Vom BIT zum PROGRAMM

Grundlagen der Computertechnologie © Herbert Paukert (Version 4.1)

Das vorliegende Skriptum ist eine Einführung in die Computertechnologie. Es soll dem Leser einen Einblick in das Innenleben von Computersystemen ermöglichen. Das Skriptum ist keine Praxisanleitung für die Arbeit am Computer, sondern eine Beschreibung der theoretischen Grundlagen von Computersystemen. Der Autor ist der Meinung, dass zum Verstehen der Funktionsweise des Computers ein entsprechendes Hintergrundwissen erworben werden sollte. Genau dieses Hintergrundwissen soll das vorliegende Skriptum vermitteln, ohne jedoch auf die komplizierten technischen Details genauer einzugehen.

1 Die Informationsverarbeitung 2

1.1 Allgemeine Begriffsbestimmungen 2
1.2 Interne Datenformate 4
1.3 Die Darstellung negativer Zahlen 7

2 Die Schaltkreise des Computers 8

2.1 Grundbegriffe der Aussagenlogik 8
2.2 Logische Schaltungen 10

3 Der Aufbau des Computers 18

3.1 Ein einfaches Computermodell 18
3.2 Die Geräteeinheiten des Systems 19
3.3 Die Informationsübertragung 20
3.4 Die Informationsspeicherung 21
3.5 Ein erweitertes Computermodell 22
3.6 Das Basis-Input-Output-System (BIOS) 24
3.7 Die Interruptverwaltung 24
3.8 Schnittstellen, die Tore zur Außenwelt 26
3.9 Ein komplettes Computersystem 27

4 Die Grundfunktionen des Betriebssystems 29

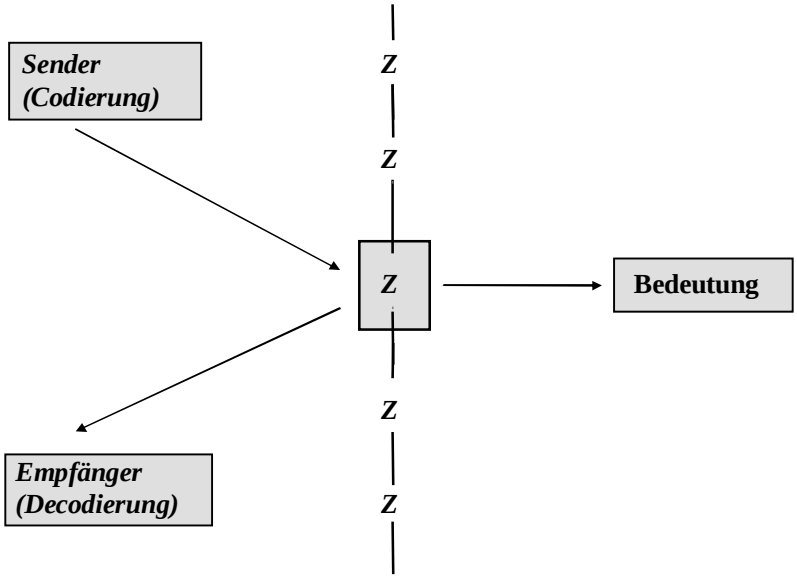
4.1 Die fünf Hauptaufgaben des Betriebssystems 29
4.2 Das Windows Application Programming Interface 32

5 Die Programmierung des Computers 33

1 DIE INFORMATIONSVERARBEITUNG

1.1 Allgemeine Begriffsbestimmungen

Wir leben heute in einer so genannten Informationsgesellschaft. **Information** besteht aus Nachrichten, welche ein Sender an einen Empfänger übermittelt. Eine Nachricht ihrerseits besteht meistens aus einer geordneten Aufeinanderfolge von Zeichen, beispielsweise den Buchstaben eines Alphabets. Dabei ist die Abfolge der sprachlichen Zeichen durch die Grammatik bzw. die Syntaktik geregelt. Bestimmten Zeichengruppen (Z) werden bestimmte Bedeutungen zugeordnet. Das sind dann die Wörter, deren Bedeutungen die semantische Ebene einer Sprache bilden. Neben Grammatik und Semantik ist die Ebene des physischen Datenaustausches ein wesentlicher Bereich jeder Information. Hier werden die Zeichen vom Sender verschlüsselt **(Codierung),** auf ein Medium übertragen und schließlich vom Empfänger entschlüsselt **(Decodierung).**



Computer sind Maschinen, welche der schnellen Verarbeitung großer Informationsmengen dienen. Im Grunde sind sie nichts anderes als ein Netzwerk von miteinander verdrahteten elektronischen **Schaltelementen** (z.B. Transistoren). Das sind Bauelemente der Halbleitertechnik, die nur zwei Schaltzustände annehmen können, die als **binäre Signale** (BIT = **BI**nary digi**T**) bezeichnet werden.

„0“ = Schalter aus: Es fließt KEIN elektrischer Strom.
„1“ = Schalter ein: Es fließt elektrischer Strom.

Diese elektronischen Schaltelemente werden zu integrierten Schaltkreisen verdrahtet, welche entweder der Speicherung binärer Signale (passive Bausteine) oder der Verarbeitung binärer Signale (aktive Bausteine) dienen. Auf Grund dieser technischen Gegebenheit muss somit jede Information in einen Code verschlüsselt werden, der nur zwei Grundzeichen aufweist (binäres bzw. duales Zahlensystem).

Datenspeicherung heißt, dass die Zustände (0 oder 1) von passiven Bauelementen bleibend so abgeändert werden, dass sie der Folge von eingegebenen binären Signalen entsprechen (z.B. bei der Abspeicherung von Textzeichen). Üblicherweise werden acht binäre Signalspeicher, die somit genau **acht Bit** aufnehmen können, zu einer Speicherzelle (**ein Byte**) zusammengefasst. **Datenverarbeitung** heißt, dass in aktiven Bauelementen eine Folge von eingegebenen binären Signalen in gewünschter Weise zu einer neuen Folge von binären Signalen umgeformt und ausgegeben wird (z.B. bei der Addition von zwei Zahlen).

Gebräuchliche Maße für Informationsmengen:

1 BIT	=	Informationsgehalt eines binären Signals (0, 1)
1 BYTE	=	8 Bit (1 Byte = Standard-Einheit)
1 KILOBYTE	=	2^{10} = 1 024 Byte (1 KB)
1 MEGABYTE	=	2^{20} = 1 048 576 Byte (1 MB)
1 GIGABYTE	=	2^{30} = 1 073 741 824 Byte (1 GB)

Einfache und direkte Umwandlung von Zahlen:

Dezimalsystem	Binärsystem	Dezimalsystem	Binärsystem
0	0	15	1111
2^0 = 1	1	2^4 = 16	10000
2^1 = 2	10	17	10001
3	11	18	10010
2^2 = 4	100	19	10011
5	101	20	10100
6	110	....	....
7	111	254	11111110
2^3 = 8	1000	255	11111111
9	1001	2^8 = 256	100000000
....	.....	257	100000001

Die Umwandlung einer Zahl vom 10er-System in das 2er-System erfolgt einfach durch Zerlegung der Zahl in eine Summe von 2er-Potenzen:

Z.B. $25 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 11001$.

Je nach der Art der Information kann zwischen **Daten** und **Befehlen** unterschieden werden. So enthält beispielsweise ein Datenbanksystem einerseits Personaldaten einer Personengruppe und andererseits auch Befehle zur Verwaltung dieser Daten (u.a. zum Durchsuchen oder Sortieren des Datenbestandes). Sowohl Daten als auch Befehle müssen binär codiert werden, damit sie in den Schaltelementen gespeichert werden können. Das Innere eines Computers stellt letztendlich nichts anderes als eine riesige, aber wohl strukturierte Menge von 0 und 1 dar. Die einfachste Strukturierung besteht darin, dass jeweils acht Bit zu einem Byte zusammengefasst werden.

Unter einer **Datei** versteht man eine Menge von zusammengehörigen Bytes, welche auf einem Datenträger (z.B. Festplatte) unter einem symbolischen Namen abgespeichert werden kann. Wenn die Bytes Daten repräsentieren, dann handelt es sich um eine **Datendatei** (z.B. Textdateien). Wenn die Bytes hingegen Befehle repräsentieren, dann handelt es sich um eine ausführbare **Programmdatei** (z.B. Computerspiele).

In der elektronischen Datenverarbeitung (EDV) gewinnen zwei Begriffe immer mehr an Bedeutung, nämlich die **Objekte** und die **Ereignisse.** Ein wichtiger Zweck aller Information ist die Beschreibung der Welt. Betrachten wir als Beispiel das Weltobjekt „TISCH“. Dieses Objekt kann durch bestimmte Eigenschaften (Tischplatte, Tischbeine usw.) beschrieben werden. Außerdem gibt es aber auch bestimmte Operationen, die mit dem Objekt ausgeführt werden können (Tisch verschieben, Tisch decken usw.). Das Objekt „TISCH“ kann daher einerseits durch seine Datenmerkmale und andererseits durch die zulässigen Operationen bzw. Funktionen beschrieben werden. Überträgt man dieses Modell auf die EDV, dann versteht man unter einem **Objekt** eine Zusammenfassung von Datenmerkmalen und zugehörigen Bearbeitungsmethoden. So enthält die grafische Bedienungsoberfläche moderner Programme mehrere typische Objekte, wie beispielsweise Fenster (Windows) und Schaltflächen für das Bedienungsgerät „Maus“. Mit diesen Objekten können vom Benutzer mithilfe der Maus bestimmte Operationen durchgeführt werden (Verschieben, Vergrößern, Anklicken usw.).

Neben dem Begriff des **Objektes** ist auch der Begriff des **Ereignisses** in der modernen EDV sehr wichtig. Ereignisse sind Vorfälle (Events), welche während der Arbeit im Computersystem auftreten, beispielsweise ein Klick mit der Maus oder ein Druck auf eine Taste der Tastatur.

Ereignisse werden vor allem durch Betätigung von Eingabegeräten ausgelöst und müssen durch das laufende Computerprogramm in passender Weise behandelt, d.h. richtig beantwortet werden. Moderne Computerprogramme sind **objektorientiert** und **ereignisgesteuert;** das gilt besonders für die Betriebssysteme (z.B. WINDOWS) und die Anwenderprogramme (z.B. WORD).

1.2 Interne Datenformate

Grundsätzlich muss man zwischen zwei Datenformaten unterscheiden, entweder numerische Daten oder Textdaten. Dabei wird eine Folge von nebeneinander liegenden Bytes verschieden interpretiert, entweder als Folge von Ziffern, welche eine Zahl darstellen (Number), oder als Kette von Zeichen (String).

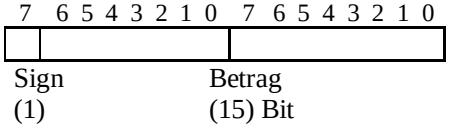
NUMERISCHE DATEN

Die Bytes werden als Zahlen interpretiert. Man unterscheidet hier wiederum zwischen GANZEN Zahlen (Integer) und DEZIMAL-Zahlen (Real).

Integer-Zahlen (Ganze Zahlen)

Diese können 1, 2 oder mehrere Byte lang sein. Grundsätzlich wird dabei immer das höchste Bit (Most Significant Bit, MSB) für das Vorzeichen reserviert (Signum SIGN: + = 0, – = 1). Der Betrag ist in den nachfolgenden Bits verschlüsselt. Die 8 Bit innerhalb eines Byte werden von der höchsten bis zur niedersten Position mit den Ziffern 7, 6, ... 1, 0 fortlaufend nummeriert.

Beispiel: 2-Byte-Zahlen



Betrag von X = $|X| < 2^{15} = 32768$

Beispiel: 300 = 1*256 + 44
= 00000001 00101100

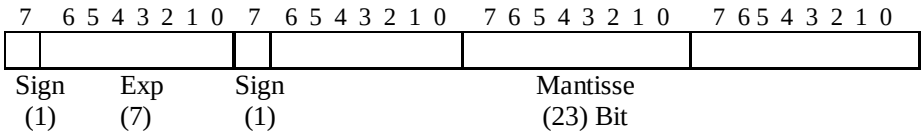
Real-Zahlen (Gleitkomma-Zahlen)

Diese werden in einer normalisierten Exponentialform dargestellt:

$X = m * 10^a$ m = Mantisse, $1 \le m < 10$
a = Exponent

Beispiel: $5716.82 = 5.71682 * 10^3$

Exponent und Mantisse werden getrennt in verschiedenen Bytes gespeichert, z.B. in 1 und 3 Byte. Durch den Exponenten ist die Größenordnung, durch die Mantisse die Genauigkeit der Zahl bestimmt. Meistens wird nicht 10, sondern 2 als Basis genommen.



Größenordnung (Exponent): $|a| < 2^7 = 128$, $10^h = 2^{128}$ h = 38
Genauigkeit (Mantisse): $|m| < 2^{23}$, $10^d = 2^{23}$ d = 7

Somit können 4-Byte-Dezimalzahlen mit einer Genauigkeit von 7 Nachkommastellen und bis zu einer Größenordnung von 10^{38} dargestellt werden.

STRING-DATEN

Jeder Text, der in den Computer über die Tastatur eingegeben wird, besteht aus einer Kette von Zeichen (String). Jedes einzelne Zeichen (Charakter) wird durch genau ein Byte, also einer Folge von acht Bit, verschlüsselt.

Z.B.: „**A**“ = 65 = 01000001
„+“ = 43 = 00101011

Weil ein Byte aus acht Bit besteht, können $2^8 = 256$ verschiedene Bitmuster in einem Byte dargestellt werden (00000000, 00000001, 00000010, ..., 11111111), also genau 256 verschiedene Zeichen. Das ist aber mehr als ausreichend für den Grundzeichen-Vorrat einer Sprache.

Z.B.: „**ADE**“ = 65 68 69 = 01000001 01000100 01000101
Das Wort ADE wird somit mittels 3*8 = 24 Bit dargestellt.

Die Codierung eines Textzeichens durch ein Bitmuster erfolgt nach den Vorschriften des so genannten ASCII-Codes (American Standard Code for Information Interchange). Als Hinweis, dass es sich bei einer Zahl um einen ASCII-Code handelt, wird oft das Zeichen # vorangestellt.

0 bis 31:	Code von speziellen Steuersignalen für Drucker und Bildschirm
32 bis 127:	Code von Buchstaben, Ziffern und Sonderzeichen
128 bis 255:	Code von bestimmten Grafikzeichen (z.B. IBM-Zeichensatz)

In Textdateien schließen die Textzeilen mit den Steuerzeichen CARRIAGE RETURN (CR, #13) und LINE FEED (LF, #10) ab, wodurch ein Zeilenvorschub bewirkt wird. Ein Seitenvorschub wird mit FORM FEED (FF, #12) gekennzeichnet. Das Ende einer Textdatei kann durch END OF FILE (EOF, #26) markiert werden. Bestimmte Steuerzeichen beziehen sich auch auf den Positionszeiger am Bildschirm **(Cursor),** beispielsweise bewirkt ein so genanntes BACK SPACE (BS, #8) einen Rückwärtsschritt des Cursors um eine Spalten-Position und die Löschung des dort befindlichen Zeichens.

Außerdem werden von Textverarbeitungsprogrammen in die Texte druckerspezifische Steuerzeichen gesetzt, welche den Ausdruck des nachfolgenden Textes gestalten (besondere Schriftauszeichnungen). Zumeist werden diese Steuercodes mit dem ESC-Signal (ASCII-Code 27) eingeleitet. Einige Beispiele solcher Escape-Sequenzen für EPSON/IBM-Drucker sind:

Unterstreichen EIN:	ESC - 1	Unterstreichen AUS:	ESC - 0
Fettschrift **EIN**:	ESC E	Fettschrift AUS:	ESC F
Hochstellen ^{EIN}:	ESC S 0	Hochstellen AUS:	ESC T
Tiefstellen $_{EIN}$:	ESC S 1	Tiefstellen AUS:	ESC T

Vereinfachte Darstellung im Hexadezimal-System

Mithilfe von vier Bit (Halbbyte) können $2^4 = 16$ Bitmuster erzeugt werden. Wenn man nun ein Zahlensystem mit genau 16 verschiedenen Grundziffern (0, 1, 2, 3, 4, ... 9, A, B, C, D, E, F) verwendet, dann kann man den Inhalt von einem Halbbyte nur durch eine einzige solche Grundziffer darstellen. Der Speicherinhalt von einem ganzen Byte wird demnach mithilfe von zwei hexadezimalen Ziffern dargestellt (z.B. 1010 0111 = A7h = 167). Hexadezimale Zahlen werden mit nachgestelltem **h** gekennzeichnet. Die Umwandlung einer Zahl vom 10er-System in das 16er-System erfolgt einfach durch Zerlegung der Zahl in eine Summe von 16er-Potenzen:

z.B. $24\,000 = 5 * 16^3 + 13 * 16^2 + 12*16^1 + 0 * 16^0 = 5DC0h.$

ASCII-Tabelle:

0		1		2		3		4		5		6		7	
8	BS	9		10	LF	11		12	FF	13	CR	14		15	
16		17		18		19		20		21		22		23	
24		25		26	EOF	27	ESC	28		29		30		31	
32		33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
58	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	
128		129	ü	130	é	131	â	132	ä	133	à	134	å	135	ç
136	ê	137	ë	138	è	139	ï	140	î	141	ì	142	Ä	143	Å
144	É	145	æ	146	Æ	147	ô	148	ö	149	ò	150	û	151	ù
152	ÿ	153	Ö	154	Ü	155	¢	156	£	157	¥	158	₧	159	ƒ
160	á	161	í	162	ó	163	ú	164	ñ	165	Ñ	166	ª	167	º
168	¿	169	⌐	170	¬	171	½	172	¼	173	¡	174	«	175	»
176	░	177	▒	178	▓	179	│	180	┤	181	╡	182	╢	183	╖
184	╕	185	╣	186	║	187	╗	188	╝	189	╜	190	╛	191	┐
192	└	193	┴	194	┬	195	├	196	─	197	┼	198	╞	199	╟
200	╚	201	╔	202	╩	203	╦	204	╠	205	═	206	╬	207	╧
208	╨	209	╤	210	╥	211	╙	212	╘	213	╒	214	╓	215	╫
216	╪	217	┘	218	┌	219	█	220	▄	221	▌	222	▐	223	▀
224	α	225	ß	226	Γ	227	π	228	Σ	229	σ	230	µ	231	τ
232	Φ	233	Θ	234	Ω	235	δ	236	∞	237	ø	238	ε	239	∩
240	≡	241	±	242	≥	243	≤	244	⌠	245	⌡	246	÷	247	≈
248	°	249	∙	250	·	251	√	252	ⁿ	253	²	254	■	255	

Beispiel: Codierung des Strings „**Eva**“ mit der ASCII-Tabelle im 10er-, 16er- und 2er-System.

E = 69, v = 118, a = 97

$69 = 4 * 16^1 + 5 * 16^0 = 45h = \underset{4h}{\underline{0100}}\ \underset{5h}{\underline{0101}} = 1 * 2^6 + 1 * 2^2 + 1 * 2^0$

$118 = 7 * 16^1 + 6 * 16^0 = 76h = \underset{7h}{\underline{0111}}\ \underset{6h}{\underline{0110}} = 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 1 * 2^2 + 1 * 2^1$

$97 = 6 * 16^1 + 1 * 16^0 = 61h = \underset{6h}{\underline{0110}}\ \underset{1h}{\underline{0001}} = 1 * 2^6 + 1 * 2^5 + 1 * 2^0$

E= 69 = 45h = 01000101
v= 118 = 76h = 01110110
a= 97 = 61h = 01100001

Wichtiger Hinweis: Windows benutzt zur Darstellung von Textzeichen nicht den **ASCII**-Code, sondern den **ANSI**-Code (American National Standard Institute). Die beiden Systeme unterscheiden sich in einigen Codes. Dies kann zu Problemen führen, wenn man alte MSDOS-Texte in WINDOWS einliest. In heutiger Zeit kommt dem UniCode-Zeichensatz immer größere Bedeutung zu. In diesem Zeichensatz besteht ein Zeichen aus zwei Byte. Auf diese Weise lassen sich 65 536 Zeichen darstellen und somit können die vielfältigen Zeichen der verschiedenen Alphabete (z.B. lateinisches, griechisches, kyrillisches, arabisches) und Schriften (z.B. japanisch, chinesisch) in einem Zeichensatz zusammengefasst werden. Außerdem können mathematische, kaufmännische und technische Sonderzeichen im Unicode codiert werden. Die ersten 256 Zeichen des Unicode-Zeichensatzes stimmen mit jenen des ANSI-Zeichensatzes überein.

1.3 Die Darstellung negativer Zahlen

Die wichtigste arithmetische Schaltung im Computer-Rechenwerk ist die **Addier-Schaltung,** welche die Summe zweier positiver ganzer Zahlen berechnet. Wie werden aber negative Zahlen intern dargestellt und wie kann die Subtraktion realisiert werden? Die Lösung dieser beiden Fragen soll der Einfachheit wegen im Folgenden nur an Ein-Byte-Zahlen (8 Bit) erklärt werden, d.h. der Gültigkeitsbereich der Zahlen ist 0, 1, 2, 3, ... 253, 254, 255.

Um die Zahl x (z.B. 14 = 0000 1110 = 0Eh) zu negieren, werden zwei Schritte durchgeführt:

1. Schritt: Bildung des ***Einer***-Komplementes x1 von x. Dabei werden durch eine entsprechende **Komplement-Schaltung** im Rechenwerk 0 durch 1 und 1 durch 0 ersetzt.
x = 0000 1110 = 0Eh = 14
x1 = 1111 0001 = F1h = 241
Die Summe von x und x1 ergibt immer 255, also ist x1 = (255 - x).

2. Schritt: Bildung des ***Zweier***-Komplementes x2 von x, d.h. die Differenz x2 = (256 - x) .
Somit ist das *Zweier*-Komplement um 1 größer als das *Einer*-Komplement.
x2 = x1 + 1 = 1111 0001 + 1 = 1111 0010 = F2*h* = 242

Das *Zweier*-Komplement einer Zahl x wird nun als negative Zahl (-x) interpretiert. In der nachfolgenden Tabelle sind die Zahlenwerte aufgelistet:

x (positiv)	-x = 256 – x (negativ)
1 = 0000 0001 = 01h	255 = 1111 1111 = FFh = -1
2 = 0000 0010 = 02h	254 = 1111 1110 = FEh = -2
.....	.....
126 = 0111 1110 = 7Eh	130 = 1000 0010 = 82h = -126
127 = 0111 1111 = 7Fh	129 = 1000 0001 = 81h = -127

Wie man aus der Tabelle deutlich ersieht, ist bei allen negativen Zahlen das höchste Bit immer 1, bei positiven Zahlen hingegen ist es immer 0 (MSB, Most Significant Bit). Das MSB wird daher auch Vorzeichen-Bit genannt. Daraus folgt, dass 128 = 1000 0000 = 80h negativ interpretiert werden kann. So ergibt sich der Zahlenbereich –128, –127, –126, ... –2, –1, 0, 1, 2, ... 126, 127.

Der Grund für diese eigenartige Definition von negativen Zahlen als *Zweier*-Komplement liegt nun darin, dass dadurch die Subtraktion direkt auf die Addition zurückgeführt werden kann. Das soll wieder am Beispiel von Ein-Byte-Zahlen erläutert werden.

Beispiel: **20 – 14 = 20 + (256 – 14) – 256 = 6**

Diese Umformung zeigt, dass anstelle der Subtraktion die Addition des *Zweier*-Komplementes verwendet werden kann, wobei ein möglicher Überlauf in das (nicht vorhandene) neunte Bit ignoriert wird. Daraus ergibt sich folgendes Subtraktionsverfahren:

```
Minuend              :    0001 0100 (20)
Subtrahend           :  + 1111 0010 (256-14 = 242 = -14)
(im 2er-Komplement)     -----------
Summenbildung        :  1 0000 0110 (262)
Überlauf ignorieren  :    0000 0110 (262-256 = 6)
```

So wird die Subtraktion mithilfe von Komplementbildung und nachfolgender Addition erzeugt:

Y – X = Y + (256 – X) – 256

Komplement- und Subtraktionsverfahren gelten in gleicher Weise auch für Mehr-Byte-Zahlen.

2 DIE SCHALTKREISE DES COMPUTERS

2.1 Grundbegriffe der Aussagenlogik

Die Aussagenlogik analysiert die formale Struktur von Aussagenverknüpfungen (Satzgefügen). Um die formale Struktur einer Aussagenverknüpfung zu erhalten, werden die Aussagen als Ganzes durch Variable (a, b, c, ...) ersetzt. Für die Verknüpfungsoperationen werden feststehende Operationssymbole (die **Junktoren** „und“, „oder“, „wenn-dann“, „nicht“, ...) eingesetzt.

Beispiel: Immer wenn es regnet, dann ist es nass. Es ist nicht nass. Also regnet es nicht.

Wird die Folgerung (wenn-dann) durch das Symbol $\rightarrow$, die Konjunktion (und) durch $\wedge$, die Negation (nicht) durch $\neg$ gekennzeichnet, so lautet die vollständige Formalisierung:

Beispiel: $((a \rightarrow b) \wedge \neg b) \rightarrow \neg a$

Von jeder Aussage wird vorausgesetzt, dass sie entweder **wahr** (w) oder **falsch** (f) ist. Dies ist die Grundannahme der klassischen, zweiwertigen Logik. Für die Verknüpfung von Aussagen existieren bestimmte Operatoren (Junktoren**)**: Die Negation (nicht a, $\neg a$), die Konjunktion (a und b, $a \wedge b$), die Disjunktion (a oder b, $a \vee b$), die Antivalenz (entweder a oder b, $a \times b$), die Implikation (wenn a dann b, $a \rightarrow b$), die Äquivalenz (wenn a dann b, und umgekehrt, $a \leftrightarrow b$), die NOR-Verknüpfung (weder a noch b) und andere Aussagenverknüpfungen. Die Bedeutung der Junktoren wird durch ihre **Wahrheitstabellen** festgelegt. Diese Tabellen bestimmen den Wahrheitswert der Aussagenverknüpfung für die verschiedenen Belegungen der Einzelaussagen mit wahr und falsch. In den folgenden Diagrammen sind die Wahrheitstabellen für die wichtigsten Junktoren aufgelistet.

a	$\neg$ a
f	w
w	f

Die **NEGATION** (nicht a) ist dann wahr, wenn a falsch ist, und dann falsch, wenn a wahr ist.
Abkürzung: NOT, $\neg$

a	b	a $\wedge$ b
f	f	f
f	w	f
w	f	f
w	w	w

Die **KONJUNKTION** (a und b) ist nur dann wahr, wenn sowohl die eine als auch die andere Aussage wahr ist.
Abkürzung: AND, $\wedge$

a	b	a $\vee$ b
f	f	f
f	w	w
w	f	w
w	w	w

Die **DISJUNKTION** (a oder b) ist immer dann wahr, wenn mindestens eine der beiden Aussagen wahr ist.
Abkürzung: OR, $\vee$ (einschließendes ODER)

a	b	a $\times$ b
f	f	f
f	w	w
w	f	w
w	w	f

Die **ANTIVALENZ** (entweder a oder b) ist immer dann wahr, wenn die beiden Aussagen entgegengesetzte Wahrheitswerte haben. Abkürzung: XOR, $\times$ (ausschließendes ODER)

a	b	$a \rightarrow b$
f	f	w
f	w	w
w	f	f
w	w	w

Die **IMPLIKATION** (aus a folgt b) ist nur dann falsch, wenn die Prämisse a wahr und die Konklusion b falsch ist.
Abkürzung: IMP, $\rightarrow$

a	b	$a \leftrightarrow b$
f	f	w
f	w	f
w	f	f
w	w	w

Die **ÄQUIVALENZ** (a gleichwertig zu b) ist immer dann wahr, wenn die beiden Aussagen die gleichen Wahrheitswerte haben.
Abkürzung: EQU, $\leftrightarrow$

Die Gleichwertigkeit ($L \leftrightarrow R$ bzw. $L = R$) von zwei komplexen Aussagenverknüpfungen L und R kann folgendermaßen nachgewiesen werden. Man berechnet gemäß den Wahrheitstabellen für alle möglichen Belegungen der Teilaussagen mit den Werten wahr (w) und falsch (f) die Wahrheitswerte von den beiden Aussagenverknüpfungen L und R. Stimmen diese für alle Belegungen überein, d.h., haben L und R immer die gleichen Wahrheitswerte, dann sind L und R gleichwertig. Im Folgenden werden oft komplexe Aussagenverknüpfungen durch andere ersetzt. Der Beweis ihrer Gleichwertigkeit kann mit dieser Methode leicht durchgeführt werden.

Beispiel: Es soll die Gleichwertigkeit der beiden Ausdrücke $\neg(a \wedge b)$ und $(\neg a \vee \neg b)$ nachgewiesen werden, was am besten mithilfe einer Wertebelegungstabelle geschieht. Dabei werden für jede Wertebelegung der Einzelaussagen die Wahrheitswerte der zwei Aussagenverknüpfungen berechnet und sodann auf Übereinstimmung geprüft.

a	b	$\neg(a \wedge b)$	$(\neg a \vee \neg b)$
f	f	$\neg(f \wedge f) = \neg f = \boldsymbol{w}$	$(\neg f \vee \neg f) = (w \vee w) = \boldsymbol{w}$
f	w	$\neg(f \wedge w) = \neg f = \boldsymbol{w}$	$(\neg f \vee \neg w) = (w \vee f) = \boldsymbol{w}$
w	f	$\neg(w \wedge f) = \neg f = \boldsymbol{w}$	$(\neg w \vee \neg f) = (f \vee w) = \boldsymbol{w}$
w	w	$\neg(w \wedge w) = \neg w = \boldsymbol{f}$	$(\neg w \vee \neg w) = (f \vee f) = \boldsymbol{f}$

Eine besondere Klasse von Aussagenverknüpfungen sind jene, welche bei jeder Belegung ihrer Einzelaussagen mit wahr oder falsch immer den Wert wahr erhalten. Solche Aussagenverknüpfungen heißen logisch **allgemein gültig**. Man bezeichnet sie auch als **Tautologien** und sie stellen die universellen Spielregeln unseres formal-richtigen Denkens dar.

Beispiel: *(a) Immer wenn es regnet, dann ist es nass.*
(b) Es ist nicht nass.

(c) Daher hat es nicht geregnet.

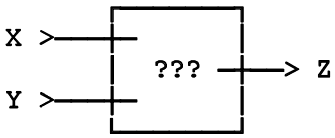
Diese Verküpfung $((\mathbf{a \rightarrow b}) \wedge \mathbf{\neg b}) \rightarrow \mathbf{\neg a}$ ist eine Tautologie, welche als **Kontrapositionsregel** bezeichnet wird. Die Allgemeingültigkeit kann mittels Wahrheitstabellen nachgewiesen werden:

a | b | ((a→b)∧ ¬b)→ ¬a

f | f | w, weil ((f→f)∧ ¬f)→ ¬f = ((w ∧ w)→ w) = (w → w) = w
f | w | w, weil ((f→w)∧ ¬w)→ ¬f = ((w ∧ f)→ w) = (f → w) = w
w | f | w, weil ((w→f)∧ ¬f)→ ¬w = ((f ∧ w)→ f) = (f → f) = w
w | w | w, weil ((w→w)∧ ¬w)→ ¬w = ((w ∧ f)→ f) = (f → f) = w

2.2 Logische Schaltungen

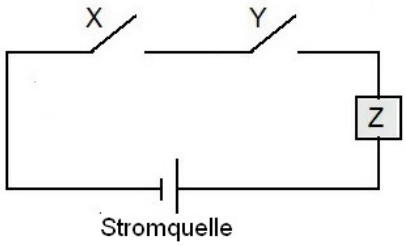
Ein Schaltkreis besteht aus miteinander verdrahteten elektronischen Schaltelementen (z.B. Transistoren), welche grundsätzlich nur zwei Zustände (0 oder 1) annehmen können. Jede Information kann durch eine Folge von binären Signalen (0 oder 1) verschlüsselt werden. Die **Eingänge** eines elektronischen Schaltkreises werden mit solchen binären Signalen beschickt. Im **Inneren** des Schaltkreises kommt es dann durch die jeweilige Verdrahtung der Schaltelemente zu einer entsprechenden Informationsverarbeitung, sodass am **Ausgang** das gewünschte Signal entsteht. Dabei bedeutet „Signal 0“ keine elektrische Spannung (bzw. kein Strom fließt) und „Signal 1“ eine schwache positive Spannung (bzw. Strom fließt). In einem Schaltkreis werden Signalzustände an den Eingängen zu Signalzuständen an den Ausgängen umgeformt.



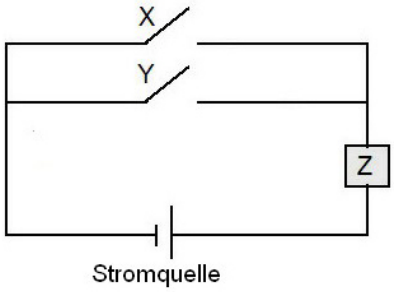
Blockbild eines Schaltkreises mit den zwei Eingängen X,Y und dem Ausgang Z.

Wird eine Aussagenverknüpfung durch eine entsprechende Verknüpfung binärer Schaltelemente technisch nachgebaut, dann spricht man von einem **logischen Schaltkreis** (Gatter). So kann das logische AND als serielle Schaltung und das logische OR als parallele Schaltung realisiert werden. In den nachfolgenden Abbildungen sind X und Y einfache mechanische Eingangsschalter und die Glühlampe Z stellt den Ausgang des Schaltkreises dar.

Die AND-Schaltung:



Die OR-Schaltung:



Offensichtlich liegt eine **Entsprechung** zwischen solchen Schaltungen und den Aussagenverbindungen unseres sprachlichen Denkens vor:

Schaltelement	=	einfache Aussage
Schaltkreis	=	Aussagenverknüpfung
Signal **0**	=	falsch **(f)**
Signal **1**	=	wahr **(w)**

Im Folgenden sollen die wichtigsten logischen Schaltungen für NOT, OR und AND nur mithilfe von NOR-Schaltungen aufgebaut werden. Danach wird der so genannte Halb-Addierer (HA) entwickelt. Dieser Schaltkreis besteht aus zwei Eingängen, die mit einstelligen binären Zahlen, also 0 oder 1, belegt werden. Am Ausgang soll dann die Summe dieser beiden Zahlen binär dargestellt werden.

Der nächste arithmetische Schaltkreis ist der Voll-Addierer (VA), welcher drei einstellige Binärzahlen addieren kann. Zum Abschluss wird ein paralleles 4-Bit-Addierwerk entwickelt, welches die Summe von zwei eingegebenen vierstelligen Binärzahlen ermittelt und ausgibt. Solche Schaltkreise befinden sich in der zentralen Recheneinheit von Computern.

Die Rückführung der logischen und arithmetischen Schaltungen auf nur einen Schaltkreis (z.B. auf die NOR-Schaltung) ist von großer praktischer Bedeutung, denn dadurch genügt die serienmäßige Produktion von einer einzigen Schaltung, um bei entsprechender Verdrahtung von tausenden solcher gleichartiger Bauelemente auf kleinster Fläche alle gewünschten Funktionen und Leistungen zu realisieren (IC = Integrated Circuit, auch Chip genannt).

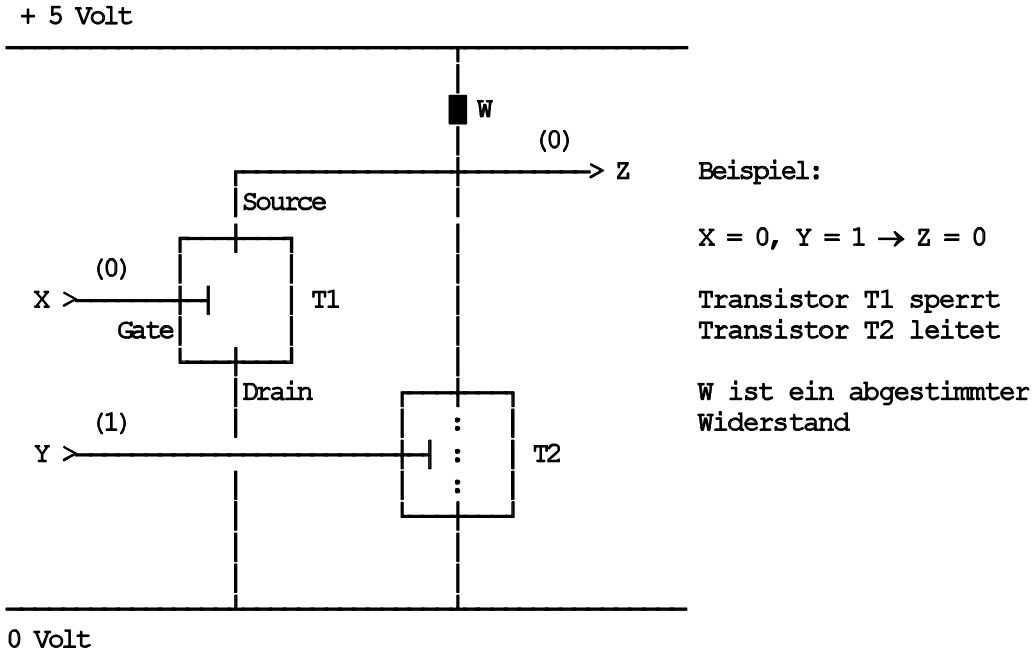
[1] Das NOR-Gatter

a	b	a NOR b
f	f	w
f	w	f
w	f	f
w	w	f

Wertetabelle der NOR-Schaltung
Dieser Schaltkreis mit 2 Eingängen und 1 Ausgang entspricht der logischen Aussagenverknüpfung WEDER – NOCH. Z = X NOR Y ist nur dann wahr, wenn weder X noch Y wahr sind.

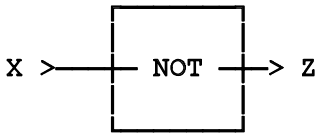
Das NOR-Gatter ist ein grundlegender Schaltkreis, mit dessen Hilfe alle logischen und arithmetischen Schaltungen aufgebaut werden können. Das NOR-Gatter besteht im Wesentlichen aus zwei parallel geschalteten Transistoren (T1, T2) und einem entsprechend abgestimmten Widerstand (W). Die Feldeffekt-Transistoren sind Halbleiter-Bauelemente, welche drei elektrische Kontaktstellen enthalten (Gate, Source und Drain). Die Gates der Transistoren werden mit den Eingangs-Signalen belegt. Nur wenn am Gate Signal 1, d.h. eine schwache positive Spannung, liegt, wird zwischen Source und Drain elektrischer Strom geleitet – sonst ist die Stromleitung gesperrt. Diese Transistoren wirken somit als einfache EIN-AUS-Schalter.

Wenn an beiden Eingängen X und Y das Signal 0 liegt, dann **sperren** beide Transistoren und am Ausgang Z des Schaltkreises liegt eine schwache positive Spannung, also das Signal 1. Wenn zumindest an einem Eingang das Signal 1 liegt, dann **leitet** zumindest ein Transistor. Weil der Transistor einen sehr kleinen Innenwiderstand darstellt, erfolgt der eigentliche Spannungsabfall am oberen Ausgangswiderstand W, so dass der Ausgang Z den Signalzustand 0 aufweist.



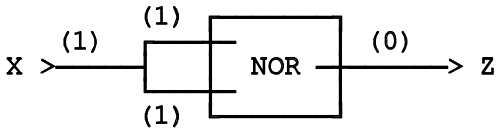
Schematischer Schaltplan des NOR-Gatters

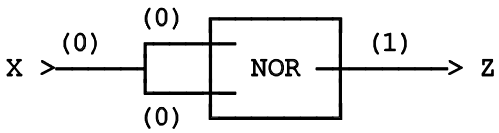
[2] Das NOT-Gatter



Dieser Schaltkreis mit 1 Eingang
und 1 Ausgang entspricht der logischen
Aussagenverknüpfung NOT.
Z = NOT X ist nur dann wahr, wenn
X falsch ist und umgekehrt.

Wertetabelle:	X	Z
0 = Falsch	0	1
1 = Wahr	1	0

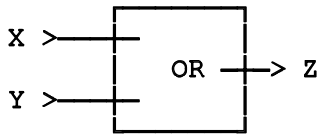




Schaltplan: Rückführung von NOT auf NOR: Z = NOT X = (X NOR X)

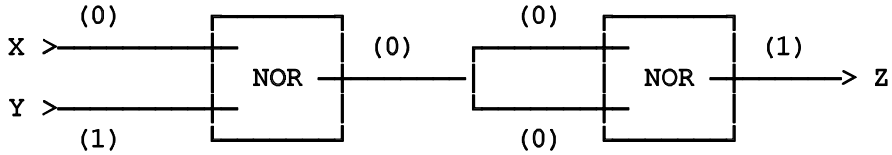
Als Übungsaufgabe kann die Richtigkeit der NOT-Reduktion bewiesen werden. Dazu ist es nur notwendig, die aussagenlogische Gleichwertigkeit der beiden Ausdrücke nachzuprüfen.

[3] Das OR-Gatter



Dieser Schaltkreis mit 2 Eingängen
und 1 Ausgang entspricht der logischen
Aussagenverknüpfung ODER.
Z = X OR Y ist nur dann wahr, wenn
mindestens eine Aussage X oder Y wahr ist.

Wertetabelle:	X	Y	Z
	0	0	0
0 = Falsch	0	1	1
1 = Wahr	1	0	1
	1	1	1

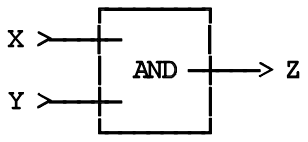


Schaltplan: Rückführung von OR auf NOR: Z = (X OR Y) = (X NOR Y) NOR (X NOR Y)

Beispiel: X = 0, Y = 1 → Z = 1

Als Übungsaufgabe kann die Richtigkeit der OR-Reduktion bewiesen werden. Dazu ist es nur notwendig, die aussagenlogische Gleichwertigkeit der beiden Ausdrücke nachzuprüfen.

[4] Das AND-Gatter

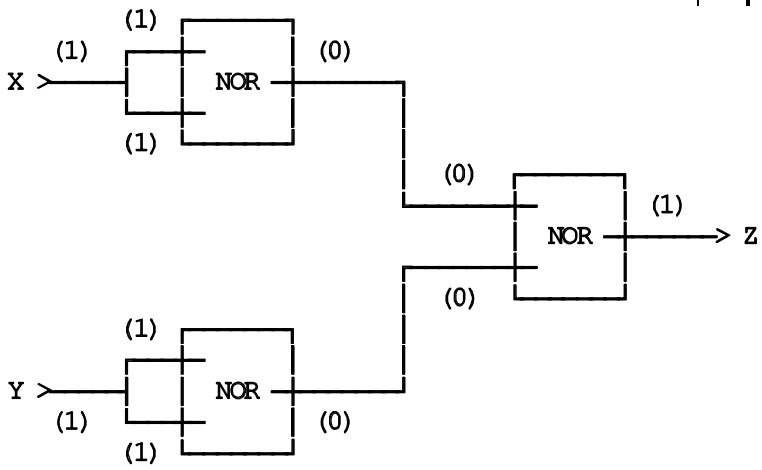


Dieser Schaltkreis mit 2 Eingängen
und 1 Ausgang entspricht der logischen
Aussagenverknüpfung UND.
Z = X AND Y ist nur dann wahr,
wenn sowohl X als auch Y wahr sind.

Wertetabelle:

0 = Falsch
1 = Wahr

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1



Schaltplan: Rückführung von AND auf NOR: Z = (X AND Y) = (X NOR X) NOR (Y NOR Y)

Beispiel: X = 1, Y = 1 $\rightarrow$ Z = 1

Als Übungsaufgabe kann die Richtigkeit der AND-Reduktion bewiesen werden. Dazu ist es nur notwendig, die aussagenlogische Gleichwertigkeit der beiden Ausdrücke nachzuprüfen.

[5] Weitere wichtige logische Schaltungen

Die **Äquivalenz** (X EQU Y bzw. X = Y). Sie ist nur dann wahr, wenn beide Teilaussagen immer denselben Wahrheitswert haben. In allen anderen Fällen ist sie falsch.

Die **Antivalenz** (X XOR Y) entspricht dem ausschließenden ODER, also ENTWEDER X ODER Y. Sie ist nur dann wahr, wenn die beiden Teilaussagen verschiedene Wahrheitswerte haben. Sie ist also die Negation der Äquivalenz. (X XOR Y) = NOT (X EQU Y).

Der **Shefferschе Strich** (X | Y) = (X NAND Y) entspricht NOT (X AND Y) und ist nur dann wahr, wenn X und Y falsch sind. In allen anderen Fällen ist er falsch.

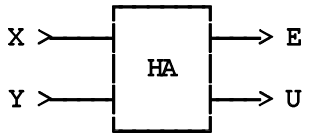
Wertetabelle:

0 = Falsch
1 = Wahr

X	Y	X EQU Y	X XOR Y	X NAND Y
0	0	1	0	1
0	1	0	1	0
1	0	0	1	0
1	1	1	0	0

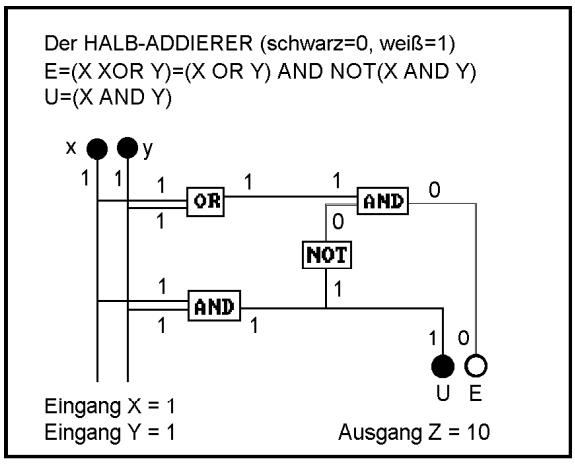
Wertetabelle von Äquivalenz, Antivalenz und Shefferschem Strich.

[6] Der Halb-Addierer (HA) addiert zwei einstellige Binärzahlen (Bits) X, Y. Er hat zwei Ausgänge, einen für die Einerstelle E und den anderen für die Zweierstelle U (Überlauf) der Summe. Ein Überlauf ist deswegen notwendig, weil 1 + 1 = 2 ist und die Zahl 2 binär als 10 codiert wird.



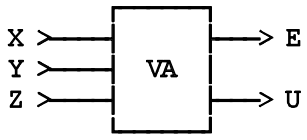
Wertetabelle:

X	Y	U	E	(dez)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	2



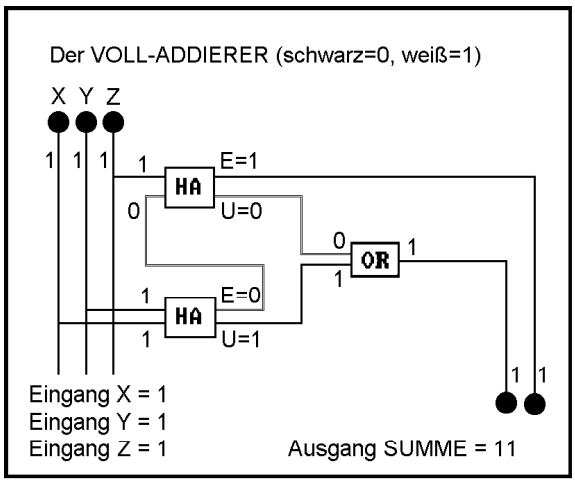
E = (X XOR Y) = (X OR Y) AND NOT (X AND Y)
U = (X AND Y)

[7] Der Voll-Addierer (VA) addiert drei einstellige Binärzahlen (Bits) X, Y, Z. Er hat zwei Ausgänge, einen für die Einerstelle E und den anderen für die Zweierstelle U (Überlauf) der Summe. Meistens wird der Volladdierer durch 2 Halbaddierer und eine OR-Schaltung realisiert.



Acht Möglichkeiten für die Addition von 3 Bits:

X + Y + Z (bin)	(dez)
0 + 0 + 0 = 00	= 0
0 + 0 + 1 = 01	= 1
0 + 1 + 0 = 01	= 1
1 + 0 + 0 = 01	= 1
0 + 1 + 1 = 10	= 2
1 + 0 + 1 = 10	= 2
1 + 1 + 0 = 10	= 2
1 + 1 + 1 = 11	= 3



E = (X XOR Y) XOR Z
U = (X AND Y) OR (X AND Z) OR (Y AND Z)

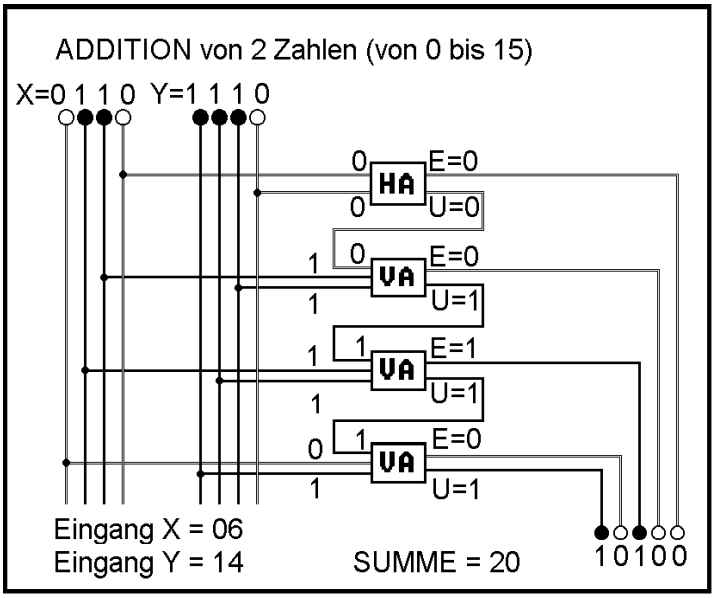
Beispiel:
Eingänge X = 1, Y = 1, Z = 1
Ausgang SUME = 11 (=3)

[8] Das Addierwerk

In einem parallelen 4-Bit-Addierwerk werden zwei vierstellige Binärzahlen addiert. Diese Schaltung ist aus einem Halb-Addierer und drei hintereinander geschalteten Voll-Addierern aufgebaut. Ein etwaiger Überlauf bei der Addition zweier Ziffern muss mithilfe eines Voll-Addierers zur Summe der nächsten beiden Ziffern hinzugezählt werden.

Beispiel: 6 + 14 = 20

```
  0110    ( 6)
+ 1110    (14)
=======
(1110 )  (Überläufe der Addierschaltungen)
-------
 10100   (Stellen der Addierschaltungen)
  ||||
  |||└── HA   ( 0+0)
  ||└─── VA   (0+1+1)
  |└──── VA   (1+1+1)
  └───── VA   (1+0+1)
```



[9] Die Grundrechnungsarten

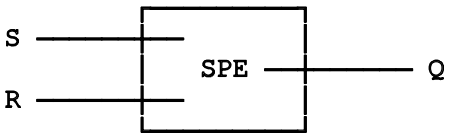
Logische Operationen verwenden wir täglich, wenn wir **Rechnungen** ausführen und auch **Entscheidungen** fällen. Genau das sind aber auch die beiden Hauptleistungen der zentralen Prozessoreinheit eines Computers.

Rechnungen mit Zahlen kann man im Wesentlichen auf das **Addieren** von natürlichen Zahlen im binären Code zurückführen. Die Subtraktion entspricht einer Addition mit der Gegenzahl. Die Multiplikation ist nichts anderes als eine Addition von lauter gleichen Summanden, und die Division ist nichts anderes als eine Subtraktion von lauter gleichen Subtrahenden.

Subtraktion: 7 – 4 = 7 + (–4) = 3
Multiplikation: 3 * 5 = 3 + 3 + 3 + 3 + 3 = 15
Division: 16 / 3 = 16 – 3 – 3 – 3 – 3 – 1 (Rest) = 5 und 1 Rest
(3 * 5)

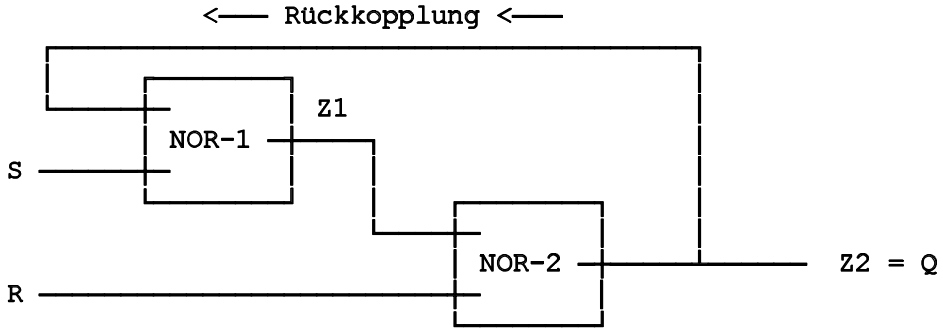
[10] Binäre Signalspeicher

Ein binärer Signalspeicher (SPE) muss in der Lage sein, ein binäres Signal (S) aufzunehmen und dieses auch dann noch zu behalten (speichern), wenn das auslösende Signal selbst bereits wieder verschwunden ist. Meistens wird zusätzlich noch verlangt, dass eine solcherart gesetzte Speicherzelle durch ein eigenes Rückstellsignal (R) wieder gelöscht werden kann.



Blockbild einer binären Speicherzelle

Im Ruhezustand führt der Speicherausgang Q ein 0-Signal. Wenn jetzt über den Setzeingang S ein 1-Signal eingegeben wird, so wird der Speicherausgang Q auf 1 gezwungen. Wird Eingang S wieder auf 0 gesetzt, bleibt der Speicherausgang Q im Zustand 1. Eine neuerliche Beschickung des Setzeinganges S mit dem Signal 1 ändert am Ausgangszustand Q der Speicherzelle nichts. Gelöscht kann der Speicher nur dann werden, wenn über den Rückstelleingang R ein 1-Signal zugeführt wird. Technisch kann ein solcher binärer Signalspeicher mithilfe von zwei rückgekoppelten NOR-Gattern verwirklicht werden (RS-Flip-Flop bzw. LATCH).



Wertetabelle

S	R	Z1	Z2= Q	
0	0	1	0	Speicher in Ruhelage
1	0	0	1	Speicher wird gesetzt
0	0	0	1	Speicher bleibt gesetzt
0	1	0	0	Speicher wird gelöscht
0	0	1	0	Speicher bleibt gelöscht
1	1	--	--	NICHT möglicher Zustand der Eingangs-Signale

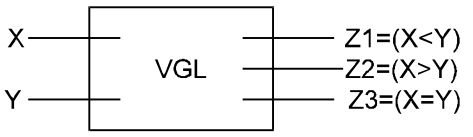
Unter der Annahme, dass der Speicherausgang Q in Ruhelage auf 0 steht, wird der Ausgang des ersten NOR-Gatters Z1 auf 1 gezwungen. Beginnend mit diesem Anfangszustand zeigt die obige Tabelle die verschiedenen Zustände der Speicherzelle für die verschiedenen Eingangssignale an. Eine Eingangs-Situation, wo sowohl das Setzsignal S als auch das Rückstellsignal R auf 1 stehen, ergibt keinen Sinn und ist daher ausgeschlossen.

Ausgehend von dieser statischen Basisschaltung werden zeitlich getaktete, binäre Speicherzellen entwickelt (T-Flip-Flop), welche die kleinsten Speicherbausteine des Computer-RAM bilden.

[11] Logische Entscheidungen

Entscheidungen können im Wesentlichen auf das **Vergleichen** von Zahlen im binären Code zurückgeführt werden. Das Grundproblem dabei, nämlich der Vergleich zweier Binärziffern, kann mithilfe von drei **Vergleichsschaltungen** (VGL) gelöst werden:

X	Y	<	>	=
0	0	0	0	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1



Dabei sind X und Y die beiden binären Ziffern. Die drei Ausgänge der Schaltung entsprechen den Aussagen (X<Y), (X>Y) und (X=Y). Auch diese Vergleichsschaltungen (VGL) können durch logische Operationen dargestellt werden. Durch Einsetzen von 0 und 1 in die Formeln wird die Richtigkeit der Wertetabelle (VGL) nachgewiesen.

(X < Y) = (NOT X) AND Y = ($\neg X \wedge Y$)
(X > Y) = X AND (NOT Y) = ($X \wedge \neg Y$)
(X = Y) = NOT (X XOR Y) = ($X \vee \neg Y$) $\wedge$ ($\neg X \vee Y$)

Nachweis, dass die Schaltung (X = Y) = ($X \vee \neg Y$) $\wedge$ ($\neg X \vee Y$) die Gleichheit (Äquivalenz) von X und Y überprüft :

(a) X = 0, Y = 0
($X \vee \neg Y$) $\wedge$ ($\neg X \vee Y$) = ($0 \vee 1$) $\wedge$ ($1 \vee 0$) = $1 \wedge 1 = 1$

(b) X = 0, Y = 1
($X \vee \neg Y$) $\wedge$ ($\neg X \vee Y$) = ($0 \vee 0$) $\wedge$ ($1 \vee 1$) = $0 \wedge 1 = 0$

(c) X = 1, Y = 0
($X \vee \neg Y$) $\wedge$ ($\neg X \vee Y$) = ($1 \vee 1$) $\wedge$ ($0 \vee 0$) = $1 \wedge 0 = 0$

(d) X = 1, Y = 1
($X \vee \neg Y$) $\wedge$ ($\neg X \vee Y$) = ($1 \vee 0$) $\wedge$ ($0 \vee 1$) = $1 \wedge 1 = 1$

Zusammenfassung

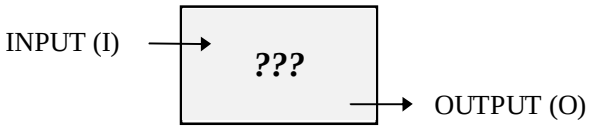
Somit lassen sich die beiden Hauptleistungen, das ***Rechnen*** und das ***Entscheiden*** auf die logischen Grundoperationen zurückführen. Man kann aber noch einen Schritt weitergehen und alle logischen Grundoperationen mit einer einzigen realisieren, z.B. mittels NOR-Verknüpfung. Dieses Ergebnis ist von großer praktischer Bedeutung, denn dadurch genügt die serienmäßige Produktion von einem einzigen logischen Schaltkreis, um bei entsprechender Verdrahtung von tausenden solcher gleichartiger Bauelemente auf kleinster Fläche alle gewünschten Funktionen und Leistungen zu realisieren (IC = Integrated Circuit, Chip).

Neben diesen aktiven Schaltungen zur Informationsverarbeitung (Rechnen, Entscheiden) gibt es auch elektronische Schaltungen zur bitweisen Informationsspeicherung. Eine solche Speicherzelle für ein Byte besteht aus 8 binären Signalspeichern, die stabil in einem eingestellten Zustand (0 oder 1) verbleiben, solange sie nicht neu gesetzt werden. Technisch können diese binären Signalspeicher mithilfe von zwei rückgekoppelten NOR-Schaltungen verwirklicht werden (RS-Flip-Flop, Latch). Die gesamte Hardware ist aus solchen elektronischen Schaltkreisen aufgebaut.

3 DER AUFBAU DES COMPUTERS

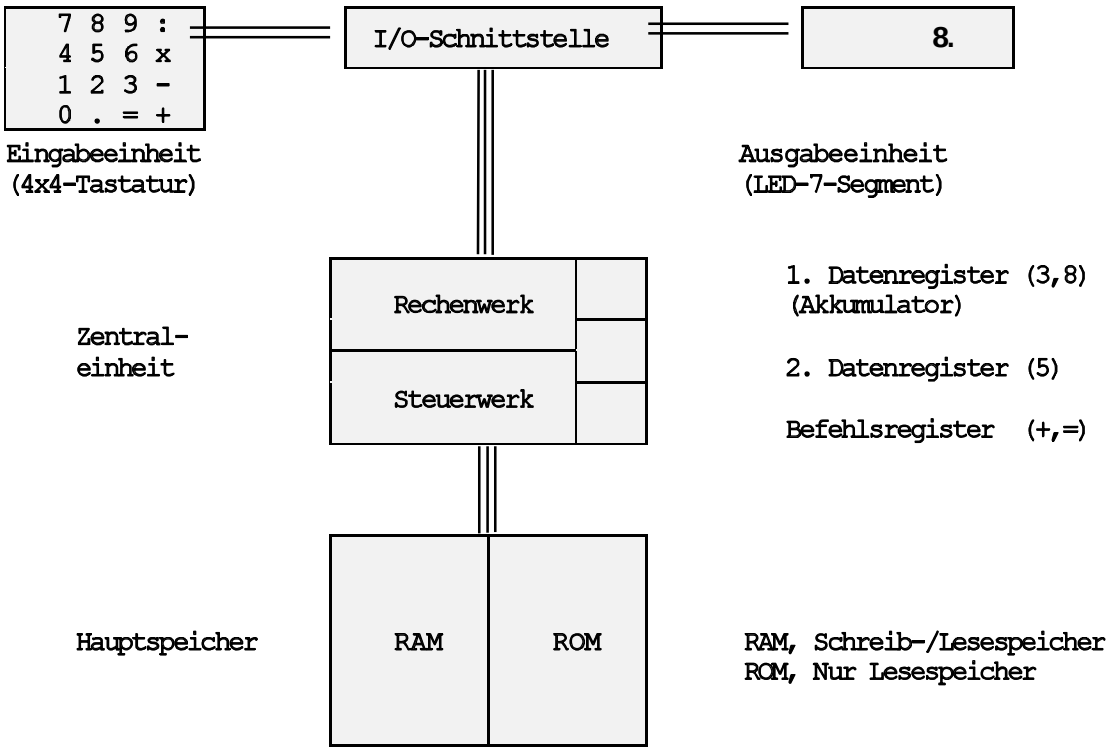
3.1 Ein einfaches Computermodell

Der erste Zugang des Menschen zum Computer erfolgt im Allgemeinen nach 2 Prinzipien, dem BLACK-BOX-Prinzip und dem EVA-Prinzip (Eingabe – Verarbeitung – Ausgabe). Nach diesen Prinzipien erfolgt die Handhabung der meisten modernen, hochtechnologisierten Geräte (Fernseher, Waschmaschinen, usw.). Das Gerät selbst ist eine Black-Box, deren Innenleben weitgehend unbekannt ist. Für den Benutzer ist nur die richtige Handhabung der Bedienungselemente von Relevanz (d.h. der Eingabe-/Ausgabe-Schnittstelle, Eingabe = **I**nput, Ausgabe = **O**utput).



BLACK-BOX

Folgendes einfache Modell soll einen ersten tieferen Einblick in die Black-Box eines Computers ermöglichen (Rechenbeispiel: 3 + 5 = 8):



Anmerkung: Unter **Schreiben** (Write) versteht man den Datentransfer von der Zentraleinheit in den Hauptspeicher. Unter **Lesen** (Read) versteht man den Datentransfer vom Hauptspeicher in die Zentraleinheit. Will man an unserem einfachen Rechnermodell die Rechnung 3 + 5 = 8 durchführen, dann müssen bestimmte Aktionen gesetzt werden. Dabei ruft jede Aktion ein so genanntes **Mikroprogramm** im Steuerwerk auf, welches im Rechenwerk die entsprechenden **Schaltkreise** steuert. Nachfolgend werden die einzelnen Befehle zur Ausführung der Rechnung detailliert beschrieben.

1. Schritt: Eintasten der Zahl „3“.

Der Computer liest und decodiert das Tastatursignal. Das entsprechende Bitmuster (3=00000011) wird über Verbindungsleitungen in das erste Datenregister (Akkumulator) der Zentraleinheit gebracht. Außerdem erfolgt eine optische Darstellung in der Ausgabeeinheit.

2. Schritt: Eintasten des Befehls „+“.

Der Computer liest und decodiert das Tastatursignal. Der entsprechende Befehlscode wird in das Befehlsregister der Zentraleinheit gebracht. Außerdem erfolgt eine optische Darstellung in der Ausgabeeinheit.

3. Schritt: Eintasten der Zahl „5“.

Der Computer liest und decodiert das Tastatursignal. Das entsprechende Bitmuster (5=00000101) wird über Verbindungsleitungen in das zweite Datenregister der Zentraleinheit gebracht. Außerdem erfolgt eine optische Darstellung in der Ausgabeeinheit.

4. Schritt: Eintasten des Befehls „=“.

Der Computer liest und decodiert das Tastatursignal. Dann wird ein im STEUERWERK fest verankertes Mikroprogramm ausgelöst, welches dem im Befehlsregister stehenden Befehl „+“ entspricht und welches im RECHENWERK die Addition der beiden Zahlen bewirkt. Das Ergebnis (8=00001000) wird in das erste Datenregister gebracht und in der Ausgabeeinheit dargestellt. Die Rechnung im binären Zahlencode lautet: [00000011] + [00000101] = [00001000].

Zur Bewältigung der einfachen Rechenaufgabe 3 + 5 = 8 muss also zwischen mehreren Geräteeinheiten (UNITS) ein geordneter Informationsfluss stattfinden und das Zusammenspiel der Units muss in sinnvoller Weise gesteuert werden. Diese Aufgaben (Tastaturlesung, Codierung, Decodierung, ...) werden von entsprechenden, eingespeicherten Dienstprogrammen bewältigt.

3.2 Die Geräteeinheiten des Systems

- **Die Zentraleinheit** (**CPU** = **C**entral **P**rocessing **U**nit). Sie gliedert sich in drei Bereiche:

a) Die Register: Sie dienen als Zwischenspeicher für Daten und Befehle.

b) Das Steuerwerk: Jeder einlangende Befehl wird hier decodiert und aktiviert ein entsprechendes Steuerprogramms (Mikroprogramm). Dadurch wird der interne Datenverkehr und die Auswahl der Operationen im Rechenwerk kontrolliert.

c) Das Rechenwerk: Das Rechenwerk führt die Anweisungen des Steuerwerkes mit den Daten aus den Registern durch (d.h. die Daten werden entweder arithmetisch oder logisch verknüpft).

- **Der Hauptspeicher** (Memory). Man unterscheidet zwei Bereiche (ROM und RAM):

a) ROM: Im Read-Only-Memory (Nur-Lese-Speicher) sind Hilfs- und Dienstprogramme fest verankert, welche zur geordneten und reibungslosen Informationsverarbeitung unbedingt erforderlich sind. Hier kann nichts hineingespeichert werden.

b) RAM: Im Random-Access-Memory (Wahlfreier-Zugriffs-Speicher) können vom Benutzer Daten und Befehle sowohl eingeschrieben als auch ausgelesen werden.

- **Das Bussystem**

Man unterscheidet 3 Arten von Verbindungsleitungen, auf denen Information transportiert wird:

a) Der Datenbus: Hier werden die Daten transportiert.

b) Der Adressbus: Hier werden die Adressen transportiert, die angeben, wohin die Daten verschickt werden.

c) Der Steuerbus: Er leitet die Steuersignale, welche angeben, ob Daten in die adressierten Speicherzellen geschrieben oder gelesen werden sollen. Der Steuerbus wird auch Kontrollbus genannt.

- **Die Schnittstellen-Adapter** (Interfaces, Controller)

Sie kontrollieren den Datenverkehr zwischen den zentralen und den peripheren Geräteeinheiten.

- **Die peripheren Geräteeinheiten**

a) Ein- und Ausgabegeräte: Tastatur, Maus, Scanner, Bildschirm, Drucker, ...
b) Externe Speicher: Festplatte, DVD- Laufwerk, USB-Stick, ...
c) Spezielle Geräte zur Datenerfassung einerseits und Prozesssteuerung andererseits.

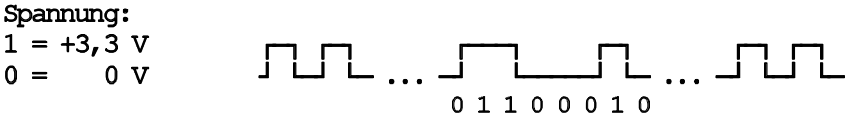
3.3 Die Informationsübertragung

Impulsgeneratoren (Taktgeber) sind elektronische Bauelemente, welche binäre Signale erzeugen und zeitlich steuern können. Eine solche einfache elektrische Impulssetzung heißt auch eine Taktgebung. Dabei ergibt sich ein periodischer, rechteckiger Verlauf der elektrischen Spannung. Schematische Darstellung des zeitlichen Verlaufes von binären Taktsignalen:

Taktdauer, z.B. $^1/_{500}$ Mikrosekunden (das sind millionstel Bruchteile von einer Sekunde). Unter Frequenz versteht man die Anzahl von Takten in einer Sekunde (Hertz, Hz), beispielsweise 500 Millionen Hertz (500 Megahertz, MHz). Die Taktrate ist ein Maß für die Arbeitsgeschwindigkeit der Zentraleinheit. Es gibt grundsätzlich zwei Formen der Informationsübertragung, die serielle und die parallele Datenübetragung.

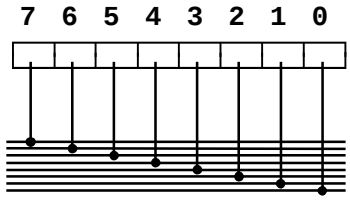
- **Die bitserielle Informationsübertragung:**

Die Bits werden zeitlich hintereinander durch die Leitung geschickt. Dabei wird der vorgegebene Maschinentakt zugrunde gelegt, sodass ein Takt einem Bit entspricht.



- **Die bitparallele Informationsübertragung:**

Die Bits eines Bitmusters werden zeitgleich durch nebeneinander liegende Leitungen geschickt; z.B. können die 8 Bitsignale von einem Byte gleichzeitig auf 8 parallelen Leitungen des Datenbusses transportiert werden (siehe unten stehende Abbildung). Auch 64 Bitsignale (= 8 Byte) können auf 64 parallelen Leitungen des Datenbusses befördert werden.



Eine Speicherzelle (Byte) mit 8 binären Signalspeichern (Bits). Der Zahlenwert eines Bytes reicht von 0 bis $2^8 - 1 = 255$, d.h. von 00000000, 00000001 bis 11111111.

8 parallele Leitungen bilden den Datenbus. Ist der Datenbus 32 Bit breit, so versorgt er zugleich vier Speicherzellen.

In Computersystemen gibt es oft Schnittstellen, wo die bitparallele Information in eine bitserielle umgewandelt wird und umgekehrt. Dafür sind so genannten SCHIEBE-Register, notwendig. Als Leistungsmerkmale des Prozessors können die Breite des Datenbusses (8, 16, 32 oder 64 Bit) und die Taktfrequenz (500 Megahertz oder mehr) herangezogen werden.

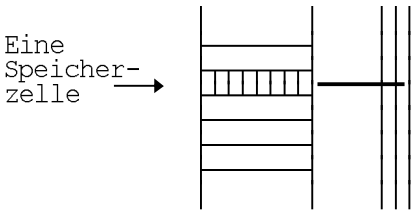
3.4 Die Informationsspeicherung

Eine Speicherzelle besteht aus **acht** nebeneinander liegenden binären Signalspeichern. Sie ist also genau **ein Byte** breit. Binäre Signalspeicher werden durch spezielle, elektronische FLIP-FLOP-Schaltungen realisiert. Sie verbleiben im gesetzten Zustand (0 = 0 Volt, 1 = +5 Volt), solange sie nicht gelöscht werden. Der Inhalt einer solchen Speicherzelle wird über den Datenbus bitparallel übertragen **(Lesen** oder **Schreiben)**.

Hinweis: Die in Bytes gespeicherten Daten können verschiedene Bedeutung haben, beispielsweise als Zahlen oder als Textzeichen. Das Wort „EVA“ besteht aus drei Bytes, wobei jedes Byte den so genannten **ANSI**-Code (**A**merican **N**ational **S**tandard **I**nstitute) der Buchstaben enthält: E = [69] = [0100 0101], V = [86] = [0101 0110], A = [65] = [0100 0001].

Ein Speicher besteht aus vielen solchen Speicherzellen. Dabei muss jede einzelne Zelle genau adressierbar sein. Die entsprechenden Adressleitungen bilden den Adressbus. Die Umwandlung einer am Adressbus anliegenden Adresse in den Zugriff auf die passende Speicherzelle wird von einer eigenen elektronischen Schaltung bewältigt (ADRESS-Decoder).

Außerdem muss noch bekannt gegeben werden, ob der Inhalt einer Zelle auf den Datenbus gelegt werden soll (read, lesen) oder ob der Inhalt vom Datenbus in die Zelle gespeichert werden soll (write, schreiben). Die entsprechenden Signale werden am Steuerbus geleitet.

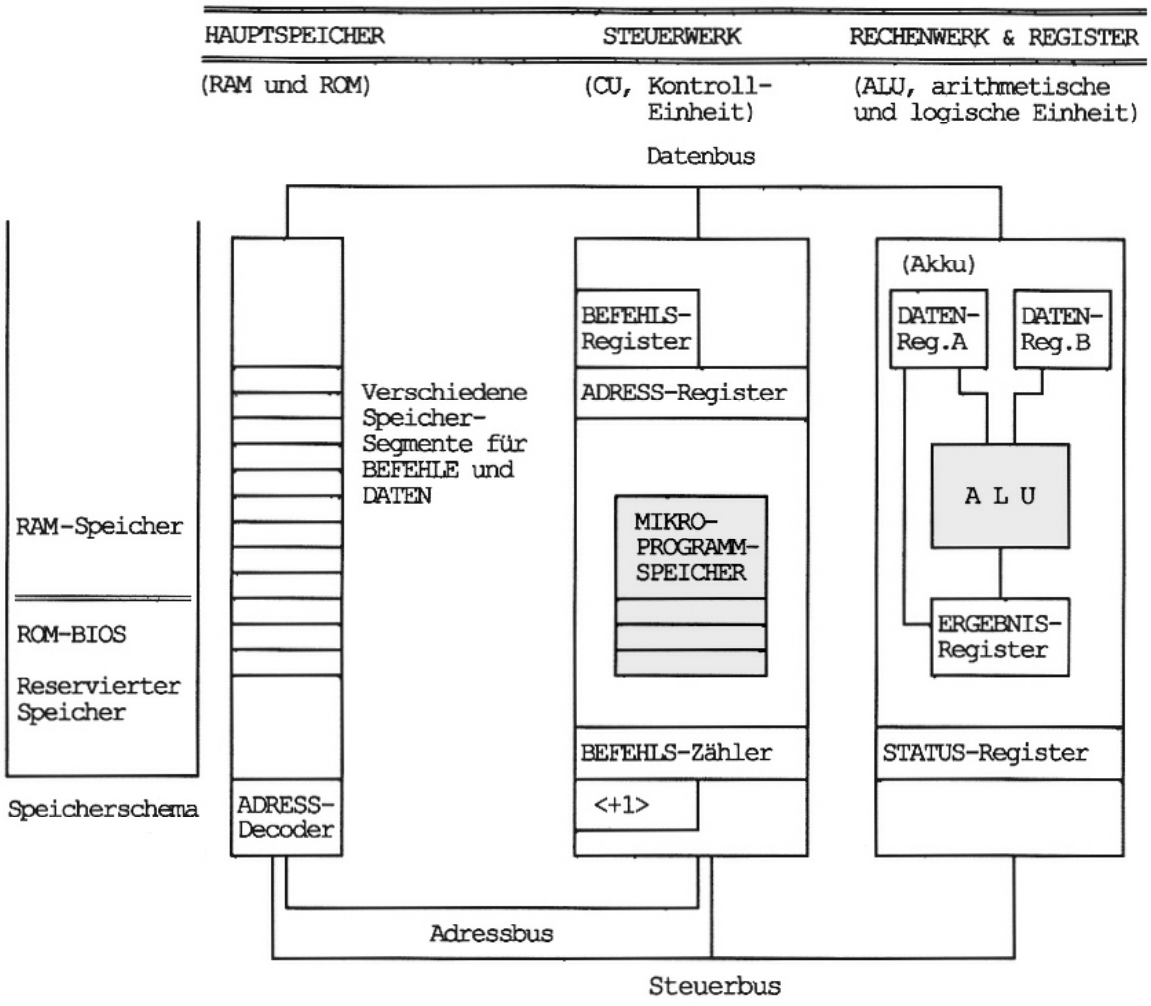


Ein einfaches Speichermodell:

Den Hauptspeicher kann man sich als einen Kasten und die Zellen als seine Schubladen vorstellen, die alle mit Nummern versehen sind. In jeder Zelle (Byte) befindet sich ein Bitmuster aus acht Bits (0,1).

Bei 16 MB Speicher ($16 * 2^{20} = 2^{24} = 16\,777\,216$ Byte) sind zum gezielten Zugriff auf jedes Byte genau 24 parallele Adressleitungen notwendig. Bei 16 GB Speicher ($16 * 2^{30} = 2^{34}$ Byte) sind dann 34 parallele Adressleitungen notwendig.

3.5 Ein erweitertes Computermodell



Die Zentraleinheit (**CPU**) gliedert sich in das Rechenwerk (**ALU**) und die Kontrolleinheit (**CU**), welche das eigentliche Steuerwerk mit seinen fest eingespeicherten Steuerfunktionen enthält. Diese werden auch als Mikroprogramm bezeichnet. Daneben befinden sich hier ein oder mehrere Befehlsregister, welche jene Befehle aufnehmen, die über das Adressregister aus dem Hauptspeicher geholt werden. Das Adressregister enthält die Adressen von Daten oder Befehlen, auf die mithilfe des Adressbusses und des Adressdecoders im Hauptspeicher zugegriffen wird. Nach jeder Befehlsausführung wird der Befehlszähler automatisch um Eins <+1> erhöht.

Das Rechenwerk (**ALU**) nimmt in seinen Datenregistern jene Daten auf, welche im Hauptspeicher adressiert wurden, und führt damit dann jene arithmetischen und logischen Operationen aus, welche durch den im Befehlsregister stehenden Befehl veranlasst werden. Das Ergebnis wird in ein entsprechendes Register gestellt und kann von dort in die Zellen des Hauptspeichers geschrieben werden. Im Statusregister werden etwaige Fehler oder Zahlenüberläufe, die bei den Operationen im Rechenwerk auftreten können, in entsprechenden Kontrollbits vermerkt.

Der Hauptspeicher schließlich enthält in seinen Zellen entweder Daten oder Befehle. Diese beiden Speicherbereiche müssen streng voneinander getrennt werden, denn wenn ein Programm gerade Daten verarbeitet und abspeichert, so dürfen diese nicht die Befehle des Programmes überschreiben. Nur eine ordentliche Speicherverwaltung kann solche fatalen Fehler vermeiden.

Jede Arbeit im Computer – ob eine Textverarbeitung oder eine Datenbankverwaltung – wird von Programmen gesteuert. Eine so genannte höhere Programmiersprache besteht aus symbolischen Befehlen, die den Computer anweisen, bestimmte Tätigkeiten auszuführen. Dabei werden auch die im Hauptspeicher adressierten Speicherbereiche durch Symbole (Variablen) gekennzeichnet, sodass der Programmierer sich um die eigentliche Speicherverwaltung nicht selbst kümmern muss. Diese wird von speziellen Hilfsprogrammen des Systems übernommen. Damit das alles gelingt, muss der symbolische Programmbefehl in eine Folge von Maschinenbefehlen (Makroprogramm) übersetzt und an die Stellen der Variablen echte Adressen eingesetzt werden. Dieser Vorgang heißt Compilation (Übersetzung) und wird mit den erwähnten speziellen Hilfsprogrammen durchgeführt. Ein Maschinenbefehl seinerseits besteht aus einer bestimmten Anzahl von binären Signalen (0 oder 1), welche in der Zentraleinheit den Ablauf von Steuerwerksfunktionen (Mikroprogramm) veranlassen. Das nachfolgende Beispiel der Addition zweier Zahlen soll die Arbeitsweise des Mikroprozessors schematisch darstellen.

LET Z = X + Y	***LDA 1000*** ***ADD 1001*** ***STO 1002***	***101100101110*** ***.....*** ***.....***
Befehl in der Hochsprache	***Maschinenbefehle (Makroprogramm)***	***Steuerwerksfunktionen (Mikroprogramm)***

LET Z = X + Y	***ist ein Befehl aus der Hochsprache BASIC und weist der Variablen Z die Summe der Variablen X und Y zu.***
LDA 1000	***ist ein Maschinenbefehl und lädt das Akkumulator-Register der CPU mit dem Inhalt der Speicherzelle 1000 (symbolisiert durch X).***
ADD 1001	***addiert zum Akkumulator den Inhalt von Zelle 1001 (symbolisiert durch Y).***
STO 1002	***speichert den Akkumulator-Inhalt in die Zelle 1002 (symbolisiert durch Z).***

Somit ergibt sich eine dreifache Befehlshierarchie:

I. Der symbolische Befehl einer Hochsprache

Er wird von einem eigenen Sprachübersetzungsprogramm (Interpreter, Compiler) in eine entsprechende Folge von Maschinenbefehlen übersetzt (Maschinenprogramm, Makroprogramm).

II. Der Maschinenbefehl

Er wird von der Zentraleinheit in drei Phasen abgearbeitet, welche ihrerseits vom Steuerwerk überwacht werden (man nennt dies nach dem Informatiker und Mathematiker John von Neumann einen „VON NEUMANN"-Zyklus):

1. Befehl holen
2. Befehl decodieren
3. Befehl ausführen

Die drei Phasen werden in mehreren Maschinentakten abgearbeitet, sodass die Arbeitsgeschwindigkeit des Mikroprozessors von der Taktfrequenz des zentralen Taktgebers abhängt.

III. Das Mikroprogramm

Dieses ist eine Folge von binären Signalzuständen, die fest im Steuerwerks-ROM verankert sind. Das Mikroprogramm wird von einem Maschinenbefehl ausgewählt und steuert die einzelnen Schaltungen im Rechenwerk (z.B. Addieren, Negieren bzw. Komplementieren). Ein Mikrobefehl entspricht genau einem Maschinentakt. Von der Vielfältigkeit der Mikroprogramme im Steuerwerk hängt die Funktionsmächtigkeit des Mikroprozessors ab.

Der Cache-Speicher

Ein wichtiges Leistungsmerkmal des PCs ist die Zugriffszeit der CPU auf die Zellen des Hauptspeichers. Um diese zu verringern besitzen moderne Computer (die Nachfolger des INTEL 8086) so genannte **Cache-Speicher.** Diese sind nur einige hundert Kilobyte groß, aber wesentlich schneller als der Hauptspeicher. In einem Cache werden häufig von der CPU benötigte Daten bereitgehalten, sodass der Prozessor diese Daten viel schneller zur Verfügung hat und nicht auf den relativ langsamen Hauptspeicher warten muss. Soll die CPU Daten aus dem Hauptspeicher lesen, so prüft ein **Cache-Controller** zuerst, ob diese bereits im Cache-Speicher vorhanden sind. Ist dies der Fall, so werden die Daten der CPU sofort übergeben. Sonst liest der Cache-Controller die Daten aus dem Hauptspeicher und gibt sie gleichzeitig an den Prozessor weiter. Soll die CPU umgekehrt Daten schreiben, so werden diese mit hoher Geschwindigkeit in den Cache-Speicher geschrieben. Der Cache-Controller kümmert sich anschließend darum, dass die Daten auch in den Hauptspeicher übertragen werden. Ein ähnliches Verhalten legen auch wir Menschen an den Tag: Wenn wir z.B. Schreibarbeiten zu erledigen haben, so nehmen wir die Unterlagen aus dem Regal, die wir wahrscheinlich benötigen. Das Regal stellt dabei den Hauptspeicher und der Schreibtisch den Cache-Speicher dar und wir selbst sind der Cache-Controller. Tritt nun ein Problem auf, so werden wir weitere Unterlagen aus dem Regal auf den Schreibtisch holen. Ist dieser voll, der Cache-Speicher also erschöpft, so müssen wir die wahrscheinlich am wenigsten benötigten Unterlagen wieder ins Regal zurückstellen, bevor neue Unterlagen Platz haben.

3.6 Das Basis-Input-Output-System (BIOS)

Das **BIOS** ist ein mehrteiliges Dienstprogramm, das im Festspeicher (ROM) abgelegt ist. Es läuft immer automatisch beim Starten des Computers ab (**Booten**). Seine Hauptaufgaben sind:

- **Power-On-Selftest** (POST)
  Beim Booten erfolgt zuerst ein Prüftest für Prozessor und Hauptspeicher.
- **Initialisierungen** von Registerinhalten und internen Schalterstellungen.
- **Boot-Strap**
  Im letzten Teil des Bootens wird zunächst ein einleitender Ladesatz (Boot-Record) vom Festspeicher (ROM) geladen. Danach werden Dienstprogramme des Betriebssystems blockweise von der primären Festplatte in den Hauptspeicher geladen.

Die im **ROM** eingespeicherten Hilfsprogramme des **BIOS** können durch so genannte Interrupts aufgerufen werden. Beim **Booten** werden die Startadressen der 256 Interrupt-Service-Routinen in die ersten 1024 Byte des **RAM** geschrieben. Man nennt diesen Bereich auch **Interrupt-Tabelle**.

3.7 Die Interrupt-Verwaltung

Ein **Interrupt** ist eine Unterbrechung der laufenden Arbeit der Zentrale, welche durch ein bestimmtes **Ereignis** ausgelöst wird (z.B. durch das Drücken einer Taste der Tastatur). Dabei wird in eine Unterbrechungsroutine (**Interrupt-Handler** bzw. **Interrupt-Service-Routine, ISR**) verzweigt. Das ist ein im Hauptspeicher befindliches Dienstprogramm des Betriebssystems. Wenn der Interrupt nicht abgelehnt wird, dann wendet sich diese Service-Routine dem auslösenden Ereignis zu und stellt spezifische Verarbeitungsmöglichkeiten zur Verfügung, beispielsweise die Darstellung des eingetasteten Zeichens am Bildschirm. Nach der Beendigung der ISR wird die unterbrochene Arbeit wieder fortgesetzt. Solche Unterbrechungen können von einer **Hardware-Einheit,** beispielsweise der Tastatur (externer Interrupt), oder von einer **Software**, d.h. einem Anwenderprogramm (interner Interrupt), ausgelöst werden. Jedem Interrupt wird eine bestimmte Kennnummer **nn** zugeordnet **(INT nn)**.

Ein Beispiel aus dem Alltag soll das Interrupt-Konzept anschaulich beschreiben: Ein Haushalt einer Familie entspricht dem kompletten Computersystem. Auf Grund ihrer zentralen Bedeutung soll die Küche der CPU entsprechen, wo die Mutter gerade mit dem Tellerwaschen beschäftigt ist. Dabei werden die Teller auf einen Stapel **(Stack)** abgelegt. Plötzlich ertönt aus dem Kinderzimmer Geschrei (Interrupt-Aufruf von einem externen Gerät). Die Mutter hat nun zwei Möglichkeiten: Entweder sie ignoriert den Ruf oder sie reagiert darauf. Im zweiten Fall legt sie den letzten Teller zurück und merkt sich seine Position am Stapel. Hierauf geht sie ins Kinderzimmer und ergreift entsprechende Maßnahmen, um das Kind zu versorgen (Interrupt-Service-Routine). Sodann kehrt sie in die Küche zurück und setzt ihre Arbeit dort fort, wo sie vor der Unterbrechung stehen geblieben ist. Die Einrichtung eines **Stack-Speichers**, wo kurzfristig (nach dem Prinzip **Last in – First out**) Daten aufgestapelt und dann wieder abgehoben werden, erweist sich als wichtiges Hilfsmittel bei der Durchführung von Programmunterbrechungen. In unserem Beispiel entspricht die Mutter dem im Computer arbeitenden Betriebssystem und das Kind einem externen Gerät.

Die **HARDWARE-Interrupt-Verwaltung:** Hier befinden sich die Service-Routinen des ROMs, welche jene Unterbrechungen registrieren und beantworten, die von externen Bausteinen (z.B. Tastatur, Maus) ausgelöst werden. Dabei wird der Datenfluss auf den Ein-/Ausgabe-Kanälen geregelt. Um Kollisionen von verschiedenen gleichzeitig auftretenden Interrupts zu vermeiden, werden den einzelnen Interrupts unterschiedliche Prioritäten zugeordnet. Ein eigener elektronischer Baustein, der **Interrupt-Controller**, dient der Verwaltung der von der Peripherie in die Zentrale einlangenden Unterbrechungs-Anforderungen (**IRQ, Interrupt Request**).

Wichtige HARDWARE-Interrupt-Anforderungen	
IRQ 01h	Tastatursignal senden
IRQ 04h	Erste serielle Schnittstelle (COM1)
IRQ 07h	Erste parallele Schnittstelle (LPT1)
IRQ 08h	CMOS-Echtzeituhr
IRQ 09h	Tastatursignal empfangen
IRQ 0Eh	Festplatte

Die **SOFTWARE-Interrupt-Verwaltung:** Diese Dienstprogramme des ROMs werden von einer im Computer laufenden Software (Anwenderprogramm) über bestimmte Interrupt-Kennnummern aufgerufen. Auch diese Dienstprogramme dienen im Wesentlichen der Ein- und Ausgabe von Daten. Ihr Aufruf wird als interner Interrupt bezeichnet – im Gegensatz zu den externen Interrupts, die von einem Ereignis auf der Hardware-Ebene ausgelöst werden.

Hardware- und Software-Interrupts arbeiten zwar unabhängig voneinander, jedoch in enger Kooperation. Der Hardware-Interrupt-Teil der Tastaturverwaltung (INT 9h) beispielsweise reagiert auf ein Tastatursignal (IRQ 1h), codiert und speichert es in einem Tastaturpuffer. Mithilfe des Dienstprogrammes (INT 16h) können diese Tastencodes in die zentralen Datenregister der CPU überstellt und von dort an ein laufendes Programm übergeben werden.
Das Dienstprogramm INT 17h dient der Unterstützung der parallelen Druckerschnittstelle. Es enthält drei einfache Unterprogramme, die mit 0, 1 und 2 nummeriert sind. Nummer 0 schickt aus den zentralen Datenregistern der CPU die einzelnen Byte zum Drucker; Nummer 1 initialisiert den Drucker; Nummer 2 meldet den Druckerstatus, z.B. wenn kein Papier vorhanden ist.

Interrupt-Nummern wichtiger ROM-interner Dienstprogramme	
INT 10h	Daten am Bildschirm ausgeben
INT 11h	Systemkonfiguration erfragen
INT 13h	DISK-Funktionen (z.B. Formatieren)
INT 16h	Daten von der Tastatur abfragen
INT 17h	Drucker-Funktionen (z.B. Ausdrucken)
INT 19h	Boot-Strap, startet den BOOT-Vorgang

Die **PROZESSOR-Interrupt-Verwaltung:** Neben der Hardware und der Software kann auch der Prozessor selbst einen Interrupt auslösen. Beispielsweise wird eine so genannte ***Exception*** ausgelöst, wenn ein interner Fehler (Speicherzuordnungsfehler, Nulldivision) aufgetreten ist.

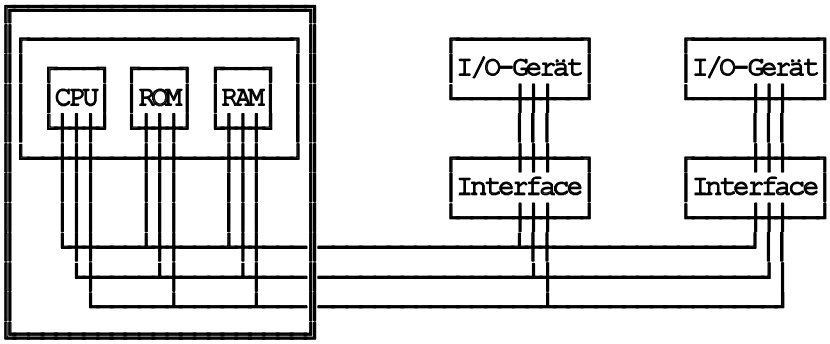
3.8 Schnittstellen, die Tore zur Außenwelt

Neben Zentraleinheit (CPU) und Hauptspeicher (RAM) sind die so genannten Peripheriegeräte (Tastatur, Bildschirm, Drucker usw.) wichtige Bestandteile eines Computersystems. Sie dienen der **Eingabe** (I, Input) und der **Ausgabe** (O, Output) von Daten. Bei einer Eingabe liest die CPU die Daten aus einer externen Quelle in ein zentrales Datenregister. Bei einer Ausgabe schreibt die CPU die Daten aus einem zentralen Datenregister in ein externes Ziel. Der Datenaustausch zwischen Zentraleinheit und Peripherie muss sorgfältig organisiert werden. Zwischen dem Bussystem der Zentraleinheit und den Anschlüssen eines externen Gerätes liegt eine **Schnittstelle** (Interface), welche durch entsprechende Anpassungsschaltungen (Adapter, Controller) verwaltet wird. Ein solcher Baustein besteht aus hochintegrierten Schaltungen, welche zumeist auf einer eigenen Steckkarte realisiert sind.

Die **Adressen** der Anschlüsse (**I/O-Tore, Ports**) der einzelnen Schnittstellencontroller liegen nicht im Systemspeicher, sondern bilden einen eigenen Bereich von 64 Kilobyte, der mithilfe der Adressleitungen des Systembusses adressiert wird.

Über den externen **Datenbus** werden die Daten in die I/O-Tore übertragen, in einem Datenpuffer zwischengespeichert und von dort weiter in das angeschlossene I/O-Gerät transferiert. Der Datenverkehr kann natürlich auch in umgekehrter Richtung verlaufen.

Der **Kontrollbus** muss neben den Signalen für das Lesen (MEMR) und das Schreiben (MEMW) des Hauptspeichers auch noch die entsprechenden Signale für das Lesen (IOR) und Schreiben (IOW) eines I/O-Tores führen. Dann gibt es noch Leitungen zur Interrupt-Steuerung. Auch für den Systemtakt und weitere Kontrollfunktionen sind Leitungen vorhanden.



Grundsätzlich können parallele und serielle Schnittstellen unterschieden werden. Beispielsweise kann ein PC standardmäßig 3 parallele Schnittstellen verwalten, die man symbolisch mit LPT1, LPT2, LPT3 bezeichnet. Die softwaremäßige Verwaltung eines angeschlossenen Druckers kann über den Interrupt INT 17h erfolgen.

Zum Abschluss muss noch erwähnt werden, dass über die Datenbit des parallelen Anschlusses auch externe RELAIS ein- und ausgeschaltet werden können, sodass die parallele Schnittstelle auch für **technische Steuerungsaufgaben** von angeschlossenen Maschinen verwendet wird. Werden hingegen die Status-Anschlüsse von einem externen Gerät mit Signalen (1 = +5 Volt und 0 = 0 Volt) beschickt, dann können über die parallele Schnittstelle auch externe Signale erfasst werden (**Erfassung von Messdaten** und deren Darstellung und Auswertung). Zwei Beispiele aus der Praxis sollen diese Messdatenerfassung demonstrieren.

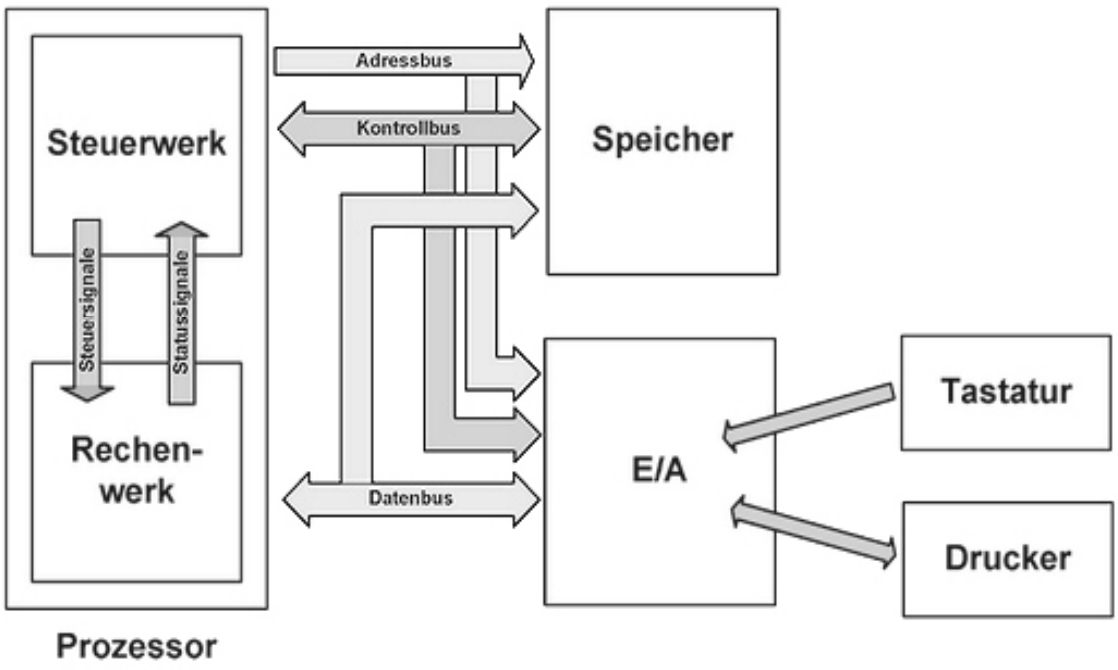
Erstens liefern beispielsweise **Photodioden** mit entsprechenden elektronischen Schaltungen einfache Signale (0 oder 1), wodurch erfolgte Unterbrechungen von Lichtschranken registriert werden. So ist es möglich, Überwachungseinrichtungen technisch zu realisieren.

Zweitens kann beispielsweise das externe Gerät ein **Thermoelement** sein, das entsprechend der Umgebungstemperatur eine elektrische Spannung erzeugt. Diese wird innerhalb eines bestimmten Bereiches durch einen so genannten **ADC-Wandler** (Analog Digital Conversion) in ein entsprechendes Bitmuster umgewandelt und in das Status-Register der parallelen Schnittstelle transportiert. Mit einem geeigneten Programm wird dieses Statusbyte in bestimmten Zeitschritten periodisch ausgelesen und ausgewertet. Die Auswertung kann darin bestehen, dass der zeitliche Verlauf der Messdaten am Bildschirm grafisch dargestellt wird (z.B. Temperaturkurve).

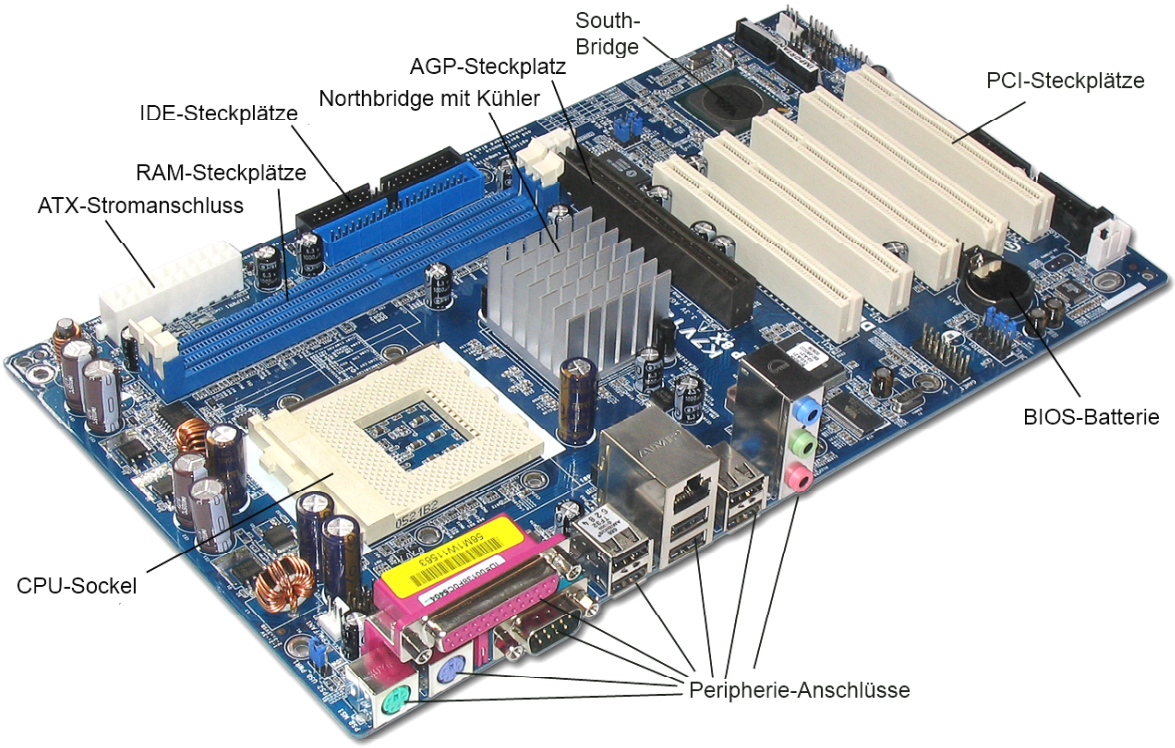
Ein **ADC-Wandler** ist ein elektronischer Baustein, der auf seiner Eingangsseite eine elektrische Spannung erhält, welche kontinuierlich jeden Wert zwischen zwei Bereichsgrenzen annehmen kann, was als analoge Messgröße X bezeichnet wird. Besteht auf der anderen Seite sein Ausgang aus beispielsweise einem Byte, dann wird die Differenz zwischen den Eingangsgrenzen in 256 gleich große Stufen zerlegt und jede solche Stufe entspricht genau einem Bit. So kann der reelle Wert der eingelangten Messgröße X in eine entsprechende ganze Zahl Z von 0 bis 255, also in ein 8-Bit-Muster umgewandelt werden, welches dann zur Ausgabe kommt. Natürlich bewirkt eine solche Digitalisierung eine starke Vergröberung der Messdaten-Erfassung.

Die Umkehrung einer **ADC**-Wandlung ist eine **DAC**-Wandlung (Digital Analog Conversion). Dabei wird ein digitales Signal (also ein bestimmtes Bit-Muster) in ein entsprechendes analoges Signal (beispielsweise in eine kontinuierlich veränderliche elektrische Spannung) umgewandelt. Solche DAC-Wandlungen finden beispielsweise beim Fernsehen ihre Anwendung. Dabei werden die gesendeten digitalen Signale in analoge elektrische Spannungen zur Gerätesteuerung konvertiert.

3.9 Ein komplettes Computersystem



Die Hauptplatine (Motherboard) mit CPU, RAM-Speicher und Kontroll-Bausteinen eines modernen Desktop-Computers



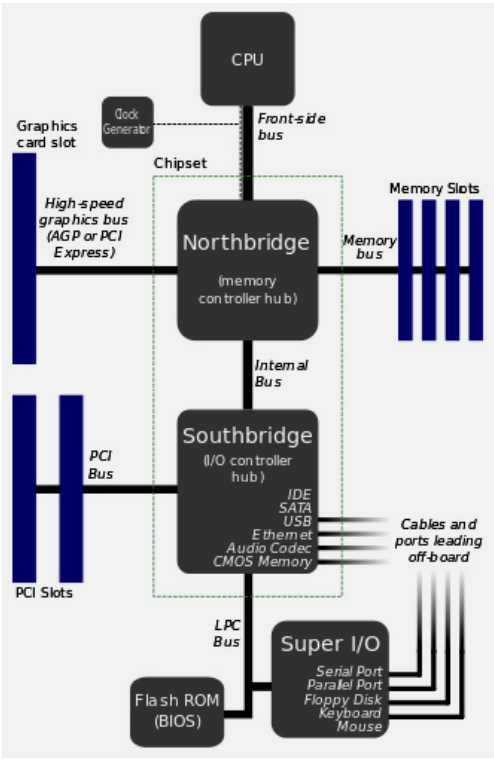
Laptop „Acer Aspire E5-575G-54T8“

CPU: Intel Core i5-7200U, 2 Prozessor-Kerne
Taktfrequenz = maximal 3,1 GHz (GigaHertz)
Adressbus = 36 Bit,
Datenbus = 64 Bit.

Festplatte: 1000 GB (SATA) und 128 GB (SSD)
Grafikkarte: nVidia GForce GTX 950M, 2 GB
Monitor: 15,6-Zoll Bilddiagonale,
HD-Auflösumg (1920 x 1080 Pixel),
Bildformat 16:9, entspiegelt, matt.

DVD-Brenner
2 Lautsprecher, 1 Mikrophon, 1 Kamera, 1 Kartenleser
2 Videoausgänge (HDMI, VGA)
3 USB-Anschlüsse
Datenkommunikation (Ethernet, WLAN, Bluetooth)
Akku mit 2800 mAh, ca. 6 h Laufzeit

Betriebssystem: WINDOWS 10, Home.



4 DIE GRUNDFUNKTIONEN DES BETRIEBSSYSTEMS

Damit die einzelnen Teile der Hardware reibungslos miteinander arbeiten können, muss ein so genanntes Betriebssystem diese Koordination und Verwaltung übernehmen. Ein Betriebssystem besteht aus einer Vielzahl von Programmdateien, welche zwischen Mensch und Hardware vermitteln. Erst dadurch wird eine komfortable Bedienung des Computers möglich.

MENSCH
ANWENDERSOFTWARE
BETRIEBSSYSTEM
HARDWARE

Der Anwender hat mit dem Betriebssystem und seinen Programmroutinen im Normalfall wenig zu tun. Er kommuniziert hauptsächlich mit der entsprechenden Anwendersoftware (Textverarbeitung, Tabellenkalkulation, Grafikverarbeitung, Datenbankprogramm), die sich ihrerseits des Betriebssystems bedient, um die Hardware zu steuern. Man unterscheidet bei einem Betriebssystem zwischen ***Kernprogrammen*** (zum Funktionieren unerlässliche Routinen) und ***Zusatzprogrammen.*** Diese befinden sich allesamt auf der Festplatte.

Eine ***Datei*** ist eine Menge von zusammengehörigen Bytes, auf die mithilfe eines symbolischen Namens zugegriffen werden kann. Grundsätzlich gibt es zwei Arten von Dateien:

Programmdateien**:** Ihre Bytes stellen Befehle dar, welche von der CPU als solche erkannt und hintereinander ausgeführt werden. In Wirklichkeit steuert ein solcher Befehl nur das Öffnen oder Schließen bestimmter elektronischer Schalter, wodurch dann die gewünschte Aktion erreicht wird (z.B. Auswahl eines Addierschaltkreises im Rechenwerk, Farbänderungen am Monitor, Transfer von Bytes aus dem RAM auf die Festplatte, usw.). So besteht beispielsweise die Textverarbeitung WORD aus vielen genau aufeinander abgestimmten Programmroutinen.

Datendateien**:** Ihre Bytes stellen keine Befehle, sondern schlichte Daten dar. Beispielsweise werden mithilfe eines Textverarbeitungsprogrammes Textzeichen entsprechend dem ANSI-Code binär verschlüsselt (d.h. in eine Folge von 0 und 1). Daneben gibt es Kalkulationstabellen (enthalten Zahlen), Grafikdateien (enthalten Bitmuster zur Farbcodierung der Bildpunkte), usw.

Nach dem Einschalten des Computers werden die Kernprogramme des Betriebssystems stufenweise von der Festplatte in den RAM-Speicher geladen und verbleiben dort resident bis zum Ausschalten des Gerätes. Dieser Startvorgang wird als ***Booten*** bezeichnet.

4.1 Die fünf Hauptaufgaben des Betriebssystems

(1) Verwaltung der Computer-Peripherie (Input/Output-Management)

Kontrolle und Regelung des Datenverkehrs zwischen der CPU und den I/O-Geräten. Bei der Installation eines neuen I/O-Gerätes (z.B. Drucker oder Scanner) muss unbedingt ein vom Hersteller mitgeliefertes gerätespezifisches Treiberprogramm von einer originalen CD oder DVD auf die Festplatte kopiert und ins Betriebssystem integriert werden. Auf diese gerätespezifischen Treiber greifen dann das Betriebssystem und in weiterer Folge auch die Anwendersoftware zu, wenn sie mit dem Gerät kommunizieren wollen. Die I/O-Geräte werden mit symbolischen Namen angesprochen (Laufwerk A:, Festplatte C:, usw.). Der so genannte ***Hardware Abtraction Layer*** (HAL) bildet die physischen Bus-Adressen auf logische Adressen ab und ermöglicht dadurch eine vom jeweiligen Rechner unabhängige I/O-Verwaltung.

(2) Verwaltung der externen Speicher (File-Management)

Das Betriebssystem ist verantwortlich für die ordentliche Abspeicherung und Ladung von Dateien. Es regelt den Datenverkehr zwischen Hauptspeicher und dem entsprechenden externen Speichermedium. Es ermöglicht die grundlegenden ***Datei-Operationen*** wie Inhaltsanzeigen, Kopieren, Löschen und Umbenennen. Es sorgt aber auch für den Schutz der Dateien durch Vergabe von ***Zugriffsrechten***. Hiezu ist für jede Datei ein Attributbyte vorgesehen, das verschiedene Zugriffsmöglichkeiten festlegt (z.B. offen und ungeschützt, versteckt, schreibgeschützt, usw.). Dadurch wird erst Datensicherheit gewährleistet. Zur besseren Übersicht können auf dem externen Speicher so genannte Verzeichnisse oder Ordner eingerichtet werden, sodass eine baumartige Gliederung von Verzeichnissen entsteht. Der ***Verzeichnisbaum*** beginnt immer im Stammverzeichnis (Wurzelverzeichnis, Root-Directory) und ästelt sich in die einzelnen Unterverzeichnisse auf. Dateien und Verzeichnisse werden durch symbolische Namen angesprochen, wobei für das Stammverzeichnis immer der Backslash (\) verwendet wird. Der Backslash dient auch als Trennzeichen in Verzeichnispfaden. So bezeichnet C:\TEXT\PRIVAT ein Unterverzeichnis PRIVAT im Verzeichnis TEXT der Festplatte **C:**. Die Dateinamen bestehen aus acht oder mehr Buchstaben, einem Punkt, und drei oder mehr Zeichen als Erweiterung (Extension). Programmdateien (Applikationen) haben meist die Erweiterung EXE (z.B. PAINT.EXE).

(3) Verwaltung des Hauptspeichers (Memory-Management)

Beim Laden von Programm- oder Datendateien von der Festplatte in den RAM dürfen die schon dort residierenden Kernprogramme des Betriebssystems nicht überschrieben werden. Diese RAM-Bereiche müssen ***protected*** (geschützt) und ***privilegiert*** behandelt werden. Jedoch können andere Speicherblöcke (Segmente) besetzt oder wieder freigegeben werden. Es muss daher ein dauerndes Protokoll über die aktuelle Speicherbelegung geführt werden, was mithilfe eigener Kontrolleinrichtungen (***Segment-Descriptoren*** und ***Paging***) geschieht.

(4) Ausführen von Programmen (Task-Management)

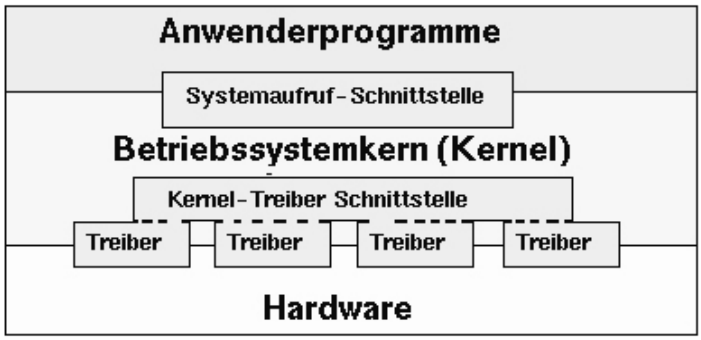
Jedes auszuführende Programm (***Task***) muss in den RAM geladen und die Adresse des ersten Programmbefehls der CPU übermittelt werden. Danach wird das Programm durch die CPU schrittweise, Befehl um Befehl, abgearbeitet. Nach der Beendigung des Programmes muss es wieder aus dem RAM entfernt, also die entsprechenden Speicherblöcke freigegeben werden. Eine Spezialfunktion ist das ***Multitasking,*** wobei mehrere Programmroutinen (***Tasks***) parallel in verschiedenen RAM-Bereichen ablaufen, ohne sich gegenseitig zu behindern. Problemsituationen beim Multitasking sind die so genannten Contentions, worunter man Konkurrenzsituationen versteht, wo zwei verschiedene Tasks zur selben Zeit ein bestimmtes Objekt (z.B. ein File, eine Schnittstelle oder auch einen Interrupt) benutzen wollen. Wird die Zugriffsreihenfolge auf dieses Objekt nicht vom Betriebssystem (vom ***Scheduler***) ordentlich kontrolliert, dann kann es zu fatalen Fehlern kommen. Der ***Scheduler*** teilt jedem Task eine bestimmte Prozessor-Zeit und auch die notwendigen Betriebsmittel zu (Code- und Datensegmente im RAM). Dazu verwendet er einen ***Stack***-Speicher, auf dem die einzelnen Tasks in einer Warteschlage eingetragen sind. Zu erwähnen ist noch, dass jeder ***Task*** (=***Prozess***) in verschiedene Ausführungsstränge (***Threads***) zerlegt wird, deren Verwaltung auch dem Scheduler obliegt.

(5) Der Dialog mit dem Benutzer (Communication-Management)

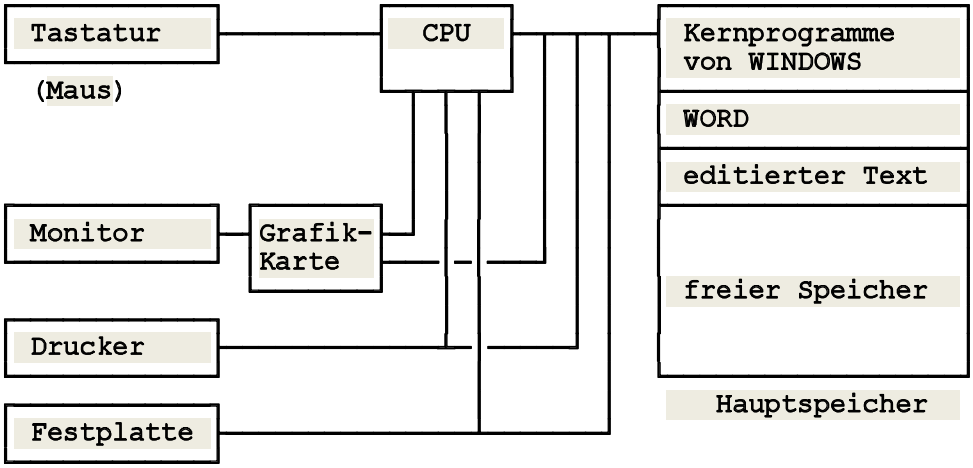
Über eine eigene ***Benutzeroberfläche*** müssen die Botschaften des Benutzers (Ereignisse wie Tastendruck, Mausklick oder Eingaben) erkannt und die entsprechenden Aktionen durchgeführt werden. Solche Ereignisse unterbrechen den Programmablauf und werden daher ***Interrupts*** genannt, die entsprechend verwaltet werden müssen. Betriebssysteme können mit eingetasteten Befehlen im Textmodus oder mit mausbedienten Objektsymbolen im Grafikmodus arbeiten.

MSDOS ist das älteste Betriebssystem der Firma Microsoft, welches befehlsorientiert im Textmodus arbeitet, d.h. die Befehle müssen vom Benutzer über die Tastatur eingegeben werden, was aber oft sehr mühsam ist. Die drei Kernprogramme von MSDOS sind IO.SYS, MSDOS.SYS (versteckt und geschützt) und COMMAND.COM (offen und ungeschützt).

Das Betriebssystem **WINDOWS** hingegen arbeitet objektorientiert und ereignisorientiert im Grafikmodus, wobei die Maus als Eingabegerät eine wichtige Stellung einnimmt. Der ***Windows-Explorer*** übernimmt erstens die Dateiverwaltung und zweitens die Verwaltung der Benutzeroberfläche (***Desktop***), die aus Fenstern (Windows), Icons, Taskleiste und Startmenü besteht. In einer eigenen Datenbank (***Registry***) sind alle wichtigen Systemeinstellungen und Verknüpfungen der installierten Dateien registriert. Die nachfolgende Abbildung zeigt die Vermittlung des Betriebssytems zwischen der Hardware und den Anwenderprogrammen. Fast alle beschriebenen Aufgaben des Betriebssystems werden durch Dienstprogramme erledigt, die man in verschiedene Schichten (***Layers***) aufgliedert und als Kern des Betriebssystems (***Kernel***) zusammenfasst.



Die folgende Abbildung zeigt im rechten Teil die Belegung des Hauptspeichers, wenn mit dem Textverarbeitungsprogramm WORD in WINDOWS ein Text bearbeitet bzw. editiert wird.



Ist dabei der Text mit ev. eingefügten Grafiken (embedded objects) für den noch verbleibenden freien RAM zu groß, dann müssen Daten temporär auf die Festplatte ausgelagert und bei Bedarf wieder eingelagert werden, was als ***Page-Swapping*** (***Paging***) bezeichnet wird. Dadurch wird ein ***virtueller Speicher*** eingerichtet, der dauernd verändert und angepasst wird, was wiederum mit entsprechenden Kontrolleinrichtungen verwaltet wird. Durch die konstanten, kleinen Speicherseiten können größere, unbelegte Speicherbereiche (Fragmentierung) vermieden werden.

Hinweis: Im Gegensatz zu älteren 32-Bit-Prozessoren verwenden 64-Bit-Prozessoren vor allem das ***Paging-Konzept*** anstelle der ***Segmentierung im Protected Mode***. Dabei wird jedem ***Task*** (=Prozess) ein von den anderen Tasks getrennter virtueller Adressraum zur Verfügung gestellt.

4.2 Das Windows Application Programming Interface (WinAPI)

WinAPI ist die Schnittstelle für Windows-Anwendungsprogramme. Mit ihrer Hilfe können vom Programmierer in höheren Programmiersprachen wie C oder DELPHI über eigene Programmroutinen die Dienstprogramme des Betriebssystems aufgerufen werden. Diese Programmroutinen sind in dynamischen Dateibibliotheken (DLLs) gespeichert, beispielsweise in ***kernel32.dll***, die bereits im Betriebssystem integriert sind.

Die nachfolgenden Funktionen sind in der Programmiersprache DELPHI geschrieben und stützen sich auf WinAPI. Sie greifen tief in das Task-Management von WINDOWS ein und sollen dem Leser einen Eindruck über systemnahes Programmieren vermitteln.

Ein praktisches Anwendungsbeispiel dazu: Ein Foto ist als eine Grafikdatei „bild.jpg“ in einem Windows-Ordner abgespeichert. Mit einem doppelten Mausklick auf den Dateinamen wird automatisch ein vorhandenes Grafikprogramm aufgerufen, mit dessen Hilfe das Foto am Bildschirm dargestellt wird. Voraussetzung dafür ist, dass den Grafikdateien mit der Extension „jpg“ das Grafikprogramm, beispielsweise „paint.exe“, zugeordnet (registriert) wurde. Die Grafikdatei nennt man dann „Client“ und das zugeordnete Grafikprogramm heißt „Server“.

Mit der ersten Funktion ***ExecuteFile*** kann beispielsweise aus einem DELPHI-Programm das zu einer Grafikdatei (Client) registrierte Grafikprogramm (Server) aufgerufen und die Grafikdatei geöffnet werden. Die zweite Funktion ***GetExeForFile*** ermittelt das Serverprogramm zu einer Clientdatei. Die dritte Funktion ***KillTask*** beendet ein laufendes Programm.

function ExecuteFile(const FileName, Params, DefaultDir: string; ShowCmd: Integer): THandle;

```
// Eine Datei oder ein Programm "FileName" ausführen
var zFileName, zParams, zDir: array[0..79] of Char;
begin
  Result := ShellExecute(Application.MainForm.Handle,
                         nil,
                         StrPCopy(zFileName, FileName),
                         StrPCopy(zParams, Params),
                         StrPCopy(zDir, DefaultDir),
                         ShowCmd);
end;
```

function GetExeForFile(const FileName: String): String;

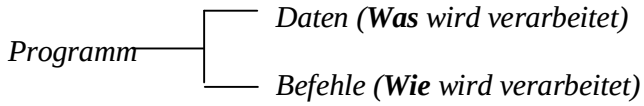
```
// Zu einer Datendatei (Client) das registrierte Windows-Programm (Server) ermitteln
var x: Integer;
begin
  SetLength(Result, MAX_PATH);
  if FindExecutable(PChar(FileName), nil, PChar(Result)) >= 32 then
     SetLength(Result, StrLen(PChar(Result)))
  else Result := IntTostr(x);
end;
```

function KillTask(ExeFileName: string): integer;

```
// Einen laufenden Task "ExeFileName" aus dem Speicher entfernen (beenden)
const PROCESS_TERMINATE = $0001;
var   ContinueLoop: BOOL;
      FSnapshotHandle: THandle;
      FProcessEntry32: TProcessEntry32;
begin
  result := 0;
  FSnapshotHandle := CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  FProcessEntry32.dwSize := Sizeof(FProcessEntry32);
  ContinueLoop := Process32First(FSnapshotHandle,FProcessEntry32);
  while integer(ContinueLoop) <> 0 do begin
    if ((UpperCase(ExtractFileName(FProcessEntry32.szExeFile)) = UpperCase(ExeFileName))
       or (UpperCase(FProcessEntry32.szExeFile) = UpperCase(ExeFileName))) then
          Result := Integer(TerminateProcess(OpenProcess(PROCESS_TERMINATE, BOOL(0),
                                                FProcessEntry32.th32ProcessID), 0));
    ContinueLoop := Process32Next(FSnapshotHandle,FProcessEntry32);
  end;
  CloseHandle(FSnapshotHandle);
end;
```

5 DIE PROGRAMMIERUNG DES COMPUTERS

Am Anfang steht immer ein Problem. Dieses wird vom Programmierer analysiert und ein Lösungsverfahren (Algorithmus) entwickelt. Der Algorithmus wird sodann mit Hilfe einer Programmiersprache in einer Folge von entsprechenden Befehlen formuliert. Die Befehle bilden das Programm und steuern den Computer derart, dass die Problemlösung realisiert wird. Ein Programm enthält daher sowohl die relevanten Daten als auch die entsprechenden Befehle zur Verarbeitung dieser Daten.



Der große Vorteil der Programmierung in einer höheren Computersprache ist die symbolische Codierung. Der Zugriff auf die Daten erfolgt mittels einer symbolischen Adressierung. D.h. der Programmierer kann die Daten über Variablenbezeichner (das sind beliebig wählbare Namen) ansprechen und braucht sich nicht um deren effektive Speicheradressen kümmern. Auch die Befehle werden durch symbolische Schlüsselworte dargestellt und entsprechen meist sehr umfangreichen internen Maschinencodes. Einfache symbolische Adressierung der Daten und mächtige symbolische Befehlsworte kennzeichnen somit eine Hochsprache, die dadurch relativ maschinenunabhängig und komfortabel wird. Damit zwischen Daten und Befehlen keine Verwechslungen auftreten, werden sie in getrennten Bereichen des Hauptspeichers abgelegt.

Vor der Verwendung von Daten in einem Programm muss deren Speicherformat festgelegt werden, damit der Computer den Daten entsprechende Bytes des Speichers zuordnen kann. Will man zum Beispiel eine Berechnung durchführen und das Ergebnis in der Variablen *Resultat* abspeichern, so muss dem Computer am Beginn des Programms mitgeteilt werden, dass die Variable *Resultat* eine reelle Zahl enthalten soll. Dies geschieht im **Definitionsteil** des Programms.

Variable und Konstante (Daten, deren Werte sich während der Laufzeit des Programmes nicht ändern) werden erzeugt, um damit Berechnungen durchzuführen. Zu diesem Zweck muss eine Menge von zulässigen Operationen zur Verfügung stehen (z.B. die Addition "+"). Aus primitiven Operationen werden mächtige Operationsabläufe zusammengesetzt. Das erfolgt im **Ablaufteil** des Programmes. Grundlegende Programmbefehle sind Wertzuweisung und Wertevergleich:

Wertzuweisung: Einer Variablen (***X***) wird der Wert einer anderen Variablen (***Y***) oder ein Operationsergebnis zugewiesen (z.B. ***X := Y*** oder ***X := Y + Z***).

Wertevergleich: Test auf Gleichheit zweier Variablen (z.B. ***X = Y*** oder ***X = Y + Z***). Dabei wird immer der linke Wert mit dem rechten Wert auf Übereinstimmung verglichen.

Das Schreiben der Programmbefehle (Quelltext) erfolgt mit einem komfortablen **Editor**. Die Umwandlung des symbolischen Codes in den maschinenverständlichen Binärcode übernimmt ein eigenes Übersetzungsmodul, der **Compiler**. Mit Hilfe eines **Linkers** werden den symbolischen Adressen echte Speicherplätze zugeordnet. Dann erst ist ein lauffähiges Programm entstanden.

• Ein Schubladen-Modell des Speichers

Einer Variablen sind folgende drei Bestimmungen zugeordnet:

Name: Durch diesen wird der Speicherplatz adressiert (d.h. wo die Variable gespeichert ist).
Typ: Dieser definiert die Struktur des Speicherplatzes (d.h. wie die Bits angeordnet sind).
Wert: Der Inhalt des Speicherplatzes (d.h. Auswertung der Bits, z.B. als Zahl oder Zeichen).

Vereinfacht kann eine solche Variable als eine mit einem Namensschild versehene Schublade im Speicherkasten des Computers aufgefasst werden. Damit laufen in vereinfachter Weise bei einer **Wertzuweisung** ***X := Y + Z*** folgende Arbeitsschritte im Computer ab:

(1) Suche im Speicherkasten die Schublade ***Y***.
(2) Transportiere ihren Inhalt in ein Register in der Zentrale.
(3) Suche im Speicherkasten die Schublade ***Z***.
(4) Transportiere deren Inhalt in ein Register in der Zentrale.
(5) Transportiere die Registerinhalte ins Rechenwerk und führe dort den Additionsbefehl (+) aus.
(6) Stelle dann das Ergebnis in ein zentrales Register zurück.
(7) Transportiere das Rechenergebnis aus dem zentralen Register in die Speicher-Schublade ***X***.

Ein **Wertevergleich** ***X = Y*** kann in vereinfachter Weise folgendermaßen dargestellt werden:

(1) Suche im Speicherkasten die Schublade ***Y***.
(2) Transportiere ihren Inhalt in ein zentrales Speicherregister.
(3) Suche im Speicherkasten die Schublade ***X***.
(4) Transportiere ihren Inhalt in ein zentrales Speicherregister.
(5) Transportiere diese zwei Registerinhalte in das Rechenwerk und vergleiche sie dort. Setze, je nach Ausgang des Vergleiches (gleich, ungleich), ein Statusbit (1 = true, 0 = false) im Statusregister der Zentrale.
(6) Das Vergleichsergebnis kann dann vom Programm im zentralen Statusregister abgelesen und zur Steuerung des weiteren Programmablaufes verwendet werden.

Die symbolischen Programmbefehle werden im Hauptspeicher in einen maschinenverständlichen Binärcode abgelegt und von der Zentrale des Computers (CPU) hintereinander abgearbeitet. Dabei erfolgt die Durchführung eines einzelnen Maschinenbefehls in einem dreiteiligen Zyklus: ***Befehl holen*** (vom Hauptspeicher in die Zentrale), ***Befehl decodieren*** und ***Befehl ausführen***. Die Arbeitsgeschwindigkeit dieses Maschinenzyklus hängt u.a. von der Taktrate der CPU ab.

• Erstes Programmbeispiel „Sortieren von Namen“

Eine Liste von Personennamen wird über die Tastatur in einen dafür reservierten Bereich des Hauptspeichers eingegeben. Dort erfolgt eine alphabetische Sortierung. Zum Schluss wird die sortierte Liste am Drucker ausgegeben.

Der eigentliche Kern des Programmes liegt in der Entwicklung eines geeigneten Sortierverfahrens. Dazu gibt es verschiedene Möglichkeiten: Beispielsweise durchläuft man mehrmals schrittweise die eingespeicherte Liste und vergleicht jedes Listenelement mit seinem Nachfolger. Steht das Element im Alphabet hinter seinem Nachfolger, dann müssen die beiden Elemente in der Liste ihre Plätze tauschen. Verglichen werden dabei die ANSI-Codes der einzelnen Textzeichen der Listenelemente. Am Ende des Verfahrens erhält man eine sortierte Liste.

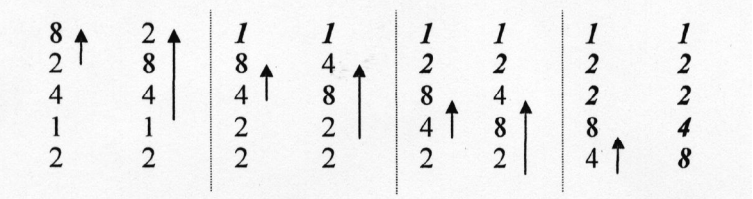
Das Programm zur Problemlösung könnte in folgende Teilschritte (Module) zerlegt werden:

→ Programmbeginn.
→ Reservierung eines Speicherbereichs von Textvariablen (Strings) für die Personennamen.
→ Tastatureingabe der Personennamen und deren Abspeicherung auf die vorher reservierten Textvariablen.
→ Sortierung der Variablenliste im Hauptspeicher mit einem entsprechenden Sortierverfahren. Dieses kann als eigenes Unterprogramm das Hauptprogramm ergänzen.
→ Ausgabe der Textvariablen am Drucker.
→ Programmende.

• Zweites Programmbeispiel „Sortieren von Zahlen“

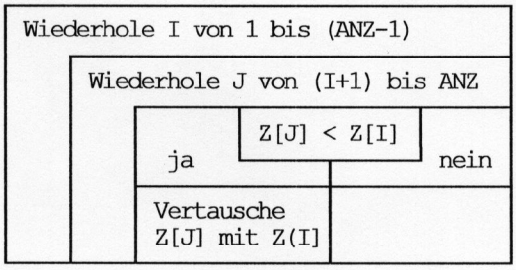
Sortieren und Durchsuchen von Datenbeständen im Hauptspeicher des Computers zählen zu den Grundaufgaben der EDV. Hier soll ein einfaches Sortierverfahren (Sortieren durch Austauschen) ausführlich besprochen werden.

In aufsteigender Folge wird jedes Datenelement mit dem ersten verglichen, und wenn es kleiner als dieses ist, dann wird es mit diesem vertauscht. Wenn der ganze Datenbereich durchlaufen ist, steht das kleinste Element an erster Stelle. In einem zweiten Durchlauf wird das zweitkleinste Element an die zweite Stelle befördert, in einem dritten Durchlauf das drittkleinste Element an die dritte Stelle, usw. Am Ende aller Durchläufe ist der Bereich sortiert. Zur Illustration sollen fünf Zahlen (8, 2, 4, 1, 2) sortiert werden:



Zur Sortierung dieser 5 Zahlen wurden also in 4 Durchläufen genau 7 Vertauschungen vorgenommen.

Struktogramm:



Das nachfolgende Programmlisting ist in der Programmiersprache „DELPHI“ geschrieben und stellt das kurze Unterprogramm „SORT“ dar. Die globale Variable ***Z*** benennt einen indizierten Speicherbereich, wo ***ANZ*** ganze Zahlen abgespeichert sind. ***X, I, J*** sind lokale Variable, welche in den beiden ineinander geschachtelten Wiederholungsschleifen (***for – do***) verwendet werden. Ein zusammengehöriger Programmblock wird immer von (***begin – end***) eingeschlossen. Zentral in der inneren Scheife ist der Vergleichsbefehl (***if – then***). Je nach Vergleichsergebnis werden mithilfe der Variablen ***X*** die zwei indizierten Variablen ***Z[I]*** und ***Z[J]*** getauscht (Dreieckstausch). Als Vergleichsoperator wird < verwendet. Der Befehl ***:=*** weist Variablen ihre Werte zu.

```
procedure Sort(var Z: Bereich; ANZ: Integer);
var X: Integer;
    I,J: Integer;
begin
  for I := 1 to ANZ-1 do begin
    for J := I+1 to ANZ do begin
      if Z[J] < Z[I] then begin
         X := Z[I]; Z[I] := Z[J]; Z[J] := X;
      end;
    end;
  end;
end;
```

- **Drittes Programmbeispiel „Einfügen in eine Liste“**

Eine sortierte Liste von ganzen Zahlen soll vorliegen. Eine neue Zahl soll in diese sortierte Liste an der richtigen Stelle eingefügt werden. Zur Erledigung dieser Aufgabe stellen wir zunächst die Zahl an das Ende der Liste. Dann vergleichen wir – beginnend mit dem letzten Listenelement – jedes Element mit seinem Vorgänger. Ist das Element kleiner als sein direkter Vorgänger, so tauschen die beiden Platz. Dadurch ist das Element um eine Stelle nach vor gerückt. Dieses Verfahren wird dann abgebrochen, wenn das Element größer oder gleich seinem Vorgänger ist. Dann ist die neue Zahl an der richtigen Stelle eingefügt.

Das Problem und seine Lösung

Unsere Beispiele demonstrieren sehr anschaulich das allgemeine Schema von Problemlösungen:

[1] Es existiert ein unerwünschter Anfangszustand ***AZ***.

[2] Der Problemlöser hat einen erwünschten Zielzustand ***ZZ*** vor Augen.

[3] Der Problemlöser sucht nach einer Transformation, die aus einer Folge von Operationen besteht, welche den Anfangszustand über Zwischenstufen in den Zielzustand überführen (***AZ → ZZ***).

Meistens ist leider die Problemlösung nicht sofort einsehbar (evident), sondern sie wird durch verschiedene Barrieren erschwert. Beispielsweise könnte der Zielzustand zu grob und zu wenig präzise formuliert sein oder es stehen keine zielführenden Operationen zur Verfügung.

Die Entwicklung eines Programmes zur Lösung von Problemen erfolgt in bestimmten Schritten:

PROBLEMSTELLUNG
LÖSUNGSENTWURF
PROGRAMMIERUNG
PROGRAMMTESTUNG

Der wesentliche Schritt dabei ist der Programmentwurf. Er gliedert sich in zwei Abschnitte:

→ **Analyse des Problems**

Analyse der Ausgabedaten (Was will ich erreichen?)
Analyse der Eingabedaten (Was ist vorgegeben?)
Analyse des Lösungsverfahrens (Algorithmus: Wie erreiche ich das Ziel?)

→ **Darstellung des Lösungsverfahrens**

in umgangsprachlicher Form
als Struktogramm
als Flussdiagramm

Die wichtigsten Richtlinien für einen strukturierten Programmentwurf sind der ***Top-Down-Entwurf*** und die ***Modularisierung***. Darunter versteht man einerseits die schrittweise Verfeinerung der Problemlösung (vom Groben zum Feinen) und andererseits die Gliederung in verschiedene, wohldefinierte Teilbereiche (Module).