Grundlagen der Technischen Informatik 2

Digitaltechnik
Rechneraufbau

Prof. Dr. U. Kebschull
Technische Informatik
kebschull@informatik.uni-leipzig.de

Übersicht

- Einleitung
- Schaltnetze
  - KV-Diagramme
  - Minimierung nach Quine MC-Cluskey
  - Bündelminimierung mit KV-Diagrammen
- 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:

- 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)

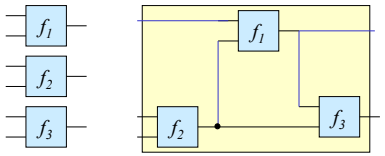
0 Einleitung

Der Entwurf elektronischer Systeme ist gekennzeichnet durch:

- Zunahme der Komplexität und Integrationsdichte
- höhere Packungsdichten aufgrund geringerer Strukturgrößen
- steigende Anforderungen (Platzbedarf, Taktrate, Leistungsverbrauch, Zuverlässigkeit)
- kurze Entwicklungszeiten (time to market)
- Wiederverwendung von Entwurfsdaten (Re-use)

Die Entwicklung elektronischer Systeme ist bei der heutigen Komplexität nur durch eine strukturierte Vorgehensweise beherrschbar!

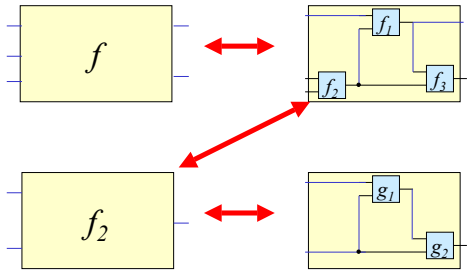
Grundprinzip des Entwurfs



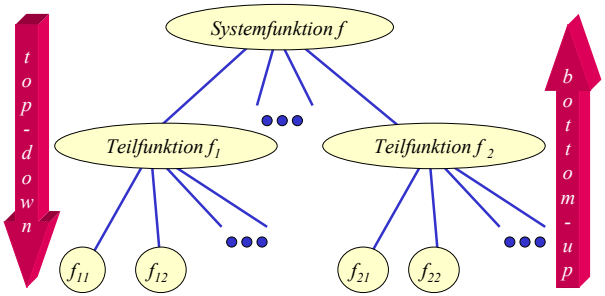
Komponenten + *Struktur*

= *gewünschtes Verhalten*

Abstraktion und Detaillierung



„top-down“ und „bottom-up“



Technische Kriterien für den Entwurf von Schaltnetzen

- Korrekte Realisierung unter Beachtung des statischen und dynamischen Verhaltens der verwendeten Bauelemente
- Berücksichtigung technischer Beschränkungen (Anzahl der Eingänge, begrenzte Belastbarkeit der Ausgänge, zur Verfügung stehende Bausteine (Bausteinbibliothek), Temperaturgrenzen, Speicherplatz (bei PLAs), Taktfrequenz)
- Gewährleistung hoher Systemzuverlässigkeit (leichte Testbarkeit, Selbsttest, Fehlertoleranz, zuverlässiger Betrieb)
- Berücksichtigung von Forderungen an die Gebrauchseigenschaften (universelle Einsatzmöglichkeit, großer Funktionsumfang)
- Berücksichtigung technologischer Nebenbedingungen (Kühlung, Versorgungsspannung)
- Vermeidung von Störeinflüssen (elektromagnetische Felder)

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
  - ⇨ Stromverbrauch

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 aus
  - ⇨ Korrektheit der Realisierung
  - ⇨ Einhaltung der technologischen Grenzen
  - ⇨ ökonomische Kriterien

☞ Wir betrachten in dieser Vorlesung nur die Minimierung des Realisierungsaufwands

1 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

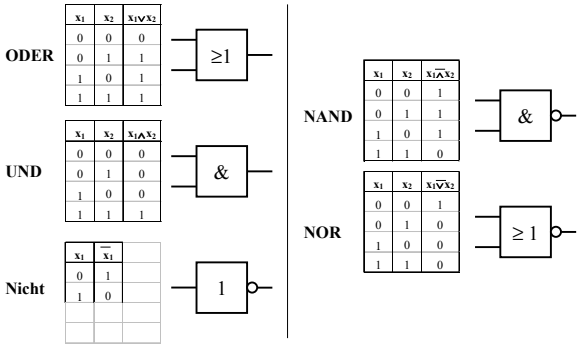
Darstellung Boolescher Funktionen durch Funktionstabellen

- 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

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

Darstellung einiger zweistelliger Funktionen



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

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

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

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

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

Zusammenfassung der wichtigsten Begriffe aus TI1

Boolesche Variable	Variable, die den Wert wahr (1) oder falsch (0) annehmen kann
Produktterm:	UND-Verknüpfung von Booleschen Variablen
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, des genau eine „0“-Stelle einer booleschen Funktion beschreibt
disjunktive Normalform:	Darstellung der Funktion, die nur aus Mintermen besteht (DNF)
konjunktive Normalform:	Darstellung der Funktion, die nur aus Maxtermen besteht (KNF)

Beispiel

Implikant $x_2 \wedge x_0$ → $x_2 \wedge x_1 \wedge x_0$, $x_2 \wedge \bar{x}_1 \wedge x_0$

- DNF $f(x_2, x_1, x_0) = x_2x_1x_0 \vee x_2\bar{x}_1x_0 \vee \bar{x}_2x_1\bar{x}_0 \vee \bar{x}_2\bar{x}_1\bar{x}_0$

Minterm

Boolesche Variable

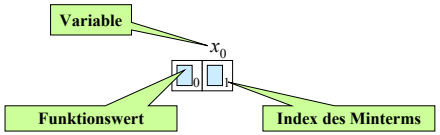
Maxterm

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

Implikat $x_2 \vee x_0$ → $x_2 \vee x_1 \vee x_0$, $x_2 \vee \bar{x}_1 \vee x_0$

1.1 KV-Diagramme

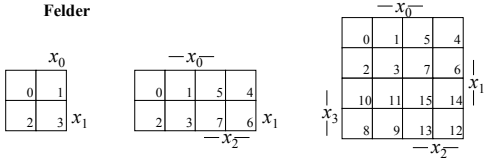
- Nach Karnaugh und Veitch
- Möglichkeit, Boolesche Funktionen übersichtlich darzustellen
  - ⇨ bis 6 Variablen praktisch einsetzbar
- Ausgangspunkt ist ein Rechteck mit 2 Feldern



KV-Diagramme

- Beispiele

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

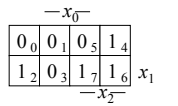


Eigenschaften von KV-Diagrammen

- 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

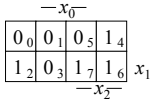
$$f(x_2,x_1,x_0) = x_1\overline{x}_0 \vee x_2x_1 \vee x_2\overline{x}_1\overline{x}_0$$



KV-Diagramme über die KNF

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

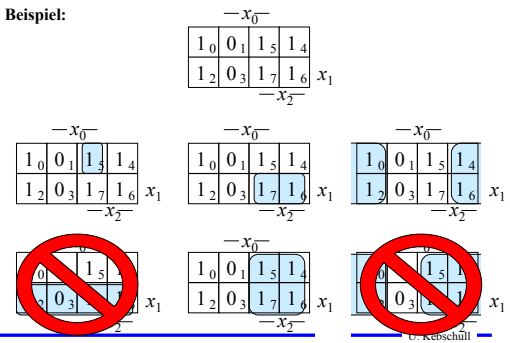
$$f(x_2,x_1,x_0) = x_1\overline{x}_0 \vee x_2x_1 \vee x_2\overline{x}_1\overline{x}_0$$



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

Minimalformen aus KV-Diagrammen

- Zusammenfassen von Mintermen zu Implikanten
- Beispiel:



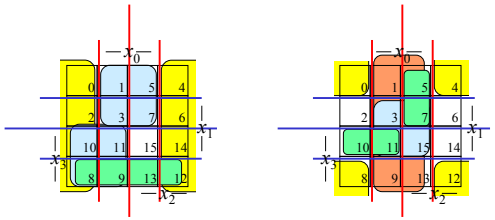
Implikant k-ter Ordnung

Def. 1.1: Es sei eine Boolesche Funktion $f(x_0,\ldots,x_{n-1}):B^n \to B$ 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

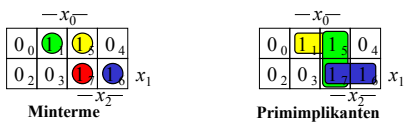


Primimplikant

Def. 1.2: Es sei eine Boolesche Funktion $f(x_0,\ldots,x_{n-1}):B^n \to B$ 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

$$f(x_2,x_1,x_0) = x_2x_1\overline{x}_0 \vee x_2x_0 \vee \overline{x}_2\overline{x}_1x_0$$



Überdeckung

Satz 1.1: 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

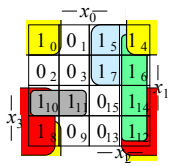
Def. 1.3: Es sei eine Boolesche Funktion $f(x_0,\ldots,x_{n-1}): B^n \to B$ gegeben. Ein Implikant p heißt **Kernprimimplikant**, wenn er einen Minterm überdeckt, der von keinem anderen Primimplikant überdeckt wird.

- 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

$$f(x_3,x_2,x_1,x_0) = \bar{x}_3\bar{x}_2\bar{x}_1\bar{x}_0 \vee \bar{x}_3x_2\bar{x}_1\bar{x}_0 \vee \bar{x}_3x_2\bar{x}_1x_0 \vee \bar{x}_3\bar{x}_2x_1x_0 \vee \bar{x}_3x_2x_1x_0 \vee x_3\bar{x}_2\bar{x}_1\bar{x}_0 \vee x_3\bar{x}_2x_1x_0 \vee x_3x_2\bar{x}_1\bar{x}_0 \vee x_3x_2x_1\bar{x}_0$$
$$= MINt(0,\ 4,\ 5,\ 6,\ 7,\ 8,\ 10,\ 11,\ 12,\ 14)$$

DNF

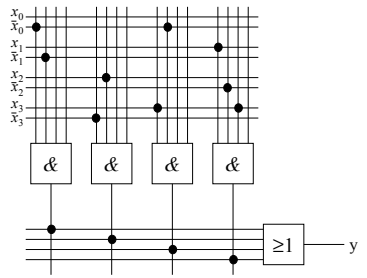


e $\bar{x}_1\bar{x}_0$ (0,4,8,12)
e $\bar{x}_3x_2$ (4,5,6,7)
$x_2\bar{x}_0$ (4,6,12,14)
$x_3\bar{x}_0$ (8,10,12,14)
e $x_3\bar{x}_2x_1$ (10,11)

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

DMF

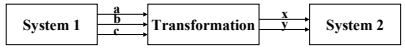
Realisierung als „Programmable Logic Array (PLA)



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

1.2 Bündelminimierung

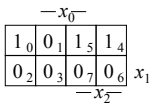
- Funktionen mit mehreren Ausgängen werden gemeinsam minimiert
- gemeinsame Implikanten sollten mehrfach genutzt werden
- Beispiel: Transformation eines Codes



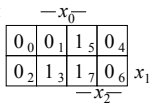
- Transformationstabelle

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:

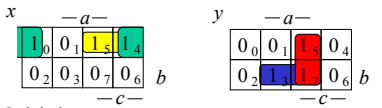


y:

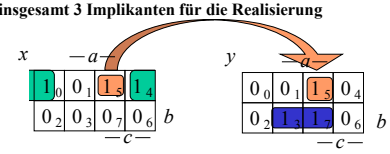


Bündelminimierung

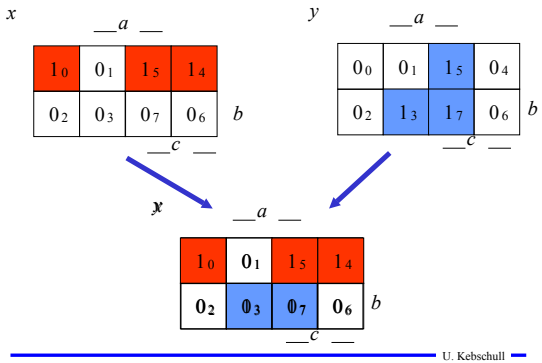
- Getrennte Minimierung
  - ⇨ insgesamt 4 Implikanten für die Realisierung



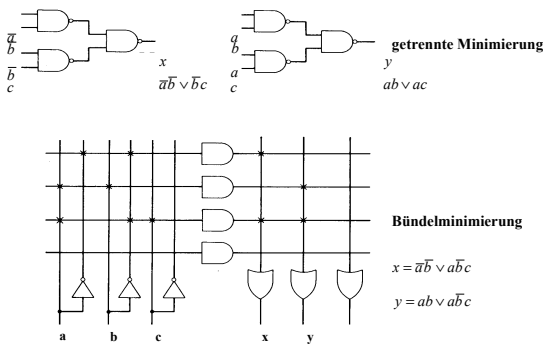
- Bündelminimierung
  - ⇨ insgesamt 3 Implikanten für die Realisierung



Bündelminimierung



Bündelminimierung

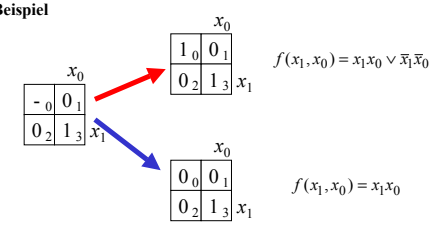


1.3 Unvollständig definierte Funktionen

- 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

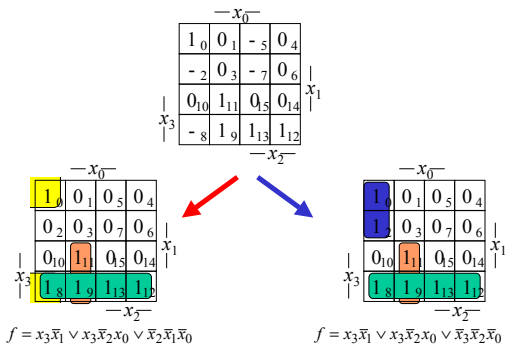
- Beispiel



$f(x_1,x_0) = x_1x_0 \vee \bar{x}_1\bar{x}_0$

$f(x_1,x_0) = x_1x_0$

Minimierung unvollständiger Boolescher Funktionen



1.4 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

$$\ddot{u}_f = w_C(w_A \vee w_B)w_A(w_B \vee w_C)w_C(w_A \vee w_B)w_A(w_B \vee w_C)w_Cw_Cw_Cw_C$$
$$= w_C(w_A \vee w_B)w_A(w_B \vee w_C)$$
$$= (w_Cw_A \vee w_Cw_B)(w_Aw_B \vee w_Aw_C)$$
$$= w_Cw_Bw_A \vee w_Aw_C$$
$$(= w_Aw_C)$$

Systematische Lösung des Überdeckungsproblems

- Ergebnis nach der Vereinfachung: $\ddot{u}_f = w_Cw_Bw_A \vee w_Aw_C$
- Damit f ganz überdeckt ist, muss $\ddot{u}_f$ eine Tautologie sein
  - ⇨ man sucht einen konjunktiven Term mit minimalen Kosten

$w_Cw_Bw_A$ **Kosten :** $3+3+2=8$

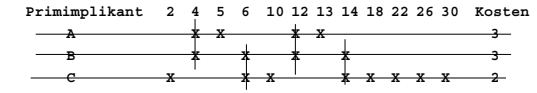
w_Aw_C **Kosten :** $3+2=5$

- 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

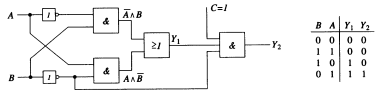
Heuristische Verfahren

- 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

1.5 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

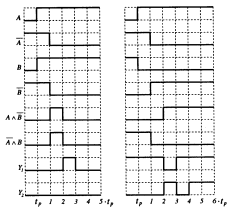
Entstehung von Hazards



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

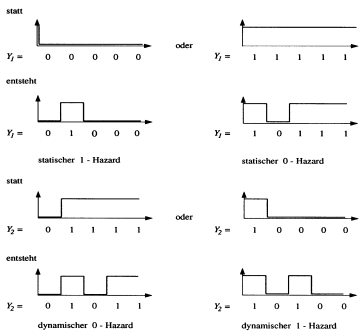
$X_t \wedge \overline{X}_{t-k}$ **müsste Null liefern**
statischer 1-Hazard bei einem Übergang von X: 0→1

$X_t \vee \overline{X}_{t-k}$ **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 →X=1 darf am Ausgang nur ein zu X_{t-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 →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



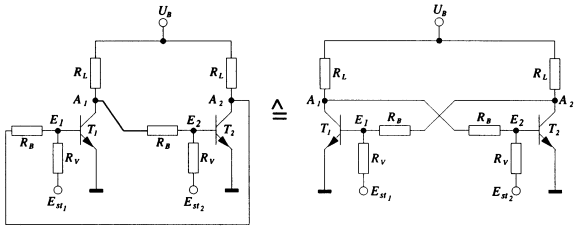
Behebung von Hazards

- 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

2 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

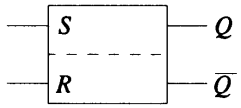
Funktionsprinzip

- 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

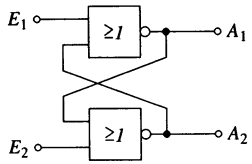
  gebaut werden
- RS-Flipflop
  - ⇨ wenn die Eingänge den Wert 0 haben, bleibt der vorherige Zustand stabil
  - ⇨ wird S =1, wird Q=1 und $\overline{Q}$=0
  - ⇨ wird R=1, wird Q=0 und $\overline{Q}$=1
  - ⇨ S=1 und gleichzeitig R=1 sind nicht zulässig



Schaltzeichen für ein RS-Flipflop nach DIN

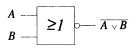
RS-Flipflop aus NOR-Gattern

- 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



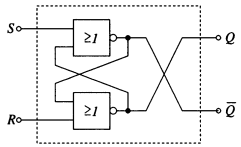
B	A	$\overline{A \vee B}$
0	0	1
0	1	0
1	0	0
1	1	0

RS-Flipflop aus NOR-Gattern

- Ein RS-Flipflop entsteht durch Vertauschen der Ausgänge

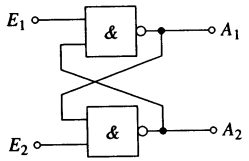
Funktionstabelle

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



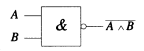
RS-Flipflop aus NAND-Gattern

- 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	



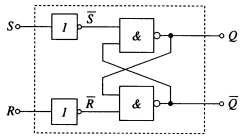
B	A	$\overline{A \wedge B}$
0	0	1
0	1	1
1	0	1
1	1	0

RS-Flipflop aus NAND-Gattern

- Ein RS-Flipflop entsteht durch Negation der Eingänge

Funktionstabelle

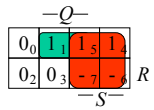
S	R	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	1	1	(wie vorher) speichern	
0	1	1	0	0	1
1	0	0	1	1	0
1	1	0	0	(1	1) unzulässig



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

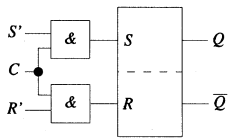


$Q_{n+1} = S \vee (\overline{R} \wedge Q_n)$

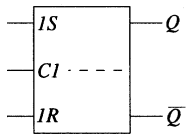
- Diese Gleichung heißt auch Funktionsgleichung oder Übergangsfunktion eines RS-Flipflops
  - ⇨ das Verhalten eines Flipflops kann durch eine Schaltfunktion beschrieben werden

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

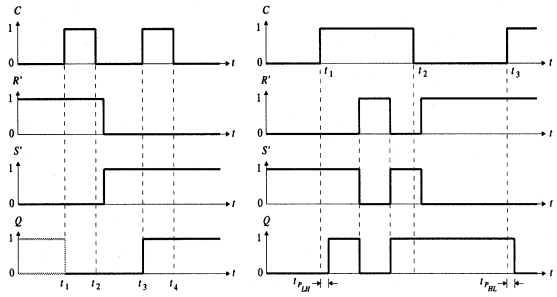
- Aus der Übergangsfunktion des RS-Flipflos

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

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

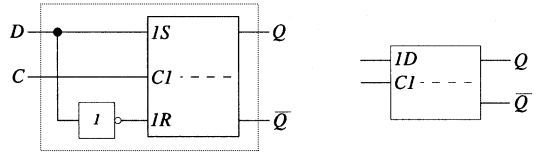
$$Q_{n+1} = (C \wedge S') \vee ((\overline{C \wedge R'}) \wedge Q_n)$$

Impulsdiagramm für Taktzustandssteuerung



D-Flipflop mit Zustandssteuerung

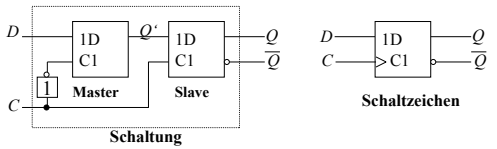
- Das D-Flipflop entsteht aus einem RS-Flipflop mit Zustandssteuerung, durch die Negation des Setzsignals S



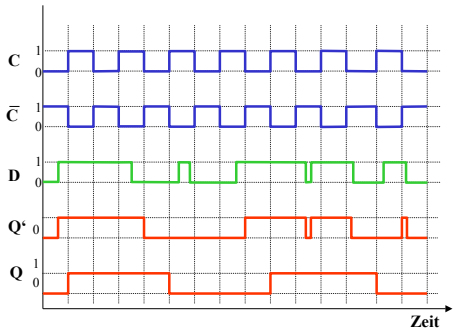
C	D	Q_n	Q_{n+1}	
0	-	0	0	speichern
0	-	1	1	speichern
1	0	-	0	rücksetzen
1	1	-	1	setzen

Master-Slave D- Flipflop

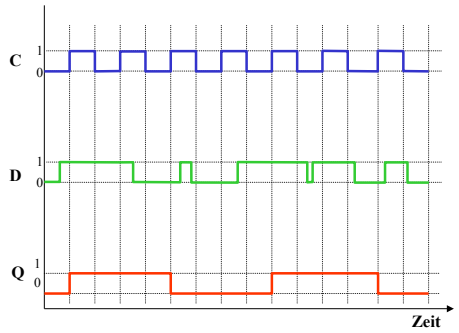
- Probleme beim Verketten von Flipflops
  - ⇨ bei C=1 rutschen die Eingänge bis zum Ausgang durch
  - ⇨ Anwendung: Schieberegister, Zähler
- Lösung: (positiv) flankengesteuertes Flipflop
  - ⇨ zwei D-Flipflops werden hintereinander geschaltet
  - ⇨ das erste Flipflop erhält den negierten Takt
  - ⇨ Master-Slave-Prinzip



Impulsdiagramm des Master-Slave D-Flipflops

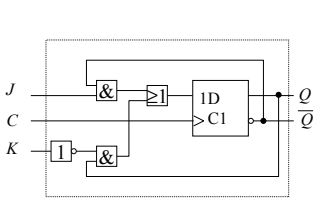


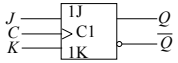
Impulsdiagramm des Master-Slave D-Flipflops



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



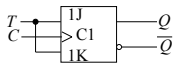


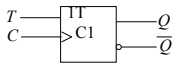
J	K	Q_{n+1}	
0	0	Q_n	speichern
0	1	0	rücksetzen
1	0	1	setzen
1	1	$\overline{Q}_n$	wechseln

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





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

Funktionstabelle

3 Schaltwerke

3.1 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

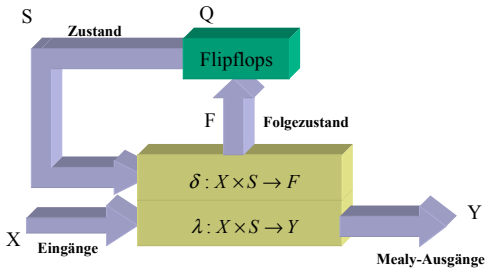
Beschreibung von endlichen Automaten

- 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, δ, λ, 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

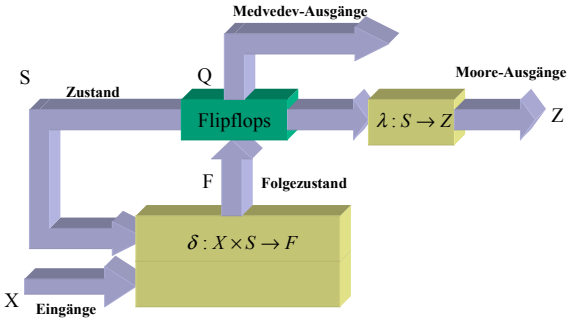
Mealy- und Moore-Automaten

- Die Zustände eines endlichen Automaten werden in Flipflops gespeichert
  - ⇨ möglich sind D-, T-, JK-, RS-Flipflops
- Der aktuelle Zustand wird an den Eingang der Schaltung rückgekoppelt
- Man unterscheidet Mealy- und 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



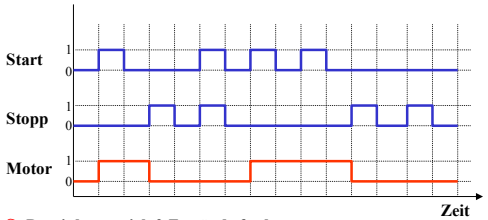
3.2 Darstellung endlicher Automaten

- 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

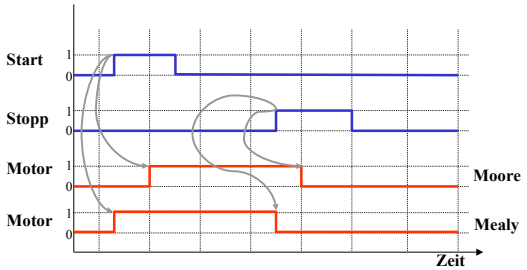
- Beschreibung der Funktion:
  - ⇨ an den Eingängen befinden sich zwei Tasten : (Start und Stopp)
  - ⇨ die Schaltung liefert ein Ausgangssignal, mit dem ein Gerät ein- oder ausgeschaltet werden kann
  - ⇨ wird die Starttaste gedrückt, soll das Gerät eingeschaltet werden
  - ⇨ es soll eingeschaltet bleiben, 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

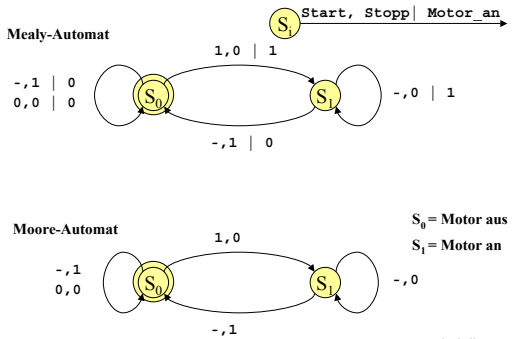


- Damit lassen sich 2 Zustände festlegen:
  - ⇨ Zustand s_0: Ausgabe von Motor=0 und warten auf Start=1 und Stopp=0
  - ⇨ Zustand s_1: Ausgabe von Motor=1 und warten auf Stopp=1

Mealy und Moore Verhalten



Automatengraph



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 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

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

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

Medvedev- und Moore-Automaten

- **Auch Moore-Automaten können während des Übergangs Fehlimpulse (Glitches, Hazards) auslösen**
  - ⇨ **unterschiedliche Laufzeiten in der Schaltung**
  - ⇨ **01 nach 10 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**

3.3 Analyse und Entwurf von Schaltwerken

Grundlegende Realisierung von Automaten

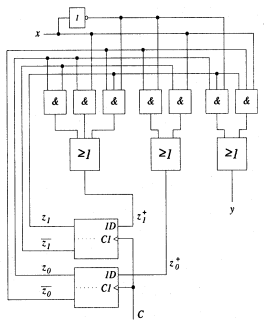
- **Asynchrone Realisierung**
  - ⇨ **Zustandsspeicher durch Rückkopplung**
  - ⇨ **es gibt keinen zentralen Takt**
  - ⇨ **die Zustandsspeicher (Flipflops) können zu jedem Zeitpunkt ihren Wert ändern**
  - ⇨ **self-timed**
- **Synchrone Realisierung**
  - ⇨ **Rückkopplung nur durch flanken- oder pegelgetriggerte 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**

3.3.1 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**

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

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

  - ⇨ **für die Ausgabefunktion**

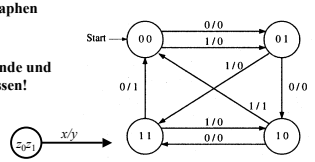
$$y = (z_0 \wedge z_1 \wedge \bar{x}) \vee (\bar{z}_0 \wedge z_1 \wedge x)$$

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!

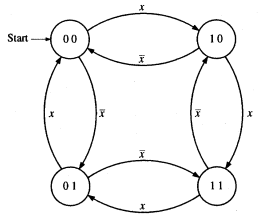


3.3.2 Entwurf von Schaltwerken

- 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



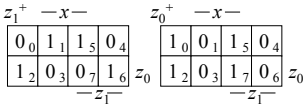
Automatengraph

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



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

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

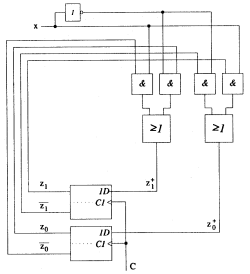
$$z_0^+ = (\bar{z}_1 \wedge \bar{x}) \vee (z_1 \wedge x)$$

Das Schaltwerk

- Die minimierten Übergangsfunktionen können schließlich in einem Schaltplan gezeichnet werden

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

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

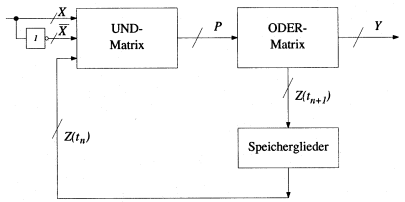


3.4 Technische Realisierung von Schaltwerken

- 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

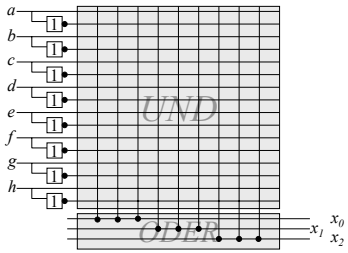
Realisierung mit einem PLA

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



Realisierung mit einem PAL

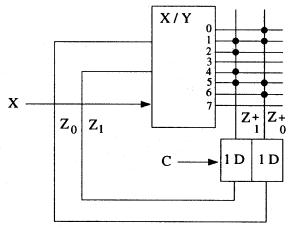
- Programmable Array Logic
  - ⇨ die ODER-Matrix ist vorgegeben
  - ⇨ es steht eine feste Anzahl von Implikanten pro Ausgang zur Verfügung
  - ⇨ die UND-Matrix ist programmierbar



Realisierung mit einem ROM

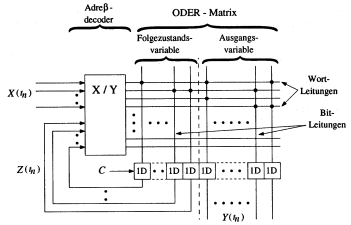
- Technische Realisierung durch ein PROM, EPROM, EEPROM
- Die UND-Matrix ist durch den Adressdekodierer vorgegeben
  - ⇨ alle Minterme sind implementiert
  - ⇨ direkte Implementierung der Funktionstabelle

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 Steuerwerke in Mikroprozessoren

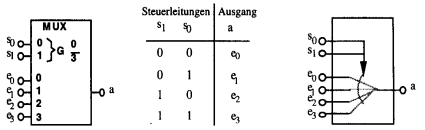


4. Spezielle Schaltnetze und Schaltwerke

- 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

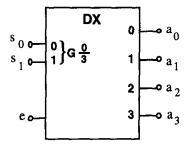
- Mehrere Eingänge, ein Ausgang
- über n Steuerleitungen können 2^n Eingänge ausgewählt und an den Ausgang durchgeschaltet werden



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

Demultiplexer

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

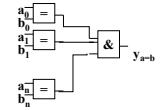


s_1	s_0	a_0	a_1	a_2	a_3
0	0	e	0	0	0
0	1	0	e	0	0
1	0	0	0	e	0
1	1	0	0	0	e

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

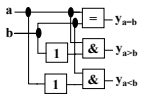
Vergleicher (Komparatoren)

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



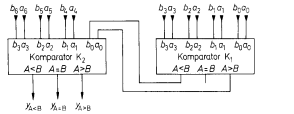
- 1-Bit Komparator mit Größenvergleich

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

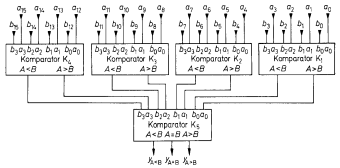


Komparatoren

- Serielle Erweiterung



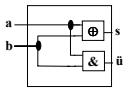
- Parallele Erweiterung



Addierer

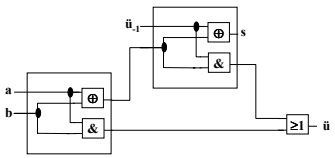
- Halbaddierer

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



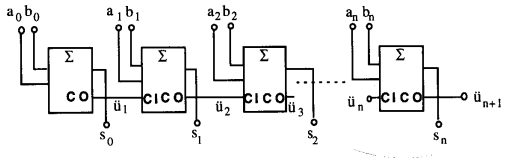
- Volladdierer

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

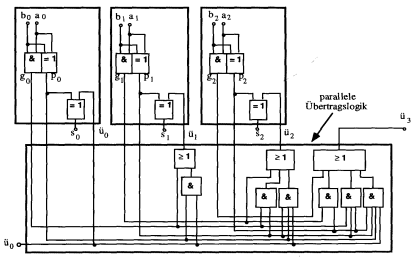
- Der Übertrag des Volladdierers $ü_i$ wird mit c_{i+1} verbunden



Addierer mit paralleler Übertragslogik

- Allgemein:

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

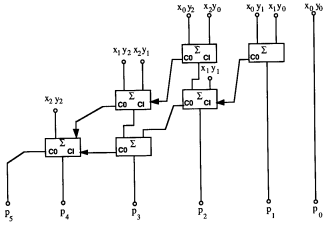


Multiplizierer

- Parallele Multiplikation durch Addierwerk

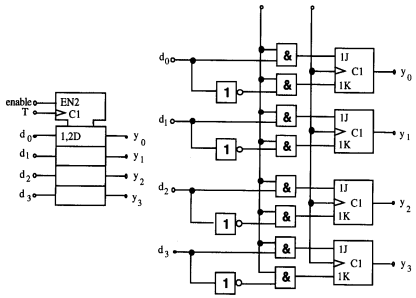
$$p = x \cdot y = \left(\sum_{i=0}^{n-1} x_i \cdot 2^i\right) \cdot \left(\sum_{j=0}^{n-1} y_j \cdot 2^j\right) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} 2^{i+j} x_i y_j$$

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



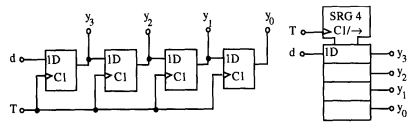
Register

- Speicherung einer n-stelligen Zahl durch n Flipflops



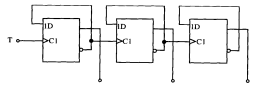
Schieberegister

- 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)

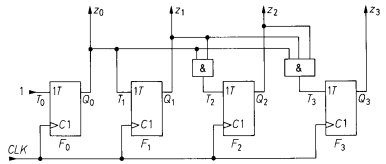


Zähler

- Einfacher Dualzähler durch Rückkopplung
- Asynchroner Ripple Carry Zähler

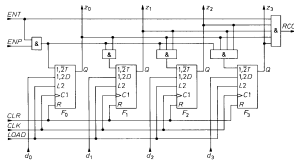


- Synchroner Dualzähler durch Carry-Look-Ahead-Logik

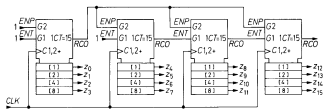


Zähler

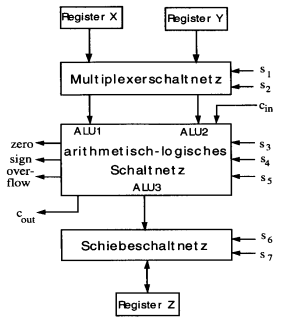
- Praktische Ausführung eines Zählers



- Kaskadierung eines Zählers



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 ∨ ALU2
1	0	0	ALU1 ∧ ALU2
1	0	1	$\overline{\text{ALU1}}$ ∧ ALU2
1	1	0	ALU1 ⇎ ALU2
1	1	1	ALU1 ↔ ALU2

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

Bauelemente eines Rechnersystems

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

5 Rechnerarithmetik

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

5.1 Formale Grundlagen

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

7.1.1 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 von Dezimalsystem in ein Zahlensystem zur Basis b

- Euklidischer Algorithmus
  - ⇨ die einzelnen Ziffern werden sukzessive berechnet

$$Z = z_n b^n + z_{n-1} b^{n-1} + \ldots + z_1 b + z_0 + z_{-1} b^{-1} + z_{-2} b^{-2} + z_{-m} b^{-m}$$
$$= y_p b^p + y_{p-1} b^{p-1} + \ldots + y_1 b + y_0 + y_{-1} b^{-1} + y_{-2} b^{-2} + y_{-q} b^{-q}$$

  - ⇨ Algorithmus
    1. Berechne P gemäß der Ungleichung $b^{n-1} \le Z < b^n$
    2. Ermittle y_p und den Rest R_p durch Division von Z durch b^p
       $y_p = Z \text{ div } b^p$; $R_p = Z \bmod b^p$; $y_p = \{0, 1, \ldots, b\text{-}1\}$
    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	$16^3 \le 15741{,}233_{10} < 16^4$	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:

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

- Die gegebene Dezimalzahl wird sukzessive durch die Basis b dividiert
  - ⇨ Die jeweiligen ganzzahligen Reste ergeben die Ziffern der Zahl X_b
  - ⇨ Reihenfolge: niedrigstwertige 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

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

- 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

0011	0100,	1101	0100
3	4,	D	4

5.1.2 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

Gray-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	Gray-Codierung
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

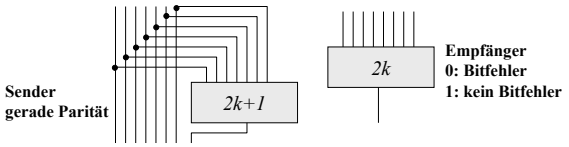
ASCII-Tabelle

	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



5.1.3 Darstellung negativer Zahlen

- Für die Darstellung von Zahlen in Rechnern werden 4 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

5.1.4 Fest- und Gleitkommazahlen

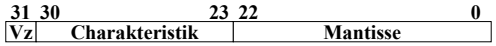
- Darstellung von Zahlen mit einem Komma
- Festkommadarstellung
  - ⇨ Festlegung der Stelle in einem Datenwort

0	1	0	1	1	0	0	1	0	,	1	1	0	0	0	0	0

  - ⇨ 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



Normalisierte Gleitkommadarstellung

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

$$\frac{1}{2} \leq \text{Mantisse} < 1$$

  - ⇨ 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

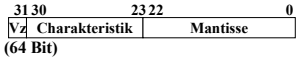
0	100 0110 1	110 1111 0111 1100 0000 0000

  - ⇨ Gleitkommadarstellung, normiert, implizite erste 1

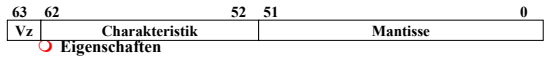
0	100 0110 1	101 1110 1111 1000 0000 0000

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

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

5.2 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

5.3 Multiplikation und Division

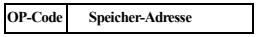
- Prinzip der Multiplikation: Schieben und Addieren
- Multiplikation von Zahlen im Zweierkomplement:
  - ⇨ die Zahlen werden in eine Form mit Betrag und Vorzeichen konvertiert
  - ⇨ die Beträge werden Multipliziert (kaskadiertes Addierwerk)
  - ⇨ das neue Vorzeichen wird berechnet (Exklusiv-ODER-Verknüpfung)
- 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

6 Ein minimaler Rechner

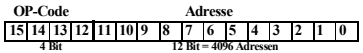
- Der Toy-Prozessor
  - ⇨ Quelle: Phil Kopmann, Microcoded versus Hard-Wired Logic
  - ⇨ Byte Januar 87, S. 235
- einfacher aber vollständiger Mikrorechner
- einfacher Aufbau mit Standardbausteinen
- RISC-Rechner
  - ⇨ alle Befehle in einem Takt (2 Phasen Takt)
  - ⇨ sehr einfacher Befehlssatz (12 Befehle)

Spezifikation des Toy-Rechners

- 1-Adress-Maschine
- Zielregister ist immer der Akkumulator (ACCU)



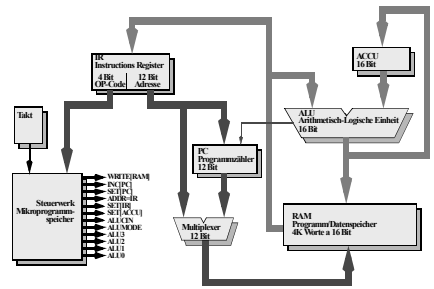
- Befehlsformat



- Komponenten (Speicher CPU)

RAM:	4096 * 16 Bit
ALU:	4 * 74181 ALU-Baustein
ACC:	Register
IR:	Instruktionsregister
PC:	Programmzähler
MUX:	Multiplexer

Blockschaltbild



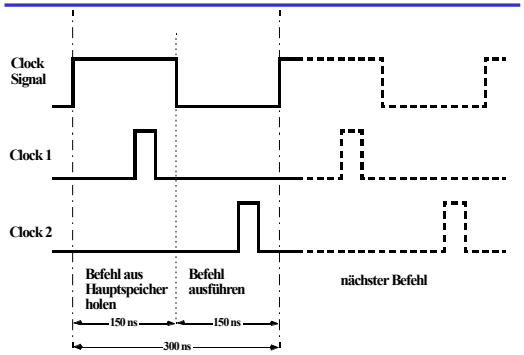
Befehlssatz

Opcode	Operation		Beschreibung
0	STO	<Adresse>	speichere den ACCU ins RAM an die Adresse
1	LDA	<Adresse>	lade 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		nicht benutzt
13	NOP		nicht benutzt
14	NOP		nicht benutzt
15	NOP		nicht benutzt

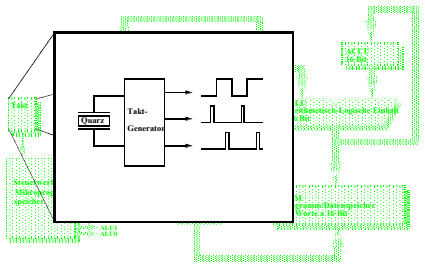
Spezifikation der Befehle

OpCode	Operation	Zyklus	Beschreibung
0	STO	1	ADDR=IR; ALU=ACC; WRITE(RAM)
		2	ADDR=PC; SET(IR); INC(PC)
1	LDA	1	ADDR=IR; ALU=RAM; SET(ACC)
		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=ACC+RAM; SET(ACC)
		2	ADDR=PC; SET(IR); INC(PC)
...			
9	INC	1	ALU=ACC+1; SET(ACC)
		2	ADDR=PC; SET(IR); INC(PC)
...			
12-15	NOP	1	ADDR=PC; SET(IR); INC(PC)
		2	

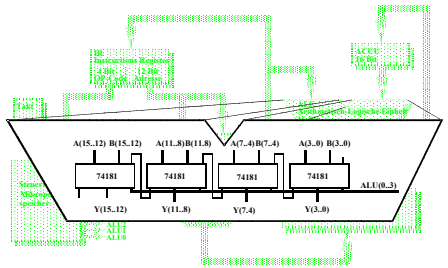
Ablaufsteuerung



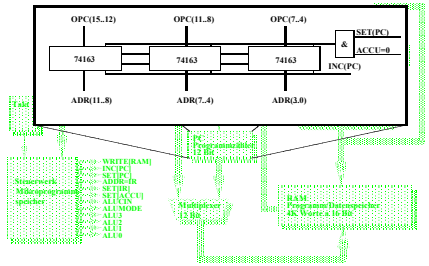
Komponente 1: Der Taktgenerator



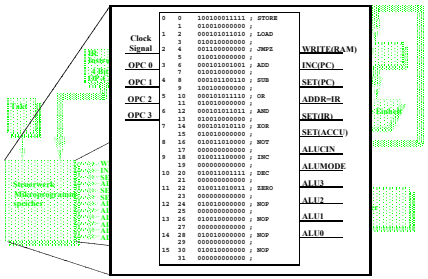
Komponente 2: Die ALU



Komponente 3: Der Befehlszähler



Das Steuerwerk als ROM



Ablauf eines Maschinenbefehls

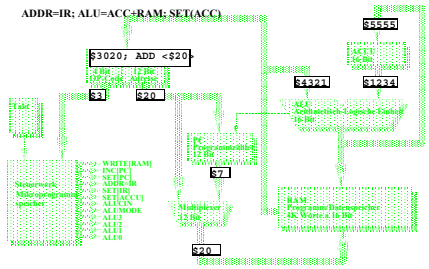
- Ab der Speicherstelle $0007 steht die Befehlssequenz:

  $3020 ; ADD <$20>
  $0030 ; STO <$30>

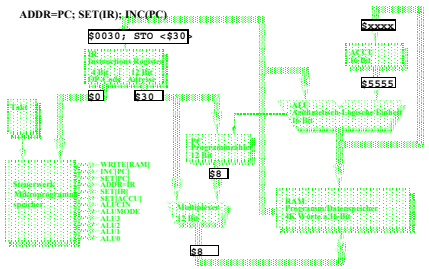
- Der Accuinhalt ist $1234.
- Der Inhalt der Speicherstelle $20 ist $4321

- Wie werden die Befehle abgearbeitet?

Ablauf eines Maschinenbefehls (Phase 1)



Ablauf eines Maschinenbefehls (Phase 2)

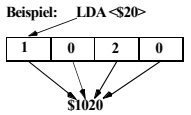


Ein Beispielprogramm

```
; Variablen:
;  Loopcount=$20, Number=$21 (enthaelt zunaechst 0)
; Labels:
;  loop=$2, end=$b
;
$0020       ; STO Loopcount    ; Auswerten des initialen
                               ; Accuinhalts
$200b       ; BRZ end          ; Schon fertig?
#-----------------------------
#loop:
$1021       ; LDA Number       ; nat. Zahl mitzaehlen
$9000       ; INC
$0021       ; STO Number
$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
$200c       ; STP                  ; Endlosschleife
```


Assemblierung des Programms

0	0	STO
1	1	LDA
2	2	BRZ
3	3	ADD
4	4	SUB
5	5	OR
6	6	AND
7	7	XOR
8	8	NOT
9	9	INC
10	A	DEC
11	B	ZRO
12	C	NOP
13	D	NOP
14	E	NOP
15	F	NOP



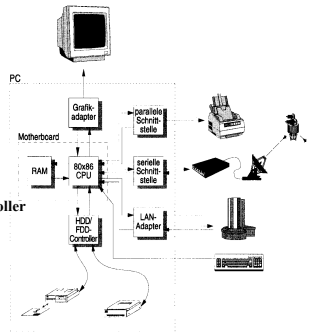
Assemblieren:
Assembler als Kommentar schreiben
Adressen der Labels für Sprünge feststellen
Adressen für Variablen festlegen
Hexcode aus OP-Codetabelle und aus Labels/Variablenadressen berechnen

Unterschiede zu realen Rechnern

	Toy Rechner	reale Prozessoren
Wortlänge	12 Bit	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

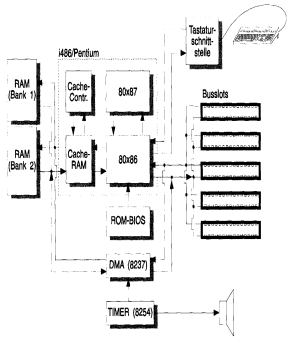
7 Aufbau von Rechnersystemen

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

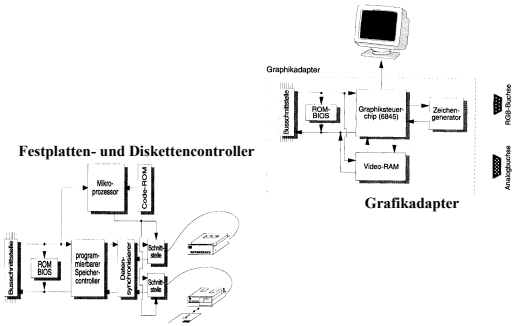


Hauptkomponenten der Zentraleinheit

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

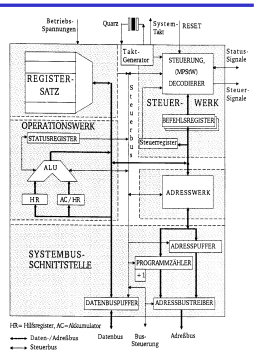


Peripherie



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

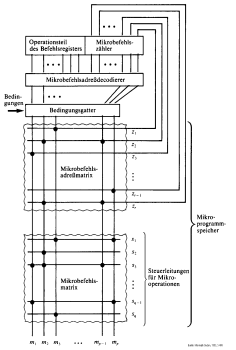


Das Steuerwerk

- 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

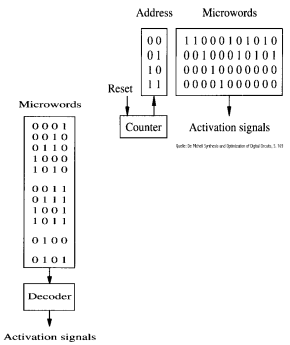
Mikroprogrammierung

- 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



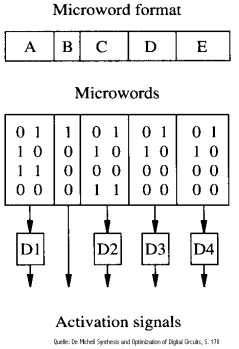
Vertikale und horizontale Mikroprogrammierung

- Horizontale Mikroprogrammierung
  - ⇨ jedes Ausgangssignal erhält eine eigene Steuerleitung
- Vertikale Mikroprogrammierung
  - ⇨ Die Ausgangssignale werden über einen Multiplexer angesteuert

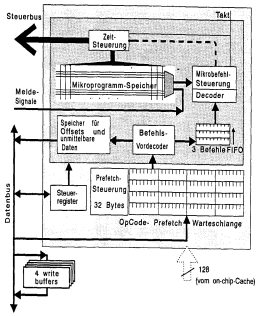


Mischformen

- Unabhängige Teile des horizontalen Mikrobefehlsworts werden zusammengefaßt und vertikal kodiert

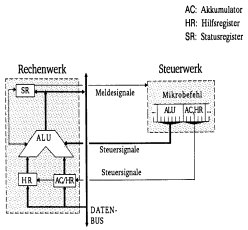


Das Steuerwerk des Intel 486



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
- 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 insbesondere:	(*shift*)
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 insbesondere:	(*rotate*)
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

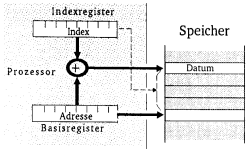
Mnemonic	Bedeutung
NOP	keine Operation, nächsten Befehl ansprechen (*no operation*)
WAIT	Warten, bis ein Signal an einem speziellen Eingang auftritt
SYNC	Warten auf einen Interrupt
HALT, STOP	Anhalten des Prozessors, Beenden jeder Programmausführung
RESET	Ausgabe eines Rücksetz-Signals für die Peripherie-Bausteine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

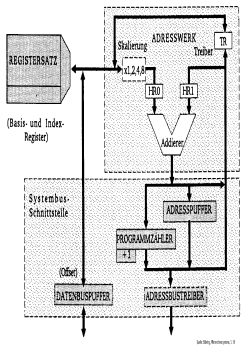


Die Register im Intel 80x86

- 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
  - ⇨ date 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 Spatelbereichs

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



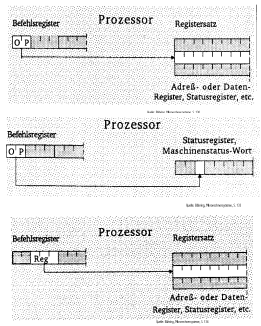
Adressierungsarten

Adressierungsarten

- Register-Adressierung
  - implizit — Flag
  - explizit
- einstufige Speicher-Adressierung
  - unmittelbar
  - direkt
    - absolut
    - Zero-Page
    - Seiten(-Register)
  - Register-indirekt
  - indiziert
    - Speicher-relativ
    - Register-relativ
    - Register-relativ mit Index
  - Programmzähler-relativ
- zweistufige Speicher-Adressierung
  - indirekt absolut
  - indirekt Register- indirekt
  - indirekt indiziert
    - Speicher-relativ
    - Register-relativ
    - Register-relativ mit Index
  - indiziert indirekt
  - indirekt Programmzähler-relativ

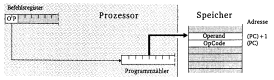
Register- Adressierung

- 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
    - drecrement R0

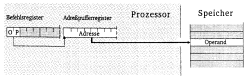


Einstufige Adressierung

- Unmittelbare Adressierung
  - ⇨ Der Befehl enthält den Operanden
  - ⇨ Beispiel: LDA #$A3
    - load accu 3_{16}

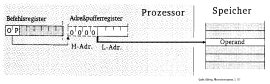


- Absolute Adressierung
  - ⇨ Das Speicherwort das dem Befehls folgt enthält die Adresse
  - ⇨ Beispiel: JMP $07FE

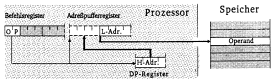


Seitenadressierung

- 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 $7 um 1

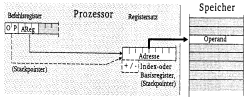


- Seiten-Register-Adressierung
  - ⇨ Höherwertige Adressteil wird von einem Register zur Verfügung gestellt
  - ⇨ Beispiel: LDA R0, <$7F



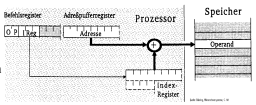
Register-Indirekte Adressierung

- 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

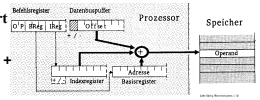


Indizierte Adressierung

- 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

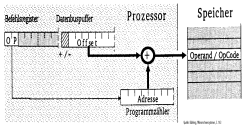


- Register-relative Addressierung mit Offset
  - ⇨ Der Basiswert befindet sich in einem speziellen Basisregister
  - ⇨ Ein der Inhalt des Indexregister und ein Offset wird zum Basiswert addiert
  - ⇨ autoincrement und autodecrement
  - ⇨ Beispiel: `ST R1, $A7(B0)(I0)+`
    - Speichere R1 an die Adresse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



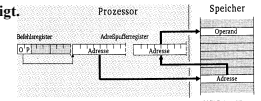
Programmzähler-relative Adressierung

- 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

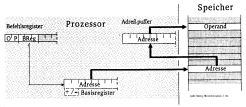


Zweistufige Speicheradressierung

- 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

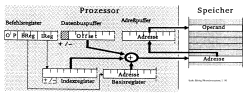


- Indirekte Register-indirekte Adressierung
  - ⇨ Der Befehl bezeichnet ein Register, dessen Inhalt die Speicherzelle ist, deren Inhalt als Adresse für das Speicherwort verwendet wird
  - ⇨ Beispiel: LD R1, ((R0))
    - Lade R1 mit dem Inhalt der Adresse, die in in der Speicherzelle steht, auf die R0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

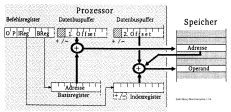


Zweistufige Speicheradressierung

- 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

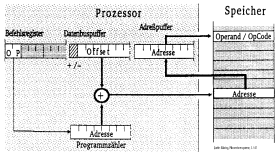


- Indizierte indirekte Adressierung
  - ⇨ Die Adresse des Speicherworts wird aus dem 1. Offset und dem Basisregister gebildet. Der Inhalt dieses Speicherworts wird zum Indexregister und dem 2. Offset hinzuaddiert und bildet die Adresse des gesuchten Speicherworts
  - ⇨ Beispiel: `INC $A7($10(B0))(I2)`
    - 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.



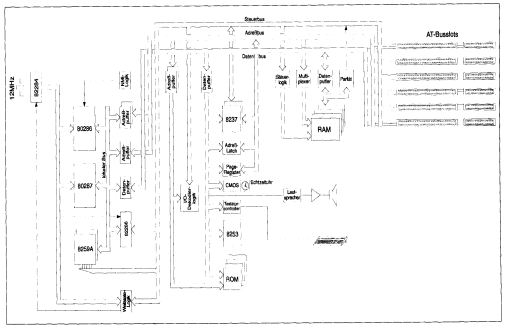
8 Rechner- und Gerätebusse

- Busse verbinden Komponenten eines Rechnersystems
  - ⇨ Datenbus 8 bis 64 Bit
  - ⇨ Adressbus 16 bis 64 Bit
  - ⇨ Steuerbus
- Rechnerbusse
  - ⇨ 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
- Gerätebusse
  - ⇨ Busse, die externe Komponenten mit einem Rechnersystem verbinden
  - ⇨ IEC Gerätebus
  - ⇨ EIDE Festplatten
  - ⇨ SCSI Geräte und Festplattenbus

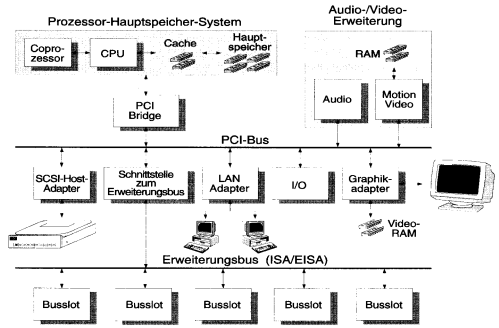
Interne Busse im PC

- lokaler Bus (Daten und Adressen)
  - ⇨ am Prozessor
- Systembus (Daten und Adressen)
  - ⇨ zentraler Bus
  - ⇨ Verbindung zu den Steckplätzen (ISA/EISA)
- Speicherbus (Daten und Adressen)
  - ⇨ Verbindung des Systembusses mit den Speicherbausteinen
  - ⇨ gemultiplexte Adressen
- X-Bus (Daten und Adressen)
  - ⇨ Adressierung der Komponenten des Motherboards

PC-Interne Busse im AT

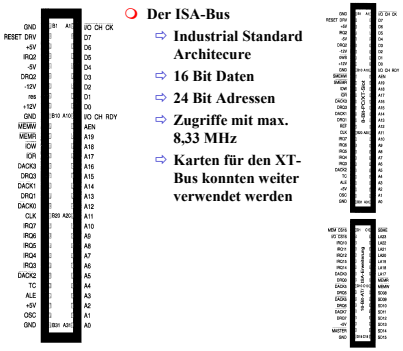


Moderne PC-Busstrukturen (PCI)



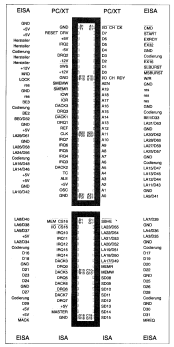
Der PC/XT-Bus und der ISA-Bus

- Der PC/XT-Bus
  - ⇨ Systembus
  - ⇨ 8 Bit Daten
  - ⇨ 20 Bit Adressen
  - ⇨ Zugriffe mit max. 8 MHz
  - ⇨ 16-Bit-Zugriffe beim XT mussten auf 2 8-Bit-Zugriffe abgebildet werden
- Der ISA-Bus
  - ⇨ Industrial Standard Architecure
  - ⇨ 16 Bit Daten
  - ⇨ 24 Bit Adressen
  - ⇨ Zugriffe mit max. 8,33 MHz
  - ⇨ Karten für den XT-Bus konnten weiter verwendet werden



Der EISA-Bus

- Extended ISA
- Evolutionäre Weiterentwicklung des ISA-Busses
- 32 Bit Daten
- 32 Bit Adressen
- Zugriffe mit max. 8.33 MHz
- Steckplatz ist kompatibel zu ISA Steckkarten
  - ⇨ ISA-Pins liegen tiefer und werden von den alten ISA-Karten nicht erreicht

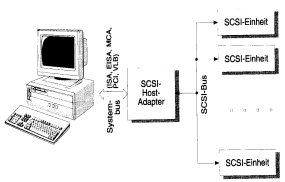


Der PCI-Bus

- Entkopplung von Prozessor und Erweiterungsbus durch eine Bridge
- 32-Bit-Standardbusbreite mit maximal 133MByte/s Transferrate
- Erweiterung auf 64 Bits mit maximal 266MByte/s Übertragungsrate
- Unterstützung von Mehrprozessorsystemen
- Burst-Transfers mit beliebiger Länge
- Unterstützung von 5V- und 3,3V-Versorgungsspannungen
- Write Posting und Read Prefetching
- Multimaster-Fähigkeit
- Betriebsfrequenz von 0MHz bis maximal 33MHz
- zeitlich gemultiplexter Adress- und Datenbus für geringe Pin-Anzahl
- Unterstützung für ISA/EISA/MCA
- Konfigurierung über Software und Register
- prozessorunabhängige Spezifikation

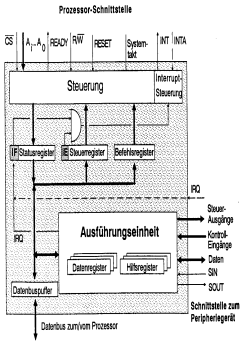
Gerätebusse: Der SCSI-Bus

- Small Computer Systems Interface
  - ⇨ 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 außer Festplatten berücksichtigt
  - ⇨ Fast SCSI
    - maximale Taktfrequenz wurde auf 10 MHz erhöht
  - ⇨ Wide SCSI
    - 16 Bit und 32 Bit Erweiterung der Datenbreite



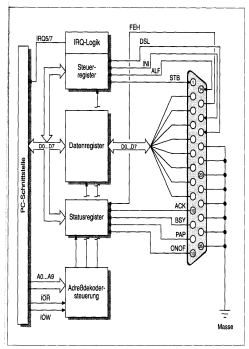
9 E/A-Steuerungen

- 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)



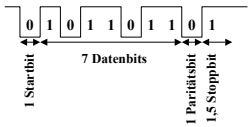
Die parallele Schnittstelle

- Verbindung zum Drucker
  - ⇨ 8 Bit Daten
  - ⇨ einfacher Aufbau
  - ⇨ normalerweise nur Schreiben
  - ⇨ bei Lesezugriff auf das Datenregister werden die Werte im Datenregister mit den momentan anliegenden Daten mit ODER Verknüpft



Serielle Datenübertragung

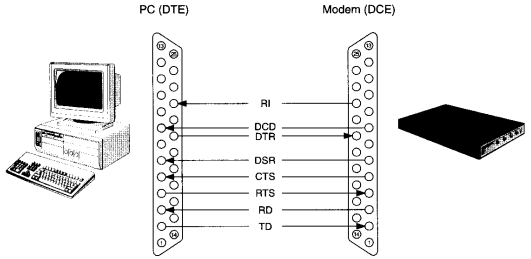
- Baud: Schrittgeschwindigkeit
- Aufbau einer Übertragungseinheit
  - ⇨ Startbit
    - Kennzeichnet den Anfang einer Übertragung
  - ⇨ Datenbits
    - das zu übertragende Datum
    - ASCII-Kodierung der Daten
  - ⇨ Paritätsbit
    - Prüfbit zum Feststellen der Korrektheit der Übertragung
    - gerade Parität: die Zahl der 1en wird zu einer geraden Anzahl ergänzt
  - ⇨ Stoppbit
    - Markiert das Ende einer Übertragungseinheit
- Das Startbit wird mit 8-facher Rate abgetastet



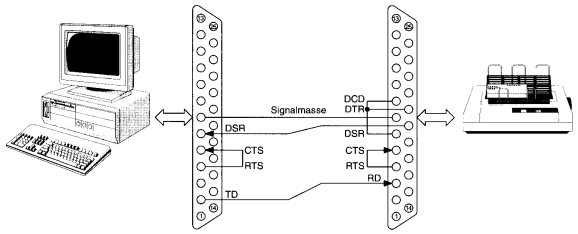
Die RS232-Schnittstelle

- 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

Anschluß eines Modems

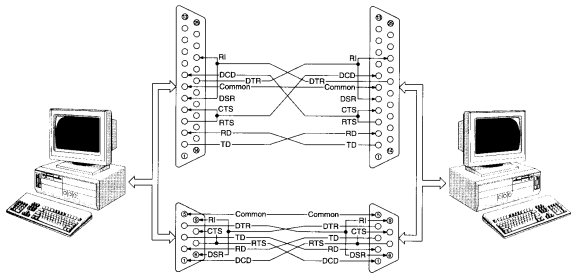


Anschluß eines Peripheriegeräts

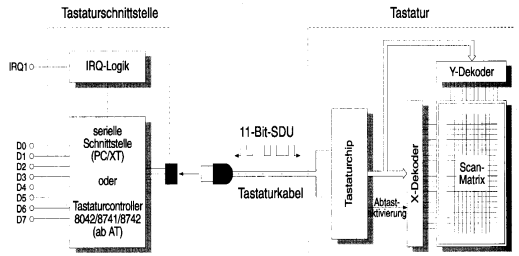


Verbindung zwischen zwei Computern

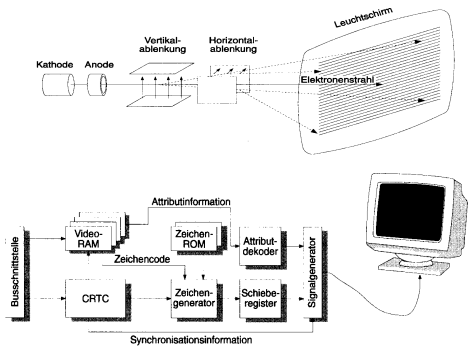
Link-Kabel



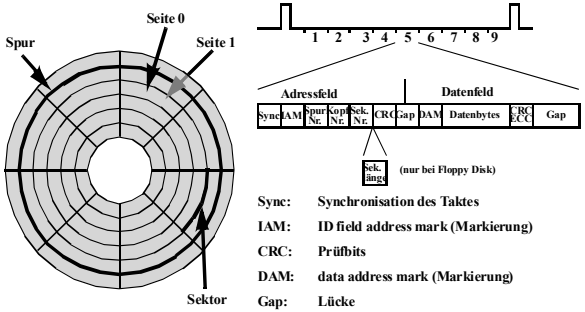
Tastatur



Graphikadapter



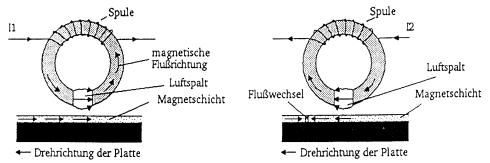
Sektoren einer Festplatte



Sync: Synchronisation des Taktes
IAM: ID field address mark (Markierung)
CRC: Prüfbits
DAM: data address mark (Markierung)
Gap: Lücke

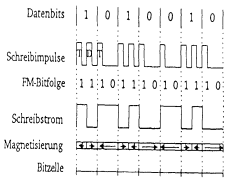
Prinzip der Datenspeicherung

- 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.



Das Frequenzmodulations-Verfahren (FM)

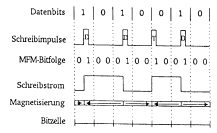
- **Prinzip: Zu Beginn jeder Bitzelle wird ein Taktimpuls T abgespeichert. Nur wenn der Inhalt gleich „1“ sein soll, folgt in der Mitte der Bitzelle das Datum D als weiterer Impuls.**
- **Dieses Verfahren ist relativ langsam und Speicherplatzintensiv, da in jeder Bitzelle mit dem Datenbit auch der Takt aufgezeichnet werden muss. Es wird auch als Format mit einfacher Schreibdichte (single density) bezeichnet.**



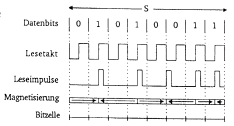
FM-Aufzeichnungsverfahren

Das modifizierte Frequenzmodulations-Verfahren (MFM)

- **Prinzip: es wird nur in solchen Zellen ein Taktimpuls abgelegt, in denen auch ein „1“-Datenbit gespeichert werden soll. Dadurch benötigt jede Bitzelle nur noch den halben Platz auf der magnetischen Oberfläche (double density Format).**
- **Soll eine „1“ geschrieben werden, wird ein positiver Datenimpuls D in der Mitte geschrieben. Bei einer „0“ wird ein Taktimpuls T am Anfang der Zelle abgelegt, wenn im Takt vorher nicht eine „1“ geschrieben wurde.**
- **Damit wird bei einer Folge von „0“-Bits. Der Takt am Anfang einer jeden Bitzelle abgespeichert und ermöglicht so die Synchronisation beim Lesen der Daten.**
- **Beim MFM-Format spricht man von einem Format mit doppelter Schreibdichte (double density).**



MFM-Aufzeichnungsverfahren

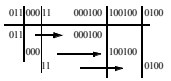


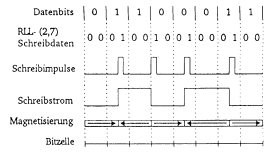
Rückgewinnung der Daten beim MFM-Verfahren

Das RLL-Verfahren

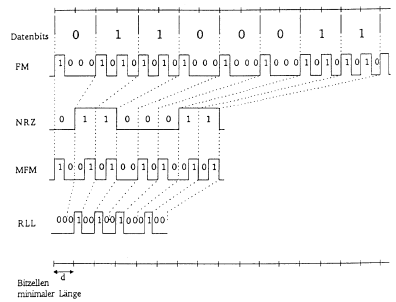
- **Ziel dieses Verfahrens ist, die Aufzeichnung von „0“-Läufen zu begrenzen. Dies wird durch eine geeignete Kodierung der Daten erreicht.**
- **RLL-(2,7) bedeutet, dass zwischen zwei „1“-Bits mindestens 2 jedoch höchstens 7 „0“-Bits liegen.**
- **Neben dem zu kodierenden Bit werden zusätzlich noch ein oder zwei folgende Bits berücksichtigt (kontextabhängig)**

Bitkombination Bit	Kontext	RLL-(2,7)-Code	
1	0	10	00
1	1	01	00
0	00	10	0100
0	10	00	1000
0	11	00	0100
0	010	00	001000
0	011	00	100100

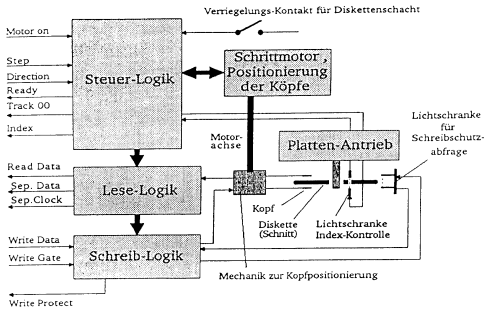




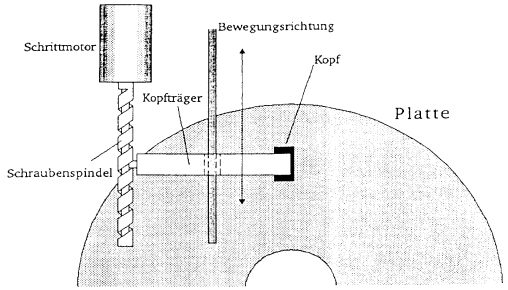
Vergleich des Speicherbedarfs der verschiedenen Aufzeichnungsverfahren



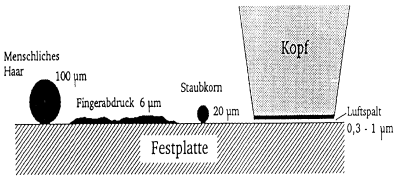
Aufbau eines Floppy-Disk-Laufwerks



Floppy

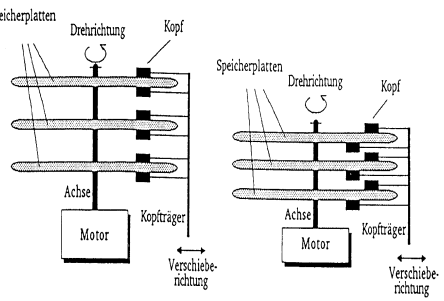


Größenverhältnisse im Festplatten-Laufwerk



Größenvergleich

Aufbau eines Festplatten-Laufwerks



Schematischer Aufbau einer Festplatte

Lokale Netzwerke

- **Zugriffsverfahren:**
  - **CSMA/CD**
    Carrier Sense Multiple Access / Collision Detect Eine Station, die Daten senden will muss zunächst auf das Übertragungsmedium hören. Ist dieses frei, darf sie sofort senden. Kommt es dennoch zu einer Kollision, weil zwei Stationen gleichzeitig senden wollen, so unterbricht sie die Übertragung und wartet eine zufällige Zeitspanne
  - **Token Passing**
    Ein freies Token durchläuft permanent den Ring. Nur die Station, die das Token besitzt darf senden
  - **Token Bus**
    Die Netztopologie ist ein Bus, die Zugriffsberechtigung wird über ein Token geregelt