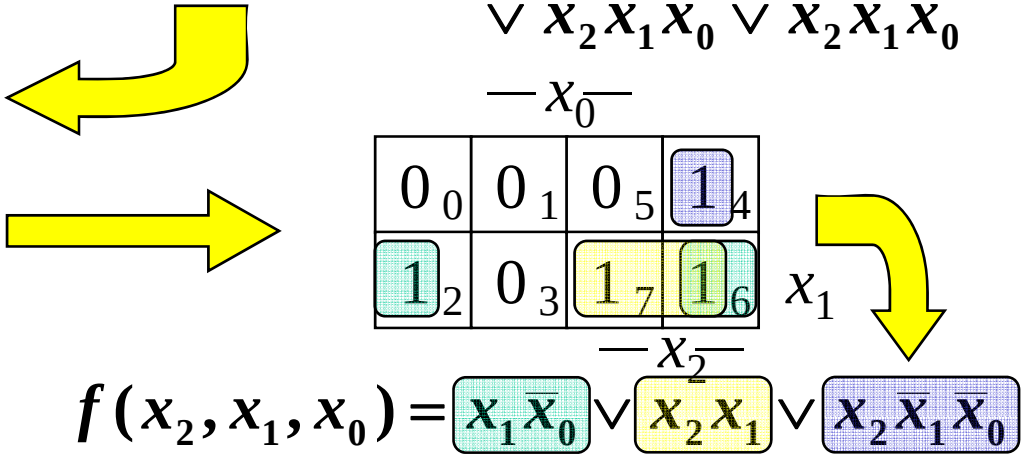
$\overline{x_2}$

Eigenschaften von KV-Diagrammen

- Jedes Feld ist ein Funktionswert
  - ⇨ Ein Minterm der Funktion
  - ⇨ Eindeutige Variablenzuordnung
- Oft werden x_1 und x_2 vertauscht
  - ⇨ Lediglich eine andere Numerierung der Felder
  - ⇨ Kein Einfluss auf das Minimierungsverfahren
- Aufstellen der KV-Diagramme über die Funktionstabelle:

Index	x_2	x_1	x_0	y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

$$f(x_2, x_1, x_0) = \bar{x}_2 x_1 \bar{x}_0 \vee x_2 \bar{x}_1 \bar{x}_0 \vee x_2 x_1 \bar{x}_0 \vee x_2 x_1 x_0$$



$$f(x_2, x_1, x_0) = x_1 \bar{x}_0 \vee x_2 x_1 \vee x_2 \bar{x}_1 \bar{x}_0$$

KV-Diagramme über die KNF

- **Argumentation über die Nullstellen der Funktion**
  - ⇨ **Jede Nullstelle entspricht einem Maxterm**
- **Beispiel**

$$f(x_2, x_1, x_0) = x_1\bar{x}_0 \vee x_2x_1 \vee x_2\bar{x}_1\bar{x}_0$$

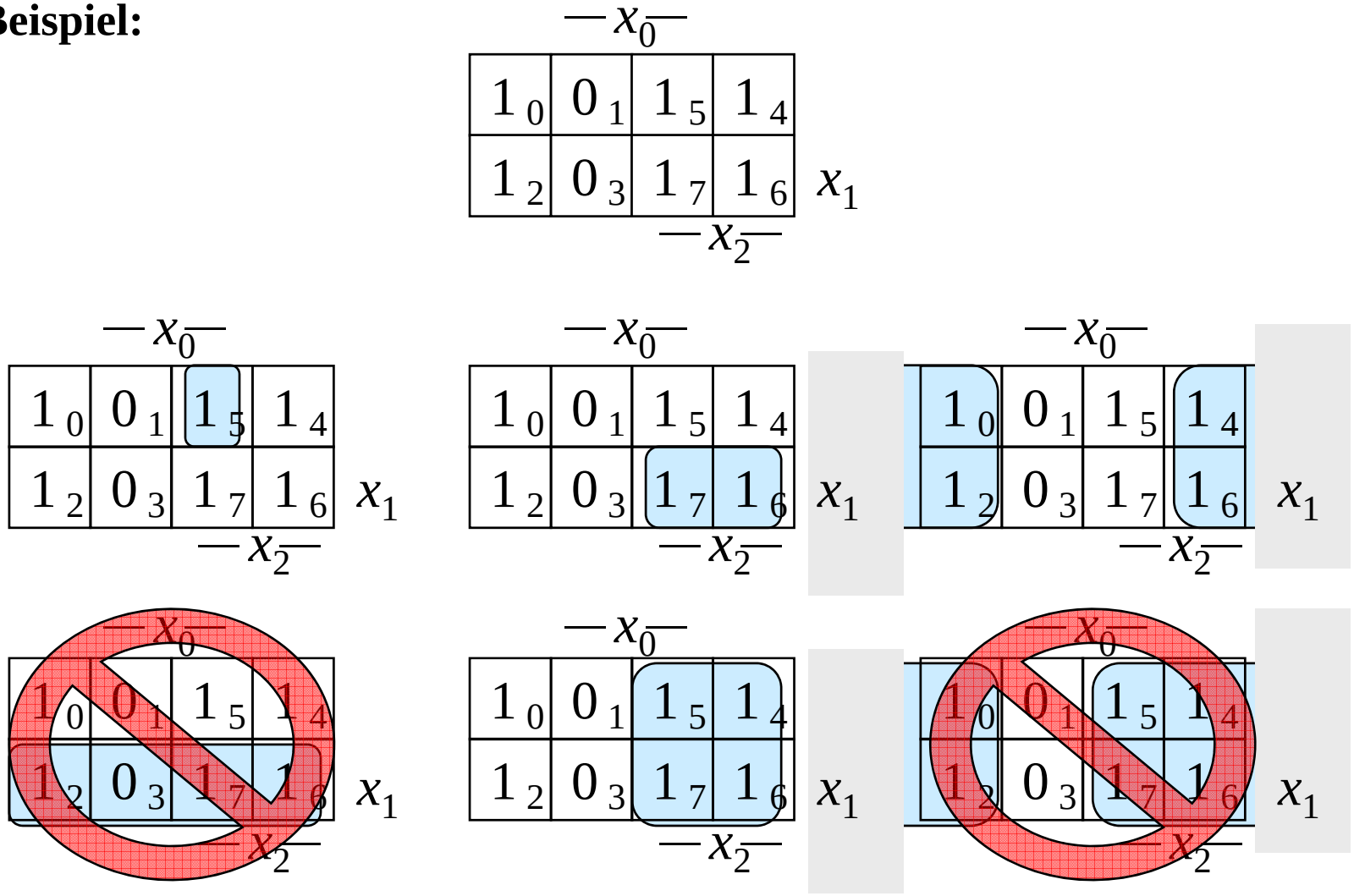
$\overline{x_0}$			
0_0	0_1	0_5	1_4
1_2	0_3	1_7	1_6
		$\overline{x_2}$	

x_1

$$f(x_2, x_1, x_0) = (x_2 \vee x_1 \vee x_0) \wedge (x_2 \vee x_1 \vee \bar{x}_0) \wedge$$
$$(x_2 \vee \bar{x}_1 \vee \bar{x}_0) \wedge (\bar{x}_2 \vee x_1 \vee \bar{x}_0)$$

Minimalformen aus KV-Diagrammen

- **Zusammenfassen von Mintermen zu Implikanten**
- **Beispiel:**



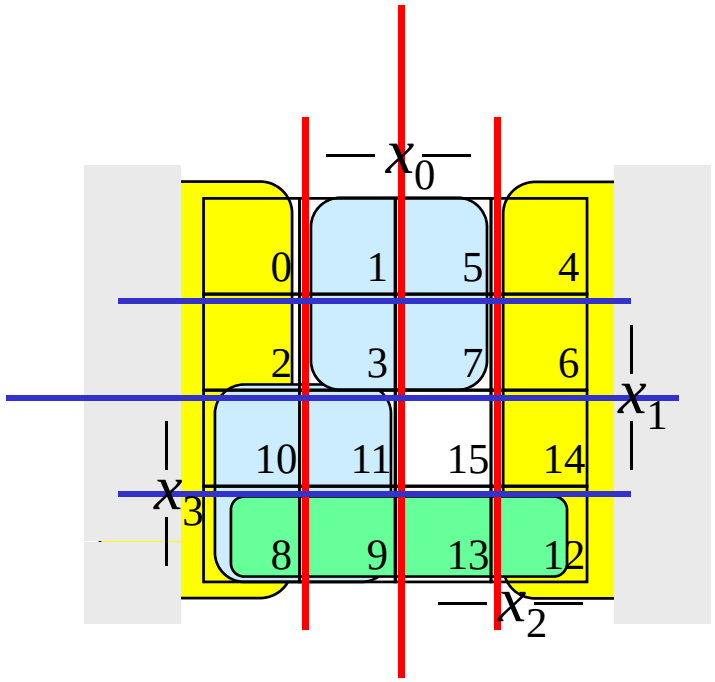
Implikant k-ter Ordnung

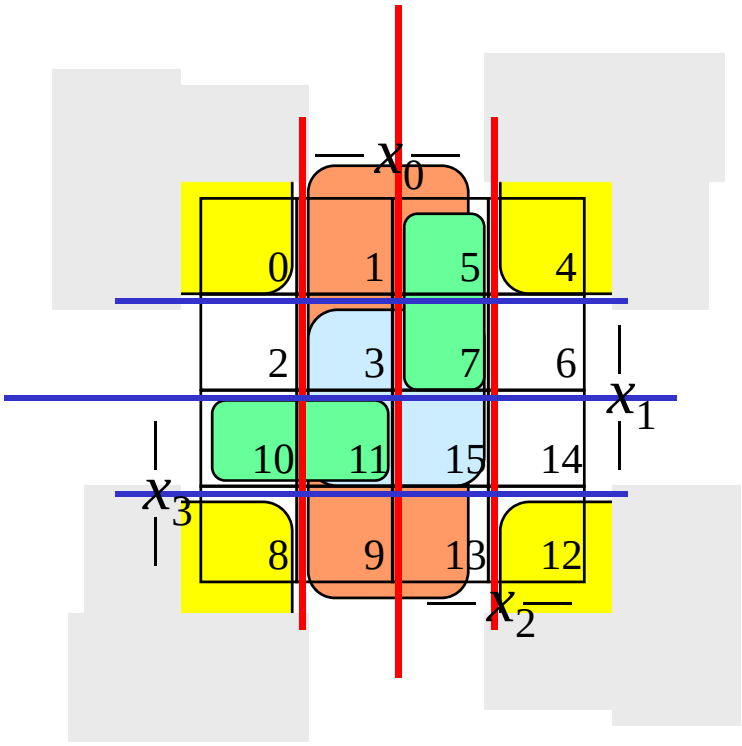
Def. 10.1: Es sei eine Boolesche Funktion $f(x_0,\ldots,x_{n-1}):B^n \rightarrow B$ gegeben. Ein Implikant k-ter Ordnung umfaßt 2^k Felder eines KV-Diagramms.

- **Man erhält**
  - ⇨ **Implikanten 0-ter Ordnung** — **Minterme**
  - ⇨ **Implikanten 1-ter Ordnung** — **Zusammenfassung zweier Minterme**
  - ⇨ **Implikanten 2-ter Ordnung** — **Zusammenfassung zweier Implikanten 1-ter Ordnung**
  - ⇨ **usw.**

Finden möglicher Zusammenfassungen

- **Finden von 1-Blöcken, die symmetrisch zu denjenigen Achsen, an denen eine Variable von 0 auf 1 wechselt**
- **Jede Funktion läßt sich als disjunktive Verknüpfung solcher Implikanten darstellen**
- **Beispiele**



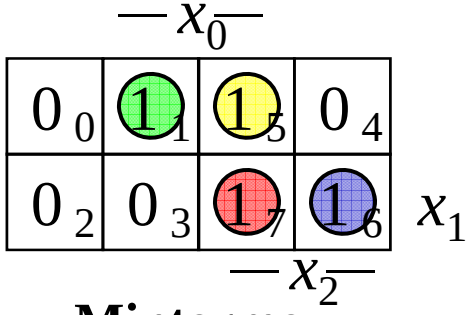


Primimplikant

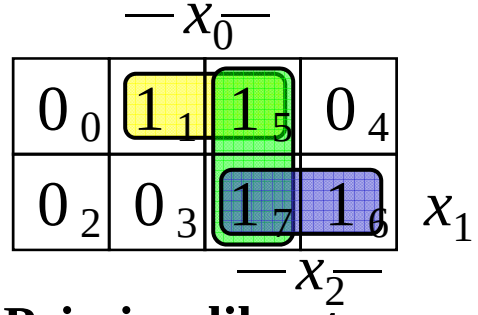
Def. 10.2: Es sei eine Boolesche Funktion $f(x_0,\ldots,x_{n-1}):B^n \rightarrow B$ gegeben. Ein Implikant p heißt Primimplikant, wenn es keinen Implikanten q gibt, der p impliziert.

- **Ein Primimplikant p ist von größtmöglicher Ordnung**
  - ⇨ **Primimplikanten sind einfach aus einem KV-Diagramm herauszulesen**
  - ⇨ **man sucht die größtmöglichen Implikanten**

$$f(x_2,x_1,x_0) = x_2x_1\bar{x}_0 \vee x_2x_1x_0 \vee x_2\bar{x}_1x_0 \vee \bar{x}_2\bar{x}_1x_0$$



Minterme



Primimplikanten

$$f(x_2,x_1,x_0) = x_2x_1 \vee x_2x_0 \vee \bar{x}_1x_0$$

Überdeckung

Satz 10.1: Zu jeder Booleschen Funktion *f* gibt es eine minimale Überdeckung aus Primimplikanten

Bew. (Skizze):

Angenommen wir haben eine minimale Überdeckung der Funktion, die einen Implikanten *k* besitzt, der kein Primimplikant ist.

⇒ Dieser Implikant *k* kann durch einen Primimplikant *p* ersetzt werden, der *k* enthält

⇒ Das Ergebnis ist eine Überdeckung der Funktion *f* aus Primimplikanten mit der gleichen Anzahl von Termen

⇒ Die Überdeckung ist minimal

- **Einschränkung des Suchraums**
  - ⇨ **man braucht nur die Primimplikanten für die Minimierung betrachten**

Kernprimimplikant

Def. 10.3: Es sei eine Boolesche Funktion $f(x_0,\ldots,x_{n-1}):B^n \rightarrow B$ gegeben. Ein Implikant p heißt Kernprimimplikant, wenn er einen Minterm überdeckt, der von keinem anderen Primimplikant überdeckt wird.

- **Man nennt solche Primimplikanten auch essentielle Primimplikanten**
  - ⇨ **Ein Kernprimimplikant muss auf jeden Fall in der disjunktiven Minimalform vorkommen**
- **Ziel der Minimierung:**
  - ⇨ **Überdecken der Funktion durch Kernprimimplikanten und möglichst wenigen zusätzlichen Primimplikanten**
- **Zwei Schritte**
  1. **Finde alle Primimplikanten**
  2. **Suche eine Überdeckung der Funktion mit möglichst wenigen Primimplikanten**

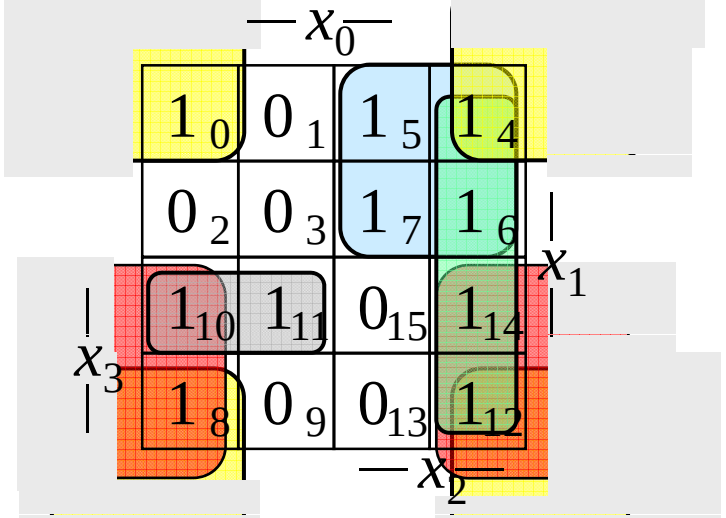
Beispiel

$$f(x_3, x_2, x_1, x_0) = \bar{x}_3\bar{x}_2\bar{x}_1\bar{x}_0 \vee \bar{x}_3x_2\bar{x}_1\bar{x}_0 \vee \bar{x}_3x_2\bar{x}_1x_0 \vee$$

$$\bar{x}_3x_2x_1\bar{x}_0 \vee \bar{x}_3x_2x_1x_0 \vee x_3\bar{x}_2\bar{x}_1\bar{x}_0 \vee \qquad \textbf{DNF}$$

$$x_3\bar{x}_2x_1\bar{x}_0 \vee x_3\bar{x}_2x_1x_0 \vee x_3x_2\bar{x}_1\bar{x}_0 \vee x_3x_2x_1\bar{x}_0$$

$$= \mathit{MINt}(0, 4, 5, 6, 7, 8, 10, 11, 12, 14)$$

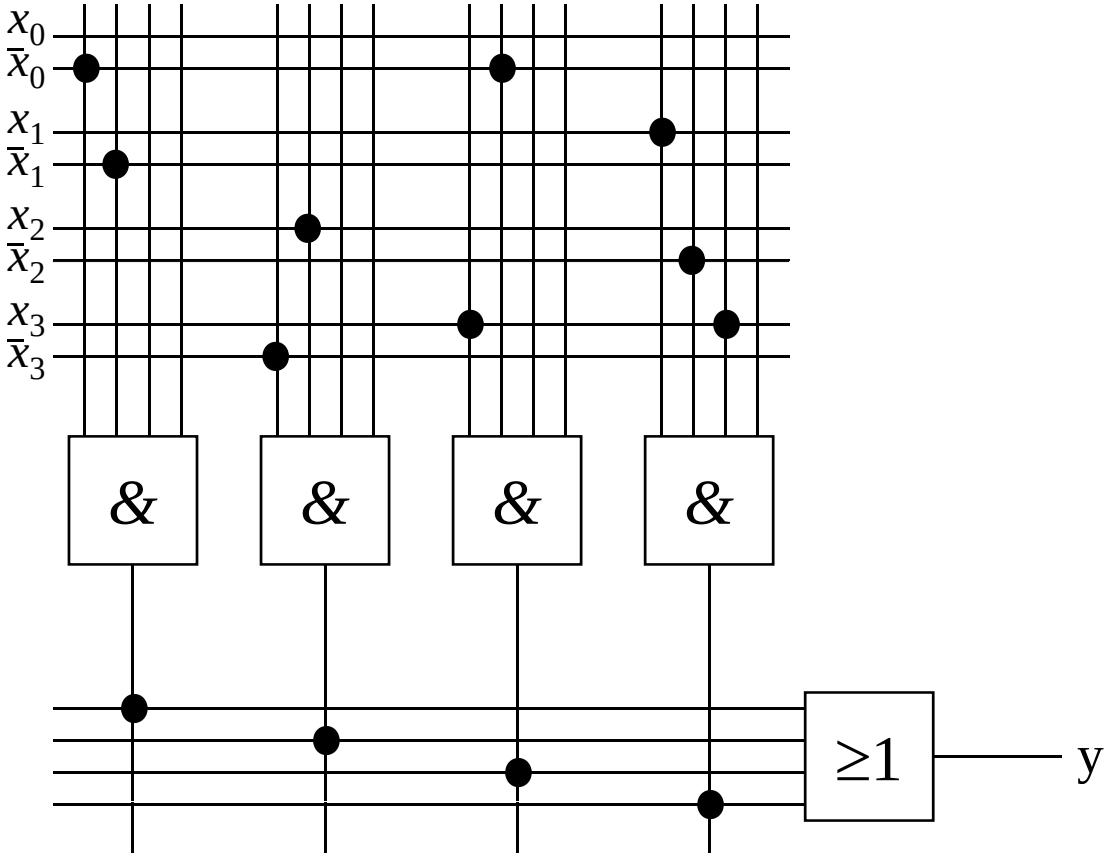


e	$\bar{x}_1\bar{x}_0$	(0,4,8,12)
e	$\bar{x}_3x_2$	(4,5,6,7)
	$x_2\bar{x}_0$	(4,6,12,14)
	$x_3\bar{x}_0$	(8,10,12,14)
e	$x_3\bar{x}_2x_1$	(10,11)

$$f(x_3, x_2, x_1, x_0) = \bar{x}_1\bar{x}_0 \vee \bar{x}_3x_2 \vee x_3\bar{x}_0 \vee x_3\bar{x}_2x_1$$

$$= \bar{x}_1\bar{x}_0 \vee \bar{x}_3x_2 \vee x_2\bar{x}_0 \vee x_3\bar{x}_2x_1 \qquad \textbf{DMF}$$

Realisierung als „Programmable Logic Array (PLA)



$$f(x_3, x_2, x_1, x_0) = \bar{x}_1\bar{x}_0 \vee \bar{x}_3 x_2 \vee x_3\bar{x}_0 \vee x_3\bar{x}_2 x_1$$

Bündelminimierung

- **Funktionen mit mehreren Ausgängen werden gemeinsam minimiert**
  - ⇨ **gemeinsame Implikanten sollten mehrfach genutzt werden**
- **Beispiel: Transformation eines Codes**

- **Transformationstabelle**

System1			System2	
c	b	a	x	y
0	0	0	1	0
0	0	1	0	0
0	1	1	0	1
0	1	0	0	0
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	1

x:

$\overline{x_0}$			
1_0	0_1	1_5	1_4
0_2	0_3	0_7	0_6

x_1 $\overline{x_2}$

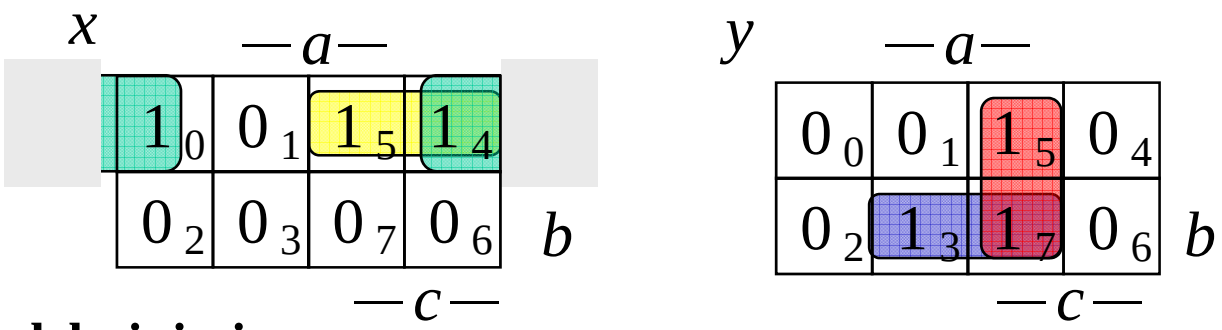
y:

$\overline{x_0}$			
0_0	0_1	1_5	0_4
0_2	1_3	1_7	0_6

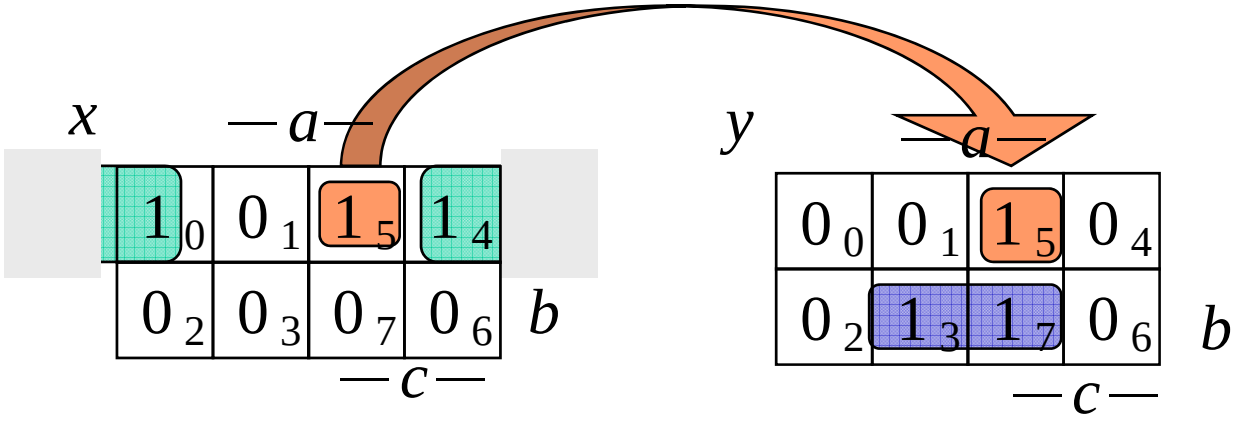
x_1 $\overline{x_2}$

Bündelminimierung

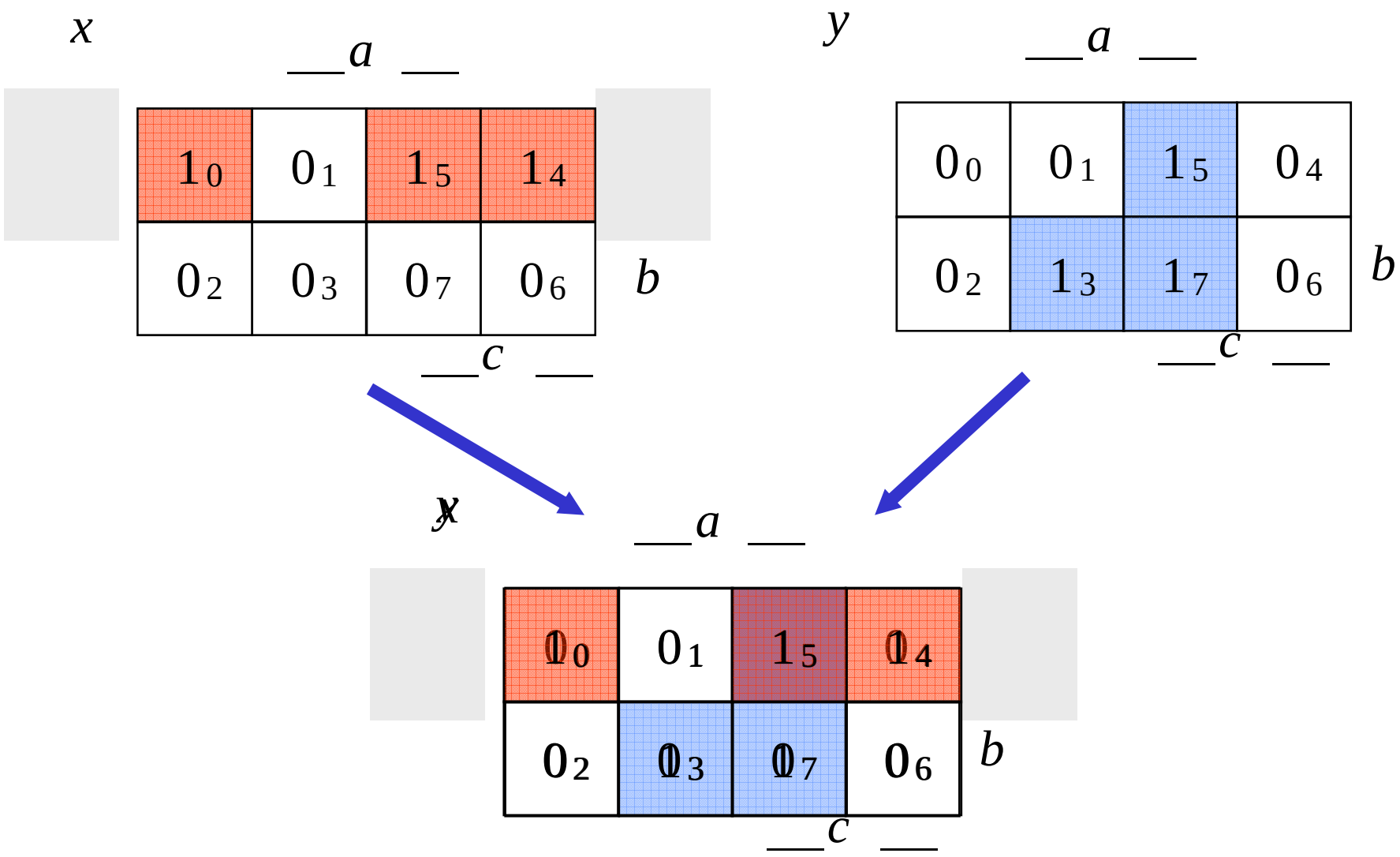
- **Getrennte Minimierung**
  - ⇨ **insgesamt 4 Implikanten für die Realisierung**
    - **Je 2 pro Ausgang**



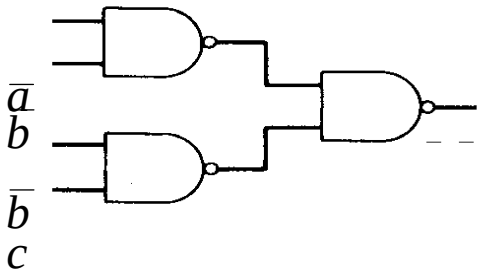
- **Bündelminimierung**
  - ⇨ **insgesamt 3 Implikanten für die Realisierung**

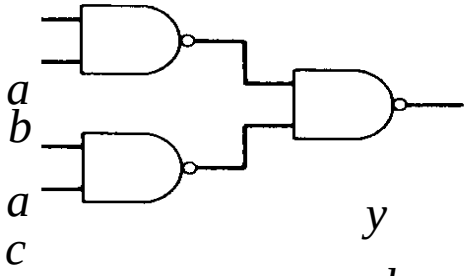


Bündelminimierung



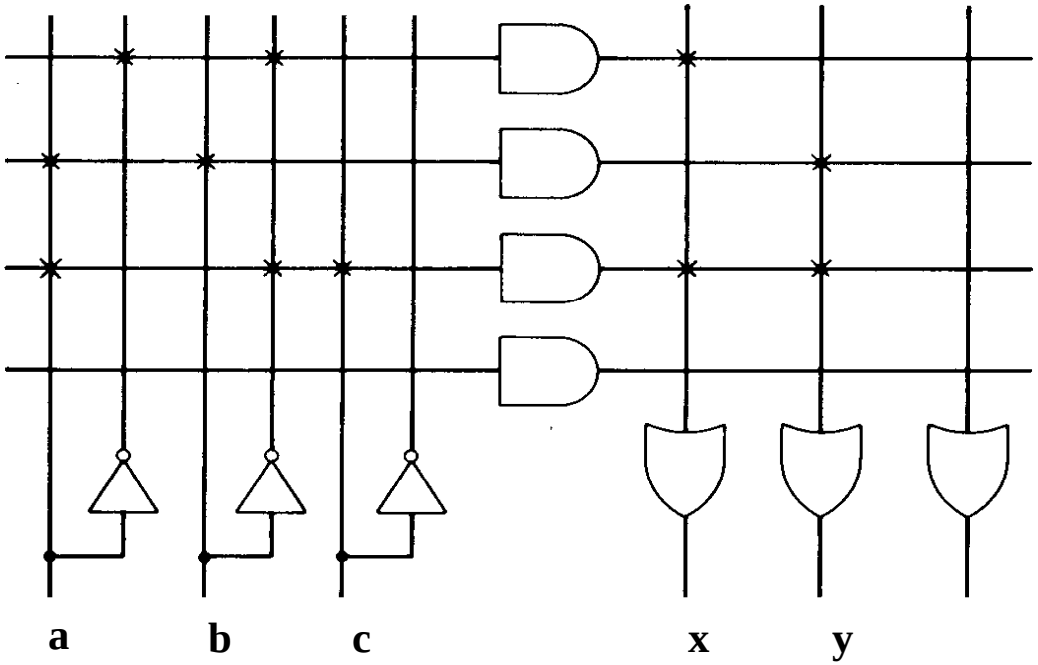
Bündelminimierung





getrennte Minimierung

6 Gatter



Bündelminimierung

5 Gatter

$x = \bar{a}\bar{b} \vee a\bar{b}c$

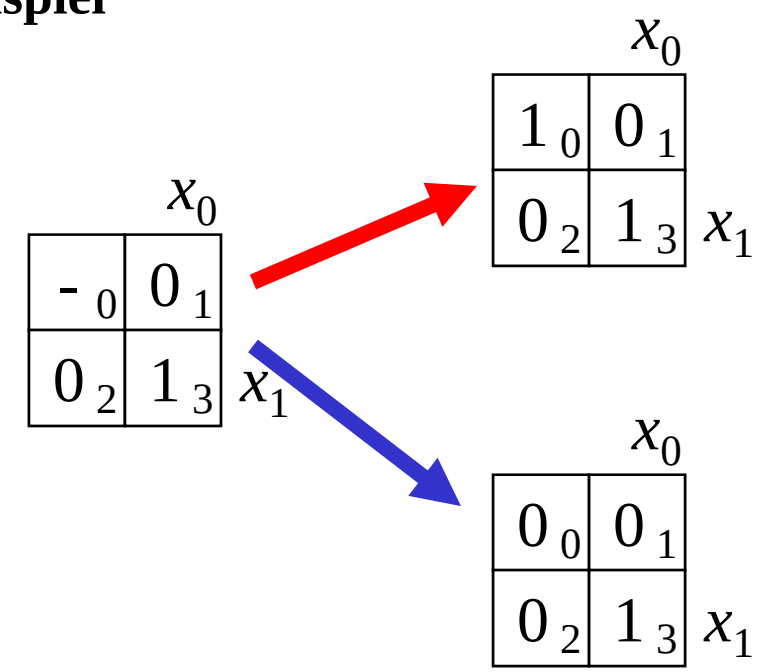
$y = ab \vee a\bar{b}c$

Unvollständig definierte Funktionen

- **Bisher war für alle Belegungen der Eingänge ein Funktionswert festgelegt**
  - ⇨ **in praktischen Fällen kommt es sehr häufig vor, dass die Funktionswerte für bestimmte Eingangsbelegungen frei wählbar sind**
  - ⇨ **diese Funktionswerte sind frei verfügbar**
- **Solche Funktionen heißen unvollständig oder partiell definierte Funktionen**
  - ⇨ **die nicht verwendeten Eingangsbelegungen heißen auch Don‘t-care-Belegungen**
  - ⇨ **in KV-Diagrammen werden diese Felder mit einem „-“ gekennzeichnet**
- **wichtiges Potential für die Minimierung!**
  - ⇨ **um eine DMF zu erhalten, müssen diese mit „0“ oder „1“ belegt werden**

Minimierung unvollständiger Boolescher Funktionen

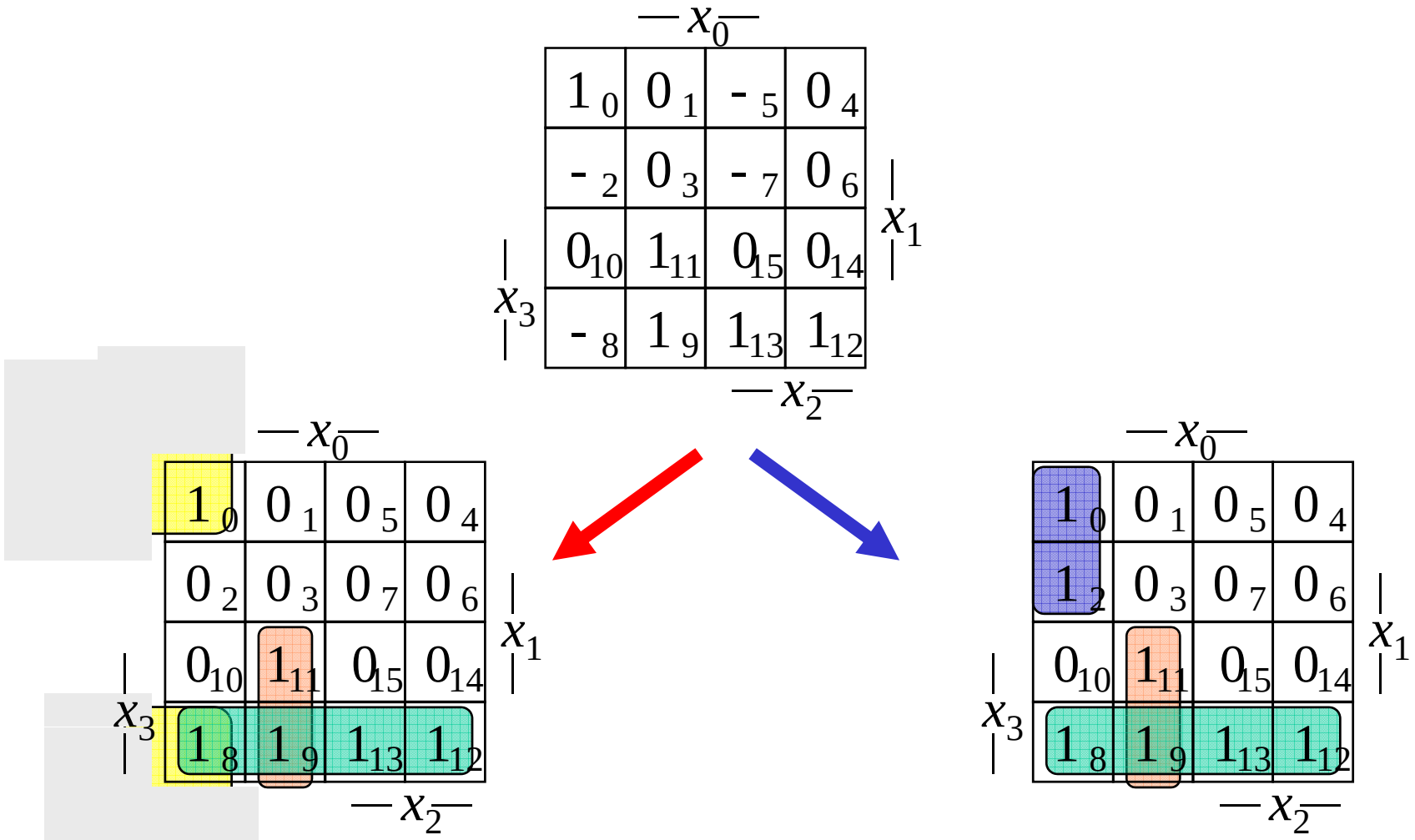
- **Beispiel**



$$f(x_1, x_0) = x_1 x_0 \vee \bar{x}_1 \bar{x}_0$$

$$f(x_1, x_0) = x_1 x_0$$

Minimierung unvollständiger Boolescher Funktionen



Das Verfahren nach Quine-McCluskey

- **KV-Diagramme mit mehr als 6 Variablen werden sehr groß und unübersichtlich**
  - ⇨ **dieses Problem wurde zuerst von Quine und McCluskey erkannt und gelöst**
  - ⇨ **das Verfahren nach Quine-McCluskey ist ein tabellarisches Verfahren**
  - ⇨ **es führt auf eine DMF (disjunktive minimale Form)**
- **Ausgangspunkt ist die Funktionstabelle der Funktion**
  - ⇨ **nur die Minterme werden berücksichtigt**
- **Der Suchraum wird eingeschränkt, weil der Satz 1.1 gilt:**
  - ⇨ **zu jeder Booleschen Funktion *f* gibt es eine minimale Überdeckung aus Primimplikanten**
- **Verfahren nach Quine-McCluskey in 2 Schritten:**
  1. **Schritt: berechne alle Primimplikanten**
  2. **Schritt: suche eine minimale Überdeckung aller Minterme**

Beispiel: Die vollständige Funktionstabelle

Nr.	e	d	c	b	a	y
0	0	0	0	0	0	0
1	0	0	0	0	1	0
2	0	0	0	1	0	1
3	0	0	0	1	1	0
4	0	0	1	0	0	1
5	0	0	1	0	1	1
6	0	0	1	1	0	1
7	0	0	1	1	1	0
8	0	1	0	0	0	0
9	0	1	0	0	1	0
10	0	1	0	1	0	1
11	0	1	0	1	1	0
12	0	1	1	0	0	1
13	0	1	1	0	1	1
14	0	1	1	1	0	1
15	0	1	1	1	1	0

Nr.	e	d	c	b	a	y
16	1	0	0	0	0	0
17	1	0	0	0	1	0
18	1	0	0	1	0	1
19	1	0	0	1	1	0
20	1	0	1	0	0	0
21	1	0	1	0	1	0
22	1	0	1	1	0	1
23	1	0	1	1	1	0
24	1	1	0	0	0	0
25	1	1	0	0	1	0
26	1	1	0	1	0	1
27	1	1	0	1	1	0
28	1	1	1	0	0	0
29	1	1	1	0	1	0
30	1	1	1	1	0	1
31	1	1	1	1	1	0

1. Schritt: Berechnung aller Primimplikanten

- **Schreibweise**
  - ⇨ **`1`** **steht für eine nicht negierte Variable**
  - ⇨ **`0`** **steht für eine negierte Variable**
  - ⇨ **`-`** **steht für eine nicht auftretende Variable**
- **Man betrachtet nur die Minterme**
  - ⇨ **1-Stellen der Funktion**
- **Die Minterme werden geordnet**
  - ⇨ **Gruppen mit der gleichen Anzahl von Einsen**
  - ⇨ **innerhalb der Gruppen: aufsteigende Reihenfolge**
  - ⇨ **man erhält die 1. Quinesche Tabelle, 0. Ordnung**
- **Minterme benachbarter Gruppen die sich nur in 1 Variable unterscheiden werden gesucht**
  - ⇨ **diese können durch Streichen der Variable zusammengefaßt werden**
  - ⇨ **man erhält die 1. Quineschen Tabellen höherer Ordnung**

Beispiel: 1. Quinesche Tabelle

Nr.	e	d	c	b	a
2	0	0	0	1	0
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
10	0	1	0	1	0
12	0	1	1	0	0
18	1	0	0	1	0
13	0	1	1	0	1
14	0	1	1	1	0
22	1	0	1	1	0
26	1	1	0	1	0
30	1	1	1	1	0

0. Ordnung

Nr.	e	d	c	b	a
2,6	0	0	-	1	0
2,10	0	-	0	1	0
2,18	-	0	0	1	0
4,5	0	0	1	0	-
4,6	0	0	1	-	0
4,12	0	-	1	0	0
5,13	0	-	1	0	1
6,14	0	-	1	1	0
6,22	-	0	1	1	0
10,14	0	1	-	1	0
10,26	-	1	0	1	0
12,13	0	1	1	0	-
12,14	0	1	1	-	0
18,22	1	0	-	1	0
18,26	1	-	0	1	0
14,30	-	1	1	1	0
22,30	1	-	1	1	0
26,30	1	1	-	1	0

1. Ordnung

Nr.	e	d	c	b	a	
2,6,10,14	0	-	-	1	0	
2,6,18,22	-	0	-	1	0	
2,10,18,26	-	-	0	1	0	
4,5,12,13	0	-	1	0	-	*A*
4,6,12,14	0	-	1	-	0	*B*
6,14,22,30	-	-	1	1	0	
10,14,26,30	-	1	-	1	0	
18,22,26,30	1	-	-	1	0	

2. Ordnung

Nr.	e	d	c	b	a	
2,6,10,14						
18,22,26,30	-	-	-	1	0	*C*

3. Ordnung

2. Schritt: Suche einer minimalen Überdeckung

- **Aufstellen der 2. Quineschen Tabelle**
  - ⇨ **alle Primimplikanten werden zusammen mit der Nummer des Minterms aus dem sie hervorgegangen sind in eine Überdeckungstabelle eingetragen**
- **Kosten für einen Primimplikanten:**
  - ⇨ **Anzahl der UND-Eingänge (Anzahl der Variablen des Terms)**

Primimplikant	2	4	5	6	10	12	13	14	18	22	26	30	Kosten
A		X	X			X	X						3
B		X		X		X		X					3
C	X			X	X			X	X	X	X	X	2

- **Aufgabe: Finden einer Überdeckung aller Minterme mit minimalen Kosten**

Systematische Lösung des Überdeckungsproblems

- **Aufstellung einer Überdeckungsfunktion $\ddot{u}_f$**
  - ⇨ **w_A, w_B und w_C sind Variablen, die kennzeichnen, ob ein entsprechender Primimplikant in der vereinfachten Darstellung aufgenommen wird, oder nicht**
  - ⇨ **Konjunktive Form über alle den jeweiligen Minterm überdeckenden Primimplikanten**

Primimplikant	2	4	5	6	10	12	13	14	18	22	26	30
A		X	X			X	X					
B		X		X		X		X				
C	X			X	X			X	X	X	X	X

$$\ddot{u}_f = w_C(w_A \vee w_B)w_A(w_B \vee w_C)w_C(w_A \vee w_B)w_A(w_B \vee w_C)w_C w_C w_C w_C$$

$$= w_C(w_A \vee w_B)w_A(w_B \vee w_C)$$

$$= (w_C w_A \vee w_C w_B)(w_A w_B \vee w_A w_C)$$

$$= w_C w_B w_A \vee w_A w_C$$

$$(= w_A w_C)$$

Systematische Lösung des Überdeckungsproblems

- **Ergebnis nach der Vereinfachung:** $\ddot{u}_f = w_C w_B w_A \vee w_A w_C$

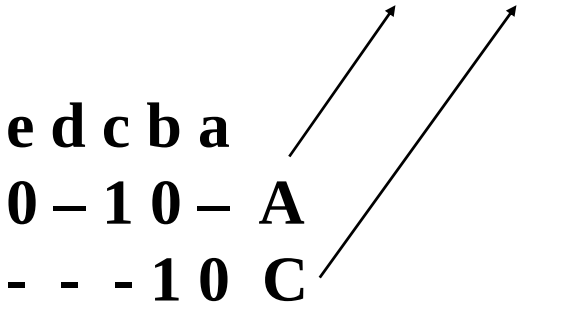
- **Damit f ganz überdeckt ist, muss $\ddot{u}_f$ eine Tautologie sein**
  - ⇨ **man sucht einen konjunktiven Term mit minimalen Kosten**

$w_C w_B w_A$ **Kosten :** $3+3+2=8$

$w_A w_C$ **Kosten :** $3+2=5$

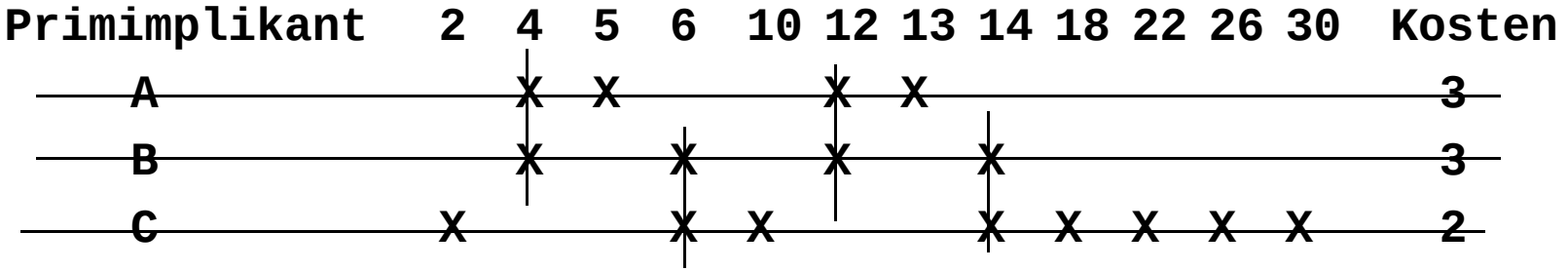
- **Als Endergebnis der Minimierung für die Funktion f erhält man**

$$f(e,d,c,b,a) = \bar{e}c\bar{b} \vee b\bar{a}$$



Vereinfachung des Überdeckungsproblems

- Die Primimplikantentabelle kann reduziert werden, indem essentielle Primterme (Kernprimimplikanten) und die von ihnen überdeckten Minterme gestrichen werden
  - ⇨ tragen mit einem einzigen „X“ zu einer Spalte bei
  - ⇨ müssen auf jeden Fall in der Überdeckung enthalten sein
- In diesem Beispiel sind dies die beiden Primimplikanten A und C



  - ⇨ A: 5, 13
  - ⇨ C: 2, 10, 18, 22, 26, 30
  - ⇨ B ist vollständig überdeckt und kann ebenfalls gestrichen werden

Aufwandsbetrachtungen

- **Alle Verfahren benötigen 2 Schritte**
  - ⇨ **1. Erzeugen aller Primimplikanten (Primimplikate)**
  - ⇨ **2. Auswahl der Primiplikanten (Primimplikate), welche die Minterme (Maxterme) mit minimalen Kosten überdecken**
- **Die Anzahl der Primimplikanten (Primimplikaten) kann exponentiell steigen**
  - ⇨ **Es gibt Funktionen mit** $\frac{3^n}{n}$ **Primimplikanten**
- **Das Überdeckungsproblem ist NP-Vollständig**
  - ⇨ **es besteht wenig Hoffnung einen Algorithmus zu finden, der dieses Problem polynomial mit der Zahl der Eingabevariablen löst**

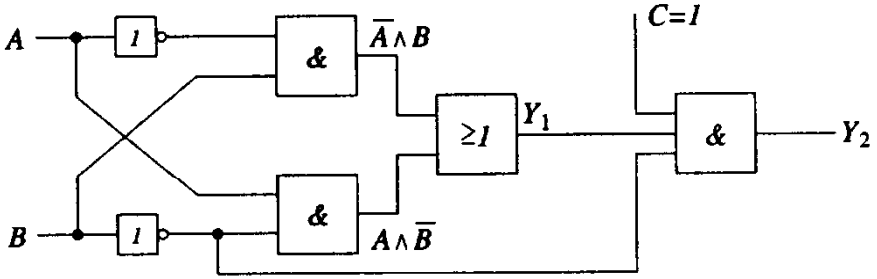
Heuristische Verfahren

- **Heuristische Minimierungsverfahren werden eingesetzt,**
  - ⇨ **wenn die zweistufige Darstellung optimiert werden muss, aber**
  - ⇨ **nur begrenzte Rechenzeit und Speicherplatz zur Verfügung steht**
- **Die meisten heuristischen Minimierungsansätze basieren auf einer schrittweisen Verbesserung der Schaltung**
- **Unterschiede zu exakten Verfahren:**
  - ⇨ **man wendet eine Menge von Transformationen direkt auf die Überdeckung des *ON-Sets* an**
  - ⇨ **man definiert die Optimierung als beendet, wenn diese Transformationen keine Verbesserungen mehr bringen**

Laufzeiteffekte in Schaltnetzen

- **Bisher wurden Schaltnetze mit idealen Verknüpfungsgliedern betrachtet**
  - ⇨ **die Verknüpfungsgliedern besaßen keine Signallaufzeit**
- **Bei realen Verknüpfungsgliedern dürfen Signallaufzeiten nicht vernachlässigt werden**
  - ⇨ **Schaltvariablen können Werte annehmen, die theoretisch oder bei idealen Verknüpfungsgliedern nie auftreten könnten**
- **Solche Störimpulse nennt man Hazards**
  - ⇨ **sie treten als Antwort auf die Änderung der Werte der Eingangsvariablen auf**

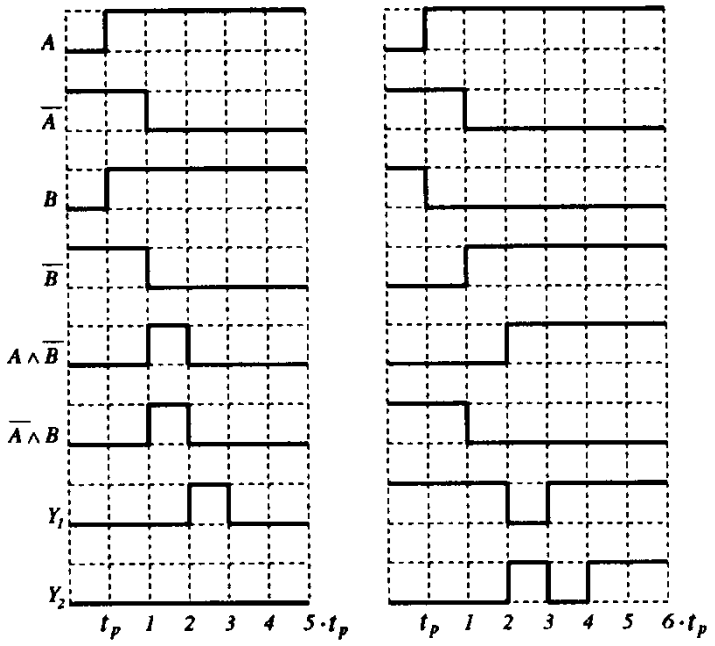
Entstehung von Hazards



a) Schaltnetz

B	A	Y_1	Y_2
0	0	0	0
1	1	0	0
1	0	1	0
0	1	1	1

b) Funktionstabelle



a) Impulsdiagramm

Statische Hazards

- **Statische Hazards sind Störimpulse aus einer Verknüpfung, die theoretisch konstant Null oder Eins liefern müsste**

$X_t \wedge \overline{X}_{t-k}$ **müsste Null liefern**
statischer 1-Hazard bei einem Übergang von X: 0→1

$X_t \vee \overline{X}_{t-k}$ **müsste Eins liefern**
statischer 0-Hazard bei einem Übergang von X: 1→0

Dynamische Hazards

- **Dynamische Hazards entstehen als zusätzliche Übergänge beim Ausgang eines Schaltnetzes**
- $X_t \wedge \overline{X}_{t-k} \vee X_l$ **, mit** $l > k$
  - ⇨ **bei einem Übergang von** $X{=}0 \rightarrow X{=}1$ **darf am Ausgang nur ein zu** $X_{t\text{-}l}$ **synchroner 0 →1 Übergang auftreten**
  - ⇨ **durch den vorgeschalteten statischen Hazard kommt es aber zu einer zusätzlichen 0 →1 Flanke**
- $X_t \wedge (\overline{X}_{t-k} \vee X_l)$ **, mit** $l > k$
  - ⇨ **bei einem Übergang von** $X{=}0 \rightarrow X{=}1$ **darf am Ausgang nur ein zu** X_t **synchroner 0 →1 Übergang auftreten**
  - ⇨ **durch den vorgeschalteten statischen Hazard kommt es aber zu einer zusätzlichen 0 →1 Flanke**

Klassifikation von Hazards

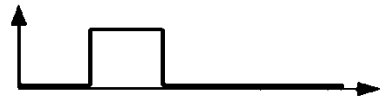
statt

$Y_1 =$ 0 0 0 0 0

oder

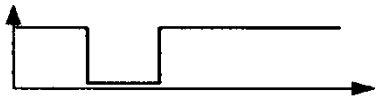
$Y_1 =$ 1 1 1 1 1

entsteht



$Y_1 =$ 0 1 0 0 0

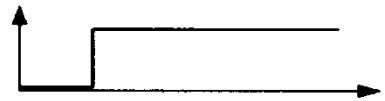
statischer 1 - Hazard



$Y_1 =$ 1 0 1 1 1

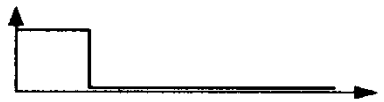
statischer 0 - Hazard

statt



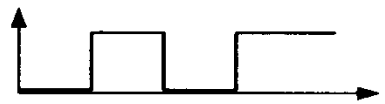
$Y_2 =$ 0 1 1 1 1

oder



$Y_2 =$ 1 0 0 0 0

entsteht



$Y_2 =$ 0 1 0 1 1

dynamischer 0 - Hazard

$Y_2 =$ 1 0 1 0 0

dynamischer 1 - Hazard

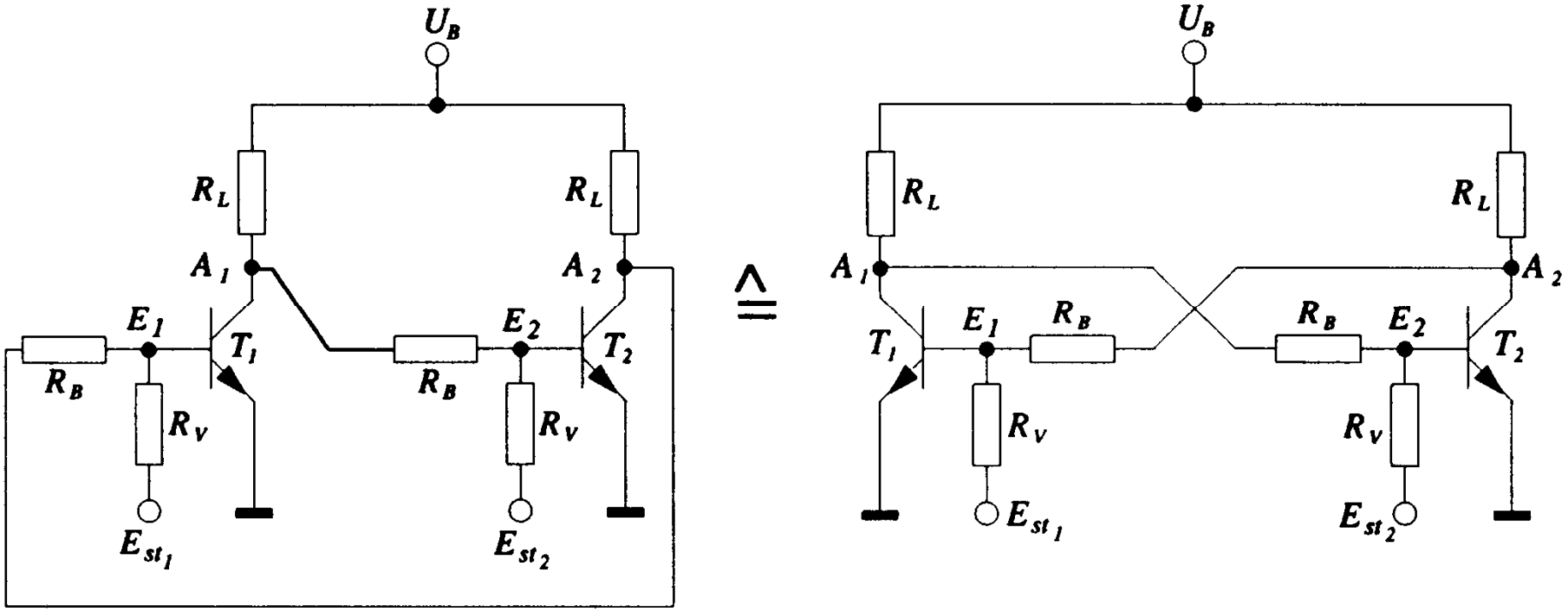
Behebung von Hazards

- Hazards können die Funktion von Schaltnetzen stören
  - ⇨ falsche Werte können an den Eingang eines Schaltnetzes zurückgekoppelt werden
- Um solche Fehler zu vermeiden werden taktflankengetriggerte Speicherglieder in die Rückkopplung eingefügt
- Die Signale werden erst übernommen, wenn die Hazards abgeklungen sind
  - ⇨ nur stabile, gültige Werte werden übernommen
  - ⇨ synchrone Schaltwerke: Schaltwerke, die durch einen zentralen Takt gesteuert werden
- Hazards haben einen Einfluss auf die maximale Schaltgeschwindigkeit
  - ⇨ maximaler Takt
  - ⇨ Entfernung von Hazards führt zu einer Erhöhung der Geschwindigkeit einer Schaltung

Speicherglieder

- Speicherglieder dienen der Aufnahme, Speicherung und Abgabe von Schaltvariablen
  - ⇨ Ein Speicherglied ist ein bistabiles Kippglied
  - ⇨ Flipflop
- Zwei Zustände
  - ⇨ Zustand 1: Setzzustand
  - ⇨ Zustand 0: Rücksetzzustand
- Übernahme des Zustands kann erfolgen
  - ⇨ taktunabhängig (nicht taktgesteuert)
  - ⇨ taktabhängig (taktgesteuert)
    - taktzustandsgesteuert
    - taktflankengesteuert
- Die unterschiedlichen Arten der Ansteuerungen führen zu unterschiedlichen Flipflop-Typen

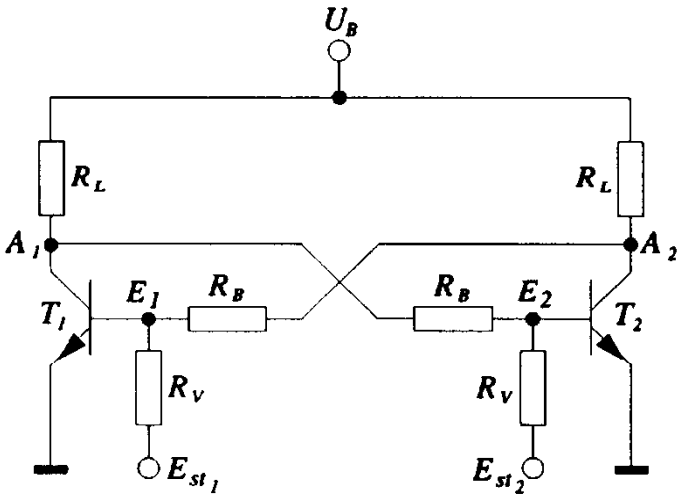
Funktionsprinzip



- **Rückkopplung**
  - ⇨ **Wirkprinzip aller bistabilen Kippschaltungen**
  - ⇨ **Ein Kippvorgang eines stabilen Zustands in den anderen wird durch E_{st1} und E_{st2} ausgelöst**

Funktionsprinzip

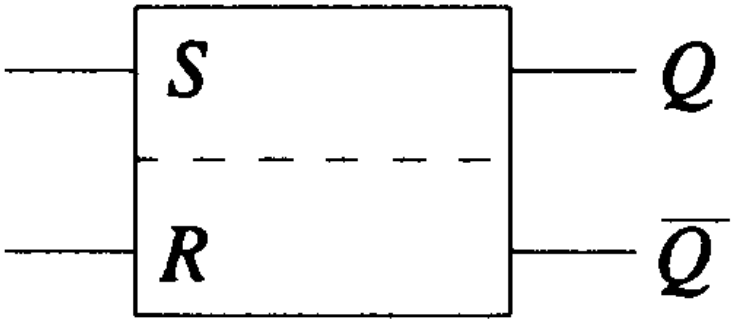
- **Nach dem Anlegen von U_B sei T_2 leitend, T_1 sperrt**
  - ⇨ **A_1 besitzt H-Pegel und A_2 besitzt L-Pegel**
  - ⇨ **dieser Zustand ist stabil**
- **Wird E_{st1} auf H-Pegel gesetzt, so**
  - ⇨ **wird T_1 leitend, A_1 geht auf L-Pegel**
  - ⇨ **T_2 sperrt und A_2 geht auf H-Pegel**
  - ⇨ **dieser Zustand ist ebenfalls stabil**
- **Wird E_{st2} auf H-Pegel gesetzt, so**
  - ⇨ **wird T_2 leitend, A_2 geht auf L-Pegel**
  - ⇨ **T_1 sperrt und A_1 geht auf H-Pegel**
  - ⇨ **dieser Zustand ist wiederum stabil**
- **Werden E_{st1} und E_{st2} auf H-Pegel gesetzt, so**
  - ⇨ **leiten beide Transistoren, die Rückkopplung wird unwirksam**
  - ⇨ **dieser Zustand ist nicht stabil**
  - ⇨ **unzulässige Eingangsbelegung**



RS-Flipflop

- **Bistabile Kippschaltungen können aus rückgekoppelten**
  - ⇨ **Transistoren**
  - ⇨ **NOR-Gattern**
  - ⇨ **NAND-Gattern**

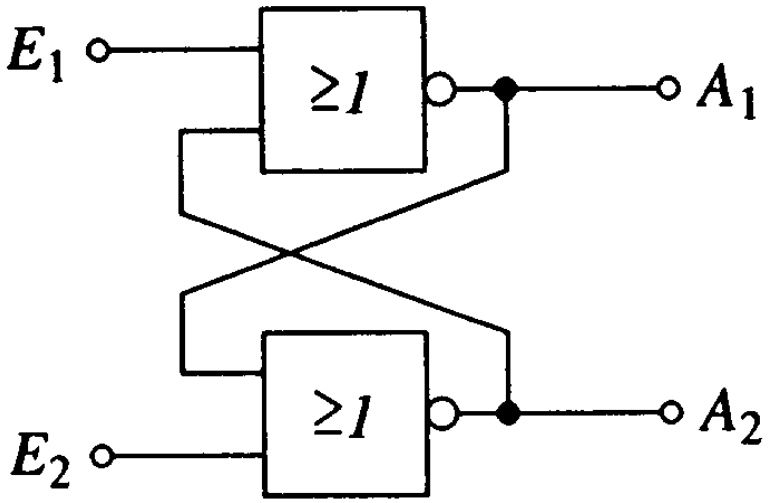
  gebaut werden
- **RS-Flipflop**
  - ⇨ **wenn die Eingänge den Wert 0 haben, bleibt der vorherige Zustand stabil**
  - ⇨ **wird S =1, wird Q=1 und $\overline{Q}$=0**
  - ⇨ **wird R=1, wird Q=0 und $\overline{Q}$=1**
  - ⇨ **S=1 und gleichzeitig R=1 sind nicht zulässig**



Schaltzeichen für ein RS-Flipflop nach DIN

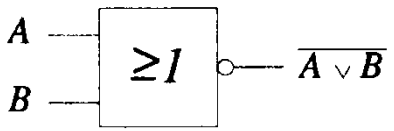
RS-Flipflop aus NOR-Gattern

- **Liegt an einem Eingang eines NOR-Gatters eine 1 an, so geht der entsprechende Ausgang auf 0**
- **Liegen an beiden Eingängen eine 0 an, so bleiben die Ausgänge erhalten**



Funktionstabelle der Ausgänge A_1 und A_2

E_1	E_2	A_1	A_2
0	0	(wie vorher) speichern	
0	1	1	0
1	0	0	1
1	1	(0	0) unzulässig



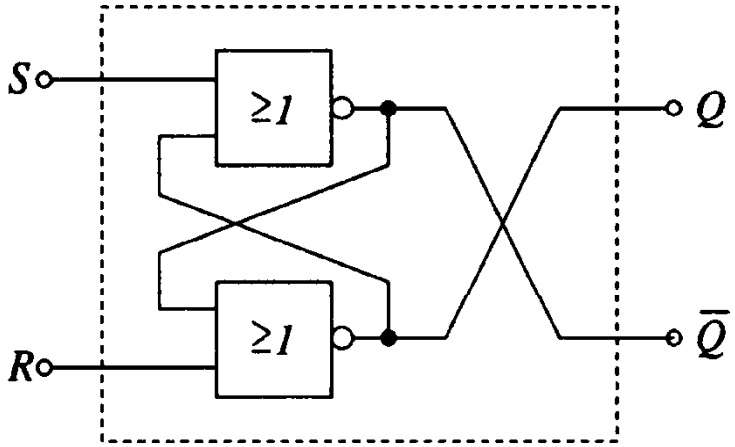
B	A	$\overline{A \vee B}$
0	0	1
0	1	0
1	0	0
1	1	0

RS-Flipflop aus NOR-Gattern

- **Ein RS-Flipflop entsteht durch Vertauschen der Ausgänge**

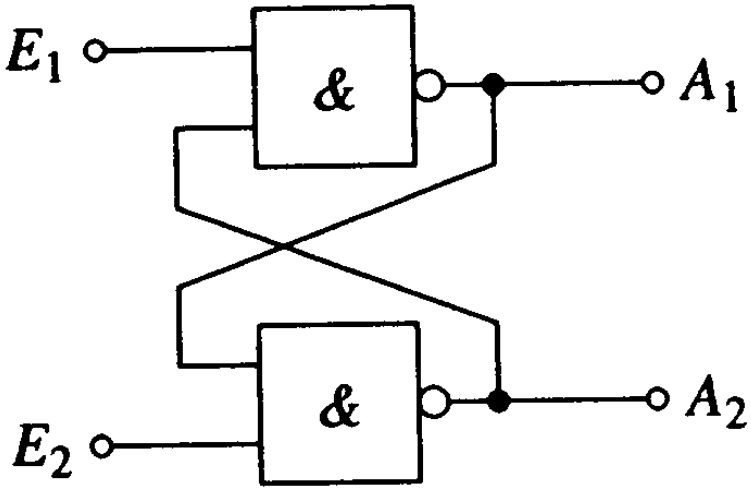
Funktionstabelle

S	R	Q	$\overline{Q}$	
0	0			(wie vorher) speichern
0	1	0	1	
1	0	1	0	
1	1	(0	0)	unzulässig



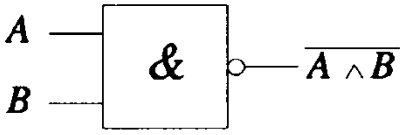
RS-Flipflop aus NAND-Gattern

- **Liegt an beiden Eingängen eines NAND-Gatters eine 1 an, so geht der entsprechende Ausgang auf 0**
- **Liegen an beiden Eingängen der Schaltung eine 1 an, so bleiben die Ausgänge erhalten**



Funktionstabelle der Ausgänge A_1 und A_2

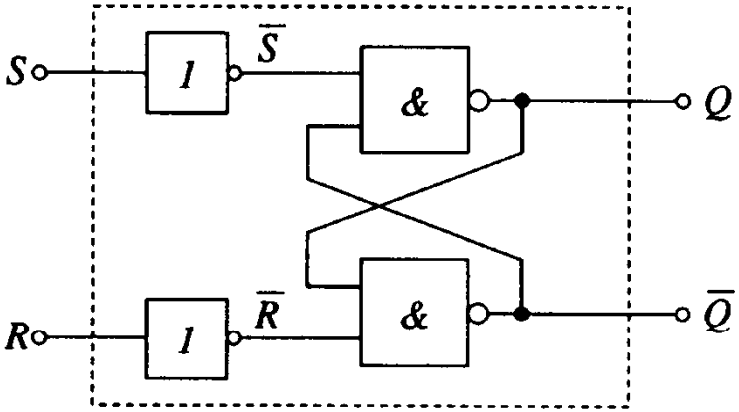
E_1	E_2	A_1	A_2
0	0	(1	1) (unzulässig)
0	1	1	0
1	0	0	1
1	1	(wie vorher) speichern	



B	A	$\overline{A \wedge B}$
0	0	1
0	1	1
1	0	1
1	1	0

RS-Flipflop aus NAND-Gattern

- **Ein RS-Flipflop entsteht durch Negation der Eingänge**



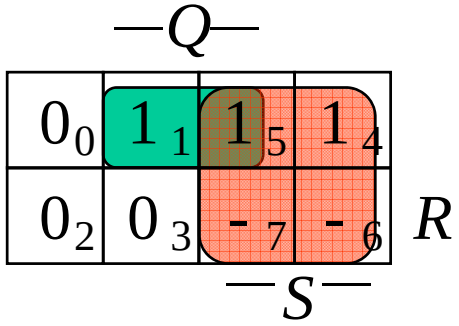
Funktionstabelle

S	R	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
0	0	1	1	(wie vorher)		speichern
0	1	1	0	0	1	
1	0	0	1	1	0	
1	1	0	0	(1	1)	unzulässig

Zustandsfolgetabelle

- Das Ausgangssignal ändert sich zeitversetzt nach der Signaländerung am Eingang
- Zeitverhalten wird in einer Zustandsfolge dargestellt
  - ⇨ Q_n ist der Wert vor der Signaländerung
  - ⇨ Q_{n+1} ist der Wert nach der Signaländerung

S	R	Q_n	Q_{n+1}	
0	0	0	0	speichern
0	0	1	1	speichern
0	1	0	0	rücksetzen
0	1	1	0	rücksetzen
1	0	0	1	setzen
1	0	1	1	setzen
1	1	0	-	unzulässig
1	1	1	-	unzulässig

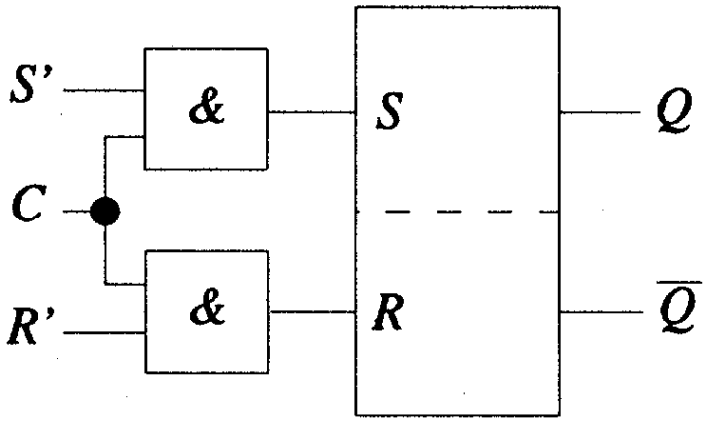


$$Q_{n+1} = S \vee (\overline{R} \wedge Q_n)$$

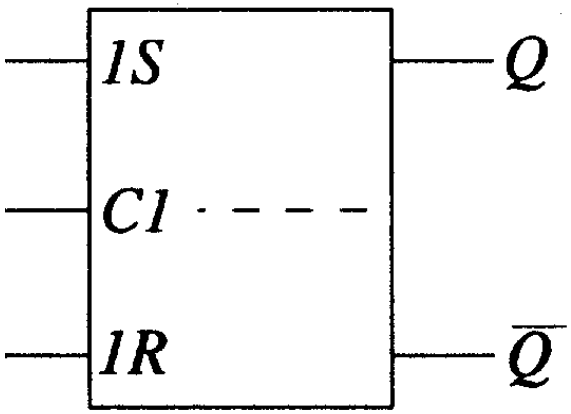
- Diese Gleichung heißt auch Funktionsgleichung oder Übergangsfunktion eines RS-Flipflops
  - ⇨ das Verhalten eines Flipflops kann durch eine Schaltfunktion beschrieben werden

RS-Flipflop mit Zustandssteuerung

- **Beim RS-Flipflop wird der Ausgang sofort nach Anlegen der Eingangssignale gesetzt**
  - ⇨ **zur Vermeidung von Hazards wird häufig gefordert, dass ein Flipflop seinen Wert nur zu bestimmten Zeitpunkten ändert**
  - ⇨ **Synchrone Schaltwerke**
  - ⇨ **Einführung eines Taktsignals**



Schaltung



Schaltzeichen

RS-Flipflop mit Zustandssteuerung

Funktionstabelle

C	S	R	Q_n	Q_{n+1}	
0	0	0	0	0	
0	0	0	1	1	keine Änderung
0	0	1	0	0	des
0	0	1	1	1	Ausgangs-
0	1	0	0	0	zustands
0	1	0	1	1	d.h.
0	1	1	0	0	Speichern
0	1	1	1	1	
1	0	0	0	0	speichern
1	0	0	1	1	speichern
1	0	1	0	0	rücksetzen
1	0	1	1	0	rücksetzen
1	1	0	0	1	setzen
1	1	0	1	1	setzen
1	1	1	0	-	unzulässig
1	1	1	1	-	unzulässig

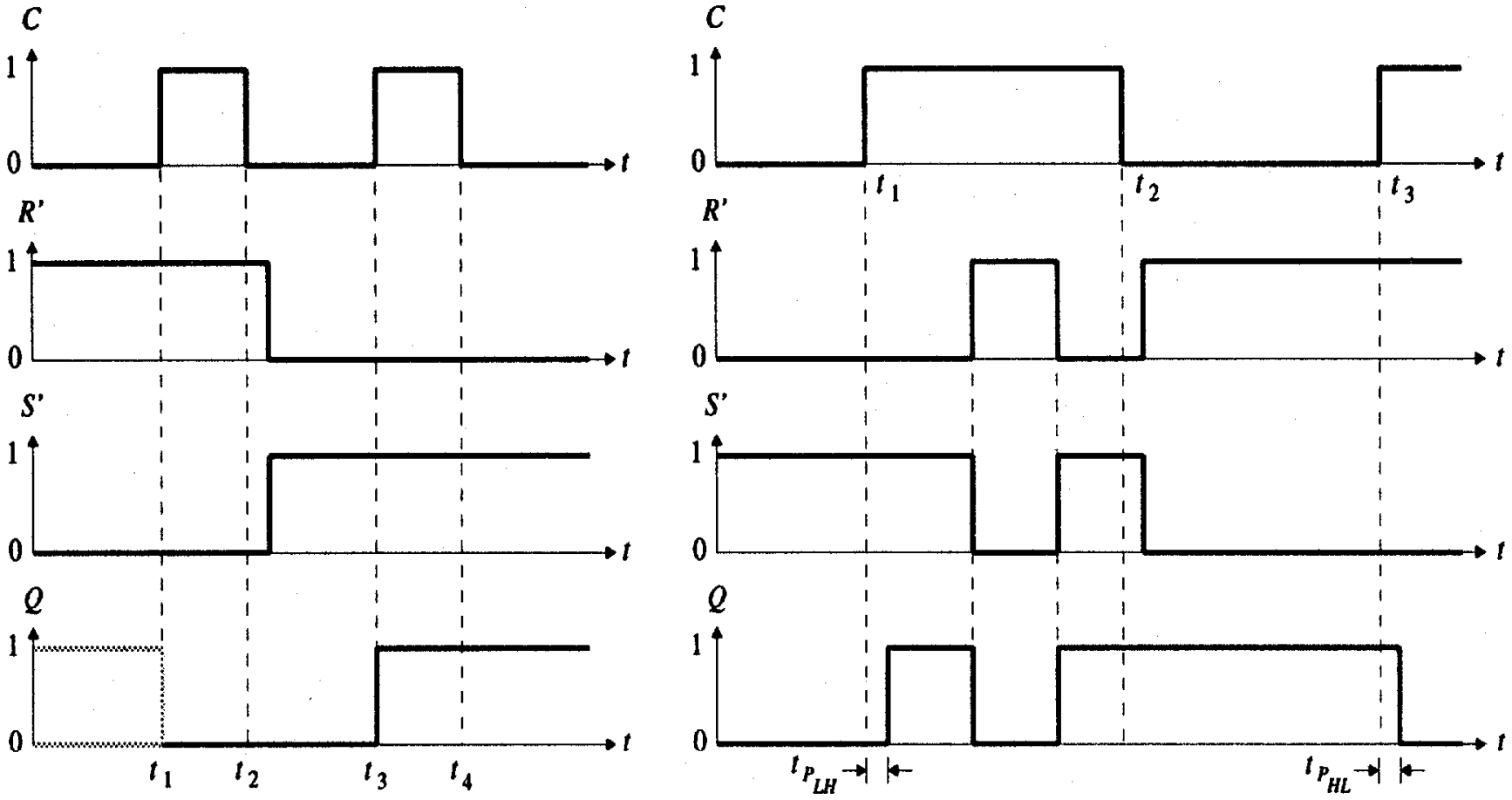
- **Aus der Übergangsfunktion des RS-Flipflos**

$$Q_{n+1} = S \vee (\overline{R} \wedge Q_n)$$

mit $S = (C \wedge S')$ **und** $R = (C \wedge R')$

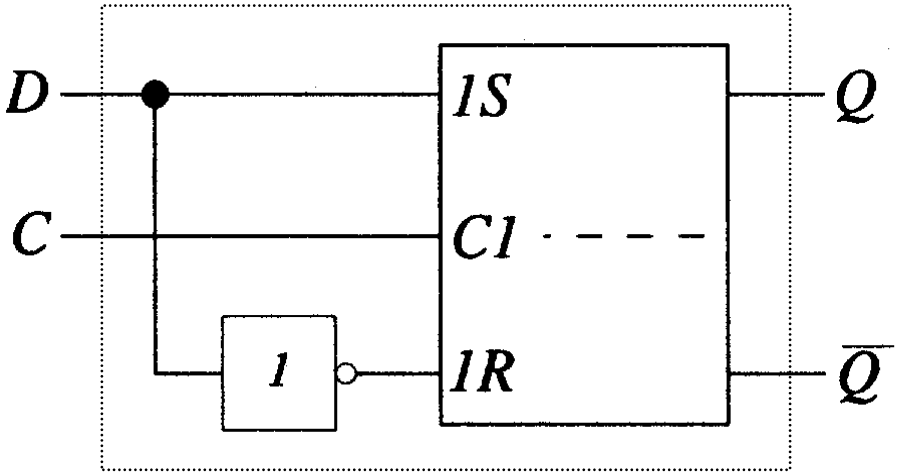
$$Q_{n+1} = (C \wedge S') \vee ((\overline{C \wedge R'}) \wedge Q_n)$$

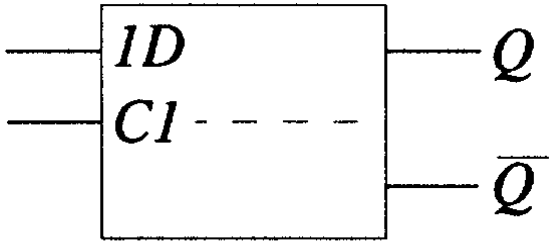
Impulsdiagramm für Taktzustandssteuerung



D-Flipflop mit Zustandssteuerung

- Das D-Flipflop entsteht aus einem RS-Flipflop mit Zustandssteuerung, durch die Negation des Setzsignals S

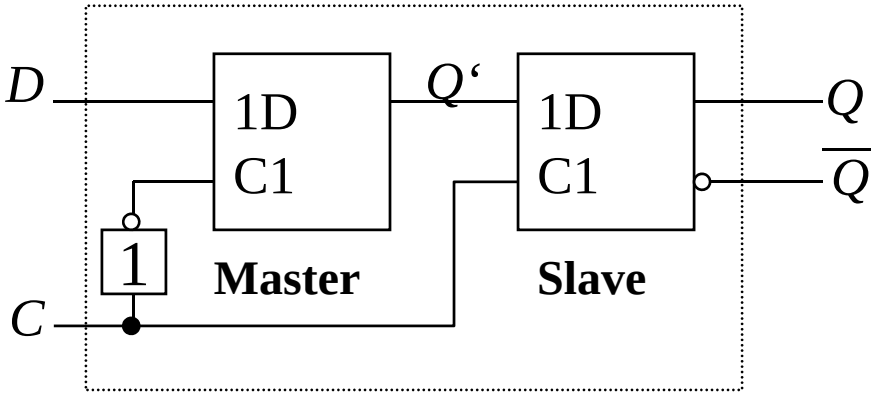




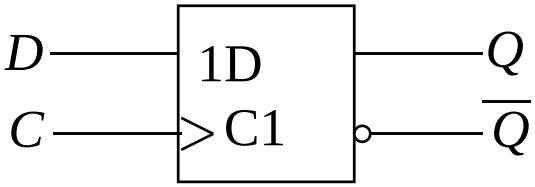
C	D	Q_n	Q_{n+1}	
0	-	0	0	speichern
0	-	1	1	speichern
1	0	-	0	rücksetzen
1	1	-	1	setzen

Master-Slave D- Flipflop

- **Probleme beim Verketten von Flipflops**
  - ⇨ **bei C=1 rutschen die Eingänge bis zum Ausgang durch**
  - ⇨ **Anwendung: Schieberegister, Zähler**
- **Lösung: (positiv) flankengesteuertes Flipflop**
  - ⇨ **zwei D-Flipflops werden hintereinander geschaltet**
  - ⇨ **das erste Flipflop erhält den negierten Takt**
  - ⇨ **Master-Slave-Prinzip**

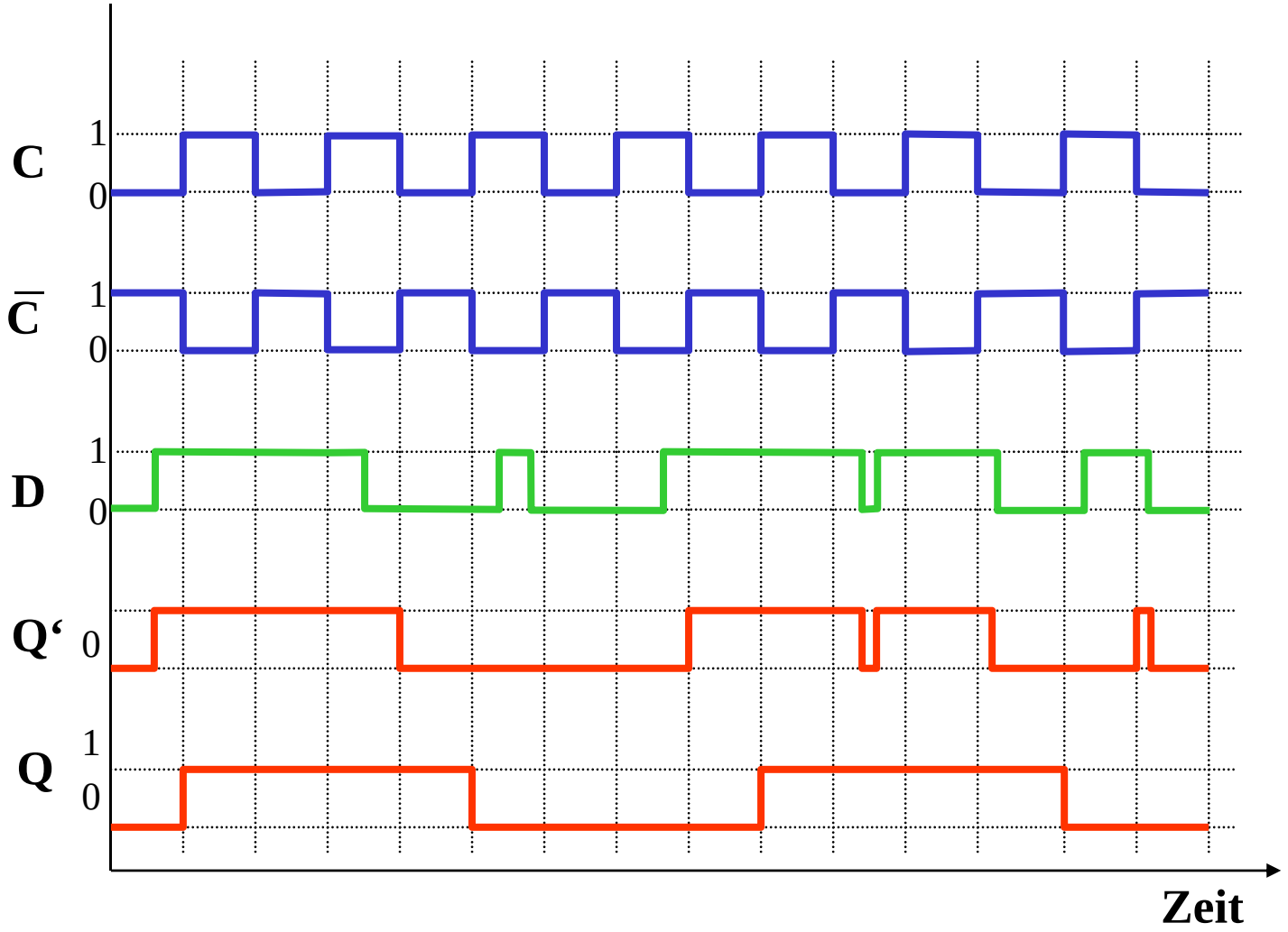


Schaltung

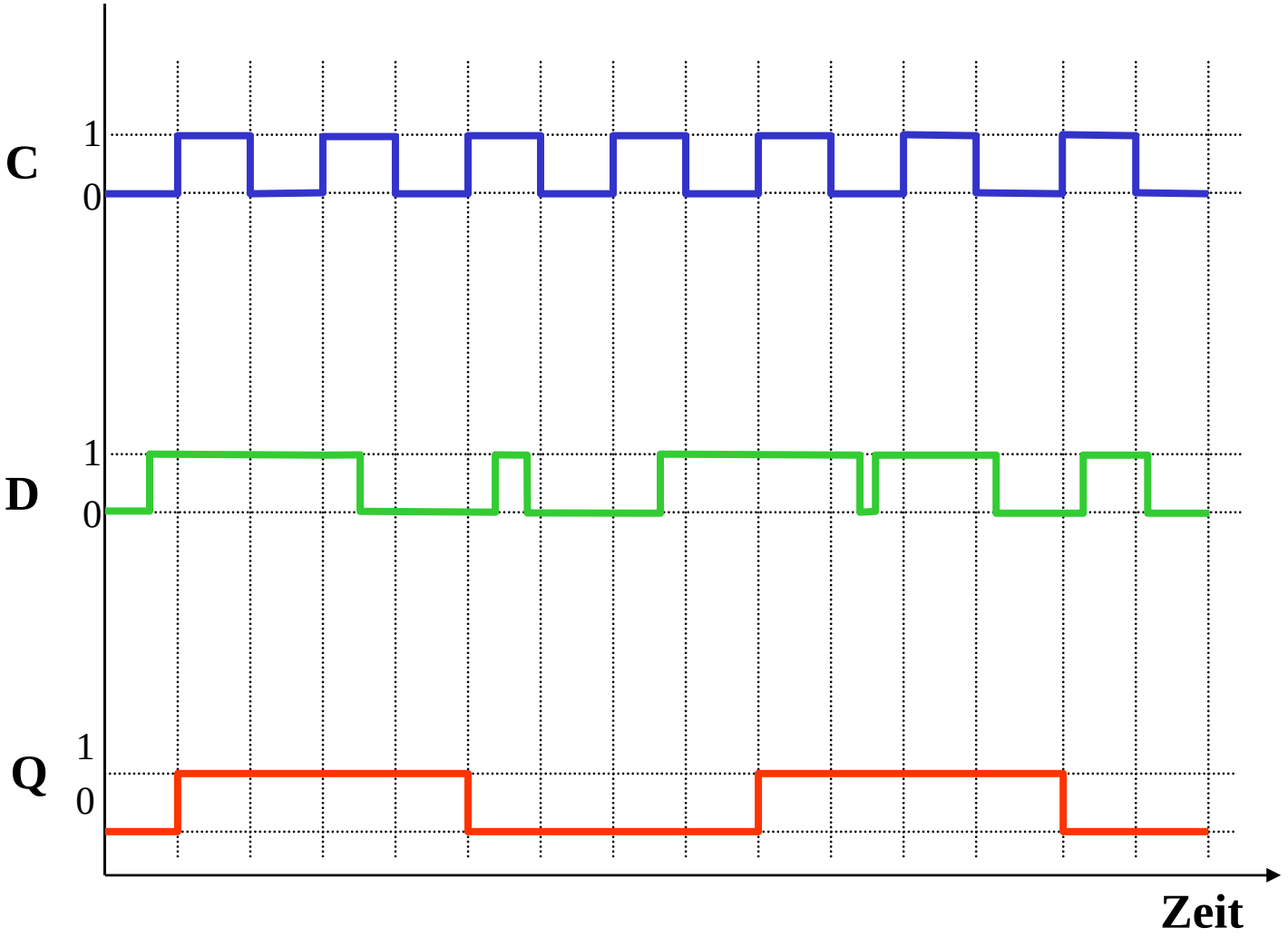


Schaltzeichen

Impulsdiagramm des Master-Slave D-Flipflops

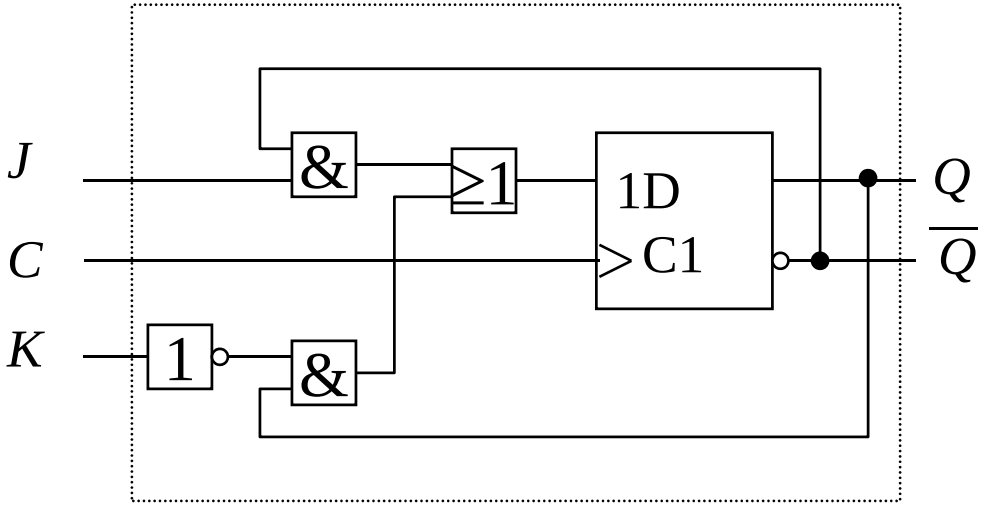


Impulsdiagramm des Master-Slave D-Flipflops

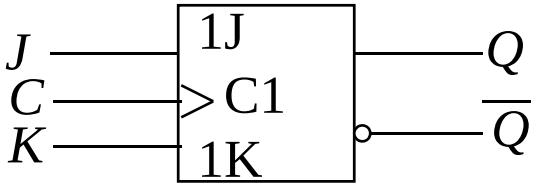


JK-Flipflop

- **Neben den Funktionen speichern, setzen und rücksetzen, macht es Sinn für die undefinierte Belegung R=S=1 die weitere Funktion wechseln zu definieren**
  - ⇨ **Man erreicht dies durch Rückführung der Ausgänge an den Eingang**



Schaltung



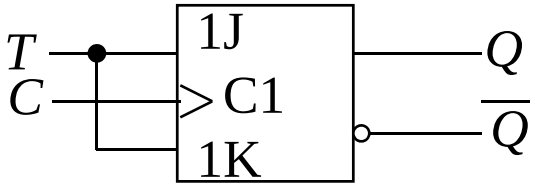
Schaltzeichen

J	K	Q_{n+1}	
0	0	Q_n	speichern
0	1	0	rücksetzen
1	0	1	setzen
1	1	$\overline{Q}_n$	wechseln

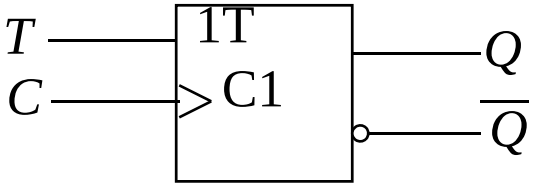
Funktionstabelle

Master-Slave T-Flipflop

- **Ein T-Flipflop besitzt wie das D-Flipflop nur einen Eingang**
  - ⇨ **ist dieser gleich 1, wechselt das Flipflop seinen Wert**
  - ⇨ **T steht für toggle**



Schaltung



Schaltzeichen

T	Q_{n+1}	
0	Q_n	speichern
1	$\overline{Q}_n$	wechseln

Funktionstabelle

Schaltwerke

Formale Grundlagen

- **Schaltnetze**
  - ⇨ **die Ausgabe einer Schaltung hängt nur von den Werten der Eingabe zum gleichen Zeitpunkt ab**
  - ⇨ **man nennt sie auch kombinatorische Schaltungen**
- **Schaltwerke**
  - ⇨ **die Ausgabe einer Schaltung kann von den Werten der Eingabe zu vergangenen Zeitpunkten abhängen**
  - ⇨ **alle Abhängigkeiten von Werten der Vergangenheit werden in einem Zustand zusammengefasst**
  - ⇨ **sie sind Implementierungen von deterministischen endlichen Automaten**

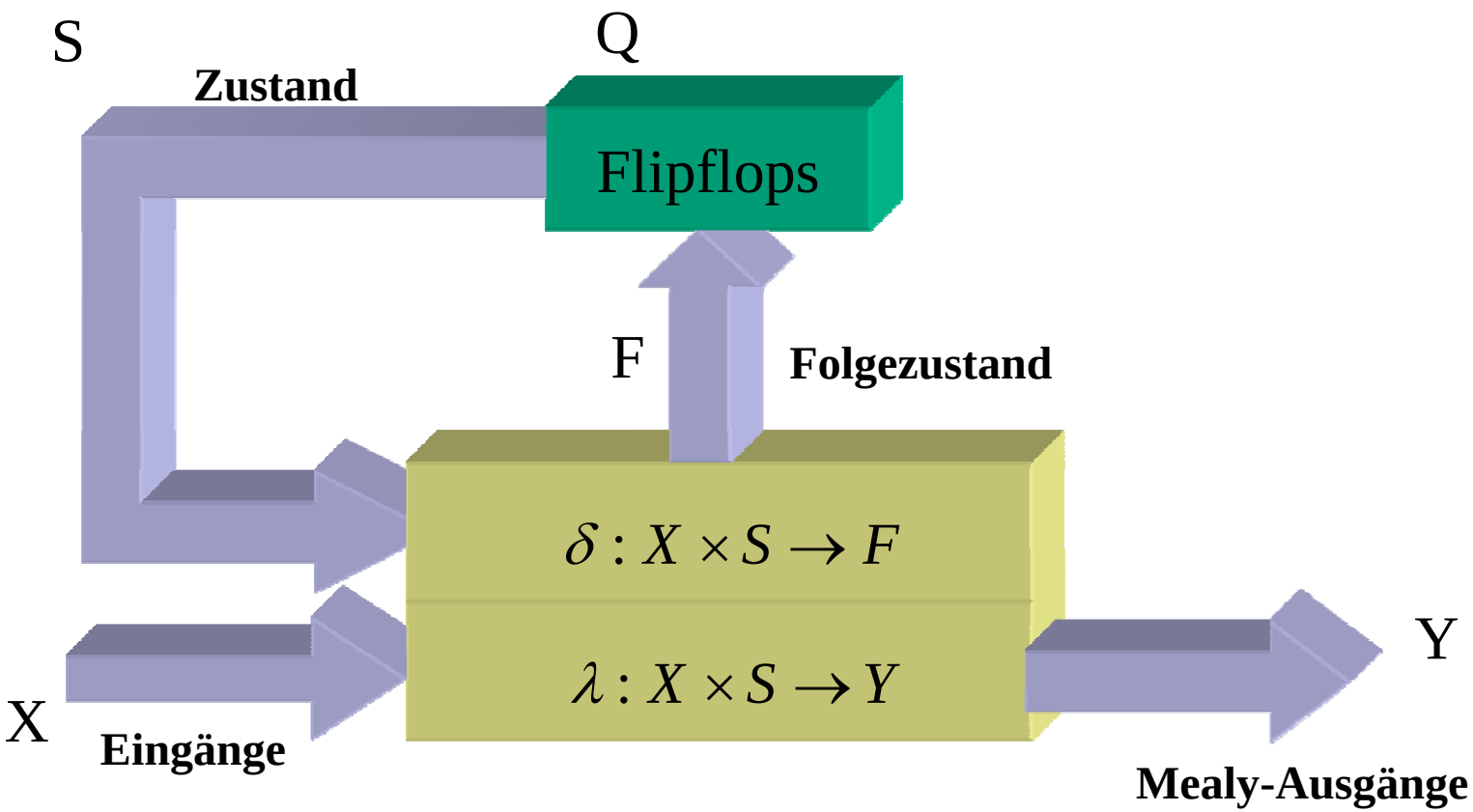
Beschreibung von endlichen Automaten

- **Andere Namen für endliche Automaten sind:**
  - ⇨ **finite state machine, FSM**
  - ⇨ **sequentielle Schaltungen**
  - ⇨ **Schaltungen mit Speicherverhalten**
- **Aus der Automatentheorie:**
  Ein endlicher Automat ist ein Quintupel $A=(X, Y, S, \delta, \lambda, s_0)$
  - ⇨ **eine endliche Menge von Eingangsbelegungen** X
  - ⇨ **eine endliche Menge von Ausgangsbelegungen** Y
  - ⇨ **eine endliche Menge von Zuständen** S
  - ⇨ **eine Zustandsübergangsfunktion** $\delta : X \times S \rightarrow S$
  - ⇨ **eine Ausgabefunktion**
    $\lambda : X \times S \rightarrow Y$ **(Mealy Verhalten)**
    $\lambda : S \rightarrow Y$ **(Moore Verhalten)**
  - ⇨ **und er besitzt einen Startzustand** s_0

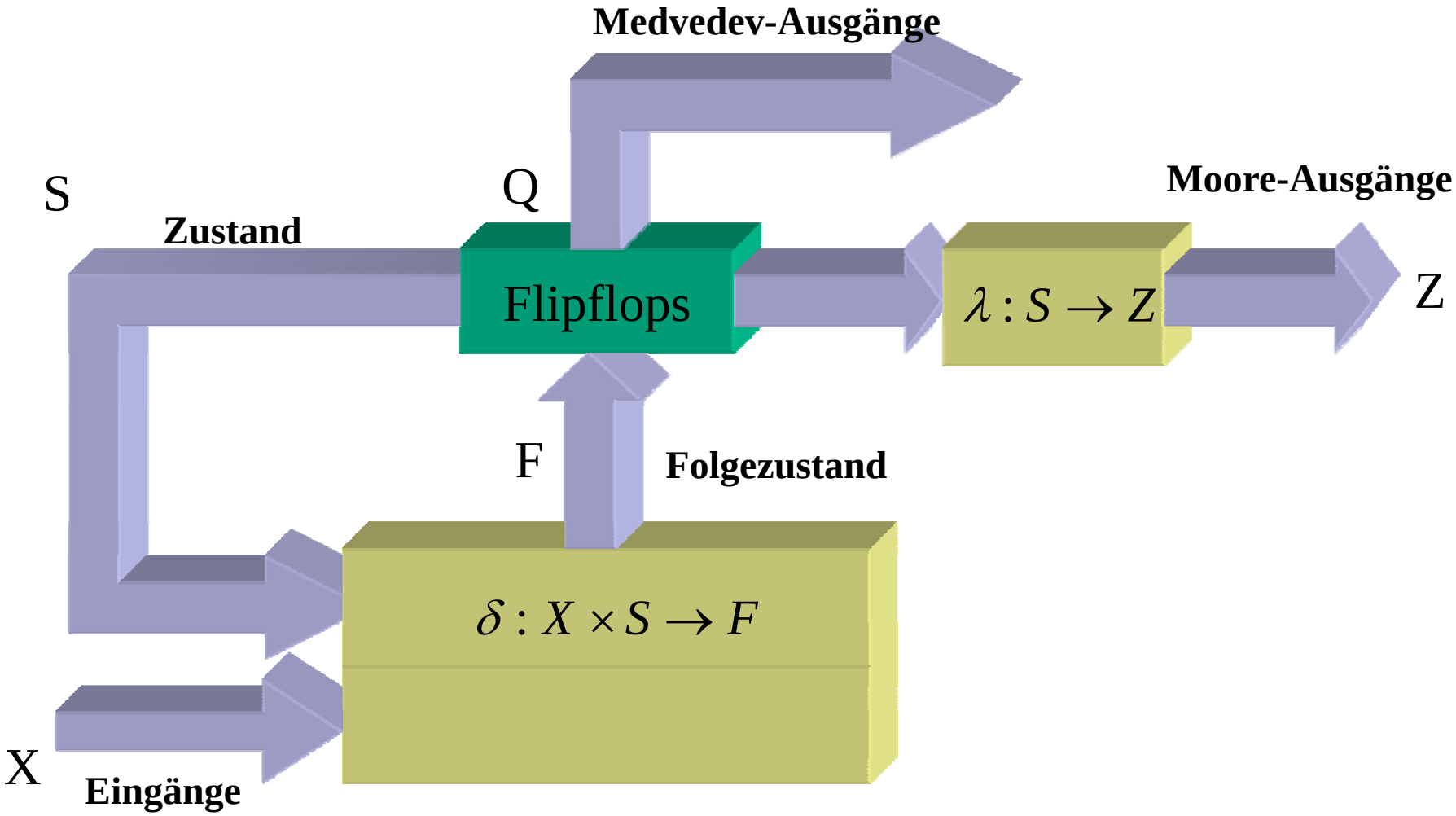
Mealy- und Moore-Automaten

- Die Zustände eines endlichen Automaten werden in Flipflops gespeichert
  - ⇨ möglich sind D-, T-, JK-, RS-Flipflops
  - ⇨ Der aktuelle Zustand wird an die Eingänge der Schaltung rückgekoppelt
- Man unterscheidet Mealy-, Moore- und Medvedev-Automaten:
- Mealy:
  - ⇨ Ausgangsleitungen können sich ändern, auch wenn keine Taktflanke aufgetreten ist
- Moore:
  - ⇨ Änderung von Ausgangsleitungen nur mit Änderung eines Taktimpulses
- Medvedev:
  - ⇨ Spezialfall des Moore-Automaten
  - ⇨ die Ausgänge sind die Zustandsbits der Flipflops

Struktur eines Mealy-Automaten



Struktur eines Moore-Automaten



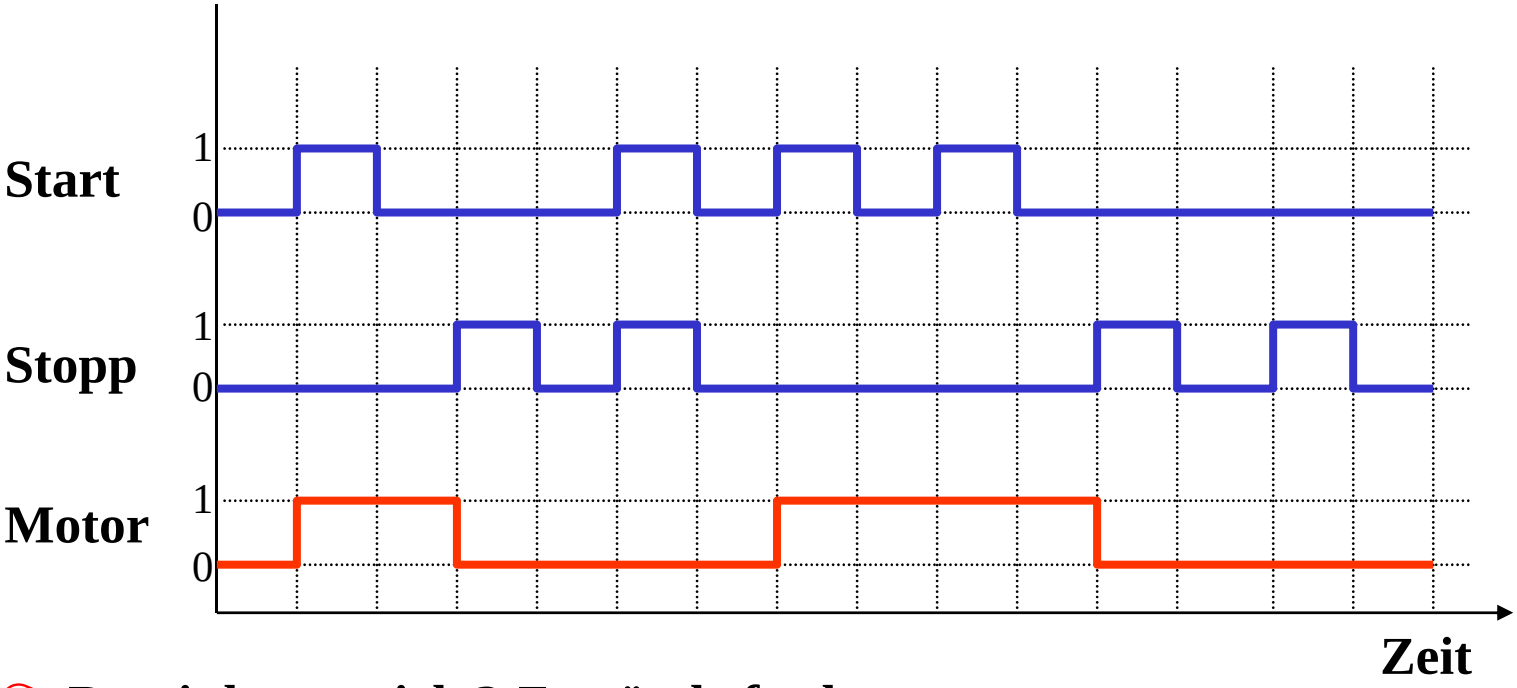
Darstellung endlicher Automaten

- **Die Aufgabenstellung liegt meist in einer nicht formalisierten Form vor**
- **Um beim Entwurf von Schaltwerken systematische und möglichst auch rechnergestützte Entwurfsverfahren einsetzen zu können, muss eine formalisierte Beschreibung verwendet werden**
- **Häufig verwendete Darstellungsformen sind:**
  - ⇨ **Zeitdiagramm**
  - ⇨ **Automatengraph**
  - ⇨ **Ablauftabelle**
  - ⇨ **Schaltfunktionen**
  - ⇨ **Automatentabelle**

Beispiel: Selbsthalteschaltung

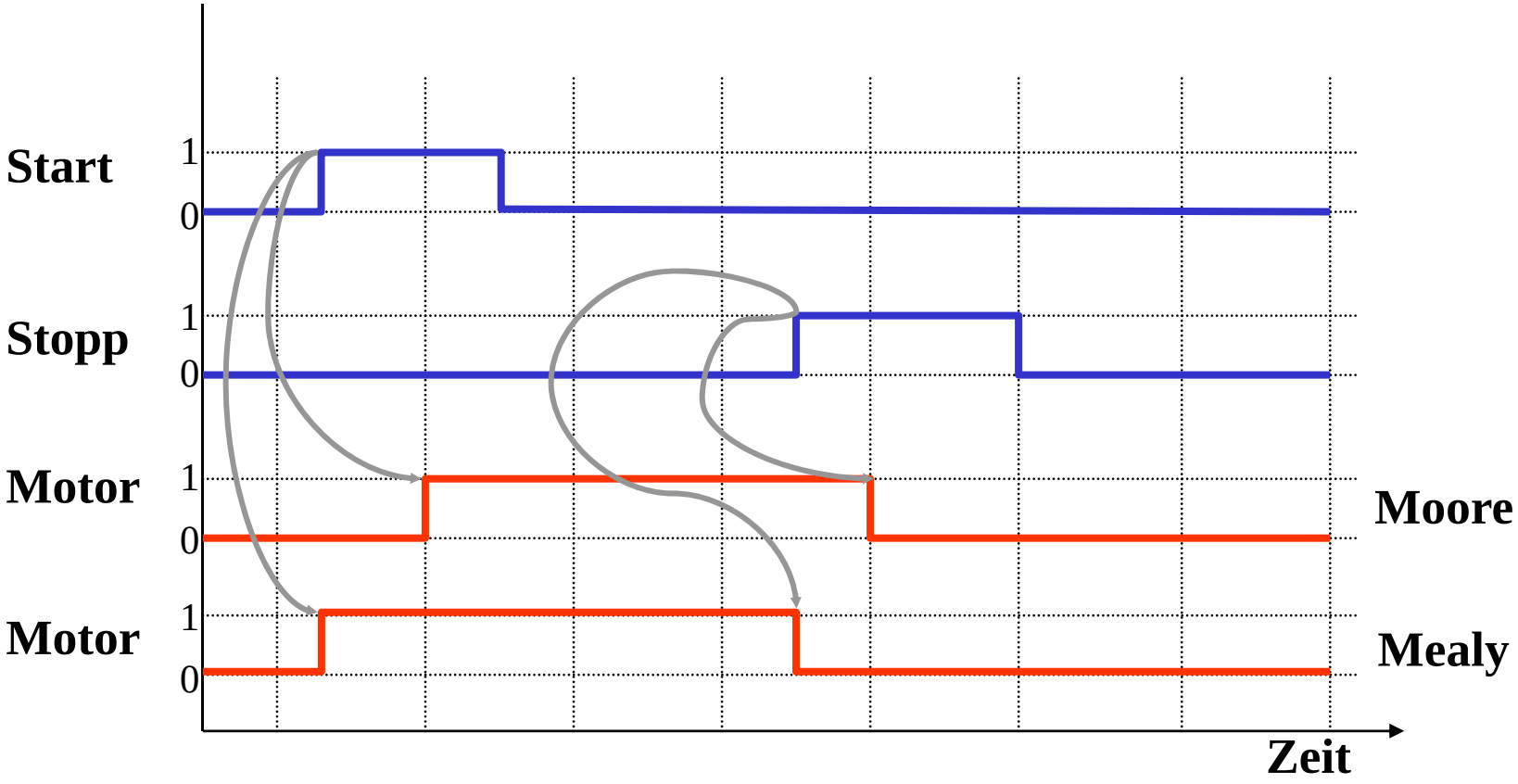
- **Beschreibung der Funktion:**
  - **an den Eingängen befinden sich zwei Tasten : (Start und Stopp)**
  - **die Schaltung liefert ein Ausgangssignal, mit dem ein Gerät ein- oder ausgeschaltet werden kann**
  - **wird die Starttaste gedrückt, soll das Gerät eingeschaltet werden**
  - **es soll eingeschaltet bleiben, auch wenn die Starttaste wieder losgelassen wird**
  - **das Gerät soll ausgeschaltet werden, sobald die Stopptaste betätigt wird**
- **zu klären ist:**
  - **was passiert, wenn beide Tasten gleichzeitig betätigt werden?**
  - **was passiert, wenn die Starttaste gedrückt wird, obwohl das Gerät eingeschaltet ist?**
  - **was passiert, wenn das Gerät ausgeschaltet ist und die Stopptaste gedrückt wird?**

Zeitdiagramm



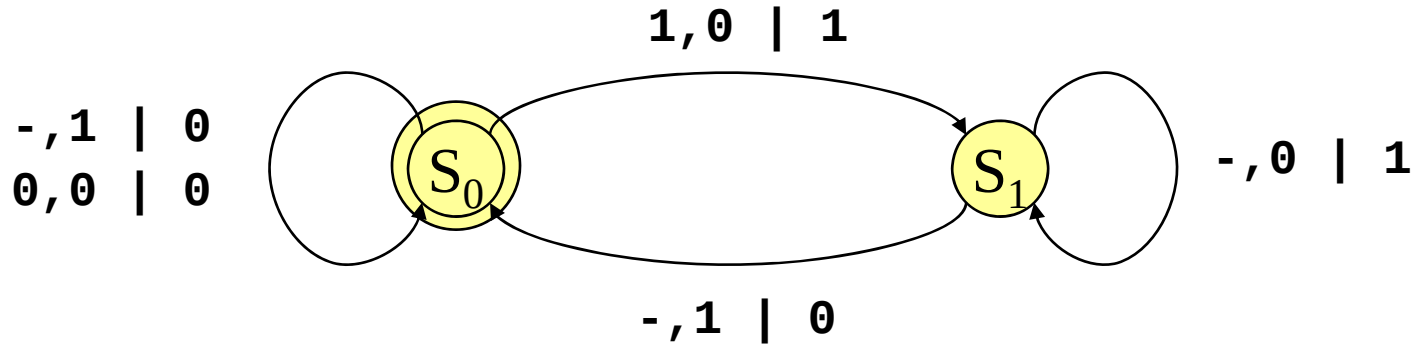
- **Damit lassen sich 2 Zustände festlegen:**
  - ⇨ **Zustand s_0: Ausgabe von** Motor=0 **und warten auf** Start=1 **und** Stopp=0
  - ⇨ **Zustand s_1: Ausgabe von** Motor=1 **und warten auf** Stopp=1

Mealy und Moore Verhalten

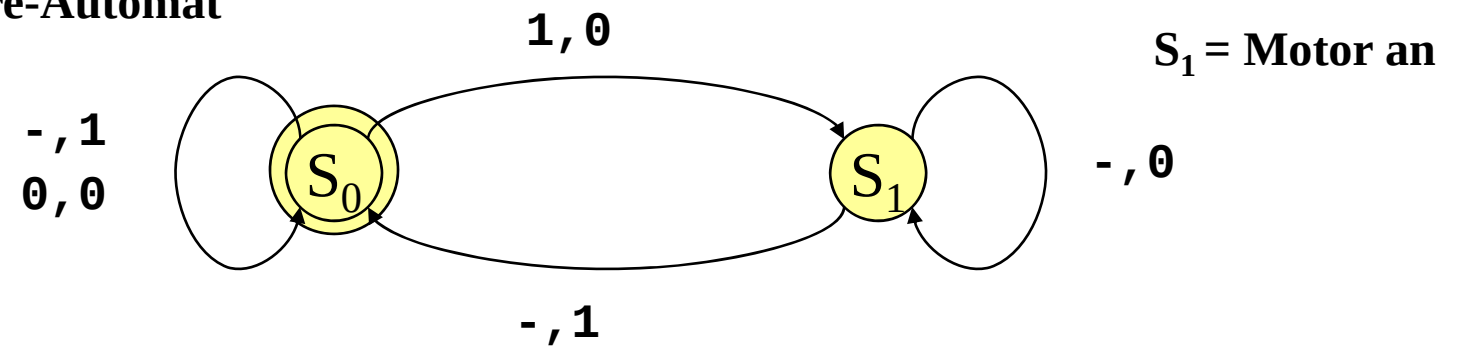


Automatengraph

Mealy-Automat



Moore-Automat



S_0 = Motor aus

S_1 = Motor an

Ablauftabelle

Mealy-Ablauftabelle

Eingang	Zustand		Folgezustand	Ausgang
-, 1	S0		S0	0
0, 0	S0		S0	0
1, 0	S0		S1	1
-, 0	S1		S1	1
-, 1	S1		S0	0

Moore-Ablauftabelle

Eingang	Zustand / Ausgang	Folgezustand
-, 1	S0/ 0	S0
0, 0	S0/ 0	S0
1, 0	S0/ 0	S1
-, 0	S1/ 1	S1
-, 1	S1/ 1	S0

Interpretation der Ablauftabelle

Wenn	**wir im Zustand 0 sind**
und	**zusätzlich Start = 1 und Stop = 0 gilt,**
dann	**wird Motor_an zu 1**
und	**wir gehen mit dem nächsten Takt in den Zustand 1**

Schaltfunktionen

- **Aus der Ablauftabelle lassen sich die die Ausgabe- und die Zustandsübergangsfunktion ablesen:**

x_1, x_2	Zustand S	Folgezustand S^+	Ausgang y
-, 1	S0	S0	0
0, 0	S0	S0	0
1, 0	S0	S1	1
-, 0	S1	S1	1
-, 1	S1	S0	0

- **Übergangsfunktion:** $s_0^+ = (x_2 \wedge s_0) \vee (\bar{x}_1 \wedge \bar{x}_2 \wedge s_0) \vee (x_2 \wedge s_1)$

$$s_1^+ = (x_1 \wedge \bar{x}_2 \wedge s_0) \vee (\bar{x}_2 \wedge s_1)$$

- **Ausgabefunktion:** $y = (x_1 \wedge \bar{x}_2 \wedge s_0) \vee (\bar{x}_2 \wedge s_1)$ **Mealy-Automat**

$y = s_1$ **Moore-Automat**

Automatentabelle

Zustand	Folgezustand/Ausgang Start/Stopp 0/0	0/1	1/0	1/1
s_0	$s_0/0$	$s_0/0$	$s_1/1$	$s_0/0$
s_1	$s_1/1$	$s_0/0$	$s_1/1$	$s_0/0$

Mealy-Automat

Zustand	Folgezustand Start/Stopp 0/0	0/1	1/0	1/1	Ausgang
s_0	s_0	s_0	s_1	s_0	0
s_1	s_1	s_0	s_1	s_0	1

Moore-Automat

- **In der Automatentabelle werden die Zustände senkrecht und alle möglichen Eingangsbelegungen waagrecht dargestellt**
  - ⇨ **an den Schnittpunkten werden die Folgezustände eingetragen**
  - ⇨ **Mealy-Automat: Die Ausgabe wird dem Folgezustand zugeordnet**
  - ⇨ **Moore-Automat: Die Ausgabe wird dem Zustand zugeordnet**

Medvedev- und Moore-Automaten

- **Auch Moore-Automaten können während des Übergangs Fehlimpulse (Glitches, Hazards) auslösen**
  - ⇨ **unterschiedliche Laufzeiten in der Schaltung**
  - ⇨ **0→1 nach 1→0 Übergänge der Zustandsübergangsfunktion ohne Änderung des Ausgangswerts**
- **Medvedev-Automaten besitzen am Ausgang ein Flipflop**
  - ⇨ **keine Fehlimpulse**
  - ⇨ **Ausgangswert muss einen Takt früher berechnet werden**

Eingang	Zustand / Ausgang	Folgezustand	Eingang	Zustand / Ausgang	Folgezustand
-, 1	S0/ 0	S0	-, 1	S0/ 0	S0
0, 0	S0/ 0	S0	0, 0	S0/ 0	S0
1, 0	S0/ 0	S1	1, 0	S0/ 1	S1
-, 0	S1/ 1	S1	-, 0	S1/ 1	S1
-, 1	S1/ 1	S0	-, 1	S1/ 0	S0

Moore-Automat **Medvedev-Automat**

Analyse und Entwurf von Schaltwerken

Grundlegende Realisierung von Automaten: Synchron oder Asynchron

- **Asynchrone Realisierung**
  - ⇨ **Zustandsspeicher durch Rückkopplung**
  - ⇨ **es gibt keinen zentralen Takt**
  - ⇨ **die Zustandsspeicher (Flipflops) können zu jedem Zeitpunkt ihren Wert ändern**
    - **Laufzeiten beachten!**
- **Synchrone Realisierung**
  - ⇨ **Rückkopplung nur durch flanken- oder pegelgetriggerte Flipflops**
  - ⇨ **die Taktleitungen aller Flipflops sind miteinander verbunden (oder hängen nach einem festen Zeitschema voneinander ab)**
- **Obwohl asynchrone Realisierungen auch eine gewisse praktische Bedeutung besitzen, werden hier nur synchrone Realisierungen betrachtet**

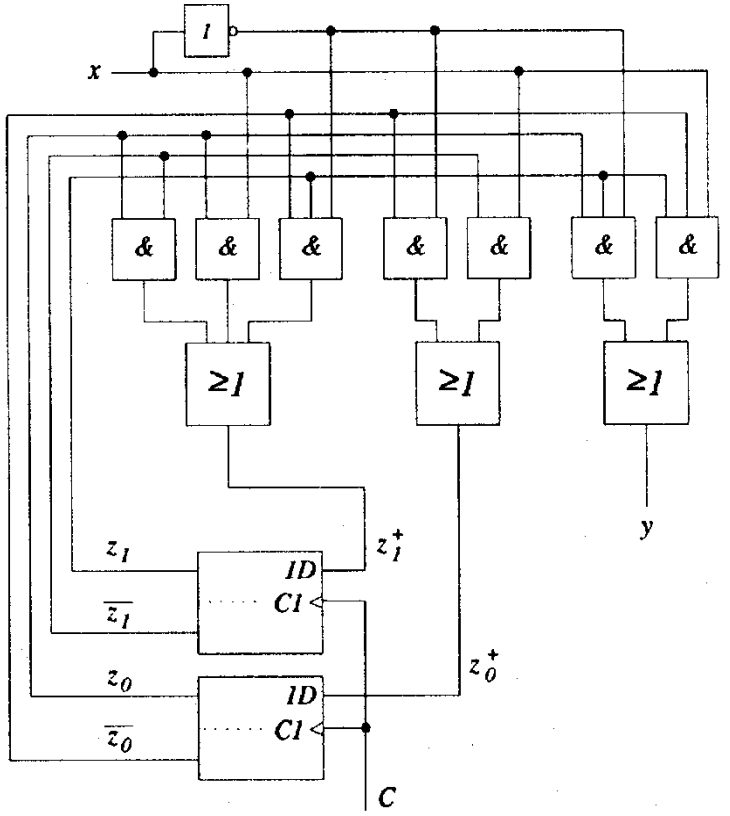
Analyse von Schaltwerken

- **Ein Schaltwerk zu analysieren bedeutet, sein Schaltverhalten durch**
  - ⇨ **eine Zustandstabelle**
  - ⇨ **dessen Schaltfunktion oder**
  - ⇨ **einen Zustandsgraph**

  zu beschreiben
- **Prinzipielles Vorgehen:**
  - ⇨ **von einem gegebenen Schaltplan werden zunächst die Ausgabe und Übergangsfunktion abgeleitet**
  - ⇨ **ein Anfangszustand wird angenommen**
  - ⇨ **mit den Werten der Eingangsvariablen werden die Folgezustände abgeleitet**
  - ⇨ **auf diese Weise entstehen die Ablauftabellen**
  - ⇨ **aus den Ablauftabellen kann der Automatengraph abgeleitet werden**

Beispiel: Ausgangspunkt - der Schaltplan

- **Grundlegende Charakterisierungen**
  - **synchrones Schaltwerk**
  - **Eingang *x* und Ausgang *y* bestehen je aus einer Variablen**
  - **das Schaltwerk enthält 2 D-Flipflops**
  - **es kann maximal 4 Zustände besitzen**
  - **Das Schaltwerk ist ein Mealy-Automat**



Die Schaltfunktion

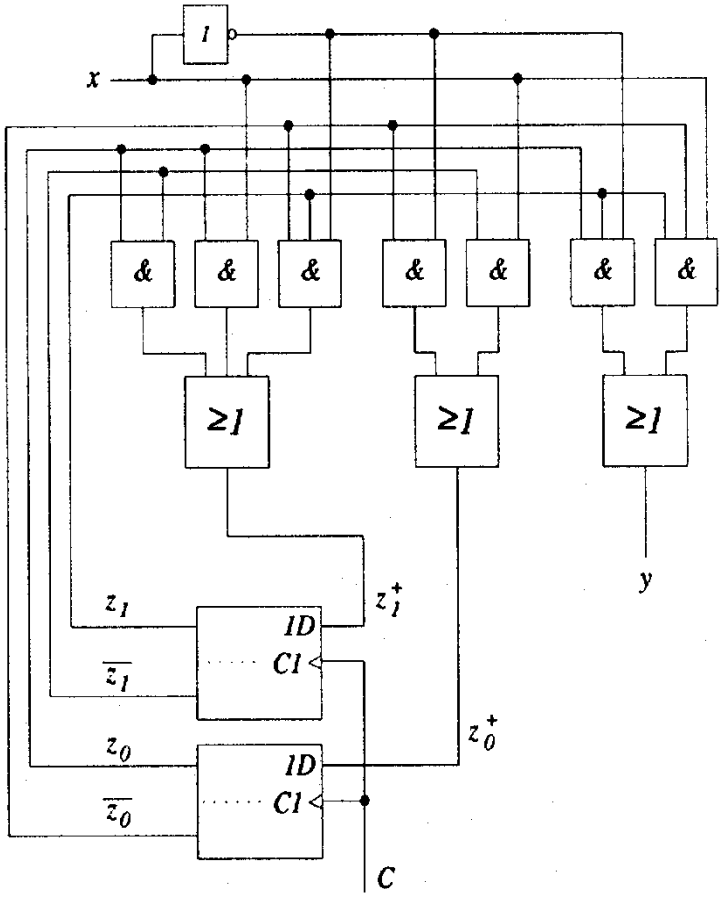
- **Aus dem Schaltplan läßt sich ablesen:**
  - ⇨ **für die Übergangsfunktion**

$$z_0^+ = (\overline{z}_0 \wedge \overline{x}) \vee (\overline{z}_1 \wedge x)$$

$$z_1^+ = (z_0 \wedge \overline{z}_1) \vee (z_0 \wedge x) \vee (\overline{z}_0 \wedge z_1 \wedge \overline{x})$$

  - ⇨ **für die Ausgabefunktion**

$$y = (z_0 \wedge z_1 \wedge \overline{x}) \vee (\overline{z}_0 \wedge z_1 \wedge x)$$

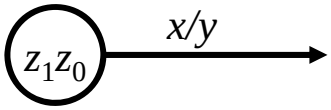


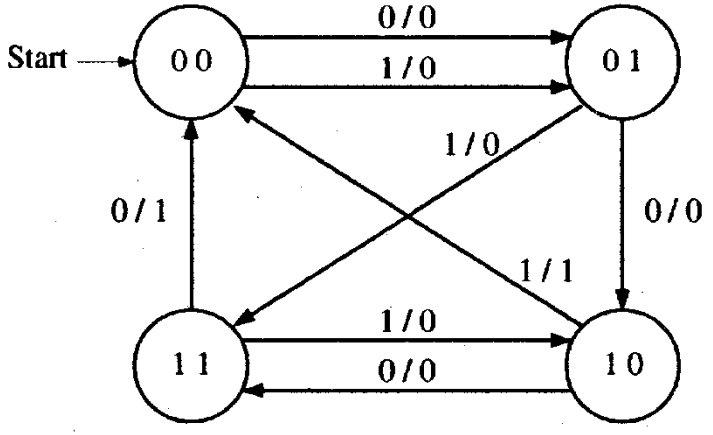
Die Ablauftabelle und der Automatengraph

- **Aufstellen der Ablauftabelle über die Auswertung der Funktionen für z_0, z_1 und y**
  - ⇨ **alle Belegungen der Eingangsvariablen**
  - ⇨ **alle Belegungen der Zustandsvariablen**

z_1	z_0	x	z_1^+	z_0^+	y
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	1	0
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1	1	0	0

- **Aufstellen des Automatengraphen über die Auswertung der Ablauftabelle**
  - ⇨ **Beschriftung der Zustände und Übergänge nicht vergessen!**



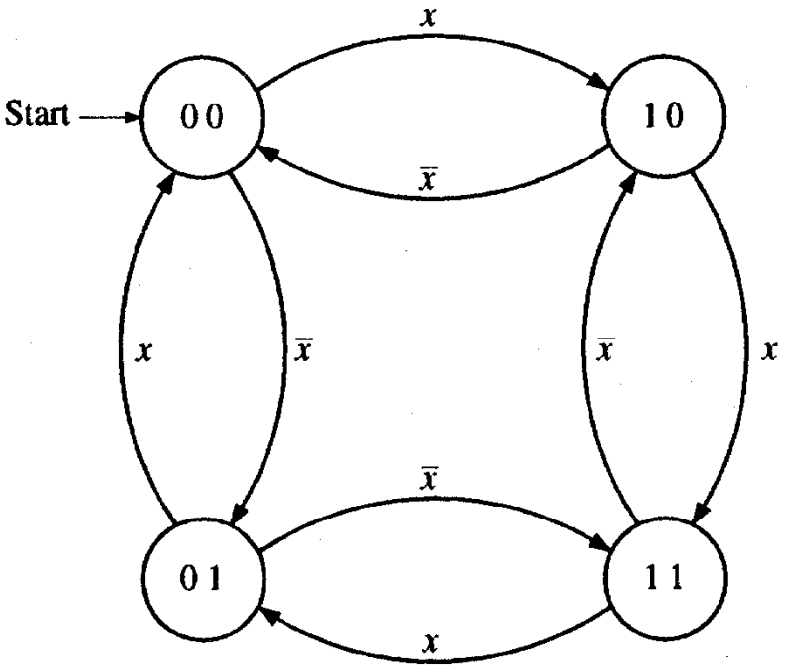


Entwurf von Schaltwerken

- **Prinzipielles Vorgehen:**
  - ⇨ **Festlegen der Zustandsmenge**
    - **daraus ergibt sich die Anzahl der erforderlichen Speicherglieder**
  - ⇨ **Festlegen des Anfangszustands**
  - ⇨ **Definition der Ein- und Ausgangsvariablen**
  - ⇨ **Darstellung der zeitlichen Zustandsfolge in Form eines Zustandsgraphen**
  - ⇨ **Aufstellen der Ablauftabelle**
  - ⇨ **Herleitung der Übergangs- und Ausgabefunktionen**
  - ⇨ **Darstellung der Übergangs- und Ausgabefunktionen in einem KV-Diagramm und Minimierung**
  - ⇨ **Darstellung des Schaltwerks in einem Schaltplan**

Beispiel: ein umschaltbarer Zähler

- **Es soll ein zweistelliger Gray-Code-Zähler entworfen werden, der sowohl vorwärts als auch rückwärts zählen kann**
- **Die Umschaltung der Zählrichtung erfolgt über die Eingangsvariable** x
  - ⇨ **für** x=0 **ist die Zählfolge**
    `00 - 01 - 11 - 10`
  - ⇨ **für** x=1 **ist die Zählfolge**
    `00 - 10 - 11 - 01`
- **Die Ausgangsvariablen sind identisch mit den Zustandsvariablen, da der Zählerstand angezeigt werden soll**
  - ⇨ **Moore-Automat**



Automatengraph

Ablauftabelle und die Übergangsfunktionen

- **Die Ablauftabelle kann direkt aus dem Automatengraph abgeleitet werden**
  - ⇨ **die linke Seite enthält alle Wertekombinationen, die z_0, z_1 und x einnehmen können**
  - ⇨ **die rechte Seite enthält die Werte der Folgezustände**

z_1	z_0	x	z_1^+	z_0^+
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	1

- **Aus der Ablauftabelle können die KV-Diagramme für z_0 und z_1 aufgestellt werden**

z_1^+ — x —

0_0	1_1	1_5	0_4
1_2	0_3	0_7	1_6

z_0

— z_1 —

z_0^+ — x —

1_0	0_1	1_5	0_4
1_2	0_3	1_7	0_6

z_0

— z_1 —

- **Aus den KV-Diagrammen lassen sich die minimierten Übergangsfunktionen ablesen**

$$z_1^+ = (\bar{z}_0 \wedge x) \vee (z_0 \wedge \bar{x})$$

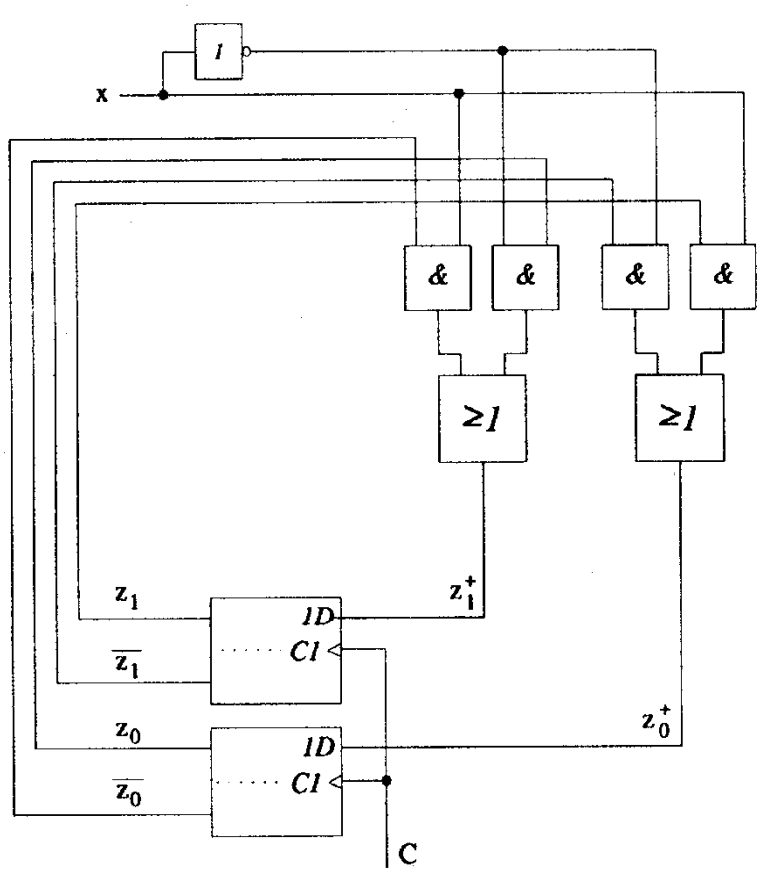
$$z_0^+ = (\bar{z}_1 \wedge \bar{x}) \vee (z_1 \wedge x)$$

Das Schaltwerk

- **Die minimierten Übergangsfunktionen können schließlich in einem Schaltplan gezeichnet werden**

$$z_0^+ = (\overline{z}_0 \wedge x) \vee (z_0 \wedge \overline{x})$$

$$z_1^+ = (\overline{z}_1 \wedge \overline{x}) \vee (z_1 \wedge x)$$

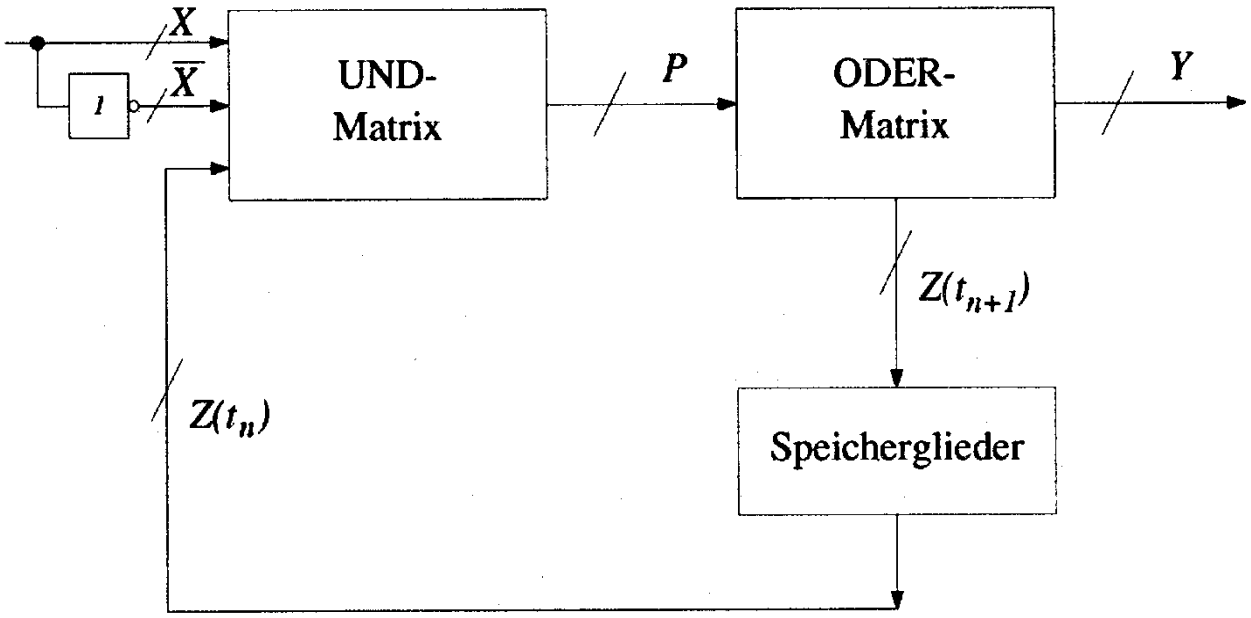


Technische Realisierung von Schaltwerken

- **Realisierung mit diskreten Bauelementen**
  - ⇨ **Verknüpfungsglieder**
  - ⇨ **Speicherglieder**
- **Die Bauelemente werden entsprechend der Aufgabenstellung durch eine feste Verdrahtung miteinander verbunden**
- **Solche Schaltwerksrealisierungen können nur eine feste Aufgabe erfüllen**
  - ⇨ **das Schaltwerk ist nicht flexibel**
  - ⇨ **bei einem Fehler in der Verdrahtung kann keine Korrektur vorgenommen werden**
- **Die Bauelemente stehen als integrierte Schaltkreise zur Verfügung**

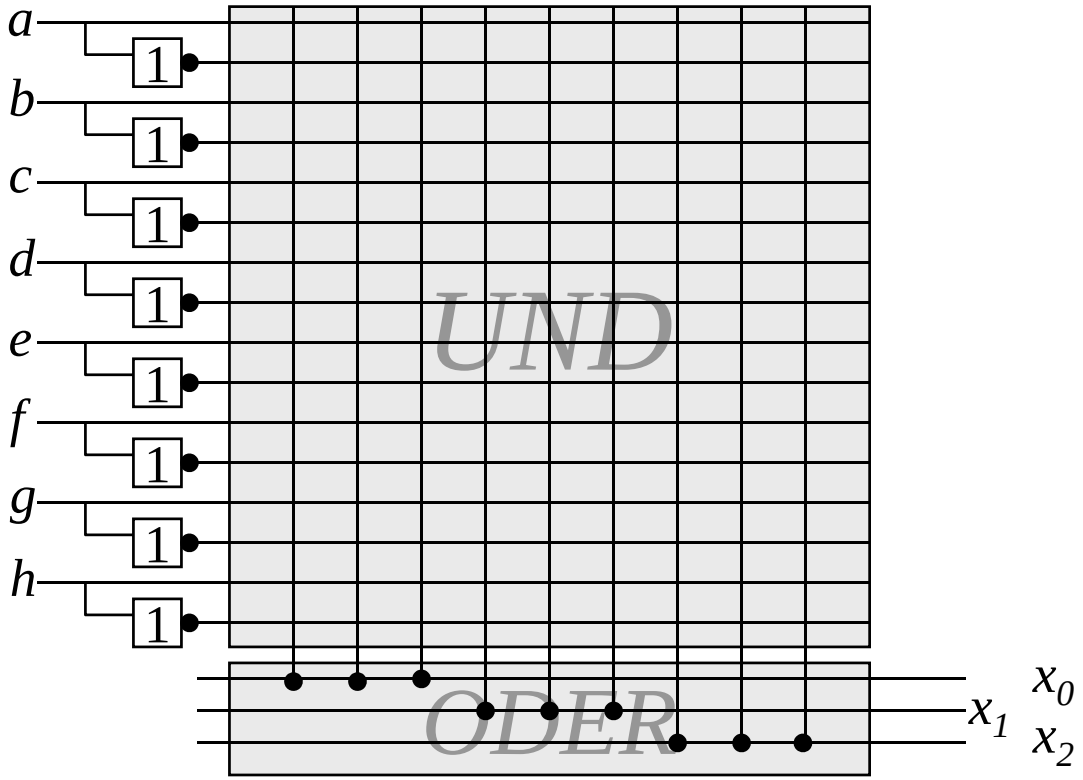
Realisierung mit einem PLA

- **Programmable Logic Array**
  - ⇨ **Technische Realisierung der DMF**
  - ⇨ **UND- und ODER-Matrix sind frei programmierbar**



Realisierung mit einem PAL

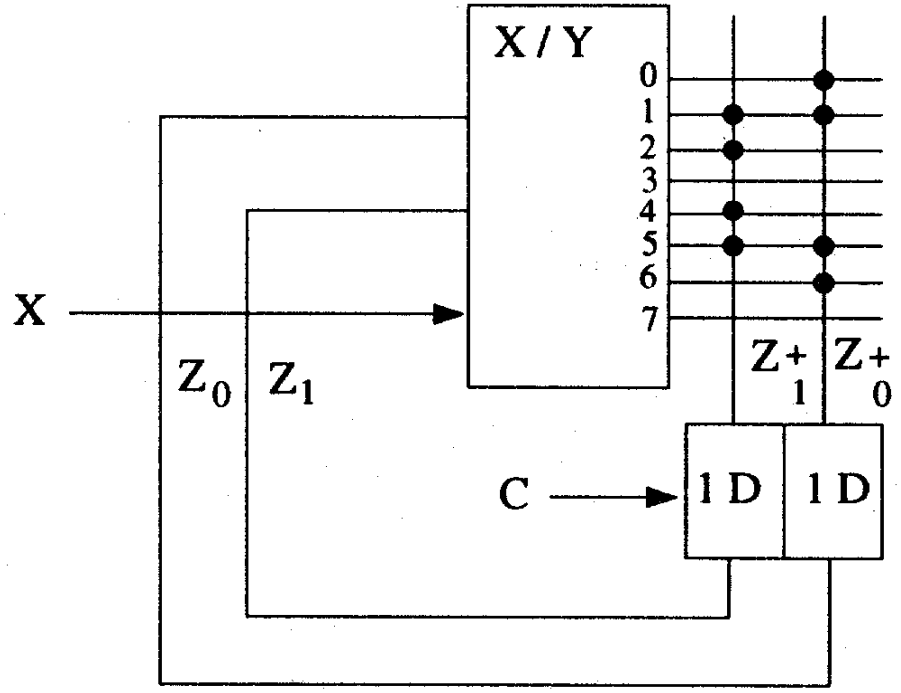
- **Programmable Array Logic**
  - ⇨ **die ODER-Matrix ist vorgegeben**
  - ⇨ **es steht eine feste Anzahl von Implikanten pro Ausgang zur Verfügung**
  - ⇨ **die UND-Matrix ist programmierbar**



Realisierung mit einem ROM

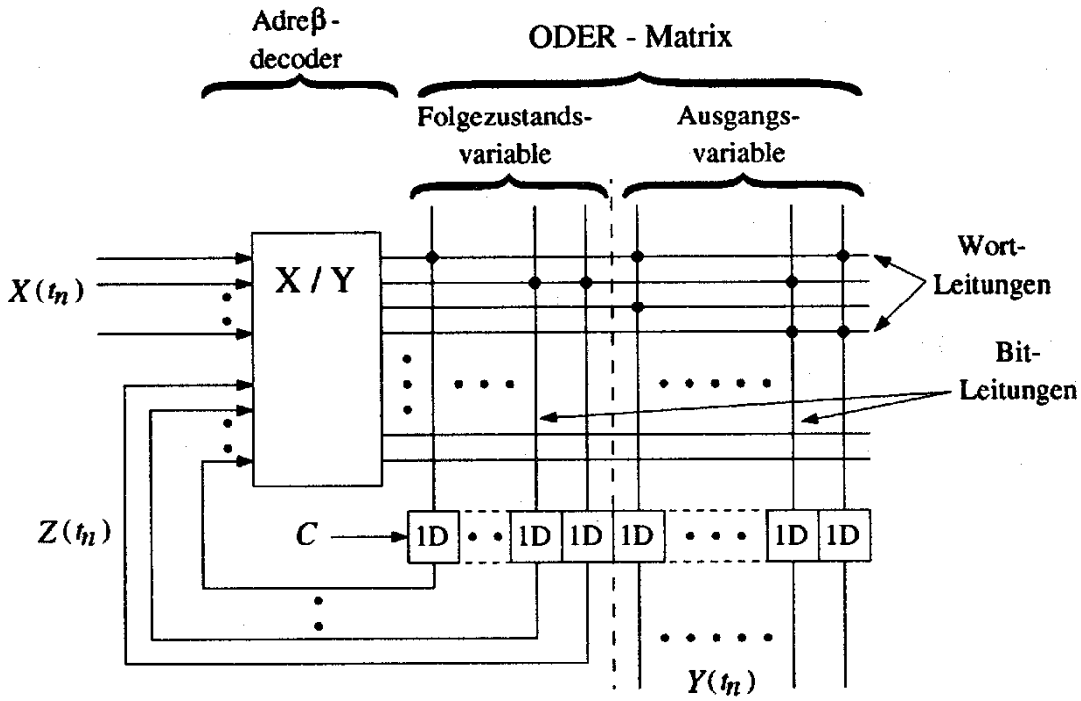
- Technische Realisierung durch ein PROM, EPROM, EEPROM
- Die UND-Matrix ist durch den Adressdekodierer vorgegeben
  - ⇨ alle Minterme sind implementiert
  - ⇨ direkte Implementierung der Funktionstabelle

X	Z_1	Z_0	Z_1^+	Z_0^+
0	0	0	0	1
0	0	1	1	1
0	1	1	1	0
0	1	0	0	0
1	0	0	1	0
1	1	0	1	1
1	1	1	0	1
1	0	1	0	0



Realisierung mit einem ROM

- Auch die Ausgabefunktion kann mit einem ROM realisiert werden
  - ⇨ Wortorientierung des ROMs wird ausgenutzt
  - ⇨ Mikroprogramm
  - ⇨ Mögliche Implementierung des Steuerwerkes in Mikroprozessoren

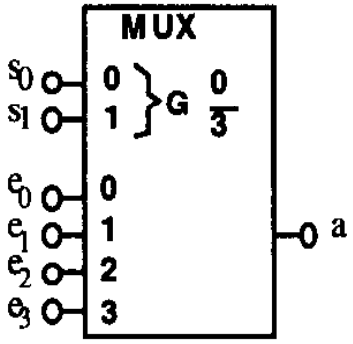


Spezielle Schaltnetze und Schaltwerke

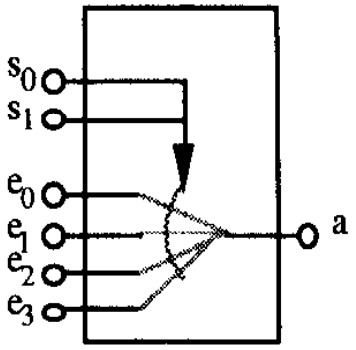
- Für die Implementierung komplexer Schaltungen werden häufig immer wieder kehrende Bausteintypen verwendet
- Typische Schaltnetze sind
  - ⇨ Multiplexer/Demultiplexer
  - ⇨ Vergleicher
  - ⇨ Addierer
  - ⇨ Multiplizierer
- Typische Schaltwerke sind
  - ⇨ Register
  - ⇨ Schieberegister
  - ⇨ Zähler

Multiplexer

- Mehrere Eingänge, ein Ausgang
- über n Steuerleitungen können 2^n Eingänge ausgewählt und an den Ausgang durchgeschaltet werden



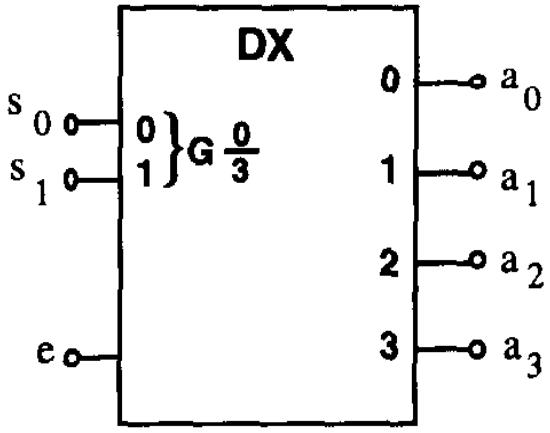
Steuerleitungen		Ausgang
s_1	s_0	a
0	0	e_0
0	1	e_1
1	0	e_2
1	1	e_3



Schaltbild und logisches Verhalten eines 1-aus-4-Multiplexers

Demultiplexer

- Ein Eingang wird auf einen aus 2^n Ausgängen durchgeschaltet

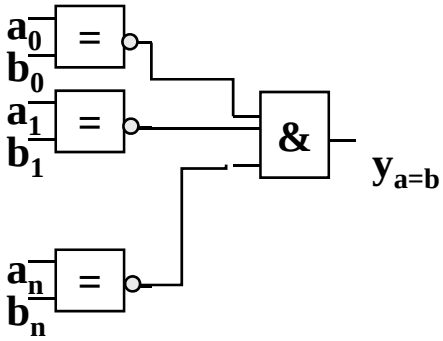


s_1	s_0	a_0	a_1	a_2	a_3
0	0	e	0	0	0
0	1	0	e	0	0
1	0	0	0	e	0
1	1	0	0	0	e

Schaltbild und logisches Verhalten eines 1-auf-4-Demultiplexers

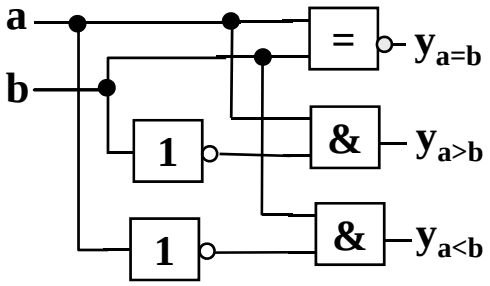
Vergleicher (Komparatoren)

- **Vergleich zweier Zahlen**
  - ⇨ **A=B, A<B, A>B**
- **Gleichheit bedeutet, dass alle Bits übereinstimmen**



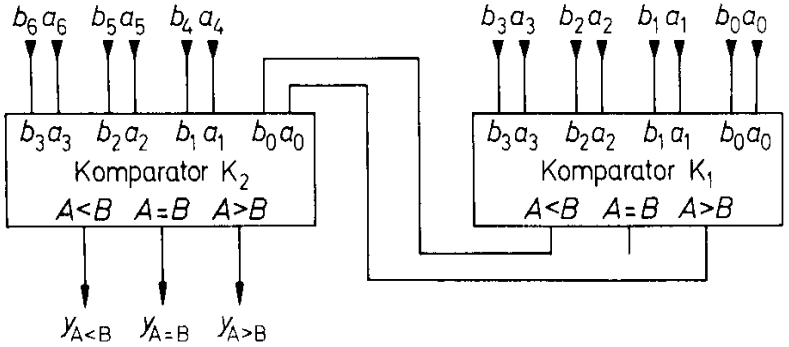
- **1-Bit Komparator mit Größenvergleich**

a	b	$y_{a>b}$	$y_{a=b}$	$y_{a<b}$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

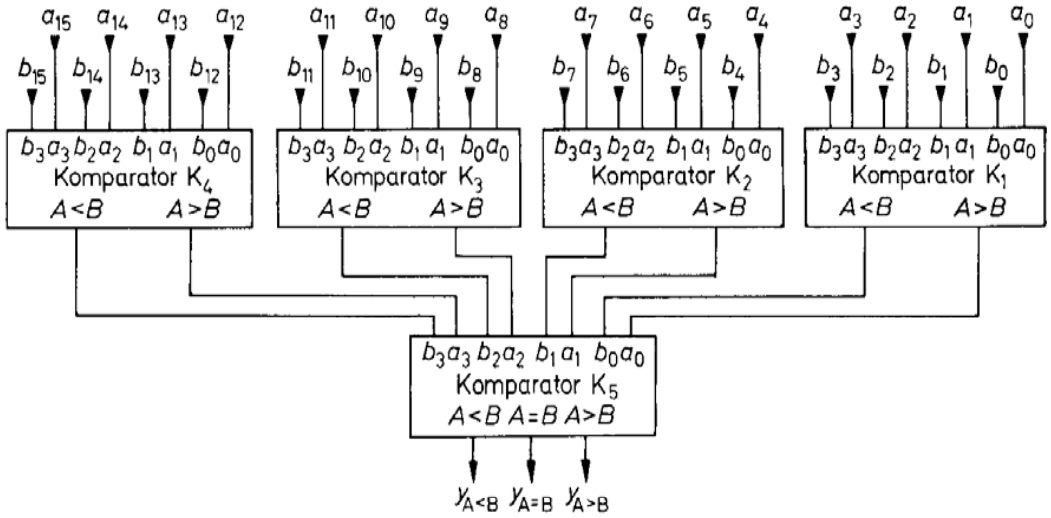


Komparatoren

- **Serielle Erweiterung**



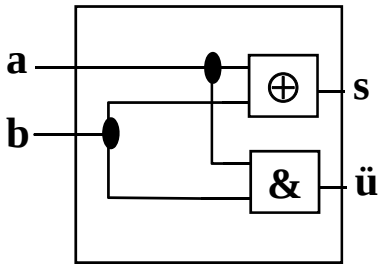
- **Parallele Erweiterung**



Addierer

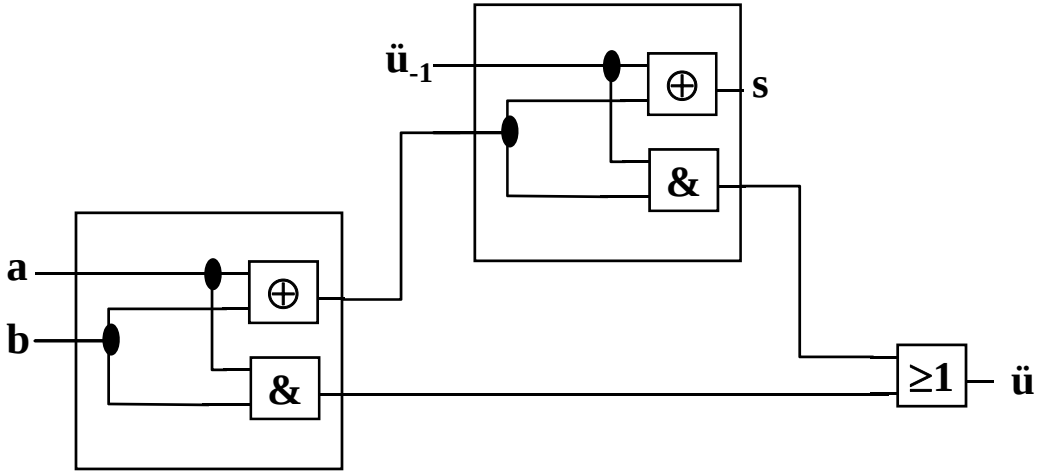
Halbaddierer

a	b	s	ü
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



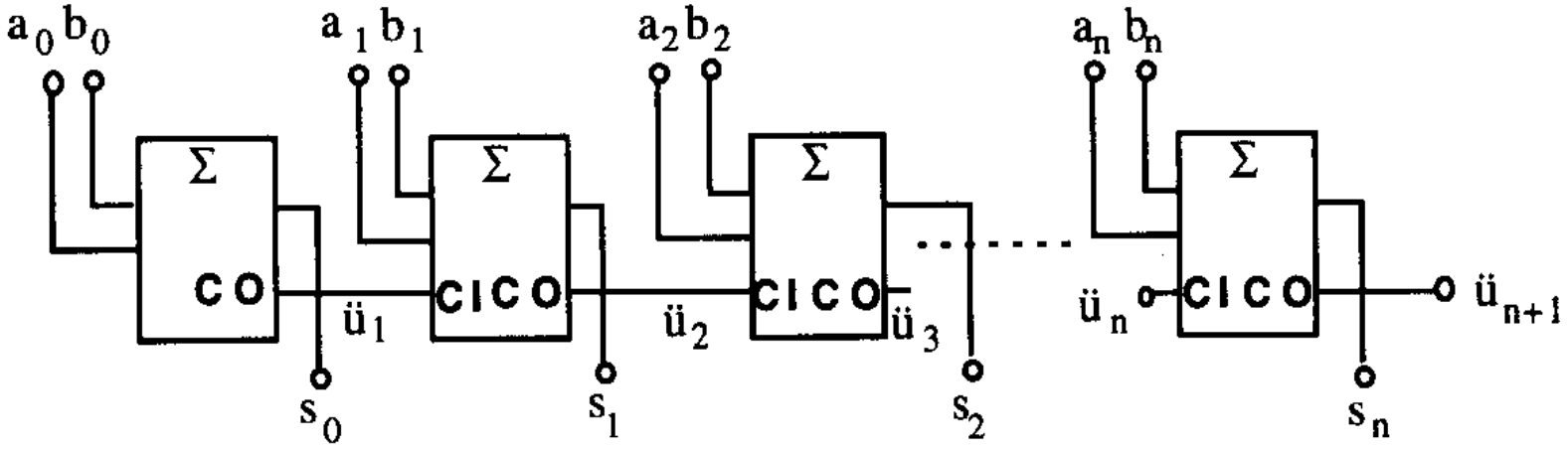
Volladdierer

a	b	$ü_{-1}$	s	ü
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Addition mit seriellem Übertrag

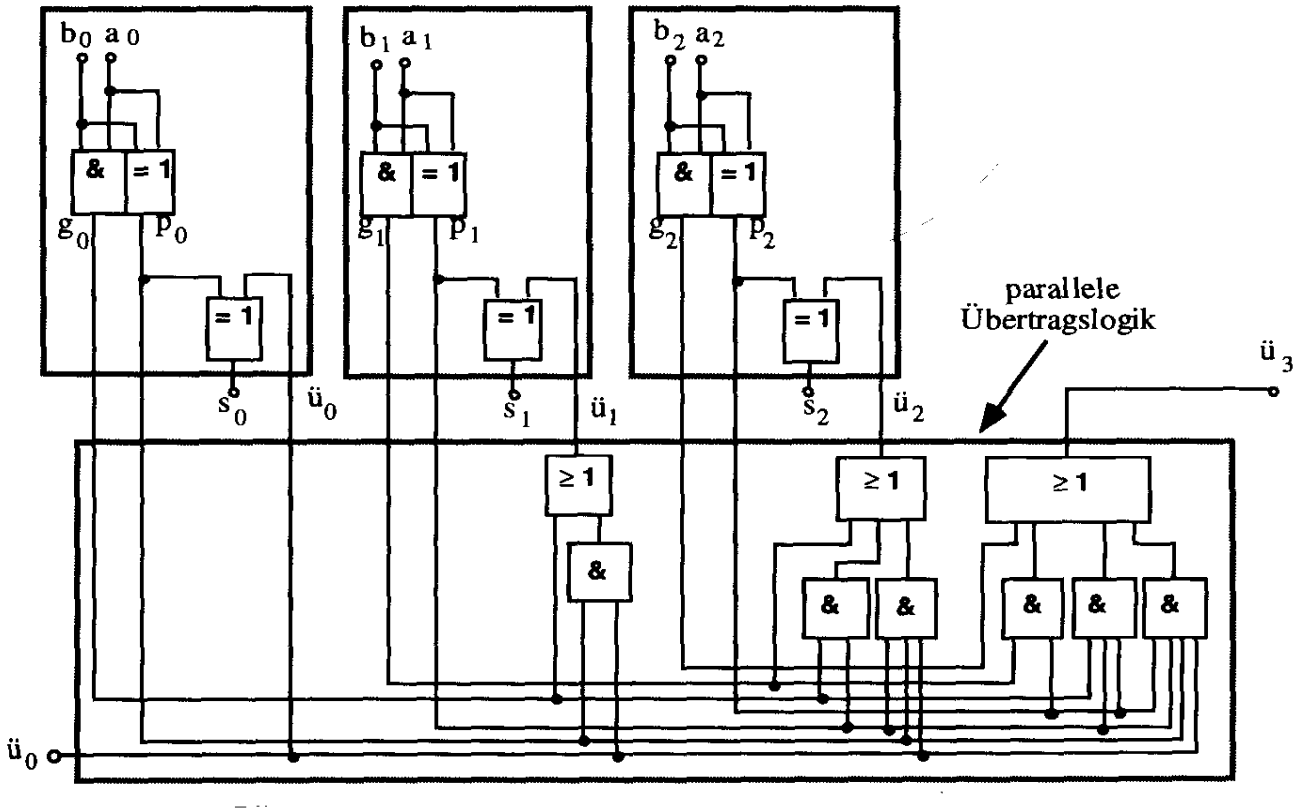
- Der Übertrag des Volladdierers $ü_i$ wird mit c_{i+1} verbunden



Addierer mit paralleler Übertragslogik

- **Allgemein:**

$$c_i = a_i b_i \vee (a_i \oplus b_i) c_{i-1}$$



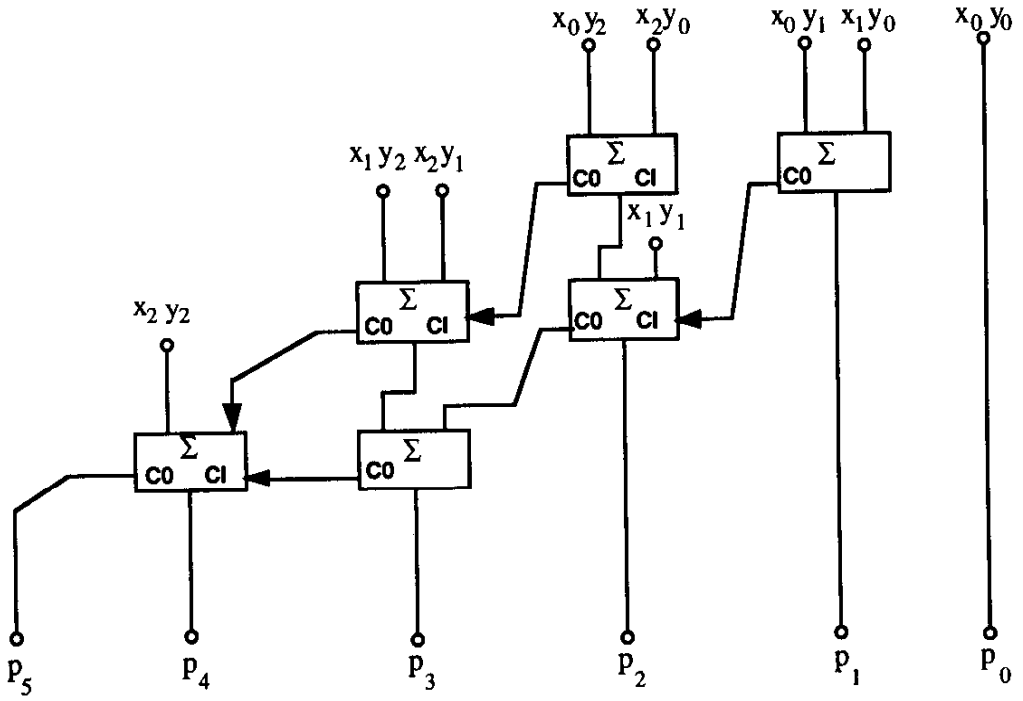
Multiplizierer

- **Parallele Multiplikation durch Addierwerk**

$$p = x \cdot y = \left(\sum_{i=0}^{n-1} x_i \cdot 2^i \right) \cdot \left(\sum_{j=0}^{n-1} y_j \cdot 2^j \right) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \cdot 2^{i+j} x_i y_j$$

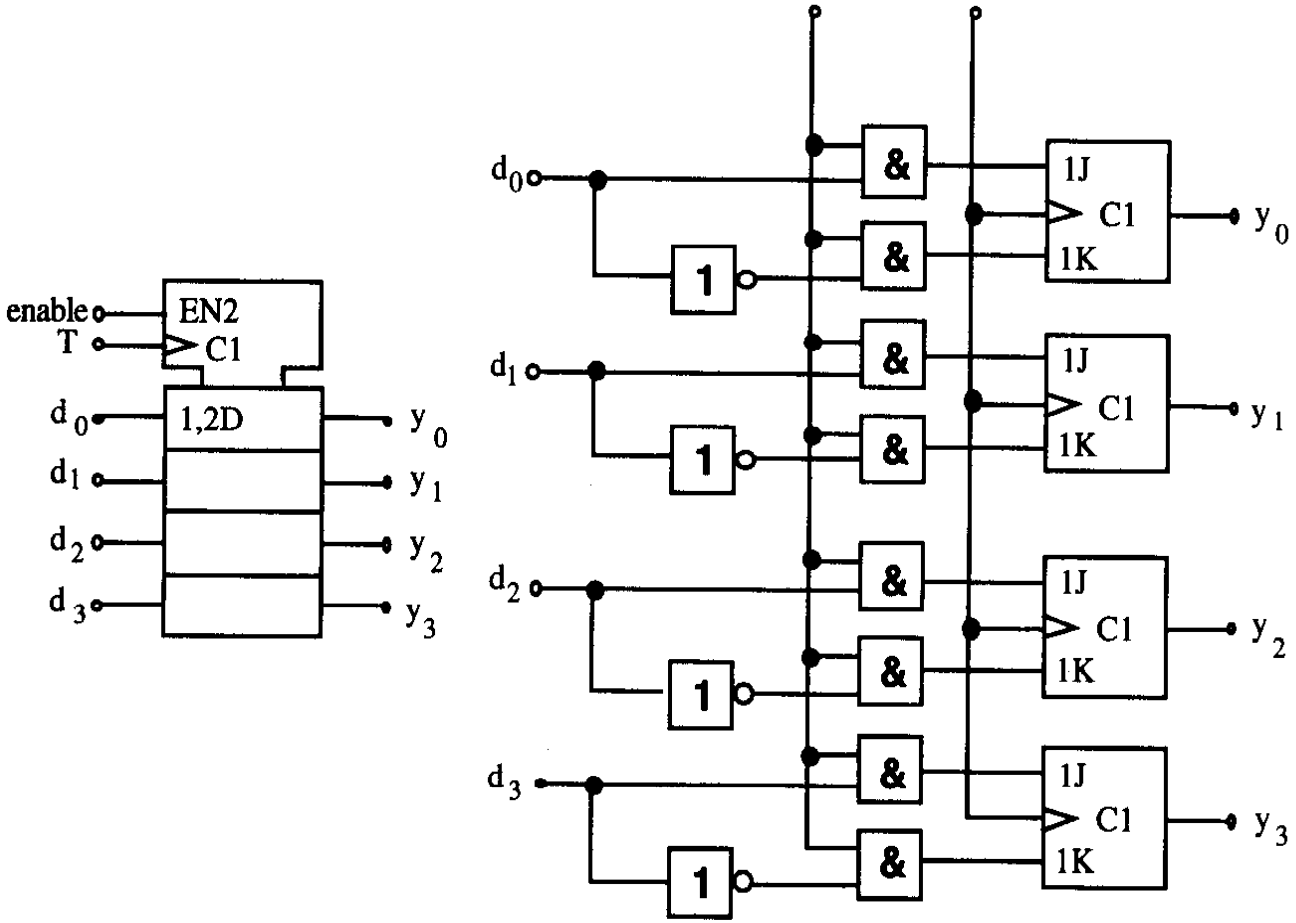
- **für n= 3: ($x_i y_j$ steht für x_i UND y_i)**

1	1	0	•	1	0	1
1	1	0				
	0	0	0			
		1	1	0		
1	1	1	1	0		



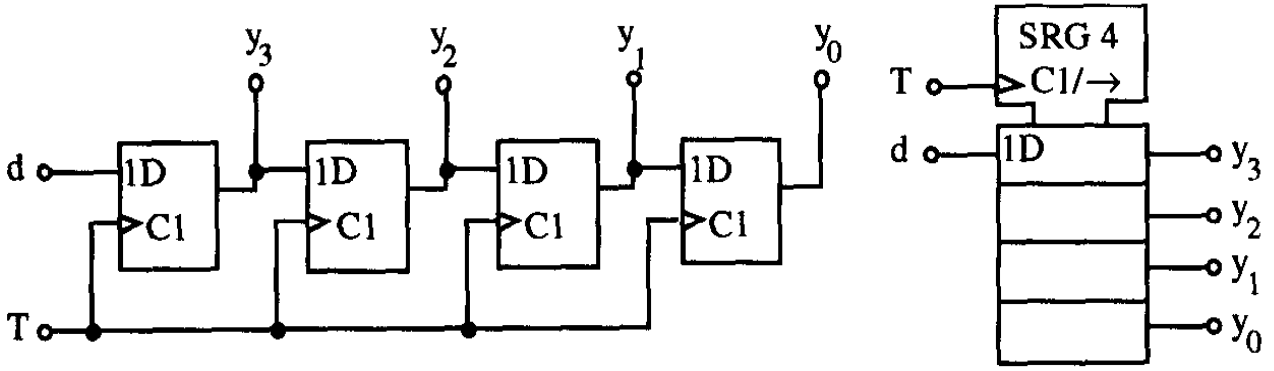
Register

- Speicherung einer n-stelligen Zahl durch n Flipflops



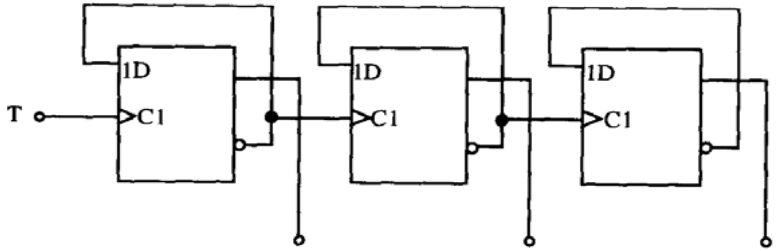
Schieberegister

- Kette von Flipflops
- Anwendungen:
  - ⇨ Serien-Parallel-Wandlung
  - ⇨ Parallel-Serien-Wandlung
  - ⇨ FIFO oder Stapel-Speicher
  - ⇨ Multiplikation mit 2 oder Division durch 2
  - ⇨ mit Rückkopplung zur Erzeugung komplexer Signalfolgen (Sequenzer)

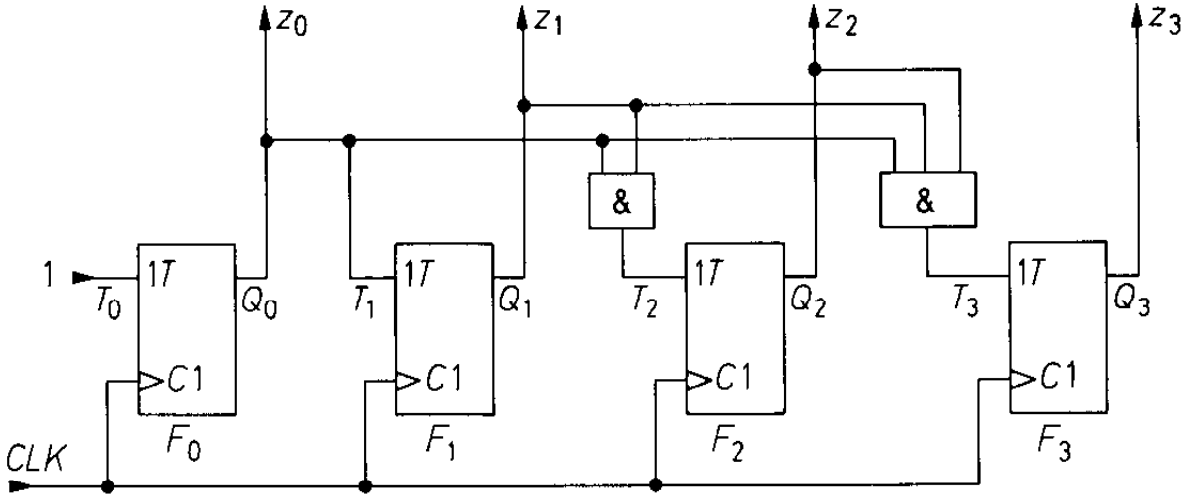


Zähler

- Einfacher Dualzähler durch Rückkopplung
- Asynchroner Ripple Carry Zähler

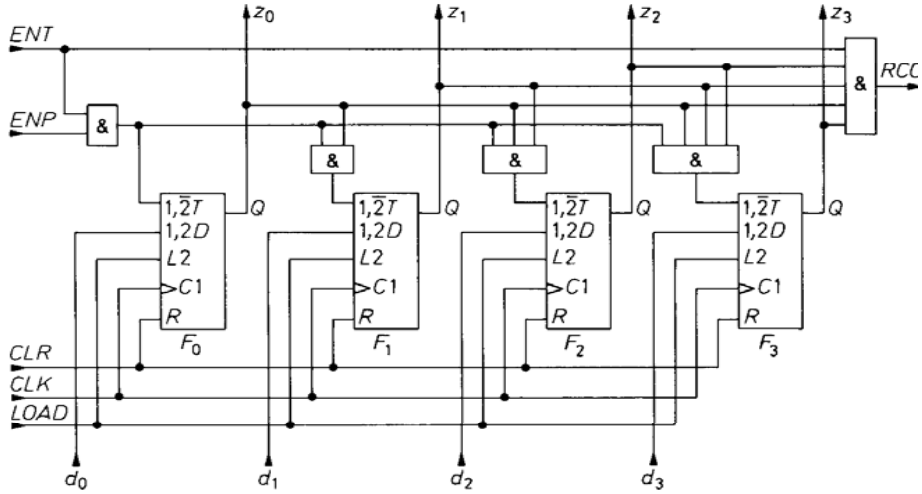


- Synchroner Dualzähler durch Carry-Look-Ahead-Logik

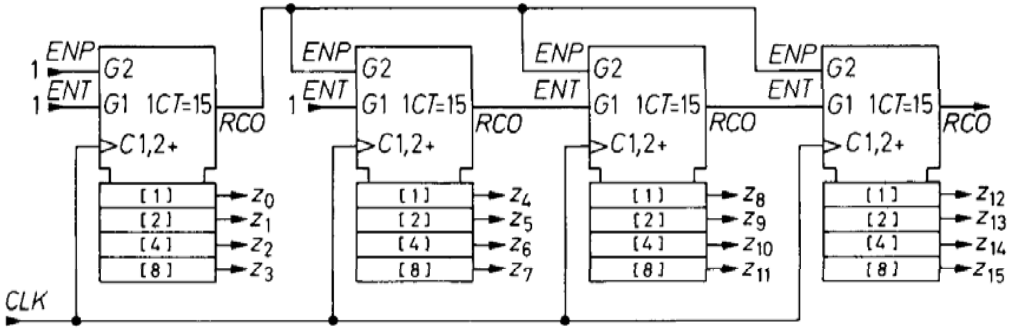


Zähler

- **Praktische Ausführung eines Zählers**



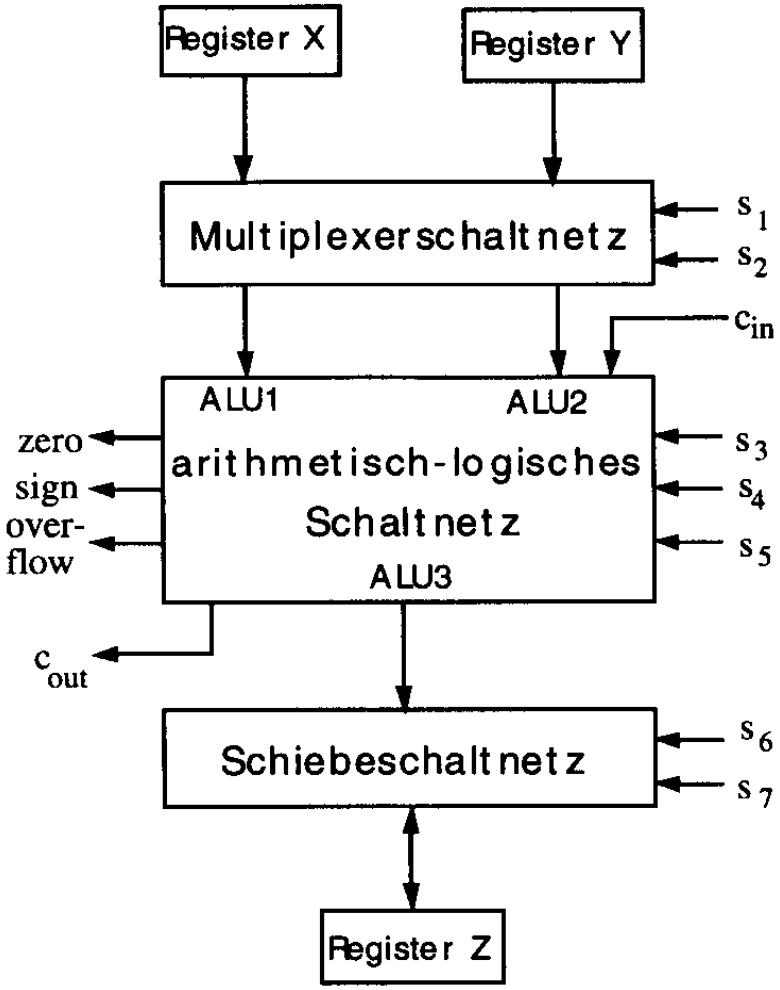
- **Kaskadierung eines Zählers**



Bauelemente eines Rechnersystems

- Multiplexer und Demultiplexer zur Steuerung des Datenflusses
- Zähler für die Programmsteuerung
- ALU
  - ⇨ Register
  - ⇨ Addierer
  - ⇨ Multiplizierer
  - ⇨ Schieberegister
- Speicherzellen
  - ⇨ RAM
  - ⇨ ROM

Aufbau einer ALU



s_1	s_2	ALU1	ALU2
0	0	X	Y
0	1	X	0
1	0	Y	0
1	1	Y	X

s_3	s_4	s_5	ALU3
0	0	0	ALU1+ALU2+c_{in}
0	0	1	ALU1-ALU2-$\overline{c_{in}}$
0	1	0	ALU2-ALU1-$\overline{c_{in}}$
0	1	1	ALU1 $\vee$ ALU2
1	0	0	ALU1 $\wedge$ ALU2
1	0	1	$\overline{\text{ALU1}}$ $\wedge$ ALU2
1	1	0	ALU1 $\not\leftrightarrow$ ALU2
1	1	1	ALU1 $\leftrightarrow$ ALU2

s_6	s_7	Z
0	0	ALU3
0	1	ALU3 / 2
1	0	ALU3 * 2
1	1	Z speichern

Rechnerarithmetik

- **Die Rechnerarithmetik behandelt**
  - ⇨ **die Darstellung von Zahlen**
  - ⇨ **Verfahren zur Berechnung der vier Grundrechenarten**
  - ⇨ **Schaltungen, die diese Verfahren implementieren**

Formale Grundlagen

- **Menschen rechnen und denken im Dezimalsystem**
- **Die meisten Rechner verwenden das Dualsystem**
  - ⇨ **man benötigt Verfahren der Konvertierung, die sich algorithmisch umsetzen lassen**

Zahlensysteme

- **Stellenwertsysteme**
  - ⇨ **jeder Position i der Ziffernreihe ist ein Stellenwert zugeordnet welcher der Potenz b^i der Basis b eines Zahlensystems entspricht** $z_n z_{n-1} \ldots z_1 z_0 . z_{-1} z_{-2} z_{-m}$
  - ⇨ **der Wert X_b ergibt sich aus der Summe der Werte aller Einzelstellen**

$$X_b = z_n b^n + z_{n-1} b^{n-1} + \ldots + z_1 b + z_0 + z_{-1} b^{-1} + z_{-2} b^{-2} + z_{-m} b^{-m} = \sum_{i=-m}^{n} z_i b^i$$

Die wichtigsten Zahlensysteme

b	Zahlensystem	Ziffern
2	Dualsystem	0, 1
8	Oktalsystem	0, 1, 2, 3, 4, 5, 6, 7
10	Dezimalsystem	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16	Hexadezimalsystem	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- Dualsystem kann direkt auf 2-wertige Logik umgewandelt werden
- Oktal- und Hexadezimalsystem sind Kurzschreibweisen der Zahlen im Dualsystem
  - ⇨ sie lassen sich leicht in Zahlen des Dualsystems umwandeln

Umwandlung vom Dezimalsystem in ein Zahlensystem zur Basis b

- **Euklidischer Algorithmus**
  - ⇨ **die einzelnen Ziffern werden sukzessive berechnet**

$$Z = z_n 10^n + z_{n-1} 10^{n-1} + \ldots + z_1 10 + z_0 + z_{-1} 10^{-1} + z_{-2} 10^{-2} + z_{-m} 10^{-m}$$

$$= y_p b^p + y_{p-1} b^{p-1} + \ldots + y_1 b + y_0 + y_{-1} b^{-1} + y_{-2} b^{-2} + y_{-q} b^{-q}$$

  - ⇨ **Algorithmus**
    1. **Berechne P gemäß der Ungleichung** $b^p \le Z < b^{p+1}$
    2. **Ermittle** y_p **und den Rest** R_p **durch Division von** Z **durch** b^p

       $y_p = Z \; div \; b^p;$ $\quad R_p = Z \; mod \; b^p;$ $\quad y_p = \{0, 1, \ldots, b\text{-}1\}$
    3. **Wiederhole 2. für** $i = p\text{-}1$ **und ersetze dabei nach jedem Schritt** Z **durch** R_i**, bis** $R_i = 0$ **oder bis** b_i **klein genug ist**

Beispiel

- **Umwandlung von** $15741{,}233_{10}$ **ins Hexadezimalsystem**

1. Schritt	$16^3 \leq 15741{,}233_{10} < 16^4$	**höchste Potenz** 16^3
2. Schritt	$15741{,}233_{10} : 16^3 = 3$	**Rest** 3453,233
3. Schritt	$3453{,}233 : 16^2 = D$	**Rest** 125,233
4. Schritt	$125{,}233 : 16 = 7$	**Rest** 13,233
5. Schritt	$13{,}233 : 1 = D$	**Rest** 0,233
6. Schritt	$0{,}233 : 16^{-1} = 3$	**Rest** 0,0455
7. Schritt	$0{,}0455 : 16^{-2} = B$	**Rest** 0,00253
8. Schritt	$0{,}00253 : 16^{-3} = A$	**Rest** 0,000088593
9. Schritt	$0{,}000088593 : 16^{-4} = 5$	**Rest** 0,000012299

Fehler

Ergebnis: $15741{,}233_{10} = 3D7D{,}3BA5_{16}$

Umwandlung vom Dezimalsystem in eine Zahl zur Basis b

- **Horner-Schema**
  - **Eine ganze Zahl X_b kann auch in der folgenden Form dargestellt werden:**

$$X_b = (((...(((y_n b + y_{n-1})b + y_{n-2})b + y_{n-3})b...)b + y_1)b + y_0$$

- **Die gegebene Dezimalzahl wird sukzessive durch die Basis b dividiert**
  - **Die jeweiligen ganzzahligen Reste ergeben die Ziffern der Zahl X_b**
  - **Reihenfolge: niederwertige zur höchstwertige Stelle**
- **Beispiel: Umwandlung von 15741_{10} ins Hexadezimalsystem**

$15741_{10} : 16 = 983$	**Rest** 13	(D_{16})
$983_{10} : 16 = 61$	**Rest** 7	(7_{16})
$61_{10} : 16 = 3$	**Rest** 13	(D_{16})
$3_{10} : 16 = 0$	**Rest** 3	(3_{16})

Ergebnis: $15741_{10} = 3D7D_{16}$

Umwandlung des Nachkommateils

- **Der Nachkommateil einer Zahl X_b kann in der folgenden Form dargestellt werden**

$$Y_b = ((...((y_{-m}b^{-1} + y_{-m+1})b^{-1} + y_{-m+2})b^{-1} + ... + y_{-2})b^{-1} + y_{-1})b^{-1}$$

- **Sukzessive Multiplikation des Nachkommateils der Dezimalzahl mit der Basis b des Zielsystems ergibt nacheinander die y_{-i}**
- **Beispiel: Umwandlung von $0{,}233_{10}$ ins Hexadezimalsystem**

0,233 * 16	= 3,728	$z_{-1} = 3$
0,728 * 16	= 11,648	$z_{-2} = B$
0,648 * 16	= 10,368	$z_{-3} = A$
0,368 * 16	= 5,888	$z_{-4} = 5$

Ergebnis: $0{,}233_{10} = 0{,}3BA5_{16}$

Umwandlung einer Zahl zur Basis *b* ins Dezimalsystem

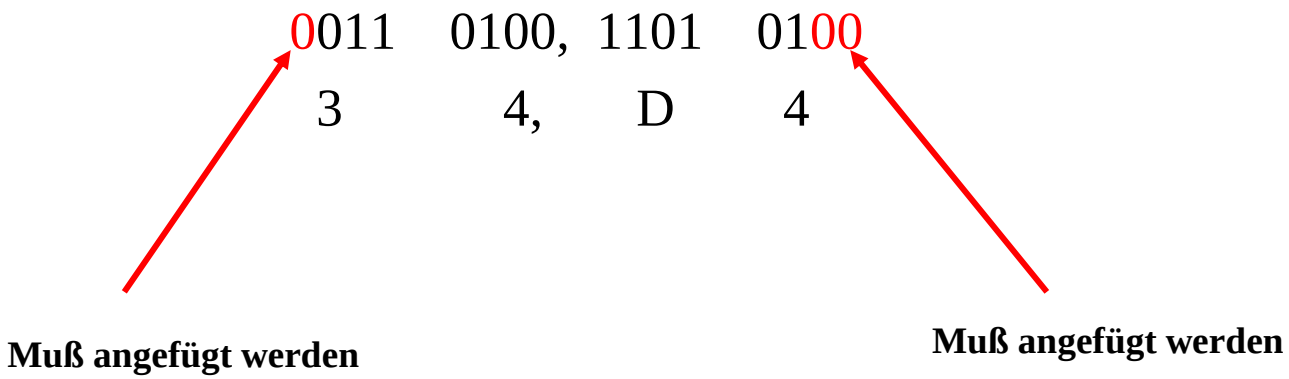
- **Werte der einzelnen Stellen werden mit deren Wertigkeit multipliziert und aufsummiert**
- **Beispiel: Umwandlung von** 101101,1101 **ins Dezimalsystem**

101101,1101

$1 * 2^{-4}$	=	0,0625
$0 * 2^{-3}$	=	0
$1 * 2^{-2}$	=	0,25
$1 * 2^{-1}$	=	0,5
$1 * 2^{-0}$	=	1
$0 * 2^{1}$	=	0
$1 * 2^{2}$	=	4
$1 * 2^{3}$	=	8
$0 * 2^{4}$	=	0
$1 * 2^{5}$	=	32
		$45{,}8125_{10}$

Weitere Umwandlungen

- **Umwandlung zwischen zwei beliebigen Zahlensystemen**
  - ⇨ **zwei Schritte: Umwandlung ins Dezimalsystem und danach vom Dezimalsystem ins Zielsystem**
- **Spezialfall: Eine Basis eine Potenz der anderen Basis**
  - ⇨ **Umwandlung erfolgt durch Zusammenfassen der Stellen**
  - ⇨ **Beispiel: Umwandlung von** $0110100{,}110101_2$ **ins Hexadezimalsystem**



Kodierung zur Zahlen- und Zeichendarstellung

- **Die Dezimalzahlen können auch ziffernweise in eine Binärdarstellung überführt werden**
  - ⇨ **um die 10 Ziffern 0 bis 9 darstellen zu können, benötigt man 4 Bit**
  - ⇨ **eine solche 4er-Gruppe wird Tetrade genannt**
    - **Pseudotetraden: 6 der 16 Kodierungen stellen keine gültigen Ziffern dar**
- **BCD**
  - ⇨ **Binary Coded Decimals**
  - ⇨ **man verwendet das Dualäquivalent der ersten 10 Dualzahlen**
  - ⇨ **Beispiel:**

$$8127_{10} = 1000\ 0001\ 0010\ 0111_{BCD} = 1111110111111_2$$

  - ⇨ **Nachteile der BCD-Kodierung**
    - **höherer Platzbedarf**
    - **aufwändige Implementierung der Rechenoperationen**

Grey-Kodierung

- **Einschrittige Kodierung**
  - ⇨ **bei benachbarten Zahlen ändert sich nur ein Binärzeichen**
- **Vorteil**
  - ⇨ **keine Hazards bei der Analog/Digitalwandlung und bei Abtastern**
- **Nachteil**
  - ⇨ **keine Stellenwertigkeit**
  - ⇨ **aufwändige Rechenoperationen**

Dezimalzahl	Grey-Kodierung
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Kodierung von Zeichen

- **American Standard Code for Information Interchange (ASCII)**
  - ⇨ **7 Bit-Kodierung für 128 Zeichen**
  - ⇨ **2*26 Zeichen, 10 Ziffern und 32 Kommunikationssteuerzeichen**
- **Umlaute und Sonderzeichen sind nicht enthalten**
  - ⇨ **8-Bit Erweiterungen unterschiedlicher Computerhersteller**
  - ⇨ **Andere Verwendung des 8. Bits: Paritätsprüfung**

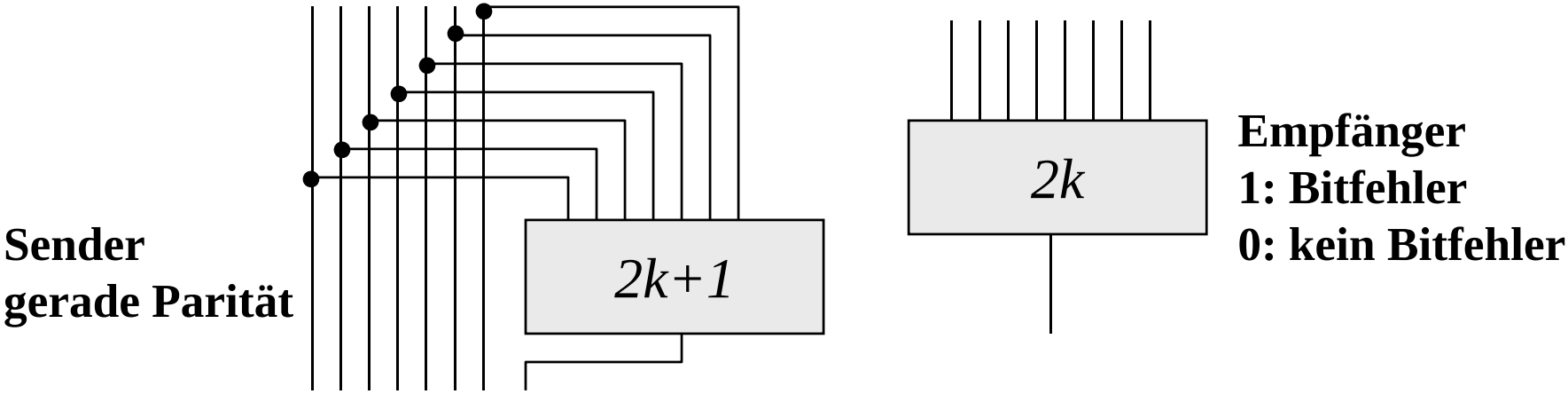
ASCII-Tabelle

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	'	p
0001	SOH	DC 1	!	1	A	Q	a	q
0010	STX	DC 2	"	2	B	R	b	r
0011	ETX	DC 3	#	3	C	S	c	s
0100	EOT	DC 4	$	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	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Die höchstwertigen Bits der Kodierung eines Zeichens sind in der Kopfzeile abzulesen, die niederwertigen Bits in der ersten Spalte (Beispiel: A $\rightarrow 100\,0001_2$).

Paritätsprüfung

- **Problem:**
  - ⇨ **Erkennung von Übertragungsfehlern**
- **Prinzip:**
  - ⇨ **die 7-Bit Kodierung wird beim Sender so auf 8 Bit ergänzt, dass stets eine gerade (ungerade) Anzahl von Einsen ergänzt**
    - **gerade (ungerade) Parität**
  - ⇨ **beim Empfänger wird diese Eigenschaft überprüft**
    - **falls bei der Übertragung ein Bitfehler auftritt, wird dieser erkannt**



Beispiel: Paritätsprüfung

- **Gerade Parität (even parity)**
  - ⇨ **Paritätsbit ‚0' wenn gerade, ‚1' wenn ungerade**
  - ⇨ **Codewort 0110110 hat 4 ‚1', also gerade →00110110**
  - ⇨ **Codewort 0111110 hat 5 ‚1', also gerade →10111110**

- **Ungerade Parität (odd parity)**
  - ⇨ **Paritätsbit ‚0' wenn ungerade, ‚1' wenn gerade**
  - ⇨ **Codewort 0110110 hat 4 ‚1', also gerade →10110110**
  - ⇨ **Codewort 0111110 hat 5 ‚1', also gerade →00111110**

Darstellung negativer Zahlen

- **Für die Darstellung von Zahlen in Rechnern werden vier verschiedene Formate benutzt**
  - ⇨ **Darstellung mit Betrag und Vorzeichen**
  - ⇨ **Stellenkomplement (Einerkomplement)**
  - ⇨ **Zweierkomplement**
  - ⇨ **Offset-Dual-Darstellung (Charakteristik)**

Darstellung mit Betrag und Vorzeichen

- **Die erste Stelle der Zahl wird als Vorzeichen benutzt**
  - ⇨ **0: Die Zahl ist positiv**
  - ⇨ **1: Die Zahl ist negativ**
- **Beispiel:**
  - ⇨ `0001 0011 = + 19`
  - ⇨ `1001 0011 = - 19`
- **Nachteile dieser Darstellung**
  - ⇨ **bei Addition und Subtraktion müssen die Vorzeichen getrennt betrachtet werden**
  - ⇨ **es gibt 2 Repräsentanten der Zahl 0**
    - **positives und negatives Vorzeichen**

Einerkomplement

- **Jede Ziffer der Binärzahl wird negiert**
  - ⇨ **negative Zahlen werden ebenfalls durch eine 1 an der 1. Stelle gekennzeichnet**
- **Vorteil:**
  - ⇨ **die 1. Stelle muss bei Addition und Subtraktion nicht gesondert betrachtet werden**
- **Beispiel:**

	2		0010	
+	-3	+	1100	(Komplement: 0011)
=	-1	=	1110	(Komplement: 0001)

- **Nachteil:**
  - ⇨ **es gibt 2 Repräsentanten der Zahl 0:**
    - **0000 und 1111**

Zweierkomplement

- **Addiert man zum Einerkomplement noch 1 hinzu, dann fallen die beiden Darstellungen der Zahl 0 durch den Überlauf wieder aufeinander**
  - **Die Zahl 0** `0000`
  - **Einerkomplement** `1111`
  - **Zweierkomplement** `1111 + 0001 = 0000`
- **Vorteile**
  - **das 1. Bit enthält das Vorzeichen**
  - **direkte Umwandlung der Zahl Z über die Stellenwertigkeit**

- **Beispiel** $Z = -z_n \cdot 2^n + z_{n-1} \cdot 2^{n-1} + \ldots + z_1 \cdot 2 + z_0$
  - **Die Zahl** 54 = 00110110_2
  - **mit Vorzeichenbit** -54_{10} = 10110110_2
  - **Einerkomplement** = 11001001_2
  - **Zweierkomplement** = 11001010_2

Addition im Zweierkomplement

- **Beispiel:**

```
    73        01001001
   -54        11001010
  ------------------------
 =  19     (1)00010011
```

- **Beispiel:**

```
    37        00100101
   -54        11001010
  ------------------------
 = -17        11101111     (00010001)
```

Charakteristik

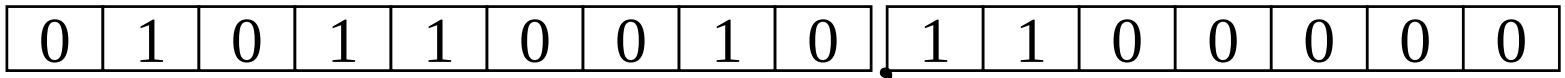
- **Hauptsächlich in der Darstellung von Exponenten für Gleitkommazahlen**
  - ⇨ **der gesamte Zahlenbereich wird durch die Addition einer Konstanten so nach oben verschoben, dass die kleinste Zahl die Darstellung 0...0 erhält**

- **Übersicht der Zahlendarstellungen**

Dez.	Betrag mit Vorz.	Einerkomp.	Zweierkomp.	Charakteristik
-4	---	---	100	000
-3	111	100	101	001
-2	110	101	110	010
-1	101	110	111	011
0	100 oder 000	000 oder 111	000	100
1	001	001	001	101
2	010	010	010	110
3	011	011	011	111

Fest- und Gleitkommazahlen

- Darstellung von Zahlen mit einem Komma
- Festkommadarstellung
  - ⇨ Festlegung der Stelle in einem Datenwort



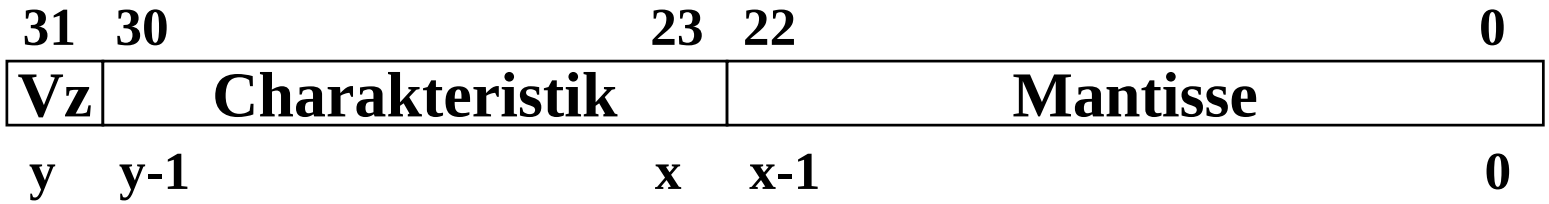
  - ⇨ wird heute hardwareseitig nicht mehr eingesetzt
- Gleitkommadarstellung
  - ⇨ Angabe der Stelle des Kommas in der Zahlendarstellung

$$Z = \pm \text{Mantisse} \cdot b^{\text{Exponent}}, b \in \{2,16\}$$

  - ⇨ negative Zahlen werden meist in Betrag und Vorzeichen dargestellt (kein Zweierkomplement)
  - ⇨ sowohl für die Mantisse als auch für die Charakteristik wird eine feste Anzahl von Speicherstellen vorgesehen

Fest- und Gleitkommazahlen

- **Gleitkommadarstellung**
  - **Bei der Mantisse ist die Lage des Kommas durch Vereinbarung festgelegt**
  - **Der Exponent ist eine ganze Zahl, die in Form seiner Charakteristik dargestellt wird**
  - **Charakteristik und Mantisse werden definiert festgelegt**
    - **Anzahl der Speicherbits**
  - **Länge der Charakteristik y-x bestimmt den Zahlenbreich**
  - **Länger der Mantisse x legt die Genauigkeit fest**



$$Dezimahlzahl = (-1)^{Vz} \cdot (0, Mantisse) \cdot b^{Exponent}$$

$$Exponent = Charakteristik - b^{(y-1)-x}$$

Normalisierte Gleitkommadarstellung

- **Eine Gleitkommazahl heißt normalisiert, wenn die folgende Beziehung gilt:**

$$1 \le \text{Mantisse} < b \quad \text{oder} \quad \frac{1}{b} \le \text{Mantisse} < 1 \qquad \frac{1}{2} \le \text{Mantisse} < 1$$

  - ⇨ **bei allen Zahlen außer der 0 ist die erste Stelle hinter dem Komma immer 1**
  - ⇨ **legt man für die Zahl 0 ein festes Bitmuster fest, kann man die erste 1 nach dem Komma weglassen**
- **Beispiel: Die Zahl** 7135_{10}
  - ⇨ **Festkommazahl**

    `0 000 0000 0000 0000 0001 1011 1101 1111`$_2$
  - ⇨ **Gleitkommadarstellung, normiert**

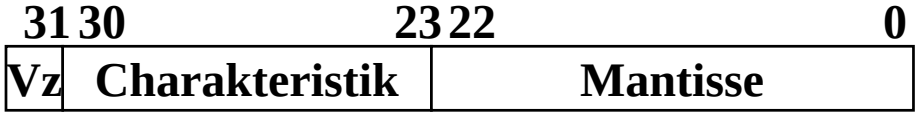
`0`	`100 0110 1`	`110 1111 0111 1100 0000 0000`

  - ⇨ **Gleitkommadarstellung, normiert, implizite erste 1**

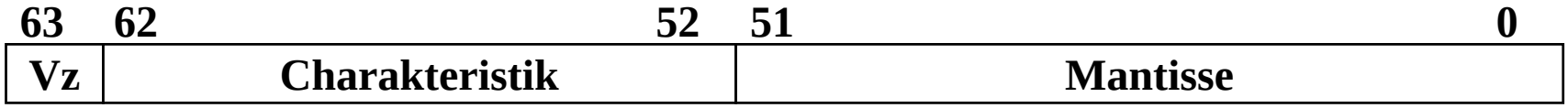
`0`	`100 0110 1`	`101 1110 1111 1000 0000 0000`

IEEE Gleitkommadarstellung

- Auch bei gleicher Wortbreite lassen sich unterschiedliche Gleitkommaformate definieren
  - ⇨ Normung durch IEEE
  - ⇨ einfache Genauigkeit (32 Bit)



  - ⇨ doppelte Genauigkeit (64 Bit)



- Eigenschaften
  - ⇨ Basis b ist gleich 2
  - ⇨ das erste Bit wird implizit zu 1 angenommen, wenn die Charakteristik nicht nur Nullen enthält
  - ⇨ Es wird so normalisiert, dass das erste Bit vor dem Komma steht

IEEE Gleitkommadarstellung

- **Zusammenfassung des 32-bit IEEE-Formats:**

Charakteristik	Zahlenwert
0	$(-1)^{Vz}$ 0,Mantisse*2^{-126}
1	$(-1)^{Vz}$ 1,Mantisse*2^{-126}
...	$(-1)^{Vz}$ 1,Mantisse*$2^{Charakteristik-127}$
254	$(-1)^{Vz}$ 1,Mantisse*2^{127}
255	Mantisse = 0: overflow, $(-1)^{Vz}\infty$
255	Mantisse $\neq$ 0: NaN (not a number)

- **Um Rundungsfehler zu vermeiden, wird intern mit 80 Bit gerechnet**

Addition und Subtraktion

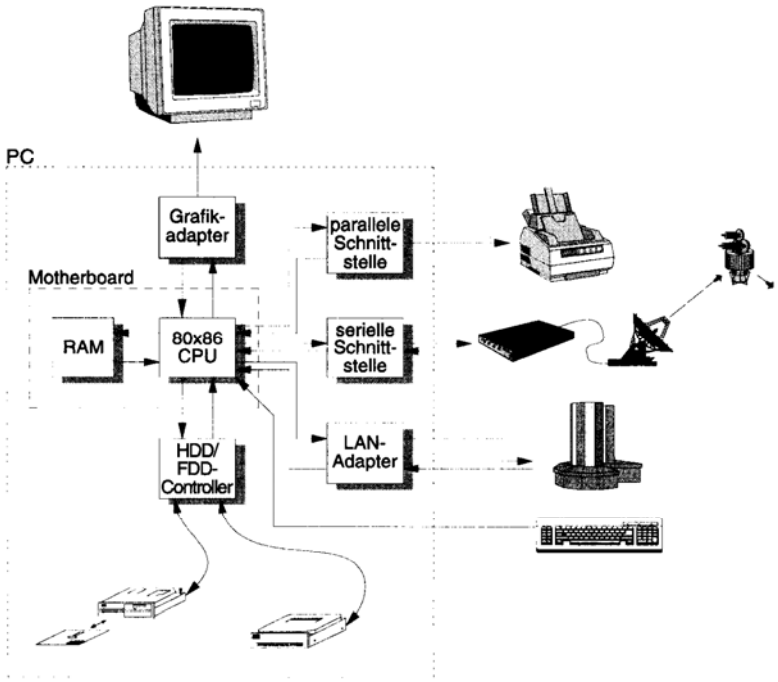
- Addition erfolgt Hilfe von Volladdierern wie im letzten Abschnitt beschrieben
  - ⇨ Ripple-Carry oder Carry-Look-Ahead Addierer
- Für die Subtraktion können ebenfalls Volladdierer verwendet werden
  - ⇨ `X -Y = X + (-Y)`
  - ⇨ Zweierkomplement berechnet sich über die Negation aller Bits mit einer 1 am ersten Übertrag des Addierers
- Bei Gleitkommazahlen müssen Mantisse und Exponent separat betrachtet werden
  - ⇨ Angleichen der Exponenten: Bilde die Differenz der Exponenten und verschiebe die Mantisse, die zum kleineren Exponenten gehört um die entsprechende Anzahl nach rechts
  - ⇨ Addition der Mantissen
  - ⇨ Normalisierung

Multiplikation und Division

- **Prinzip der Multiplikation: Schieben und Addieren**
- **Multiplikation von Zahlen im Zweierkomplement:**
  - ⇨ **die Zahlen werden in eine Form mit Betrag und Vorzeichen konvertiert**
  - ⇨ **die Beträge werden Multipliziert (kaskadiertes Addierwerk)**
  - ⇨ **das neue Vorzeichen wird berechnet (Exklusiv-ODER-Verknüpfung)**
- **Prinzip der Division: Schieben und Subtrahieren**
  - ⇨ **zwei Sonderfälle:**
    - **Division durch 0 muss eine Ausnahme auslösen**
    - **Die Division muss abgebrochen werden, wenn die vorgegebene Bitzahl des Ergebnisregisters ausgeschöpft ist**

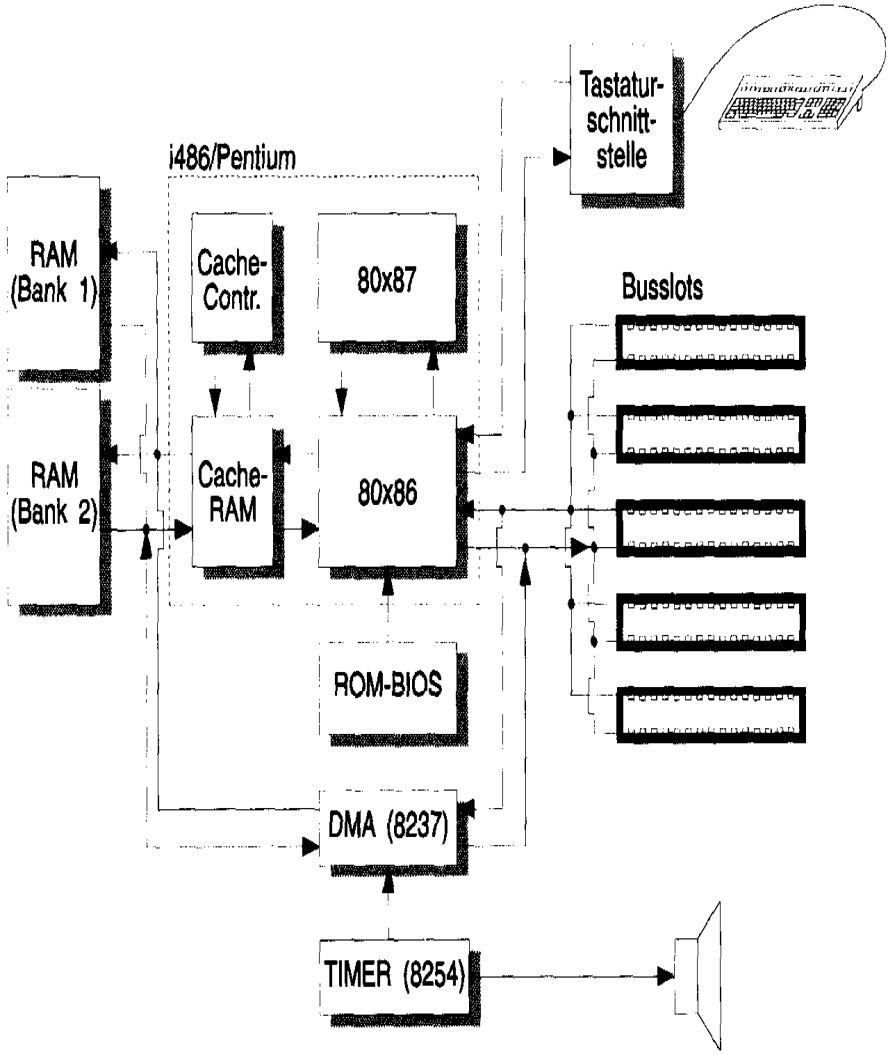
Aufbau von Rechnersystemen

- Speicher
  - ⇨ RAM, ROM, Cache
- Prozessor
  - ⇨ Integer
  - ⇨ Gleitkommaarithmetik
  - ⇨ Cachecontroller
- E/A
  - ⇨ Tastatur
  - ⇨ Grafikkarte
  - ⇨ Diskettencontroller
  - ⇨ Festplattencontroller
  - ⇨ Netzwerkkarte

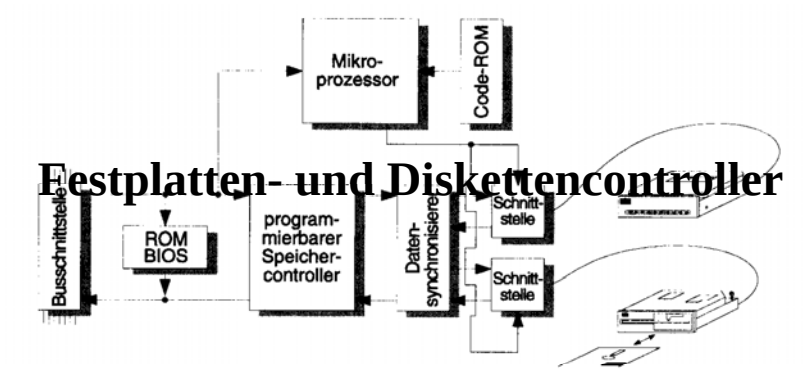


Hauptkomponenten der Zentraleinheit

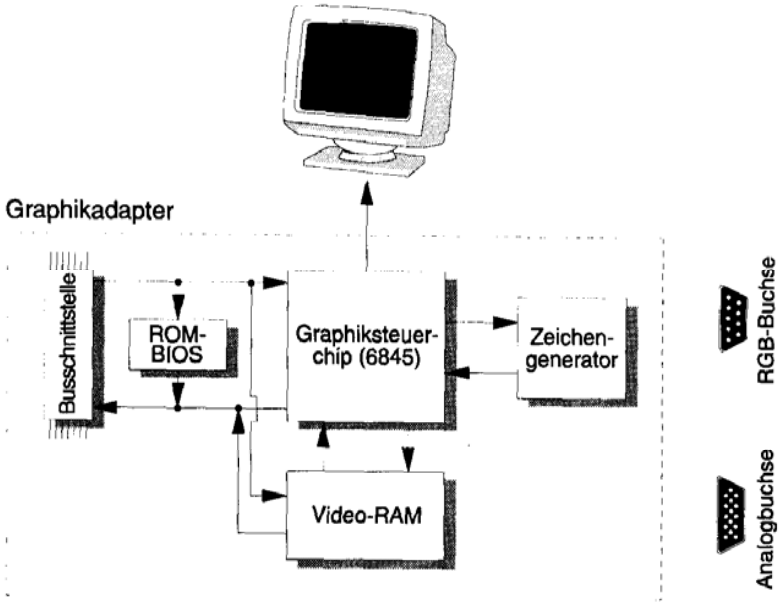
- Speicher
  - ⇨ RAM
  - ⇨ ROM
- Prozessor
  - ⇨ Integer-CPU
  - ⇨ Gleitkomma-Prozessor
  - ⇨ Cache
  - ⇨ Cachecontroller
- Bus
- Peripherie
  - ⇨ Schnittstellen
  - ⇨ Timer
  - ⇨ DMA



Peripherie



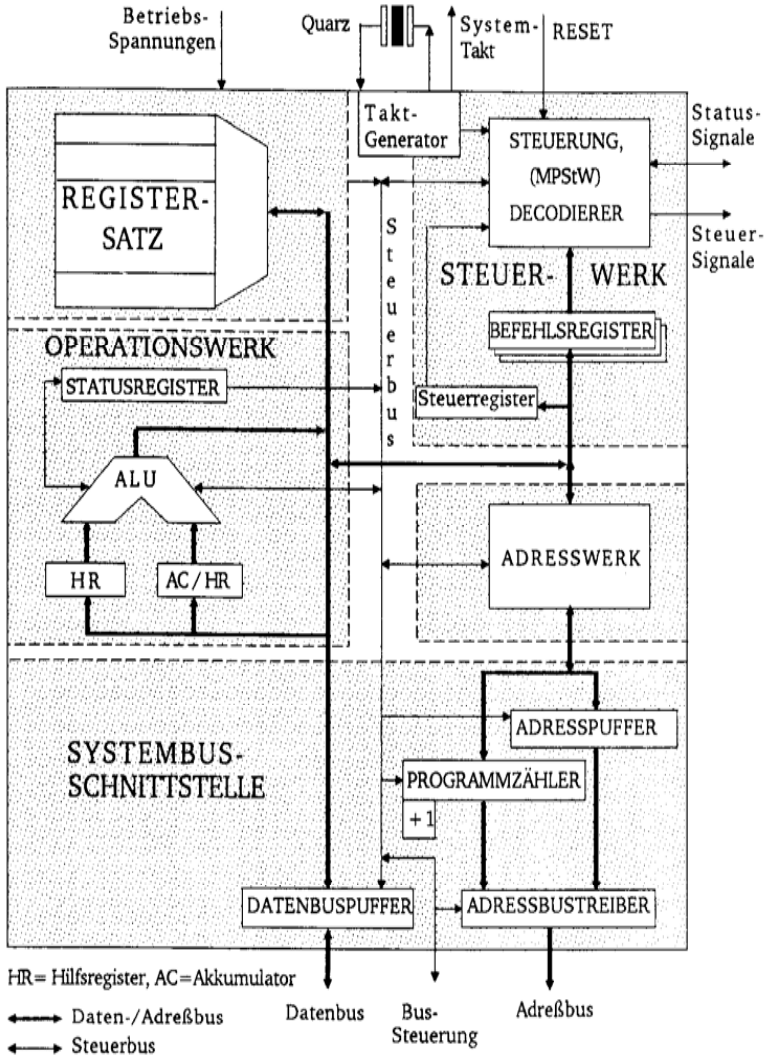
Festplatten- und Diskettencontroller



Grafikadapter

Prinzipieller Aufbau eines typischen Mikroprozessors

- **Steuerwerk**
  - ⇨ **Liefert die Steuersignale für das Rechenwerk**
  - ⇨ **Steuert den Ablauf der Operationen**
- **Rechenwerk (Operationswerk)**
  - ⇨ **führt die arithmetischen und logischen Operationen aus**
- **Registersatz**
  - ⇨ **speichert die Operanden für das Rechenwerk**
- **Adresswerk**
  - ⇨ **Berechnet die Adressen für die Befehle oder die Operanden**
- **Systembus-Schnittstelle**
  - ⇨ **Treiber**
  - ⇨ **Zwischenspeicher**
  - ⇨ **Adresszähler**

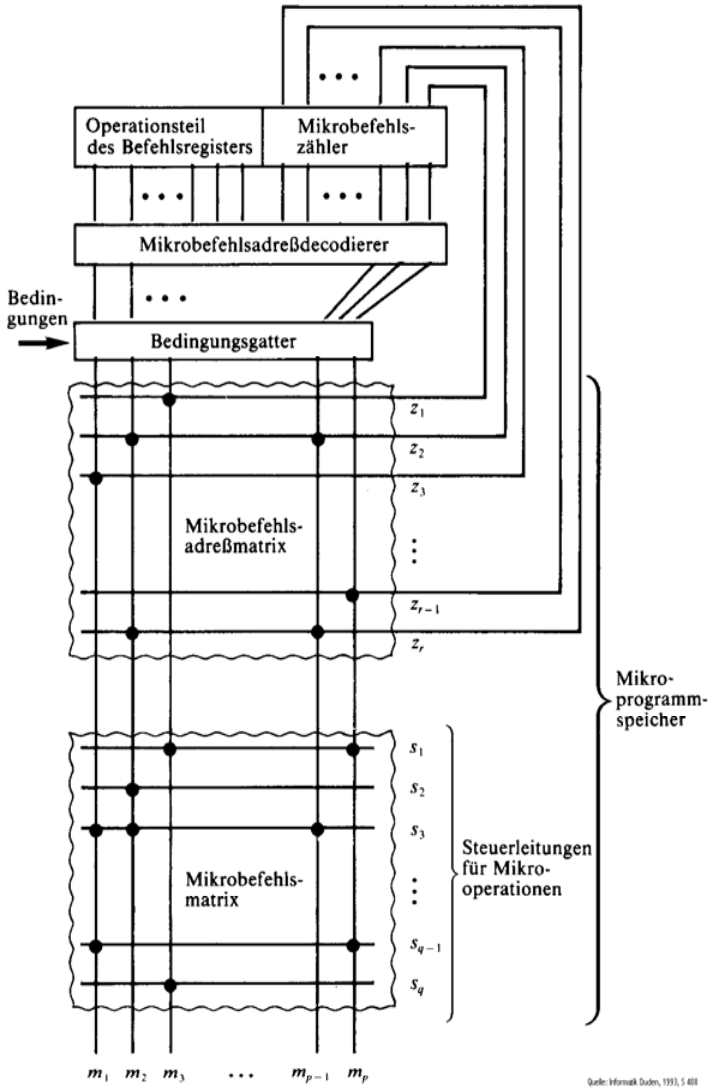


Das Steuerwerk

- Ablaufsteuerung der Befehlsbearbeitung im Operationswerk
- Synchrones Schaltwerk
- Komponenten eines typischen Steuerwerks
  - ⇨ Befehlsdekodierer: analysiert und entschlüsselt den aktuellen Befehl
  - ⇨ Steuerung: generiert die Signale für das Rechenwerk
  - ⇨ Befehlsregister: speichert den aktuellen Befehl
  - ⇨ Steuerregister: liefert Bedingungen zur Entscheidung des Befehlsablaufs
- Festverdrahtetes Steuerwerk
  - ⇨ das Steuerwerk wird als System mehrstufiger logischer Gleichungen implementiert und minimiert
- Mikroprogrammiertes Steuerwerk
  - ⇨ das Steuerwerk wird in einem ROM implementiert
- Mikroprogrammierbares Steuerwerk
  - ⇨ das Steuerwerk wird in einem RAM implementiert und wird beim Neustart des Prozessors geladen

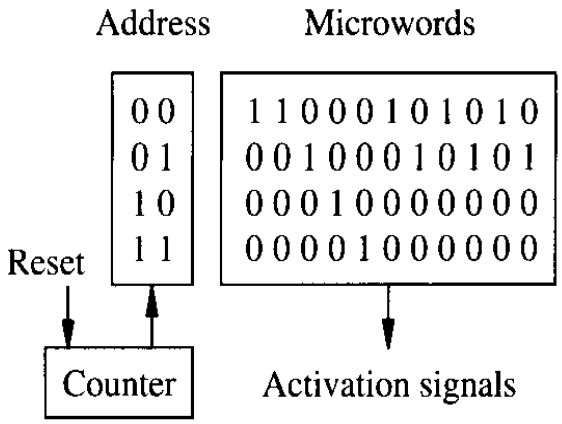
Mikroprogrammierung

- **Mikrooperationen**
  - ⇨ **elementare Operationen wie das Setzen eines Registers**
- **Mikrobefehle**
  - ⇨ **Zusammenfassung bestimmter Mikrooperationen, die zu einem Taktzeitpunkt gleichzeitig ausgeführt werden können**
- **Mikroprogrammierung**
  - ⇨ **Realisierung der Maschinenbefehle durch durch eine Folge von Elementaroperationen**



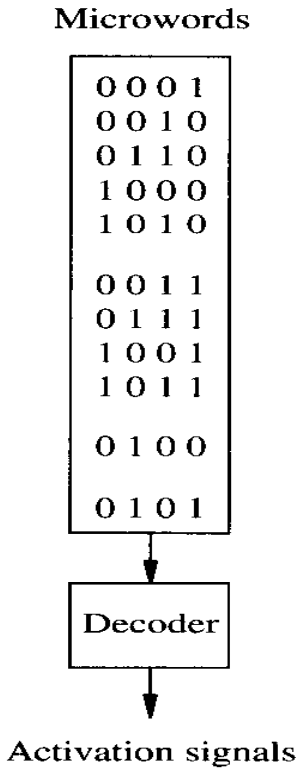
Vertikale und horizontale Mikroprogrammierung

- **Horizontale Mikroprogrammierung**
  - ⇨ **jedes Ausgangssignal erhält eine eigene Steuerleitung**



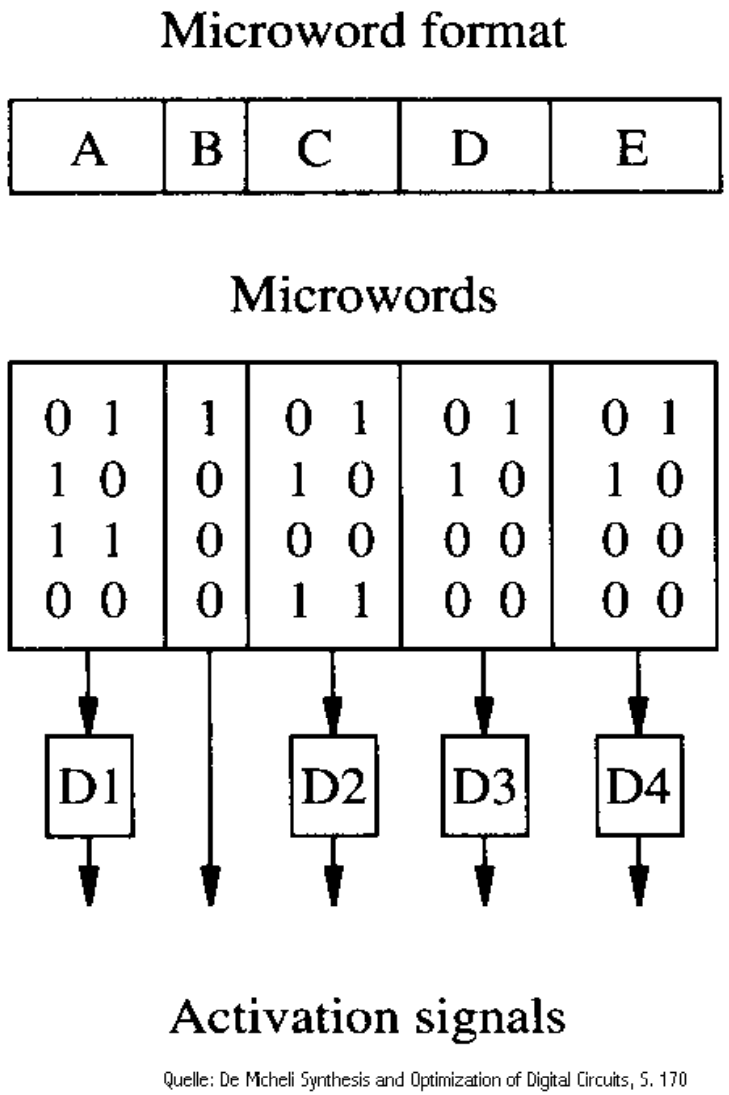
Quelle: De Micheli Synthesis and Optimization of Digital Circuits, S. 169

- **Vertikale Mikroprogrammierung**
  - ⇨ **Die Ausgangssignale werden über einen Multiplexer angesteuert**



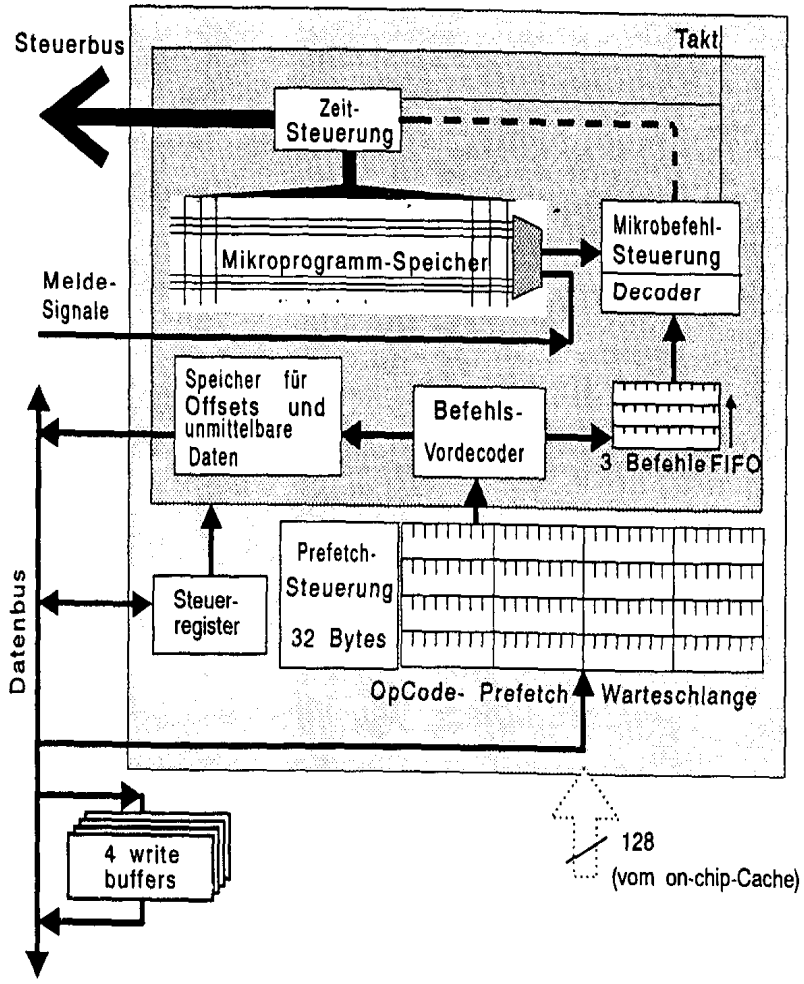
Mischformen

- **Unabhängige Teile des horizontalen Mikrobefehlsworts werden zusammengefaßt und vertikal kodiert**



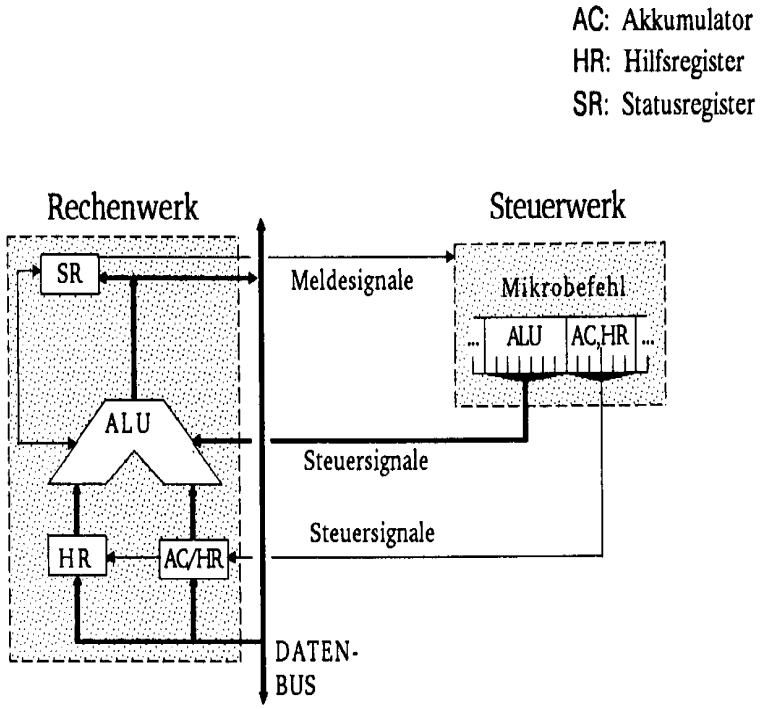
Quelle: De Micheli Synthesis and Optimization of Digital Circuits, S. 170

Das Steuerwerk des Intel 486



Das Rechenwerk

- **ALU**
  - ⇨ **berechnet alle Operationen**
- **Akkumulator**
  - ⇨ **speichert das Ergebnis einer Operation**
  - ⇨ **stellt einen Operanden zur Verfügung**
- **Hilfsregister**
  - ⇨ **stellt den zweiten Operanden zur Verfügung**
- **Statusregister**
  - ⇨ **Speichert besondere Ergebnisse**



Das Statusregister

- **Einzelne logisch unabhängige Bits**
  - **CF (Carry Flag)** — **Übertrag**
  - **ZF (Zero Flag)** — **Ergebnis der letzten Operation ist 0**
  - **SF (Sign Flag)** — **negatives Ergebnis bei der letzten Operation**
  - **OF (Overflow Flag)** — **Überlauf bei der letzten Operation**
  - **EF (Even Flag)** — **Gerades Ergebnis bei der letzten Operation**
  - **PF (Parity Flag)** — **ungerade Anzahl der ´1´-Bits**
- **Einsatz: Diese Flags werden bei bedingten Sprüngen ausgewertet**

Transfer- und Ein-/Ausgabebefehle

Mnemonic	Bedeutung
LD	Laden eines Register (*load*)
LEA	Laden eines Registers mit der Adresse eines Operanden (*load effective address*)
ST	Speichern des Inhalts eines Registers (*store*)
MOVE	Übertragen eines Datums (in beliebiger Richtung)
EXC	Vertauschen der Inhalte zweier Register bzw. eines Registers und eines Speicherwortes (*exchange*)
TFR	Übertragen eines Registerinhalts in ein anderes Register (*transfer*)
PUSH	Ablegen des Inhalts eines oder mehrerer Register im Stack
PULL (POP)	Laden eines Registers bzw. mehrerer Register aus dem Stack
STcc	Speichern eines Registerinhaltes, falls die Bedingung cc (nach Tabelle 1.14-11) erfüllt ist

Mnemonic	Bedeutung
IN, READ	Laden eines Registers aus einem Peripheriebaustein
OUT, WRITE	Übertragen eines Registerinhalts in einen Peripheriebaustein

Arithmetische und Logische Befehle

Mnemonic	Bedeutung
ABS	Absolutbetrag bilden (*absolute*)
ADD	Addition ohne Berücksichtigung des Übertrags (*add*)
ADC	Addition mit Berücksichtigung des Übertrags (*add with carry*)
CLR	Löschen eines Registers oder Speicherwortes (*clear*)
CMP	Vergleich zweier Operanden (*compare*)
COM	bitweises Invertieren eines Operanden (Einerkomplement) (*complement*)
DAA	Umwandlung eines dualen Operanden in eine Dezimalzahl (*decimal adjust A*)
DEC	Register oder Speicherwort dekrementieren (*decrement*)
DIV	Division (*divide*)
INC	Register oder Speicherwort inkrementieren (*increment*)
MUL	Multiplikation (*multiply*)
NEG	Vorzeichenwechsel im Zweierkomplement (*negate*)
SUB	Subtraktion ohne Berücksichtigung des Übertrags (*subtract*)
SBC	Subtraktion mit Berücksichtigung des Übertrags (*subtract with carry*)

Mnemonic	Bedeutung
AND	UND-Verknüpfung zweier Operanden
OR	ODER-Verknüpfung zweier Operanden
EOR	Antivalenz-Verknüpfung zweier Operanden (*exclusive or*)
NOT	Invertierung eines (Booleschen) Operanden

Flag- und Bit-Manipulationsbefehle

Mnemonic	Bedeutung	
SE<f>	Setzen eines Bedingungs-Flags	(*set*)
CL<f>	Löschen eines Bedingungs-Flags	(*clear*)
BSET	Setzen eines Bits	(*bit set*)
BCLR	Rücksetzen eines Bits	(*bit clear*)
BCHG	Invertieren eines Bits	(*bit change*)
TST	Prüfen eines bestimmten Flags oder Bits	(*test*)
BF...	Bitfeld-Befehle, insbesondere:	
BFCLR	Zurücksetzen der Bits auf '0'	(*clear*)
BFSET	Setzen der Bits auf '1'	(*set*)
BFFFO	Finden der ersten '1' in einem Bitfeld	(*find first one*)
BFEXT	Lesen eines Bitfeldes	(*extract*)
BFINS	Einfügen eines Bitfeldes	(*insert*)

(<f> Abkürzung für ein Flag, z.B. C *carry flag*)

Schiebe- und Rotationsbefehle

Mnemonic	Bedeutung	
SHF	Verschieben eines Registerinhaltes	(*shift*)
	insbesondere:	
ASL	arithmetische Links-Verschiebung	(*arithm. shift left*)
ASR	arithmetische Rechts-Verschiebung	(*arithm. shift right*)
LSL	logische Links-Verschiebung	(*logical shift left*)
LSR	logische Rechts-Verschiebung	(*logical shift right*)
ROT	Rotation eines Registerinhaltes	(*rotate*)
	insbesondere:	
ROL	Rotation nach links	(*rotate left*)
RCL	Rotation nach links durchs Übertragsbit	(*rotate with carry left*)
ROR	Rotation nach rechts	(*rotate right*)
RCR	Rotation nach rechts durchs Übertragsbit	(*rotate with carry right*)
SWAP	Vertauschen der beiden Hälften eines Registers	

Befehle zur Programmsteuerung

Sprung und Verzweigungsbefehle

Mnemonic	Bedeutung
JMP	unbedingter Sprung zu einer Adresse (*jump*)
Bcc	Verzweigen, falls die Bedingung cc erfüllt ist (*branch*)
BRA	Verzweigen ohne Abfrage einer Bedingung (*branch always*)

Unterprogrammaufrufe und Rücksprünge, Software-Interrupts

Mnemonic	Bedeutung
JSR, CALL	unbedingter Sprung in ein Unterprogramm (*jump to subroutine*)
BSRcc	Verzweigung in ein Unterprogramm, falls die Bedingung cc gilt (*branch to subroutine*)
RTS	Rücksprung aus einem Unterprogramm (*return from subroutine*)
SWI,TRAP,INT	Unterbrechungsanforderung durch Software (*software interrupt*)
RTI, RTE	Rücksprung aus einer Unterbrechungsroutine (*return from interrupt/exception*)

Bedingungen für Sprünge

cc	Bedingung	Bezeichnung
CS	CF = 1	*branch on carry set*
CC	CF = 0	*branch on carry clear* ,
VS	OF = 1	*branch on overflow*
VC	OF = 0	*branch on not overflow*
EQ	ZF = 1	*branch on zero/equal*
NE	ZF = 0	*branch on not zero/equal*
MI	SF = 1	*branch on minus*
PL	SF = 0	*branch on plus*
PA	PF = 1	*branch on parity/parity even*
NP	PF = 0	*branch on not parity/parity odd*
nicht vorzeichenbehaftete Operanden		
LO	CF = 1 (vgl. CS)	*branch on lower than*
LS	CF v ZF = 1	*branch on lower or same*
HI	CF v ZF = 0	*branch on higher than*
HS	CF = 0 (vgl. CC)	*branch on higher or same*
vorzeichenbehaftete Operanden		
LT	SF≠OF = 1	*branch on less than*
LE	ZF v (SF≠OF) = 1	*branch on less or equal*
GT	ZF v (SF≠OF) = 0	*branch on greater than*
GE	SF≠OF = 0	*branch on greater or equal*

(Bezeichnungen: ≠ Antivalenz, v logisches ODER)

Sonstige Befehle

Systembefehle

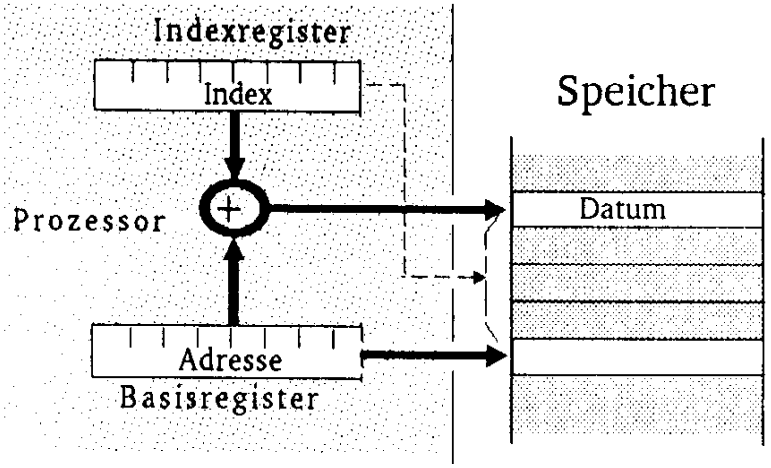
Mnemonic	Bedeutung
NOP	keine Operation, nächsten Befehl ansprechen (*no operation*)
WAIT	Warten, bis ein Signal an einem speziellen Eingang auftritt
SYNC	Warten auf einen Interrupt
HALT, STOP	Anhalten des Prozessors, Beenden jeder Programmausführung
RESET	Ausgabe eines Rücksetz-Signals für die Peripherie-Bausteine
SVC	(geschützter) Aufruf des Betriebssystem-Kerns (*supervisor call*)

Stringbefehle

Mnemonic	Bedeutung
MOVS	Transferieren eines Blocks (*move string*)
INS	Einlesen eines Blocks von der Peripherie (*input string*)
OUTS	Ausgabe eines Blocks an die Peripherie (*output string*)
CMPS	Vergleich zweier Blöcke (*compare string*)
COPS	Kopieren eines Blocks (*copy string*)
SCAS	Suchen eines Zeichens (Wortes) in einem Block (*scan string*)

Der Registersatz

- Datenregister
  - ⇨ Integerregister
  - ⇨ Akkumulator
- Adressregister
  - ⇨ Basisregister
  - ⇨ Indexregister
- Spezialregister
  - ⇨ Statusregister
  - ⇨ Programmzähler
  - ⇨ Stackpointer
  - ⇨ Segmentregister

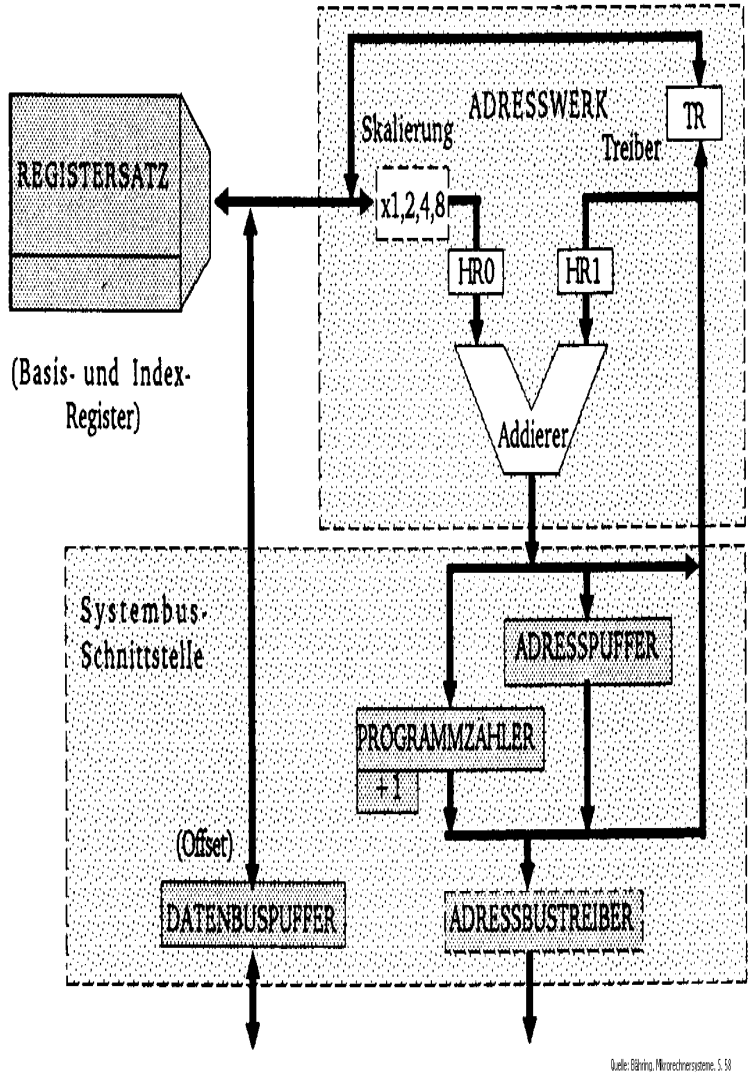


Die Register im Intel 80x86

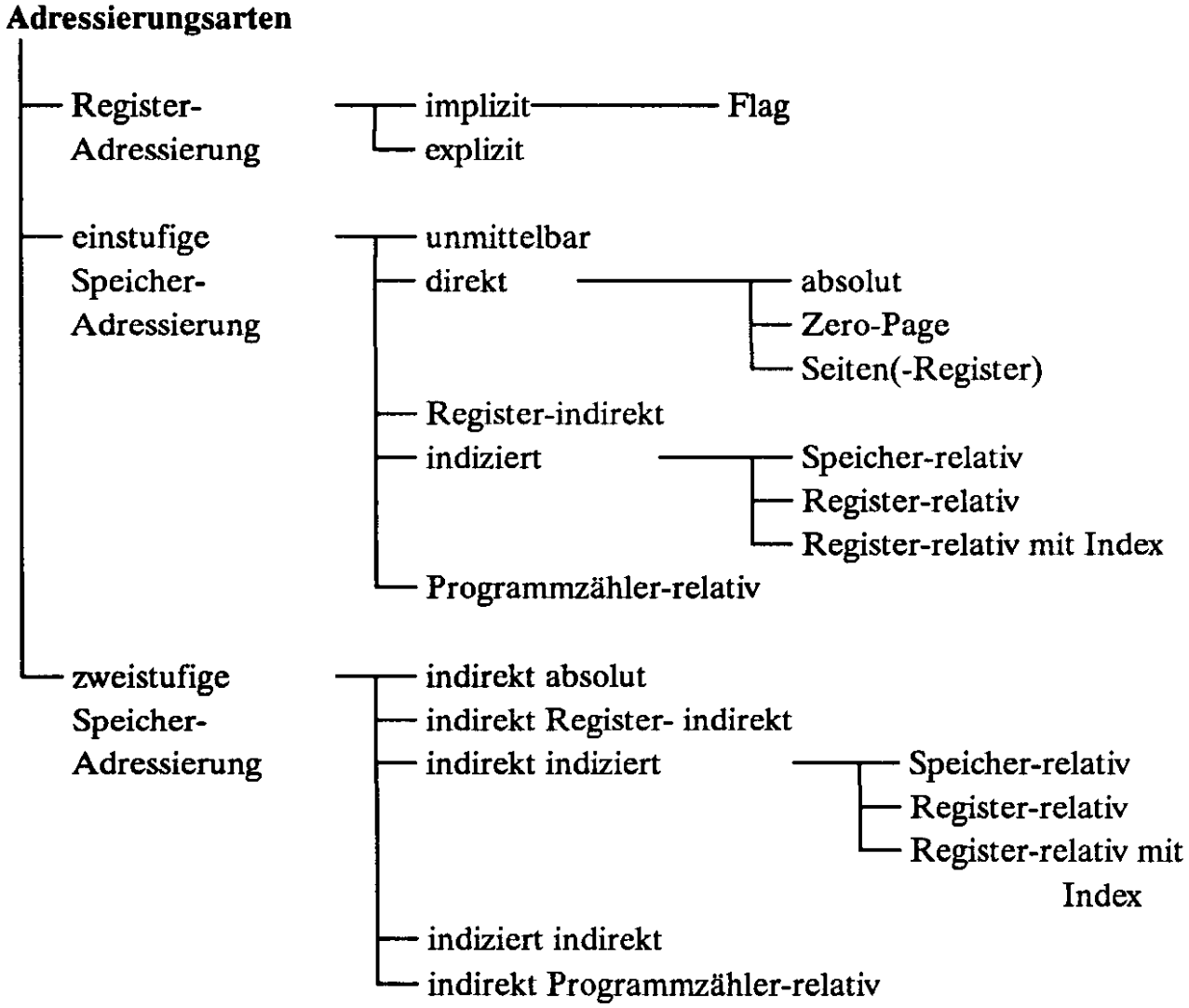
- AX (AH und AL)
  - accumlator
  - Akkumulator
- BX (BH und BL)
  - base register
  - Basisregister zur Adressierung der Anfangsadresse einer Datenstruktur
- CX (CH und CL)
  - count register
  - Schleifenzähler, wird bei Schleifen und Verschiebeoperationen benötigt
- DX
  - data register
  - Datenregister Register für den zweiten Operand
- SI und DI
  - source register und destination register
  - Indexregister für die Adressierung von Speicherbereichen
- SP
  - stack pointer
  - Verwaltung eines Stapelbereichs

Das Adresswerk

- Nach den Vorgaben des Steuerwerks werden Speicheradressen gebildet
  - ⇨ aus Registerinhalten
  - ⇨ aus Speicherzellen
- Adressaddierer
- TR-Register speichert den Inhalt des aktuellen Adresszählers bei Sprüngen
- Adressprüfung bei Byte-, Halbwort-, Doppelwort- und Quadwort-Zugriffen

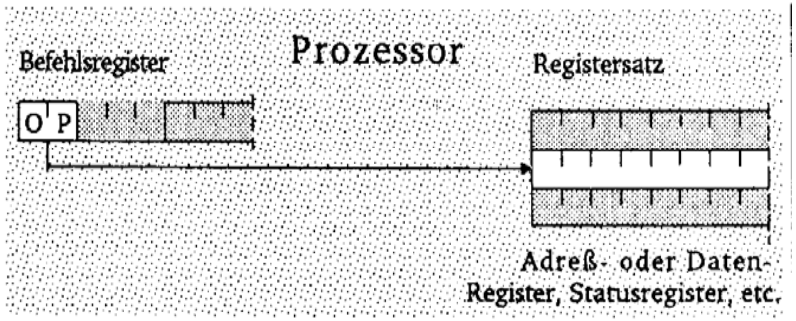


Adressierungsarten

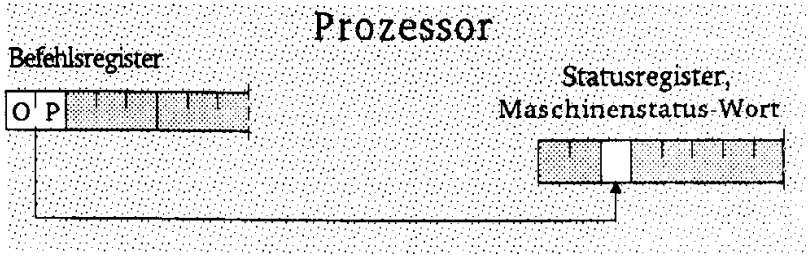


Register- Adressierung

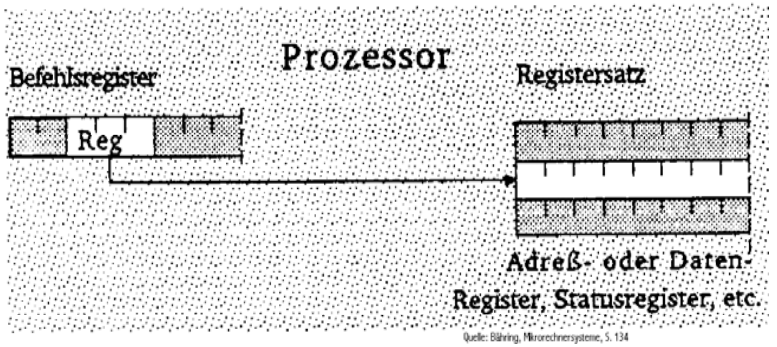
- **Implizite Adressierung**
  - ⇨ **Adresse des Operands ist im OP-Code enthalten**
  - ⇨ **Beispiel: LSRA**
    - **logical shift right accumulator**
- **Flag-Adressierung**
  - ⇨ **ein einzelnes Bit wird angesprochen**
  - ⇨ **Beispiel: SEC**
    - **set carry flag**
- **Explizite Adressierung**
  - ⇨ **Adresse des Operandenregisters wird im OP-Code angegeben**
  - ⇨ **Beispiel: DEC r0**
    - **decrement R0**



Quelle: Bähring, Mikrorechnersysteme, S. 130



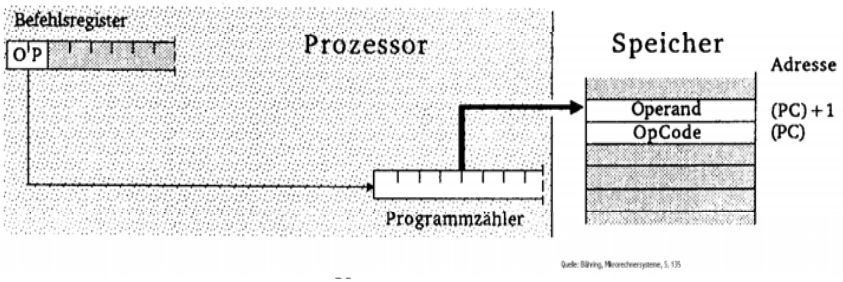
Quelle: Bähring, Mikrorechnersysteme, S. 133



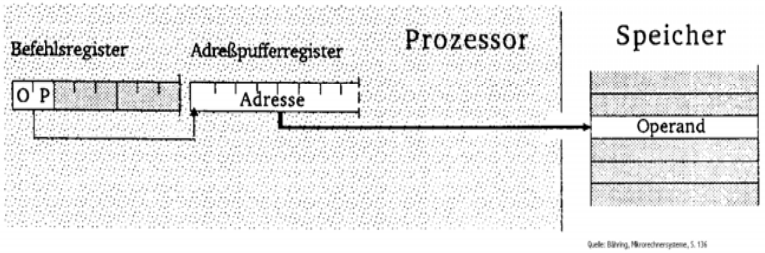
Quelle: Bähring, Mikrorechnersysteme, S. 134

Einstufige Adressierung

- **Unmittelbare Adressierung**
  - ⇨ **Der Befehl enthält den Operanden**
  - ⇨ **Beispiel: LDA #$A3**
    - **load accu $A3_{16}$**

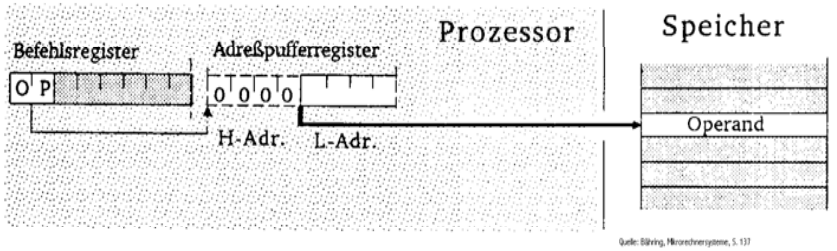


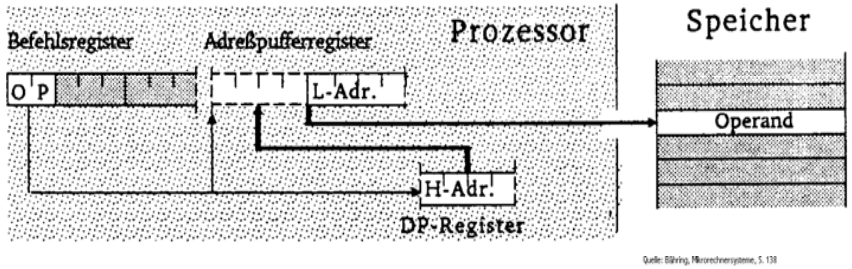
- **Absolute Adressierung**
  - ⇨ **Das Speicherwort das dem Befehl folgt enthält die Adresse**
  - ⇨ **Beispiel: JMP $07FE**



Seitenadressierung

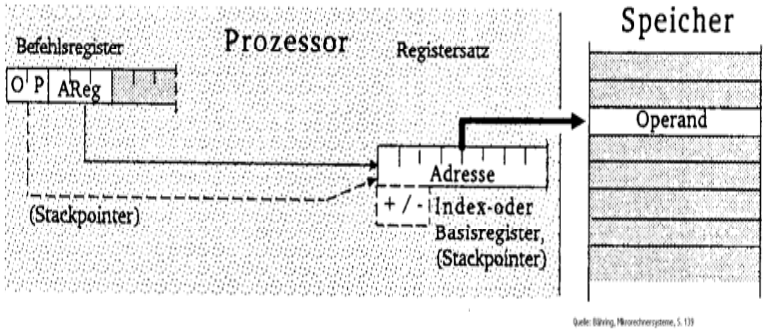
- Bei Prozessoren mit unterschiedlicher Daten- und Adressbusbreite
  - ⇨ man spart Speicherplatz und Zeit des Lesens der höherwertigen Bits
- Zero-Page Adressierung
  - ⇨ schneller Zugriff auf die Speicherseite 0
  - ⇨ Beispiel: INC $007F
    - erhöhe Speicherzelle $7F um 1
- Seiten-Register-Adressierung
  - ⇨ Höherwertige Adressteil wird von einem Register zur Verfügung gestellt
  - ⇨ Beispiel: LDA R0, <$7F





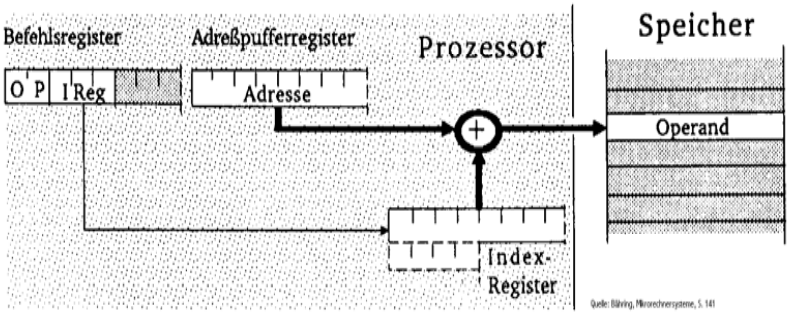
Register-Indirekte Adressierung

- Auch Zeigeradressierung
  - ⇨ Der Inhalt eines Registers wird als Adresse des Operanden verwendet
- postincrement: `LD R1,(R0)+`
  - ⇨ Lade R1 mit dem Inhalt der Speicherzelle, auf die R0 zeigt, und incrementiere anschließend R0
- preincrement: `INC +(R0)`
  - ⇨ Erhöhe zunächst das Register R0 um 1 und danach die Speicherzelle, auf die das neue R0 zeigt
- postdecrement: `LD R1,(R0)-`
  - ⇨ Lade R1 mit dem Inhalt der Speicherzelle, auf die R0 zeigt, und decrementiere anschließend R0
- predecrement: `CLR -(R0)`
  - ⇨ Dekrementiere zunächst R0 und lösche die Speicherzelle, auf die das neue R0 zeigt

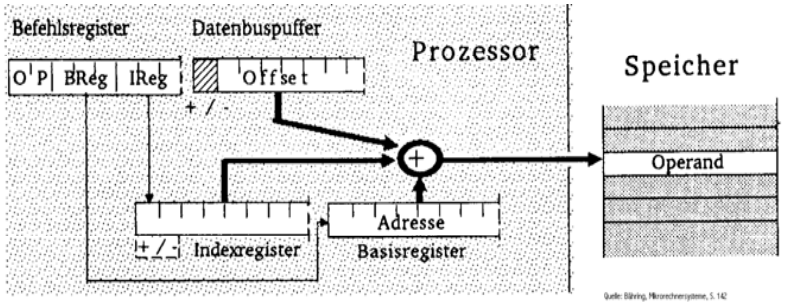


Indizierte Adressierung

- Speicher-relative Adressierung
  - ⇨ Der Basiswert, der zum Indexregister addiert wird, ist im Befehlswort enthalten
  - ⇨ Beispiel `ST R1, $A704(R0)`
    - Speichere R1 an die Adresse, die sich aus der Summe des Inhalts des Registers R0 und $A704 ergibt

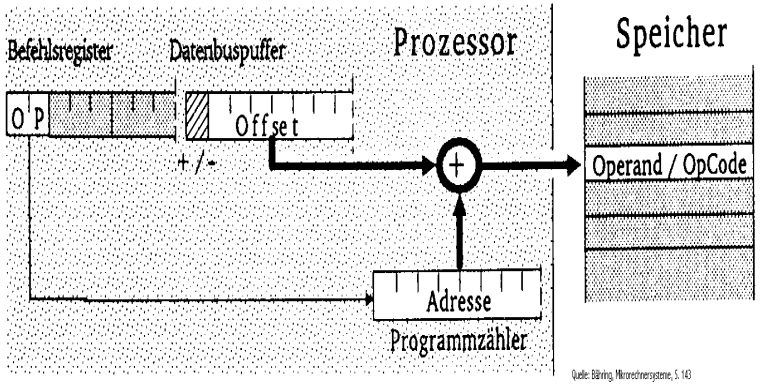


- Register-relative Adressierung mit Offset
  - ⇨ Der Basiswert befindet sich in einem speziellen Basisregister
  - ⇨ Der Inhalt des Indexregister und ein Offset wird zum Basiswert addiert
  - ⇨ autoincrement und autodecrement
  - ⇨ Beispiel: `ST R1, $A7(B0)(I0)+`
    - Speichere R1 an die Adresse die sich durch Addition von B0, I0 und dem Offset ergibt und incrementiere I0 anschließend



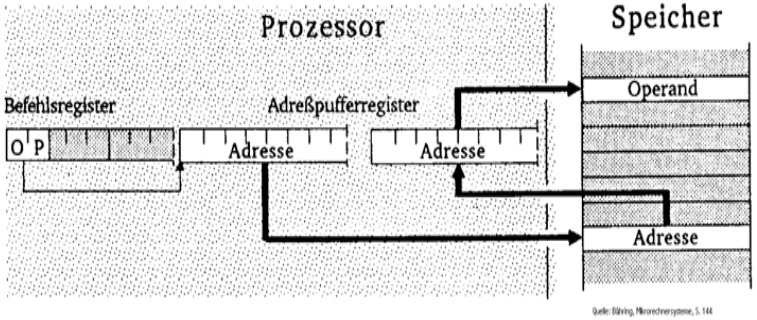
Programmzähler-relative Adressierung

- Der im Befehlscode angegebene Offset wird zum aktuellen Befehlszähler hinzuaddiert
- Beispiel: `BCS $47(PC)`
  - ⇨ Verzweige an die Adresse PC+$47 sofern das Carry-Flag gesetzt ist

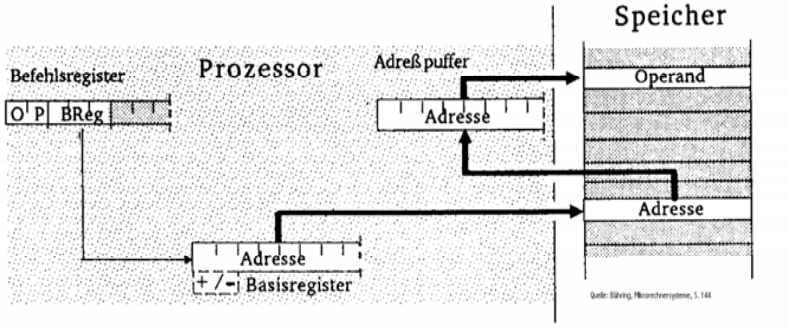


Zweistufige Speicheradressierung

- **Indirekte absolute Adressierung**
  - ⇨ **Der Befehl enthält eine absolute Adresse, die auf ein Speicherwort zeigt. Dieses Speicherwort enthält die gesuchte Adresse**
  - ⇨ **Beispiel: `LDA ($A345)`**
    - **Lade den Accu mit dem Inhalt des Speicherworts, dessen Adresse in $A345 steht**

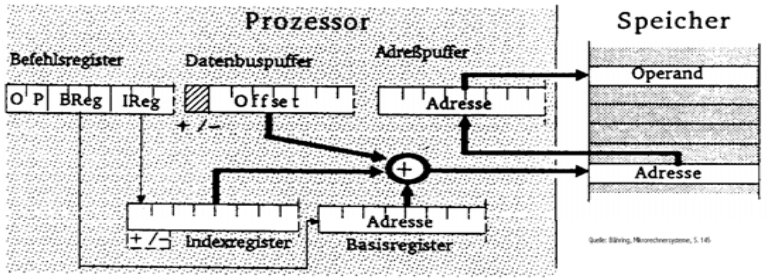


- **Indirekte Register-indirekte Adressierung**
  - ⇨ **Der Befehl bezeichnet ein Register, dessen Inhalt die Speicherzelle ist, deren Inhalt als Adresse für das Speicherwort verwendet wird**
  - ⇨ **Beispiel: LD R1, ((R0))**
    - **Lade R1 mit dem Inhalt der Adresse die in in der Speicherzelle steht, auf die R0 zeigt**

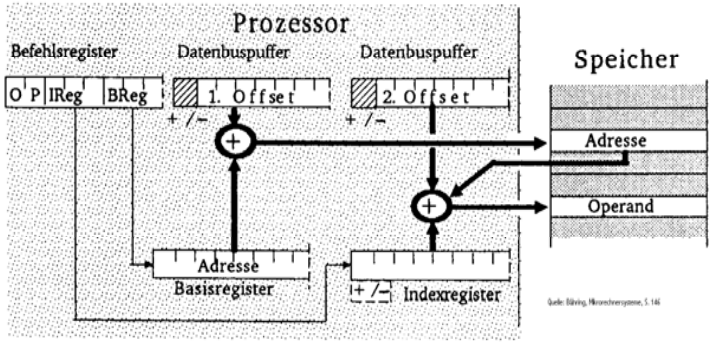


Zweistufige Speicheradressierung

- **Indirekte indizierte Adressierung**
  - ⇨ **Die Adresse des Speicherworts wird aus der Summe von Offset, Basisregister und Indexregister gebildet. Dieses Speicherwort enthält die Adresse des Ziels**
  - ⇨ **Beispiel: `INC ($A7(B0)(I0))`**
    - **Erhöhe die Speicherzelle mit der Adresse $A7+B0+I0 um 1**

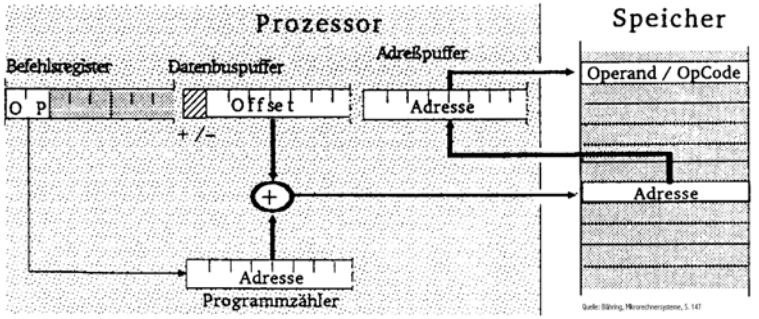


- **Indizierte indirekte Adressierung**
  - ⇨ **Die Adresse des Speicherworts wird aus dem 1. Offset und dem Basisregister gebildet. Der Inhalt dieses Speicherworts wird zum Indexregister und dem 2. Offset hinzuaddiert und bildet die Adresse des gesuchten Speicherworts**
  - ⇨ **Beispiel: `INC $A7($10(B0))(I2)`**
    - **Addiere den Offset $10 zum Inhalt des Basisregisters. Der Inhalt dieser Speicherzelle plus Indexregister und zweiter Offset $A7 ergibt den Wert der gesuchten Adresse**



Zweistufige Speicheradressierung

- **Indirekte Programmzähler-relative Adressierung**
  - ⇨ **Die Summe aus Programmzähler und Offset ergeben die Adresse, die auf das Ziel zeigt**
  - ⇨ **Beispiel: JMP ($A7(PC))**
    - **Springe an die Stelle die im Speicherwort mit der Adresse PC plus $A7 steht.**



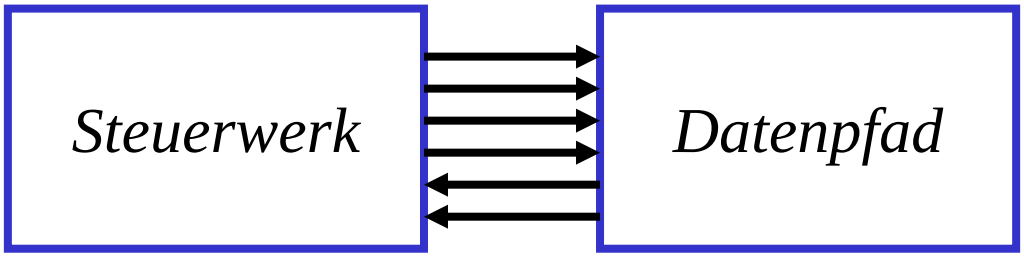
Prüfungstermin

- **"Grundlagen der Technischen Informatik 2„**
  - ⇨ **31.7.2008, 15.30 Uhr , Gr Hs CLI**
  - ⇨ **An- und Abmeldeende: 04.07.08**
  - ⇨ **Voraussetzung: Schein Hardware-Praktikum**

- **"Grundlagen der Technischen Informatik 1" (Wh.)**
  - ⇨ **31.07.08, 9.00 Uhr, KH 2-04**
  - ⇨ **Teilnahmeberechtigt und gleichzeitig verpflichtet sind nur Studenten, die die Klausur vom WS.07/.08 nicht bestanden haben oder entschuldigt gefehlt haben**

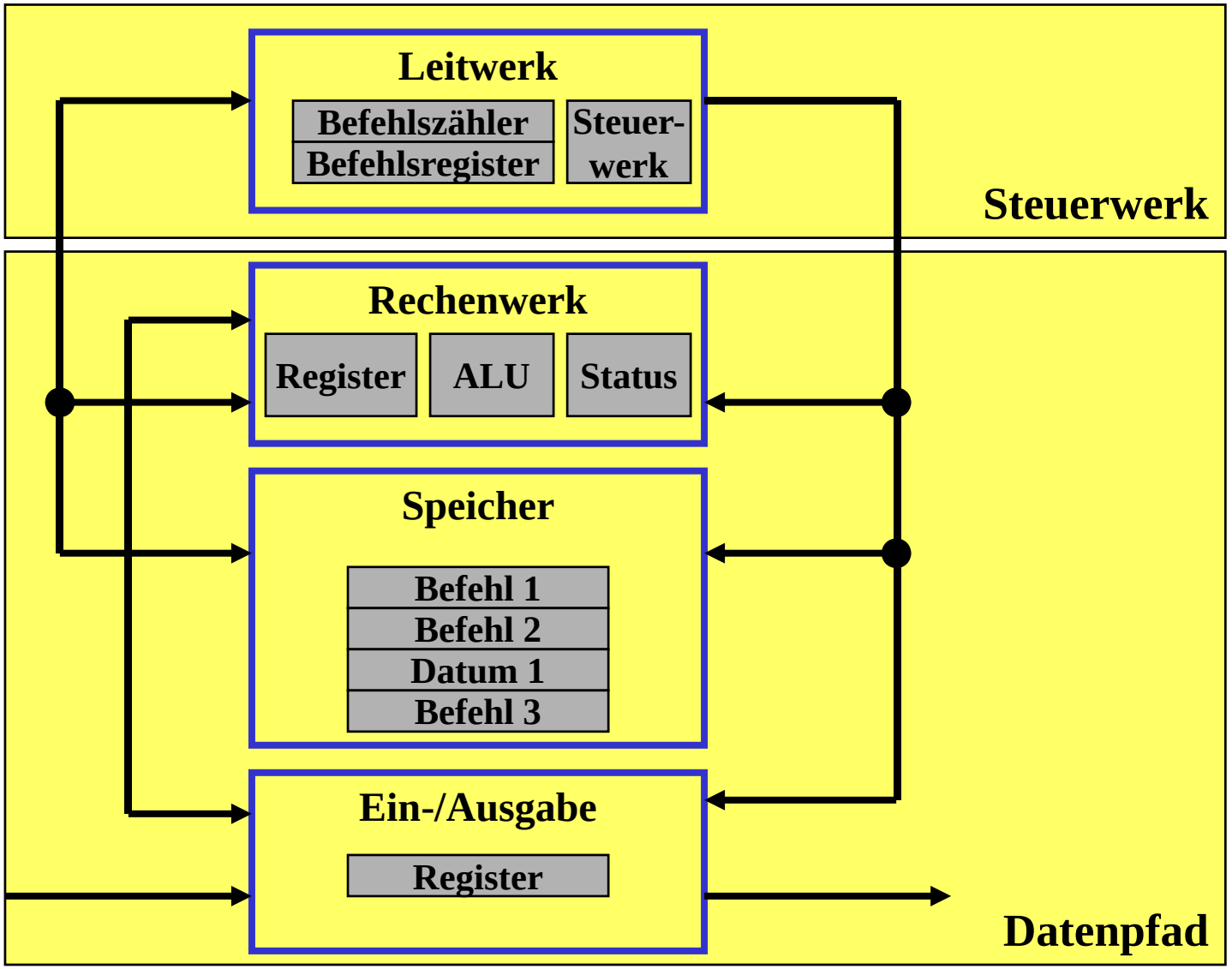
Ein einfacher Rechner

- **Von Neumann Architektur**
  - ⇨ **Ein-/Ausgabe**
  - ⇨ **Speicher**
  - ⇨ **Rechenwerk**
  - ⇨ **Leitwerk (Steuerwerk)**



- **Erweiterung von Steuerwerken**
  - ⇨ **Ein Steuerwerk bestimmt den Ablauf der Berechnung**
  - ⇨ **Der Datenpfad bestimmt den Fluss der Operanden und Ergebnisse**
  - ⇨ **Daten und Programm stehen in einem gemeinsamen Speicher**

Von Neumann Architektur



Befehlsablauf im von Neumann-Rechner

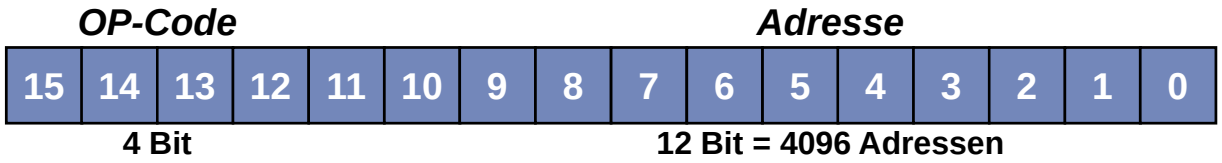
- **Lesen**
  - ⇨ **Einen neuen Programmzähler-Wert (PC) bestimmen**
  - ⇨ **Bestimmung der Speicheradresse des Quelloperanden**
  - ⇨ **Lesezugriff auf den Speicher**
  - ⇨ **Speichern des gelesenen Wertes im Zielregister**
- **Schreiben**
  - ⇨ **Einen neuen Programmzähler-Wert (PC) bestimmen**
  - ⇨ **Bestimmung der Speicheradresse des Zieloperanden**
  - ⇨ **Lesezugriff auf das Quellregister**
  - ⇨ **Schreibzugriff auf den Speicher**

Befehlsablauf im von Neumann-Rechner

- **Verknüpfung von Operanden**
  - ⇨ **Einen neuen Programmzähler-Wert (PC) bestimmen**
  - ⇨ **Auslesen der Operanden aus dem Registerblock**
  - ⇨ **Verknüpfung der Operanden in der ALU**
  - ⇨ **Schreiben des Ergebnisses in den Registerblock**
- **Verzweigungen und Sprünge**
  - ⇨ **Einen neuen Programmzähler-Wert (PC) bestimmen**
  - ⇨ **Berechnung der Adresse des Sprungziels**
  - ⇨ **Prüfung der Sprungbedingung (bei Verzweigungen)**
  - ⇨ **Überschreiben des Befehlszählers, wenn der Sprung ausgeführt werden soll**

Der Toy-Rechner: Allgemeine Informationen

- **Quelle: Phil Koopman, *Microcoded versus hard-wired control*, BYTE, Januar 1987, S. 235**
- **Spezifikation**
  - **einfacher aber vollständiger Mikrorechner**
  - **einfacher Aufbau mit Standardbausteinen**
  - **RISC-Rechner**
    - sehr einfacher Befehlssatz (12 Befehle)
    - alle Befehle in einem Takt (2 Phasen-Takt)
  - **Befehlsformat**

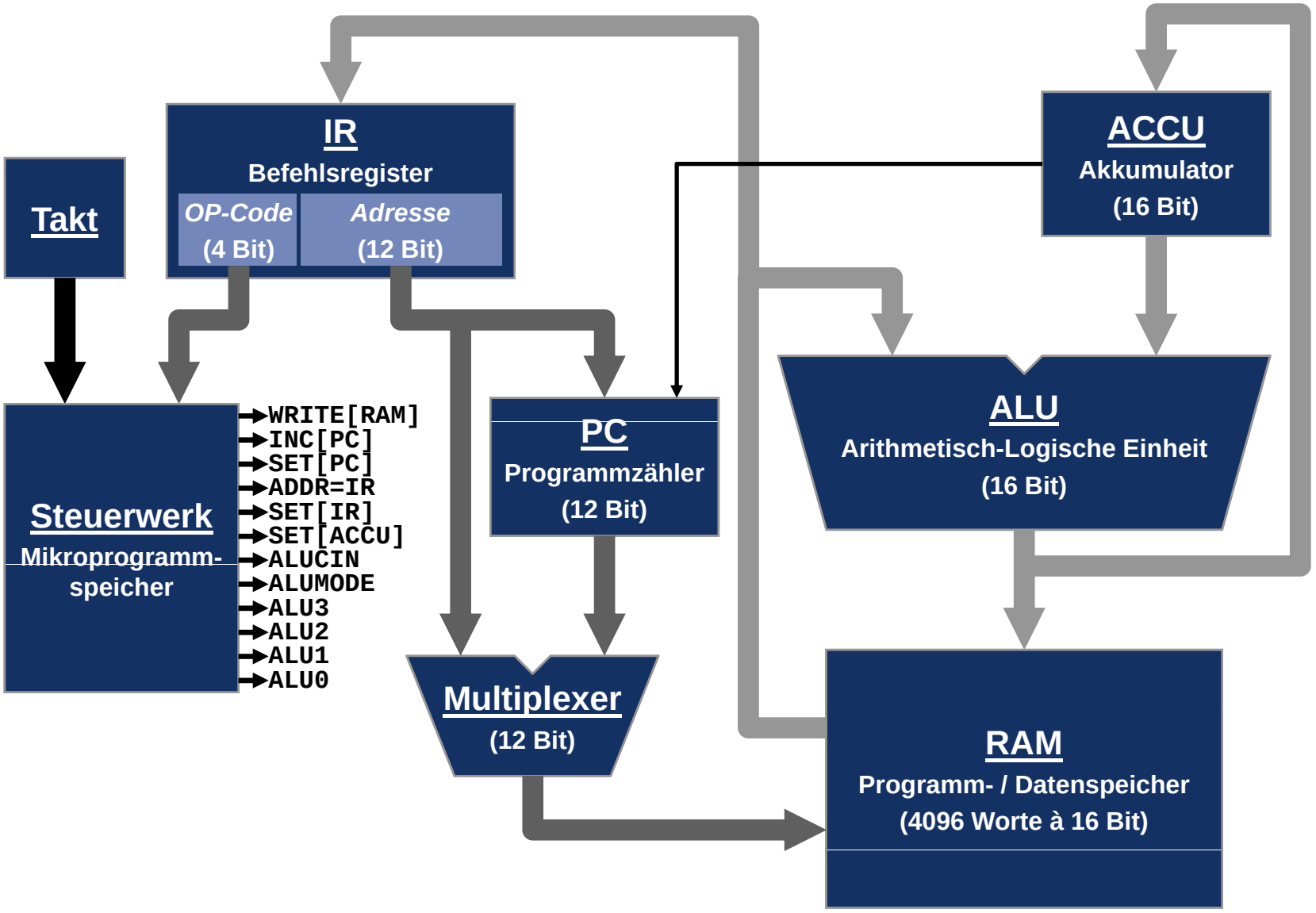


  - **1-Adress-Maschine**
    - Akkumulator liefert implizit immer einen der Operanden
    - Befehl spezifiziert zweiten Operanden (falls notwendig)
    - Ergebnis einer Operation wird immer in den Akkumulator geschrieben

Toy-Befehlssatz

	Mnemonic	Bedeutung
0	`STO <Adresse>`	speichere den Inhalt des ACCUs ins RAM
1	`LDA <Adresse>`	lade den ACCU mit dem Inhalt der Adresse
2	`BRZ <Adresse>`	springe nach Adresse, wenn der ACCU Null ist
3	`ADD <Adresse>`	addiere den Inhalt der Adresse zum ACCU
4	`SUB <Adresse>`	subtrahiere den Inhalt der Adresse vom ACCU
5	`OR  <Adresse>`	logisches ODER des ACCUs mit dem Inhalt der Adresse
6	`AND <Adresse>`	logisches UND des ACCUs mit dem Inhalt der Adresse
7	`XOR <Adresse>`	logisches ExODER des ACCUs mit dem Inhalt der Adresse
8	`NOT`	logisches NICHT der Bits im ACCU
9	`INC`	inkrementiere den ACCU
10	`DEC`	dekrementiere den ACCU
11	`ZRO`	setze den ACCU auf Null
12	`NOP`	keine Operation
13	`NOP`	keine Operation
14	`NOP`	keine Operation
15	`NOP`	keine Operation

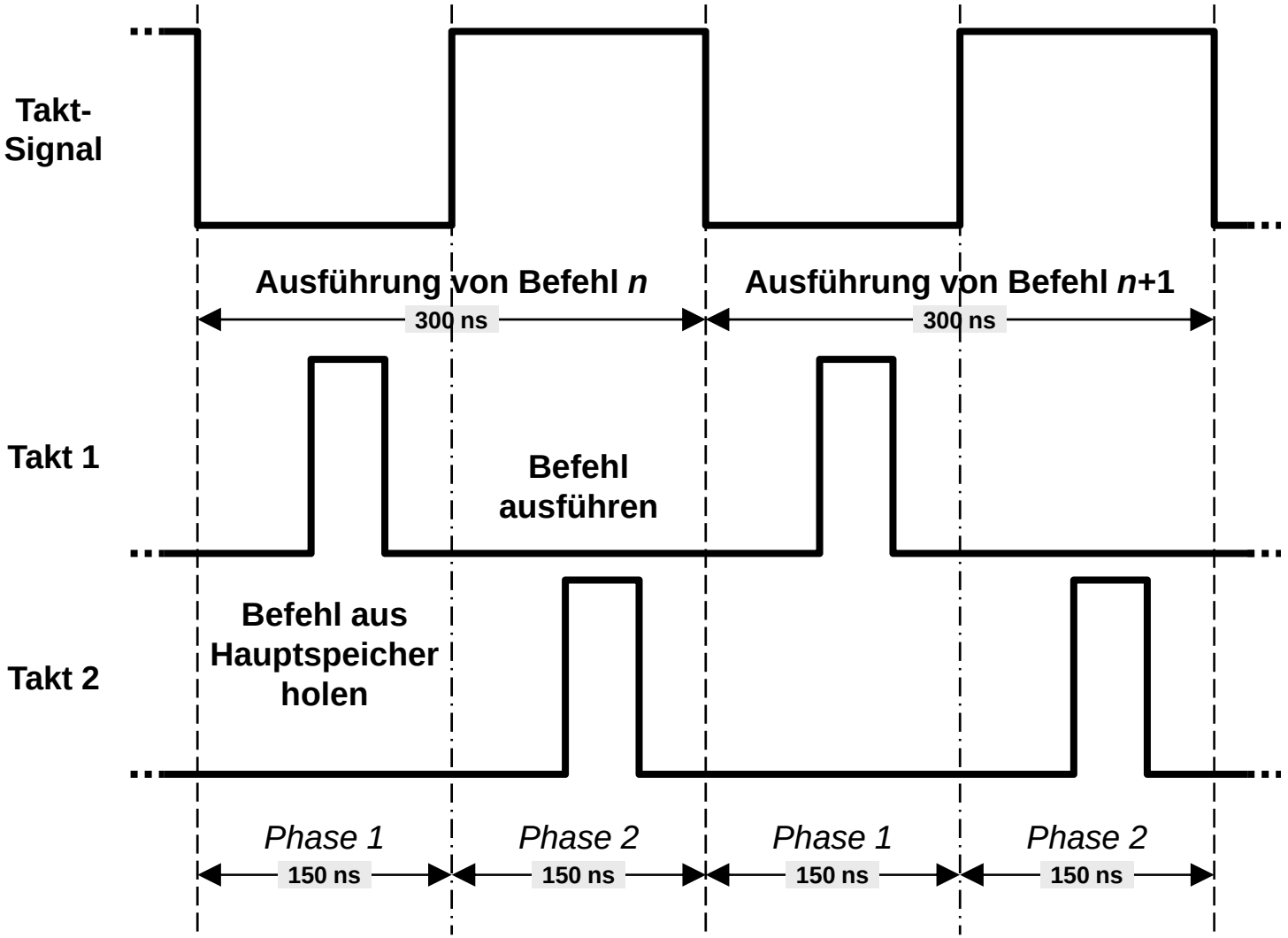
Blockschaltbild



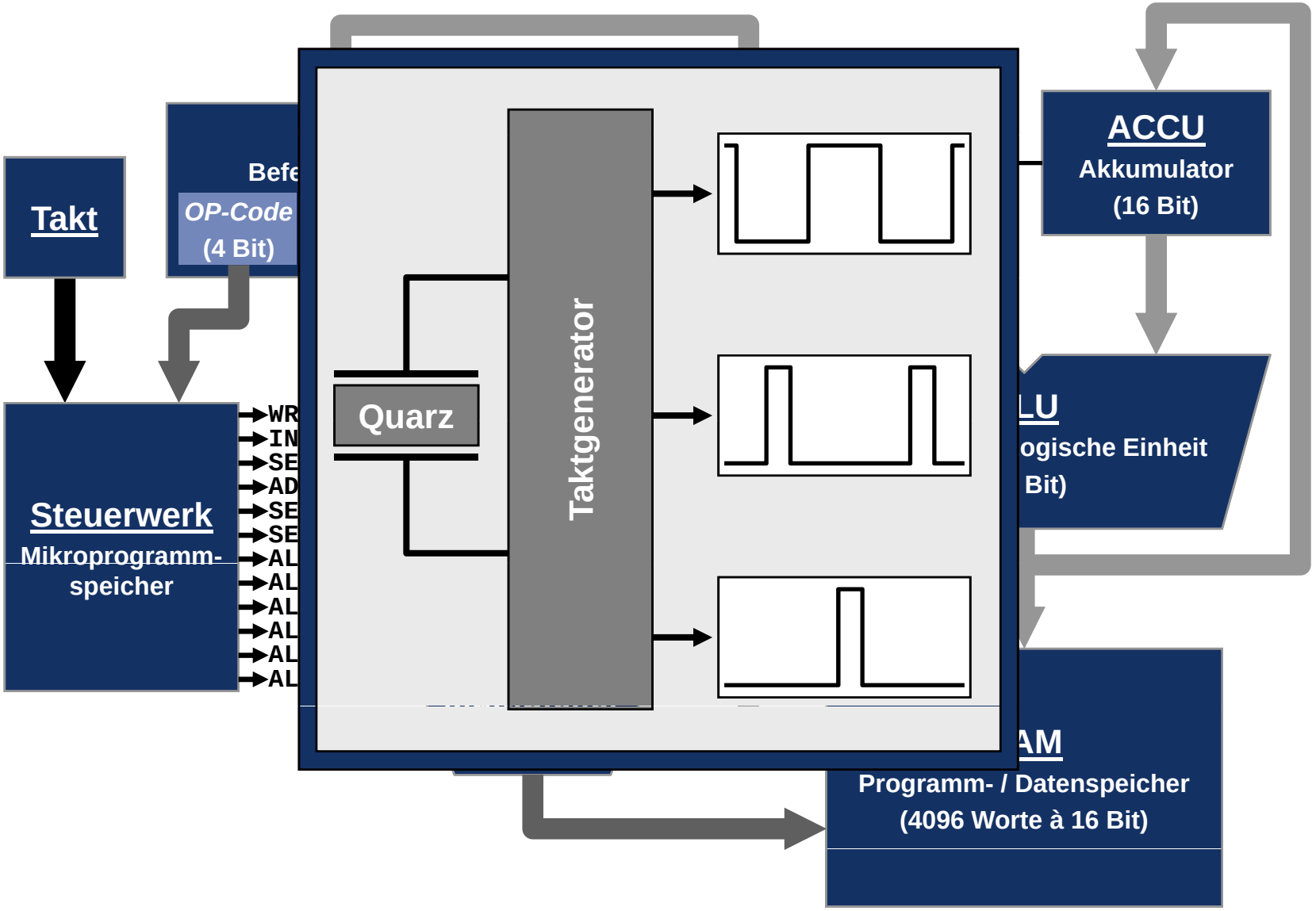
Steuerung im 2-Phasen-Takt

OP-Code	Operation	Phase	Steuerung
0	STO	1	ADDR=IR; ALU=ACCU; WRITE[RAM]
		2	ADDR=PC; SET[IR]; INC[PC]
1	LDA	1	ADDR=IR; ALU=RAM; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
2	BRZ	1	SET[PC]
		2	ADDR=PC; SET[IR]; INC[PC]
3	ADD	1	ADDR=IR; ALU=ACCU+RAM; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
4	SUB	1	ADDR=IR; ALU=ACCU–RAM; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
...			
8	NOT	1	ALU=~ACCU; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
9	INC	1	ALU=ACCU+1; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
...			
12 – 15	NOP	1	ADDR=PC; SET[IR]; INC[PC]
		2	–

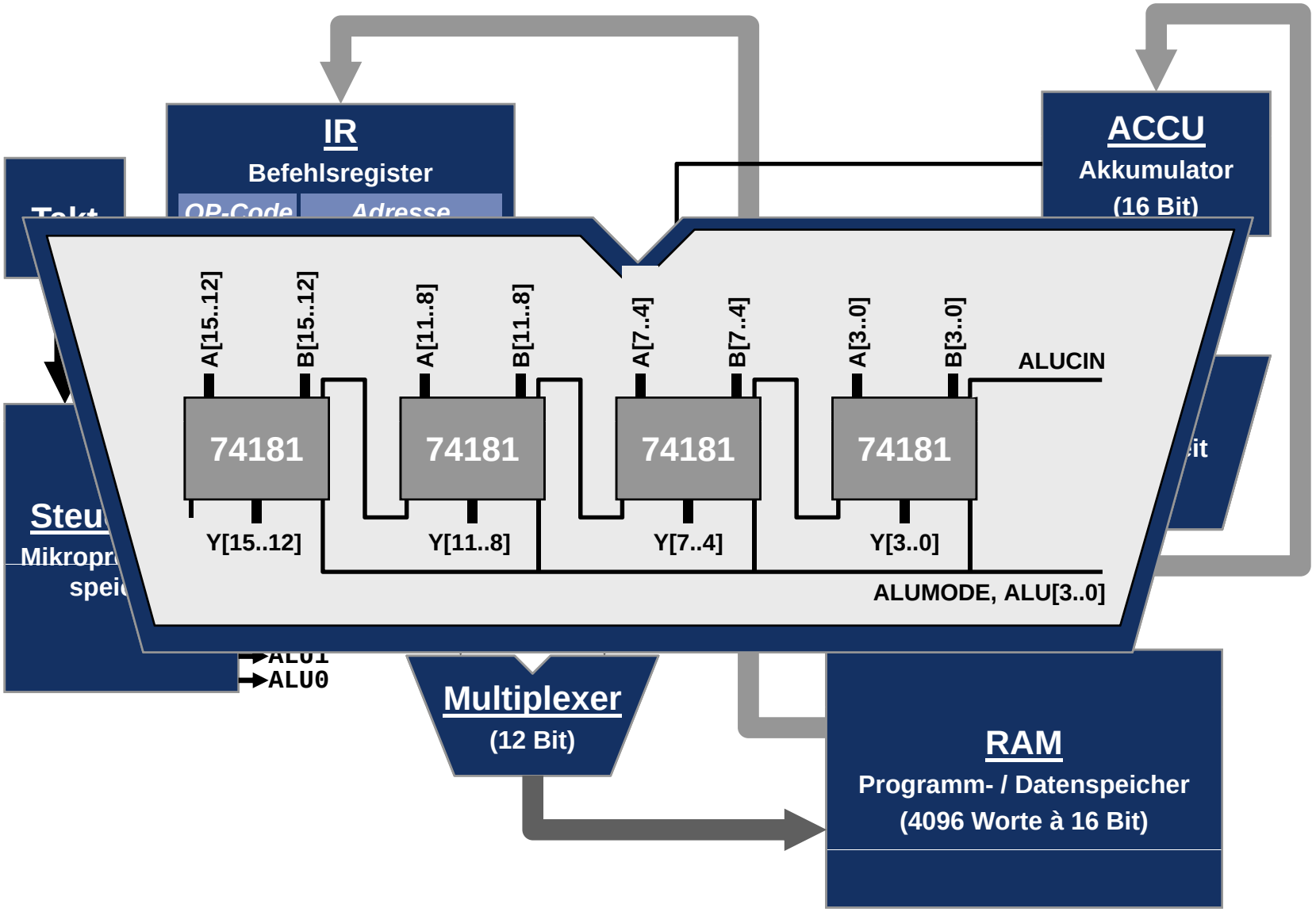
Ablaufsteuerung – 2-Phasen-Takt



Der Taktgenerator



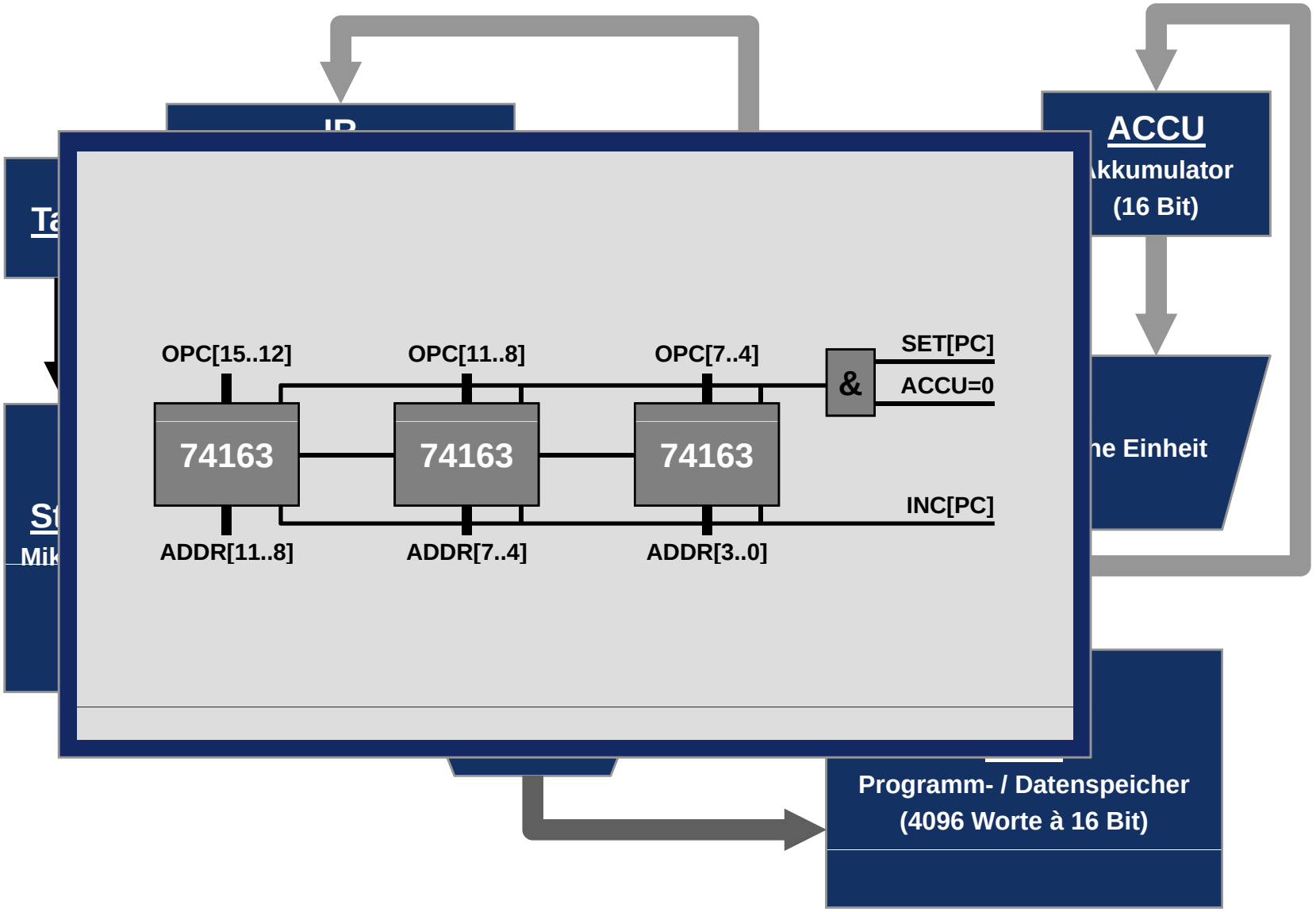
Die ALU

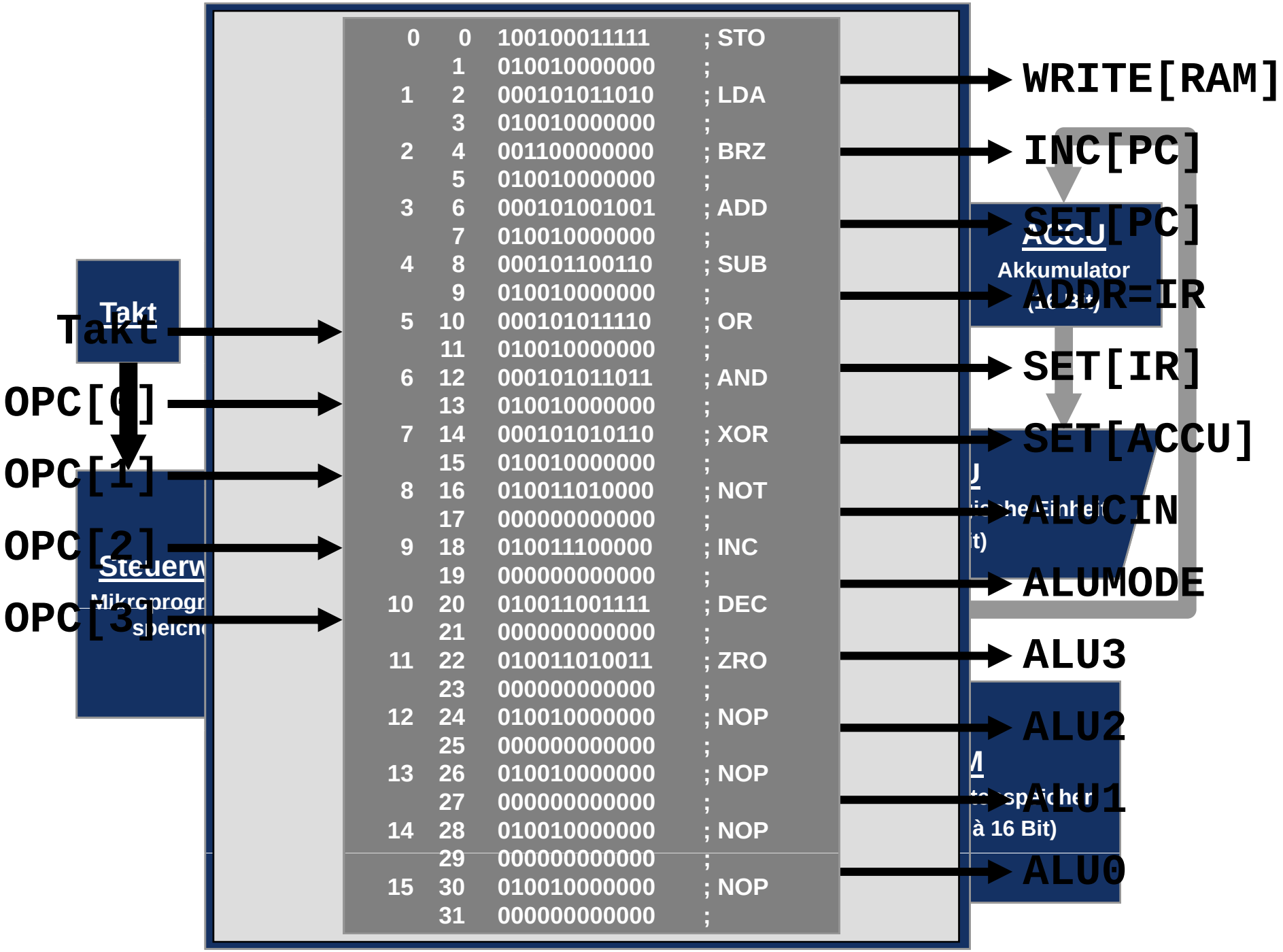

IR
Befehlsregister
OP-Code
Adresse
ACCU
Akkumulator
(16 Bit)
A[15..12]
B[15..12]
A[11..8]
B[11..8]
A[7..4]
B[7..4]
A[3..0]
B[3..0]
ALUCIN
74181
74181
74181
74181
Y[15..12]
Y[11..8]
Y[7..4]
Y[3..0]
ALUMODE, ALU[3..0]
Multiplexer
(12 Bit)
RAM
Programm- / Datenspeicher
(4096 Worte à 16 Bit)

Ansteuerung einer ALU vom Typ TTL-Series 74181

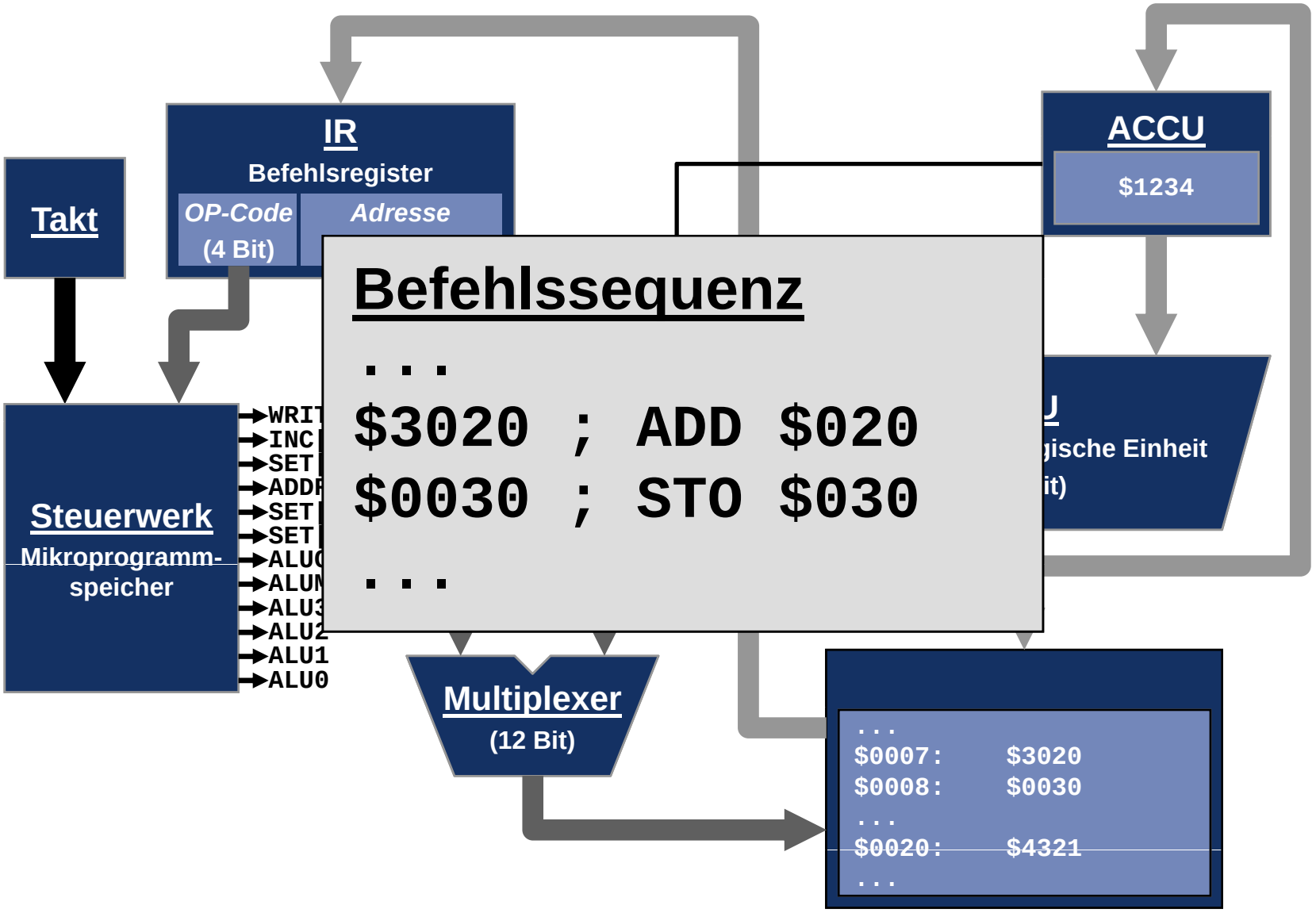
ALU3	ALU2	ALU1	ALU0	ALUMODE = 0 (arithmetischer Modus)		ALUMODE = 1 (logischer Modus)
				ALUCIN = 0	ALUCIN = 1	
0	0	0	0	A	A + 1	¬A
0	0	0	1	A | B	(A | B) + 1	¬(A & B)
0	0	1	0	A | ¬B	(A | ¬B) + 1	¬A & B
0	0	1	1	–1	0	0
0	1	0	0	A + (A & ¬B)	A + (A & ¬B) + 1	¬(A & B)
0	1	0	1	(A | B) + (A & ¬B)	(A | B) + (A & ¬B) + 1	¬B
0	1	1	0	A – B – 1	A – B	A ⊕ B
0	1	1	1	(A & B) – 1	A & B	A & ¬B
1	0	0	0	A + (A & B)	A + (A & B) + 1	¬A | B
1	0	0	1	A + B	A + B + 1	¬(A ⊕ B)
1	0	1	0	(A | ¬B) + (A & B)	(A | ¬B) + (A & B) + 1	B
1	0	1	1	(A & B) – 1	A & B	A & B
1	1	0	0	A + A	A + A + 1	1
1	1	0	1	(A | B) + A	(A | B) + A + 1	A | ¬B
1	1	1	0	(A | ¬B) + A	(A | ¬B) + A + 1	A | B
1	1	1	1	A – 1	A	A

Der Befehlszähler

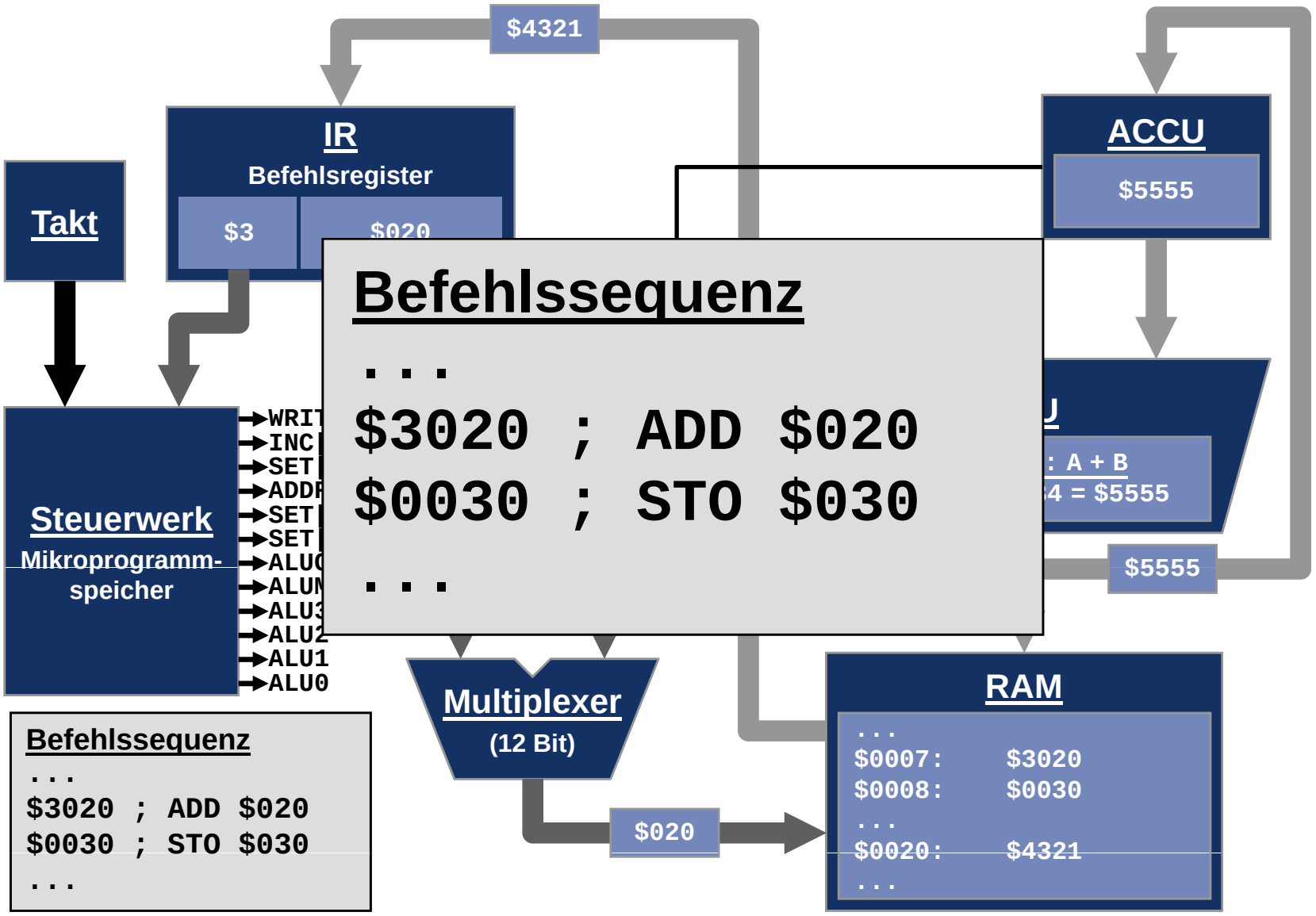



Takt
OPC[0]
OPC[1]
OPC[2]
OPC[3]
Steuerw
Mikroprogr
speich
0 0 100100011111 ; STO
1 010010000000 ;
1 2 000101011010 ; LDA
3 010010000000 ;
2 4 001100000000 ; BRZ
5 010010000000 ;
3 6 000101001001 ; ADD
7 010010000000 ;
4 8 000101100110 ; SUB
9 010010000000 ;
5 10 000101011110 ; OR
11 010010000000 ;
6 12 000101011011 ; AND
13 010010000000 ;
7 14 000101010110 ; XOR
15 010010000000 ;
8 16 010011010000 ; NOT
17 000000000000 ;
9 18 010011100000 ; INC
19 000000000000 ;
10 20 010011001111 ; DEC
21 000000000000 ;
11 22 010011010011 ; ZRO
23 000000000000 ;
12 24 010010000000 ; NOP
25 000000000000 ;
13 26 010010000000 ; NOP
27 000000000000 ;
14 28 010010000000 ; NOP
29 000000000000 ;
15 30 010010000000 ; NOP
31 000000000000 ;
WRITE[RAM]
INC[PC]
SET[PC]
ACCU
Akkumulator
ADDR=IR
SET[IR]
SET[ACCU]
ALUCIN
ALUMODE
ALU3
ALU2
ALU1
ALU0
à 16 Bit)

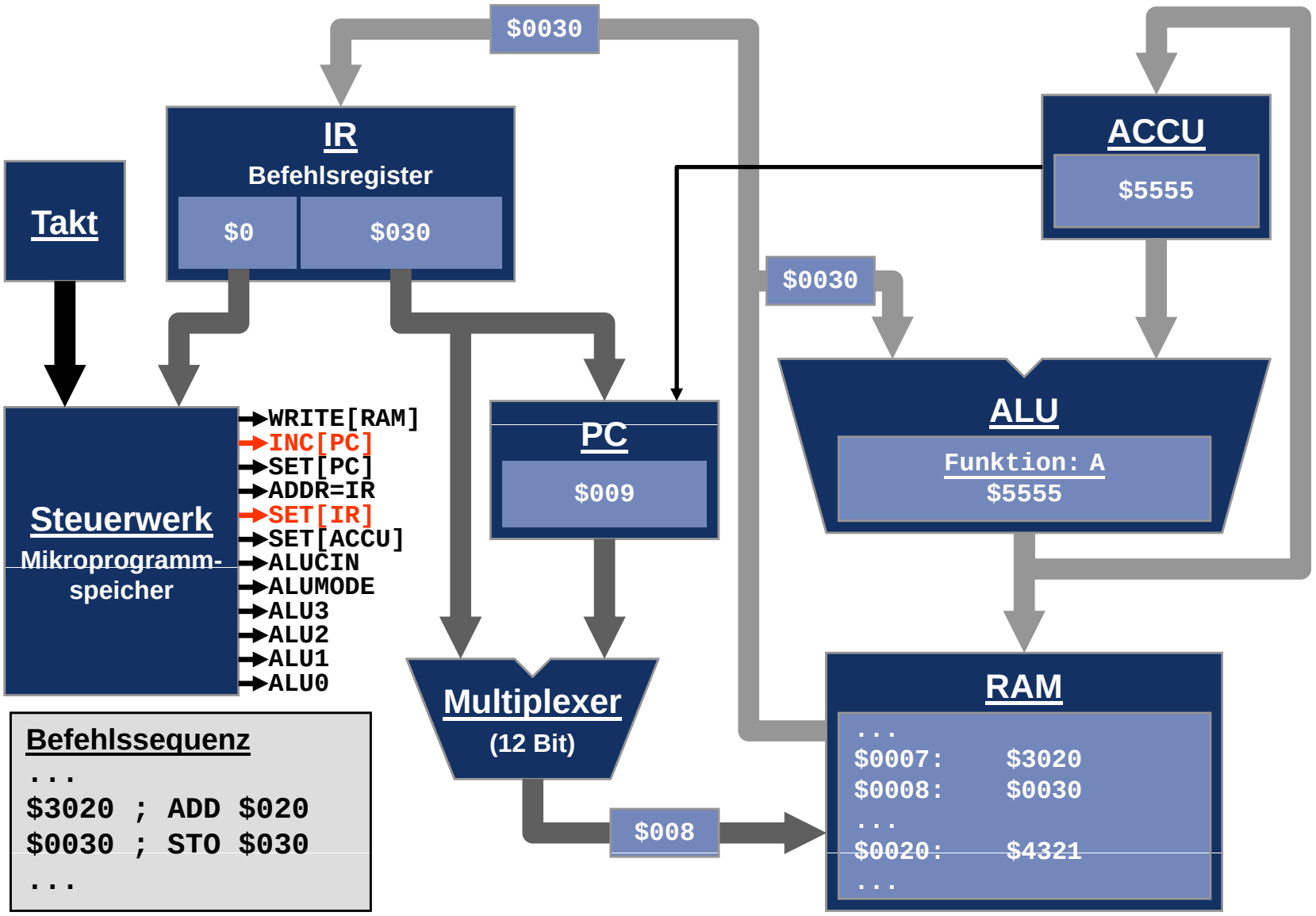
Ablauf eines Maschinenbefehls: Ausgangssituation



Ablauf eines Maschinenbefehls: Phase 1



Ablauf eines Maschinenbefehls: Phase 2



Toy-Programm

```
; Variablen:
;   Loopcount=$20, Result=$21 (enthaelt zunaechst 0)
; Labels:
;   loop=$2, end=$b
;
$0020 ; STO Loopcount      ; Auswerten des initialen ACCU-Inhalts
$200b ; BRZ end            ; Schon fertig?
#-------------------------
#loop:
$1021 ; LDA Result         ; Schleifenzaehler zu Result addieren
$3020 ; ADD Loopcount
$0021 ; STO Result
$1020 ; LDA Loopcount      ; Schleifenzaehler aktualisieren
$a000 ; DEC
$0020 ; STO Loopcount
$200b ; BRZ end            ; Fertig?
$b000 ; ZRO                ; Nein,
$2002 ; BRZ loop           ; dann wieder von vorn
#-------------------------
#end:
$b000 ; ZRO
$200b ; BRZ end            ; Endlosschleife
```

Der Toy-Emulator

```
================================================================================

   Accu    :  $    1                Address :   Contents
   PC      :  $    7               ----------------------------------------
   InsnReg :  STO $20
                                     $   0  :   $  20  $200b  $1021  $3020
---------------------------------    $   4  :   $  21  $1020  $a000  $  20
                                     $   8  :   $200b  $b000  $2002  $b000
   Steps   :          79            $   c  :   $200c  $   0  $   0  $   0
                                     $  10  :   $   0  $   0  $   0  $   0
---------------------------------    $  14  :   $   0  $   0  $   0  $   0
                                     $  18  :   $   0  $   0  $   0  $   0
        T O Y - EMULATOR             $  1c  :   $   0  $   0  $   0  $   0
        (c) 1992-95, gjh             $  20  :   $   2  $  36  $   0  $   0
                                     $  24  :   $   0  $   0  $   0  $   0
   <Space>   Single Step             $  28  :   $   0  $   0  $   0  $   0
   <S>       Enter # of Steps        $  2c  :   $   0  $   0  $   0  $   0
   <A>       Set Accumulator         $  30  :   $   0  $   0  $   0  $   0
   <P>       Set Program Counter     $  34  :   $   0  $   0  $   0  $   0
   <M>       Set RAM Cell            $  3c  :   $   0  $   0  $   0  $   0
   <+>, <->  Scroll RAM-Window       $  40  :   $   0  $   0  $   0  $   0
                                     $  44  :   $   0  $   0  $   0  $   0
   <Q>, <^C> Game Over

================================================================================
```

Der Toy-Emulator

```
================================================================================

   Accu    :  $    1                Address :   Contents
   PC      :  $    8               ----------------------------------------
   InsnReg :  BRZ $b
                                     $   0  :   $  20  $200b  $1021  $3020
---------------------------------    $   4  :   $  21  $1020  $a000  $  20
                                     $   8  :   $200b  $b000  $2002  $b000
   Steps   :          80            $   c  :   $200c  $   0  $   0  $   0
                                     $  10  :   $   0  $   0  $   0  $   0
---------------------------------    $  14  :   $   0  $   0  $   0  $   0
                                     $  18  :   $   0  $   0  $   0  $   0
        T O Y - EMULATOR             $  1c  :   $   0  $   0  $   0  $   0
        (c) 1992-95, gjh             $  20  :   $   1  $  36  $   0  $   0
                                     $  24  :   $   0  $   0  $   0  $   0
   <Space>   Single Step             $  28  :   $   0  $   0  $   0  $   0
   <S>       Enter # of Steps        $  2c  :   $   0  $   0  $   0  $   0
   <A>       Set Accumulator         $  30  :   $   0  $   0  $   0  $   0
   <P>       Set Program Counter     $  34  :   $   0  $   0  $   0  $   0
   <M>       Set RAM Cell            $  3c  :   $   0  $   0  $   0  $   0
   <+>, <->  Scroll RAM-Window       $  40  :   $   0  $   0  $   0  $   0
                                     $  44  :   $   0  $   0  $   0  $   0
   <Q>, <^C> Game Over

================================================================================
```

Unterschiede zu realen Rechnern

	Toy Rechner	Reale Prozessoren
Wortlänge	16 BitDaten 12 Bit Adresssen	bis 100 Bit
Mikroinstruktionen	1 Routine pro Maschinenbefehl	mehrere Routinen pro Maschinenbefehl
Umfang des Mikroprogramms	384 Bit	300 000 Bit
Verzweigungsbefehle	1 Verzweigungsbefehl	10-33 Verzweigungsbefehle
Adressierungsmodi	1 Addressierungsmodus	1-21 Addressierungsmodi
Befehlssatz	12 Befehle	100 - 300 Befehle
Registersatz	1 Register (Akku)	32 - 512 Register

Der Bus als Kommunikationsmedium

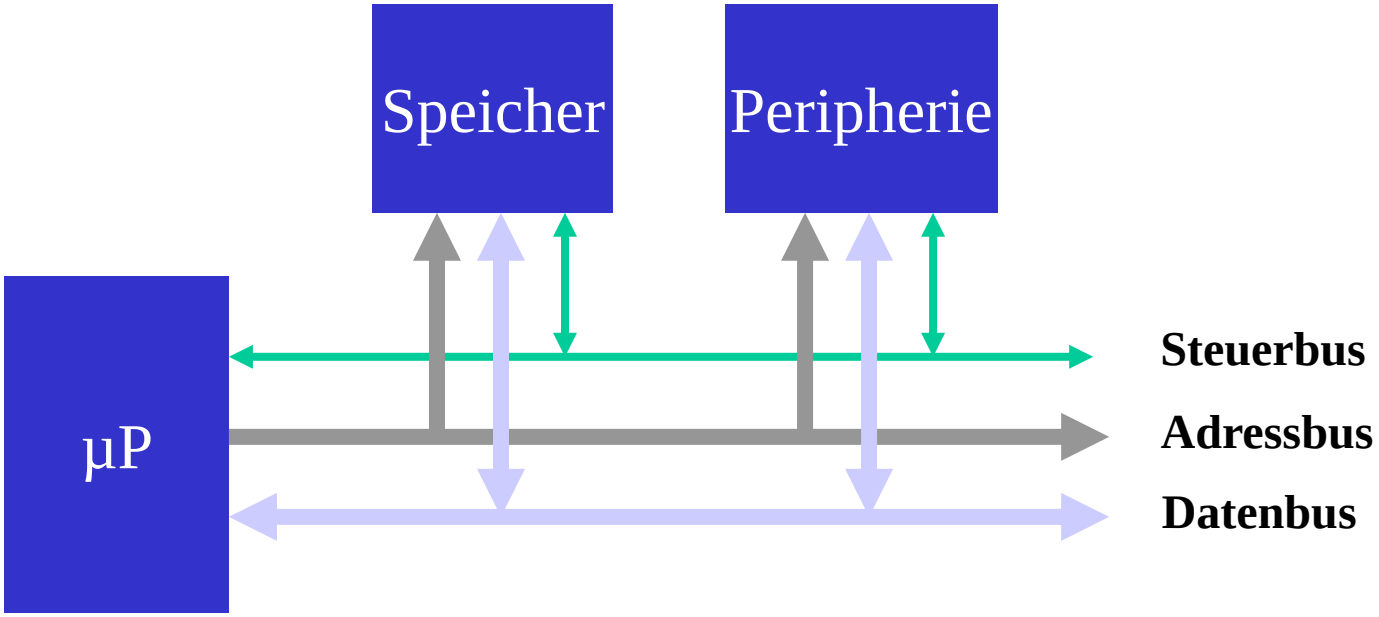
- Bus ist eine Sammelleitung zur Übertragung von Daten zwischen mehreren Funktionseinheiten einer Rechenanlage
- Es muss dafür gesorgt werden, dass
  - ⇨ verschiedene Geräte nicht gleichzeitig Daten senden
    - Bus Arbitrierung
  - ⇨ nur solche Geräte die Daten empfangen, für die sie bestimmt sind
    - Bus Protokoll
- Ein Bus besteht in der Regel aus
  - ⇨ Datenleitungen
  - ⇨ Adressleitungen
  - ⇨ Steuerleitungen

Rechner- und Gerätebusse

- Busse verbinden Komponenten eines Rechnersystems
  - Datenbus 8 bis 64 Bit
  - Adressbus 16 bis 64 Bit
  - Steuerbus
- Systembusse
  - Busse, die rechnerinterne Komponenten verbinden
  - AT-Bus PC/XT (8088/ 8086)
  - ISA-Bus AT (80286)
  - EISA 80386 und 80486
  - VESA ab 80486
  - PCI ab 80486 bis Pentium4
  - PCIe seit ca. 2005 schrittweiser Ersatz für PCI und AGP
- Gerätebusse
  - Busse, die externe Komponenten mit einem Rechnersystem verbinden
  - IEC Gerätebus
  - EIDE Festplatten
  - SCSI Geräte und Festplattenbus

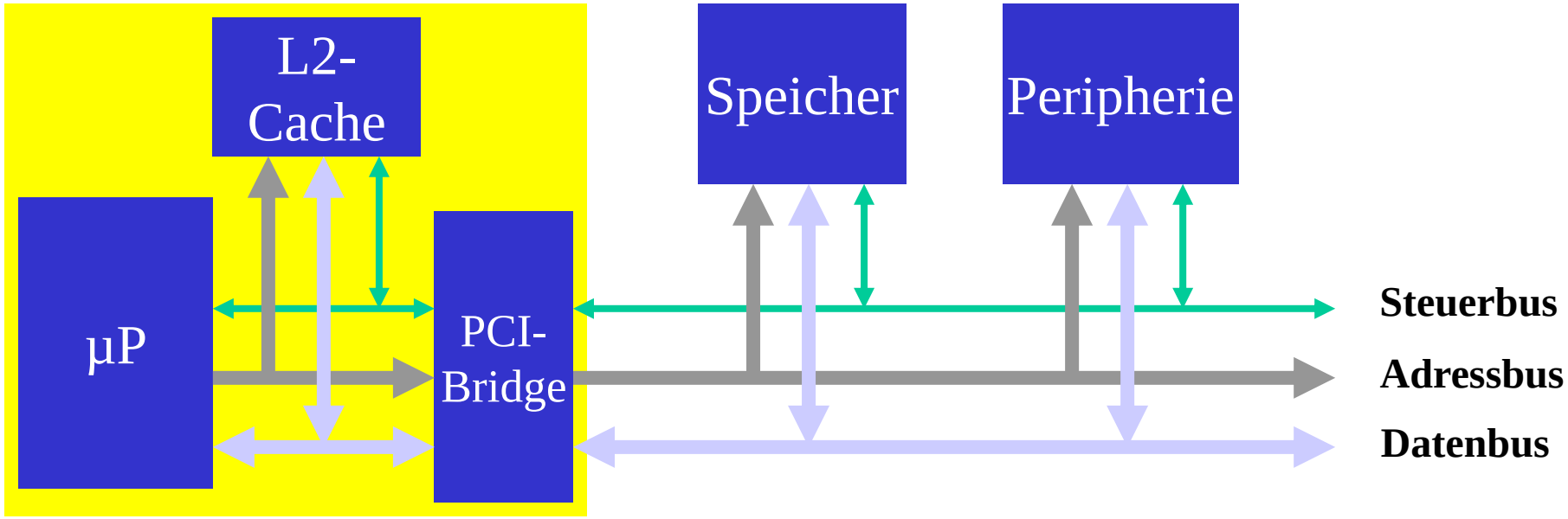
Systembusse

- Verbindung der Komponenten innerhalb eines Rechnersystems
- Prozessorabhängige Busse
  - ⇨ Busschnittstelle des verwendeten Mikroprozessors
  - ⇨ einfach zu realisieren
  - ⇨ auf einen Prozessor zugeschnitten

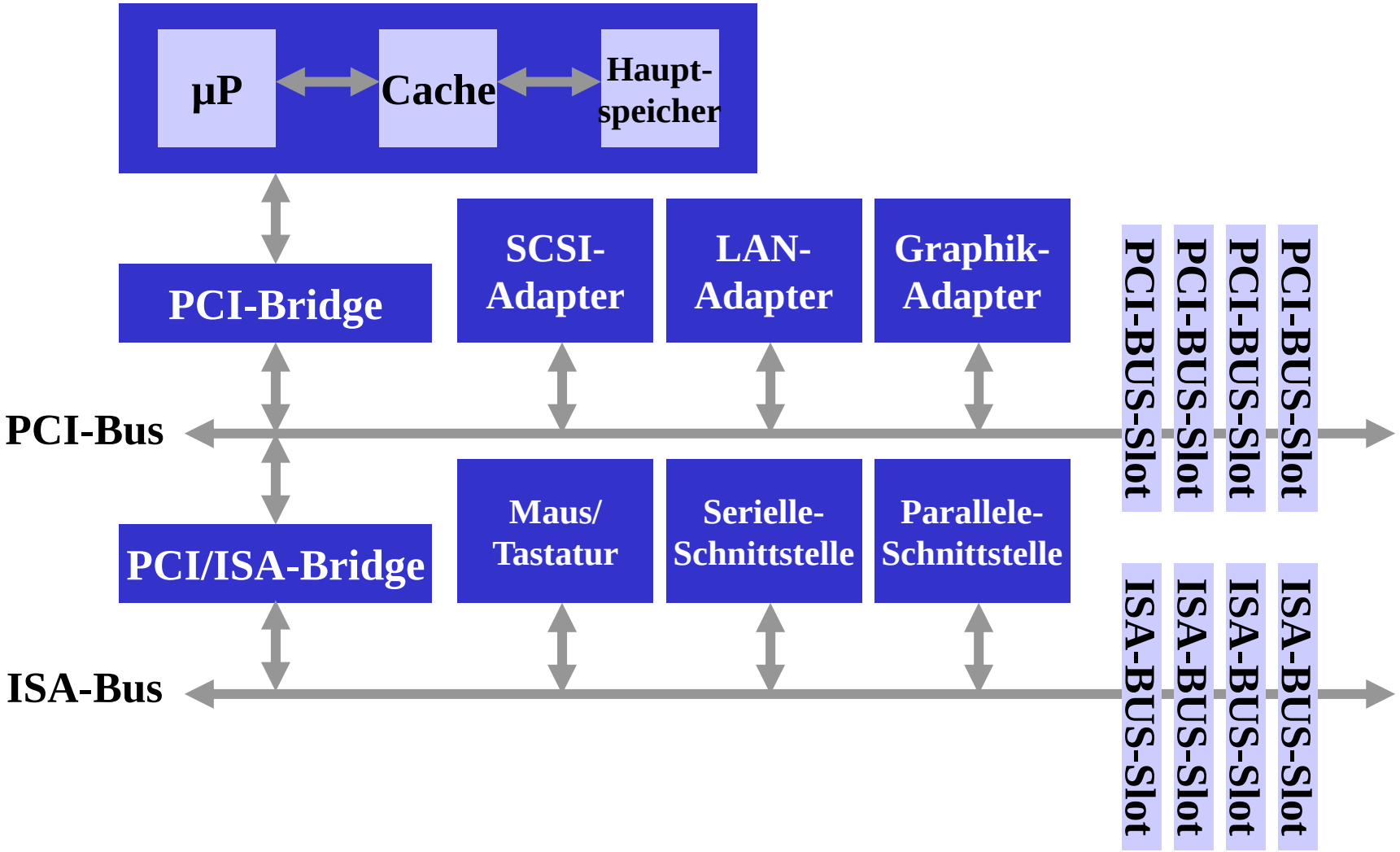


Prozessorunabhängige Systembusse

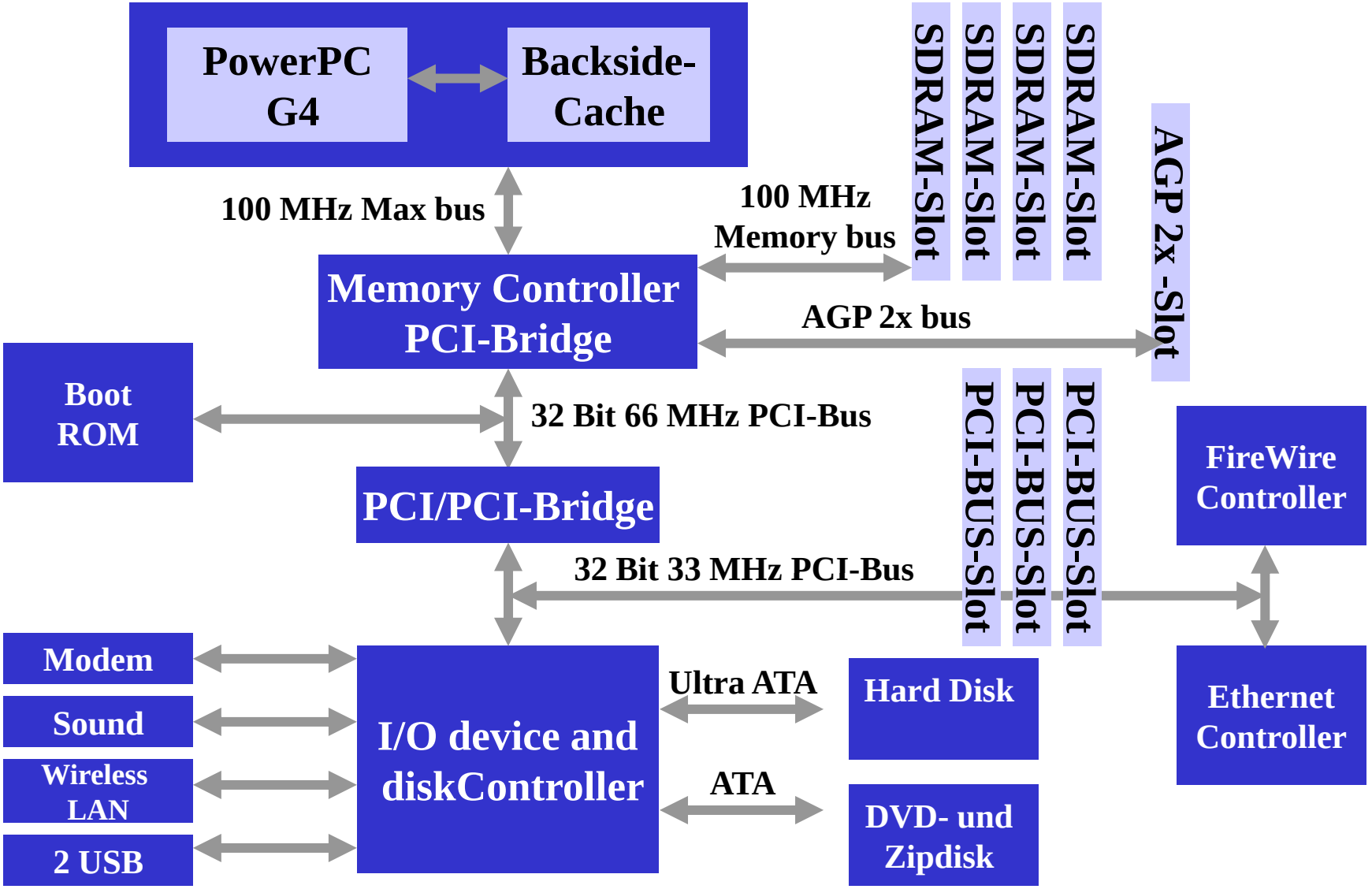
- Der Prozessor ist über eine Bus Brücke mit den Komponenten verbunden
  - ⇨ z.B. PCI Bridge (Peripheral Component Interconnect)
  - ⇨ erhöhter Hardwareaufwand
  - ⇨ die Komponenten können für verschiedene Prozessoren verwendet werden (PPC 403, Pentium, I860, ...)



Beispiel: Busstruktur eines PC

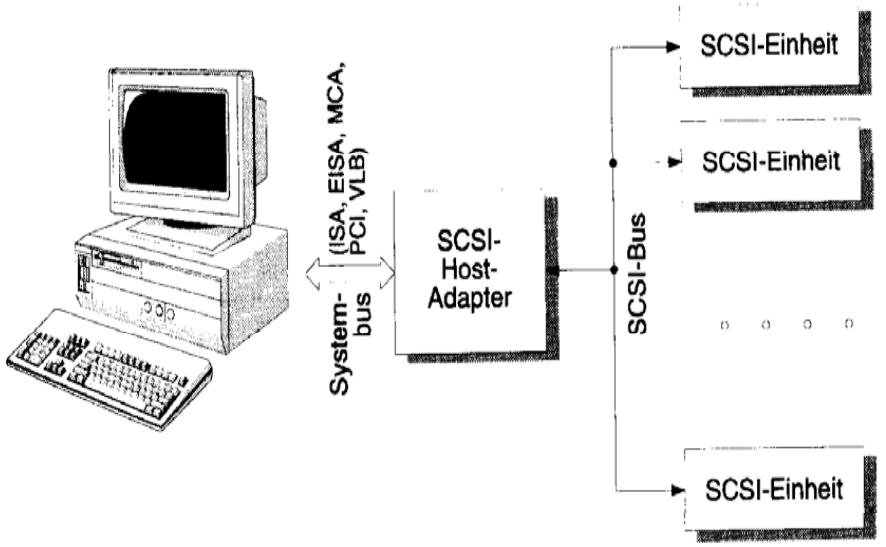


Beispiel: Busstruktur des Power Mac G4



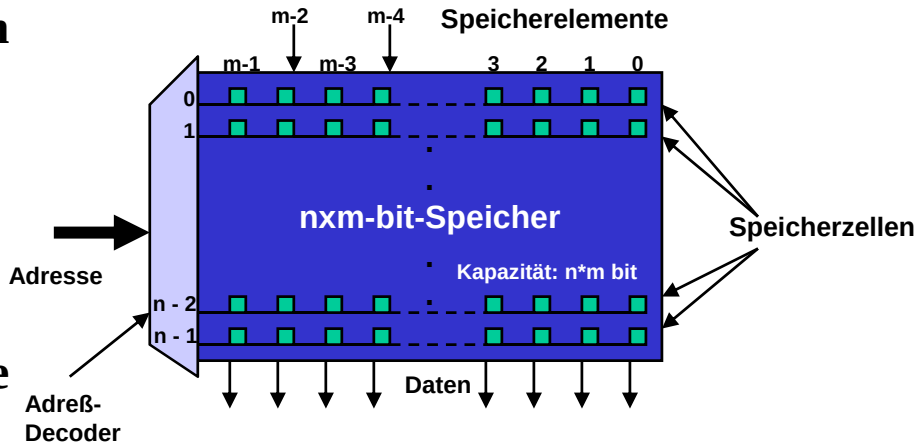
Gerätebusse: Der SCSI-Bus

- Small Computer Systems Interface (SCSI)
  - ⇨ Maximal 8 Einheiten
  - ⇨ 8 Bit Übertragung
  - ⇨ Identifikation durch SCSI-ID
  - ⇨ Terminierung durch Abschlußwiderstand
- Weitere SCSI-Standards
  - ⇨ SCSI-II
    - Erster richtiger Standard, der am gleichen Bus auch andere Geräte auf Festplatten berücksichtigt
  - ⇨ Fast SCSI
    - maximale Taktfrequenz wurde auf 10 MHz erhöht
  - ⇨ Wide SCSI
    - 16 Bit und 32 Bit Erweiterung der Datenbreite



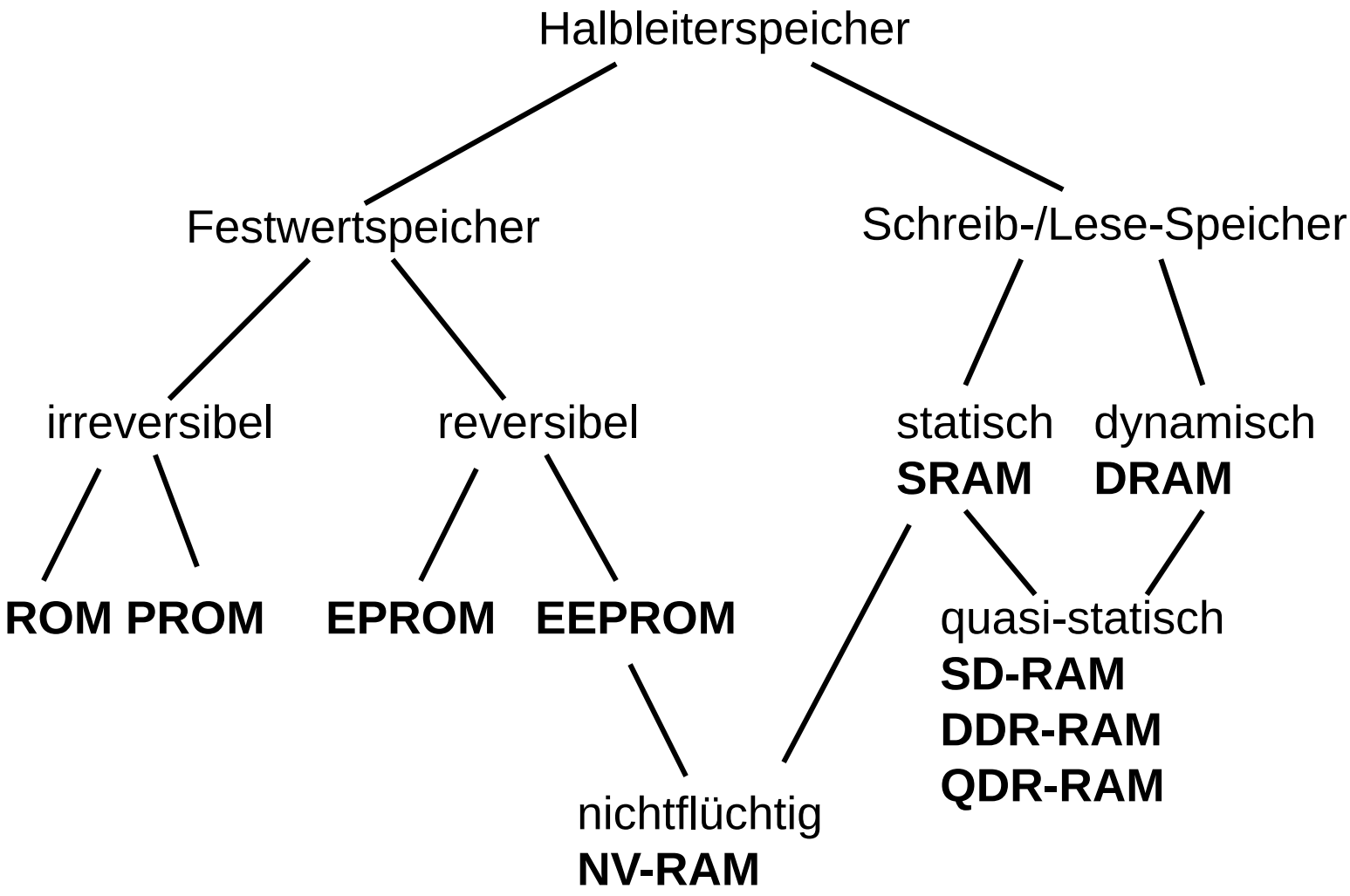
Aufbau von Speicherzellen

- Speicherung von Daten oder von logischen Funktionen
- Arten der Speicherung
  - ⇨ irreversibel programmierbare Speicherzellen
  - ⇨ reversibel programmierbare Speicherzellen
  - ⇨ spezielle Transistorschaltungen als statisches Speicherelement
  - ⇨ Speicherung der Daten in einem Kondensator
- Speicherung der kleinsten Informationseinheit (Bit) in einem Speicherelement

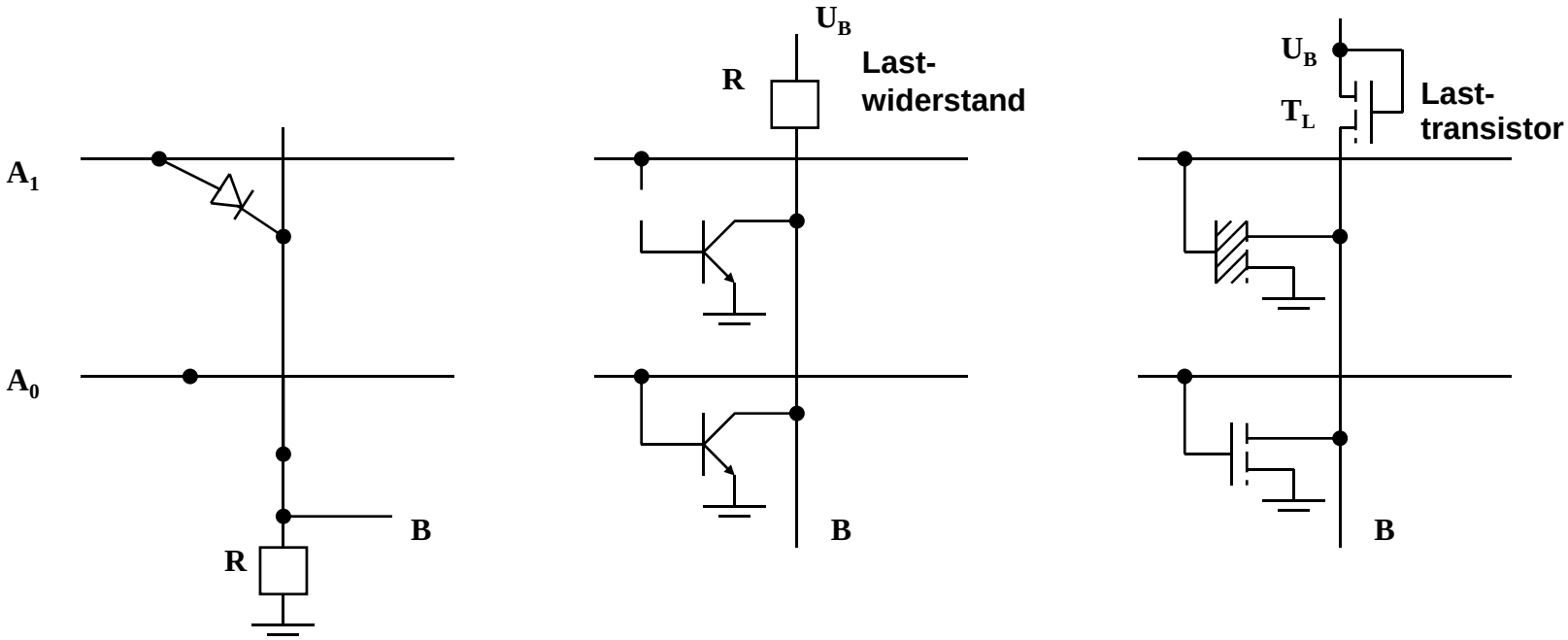


- Speicherzelle
  - ⇨ Speicherelemente, die unter einer gemeinsamen Adresse ansprechbar sind
- Speicherwort
  - ⇨ Datenbusbreite
- Organisation
  - ⇨ Anzahl der Speicherzellen
  - ⇨ Anzahl der Speicherelemente
  - ⇨ n*m Bit
- Kapazität
  - ⇨ Zahl der Speicherelemente

Klassifizierung von Halbleiterspeichern

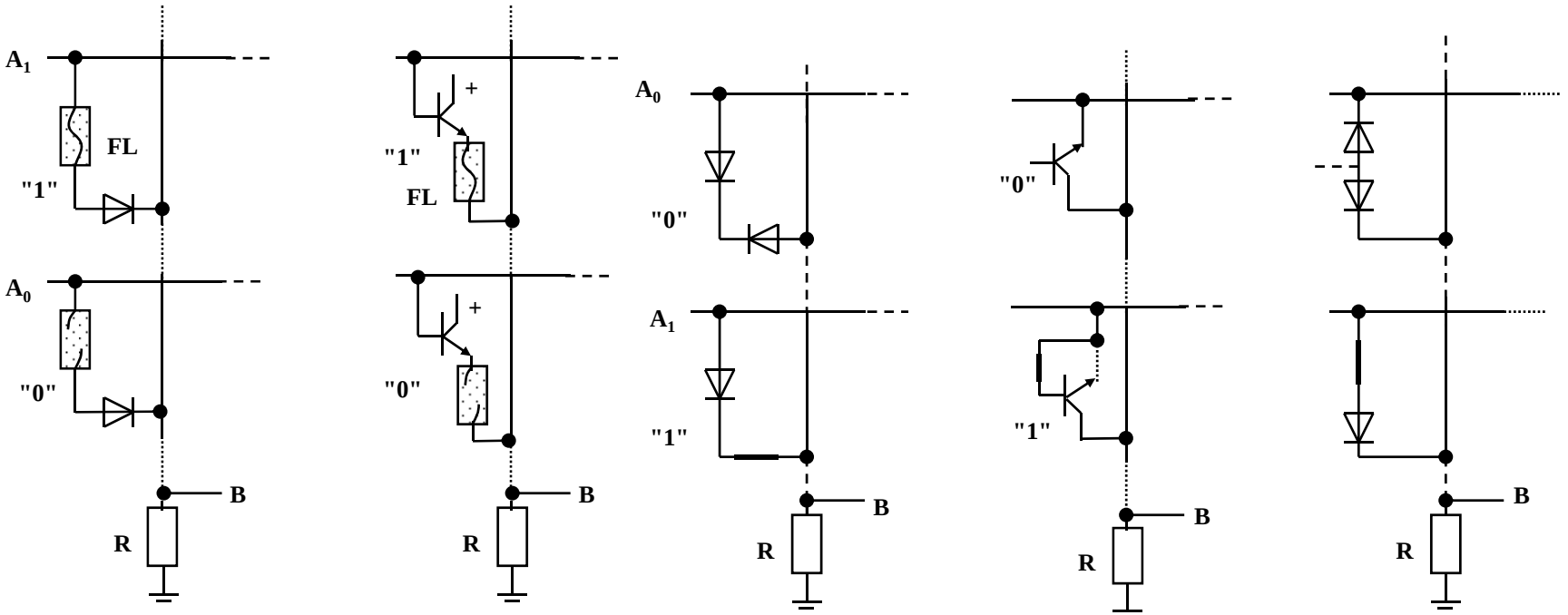


Speicherzellen für maskenprogrammierbare Speicherelemente



- Maskenprogrammierbare Speicherelemente erhalten ihre Information bei der Herstellung des Chips
  - ⇨ Information steht auf einer der Masken
  - ⇨ Inhalt ist nicht veränderbar
- Bauelemente wie Dioden, Bipolar- oder MOS-Transistoren werden bei der Herstellung deaktiviert
  - ⇨ Bei MOS-Transistoren ist die Dicke der Gate-Isolation ausschlaggebend

Speicherzellen für programmierbare Speicherelemente

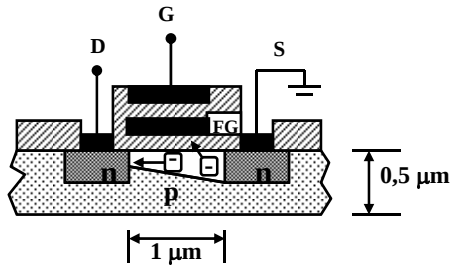


Speicherzellen mit Schmelzsicherungen

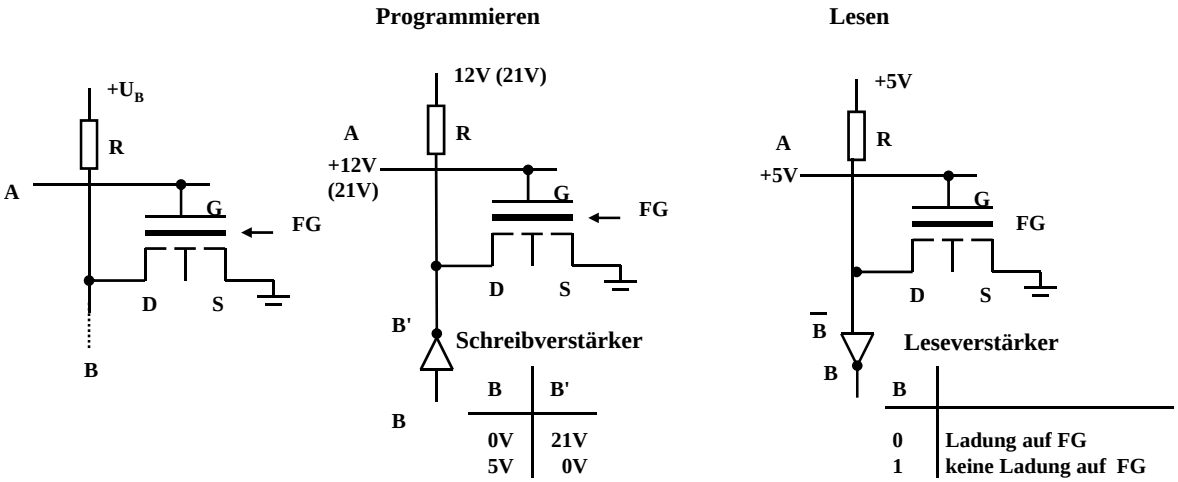
AIM-Speicherzellen

- Programmierung in Programmiergerät durch Überspannungen
  - ⇨ Schmelzsicherung
  - ⇨ Zerstören von Dioden (dauernd leitend)
- Information ist nur einmal schreibbar und kann nicht verändert werden

Löschbare Speicherelemente



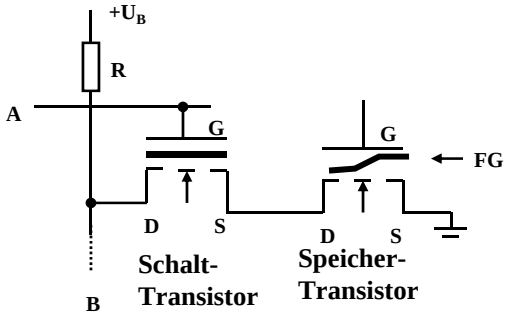
Gesamte Zellen-Fläche: 36 mm²

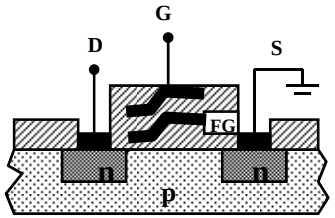


Programmieren und Lesen einer EPROM-Zelle

- Löschen durch UV-Licht
- FAMOS: floating gate avalance MOS-transisistor
  - ⇨ Besitzt zweites Gate, das vollständig isoliert ist
  - ⇨ Speicherung der Ladung über 30 Jahre
- Programmierung durch hohe Spannung (12-21 V)
  - ⇨ Elektronen werden angezogen
- Lesen durch Anlegen einer niederen Spannung (5 V)
  - ⇨ ist das Floating-Gate geladen, schaltet der Transistor nicht

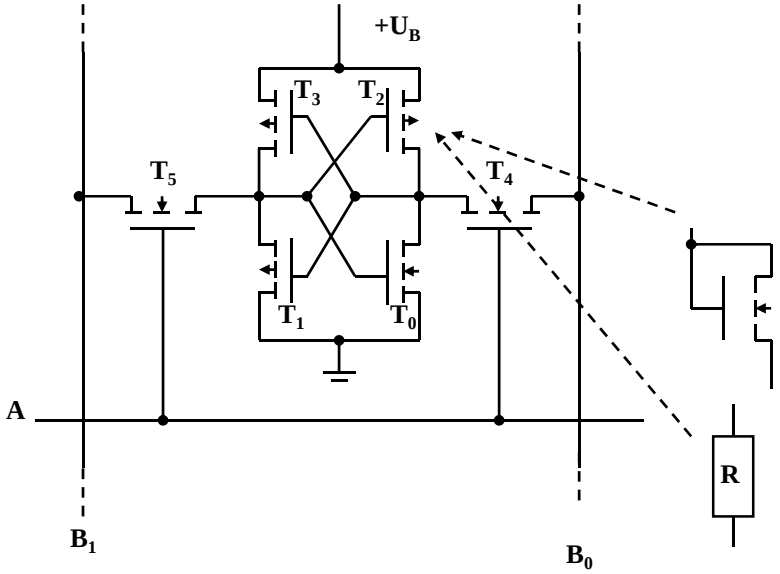
Elektrisch löschbare Speicherelemente





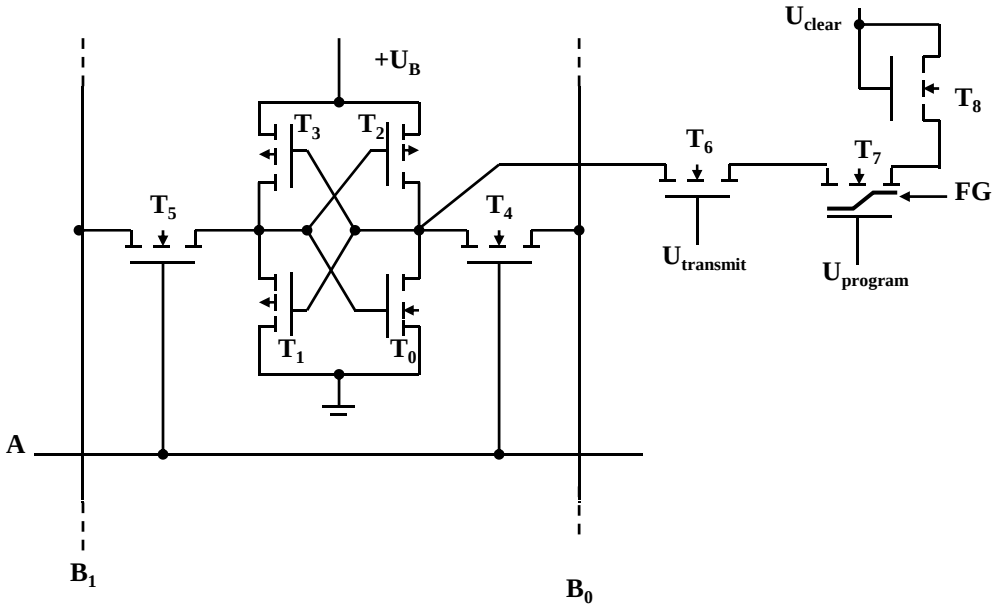
- **Dünne Isolierschicht des Floating Gates**
  - ⇨ **Lesen: Wenn das Floating Gate des Transistors geladen ist, sperrt dieser**
  - ⇨ **Löschen: Hohe Spannung (21 V) am Gate-Anschluß des Transistors lädt das Floating Gate (U_B = 0V)**
  - ⇨ **Programmieren: 0 V am Gate und eine hohe Spannung am Drain-Anschluß des Transistors entlädt einzelne Floating Gates (logisch 0)**

Statische MOS-Speicherelemente



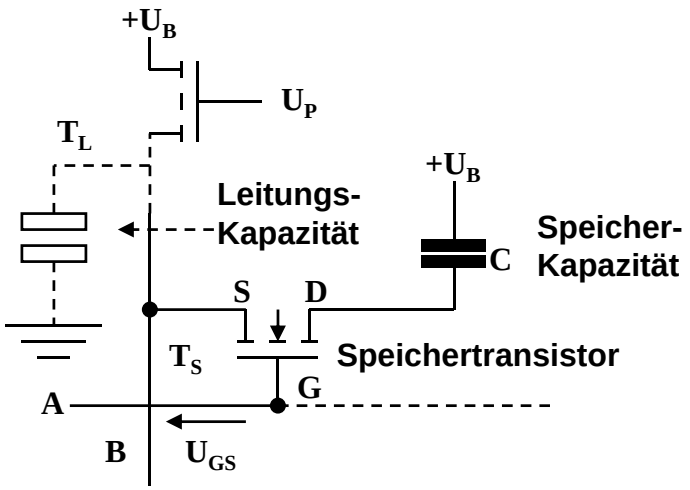
- **6-Transistorzelle**
  - ⇨ **Statt T_2 und T_3 können auch n-MOS-Transistoren oder Widerstände eingesetzt werden**
  - ⇨ **T_4 und T_5 dienen zur Ankopplung an die Bitleitungen**

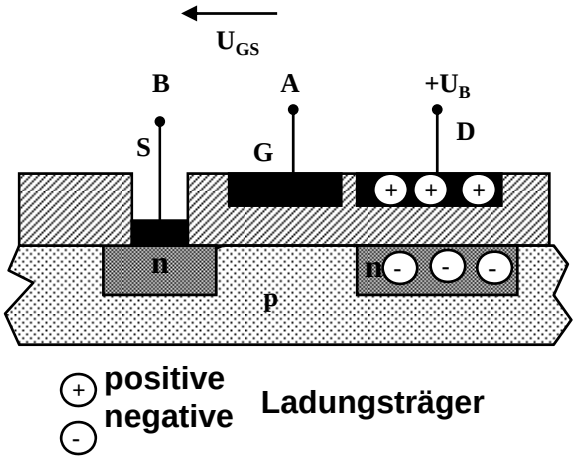
NVRAM-Speicherelemente



- **Kombination eines statischen mit einem EEPROM Speicherelement**
  - ⇨ **wenn die Spannung abfällt oder das Gerät eingeschaltet wird, findet eine Übertragung von bzw. in die EEPROM-Zelle statt**

Dynamische Speicherelemente

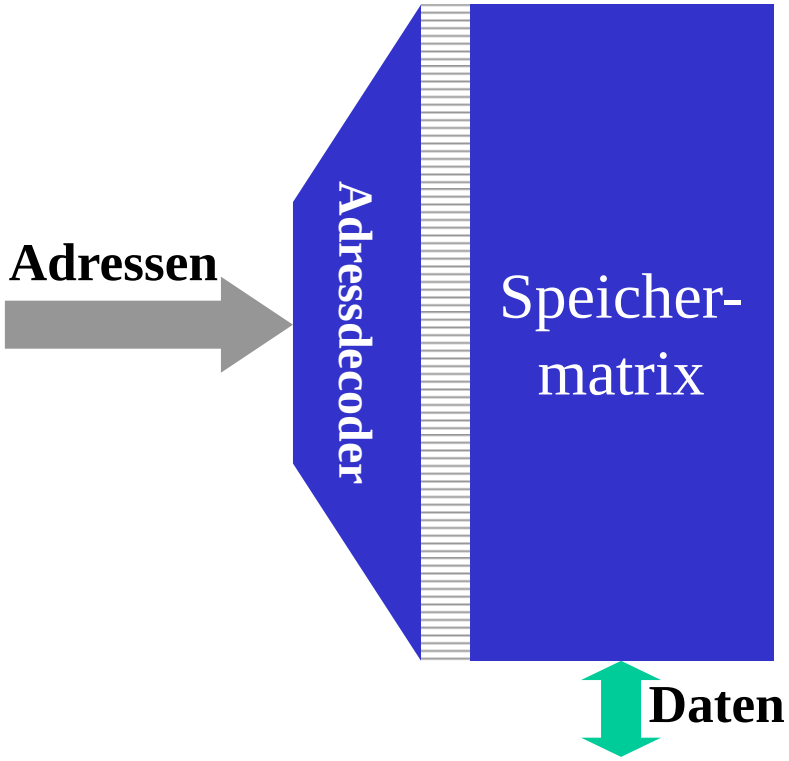




- Die Information wird in einem Kondensator gespeichert
  - ⇨ vergrößerte Drain-Zone
  - ⇨ isoliert zur Spannungsversorgung
- Kapazität 0,1 bis 0,5 pF, 100.000 bis 150.000 Elektronen
  - ⇨ Selbstentladung nach ca. 2 ms
- Speichern entspricht dem Laden des Kondensators
- Lesen entlädt den Kondensator
  - ⇨ Daten müssen wieder zurückgeschrieben werden

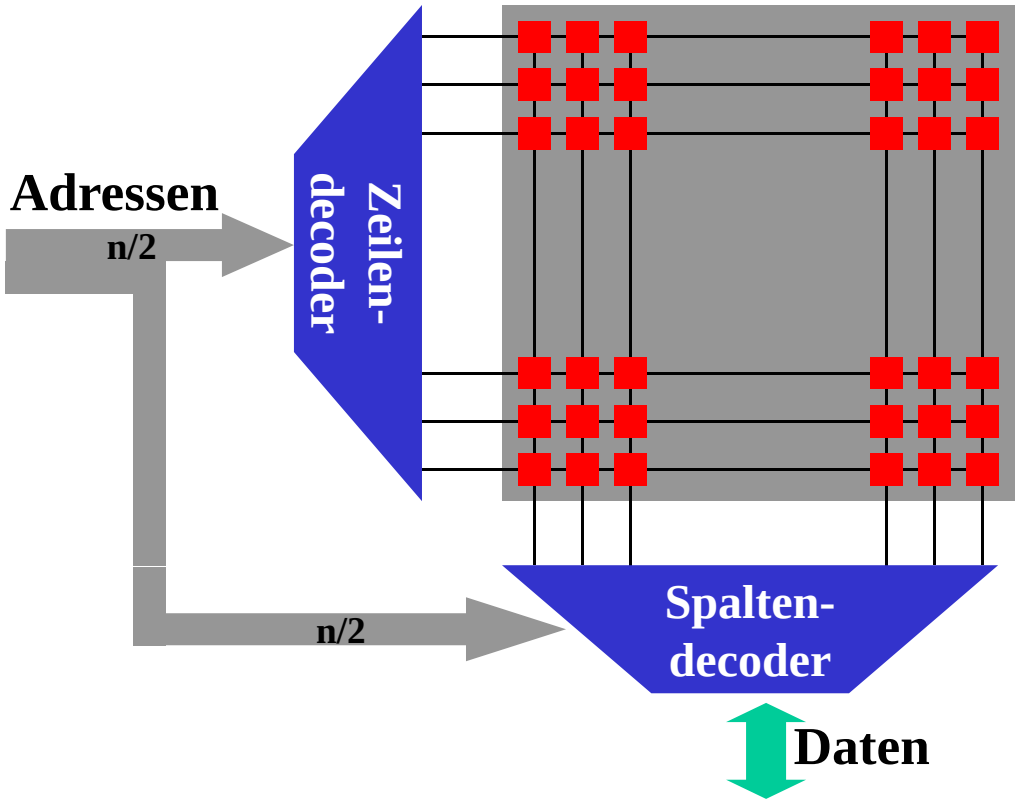
Äußere Organisation

- **Organisation des Speicherbausteins als Wortbreite und Adressbereich**
- **Beispiele: 512k*8, 4M*1**

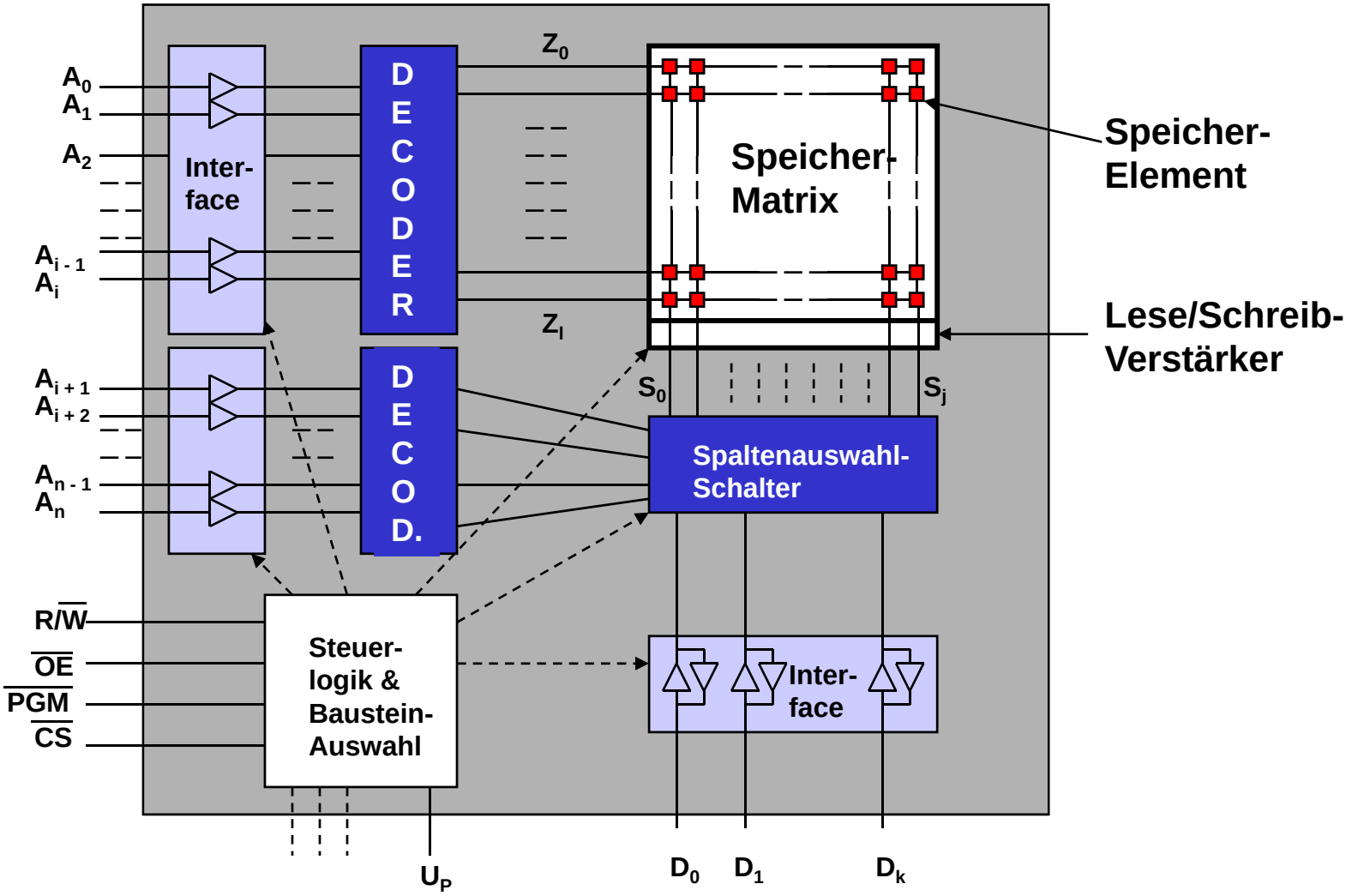


Innere Organisation

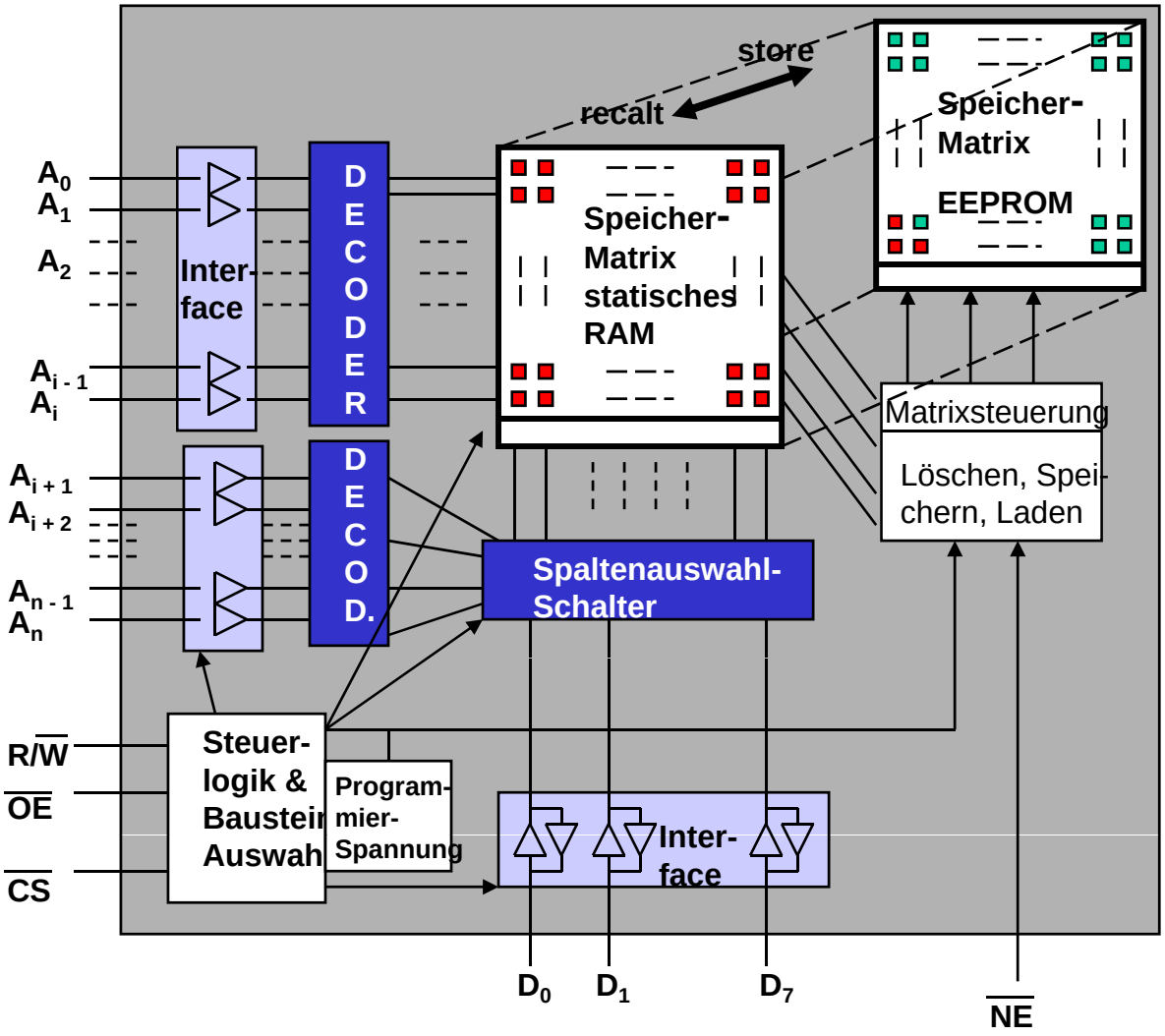
- Anordnung der Speicherzellen in einer quadratischen Matrix
  - ⇨ ergibt die minimale Anzahl der Ansteuerleitungen
- Aus den $2^{n/2}$ Datenbits werden im Spaltendecoder m Bits ausgewählt



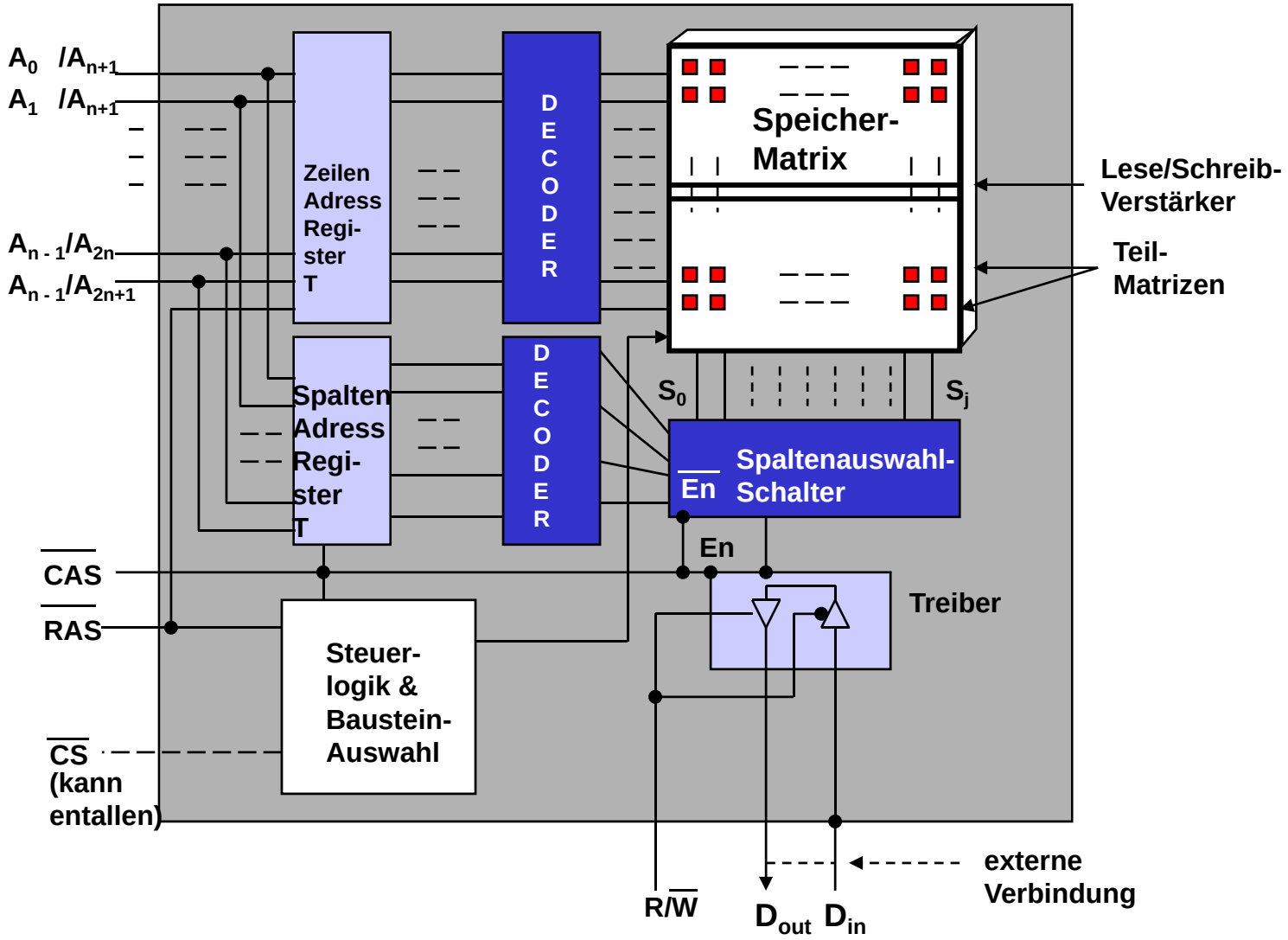
Aufbau eines Speicherbausteins



NVRAM-Bausteine

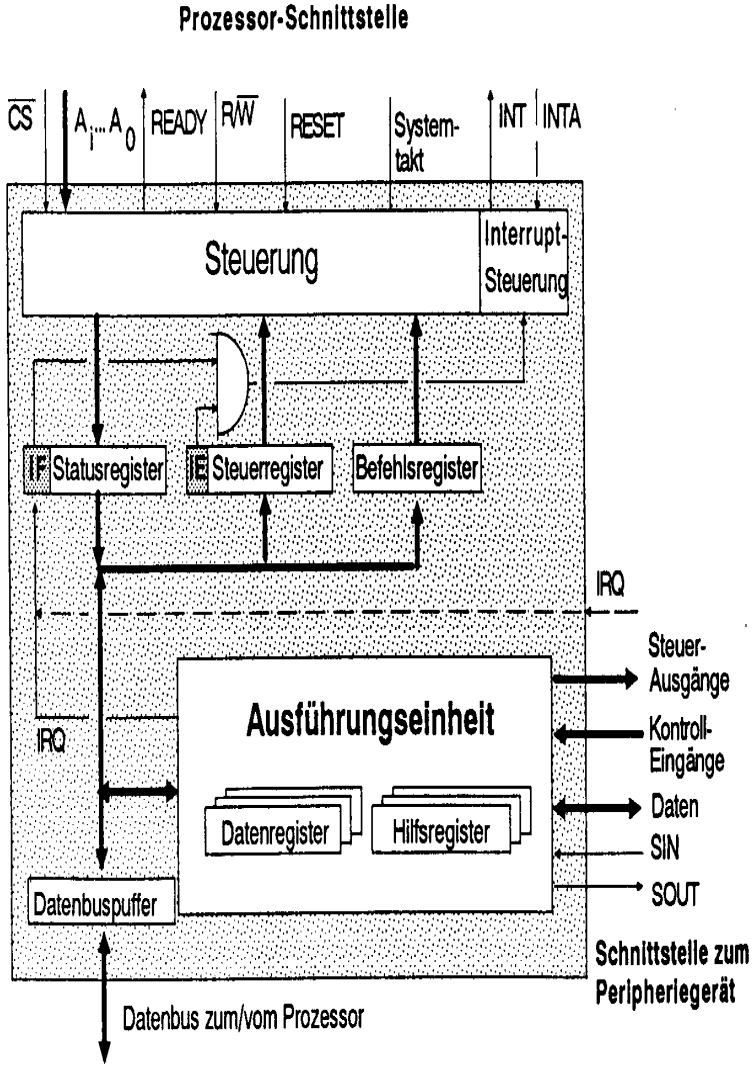


Dynamische RAM-Bausteine



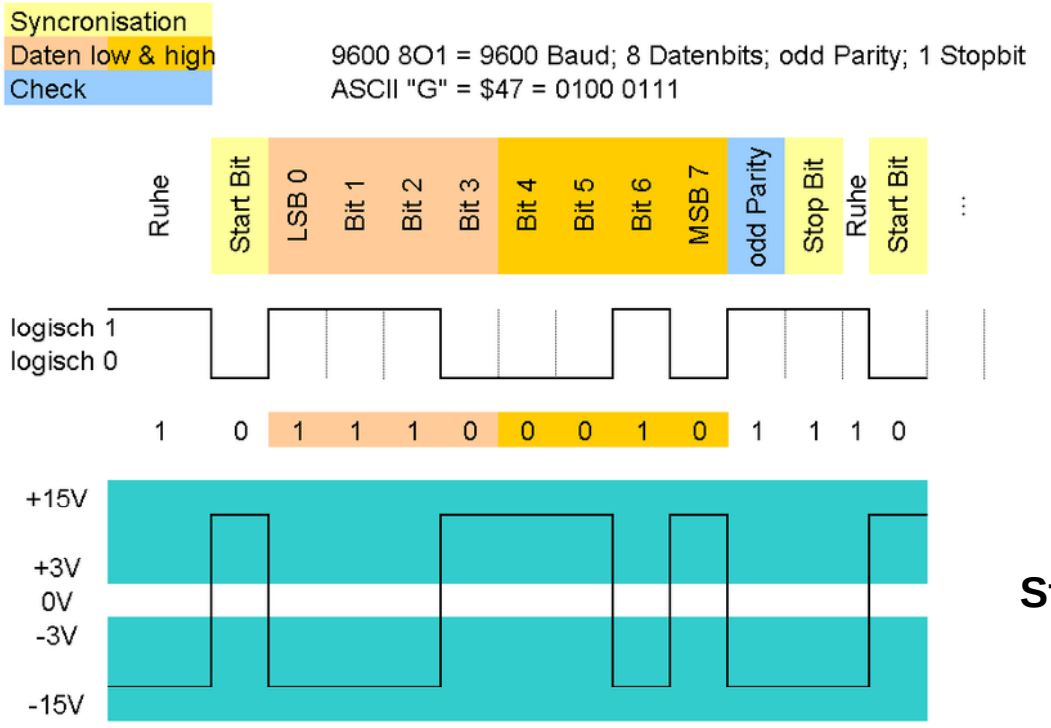
10 E/A und Peripheriegeräte

- Ein- und Ausgabe erfolgt über spezielle Speicherstellen im Adressraum des Prozessors
  - ⇨ Memory Mapped
  - ⇨ spezielle I/O-Befehle
- Adressdekodierung erzeugt das CS-Signal (chip select)
- Der Prozessor kommuniziert über
  - ⇨ Datenregister (Lesen und Schreiben der Daten)
  - ⇨ Statusregister (Zustand des Bausteins)
  - ⇨ Steuerregister (Betriebsart des Bausteins)



Serielle Schnittstelle: RS232

- **Synchronisation über den Aufbau des Datenworts**
  - ⇨ **Start: Startbit**
  - ⇨ **1,...,n: 5 bis 8 Datenbits**
  - ⇨ **P: Parität**
  - ⇨ **Stopp: 1, 1.5 oder 2 Stoppbits**



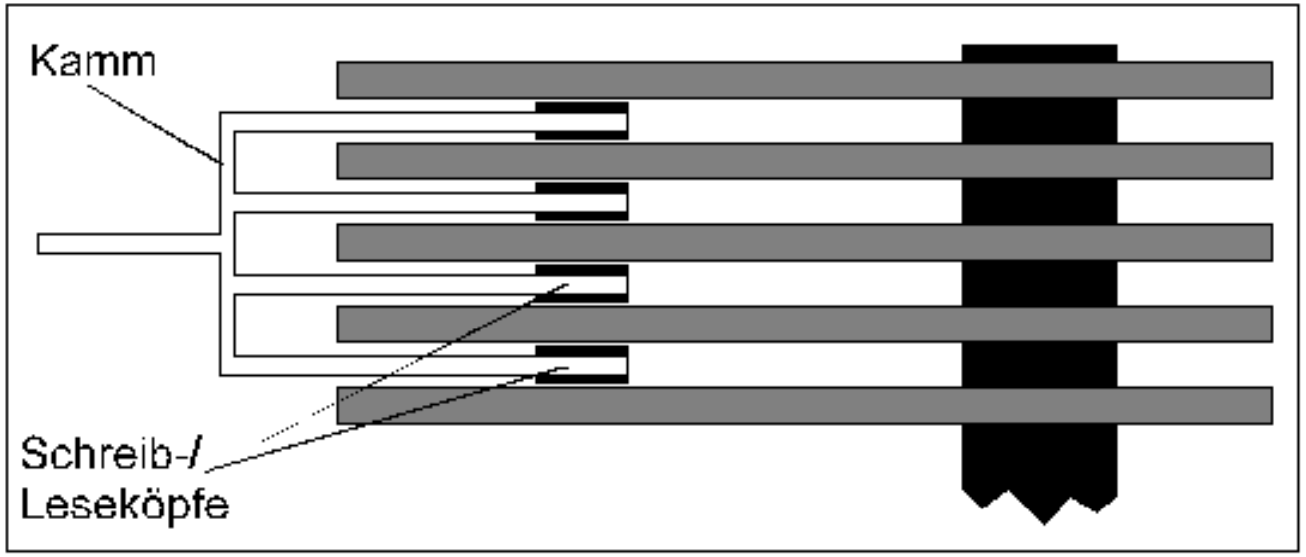
Beispiel: EIA 232 (RS232)

Steuerspannung

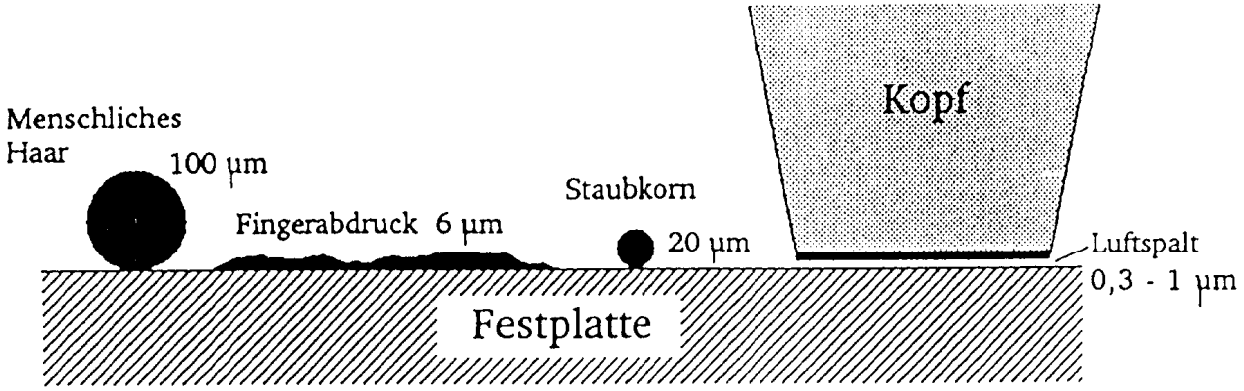
Die RS232-Schnittstelle

- **RTS: request to send**
  - ⇨ **Sendeteil einschalten**
- **CTS: clear to send**
  - ⇨ **Übertragungseinrichtung sendebereit**
- **DCD: data carrier detect**
  - ⇨ **Trägersignal erkannt**
  - ⇨ **Empfangsteil einschalten**
- **DSR: data set ready**
  - ⇨ **Übertragungseinrichtung betriebsbereit**
- **DTR: data terminal ready**
  - ⇨ **Empfangseinrichtung betriebsbereit**

Aufbau eines Festplatten-Laufwerks



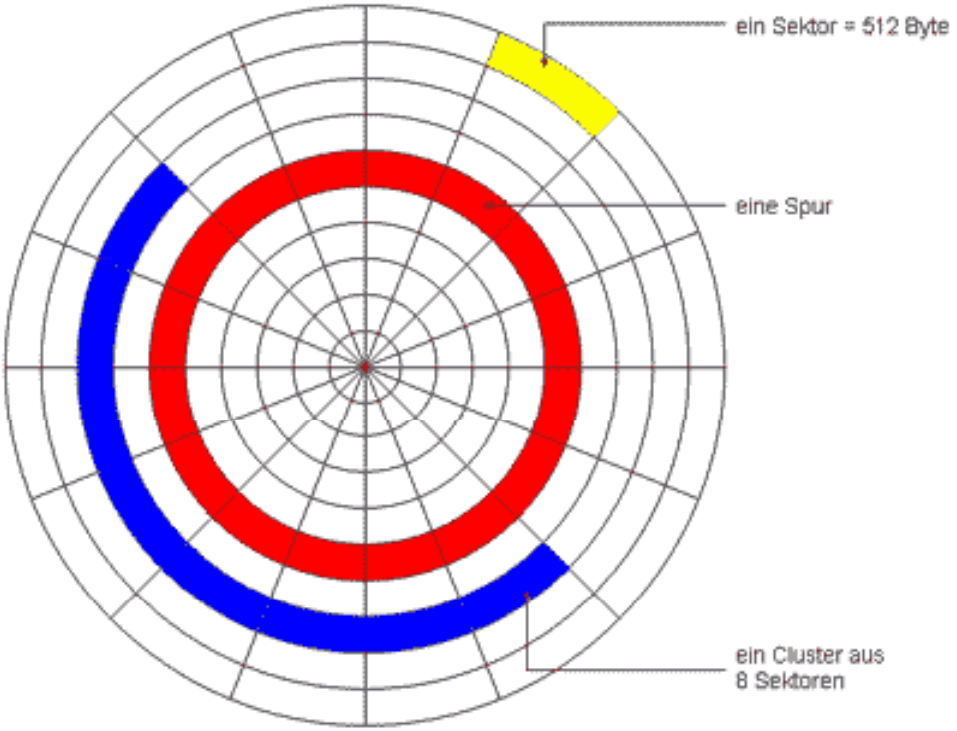
Größenverhältnisse im Festplatten-Laufwerk



Größenvergleich

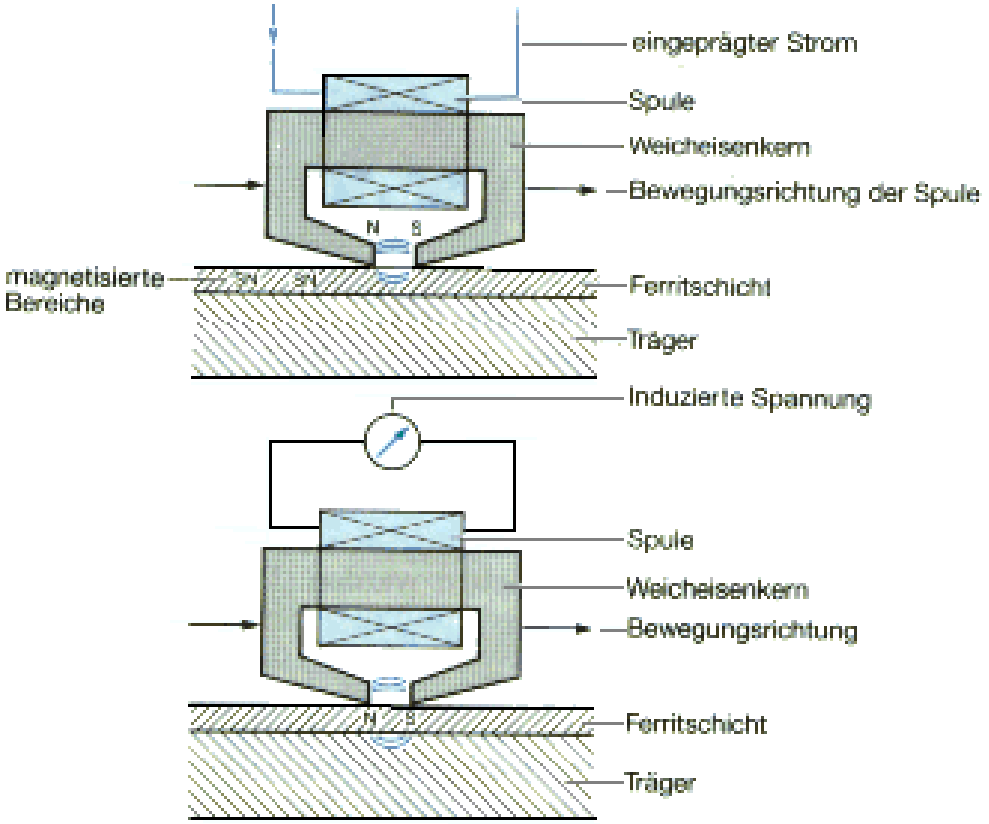
Sektoren einer Festplatte

- **Sektor: 512 Byte**
- **Spur: „Lesestreifen“**
  - ⇨ **Sektoren auf äußeren Spuren sind flächenmäßig größer**
  - ⇨ **Trotzdem 512 Byte!**
- **Cluster: Verbund von Sektoren; Größe abhängig von der Partitionierung**



Prinzip der Datenspeicherung

- **Das Prinzip der Datenaufzeichnung besteht darin, die Oberfläche der Platte informationsabhängig zu magnetisieren.**
- **Zur Unterscheidung der „0“- und „1“-Bits wird die Richtung der Magnetisierung verändert. Jede Änderung der Magnetisierungsrichtung wird als flusswechsel bezeichnet.**



Zukunft der Festplatte

- **SSD**
  - ⇨ **Solid State Drive**
  - ⇨ **Basier auf Speicherbausteinen**
    - **SRAM**
    - **Flash-Speicher**
  - ⇨ **Besteht aus Speicherbausteinen, wird aber wie eine Festplatte angesprochen**
    - **Schnellerer Zugriff auf Daten**
    - **Bis zu 64 GB**
      - **Stand: Juni 2008**

Zusammenfassung

- **TI1**
  - ⇨ **Elektrotechnische Grundlagen**
    - **Einfache physikalische Zusammenhänge, die verwendet werden um Schaltvorgänge in Rechnersystemen durchzuführen**
  - ⇨ **Halbleitertechnologie**
    - **Funktionsweise von Dioden und Transistoren**
    - **Einsatz von Transistoren als Schalter**
    - **CMOS-Schaltungen**
  - ⇨ **Digitale Grundlagen**
    - **Entwurf und Darstellung von Schaltnetzen**

Zusammenfassung

- **TI2**
  - ⇨ **Digitaltechnik**
    - **Optimierung von Schaltnetzen und Schaltwerken**
  - ⇨ **Komponenten digitaler Systeme**
    - **Funktion und Aufbau komplexer Bausteine**
    - **Komponenten aus denen Rechnersysteme aufgebaut sind**
  - ⇨ **Rechnerarithmetik**
    - **Darstellung von Zahlen und Zeichen in Rechnersystemen**
    - **Algorithmen zur Berechnung von Operationen wie die vier Grundrechenarten**
  - ⇨ **Aufbau und Funktionsweise einfacher Rechnersysteme**
    - **Komponenten**
    - **Busse**
    - **Speicher**
    - **Peripherie**