1 Vorbemerkungen

1.1 Beispielliteratur

- John P. Hayes: "Computer Architecture and Organization". Mc Graw-Hill 1988
- John L. Hennesy and David A. Patterson: "Computer Architecture. A Quantitative Approach". Morgan Kaufmann 1993
- John L. Hennessy and David A. Patterson: "Computer Organization & Design. The Hardware/Software Interface". Morgan Kaufmann 1994
- W. Schiffmann und R. Schmitz: "Technische Informatik 1 und 2". Springer Lehrbuch 1992
- Manfred Broy: "Informatik. Grundlegende Einführung Teil II". Springer Lehrbuch 1993

1.2 Einordnung

Informatik = Information + Automatik

- Ingenieurwissenschaft
- immaterieller Werkstoff: Information
- Wissenschaft von der Verarbeitung von Information durch Computer
- Erforschung des Aufbaus und der Funktion solcher Anlagen
- Entwicklung von Verfahren zur Anwendung von Computern

Gliederung:

- Theoretische Informatik
- Technische Informatik
- Praktische Informatik
- Angewandte Informatik

1.3 Rechnerentwurfsebene

Digitale Datenverarbeitung: r: $N_1 \rightarrow N_2$

a_r sei ein zugehöriger Algorithmus

DVA (= Datenverarbeitungsanlage)

$E := code_E (N_1)$	*Eingabedaten*
$A := code_A (r(N_1))$	*Ausgabedaten*
$a := code (a_r)$	*Programm*
$f(E, a) \rightarrow A$	*beschreibt die von der DVA durchgeführte Transformation*

Problem: Wie muß DVA organisiert sein, um *f* leisten zu können?

DVA-System von verbundenen Komponenten

1.3.1 Darstellung durch einen Graphen

- Knoten K (Datenverarbeitende Komponenten)
- Kanten $\subseteq K \times K$ (jede Kante repräsentiert einen Kanal zum Nachrichtenaustausch)

1.3.2 Darstellung durch ein Blockdiagramm

System bestimmt durch

- Struktur
- Verhalten

Design - Problem:

geg.: gewünschtes Verhalten
Menge verfügbarer Komponenten

ges.: Struktur, bestehend aus diesen Komponenten mit gewünschtem Verhalten und minimale Kosten

Vorgehensweise:

- Zerlegung in kleinere Teilprobleme
- unabhängige Lösung der Teilprobleme

Design - Ebenen:

Einteilung A:	**Einteilung B:**
Material, physikalische Effekte	
Bauelemente	
Grundschaltungen	Gatter Ebene
Schaltnetze	Register-Transfer-Ebene
Schaltwerke	
Systemkomponenten	Prozessorebene
System	

Ebene	**Komponenten**	**IC-Dichte**	**Dateneinheit**	**Zeiteinheiten**
Prozessor	CPU, EAP Speicher I/O-Geräte	LSI, VLSI	Programme Dateien	10^{-3} - 10^{3} sec
Register-Transfer	Register, Schaltnetze Schaltwerke	MSI	Worte	10^{-9} - 10^{-6} sec
Gatter	logische Gatter	SSI	Bits	10^{-10} - 10^{-8} sec

SSI (= Small Scale Integration)

MSI (=Medium Scale Integration)

LSI (=Large Scale Integration)

VLSI (=Very Large Scale Integration)

2 Darstellung von Information

2.1 Information

Information: Grundbegriff der Informatik
(Kenntnis über Sachverhalte u. Vorgänge)
Definition nur durch andere, ebenfalls undefinierte Grundbegriffe möglich
⇒ häufig einfach als Grundbegriff eingeführt

Nachricht: dargestellte Information zum Zwecke der Übertragung
abstrakte Information wird auf konkrete Nachricht abgebildet
keine eindeutige Zuordnung (Sprachen)
zu einer Nachricht muß eine Interpretationsvorschrift existieren, die vom Empfänger unabhängig ist
(Bsp.: Gleiche Pressemeldung wird von verschiedenen Leuten verschieden interpretiert)

Daten: dargestellte Information zum Zwecke der Verarbeitung

Bemerkung: häufig werden Information, Nachricht und Daten synonym benutzt!

Darstellung von Information: Daten und Nachrichten

- digital (durch Zeichensysteme) z. B. binärer Zeichensatz
- analog (als kontinuierliche Funktion)

Hierarischer Zeichensatz:

- Bit Bitkette
- Byte Zeichenkette

...

2.2 Darstellung von Text in Rechnern

Alphanumerische Texte sind Folgen von Zeichen

a, b, ..., z, A, B, ..., Z, 0, 1, ..., 9, , ;, ... usw.

Die einzelnen Zeichen werden codiert. Heute sind **8 Bits = 1 Byte** üblich.

Häufig wird **ASCII-Code** (=American Standard Code for Information Interchange) benutzt.

Deshalb sind die Wortlängen heutiger Rechner Vielfache von Bytes (8, 16, 32, ...). Wort ist hier die Informationseinheit, die ein Rechner bei der Verarbeitung intern benutzt.

Dadurch verhütet man, daß Zeichen auf mehrere Wörter verteilt werden oder bei Text Bits verschwendet werden.

2.3 Zahlendarstellung

2.3.1 Zahlensysteme

Def.: Ein polyadisches Zahlensystem mit der Basis $B \in N$, $B > 1$ ($B, b \in N_0$), ist die Darstellung jeder reellen Zahl a als Summe von gewichteten Potenzen von B

$$a = \pm \sum_{i=-\infty}^{n} b_i B^i \qquad b_i \in \{0,1,2,\ldots B-1\}$$

(b_i heißen Ziffern)

Jede reelle Zahl läßt sich derart darstellen. Insbesondere gibt es zu jeder natürlichen Zahl eine Darstellung

$$m = \pm \sum_{i=0}^{n} b_i B^i$$

Die Darstellung ist nicht notwendigerweise eindeutig:

$$1 = 1.10^0 = \sum_{i=-\infty}^{-1} 9.(10)^i = 0{,}999\ldots.$$

Deshalb betrachten wir die Menge

$$D(B) = \left\{ a | a \in R, a = \pm \sum_{i=-m}^{n} b_i B^i ; m,n \in N_0 \right\}$$

Satz:

$$D(B) = \left\{ a | a \in R, \exists k,l \in N_0 : a = k^{-1} l \right\}$$

Beweis:

$$a = \sum_{i=-m}^{n} b_i B^i = \underbrace{B^{-m}}_{B^m = k} \underbrace{\sum_{i=0}^{m+n} b_i B^i}_{l}$$

Bemerkung: D(10) ≠ D(7), da

z. B. $0,\overline{142857} = 1,7^{-1}$

$0,\overline{142857}$ ist aber unendlich, was der Definition von D(B) widerspricht!

Normierung: $b_n \neq 0$, falls $n > 0$ und (führende Nullen streichen)

$b_{-m} \neq 0$ falls $m \geq 0$ (nachfolgende Nullen streichen)

lediglich $0 = 0.B^0$

für $x \in$ D(B) ist die normierte Darstellung eindeutig

Wichtige Zahlensysteme:

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

Unterschied Dual-Binär:

- Dualsystem ist ein Zahlensystem
- Zur Darstellung der Dualziffern "0" und "1" benötigt man ein Binärsystem (Informationsdarstellung mittels Zeichen)

Sei $x \in D(B)$ mit der normierten Summe

$$x = \pm \sum_{i=-m}^{n} b_i B^i$$

$(b_n b_{n-1} ... b_0 b_{-1} .. b_{-m})_B$ bzw.

$(b_n b_{n-1} ... b_0)_B$ bzw. für $n < 0$

$(0,000...0 b_n b_{n-1} ... b_{-m})_B$

Die Klammern werden häufig auch weggelassen!

Bsp.: $17_{10} = 15_{12} = 11_{16} = 21_8 = 10001_2$

2.3.1.1 Konvertierung:

geg.: eine Zahl

ges.: Darstellung im System mit der Basis B

2.3.1.1.1 sukzessive Division mit Rest:

2.3.1.1.1.1 natürliche Zahl

$$s \in N_0: \; s = \sum_{i=0}^{n} b_i B^i = b_n B^n + b_{n-1} B^{n-1} + \ldots + b_2 B^2 + b_1 B^1 + b_0$$

$$s = \underbrace{\left(\underbrace{\left(\underbrace{\left(\ldots \left(\left(b_n B + b_{n-1} \right) + b_{n-2} \right) B + \ldots + b_3 \right) B + b_2}_{s_2} \right) B + b_1}_{s_1} \right) B + b_0}_{s_0}$$

$$s = s_0 B + b_0 = \underbrace{\left(s_1 B + b_1 \right)}_{s_0} B + b_0 = \underbrace{\left(\left(s_2 B + b_2 \right) B + b_1 \right)}_{s_1} B + b_0$$

⇒ **Vorgangsweise:**

$s:B = s_0$ b_0 Rest

$s_0:B = s_1$ b_1 Rest

...

$s_{n-2}:B = s_{n-1}$ b_{n-1} Rest

$s_{n-1}:B = 0$ b_n Rest

⇒ $s = (b_n b_{n-1} \ldots b_1 b_0)_B$

Bsp.: $23_{10} = ?_2$

23:2 = 11 1 Rest

11:2 = 5 1 Rest

5:2 = 2 1 Rest

2:2 = 1 0 Rest

1:2 = 0 1 Rest

⇒ $23_{10} = (10111)_2$

Bsp.: $(10111)_2 = ?_{10}$

10111:1010 = 10 11 Rest (entspricht 3_{10})

10:1010 = 0 1 Rest (entspricht 2_{10})

$\Rightarrow (10111)_2 = 23_{10}$

2.3.1.1.1.2 echt gebrochene Zahl |q| < 1

$$q = \sum_{-m}^{-1} b_i B^i = b_{-1}B^{-1} + b_{-2}B^{-2} \ldots$$

$$= \underbrace{B^{-1}\left(b_{-1} + B^{-1}\left(b_{-2} + B^{-1}\left(b_{-3}+\ldots\right)\right)\right)}_{0,b_{-1}b_{-2}\ldots}$$

$$qB = b_{-1} + \underbrace{B^{-1}\left(b_{-2} + B^{-1}\left(b_{-3}+\ldots\right)\right)}_{q_{-1}}$$

$$q_{-1}B = b_{-2} + \underbrace{B^{-1}\left(b_{-3}+\ldots\right)}_{q_{-2}}$$

Bsp.:

$0,13_{10} = ?_2$

0,13.2 = 0 + 0,26

0,26.2 = 0 + 0,52

0,52.2 = 1 + 0,04

0,04.2 = 0 + 0,08

0,08.2 = 0 + 0,16

0,16.2 = 0 + 0,32

0,32.2 = 0 + 0,64

0,64.2 = 1 + 0,28

...

i. a. nicht endlich umwandlbar!

$\Rightarrow 0,13_{10} = (0,00100001)_2$

2.3.1.1.1.3 beliebig gebrochene Zahl

$$x = \sum_{-m}^{n} b_i B^i = \underbrace{\sum_{0}^{n} b_i Bi}_{s \in N_0} + \underbrace{\sum_{-m}^{-1} b_i B^i}_{q}$$

⇒ vorderen Teil als natürliche Zahl und hinteren Teil als echt gebrochene Zahl behandeln.

Anschließend einfach addieren.

2.3.1.1.2 Sukzessive Multiplikation mit Addition

$s \in N_0$

Konvertierung $B \rightarrow B^*$

$b_i = (a_{r-1}a_{r-2}...a_0)_{B^*} = \underline{b}_i$

$$s = \sum_{i=0}^{n} b_i B^i = \underline{b}_n \underline{B}^n + \underline{b}_{n-1} \underline{B}^{n-1} \ldots = \Big(\big(\ldots\big(\big(\underline{b}_n \underline{B} + \underline{b}_{n-1}\big)\underline{B} + \underline{b}_{n-2}\big)\underline{B} + \ldots + \underline{b}_2\big)\underline{B} + \underline{b}_1\Big)\underline{B} + \underline{b}_0$$

Bsp.:

$(111001)_2 = ?_{10}$

ersetze jede Stelle durch die Zahl mit der Basis 10:

111001 ⇒ 1 1 1 0 0 1 (da $1_2 = 1_{10}$ und $0_2 = 0_{10}$)

dann von vorne beginnend, einmal multiplizieren (mit der neue Basis 2) und einmal addieren!

1.2 = 2

2 + 1 = 3

3.2 = 6

6 + 1 = 7

7.2 = 14

14 + 0 = 14

14.2 = 28

28 + 0 = 28

28.2 = 56

56 + 1 = 5710

somit ist $(111001)_2 = 5710_{10}$

Bsp.:

$57_{10} = ?_2$

ersetze jede Stelle durch die Zahl mit der Basis 2:

$57 \Rightarrow 101\ 111$ (da $5_{10} = 101_2$ und $7_{10} = 111_2$)

dann von vorne beginnend, einmal multiplizieren (mit der neuen Basis $10_{10} = 1010_2$) und einmal addieren!

$101.1010 = 110010$

$110010 + 111 = 111001_2$

2.3.1.1.3 Konvertierung von 2er Potenzen:

$$a = \sum_{i=0}^{n} b_i B^i \qquad (B = 2^r)\ (0 \le b_i \le 2^r - 1)$$

$$b_i = \sum_{j=0}^{r-1} a_{ij} 2^j \qquad (j \le r - 1)$$

eventuell führende Nullen behalten!

$$a = \sum_{i=0}^{n} \left(\sum_{j=0}^{r-1} a_{ij} 2^j \right) 2^{ri} = \sum_{i=0}^{n} \left(\sum_{j=0}^{r-1} a_{ij} 2^{ri+j} \right) = \sum_{k=0}^{nr+r-1} c_k 2^k$$

wobei $c_k = a_{ij}$ eindeutig ist, falls $k = ri + j$

Beweis:

k = 0	i = 0	j = 0
k = 1	i = 0	j = 1
...	...	...
k = r - 1	i = 0	j = r - 1
k = r	i = 1	j = 0
k = r + 1	i = 1	j = 1
...	...	...
k = 2r - 1	i = 1	j = r - 1
k = 2r	i = 2	j = 0
...	...	...

Bsp.: $7851_{16} = ?_2$ (da Basis $16 = 2^4 \Rightarrow$ 4 Stellen)

7				8				5				1			
0	1	1	1	1	0	0	0	0	1	0	1	0	0	0	1

$\Rightarrow 111100001010001_2$

$111100001010001_2 = ?_8$ (da $8 = 2^3 \Rightarrow$ 3 Stellen)

1	1	1	1	0	0	0	0	1	0	1	0	0	0	1
		7			4			1			2			1

$\Rightarrow 74121_8$

Wenn man also vom 16er System in das 8er System wechseln will, so geschieht das über den "Umweg" des 2er Systems.

2.3.2 Darstellung von Zahlen im Rechner

nur eine endliche Menge von Zahlen ist darstellbar!

2.3.2.1 Ganze Zahlen

Annahme: es stehen n Bits zur Darstellung von ganzen Zahlen zur Verfügung (1 Wort) $\Rightarrow$ höchstens 2^n Zahlen können dargestellt werden.

Absicht: Teilmenge der ganzen Zahlen soll repräsentiert werden und zwar soll etwa die Hälfte negativ sein!

In der Mathematik werden üblicherweise negative Zahlen mittels Vorzeichen dargestellt:

Nachteile: gesonderte Vorzeichenbehandlung $\Rightarrow$ eigenes Subtraktionswerk

deshalb: Rückführung der Subtraktion auf Addition durch Komplementierung

$a - b = a + K_0(b) - K$ (mit $K_0(b) = K - b$)

K muß dabei einfach sein, damit K - b nichts kostet!

Sei $b > 0$ und b Dualzahlen mit n Bits im erlaubten Bereich, so ist

$-b = K_0(b)$ mit n Bits

Bemerkung: erlaubter Bereich muß so gewählt werden, daß Eindeutigkeit gewährleistet ist:

$K_0(K_0(b)) = b$

Wahl von K:

1-Komplement: $K = 2^n - 1$ größte darstellbare Zahl

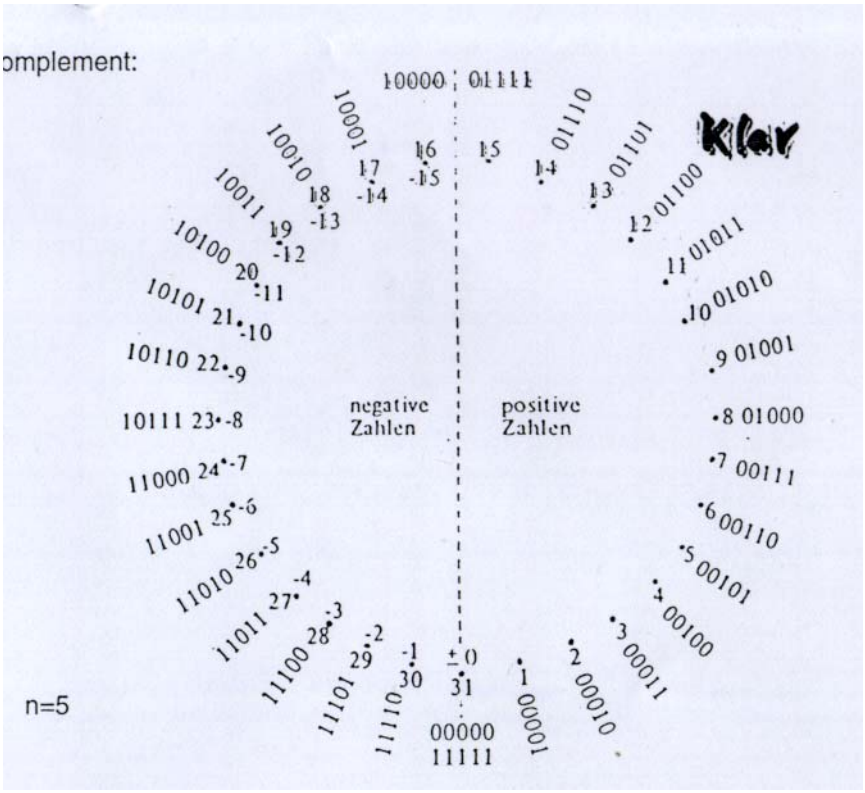
2-Komplement: $K = 2^n$ größte darstellbare Zahl

Berechnung des Komplements:

1-Komplement: bitweises "umklappen" der Stellen (aus 0 wird 1 und aus 1 wird 0)

2-Komplement: bilde 1-Komplement und addiere dazu 1.

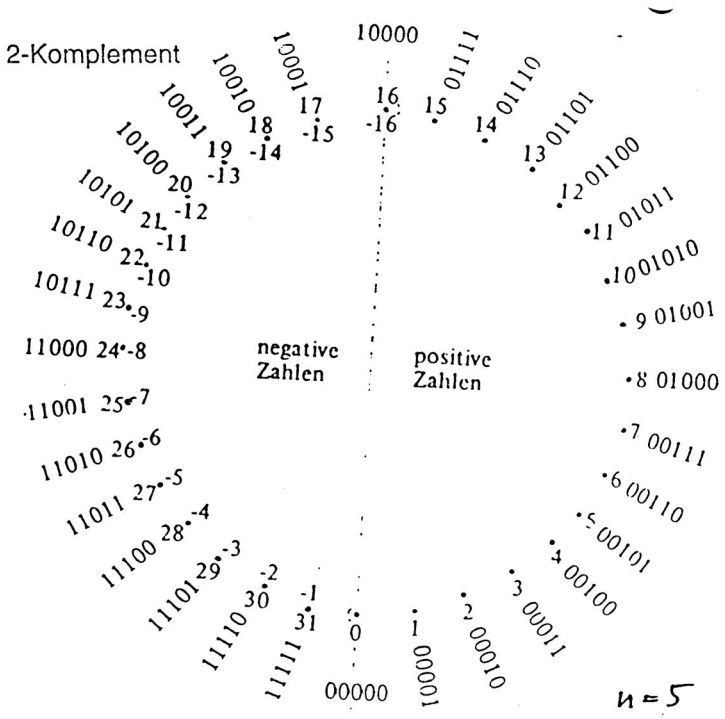
2.3.2.1.1 1-Komplement:



Bemerkung:

- alle negativen Zahlen haben an der ersten Stelle eine "1".
- Darstellung der Null ist nicht eindeutig: es gibt somit eine 0...0 = +0 und 1...1 = -0!

2.3.2.1.2 2-Komplement:



Bemerkung:

- Unsymmetrie: eine negative Zahl ist mehr darstellbar
- Aufwand liegt bei der Komplementenbildung
- Null ist nun eindeutig

z. B.: $K_2(01010) = K_1(01010) + 00001 = 10101 + 00001 = 10110$

Bsp.:

$5_{10} = 00101_2$

$9_{10} = 01001_2$

$K_2(00101) = 11011$

$K_2(01001) = 10111$

5 + 9:	00101	9 - 5:	01001
	01001		11011
	01110 (=14_{10})		00100 (= 4_{10})
5 - 9:	00101	-5 - 9:	11011
	10111		10111
	11100 (= -4_{10})		10010 (-14_{10})

2.3.2.2 Bereichsüberschreitungen

2.3.2.2.1 2-Komplement

Das 2-Komplement erlaubt nur die Darstellung von ganzen Zahlen z mit
$-2^{n-1} \le z \le 2^{n-1} - 1$

Beispiel:
n = 6: ($-32 \le z \le 31$)

-27_{10}	100101_2
-8_{10}	111000_2
-35_{10}	$\not{1}011101_2$

Es kommt zu einer Bereichsüberschreitung! Beim 2-Komplement kann diese einfach unter den Tisch fallen gelassen werden. Dann jedoch kommt es dazu, daß 2 negative Zahlen (führende 1) nicht zu einem positiven Ergebnis (führende 0) führen kann. Sowohl in diesem Fall als auch in dem umgekehrten (2 positive Zahlen werden negativ) muß somit das Ergebnis verworfen werden.

2.3.2.2.2 1-Komplement

Kommt es zu einem Überlauf, so muß die 1 (Überlauf) hinten dazu addiert werden (= Rücklauf). Dann wie beim 2-Komplement die Vorzeichen auf Korrektheit überprüfen.

2.3.2.3 Rationale Zahlen

2.3.2.3.1 Festpunktdarstellung

Zahlen werden als n-stellige Dualzahlen bzw. Komplemente dargestellt. Es gibt eine fixe Anzahl von Vor- und Nachkommastellen.

Seien r Bits vor und s Bits (n = r + s) nach dem Komma vorgesehen für die Darstellung

$$a_{r-2}...a_0a_{-1}...a_{-s} = \pm 2^{-s} \sum_{i=0}^{r+i-2} a_{i-s} 2^i$$

Beispiel:
Stelle $-1{,}5_{10}$ im Dualsystem mit 5 Vor- und 3 Nachkommastellen dar.
$1{,}5_{10} \Rightarrow 00001{,}100_2 \Rightarrow$ 1-Komplement: $11110{,}011_2 \Rightarrow +00000{,}001_2$:
$-1{,}5_{10} = 11110{,}100_2$

Der Nachteil der Festpunktdarstellung liegt darin, daß viele rationale Zahlen nur unbefriedigend dargestellt werden können. Es werden nämlich oft signifikante Stellen einfach abgeschnitten.

2.3.2.3.2 Gleitpunkt-Darstellung (floating point)

Jede Zahl x wird dargestellt als

$x = M.B^E$ (Mantisse, Basis, Exponent)

für B ist 2 oder 16 gebräuchlich

Stehen e Stellen für den Exponent zur Verfügung, so wird meist die Charakteristik

$$C = E + \frac{1}{2}B^e$$

angegeben.

Ad Beispiel: B = 2 und e = 8:

$$C = E + \frac{1}{2}B^e = E + \frac{1}{2}2^8 = E + 2^7$$

steht ein **Exponent** von
-128_{10} bis 127_{10} bzw.

Eine **Charakteristik** von
0_{10} bis 255_{10} zur Verfügung.

Standard:
1 Bit Vorzeichen
8 Bit Exponent
23 Bit Mantisse

<u>Probleme beim Rechnen:</u>

- *Addition, Subtraktion:* Charakteristiken müssen vorher angeglichen werden
- *Multiplikation, Division:* Angleichung nicht nötig, aber mehrmalige Multiplikationen lassen die relevanten Bits in der Mantisse immer mehr nach rechts wandern

Somit wird eine Normierung notwendig, die allerdings sehr zeitaufwendig ist. Dieser Standard ist (z. B. IBM) weit verbreitet. Es wird eine nicht vorhandene Genauigkeit vorgegaukelt.

2.3.2.3.3 IEEE-Format

32-Bit:

$z = (-1)^V 2^{C'-127}(1,M')_2$ mit $0 < C' < 255$

V: 1 Bit
C': 8 Bit
M': 23 Bit

Beispiel $-1,5_{10}$:
$-1,5_{10} = 1,1_2.2^0$
V = 1 (da negative Zahl)
C' - 127 = 0 ⇒ C' = 127
M' = ,10000000000000000000000
⇒
1 01111111 10000000000000000000000

Spezialvereinbarungen:

C' = 0:	M' = 0:	z = 0
	M' ≠ 0:	Zahl ist zu klein
0 < C' < 255:		Zahl ist im erlaubten Bereich
C' = 255:	M' ≠ 0:	ungültiges Ergebnis (z. B. Division durch Null)
	M' = 0:	z = ± ∞

64-Bit:

$z = (-1)^V 2^{C'-1023}(1,M')$

V: 1 Bit
C': 11 Bit
M': 52 Bit

3 Gatter-Ebene

Verknüpfung (Verarbeitung) binärer Daten mittels binärer Operationen zu neuen binären Daten

3.1 Boole'sche Algebra

Eine Menge B zusammen mit den Verknüpfungen "+" und "." heißt Boole'sche Algebra, wenn für a, b, c ∈ B gilt:

1.	ab ∈ B a + b ∈ B	Abgeschlossenheit
2.	es gibt Elemente 0,1 ∈ B a + 0 = a a 1 = a	Existenz von 0 und 1
3.	a + b = b + a ab = ba	Kommutativgesetz
4.	a (b + c) = ab + ac a + (bc) = (a + b)(a + c)	Distributivgesetz
5.	a + (b + c) = (a + b) + c a(bc) = (ab)c	Assoziativgesetz
6.	zu jedem a existiert ein $\bar{a}$ $a\bar{a} = 0$ $a + \bar{a} = 1$	Existenz des Komplements

Weiterhin gelten:

7.	a + a = a aa = a	Idempotenzgesetz
8.	$\overline{a + b} = \bar{a}\bar{b}$ $\overline{ab} = \bar{a} + \bar{b}$	De Morgan's Gesetz
9.	$\bar{\bar{a}} = a$	Involution

SCHALTALGEBRA

Sie ist eine spezielle Boole'sche Algebra über der Menge {0, 1} mit den Verknüpfungen UND, ODER und NICHT.

X	Y	XY (X UND Y)	X + Y (X ODER Y)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

X	$\bar{X}$ (NICHT X)
0	1
1	0

Boole'scher Ausdruck:

1. 0 und 1 sind Konstanten
2. Variable x ist ein Symbol, das für einen der beiden Werte 0 oder 1 steht
3. BA = Menge aller Boole'schen Ausdrücke
   - 3.1. Konstante, Variable $\in$ BA
   - 3.2. A, B $\in$ BA $\Rightarrow$ (A + B), (AB), $(\overline{A}) \in$ BA

Dualitätsprinzip der Boole'schen Algebra:

Wenn ein Ausdruck (Gleichung) wahr ist, dann ist auch der duale Ausdruck wahr. Der duale Ausdruck ergibt sich, indem man "+" und "." sowie 1 und 0 vertauscht.

3.2 Schaltfunktionen

Vektor(schalt)funktion (combinational network):
f: $\{0, 1\}^n \rightarrow \{0, 1\}^s$

Schaltfunktion (combinational function):
f: $\{0,1\}^n \rightarrow \{0, 1\}$

Vektorschaltfunktionen lassen sich als s Schaltfunktionen mit gleichen Eingabevariablen auffassen.

3.2.1 Darstellung von Schaltfunktionen

3.2.1.1 Darstellung durch Wahrheitstabelle

Diese spezifiziert zu jeder Kombination (x_1, x_2, ..., x_n) die Ausgabe (y_1, y_2, ..., y_n).

x_1	x_2	x_3	y_1	y_2
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

Tabellengröße für Schaltfunktionen:
2^n Zeilen mit je zwei möglichen Ausgabewerten: 0 oder 1

3.2.1.2 Darstellung durch Boole'sche Ausdrücke

Wie gewinne ich zu einer tabellarisch gegebenen Schaltfunktion einen äquivalenten Boole'schen Ausdruck?

3.2.1.3 Darstellung durch Minterme

Ein Minterm m ist ein Ausdruck der Form

$$m = x_1^{\delta_1} x_2^{\delta_2} \dots x_n^{\delta_n}$$

wobei $\delta_i \in \{0, 1\}$ und

$$x_i^{\delta_i} = \begin{cases} x_i & \text{bei } \delta_i = 1 \\ \overline{x}_i & \text{bei } \delta_i = 0 \end{cases}$$

Beispiel:

$$A(\vec{x}) = x_1 \overline{x}_2 x_3 = \begin{cases} 1 & \text{wenn } x_1 = 1, x_2 = 0, x_3 = 1 \\ 0 & \text{sonst} \end{cases}$$

Ordne nun jedem $\vec{x}_v \in \{0,1\}^n$ den Minterm m_v zu, für den gilt:

$$m_v(\vec{x}_v) = 1$$

Jede Schaltfunktion f: $\{0, 1\}^n \rightarrow \{0, 1\}$ läßt sich eindeutig durch einen Ausdruck aus disjunktiv (durch ODER) verknüpften Mintermen der Variablen x_1, x_2, ..., x_n darstellen (Disjunktive Kanonische Form)

$$f_{DKF}(x_1, x_2, \dots, x_n) = \sum_{\substack{f(\vec{x}_v)=1 \\ m_v(\vec{x}_v)=1}} m_v$$

Beispiel:

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

$$\Rightarrow f_{DKF} = \overline{x}_1 \overline{x}_2 \overline{x}_3 + \overline{x}_1 x_2 x_3 + x_1 x_2 \overline{x}_3$$

Es existieren i. a. einfachere Ausdrücke, wie z. B. die Disjunktive Normalform, welche eine disjunktive Verknüpfung von Konjunktionen darstellt, wobei δ_i angibt, ob die Variable x_i direkt ($\delta_i = 1$), als Negation ($\delta_i = 0$) oder nicht ($\delta_i = 2$) auftritt.

$$f_{DNF}(x_1, x_2, \dots x_n) = \sum \left(x_1^{\delta_1} x_2^{\delta_2} \dots x_n^{\delta_n} \right)$$

f_{DNF} ist nicht mehr eindeutig zu f!

3.2.1.4 Darstellung durch Maxterme

Ein Maxterm m ist ein Ausdruck der Form

$$m = x_1^{\delta_1} + x_2^{\delta_2} + \ldots + x_n^{\delta_n}$$

wobei $\delta_i \in \{0, 1\}$ und

$$x_i^{\delta_i} = \begin{cases} x_i & \text{bei } \delta_i = 0 \\ \overline{x}_i & \text{bei } \delta_i = 1 \end{cases}$$

Beispiel:

$$A(\vec{x}) = x_1 + \overline{x}_2 + x_3 = \begin{cases} 0 & \text{wenn } x_1 = 1, x_2 = 0, x_3 = 1 \\ 1 & \text{sonst} \end{cases}$$

Jede Schaltfunktion f: $\{0, 1\}^n \rightarrow \{0, 1\}$ läßt sich eindeutig durch einen Ausdruck aus konjunktiv (durch UND) verknüpften Maxtermen der Variablen x_1, x_2, ..., x_n darstellen (Konjunktive Kanonische Form)

$$f_{KKF}(x_1, x_2, \ldots, x_n) = \prod_{\substack{f(\bar{x}_v)=0 \\ m_v(\bar{x}_v)=0}} m_v$$

ad Beispiel:

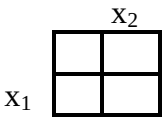
x_1	x_2	x_3	f	
0	0	0	1	
0	0	1	0	←
0	1	0	0	←
0	1	1	1	
1	0	0	0	←
1	0	1	0	←
1	1	0	1	
1	1	1	0	←

$$f_{KKF} = (x_1 + x_2 + \overline{x}_3)(x_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_1 + x_2 + x_3)(\overline{x}_1 + x_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3)$$

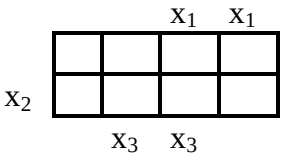
3.2.1.5 Repräsentation einer Schaltfunktion mit KV-Diagramm

Ein Karnaugh-Veitch-Diagramm (KV-Diagramm) ist die grafische Darstellung einer Wertetabelle oder Schaltfunktion. Das Diagramm ist in Matrixform angeordnet und jedes Matrixfeld entspricht einem Minterm.

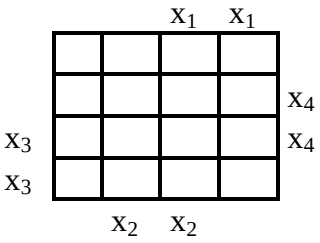
mit 2 Variablen:



mit 3 Variablen:



mit 4 Variablen:



Eine Verknüpfungsbasis ist eine Menge von primitiven Verknüpfungen v_i: $\{0, 1\}^2 \rightarrow \{0, 1\}$ mit deren Hilfe sich alle Schaltfunktionen darstellen lassen.

(+, ., Negation) ist eine Verknüpfungsbasis.

Behauptung: NAND(x,y) = (x | y) = $\overline{xy}$ (Sheffer-Funktion) ist eine Verknüpfungsbasis

Beweis: es genügt zu zeigen, daß Negation, UND und ODER durch NAND darstellbar sind:

Negation $f(x) = \bar{x} = (\overline{xx}) = x|x = NAND(x, x)$

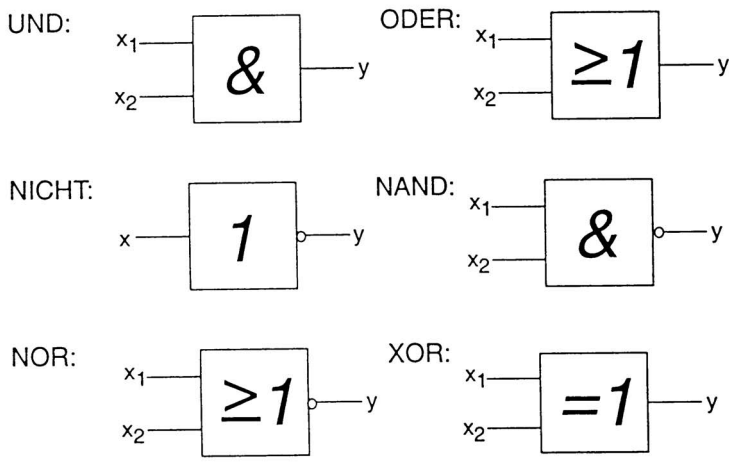
UND $f(x, y) = xy = xy + xy = \overline{\overline{xy + xy}} = \overline{\overline{xy}\,\overline{xy}} = \overline{(x|y)(x|y)} = (x|y)|(x|y)$

ODER $f(x, y) = x + y = \overline{\overline{x + y}} = \overline{\bar{x}\bar{y}} = \bar{x}|\bar{y} = (x|x)|(y|y)$

3.3 Schaltnetze

Die Realisierung einer Schaltfunktion heißt kombinatorischer Schaltkreis oder Schaltnetz. Schaltnetze werden aus Standardkomponenten, den sogenannten Gattern, aufgebaut. Gatter realisieren primitive Schaltfunktionen.

Grundgatter:



Alle Grundgatter (bis auf NICHT) lassen sich zu Gattern mit k Eingängen erweitern. In der Praxis ist k mit 8 beschränkt.

Mittels der Gatter lassen sich alle Schaltnetze realisieren.

Ein Schaltnetz heißt wohlgeformt wenn
- es azyklisch ist (es enthält keine geschlossene Schleife)
- Ausgänge sind nicht direkt verbunden

Die Tiefe oder die Anzahl der Stufen eines Schaltnetzes ist gegeben durch die Maximalanzahl von Gattern, die auf einem Weg zwischen einem Eingang und einem Ausgang liegen.
Ein Schaltnetz mit der Tiefe k heißt k-stufig.

3.4 Minimierung

Zu jeder Funktion f gibt es mehrere Ausdrücke in DNF, die f repräsentieren. Minimierung ist das Ermitteln eines möglichst einfachen und kurzen Ausdrucks, der zu einem gegebenen Ausdruck äquivalent ist, d. h. der die gleiche Schaltfunktion darstellt.

Beispiel:
abc + ab

Minimierung:
abc + ab = abc + ab1 = ab(c + 1) = ab

Somit ist ab die Minimierung von abc + ab

Da man sowohl nach Preisen als auch nach Technologie (als Beispiele) minimieren kann, gibt es kein einheitliches Maß, nach dessen minimiert werden sollte.

3.4.1 Verfahren von Quine-McCluskey

(Dieses Verfahren wird an Hand eines Beispieles erklärt. Siehe auch Übung Bsp. 5.5)

Gegeben:
Ein Ausdruck f_A, der die Funktion f darstellt.

$$f_A = ab + \bar{a}\bar{b}c + b\bar{c}d + a\bar{b}\bar{d} + \bar{a}\bar{b}\bar{c}$$

1. Schritt:
Bilde aus f_A die DKF
durch Erweiterungen der Art

$$ab = abc + ab\bar{c} = abcd + abc\bar{d} + ab\bar{c}d + ab\bar{c}\bar{d}$$

erhält man

$$f_{DKF} = abcd + abc\bar{d} + ab\bar{c}d + ab\bar{c}\bar{d} + a\bar{b}c\bar{d} + a\bar{b}\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}cd + \bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}\bar{c}\bar{d}$$

2. Schritt:
Bestimmung der Primimplikanden
Beginne mit P = {} und T = {Minterme}

1. Zerlege T so in Klassen K_i, daß K_i aus allen Termen, die genau i nichtnegierte Variablen enthalten, besteht
   Setze T = {}
2. Versuche, gemäß Grundidee, Elemente der Klasse K_{i-1} und K_i zu verschmelzen. Jedes Elementpaar, das verschmolzen wird, wird abgehakt. Das Ergebnis wird mit T vereinigt. Solange bis keine Verschmelzung mehr möglich ist.
3. P = P $\cup$ {alle Terme, die in 2. Nicht abgehakt wurden}
4. Ist T = {}, dann fertig. P = {Primimplikanden} sonst weiter bei 1.

Klasse	Gewicht der Minterme	Minterme	Verschm.		Verschm.	
			versch. Minterme	neue Terme mit 3 Variablen	versch. Minterme	neue Terme mit 2 Variablen
K_0	0	$\bar{a}\bar{b}\bar{c}\bar{d}$ ✓				
K_1	1	$\bar{a}\bar{b}\bar{c}d$ ✓	0,1	$\bar{a}\bar{b}\bar{c}$ ✓	0,1-2,3	$\bar{a}\bar{b}$
	2	$\bar{a}\bar{b}c\bar{d}$ ✓	0,2	$\bar{a}\bar{b}\bar{d}$ ✓	0,2-1,3	$\bar{a}\bar{b}$
	8	$a\bar{b}\bar{c}\bar{d}$ ✓	0,8	$\bar{b}\bar{c}\bar{d}$ ✓	0,8-2,10	$\bar{b}\bar{d}$
K_2	3	$\bar{a}\bar{b}cd$ ✓	1,3	$\bar{a}\bar{b}d$ ✓	8,10-12,14	$a\bar{d}$
	5	$\bar{a}b\bar{c}d$ ✓	1,5	$\bar{a}\bar{c}d$	8,12-10,14	$a\bar{d}$
	10	$a\bar{b}c\bar{d}$ ✓	2,3	$\bar{a}\bar{b}c$ ✓		
	12	$ab\bar{c}\bar{d}$ ✓	2,10	$\bar{b}c\bar{d}$ ✓		
			8,10	$a\bar{b}\bar{d}$ ✓		
			8,12	$a\bar{c}\bar{d}$ ✓		
K_3	13	$ab\bar{c}d$ ✓	5,13	$b\bar{c}d$	12,13-14,15	ab
	14	$abc\bar{d}$ ✓	10,14	$ac\bar{d}$ ✓	12,14-13,15	ab
			12,13	$ab\bar{c}$ ✓		
			12,14	$ab\bar{d}$ ✓		
K_4	15	$abcd$ ✓	13,15	abd ✓		
			14,15	abc ✓		

Als Primimplikanden erhält man nun alle nicht abgehakten Terme:

$\bar{a}\bar{c}d, b\bar{c}d, \bar{a}\bar{b}, \bar{b}\bar{d}, a\bar{d}, ab$

3. Schritt:

Bestimmung der wesentlichen Primimplikanden.

Stelle eine Tabelle auf, wobei für jeden Minterm eine Spalte und für jeden Primimplikanden eine Zeile aufgebaut wird. Falls der Primimplikand den Minterm enthält, so setze in die Spalte ein X.

	0	1	2	3	5	8	10	12	13	14	15
$\bar{a}\bar{c}d$		X			X						
$b\bar{c}d$					X				X		
$\bar{a}\bar{b}$	X	X	X	X							
$\bar{b}\bar{d}$	X		X			X	X				
$a\bar{d}$						X	X	X		X	
ab								X	X	X	X

Streichen aller dominierenden Primimplikanden;
Streiche jene Spalte und Zeile, in der jeweils nur 1 Stern steht!

	0	1	2	3	5	8	10	12	13	14	15
$\bar{a}\bar{c}d$		X			X						
$b\bar{c}d$					X				X		
$\bar{a}\bar{b}$	X	X	X	X							
$\bar{b}\bar{d}$	X		X			X	X				
$a\bar{d}$						X	X	X		X	
ab								X	X	X	X

Weiters kann man nun keine Streichungen mehr vornehmen. Daher faßt man die übrig gebliebenen Minterme in einer Restmatrix zusammen. Man sieht nun, daß die Primimplikanden $\bar{a}\bar{c}d$ und $b\bar{c}d$ durch den Minterm 5 überdeckt werden. Die Primimplikanden $\bar{b}\bar{d}$ und $a\bar{d}$ können durch die Minterme 8 oder 10 überdeckt werden.

	5	8	10
$\bar{a}\bar{c}d$	X		
$b\bar{c}d$	X		
$\bar{b}\bar{d}$		X	X
$a\bar{d}$		X	X

Die Lösung (f_{DMF}) setzt sich nun aus den wesentlichen (gestrichenen) Primimplikanden und dem übriggebliebenem Rest zusammen.

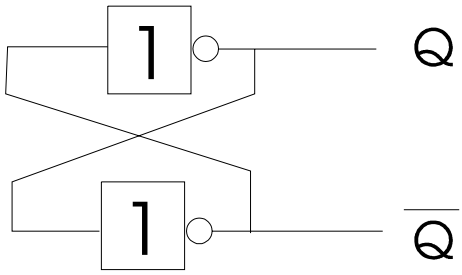
$$f_{DMF} = \underbrace{ab + \bar{a}\bar{b}}_{\substack{\text{wesentliche} \\ \text{Primimplikanden}}} + \underbrace{\begin{cases} \bar{b}\bar{d} + \bar{a}\bar{c}d \\ \bar{b}\bar{d} + b\bar{c}d \\ a\bar{d} + \bar{a}\bar{c}d \\ a\bar{d} + b\bar{c}d \end{cases}}_{\text{Restüberdeckung}}$$

3.5 Schaltwerke

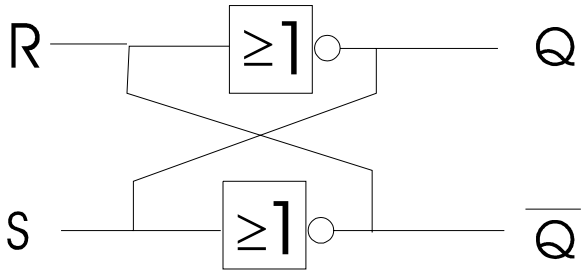
Schaltwerke (Sequentielle Schaltkreise) sind logische Schaltkreise, deren Ausgabe von der Vergangenheit abhängig ist, d. h. sie sind in der Lage Information zu speichern. Insbesondere heißt dies: sie realisieren zeitabhängige Schaltfunktionen.

Ein Schaltkreis, der 1 bit Information speichert, heißt Flipflop.

Grundschaltung eines Flipflops:



RS-Flipflop (Rücksetz-Setz-Flipflop):



Voraussetzung:
feste Zeitpunkte $t_0 < t_1 < t_2 < ... < t_n$ zum Umschalten mit
$t_i = t_{i-1} + \Delta$.

Wenn weiters gilt:
$Q_n = Q(t_n)$ und
$\overline{Q_n} = \overline{Q}(t_n)$

Dann hat das RS-Flipflop folgendes Verhalten:

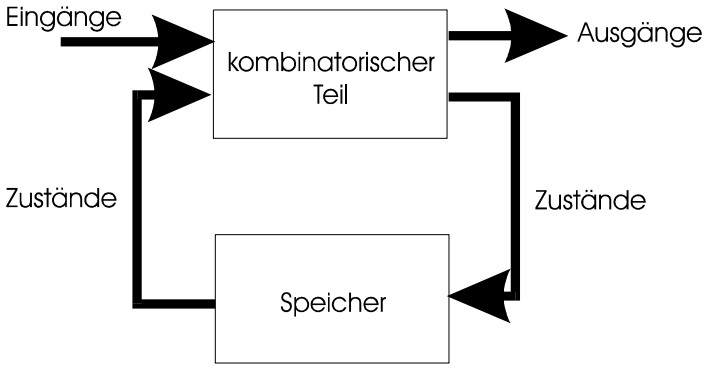
R_n	S_n	Q_{n+1}	$\overline{Q}_{n+1}$
0	0	Q_n	$\overline{Q}_n$
0	1	1	0
1	0	0	1
1	1	unerlaubt	

Im Zustand 1/1 würden an beiden Ausgängen eine Null anlegen ⇒ somit unerlaubt, da nicht mehr komplementär. Würde man anschließend 2 Nullen an den Eingang legen, so entstünde ein instabiler Zustand.

HUFFMAN MODELL EINES SCHALTWERKES:

Schaltwerke, die beim Rechnerentwurf benutzt werden, haben eine Struktur, die durch das Hufmann Modell beschrieben werden kann.

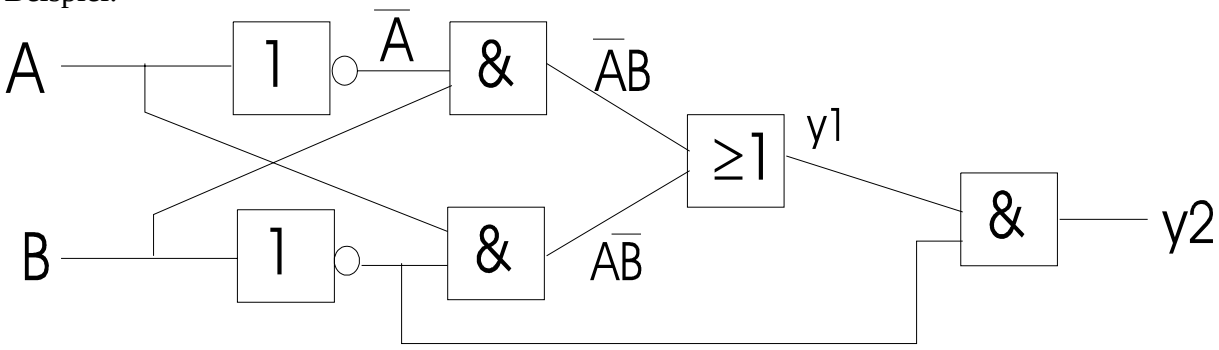
Ein Schaltwerk besteht aus einem kombinatorischen Schaltkreis und einem Speicher, welcher aus Fliflops aufgebaut wird.



Probleme:

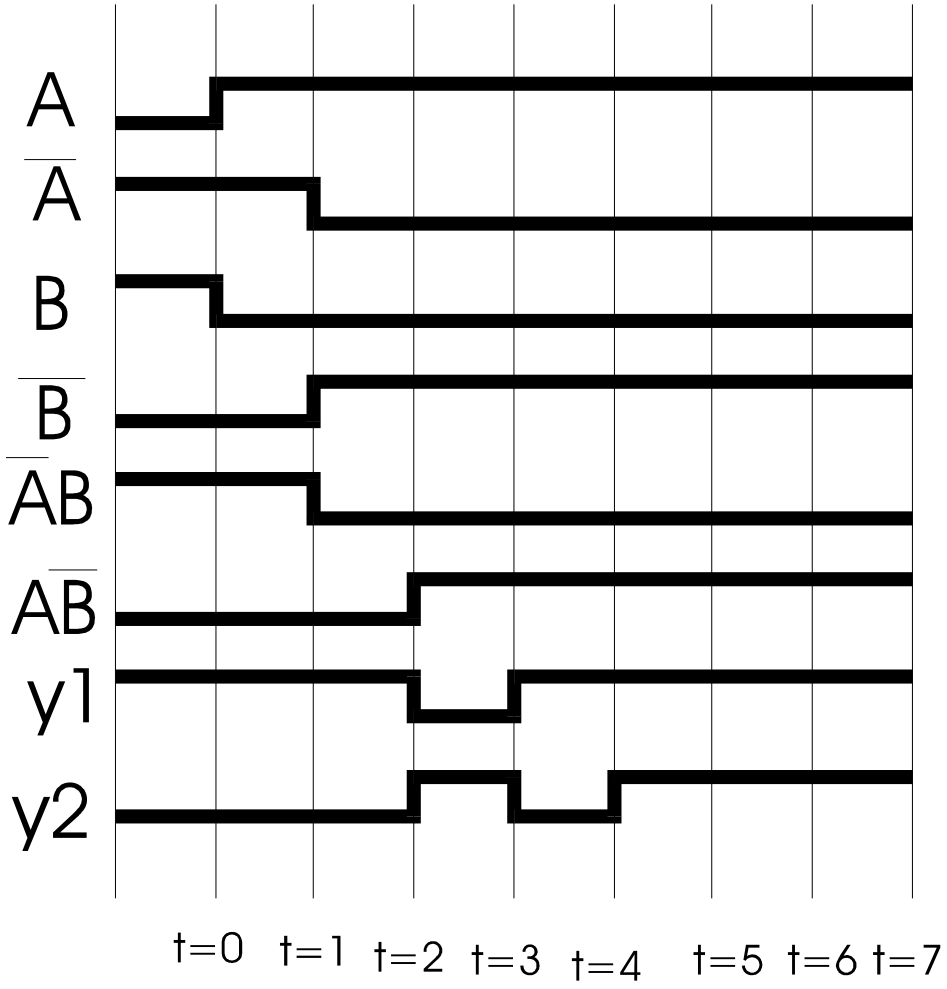
- Individuelle Gatter haben unterschiedliche Delays
- Reaktion der Ausgänge auf die Eingänge kann zeitlich variieren

Beispiel:

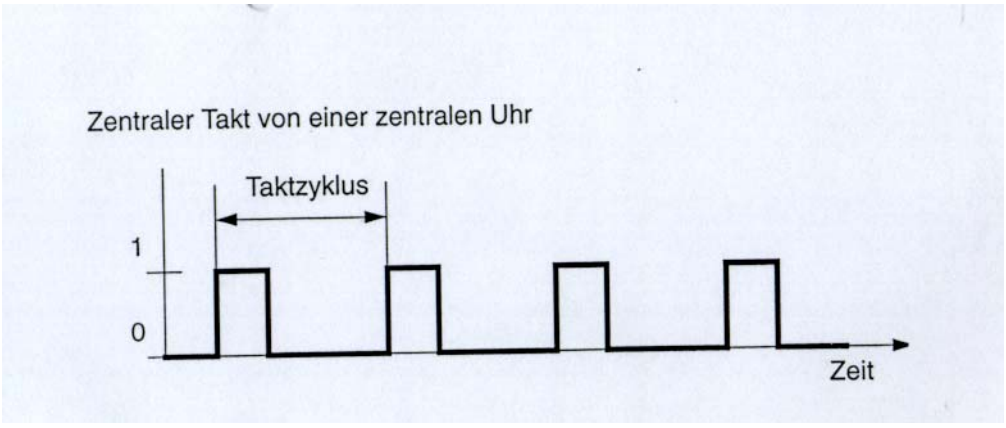


A	B	y_1	y_2
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	0

Annahme: Gatterlaufzeit = 1 Einheit und es wird A = 0 und B = 1 an die Schaltung angelegt



Somit wird ein zentraler Takt von einer zentralen Uhr eingefügt.



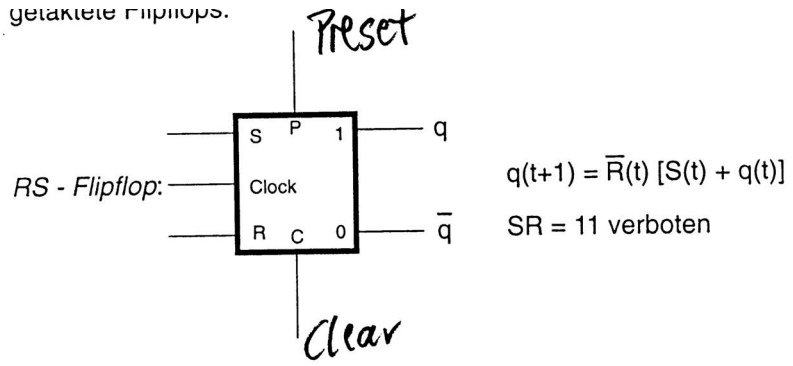
Takt s:
- Speicher übernimmt Zustand
- kombinatorischer Schaltkreis reagiert auf Änderungen
- nach einiger Zeit: Ausgänge stabil

Takt s+1:
- Speicher übernimmt Zustand

Schaltkreise mit zentraler Uhr heißen synchron.

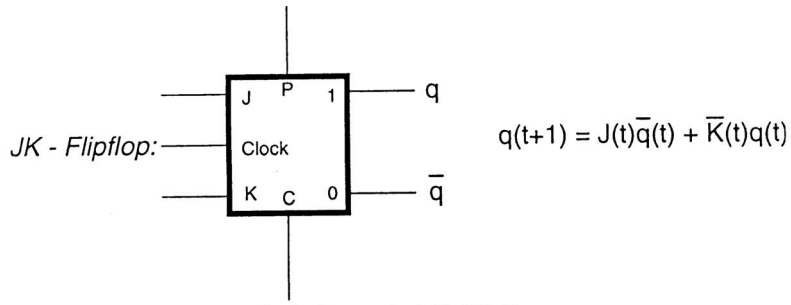
Asynchrone Schaltkreise sind extrem schwierig zu entwerfen.

getaktete Flipflops:



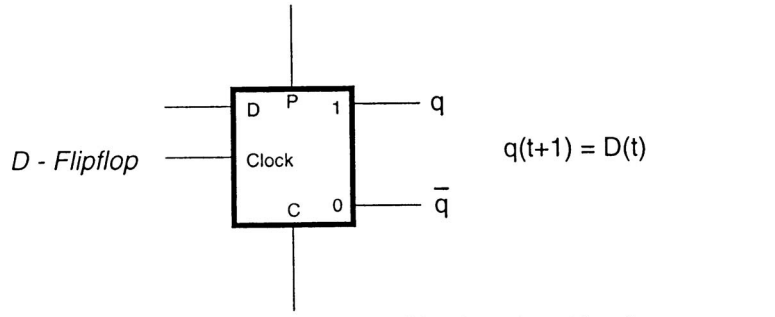
$q(t+1) = \overline{R}(t)\,[S(t) + q(t)]$

SR = 11 verboten



$q(t+1) = J(t)\overline{q}(t) + \overline{K}(t)q(t)$

gleiches Verhalten wie RS-Flipflop
nur 1 1 erlaubt: bewirkt Komplementierung des Zustandes



$q(t+1) = D(t)$

(Delay) d.h. getaktete Version einer Verzögerung

JK-FLIPFLOP:

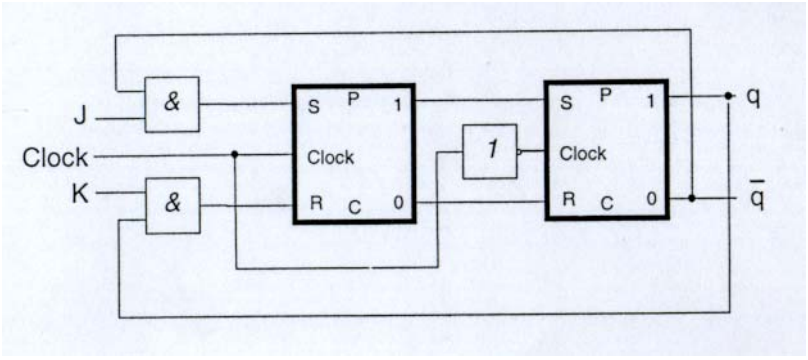
$$Q(t+1) = J(t)\overline{Q}(t) + \overline{K}(t)Q(t)$$

J	**K**	**Q(t+1)**
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

Der Takt muß nun entsprechend lange anliegen, damit alle Flipflops schalten können. Der Taktimpuls darf aber nicht zu lange sein, da sonst möglicherweise bereits neue

Eingangssignale übersehen werden, wodurch ein falsches Verhalten der Schaltung entstehen kann.

⇒



Master-Slave-Flipflop ist zustandsgesteuert:

- Master reagiert, falls Takt auf 1
- Slave reagiert, falls Takt auf 0

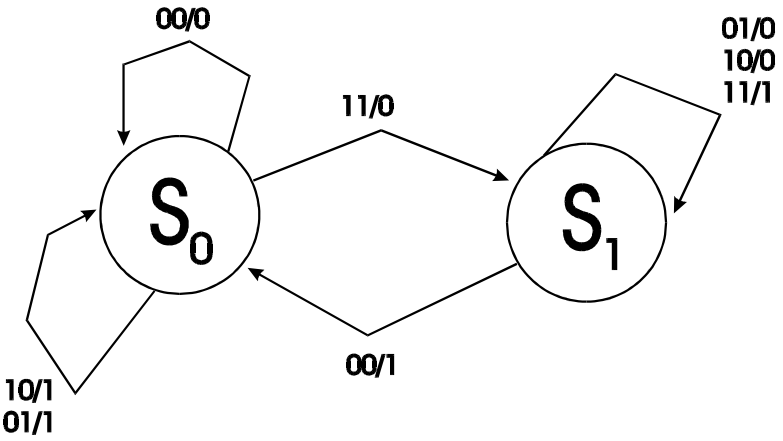
Design Methode für sequentielle Schaltkreise (= **HUFFMAN MODELL**)
(siehe Übungen 7.2)

1. Anzahl der Zustände von Speicher M festlegen
2. Konstruiere Zustandsgraph meist als Tabelle; eventuell Zustandsminimierung (algorithmisch schwierig)
3. Flipflop-Typ wählen. Jedem Knoten wird binärer Code zugeordnet
   für n Zustände benötigt man mindestens $\lceil ld(n) \rceil$ Flipflops.
4. Zu jedem Zustand und jeder Kombination von Eingangssignalen den Folgezustand und die Ausgabesignale ermitteln

Implementierung wie bei kominatorischen Schaltkreisen:

- Konstruktion einer Wahrheitstabelle eines kominatorischen Schaltkreises C mit gewünschten Verhalten (Änderung von Zustand und Ausgabesignalen).
- Bestimmung eines minimalen Boole'schen Ausdrucks fü C.
- Realisierung durch Gatter

Beispiel: serieller Addierer (Bit für Bit wird addiert)



Erklärung:
Im Zustand S_0 hat der Zustandsgraph keinen Übertrag. Kommt 1 und 1 in das Addierwerk, so wird 0 ausgegeben und das Schaltwerk kommt in den Zustand S_1, wo es sich den Übertrag merkt. Kommt als Eingabe 00, so wird der Übertrag (1) ausgegeben und das Addierwerk geht in den Zustand S_0 (kein Übertrag) zurück.

Zustandsgraph als Tabelle:

Zustand	**00**	**01**	**10**	**11**
S_0	S_0,0	S_0,1	S_0,1	S_1,0
S_1	S_0,1	S_1,0	S_1,0	S_1,1

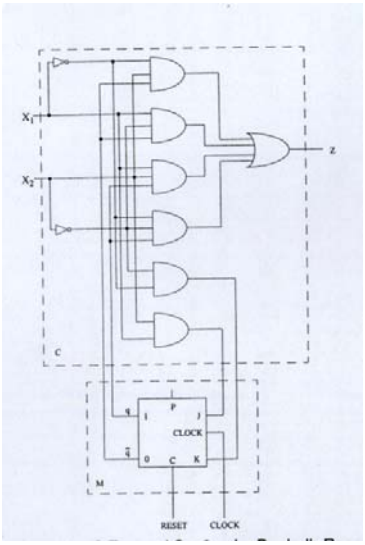
Implementierung durch 1 JK-Flipflop!

Wahrheitstabelle (Transistionstabelle)

	Eingaben			Ausgaben		
S(t+1)	S(t)	$x_1(t)$	$x_2(t)$	J(t)	K(t)	z(t)
0	0	0	0	0	d	0
0	0	0	1	0	d	1
0	0	1	0	0	d	1
1	0	1	1	1	d	0
0	1	0	0	d	1	1
1	1	0	1	d	0	0
1	1	1	0	d	0	0
1	1	1	1	d	0	1

S(t) Übertrag
$x_1(t)$ 1. Zahl
$x_2(t)$ 2. Zahl
d don't care (entweder 0 oder 1; es spielt keine Rolle)

keine Ahnung, wie es weitergeht

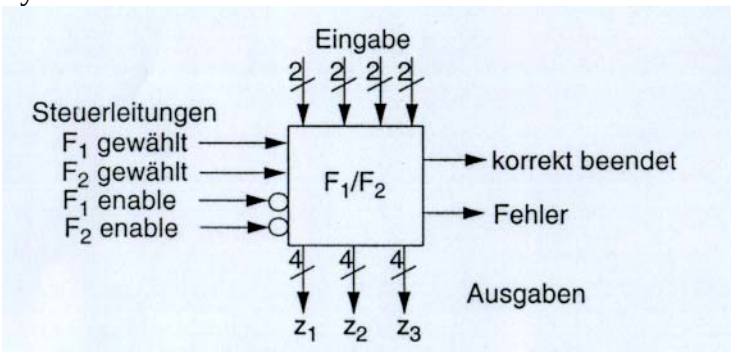


4 Register-Transfer-Ebene

Wortorientierte Komponenten

Schaltkreistype	Komponenten	Funktionen
kombinatorisch	Wortgatter	Boole'sche Operationen
	Multiplexer	Datenstuereung, Generierung allg. Fkt.
	(De-)Kodierer	Kodierung und Dekodierung
	Programmierbare Felder	Generierung allgemeiner Funktionen
	Arithmetische Elemente (Addierer, ALUs)	Numerische Operationen
sequentiell	(Schift)Register	Speicherung von Information Seriell-parallel Konvertierung
	Zähler	Kontrolle/Timing-Signal-Erzeugung

Symbol:



Steuerleitungstypen:

- Auswahl: Funktion auswählen
- Enable: Zeitdauer der Auswahl festlegen

Bus:
mehrere zu einer Einheit zusammengefaßte Verbindungsleitungen zwischen wortorientierten Komponenten (stark vereinfacht)

Beschreibungstechniken:

- Zustandstabellen
- Blockdiagramme
- Beschreibungssprachen

4.1 Register-Transfer-Ebene: Komponenten

4.1.1 Wortgatter

Ein Wortgatter realisiert die Funktion

$Z = f(X,Y)$

wobei:

f:	Boole'sche Operation
X,Y,Z:	Wörter (je m Variable) mit $X = (x_1, x_1, ..., x_{m-1})$
$z = f(X,Y)$:	definiert durch $z_i = f(x_i,y_i)$

oder

$Z = f(X,y)$

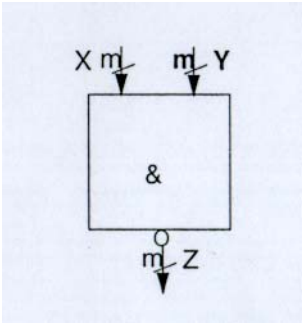
wobei:

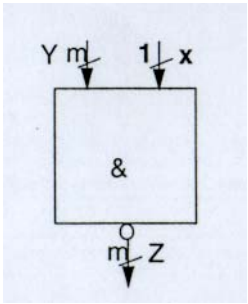
y:	Boole'sche Variable
$Z = f(X,y)$:	$z_i = f(x_i,y)$

Beispiele:

$Z = \overline{XY}$ Bedeutung: $(z_0, z_1, \ldots, z_{m-1}) = (\overline{x_0 y_0}, \overline{x_1 y_1}, \ldots, \overline{x_{m-1} y_{m-1}})$

$Z = \overline{xY}$ Bedeutung: $(z_0, z_1, \ldots, z_{m-1}) = (\overline{x y_0}, \overline{x y_1}, \ldots, \overline{x y_{m-1}})$

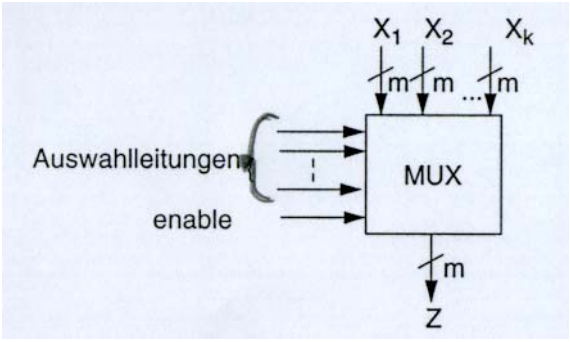




Bemerkung:
Wortgatter selten benutzt, da ziemlich einfache Funktionen sind, die durch einfache Gatter beschreibbar sind.

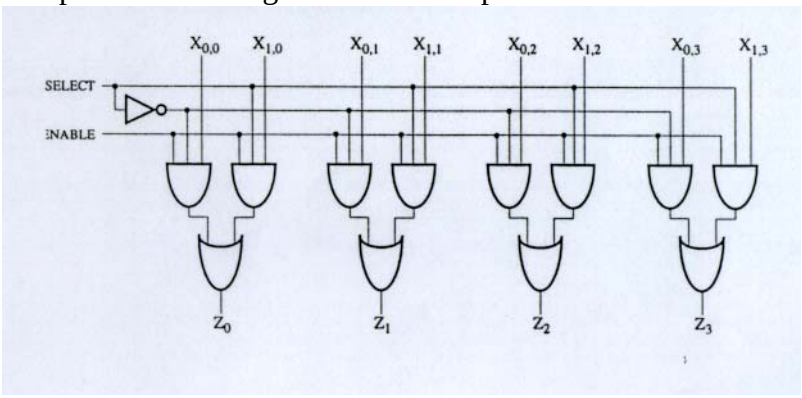
4.1.2 Multiplexer

Ein Multiplexer hat die Aufgabe, einen von k Bussen an einen Ausgabebus durchzuschalten. Falls es sich um k Busse mit je m Bit handelt, spricht man von einem k-Eingabe-m-Bit-Multiplexer (k-Eingabe-MUX)



Üblich: $k = 2^p$ dann kann SELECT mit p Bits codiert werden

Beispiel: 2-Eingabe-4-Bit-Multiplexer



Implementierung eines k^q-Eingabe-Multiplexer durch k-Eingabe-Multiplexer

Bemerkung:
Baum der Tiefe q
In jeder Ebene wird bei jedem MUX der gleiche Eingang ausgewählt.

Beispiel: 2^3-Eingabe-Multiplexer

MULTIPLEXER ALS GENERATOR FÜR SCHALTFUNKTION F:

Sie beliebige DKF gegeben durch

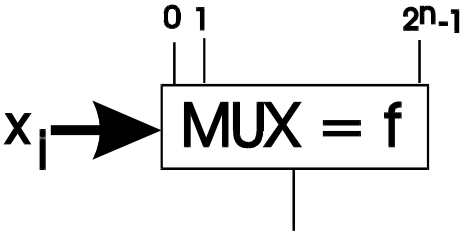
$$DKF(y_1, y_2, \ldots, y_n) = \sum_{i \in D} m_i$$

mit $m_i = y_1^{\delta^1} y_2^{\delta^2} \ldots y_n^{\delta^n}$

$i = (d^1, d^2, \ldots, d^n)_2$ und $i \in D$ m_i ist Minterm von f

	x_1 ... x_n	f
0		
1		
....		
2^n-1		

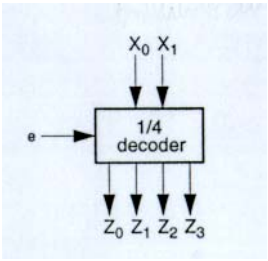
2^n Eingänge ⇒

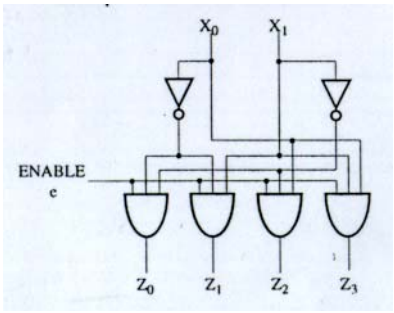


f läßt sich realisieren, indem $x_i = \begin{cases} 1 & falls \; i \in D \\ 0 & sonst \end{cases}$

4.1.3 Dekodierer

Ein Dekodierer wählt aus 2^n Ausgangsleitungen mittels n Eingabeleitung eine aus ($1|2^n$-Dekodierer)
Eingabe $x_0, x_1, ..., x_{n-1}$ wählt r-te Leitung aus für $r = (x_0 x_1 ... x_{n-1})_2$





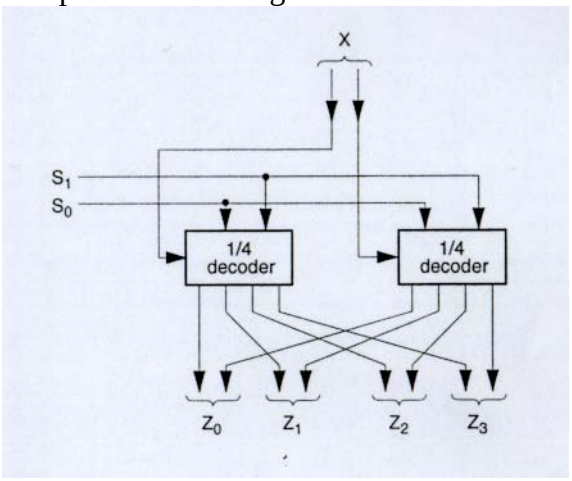
Einsatzbeispiel:
Adressierung von Speichern

4.1.4 Demultiplexer

Ein Demultiplexer schaltet einen Bus auf einen von k Bussen

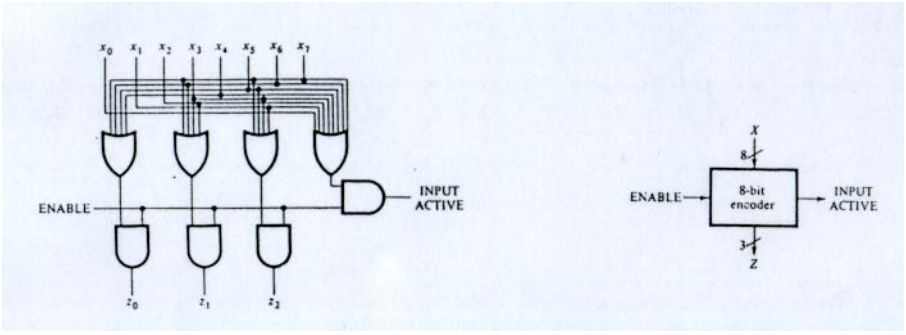
DEMUX ist bei m Bit breiten Bussen durch m $1|2^k$-Dekodierer implementierbar

Beispiel: 4-Ausgabe-2-Bit DEMUX



4.1.5 Kodierer

Ein Kodierer realisiert die Umkehrfunktion zu einem Dekodierer. Es werden dabei durch dem m-ten von 2^k Eingangsleitungen diejenigen Ausgangsleitungen ausgewählt, die $m = (...)_2$ entsprechen.



Bemerkung: Input-aktiv notwendig

Problem: 2 Leitungen aktiv!
z. B. x1 = x2 = 1 $\Rightarrow$ z = (0,1,1) entspricht x3 = 1

Lösungen:

- zusätzliche Kontrolle, daß keine 2 Leitungen aktiv
  z. B. $kon = \overline{\sum_{i \neq j} x_i x_j}$
- Prioritäts-Kodierer: falls mehr als eine Leitung aktiv (z. B. x_i,x_j,x_k) wird die Schaltung so ausgelegt, daß x_i mit i > j,k als aktiv unterstellt wird.

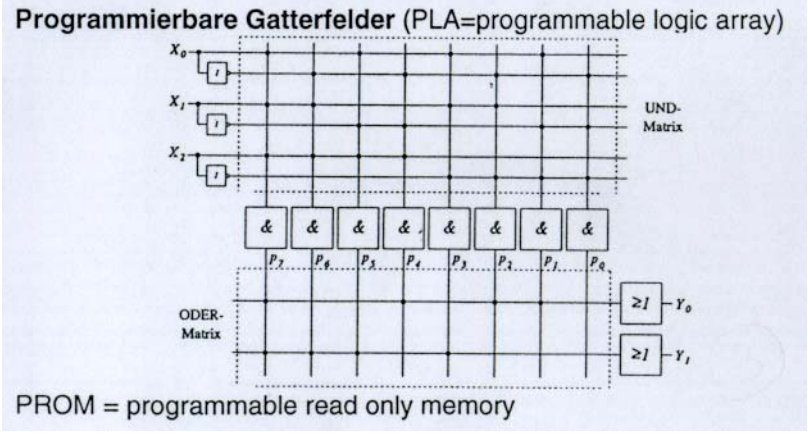
4.1.6 Felderlogik

Felderlogik erlaubt die Realisierung von Schaltnetzen durch Standardbausteine.

Standardbausteine sind in integrierter Technik hergestellte Chips, die
- für Massenproduktion in integrierter Technik geeignete Strukturen realisieren
- die nachträgliche Modifikationen der Struktur zur Anpassung an Kundenwünsche erlauben

Beispiele:
einfache Schaltnetze: Anpassung an Kundenwünsche durch nachträgliche Entfernung von Verbindungen

Programmierbare Gatterfelder (PLA = programmable logic array)



$y_0 := x_0x_1x_2 + \overline{x}_0\overline{x}_1x_2 + \overline{x}_0x_1\overline{x}_2 + x_0\overline{x}_1\overline{x}_2$

PROM = programmable read only memory

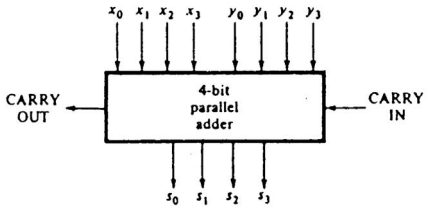
für komplexe Schaltnetze: nicht programmierbare Gatterfelder (gate arrays)
- große Felder einzelner Transistoren oder Gatter
- nachträgliches Anbringen von Leiterbahnen durch den Hersteller gemäß Kundenwunsch

4.1.7 Arithmetische Elemente

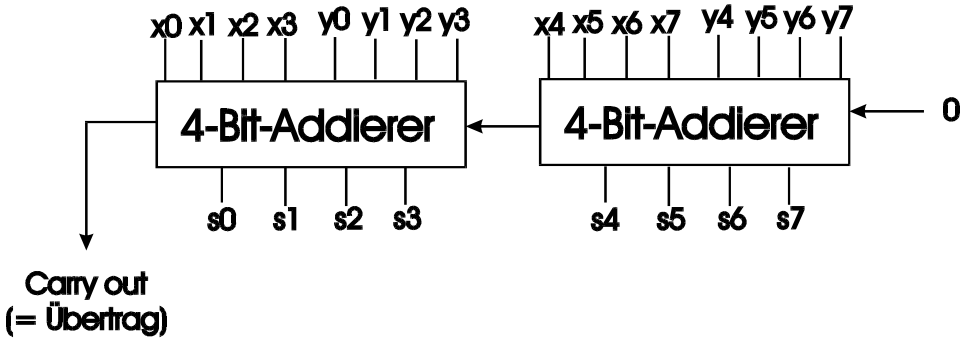
Nur einfache arithmetische Komponenten werden als Register-Level-Komponenten auf Gatterebene realisiert. Floating-Point oder Integer Multiplizierer/Dividierer sind zu komplex.

4.1.7.1 Integer-Addierer/Subtrahierer

4-Bit-Addierer:



8-Bit-Addierer:



Festpunktaddierer:

X	+	Y	→	C, Z
n Bit		n Bit		(n+1) Bit

Serieller Addierer:
in n Zyklen werden 2 n-Bit-Zahlen addiert

Parallele Addierer:
in einem Zyklus werden 2n-Bit-Zahlen addiert, meist aufgebaut aus 1-Bitaddierern (Volladdierern):
$C_{i+1} + x_i + y_i \rightarrow C_i, z_i$

x_i	y_i	C_{i+1}	C_i	z_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$c_i = y_i c_{i+1} + x_i c_{i+1} + x_i y_i$

$z_i = (x_i \oplus y_i) \oplus c_{i+1}$

Ripple Carry Adder:

n Volladdierer hintereinanderschalten
carry wird sequentiell durchgereicht ⇒ maximale Verzögerung: n*Delay des Volladdierers

Beschleunigung des Addierers durch
CarryLookAhead-Technik (carries vorher berechnen)

$c_i = x_i y_i + y_i c_{i+1} + x_i c_{i+1}$
$= x_i y_i + (x_i + y_i) c_{i+1}$

wir setzen
$x_i y_i =: g_i$ und
$x_i + y_i =: p_i$

Beispiel:
n = 4:
$c_3 = g_3 + p_3 c_{in}$
$c_2 = g_2 + p_2 g_3 + p_2 p_3 c_{in}$
$c_1 = g_1 + p_1 g_2 + p_1 p_2 g_3 + p_1 p_2 p_3 c_{in}$
$c_0 = g_0 + p_0 g_1 + p_0 p_1 g_2 + p_0 p_1 p_2 g_3 + p_0 p_1 p_2 p_3 c_{in}$

Vorteil:
- c_i abhängig von X und Y und c_{i+k}
- 3-stufige Gatterschaltung unabhängig von k

üblicherweise wird diese Technik nur im Bereich $4 \le k \le 8$ verwendet!

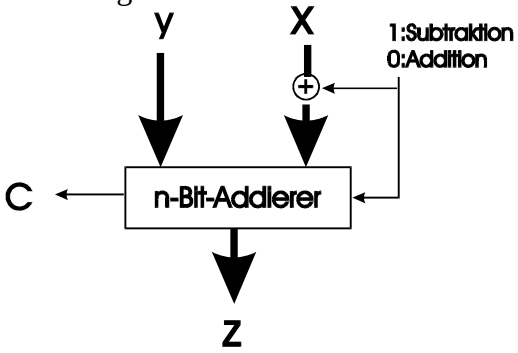
4.1.7.2 Paralleladdierern für Addition und Subtraktion im 2-Komplement

$Z = Y \pm X$

$$Z = Y - X = Y - Ko(X) = Y + \left(\left(\overline{x}_0, \overline{x}_1, \ldots \overline{x}_{n-1}\right) + 1\right)$$
$$= Y + \left(x_0 \oplus 1, x_1 \oplus 1, \ldots, x_{n-1} \oplus 1\right) + 1$$

$$Z = Y + X = Y + \left(x_0 \oplus 0, x_1 \oplus 0, \ldots, x_{n-1} \oplus 0\right) + 0$$

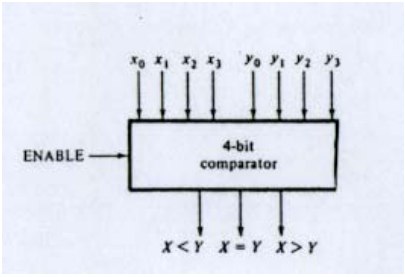
Schaltung:



Bemerkung:
- Schaltung ist kombinatorischer Schaltkreis

- liefert nach Anlegen der Eingangswerte nach gewisser Verzögerung ständig das Ergebnis

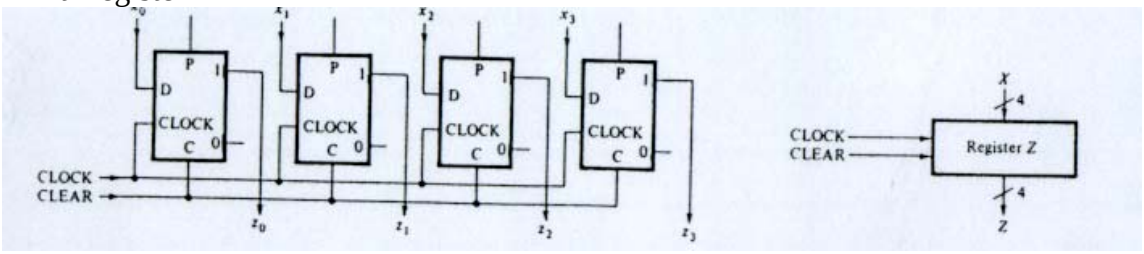
4.1.7.3 Komparatoren



4.1.8 Register

ein m-Bit-Register ist eine geordnete Menge von m Flipflops zur Speicherung eines m-Bit-Wortes (jeder behandelte Typ geeignet).

4-Bit-Register

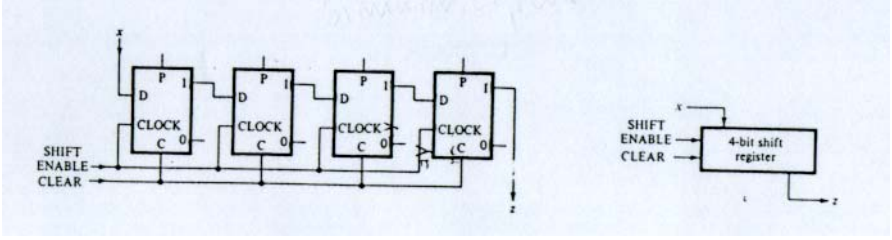


Shiftregister:

- Rechtsshift: $(z_0, z_1, ..., z_{m-2}, z_{m-1}) \rightarrow (0, z_0, z_1, ..., z_{m-2})$
- Linksshift: $(z_0, z_1, ..., z_{m-2}, z_{m-1}) \rightarrow (z_1, z_2, ..., z_{m-2}, z_{m-1})$
- signed-shift: $(z_0, z_1, ..., z_{m-2}, z_{m-1}) \rightarrow (z_0, z_0, z_1 ..., z_{m-2})$

somit bleibt beim signed-shift das Vorzeichen erhalten!

Serieller Rechts-Shifter:

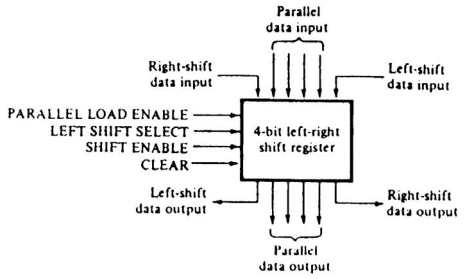


Rotation von Daten:

$(z_0, z_1, ..., z_{m-2}, z_{m-1}) \rightarrow (z_1, z_2, ..., z_{m-1}, z_0)$

$(z_0, z_1, ..., z_{m-2}, z_{m-1}) \rightarrow (z_{m-1}, z_0, z_1, ..., z_{m-3}, z_{m-2})$

Allgemeines 4-Bit-Shift-Register:

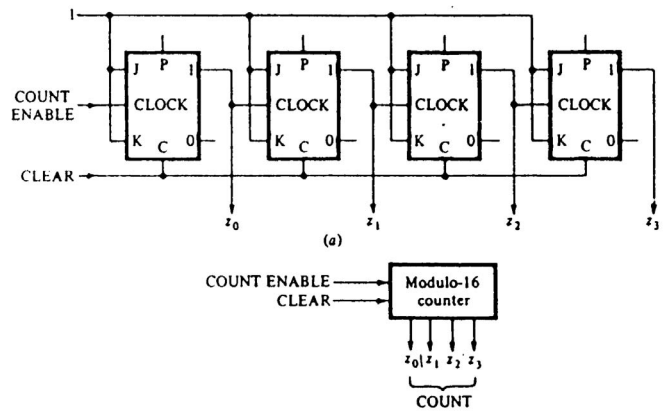


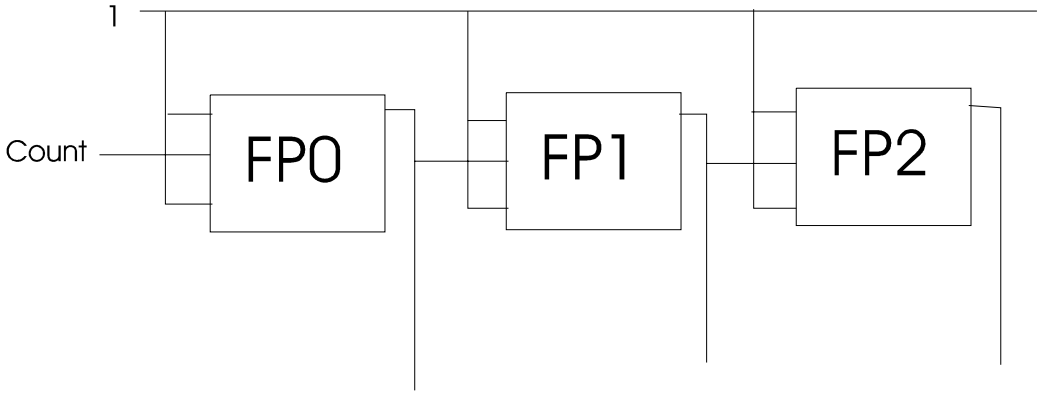
Typische Anwendungen:

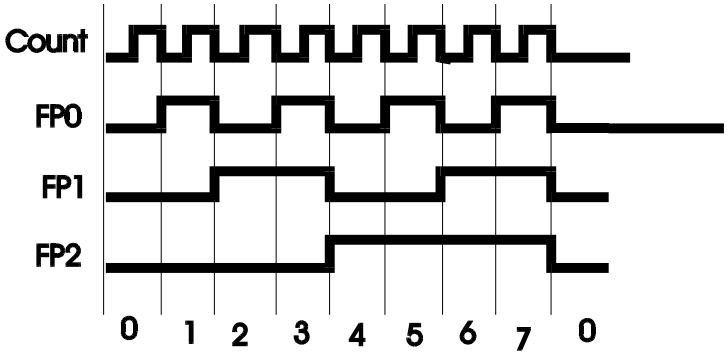
- Speicherung von Daten (parallel und seriell)
- Konvertierungen (Seriell → parallel und umgekehrt)
- Ausführung arithmetischer Operationen
- Links-(Rechts-)-Shift: Multiplikation (Division) mit (durch) 2

Zähler:
in einem Register werden Signale modulo k gezählt

Beispiel:
modulo-16-Zähler aus 4 Master-Slave J-K-Flipflops

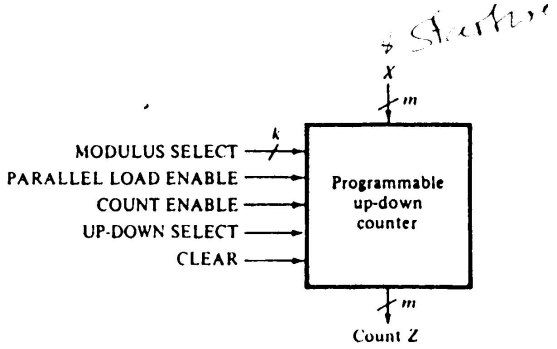






$\Rightarrow$ Schaltung zählt modulo $2^3 = 8$

Allgemeiner Zähler:



Beispieleinsatzgebiet:
Programmzähler

4.1.9 Busse

Bus: Menge von Verbindungsleitungen zum Transfer aller Bits eines w-Bit-Wortes von einer bestimmten Quelle zu einem spezifizierten Ziel

Zugeordneter (deticated) Bus verbindet genau eine Quelle mit einem Ziel (uni- oder biderektional)

Bei n Einheiten benötigt man n(n-1) Busse, um alle möglichen Verbindungen mittels zugeordneten Bussen herstellen zu können.

Problem (sehr teuer):
- Bustreiber
- Kontrolle
- Kabelpreise
- Pinnzahl

Gemeinsamer (shared) Bus:
Register A → Datenbus
Datenbus → Register B

4.2 Design-Methode auf Register-(Transfer)-Level

Ziel: Entwurf eines Befehlsmengen-Prozessors

Befehlsmengenprozessor (General purpose processor z. B. Pentium):
Maschine auf Register-(Transfer)-Level (komplexes Schaltwerk)
Struktur: aufgebaut mit Komponenten dieser Ebene
Verhalten: ist gegeben durch endliche Menge von Befehlen $S = \{F_i\}$

Befehl:
Operation bzw. Funktion F_i, die auf unterschiedlichen Typen von Wörtern X_i operiert.
Jeder Befehl F_i wird implementiert durch eine Folge von elemenaren Register-Transfer-Operationen sogenannten Mikrooperationen f:

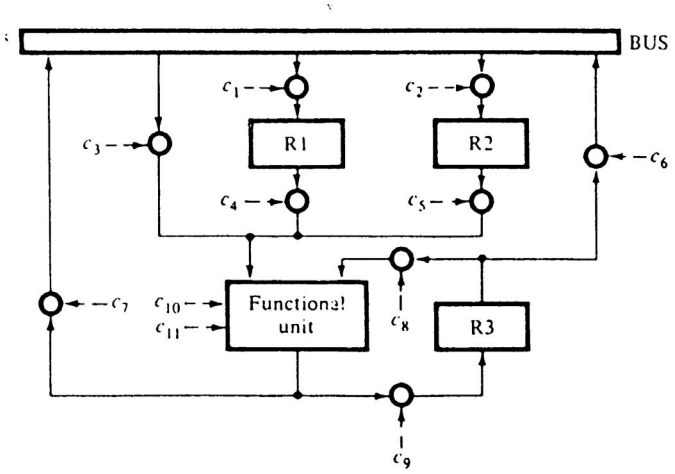
$$Z = f(X_1, X_2, ..., X_k) = f_r^i(...f_2^i(f_1^i(X_1, X_2, ...X_k)...)$$

Die einzelnen Mikrooperationen werden durch Kontrollsignale ausgelöst.

Wegen der Komplexität unterscheidet man zwischen
- Datenverarbeitungsteil
- Kontrollteil

Einfachere Fälle: Kontrollteil als sequentieller Schaltkreis (Gate-Level)
Komplexe Fälle: Kontrollteil selbst eine Register-Transfer-Maschine, die einfacher als das Original ist

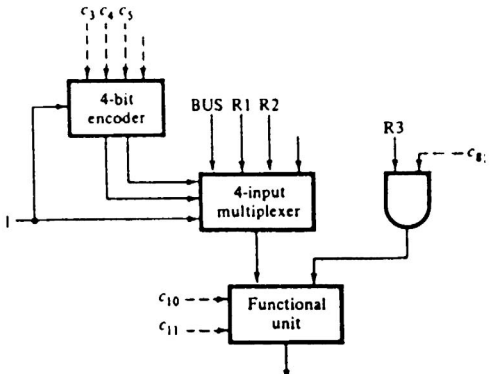
Register-Level-Schaltkreis (insbesondere Datenverarbeitungteils)
ist ein Netzwerk bestehend aus:
- Register-Level-Komponentne
- gerichteten Verbindungen zwischen den Komponenten und den Datenpfaden
- jeder Datenpfad enthält einen Kontrollpunkt, der je nach Steuerung den entsprechenden Datenpfad
    - blockiert oder
    - nicht blockiert



Kontrollpunkte werden durch logische Schaltkreise implementiert und durch Kontrollsignale gesteuert, die von der Kontrolleinheit gesteuert werden.

Beispiel:
Funktionseinheit

it



Vorgehensweise beim Design:

1. Definiere Menge der Funktion S und die hierfür notwendigen Folgen von Register-Transfer-Operationen unter Berücksichtigung der Möglichkeiten verfügbarer Komponenten.
2. Analysiere S um die Komponenten und deren Anzahl für den Datenverarbeitungsteil festzustellen.
3. Konstuiere ein Blockdiagramm D für Datenverarbeitungsteil. Bestimme die notwendigen Verbindungen zwischen den Komponenten so, daß die Leistungs- und Kostennebenbedingungen erfüllt sind.
4. Bestimme zu S und D die Kontrollpunkte und die Kontrollsignale, die gebraucht werden.
5. Entwerfe die Kontrolleinheit für die Maschine, sodaß die notwendigen Kontrollsignale in der gewünschten Reihenfolge, wie sie von S spezifiziert werden, erzeugt werden.
6. Vereinfachung wo möglich.

Beispiel:
Entwurf eines Festpunkt-Multiplizierers
gegeben:
$X = x_0x_1x_2...x_7$
$Y = y_0y_1y_2...y_7$
mit x_0 und y_0 als Vorzeichen und
$X_M = 0,x_1x_2...x_7$ (Betrag = Magnitude von X)

gesucht:
$P = X * Y$
$P = p_0p_1p_2...p_{14}$,
wobei p_0 das Vorzeichen ist.

Berechnung:
$p_0 = x_0 \oplus y_0$
$P_M = X_M * Y_M$

P_M-Berechnung:
$P0 = 0$
für i = 0, 1, 2,...6 berechne: $P_i := P_i + x_{7-i}Y_M$, $P_{i+1} := 2^{-1}P_i$

1	0	1	1	*	1	0	1	0
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	
0	0	0	1	0	1	1		
0	0	0	1	0	1	1		
0	0	0	0	0	0			
0	0	0	1	0	1			
0	1	0	1	1				
0	1	1	0	1				

Wie man sieht, bleibt bei der Multiplikation immer die letzte Stelle unverändert. Es reicht somit aus, einen 7-Bit-Addierer zu verwenden, der nach jedem Addieren um eine Stelle nach links geschoben wird:
(Die Vorzeichen sind grau hinterlegt!)

A					Q					M				
0	0	0	0	0	1	1	0	1	0	1	1	0	1	1

Da die letzte Stelle von Q eine Null ist, wird der Multiplikand 0-mal zum Accumulator dazuaddiert:

A					Q					M				
0	0	0	0	0	1	1	0	1	0	1	1	0	1	1

Danach wird eine Rechts-Shift-Operation durchgeführt. Die Vorzeichen wandern somit um eine Stelle nach rechts:

A					Q					M				
0	0	0	0	0	0	1	1	0	1	1	1	0	1	1

Nun zeigt die letzte Stelle von Q eine Eins, wodurch der Multiplikand 1-mal zum Accumulator dazuaddiert wird.

A					Q					M				
0	1	0	1	1	0	1	1	0	1	1	1	0	1	1

Danach wird eine Rechts-Shift-Operation durchgeführt. Die Vorzeichen wandern somit um eine Stelle nach rechts:

A					Q					M				
0	0	1	0	1	1	0	1	1	0	1	1	0	1	1

A					Q					M				
0	0	0	1	0	1	1	0	1	1	1	1	0	1	1

A					Q					M				
0	1	1	0	1	1	1	0	1	1	1	1	0	1	1

A					Q					M				
0	0	1	1	0	1	1	1	0	1	1	1	0	1	1

Nun steht das Vorzeichen von Q direkt vor M. D. h.: Die Multiplikation ist fertig abgearbeitet. Die beiden Vorzeichen werden nun exclusiv geodert (XOR) und an die 1. Stelle von Accumulator gestellt. Das alte Vorzeichen von Q wird vernichtet (zu Null gemacht!):

A					Q					M				
0	0	1	1	0	1	1	1	0	0	1	1	0	1	1

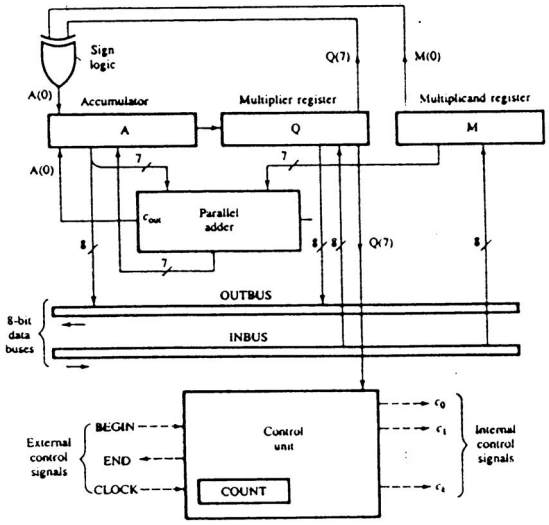
Das Ergebnis steht somit in Accumulator und Q.

Anstatt Hardware:

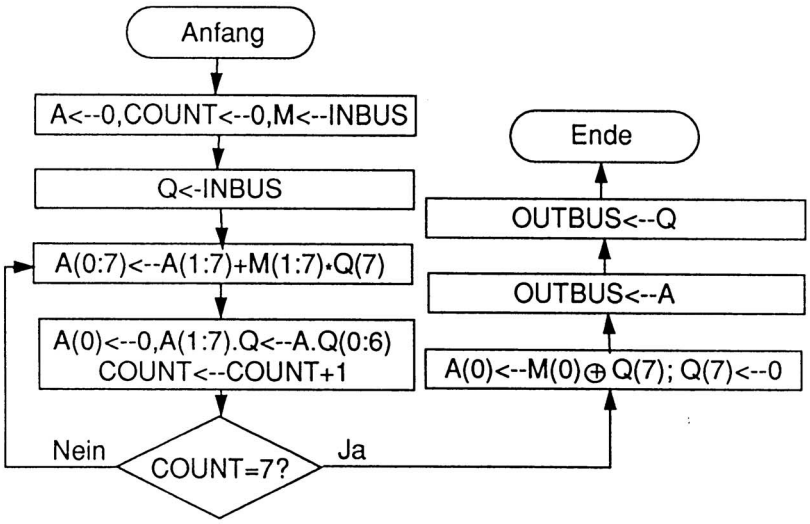
- zwei 8-Bit-Register (Q, M für X und Y; die ursprünglichen Zahlen)
- ein 16-Bit-Register (A für P; Ergebnis)
- eine 7-Bit-Paralleladdierer
- Shiftoperation ⇒ A sein ein paralleles Eingabe-Shift-Register

genügt nun ein 8-Bit-Register und ein 16-Bit-Register (in dem A und Q gemeinsam stehen)

Schaltung:



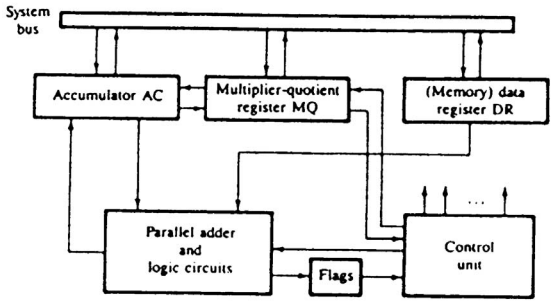
Leitwerksoperationen:



Liste der Steuersignale:

c_0 lösche Akku ($0 \rightarrow$ Akku)
c_1 lösche COUNT
c_2 lade A(0)
c_3 lade Multiplikandenregister M vom INBUS
c_4 lade Multiplikationsregister Q aus INBUS
c_5 lade Hauptausgänge des Addierers nach A(1:7)
c_6 wähle M oder O in den rechten Eingang des Addierers
c_7 Rechtsshift von A.Q
c_8 erhöhe COUNT um 1
c_9 wähle c_{OUT} oder $M(0) \oplus Q(7)$ und lade nach A(0)
c_{10} lösche Q(7)
c_{11} transferiere Inhalt von A nach OUTBUS
c_{12} transferiere Inhalt von Q nach OUTBUS

allgemeine ALU:



Befehle:

Addition	$AC + DR \rightarrow AC$
Subtraction	$AC - DR \rightarrow AC$
Multiplication	$DR \times MQ \rightarrow AC.MQ$
Division	$MQ / DR \rightarrow AC.MQ$
AND	$AC \wedge DR \rightarrow AC$
OR	$AC \vee DR \rightarrow AC$
EXCLUSIVE-OR	$AC \oplus DR \rightarrow AC$
NOT	$\overline{AC} \rightarrow AC$

5 Prozessor Level

(General Purpose Processor)

4 Gruppen von Komponenten
- Befehlsmengenprozessoren
- Speicher
- E/A-Geräte
- Verbindungsnetzwerk

Informationseinheiten
- Programme
- Datenmangen

Transformationseinheiten
- Worte
- Wortmengen

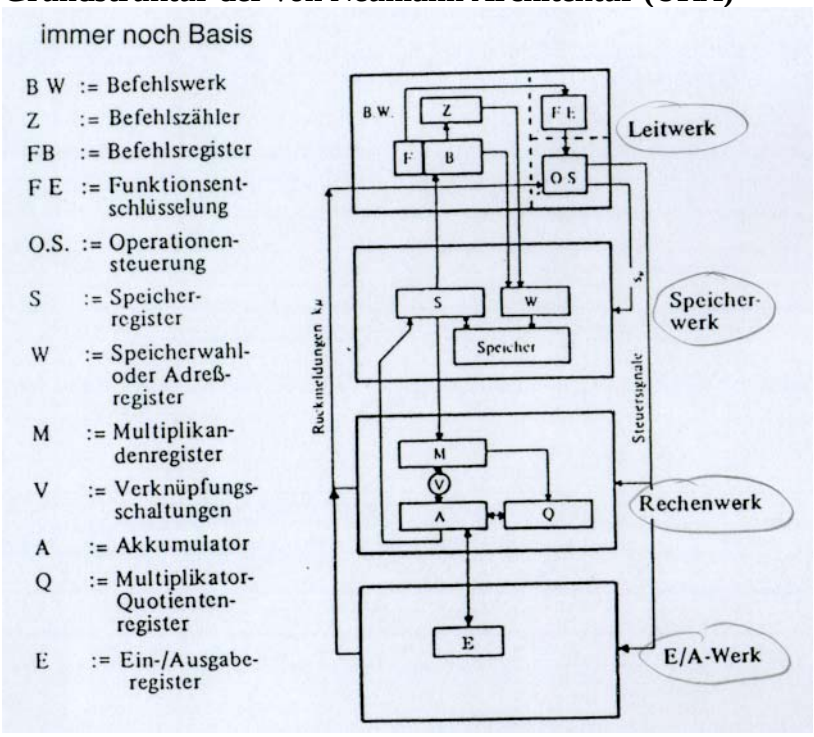
Typische Fragestellungen
- Verarbeitungszeit für ein Programm
- Wieviel Speicher
- Ausnutzung der Komponenten

Control arbeitet nicht mehr einzelne Befehle ab, sondern holt sich die Befehle aus dem Speicher, wo das Programm steht.

5.1 Konzept des URA: (Darstellung nach W. Händler)

1. Der Rechenautomat wird logisch und räumlich in Teile zerlegt:
   1. 1. Leitwerk
   2. Ein-/Ausgaben 2. Speicherwerk
   3. 3. Rechenwerk
2. Zu jedem Problem wird eine Bearbeitungsvorschrift (Algorithmus), das Programm, von außen eingegeben und im Speicher abgelegt. Erst dieses Programm macht den Automaten arbeitsfähig. Die Struktur des Automaten ist problemunabhängig.
3. Programm und Daten selbst im Speicher.
4. Speicher ist Folge von Speicherzellen, die durchnumerkiert sind. Die Nummer eines Peicherplatzes heißt seine Adresse. Über die Adresse kann der Inhalt gelesen oder geändert werden.
5. Programm besteht aus Folge von Befehlen (elementare Operationen), die in konsekutiven Speicherzellen abgelegt sind und i. a. in dieser Reihenfolge abgearbeitet werden; man erhält die Adresse des nächsten Befehls durch Erhöhung der Adresse um 1.
6. Es gibt Sprungbefehle, die bewirken, daß nach der Bearbeitung des Befehls mit Adresse s nicht der Behfehl mit der Adresse s+1 ausgeführt wird, sondern einer mit der Adresse t.
7. Es gibt bedingte Sprübnge, d. h. Sprung wird nur ausgeführt, falls eine bestimmte Bedinung erfüllt ist. Sonst wird der Befhel mit der nächsthöheren Adresse verwendet ⇒ Entscheidungen können in Abhängigkeit von zunächst unbekannter Größen getroffen werden.
8. Das duale Zahlensystem wird benutzt.

Grundstruktur der von Neumann Architektur (URA)



besteht aus 4 Teilen:

- Leitwerk: Programmablaufsteuerung
- E/A-Werk: Ein- und Ausgabe von Daten und Programmen
- Speicherwerk: Programme und Daten speichern
- Rechenwerk: Durchführung von Rechenoperationen und logischen Verknüpfungen

Prozessor = CPU (Central Process Unit)
= Rechenwerk + Leitwerk

Rechenwerk:
ist ein Schaltwerk, Zustand Z ist durch die in den Rechenwerksregistern enthaltenen Informationen definiert. Eingangsgrößen X sind die Operanden und die vom Leitwerk kommenden Steuersignale s_v, Ausgangsgröße Y die Zwischenergebnisse und die Entscheidungskriteriene k_m, die zum Leitwerk zurückgehen.

Das Rechenwerk eines Rechenautomaten muß außer arithmetischen auch logische Operationen durchführen können. Im englischen Sprachgebrauch hat man deshalt den begriff "arithmetic unit" zugusten des Begriffs "arithmetical and locial unit: ALU" aufgegeben. Sinngemäß müßte man im Deutschen Begriffe wird Verknüpfungswerk oder Verarbeitungswerk wählen.

Leitwerk:

- steuert den Programmablauf
- koordiniert die Zusammenarbeit der einzelen Werke

Den Programmablauf regelt das Befehlswerk, den Befehlsablauf die Operationensteuerung. Zwischen beiden liegt die Funktionsentschlüsselung.

Der Befehlstyp:
Bei den hier angestellten Betrachtungen wird stet angenommen, daß Einadreßbefehle vorliegen. Der Grundgedanke des Einadreßbefehls ist das Akkumulieren der Zwischenergebnisse in einem besonderem Register, dem Akkumulator, wobei zusätzlich angenommen wird, daß sich der erste Operand bereits im Akkumulator befinde, währen der zweiter gemöß der angegebene Adresse aus dem Speicher geholt wird. Damit ist es möglich, mit nur einer Adresse auszukommen.

Befehlswerk:
Es enthält die für die Programmsteuerung notwendigen Daten, nämlich in FB den geraden anstehen Befehl, wobei gilt:
(F) := Operationsteil des Befehls (Funktionsteil)
(B) := Adreßteil (z. B. Adresse des Operanden)

Für alle Befehle gemeinsam und gleich ist die
Befehlsholphase:
ein Befehlt wird aus dem Speicher geholt, der Befhelszählerinhalt wird um 1 erhöht.

In der Entschlüsselungsphase tritt eine breitgefächerte Verzweigung ein, jeder Befehl t_i wird auf einem gesonderten Weg weiterverfolgt.

In der Ausführungsphase unterscheidet man vier Befehlstypen:
- Verknüpfungsbefehle: Es wird ein Operand nach M geholt und mit dem Inhalt von A gemäß $_i$ verknüpft.
- Einfache Speicherbefehle: Der Inhalt des Akkumulators wird an einen vorgegebenen Speicherplatz gebracht.
- Sprungbefehle: s wird entweder vor der nächsten Holphase gehalten oder zu einer anderen Adresse gesprungen.
- Bedingte Sprungbefehle: Hier erfolgt der Sprung nur, wenn gewisse Bedingungen β_i erfüllt sind.

Beim Start des Programmes wird zunächst die Adresse des ersten Befehls in den Befehlszähler Z gebracht, dann wiurd ein Startsignal gegeben. Der Programmablauf beginnt mit der Befehlsholphase des ersten Befehls. Im einzelen sind in der Befehlsholphase und den Ausführungsphasen folgende Transporte und Verknüpfungen durchzuführen:

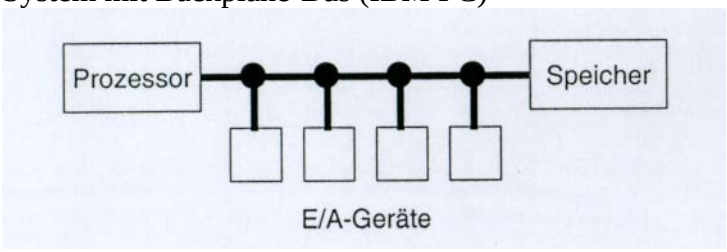
hier fehlt etwas!

5.2 Design-Technik

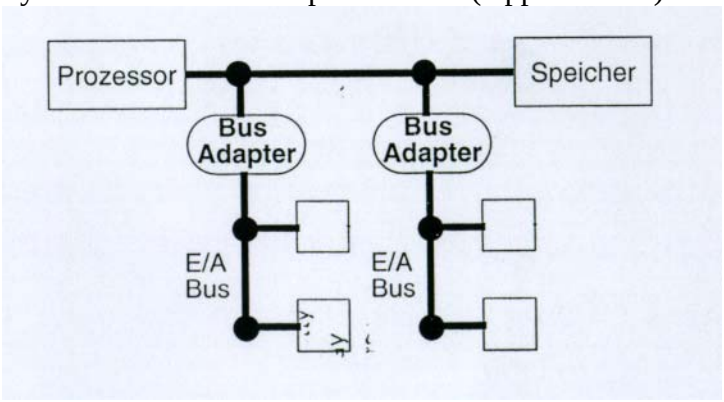
Prototypen unter gewissen Leistungsbedingungen
Bestimmung der Leistung des Prototypen: ungenügend $\Rightarrow$ Redesign

PROTOTYPEN-BEISPIELE:

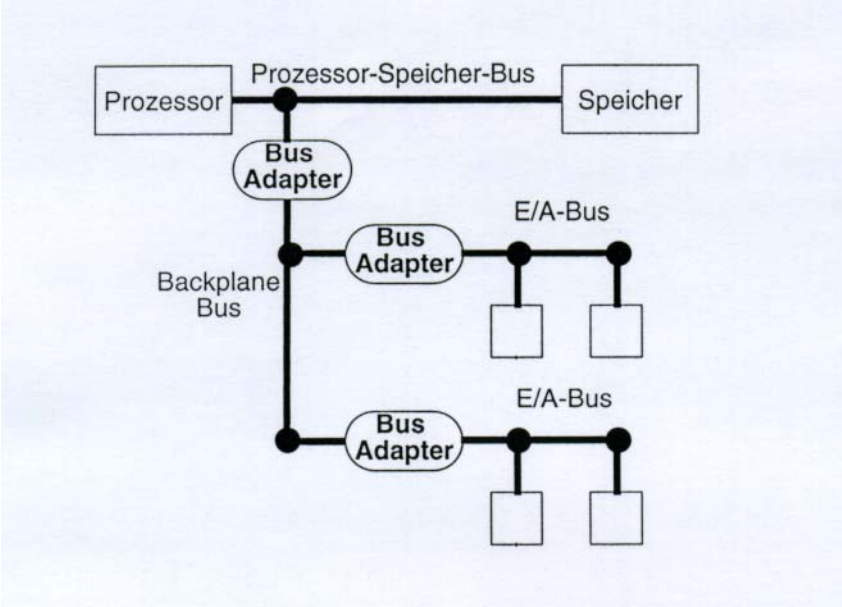
System mit Backplane-Bus (IBM PC)



System mit Prozessor-Speicher-Bus (Apple Mac II)



System mit Backplane-Bus und Prozessor-Speicher-Bus (RS 6000, Silicon Graphics)



5.3 Bewertung von Rechenanlagen

Maßstäbe

- Geschwindigkeit
- Durchsatz
- Wartezeit
- Auslastung
- rel. Leistung
- Zuverlässigkeit
- Benutzerfreundlichkeit

Ziel:

- Rechnerauswahl
- Rechnerentwurf
- Tuning von Rechenanlagen

5.3.1 Leistung und Speedup

relative Leistung:
Rechner A (t_A) ist umd n% schneller als Rechner B (t_B)

t_A ... Zeit, die A für ein Programm braucht

$$Speedup = \frac{t_B}{t_A}$$

5.3.2 Hardwaremaße und Parameter

CPU-Performance:
CPI = Clocks Per Instruction

mittlere Anzahl von Takten/Instruktion

Operationsgeschwindigkeiten:
Als Maße dienen:
- Befehlsausführungszeiten (ADD, MULT, ...)
- Taktfrequenzen
- MIPS: Millions of Instructions Per Second
- MOPS: Millions of Operations Per Second
- MFLOPS: Millions of FLoatingpoint Operations Per Second
- MLIPS: Millions of Logical Interferences Per Second

Nachteil ist die Abhängigkeit von:
- Befehlsvorrat
- untersuchtem Programm
- Wortlänge (32 bit oder 64 bit)
- globaler Systemstruktur (Pipelining, Parallelität)

Vorteil:
- geringer Aufwand

MIXE (veraltet):
Suche nach "mittleren" Operationszeit
t_i: Ausführungszeit für Maschinenbefehl i (i z. B. Addition, Multiplikation)
p_i: Realtive Häufigkeit des Maschinenbefehls i

von Neumann Mix: $T = \frac{2t_A + t_M}{3}$

Weitere Mixe: GAMM, GIBSON, UNI "Weltformel" von Knight

Kernprogramme:
Die Befehle eines Programmes werden gezählt. Die Ausführungszeiten der Befehle werden addiert. Die Programme werden nicht ausgeführt.

Nachteil:
- SW wird nicht berücksichtigt
- bei Parallelitätseigenschaften nicht durchführbar
- sehr viel Aufwand
- kaum ein Vorteil gegenüber MIXEN (werden daher kaum verwendet)

5.3.3 Laufzeitmessung bestehender Programme

- Vorteil: Eigenschaften von Software und die des Betriebssystems können gemessen werden
- Nachteil: Laufzeitmessungen liefern nur Vergleichszahlen, Subjektive Messung, viele Parameter

Benchmarks
Synthetische Benchmarks:
Nur zu Meßzwekcen erstellte Programme (Pogrammpakete). Die Ausführungszeiten werden gemessen. Meistens wird die Laufzeit mit der der VAX-11/780 verglichen.

Numerische Vertreter:
- Whetstone
- Drystone

nicht numerische Vertreter:
- Sieve
- Puzzle
- Ackermann
- Hanoi
- 8 Damen

Kernels:
Das sind kurze, numerische Programme, die aber nicht auf das jeweilige Anwenderprofil eingehen.
Vertreter:
- Livermore Loops
- Linpack

Geschichte zu Linpack: IBM hatte eine optimal compilierte Version des Linpacks und hat diese in einer ihrer Maschinen implementiert. Als nun Linpack getestet werden sollte, erkannte die Maschine das Testprogramm und führte mittels einer Schaltung "switch to Linpack" die vorher einprogrammierte Testversion aus, und war somit am schnellsten. Man vermutet, daß andere PC-Hersteller in ähnlicher Weiser vorgingen, jedoch war IBM der Hersteller, dem man auf die Schliche kam.

Reale Programme liefern allerdings den besten Vergleich. Am besten ist eine Arbeitslast (Workload) der eigenen Maschine.

5.3.4 Messungen des Betriebs bestehender Anlagen

Dafür werden sogenannte Monitore verwendet. Ein Monitor ist ein Aufzeichnungselement von "Ereignissen" im Rechner. Es gibt HW-, FW- und SW-Monitore.

Varianten des Monitoring:
- Kontinuierliche/Sporadische Messung
- Gesamtdaten (tracing)/Übersichtsdaten (pre processing)
- Echtzeitverabeitung/spätere Verarbeitung (post processing)
- Benutzerinformation/Systeminformation
- HW-Messung/SW-Messung

Folgende Probleme treten auf:
- eine Gesamtdatenaufzeichnung ergibt eine enorme Datenmenge
- sporadische Messungen sind dagegen gegebenenfalls falsch
- SW & FM Monitore messen sich selbts mit (sie fügen zusätzlichen Code ein, wodurch es vorkommen kann, daß ein Programm nicht mehr läuft, wenn man den Monitor entfernt)
- HW Monitore setzen einen Physikalischen Zugang voraus

5.3.5 Modelltheoretische Verfahren

Erstellung von Ablaufmodellen mit Annahmen über Struktur und Betrieb des Rechners und über Prozesse

Simulation:
Programmierte Nachbildung des zu untersuchenden Systems, um verschiednene Systemparameter zu ermitteln
- event driven
- trace driven

Analytische Modelle:
Untersuchung des dynamischen Ablaufverhaltens im System und die Aufdeckung von Engpässen. Beruht meistens auf stochastischen Modelln (Warteschlangensystem).
Meistens müssen stark vereinfachende Annahmen getroffen werden, damit eine geschlossene Lösung möglich ist.
Daher werden oft Simulation und analytische Modelle gemeinsam verwendet.

5.3.6 Zuverlässigkeit eines Systems

Der Ausfall eines Rechners kann katastrophale Folgen haben:
- Echzeitbetrieb: Produktionsausfälle in Walzwerk
- Netzwerkbetrieb: Leistungsverlust
- Teilnehmerbetrieb: lange Wartezeiten

Vorsorge:
- stand by: Verdoppelung und bei Ausfall umschalten
- fail soft: Ausfall $\Rightarrow$ reduzierte Leistung (bei Multiprozessoren)
- vergleichen: parallel arbeitende Elemente erzeugen gleiches Ergebnis (Valuator; "goldene Element")

5.4 Rechnerarchitektur

5.4.1 Definition der Rechnerarchitektur

Die Rechnerarchitektur ist die Schnittstelle zwischen Hardware und Software
Die Rechnerarchitektur beschreibt die bestehenden und zukünftigen Rechnerarchitekturen.
Die Rechnerarchitektur betrachtet den Aufbau und die Eigenschaften von Rechenanlagen, ihren Komponenten und deren Zusammenwirken.

Rechnerarchitektur (Definition):

- allgemeine Strukturlehre mit ihren Hilfsmitteln
- ingenieurwissenschaftliche Disziplin, die bestehende und zukünftige Rechenanlagen beschreibt, vergleicht, beurteilt, verbessert und entwirft
- betrachtet dabei Aufbau und Eigenschaften der ganzen Rechenanlage, deren Teile (Komponenten) und Verbindungen

Schichten einer Computerarchitektur

- **Befehlsarchitektur:** Die Beschreibung des Verhaltens eines Rechners auf einer abstrakten Ebene. Dies umfaßt alles, was nach außen sichtbar zur Software ist:
    - Maschinentypen,
    - Befehlssatz,
    - Unterbrechungssystem,
    - usw.
- **Implementierung:** Die Darstellung der internen Struktur und Arbeitsweise auf einer abstrakten Beschreibungsebene. Dies beinhaltet
    - **Rechnerstruktur:** statische Topologie der Hardware-Betriebsmittel und ihre Verbindungen
    - **Informationsstruktur:** die interne Repräsentation von Code und Daten auf den hardware-Betriebsmitteln, z. B.
        - die logische Struktur der Signale,
        - die Speicherstruktur,
        - usw.
    - **Operationsprinzip:** die Darstellung, wie die Informationsstruktur unter Berücksichtigung der Betriebsmittel manipuliert wird (z. B. wie wird ein Befehl abgearbeitet?)
- **Realisierung:** wie wird das Ganze in Hartdware gegossen?

5.4.2 Gestaltungsgrundsätze

5.4.2.1 Konsistenz:

Eigenschaft eines Systems mit folgerichtigem, schlüssigem Aufbau, d. h. bei Vorgabe eines Teils des Systems muß der Rest "vorhersagbar" sein.

Beispiel:
Erweiterung des Befehlsvorrates einer Rechenanlage um den Befehl "Quadratwurzel von" bei konsistenter Architektur:

- Realisierung des neuen Befehls weitgehend festgelegt
- Daten und Befehlsformat wie bei anderen arithmeitschen Befehlen
- Rundung und Genauigkeit wie bei anderen Resultaten
- Wurzel von negativer Zahl führt zum selben Exception Handling wie Division durch Null

Gegenbeispiel:
IBM-360 Floating-Point-Operation: "Halbiere" ohne Normalisierung des Ergebnisses (Implementierungsschwierigkeiten).

5.4.2.2 Orthogonalität:

Ein orthogonales System ist so aufgebaut, daß

- funktionell unabhängige Teilelemente auch unabhängig voneinander spezifiereirt und realisiert sind.
- Die Verbindung der Teilemenete zu einem geschlossenen System ist durch genormte Schnittstellen hergestellt, sodaß bei Elementausfällen oder -änderungen lediglich das betroffene Element ausgewechsel wird, während das übrige Gesamtsystem unverändert bleibt.

Die Orthogonalität wird oft auch als Modularität bezeichnet.

Beispiel:
Ein Verstoß gegen die Orthogonalität liegt etwa bei der Rechenanlage CD 3300 vor: Das Steuerwerk ist als eine Einheit nicht existent, die einzelnen Funktionen des Steuerwerks sind verteilt in anderen Elementen der gesamten Zentraleinheit verstreut. Dies erlaubt zwar ein vollen "Ausreizen" der Leistung der Anlage (die Schnittstellen werden eingespart), verkompliziert aber beispielsweise Änderungen der Steuerung enorm

Kosten der Orthogonalität:

- Schnittstellen
- Zeit
- Geld

5.4.2.3 Symmetrie:

Eigenschaft eines Systems, mathematisch symmetrische Dinge im System auch symmetrisch zu verarbeiten.

Beispiel:
Positive und negative Zahlen sollten an gleicher Stelle verwendet werden dürfen
Wenn der Befehl "Verzweige falls größer" existiert, so existiert der Befehl "Verzweige falls kleiner"

5.4.2.4 Angemessenheit:

Die Elemente eines Systems sind angemessen, wenn sie bei der vorgesehenen Problemstellung ausgeschöpft werden.

Beispiel:
Für einen sehr schnellen Prozessor ist eine schnelle Peripherie angemessen.

Gegenbeispiel:
Vorzeichen der Null im 1-Komplement: $\Rightarrow$ kein Vorteil bei der Problemlösung zusätzliches Statement wie "+0 = -0" ist nötig

Sparsamkeit:

Die Sparsamkeit versucht die Kosten eines Systems in Abhängigkeit von der gegebenen Technologie möglichst gering zu halten. Insbesondere sollten keine relativ unwichtigen Dinge realisiert werden.

Beispiel:
Stretch variable Byte-Länge (1-8 Bits) wieder aufgeben, da mehr Probleme erzeugt als gelöste wurden (z. B. match zwischen Bytes verschiedener Länge).

Originalton BLAAUW:
There appears to be a phase in each technological development cycle where there is a tendency to add bells and whistles.

Beispiel:
"Schnelle Multiplikation" in TR 4 so teuer, wie alle anderen Operationen zusammen. Zusätzlich nicht angemessen!

5.4.2.5 Transparenz:

Ein System ist transparent, wenn die Funktion des Gesamtsystems grob überschaubar ist (Details müssen nicht explizit zu erkenne sein).

Beispiel:;
Über Statusmeldungen, Quittungen etc. sollte der Benutzer Überblick über die ordnungsgemäße Funktion einer Rechenanlage gewinnen. Keine Einsicht in Registerbelegung notwendig.

Rechnernetz:
- Zielrechneradresse angeben
- Weg durchs Netz ist uninteressant

5.4.2.6 Virtualität:

Verdeckung der Begrenzung einer Implementierung

Beispiel: Virtueller Speicher

Nachteil: zusätzliche Kosten
Gefahr der Umgeheung durch Programmierer?

5.4.2.7 Kompatibilität:

Systeme (Rechner, Komponenten) sind kompatibel:
- bei gleichem Arbeitsauftrag (Programm, Daten, Funktion)
- unabhängig von der Zeit und
- der Anordnung der Einzelelemente

⇒ gleiche Ergebnisse

Beispiel:
Familie 360/370
verschieden schnelle Zentraleinheiten
Übergang:
- keine Änderung der Programme
- keine Änderung der Restkonfiguration

Bemerkungen:
- kompatible Rechner haben gleiche Befehlsarchitektur
- Geschwindigkeit geht nicht ein in die Definition
- kompatible Rechnerkomponenten ⇒ schrittweise Rekonfiguration möglich (Leistungssteigerung, Erweiterung des Anwendungsbereichses)

Kosten:
- genormte Schnittstellen
- Vernachlässigkbar wegen SW-Kosten

5.4.2.8 All-Anwendbarkeit

Ein System ist all-anwendbar, wenn es nicht auf eine einzige Anwendung zugeschnitten ist, sondern durch seine Flexibilität eine möglichst große Klasse von Problemen lösen kann (Vollständigkeit).

5.4.2.9 Dynamische Erweiterbarkeit (open-endedness)

Die dynamische Erweiterbarkeit charakterisiert ein System, das in jedem (erlaubten) Zustand die Erweiterung seiner Fähigkeiten zuläßt, dennoch aber in jedem Zwischenzustand in sich abgeschlossen und funktionsfähig ist.

Beispiel:
Mikropogrammierbare Rechner sind dynamsich erweiterbar, da der jeweilige Befehlssatz durch Hinzufügen neuer Instruktionen erweitert werden kann.

Typisch für die "open-endedness" sind auch nicht voll ausgenützte Codes, Konfiguarionstabellen (allgemein: Adreßräume).

5.4.3 Einteilung von Rechenanlagen

Art der Anwendung:
- Taschen-, Tischrechner, PC
- Rechner mit mittlerer DV
- Prozeßrechner
- Großrechner
- Spezialrechner
- ...

Art des Betriebs:
- Einzelbetrieb, Stapelbetrieb, Teilnehmerbetrieb
- funktionsorietierter Betrieb
- Realzeit
- Verbundbetrieb
- ....

Größe des Rechners
- Mikro, Mini, Midi
- Großrechner
- Superrechner
- ...

Art der Technologie

Art der Topologie und des Parallelismus
- Nebenläufigkeit, Fließbandverarbeitung
- RISC (= Reduced Instruction Set) - CISC (= Complex Instruction Set)

6 Befehlsarchitekturen

Befehlssatz: Menge der Maschienbefehle eines Rechners

6.1 Charakteristische Merkmale

6.1.1 Befehlsformat

n Operanden werden einer Operation unterworfen

Operationscode	Adreßfelder			
f	X_1	X_2	...	X_m

Häufig ist als Platzersparnis m ≤ n (z. B. ist im Befehl ADD 100 der Operand in ADD versteckt: der Befehl müßte eigentlich heißen: ADD Akku, 100)

$X_1, X_2, ..., X_m$ heißen explizite Operanden

der Rest heißt implizite Operanden

Man spricht von einer m-Adreßmaschine (normalerweise m = 1, 2 oder 3)

6.1.2 Adreßmodi

Adreßmodus:

- spezifiziert die Berechnung der Adresse des Operanden

Opcode • Adreßmodus • Zusatz

Adreßmodus und Zusatz bilden den Operand

- ist implizit im Befehlscode enthalten oder Teil des Adreßfeldes

Klassen

- absolut
- realtiv
- indirekt
- immediate

6.1.2.1 immediate

Operand ist unmittelbar im Adreßfeld (Konstante)

LW R8, #2000 bewirkt, daß die Zahl 2000 in das Register 8 geladen wird

6.1.2.2 indirekt

Adreßfeld beschreibt eine Adresse, dort steht die Adresse des Operanden

LW R8, (2000) bewirkt, daß die in 2000 stehende Adresse in das Register R8 geladen wird

6.1.2.3 absolut

Adresse steht selbst im Adreßfeld
LW R8, 2000 Die Adresse 2000 wird in das Register 8 geladen

6.1.2.4 relativ

Adreßfeld enthält ein sogenanntes Displacement D und die Adresse (Nummer) eines Registers R. Effektive Adresse ergibt sich als Summe D + (R)
R1: 3000 im Register 1 steht die Adresse 3000
LW R8, 2000 (R1) bewirkt das Laden der Adresse 5000 in Register 8

Vorteile:

- R implizit: kürzerer Befehl
- Programm oder Teil eines Programmes wird im Speicher verschiebbar, wenn alle Befehle, die auf den Speicher zugreifen, relative Adressierung benutzen.
  Zum Verschieben muß der Inhalt von R geändert werden. R heißt dann Basisregister
- R kann als Indexregister benutzt werden. D entspricht dann der Adresse des Elements V(0) eines Vektors. Für (R) = i ergibt sich die Adresse von V(i)

6.1.3 Anzahl der expliziten Adressen im Befehl (Speicher und Register)

üblich sind 1 - 3

Die Anzahl der expliziten Adressen beeinflußt die Länge der Befehle, die Länge der Programme und die Komplexität der Dekodierung (Schaltkreise).

Beispiel Addition (A = Akkumulator)
1-Adreßbefehl:
ADD X (A) + (X) $\rightarrow$ A (URA!)

2-Adreßbefehl: (es gibt 2 Implementierungen)
ADD X,Y (X) + (Y) $\rightarrow$ A
ADD X,Y (X) + (Y) $\rightarrow$ X

3-Adreßbefehl: (heute üblich)
ADD X,Y,Z (Y) + (Z) $\rightarrow$ X

Bemerkung:
m-Adreßmaschinen haben i. a. auch Befehle mit weniger Adreßfeldern

Einen Sonderfall bildet die 0-Adreßmaschine (= Kellermaschine)

Keller: geordnete Menge von Speicherplätzen, wobei nur auf die beiden obersten, gefüllten Plätze zugegriffen werden kann. Kellerzeiger (stack pointer) s zeigt auf obersten gefüllten Platz. Push down store

es gibt nur 2 Befehle mit einer Speicheradresse:

PUSH A s + 1 → s und A → s
POP A s → A und s - 1 → s

arithmetische Befehle enthalten keine Adresse:
VERK s - 1 → s und s VERK s + 1 → s

günstig zur Abarbeitung arithmetischer Ausdrücke durch Verwendung der umgekehrt polnischen Notation (erst Operanden, dann Operation):
Statt X + Y schreibt man X Y +

Beispiel:
X := A * B + C * D UPN: A B * C D * +

Das Programm dazu sieht wie folgt aus:
PUSH A
PUSH B
MULT
PUSH C
PUSH D
MULT
ADD
POP X

Die 0-Adreßmaschine ist heute ausgestorben aber der Keller ist in vielen modernen Maschinen realisiert. Er ist meist als Stapel (stack) implementiert, d. h. es besteht die Möglichkeit, auf alle Elemente des Stapes über den Stackpointer zuzugreifen.

6.1.4 Anzahl der explizit angegebenen Speicherzugriffe

3-Adreßmaschine:
0, 1, 2 oder 3 Speicheradressen pro Befehl
3, 2, 1 oder 0 Registeadressen pro Befehl

Vorteil von Registeradressen: kürzere Befehle

Vorteil von mehr Speicheradressen: weniger Befehle
Nachteil von mehr Speicheradressen: Hardware aufwendiger

Load&Store-Maschine (heute weit verbreitet)
3-Register Befehle
Speicherzugriffe nur:
LW R, Speicheradresse
SW Speicheradresse, R

Beispiel:
X := A * B + C * D

1-Adreß (1 Speicher)	2-Adreß (2 Speicher)	3-Adreß (3 Speicher)	Load&Store
LW A	MOVE T,A	MULT X,A,B	LW R1, A
MULT B	MULT T,B	MULT T,C,D	LW R2, B
SW T	MOVE X,C	ADD X,X,T	MULT R1,R1,R2
LW C	MULT X,D		LW R2,C
MULT D	ADD X,T		LW R3,D
ADD T			MULT R2,R2,R3
SW X			ADD R1,R1,R2
			SW X,R1

6.1.5 Befehlsklassen

Datentransfer	move	Speicher ←→ Speicher
		Register ←→ Register
	load	Speicher → Register
	store	Register → Speicher
arithmetische	ADD, SUB, MULT, DIV, usw.	
logische	Boole'sche Wortoperationen	
Kontrollbefehle	(bedingte) Sprünge, Unterprogrammaufrufe	
E/A	Einleitung und Überwachung	

7 Index

W

Z