1. Schaltfunktionen und ihre Darstellung

- Zahlendarstellungen
- Boolesche Algebra
- Schaltfunktionen und Boolesche Funktionen
- Schaltnetze

b-adische Darstellung natürlicher Zahlen

Sei $b \in \mathbf{N}$ mit $b > 1$. Dann ist jede natürliche Zahl z mit $0 \leq z \leq b^n - 1$ (und $n \in \mathbf{N}$) eindeutig als Wort der Länge n über Σ_b darstellbar durch

$$z = \sum_{i=0}^{n-1} z_i b^i$$

mit $z_i \in \Sigma_b = \{0, \ldots, b-1\}$ für $i = 0, \ldots, n-1$.

Als vereinfachende Schreibweise ist dabei die folgende **Ziffernschreibweise** üblich:

$$z = (z_{n-1} z_{n-2} \ldots z_1 z_0)_b$$

Wichtiger Spezialfall: $b = 2$ ("**Dualdarstellung**" natürlicher Zahlen)

Zahlendarstellungen

Endliches Alphabet:

z.B. im Dezimalsystem: $\Sigma_{10} = \{0, 1, 2, \ldots, 9\}$

allgemein: $\Sigma_b = \{0, 1, 2, \ldots, b-1\}$, wobei b **Basis** genannt wird

b-adische Zahlen mit **endlicher Wortlänge** n: Σ_b^n

Beispiel: $00456 \in \Sigma_{10}^5$.

wichtige Zahlensysteme in der Informatik:

Dual-/Binärsystem: $\Sigma_2 = \{0, 1\}$

Oktalalsystem: $\Sigma_8 = \{0, 1, 2, \ldots, 7\}$

Hexadezimalsystem: $\Sigma_{16} = \{0, 1, 2, \ldots, 9, A, \ldots, F\}$

Boolesche Algebra

Erklärt man auf B drei Verknüpfungen wie folgt: Seien $x, y \in B$:

$$x \cup y := \mathrm{Max}(x, y)$$

$$x \cap y := \mathrm{Min}(x, y)$$

$$\overline{x} := 1 - x$$

so ist $(B, \cup, \cap, \bar{})$ eine **Boolesche Algebra**, d.h. ein distributiver, komplementärer Verband, in welchem es ein kleinstes (0) und ein größtes (1) Element gibt.

Gesetze einer Booleschen Algebra

(a) **Kommutativgesetze:** $x \cup y = y \cup x,\ x \cap y = y \cap x$

(b) **Assoziativgesetze:**
$(x \cup y) \cup z = x \cup (y \cup z),\ (x \cap y) \cap z = x \cap (y \cap z)$

(c) **Verschmelzungsgesetze:** $(x \cup y) \cap x = x,\ (x \cap y) \cup x = x$

(d) **Distributivgesetze:**
$x \cap (y \cup z) = (x \cap y) \cup (x \cap z),\ x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$

(e) **Komplementgesetze:** $x \cup (y \cap \overline{y}) = x,\ x \cap (y \cup \overline{y}) = x$

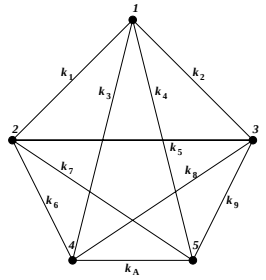
(f) $x \cup 0 = x,\ x \cap 0 = 0,\ x \cap 1 = x,\ x \cup 1 = 1$

(g) **de Morgansche Regeln:** $\overline{x \cup y} = \overline{x} \cap \overline{y},\ \overline{x \cap y} = \overline{x} \cup \overline{y}$

(h) $x = x \cup x = x \cap x = \overline{\overline{x}}$

Weitere Beispiele

- Existenz eines Euler-Kreises in einem Graphen mit 5 Knoten



- Existenz eines Hamilton-Kreises in einem Graphen mit 250 Punkten

Schaltfunktionen

Definition: Seien $n, m \in \mathbf{N}$, $n, m \geq 1$. Dann heißt eine Funktion $\mathcal{F} : B^n \to B^m$ Schaltfunktion.

Beispiele:

- Addition von zwei 16-stelligen Dualzahlen
- Multiplikation von zwei 16-stelligen Dualzahlen
- Sortieren von 30 16-stelligen Dualzahlen
- Primzahltest

Boolesche Funktionen

Eine Schaltfunktion $f : B^n \to B$ heißt (n-stellige) **Boolesche Funktion**.

Zusammenhang zu Schaltfunktionen:

Sei $\mathcal{F} : B^n \to B^m$ mit $\mathcal{F}(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$. Setzt man für jedes $i \in \{1, \ldots, m\}$

$$f_i : B^n \to B,$$

def. durch

$$f_i(x_1, \ldots, x_n) = y_i,$$

so ist $\mathcal{F}$ wie folgt darstellbar:

$$\mathcal{F}(x_1, \ldots, x_n) = (f_1(x_1 \ldots x_n), f_2(x_1 \ldots x_n), \ldots, f_m(x_1 \ldots x_n))$$

für alle $x_1, \ldots, x_n \in B$.

1-stellige Boolesche Funktionen

x	$f_0(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$
0	0	0	1	1
1	0	1	0	1

Es gilt: $f_0(x) \equiv 0, \quad f_1(x) = x, \quad f_2(x) = \overline{x}, \quad f_3(x) \equiv 1$

2-stellige Boolesche Funktionen I

(1)		$x \cdot \overline{x}$	$x \cdot y$	$x \cdot \overline{y}$	x	$\overline{x} \cdot y$	y	$\nleftrightarrow$	$x + y$
(2)		$\equiv 0$	Min	$>$	x	$<$	y	$\neq$	Max
(3)			$\wedge$	$\not\rightarrow$	x	$\not\leftarrow$	y	$\nleftrightarrow$	$\vee$
x	y	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

2-stellige Boolesche Funktionen II

(1)		$\overline{x+y}$		$\overline{y}$	$x + \overline{y}$	$\overline{x}$	$\overline{x} + y$	$\overline{x \cdot y}$	$x + \overline{x}$
(2)		1-Max	$=$	1-y	$\geq$	**1-x**	$\leq$	1-Min	$\equiv 1$
(3)		NOR	$\leftrightarrow$	$\neg y$	$\leftarrow$	$\neg x$	$\rightarrow$	NAND	
x	y	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Beispielfunktion

Sei $f : B^3 \to B$ gegeben durch

i	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Minterme (z.B.)

$m_3(x_1, x_2, x_3) = \overline{x}_1 \cdot x_2 \cdot x_3,$

$m_4(x_1, x_2, x_3) = x_1 \cdot \overline{x}_2 \cdot \overline{x}_3$

Darstellungssatz für Boolesche Funktionen

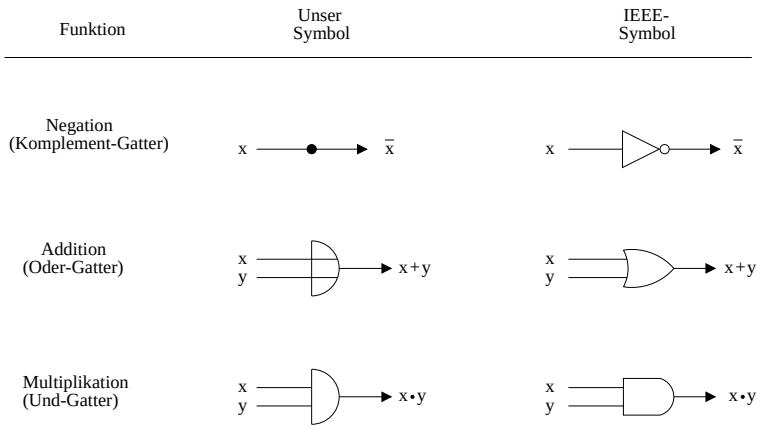
Jede Boolesche Funktion $f : B^n \to B$ ist eindeutig darstellbar als **Summe der Minterme ihrer einschlägigen Indizes**, d.h.

ist $I \subseteq \{0, \ldots, 2^n - 1\}$ die Menge der einschlägigen Indizes von f, so gilt

$$f = \sum_{i \in I} m_i,$$

und keine andere Minterm-Summe stellt f dar.

Grundbausteine zur Realisierung Boolescher Funktionen



Beispiel (Forts.)

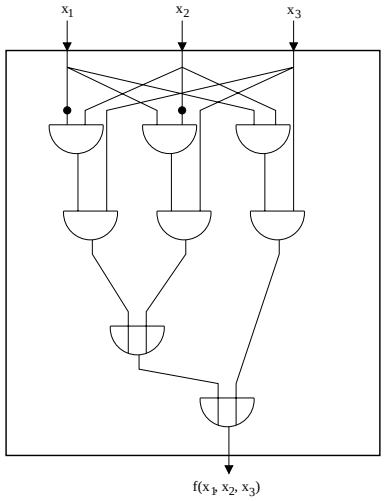
DNF:

$$\begin{aligned} f(x_1, x_2, x_3) &= m_3 + m_5 + m_7 \\ &= \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 x_3 + x_1 x_2 x_3 \end{aligned}$$

KNF:

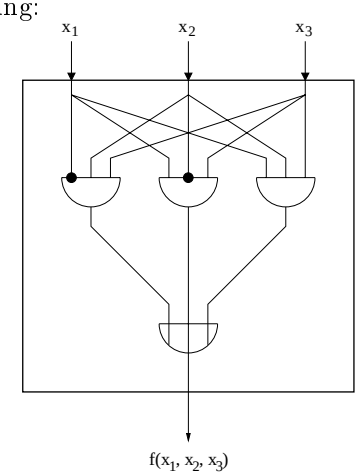
$$\begin{aligned} f(x_1 x_2 x_3) &= M_0 \cdot M_1 \cdot M_2 \cdot M_4 \cdot M_6 \\ &= (x_1 + x_2 + x_3) \cdot (x_1 + x_2 + \overline{x_3}) \cdot (x_1 + \overline{x_2} + x_3) \\ &\quad \cdot (\overline{x_1} + x_2 + x_3) \cdot (\overline{x_1} + \overline{x_2} + x_3) \end{aligned}$$

Beispiel (Forts.)

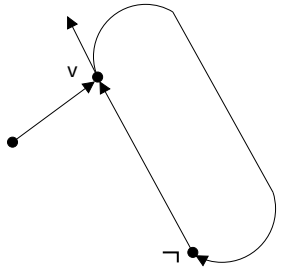


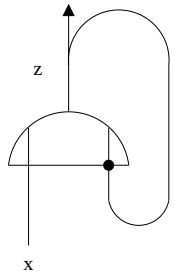
Beispiel (Forts.)

Alternative Schaltung:

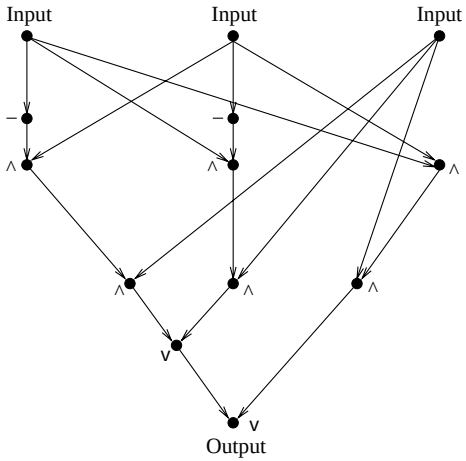


Flimmerschaltung





DAG-Darstellung



2. Spezifische Schaltnetze und ihre Verbesserung

- Entwurf von Schaltnetzen
- Multiplexer und verwandte Bausteine
- Addiernetze
- Vereinfachung von Schaltnetzen
- Fehlerdiagnose von Schaltnetzen

Realisierung auf Transistorebene

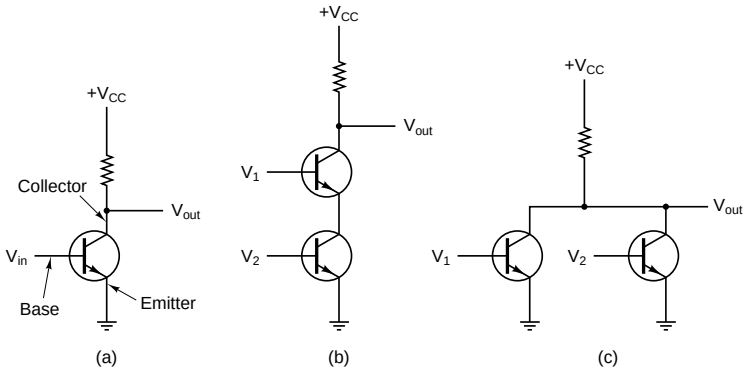
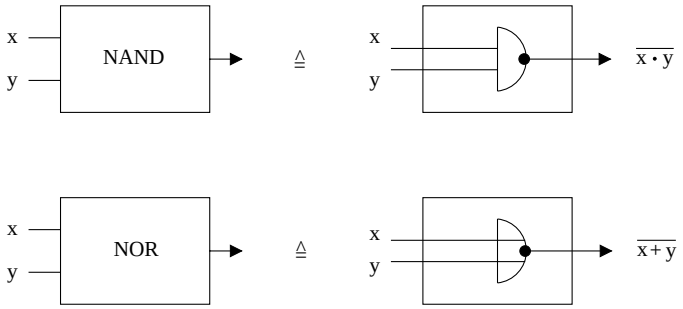


Figure 3-1. (a) A transistor inverter. (b) A NAND gate. (c) A NOR gate.

Nand und Nor



Satz: Nand und Nor sind funktional vollständig.

Realisierung der Funktion e (Euler-Kreis)

Ein Graph heißt **zusammenhängend**, falls jeder Knoten von jedem anderen durch einen Pfad erreichbar ist.

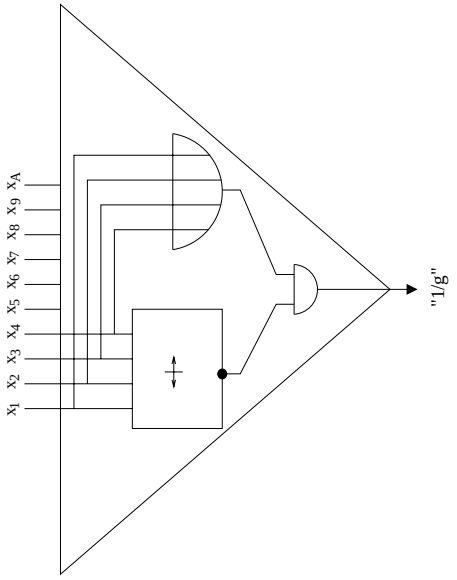
Der **Grad** eines Knoten stellt die Anzahl der von dem Knoten ausgehenden Kanten dar.

Satz:
Ein zusammenhängender Graph mit 5 Knoten hat einen Euler-Kreis gdw. jeder Knoten den Grad 2 oder 4 hat.

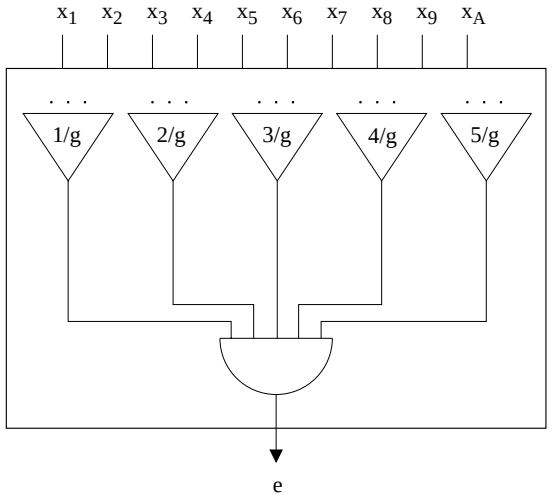
Test auf geraden Grad

"Knoten 1 hat Grad 2 oder 4" darstellbar als

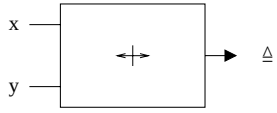
$$\overline{x_1 \leftrightarrow x_2 \leftrightarrow x_3 \leftrightarrow x_4} \wedge (x_1 \vee x_2 \vee x_3 \vee x_4)$$

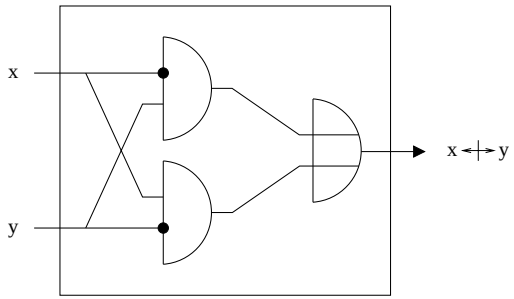


Gesamtes Schaltnetz für e

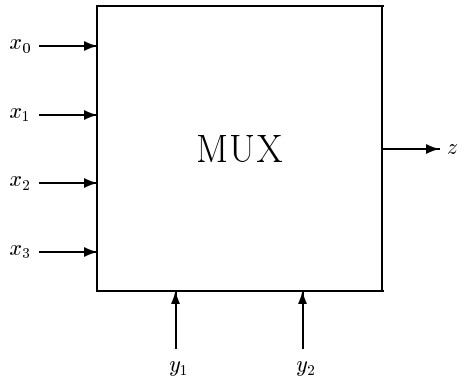


Xor



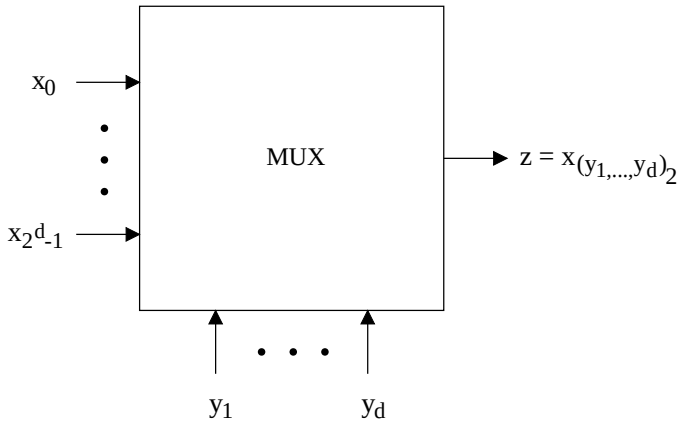


2-MUX (Prinzip)

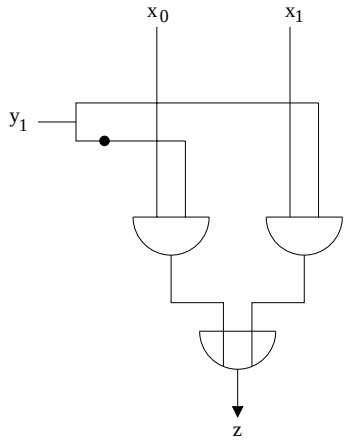


y_1	y_2	z
0	0	x_0
0	1	x_1
1	0	x_2
1	1	x_3

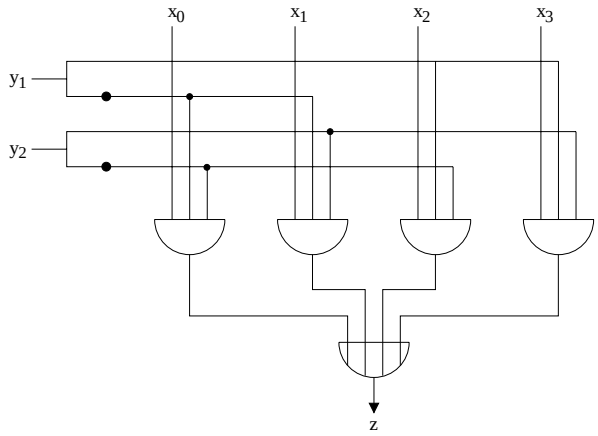
Allgemeiner MUX-Aufbau



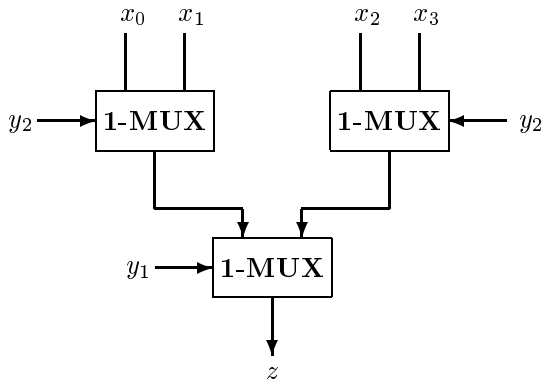
1-MUX



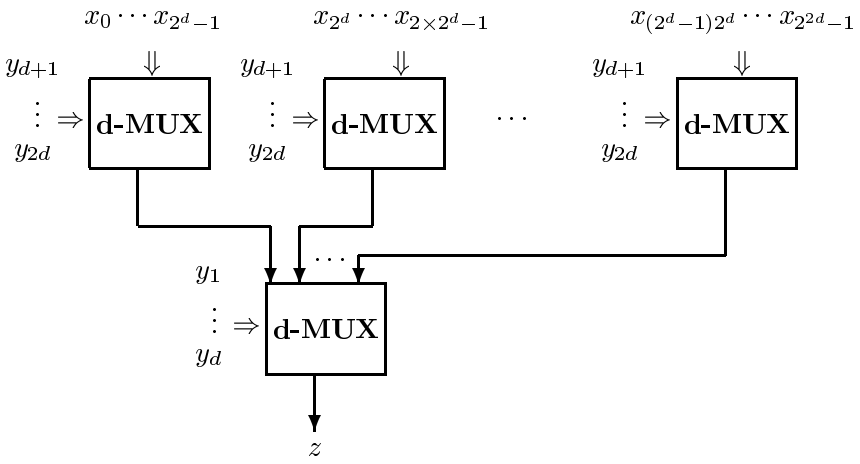
Realisierung eines 2-MUX



Systematische Konstruktion des 2-MUX

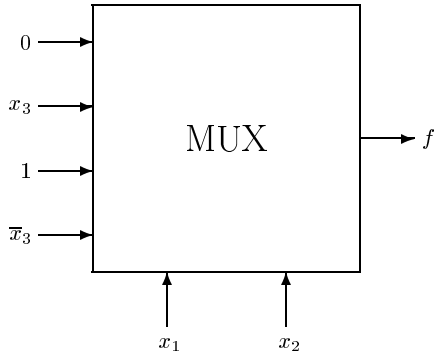


Top-Down-Multiplexer-Entwurf



Realisierung einer Funktion

f dargestellt in alleiniger Abhängigkeit von x_1 und x_2:



x_1	x_2	f
0	0	0
0	1	x_3
1	0	1
1	1	$\overline{x}_3$

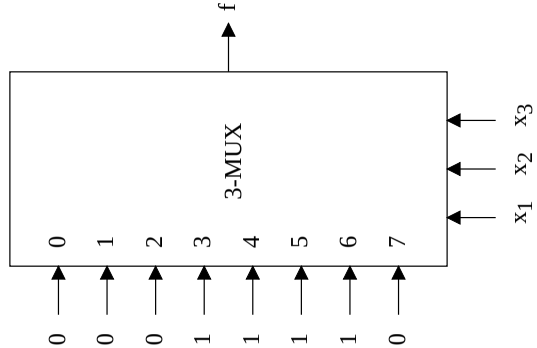
MUX zur Realisierung Boolescher Funktionen

Betrachte:

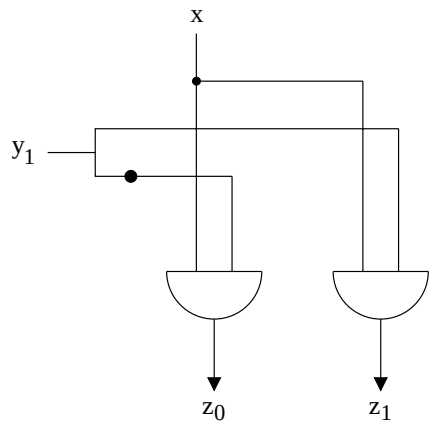
x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Hardware-Lookup

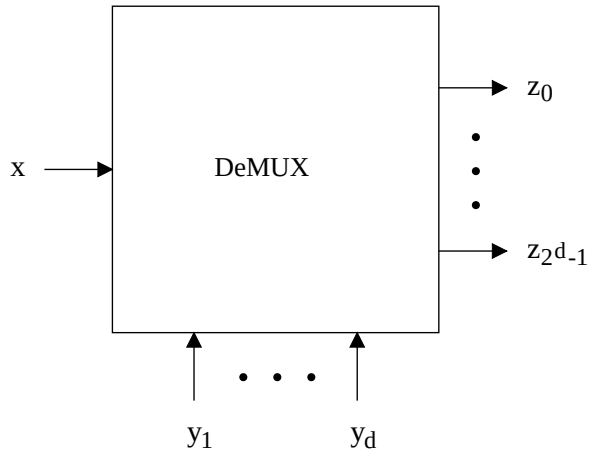
$f(x_1, x_2, x_3) = m_3 + m_4 + m_5 + m_6.$



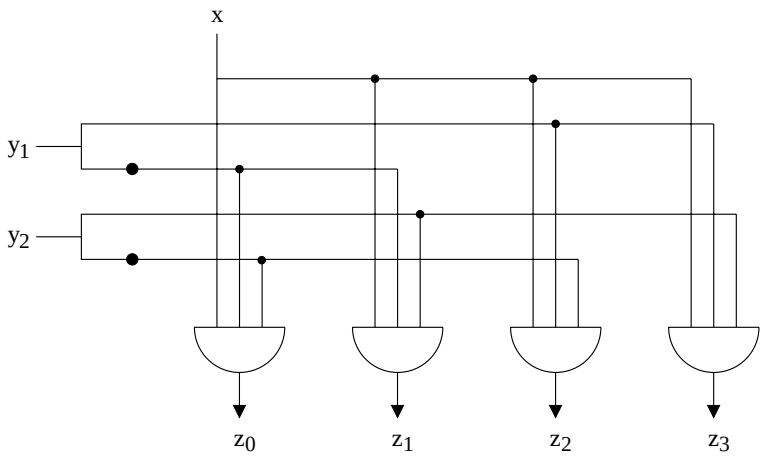
1-DeMUX



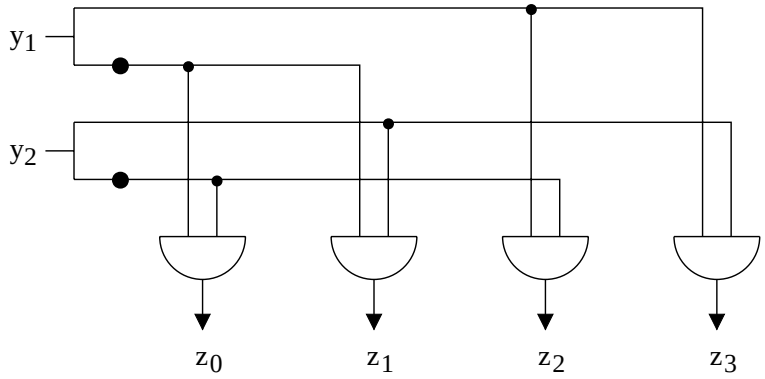
Allgemeiner Aufbau eines DeMUX



2-DeMUX

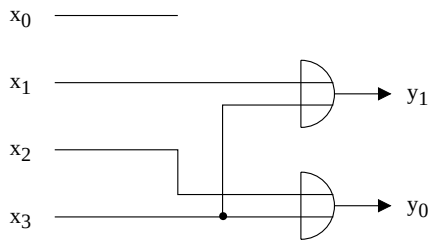


2×4-Decoder



4×2-Encoder

x_0	x_1	x_2	x_3	y_0	y_1
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1



Realisierung Boolescher Funktionen (1)

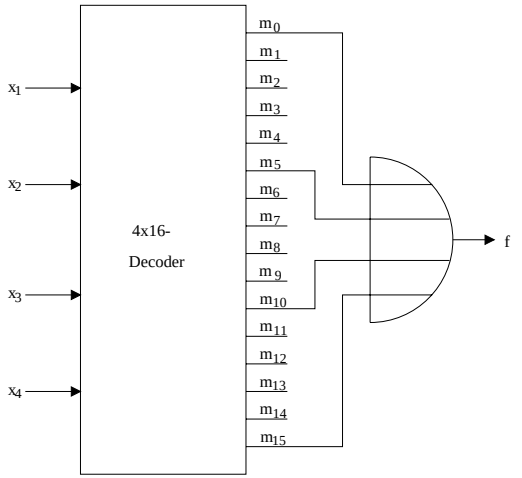
Beispiel:

$$f(x_1, x_2, x_3, x_4) = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1 x_2 \overline{x}_3 x_4 + x_1 x_2 x_3 x_4 + x_1 \overline{x}_2 x_3 \overline{x}_4$$

1. mittels MUX: siehe oben

Realisierung Boolescher Funktionen (2)

2. mittels Decoder:



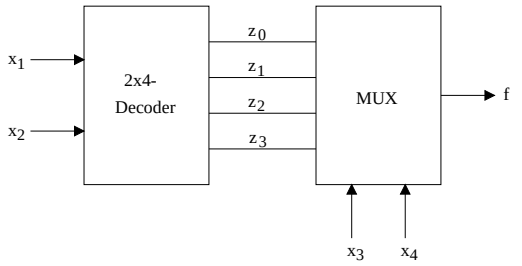
Realisierung Boolescher Funktionen (3)

3. mittels Kombination von Decoder und MUX:

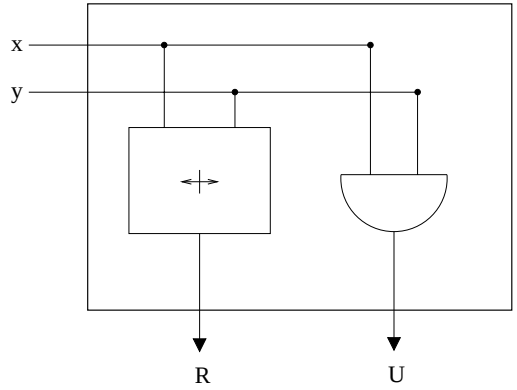
Es gibt 4 Input-Kombinationen, für welche $f = 1$ gilt:

$x_1x_2 = 00$ und $x_3x_4 = 00$, $x_1x_2 = 01$ und $x_3x_4 = 01$,

$x_1x_2 = 11$ und $x_3x_4 = 11$, $x_1x_2 = 10$ und $x_3x_4 = 10$.

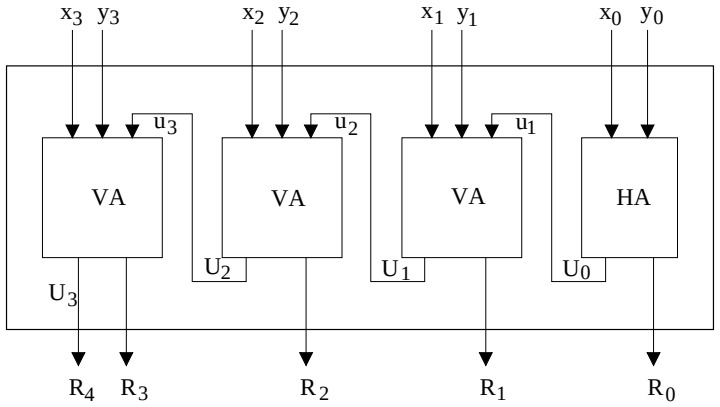


Halbaddierer

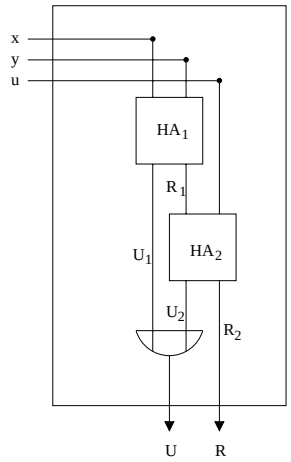


Addiernetz für zwei 4-stellige Dualzahlen

asynchrones (Parallel-) Addiernetz, Ripple-Carry-Adder:



Volladdierer



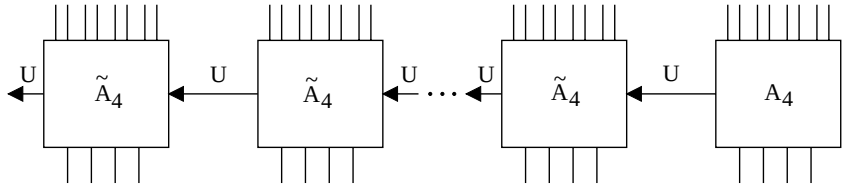
$U_1 = x \cdot y$

$R_1 = x \leftrightarrow y$

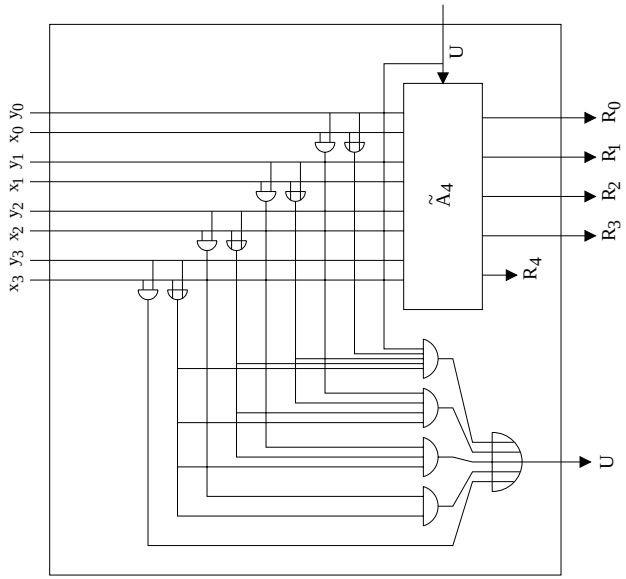
$U_2 = (x \leftrightarrow y) \cdot u$

$R_2 = (x \leftrightarrow y) \leftrightarrow u$

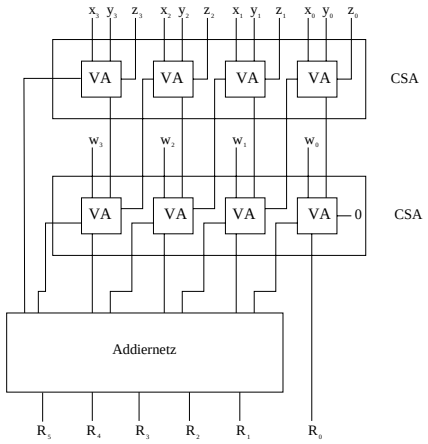
n-stelliges Addiernetz



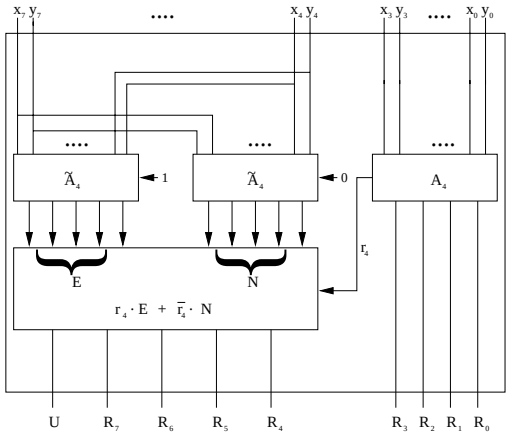
Carry-Bypass-Addiernetz



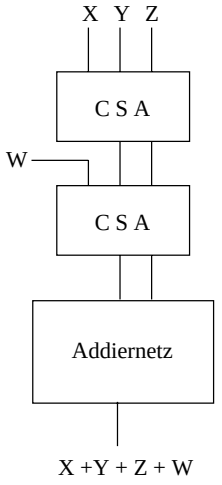
Carry-Save-Addiernetz



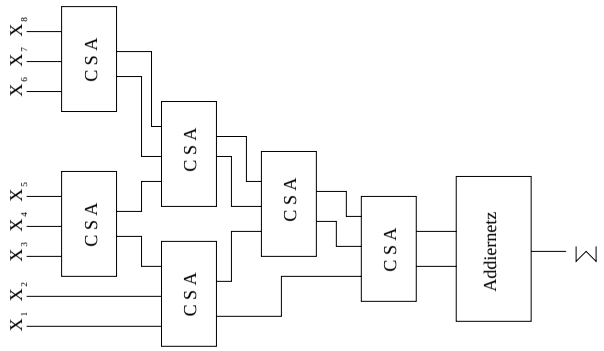
Carry-Select-Addiernetz



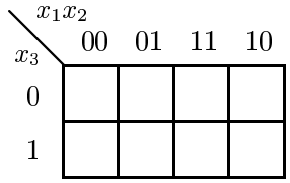
Prinzip der Carry-Save-Addition

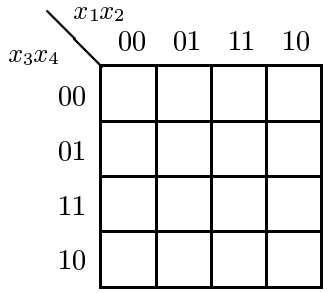


Wallace-Tree



Karnaugh-Diagramme für $n = 3, 4$





Vereinfachung Boolescher Funktionen

Beispiel 1:

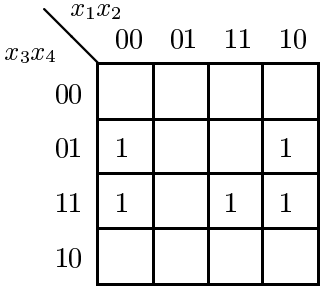
$$\begin{aligned} f(x_1, x_2, x_3) &= \overline{x}_1 x_2 x_3 + x_1 x_2 x_3 \\ &= \underbrace{(\overline{x}_1 + x_1)}_{=1} x_2 x_3 \\ &= x_2 x_3 \end{aligned}$$

Beispiel 2:

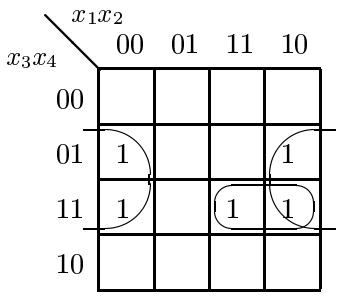
$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= x_1\overline{x}_2x_3x_4 + x_1\overline{x}_2\overline{x}_3x_4 + x_1x_2x_3x_4 + \overline{x}_1\overline{x}_2\overline{x}_3x_4 + \overline{x}_1\overline{x}_2x_3x_4 \\ &= x_1\overline{x}_2x_4 + x_1x_3x_4 + \overline{x}_2\overline{x}_3x_4 + \overline{x}_1\overline{x}_2x_4 \\ &= \overline{x}_2x_4 + x_1x_3x_4 + \overline{x}_2\overline{x}_3x_4 \end{aligned}$$

Beispiel

$$f = x_1\overline{x}_2x_3x_4 + x_1\overline{x}_2\overline{x}_3x_4 + x_1x_2x_3x_4 + \overline{x}_1\overline{x}_2\overline{x}_3x_4 + \overline{x}_1\overline{x}_2x_3x_4$$



Beispiel (Forts.)



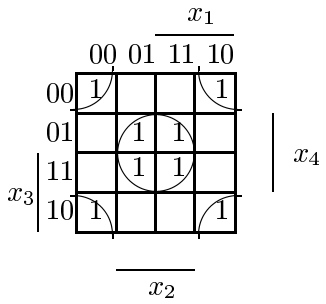
Ergebnis: $f = \overline{x}_2x_4 + x_1x_3x_4$

Ausnutzung von *Don't Cares*

x	x_1	x_2	x_3	x_4	f
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	D
B	1	0	1	1	D
C	1	1	0	0	D
D	1	1	0	1	D
E	1	1	1	0	D
F	1	1	1	1	D

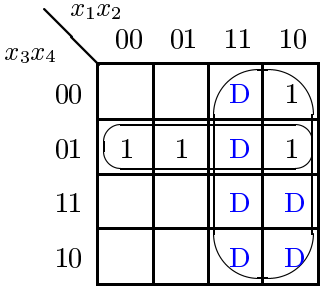
Weiteres Beispiel

$$f(x_1,x_2,x_3,x_4) = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3x_4 + x_1x_2\overline{x}_3x_4 + \overline{x}_1x_2x_3x_4 + x_1x_2x_3x_4 + \overline{x}_1\overline{x}_2x_3\overline{x}_4 + x_1\overline{x}_2x_3\overline{x}_4$$



Vereinfachte Form: $f = x_2x_4 + \overline{x}_2\overline{x}_4$

Ausnutzung von *Don't Cares* (Forts.)



Ergebnis: $f(x_1,x_2,x_3,x_4) = x_1 + \overline{x}_3x_4$

Bsp. zum Quine/McCluskey-Verfahren (1)

$$f = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2x_3\overline{x}_4 + x_1\overline{x}_2x_3x_4 + x_1x_2\overline{x}_3\overline{x}_4 + x_1x_2\overline{x}_3x_4 + x_1x_2x_3\overline{x}_4$$

Minterme gemäß Anzahl der Negationen:

Gruppe	Minterm	einschlägiger Index	Dezimaldarstellung des Index
1	$x_1\overline{x}_2x_3x_4$	1011	11
	$x_1x_2\overline{x}_3x_4$	1101	13
	$x_1x_2x_3\overline{x}_4$	1110	14
2	$\overline{x}_1x_2x_3\overline{x}_4$	0110	6
	$x_1x_2\overline{x}_3\overline{x}_4$	1100	12
3	$\overline{x}_1x_2\overline{x}_3\overline{x}_4$	0100	4
4	$\overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4$	0000	0

Bsp. zum Quine/McCluskey-Verfahren (2)

Nach (erster) Anwendung der Resolutionsregel:

Gruppe	Implikant	Index	Minterm-Nummern
1	$x_1\overline{x}_2x_3x_4$	1011	11
	$x_2x_3\overline{x}_4$	*110	6, 14
	$x_1x_2\overline{x}_3$	110*	12, 13
	$x_1x_2\overline{x}_4$	11*0	12, 14
2	$\overline{x}_1x_2\overline{x}_4$	01*0	4, 6
	$x_2\overline{x}_3\overline{x}_4$	*100	4, 12
3	$\overline{x}_1\overline{x}_3\overline{x}_4$	0*00	0, 4

Alle Primimplikanten:

Gruppe	Implikant	Index	Minterm-Nummern
1	$x_1\overline{x}_2x_3x_4$	1011	11
	$x_1x_2\overline{x}_3$	110*	12, 13
	$x_2\overline{x}_4$	*1*0	4, 6, 12, 14
3	$\overline{x}_1\overline{x}_3\overline{x}_4$	0*00	0, 4

Bsp. zum Quine/McCluskey-Verfahren (3)

Implikationsmatrix:

Primimplikant \ Minterm	0	4	6	11	12	13	14
$x_1\overline{x}_2x_3x_4$	0	0	0	1	0	0	0
$x_1x_2\overline{x}_3$	0	0	0	0	1	1	0
$x_2\overline{x}_4$	0	1	1	0	1	0	1
$\overline{x}_1\overline{x}_3\overline{x}_4$	1	1	0	0	0	0	0

Kostengünstigste Darstellung:

$$x_1\overline{x}_2x_3x_4 + x_1x_2\overline{x}_3 + x_2\overline{x}_4 + \overline{x}_1\overline{x}_3\overline{x}_4$$

Schaltungsabhängige Fehlerdiagnose

Beispiel:

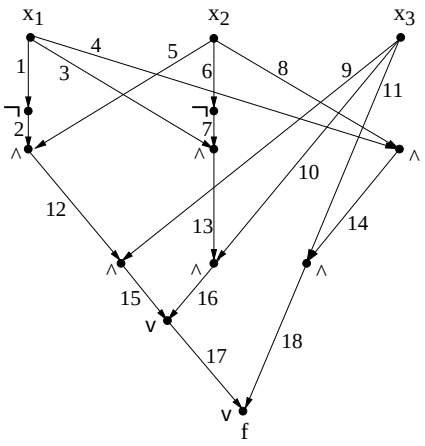
$f(x_1, x_2, x_3) = \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3 + x_1x_2x_3$

Annahmen:

- Es tritt im gegebenen Schaltnetz höchstens ein Fehler auf;
- der Defekt, welcher einen Fehler verursacht, ist ein gerissener Verbindungsdraht.

Hier: **0-Verklemmung** bzw. **Stuck-at-Zero-Fault**

DAG mit Draht-Nummern



Fehlermöglichkeiten (Ausfalltafel)

x_1	x_2	x_3	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	1	1	0	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1	1	0	1	1
1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	1	1	1	0	1

x_1	x_2	x_3	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	1	1	0	1	0	1
1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	1	0	1	1	0	0	1
1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	1	1	0	1	1	1	0

Darstellungen der f_i

$$
\begin{aligned}
f_1 &= \overline{0} \cdot x_2x_3 + x_1\overline{x}_2x_3 + x_1x_2x_3 = x_2x_3 + x_1x_3\\
f_2 &= 0 \cdot x_2x_3 + x_1\overline{x}_2x_3 + x_1x_2x_3 = x_1\overline{x}_2x_3 + x_1x_2x_3 = x_1x_3\\
f_3 &= \overline{x}_1x_2x_3 + x_1x_2x_3 = x_2x_3\\
f_4 &= \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3\\
f_5 &= x_1\overline{x}_2x_3 + x_1x_2x_3 = x_1x_3\\
f_6 &= \overline{x}_1x_2x_3 + x_1x_3\\
f_7 &= x_2x_3\\
f_8 &= \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3\\
f_9 &= x_1x_3\\
f_{10} &= x_2x_3\\
f_{11} &= \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3\\
f_{12} &= 0 \cdot x_3 + x_1\overline{x}_2x_3 + x_1x_2x_3 = x_1x_3\\
f_{13} &= x_2x_3\\
f_{14} &= \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3\\
f_{15} &= x_1x_3\\
f_{16} &= x_2x_3\\
f_{17} &= x_1x_2x_3\\
f_{18} &= \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3
\end{aligned}
$$

Ausfallmatrix

Es gilt:

$$
\begin{aligned}
f_1 &= f_6\\
f_2 &= f_5 = f_9 = f_{12} = f_{15}\\
f_3 &= f_7 = f_{10} = f_{13} = f_{16}\\
f_4 &= f_8 = f_{11} = f_{14} = f_{18}
\end{aligned}
$$

x_1	x_2	x_3	f_1	f_2	f_3	f_4	f_{17}
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	1	0	1	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	0
1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	1

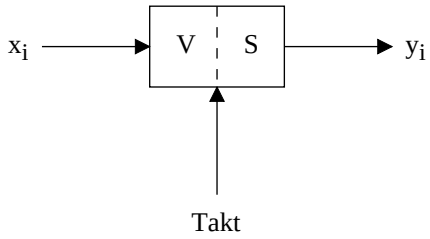
Fehlermatrix

Zeilen-Nr.	x_1	x_2	x_3	$f \nleftrightarrow f_1$	$f \nleftrightarrow f_2$	$f \nleftrightarrow f_3$	$f \nleftrightarrow f_4$	$f \nleftrightarrow f_{17}$
0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	0	1	1	0	1	0	0	1
4	1	0	0	0	0	0	0	0
5	1	0	1	0	0	1	0	1
6	1	1	0	0	0	0	0	0
7	1	1	1	0	0	0	1	0

4. Schaltungen mit Delays (Schaltwerke)

- Delays für die Taktung von Schaltnetzen
- Addierwerke
- Latches und Flip-Flops
- (Halbleiter-)Speicher

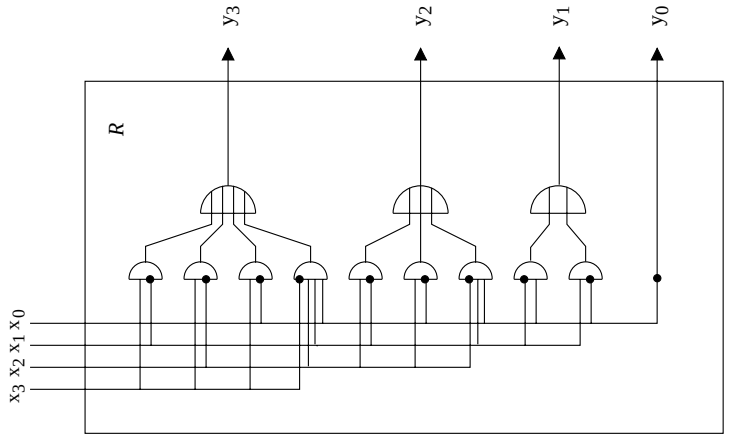
Delay



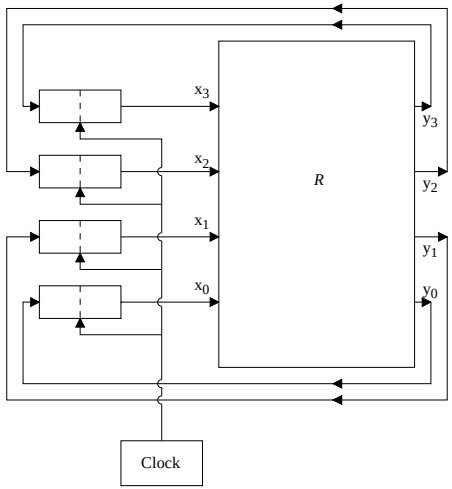
(1) **Arbeitsphase**: Der Inhalt von S wird „nach rechts“ abgegeben; er steht als Signal y_i zur Verfügung. Ein Signal x_i wird in V „abgelegt“. V und S sind durch eine Sperre getrennt.

(2) **Setzphase**: Eine zentrale Synchronisation (die Clock, welche Taktimpulse erzeugt) hebt die Sperre kurzzeitig auf und bewirkt dadurch die Abgabe des Inhalts von V an S.

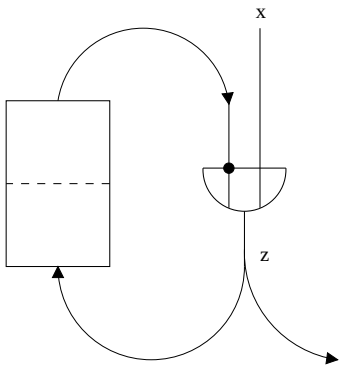
4-Bit-Ringzähler



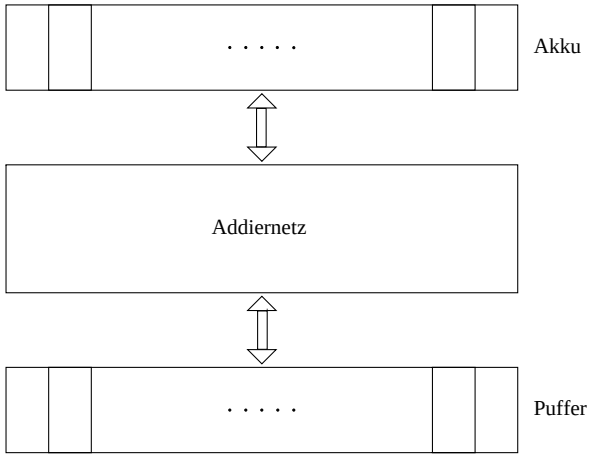
4-Bit-Ringzähler mit Delays



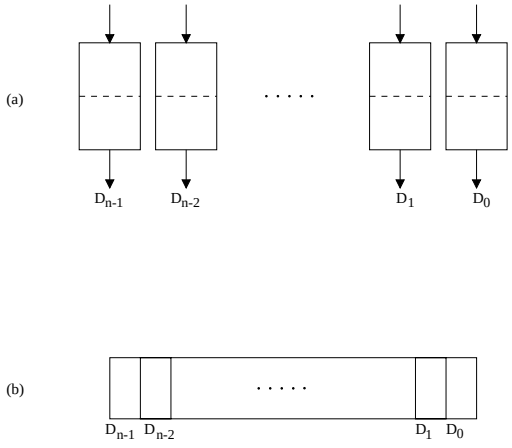
Flimmerschaltung (ok mit Delay)



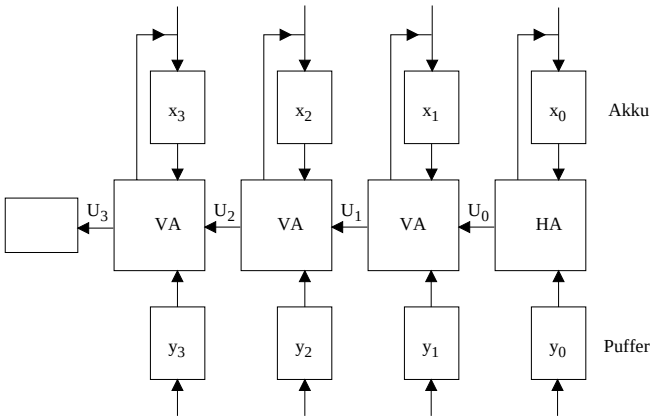
Addierwerk (Organisationsplan)



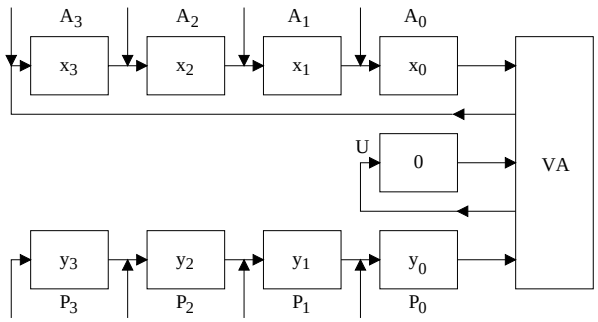
n-stelliges Register



4-Bit-Parallel-Addierwerk (Ripple-Carry-Adder mit Delays)



4-Bit-Serien-Addierer

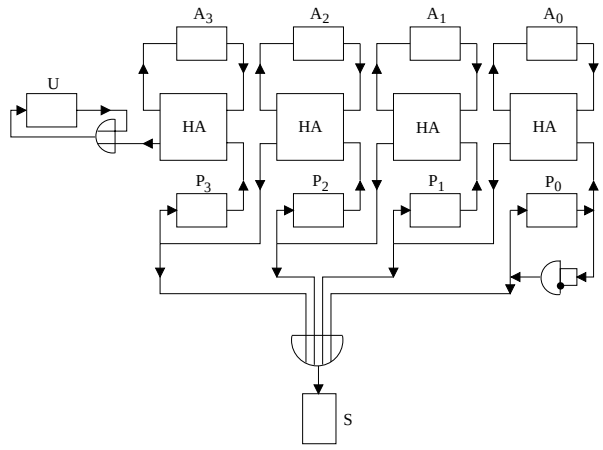


Arbeitsweise (Beispiel)

Aufgaben: 13+11, 10+12, 15+15, 9+10, 0+0

Zeile	S	U	Puffer-Inhalt dual $P_3P_2P_1P_0$	Puffer-Inhalt dezimal	Akku-Inhalt dual $A_3A_2A_1A_0$	Akku-Inhalt dezimal	Schritt
1	0	0	0000	0	0000	0	
2	1	0	1101	13	1011	11	1
3	1	1	0010	2	0110	22	2
4	1	1	0100	4	0100	20	3
5	1	1	1000	8	0000	16	4
6	0	1	0000	0	1000	24	5
7	1	0	1010	10	1100	12	1
8	0	1	0000	0	0110	22	2
9	1	0	1111	15	1111	15	1
10	1	1	1110	14	0000	16	2
11	0	1	0000	0	1110	30	3
12	1	0	1001	9	1010	10	1
13	0	1	0000	0	0011	19	2
14	1	0	0000	0	0000	0	1
15	0	0	0000	0	0000	0	2

4-Bit-von Neumann-Addierwerk

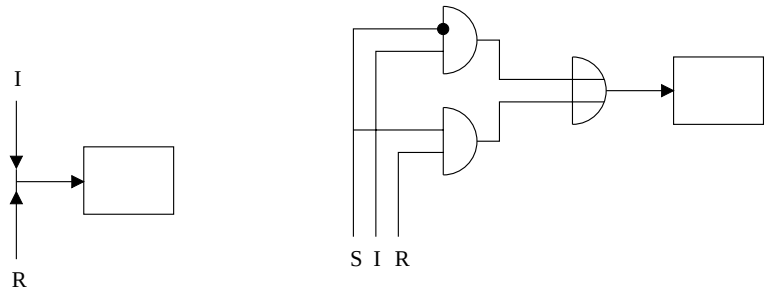


Fan-In-Problem bei Delay-Eingängen (1)

$$f(S,I,R) := \begin{cases} I & \text{falls } S=0 \\ R & \text{falls } S=1 \end{cases}$$

S	I	R	f	Output
0	0	0	0	I
0	0	1	0	I
0	1	0	1	I
0	1	1	1	I
1	0	0	0	R
1	0	1	1	R
1	1	0	0	R
1	1	1	1	R

Fan-In-Problem bei Delay-Eingängen (2)



(a) (b)

Clocks

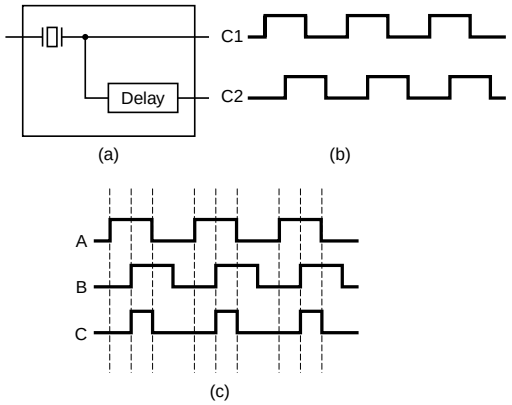
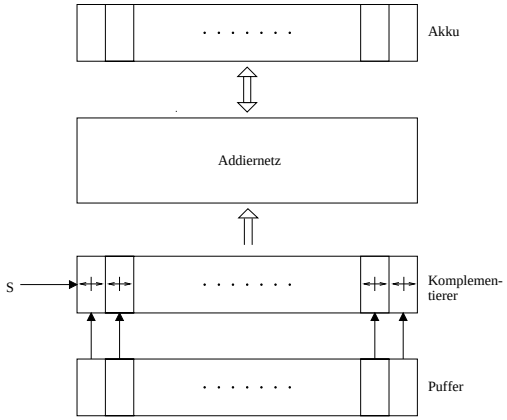
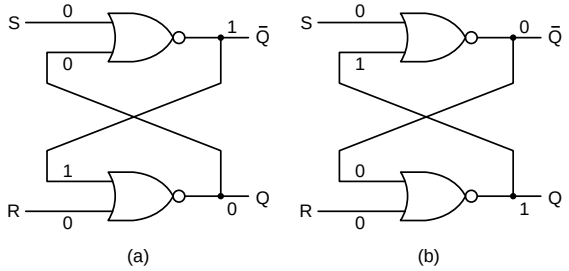


Figure 3-21. (a) A clock. (b) The timing diagram for the clock. (c) Generation of an asymmetric clock.

Organisation eines kombinierten Addier-/Subtrahierwerks



Prinzip des NOR-Latches



A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Figure 3-22. (a) NOR latch in state 0. (b) NOR latch in state 1. (c) Truth table for NOR.

Getaktetes SR Latch

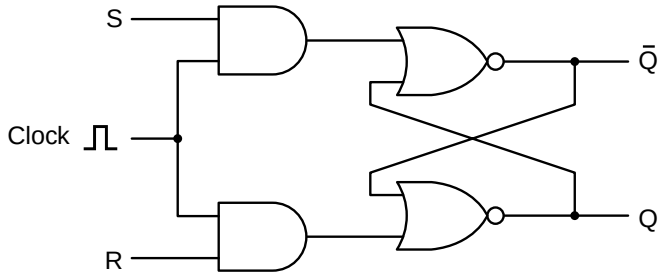


Figure 3-23. A clocked SR latch.

Prinzip eines Pulsgenerators

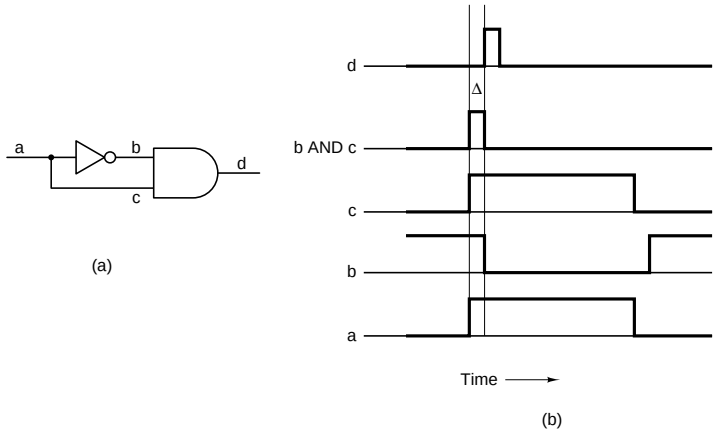


Figure 3-25. (a) A pulse generator. (b) Timing at four points in the circuit.

Getaktetes D Latch

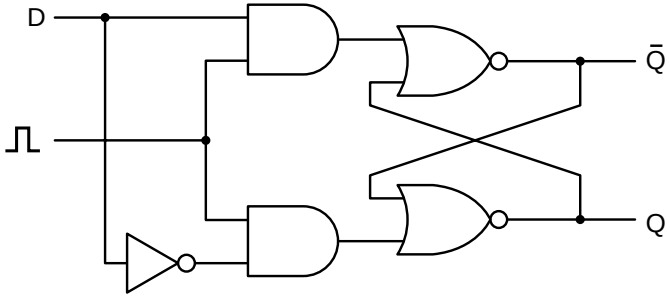


Figure 3-24. A clocked D latch.

D Flip-Flop

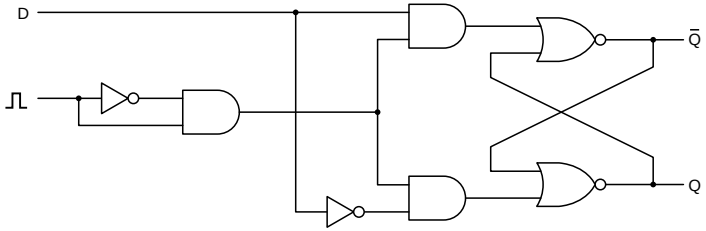


Figure 3-26. A D flip-flop.

D Latches und Flip-Flops

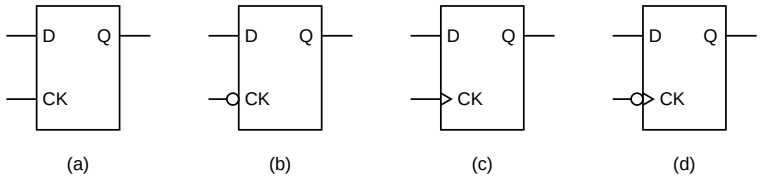


Figure 3-27. D latches and flip-flops.

4×3 Speicher

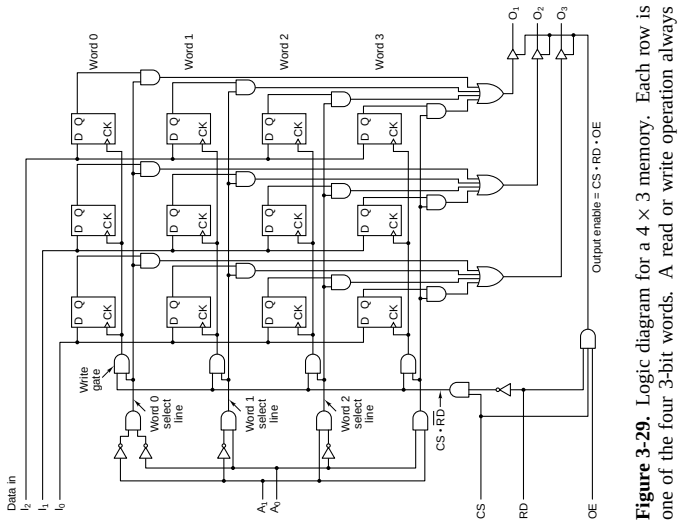


Figure 3-29. Logic diagram for a 4×3 memory. Each row is one of the four 3-bit words. A read or write operation always reads or writes a complete word.

Zwei D-Flip-Flops und ein 8-bit Register

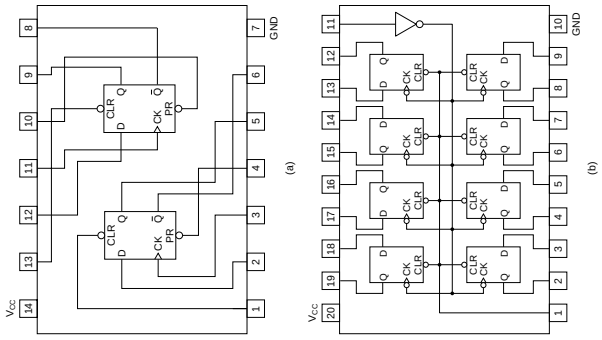


Figure 3-28. (a) Dual D flip-flop. (b) Octal flip-flop.

(Invertierender) Schalter

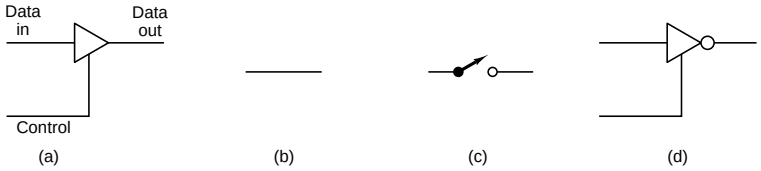


Figure 3-30. (a) A noninverting buffer. (b) Effect of (a) when control is high. (c) Effect of (a) when control is low. (d) An inverting buffer.

Organisation eines 4Mbit Speicherchips

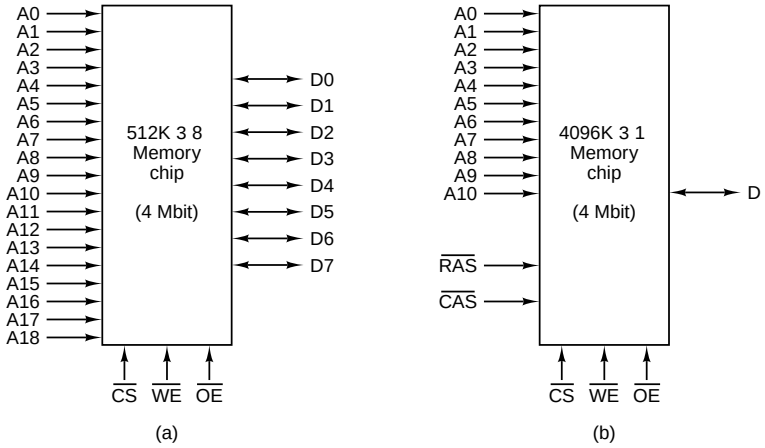
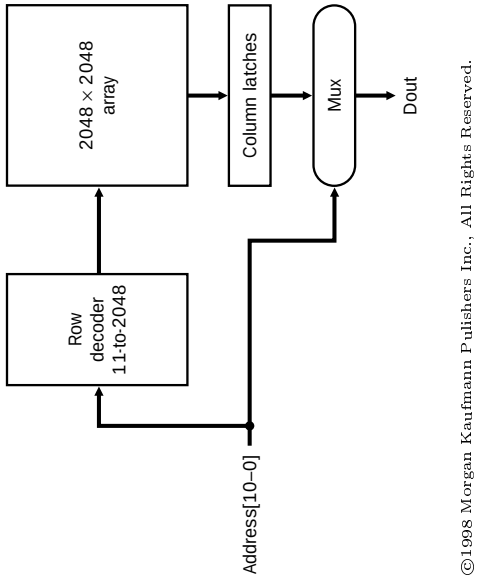


Figure 3-31. Two ways of organizing a 4-Mbit memory chip.

Prinzip eines 4MBit DRAM Speichers



RAM-Speicher

RAM = **R**andom **A**ccess **M**emory

- SRAM (**S**tatic RAM)
  - Verwendung von Flip-Flops/Latches (wie Folie "4×3 Speicher")
  - sehr schnell (<10ns Zugriffszeit)
  - Verwendung als **2-level Cache**.
- DRAM (**D**ynamic RAM)
  - Pro Speicherzelle 1 Transistor und 1 Kondensator
  - dadurch hohe Speicherdichte und geringe Kosten
  - Speicherinhalte müssen "aufgefrischt" werden
  - langsamer als SRAM (ca. 50ns Zugriffszeit)
  - Verwendung als **Hauptspeicher**

5. Darstellung von Zahlen/Zeichen im Rechner

- Darstellung ganzer Zahlen, insbes. Komplementdarstellungen
- Darstellung von Gleitkomma-Zahlen
- Multiplikation
- Darstellung alphanumerischer Daten

Komplementdarstellungen

Sei $x = (x_{n-1}, \ldots, x_0)_2 \in B^n$ eine n-stellige Dualzahl.

(i) $K_1(x) := (1 \nleftrightarrow x_{n-1}, \ldots, 1 \nleftrightarrow x_0)_2$ heißt *Einer-Komplement* von x.

(ii) $K_2(x) := (1 \nleftrightarrow x_{n-1}, \ldots, 1 \nleftrightarrow x_0)_2 + 1 = K_1(x) + 1$ (modulo 2^n) heißt *Zweier-Komplement* von x.

Beispiel: $x = 10110010$:

$$K_1(x) = 01001101$$
$$K_2(x) = 01001110$$

Darstellung ganzer Zahlen

Bisher wurden nur natürliche Zahlen betrachtet. Wie sollte man ganze Zahlen darstellen?

Der konzeptionell einfachste Ansatz:

Dualdarstellung wie bisher plus Vorzeichen-Bit
$0 = +$, $1 = -$, Bsp. $+5 = 0101$, $-5 = 1101$

Nachteile: 0 hat zwei Darstellungen und man braucht sowohl Addier- als auch Subtrahierwerk:

1.	$+x, +y$	$x + y$	Add.
2.	$-x, -y$	$-(x + y)$	Add.
3.	$+x, -y$ $(x \geq y)$	$x - y$	Subtr.
	$-x, +y$ $(y \geq x)$	$y - x$	Subtr.
4.	$+x, -y$ $(x < y)$	$-(y - x)$	Subtr.
	$-x, +y$ $(y < x)$	$-(x - y)$	Subtr.

Alternative Darstellungen ganzer Zahlen

Bitfolge	Darstellung in Dezimalnotation Vorz./Betrag	K_1	K_2
0000	+0	+0	0
0001	+1	+1	+1
0010	+2	+2	+2
0011	+3	+3	+3
0100	+4	+4	+4
0101	+5	+5	+5
0110	+6	+6	+6
0111	+7	+7	+7
1000	−0	−7	−8
1001	−1	−6	−7
1010	−2	−5	−6
1011	−3	−4	−5
1100	−4	−3	−4
1101	−5	−2	−3
1110	−6	−1	−2
1111	−7	−0	−1

BCD-Code

Binary **C**oded **D**ecimal

Ziffern werden einzeln mit 4 Bits als Dualzahl dargestellt.

Beispiel: 4739 = 0100 0111 0011 1001

6 Bitmuster ungenutzt, erlaubt Darstellung der Vorzeichen:
z.Bsp. + = 1010 und − = 1011

Nachteil: erschwerte Addition.

Beispiel: 4739 + 1287 = 6026

0100	0111	0011	1001
0001	0010	1000	0111
0101	1001	1100	0000

z.Bsp. Fehler bei der letzten Stelle wegen des Übertrags.

Festkomma-Darstellung

- Komma rechts von der Stelle mit dem niedrigsten Wert:
  Ein n-Bit Wort $(x_{n-1},\ldots,x_0)_2$ stellt dann die Zahl
  $$z = \sum_{i=0}^{n-1} x_i \cdot 2^i \quad \text{dar.}$$
- Komma links von der Stelle mit dem höchsten Wert:
  Ein n-Bit Wort $(x_1,\ldots,x_n)_2$ stellt dann die Zahl
  $$z = \sum_{i=1}^{n} x_i \cdot 2^{-i} \quad \text{dar.}$$
- Allgemein stellt eine Bitfolge $(x_{n-1},\ldots,x_1,x_0,x_{-1},\ldots,x_{-m+1},x_{-m})_2$, falls der Punkt rechts von der Stelle x_0 angenommen wird, die Zahl
  $$x = \sum_{i=-m}^{n-1} x_i 2^i \quad \text{dar.}$$

BCD-Code (2)

Korrektur: 6 aufaddieren bei jedem Übertrag und bei jeder ungültigen BCD-Darstellung:

0101	1001	1100	0000
			0110
0101	1001	1100	0110
		0110	
0101	1010	0010	0110
	0110		
0110	0000	0010	0110
6	0	2	6

Gleitkomma-Darstellung

- Jede Zahl z wird in der Form $z = \pm m \times b^{\pm d}$ dargestellt mit m *Mantisse*, d *Exponent*, b Basis für den Exponenten.
- Die Basis ist für alle auftretenden Exponenten die gleiche; daher rechnerinterne Darstellung einer Gleitkomma-Zahl:
  $$(\pm m\ ,\ \pm d)$$
- Eine Gleitkomma-Zahl der Form $\pm m \cdot b^{\pm d}$ heißt *normalisiert*, falls gilt:
  $$\frac{1}{b} \le |m| < 1$$
  Im Fall $b = 2$ (als Basis für Exponent *und* Mantisse) folgt hieraus unmittelbar
  $$\frac{1}{2} \le |m| < 1$$

Multiplikation

Schulmethode:

Sei x der Multiplikand, $y = (y_{n-1}, \ldots, y_0)$ der Multiplikator, dann ist

$$\begin{aligned} x \cdot y &= x \cdot y_0 + x \cdot y_1 \cdot 2 + x \cdot y_2 \cdot 2^2 + \ldots + x \cdot y_{n-1} \cdot 2^{n-1} \\ &= \sum_{i=0}^{n-1} x \cdot y_i \cdot 2^i \end{aligned}$$

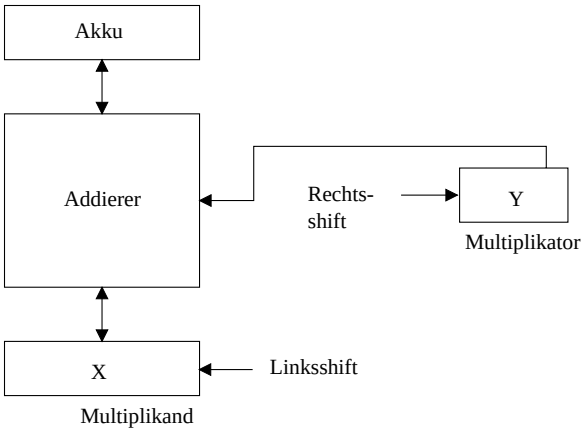
In der Praxis ist es sinnvoll, jeden Term der Form $x \cdot y_i \cdot 2^i$ zu addieren, sobald er generiert wurde:

$$x \cdot y = (\ldots((x \cdot y_0 + x \cdot y_1 \cdot 2) + x \cdot y_2 \cdot 2^2) \ldots + x \cdot y_{n-1} \cdot 2^{n-1})$$

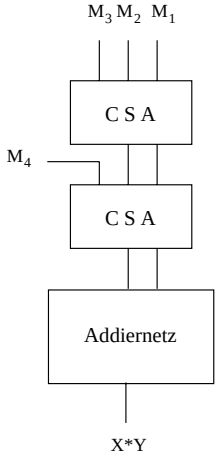
Carry-Save-Multiplikation (1)

				x_3	x_2	x_1	x_0	x
				$\times\ y_3$	y_2	y_1	y_0	y
0	0	0	0	x_3y_0	x_2y_0	x_1y_0	x_0y_0	M_1
0	0	0	x_3y_1	x_2y_1	x_1y_1	x_0y_1	0	M_2
0	0	x_3y_2	x_2y_2	x_1y_2	x_0y_2	0	0	M_3
0	x_3y_3	x_2y_3	x_1y_3	x_0y_3	0	0	0	M_4

Schaltung zur Multiplikation



Carry-Save-Multiplikation (2)



Regeln für das Rechnen mit Gleitkomma-Zahlen

Seien $x = m_x \cdot 2^{d_x}$, $y = m_y \cdot 2^{d_y}$:

$$
\begin{aligned}
x + y &= (m_x \cdot 2^{d_x - d_y} + m_y) \cdot 2^{d_y} \text{ falls } d_x \leq d_y \\
x - y &= (m_x \cdot 2^{d_x - d_y} - m_y) \cdot 2^{d_y} \text{ falls } d_x \leq d_y \\
x \cdot y &= (m_x \cdot m_y) \cdot 2^{d_x + d_y} \\
x : y &= (m_x : m_y) \cdot 2^{d_x - d_y}
\end{aligned}
$$

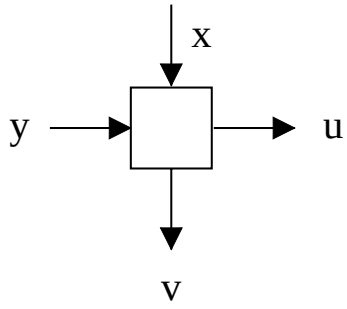
8-Bit ASCII Code

Bits 7654 3210	P000	P001	P010	P011	P100	P101	P110	P111
0000	NULL	DC_0		0	@	P	‘	p
0001	SOM	DC_1	!	1	A	Q	a	q
0010	EOA	DC_2	“	2	B	R	b	r
0011	EOM	DC_3	#	3	C	S	c	s
0100	EOT	DC_4	$	4	D	T	d	t
0101	WRU	ERR	%	5	E	U	e	u
0110	RU	SYNC	&	6	F	V	f	v
0111	BELL	LEM	’	7	G	W	g	w
1000	FE	S_0	(	8	H	X	h	x
1001	HT/SK	S_1	)	9	I	Y	i	y
1010	LF	S_2	*	:	J	Z	j	z
1011	V/TAB	S_3	+	;	K	[	k	
1100	FF	S_4	,	<	L	\	l	ACK
1101	CR	S_5	–	=	M	]	m	UC
1110	SO	S_6	.	>	N	↑	n	ESC
1111	SI	S_7	/	?	O	←	o	DEL

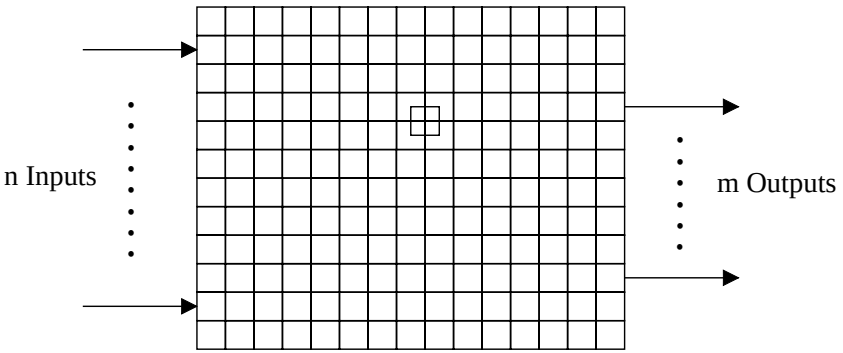
6. Programmierbare Logische Arrays (PLAs)

- Aufbau von PLAs
- Programmierung von PLAs
- Faltung von PLAs
- Anwendung 1: ROMs
- Anwendung 2: Mikroprogrammierung

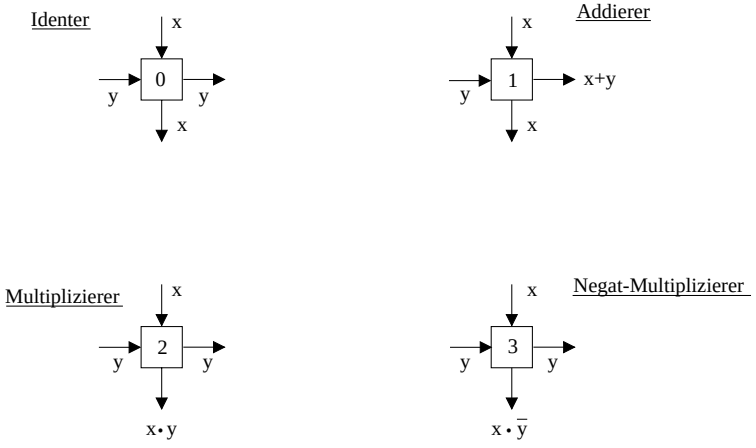
Ein Gitterpunkt



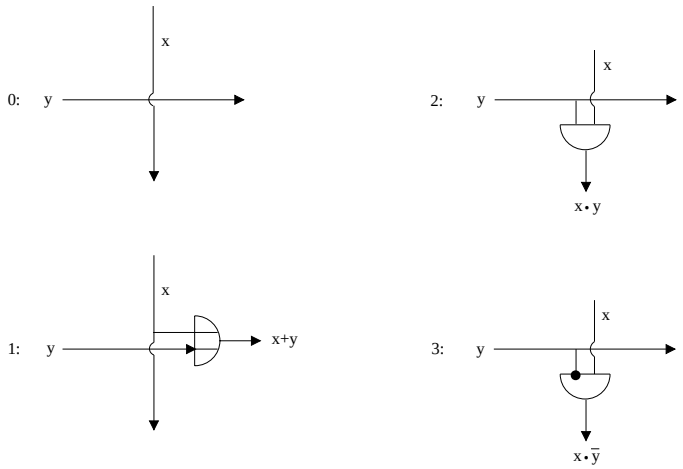
Aufbau eines PLAs



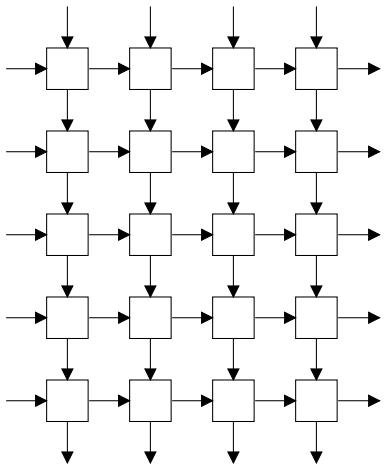
Bausteintypen



Realisierung der Bausteintypen



PLA-Schema zum Beispiel



Ein Beispiel

- $n = 5$ Inputs an der linken Seite
- $m = 5$ Outputs an der rechten Seite
- $k = 4$ Spalten

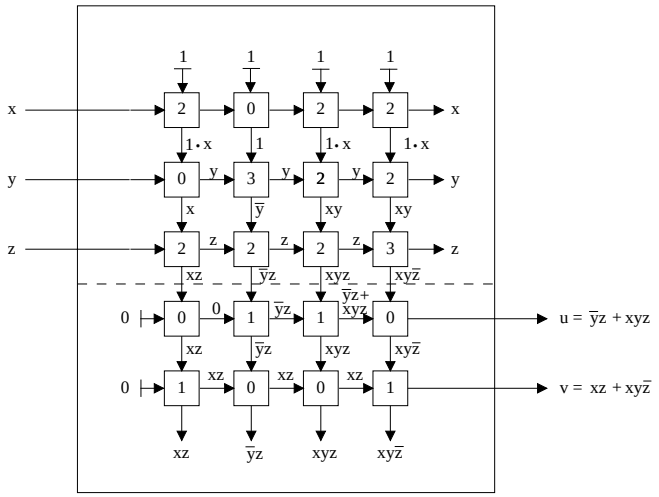
Es soll

$$F : B^3 \to B^2, \text{ definiert durch}$$

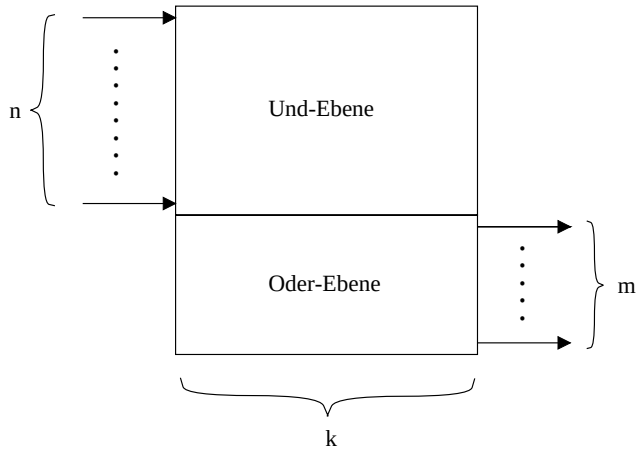
$$F(x, y, z) := (\underbrace{\overline{y}z + xyz}_{u}, \underbrace{xz + xy\overline{z}}_{v})$$

realisiert werden.

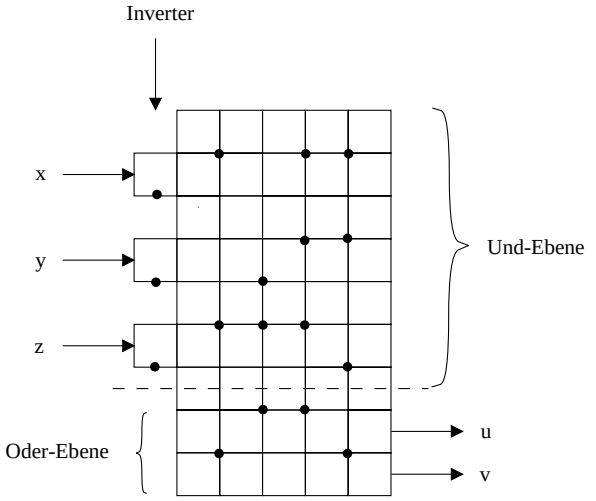
Realisierung



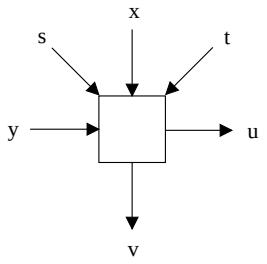
Allgemeiner PLA-Aufbau



Punkt-orientierte PLA-Darstellung



Zur Programmierung von PLAs

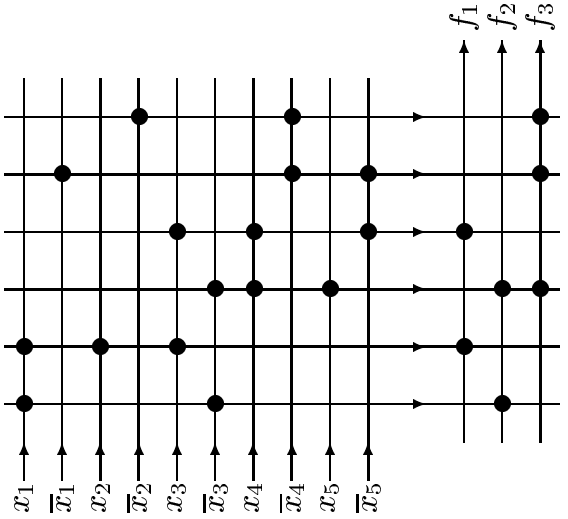


Baustein-Typ	s	t	v	u
0	0	0	x	y
1	0	1	x	$x+y$
2	1	0	$x \cdot y$	y
3	1	1	$x \cdot \overline{y}$	y

Daraus liest man ab: $u = y + \overline{s}tx \qquad v = \overline{s}x + sx(t \nleftrightarrow y)$

Faltung von PLAs

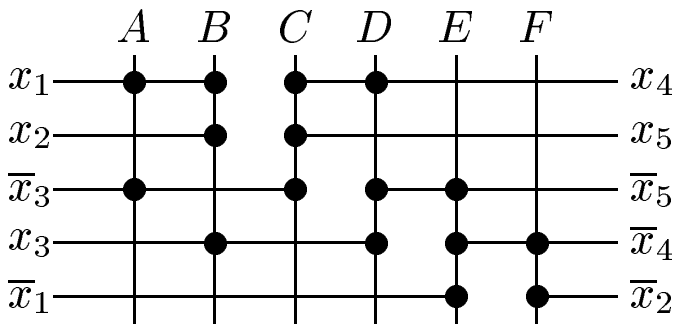
PLA für eine Funktion $F : B^5 \to B^3$:



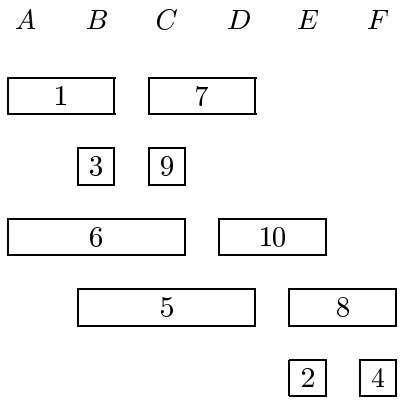
Überdeckungsmatrix

		A	B	C	D	E	F
1	x_1	1	1				
2	$\overline{x}_1$					1	
3	x_2		1				
4	$\overline{x}_2$						1
5	x_3		1		1		
6	$\overline{x}_3$	1		1			
7	x_4			1	1		
8	$\overline{x}_4$					1	1
9	x_5			1			
10	$\overline{x}_5$				1	1	

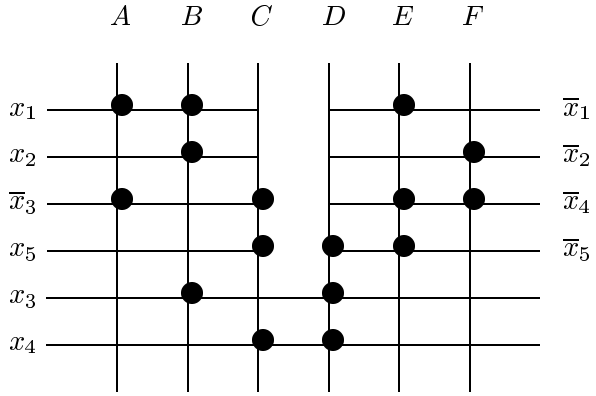
Faltung der Und-Ebene



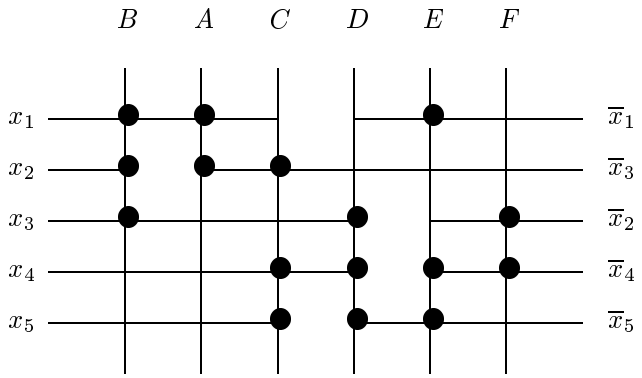
Matrix nach Zeilenüberlagerung



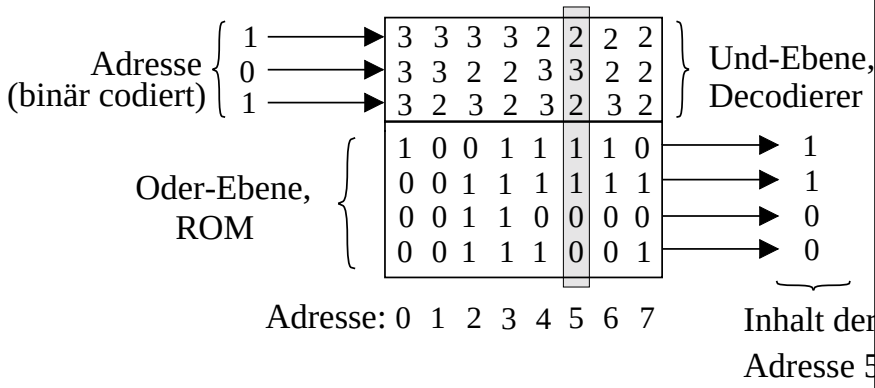
Block-Faltung



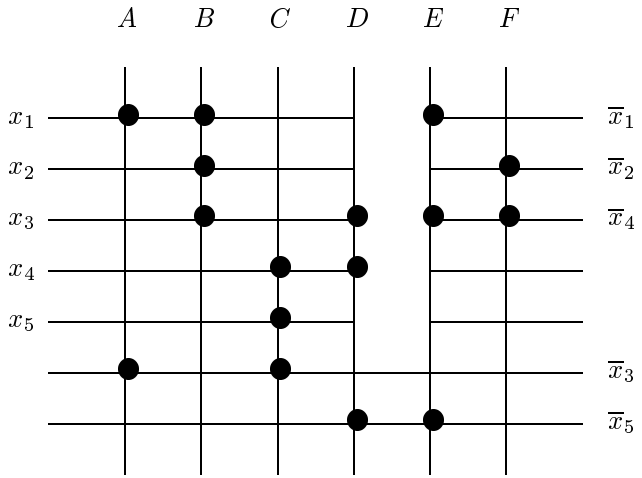
Bedingte Faltung



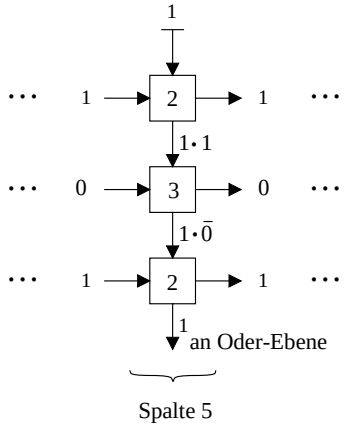
Beispiel eines ROM



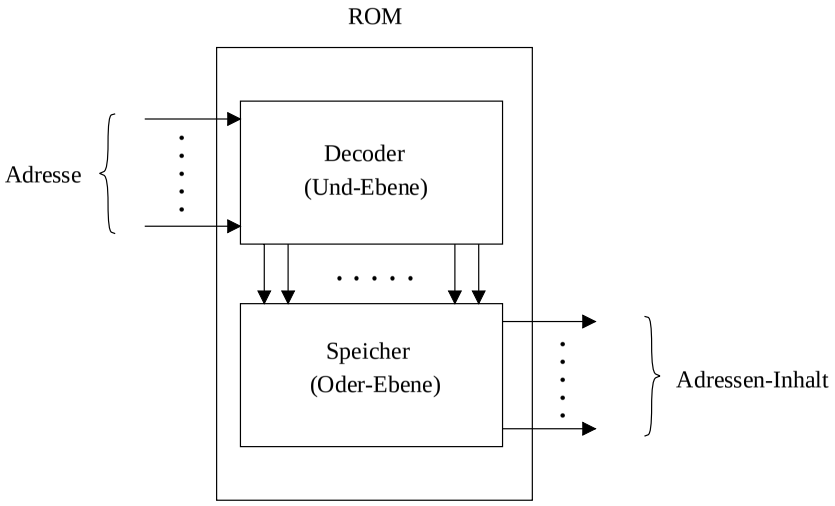
Bedingte Block-Faltung



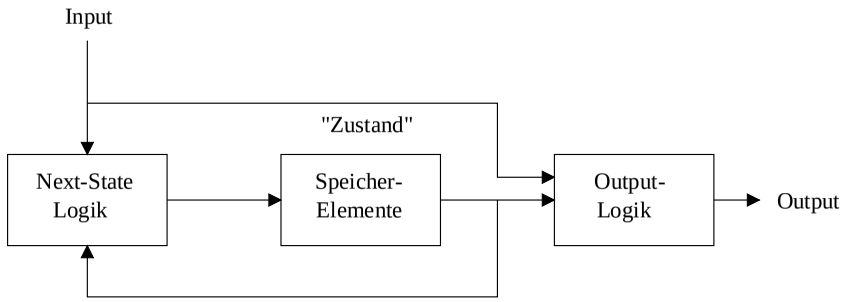
Auswahl einer Adresse



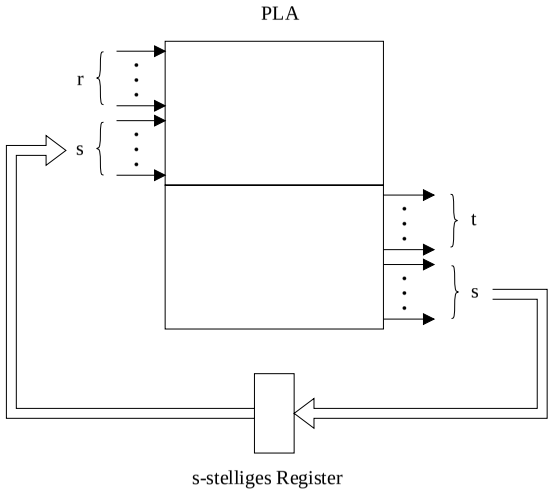
PLA als ROM



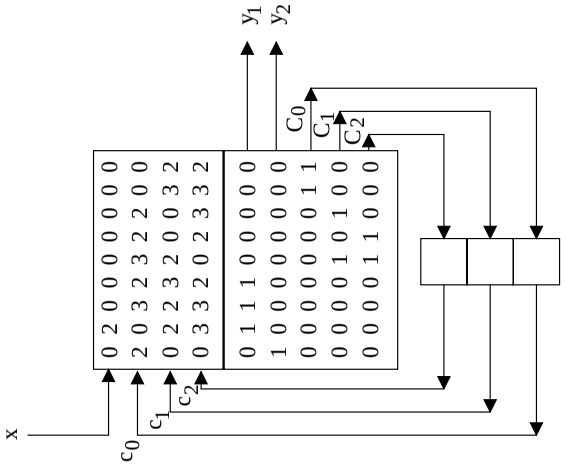
Prinzip eines sequentiellen Rechners



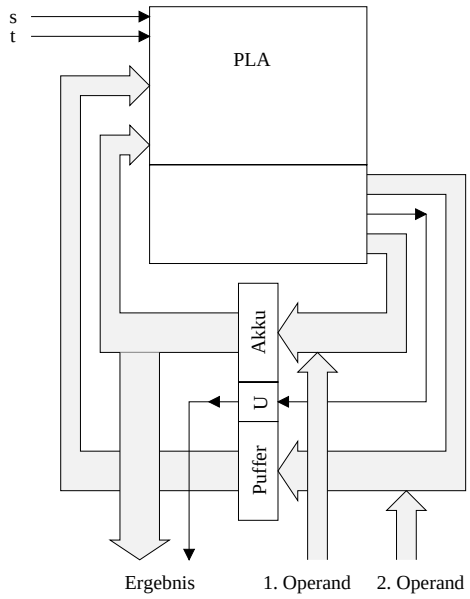
Schaltwerkrealisierung durch ein PLA



Beispiel: dreistelliger Ringzähler im Gray-Code



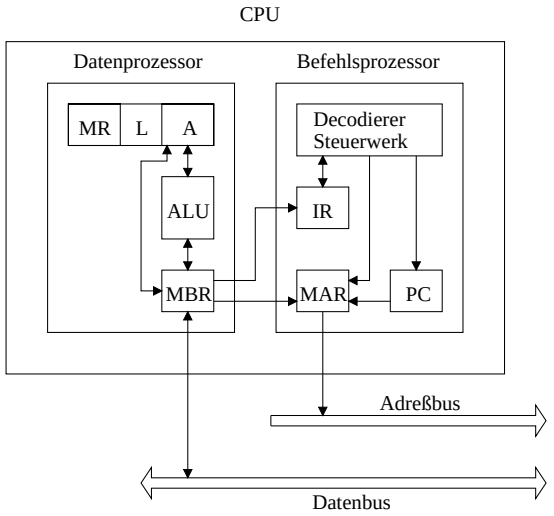
Prinzip eines Addierers bei Verwendung eines PLAs



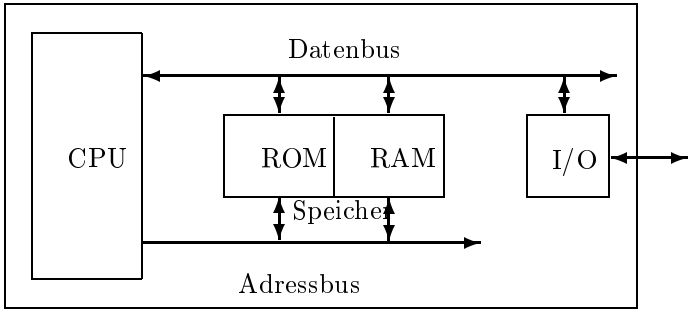
8. Organisationsplan eines von Neumann-Rechners

- Struktur eines von Neumann-Rechners
- Struktur und Arbeitsweise einer CPU
- Speicher
- Busse
- I/O

Struktur einer CPU



Struktur eines von Neumann-Rechners



Charakteristika des von Neumann-Rechners

1. Zu jedem Zeitpunkt führt die CPU *genau einen Befehl* aus, und dieser kann (höchstens) *einen* Datenwert bearbeiten (SISD-Prinzip).
2. Alle Speicherworte (d. h. Inhalte der Speicherzellen) sind als Daten, Befehle oder Adressen brauchbar. Die jeweilige Verwendung richtet sich nach dem momentanen Kontext.
3. Da Daten und Programme nicht in getrennten Speichern untergebracht werden, besteht grundsätzlich keine Möglichkeit, die Daten vor ungerechtfertigtem Zugriff zu schützen.

Fetch/Execute-Zyklus

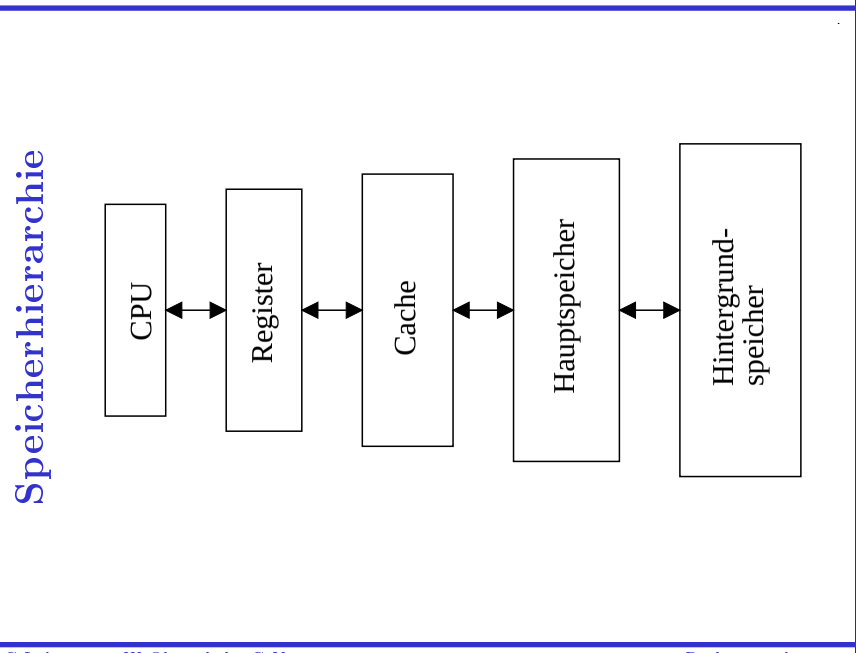
2-Phasen-Konzept der Befehlsverarbeitung:

1. In der *Fetch-Phase* wird der Inhalt von PC nach MAR gebracht und der Inhalt dieser Adresse aus dem Speicher über MBR nach IR geholt. Der Rechner geht jetzt davon aus, *dass* es sich um einen Befehl handelt.

   Der Decodierer erkennt, um *welchen* Befehl und um welchen Befehls*typ* es sich handelt und veranlaßt ggf. die Bereitstellung von Operanden.

   Schließlich muss der Inhalt von PC aktualisiert werden.

2. In der darauf folgenden *Execution-Phase* erfolgt die Befehlsausführung sowie eine Initiierung der Fetch-Phase für den nächsten Befehl.



Fetch-Phase

MAR ← PC;

MBR ← <MAR>;

IR ← MBR;

decodiere IR;

falls kein Sprungbefehl

dann { stelle Operanden bereit; PC ← PC + 1 }

sonst PC ← Sprungzieladresse;

Beispiel einer Cache-Hierarchie

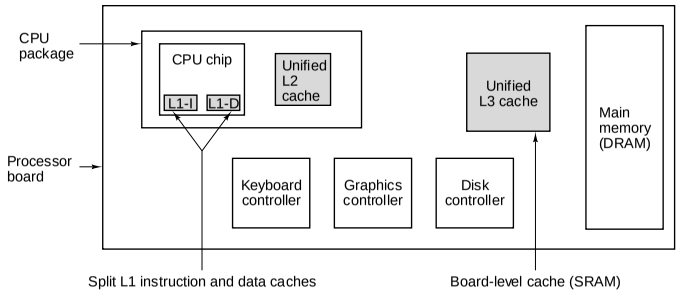
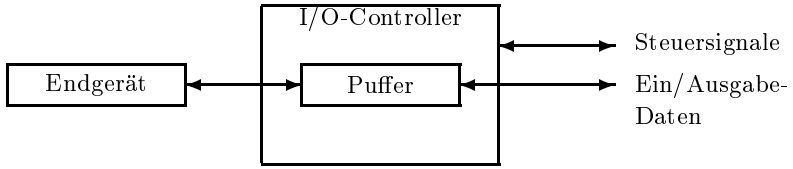


Figure 4-37. A system with three levels of cache.

Organisation einer I/O-Einheit



3 Arten der Datenübertragung:

- programmierter I/O
- interrupt-gesteuerter I/O
- Direct Memory Access

Busse am Beispiel des Pentium II

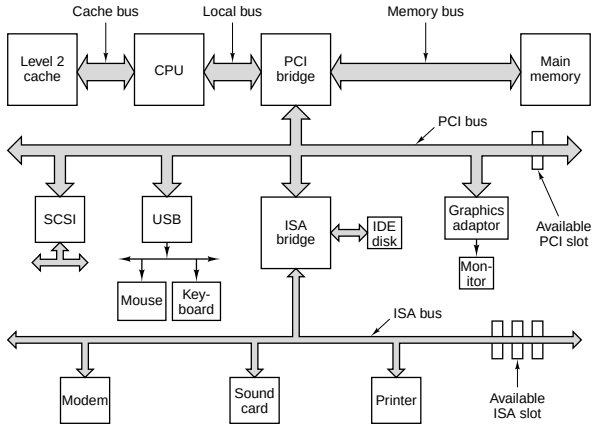


Figure 3-50. Architecture of a typical Pentium II system. The thicker buses have more bandwidth than the thinner ones.

Direct Memory Access (DMA)

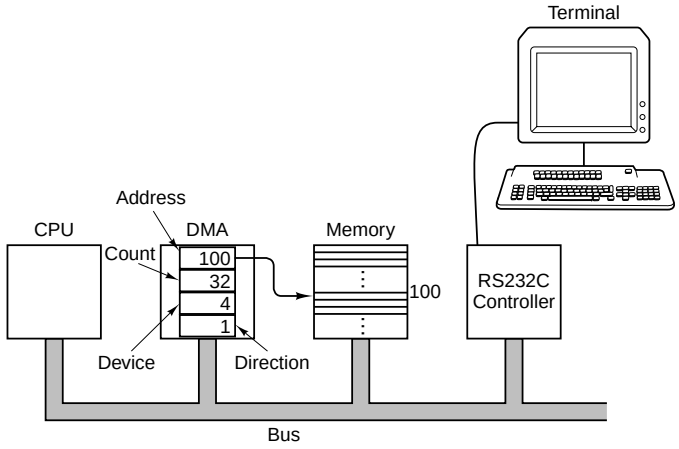


Figure 5-32. A system with a DMA controller.

Klassifikation der von Neumann-Rechnern

Anhand der Leistungsfähigkeit:

- PCs
- Workstations
- Großrechner (Mainframes)

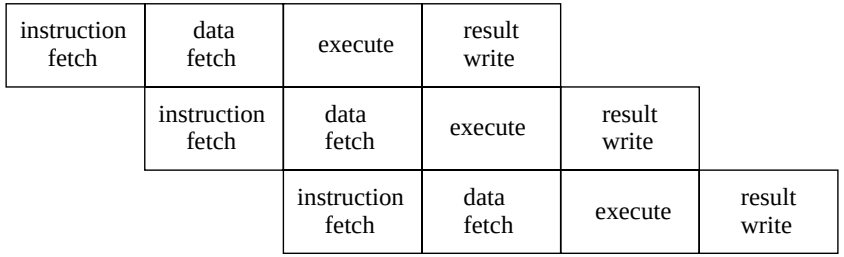
Anhand des Maschinenbefehlssatzes:

- CISC-Rechner
- RISC-Rechner

Probleme von CISC-Prozessoren

1. Der von Neumannsche Flaschenhals verhindert, dass die Geschwindigkeit, mit welcher auf einen Speicher zugegriffen werden kann, mit der einer CPU vergleichbar ist.
2. Eine Reihe von Instruktionen bzw. Kombinationen von Instruktion und Adressierungsart wird nur in speziellen Anwendungen verwendet; dennoch muss der entsprechende Mikrocode vorgesehen werden.
3. Die Mikroprogrammierung des Steuerwerks ist langsamer als eine fest verdrahtete Steuerung.

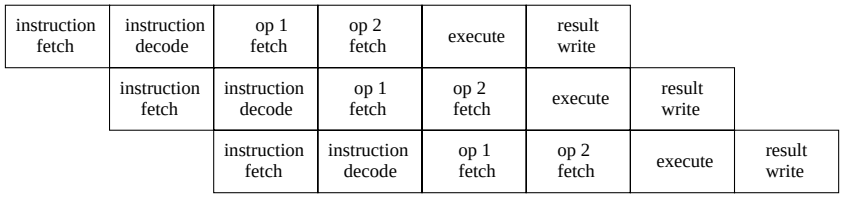
Einfaches Befehlsphasen-Pipelining



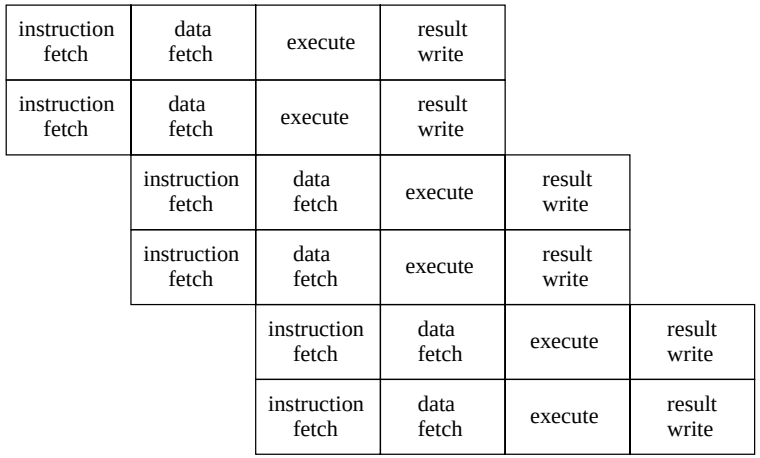
Merkmale von RISC-Prozessoren

- Der Befehlssatz umfasst i.a. nur wenige Instruktionen sowie Adressierungsarten;
- durch Beschränkung auf elementare Grundfunktionen können die meisten Befehle innerhalb von einem Maschinen-Zyklus ausgeführt werden;
- auf den Hauptspeicher wird nur mit speziellen Load- und Store-Befehlen zugegriffen („Load/Store-Architektur“);
- die Befehlsausführung wird unterstützt durch zusätzliche Hardware wie etwa eine große Anzahl von Registern;
- der Befehlsdecodierer bzw. das Steuerwerk ist fest verdrahtet;
- zur Unterstützung einer schnellen Befehls- und Operanden-Decodierung haben alle Instruktionen ein festes Format.

Superpipelining



Superskalar-Architektur



Alternativen zum von Neumann-Konzept

Flynn-Klassifikation:

Single Instruction	—	Single Data:	SISD
Single Instruction	—	Multiple Data:	SIMD
Multiple Instruction	—	Single Data:	MISD
Multiple Instruction	—	Multiple Data:	MIMD