SKRIPT[1] ZUR LEHRVERANSTALTUNG

GRUNDZÜGE DER BETRIEBSWIRTSCHAFTSLEHRE

Teil A 3: Entscheidung und Information

Grundlagen der computergestützten betrieblichen Informationsverarbeitung

WS 2004/05

Univ.-Prof. Dr.-Ing Günter Schmidt
Lehrstuhl für Betriebswirtschaftslehre, insbesondere Informations- und Technologiemanagement
Universität des Saarlandes

Überarbeitet von Benjamin Olschok
(Version 1.0)

[1]Teile des Inhalts basieren auf [ABCW02], [FS01], [Jan98] und [Sch99].

INHALTSVERZEICHNIS

1 MOTIVATION UND BEGRIFFE

Zu Beginn wollen wir zur Motivation ein Warenwirtschaftssystem als Beispiel der computergestützten Informationsverarbeitung etwas genauer betrachten. Mit Warenwirtschaftssystemen kommt man in jedem größeren Handelsbetrieb in Kontakt. Die wesentlichen Komponenten des Systems und ihr Zusammenwirken sind in Abbildung 1-1 dargestellt.

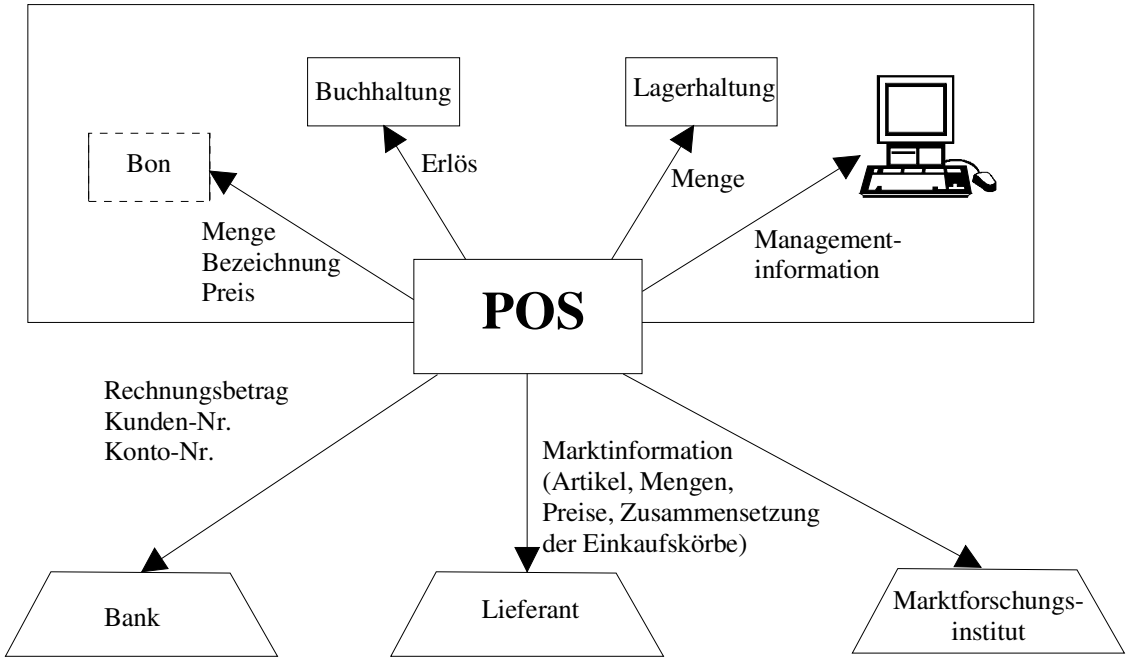


Abb. 1-1: *Komponenten eines Warenwirtschaftssystems nach [MBKPS01]*

Der Point of Sale (POS) ist in diesem Beispiel die zentrale Komponente, von der alle anderen Komponenten Informationen beziehen. Diese werden im Unternehmen (Buchhaltung, Lagerhaltung, Managementinformationen) und über das Unternehmen hinaus (Bank, Lieferanten, Marktforschungsinstitute) weitergeleitet.

Bevor ein solches System zum Einsatz kommt, bedarf es vielfältiger konstruktiver und organisatorischer Aktivitäten. Um diese verstehen zu können, sind zunächst die wichtigsten Begriffe, die bei diesen Aktivitäten eine Rolle spielen, zu klären.

Grundlegend für den Bereich der computergestützten Informationsverarbeitung ist der Begriff der Menge. Eine *Menge* ist die Kollektion von unterscheidbaren Objekten. Die Objekte nennt man *Elemente*. Die Elemente einer Menge werden in geschweiften Klammern aufgelistet. Schreiben wir $V = \{\text{Auto, Zug, Flugzeug, Boot}\}$, so meinen wir damit eine Menge V, welche die vier Elemente Auto, Zug, Flugzeug und Boot enthält. Ein Spezialfall ist eine Menge, die kein Element enthält; die leere Menge wird durch das Symbol $\varnothing$ dargestellt. Standardmengen sind die der natürlichen Zahlen $\mathbb{N}$, der ganzen Zahlen $\mathbb{Z}$, der rationalen Zahlen $\mathbb{Q}$ und der reellen Zahlen $\mathbb{R}$.

Man schreibt $s \in S$ bzw. $t \notin S$, wenn s in S enthalten bzw. t nicht in S enthalten ist. Ist eine Menge R in einer Menge S enthalten, d.h. jedes Element von R ist auch Element von S, so heißt R Teilmenge von S (geschrieben $R \subseteq S$). R ist echte Teilmenge von S (geschrieben $R \subset S$), wenn R Teilmenge von S ist und mindestens ein Element von S nicht in R ist.

Mengen lassen sich durch eine Eigenschaft P darstellen, die alle ihre Elemente erfüllen müssen: $\{s \mid P(s)\}$. So beschreibt die Menge $I = \{x \mid 2<x<5\}$ das offene Intervall (2,5). Die Menge aller Teilmengen von S, d.h. $\wp(S) = \{R \mid R \subseteq S\}$, heißt Potenzmenge von S, wobei card $|\wp(S)| = 2^{|S|}$. Möchte man die Anzahl der Elemente einer Menge S angeben, so bestimmt man die Kardinalität von S und schreibt $|S|$. Die Kardinalität der obigen Menge V ist vier, d.h. $|V| = 4$.

Folgende Operationen kann man auf Mengen ausführen. O.B.d.A. seien die Operatoren nur für die beiden Mengen X und Y definiert:

Vereinigung:	$X \cup Y = \{z \mid z \in X \text{ oder } z \in Y\}$
Durchschnitt:	$X \cap Y = \{z \mid z \in X \text{ und } z \in Y\}$
Differenz:	$X \setminus Y = \{z \mid z \in X \text{ und } z \notin Y\}$.
Kartesisches Produkt:	$X \times Y = \{(x, y) \mid x \in X \text{ und } y \in Y\}$

Sei R Teilmenge von S, dann heißt $\overline{R} = \{s \mid s \in S \text{ und } s \notin R\} = S \setminus R$ das Komplement der Menge R. Jede Teilmenge $R \subseteq S_1 \times \ldots \times S_n$ heißt n-stellige Relation. Für n=2 spricht man von einer

binären Relation. Eine spezielle binäre Relation ist die partielle Abbildung. Eine partielle Abbildung f ordnet jedem Element einer Menge A höchstens ein Element der Menge B zu (f: $A \rightarrow_p B$). Eine partielle Abbildung ist eine Relation $R \subseteq A \times B$ mit der Eigenschaft: wenn (a,b) und (a,c) in R liegen, muss $b=c$ sein. Eine (totale) Abbildung ordnet *jedem* Element aus A *genau ein* Element aus B zu. Hierbei gilt, dass zu jedem $a \in A$ genau ein $b \in B$ mit $(a,b) \in R$ existiert (f: $A \rightarrow B$). In gleicher Weise lassen sich n-stellige Abbildungen definieren. Ist die Menge B ein Zahlenbereich, z.B. die Menge der reellen Zahlen, dann nennt man eine Abbildung auch eine *Funktion*.

Häufig auftretende Funktionen sind Polynome, Exponentialfunktionen und Logarithmen. Polynome sind von der Form

$p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$,

mit x als Variable und $n \in \mathbb{N}_0$; n heißt Grad des Polynoms p. Ist $n=1$, ist das Polynom linear. Exponentialfunktionen enthalten einen Term k^x mit $k > 1$. Das Wachstum von Exponentialfunktionen ist deutlich größer als das von Polynomen. Logarithmen bilden die Umkehrfunktion von Exponentialfunktionen. Es gilt für $a > 0$ und eine Zahl $r \in \mathbb{R}$ mit $r > 1$: $\log_r(a) = x$ genau dann, wenn $r^x = a$ ist.

Um Funktionen $f(x)$ und $g(x)$ miteinander vergleichen zu können, wird ihr Verhalten für große Werte von x analysiert. So wächst beispielsweise $f_1(x) = x^2$ schneller als $f_2(x) = 5x$. Für eine Funktion f definiert man die Menge $O(f)$ (lies "Ordnung") aller Funktionen, die bis auf eine Konstante für große x höchstens so rasch wachsen wie f. Eine Funktion $g(x)$ ist $O(f(x))$, wenn Konstanten c und $x_0 \in \mathbb{N}$ existieren, so dass $|g(x)| \leq c|f(x)|$, für alle $x \in \mathbb{N}$; $x \geq x_0$.

Ein wichtiges Hilfsmittel zur Repräsentation einer Anwendung ist ein *Graph*. Es sei V die Menge der Objekte und E eine binäre (Nachbarschafts-) Relation, bestehend aus einer Teilmenge von $V \times V$. Ein Graph G besteht aus der Menge der *Knoten* V und der Menge der *Kanten* E. *Knotenmarkierungen* und *Kantenmarkierungen* ordnen Knoten und Kanten numerische oder symbolische Ausdrücke zu. Sind zwei Knoten v und w, $v,w \in V$, durch eine Kante $e \in E$

miteinander verbunden, dann sind v und w inzident mit der Kante e. Ist v=w, so nennt man e eine Schlinge. In Abbildung 1-2 ist ein ungerichteter Graph $G = (V, E)$ mit $V = \{1, 2, 3\}$ und $E = \{(1,2), (2,3), (3,1), (3,3)\}$ dargestellt.

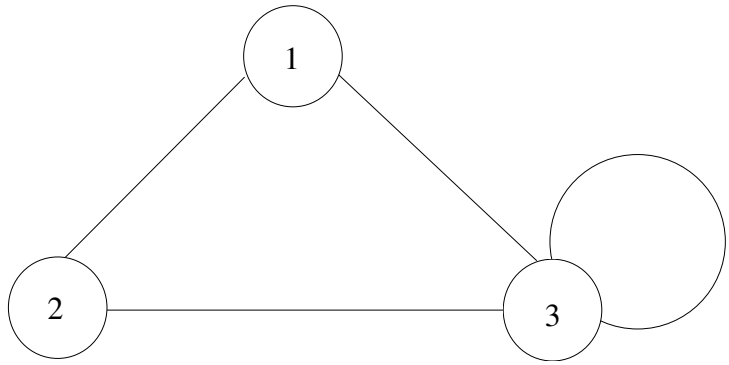


Abb. 1-2: *Ungerichteter Graph*

Wenn es sich um gerichtete Beziehungen handelt, wird bei einer Relation zwischen (a,b) und (b,a) unterschieden. Bei dem Graphen handelt es sich dann um einen *Digraphen* bzw. gerichteten Graphen (vgl. Abbildung 1-3). Die Beziehung zwischen zwei Knoten heißt dann Pfeil und die Menge der Pfeile wird mit A bezeichnet. Sind Anfangs- und Endknoten eines Pfeiles identisch, so spricht man von einer gerichteten Schlinge. Zu jedem gerichteten Graphen D kann dann ein ungerichteter Graph G angegeben werden, wenn wir Pfeile der Form (a,b) und (b,a) durch eine Kante (a,b) ersetzen können.

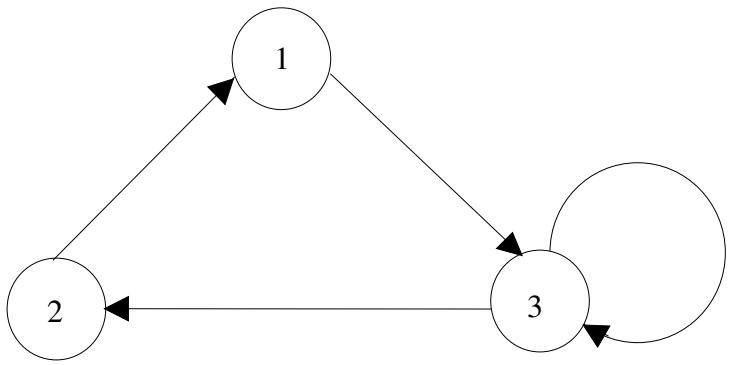


Abb. 1-3: *Gerichteter Graph (Digraph)*

Eine endliche Folge von durch Knoten verbundenen Kanten (Pfeilen) eines Graphen (Digraphen) bildet einen *Kantenzug (Pfeilzug)*. Sind Anfangs- und Endknoten verschieden, so handelt es sich um einen *offenen*, im anderen Fall um einen *geschlossenen* Kanten- bzw. Pfeilzug, der auch Zyklus genannt wird. Ein Kanten- bzw. Pfeilzug, in dem jede Kante bzw. jeder Pfeil nicht mehr als einmal vorkommt, heißt *Weg*. Beispielsweise bildet der Weg (1,2), (2,3) und (3,1) in Abbildung 1-3 einen Zyklus. Beim Graphen heißt ein offener Weg *Kette* und ein geschlossener Weg *Kreis*. Beim Digraphen heißt der offene Weg *Pfad*, der geschlossene *Schleife*. Ist kein Knoten mit mehr als zwei Kanten bzw. Pfeilen eines Weges inzident, spricht man von einem *einfachen* Weg. Ein Graph ist *zusammenhängend*, wenn zwischen je zwei verschiedenen Knoten des Graphen eine Kette besteht.

Zwei Knoten heißen *benachbart*, wenn sie durch einen Kanten- bzw. Pfeilzug verbunden sind. Bei einem Pfeil (a,b) ist a der direkte Vorgänger von b und b der direkte Nachfolger von a. Jeder Knoten u, der von einem Knoten v erreicht werden kann, heißt Nachfolger von v. Die Menge der Nachfolger von v wird mit $N(v)$ bezeichnet. Alle Knoten, von denen aus v erreicht werden kann, heißen Vorgänger von v und werden mit $V(v)$ bezeichnet. Für den Graphen in Abbildung 1-3 gilt, dass $N(2) = \{1, 3\}$ und $V(1) = \{2, 3\}$.

Informationen sind Daten, die bezogen auf einen Zweck eine Bedeutung haben. Daten bestehen aus Zeichen, die kontinuierlich oder diskret auftreten. So wird elektrischer Strom durch kontinuierliche Zeichen und die Signale einer Ampel durch diskrete Zeichen beschrieben. Kontinuierliche Daten werden durch reelle Zahlen und diskrete Daten durch natürliche Zahlen repräsentiert. Die heutige Computertechnologie verarbeitet diskrete Zeichen. Die Menge der Zeichen, die man zur Darstellung von Daten benutzen möchte, heißt Alphabet. Ein Alphabet ist eine endliche Menge $A = \{a_1, a_2, ..., a_n\}$. Die Elemente heißen Zeichen oder Buchstaben. Die Menge $W = \{w_1\, w_2\, ...\, w_m \mid m \in \mathbb{N}_0, w_i \in A \text{ für } i=1,2,...,m\}$ heißt Menge der Wörter über A. Sei u ein Wort $u = w_1\, w_2 ... w_m$ so ist $m = |\, u\, |$ die Länge des Wortes u.

Informations- und Kommunikationssysteme (*IuK-Systeme*) dienen der Bereitstellung von Informationen zur Problemlösung. Dazu benötigen wir Bedingungen (Aussagen), Anweisungen (Funktionen), Dokumente (Daten) und eine Ablaufvorschrift (Kontrollstruktur). Eine Bedingung oder Aussage ist eine Formulierung, der man die Wahrheitswerte wahr oder falsch zuordnen kann. Aussagen können mit Hilfe von Operatoren wie beispielsweise **`and`**, **`or`** und **`not`** zu neuen Aussagen verknüpft werden. Es sei E eine Menge elementarer Aussagen. Dann ist die Menge A der Aussagen über E definiert als kleinste Menge mit den folgenden Eigenschaften: (1) ist $B \in E$, dann gilt auch $B \in A$; (2) ist $B \in A$, dann ist auch **`not`** $B \in A$; (3) sind B_1 und B_2 aus A, dann sind auch (B_1 **`and`** B_2) und (B_1 **`or`** B_2) aus A.

Ein System umfasst eine *Menge* von Elementen und eine Menge von Beziehungen zwischen den Elementen. Ein System läßt sich beschreiben durch eine Menge von Elementen und eine Menge von Relationen, die die Beziehungen der Elemente angibt und eine Teilmenge des kartesischen Produkts der Teilmenge ist. Ein System hat eine Systemgrenze, Input, Output und ein Systemziel [Sch99].

IuK-Systeme lassen sich im *engeren* und im *weiteren* Sinne betrachten. Im engeren Sinne sind die Elemente Hardware (Rechner, Maschinen) und Software (Anweisungen zur Benutzung der Rechner bzw. Maschinen); im weiteren Sinne ist auch der menschliche Nutzer ein Element eines IuK-Systems. Wissenschaftliche Disziplinen, die sich mit dem Entwurf und Einsatz von IuK-Systemen für Unternehmen beschäftigen, sind *Informationsmanagement* und *Wirtschaftsinformatik*. Informationsmanagement beschäftigt sich mit der Nutzung der Ressource Information für möglichst gute Lösungen betriebswirtschaftlicher Probleme [Sch99]. Wirtschaftsinformatik befasst sich mit Systemen der computergestützten Informationsverarbeitung im Betrieb [MBKPS01].

Die Bereitstellung von Informationen lässt sich weiter unterscheiden in Beschaffung, Verarbeitung, Aufbewahrung und Weiterleitung. Beispiele betrieblicher IuK-Systeme, die diese Aufgaben unterstützen, sind:

- Kommunikationssysteme zur Beschaffung und Weiterleitung,
- Datenbanksysteme zur Aufbewahrung,
- Administrationssysteme zur Verwaltung und Abrechnung,
- Planungs- und Dispositionssysteme zur Entscheidungsfindung.

Eingesetzt werden IuK-Systeme für unterschiedliche betriebliche Funktionen wie Forschung und Entwicklung, Vertrieb, Beschaffung, Leistungserstellung, Finanzierung, Personal, Verwaltung, Rechnungswesen, etc. und in unterschiedlichen Branchen wie Industrie, Handel, Verkehr, Banken oder Versicherungen. Ein Überblick zu verschiedenen betrieblichen Informationssystemen findet man in [Sch99].

2. REPRÄSENTATION VON DATEN UND FUNKTIONEN

Wesentliche Elemente von IuK-Systemen sind Daten und Funktionen. In diesem Teil wird ihre Darstellung in Rechnern beschrieben.

2.1 DARSTELLUNG VON DATEN

Daten bestehen aus *Zeichen*. Ein Zeichen z ist ein Element aus einer endlichen Menge Z von (unterscheidbaren) Elementen, dem Alphabet oder Zeichenvorrat. Beispiele für Alphabete sind die Menge der großen Buchstaben {A, B, ..., Z} oder die Menge der Dezimalziffern {0, 1, ..., 9}.

Daten in Textform können aus einer Zeichenfolge beliebiger Länge von großen und kleinen Buchstaben, Dezimalziffern sowie Sonderzeichen bestehen. Eine endliche Zeichenfolge, die als Einheit betrachtet wird, heißt *Wort*.

Die digitale Darstellung von Zahlen benutzt das binäre Alphabet $B = \{0,1\}$. Seine Elemente heißen Binärzeichen bzw. Bit (binary digit). Der Begriff Bit ist in gleicher Weise wie die Begriffe Buchstabe oder Dezimalziffer zu verwenden; man spricht vom Bit 0 bzw. vom Bit 1 oder von n Bits als Bitfolge der Länge n.

Um Daten durch Rechner verarbeiten zu können, wird jedes Zeichen eineindeutig auf eine Bitfolge abgebildet. Eine solche Abbildung wird auch Code genannt. Eine Reihe von Codes sind international standardisiert. Einer der wichtigsten ist ASCII (American Standard Code for Information Interchange). ASCII ist eine Abbildung der Form

ASCII: $A \rightarrow B^7$.

Die Menge *A* besteht aus den großen und kleinen Buchstaben, den Dezimalziffern und einer Reihe von Sonderzeichen. B^7 ist die Menge der 7-stelligen Worte über dem Alphabet $B = \{0,1\}$, d.h.

$$B^7 = B \times B \times B \times B \times B \times B \times B.$$

Die Kardinalität der Menge B^7 beträgt $| B^7 | = 2^7 = 128$ Elemente. *A* kann daher maximal 128 Zeichen umfassen. Die bei ASCII gewählte Abbildung ist in Tabelle 2-1 dargestellt. Die Positionen der einzelnen Bits in den 7-stelligen Bitfolgen werden dabei von rechts nach links mit 0 bis 6 bezeichnet.

ASCII ordnet beispielsweise dem Buchstaben 'A' die Bitfolge 100 0001, dem Sonderzeichen '$' die Bitfolge 010 0100 zu. Die ASCII-Zeichen der ersten beiden Spalten (000 0000 bis 001 1111) sind Steuerzeichen für die Datenübertragung und -formatierung (z.B. ACK = acknowledge, FF = form feed, CR = carriage return).

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

Tab. 2-1: *ASCII Codierung*

Die 128 möglichen Bitfolgen lassen sich der Größe nach sortieren, so dass

000 0000 < 000 0001 < 000 0010 < 000 001 1 < ... < 111 1111.

ASCII ist so vereinbart, dass beim Vergleich von zwei ASCII-Zeichen die Sortierung des Alphabets *A* erhalten bleibt.

Die manuelle Schreibweise langer Bitfolgen ist unhandlich. Um sie durch eine kürzere Schreibweise zu ersetzen, wird die Abbildung

HEX: $B^4 \rightarrow S$

mit $S = \{0, 1, ..., 9, A, B, ..., F\}$ eingeführt. Die Elemente von *S* heißen Hexadezimalziffern. Es gilt die in Tabelle 2-2 dargestellte Zuordnung.

Bit-folge	Hex-ziffer	dez. Wert	Bit-folge	Hex-ziffer	dez. Wert
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

Tab. 2-2: *Codierung mit Hexadezimalziffern*

Bei der Codierung von ASCII Zeichen mit Hexadezimalziffern wird eine erweiterte Wortmenge B^8 verwendet, bei der die Bits der (von links gezählten) Positionen 1 bis 7 mit der Definition von ASCII übereinstimmen; davor wird auf der Position 0 das Bit 0 eingeführt. Die hexadezimale Schreibweise eines ASCII-Zeichens wird somit durch die Abbildung $A \rightarrow B^8 \rightarrow S^2$ bestimmt. Un-

ter Verwendung dieser Darstellung ist der Wert des Buchstabens 'A' codiert als $0100\ 0001_B$ oder 41_H . Die Indizes B und H identifizieren den Code als Bitfolgen bzw. Folgen von Hexadezimalziffern.

Weitere Beispiele für Codes sind Extended ASCII, eine von IBM definierte Erweiterung von ASCII (eASCII: $A' \rightarrow B^8$) sowie der im Bereich der Großrechnerfamilien von IBM und Siemens übliche Extended Binary Coded Decimal Interchange Code (EBCDIC: $A'' \rightarrow B^8$). Die Mengen A' und A'' umfassen jeweils 256 Zeichen (vgl. Tabelle 2-3).

Zeichen	eASCII	dezimal	EBCDIC	dezimal
1	0011 0001	(49)	1111 0001	(241)
5	0011 0101	(53)	1111 0101	(245)
9	0011 1001	(57)	1111 1001	(249)
A	0100 0001	(65)	1100 0001	(193)
a	0110 0001	(97)	1000 0001	(129)
R	0101 0010	(82)	1101 1001	(217)
r	0111 0010	(114)	1001 1001	(153)
T	0101 0100	(84)	1110 0011	(227)
t	0111 0100	(116)	1010 0011	(163)
+	0010 1011	(43)	0100 1110	(78)
?	0011 1111	(63)	0110 1111	(111)

Tab. 2-3: *Beispiele für eASCII und EBCDIC Codierung*

Die Darstellung von *Zahlen* kann auf verschiedene Weise erfolgen. Beispielsweise benutzt das uns vertraute Dezimalsystem die Basis 10, das Dualsystem die Basis 2, das Oktalsystem die Basis 8 und das Hexadezimalsystem die Basis 16:

Dezimal: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ und Basis 10

Beispiel: $4*10^2 + 0*10^1 + 9*10^0 = (409)_{10}$

Dual: {0, 1} und Basis 2

Beispiel: $409 = 1*2^8+1*2^7+1*2^4+1*2^3+1*2^0 = (110011001)_2$

Oktal: {0, 1, 2, 3, 4, 5, 6, 7} und Basis 8

Beispiel: $409 = 6*8^2 + 3*8^1 + 1*8^0 = (631)_8$

Hexadezimal: {0, ..., 9, A, B, ..., F} und Basis 16

$409 = 1*16^2 + 9*16^1 + 9*16^0 = (199)_{16}$

Allgemein: $Z = \Sigma_{i=0,...,n-1} \; z_i * b^i$

mit Z als Wert einer Zahl, z_i als Ziffern und b als Basis

Die folgende Tabelle 2-4 stellt die Darstellungsweisen beispielhaft im Zusammenhang dar.

Dezimal	Dual	Oktal	Hexadezimal
1	00001	01	01
2	00010	02	02
3	00011	03	03
4	00100	04	04
5	00101	05	05
6	00110	06	06
7	00111	07	07
8	01000	10	08
9	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

Tab. 2-4: *Beispiele für Zahlendarstellungen*

Eine einfache Form der Darstellung von Zahlen des Dezimalsystems unter Verwendung des binären Alphabets {0,1} ist die ziffernweise Codierung mit ASCII. Diese Darstellungsform heißt Dezimalcode. Der Dezimalcode wird vor allem für den Nachrichtenaustausch mit Geräten zur Mensch-Maschine-Kommunikation eingesetzt (Bildschirme, Drucker usw.). Zur ausschließlichen Darstellung von Zahlen ist er allerdings unwirtschaftlich, da für die Codierung der zehn Ziffern des Dezimalsystems nur 10 von 128 ASCII-Zeichen bzw. 10 von 256 eASCII-Zeichen verwendet werden. Außerdem eignet sich der Dezimalcode nur bedingt zur Durchführung arithmetischer Operationen in Rechnern. Aus diesem Grund werden zwei weitere Codes eingeführt.

BINÄRER FESTPUNKTCODE

Der binäre Festpunktcode dient zur Darstellung von Zahlen mit konstanter Anzahl von Dezimalstellen. Festpunktzahlen sind durch Multiplikation mit einem konstanten Faktor auf ganze Zahlen zurückführbar. Zum Beispiel können Euro-Beträge mit zwei Stellen nach dem Dezimalkomma durch Multiplikation mit dem Faktor 100 immer als ganze Zahlen dargestellt werden. Der binäre Festpunktcode beruht auf der Transformation einer im Dezimalsystem dargestellten, positiven ganzen Zahl Z in eine Zahl zur Basis 2. Die Koeffizienten c_i sind Bits. Der binäre Festpunktcode transformiert Z in die nachstehende Form:

Ziffern von Z: $c_{n-1}\ c_{n-2} \ldots c_0$

Wert von Z: $Z = c_{n-1} * 2^{n-1} + c_{n-2} * 2^{n-2} + \ldots + c_0 * 2^0$

Dagegen wird im Dezimalsystem Z wie folgt als Polynom mit der Basis 10 dargestellt; die Koeffizienten z_i sind Dezimalziffern:

Ziffern von Z: $z_{n-1}\ z_{n-2} \ldots z_0$

Wert von Z: $Z = z_{n-1} * 10^{n-1} + z_{n-2} * 10^{n-2} + \ldots + z_0 * 10^0$

In den Tabellen 2-2 und 2-3 lassen sich entsprechende Beispiele für $n = 4$ und $n = 8$ finden.

BINÄRER GLEITPUNKTCODE

Eine reelle Zahl r wird dargestellt in der Form

$r = m * b^e$.

m heißt Mantisse, e Exponent zur Basis b. Es gilt:

- e ist eine ganze Zahl,
- m wird so gewählt, dass $b^{-1} < |m| < 1$ oder $m=0$ gilt,
- die Vorzeichen von m und e lassen sich berücksichtigen.

Bei dieser Darstellung wird somit eine reelle Zahl in die Komponenten Mantisse und Exponent zerlegt. Dabei hat beim binären Gleitpunktcode die Basis b den Wert 2. Mantisse und Exponent werden als Festpunktzahlen im binären Festpunktcode dargestellt.

In einem für uns leichter lesbaren "dezimalen Gleitpunktcode" wird r = 3,14159 dargestellt als $0{,}314159 * 10^1$ mit $m = 0{,}314159$, $b = 10$ und $e = 1$.
m und e werden als Tupel (0,314159;1) dargestellt.

Gängige Wortlängen für reelle Zahlen sind 32 und 64 Bits. Im ersten Fall sind 24 Bits für die Mantisse und 8 Bits für den Exponenten vorgesehen.

2.2 DATENSTRUKTUREN UND DATENORGANISATION

Häufig werden Daten durch Relationen repräsentiert. Relationen lassen sich als Mengen von gleichartig strukturierten Tupeln darstellen, wobei jedes Tupel ein Objekt oder eine Beziehung der Vorstellungswelt beschreibt. Wir wollen annehmen, dass ein Unternehmen Würfel herstellt (vgl. Abbildung 2-1). Das Produkt soll mit Hilfe von Relationen beschrieben werden.

Schreiben wir (x,y) für die Beziehung 'x grenzt an y' mit $x \neq y$ und $x,y \in S = \{a, b, c, d, e, f\}$, so lässt sich die Hülle eines Würfels durch die Relation

R = {(a,b), (a,c), (a,e), (a,f), (b,c), (b,d), (b,f), (c,e), (c,d), (d,e), (d,f), (f,e)} beschreiben.

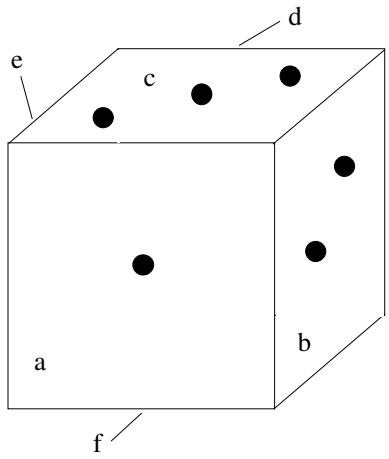


Abb. 2-1: *Würfel*

Eine einfache Darstellungsform von Relationen sind Tabellen. Bei unserem Beispiel repräsentieren die Paare (x,y) die Spalten einer Tabelle und die Zeilen repräsentieren verschiedene Würfel. Zu ihrer besseren Unterscheidung können wir den Flächen noch zusätzliche Attribute wie 'Farbe' und 'Zahl' zuordnen (vgl. Abbildung 2-2).

	a	b	c	d	e	f
Farbe (W_1)	rot	grün	blau	gelb	schwarz	weiss
Farbe (W_2)	blau	rot	gelb	schwarz	weiß	grün
Zahl (W_1)	1	2	3	6	5	4
Zahl (W_2)	3	6	5	4	1	2
•	•	•	•	•	•	•
•	•	•	•	•	•	•
•	•	•	•	•	•	•

Abb. 2-2: *Relation als Tabelle*

Zur graphischen Darstellung der Relation *R* verwenden wir Knoten *x* und *y* sowie Kanten (*x*,*y*). Ein Knoten repräsentiert eine Fläche und eine Kante eine Relation, die zwischen Flächen besteht (vgl. Abbildung 2-3).

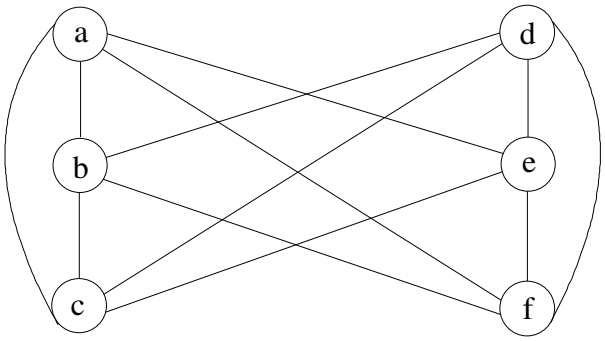


Abb. 2-3: *Relation als Graph*

Wir geben im Folgenden einen kurzen Überblick über weitere elementare Datenstrukturen. Dabei gehen wir von folgenden Zusammenhängen aus. Die kleinste abzubildende Einheit ist ein Datenelement; mehrere Datenelemente bilden einen Datensatz und mehrere Datensätze bilden eine Datei. Ein Schlüssel dient der eindeutigen Identifikation von Daten.

2.2.1 LINEARE FELDER

Ein lineares Feld, auch Liste genannt, kann man sich als eine Spalte oder Zeile einer Tabelle vorstellen. Dabei hat eine Zeile soviele Datenelemente wie es Spalten gibt und eine Spalte hat soviele wie es Zeilen gibt (vgl. Abbildung 2-4).

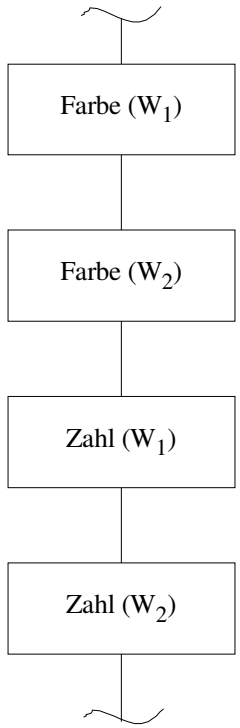


Abb. 2-4: *Ein Beispiel für Listen*

Lineare Felder werden durch eine einfache Nachfolger-Relation beschrieben. Jedes Datenelement weist auf sein einziges Nachfolger-Datenelement. Dies lässt sich auf zwei Arten abbilden: sequenziell und gestreut. Bei der gestreuten Speicherung unterscheidet man verkettet oder unverkettet.

Bei der *sequenziellen* Speicherung werden Felder in einer unverketteten (sortierten) Reihenfolge abgelegt. Das Hinzufügen, Aufsuchen und Löschen eines linearen Feldes bei sequenzieller

Speicherung ist sehr mühsam (vgl. Abbildung 2-5). Zum Beispiel müssen beim Einfügen eines Datenelementes alle nachfolgenden Elemente verschoben werden.

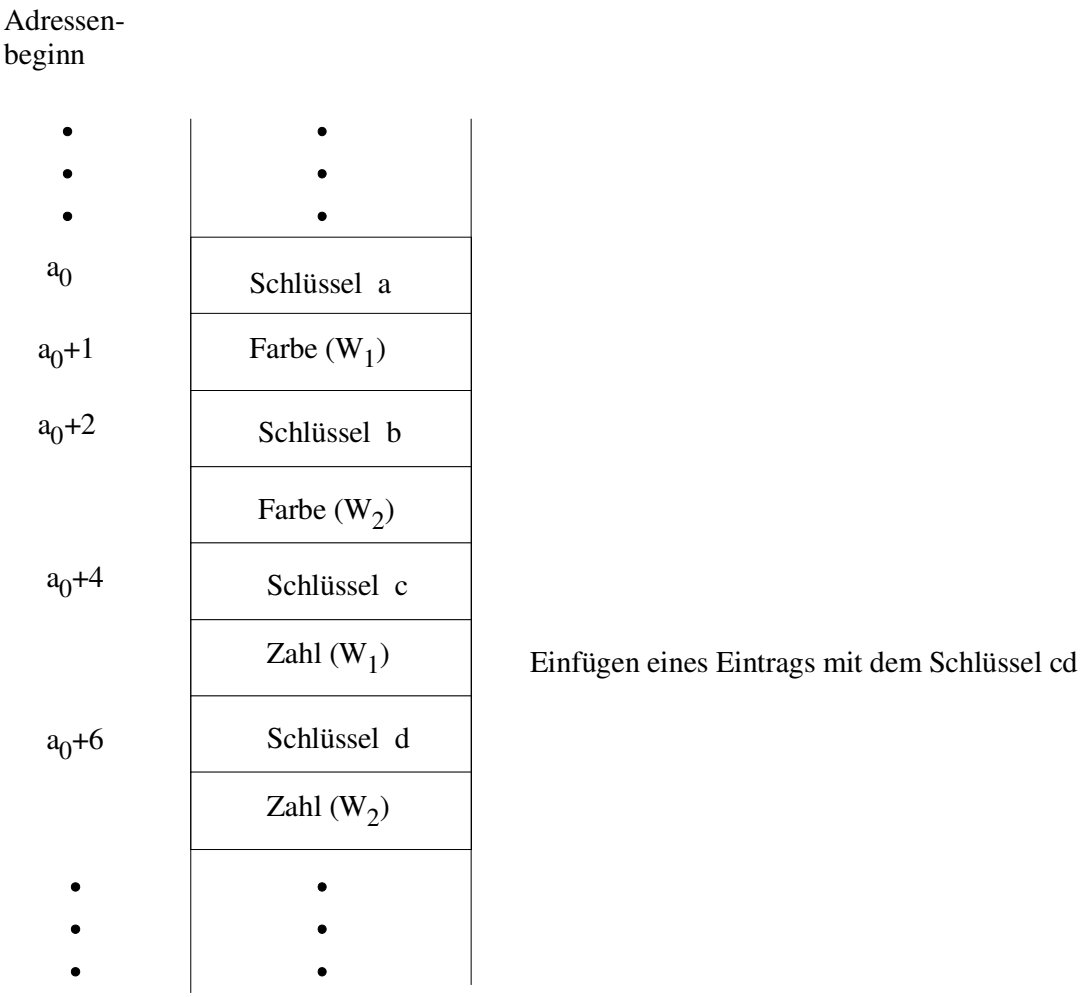


Abb. 2-5: *Sequenzielle Speicherung*

Bei der *gestreuten* Speicherung mit Verkettung werden so genannte Zeigerfelder verwendet, um die Adresse des Startfeldes, eines folgenden Feldes oder das Ende einer Verkettung anzugeben. Es ist möglich, Datenelemente über die Verkettung einzufügen, ohne eine Verschiebung von Datensätzen vorzusehen. In Abbildung 2-6 wird beispielsweise der Datensatz cd zwischen die Datensätze c und

d eingefügt. Es werden nur Zeiger geändert, ohne Datensätze zu verschieben.

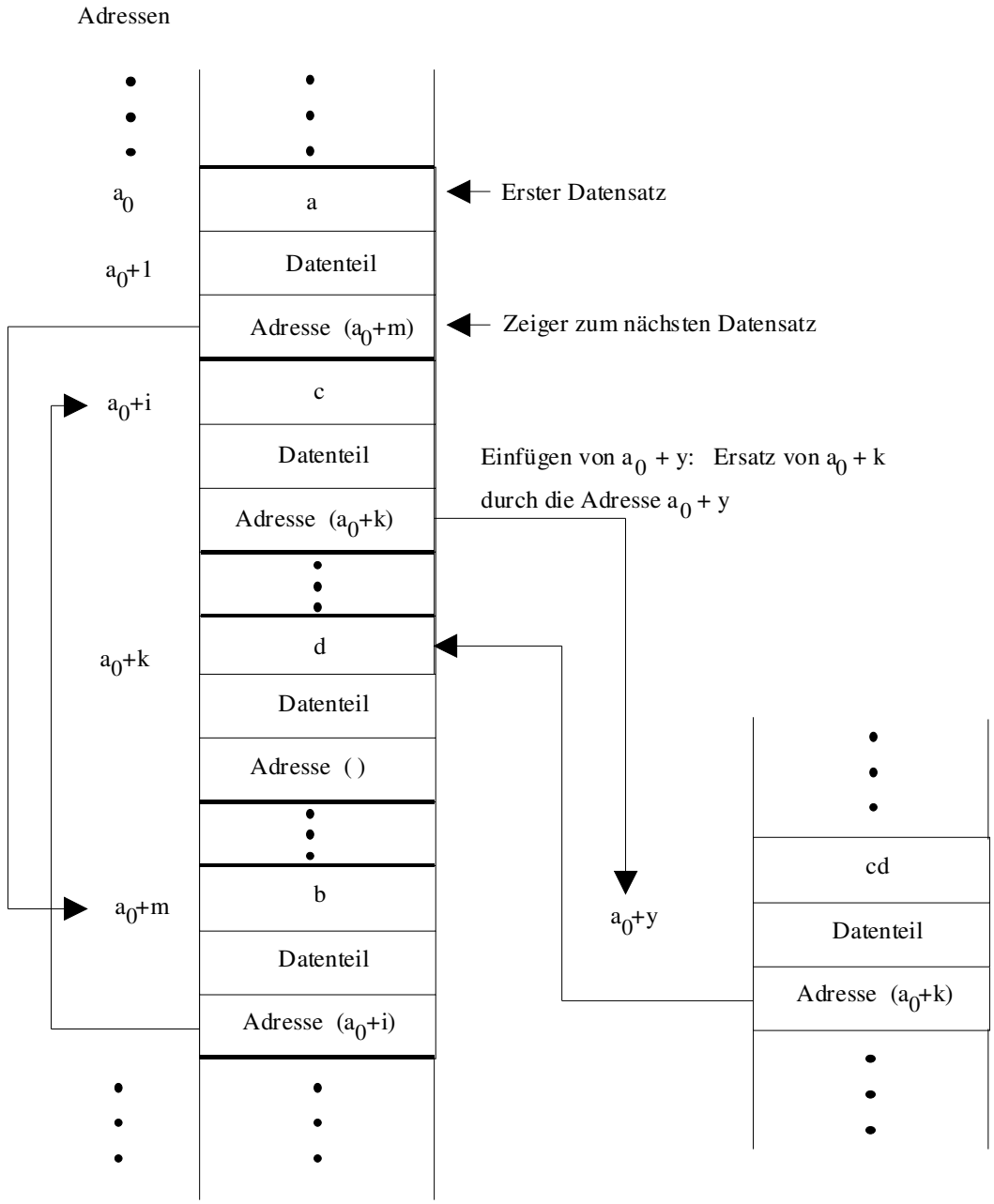


Abb. 2-6: *Gestreute Speicherung mit Verkettung*

Das Prinzip der unverketteten gestreuten Speicherung liegt in der Errechnung einer Speicheradresse aus einem Suchschlüssel. Die Funktion, die aus einem numerischen Schlüssel (NS) eine Speicheradresse errechnet, heißt Hash-Funktion. Folgendes Beispiel verdeutlicht das Arbeiten mit Hash-Funktionen.

Sollen 200 Datensätze mit den numerischen Schlüsseln 100 bis 299 zu je 10 Sätzen pro Spur den Spuren 20 bis 39 einer Platte zugeordnet werden, lautet die Speicherfunktion:

(NS-100) / 10 = *q* mit Rest *r* ; Spur-Nummer: *q*+20 ; Position auf Spur: *r*+1;
d.h. der Datensatz mit dem numerischen Schlüssel 150 hat die Speicheradresse Spur 25, Position 1.

Spezielle lineare Felder sind der Stapel (Keller) und die Schlange. Beim Stapel kann ein neues Element nur an das zuletzt Abgelegte angehängt werden; die Entnahme erfolgt nach dem LIFO (Last-In-First-Out) Prinzip (vgl. Abbildung 2-7).

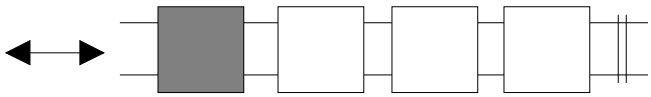


Abb. 2-7: *Stapel*

Bei der Datenstruktur vom Typ der Schlange erfolgt die Entnahme nach dem FIFO (First-In-First-Out) Prinzip, während das Hinzufügen dem Vorgehen beim Stapel folgt (vgl. Abbildung 2-8).

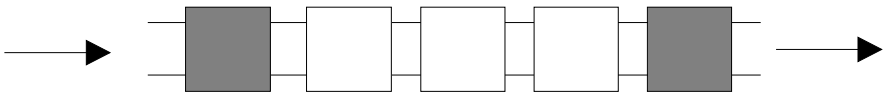


Abb. 2-8: *Schlange*

Stapel sind sehr häufig verwendete Datenstrukturen. Ein Problem liegt in dem unbestimmten Platzbedarf, der den Programmierer zwingt, für eine maximale Anzahl von Elementen Platz zu lassen. Bei zwei Stapeln ist es zweckmäßig, sie sequenziell mit den Zu- und Abgangsfeldern zueinander zu speichern. Einer der Stapel läuft erst dann über, wenn beide Stapel keinen Platz mehr finden (vgl. Abbildung 2-9).

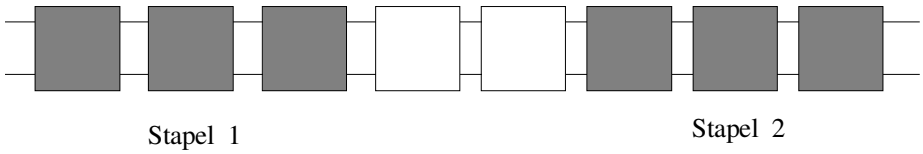


Abb. 2-9: *Zwei Stapel*

Bei einer Schlange ist es zweckmäßig, eine ringförmige Speicherungsform anzuwenden, um ein Wandern der Schlange durch den linearen Speicher zu verhindern (vgl. Abbildung 2-10).

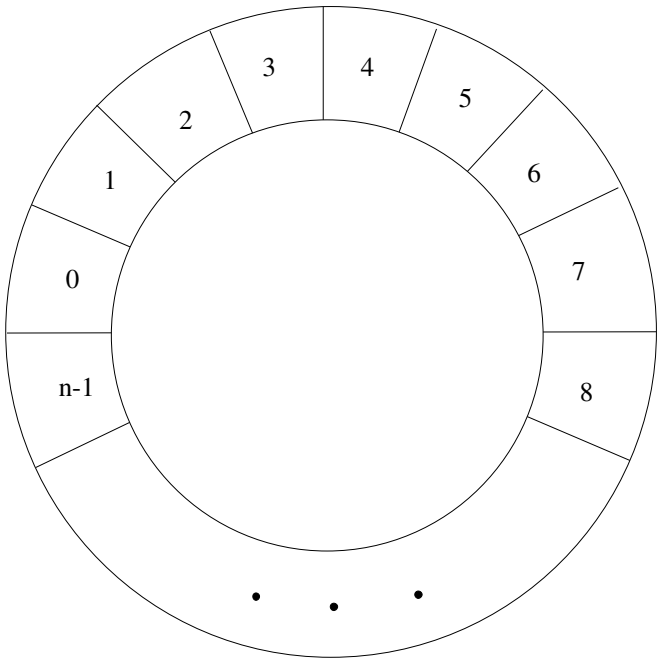


Abb. 2-10: *Schlange mit ringförmiger Speicherungsform*

2.2.2 BÄUME

Ein Baum stellt eine Ordnung auf einer Menge von Objekten her. Beispielsweise beschreibt ein Stammbaum die Menge der Ahnen eines Menschen. Bäume sind spezielle Graphen mit Wurzel-, Blatt- und Zwischenknoten. Ein Knoten ohne Vorgänger heißt *Wurzelknoten*, ein Knoten ohne Nachfolger heißt *Blattknoten*. Existiert jeweils genau eine einfache Kette zwischen je zwei Knoten, so heißt der betreffende Graph *Baum*. Ein gerichteter Baum B ist ein gerichteter Graph mit genau einem Wurzelknoten r und genau einem Pfad zwischen r und jedem Knoten $v \in V$, $v \neq r$ mit r als Anfangsknoten und v als Endknoten des Pfades. Ein gerichteter Baum ist in Abbildung 2-11 dargestellt.

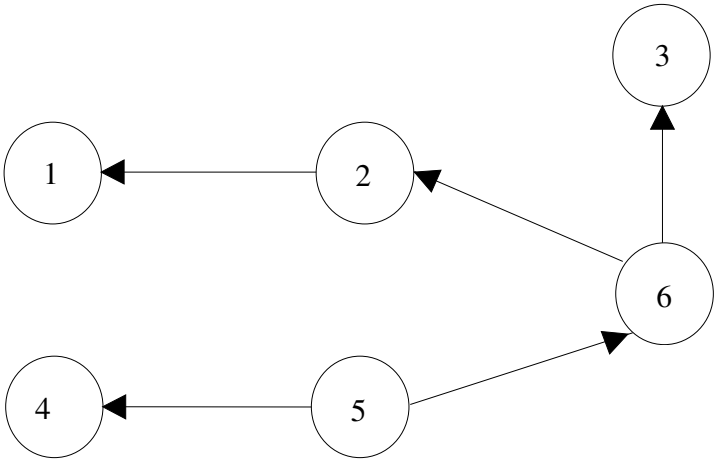


Abb. 2-11: *Gerichteter Baum*

In einem gerichteten Baum nennen wir den direkten Nachfolger i+1 eines Knotens i auch Sohnknoten, bzw. i wird als Vaterknoten von i+1 bezeichnet. In einem gerichteten Baum ist jeder Knoten, der nicht Blattknoten ist, Wurzel eines Unterbaumes. In Abbildung 2-11 ist der Knoten 5 Vaterknoten von 4 und 6; Knoten 2 und 3 sind Sohnknoten von 6. Der Knoten 5 ist Wurzelknoten des Baumes. Der Knoten 6 ist Wurzelknoten des Unterbaumes bestehend aus 1, 2, 3 und 6. Die Knoten 1, 3 und 4 sind Blattknoten.

Baumstrukturen sind eines der wesentlichsten graphischen Modelle in der Datei- und Zugriffsmodellierung. Baumstrukturen werden physisch verkettet implementiert, indem man Zeigerfelder zu den Sohnknoten in den Datensatz einbezieht. Eine der wichtigsten Baumstrukturen sind binäre Bäume.

Ein binärer Baum *B* besteht aus einer endlichen Menge von Knoten *V*, die entweder leer ist oder aus einem Wurzelknoten *r* zusammen mit den Knoten von zwei binären Teilbäumen besteht, die linker und rechter Teilbaum genannt werden (vgl. Abbildung 2-12).

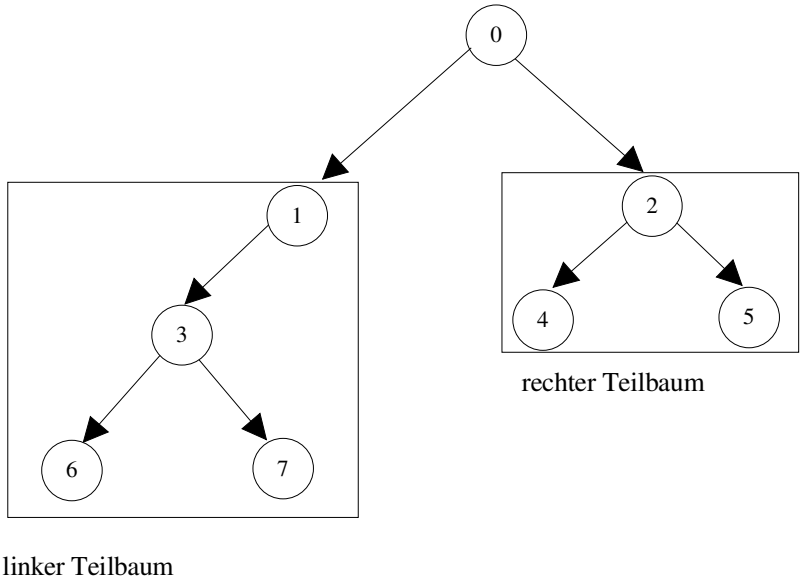


Abb. 2-12: *Binärer Baum*

Ein binärer Baum wird voll besetzt genannt, wenn jeder Knoten entweder Wurzel zweier nicht leerer Teilbäume ist oder Blattknoten ist. Ein voll besetzter Binärbaum mit *k* Ebenen hat genau dann 2^k-1 Knoten, wenn alle Blattknoten auf der *k*-ten Ebene liegen. Die Anzahl der Knoten auf der *k*-ten Ebene beträgt dann 2^{k-1}.

2.2.3 DATEIORGANISATION

Dateien bestehen aus Datensätzen und können als Tabelle interpretiert werden. Die Dateifelder entsprechen dann den Spalteneintragungen je Zeile. Die Feldnamen entsprechen den Spaltennamen. Die Zeilen entsprechen den unterschiedlichen Datensätzen. Typische Dateioperationen sind das Suchen, Ändern, Einfügen und Löschen eines Satzes in einer Datei.

Dateien können index-sequenziell gespeichert werden. Hierbei werden für jede Datei zwei weitere Dateien angelegt: eine Indexdatei und eine Überlaufdatei. Die Indexdatei enthält gewöhnlich nur den Schlüssel des Datensatzes aus der Primärdatei und ein Zeigerfeld, welches seine Speicheradresse angibt. Sucht man nun nach einem Satz mit einem entsprechenden Schlüssel, so genügt es, die Indexdatei zu durchsuchen, die einen wesentlich geringeren Umfang als die Primärdatei aufweist. Da in der Primärdatei die Sätze sequenziell gespeichert sind, ist es zur Vermeidung einer ständigen Reorganisation üblich, eine Überlaufdatei anzulegen. In dieser werden die Sätze, welche in die Primärdatei einzufügen wären, bis zur nächsten Reorganisation gespeichert.

Mitunter werden auch mehrere Indexstufen unterschieden, d.h. es werden Indexdateien über Indexdateien angelegt. Dies ist insbesondere dann der Fall, wenn die Indexdatei nicht alle Schlüssel einer Primärdatei enthält, sondern nur jeden k-ten ($k > 1$). Das Modell derartiger Dateien entspricht einem sogenannten Mehrwegebaum. Die bekannteste Version hiervon ist der sogenannte B-Baum.

Abbildung 2-13 zeigt einen dreistufigen Index eines 3-Wege-Baumes, auf dessen unterster Ebene - dem sogenannten Blattknoten - Zeiger gespeichert sind, die zu den eigentlichen Daten weisen.

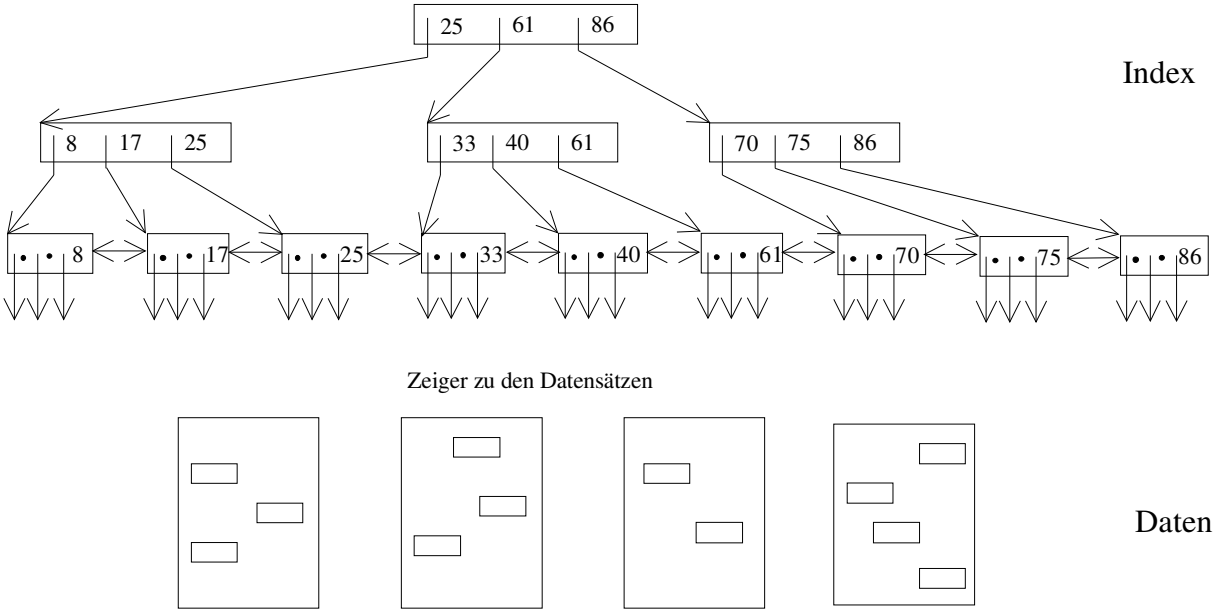


Abb. 2-13: *Beispiel einer Mehrwegebaumorganisation*

2.3 FUNKTIONSBESCHREIBUNGEN

Funktionen benutzen Algorithmen. Unter einem Algorithmus versteht man die exakte Beschreibung einer Verarbeitungsvorschrift, die ohne zusätzliche Erläuterungen von einer Maschine ausgeführt werden kann. Wir wollen den Algorithmusbegriff an einem Beispiel verdeutlichen. Angenommen, die Aufgabe besteht darin, zwei Listen mit reellen Zahlen zu verschmelzen. In beiden Ausgangslisten sind die Zahlen nicht-fallend geordnet. Eine solche Reihung soll auch in der Liste nach der Verschmelzung erreicht werden.

Algorithmus 123 *merge*.

Input: Two sequences of reals, $\boldsymbol{e} = (e[1],\cdots, e[n])$ and $\boldsymbol{f} = (f[1],\cdots, f[m])$, both sorted in non-decreasing order.

Output: Sequence $\boldsymbol{g} = (g[1],\cdots, g[n+m])$ in which all elements are arranged in non-decreasing order.

```
begin
i := 1; j := 1; k := 1;     -- initialization of counters
while (i ≤ n) and (j ≤ m) do
        -- the while loop merges elements of sequences e and f into g;
        -- the loop is executed until all elements of one of the sequences are merged
    begin
    if e[i] < f[y]
    then begin g[k] := e[i]; i := i + 1; end
    else begin g[k] := f[j]; j := j + 1; end;
    k := k + 1;
    end;
if i ≤ n        -- not all elements of sequence e have been merged
then for l := i to n do g[k + l – i] := e[l]
else
    if j ≤ m        -- not all elements of sequence f have been merged
    then for l := j to m do g[k + l - j] := f[l];
end;
```

Der Algorithmus ermittelt die nicht-fallende Reihung $\boldsymbol{g}$ aller Elemente von $\boldsymbol{e}$ und $\boldsymbol{f}$.

Ein Algorithmus besteht aus zwei Teilen: einen Kopf und eine Methode. Der Kopf beginnt mit dem Schlüsselwort *Algorithmus*, gefolgt von einer identifizierenden Zahl, einem optionalen Deskriptor

und, falls existent, eine Referenz auf die oder den Urheber des Algorithmus. Input- und Outputparameter werden in den beiden Feldern *Input* (*Instance*) und *Output* (*Answer*) vereinbart. Die *Methode* ist eine Kollektion von mehreren Anweisungen. Ein Block von mehreren Anweisungen ist eingebettet durch **`begin`** und **`end`**. Eine Anweisung kann wiederum ein Block, eine Zuweisung, eine Else- oder Case-Operation, eine Schleife (**`for`**, **`while`**, **`repeat ··· until`**, oder ein allgemeiner **`loop`**), ein **`call`** eines anderen Algorithmus, ein Abbruch einer Schleife (**`exit loop`**, etc.), etc. sein. Case-Anweisungen zerlegen die Ausführung eines Algorithmus in mehrere Äste, abhängig vom Wert der Kontrollvariablen. Loop-Anweisungen können solche Formulierungen enthalten, wie "**`for all`** $a \in M$ **`do ···`**", oder "**`while`** $M \neq \emptyset$ **`do ···`**". Wenn die Ausführung einer Schleife durch eine Exit-Anweisung unterbrochen wird, springt der Algorithmus zur ersten Anweisung nach der Schleife. Kommentare werden auf jeder Zeile durch zwei Minuszeichen (--) eingeleitet und durch das Ende der Zeile (EOL) abgeschlossen.

Zur Darstellung von Funktionsbeschreibungen gibt es viele Möglichkeiten. Weit verbreitet sind Programmablaufpläne (PAP) und Struktogramme. Sie dienen der Darstellung des Ablaufs (Kontrollstruktur) eines Programms (Algorithmen). Die wichtigsten Kontrollstrukturen sind:

- Sequenz,
- Alternative,
- Wiederholung,
- Rekursion und
- Parallelität.

Die Symbole von Programmablaufplänen nach DIN 66001 sind in Abbildung 2-14 dargestellt.

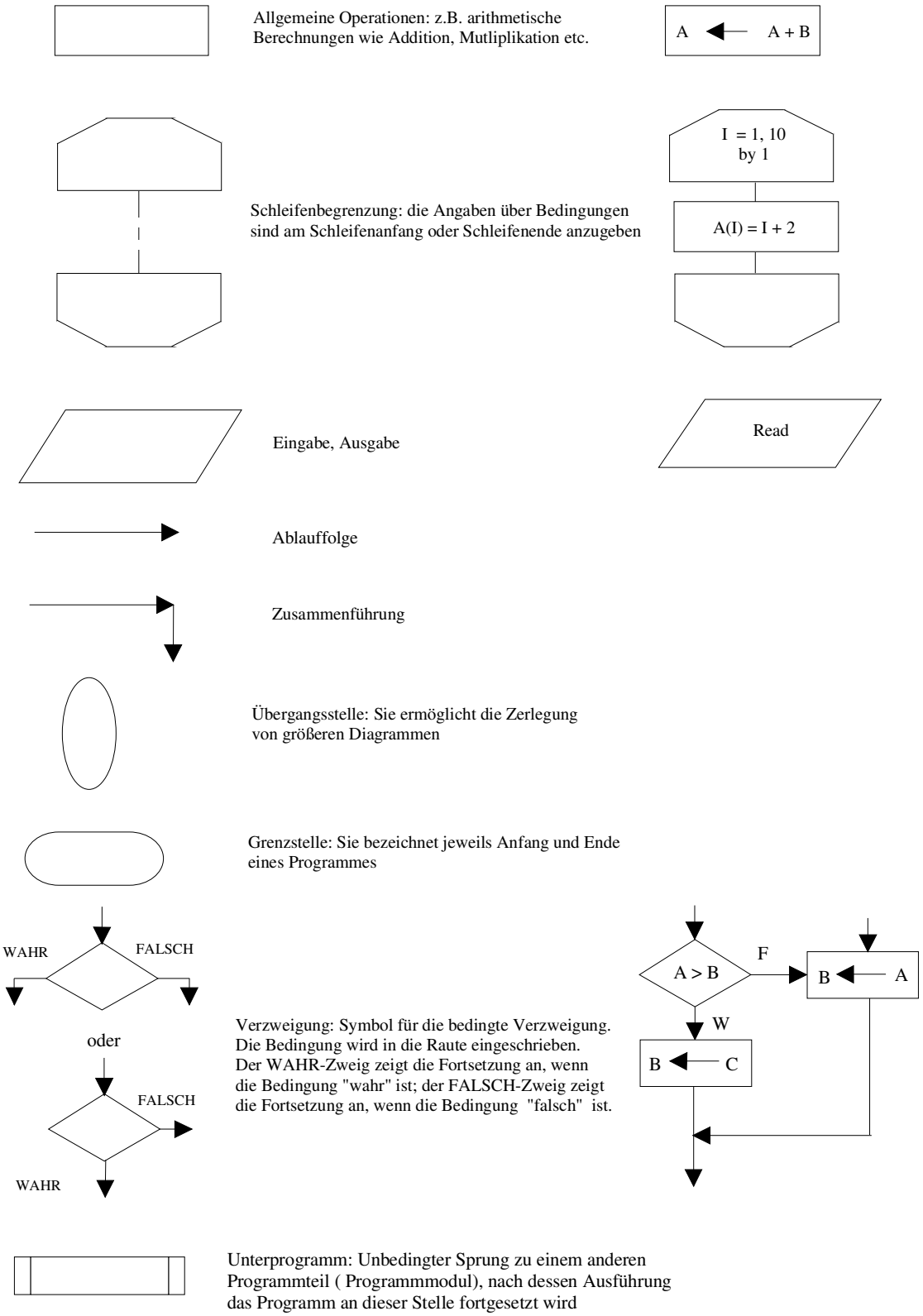


Abb. 2-14: *Symbole für Programmablaufpläne*

PAP und Struktogramme werden jedoch nicht nur zur Darstellung von Algorithmen verwendet, die für Rechner geschrieben werden, sondern dienen in allen Bereichen des menschlichen Lebens zur Darstellung von Abläufen. Zur Veranschaulichung der Verwendung von Sprachelementen dient Abbildung 2-15.

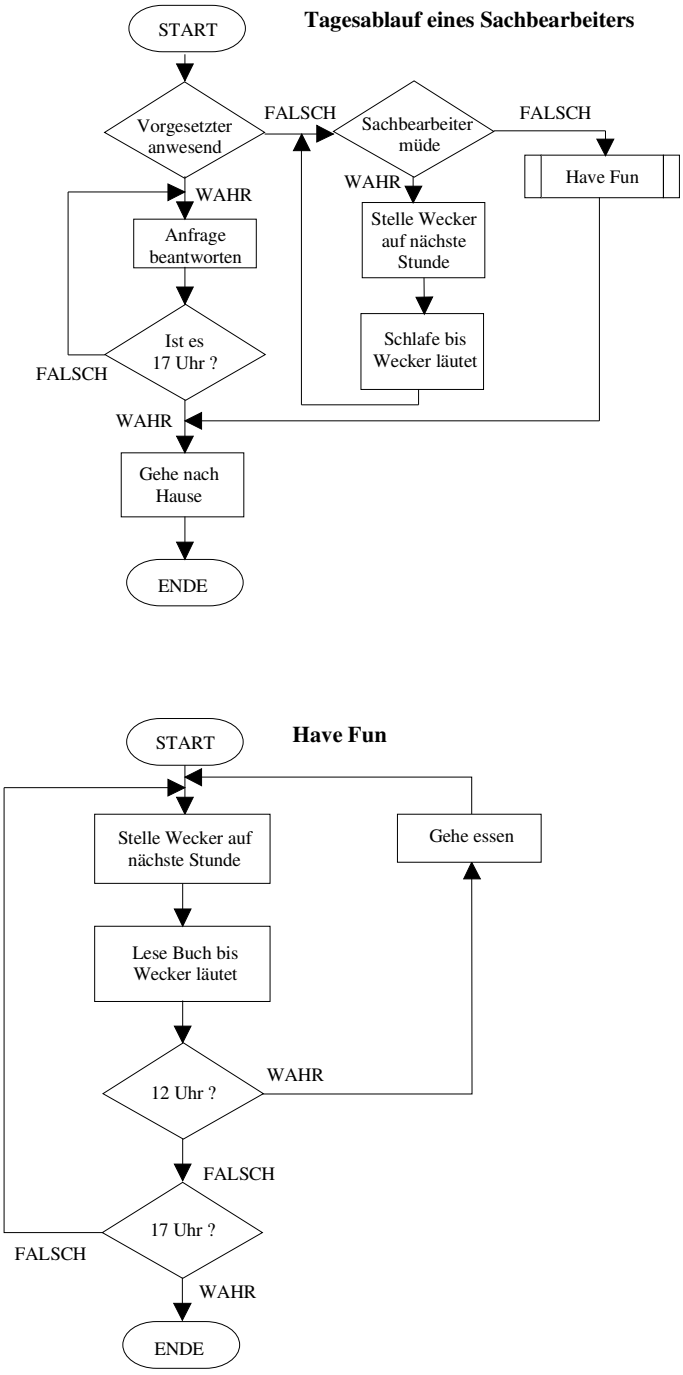


Abb. 2-15: *Beispiel für einen Programmablaufplan*

Programmablaufpläne führen oft zu unübersichtlichen und unnötig langen Darstellungen. Struktogramme versuchen diese Nachteile zu vermeiden. Ein Struktogramm greift auf die folgenden Elemente zurück.

Das Anweisungsdiagramm dient zur Darstellung eines Verarbeitungsschrittes, zum Beispiel einer Wertzuweisung, einer Ein-, bzw. Ausgabeanweisung oder eines Unterprogramm-Aufrufes (vgl. Abbildung 2-16).

Abb. 2-16: *Anweisungsdiagramm*

Das Entscheidungsdiagramm wird zur Repräsentation der bedingten Verzweigung verwendet. Das zentrale Dreieck enthält die Bedingung, die Dreiecke rechts und links enthalten die möglichen Wahrheitswerte Wahr und Falsch. Abhängig von der Antwort wird Anweisung 1 oder Anweisung 2 ausgeführt (vgl. Abbildung 2-17).

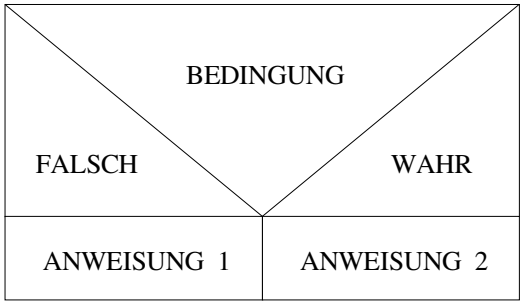


Abb. 2-17: *Entscheidungsdiagramm*

Das While-Do-Diagramm dient als Wiederholungssymbol, bei dem die Ausführung der Anweisung(en) von der einleitenden Bedingung abhängig gemacht wird. Ist die Bedingung nicht erfüllt, so

wird (werden) die Anweisung(en) nicht (mehr) ausgeführt. (vgl. Abbildung 2-18).

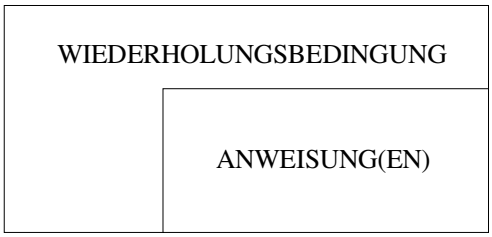


Abb. 2-18: *While-Do-Diagramm*

Das Beginn-Ende-Diagramm dient zur Hervorhebung zusammengehöriger Anweisungsteile, so genannter Blöcke (vgl. Abbildung 2-19).

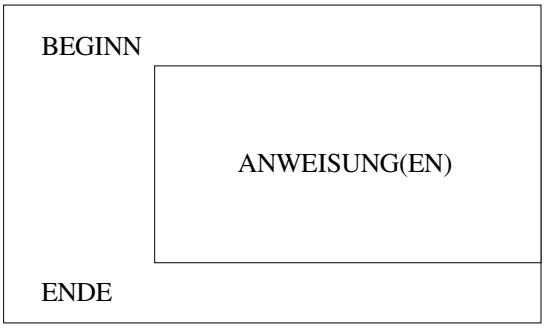


Abb. 2-19: *Beginn-Ende-Diagramm*

Das Repeat-Until-Diagramm dient als Wiederholungssymbol, bei dem die Durchführung der Anweisung(en) von einer der Endbedingungen abhängt (vgl. Abbildung 2-20).

Abb. 2-20: *Repeat-Until-Diagramm*

Das Case-Diagramm dient als Entscheidungsdiagramm für Mehrfachverzweigungen, wobei die Verzweigung vom Wert eines Ausdruckes abhängig gemacht wird. Für das in Abbildung 2-21 dargestellte Beispiel gilt der folgende Zusammenhang. Ist der Wert des Ausdrucks *i*, so wird die Anweisung A_i ausgeführt, sonst die Anweisung A_{n+1}.

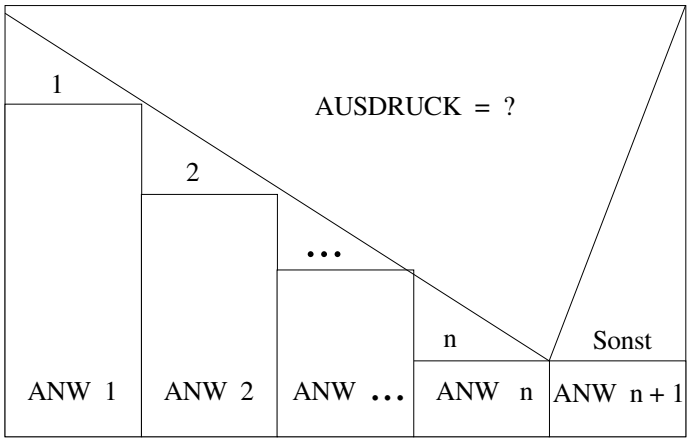


Abb. 2-21: *Case-Diagramm*

Das Beispiel, das für die Erläuterung des Programmablaufplans diente, wird auch für die Erläuterung der Anwendung eines Struktogramms benutzt (vgl. Abbildung 2-22).

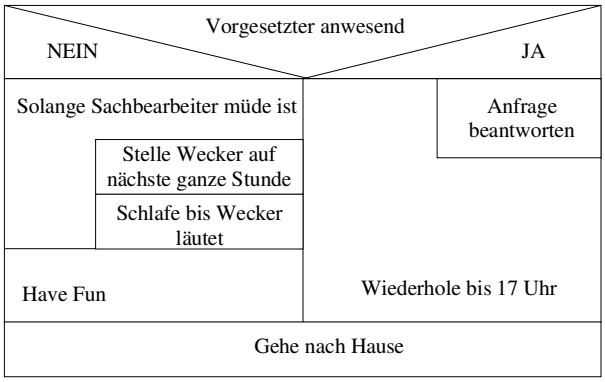


Abb. 2-22a: *Beispiel für ein Struktogramm*

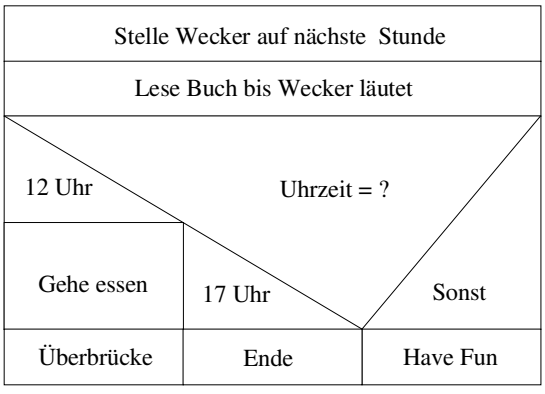


Abb. 2-22b: *Beispiel für ein Struktogramm*

3 INFORMATIONSTECHNOLOGIE

Wesentliche Bestandteile der Informationstechnologie sind Rechner, Betriebssysteme, Kommunikationseinrichtungen und Programmiersprachen. In diesem Abschnitt werden Modelle dieser Basistechnologien besprochen.

3.1 RECHNER

Grundlegend für das Verständnis der meisten Rechner ist die von-Neumann-Architektur, wie sie in Abbildung 3-1 dargestellt ist.

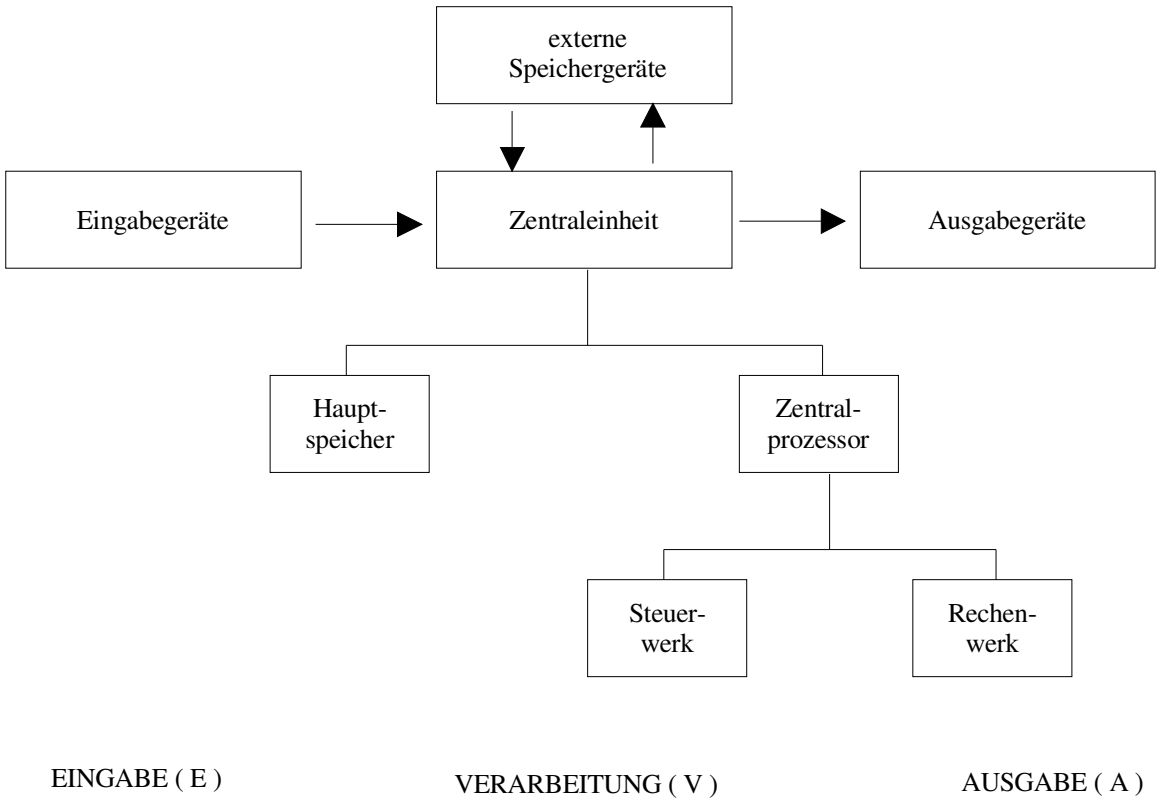


Abb. 3-1: *von-Neumann-Architektur*

Die Abarbeitung eines Programms durch einen Menschen unter Benutzung einfacher Hilfsmittel auf Basis der von-Neumann-Architektur kann man sich wie folgt vorstellen (vgl. Abbildung 3-2).

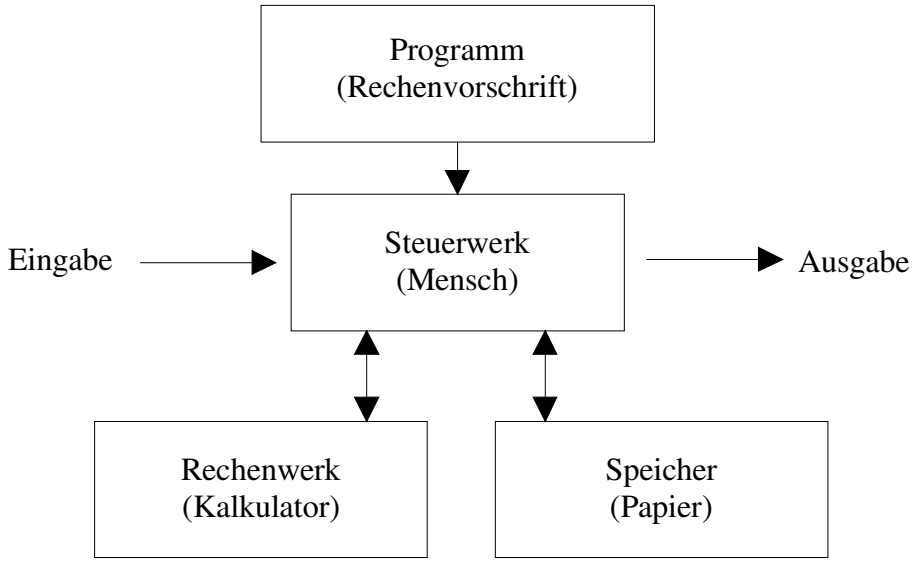


Abb. 3-2: *Menschliche Programm-Abarbeitung*

Rechner, die der von-Neumann-Architektur folgen, findet man in allen Klassen. Beispiele sind Embedded Systems, Palmtops, Notebooks, Laptops, PCs, Workstations und Mainframes (Großrechner).

3.1.1 EINFACHE BASISMASCHINE

Die einfache Basismaschine (BM) soll dazu dienen, für eine Anzahl von Funktionen F_i mit Argument x den Funktionswert $y_i = F_i(x)$ zu berechnen (vgl. Abbildung 3-3). Wir wollen annehmen, dass x und y_i ganze Zahlen sind.

Die Komponenten von BM seien:

- der Eingabeteil (ET) für die Bedienung der Maschine, bestehend aus

  einem Tastenfeld für die Eingabe von $x \in \{0, 1, ..., 9\}$ und

  einem Tastenfeld für die Auswahl der Funktion F_i, i=1,...,N.

- der Operationsteil (OT) für die Berechnung von F_i, bestehend aus

  einer Speicherzelle AS (Argumentwertspeicher) für x,

  einer Speicherzelle FS (Funktionswertspeicher) für y_i und

  einer Anzahl N von Rechenwerken zur Berechnung von F_i.

- der Ausgabeteil (AT) für die Darstellung der Speicherinhalte

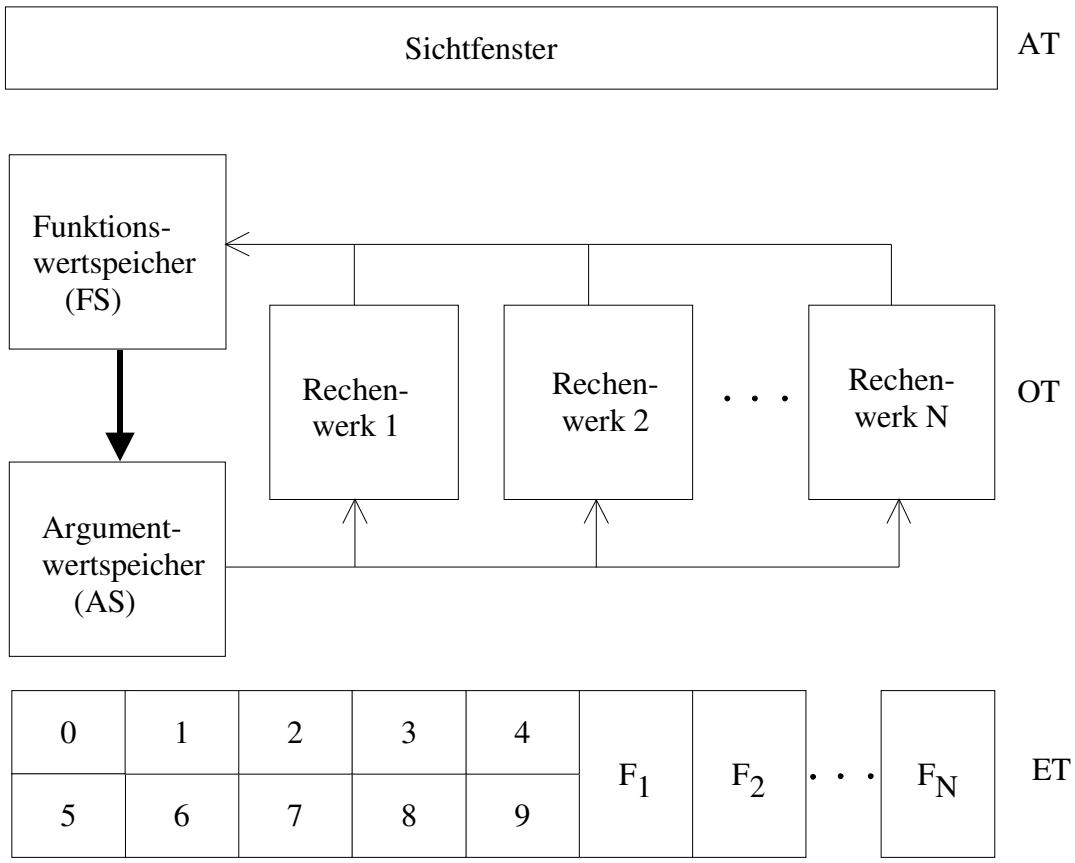


Abb. 3-3: *BM zur Berechnung mehrerer Funktionen*

Die Bedienung der BM besteht aus einer zyklischen Abfolge der in Abbildung 3-4 dargestellten Schritte. Jeder Eingabeschritt löst eine Operation und einen Ausgabeschritt aus.

Eingabe	Verarbeitung	Ausgabe
Eingeben von x über ET	Ablegen von x in AS	Anzeigen des Inhalts von AS über AT
Auswählen von F_i über ET	Berechnen von y_i und Ablegen von y_i in FS	Anzeigen des Inhalts von FS über AT

Abb. 3-4: *Funktionsweise der BM*

Für Funktionen mit zwei oder mehr Argumenten x_1 , x_2 , ... wird AS zu einer Folge von Speicherzellen (AS_1 , AS_2 , ...) erweitert. Ebenso ist es möglich, dass ein Funktionswert y_i zum Argument x einer nachfolgenden Funktionsberechnung wird. BM ist ein einfaches Modell zur Darstellung eines Rechners. Ein Beispiel ist ein nicht programmierbarer Taschenrechner ohne Speicher.

3.1.2 PROGRAMMGESTEUERTE MASCHINE

Für die Darstellung weiterer Strukturmerkmale von Rechnersystemen wird BM zu einer programmgesteuerten Maschine (PBM) erweitert. Bedienung und Verhalten von BM und PBM stimmen überein. Der Unterschied liegt in der Detaillierung des Operationsteils OT.

Die Realisierung von *N* Rechenwerken zur Berechnung von *N* Funktionen ist unwirtschaftlich, wenn mehrere der *N* Rechenwerke gleichartige Schritte ausführen. So könnte z.B. eine

Komponente für die Addition zur Berechnung mehrerer Funktionen verwendet werden. Solche Komponenten sollten nur einmal bereitgestellt werden und für alle Funktionen nutzbar sein.

Bei PBM wird daher die Berechnung der *N* Funktionen in *M* elementare Schritte zerlegt. Jeder elementare Schritt wird als Rechenwerk realisiert. Die Berechnung einer Funktion besteht nun in der Berechnung einer zugehörigen Folge von elementaren Schritten. Hierzu benötigt PBM im Vergleich zu BM folgende zusätzliche Komponenten (vgl. Abbildung 3-5):

- Einen Programmspeicher PS, der für jede Funktion F_i die zugehörige Folge elementarer Schritte enthält. Ein Schritt heißt Befehl und identifiziert ein Rechenwerk. Die Folge elementarer Schritte ist eine Befehlsfolge, die als Programm bezeichnet wird.

- Ein Steuerwerk SW, das bei der Berechnung einer Funktion das zugehörige Programm Befehl für Befehl abarbeitet und dabei die zu den einzelnen Befehlen gehörenden Rechenwerke aufruft.

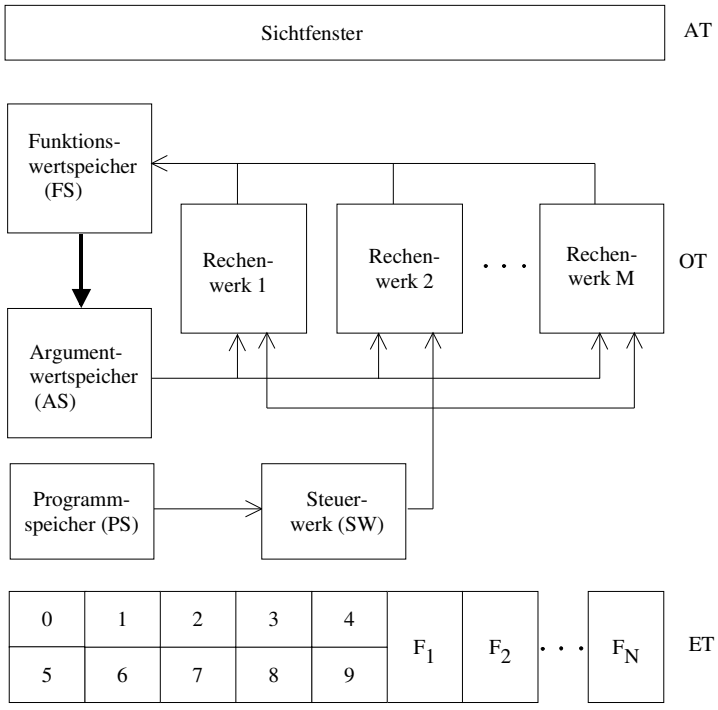


Abb. 3-5: *PBM zur Berechnung mehrerer Funktionen*

Beispiele für PBM sind Taschenrechner, deren Funktionen über "fest verdrahtete", von außen nicht modifizierbare Programme realisiert sind.

3.1.3 UNIVERSALRECHENMASCHINE

Eine PBM lässt sich zu einer Universalrechenmaschine (URM) verallgemeinern (vgl. Abbildung 3-6). Die Konzeption von URM stellt das seit Jahrzehnten dominierende Modell für Rechner dar.

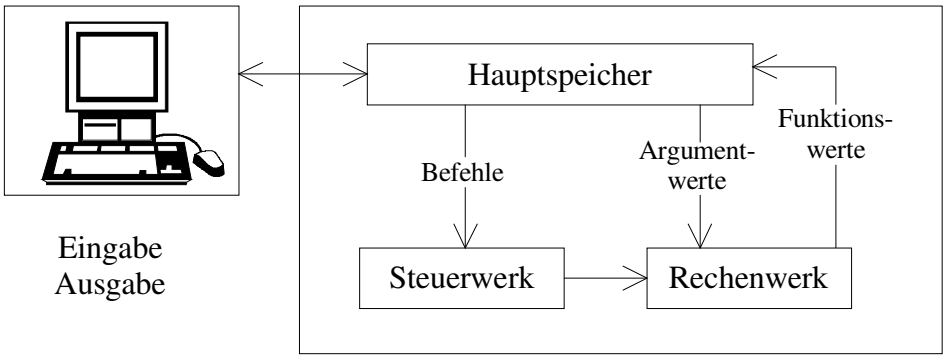


Abb. 3-6: *URM: Universalrechenmaschine*

URM unterscheidet sich von PBM in folgenden Eigenschaften:

- URM ist frei programmierbar, d.h. es können jederzeit Programme im Programmspeicher PS hinzugefügt oder gelöscht werden. URM kann somit alle Funktionen berechnen, die sich aus elementaren Schritten, für die URM Rechenwerke besitzt, berechnen lassen und als Programme verfügbar sind.
- Argumentwertspeicher AS und Funktionswertspeicher FS werden zum Operandenspeicher OS zusammengefasst.

- Programmspeicher PS und Operandenspeicher OS werden zu einem allgemeinen Hauptspeicher HS zusammengefasst. Die Bereiche von OS und PS sind innerhalb von HS variabel. Dadurch wird eine flexible Nutzung des Hauptspeichers möglich.
- ET und AT werden mit Einrichtungen für die Mensch-Maschine-Kommunikation erweitert. Zu den wichtigsten Einrichtungen gehören Bildschirm, Tastatur, Maus und Drucker. ET und AT können grundsätzlich mit jeder beliebigen Stelle von HS kommunizieren.

Die Komponenten des Operationsteils von URM werden im Folgenden genauer betrachtet.

URM-RECHENWERK

Das URM-Rechenwerk fasst alle (elementaren) Rechenwerke zusammen. Der Befehlsvorrat des URM-Rechenwerks umfasst im wesentlichen Befehle für

- den Transport von Operanden innerhalb des Hauptspeichers,
- den Transport von Operanden zwischen Hauptspeicher und Ein-/Ausgabegeräten,
- arithmetische Operationen (z.B. +, -, *, /),
- logische Operationen (z.B., $\wedge$, $\vee$,$\neg$) und
- Vergleichsoperationen (z.B. $<$, $\leq$, $=$, $\geq$, $>$, $\neq$).

Die Leistung eines URM-Rechenwerkes wird in MIPS (million instructions per second) und FLOPS (floating point operations per second) gemessen; *instruction* und *operation* sind dabei mit Befehl gleichzusetzen. Während MIPS auf den allgemeinen Befehlsvorrat eines Rechners Bezug nimmt, gibt FLOPS speziell Auskunft über die Berechnungszeiten von arithmetischen Operationen mit Gleitpunktzahlen.

URM-STEUERWERK

Bei PBM wurde davon ausgegangen, dass ein Programm eine Befehlsfolge darstellt, die bei jeder Programmausführung linear abgearbeitet wird. Im Gegensatz dazu kann bei URM von dieser linearen Folge abgewichen werden. Dabei ist es möglich, dass der Nachfolger-Befehl abhängig vom Ergebnis des Vorgänger-Befehls ist. Die jeweilige Befehlsfolge wird erst während der Ausführung des Programms, d.h. zu seiner Laufzeit, bestimmt.

Hilfsmittel zur Beschreibung nicht linearer Befehlsfolgen sind spezielle Steuerbefehle, die nicht das URM-Rechenwerk, sondern das URM-Steuerwerk ausführt. Ein Programm besteht somit aus Rechenwerksbefehlen und Steuerbefehlen. Die beiden wichtigsten Grundformen von Steuerbefehlen sind:

- Springe zu Befehl *n* (unbedingte Anweisung) und
- Springe zu Befehl *n*, falls der Funktionswert des zuletzt ausgeführten Befehls den Wert *x* hat (bedingte Anweisung).

In Abbildung 3-7 ist der Befehlsvorrat einer Beispiel-URM angegeben.
AS, FS und VS (Vergleichsspeicher) sind hier Speicherzellen im HS.

Rechenwerksbefehle	
L	Lies ein Argument über ET und trage dieses in AS ein
Ü	Übertrage den Inhalt von AS nach FS
V	Vergleiche die Inhalte von AS und FS und trage in VS eine der Aussagen 'AS $\leq$ FS' oder 'AS > FS' ein
A	Zeige den Inhalt von FS über AT an
Steuerbefehl	
Sn	Springe zu Befehl n, falls in VS die Aussage 'AS $\leq$ FS' steht

Abb. 3-7: *Befehlsvorrat einer Beispiel-URM*

Es ist ein Programm für die Berechnung der Funktion $z = \max(x,y)$ zu erstellen (x, y und z sind ganze Zahlen). Dazu steht eine URM mit dem in Abbildung 3-7 gezeigten Befehlsvorrat zur Verfügung. Die Funktion z wird z.B. durch folgendes Programm berechnet (vgl. Abbildung 3-8):

Nr.	Befehl	Kommentar
1	L	x steht in AS
2	Ü	x steht in AS und FS
3	L	y steht in AS
4	V	Vergleiche AS und FS
5	S7	Springe zu Befehl 7, falls in VS 'AS ≤ FS' steht
6	Ü	y ist größer als x und wird daher nach FS übertragen
7	A	Ausgeben des Maximums aus x und y, das in FS steht

Abb. 3-8: *Programm der Beispiel-URM*

URM-HAUPTSPEICHER

Der Hauptspeicher von URM besteht aus Speicherzellen gleicher Größe, die je nach Bedarf Befehle oder deren Operanden aufnehmen können. Zur Speicherung eines Befehls bzw. Operanden werden häufig mehrere Speicherzellen benötigt. Die Position einer Speicherzelle im Hauptspeicher heißt Adresse. Die Adressen werden fortlaufend nummeriert, beginnend bei Adresse 0. Der Bereich von Adresse 0 bis zur höchsten Adresse heißt Adressraum des Hauptspeichers. Die Adresse eines mehrere Speicherzellen belegenden Befehls bzw. Operanden ist die Adresse der ersten belegten Speicherzelle.

Eine Standardspeicherzelle kann ein Bit speichern, d.h. 0 oder 1. Ein Speicher besteht aus mehreren Standardspeicherzellen. Die Kapazität des Hauptspeichers wird in Byte (B) angegeben. Analog zum metrischen System sind hier folgende Faktoren üblich:

- K = 1024 $= 2^{10}$ ("Kilo")
- M = 1024K $= 2^{20}$ ("Mega")
- G = 1024M $= 2^{30}$ ("Giga")
- T = 1024G $= 2^{40}$ ("Tera")

ADRESSIERUNG

Die Befehle, aus denen sich die Programme von URM zusammensetzen, besitzen folgenden Aufbau (vgl. auch Abbildung 3-9).

Jeder Befehl besteht aus einem Operationscode (OpCode), gefolgt von null bis mehreren Operanden bzw. Adressen von Operanden. Der Operationscode benennt das aufzurufende Rechenwerk. Die Operanden sind die Daten, mit denen die Operation durchzuführen ist. Ein Operand ist entweder Bestandteil des Befehls oder im Befehl ist die Adresse der Speicherzelle angegeben, welche den Operanden enthält.

Die Technik zur Benennung der Speicherzellen, welche die Operanden für eine Befehlsdurchführung beinhalten, wird als Adressierung bezeichnet. Folgende Adressierungsformen finden Verwendung (vgl. Abbildung 3-9):

- Direkte Adressierung: Der Operand ist Bestandteil des Befehls.
- Absolute Adressierung: Der Befehl enthält die Adresse des Operanden in Form einer absoluten Speicheradresse.
- Relative Adressierung: Der Befehl enthält die Adresse des Operanden in Form eines Tupels von Adressen: Adresse = (Basisadresse, Distanzadresse).

Die absolute Adresse des Operanden berechnet sich aus der Summe von Basisadresse und Dis-

tanzadresse. Die Basisadresse wird häufig in einer speziellen Speicherzelle mit einer Kurzbezeichnung, dem Adressregister, gespeichert.

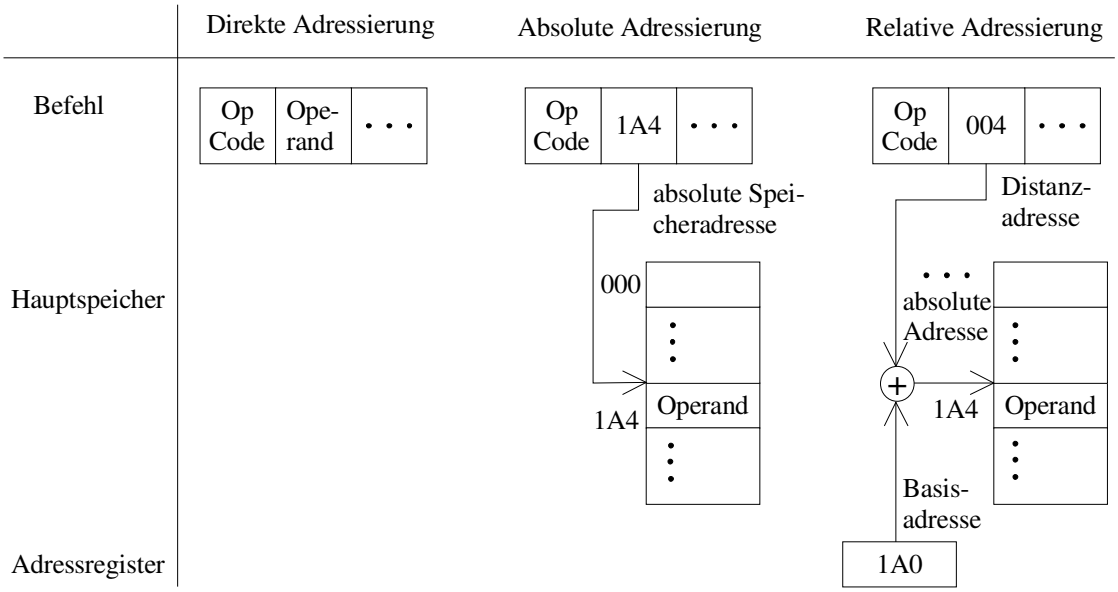


Abb. 3-9: *Adressierungsformen*

Die direkte Adressierung eignet sich für kurze Operanden mit unveränderlichem Wert, z.B. für Konstanten in Form eines einzelnen Zeichens. Die absolute Adressierung wird überwiegend im hardware- und betriebssystemnahen Bereich angewandt. Die wichtigste Adressierungsform im Bereich der Anwendungsprogrammierung ist die relative Adressierung.

Unter Verwendung des in Abbildung 3-7 eingeführten, fiktiven Befehlssatzes können u.a. folgende Befehle gebildet werden.

A 'A'	Gib das Zeichen 'A' über AT aus (direkte Adressierung)
L 700	Lies über ET ein Zeichen ein und speichere es unter der Adresse 700 (absolute Adressierung)

Ü 'A' 700	Übertrage das Zeichen 'A' an die Adresse 700 (direkte und absolute Adressierung)
L (A,200)	Lies über den Eingabeteil ein Zeichen ein und speichere es unter der Adresse (Inhalt des Adressregisters A + 200) (relative Adressierung)

3.1.4 BUSRECHNERSYSTEM (BRS)

Die Strukturierung des Operationsteils von URM in Hauptspeicher, Steuerwerk und Rechenwerk stellt eine funktionale Sicht dar. Das folgende Modell, das Busrechnersystem (BRS), berücksichtigt darüber hinaus die technischen Komponenten, die zur Realisierung dieser Funktionen von URM notwendig sind.

Wie Abbildung 3-10 zeigt, sind den Funktionen Steuerwerk, Rechenwerk und Hauptspeicher spezielle Hardware-Komponenten zugeordnet. Die Komponenten Steuerwerk und Rechenwerk bilden gemeinsam den Prozessor. Prozessoren werden als ein Hardware-Baustein realisiert. Weitere Hardware-Bausteine dienen der Realisierung des Hauptspeichers.

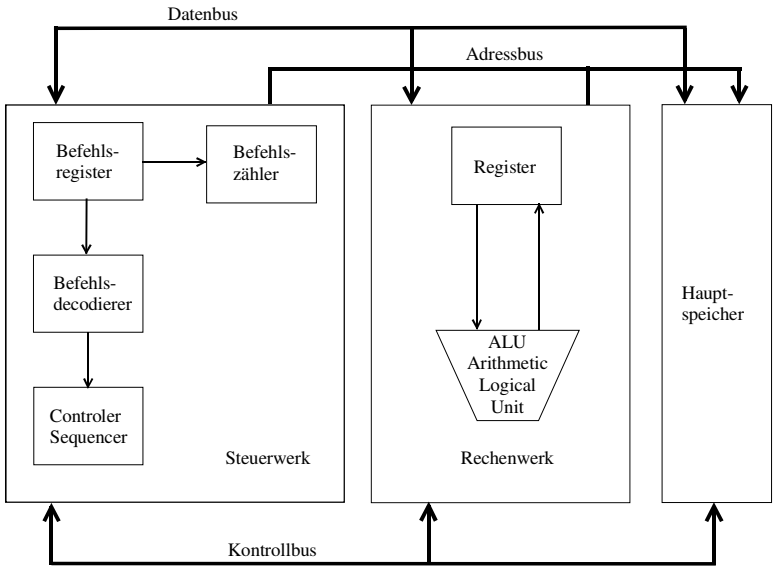


Abb. 3-10: *BRS: Busrechnersystem*

BRS ist heute das dominierende Konstruktionsprinzip für moderne Hardware-Architekturen. Erst die Verwendung standardisierter, in großen Stückzahlen herstellbarer Hardware-Bausteine erlaubt eine wirtschaftliche Konstruktion von Rechnern mit hoher Variantenvielfalt.
Kennzeichnend für dieses Konstruktionsprinzip und namensgebend für BRS sind Leitungsverbindungen, die als Bus bezeichnet werden und die Kommunikation zwischen den einzelnen Hardware-Bausteinen ermöglichen. Ein Bus wird von den beteiligten Hardware-Bausteinen gemeinsam genutzt. Er steht den jeweils kommunizierenden Hardware-Komponenten zeitabschnittsweise exklusiv zur Verfügung. Folgende Bus-Arten werden unterschieden:

- Datenbus: auf dem Datenbus werden Befehle vom Hauptspeicher zum Steuerwerk, sowie Daten zwischen Rechenwerk und Hauptspeicher in beiden Richtungen transportiert. Die Anzahl der parallelen Leitungen des Datenbusses heißt Breite des Datenbusses. Sie ist eines der kennzeichnenden Attribute für Prozessoren.

- Adressbus: der Adressbus überträgt Adressen vom Steuerwerk oder vom Rechenwerk zum Hauptspeicher, um dort eine gewünschte Speicherzelle zu identifizieren. Eine vom Steuerwerk kommende Adresse benennt den nächsten zu lesenden Befehl, vom Rechenwerk kommende Adressen benennen die Operanden des gerade ablaufenden Befehls.

- Kontrollbus: die Leitungen des Kontrollbusses werden zur Synchronisation der beteiligten Hardware-Bausteine verwendet. Es werden u.a. Signale für Anforderungsmitteilungen sowie Bereitschafts- und Fertigmeldungen übertragen.

BRS-RECHENWERK

Das BRS-Rechenwerk besteht aus der ALU (Arithmetic Logical Unit), die alle Rechenwerke umfasst, sowie aus einer Menge von speziellen Speicherzellen, die Register genannt werden. Im

Gegensatz zu Hauptspeicherzellen

- sind Register Bestandteil des Prozessors,
- werden Register durch Kurzbezeichnungen benannt (z.B. AX, BX),
- sind die Zugriffszeiten der ALU auf Register wesentlich kürzer als auf Hauptspeicherzellen.

Mit Hilfe spezieller ALU-Befehle können Operanden zwischen Hauptspeicher und Registern transportiert werden. Die Operanden von Befehlen können in Registern oder im Hauptspeicher gespeichert sein.

BRS-STEUERWERK

Das BRS-Steuerwerk ist in die Komponenten Befehlsregister, Befehlszähler, Befehlsdecodierer und Controler / Sequencer unterteilt. Befehlsregister und Befehlszähler sind wiederum spezielle Speicherzellen. Die Programmdurchführung durch das Steuerwerk besteht aus einem zyklischen Durchlauf der nachstehenden Schritte, was als Zwei-Phasen-Konzept der Befehlsverarbeitung bezeichnet wird:

(1) Fetch-Phase:
   (1a) Lesen des nächsten auszuführenden Befehls im Hauptspeicher und Übertragen in das Befehlsregister. Die Adresse des Befehls steht im Befehlszähler.
   (1b) Entschlüsseln des Operationscodes des Befehls durch den Befehlsdecodierer.
   (1c) Falls der Befehl ein Steuerbefehl ist, wird der Befehlszähler entsprechend der angegebenen Adresse gesetzt. Anderenfalls wird der Inhalt des Befehlszählers um die Befehlslänge erhöht: Er enthält nun die Adresse des sequenziell folgenden Befehls.

(2) Execution-Phase: Falls der Befehl ein Rechenwerksbefehl ist, wird die Kontrolle an den Controler / Sequencer abgegeben, der die Befehlsdurchführung in der ALU steuert.

Bei modernen Prozessoren wird der Hauptteil der Programmausführungszeit für die Speicherzugriffe benötigt. Der Zeitbedarf für die eigentliche Befehlsdurchführung im jeweiligen Rechenwerk ist dagegen vernachlässigbar. Alle Befehle und Operanden müssen über den Datenbus "gepumpt" werden. Diese Tatsache wird als von-Neumann-Flaschenhals bezeichnet und stellt eine der wesentlichen Begrenzungen dieser Architektur dar.

Eine Reihe von Architekturmerkmalen moderner Prozessoren dienen der Milderung des von-Neumann-Flaschenhalses. Hierzu gehören insbesondere:

- Cache-Speicher: die mit hoher Wahrscheinlichkeit als nächste benötigten Befehle und Daten werden auf Vorrat aus dem Hauptspeicher in einen schnellen Cache-Speicher geladen,
- Pipelining: überlappendes Ausführen der Fetch- und Execution-Phase. Parallel zur Ausführung von Befehl i wird Befehl (i+1) bereits gelesen und decodiert. Um höhere Überlappung zu erzielen, lassen sich die beiden Phasen noch weiter unterteilen,
- Sprungvorhersage: Unterstützung der Aufrechterhaltung des Befehlsflusses bei Sprungbefehlen.

CISC- UND RISC-ARCHITEKTUR

Die in den Rechnermodellen URM und BRS unterstellten Prozessoren werden üblicherweise als CISC-Prozessoren (Complex Instruction Set Computer) konstruiert. Ein typischer CISC-Prozessor verfügt über mehrere hundert Befehle, die mit einer Reihe von Adressierungsarten kombinierbar sind. Die Komplexität des Befehlssatzes erschwert es, die einzelnen Befehle im Rechenwerk durch Hardware-Schaltungen zu realisieren. Befehle werden vielmehr durch so genannte Mikroprogramme gebildet, die auf elementare Hardware-Komponenten zurückgreifen und deren Ausführungszeit wesentlich langsamer ist als die von starr geschalteten Rechenwerken. In Verbindung mit dem von-Neumann-Flaschenhals stellt diese Realisierungsform eine der wesentlichen Leistungsgrenzen von CISC-Prozessoren dar.

Den geschilderten Nachteilen wird seit Mitte der 70er Jahre durch die Entwicklung von RISC-Prozessoren (Reduced Instruction Set Computer) begegnet. Sie sind insbesondere durch folgende Merkmale charakterisiert:

- Geringe Anzahl von Befehlen und Adressierungsarten,
- Hauptspeicherzugriffe erfolgen durch spezielle Load- und Store-Befehle, alle anderen Befehle erwarten ihre Operanden in Registern,
- alle Befehle besitzen den gleichen Aufbau (Befehlsformat),
- Rechenwerk und Steuerwerk sind im Prozessor fest verdrahtet, d.h. nicht durch Mikroprogramme realisiert.

Diese Merkmale gestatten eine Reihe von Verbesserungen im Hardware-Aufbau von RISC-Prozessoren. Obwohl Programme für RISC-Prozessoren in der Regel mehr Befehle umfassen als Programme für CISC-Prozessoren, werden hierdurch höhere Rechengeschwindigkeiten erzielt.

PARALLELRECHNER

Neben von-Neumann-Rechnern gibt es auch Parallelrechnersysteme (PRS). Diese sind dadurch gekennzeichnet, dass in einem Rechner mehrere Prozessoren zeitlich parallel Programme, Programmteile oder Programmschritte ausführen. Die Prozessoren sind 'eng gekoppelt', d.h. dass mehrere Prozessoren

- auf einen gemeinsamen Speicher zugreifen und / oder
- durch ein gemeinsames Betriebssystem verwaltet werden oder
- ein gemeinsames Steuerwerk haben oder
- Informationen zur Koordination der Prozessoren austauschen.

Zur Klassifikation der unterschiedlichen Varianten von PRS wird das Schema von Flynn verwendet, welches folgende, voneinander weit gehend unabhängige Kategorien verwendet:

- Bearbeitet ein Rechner gleichzeitig unterschiedliche Befehle (multiple instruction) oder nur einen (single instruction)?
- Bearbeitet ein Rechner gleichzeitig unterschiedliche Operanden (multiple data) oder nur einen (single data)?

Durch Kombination der möglichen Ausprägungen ergeben sich vier Varianten:

SISD	Single Instruction - Single Data
SIMD	Single Instruction - Multiple Data
MISD	Multiple Instruction - Single Data
MIMD	Multiple Instruction - Multiple Data

Das SISD-Prinzip entspricht dem der klassischen von-Neumann-Maschine. Das MISD-Prinzip, nach dem ein Operand parallel von mehreren Befehlen bearbeitet wird, ist zur Programmdurchführung nicht geeignet und damit unbesetzt. Für das SIMD- und das MIMD-Prinzip wird nun jeweils ein Beispiel vorgestellt.

SIMD-PRINZIP

Die Einsetzbarkeit von PRS nach dem SIMD-Prinzip hängt vor allem vom verwendeten Lösungsverfahren ab. Dieses muss in Teilschritte zerlegbar (parallelisierbar) sein, in denen die gleiche Operation parallel auf unterschiedlichen Operanden durchgeführt wird.

Abbildung 3-11 zeigt das Funktionsschema eines PRS nach dem SIMD-Prinzip. Ein zentrales Steuerwerk SW übergibt einen (aus dem Hauptspeicher gelesenen) Befehl parallel an mehrere Rechenwerke RW_i zur Ausführung. Die zugehörigen Operanden befinden sich in den

Hauptspeicherbereichen HS_i. Die Kommunikation zwischen RW und HS sowie zwischen den einzelnen RW_i wird über das Verbindungsnetzwerk (z.B. ein internes Hochgeschwindigkeits-Busnetz) abgewickelt. Durch die RW_i -Kommunikation kann z.B. ein von RW_1 berechneter Funktionswert als Argument des nachfolgenden Befehls an RW_2 übermittelt werden.

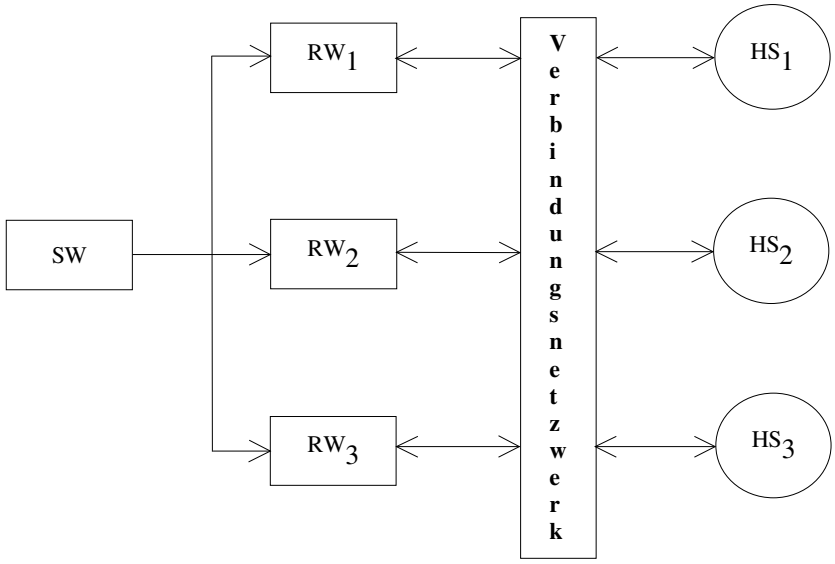


Abb. 3-11: *Parallelrechnersystem nach dem SIMD-Prinzip*

MIMD-PRINZIP

Beim MIMD-Prinzip wird zwischen PRS mit verteiltem und PRS mit gemeinsamem Hauptspeicher unterschieden. Im ersten Fall verfügt jeder Prozessor über seinen eigenen, lokalen Hauptspeicher. Im zweiten Fall greifen mehrere Prozessoren koordiniert auf einen gemeinsamen Hauptspeicher zu. Das Funktionsschema für ein PRS mit verteiltem Hauptspeicher zeigt Abbildung 3-12.

Dieses Modell enthält autonome Verarbeitungseinheiten (Knoten), bestehend aus Prozessor und lokalem Hauptspeicher, die durch ein Verbindungsnetzwerk verknüpft sind. Von wesentlicher

Bedeutung für die Leistungsfähigkeit ist die Struktur des Verbindungsnetzwerks. Neben der in Abbildung 3-12 dargestellten Ringstruktur werden Gitter-, Baum- und vor allem Hypercube-Strukturen verwendet.

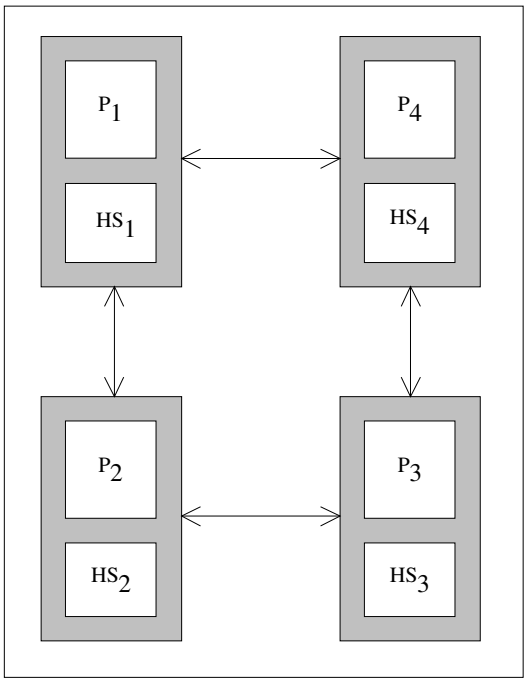


Abb. 3-12: *PRS nach dem MIMD-Prinzip mit verteiltem Hauptspeicher*

Bei der zweiten Variante, dem MIMD-Prinzip mit gemeinsamem Hauptspeicher, bestehen die Knoten lediglich aus einem Prozessor (und ggf. einem Cache-Speicher). Die Prozessoren greifen über das Verbindungsnetzwerk auf den gemeinsamen Hauptspeicher zu. Ein direkter Datenaustausch zwischen den Prozessoren ist nicht möglich.

PRS nach dem MIMD-Prinzip sind auch für IS einsetzbar, deren Lösungsverfahren in parallel ausführbare Teile zerlegbar sind. Diese müssen nicht notwendig gleichartig sein. Da jeder Prozessor eine autonome Befehlsfolge bearbeitet, ist z.B. eine parallele Durchführung unterschiedlicher Prozeduren möglich. Ein breiterer Einsatz von MIMD-Rechnern scheitert

derzeit vor allem an der mangelnden Verfügbarkeit geeigneter Lösungsverfahren sowie der zu ihrer Implementierung notwendigen Software-Konzepte und Entwicklungswerkzeuge.

3.2 RECHNERNETZE

Rechnernetze (RN) dienen der verteilten Informationsbeschaffung, -verarbeitung, -aufbewahrung und -weiterleitung. Die beteiligten Rechner befinden sich an geographisch verschiedenen Orten. Ein RN kann als ungerichteter Graph abgebildet werden, bei dem die Knoten Systemkomponenten und die Kanten mögliche Kommunikationswege darstellen.

RN dienen der Realisierung verteilter Systeme. Ein verteiltes System liegt vor, wenn

a) die Programme eines Anwendungssystems arbeitsteilig von mehreren Prozessoren durchgeführt werden,
b) der Nachrichtenaustausch zwischen den Programmen über ein die Prozessoren verbindendes Kommunikationssystem realisiert wird und
c) das Gesamtsystem aus Außensicht als ein System betrachtet werden kann.

Ziele von RN sind:

- Lastverbund (Verteilung von Rechnerleistung)
- Leistungsverbund (Zerlegung des zu bearbeitenden Problems in Teilprobleme)
- Datenverbund (Aufspaltung großer Datenbestände und gemeinsame Nutzung)
- Kommunikationsverbund (zum Informationsaustausch)
- Betriebsmittel- und Sicherheitsverbund (gemeinsame Nutzung von Ressourcen, Vorkehrungen für Komponentenausfall)

Die Übertragung von Daten in RN kann auf die folgenden Arten geschehen:

- analog (Kbit/s) mittels elektromagnetischer Wellen, oder

- digital (Mbit/s) mittels Pulse-Code-Modulation .

Begonnen hat die Entwicklung von Netzen mit analoger Übertragung und Modemzugang. Inzwischen gibt es digitale Netze zur weltweiten Kommunikation.

Bei den Netz-Betriebsarten wird unterschieden in

- Simplex: entweder nur Sende- oder nur Empfangsbetrieb,
- Halbduplex: Wechselbetrieb (analoges Netz),
- (Voll-) Duplex: Gegenbetrieb (digitales Netz).

Die Kommunikation in Netzen wird durch Kommunikationsprotokolle geregelt. Dies sind Regeln zur Organisation des Datenaustauschs. Sie beziehen sich auf die jeweilige Anwendung, den Verbindungsauf- und -abbau und die Datenübertragung. Das Kommunikationsproblem wird entsprechend der auszuführenden Funktionen in einzelne Schichten (Ebenen) zerlegt. Folgendes Beispiel verdeutlicht die Schichtenbildung:

2 verschiedensprachige Gesprächspartner kommunizieren zu einem Thema auf Ebene 3
2 Übersetzer übersetzen den Dialog in eine gemeinsame Zielsprache auf Ebene 2
2 Übermittler sorgen für den physischen Austausch der Informationen auf Ebene 1

Ein Ziel von Kommunikationsprotokollen ist es, Vereinbarungen gleicher Ebenen ändern zu können, ohne Vereinbarungen anderer Ebenen zu beeinflussen. Jede Ebene benötigt für die Erfüllung ihrer Funktion alle Funktionen der in der Hierarchie darunter liegenden Ebenen.

Das bekannteste Kommunikationsprotokoll ist im ISO-OSI (Open Systems Interconnection)-Referenzmodell geregelt. Es besteht aus den foldenden sieben Schichten:

(1) Physical Layer (Übertragungsschicht) regelt die physikalische Übertragung von Signalen auf einem Medium;

Beispiel: X.21 (CCITT) für digitale, V.24 bzw. RS-232 für analoge Übertragung;

(2) Data Link Layer (Sicherungsschicht) regelt die Übertragung von Datenblöcken auf einzelnen Abschnitten;

Beispiel: High Level Data Link Control (ISO) für synchrone Übertragung

(3) Network Layer (Vermittlungsschicht) regelt Zwischenspeicherung und Routing der Datenblöcke;

Beispiel: Leitungs-, Paket- und Zellenvermittlung; X.25 (CCITT) für Paketvermittlung;

(4) Transport Layer regelt die Bereitstellung der Endsystemverbindungen

Beispiel: Auf- und Abbau einer logischen Verbindung;

(5) Session Layer (Kommunikationssteuerschicht) regelt die Steuerung des Benutzerdialogs;

(6) Presentation Layer (Datendarstellungsschicht) legt die Vereinbarung der Datendarstellung fest; Beispiel: ASCII, EBCDIC;

(7) Application Layer (Dienstschicht) regelt den anwendungsbezogenen Austausch zwischen Teilnehmern; Beispiele: X.400 email, FTAM Filetransfer.

Übertragungsmedien beziehen sich auf die Übertragungsschicht (Ebene 1). Man verwendet:

- Verdrillte Kupferkabel (billig, aber geringe Bandbreite, hohe Störanfälligkeit, geringe Abhörsicherheit, bei großen Distanzen schlechte Übertragungsqualität)
- Koaxialkabel mit freiem Außenleiter (Grund) und isoliertem Innenleiter (Signal) (weniger störempfindlich, sowohl Basisbandverfahren (nur ein nutzbarer Informationskanal) als auch als Breitbandverfahren (mehrere Informationskanäle) einsetzbar, hohe Steifigkeit (schwer verlegbar))
- Glasfaserkabel (Lichtwellenleiter) mit einer Leitfaser, die von einem Glasmantel niedriger Brechzahl umschlossen wird (Übertragung mittels Laserlichtimpulse, Probleme bei Kopplung und Verstärkung, schnell, hohe Bandbreite, gute Verlegbarkeit, störunempfindlich, abhörsicher, Übertragungsgeschwindigkeiten von > 600 [Mbit/s]

- Richtfunk zur drahtlosen Übertragung durch elektromagnetische Wellen (Langwelle (100 kHz) bis Dezimeterwelle (500 GHz), erdgebunden (erfordert freie optische Sicht) oder via Satelliten (geostationär, Zeitverzögerung))

Zugangsverfahren für Netze werden auf der Sicherungsschicht (Ebene 2) geregelt. Man unterscheidet:

Deterministischer Zugang (ohne Kollisionen) als

- Frequency Division Multiplexing: fester Anteil der Bandbreite wird für jede Station permanent reserviert;
- Synchrones Time Division Multiplexing: ein festes, zyklisch auftretendes Zeitintervall wird jeder Station für den Zugang zugeordnet.

Random Zugang (Kollisionen möglich) als

- ALOHA: Station darf immer senden, jede Station prüft, ob sie der Empfänger ist;
- Slotted ALOHA: Station darf nur zu Beginn vorgegebener Zeitscheiben senden;
- CSMA (Carrier Sense Multiple Access): sendewillige Station hört das Übertragungsmedium ab, bevor sie sendet.

Nachfrageabhängiger Zugang (ohne Kollisionen) durch zentrale oder verteilte Steuerung als

- zentrales Polling: Stationen werden durch Adressierung von einer Zentrale angesprochen;
- dezentrales Token Passing: solange keine Daten übertragen werden, signalisiert eine spezielle Bitfolge (Token) ein Freizeichen. Eine sendewillige Station wartet das Token ab, nimmt es vom Netz und beginnt mit der Übertragung. Nach Abschluss der Übertragung wird das Token wieder hergestellt.

Leistungskriterien für Zugangsverfahren sind

- Wartezeit einer Station zwischen realisierter und möglicher Übertragung;

- Anzahl Daten pro Zeiteinheit, die übertragen werden können.

Vermittlungsverfahren werden auf der Vermittlungsschicht (Ebene 3) geregelt. Man unterscheidet:

- Durchschalte- bzw. Leitungsvermittlung (circuit switching). Die Verbindung wird in den Vermittlungsstellen auf Dauer durchgeschaltet und steht den Teilnehmern exklusiv zur Verfügung.
- Speicher- bzw. Paket-Vermittlung (store and forward switching); zu übertragende Daten werden in Blöcke begrenzter Länge aufgeteilt. Nutz- (Rumpf) und Steuerinformation (Kopf) bilden ein Paket. Die Steuerinformationen dienen der Nummerierung und der Zuordnung zu einer Verbindung.
- Zellenvermittlung (ATM - Asynchronous Transfer Mode)
  ATM arbeitet nach einem vereinfachten Paketvermittlungsverfahren. Das ATM-Verfahren basiert auf der Übermittlung von ATM-Zellen, die eine begrenzte und feste Länge haben.

Im einem Rechnernetz sind autonome Rechner durch ein Kommunikationssystem verknüpft. Die Eigenschaft der Autonomie bedeutet, dass die Rechner 'lose gekoppelt' sind, d.h. dass jeder Rechner unter der Verwaltung eines eigenen Betriebssystems selbständig Programme ausführt.

Entsprechend der räumlichen Entfernung zwischen den Rechnern und den verwendeten Kommunikationskanälen werden u.a. folgende Netz-Klassen unterschieden:

- Globale Netzwerke (Wide Area Networks, WAN) sind über Länder oder Kontinente verteilt,
- Lokale Netzwerke (Local Area Networks, LAN) umfassen typischerweise Rechner auf einem Grundstück.

Ein globales Netz mit der weltweit größten Bedeutung ist das Internet. Der Datenaustausch zwischen den Rechnern des Internet erfolgt nach den Protokollen TCP und IP. Das Internet stellt u.a. folgende Dienste bereit:

E-Mail zum Austausch persönlicher Informationen,
Usenet für thematisch gegliederte Diskussionsforen,
Telnet zum Arbeiten auf entfernten Rechnern,
FTP zum Software- und Dateitransport zwischen entfernten Rechnern,
IRC für weltweite Live-Diskussionen,
WAIS zur Suche in global verteilten Datenbeständen,
WWW als weltweit verteiltes multimediales Informationssystem.

Der bekannteste Dienst ist WWW. Hier stellen Anbieter ihre Informationen über WWW-Server zur Verfügung. Nachfrager können sich Informationen mit Hilfe von WWW-Clients, im Allgemeinen WWW-Browser genannt, besorgen. Die Informationseinheiten werden häufig als *HTML*-Dokumente angeboten, die auch erst bei Zugriff durch *CGI*-Skripte generiert werden können. Durch CGI-Skripte lassen sich auch Kommunikationsverbindungen zu Datenbanken realisieren. HTML-Seiten können neben Texten, Graphiken, Audio, Video und Animationen auch Applikationen (*Java-Applets*) enthalten. Ein Browser lädt eine HTML-Seite unter Kenntnis der Adresse, genannt *Unified Resource Locator* (URL), von einem WWW-Server mit Hilfe des *Hypertext Transfer Protocols* (HTTP) und zeigt sie am Bildschirm an. Die Vorteile des Internets machen sich Unternehmen inzwischen als Intranet (im Unternehmen) und als Extranet (zwischen Unternehmen) zunutze.

Die folgenden Ausführungen beziehen sich überwiegend auf LANs. Hier existiert eine Vielfalt an Produkten mit unterschiedlichen Übertragungsmedien, -geschwindigkeiten und Netzstrukturen.

Die Struktur eines Netzes ist in Abbildung 3-13 dargestellt. Arbeitsrechner AR führen Anwendungsprogramme durch. Das Kommunikationssystem besteht aus Vermittlungsrechnern VR, einem Übertragungsmedium, sowie Ankopplungen der VR an das Übertragungsmedium. Aufgabe eines VR ist das selbständige Senden und Empfangen von Nachrichten für einen AR. VR werden i.d.R. als eigene Hardware-Komponenten realisiert. Im Folgenden wird die Vereinigung aus AR, VR und Ankopplung als Station bezeichnet.

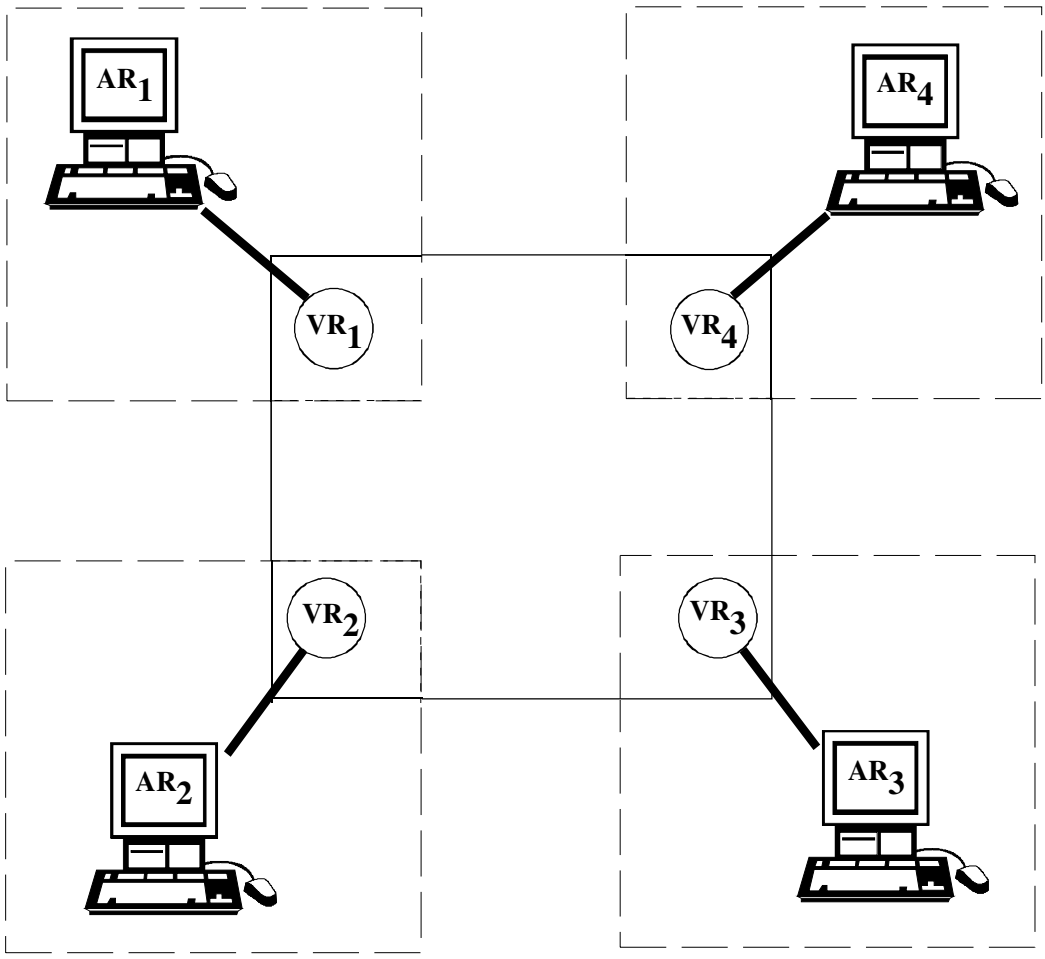


Abb. 3-13: *Rechnernetz*

Zur Verbindung von Stationen haben sich spezielle Netzstrukturen (Netztopologien) herausgebildet, von denen neben dem Sternnetz das Bus- und das Ringnetz am häufigsten anzutreffen sind.

BUSNETZ

Alle Stationen sind durch eine gemeinsame Leitungsverbindung (Bus) verknüpft (vgl. Abbildung 3-14). Der Bus kann zeitabschnittsweise der jeweils sendenden Station mit deterministischem Zugang zur Verfügung gestellt werden (Division-Multiplexing) oder es kann ein Random Zugang realisiert werden.

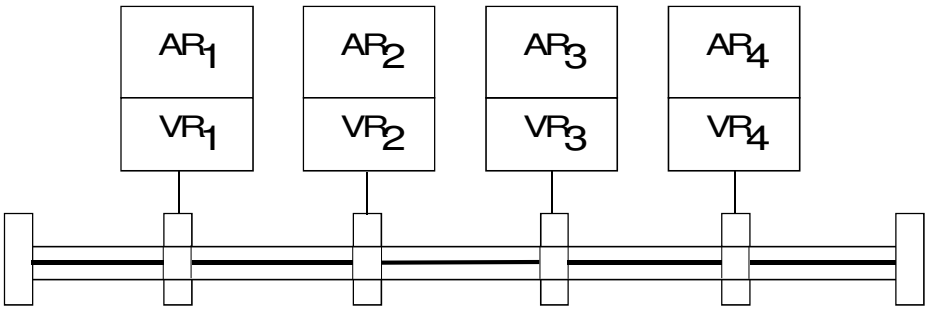


Abb. 3-14: *Busnetz*

RINGNETZ

Beim Ringnetz ist jede beteiligte Station mit ihrer linken und rechten Nachbarstation gekoppelt. Die hierzu verwendeten Ringankopplungen sind aktive Komponenten, die Transportfunktionen durchführen (vgl. Abbildung 3-15). Sendungen werden dabei von Station zu Station

weitergegeben. Bei der Steuerung des Netzzugangs dominieren nachfrageabhängige dezentrale Zugangsverfahren.

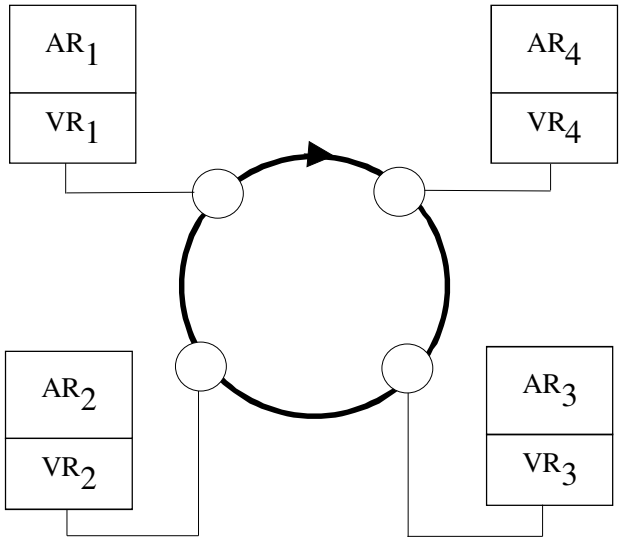


Abb. 3-15: *Ringnetz*

Bekannte LANs sind:

- Ethernet mit Bus-Struktur und CSMA/CD Zugang,
- Token Bus (Ring) mit Bus- (Ring-) Struktur und Token Passing Zugang,
- Cambridge Ring mit Ring-Struktur und Empty Slot Zugang.

3.3 SYSTEMSOFTWARE

Bei der Ausführung eines Programms werden verschiedene Aufgaben voneinander getrennt. Beispielsweise wird die Datenverwaltung mit Hilfe eines Datenbankverwaltungssystems (DBVS), die Mensch-Maschine-Kommunikation mit Hilfe eines User-Interface-Management-Systems (UIMS) und die Rechner-Rechner-Kommunikation mit Hilfe eines Netzwerk-Betriebs-

systems (NWBS) realisiert. Die Funktionen und die Schnittstellen dieser sog. Basismaschinen sind im Allgemeinen standardisiert oder besitzen zumindest eine größere Verbreitung.

Die Systemsoftwarekomponenten DBVS, UIMS und NWBS sowie das Anwendungsprogramm nutzen das Betriebssystem (BS). Das Betriebssystems besteht aus den elementaren Funktionen des Rechenwerks einer Universalrechenmaschine (URM) oder eines Rechnerverbundsystems (RVS). Das BS stellt somit das Bindeglied zwischen der Hardware sowie dem Anwendungsprogramm, UIMS, DBVS und NWBS dar.

BS, DBVS, UIMS und NWBS bilden die *Systemsoftware* eines IS. Ihr Zusammenwirken ist in Abbildung 3-16 dargestellt. Sie wird in den folgenden Abschnitten behandelt. *Middleware* umfasst die Klasse systemnaher Software zwischen Systemsoftware und IS. Zwei ausgewählte Middleware-Konzepte werden später vorgestellt.

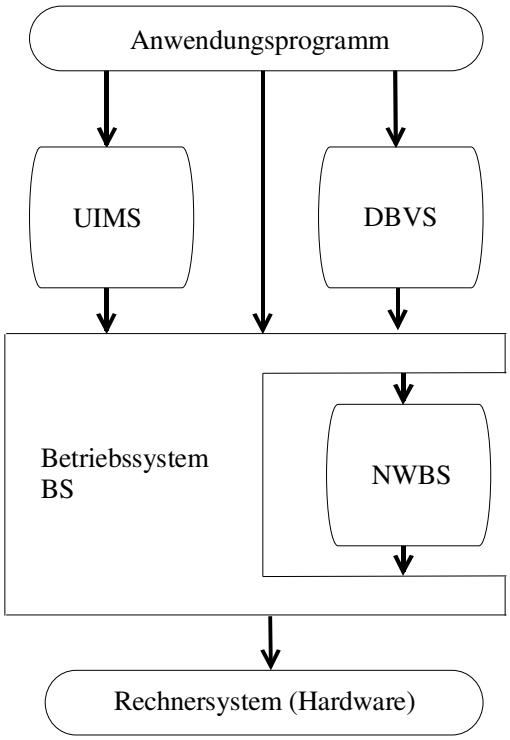


Abb. 3-16: *Systemsoftware-Schichten BS, NWBS, UIMS und DBVS*

3.3.1 BETRIEBSSYSTEME

Zentrale Aufgabe eines Betriebssystems (BS) ist die Verwaltung und Kontrolle der Betriebsmittel, wie Prozessoren, Arbeitsspeicher, peripherer Speicher, Ein- und Ausgabegeräte sowie Netzwerkschnittstellen. Aus der Sicht des BS stellt die Durchführung eines Programms einen Prozess dar, der von ihm zu verwalten ist. Im Allgemeinen werden von einem BS mehrere Prozesse gleichzeitig verwaltet. Die von den Prozessen benötigten Ressourcen werden als Betriebsmittel bezeichnet und ebenfalls vom BS bereitgestellt. Im Folgenden werden einige Konzepte der Prozess-, Speicher-, Datei- und Geräteverwaltung dargestellt.

Der *Geräteverwaltung* obliegt die Überwachung und Steuerung von Ein- und Ausgabegeräten (E /A-Geräte) und zerfällt in vier Schichten. Auf der der Hardware nächsten Schicht arbeitet die *Unterbrechungsbehandlung*. Sie ist dafür zuständig, Prozesse zu aktivieren bzw. zu deaktivieren. In der darüber liegenden Schicht übernehmen *Gerätetreiber* gerätespezifische Aktionen wie beispielsweise die Positionierung von Plattenarmen auf Spuren der Festplatte. Eine dritte, geräteunabhängige Softwareschicht übernimmt die Zwischenspeicherung von Daten und die Zuteilung von Geräten. Auf der obersten Schicht werden spezielle *E / A-Bibliotheken* für Anwendungen bereitgestellt.

Vom BS werden zwar mehrere Prozesse simultan verwaltet, der reale Prozessor kann allerdings zu jedem Zeitpunkt nur einen Prozess ausführen. Er schaltet zwischen den einzelnen Prozessen so schnell hin und her, dass der Eindruck einer simultanen Ausführung entsteht. Eine Aufgabe der *Prozessverwaltung* ist *Organisation* des Prozesswechsels. Der aktuelle Zustand des zu unterbrechenden Prozesses muss gespeichert werden, so dass er bei Wiederaufnahme des Prozesses zu einem späteren Zeitpunkt verfügbar ist. Eine andere Aufgabe ist die *Ablaufplanung* (Scheduling) der Prozesswechsel. Dabei muss entschieden werden welcher der verfügbaren Prozesse den Prozessor wie lange nutzen kann. Kriterien für die dabei zu treffenden Entscheidungen sind Fairness, Effizienz der Prozessorauslastung und Minimierung

der Antwortzeiten für wartende Prozesse. Zur Kommunikation zwischen Prozessen werden folgende Mechanismen bereitgestellt (vgl. Abbildung 3-17):

- Pipe: unidirektionaler Kommunikationskanal zwischen einem Sender- und einem Empfängerprozess. Gesendete Nachrichten werden in eine Warteschlange eingereiht und bis zur Entnahme durch den Empfänger dort gepuffert. Die Verwaltung der Pipe erfolgt nach dem FIFO-Prinzip (First In First Out).
- Queue: kann Nachrichten von mehreren Senderprozessen aufnehmen. Die Organisation der Warteschlange sowie die Entnahme der Nachrichten durch den Empfängerprozess aus der Warteschlange kann nach unterschiedlichen Kriterien erfolgen.
- Shared Memory: gemeinsamer Hauptspeicherbereich, der in beliebiger Weise als bidirektionaler Kommunikationskanal zwischen Prozessen genutzt wird. Im Gegensatz zu Pipe und Queue wird jedoch keine Warteschlange verwaltet.

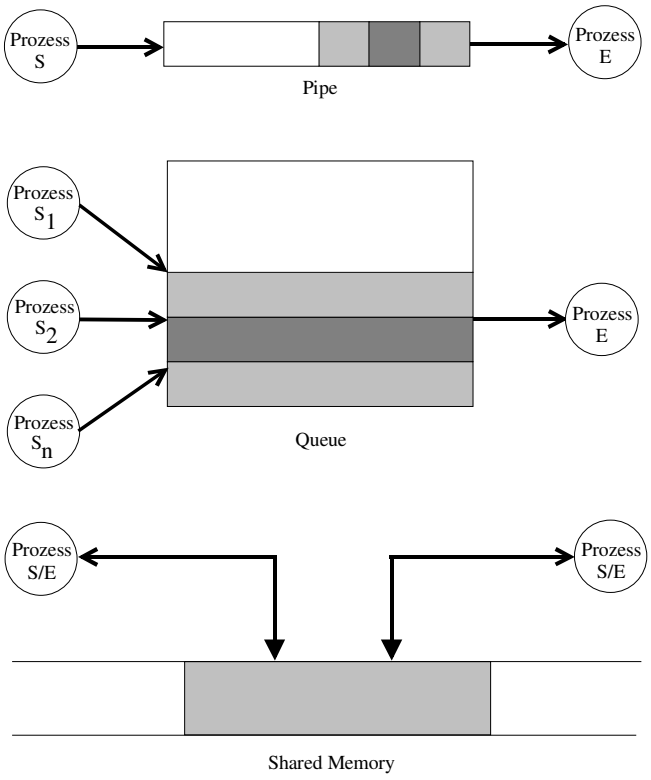


Abb. 3-17: *Möglichkeiten der Interprozess-Kommunikation*

Die *Speicherverwaltung* hat die Aufgabe, die Ein- und Auslagerung von Prozessen zwischen Arbeitspeicher und Festplatte zu organisieren sowie freien und belegten Speicherplatz zu verwalten. Ein einfaches Vorgehen zur Aufteilung ist das Swapping, bei dem ein Prozess vollständig in den Arbeitspeicher geladen wird. Ist dazu nicht genügend Platz vorhanden, wird ein anderer Prozess in den Hintergrundspeicher ausgelagert.

Ein Problem tritt auf, wenn ein Prozess mehr Speicherplatz braucht als physikalisch verfügbar ist. In diesem Fall werden die Prozesse in einzelne Fragmente aufgeteilt und nur diese im Arbeitsspeicher gehalten, die gerade benötigt werden.

Aufgabe der *Dateiverwaltung* sind die Bereitstellung von Zugriffsmechanismen auf Dateien, Strukturierungsmöglichkeiten in Form von Verzeichnissystemen und Zugriffsschutzmaßnahmen für das Lesen, Schreiben und Ausführen von Dateien. Andere Funktionen von Dateiverwaltungssystemen, die dem Benutzer aber verborgen bleiben, sind Aufteilung von Dateien auf physikalische Plattenblöcke, Behandlung defekter Plattenbereiche, Anfertigung von Sicherungskopien sowie die Gewährleistung der Konsistenz der Daten.

Beispiele kommerzieller Betriebssysteme sind MS-DOS, OS/2, UNIX, Linux und Windows XP / NT für Personal Computer, MVS und VM/370 (IBM) bzw. BS2000 (Siemens-Nixdorf) für Großrechner und Novell NetWare und LAN Manager für Netzwerke.

3.3.2 USER-INTERFACE-MANAGEMENT-SYSTEME

Eine weitere Komponente der Systemsoftware sind User-Interface-Management-Systeme (UIMS) für die MMK. Die MMK wird in Form eines Dialogs durchgeführt, der zeitlich überlappend oder im wechselseitigen Ausschluss erfolgt.

Der Ablauf der MMK ist in Abbildung 3-19 dargestellt.

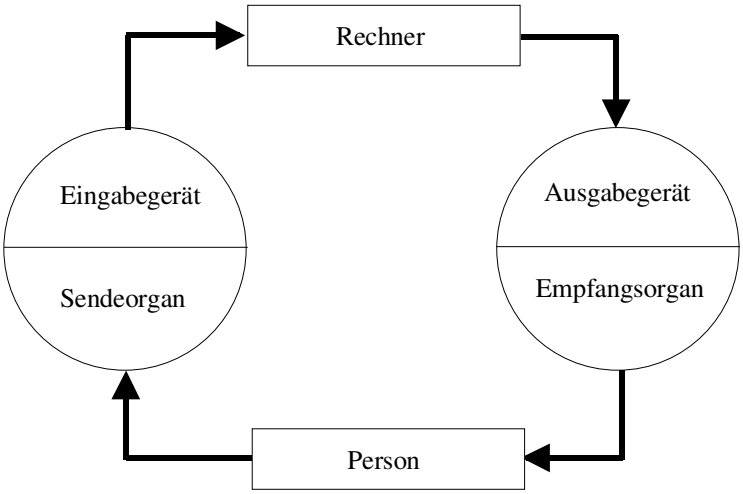


Abb. 3-18: *Ablauf der MMK*

Der Nachrichtenaustausch zwischen Mensch und Computer bedingt die Verfügbarkeit geeigneter Ein- und Ausgabegeräte der Maschine, die mit entsprechenden Sende- und Empfangsorganen des Menschen kompatibel sind. Derzeit verfügbare UIMS unterstützen insbesondere folgende Geräte und Organe:

- Eingabegeräte: Tastatur, Maus, Mikrofon, Scanner
- Ausgabegeräte: Grafikbildschirm, Lautsprecher
- Sendeorgan: Hände, Sprache
- Empfangsorgan: Augen, Ohren

Moderne UIMS beruhen auf standardisierten, generischen Objekttypen. Jeder dieser Objekttypen verfügt über Bedienelemente, über die sich Methoden des Objekttyps aufrufen lassen.

Auf der Basis der Objekttypen werden Objektausprägungen in Form von Dialogobjekten erzeugt. Die Bedienelemente eines graphischen Dialogobjekts werden mit der Maus oder der Tas-

tatur selektiert und betätigt. Hierdurch wird eine Nachricht an das Dialogobjekt gesandt, welches die Durchführung einer Methode des Dialogobjekts auslöst.

Ein Beispiel für einen generischen Objekttyp ist ein Fenster, ein Beispiel für ein Dialogobjekt ist ein konkretes Fenster bestimmter Größe, Rahmenbeschriftung etc. Zu den Bedienelementen von Fenstern gehört u.a. die Titelleiste des Fensterrahmens, die mit der Maus selektiert werden kann. Hierdurch wird beispielsweise die Methode 'VerschiebeFenster' aufgerufen. Eine andere Methode öffnet und schließt Fenster, verändert deren Größe und Position, überlagert Fenster, verwandelt Fenster in Icons und umgekehrt.

Zur Gestaltung von MMK-Schnittstellen bieten UIMS eine Reihe generischer Objekttypen an, um die konkreten, in einer Anwendung benötigten Dialogobjekte zu definieren. Die Objekttypen beinhalten graphische Bedienelemente zur Auslösung von Methoden auf den zugehörigen Dialogobjekten.

In den letzten Jahren haben sich für die verwendeten Objekttypen weitgehend akzeptierte Standards herausgebildet. Diese Standardisierung ermöglicht es, die Oberfläche unterschiedlicher IS ohne nennenswerte Einarbeitung bedienen zu können. Beispiele sind der Common User Access (CUA) Advanced Interface Design Guide, der von IBM als Teil von SAA vorgeschlagen wird, der Siemens-Nixdorf Styleguide sowie der für die nachfolgende Darstellung verwendete Microsoft Windows Styleguide.

Der Objekttyp Fenster (Window) bildet die Grundlage der MMK (vgl. Abbildung 3-19). Jeder Anwendung ist ein MDI-Fenster (Multiple Document Interface) zugeordnet, in welchem die anwendungsspezifischen Objekte dem Nutzer präsentiert werden. Der Aufbau von Fenstern und die Anordnung ihrer Bedienelemente sind grundsätzlich frei gestaltbar. Fenster sind auf dem realen Bildschirm verschiebbar und in der Größe veränderlich.

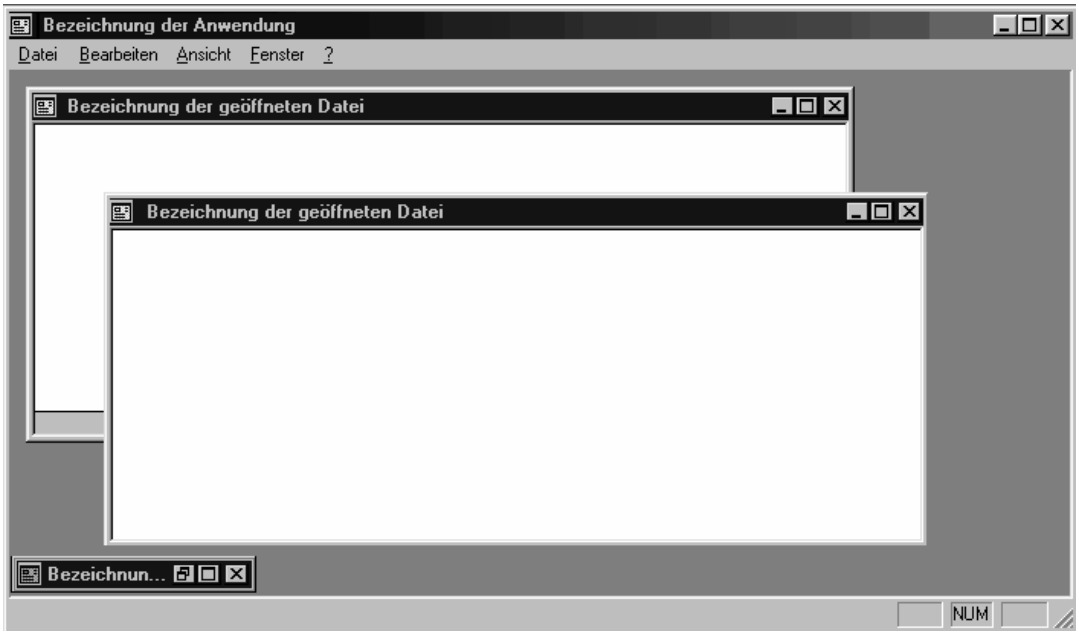


Abb. 3-19: *MDI-Fenster mit mehreren Client-Fenstern*

Ausgehend von einem MDI-Fenster können zwei weitere Arten von Fenstern geöffnet werden: Client-Fenster und Dialogboxen. Client-Fenster sind wie MDI-Fenster aufgebaut und jeweils genau einem MDI-Fenster zugeordnet. Dialogboxen (Dialog Box) sind spezielle Fenster fester Größe zur Steuerung des Dialogablaufs (vgl. Abbildung 3-20). In einer Dialogbox werden Eingaben des Nutzers angefordert.

Abb. 3-20: *Dialogbox*

Einem MDI- oder Client-Fenster sind Menüs zugeordnet, welche die Auswahl und Auslösung von Methoden auf dem Fenster ermöglichen. Die Menüs werden stets im MDI-Fenster

angezeigt und wechseln daher mit dem gerade aktiven Fenster. Standard-Menüs sind Datei (File), Bearbeiten (Edit), Ansicht (View), Fenster (Window) und Hilfe (Help). Abbildung 3-21 zeigt das Menü Bearbeiten mit den dort angebotenen Methoden.

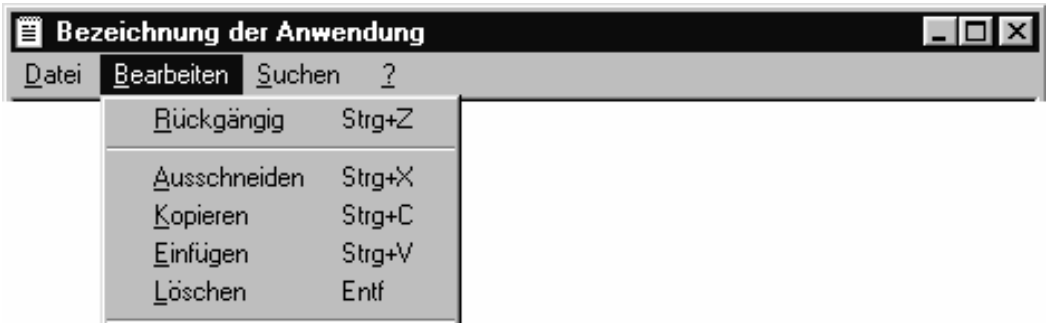


Abb. 3-21: *Menü Bearbeiten*

Nachstehend sind einige Basisobjekte für den Dialogablauf aufgeführt:

a) Aktionsknöpfe (Push Buttons): Graphische Symbole zur Auslösung von Methoden (vgl. Abbildung 3-20). Graphische Symbole zur Benennung von Objekten werden als lkone (Icons) bezeichnet.

b) Einfachauswahlknöpfe (Radio Buttons): Graphische Symbole zur Auswahl von genau einer Alternative aus einer vorgegebenen Menge von Alternativen (vgl. Abbildung 3-22).

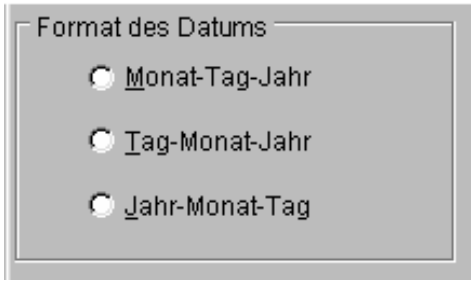


Abb. 3-22: *Einfachauswahlknöpfe*

c) Mehrfachauswahlknöpfe (Check Boxes): Graphische Symbole zur Auswahl einer oder mehrerer Alternativen aus einer vorgegebenen Menge von Alternativen (vgl. Abbildung 3-23).

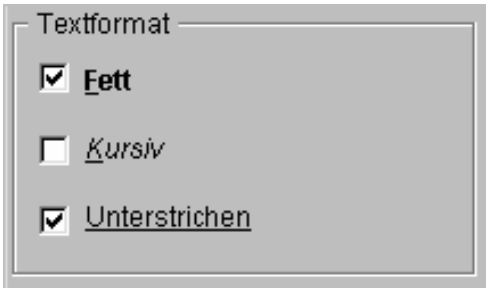


Abb. 3-23: *Mehrfachauswahlknöpfe*

d) Listen (List Boxes): Box zur Einfach- oder Mehrfachauswahl von Alternativen aus einer variablen Menge von Alternativen (vgl. Abbildung 3-24). Ist das Auswahlfeld einer Liste editierbar, so liegt eine Combo Box vor.

Abb. 3-24: *Liste*

e) Eingabefelder (Entry Fields): Ein- oder mehrzeilige Bereiche für Tastatureingaben des Nutzers (siehe Eingabefeld zu Dateiname in Abbildung 3-24).

f) Verschiebebalken (Scroll Bars): Verschieben des sichtbaren Ausschnitts über einem Objekt. Verschiebebalken können horizontal oder vertikal angeordnet sein (vgl. Abbildung 3-24).

Im Folgenden werden einige Beispiele für UIMS erläutert.

X-WINDOWS

Das X-Window-System (Kurzbezeichnung X) ist ein portables Fenstersystem für Graphikbildschirme. X wurde am Massachusetts Institute of Technology (MIT) entwickelt. Der Quellcode ist frei erhältlich (public domain). X kann somit von unterschiedlichen Herstellern adaptiert und bei Einhaltung des Standardprotokolls auch unter der Bezeichnung X vertrieben werden. X ist derzeit für eine Vielzahl von Workstations, aber auch für Personal Computer verfügbar.

Das Systemmodell von X umfasst drei Komponenten:

a) Serverprogramm: Das Serverprogramm steuert die verwendeten Ein-/Ausgabegeräte (Bildschirm, Tastatur, Maus usw.). Es stellt hierzu Funktionen zum Erzeugen von Fenstern, zum Ausgeben von Bildern und Texten in Fenstern, zum Verarbeiten von Eingaben, usw. zur Verfügung.

Jeder Serverprozess bedient die Ein-/Ausgabegeräte genau eines Bildschirmarbeitsplatzes. Bei mehreren Bildschirmarbeitsplätzen sind vom Betriebssystem mehrere Serverprozesse zu

verwalten. Ein Serverprozess führt Funktionen ausschließlich auf Anforderung von zugehörigen Clientprozessen durch.

b) Clientprogramm: Die Clientprogramme stellen aus der Sicht des Serverprogramms Anwendungen dar. Dabei sind zwei Arten von Clientprogrammen zu unterscheiden:

   - der Anwendungsteil des IS und
   - der Window Manager, der die Fenster der Anwendung verwaltet.

   Die Funktionen des Serverprogramms werden den verschiedenen Clientprogramme durch die Bibliothek Xlib bereitgestellt. Jedes Clientprogramm generiert einen eignen Prozess.

c) Kommunikationskanal: Der Kommunikationskanal dient zur Kommunikation zwischen Clientprozessen und Serverprozessen. Es werden zwei grundsätzliche Konfigurationen unterschieden:
   1. Client- und Serverprozesse laufen auf einem gemeinsamen Rechner ab. In diesem Fall wird der Kommunikationskanal über einen IPC-Mechanismus realisiert.
   2. Client- und Serverprozesse laufen auf verschiedenen Rechnern eines Rechnerverbundsystems (RVS) ab. Bei dieser Konfiguration wird der Kommunikationskanal und das Protokoll des RVS genutzt. Ein Fenster, auf dem lediglich ein Serverprozess abläuft, wird als *X-Terminal* bezeichnet.

MS-WINDOWS

MS-Windows ist von Microsoft, das primär in Verbindung mit MS-DOS eingesetzt wird, aber auch z.B. für OS/2 zur Verfügung steht. In Verbindung mit MS-DOS realisiert MS-Windows ein rudimentäres Multi-Tasking. Alle Prozesse laufen auf einem Rechner ab. Client-Server-Architekturen werden nicht unterstützt.

MS-Windows ist in der Version NT um Funktionen eines NWBS erweitert, die es ermöglichen, MS-Windows in Client-Server-Architekturen unter Nutzung von Rechnerverbundsystemen einzusetzen.

PRESENTATION MANAGER

Der Presentation Manager wird als Erweiterung des Betriebssystems OS/2 von Microsoft und IBM angeboten. Der Presentation Manager ist auf das hierarchische Prozesskonzept von OS/2 ausgerichtet.

3.3.3 MIDDLEWARE

Der Begriff Middleware beschreibt eine neuere Klasse von systemnaher Software als Bindeglied zwischen Systemsoftware und Anwendungsprogramm. Middleware unterstützt dabei speziell die Realisierung verteilter IS.
Middleware deckt ein breites und heterogenes Spektrum unterschiedlicher Konzepte ab. Im Folgenden werden zwei Middleware-Konzepte vorgestellt und an einem Beispiel erläutert.

DATENBANK-GATEWAYS

Datenbanken liefern oft Daten für unterschiedliche IS. Um einen einheitlichen Zugriff zu gewährleisten bedarf es entsprechender Schnittstellen. Datenbank-Gateways unterstützen die Kommunikation zwischen einem Client (Anwendungsteil) und unterschiedlichen, heterogenen Datenbank-Servern (Datenverwaltungsteil). Ein Ziel von Datenbank-Gateways ist es, für den

Client eine einheitliche Zugriffsschnittstelle bereitzustellen. Dadurch wird die Unabhängigkeit des Anwendungsteils vom Datenverwaltungsteil erhöht. Zum Beispiel kann dann in einfacher Weise der Datenverwaltungsteil eines Anwendungsprogramms durch ein anderes ersetzt werden.

Ein Beispiel für ein Datenbank-Gateway ist Open Database Connectivity (ODBC) von Microsoft. ODBC beruht auf dem Call Level Interface (CLI) der SQL Access Group, einer Vereinigung von Herstellern und Anwendern von Datenbanken. Die Architektur von ODBC zeigt Abbildung 3-25:

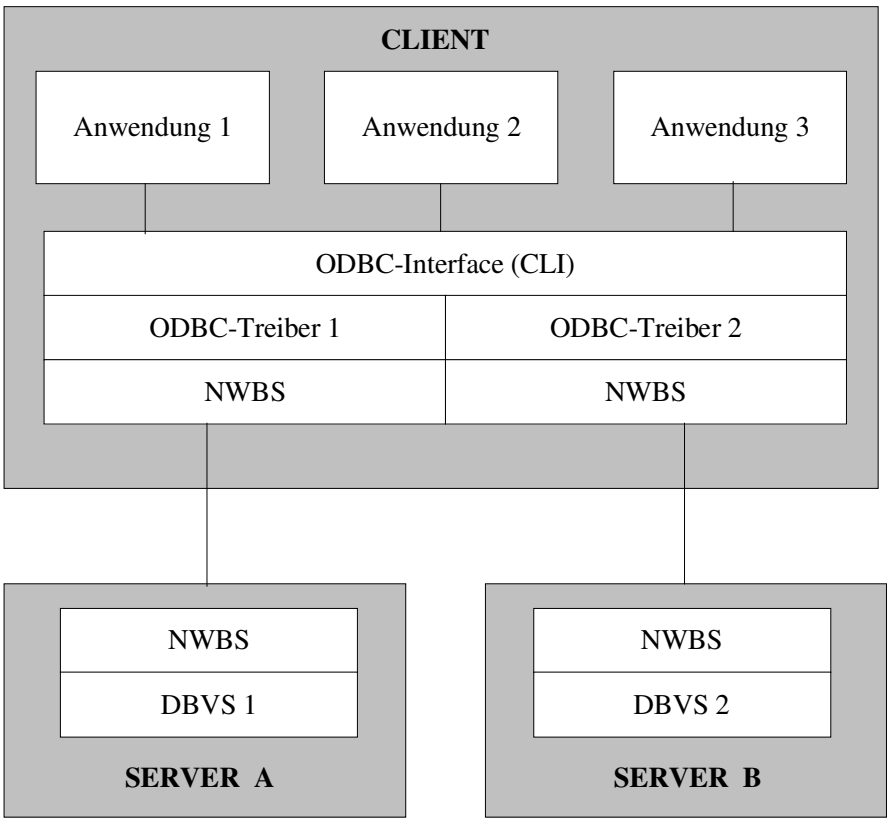


Abb. 3-25: *Datenbank-Gateway am Beispiel von ODBC nach [Rah94]*

· Client: Die einzelnen Anwendungsprogramme greifen über eine einheitliche Schnittstelle (ODBC-Interface) auf die DBVS zu. Die Anpassung an unterschiedliche Datenbank-Spra-

chen erfolgt auf der Seite des Client durch einen ODBC-Treiber. Für jedes zu unterstützende DBVS wird ein separater ODBC-Treiber benötigt. Der ODBC-Treiber greift über Funktionen des NWBS auf das Kommunikationssystem zu.

- Server: Der Server umfasst ein relationales DBVS, welches über Funktionen des NWBS auf das Kommunikationssystem zugreift.

Der Client greift transparent auf den Server zu. Client und Server können dabei lokal auf einem Rechner, oder auf unterschiedlichen Rechnerknoten eines Netzes installiert sein.

OBJECT REQUEST BROKER

Object Request Broker unterstützen die transparente Kommunikation lose gekoppelter Objekte in verteilten IS. Die Objekte folgen dem Konzept des Objekttyps und sind Komponenten des Funktionsteils, des Kommunikationsteils oder des Datenverwaltungsteils eines verteilten IS.

Das Konzept des Object Request Broker wird im Folgenden anhand der Common Object Request Broker Architecture (CORBA) der Object Management Group (OMG), einer Vereinigung von Herstellern und Anwendern, erläutert.

Abbildung 3-26 zeigt die CORBA-Architektur. Kernstück ist der eigentliche Object Request Broker (ORB Core), der die Basisdienste für die Kommunikation von Objekten bereitstellt. Der ORB Core nutzt seinerseits Funktionen des zugrunde liegenden NWBS.

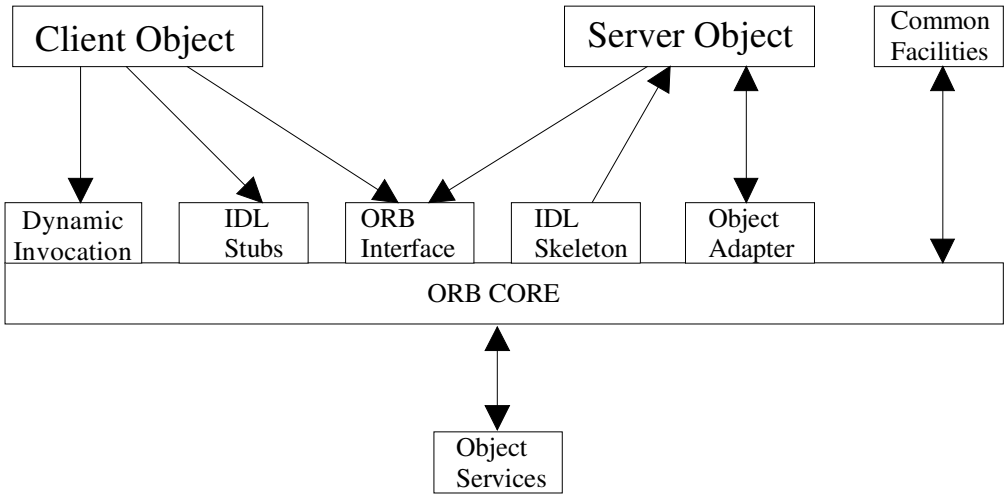


Abb. 3-26: *Object Request Broker am Beispiel von CORBA nach [OHE96]*

Die Kommunikation zwischen zwei Objekten folgt dem Client/Server-Prinzip. Es werden zwei alternative Formen der Kommunikation unterstützt:

a) Die statische Kommunikation unterstützt die häufige und zeitkritische Kommunikation von Objekten. Client- und Server-Objekte sind über vordefinierte Schnittstellen mit dem ORB verknüpft. Die Schnittstelle eines Client-Objekts wird als Stub, die des Server-Objekts als Skeleton bezeichnet. Die Schnittstellen werden automatisch aus der Schnittstellenbeschreibung der Objekte generiert, die mit Hilfe der Interface Definition Language (IDL) von CORBA erstellt werden.

b) Die dynamische Kommunikation unterstützt eine sporadische und weniger zeitkritische Kommunikation von Objekten. Dabei werden die Anordnung und die Typisierung der Aufrufparameter erst zur Laufzeit festgelegt. Die hierzu benötigten Beschreibungen sind in einem Interface Repository abgelegt, das über das ORB Interface zur Verfügung steht.

Über die Kommunikationsdienste hinaus unterstützt CORBA die Erstellung verteilter Anwendungssysteme durch Bereitstellung höherer Basismaschinen und durch Konzepte zur Erhöhung der Unabhängigkeit:

- Die Umsetzung der CORBA-Datenstrukturbeschreibung in die jeweilige Zielsprache eines Server-Objektes wird durch Object Adapter unterstützt.

- In Form der Object Services werden allgemeine Dienste zur Objektverwaltung bereitgestellt (z.B. Erzeugen, Löschen, Verlagern, persistentes Speichern von Objekten).

- Die Common Facilities umfassen eine Bibliothek vordefinierter Objekte, die von Client- und von Server-Objekten genutzt werden können. Es wird zwischen horizontalen und vertikalen Common Facilities unterschieden. Horizontale Common Facilities stellen funktional verwendbare Objektklassen zur Verfügung. Beispiele sind Objektklassen für Editieren, Drucken und Fehlerbehandlung von Objekten. Vertikale Common Facilities stellen Objektklassen für bestimmte Domänen bereit. Beispiele sind Objektklassen für Handel, Finanzdienstleistung und Gesundheitswesen.

Für CORBA sind eine Reihe von Implementierungen unterschiedlicher Hersteller verfügbar. Durch das Konzept der vertikalen Common Facilities wird die Entwicklung eines Marktes für standardisierte Softwarebausteine gefördert.

3.4 COMPILER

Zur Umsetzung von Datenstrukturen und Algorithmen werden Programmiersprachen eingesetzt. Eine Sprache ist die Gesamtheit aller möglichen Aussagen und Bedeutungen. Sie folgt einem System von Regeln (Grammatik). Jede Sprache hat eine Syntax und eine Semantik.

Die Syntax einer Sprache legt fest, welche Sätze einer Sprache zulässig sind. Beispielsweise ist 'This is a note on programming languages' nicht zulässig in der deutschen Sprache.

Die Semantik einer Sprache legt fest, welche Bedeutung zulässige Sätze haben. Beispielsweise sind der Satz 'Eine Kugel ist eckig.' und der Satz 'Dieser Satz besteht aus zehn Wörtern' falsch.

Man unterscheidet die folgenden Programmiersprachen:

- Maschinensprache ist die interne Sprache eines Rechners (z.B. steht '01011001' für 'addiere').
- Assembler ist eine maschinenorientierte Sprache, die auf einen Rechnertyp bezogen ist (z.B. steht 'ADD' für 'addiere'); sie muss vor der Ausführung in die entsprechende Maschinensprache übersetzt werden.
- Prozedurale Sprachen sind häufig auf einen Problembereich ausgerichtet (COBOL, FORTRAN, ALGOL, C, MODULA, PASCAL, BASIC).
- Deskriptive Sprachen sind häufig Abfragesprachen für Datenbanksysteme (SQL, NATURAL, ACCESS BASIC).
- Wissensorientierte Sprachen (PROLOG, LISP)
- Objektorientierte Sprachen (SMALLTALK, C++, Java)
- Endbenutzer-Sprachen (EXCEL, IFPS)

Die Übersetzung einer höheren Programmiersprache in die Maschinensprache wird durch den Compiler (vollständige Übersetzung) und / oder den Interpreter (Übersetzung Befehl für Befehl) durchgeführt. Beispielsweise wird Java zuerst in den Byte-Code kompiliert; dieser wird anschließend vom Interpreter (Java Virtual Machine) interpretiert. Compiler sind Programme, die Programme einer Quellsprache auf (grammatikalische) Fehler untersuchen und in eine Zielsprache transformieren, wobei Quell- und Objektprogramm semantisch äquivalent sind.

Ein Compiler besteht aus der lexikalischen Analyse (Scanner), der syntaktischen Analyse (Parser), der semantischen Analyse und der Code-Erzeugung. Die Aufgabe des Scanners ist es, das Quellprogramm zeichenweise einzulesen und Grundsymbole (Schlüsselwörter, Bezeichner, Zahlen, Operatoren etc.) zu erkennen. Die Überprüfung der syntaktischen Korrektheit des

Quellprogramms ist Aufgabe des Parsers. Ziel der semantischen Analyse ist die Erkennung und Überprüfung semantischer Nebenbedingungen, die sich nicht direkt aus dem Quellprogramm ableiten lassen. Bei der Code-Erzeugung wird der Code generiert, der von einem speziellen Rechner ausgeführt werden kann. Häufig wird zunächst Assemblercode erzeugt, der dann nochmal in Maschinencode übersetzt wird.

4. ENTWICKLUNG EINER PROBLEMLÖSUNG

Die Entwicklung einer computergestützten Problemlösung erfolgt in den Phasen

(1) Problembeschreibung

(2) Verbale Formulierung der Problemlösung

(3) Umsetzen in semi-formale Formulierung

(4) Programmierung der semi-formalen Formulierung

(5) Abnahme, Einsatz und Pflege des Programms

Probleme werden mit Rechnern durch die Ausführung von Programmen gelöst. Um ein Problem zu lösen, muss es zunächst korrekt und vollständig beschrieben werden. Die Problembeschreibung erfolgt zunächst verbal in einer natürlichen Sprache. Man beginnt gewöhnlich bei den *Daten* mit einer Festlegung, welche Daten in welcher Form einzugeben sind (Input-Daten) und welche Daten in welcher Form ausgegeben werden sollen (Output-Daten). Anschließend wird das Problem genauer analysiert. Ist es komplex, so wird es in einzelne, leichter lösbare Teilprobleme zerlegt und diese werden weiter zerlegt, bis man zu elementaren Problemen gelangt. Ein solches Vorgehen folgt der *Top-Down-Methode*. Dies umfasst nicht nur die sorgfältige Zerlegung eines Problems in Teilprobleme auf verschiedenen Stufen, sondern auch den Nachweis, dass durch das Zusammenwirken aller Teillösungen das Problem korrekt gelöst wird. Die Zerlegung der Problemlösung in Lösungen für Teilprobleme nennt man auch *Modularisierung*.

Die Vorteile der Top-Down-Methode sind:

- Teilprobleme sind leichter zu lösen und diese Lösungen sind leichter auf Korrektheit zu testen als das Ausgangsproblem.
- Module (Programme zur Lösung der Teilprobleme) können mehrfach Verwendung finden; man kann eine Bibliothek von Modulen anlegen.

- Änderungen können leichter durchgeführt werden. Korrekturen an einzelnen Modulen sind in ihren Wirkungen leichter zu überschauen als Korrekturen in großen Programmen.
- Die modulare Vorgehensweise erleichtert die Einschätzung des Arbeitsfortschrittes. Auf Basis der erstellten Module lässt sich das Ausmaß der erledigten und der noch ausstehenden Arbeit abschätzen.
- Die Zerlegung in Teilprobleme eignet sich besonders für Teamarbeit.

Der letztgenannte Punkt weist jedoch auch auf Probleme bei der Modularisierung hin. Um Schwierigkeiten beim Zusammensetzen verschiedener Module zu einem Programm zu vermeiden, muss eine sorgfältige Planung und Dokumentation der Module und ihrer Schnittstellen erfolgen.

Erst auf der Ebene der Teillösungen lässt sich die Qualität der Zerlegung einschätzen. Bei der Problemanalyse wird für notwendige Korrekturen oftmals der Weg zurück zum Ausgangsproblem eingeschlagen. Dies nennt man *Bottom-Up-Methode*. Dabei geht man von Teilproblemen aus und setzt diese zu einem Gesamtproblem zusammen. Die Problemanalyse ist also ein Wechselspiel der Top-Down- und der Bottom-Up-Vorgehensweise.

EIN BEISPIEL: BERECHNUNG DER MEHRWERTSTEUER

Zur Darstellung des Vorgehens bei der Entwicklung einer computergestützten Problemlösung soll als Beispiel die Berechnung der ausgewiesenen Mehrwertsteuer auf einer Rechnung dienen.

Gegeben (Input-Daten):	*n* Artikel mit Netto-Preisen, Mehrwertsteuersätze
Gesucht (Output-Daten):	Summe der abzuführenden Mehrwertsteuer (MWST)

Bevor wir mit der Problemlösung beginnen, sollten wir uns darüber klar sein, welches Problem wir lösen wollen. Der Mehrwertsteuersatz ist ein Prozentsatz, der auf Nettopreise erhoben wird. Je nach Produkt kommt ein anderer Prozentsatz in Frage. Die abzuführende MWST für einen Artikel *x* berechnet sich nach der Funktion MWST (*x*) = Netto-Preis (*x*) * MWST-Satz (*x*).

Wir gehen nach der Top-Down-Methode vor und zerlegen unser Problem in Teilprobleme, die in einer von uns anzugebenden Reihenfolge des Zusammenwirkens das Problem lösen. Wir haben zwei Möglichkeiten die abzuführende MWST zu berechnen:

(A) durch Berechnung der MWST für jeden Artikel (Teilproblem 1.1) und anschließender Addition (Teilproblem 1.2) oder

(B) durch Addition aller Netto-Preise (Teilproblem 2.1) und anschließender Berechnung der MWST (Teilproblem 2.2).

Die Möglichkeit (B) ist ausgeschlossen, wenn für die Artikel unterschiedliche Mehrwertsteuersätze anfallen.

Für (A) ergibt sich folgende Anweisungen:

Schritt 1 Lies einzelne Netto-Preise ein;

Schritt 2 Multipliziere jeden Netto-Preis mit dem entprechenden MWST-Satz;

Schritt 3 Addiere alle Ergebnisse von Schritt 2;

Schritt 4 Gib das Ergebnis von Schritt 3 aus;

Wir können (B) wie folgt aufschreiben:

Schritt 1 Lies einzelne Netto-Preise ein;

Schritt 2 Addiere alle Netto-Preise;

Schritt 3 Multipliziere das Ergebnis von Schritt 2 mit dem einheitlichen MWST-Satz;

Schritt 4 Gib das Ergebnis von Schritt 3 aus;

Beim Vergleich der Alternativen (A) und (B) fällt auf, dass die Anzahl der auszuführenden Schritte zwar gleich, jedoch der Aufwand, der sich dahinter verbirgt, unterschiedlich ist. Der Unterschied liegt in den Schritten 2 und 3. Bei (A) müssen n Multiplikationen in Schritt 2 und n Additionen in Schritt 3 ausgeführt werden; dies ergibt $2n$ Operationen. Bei (B) sind dies n Additionen in Schritt 2 und eine Multiplikation in Schritt 3; dies ergibt $n+1$ Operationen.

Für die Korrektheit des Programms müssen folgende Eigenschaften erfüllt sein:

1. Endlichkeit: beide Programme terminieren nach $3n+1$ Operationen
2. Bestimmtheit: die auszuführenden Anweisungen müssen eindeutig sein. Um die Mehrdeutigkeit der Umgangssprache zu vermeiden, werden formale (Programmier-) Sprachen verwendet.
3. Festlegung der Eingabebedingungen: Ein Programm hat keinen, einen oder mehrere Eingabewerte. (A) und (B) verarbeiten reelle Zahlen.
4. Festlegung der Ausgabebedingungen: Ein Programm hat einen oder mehrere Ausgabewerte, die in bestimmter Beziehung zu den Eingabewerten stehen. (A) und (B) erzeugen reelle Zahlen.

Weitere Anforderungen an Programme sind:

5. Effizienz: Die Effizienz eines Programms hängt von den zur Ausführung benötigten Ressourcen wie Speicher und Rechenzeit ab.
6. Universalität: Ein Programm, das zur Lösung eines Problems entworfen wurde, muss für alle zulässigen Input-Daten das Problem (korrekt) lösen.

Zur Beurteilung von Bedingung 5 kann z.B. die Anzahl der benötigten Verarbeitungsschritte herangezogen werden. Um zu sehen, was für die Erfüllung von Bedingung 2 notwendig ist, betrachten wir folgendes Kochrezept.

Zutaten (Eingabe): 35 Gramm Butter, 35 Gramm Mehl, 1/2 Ei, Salz, Muskat, Fleischbrühe.

- Schritt 1: Man nehme das Ei und vermische es gemeinsam mit Salz und Muskat mit der schaumig gerührten Butter durch 5-minütiges Rühren.
- Schritt 2: Man mische Mehl langsam darunter.
- Schritt 3: Wenn Sie eine Spritze haben, dann formen Sie damit kleine Häufchen der Masse, ansonsten nehmen Sie dafür einen Kaffeelöffel.
- Schritt 4: Lassen Sie diese Häufchen erkalten.
- Schritt 5: Geben Sie die Häufchen in die kochende Brühe und lassen Sie diese 2 Minuten kochen.

Obgleich diese Anweisungen Bedingungen 1, 3 und 4 erfüllen, ist insbesondere Bedingung 2 nicht erfüllt. Wir würden - obwohl obiges Rezept wesentliche algorithmische Elemente enthält - dieses mangels Bestimmtheit nicht als Algorithmus bezeichnen.

Ein Programm besteht aus

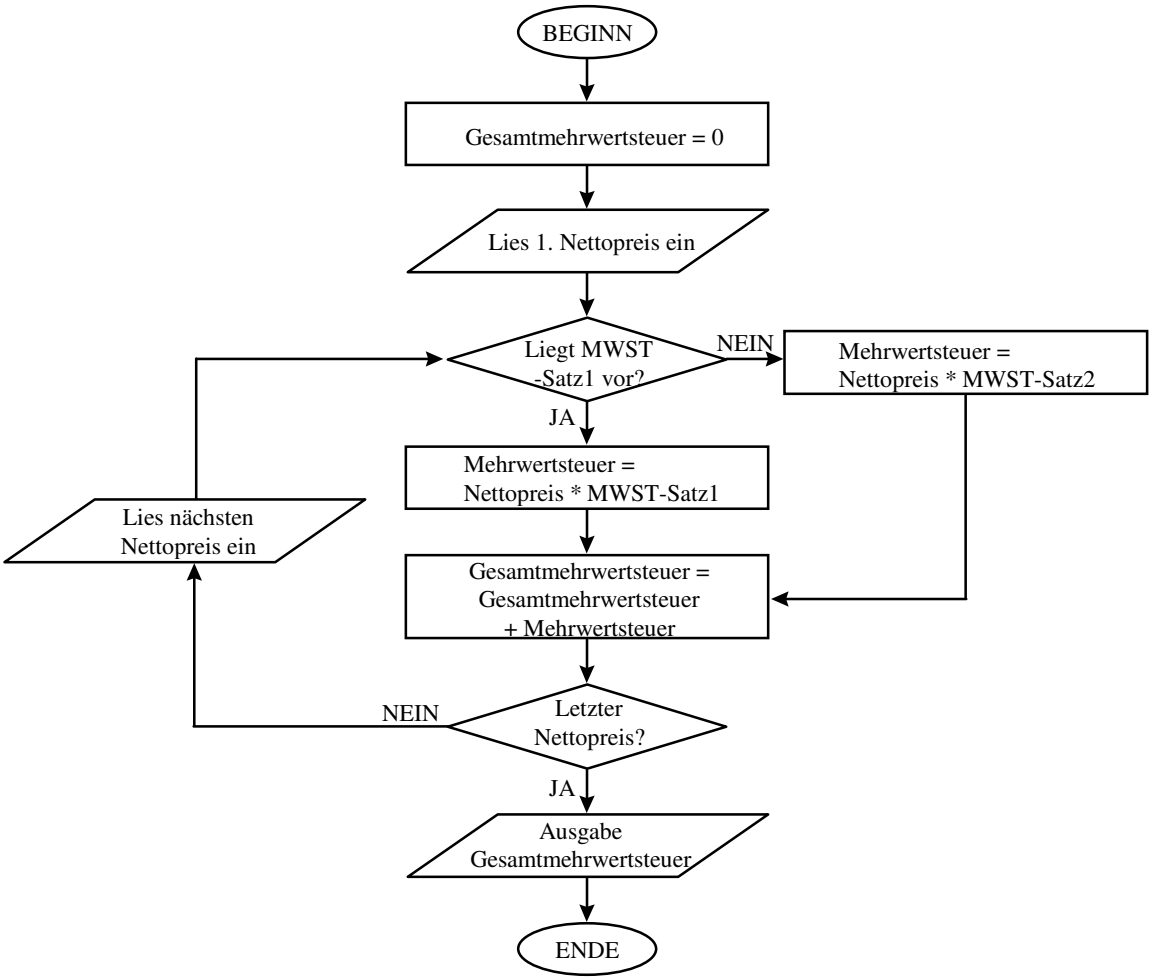
- Anweisungen, die sich auf die Manipulation von Daten beziehen und
- Festlegungen, in welcher Reihenfolge und unter welchen Bedingungen, diese Anweisungen ausgeführt werden.

Wir nennen die durch letztere Festlegungen bestimmten Abläufe die Kontrollstruktur eines Programms.

Zur semi-formalen Umsetzung der verbalen Beschreibung der Alternativen (A) und (B) lassen sich Programmablaufpläne (PAP) und Struktogramme benutzen. Diese sind für beide Möglichkeiten der Problemlösung in den Abbildungen 4-1 und 4-2 dargestellt.

Auf die Phasen (4) Programmierung der semi-formalen Formulierung und (5) Abnahme und Einsatz des Programms wird hier nicht weiter eingegangen.

PROGRAMM A



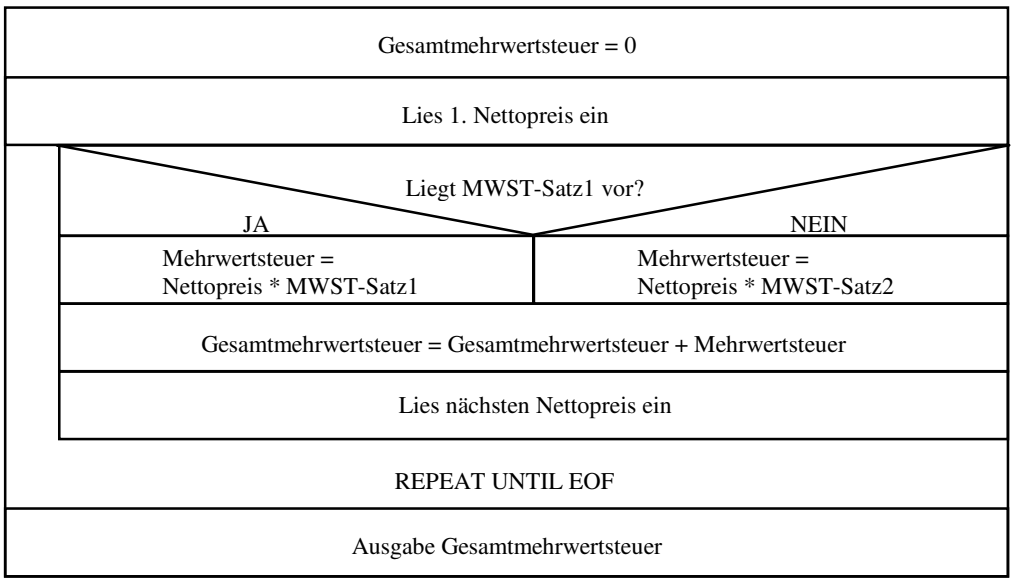
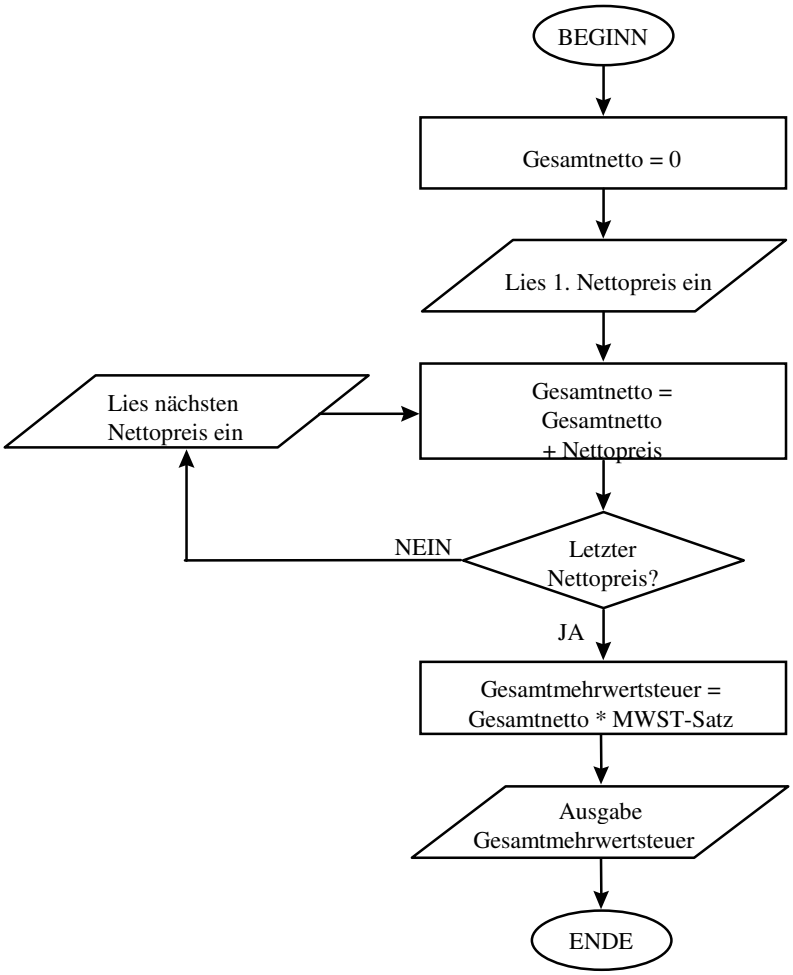


Abb. 4-1: *PAP und Struktogramm für Möglichkeit (A)*

PROGRAMM B



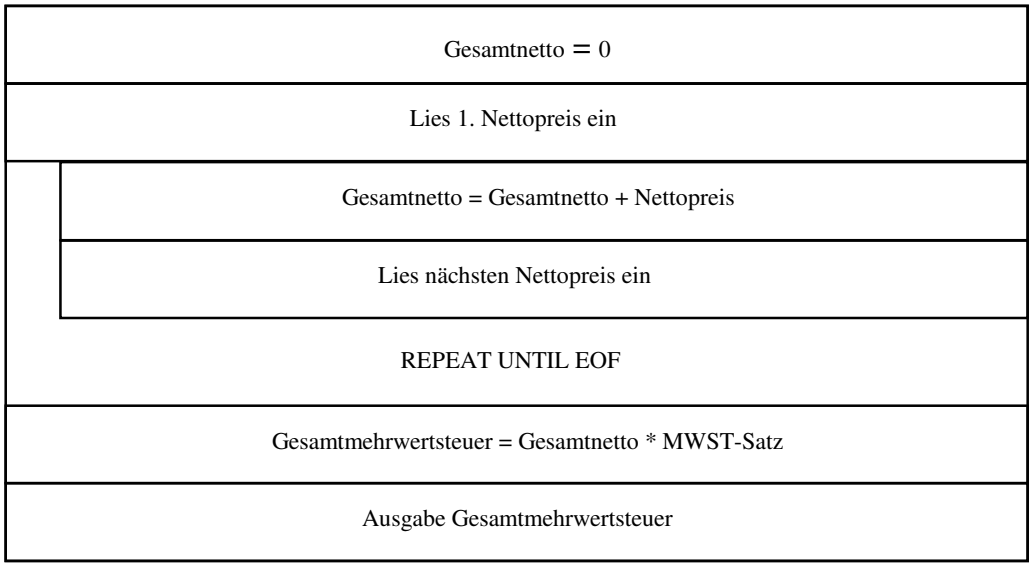


Abb. 4-2: *PAP und Struktogramm für Möglichkeit (B)*

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

ALU	Arithmetic Logical Unit
AR	Arbeitsrechner
AS	Argumentwertspeicher
ASCII	American Standard Code for Information Interchange
AT	Ausgabeteil
ATM	Asynchronous Transfer Mode
B	Byte
BM	(einfache) Basismaschine
BRS	Busrechnersystem
CCIR	Comité Consultatif International des Radiocommunications
CCITT	Comité Consultatif International Télégraphique et Téléfonique
CEN	Comité Européen de Normalisation
CEPT	Conférence Européenne des Administrations des Postes et des Télécommunications
CGI	Computer Graphics Interface
CISC	Complex Instruction Set Computer
CLI	Call Level Interface
CORBA	Common Object Request Broker Architecture
CSMA / CD	Carrier Sense Multiple Access / Collision Detection
CT	Computertechnik
CUA	Common User Access
DBVS	Datenbank-Verwaltungssystem
DIN	Deutsches Institut für Normung
E / A	Eingabe / Ausgabe
eASCII	extended ASCII
EBCDIC	Extended Binary Coded Decimal Interchange Code

ECMA	European Computer Manufacturers Association
EOF	End of File
EOL	End of Line
ET	Eingabeteil
FIFO	First In First Out
FLOPS	Floating Point Operations per Second
FS	Funktionswertspeicher
FTP	File Transfer Protocol
FTZ	Fernmeldetechnisches Zentralamt
GB	Giga Byte
HS	Hauptspeicher
HTML	Hypertext Markup Lamguage
HTTP	Hypertext Transfer Protocols
IDL	Interface Definition Language
IP	Internet Protocol
IPC	Interprocess Communication
IRC	Internet Relay Chat
IS	Informationssystem(e)
ISO	International Standardization Organization
IuK-Systeme	Informations- und Kommunikationssysteme
KB	Kilo Byte
KT	Kommunikationstechnik
LAN	Local Area Networks
LIFO	Last In First Out
MB	Mega Byte
MDI	Multiple Document Interface
MIMD	Multiple Instruction - Multiple Data
MIPS	Million Instructions per Second
MISD	Multiple Instruction - Single Data

MIT	Massachusetts Institute of Technology
MMK	Mensch-Maschine-Kommunikation
NS	Numerischer Schlüssel
NWBS	Netzwerk-Betriebssystem
ODBC	Open Database Connectivity
OMG	Object Management Group
OpCode	Operationscode
ORB	Object Request Broker
OS	Operandenspeicher
OSI	Open Systems Interconnection
OT	Operationsteil
PAP	Programmablaufplan
PBM	Programmbesteuerte Maschine
POS	Point of Sale
PRS	Parallelrechnersysteme
PS	Programmspeicher
RISC	Reduced Instruction Set Computer
RN	Rechnernetz(e)
RVS	Rechnerverbundsystem
RW	Rechenwerk(e)
SAA	Systems Application Architecture
SIMD	Single Instruction - Multiple Data
SISD	Single Instruction - Single Data
SW	Steuerwerk
TB	Tera Byte
TCP	Transmission Control Protocol
UIMS	User-Interface-Management-Systems
URL	Unified Resource Locator
URM	Universalrechenmaschine

VR	Vermittlungsrechner
VS	Vergleichsspeicher
WAIS	Wide Area Information Sever / Service
WAN	Wide Area Networks
WWW	World Wide Web
X	X-Window-System

LITERATUR

[ABCW02] Appelrath, H.-J., Boles, D., Claus, V., Wegener, I., Starthilfe Informatik, Teubner, 2002

[FS01] Ferstl, O.K., Sinz, E.J., Grundlagen der Wirtschaftsinformatik, Oldenbourg 2001

[Jan98] Janko, W.H., Informationswirtschaft I, Springer 1998

[MBKPS01] Mertens, P., Bodendorf, F., König, W., Picot, A., Schumann, M., Grundzüge der Wirtschaftsinformatik, Springer 2001

[OHE96] Orfali, R., Harkey, D., Edwards, J., The Essential Distributed Objects Survival Guide, Wiley 1996

[Rah94] Rahm, E., Mehrrechner-Datenbanksysteme, Addison-Wesley 1994

[Sch99] Schmidt, G., Informationsmanagement, Springer 1999

[SSN90] Steinmetz, R., Schmutz, H., Nehmer, J., Netz-Betriebssystem / verteiltes Betriebssystem, Informatik-Spektrum 13 (1), 1990, 38-39