Gerd Küveler
Dietrich Schwoch

Informatik für Ingenieure und Naturwissenschaftler 2

PC- und Mikrocomputer-technik, Rechnernetze

5. Auflage

vieweg

Viewegs Fachbücher der Technik

Gerd Küveler
Dietrich Schwoch

Informatik für Ingenieure und Naturwissenschaftler 2

Aus dem Programm
Elektro- und Informationstechnik

Elemente der angewandten Elektronik
von E. Böhmer

Digitaltechnik
von K. Fricke

Informatik für Ingenieure und Naturwissenschaftler 1
von G. Küveler und D. Schwoch

Bussysteme in der Automatisierungs- und Prozesstechnik
herausgegeben von G. Schnell und B. Wiedemann

Aufgabensammlung Elektrotechnik 1 und 2
von M. Vömel und D. Zastrow

Digitale Schnittstellen und Bussysteme
von F. Wittgruber

Automatisieren mit SPS
Theorie und Praxis
von G. Wellenreuther und D. Zastrow

Automatisieren mit SPS
Übersicht und Übungsaufgaben
von G. Wellenreuther und D. Zastrow

Steuerungstechnik mit SPS
von G. Wellenreuther und D. Zastrow

Mikroprozessortechnik
von K. Wüst

vieweg

Gerd Küveler
Dietrich Schwoch

Informatik für Ingenieure und Naturwissenschaftler 2

PC- und Mikrocomputertechnik, Rechnernetze

5., vollständig überarbeitete
und aktualisierte Auflage

Viewegs Fachbücher der Technik

vieweg

Bibliografische Information Der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über
<http://dnb.d-nb.de> abrufbar.

1. Auflage 1996
2., vollständig überarbeitete Auflage 1999
3., vollständig überarbeitete und erweiterte Auflage Januar 2001
4., durchgesehene und erweiterte Auflage April 2003
5., vollständig überarbeitete und aktualisierte Auflage Februar 2007

Lektorat: Reinhard Dapper

Der Vieweg Verlag ist ein Unternehmen von Springer Science+Business Media.
www.vieweg.de

Umschlaggestaltung: Ulrike Weigel, www.CorporateDesignGroup.de
Technische Redaktion: Hartmut Kühn von Burgsdorff, Wiesbaden
Druck und buchbinderische Verarbeitung: Wilhelm & Adam, Heusenstamm
Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier.
Printed in Germany

ISBN 978-3-8348-0187-6

Vorwort

Erstmals mit der neuen Auflage erscheint das vorliegende Werk in geteilter Form. Das eröffnet zum einen die Möglichkeit zu größerer Ausführlichkeit, zum anderen ist der Leser nicht gezwungen, viele Seiten zu erwerben, die ihn persönlich nicht interessieren. Zunächst wurden zwei Bände aufgelegt. Der erste Band führt in die Hochsprachen-Programmierung mit C/C++ ein, weil diese sich im Laufe der letzten Jahre zur bedeutendsten Universalsprache mit breitestem Anwendungsspektrum entwickelt hat. Der hier vorliegende zweite Band befasst sich mit PC- und Mikrocomputer-Technik sowie Computernetzwerken. Während der Band 1, zumindest was das Fortgeschrittenen-Praktikum angeht, Grundkenntnisse der höheren Mathematik voraussetzt, eignet sich der vorliegende Band auch für Techniker und Technikerschulen sowie, bezüglich des Teils II, für System-Administratoren im nicht-technischen Bereich.

Die Funktionsweise von Mikrorechnern am Beispiel des PC und seines Prozessors 80(X)86 sowie seiner Betriebssysteme ist das Kernthema des Teil I. Der Zugang erfolgt von der Softwareseite her. Die Beherrschung der Sprache C/C++ auf dem Niveau des Band 1 (ohne Objektorientierte Programmierung) wird vorausgesetzt. Kenntnisse der Maschinensprache werden für die 16-Bit-Welt anhand des Hilfsprogramms DEBUG und für die 32-Bit-Welt anhand des C++-Inline-Assemblers vermittelt.

Der Teil II bietet einen ersten Einstieg in das wichtige Gegenwarts- und Zukunftsthema Computernetzwerke. Fast jeder Rechner wird heute vernetzt betrieben und selbst private PC können weltweit kommunizieren. Durch die Möglichkeit der lokalen und globalen Rechnervernetzungen erleben wir die Entstehung einer neuen Dimension in der Daten- und Informationsverarbeitung. Die Beschäftigung mit Rechnernetzen ist daher nicht länger eine Domäne von Informatikspezialisten. Dieses Thema wurde längst in die Informatik-Lehrpläne aller technisch orientierten Ausbildungsgänge einbezogen.

Einige ausführliche „Anhänge“ ergänzen und vertiefen die im Hauptteil des Buches behandelten Themen oder dienen einfach zum Nachschlagen. Um den Preis des Buchs erträglich zu halten, wurden sie ausgegliedert, d. h. in unsere unten stehende Webseite zum Buch gestellt Die Quelltexte der im Buch abgedruckten Programme, die Lösungen zu den Übungsaufgaben, einige Tools sowie aktuelle Ergänzungen und Fehlerkorrekturen finden Sie im Internet unter

http://r5.mnd.fh-wiesbaden.de/infobuch2

Achtung: ohne „www“!

Sie haben dort auch die Möglichkeit, uns eine E-Mail zu schicken. Über Hinweise auf Fehler, Anregungen und Kritiken würden wir uns freuen.

Unser Dank gilt allen, die einen Beitrag zum Zustandekommen dieses Buches geleistet haben.

Glashütten im Januar 2007 *Gerd Küveler*
Dieburg im Januar 2007 *Dietrich Schwoch*

Inhaltsverzeichnis

Teil I: PC- und Mikrocomputer-Technik

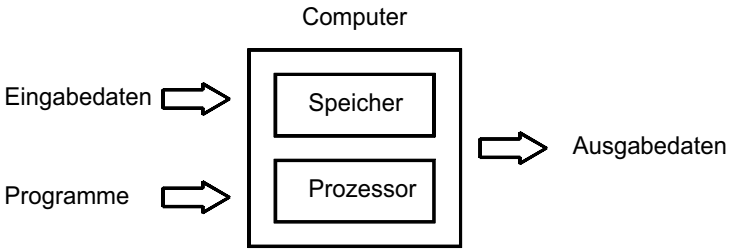
Teil I: PC- und Mikrocomputer-Technik

Der PC stellt für die weitaus meisten „Benutzer“ das Tor zur Welt der Bits und Bytes dar. Der funktionale Aufbau und die Arbeitsweise von Mikrorechnern werden deshalb am Beispiel der 80(x)86-Prozessorfamilie von INTEL behandelt, weil diese durch ihren Einsatz im „Personel Computer“ (PC) zum Industriestandard wurde und entsprechend weit verbreitet ist. Sie benötigen zur Bearbeitung der praktischen Übungen dieses Teils lediglich ihren PC unter einem Windows-Betriebssystem und den C/C++-Compiler, der Sie ggf. bereits durch den Band 1 begleitet hat. Wer an UNIX/Linux interessiert ist, sollte auch dieses installiert haben oder eine bootfähige Live-CD, wie Knoppix, verwenden. Eine über das übliche hinausgehende, zusätzliche Hard- oder Software ist nicht erforderlich.

Der Schwerpunkt liegt auf dem softwaremäßigen Zugang zum PC. Dazu gehört auch eine kleine Einführung in die Maschinensprache, die als „Muttersprache“ des Computers die Hardwarekomponenten unmittelbar erschließt. Maschinensprachen setzen allerdings Grundkenntnisse über die Funktionsweise der Hardware voraus, die hier ebenfalls vermittelt werden. Maschinensprachen sind hardwareabhängig, d. h. von Prozessor zu Prozessor verschieden. Die Strukturen ähneln sich jedoch sehr weitgehend, so dass die hier erworbenen Kenntnisse gut übertragbar sind, z. B. auf die Programmierung von Mikrocontrollern (µC). Wir gehen von dem 16-Bit-Intel-Prozessor 8086 (der Ur-PC-Prozessor) aus, zu dem auch ein Pentium IV noch kompatibel ist. Zur Lösung zahlreicher praktischer Probleme greifen wir jedoch auf Ihre C/C++-Kenntnisse zurück. Über den Inline-Assembler des C++-Compiles erschließt sich auch die 32-Bit-Welt. Bevor Sie beginnen, einen Mikrorechner auf hardware- und systemnaher Ebene zu programmieren, sind einige theoretische Vorkenntnisse über Bits, Bytes und Co. erforderlich.

1 Interne Darstellung von Informationen

Ein Mikrocomputersystem speichert und verarbeitet mehr oder weniger große Informationsmengen, je nach Anwendung und Leistungsfähigkeit.



Unter einem Mikrocomputer versteht man einen Rechner, bei dem alle Funktionen des Prozessors auf einem integrierten Baustein (*chip*), dem Mikroprozessor, vereinigt sind. Je nach Einsatzgebiet reicht die Spanne von Mikrocontrollern, das sind einfache Einplatinencomputer für Mess- und Steueraufgaben, bis zu komfortabel ausgestatteten PC und Workstations.

Die hardwarenahe Programmierung derartiger Rechner setzt Kenntnisse über die Darstellung der Informationen voraus.

Wegen der verwendeten Bauelemente eines Mikrorechners werden alle Informationen binär (dual) codiert. Je nach Art und eventuell auch geplanter Verarbeitung der Information verwendet man unterschiedliche Codes. So werden Zahlen in der Regel anders dargestellt als Zeichen, natürliche Zahlen z. B. im reinen Binärcode, Zeichen im ASCII-Code. Die Art der Verarbeitung kann jedoch auch eine Zahlendarstellung im BCD- oder gar im ASCII-Code nahe legen. Codes beruhen auf Vereinbarungen, die per Software getroffen werden.

1.1 Darstellung positiver ganzer Zahlen

Zahlen werden im täglichen Leben im Dezimalsystem dargestellt. Das gilt erst recht für ganze Zahlen. Das Dezimalsystem ist ein Stellenwertsystem. Für die Darstellung ganzer Zahlen in Stellenwertsystemen gilt:

$$X = \sum a_i b^i \,,\; i = 0,1,...$$

$$a_i \in A,\; A = \{0,1,2,...,b-1\}$$

b ist die Basis des Stellenwertsystems

■ **Beispiel** Zahl im Dezimalsystem (Basis = 10)

1398

$= 1 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 8 \times 10^0$ ■

In Zusammenhang mit einem Digitalrechner sind drei Stellenwertsysteme relevant:

- das Dezimalsystem
- das Binärsystem
- das Hexadezimalsystem

Die Bedeutungen der beiden ersten Systeme sind klar, das Hexadezimalsystem wird für eine verkürzte Darstellung von Binärwerten herangezogen, weil sich vier Binärziffern zu einer Hexadezimalziffer zusammenfassen lassen. Früher wurde statt des Hexadezimalsystems häufig das Oktalsystem (Zusammenfassung von drei Binärziffern) verwendet, inzwischen ist es jedoch „aus der Mode“ gekommen.

Mit der Darstellung von Binär- und Hexadezimalzahlen muss der harwarenah operierende Programmierer ebenso vertraut sein, wie mit der Umrechnung zwischen den Stellenwertsystemen.

1.1.1 Binär- und Hexadezimalsystem

Das Binärsystem (Dualsystem) besitzt folgende wichtige Eigenschaften:

Basis:	2
Menge der Ziffern:	{0,1}
Stellenwerte:	Potenzen von 2

2^0	2^1	2^2	2^3	2^4	...
1_{10}	2_{10}	4_{10}	8_{10}	16_{10}	...

Eine binäre Ziffer entspricht in der Datenverarbeitung einem Bit, mit dem sich zwei Zustände (0 und 1) unterscheiden lassen. Folgende „magische“ Zweierpotenzen benutzt man v. a., um die Kapazität von Rechnerspeichern anzugeben:

1 <K>	$= 2^{10}$	$= 1\,024_{10}$	(<Kilo>)
1 <M>	$= 2^{20}$	$= 1\,048\,576_{10}$	(<Mega>)
1 <G>	$= 2^{30}$	$= 1\,073\,741\,824_{10}$	(<Giga>)
1 <T>	$= 2^{40}$	$= 1\,099\,511\,627\,776_{10}$	(<Tera>)

Die Definitionen von Kilo, Mega, Giga, Tera, usw. unterscheiden sich also in der Datenverarbeitung ein wenig von der üblichen Bedeutung (1 Kilo = 10^3 = 1000, usw.).

Das Hexadezimalsystem besitzt folgende wichtige Eigenschaften:

Basis:	16
Menge der Ziffern:	{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
Bemerkung:	Die Ziffern A bis F besitzen die dezimalen Nennwerte zehn bis fünfzehn
Stellenwerte:	Potenzen von 16
	16^0 16^1 16^2 16^3 ...
	1_{10} 16_{10} 256_{10} 4096_{10} ...

Im Hexadezimalsystem ist AFFE keine Beleidigung, sondern eine gültige Zahl. Um Missverständnisse zu vermeiden, sollte man rechts unten die Basis als Index anfügen ($AFFE_{16}$).

1.1.2 Umrechnungsverfahren

Zwischen den für uns relevanten Stellenwertsystemen Dezimal, Binär und Hexadezimal gibt es sechs Umrechnungsverfahren, die nun anhand von Beispielen vorgestellt werden.

Umrechnung: Binär (Dual) -> Dezimal

Methode: Summe der Zweierpotenzen bilden

Beispiel 1: `10111`

$$= 1\cdot 2^0 + 1\cdot 2^1 + 1\cdot 2^2 + 0\cdot 2^3 + 1\cdot 2^4$$
$$= 1 + 2 + 4 + 16$$
$$= 23_{10}$$

Beispiel 2: `1100101`

$$= 1\cdot 2^0 + 0\cdot 2^1 + 1\cdot 2^2 + 0\cdot 2^3 + 0\cdot 2^4 + 1\cdot 2^5 + 1\cdot 2^6$$
$$= 1 + 4 + 32 + 64$$
$$1\cdot 2^0 + 0\cdot 2^1 + 1\cdot 2^2 + 0\cdot 2^3 = 101_{10}$$

Bemerkung: Es ist sinnvoll, mit der Umrechnung rechts zu beginnen.

Hilfstabelle: Zweierpotenzen	
n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1 024
11	2 048
12	4 096
13	8 192
14	16 384
15	32 768
16	65 536
20	1 048 575
24	16 777 216
32	4 294 967 296

Umrechnung: Hexadezimal -> Dezimal

Methode: Summe der Sechzehnerpotenzen bilden

Beispiel 1: $\texttt{1E3}_{16}$

$= 3\cdot16^0 + 14\cdot16^1 + 1\cdot16^2$

$= 3 + 224 + 256$

$= 483_{10}$

Beispiel 2: $\texttt{AFFE}_{16}$

$= 14\cdot16^0 + 15\cdot16^1 + 15\cdot16^2 + 10\cdot16^3$

$= 14 + 240 + 3840 + 40960$

$= 45054_{10}$

Bemerkung; Es ist sinnvoll, mit der Umrechnung rechts zu beginnen.

Hilfstabelle: Sechzehnerpotenzen	
n	16
0	1
1	16
2	256
3	4 096
4	65 536
5	1 048 576
6	16 777 216
7	268 435 456
8	4 294 967 296

Umrechnung: Dezimal -> Binär

Methode: Fortgesetzte Division mit Rest

Beschreibung: Die umzuwandelnde Dezimalzahl wird durch 2 dividiert, der Quotient und der Divisionsrest (0 oder 1) wie im Beispiel unten notiert. Nun wird der Quotient durch 2 dividiert, usw. (siehe Beispiel). Die Umrechnung ist beendet, sobald der Quotient den Wert 0 erreicht. Die Divisionsreste von unten nach oben notiert ergeben die gesuchte Binärzahl.

Beispiel 1:

```
223 : 2 =  111 Rest 1
111 : 2 =   55 Rest 1
 55 : 2 =   27 Rest 1
 27 : 2 =   13 Rest 1
 13 : 2 =    6 Rest 1
  6 : 2 =    3 Rest 0
  3 : 2 =    1 Rest 1
  1 : 2 =    0 Rest 1
```

Die entsprechende Binärzahl lautet: 1 1 0 1 1 1 1 1

Beispiel 2:

```
763 : 2 =  381 Rest 1
381 : 2 =  190 Rest 1
190 : 2 =   95 Rest 0
 95 : 2 =   47 Rest 1
 47 : 2 =   23 Rest 1
 23 : 2 =   11 Rest 1
 11 : 2 =    5 Rest 1
  5 : 2 =    2 Rest 1
  2 : 2 =    1 Rest 0
  1 : 2 =    0 Rest 1
```

Die Binärzahl lautet: 1 0 1 1 1 1 1 0 1 1

Umrechnung: Dezimal -> Hexadezimal

Methode: Fortgesetzte Division mit Rest

Beschreibung: Die umzuwandelnde Dezimalzahl wird durch 16 dividiert, der Quotient und der Divisionsrest (0 bis F) wie im Beispiel unten notiert. Nun wird der Quotient durch 16 dividiert, usw. (siehe Beispiel). Die Umrechnung ist beendet, sobald der Quotient den Wert 0 erreicht. Die Divisionsreste von unten nach oben notiert ergeben die gesuchte Hexadezimalzahl.

Beispiel 1:

```
443 : 16 =   27 Rest 11 ---------------------------+
 27 : 16 =    1 Rest 11 ------------------+        |
  1 : 16 =    0 Rest  1 -------------+    |        |
                                     |    |        |
Die entsprechende Hexadezimalzahl lautet:  1    B    B
```

Beispiel 2:

```
999 : 16 =   62 Rest  7 ---------------------------+
 62 : 16 =    3 Rest 14 ------------------+        |
  3 : 16 =    0 Rest  3 -------------+    |        |
                                     |    |        |
Die entsprechende Hexadezimalzahl lautet:  3    E    7
```

Hilfstabelle: Vielfache von Sechzehn

n	n ·16
1	16
2	32
3	48
4	64
5	80
6	96
7	112
8	128
9	144
10	160

Hilfstabelle: Alphabetische Hexadezimalziffern

Hexadezimalziffer:	A	B	C	D	E	F
Dezimaler Wert:	10	11	12	13	14	15

Das folgende C++-Programm rechnet die eingegebene Dezimalzahl in das ebenfalls einzugebende Ziel-Stellenwertsystem um. Bis zu welchem Stellenwertsystem liefert das Programm interpretierbare Ergebnisse?

```
// Rechnet Dezimalzahl in ein beliebiges Zielsystem um.
// Benutzt die rekursive Funktion "dest()".
#include <iostream.h>
#include <stdio.h.h>
using namespace std;
void dest(long int dec, int x);
int main(void)
{
  long int deci;
  int  base;
  cout << "Abbruch: Basis = 0 eingeben" << endl;
  do
  {
    cout << "Dezimalzahl >";
    cin >> deci;
    cout << "Basis       >";
    cin >> base;
    dest(deci, base);
    cout << endl;
  }
  while(base);
  getchar();
  return 0;
}

void dest(long int dec, int x)
{
  int ziff;
  if(x > 0)
  {
    if(dec >= x) dest(dec / x, x);
    ziff = (dec % x) + 48;
    if(ziff > '9') ziff = ziff + 7;
    cout << ((char)ziff);
  }
}
```

Umrechnung: Binär -> Hexadezimal	
Methode:	4er Bündelung
Beschreibung:	Die umzuwandelnde Binärzahl wird von rechts nach links ziffernweise in 4er Bündeln zusammengefasst. Jedes Bündel für sich wird in die entsprechende Hexadezimalziffer umgewandelt.
Beispiel 1:	111 | 0010 | 1110 | 1101 7 2 E D
Beispiel 2:	1 | 1111 | 0101 | 1010 | 0011 | 1100 | 1011 1 F 5 A 3 C B

Hilfstabelle: Binäre Ziffernbündel und Hexadezimalziffern

0000	0001	0010	0011	0100	0101	0110	0111
0	1	2	3	4	5	6	7

1000	1001	1010	1011	1100	1101	1110	1111
8	9	A	B	C	D	E	F

Umrechnung: Hexadezimal -> Binär

Methode: 4er Entbündelung

Beschreibung: Die umzuwandelnde Hexadezimalzahl wird von rechts nach links ziffernweise in je 4 Binärziffern „entbündelt“.

Beispiel 1:

```
 D      F      0      3
 |      |      |      |
1101   1111   0000   0011
```

Beispiel 2:

```
4       7       E       D      1      B      A
|       |       |       |      |      |      |
100   0111    1110    1101   0001   1011   1010
```

(Führende Nullen werden weggelassen.)

Der Darstellungsbereich für Zahlen im Rechner ist im Gegensatz zur Mathematik endlich, begrenzt durch die Speicherkapazität. In der Praxis wird man jedoch nicht den gesamten Speicher zur Darstellung *einer* Zahl verwenden. Vielmehr begnügt man sich mit einer bestimmten Anzahl von Bits. Folgende Bitgruppierungen spielen häufig eine Rolle:

1 Halbbyte = 4 Bit
= 1 **Nibble**
entspricht einer Hexadezimalziffer

1 Byte = 8 Bit
kleinste vom Rechner adressierbare Speichereinheit,
speichert z. B. ein ASCII-Zeichen
entspricht dem C-Datentyp **`char`**

1 Wort = 16 Bit
speichert z. B. eine Ganzzahl
entspricht dem C-Datentyp **short int**

1 Doppelwort = 32 Bit
entspricht als Ganzzahl dem C-Datentyp **`long int`**,
speichert aber auch eine Fließkommazahl vom Typ **float**

Nach einer anderen Definition ist ein *Wort* eine maschinenabhängige Größe. Zur Unterscheidung nennen wir diese Größe

1 Maschinenwort = Anzahl der Bits, die ein gegebener Rechner mit einem Maschinenbefehl, z. B. einer Ganzzahladdition, maximal verarbeiten kann (entspricht der maximalen Registerbreite, s. u.).

Beträgt die Maschinenwortbreite (Wortlänge) eines Mikrorechners beispielsweise 16 Bit, so gilt für die Darstellung einer binären positiven Ganzzahl:

$$X = \sum a_i 2^i \,,\; i = 0,1,...,15$$

$$a_i \in A,\; A = \{0,1\}$$

Der darstellbare Bereich liegt folglich zwischen 0 und FFFF hexadezimal bzw. 0 und 65535 dezimal.

1.1.3 Rechnen im Dualsystem

Beim Rechnen mit Dualzahlen in einem System vorgegebener Wortlänge kann es vorkommen, dass das Rechenergebnis nicht in einem Maschinenwort unterbringbar ist. Der Prozessor setzt in diesem Fall das Carry Bit in einem speziellen Flagregister (→ s. Kap. 2.2.1.5) auf „1“. Das Carry Bit C ist eins von mehreren Flag Bits, die jederzeit durch spezielle Befehle abgefragt werden können.

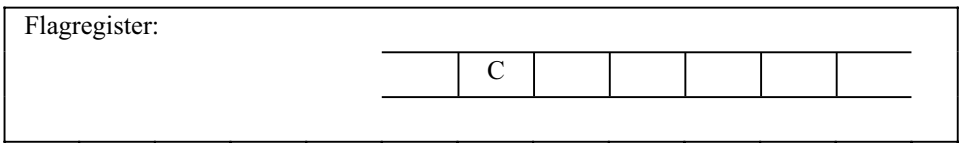
■ **Beispiel Addition im 8-Bit-System**

178 + 204 (dezimal)

binär:

```
    1 0 1 1 0 0 1 0   (178)
  + 1 1 0 0 1 1 0 0   (204)
-------------------
(1) 0 1 1 1 1 1 1 0
```

Carry Bit



■

Bei der Subtraktion wird C gesetzt, wenn eine größere von einer kleineren Zahl abgezogen wird.

Multiplikation und Division mit Dualzahlen lassen sich aus den Regeln des Dezimalsystems übertragen. Besonders einfach ergeben sich Multiplikationen und Divisionen mit 2, die durch Verschiebeoperationen erreicht werden können:

■ **Beispiel**

Multiplikation mit 2: `1 0 1 1 -> 1 0 1 1 0`

Division durch 2: `1 0 1 1 -> 0 1 0 1` (Rest 1) ■

Allgemein gilt:

Multiplikation / Division mit 2er-Potenzen	
Multiplikation mit 2^n :	Verschieben des Bitmusters um n Stellen nach links
Division durch 2^n :	Verschieben des Bitmusters um n Stellen nach rechts

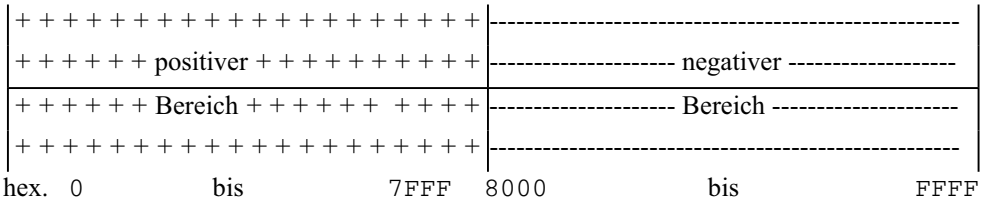
1.2 Darstellung von vorzeichenbehafteten Ganzzahlen

Für viele Anwendungen muss der Bereich der Zahlen um die negativen Ganzzahlen erweitert werden.

Negative ganze Zahlen werden im Rechner in einer speziellen Weise dargestellt.

Diese spezielle Darstellung ergibt sich aus der Forderung, im Rechner das gleiche Addierwerk für vorzeichenlose und vorzeichenbehaftete Zahlen einzusetzen.

Dazu wird der zur Verfügung stehende Bereich in zwei gleich große Portionen unterteilt. Bei 16 Bit Wortlänge sieht das folgendermaßen aus:

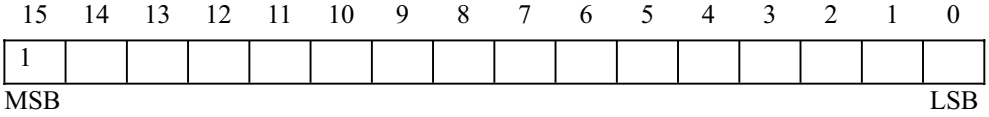


Die Erweiterung um die negativen Zahlen geht also auf Kosten des absolut darstellbaren Bereichs. Er beträgt bei 16 Bit Wortlänge:

$$-32768,...,0,...,+32767$$

Die Null wird den positiven Zahlen zugerechnet.

Negative Zahlen erkennt man daran, dass das MSB (*Most Significant Bit* den Wert „1“ besitzt. LSB bedeutet *Least Significant Bit*.



Negative Zahl -> MSB = 1

Positive Zahl -> MSB = 0

Diese Darstellung ist also wortlängenabhängig! Im Rechner geschieht dies durch die Angabe der Wortlänge, d. h. durch die Angabe der Anzahl der Bits, die zur Darstellung einer Zahl verfügbar ist (n-Bit-System).

Allgemeine Regel:

Schritt 1:		Absolutbetrag der gesuchten Zahl im n-Bit-Sytem
Schritt 2:		Bitmuster komplementieren (1er-Komplement)
Schritt 3:	+ 1	addieren
⟶		negative Zahl

Daher nennt man diese Art der Zahlencodierung 2er-Komplement-Darstellung.

Die Berechnung lässt sich, v. a. im Hexadezimalsystem, auch so durchführen (Regel b):

Schritt 1:		„größte“ Zahl bei gegebener Wortlänge
Schritt 2:	–	Absolutbetrag der gesuchten Zahl
Schritt 3:	+ 1	
⟶	=	2er-Komplement

■ **Beispiel 1 Darstellung von -1_{10} in einem 16-Bit-System:**

a) binär:

```
|1|:                0000 0000 0000 0001
komplementieren:    1111 1111 1111 1110
+1:                                 1 +
-1:                 1111 1111 1111 1111
```

b) hexadezimal:

```
„größte“ Zahl:   F F F F
                –
                +       1
                ---------
                 F F F F
```

■

■ **Beispiel 2 Darstellung von -200_{10} in einem 16-Bit-System:**

a) binär:

```
|200|:          0000 0000 1100 1000
komplementieren: 1111 1111 0011 0111
+1:                                1
-200:            1111 1111 0011 1000
```

b) hexadezimal:

```
„größte“ Zahl:   F F F F
                -     C 8
                ---------
                 F F 3 7    (Zwischenergebnis)
                +       1
                ---------
                 F F 3 8
```

■

■ **Beispiel 3 Darstellung von -100$_{10}$ in einem hypothetischen 10-Bit-System:**

a)binär:

```
|100|:          00 0110 0100
komplementieren: 11 1001 1011
+1:                        1 +
-100:            11 1001 1100
```

b)hexadezimal:

```
„größte“ Zahl:   3 F F
              -    6 4
                 3 9 B   (Zwischenergebnis)
              +      1
                 3 9 C
```

■

■ **Beispiel 4 Darstellung von -100$_{10}$ in einem 64-Bit-System:**

Es wird das Bitmuster von Beispiel 3 auf 64-Bit „erweitert“:

a)binär:

⇒ -100: **11... ...1 1001 1100**

56 führende Einsen

b)hexadezimal:

⇒ -100: **FFFFFFFFFFFFFF9C** ■

Aus dem letzten Beispiel wird die Regel erkennbar, die anzuwenden ist, wenn die Wortlänge für eine Zahlendarstellung vergrößert wird:

positive Zahl == (MSB = 0) -> führende Stellen mit „0“ auffüllen,
negative Zahl == (MSB = 1) -> führende Stellen mit „1“ auffüllen.

Die Rückrechnung kann mit den gleichen Schritten durchgeführt werden. Dies ist eine Folge des zugrunde liegenden Binärsystems.

■ **Beispiel Welche vorzeichenbehaftete Dezimalzahl ergibt FF38 bei einer Wortlänge von 16 Bit?**

MSB=1 -> „-“

a)binär:

```
FF38:            1111 1111 0011 1000
komplementieren: 0000 0000 1100 0111
+1:                               1 +
|200| ⇐         0000 0000 1100 1000
```

gesuchte Zahl: - 200$_{10}$

b) hexadezimal:

„größte“ Zahl:

```
    F F F F
-   F F 3 8
  -------------
    0 0 C 7   (Zwischenergebnis)
+         1
  -------------
      C 8
```

gesuchte Zahl: -200_{10}

■

Durch die 2er-Komplement-Darstellung wird eine Subtraktion auf eine Addition zurückgeführt, d. h. das Rechenwerk behandelt eine Subtraktion wie eine Addition:

■ **Beispiel**

13_{10} - 21_{10} [= - 8_{10}]

(dez) ⇒ 13 + (- 21)
2er-Kompl. (hex) ⇒ D + F F E B = F F F 8

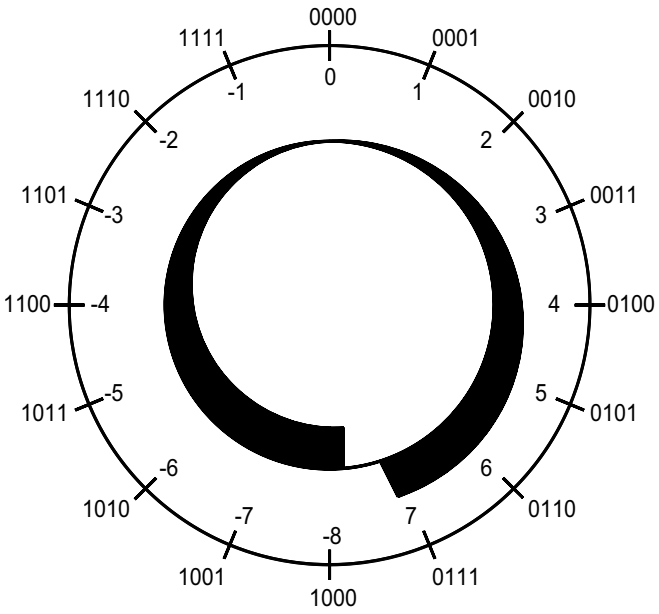
Interpretieren nach 2er-Komplement-Regel ⇒ - 8_{10}

■

Systematik von Zahlen mit Vorzeichen im n-Bit-System	
Zahl (dez)	**Binärmuster**
+/-0 :	0.......0000
+1 :	0.......0001
größte positive Zahl:	0111....1111
absolut größte negative Zahl:	1000....0000
- 1 :	1111....1111

Die Punkte stehen für beliebig viele Bit.

Die folgende Abbildung demonstriert den geschlossenen Zahlenkreis an einem 4-Bit-System.



Vorzeichenbehaftete Ganzzahlen entsprechen dem Hochsprachen-Datentyp `Integer` bzw. **`int`** (für die Sprache C).

Subtraktionen werden auf die Addition zurückgeführt. Rechenergebnisse sind falsch, wenn Bereichsüberläufe stattgefunden haben. Bereichsüberläufe werden durch Setzen des „*Overflow Bit*“ (O) im Flagregister des Prozessors festgehalten. Eine Bereichsüberschreitung bei gegebener Wortlänge erfolgt im Fall eines Vorzeichenumschlags (MSB) bei Addition von zwei positiven bzw. von zwei negativen Zahlen. Erfolgt dagegen ein Übertrag, d. h. reicht die Wortlänge zur Aufnahme des Ergebnisses nicht aus, wird das Carry Flag (C) gesetzt.

■ **Beispiel 1** 4-Bit-System

-> darstellbarer Zahlenbereich: -8...0...+7

Berechnung von 6 + 3

		Vz				
6:		0	1	1	0	
+3:	+	0	0	1	1	
		1	1			(Übertrag in Vz hinein)
+9		1	0	0	1	⇒ -7 FALSCH!

⇒ Vorzeichenumschlag, kein Übertrag

Flagregister:

		O				

Vz bedeutet Vorzeichen. ■

■ **Beispiel 2 Berechnung von -6 - 3 = (-6) + (-3)**

		Vz			
-6:		1	0	1	0
+(-3):	+	1	1	0	1
	1				
-9	(1)	0	1	1	1

(Übertrag aus VZ heraus)
⇒ +7 FALSCH!

⇒ Vorzeichenumschlag und Übertrag

Flagregister:

	C	O				

■

■ **Beispiel 3 Berechnung von +5 - 2 = (+5) + (-2)**

		Vz			
5:		0	1	0	1
+(-2):	+	1	1	1	0
	1	1			
+3	(1)	0	0	1	1

(zweiseitiger Übertrag an Vz)
richtig

⇒ kein Vorzeichenumschlag, aber Übertrag

Flagregister:

	C					

■

■ **Beispiel 4 Berechnung von +3 + 4**

		Vz			
3:		0	0	1	1
+4:	+	0	1	0	0
+7		0	1	1	1

(kein Übertrag)
richtig

⇒ kein Vorzeichenumschlag, kein Übertrag

Flagregister:

■

Merke

Zahlen ohne Vorzeichen: C beachten
Zahlen mit Vorzeichen: O beachten

ACHTUNG

Ein Overflow-Auftreten wird bei Integer-Arithmetik in Hochsprachen-Compilern häufig nicht abgeprüft, so dass ohne Warnung ein falsches Ergebnis resultiert!

Sie können diese Aussage mit einem kleinen C++-Programm überprüfen:

```
// PROGRAMM summe
// Berechnet die Summe von 1 bis N
// und laeuft ueber, wenn N zu gross wird
#include <stdio.h>
#include <iostream.h>
#include <iomanip.h>
using namespace std;
int main(void)
{
  short n, lauf, sum;  // 16 Bit !
  cout << "Grenzzahl N >";
  cin >> n;
  sum = 0;
  lauf = 1;
  do
    sum += lauf++;
  while(lauf <= n);
  cout << "Die Summe betraegt " << setw(5) << sum << endl;
  getchar();
  return 0;
}
```

Geben Sie N so ein, dass eine Summe größer als 32767 resultiert, erhalten Sie – ohne Warnung – ein falsches Ergebnis.

1.3 Darstellung gebrochener Zahlen

Gebrochene Zahlen entsprechen den C-Datentypen **float** oder **double**.

Gebrochene Zahlen werden in der Gleitkommadarstellung (Gleitpunktdarstellung) codiert. Diese Darstellung ist halblogarithmisch. Ist m die Mantisse und p der Exponent, so lässt sich jede reelle Zahl Z folgendermaßen in einem Wort darstellen:

$$Z = m \cdot 2^p$$

Nachkommastellen der gebrochenen Mantisse lassen sich durch negative 2er-Potenzen ausdrücken.

■ **Beispiel** **Wandlung der Zahl Z = 11.625 in das Binärsystem**

		2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	
	...										
	=		**1**	**0**	**1**	**1**	.	**1**	**0**	**1**	$\cdot 2^0$
11.625_{10}	=		**1**	. **0**	**1**	**1**		**1**	**0**	**1**	$\cdot 2^3$

■

Der zur Speicherung einer Gleitpunktzahl verfügbare Platz muss aufgeteilt werden zur Codierung der Mantisse, des Exponenten und des Vorzeichens.

Vz	Exponent	Mantisse

← verfügbarer Speicher, z. B. 4 Byte →

Die interne Darstellung von Gleitpunktzahlen ist herstellerabhängig, d. h. unterschiedliche Rechner stellen die gleiche Zahl unterschiedlich dar. Da dies beim Datenaustausch natürlich zu Problemen führt, wurde vor einigen Jahren die Gleitpunktdarstellung durch Normungsgremien des IEEE (*Institut of Electrical and Electronics Engineers*) standardisiert. Das IEEE-Format setzt sich heute mehr und mehr durch und wird z. B. auch in den numerischen Coprozessoren der PC benutzt.

Wir haben ein einfaches Beispiel gewählt, bei dem sich der Nachkommateil exakt mit nur drei Bit darstellen lässt:

$$1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 0.5 + 0 + 0.125 = 0.625$$

Der Standard definiert 3 Gleitpunkt-Formate

- SHORT REAL: (32 Bit) einfache Genauigkeit, C-Datentyp **float**
- LONG REAL: (64 Bit) doppelte Genauigkeit, C-Datentyp **double**
- TEMPORARY REAL: (80 Bit) erweiterte Genauigkeit

von denen hier das SHORT REAL Format vorgestellt werden soll.

IEEE-Format SHORT REAL (C/C++: float)

Bit 31		Bit 0
Vz	Characteristic c	Mantisse m
1 Bit	8 Bit	23 Bit

Vorzeichen Vz: (der Mantisse)
„0“ positiv
„1“ negativ

Der Darstellungsbereich des Exponenten zur Basis 2 beträgt
-127,...,0,...,+128.

Characteristic ***c***: ***c*** enthält die Codierung des Exponenten ***p***.
Vorzeichenbehandlung von ***p***:
Verschiebung des Exponenten um den halben Zahlenbereich für 8-Bit: 255 DIV 2 = 127.
$\boldsymbol{c} = \boldsymbol{p} + 127$

Durch diese Bereichstransformation werden positive und negative Exponenten stets durch eine positive Characteristic ausgedrückt. Die Zahl 127 ist fest vorgegeben (*excess notation* 127).

Mantisse ***m***: Normalisierung durch Anpassung des Exponenten derart, dass die Mantisse im Binärsystem stets folgende Form hat:

$|m|$ = 1.XXX...X (X=0 oder 1)

d. h. $1 \leq |m| < 2$ (dez)

Da die Vorkomma „1“ bei jeder normalisierten Mantisse auftritt, wird sie bei der Codierung als Hidden Bit einfach weggelassen und damit wertvoller Speicherplatz eingespart.

Wandlung einer Gleitpunktzahl in das IEEE-Format

1. Schritt: Wandlung (dez) → (binär)

2. Schritt: Normalisieren der Mantisse,
Anpassung des Exponenten

3. Schritt: Hidden Bit der Mantisse wegstreichen evtl. mit „0“ auf 23 Stellen auffüllen
→ Mantisse fertig

4. Schritt: Characteristic = Exponent + 127
Characteristic binärcodieren

5. Schritt: Vorzeichenbit setzen

6. Schritt: Eintragen in die IEEE-Maske und in Hexadezimalschreibweise abgreifen

■ **Beispiel 1 Wandlung von 15.625 in das IEEE-Format**

1.) 15.625 (dez) = 1 1 1 1 . 1 0 1 $\cdot 2^0$ (dual)

2.) = 1 . 1 1 1 1 0 1 $\cdot 2^3$

3.) Mantisse:
Hidden Bit entfernen
Auffüllen auf 23 Bit Länge:
$\Rightarrow$ 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

4.) Characteristic = Exponent + 127 = 3 + 127 = 130
130 (dez) = 1 0 0 0 0 0 1 0 (binär)

5.) Vorzeichen positiv ⇒ Vz = 0

6.+ 7.) IEEE-Maske:

Vz	Characteristic	Mantisse
0	1 0 0 0 0 0 1 0	1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
| 4	| 1	| 7	| A	| 0	| 0	| 0	| 0 |	

Ergebnis: 15.625_{10} : $417A0000_{16}$ ■

■ **Beispiel 2 Darstellung von - 0.171875 im IEEE-Format**

1.) 0.171875(dez) = 0 . 0 0 1 0 1 1 $\cdot 2^0$ (dual)

2.) = 1 . 0 1 1 $\cdot 2^{-3}$

3.) Mantisse:
Hidden Bit entfernen
Auffüllen auf 23 Bit Länge:
⇒ 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

4.) Characteristic = Exponent + 127 = -3 + 127 = 124
124 (dez) = 0 1 1 1 1 1 0 0 (binär)

5.) Vorzeichen negativ ⇒ Vz = 1

6.) IEEE-Maske:

Vz	Characteristic	Mantisse
1	0 1 1 1 1 1 0 0	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
| B	| E	| 3	| 0	| 0	| 0	| 0	| 0 |	

Ergebnis: -0.171875_{10} : $BE300000_{16}$ ■

Für die 1 ergibt sich das hexadezimale Muster 3F800000. Wegen des Hidden Bit wäre eine Verwechslung der 1 mit der 0 möglich. Daher muss die Zahl ±0.0 separat vereinbart werden:

(dez) (hex)	IEEE-Format
+ 0.0:	00000000
- 0.0:	80000000

Die Rückrechnung, also die Interpretation eines Binärwertes als Real-Wert im IEEE-Format, erfolgt in umgekehrter Reihenfolge.

■ **Beispiel** **Welcher float-Wert ergibt sich bei Interpretation des Binärmusters 40E4000016 nach IEEE?**

IEEE-Maske:

4	0	E	4	0	0	0	0
0	1 0 0 0 0 0 0 1	1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

V z +	Characteristic	Mantisse
	Exponent: 129 -127 = 2	Hidden Bit 1 . 1 1 0 0 1 bin -> dez: 1. + 0.5 + 0.25 + 0.03125 = 1.78125

$\Rightarrow +1.78125 \cdot 2^2 = +7.125$ ■

Zahlenbereich und Genauigkeit

Der darstellbare Zahlenbereich einer Gleitpunktdarstellung ist bestimmt durch die Länge der Characteristic. Die Genauigkeit, mit der **float**-Zahlen gespeichert werden, hängt von der Länge der Mantisse ab.

Für SHORT REAL gilt:

SHORT REAL (**float**)	
Zahlenbereich:	$\pm 3.4 \cdot 10^{38}$
Genauigkeit:	7 - 8 Stellen (dez)

Grundsätzlich folgt aus der endlichen Mantissenlänge aller Gleitpunkt-Formate allgemein:

> Die reellen Zahlen (**float**, **double**) sind prinzipiell nicht genau in Rechnern darstellbar!
> Es können nur einzelne „Punkte“ auf der reellen Zahlengeraden dargestellt werden.

Solche „Punkte“ verwendeten wir in unseren Beispielen.

Das folgende C++-Programm macht das hexadezimale Muster der eingelesenen **float**-Zahlen im IEEE-Format sichtbar:

```
// PROGRAMM real_Hex
// Inhalt eines float-Typs als Hexwert auslesen
#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
using namespace std;
int main(void)
{
  float r, *pr;
  long int hexwert , *ph;
  char goon;
  do
  {
    cout << "Bitte reelle Zahl eingeben >";
    cin >> r;
    pr = &r;
    ph = (long int *) pr;
    hexwert = *ph;
    cout << hex << hexwert << endl;
    cout << "Weiter? [j/n] >";
    cin >> goon;
    cout << endl;
  }
  while((goon == 'J') || (goon == 'j'));
  getchar();
  return 0;
}
```

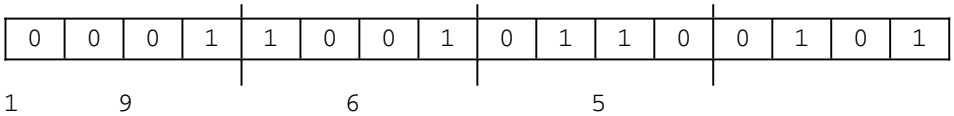
1.4 Sonstige Zifferncodes

Insbesondere für technische Anwendungen gibt es eine Reihe weiterer Codes, z. B. den Gray-Code für die Ablesung von Inkrementalgebern oder den BCD-Code, der u. a. zur Ein- und Ausgabe von Dezimalzahlen in einfachen Mikrocomputersystemen verwendet wird.

BCD-Code: *Binary Coded Decimals*
jede dezimale Ziffer wird durch eine duale Tetrade dargestellt.

■ **Beispiel**

1965



■

Der BCD-Code entspricht im Prinzip dem Hexadezimalcode, allerdings kommen nur die Ziffern 0 bis 9 vor. Da er 1/3 der Darstellungsmöglichkeiten ungenutzt lässt, ist er zu speicherplatzaufwendig für Massendatenverarbeitung. Dennoch verfügen zahlreiche Prozessoren über Befehle zur Verarbeitung von BCD-Werten.

Ein Vorteil des BCD-Code liegt darin, dass Rundungsfehler bei der dezimal/dual-Wandlung nicht auftreten können und Rechenergebnisse genau bleiben.

Beim Rechnen mit BCD-Zahlen können Sonderfälle auftreten, die eine spezielle Behandlung des Ergebnisses erfordern. Der Grund liegt darin, dass der Prozessor grundsätzlich nur dual rechnet und dabei sechs 4-Bit-Muster, sog. Pseudotetraden, die den Hexadezimalziffern A bis F entsprechen, vorkommen können. Diese gehören aber nicht zum BCD-Code. Am Beispiel der Addition von BCD-Zahlen soll dies gezeigt werden.

Es können folgende zwei Sonderfälle auftreten:

1. Eine Pseudotetrade entsteht.
2. Es entsteht ein Übertrag vom niederen auf das höhere Halbbyte (Beispiele im 8-Bit-System):

Pseudotetraden

Beispiel:	34	0011 0100		
	+28	0010 1000		
		1		
	62	0101 1100	= 5?	nicht interpretierbar!
				(Pseudotetrade)
Korrektur +6	⇒	+ 0110		
		11 1		
		0110 0010	= 62	korrekt!

Beim Auftreten von Pseudotetraden muss eine Korrektur +6 am Ergebnis angebracht werden (Überspringen der sechs Pseudotetraden).

Tritt bei der Addition ein Übertrag von dem niederen Halbbyte zum höheren Halbbyte auf, so wird prozessorintern das Hilfscarry Flag A gesetzt. Das Hilfscarry (*Auxiliary Carry*) ist ein weiteres Flag Bit.

Flagregister:

	C	O	A			

Halbbyteübertrag

Beispiel:	38	0011 1000		
	+29	0010 1001		
		1		(A gesetzt)
	67	0110 0001	= 61	falsch!
Korrektur +6	⇒	+ 0110		
		0110 0111	= 67	korrekt!

Ist nach einer BCD-Addition das A-Flag gesetzt, muss das Ergebnis, ebenso wie bei Auftreten einer Pseudotetrade, durch Addition von +6 korrigiert werden.

Es ist die Aufgabe des *Programmierers*, die Speicherinhalte entsprechend zu interpretieren und eventuell zu korrigieren. Für die Korrekturen gibt es spezielle Prozessorbefehle, die nach *jeder* BCD-Operation anzusetzen sind.

1.5 Darstellung von Zeichen

Die Darstellung von Zeichen muss von der Zahlendarstellung unterschieden werden. Während Zahlen für mathematisch-logische Aufgaben verwendet werden, dienen Zeichen zur Verarbeitung und Übertragung von Texten. Man beachte, dass Zahlen auch als Zeichen, wie Buchstaben, vorkommen können, v. a. wenn sie Bestandteile von Texten sind.

Die nachfolgende Tabelle nennt einige gebräuchliche Zeichencodes:

Code	Anwendung	Bitbreite
Baudot	Internat. Fernschreiber-Code	5
ASCII	Zeichendarstellung im Computer	7
Erweiterter ASCII	dto. IBM-kompatibler PC	8
EBCDIC	dto. „alte“ IBM- u. Siemens-Großrechner	8

Für die Textverarbeitung in Computern wird v. a. der ASCII-Code verwendet. Er ist vollständig im „Erweiterten ASCII-Code“ enthalten, der zusätzlich so genannte Halbgrafikzeichen enthält.

Jedes ASCII-Zeichen beansprucht im Speicher eines Rechners 1 Byte, beim herkömmlichen ASCII-Code wird also 1 Bit verschenkt.

Die beiden vollständigen ASCII-Tabellen finden Sie im Anhang.

ASCII-Codes werden in dezimaler, dualer oder, am Häufigsten, in der hexadezimalen Form angegeben.

Was Sie sich merken sollten:

Im ASCII-Code beginnen die Ziffern bei hexadezimal 30, die Großbuchstaben bei 41 und die Kleinbuchstaben bei 61.

Demnach bedeutet `33`(hex) ‘3’, `42`(hex) ‘B’ und `65`(hex) ‘e’.

Möchte man ein Ziffern-ASCII-Zeichen in die entsprechende Binärzahl verwandeln, so muss man `30`(hex) von dem entsprechenden ASCII-Zeichen subtrahieren.

Es wurden bisher verschiedene rechnerrelevante Codes behandelt, die zum Abspeichern und Verarbeiten von Zahlen und Zeichen dienen. Ein weiterer, ebenfalls eminent wichtiger Code ist der Maschinencode. Er enthält die Befehle, mit denen ein Rechner programmiert wird. Das Kapitel 3 ist ausschließlich diesem Code gewidmet.

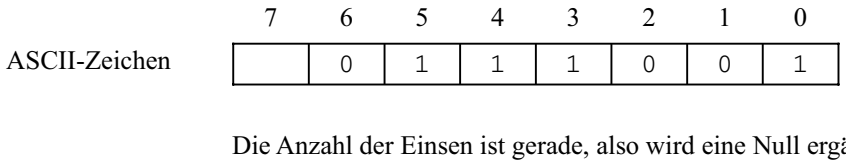
1.6 Das Prüfbitverfahren

Häufig besteht die Notwendigkeit, Daten von einem Rechner zu einem anderen, oft über große Distanzen, zu übertragen. Dabei können Übertragungsfehler auftreten. Es gibt zahlreiche Verfahren, Übertragungsfehler mit einer gewissen Wahrscheinlichkeit zu erkennen. Je höher die Entdeckungswahrscheinlichkeit, umso größer ist der Aufwand. Vor allem in Computernetzen sind derart aufwendige Verfahren unerlässlich (→ s. Teil II).

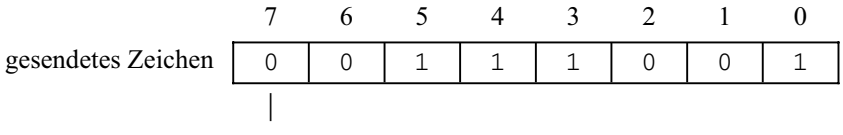
Einfach, wenn auch nur begrenzt zuverlässig, ist dagegen das Prüfbitverfahren. Nehmen wir an, es sollen 7-Bit-ASCII-Daten übertragen werden. Dann ist es nahe liegend, das ungenutzte 8. Bit des Bytes als Prüfbit zu verwenden.

Ein Prüfbit kann so gesetzt werden, dass die Anzahl der Einsen im Byte ganzzahlig ist. Enthält demnach das gesendete 7-Bit-Zeichen eine ungerade Anzahl, wird das Prüfbit „1“, anderenfalls „0“.

■ **Beispiel**



Die Anzahl der Einsen ist gerade, also wird eine Null ergänzt.



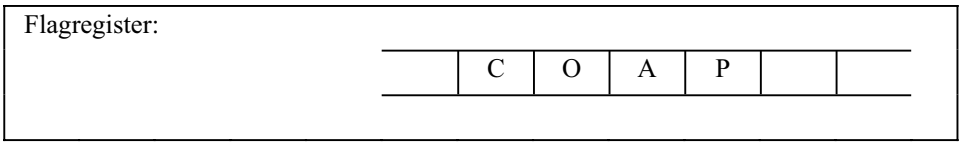
■

Dieses Verfahren prüft auf gerade Parität (*even parity)*, ein anderes prüft auf ungerade Parität (*odd parity)*. Im letzteren Fall wird auf eine ungerade Anzahl von Einsen ergänzt. Man nennt das Prüfbit auch Parity Bit.

Natürlich müssen sich Sender und Empfänger über die Art der Parität einig sein. Der Sender ergänzt das Prüfbit, der Empfänger überprüft die korrekte Parität.

Das Prüfbitverfahren eignet sich sehr gut zum Erkennen der relativ häufigen 1-Bit-Fehler. Sind beispielsweise zwei Bit fehlerhaft, kann dies trotzdem zu einer korrekten Parität führen: der Fehler bleibt unentdeckt.

Die Parität wird als Paritätsbit P im Flagregister der meisten Prozessoren geführt. Es wird jeweils die Parität des zuletzt behandelten Maschinenworts im Prozessor angegeben.



1.7 Bitoperationen mit C/C++

Die Spache C war ursprünglich als System-Implementierungssprache für UNIX entwickelt worden. Sie sollte deshalb auch Aufgaben übernehmen, die zuvor nur mit Assembler zu erledigen waren. Im Gegensatz zu vergleichbaren Hochsprachen besitzt C somit Operatoren zur Bit-Manipulation und unterstützt Bitfelder.

1.7.1 Bit-Manipulationen

C/C++ besitzt folgende Operatoren zur Bit-Manipulation:

Operator	Art der Verknüpfung
&	UND (AND)
|	ODER (OR)
^	Exklusiver ODER (XOR)
~	Komplement (0 → 1; 1 → 0)
>>	Verschieben nach rechts (right shift)
<<	Verschieben nach links (left shift)

Operationen zur Bit-Manipulation sind nur auf ganzzahlige Standard-Datentypen wie **int** und **char** anwendbar. Dabei gelten die üblichen Wahrheitstafeln, wie man sie von logischen Verknüpfungen her kennt. Nur stehen jetzt die einzelnen Bits einer Variablen für die Wahrheitswerte: 1 steht für **TRUE** und 0 für **FALSE**. So sieht die elementare Wahrheitstafel für OR, XOR und AND aus:

p	q	p | q	p^q	p&q
0	0	0	0	0
0	1	1	1	0
1	0	1	1	0
1	1	1	0	1

Wer sich ein wenig mit Digitaltechnik auskennt, dem sind solche Tabellen vertraut. Bit-Operationen spielen besonders bei Hardware nahen Anwendungen, wie Gerätetreibern, eine Rolle. Durch Maskenbytes und geeignete Verknüpfungen lassen sich Bitinformationen aus einzelnen Bytes, die z. B. einen Gerätestatus beinhalten, herausfiltern. Nehmen wir an, wir lesen von einem Gerät ein Byte, dessen Bits 0 bis 6 Nutzinformationen, das Bit 7 dagegen ein Prüfbit enthält. Dieses möchten wir in jedem Fall auf 0 setzen. Zum Ausblenden von Bits eignet sich eine AND-Verknüpfung:

■ **Beispiel**

```
// C-Function mit AND-Verknuepfung
// Blendet das MSB eines Bytes aus
unsigned char read_byte()
{
    unsigned char info_byte;
    info_byte = read_device();   // liesst ein Byte vom Geraet
    return(info_byte & 0x7F);  // AND-Maskierung mit 0111 1111
}
```

Das MSB wird immer auf 0 gesetzt. Die hexadezimale Maske ist optisch deutlicher als die entsprechende dezimale (127).

■

Das folgende Beispielprogramm setzt die Parität für ein eingelesenes 7-Bit-ASCII-Zeichens auf *gerade* (*even*), d.h. die Anzahl der ’1’ im gesamten Byte muss durch Setzen des Paritätsbits

(MSB) auf ’0’ oder ’1’ immer gerade werden. Dabei finden die Shift-Operatoren (<<, >>) Verwendung:

■ **Beispiele**

unsigned char a, n;
a = a << 1; // Die Inhalt der Variablen a wird um 1 Bit nach links geschoben
// und der so veraenderte Inhalt a wieder zugewiesen

n = 7;
a = a >> n; // Die Inhalt der Variablen a wird um 7 Bit nach rechts geschoben
// und der so veraenderte Inhalt a wieder zugewiesen ■

Nachstehend nun das angekündigte Programm:

■ **Beispiel**

```
// Demo fuer Shift-Operatoren
// Setzt das Paritaetsbit (MSB) eines eingelesenen
// 7-Bit-ASCII-Zeichens auf gerade (even)
#include <iostream.h>
#include <stdlib.h>
using namespace std;
// Bitweise Ausgabe eines Bytes
void out_bit(unsigned char ch);
int main( )
{
  unsigned char ascii, orig, bit, pari;
  int one_count;
  cout << "ASCII-Zeichen >";
  cin >> ascii;
  ascii = ascii & 0x7F; // MSB ausblenden (auf 0 setzen)
  out_bit(ascii); // Bitweise ausgeben
  orig = ascii;  // Original-Byte speichern
  one_count = 0; // Einsen-Zaehler auf 0 setzen
  for(int i = 1; i <= 7; i++) // Schleife ueber 7 Bits
  {
    bit = ascii & 0x1; // nur LSB betrachten
    if(bit) one_count++;
    ascii = ascii >> 1; // right shift um 1 Bit
    bit = ascii;
  }
  if((one_count % 2) == 0) pari = 0; // gerade Anzahl
  else pari = 0x80; // ungerade Anzahl von Einsen
  cout << endl << endl << "Paritaetsbit: "
       << (int)(pari >> 7) << endl;
  ascii = orig + pari; // ASCII-Byte mit Paritaetsbit
  out_bit(ascii); // Bitweise ausgeben
  getchar();
  return 0;
}
```

```
// Bitweise Ausgabe eines Bytes
void out_bit(unsigned char ch)
{
  short int bit;
  cout << endl;
  // Bit fuer Bit von links nach rechts ausgeben
  for(int i = 1; i <= 8; i++) // Schleife ueber 7 Bits
  {
    bit = ch & 0x80; // nur MSB erhalten
    if(bit) cout << '1';
    else cout << '0';
    ch = ch << 1; // left shift um 1 Bit
  }
}
```

■

1.7.2 Bitfelder

Bitfelder basieren auf Strukturen. Sie sind eine effektive und Platz sparende Möglichkeit, auf einzelne Bits oder Bitgruppen zuzugreifen. Man kann damit z. B. Statusworte von Geräten setzen oder auswerten. Beispiel:

Eine elektronische Spezial-Kamera in einem Forschungslabor möge das folgende Statuswort besitzen:

Kameratyp:	String
Kamera bereit:	1 Bit
Kamera belichtet gerade:	1 Bit
Anzahl der gespeicherten Bilder (max. 100):	7 Bit

Zum Abspeichern des Statusworts definieren wir eine Bitfeld-Struktur:

```
struct camera {
  char cam_type[20];  // "normale" Strukturkomponente
  unsigned ready: 1;    // Anzahl der Bits = 1
  unsigned expose: 1;   // Anzahl der Bits = 1
  unsigned stored: 7;   // Anzahl der Bits = 7
} cam_stat;
```

Für die einzelnen Bit-Komponenten sind die Datentypen **int**, **signed** und **unsigned** erlaubt. Darüber hinaus ist es zulässig, Bitfelder mit normalen Strukturkomponenten (im Beispiel oben der Kameratyp) zu kombinieren oder zu schachteln. Einzelbits sollten natürlich immer vom Typ **unsigned** sein, denn ein einzelnes Bit kann kein Vorzeichen besitzen. Der C-Compiler speichert die Bitfelder in einem oder mehreren Bytes ab, die vom LSB in Richtung MSB aufgefüllt werden. In unserem Beispiel werden zwei Bytes benötigt, da unsere drei Bitkomponenten insgesamt 9 Bits beanspruchen. Den Programmierer muss das aber nicht sonderlich interessieren, weil er, wie bei Strukturkomponenten üblich, mit Hilfe des Punktoperators oder, über Pointer, per Pfeiloperator zugreift (→ s. Band 1, Kap. 11.3). Das folgende Programm greift unser Kamerabeispiel auf und demonstriert den Umgang mit Bitfeldern.

■ **Beispiel**

```
// Demo fuer den Umgang mit Bitfeldern
// Setzt die Komponenten einer Struktur
// liest diese und gibt sie aus
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
using namespace std;
struct camera {
  char cam_type[20];
  unsigned ready: 1;
  unsigned expose: 1;
  unsigned stored: 7;
};

struct camera set_stat(struct camera cam_stat);
void out_stat(struct camera cam_stat);
int main( )
{
  struct camera status;
  status = set_stat(status); // Kamera-Status setzen
  out_stat(status); // Kamera-Status ausgeben
  getchar();
  return 0;
}

struct camera set_stat(struct camera cam_stat)
{
  strcpy(cam_stat.cam_type, "Supercam");
  cam_stat.ready = 1;
  cam_stat.expose = 0;
  cam_stat.stored = 34;
  return cam_stat;
}
void out_stat(struct camera cam_stat)
{
  cout << "Type of camera: " << cam_stat.cam_type;
  if(cam_stat.ready) cout  << endl  << "Camera ready";
  else cout << endl << "Camera not ready";
  if(cam_stat.expose) cout << endl
                           << "Camera ist exposing";
  else cout << endl << "Camera is not exposing";
  cout << endl << "Number of frames exposed: "
       << cam_stat.stored;
}
```

■

Randbemerkung: *Digitale Bilder und ihre Auflösung*

Die analoge Fotografie ist auf dem Rückzug. Marktführer wie Nikon und Minolta wollen zukünftig nur noch digitale Kameras anbieten. Nicht nur Amateure sondern auch die Profis steigen auf digitale Fotografie um, vor allem bei der Tagespresse, wo es mehr auf Schnelligkeit als auf maximale Qualität ankommt.

Ein fotografischer Film im Kleinbildformat 36 x 24 mm^2 besitzt eine typische Auflösung von 100 Linien pro Millimeter entsprechend 3600 x 2400 Pixel (8,64 Megapixel). Die Anzahl der Bildpunkte (Pixel, dots) definiert die *absolute Auflösung* und damit die mögliche Wiedergabequalität sowie die Dateigröße (ohne Dateikopf) eines Bildes.

Die Anzahl der Bildpunkte pro inch (25,4 mm) wird **dpi** (*dots per inch*) abgekürzt und definiert die *Pixeldichte* oder *relative Auflösung*. Natürlich lässt sich aus dieser Angabe unmittelbar die Pixelgröße des entsprechenden Mediums (Bildschirm, CCD-Chip, ...) errechnen. Auf dem CCD-Chip einer Digitalkamera sind die Pixel meist quadratisch, wenn der Chip zu wissenschaftlich-technischen Vermessungszwecken dient oder sie entsprechen den Größenverhältnissen des Chip-Rechtecks.

Ob die theoretische Grenze der Auflösung bei der fotografischen Aufnahme tatsächlich erreicht wir, hängt von der Qualität des Objektivs ab. Ein sehr gutes Objektiv „unterstützt“ etwa 12, ein Spitzenobjektiv bis zu 20 Megapixel. Die EOS-1Ds Mark II von Canon bietet 16,7 Megapixel und erlaubt bis zu vier Bilder pro Sekunde.

Bei 96 dpi erscheinen Dokumente auf dem Computer-Bildschirm genauso groß wie das Originaldokument in MS Office bei 100 % Zoom betrachtet. Die Auflösung 96 dpi eignet sich nur zum Testen und Archivieren. Zum Drucken benötigt man minimal 300 dpi. Für eine Kunstdruckwiedergabe sind 4800 dpi erforderlich, was selbst die Möglichkeiten einer analogen oder 8 Megapixel-Kleinbildkamera um das Doppelte übersteigt.

Je nach Anforderung benötigt man bestimmte Mindest-Auflösungen:

0,3 Megapixel: WebCam, Überwachung,

2–3 Megapixel: Schnappschüsse, Unfalldokumentation,

3–5 Megapixel: Papierabzüge bis DIN A4-Größe,

5–8 Megapixel: gute Amateurfotos,

> 8 Megapixel: professionelle Fotos,

analoge Top-Platten- oder Rollfilmkamera: Kunstdruck.

Bilddateien werden meist nicht als reine Pixeldateien (Grauwert oder Farbe: 3 Werte pro Pixel) abgespeichert, sondern zusätzlich mit Dateiheadern versehen, die technische Angaben über das Bild enthalten. Gängige PC-Formate sind BMP, JPG, GIF und TIFF. Zum Ausdruck eignen sich hoch auflösende Drucker mit farbechten Pigmenttinten und Baryth-Papier.

1.8 Aufgaben

1) Rechnen Sie folgende Dezimalzahlen in Dualzahlen um:
   13, 125, 4444, 58932

2) Rechnen Sie folgende Dualzahlen:
   `1011`, `11110001`, `1010101111` um in
   a) Dezimalzahlen
   b) Hexadezimalzahlen

3) Addieren Sie die drei Dualzahlen aus Aufgabe 2.

4) Rechnen Sie die Dezimalzahl 12345 in eine Hexadezimalzahl um.

5) Rechnen Sie die Hexadezimalzahl `3FDA` um in eine
   a) Dezimalzahl
   b) Dualzahl

6) Gegeben sei eine Wortlänge von 16 Bit. Rechnen Sie die Dezimalzahl -222 in eine 2er-Komplementzahl (hexadezimal) um.

7) Gegeben sei eine Wortlänge von 21 Bit. Rechnen Sie die Dezimalzahl -1234 in eine 2er-Komplementzahl (hexadezimal) um.

8) Kodieren Sie den Namen „Konrad Zuse“ im ASCII-Code. Wie viele Bytes Speicherplatz benötigt man zur Abspeicherung dieses Namens?

9) Eine einfache CCD-Kamera (WebCam) verfüge über 640 x 480 Pixel (Bildelemente). Jedes Pixel wird mit einer 10-Bit-Auflösung digitalisiert. Wie viel Speicherplatz benötigt ein Frame (komplettes Bild), wenn man
   a) die Bits optimal auf die Speicheradressen verteilt,
   b) nur ganze Bytes zur Abspeicherung verwendet?

   Hinweis: ein Speicherplatz hat eine Kapazität von 8 Bit (1 Byte).

10) Geben Sie zu folgenden `Integer`-Werten die internen Binärmuster in Hexadezimalschreibweise (Wortlänge 8 Bit) an:
    1 -1 24 -24 127 -128 100 -100

11) In einem 8-Bit-System (Wortlänge 8 Bit) werden folgende vorzeichenbehafteten Hexadezimalwerte addiert:

```
  60        A0        50        30
+ 30      + D0      + E0      + 40
```

    Wie werden die Flags C (*Carry*) und O (*Overflow*) beeinflusst?

12) Schreiben Sie ein kleines C-Programm, mit dem Sie interaktiv eingegebene long int-Werte hexadezimal ausgeben. Ändern Sie dazu lediglich das Programm real_hex geringfügig ab. Prüfen Sie die Ergebnisse von 4) und 10) nach.

13) Bestimmen Sie die `Integer`-Werte zu folgenden Binärmustern:
`000000FF` `800000FF` `80000000` `7FFFFFFF` `8FFFFECB`

14) Schreiben Sie ein C-Programm, mit dem Sie Hexadezimalwerte eingeben und diese als long int interpretieren und ausgeben. Prüfen Sie damit Ihre Ergebnisse von 13) nach.

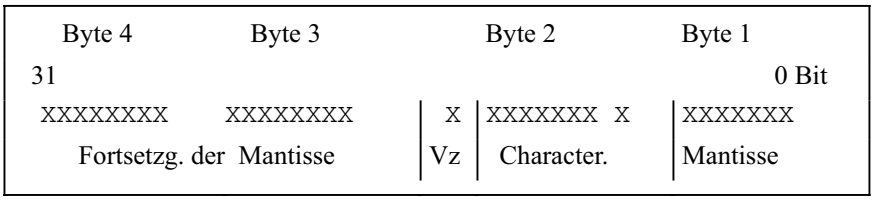
15) Es werde mit einer Wortbreite von 8 Bit gearbeitet. Welches ist die kleinste Hexadezimalzahl, die man zum Hexadezimalwert `4B` hinzuaddieren muss, um einen Overflow zu erzeugen?

16) Wie lauten die IEEE-Darstellungen der folgenden **float**-Werte:
1.0 2.0 3.0 -0.5 -4.0 3.25 -6.625 36.0 -3456.0 -0.2578125

17) Welche float-Werte werden durch folgende Hexadezimalkombinationen dargestellt (IEEE-Darstellung):
`C0400000` `3E000000` `D0100000` `C1002000`

18) Schreiben Sie ein C-Programm, mit dem Sie Hexadezimalwerte einlesen und diese als **float** -Zahlen interpretieren und ausgeben. Prüfen Sie die Ergebnisse von 17) nach.

19) Auf VAX-Rechnern wird folgende IEEE-ähnliche Gleitpunktdarstellung eingesetzt:



Vz: 1 Bit
Characteristic: 8 Bit; „excess 129“ d.h. c = exp. + 129
Mantisse: 23 Bit + 1 hidden Bit

Eine VAX schreibe den `float`-Wert 4120.25 binär in eine Datei. Die Datei wird auf einen PC übertragen und dort von einem C-Programm wieder als binärer `float`-Wert eingelesen. Welchen Wert verarbeitet der PC?

Stellen Sie sich vor, es handele sich um Kontostände!

20) Addieren Sie die folgenden BCD-Zahlen:

a) `0011 0110` + `0011 0001` b) `0101 0111` + `0001 0100`

c) `0001 1000` + `0001 1000`

21) Ergänzen Sie auf gerade Parität (*even parity*):

	7	6	5	4	3	2	1	0
ASCII-Zeichen		0	1	1	0	1	1	1

22) Schreiben Sie ein C/C++-Programm, das ein eingelesenes ASCII-Zeichen (Datentyp **unsigned char**) binär ausgibt und „verschlüsselt“, indem es alle Bits komplementiert (umdreht, unitärer Operator ~) sowie die Bitreihenfolge umkehrt: Aus Bit 0 wird Bit 7, aus Bit 1 wird Bit 6, usw. Das so verschlüsselte Byte ist ebenfalls binär auszugeben. Anschließend soll das Zeichen wieder entschlüsselt und sowohl binär als auch im ASCII-Format ausgegeben werden. Schreiben Sie dazu folgende Funktionen:

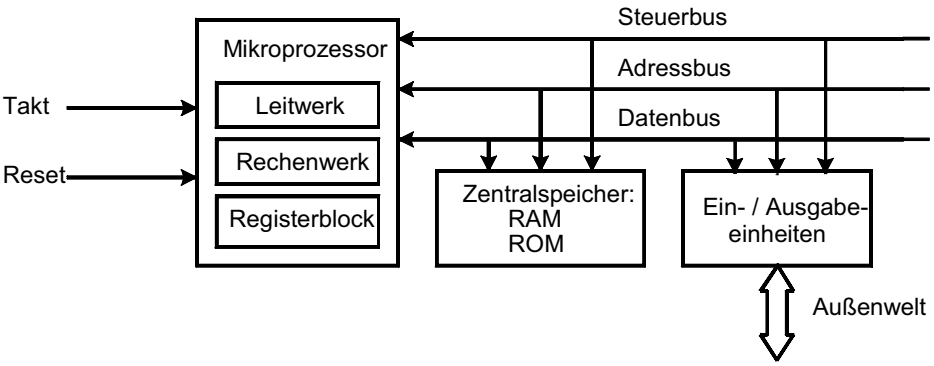
```
void out_bit(unsigned char ch);                 // Bitweise Ausgabe eines Bytes
unsigned char bit_comp(unsigned char ch);       // Bits eines Bytes komplementieren
unsigned char bit_swap(unsigned char ch);       // Bits eines Bytes vertauschen
```

2 Architektur der 80(X)86-Prozessorfamilie

Es soll zunächst der generelle Aufbau von Mikrocomputersystemen betrachtet werden. Anschließend wenden wir uns den INTEL 80(X)86 Prozessoren zu, die den Kern eines PC bilden. Unser Text orientiert sich in seinem weiteren Verlauf an diesem Prozessor- bzw. Mikrocomputertyp. Da jedoch alle vergleichbaren Systeme ähnlichen Prinzipien folgen, lassen sich die wesentlichen Aussagen leicht auf andere Typen übertragen.

2.1 Aufbau eines Mikrocomputers

Vom Prinzip her besitzen Mikrocomputer (µC) den folgenden funktionalen Grobaufbau:



Man unterscheidet drei Hauptbaugruppen:

- Prozessor oder *Central Processing Unit* (CPU)
- Zentralspeicher oder *Memory*
- Ein/Ausgabe-Einheiten oder *Input/Output* (I/O) *Units*

Verbunden sind sie über ein System von uni- und bidirektionalen Leitungen (Datenpfaden), Bus genannt.

Der Takt synchronisiert die internen Abläufe, ein Resetimpuls versetzt den Prozessor in den Grundzustand.

Im Folgenden werden diese drei Komponenten mit ihren typischen Eigenschaften kurz vorgestellt. Danach wenden wir uns dem INTEL 8086 Prozessor zu, um zu detaillierteren und praktisch verwendbaren Ergebnissen zu kommen.

2.1.1 Mikroprozessor

Ein klassischer Prozessor (CPU) besteht aus:

- Leitwerk (Befehlswerk, Steuerwerk)
- Rechenwerk oder ALU (*Arithmetic Logical Unit*)
- Registern

Die CPU steuert den gesamten Computer und löst arithmetische und logische Aufgaben. Der CPU-Chip besitzt einen Takteingang. Der Takt, von einem Quarzgenerator erzeugt, sorgt für einen synchronen Ablauf der Arbeitsvorgänge innerhalb des Prozessors. Der Resetimpuls setzt die CPU hard- und softwaremäßig auf einen definierten Ausgangszustand.

Ist die CPU auf einem Chip integriert, spricht man von einem Mikroprozessor.

Der Bau von Mikroprozessoren wurde erst durch die Metal-Oxid-Semiconductor (MOS)-Technologie ermöglicht. Die Entwicklung der Integrierten Schaltungen (ICs) führte innerhalb eines Jahrzehnts zum Mikroprozessor (µP):

Mikroprozessoren – Historischer Hintergrund	
– 60er Jahre	ICs auf TTL-Basis (bipolar)
– 1969	MOS-Technologie • leichter herstellbar • kompakter • weniger Stromverbrauch aber • langsamer
– 1971	erster Mikroprozessor

Die MOS-Technologie ermöglicht höchstintegrierte Bausteine, zu deren komplexesten Vertretern die Mikroprozessoren (µP) gehören.

Mikroprozessoren gibt es, wie die folgende Übersicht zeigt, seit Beginn der 70er Jahre:

Die ersten (wichtigen) Mikroprozessoren			
1971	INTEL	4004	4 Bit Wortlänge
1971	INTEL	8008	8 Bit Wortlänge
1974	INTEL	8080	8 Bit Wortlänge
1974	Motorola	M6800	8 Bit Wortlänge
1976	Zilog	Z80	8 Bit Wortlänge
1976	TI	9900	16 Bit Wortlänge

Der Prozessor ist in erster Linie für die Abarbeitung der Maschinenbefehle zuständig, die in (binär)codierter Form im Zentralspeicher vorliegen und in ihrer Gesamtheit ein ablauffähiges Programm bilden.

Man kann sich den Prozessor als Automaten vorstellen, der eine genau festgelegte Anzahl von Tätigkeiten ausführen kann. Diese Anzahl entspricht der Menge der Maschinenbefehle (Befehlssatz). Größenordnungsmäßig verfügt ein Mikroprozessor über 100 Befehle.

Ein typischer Maschinenbefehl soll die internen Abläufe illustrieren:

Addiere Register 2 zu Register 1 (Ergebnis nach Register 1).

Register sind eine Art Privatspeicher des Prozessors (→ s. Kap. 2.2.1).

Der Programmschrittzähler (*Instruction Pointer*, IP-Register) enthält die Speicheradresse, in der der obige Maschinenbefehl in binär codierter Form steht.
Über den Datenbus gelangt der Befehl in das Befehlsregister des Leitwerks.
Das Leitwerk interpretiert den Befehl und veranlasst seine Ausführung.
Die Inhalte von Register 1 und Register 2, also die Operanden, werden in das Rechenwerk geschoben.
Das Rechenwerk nimmt die Addition vor.
Das Ergebnis gelangt in das Register 1, der neue Status in das Status(Flag-)register.
Der Programmschrittzähler wird mit der Adresse des folgenden Befehls geladen („zeigt“ auf den folgenden Befehl).

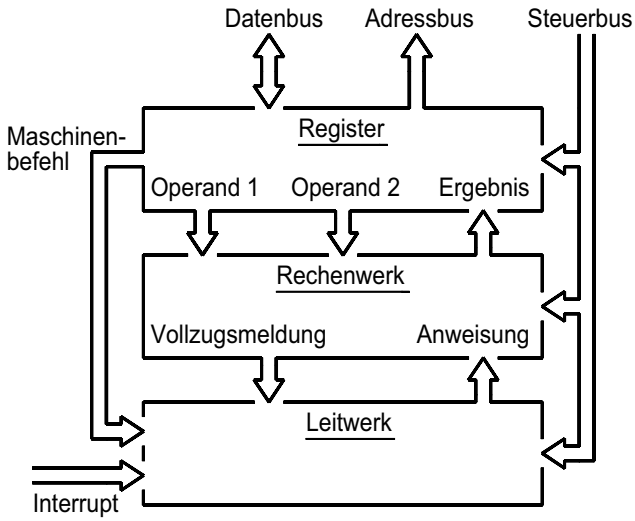
Die nicht nach außen geführten Informationswege des Prozessors bezeichnet man als internen Bus, auf den der Benutzer, im Gegensatz zum externen Bus, keinen Zugriff hat.

In aller Regel verfügen Mikroprozessoren nur über wenige Datenregister (typischerweise 2, 4, 8 oder 16). Deshalb können die Daten nicht über längere Zeit im Verlauf eines Programms in den Registern vorgehalten werden. Vielmehr findet ein ständiger Datenaustausch zwischen dem Hauptspeicher und den Datenregistern statt, der durch Transportbefehle realisiert wird. In einem realen Programm wäre unser Additionsbefehl wahrscheinlich in die nachstehende Befehlsfolge eingebettet:

1. Transportiere 1. Operanden vom Speicher nach Register 1
2. Transportiere 2. Operanden vom Speicher nach Register 2
3. Addiere Register 2 zu Register 1 (Ergebnis nach Register 1)
4. Transportiere Ergebnis in den Speicher

Bei den Transportbefehlen wird die gewünschte Speicheradresse auf den Adressbus gelegt. Je nach Transportrichtung geht die Schreib/Leseleitung des Steuerbusses auf einen der beiden möglichen Zustände (Schreiben und Lesen aus Sicht der CPU). Anschließend werden die Daten auf den Datenbus gelegt.

Der interne Aufbau lässt sich modellhaft so darstellen:



2.1.2 Zentralspeicher

Der Zentralspeicher unterteilt sich in einen flüchtigen (*volatile memory*) und einen nicht-flüchtigen (*non-volatile memory*) Teil. Nicht-flüchtig bedeutet, dass die Informationen auch nach dem Abschalten der Stromversorgung erhalten bleiben. Grundsätzlich unterscheidet man folgende beiden Speicherbaustein-Typen:

- RAM (*Random Access Memory*) = flüchtig (sofern nicht batteriegepuffert), Lesen und Schreiben möglich
- ROM (*Read Only Memory*) = nicht flüchtig, nur Lesen möglich

RAM-Typen

Bezeichnung	Bedeutung	Speichermedium	Eigenschaften
SRAM	Statische RAM	Flip-Flop	– sehr schnell, bis 10ns – nur eine Versorgungs-spannung (5 V) – teuer – geringe Speicherdichte
DRAM	Dynamische RAM	Halbleiter-kondensator	– billig – hohe Speicherdichte – (-> s. Kap. 2.2.4) – relativ langsam – benötigt Refreshimpulse aufgrund von Leckströmen (ca. alle 2 msec)
Non-Volatile RAM		batteriegepufferter RAM, Sondertyp des SRAM	– ca. 10 Jahre Informationserhaltung – geringe Speicherdichte – teuer – EPROM-Ersatz

RAMs weisen folgende typische Speicherorganisationen auf:

1 Bit, 4 Bit, 8 Bit (= 1 Byte), 16 Bit (= 1 Wort).

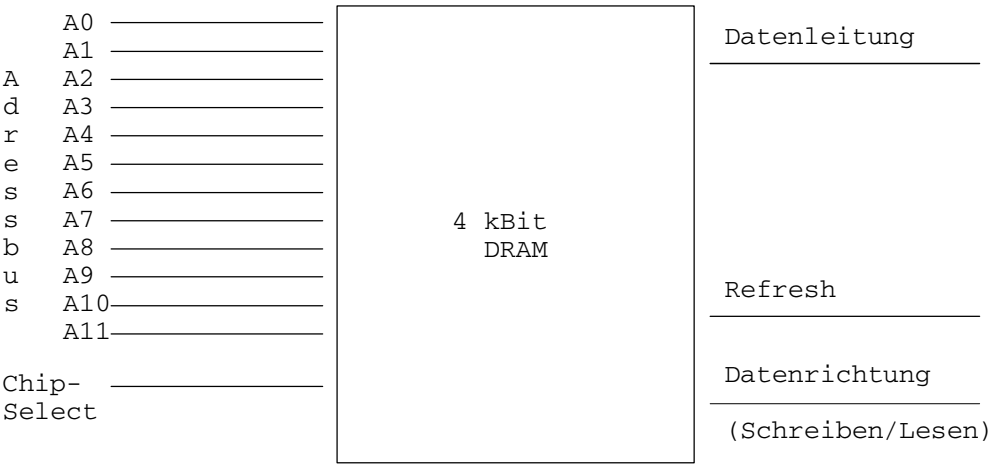
1 Bit bedeutet beispielsweise, dass jedes einzelne Bit über die Adressleitungen ansprechbar ist, bei 8 Bit sind nur Bytes elementar adressierbar.

Zwei Beispiele für verschiedene Speicherorganisationen:

1024 x 1 → 1024 Speicherzellen zu je 1 Bit = 1024 Bit (= 1kBit)

2 k x 8 → 2 k Speicherzellen zu je 8 Bit = 16 kBit (= 2KByte)

Nachstehend das Modell eines 4 kbit DRAM:



Per Chip-Select-Eingang wird der RAM-Baustein aktiviert. Anschließend lässt sich jedes der 4096 Bit einzeln über die 12 Adressleitungen ansprechen:

$$2^{12} = 4096.$$

In Mikrocomputersystemen wie PC werden heute, aus nahe liegenden Gründen (billig, hohe Speicherdichte), DRAMs verwendet, bei denen minimal einzelne Bytes adressiert werden können. Der Speicherbedarf beim PC hängt neben den geplanten Anwendungen wesentlich vom Betriebssystem ab. So genügen für Windows 98 oder ME 128 MByte, XP und Vista erwarten 256 MByte, besser mehr. In Kap. 2.2.4 werden wir näher auf die Kenngrößen moderner PC-Speicher eingehen.

ROM-Typen

Bezeichnung	Bedeutung	Eigenschaften
PROM	Programmable ROM	– spezielle Programmiergeräte – Stromstoß trennt Verbindungen von Zeilen- und Spaltenleitungen („Einbrennen“) – nur einmal programmierbar
EPROM	Erasable PROM	– Speicherzelle: Silizium-Gate, das Ladung „festhält“ – Ladungsverlust = Infomationsverlust, technisch: Bestrahlung mit UV-Lampe (= Löschen) – mehrfaches Programmieren mit EPROM-„Brenner“ (ab einigen 100 DM) – bis 64 KByte
EEPROM bzw. EAROM	Electrically erasable PROM bzw. Electrically alterable PROM	– können vom Rechner per Software beschrieben werden, auch byteweise – Informationen bleiben nach Abschalten erhalten – schreiben sehr langsam – geringe Speicherdichte – teuer – anschlusskompatibel zu EPROMS 2716 und 2732 (s. Tabelle unten)

Einige klassische EPROM-Typen:

EPROM-Typ	Speicherorganisation
2716	2 k x 8 Bit
2732A	4 k x 8 Bit
2764A	8 k x 8 Bit
27128	16 k x 8 Bit
27256	32 k x 8 Bit
27512	64 k x 8 Bit

Für die meisten EPROM-Typen liegt die Zugriffszeit bei etwa 250 Nanosekunden.

2.1.3 Ein/Ausgabe-Bausteine (I/O-Ports)

Ein/Ausgabe-Bausteine sind die Verbindungen (Interfaces) des µC zur Außenwelt (Peripherie). Sie entlasten die CPU von Routineaufgaben der Ein/Ausgabe von Daten an Laufwerke, serielle und parallele Schnittstellen, usw.

Angesprochen werden sie, je nach µC-Type über Adressen (wie Speicherplätze, z. B. M68000-Familie, = memory mapped I/O) oder spezielle Portadressen (z. B. INTEL 80(X)86-Familie).

Wir werden später auf die Programmierung von Ports zurückkommen.

2.1.4 Busleitungen

Alle Baugruppen des µCs sind untereinander, vor allem aber mit der CPU, durch Busleitungen verbunden (Omnibus = lat. „von allen“, „für alle“).

Jeder Systembus besteht aus n parallelen Leitungen, man unterscheidet:

- Steuerbus
- Adressbus
- Datenbus

Steuerbus

Die einzelnen Leitungen des Steuerbusses sind Träger spezifischer Informationen, z. B. ob gerade ein Lesen oder Schreiben von Daten erfolgen soll. Der Steuerbus ist stärker als Adress- und Datenbus auf den jeweiligen Prozessor zugeschnitten (→ s. Kap. 2.2).

Adressbus

Die einzelnen Leitungen werden meist mit $A_0 \ldots A_{n-1}$ bezeichnet.
Sie werden in erster Linie zum Ansprechen von Speicherzellen benötigt. Die Anzahl der Leitungen bestimmt den Adressraum des Rechners:

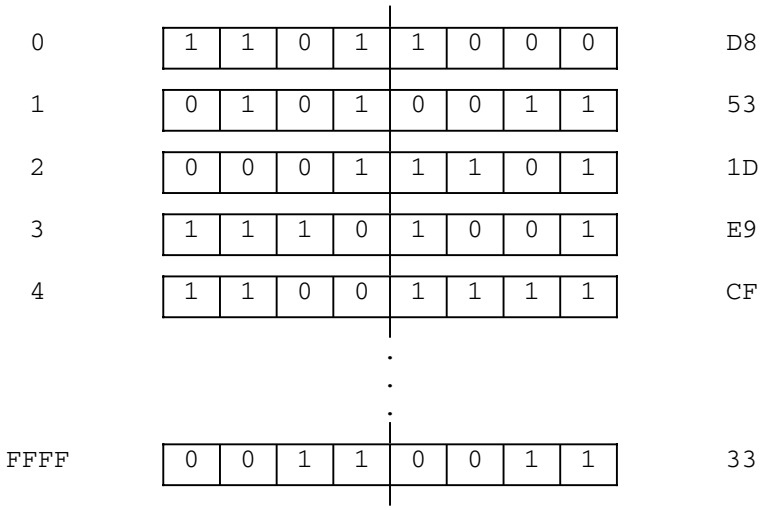
Adressraum = 2^n Byte, n = Anzahl der Adressbusleitungen

Beispiele: n = 8 -> Adressraum = 256 Byte
n = 16 -> Adressraum = 64 KByte
n = 20 -> Adressraum = 1 MByte
n = 24 -> Adressraum = 16 MByte
n = 32 -> Adressraum = 4 GByte

Der Speicher ist stets linear aufgebaut und durchnumeriert (in der Regel hexadezimal, selten dezimal angegeben). Einzelne Speicherplätze werden über diese „Hausnummer“ adressiert. Jeder elementare Speicherplatz hat stets die Kapazität von einem Byte.

Achtung	aus Sicht der CPU (und damit auch des Programmierers) ist der Speicher eines Rechners byteorientiert.
Folge:	man kann minimal auf einzelne Bytes im Speicher zugreifen (nicht jedoch auf Bits).

■ **Beispiel** Schubladenmodell eines Speicher mit einem Adressraum von 64 KByte



Natürlich sind Adressen und Inhalte rechnerintern binär realisiert, die Benennung erfolgt jedoch zumeist hexadezimal. Wir haben deshalb die einzelnen Speicherzellen optisch in zwei Nibbles unterteilt. Die gezeigten Speicherinhalte sind willkürlich. ■

Wird über den Adressbus eine Speicherzelle adressiert, so ergibt sich die Adresse aus den „quergelesenen“ binären Zuständen der Adressleitungen.

Datenbus

Der Datenbus ist ein bidirektionaler Bus zum schnellen Datenaustausch zwischen CPU und Zentralspeicher bzw. I/O-Ports. Die einzelnen Leitungen werden mit D_0 ... D_{n-1} bezeichnet.

Die Datenbusbreite beträgt bei modernen µCs mindestens 8 Bit, damit eine elementare Speicherzelle mit nur einem Zugriff ausgelesen bzw. beschrieben werden kann. Die Datenbusbreite ist mehr noch als die des Adressbusses ein Leistungskriterium für Rechner.

Gängige Datenbusbreiten: 8 Bit
16 Bit
32 Bit
64 Bit

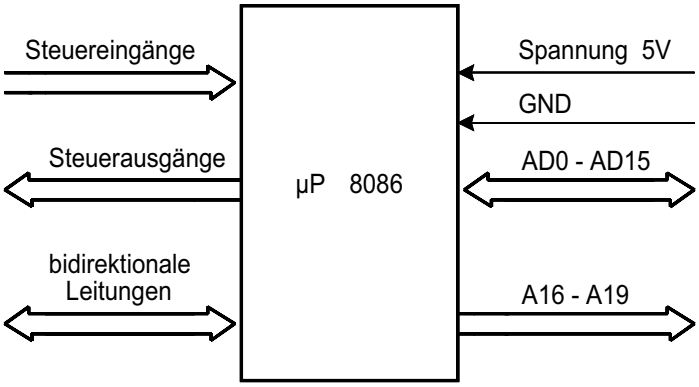
Sonstige Bussyteme

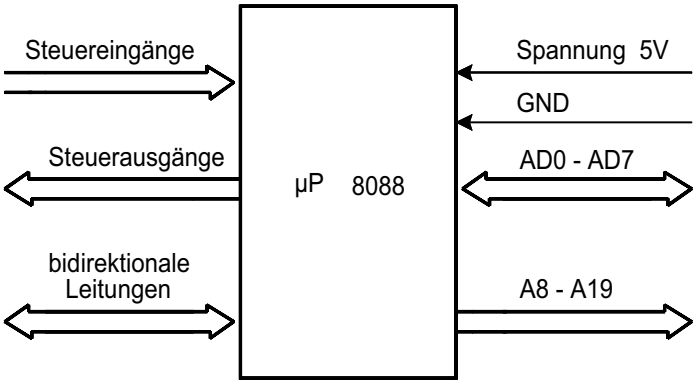
Jede CPU verfügt zusätzlich über einen *internen* Bus, der dem Benutzer jedoch nicht zugänglich ist. Über den internen Bus laufen die Informationen zwischen den Bauteilen der CPU, v. a. Leitwerk, Rechenwerk und Register.

Auch außerhalb des Computers werden oftmals Bussysteme zur Verbindung mit diversen Peripheriegeräten und anderen Computern verwendet. Beispiele: der IEC-Bus in der Messtechnik sowie der Profibus und CAN-Bus bei industriellen Automatisierungssystemen. Unmittelbar, d. h. standardmäßig ohne zusätzliche Einschubkarte, unterstützen moderne PC die Bus-systeme USB und Firewire (→ s. Kap. 2.2.5).

2.2 Hardwaremodell der INTEL 80(X)86-Prozessoren

Der 16-Bit-Prozessor 8086 ist als Nachfolger des 8-Bit-Prozessors 8080 das „Gründungsmitglied“ der 80(X)86-Familie und Kernstück des ersten PC. Daneben gibt es eine Art „Light-Version“, den 8088, der nur 8 Datenleitungen besitzt. Beide Prozessorchips besitzen je 34 Pins („Beinchen“) über die sie mit dem Bussystem verbunden sind. Es handelt sich um Steuer-, Daten- und Adressleitungen:





A*n* steht für Adressleitung Nr. *n*, AD*n* bedeutet Adress- und Datenleitung Nr. *n*. Im letzteren Fall laufen Daten und Adressanforderungen über die gleichen Anschlüsse. Über eine zusätzliche Steuerleitung muss festgelegt werden, ob aktuell Daten oder Adressen auf den Leitungen

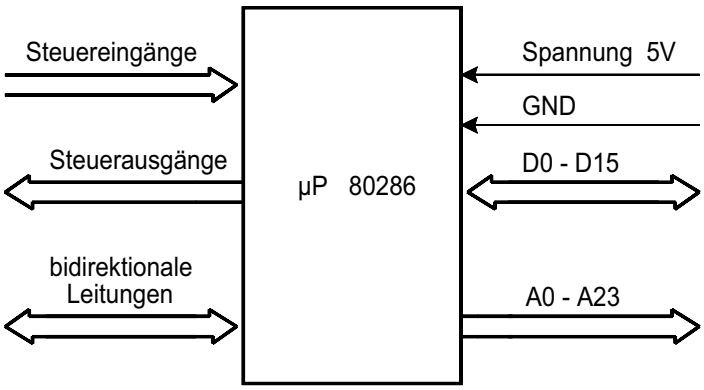
stehen. Dieser Umschaltvorgang (*multiplexing*) geht auf Kosten der Verarbeitungsgeschwindigkeit.

Beim 8088 müssen außerdem 16-Bit-Datenwörter in zwei Portionen übertragen werden, da nur acht Datenleitungen zur Verfügung stehen. Dies kostet zusätzliche Zeit, jedoch keinen zusätzlichen Programmieraufwand.

Einige wichtige Steueranschlüsse werden nachfolgend beispielhaft aufgeführt:

ALE	Adress Latch Enable: Adressleitungen aktiv
RD	READ: Lesen-Signal
CLK	CLOCK: System Takt Eingang
RESET	Programmstart bei Adresse `FFFF:0000` (Segment:Offset)
HOLD	Prozessor anhalten, externe Einheit möchte Bus steuern
INTR	maskierbarer Interrupt-Eingang
NMI	nicht-maskierbarer Interrupt Eingang
READY	keine weiteren Befehle ausführen
NM/MX	Maximum Mode, falls mehrere Prozessoren im System (dann zusätzlicher Bus-Controller 8288 notwendig)
TEST	nach dem Maschinenbefehl WAIT werden keine weiteren Befehle ausgeführt, bis Test-Leitung auf LOW geht

Eine Weiterentwicklung stellten die Prozessoren 80186, 80188 und 80286 dar Während die beiden ersteren für die PC-Entwicklung unwichtig sind, ist letztere der AT-Prozessor.



Daten- und Adressanschlüsse sind beim 80286 separat vorhanden, so dass die zeitaufwendige Umschaltung entfällt. Während der 8086/88 mit seinen 20 Adressleitungen einen Adressraum von 1 MByte erschließt, adressieren die 24 Leitungen des 80286 bis zu 16 MByte. Insgesamt ist er etwa sechs mal schneller als der 8086.

Es folgt nun ein Überblick über die Entwicklungslinie der Intel-PC-Prozessoren mit Angabe der wichtigsten technischen Kenngrößen:

– PC

Prozessor:	8088
Erscheinungsjahr:	1979
Datenbus:	8 Bit
Adressbus:	20 Bit
Realer Adressraum:	1 MByte
Virtueller Adressraum:	-
Taktrate [MHz]:	4.77 – 10
Registerbreite:	8/16 Bit
Cache-Memory:	nein
Externer Bus:	PC
Adressierungsmodus:	Real Mode
Betriebssystem:	DOS

– XT

Prozessor:	8086
Erscheinungsjahr:	1978
Datenbus:	16 Bit
Adressbus:	20 Bit
Realer Adressraum:	1 MByte
Virtueller Adressraum:	–
Taktrate [MHz]:	4.77 – 10
Registerbreite:	8/16 Bit
Cache-Memory:	nein
Coprozessor on chip:	nein
Externer Bus:	PC
Adressierungsmodus:	Real Mode
Betriebssystem:	DOS
Bemerkung:	bei gleicher Taktrate ca. 25 % schneller als der PC und 10 mal schneller als der 8-Bit-Vorgänger 8080 (1974)

– AT

Prozessor:	80286
Erscheinungsjahr:	1982
Datenbus:	16 Bit
Adressbus:	24 Bit
Realer Adressraum:	16 MByte
Virtueller Adressraum:	1 GByte
Taktrate [MHz]:	6 – 20
Registerbreite:	8/16 Bit
Cache-Memory:	nein
Coprozessor on chip:	nein
Externer Bus:	ISA
Adressierungsmodus:	Real Mode, Protected Mode (unterstützt Multitasking)
Betriebssystem:	DOS, OS/2, Windows 3.0, UNIX
Bemerkung:	ca. sechs mal schneller als der PC

– 386er

Prozessor:	80386DX
Erscheinungsjahr:	1985
Datenbus:	32 Bit

Adressbus:	32 Bit
Realer Adressraum:	4 GByte
Virtueller Adressraum:	16 TByte
Taktrate [MHz]:	25 – 40
Registerbreite:	8/16/32 Bit
Cache-Memory:	extern (meist 256 KByte SRAM, 10–20 ns)
Coprozessor on chip:	nein
Externer Bus:	ISA, EISA, Microchannel, Local-Bus, PCI
Adressierungsmodus:	Real Mode, 4-Ebenen-Protected Mode, Virtual Mode (erstmals)
Betriebssystem:	DOS, OS/2, Windows 3.1, UNIX
Bemerkung:	Die SX-Version verfügt extern nur über einen 16 Bit breiten Datenbus, daher nur ISA-Bus sinnvoll

– 486er

Prozessor:	80486DX
Erscheinungsjahr:	1989
Datenbus:	32 Bit
Adressbus:	32 Bit
Realer Adressraum:	4 GByte
Virtueller Adressraum:	16 TByte
Taktrate [MHz]:	25 – 100
Registerbreite:	8/16/32 Bit
Cache-Memory:	intern (8 KByte, 486DX4+ 16 KByte), extern
Coprozessor on chip:	ja (erstmals)
Externer Bus:	ISA, EISA, Microchannel, Local-Bus, PCI
Adressierungsmodus:	Real Mode, 4-Ebenen-Protected Mode, Virtual Mode
Betriebssystem:	(DOS), OS/2, Windows 3.11, Windows 95, UNIX
Bemerkung:	Die SX- und SX2-Version enthält keinen internen Coprozessor

– Pentium

Prozessor:	Pentium (aktuell Pentium IV)
Erscheinungsjahr:	1993
Datenbus:	64 Bit (intern)
Adressbus:	32 Bit
Realer Adressraum:	4 GByte
Virtueller Adressraum:	16 TByte
Taktrate:	60 MHz – 4 GHz
Registerbreite:	8/16/32 Bit (daher trotz 64 Bit Datenbus 32 Bit Prozessor)
Cache-Memory:	intern (8 KByte Code, 8 KByte Daten, Pentium Overdrive+ 2 x 16 KByte), extern
Coprozessor on chip:	ja, ggü. 486 verbesserte Floating Point Unit (FPU)
Externer Bus:	ISA, EISA, Microchannel, Local-Bus, PCI
Adressierungsmodus:	Real Mode, Protected Mode, Virtual Mode
Betriebssystem:	(DOS), OS/2, Windows 98, NT, XP, Vista, UNIX, Linux
Bemerkung:	zwei parallel arbeitende Adress-Pipelines (U-, V-Pipeline), daher typischerweise 3 Befehle pro Takt (486er 2 Takte, 386er 5 Takte pro Befehl). Kompatible Prozessoren der Firma AMD

Es besteht eine komplette Abwärtskompatibilität aller späteren Prozessoren zum 8086.

Die wichtigsten Eigenschaften der PC-Bussysteme:

– Industry Standard Architecture (ISA)

geeigneter Prozessor:	ab 80286
typischer Bustakt:	8.33 MHz
Datenbus:	16 Bit
Adressraum:	16 MByte
Datenübertragungsrate:	5 MByte pro Sekunde
Multimasterfähig:	nein

– VESA Local Bus (VLB)

geeigneter Prozessor:	ab 80386
typischer Bustakt:	25–50 MHz
Datenbus:	32/64 Bit
Adressraum:	4 GByte
Datenübertragungsrate:	40/64 MByte pro Sekunde
Multimasterfähig:	ja

– Microchannel (MCA)

geeigneter Prozessor:	ab 80386
typischer Bustakt:	10–25 MHz
Datenbus:	32 Bit
Adressraum:	4 GByte
Datenübertragungsrate:	40 MByte pro Sekunde
Multimasterfähig:	ja
Bemerkung:	eingeführt für die IBM-PS/2-Systeme nicht PC- und ISA-Bus-kompatibel

– Extended Industry Standard Architecture (EISA)

geeigneter Prozessor:	ab 80386
typischer Bustakt:	8.33 MHz
Datenbus:	32 Bit
Adressraum:	4 GByte
Datenübertragungsrate:	33 MByte pro Sekunde
Multimasterfähig:	ja

– Peripheral Component Interconnect (PCI)

geeigneter Prozessor:	ab 80486
typischer Bustakt:	25–33 MHz
Datenbus:	32/64 Bit
Adressraum:	17 Milliarden TByte
Datenübertragungsrate:	132/264 MByte pro Sekunde
Multimasterfähig:	ja

Die hohen Datenübertragungsraten der nicht-ISA-Bussysteme sind nicht nur eine Folge der größeren Datenbusbreite (32/64 Bit), sondern v. a. durch den Burstmode bedingt. Dabei wird immer nur eine (Anfangs-)Adresse auf den Adressbus gelegt und dann ein kompletter Datenblock gegebener Länge übertragen.

2.2.1 Prozessor-Register

Jeder Prozessor besitzt eine Reihe von internen Speichern, die als Register bezeichnet werden. Neben Datenregistern, die den Programmierer besonders interessieren, existieren Register, die für die internen Abläufe notwendig sind, z. B. der Befehlszeiger (*instruction pointer*) und das

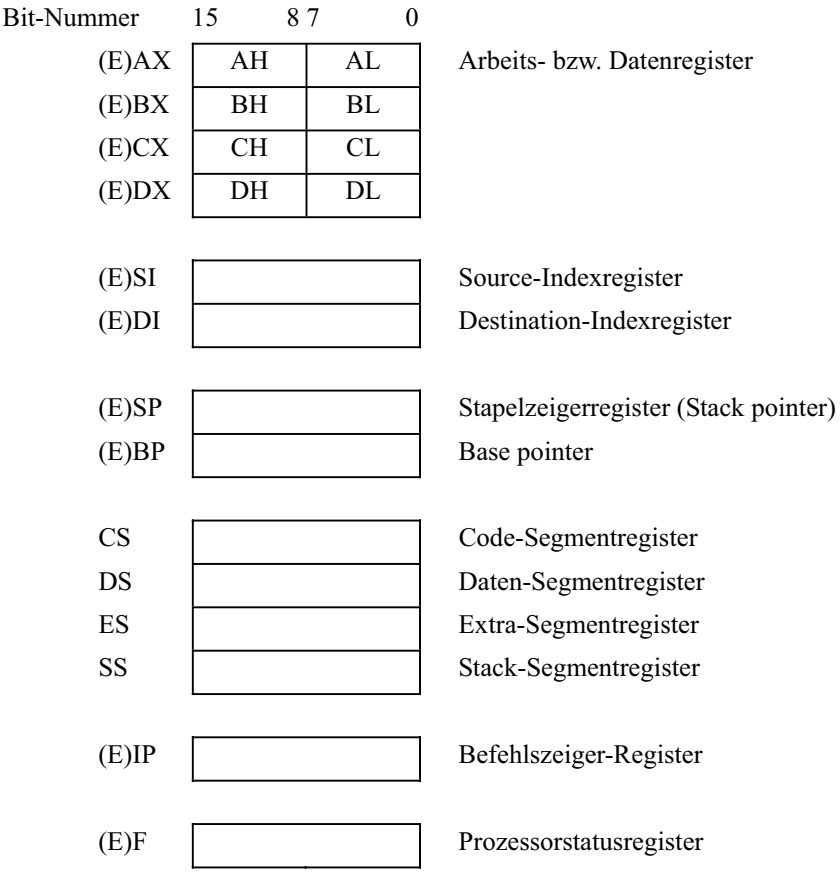
Flagregister. Die exakten Registerbezeichnungen sind typabhängig. Wir betrachten hier beispielhaft die Register der INTEL-Familie. Die Registerbreite ist meist an die Datenbusbreite gekoppelt. Die Registerbreite entspricht der *Wortlänge* des Rechners. Datenbus- und Wortlänge bestimmen die Leistungskategorie des Prozessors.

Beispiele:

8088	8/16 Bit Prozessor
8086	16 Bit Prozessor
80286	16 Bit Prozessor
80386	32 Bit Prozessor
80486	32 Bit Prozessor
Pentium	32 Bit Prozessor mit 64 Bit Datenbusinterface

Beim 8088 beträgt die Datenbusbreite nur 8 Bit, während die Datenregister 16-Bit-Worte aufnehmen können. Jedes Wort muss somit, wie bereits erwähnt, in zwei Portionen zwischen Register und Speicher transportiert werden. Man spricht auch von einem „unechten“ 16-Bit-Prozessor.

Die folgenden 16-Bit-Prozessor-Register sind allen Mitgliedern der 80(X)86-Familie gemeinsam. Die (E)-Erweiterung (E = extended) der 8 Universalregister und der Spezialregister Instruction Pointer und Flagregister auf 32 Bit existiert ab dem 80386:



Programmierermodell der INTEL-80(X)86-Prozessoren

Ab dem 80386 gibt es zwei zusätzliche Segmentregister (FS und GS) sowie einige zusätzliche Spezialregister, die hier nicht erwähnt sind.

2.2.1.1 Arbeits- oder Datenregister

Daten, die ein Prozessor verarbeiten soll, müssen in der Regel zunächst in ein Register geladen werden, damit beispielsweise das Rechenwerk darauf zugreifen kann.

Die INTEL-Familie besitzt nur vier Arbeitsregister zur Aufnahme von Daten. Sie sind grundsätzlich 16 Bit breit, können jedoch auch als 8-Bit-Halbregister verwendet werden, z. B. AX als AH und AL. Das „H“ steht für „High“, das „L“ für „Low“. Vor allem bei Textverarbeitungsproblemen ist das interessant, weil ein Halbregister ein ASCII-Zeichen aufnehmen kann. Ab dem 80386 existiert eine Datenregistererweiterung auf 32 Bit (EAX, EBX, ECX, EDX).

Die Breite eines Datenregisters bestimmt die Wortlänge:	
08-Bit-Register erlauben	08-Bit-Arithmetik, z. B. AH
16-Bit-Register erlauben	16-Bit-Arithmetik, z. B. AX
32-Bit-Register erlauben	32-Bit-Arithmetik, z. B. EAX

Alle vier allgemeine Datenregister haben darüber hinaus noch Sonderfunktionen von denen sich ihre Bezeichnungen ableiten: (E)AX = Akkumulator, (E)BX = Base Register, (E)CX = Count Register, (E)DX = Data Register.

2.2.1.2 Indexregister

Es gibt zwei Index-Register:

- SI bedeutet *Source Index* (Quellindex)
- DI bedeutet *Destination Index* (Zielindex)

Sie spielen im Zusammenhang mit „höheren“ Adressierungsarten (→ s. Kap. 3.5) eine Rolle, außerdem erleichtern sie die Verarbeitung von Zeichenketten.

Ab dem 80386 wurden diese Register als ESI und EDI auf 32 Bit erweitert.

2.2.1.3 Stackpointerregister

Der Stack ist ein reservierter Teil des RAM-Speichers, der v. a. zur Sicherung von Daten und Rücksprungadressen bei Unterprogrammen dient. Bei Hochsprachen wird der Stack vom Compiler angelegt.

Das SP-Register ist der Stackpointer. Er zeigt auf den aktuellen Eintrag im Stack. Das SP-Register wird v. a. durch spezielle Maschinenbefehle, die Unterprogramm-Befehle CALL und RET sowie die Stack-Befehle PUSH und POP, beeinflusst.

Auch der Basepointer BP zeigt auf einen Datenstapel im Hauptspeicher des Rechners. Ab dem 80386 wurden diese Register als ESP und EBP auf 32 Bit erweitert.

2.2.1.4 Befehlszeigerregister

Das IP-Register (IP = Instruction Pointer) zeigt stets auf die Speicheradresse, die den nächsten auszuführenden Befehl enthält. Man beachte, dass die Breite des IP-Registers bis einschließlich dem 80286 ebenfalls nur 16 Bit beträgt. Der Adressbus der INTEL-Prozessoren ist jedoch mindestens 20 Bit breit. In diesem Zusammenhang spielen die weiter unten (→ s. Kap. 2.2.1.6) behandelten Segmentregister eine wichtige Rolle, hier insbesondere das CS-Register.

Was bedeutet „zeigt auf“? Beispiel:

Der IP-Registerinhalt (hexadezimal)

7	F	D	E

ist eine Adresse im Zentralspeicher des Rechners.

Adresse	Inhalt	
.	.	
.	.	
7FDD	0 4	
⇒ 7FDE	1 2	auf diese Adresse zeigt das Register
7FDF	F F	
7FE0	8 0	
7FE1	4 F	

Von einem Zeiger- oder Pointerregister spricht man, wenn der Inhalt dieses Registers eine Speicheradresse ist. Wir sagen: das Register „zeigt auf“ die Speicheradresse.

Ab dem 80386 ist auch der Befehlszeiger als 32-Bit-Register EIP realisiert.

2.2.1.5 Flagregister

Das Flagregister F (Prozessorstatusregister) enthält das Prozessor-Status-Wort. Es ist 16 Bit breit (ab dem 80386 als EF 32 Bit). Die einzelnen Bits weisen auf wichtige interne Prozessorzustände hin. Man nennt diese Bits „Flags“. Ein Flag ist gesetzt, wenn es HIGH (= 1) ist. Beispielsweise enthält Bit Nr. 6 das Zero Flag (Z). Hat es den Wert 1, so war das Ergebnis der letzten arithmetischen Operation 0.

Flagregister F (= untere 16 Bit von EF):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NT	IO	PL	O	D	I	T	S	Z	0	A	0	P	1	C

Die mit ‘0’ oder ‘1’ besetzten Bitpositionen sind „reserviert“ und enthalten die angegebenen Festwerte.

Die aus Sicht des Programmierers wichtigsten Flags sind die Ereignisflags. Sie werden gesetzt, genauer gesagt auf von „0“ auf „1“ gesetzt, wenn aufgrund eines vom Prozessor ausgeführten Maschinenbefehls ein bestimmtes Ereignis eintritt. Einige Flags wurden bereits in Kap. 1 vorgestellt.

Kurzbezeichnung	Bezeichnung	Ereignis
C	*Carry*	Es erfolgt ein Übertrag aus dem MSB. Beispiel: zwei 16-Bit-Werte werden addiert. Als Ergebnis erhält man einen 17-Bit-Wert. Folge: C = 1
P	*Parity*	Im niederwertigen Byte des Ergebnisses ist die Anzahl der auf „1“ stehenden Bits gerade.
A	*Auxiliary Carry*	Übertrag von Bit 3 nach Bit 4. Erleichtert das Rechnen mit BCD-Zahlen (→ s. Kap. 13.4).
Z	*Zero*	Die Operation führt zu dem Ergebnis Null.
S	*Sign*	Die Operation führt zu einem negativen Ergebnis, sofern die Zahl vorzeichenbehaftet interpretiert wird.
O	*Overflow*	Überlauf, d. h. Vorzeichenumkehr, sofern die Zahl vorzeichenbehaftet interpretiert wird.

Die Ereignisse können vom Programm abgefragt und für den weiteren Verlauf berücksichtigt werden.

Anhand einiger Beispiele wird deutlich, welche Operationen Flags beeinflussen. Dabei gehen wir von einer 8-Bit-Wortlänge aus, wir tun also so, als würden unsere Daten in 8-Bit-Registern verarbeitet (→ s. Kap. 1.2.1.1). Wir verwenden in diesem Fall die binäre Codierung, weil die Beeinflussung der Flags in dieser Darstellung am leichtesten nachvollziehbar ist.

■ **Beispiel Carry Flag:**

```
   1101 0001
+  1000 1100
-------------
(1)0101 1101
```

Da das Ergebnisregister ebenfalls nur 8 Bit aufnehmen kann, wandert der Übertrag ins Carry Flag.
Folge: Carry Flag wird (auf 1) gesetzt.

■ **Beispiel Parity Flag:**

```
   0011 1101
+  0111 0101
-------------
   1011 0010
```

Die Anzahl der „1“ im Ergebnis ist gerade.
Folge: Parity Flag wird (auf 1) gesetzt.

■ **Beispiel Auxiliary Flag:**

```
   0011 1001     bcd-Addition    39
+  0101 1000                   +58
-------------
   1001 0001
```

Das Binärergebnis `91` (hex) entspricht nicht dem erwarteten bcd-Ergebnis 97. Der registerinterne Übertrag von Bit 3 nach Bit 4 setzt das Auxiliary Flag (auf 1). Für die bcd-Addition ist dies gleichbedeutend mit dem Auftreten einer Pseudotetrade, die eine Ergebniskorrektur von `0110` (+6dez) verlangt (→ s. Kap. 1.4).

```
   1001 0001
+  0000 0110
-------------
   1001 0111
```

Damit erhalten wir das gewünschte bcd-Ergebnis 97.

■ **Beispiel Zero Flag:**

```
   0111 1111
-  0111 1111
-------------
   0000 0000
```

Das Ergebnis ist 0. Folge: Zero Flag wird (auf 1) gesetzt.

■ **Beispiel Sign Flag:**

```
   0111 1111
+  1000 0000
-------------
   1111 1111
```

Das Ergebnis ist `FF`(hex). Während `7F`(hex) in der 2er-Komplement-Darstellung eine positive Zahl ist, ist `FF`(hex) negativ, weil das MSB auf „1“ steht (→ s. Kap. 1.2). Folge: das Sign Flag wird (auf 1) gesetzt.

■ **Beispiel Overflow Flag:**

```
   1000 0000
+  1000 0000
-------------
(1)0000 0000
```

Das Ergebnis ist `00`(hex). Während `80`(hex) in der 2er-Komplement-Darstellung eine negative Zahl ist, ist `00`(hex) positiv, weil das MSB auf „0“ steht (→ s. Kap. 1.2). Folge: das Vorzeichen hat gewechselt und das Overflow Flag wird (auf 1) gesetzt. Außerdem werden, nebenbei bemerkt, das Carry und das Zero Flag gesetzt.

■

Während die Ereignisflags also bestimmte Zustände nach einem Maschinenbefehl signalisieren, werden die Steuerflags benutzt, um den Prozessor aktiv zu beeinflussen. Im Rahmen dieser Einführung werden lediglich die beiden folgenden für uns von Interesse sein.

Kurzbezeichnung	Bezeichnung	Ereignis
I	*Interrupt Enable*	Maskierbare externe Interrupts werden zugelassen (Prozessoreingang INTR) und führen zu einer Unterbrechung des gerade laufenden Programms.
T	*Trap*	Nach Ausführung eines jeden Maschinenbefehls wird ein bestimmter Interrupt ausgelöst, der es erlaubt, den Prozessorzustand (Register, Flags) abzufragen, um so Programme zu testen (Einzelschritt-Modus).

2.2.1.6 Segmentregister

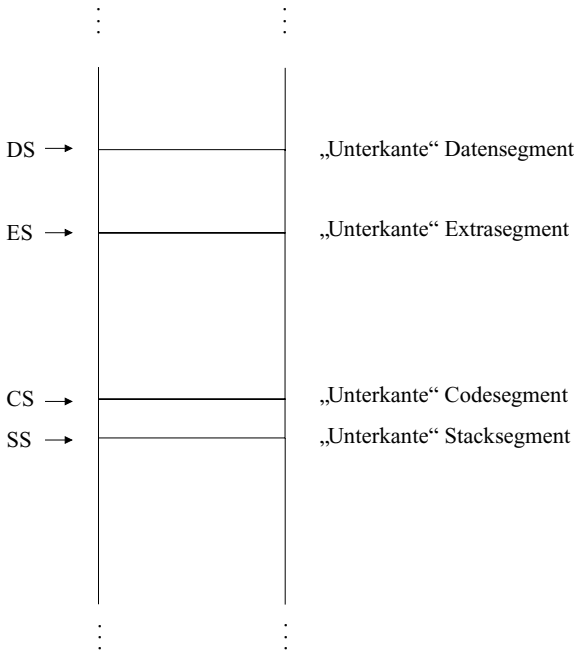
Die Segmentregister sind, je nach Adressierungsmodus (→ s. Kap.2.2.2) in unterschiedlicher Weise, an der Bildung vollständiger Speicheradressen beteiligt.

Gleichzeitig dienen sie zur Separierung verschiedener Bereiche eines Programms (Programmcodebereich, Datenbereich, usw.). Mindestens vier 16-Bit-Segmentregister besitzt jeder Prozessor der INTEL-Familie:

CS	Codesegmentregister	zeigt auf das Speichersegment mit dem gerade aktuellen Programmcode.
DS	Datensegmentregister	zeigt auf ein Speichersegment mit Daten.
ES	Extrasegment	zeigt auf ein Speichersegment mit weiteren Daten (2. Datensegment).
SS	Stacksegment	zeigt auf das Stacksegment, das zur Zwischenspeicherung von Rücksprungadressen bei Unterprogrammen und zur Datensicherung dient.

Ab dem 80386 existieren die beiden zusätzlichen Datensegmentregister FS und GS.

Die Segmentregister zeigen auf die entsprechenden Segmente im Hauptspeicher des Rechners. Reihenfolge und Art der Segmentierung bestimmt das Programm und das Betriebssystem. Die nachfolgende Skizze einer Speicherverwaltung ist demnach nur ein Beispiel:



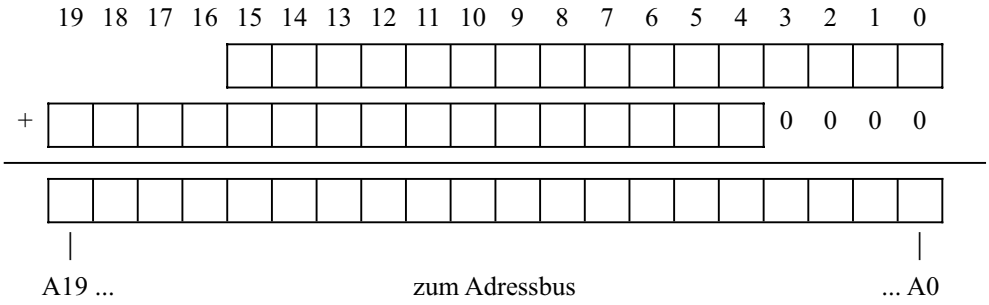
2.2.2 Die Adressierung

Dem 8086/88 stand zur vollständigen Adressbildung nur der Real-Mode zur Verfügung. Ab 80826 sind weitere Prozessor-Modi verfügbar.

Real Mode

Im so genannten Real-Modus arbeiten die Prozessoren mit einer 20-Bit-Adresse. Diesen Modus beherrschen alle Prozessoren der INTEL-Familie. Unter dem Betriebssystem DOS wird auch bei den „höheren“ Prozessoren der Real-Modus eingesetzt. In diesem Modus befindet sich jeder Prozessor (auch ein Pentium!) nach Einschalten des Rechners oder nach einem Reset. Unter modernen Betriebssystemen wie Windows und Linux ist der Real Mode nur noch das Sprungbrett zum Protected Mode (s. u.).

Die Adressbildung geschieht so, dass zu der effektiven 16-Bit-Adresse (Offsetadresse) der Wert eines Segmentregisters, die Segmentadresse, addiert wird. Dies erfolgt automatisch im Prozessor. Um eine 20-Bit-Adresse zu erhalten, wird die Segmentadresse zuvor um 4 Bitpositionen nach links geschoben (entspricht einer Multiplikation mit `10`(hex) bzw. `16`(dez)).



Errechnung der absoluten (physikalischen) Adresse nach der Formel

Offsetadresse + Segmentadresse · 10(hex) = absolute Adresse

Auf diese Weise lässt sich ein Adressraum von 1 MByte erschließen.

■ **Beispiel**

Nehmen wir an, das Codesegmentregister enthalte den Wert 4F7E(hex), der Instruction Pointer „zeige auf“ 2100(hex). Bei der Programmcode-Adressierung bilden diese beiden Register ein Paar, man schreibt

CS:IP

Berechnen wir den Wert der absoluten 20-Bit-Adresse:

```
   2100     Offsetadresse
+  4F7E     Segmentadresse
-----------
=  518E0    absolute Adresse
```

■

Zwar führt die Summe von Offset- und Segmentadresse zu einem eindeutigen Ergebnis, umgekehrt kann man im Allgemeinen von der absoluten Adresse nicht eindeutig auf eine bestimmte Offset- und Segmentadresse schließen, da es eine Vielzahl möglicher Kombinationen gibt.

Der Prozessor besitzt die vier Segmentregister CS, DS, ES und SS, die auf die Segmentkanten im Zentralspeicher zeigen. Damit ist es möglich, vier verschiedene Speicherbereiche gleichzeitig in einem Programm anzusprechen.

- das Codesegment
- das Datensegment
- das Stacksegment
- das Extrasegment (*Heap*)

Die einzelnen Segmente können völlig getrennt sein oder sich mehr oder weniger überlappen. Maximal, d. h. ohne Überlappung, kann ein einzelnes Segment 64 KByte groß sein. Das bedeutet: Solange man sich programmtechnisch innerhalb eines Segments bewegt, muss der Inhalt des entsprechenden Segmentregisters *nicht* geändert werden. In Hochsprachen nehmen Compiler und Linker dem Programmierer die Registerverwaltung ab. Sind die Segmente auch nur minimal getrennt, entspricht das betreffende Programm dem .EXE-Modell. Liegen alle vier exakt übereinander hat man es mit einem .COM-Programm zu tun.

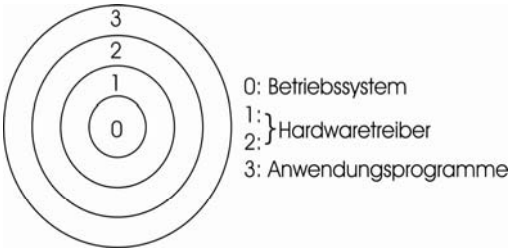
Mit der Segment-Offset-Methode lässt sich der knappe Zentralspeicher sehr effektiv verwalten: Eine Änderung der Segmentadresse um 1 bewirkt eine Änderung der absoluten (physikalischen) Adresse um nur 16 (= 1 Paragraph). Das bedeutet, die Segmentgrenzen lassen sich in Schritten von minimal 16 Byte(-adressen) verschieben.

Protected Mode

Der Protected Mode existiert seit dem 80286 und unterstützt modernere Betriebssysteme wie Windows, UNIX und Linux. Damit werden wichtige Eigenschaften wie Multitasking, Multiuserbetrieb, Multihreading, Speicher- und I/O-Schutz sowie Privilegebenen ermöglicht.

Die Speicherstrukturen für die Adressierung unterscheiden sich völlig von denen des Real Modes. Zwar gibt es weiterhin Segmente, jedoch sind die Mechanismen andere. Segmente tragen Attribute wie READonly, Write + READ und EXEConly. Eine *Descriptortabelle* im Speicher verwaltet die Segmente. Jeder *Descriptor* enthält die notwendigen Informationen wie Startadresse, Größe und diverse Attribute. Im DS-Register steht ein *Selector*, der auf einen Descriptor in der Descriptortabelle zeigt.

Jedes Programm besitzt einen bestimmten Privileglevel. Zugriffe auf eine höhere Privilegebene sind ausgeschlossen. Die Privilegebenen werden häufig als vier konzentrische Kreise dargestellt. Die innere Ebene 0 ist dem Betriebssystem vorbehalten, die Ebenen 1 und 2 den Hardwaretreibern und Ebene 3 den Anwenderprogrammen. Das Überschreiten eines definierten Speicherbereichs erzeugt einen Interrupt, der zu einer *Allgemeinen Schutzverletzung* führt. Das Betriebssystem behebt entweder den Fehler oder beendet das Programm.



Alle „alten“ Maschinenbefehle bleiben gültig, zusätzlich kann man im Protected Mode auch auf die erweiterten 32-Bit-Register zugreifen. Einige weitere Befehle sind hinzugekommen, die teilweise nur den Wechsel vom Real Mode zu Protected Mode ermöglichen.

Virtual Mode

Der Virtual- oder V86-Mode ist eine Sonderform des Protected Mode. Hier wird der vorhandene Speicher so eingeteilt, dass dem darin laufenden Programm „vorgetäuscht“ wird, es befände sich auf einem 8086. Somit kann das Programm maximal 1 MByte Speicher ansprechen. Der Virtual Mode erlaubte es z. B., auch 16-Bit-(DOS)-Anwendungen in eine Multitasking-Umgebung unter Windows einzubinden. Parallel zum Betrieb im Virtual Mode können auch Programme im Protected Mode laufen. Wir nutzen diesen Modus für unsere Einführung in die Maschinen- bzw. Assemblersprache (→ Kap. 3).

System Management Mode (SMM)

Dieser Stromsparmodus wurde mit dem Pentium eingeführt. In ihn wird gewechselt, wenn der Prozessor ein bestimmtes Signal über eine seiner Steuer-Leitungen erhält, es gibt also keinen direkten Befehl für die Umschaltung. Dafür enthält der Pentium-Befehlsvorrat einen Befehl, der den Prozessor aus seinem Schlafmodus erweckt.

2.2.3 Systemplatine

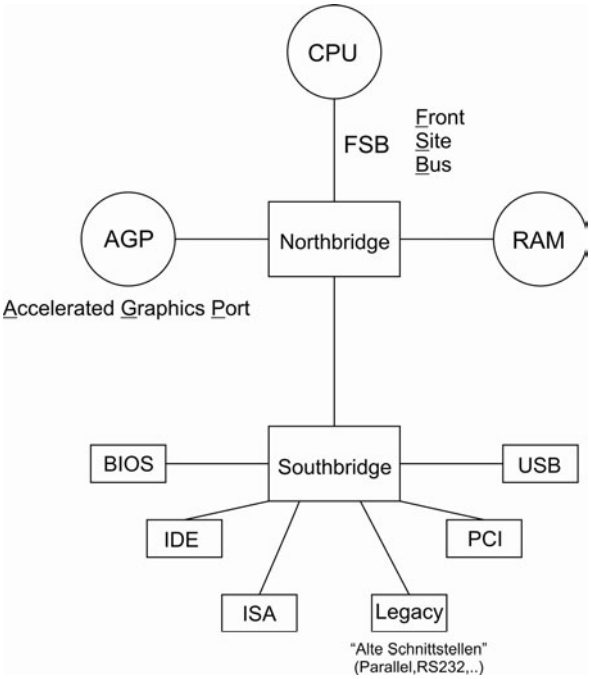
Ein komplettes Mikrocomputersystem ist heute, abgesehen von der Peripherie, auf einer Platine zusammengefasst: der Systemplatine (auch CPU-Platine, Hauptplatine, Mainboard oder Motherboard genannt).

Die Systemplatine enthält Steckplätze für den Prozessor und die Speicherbausteine sowie für Zusatzsatzkarten (Netzwerk, Grafik, Sound, …). An diesen Steckplätzen liegt der Systembus des PC an, so dass Einsteckkarten voll in das System integrierbar sind. Wegen dieses Zugriffs auf den Systembus wird der PC zu einem „offenen System“, d. h. er kann herstellerunabhängig erweitert werden. Bei Steckplatzsystemen ist die CPU-Platine selbst eine austauschbare Steckkarte.

Die Steckkarten schaffen die Verbindungen zur Peripherie, z. B. einer Data-I/O-Karte mit einer seriellen und einer parallelen Schnittstelle.

In umfangreichen Katalogen bieten diverse Firmen eine weite Palette von Einsteckkarten, z. B. für Steuerungs- und Messwerterfassungs-Zwecke, an.

Kernstück der Systemplatine der *Chip-Satz*. Er steuert den Datenaustausch mit den Mainbord-komponenten wie Prozessor und Hauptspeicher sowie den peripheren Geräten. Der PC-Chip-Satz besitzt meist aus zwei Haupt-Komponenten, der *Northbridge* und der *Southbridge*. In der Regel ist die Northbridge zuständig für das Speicherinterface (RAM und Cache), den Front-side-Bus und die Grafikschnittstelle (AGP oder PCIe 16x). Die Northbridge wird auch als Memory Controller Hub bezeichnet. Die Southbridge (Input/Output Controller Hub) verwaltet diverse Schnittstellen, deren wichtigste der Zeichnung zu entnehmen sind. Das Wort „Legacy“ kann mit „Altlasten“ übersetzt werden.



2.2.4 PC-Speicher

Der Hauptspeicher des PC ist als RAM (Random Access Memory) realisiert. Der Prozessor kann direkt und wahlfrei (lesend und schreibend) darauf zugreifen. Dieser Arbeitsspeicher enthält die aktuell laufenden Programme und deren Daten. Die Inhalte sind flüchtig und gehen mit dem Aussalten des Computers verloren. Als Sicherungsmedium dient im Allgemeinen eine Festplatte. Die maximale Größe des Arbeitsspeichers wird durch den Adressraum begrenzt, beim Pentium sind das immerhin 4 GByte. Real übliche Speichergrößen unter XP sind 512 MByte oder 1 GByte. Je größer der reale Speicher, umso weniger muss der Prozessor Daten temporär auf die Festplatte auslagern, was auf Kosten der Rechengeschwindigkeit geht. Für die RAM-Speichermodule sind auf dem Motherboard Steckplätze vorhanden, so dass Speicher nachgerüstet werden kann. Die verwendbaren Speicherchips entnimmt man dem Motherboard-Handbuch. Letzten Endes muss der RAM-Typ mit dem Chipsatz kompatibel sein.

Zurzeit gebräuchliche RAM-Typen sind:

Bezeichnung	Taktfrequenzen [MHz]	Daten-Transportleistung
SDRAM (Synchronous Dynamic RAM)	100, 133, 150, 166	0800 – 3200 MByte/s
DDR-RAM (Double Data Rate RAM)	wie SDRAM, jedoch werden zwei Datenpakete pro Takt übertragen	1600 – 6400 MByte/s
Rambus-RAM	533 (nur Pentium 4, nicht für AMD-Prozessoren)	1200 – 8400 MByte/s

Trotz der hohen Datenraten moderner RAM-Typen wird der Prozessor „ausgebremst“, wenn er unmittelbar auf den RAM zugreift. Aus diesem Grund verfügt der Prozessor seit dem 386er über zusätzliche Pufferspeicher, so genannte *Caches*. Der Level-1-Cache ist integraler Bestandteil des modernen Prozessors selbst. Der Level-2-Cache befindet sich außerhalb. Der Prozessor sucht seine Daten und Maschinenbefehle zunächst im Level-1-, dann im Level-2-Cache. Erst wenn sie weder in dem einen noch in dem anderen zu finden sind, greift er auf den Hauptspeicher zu. Besonders die Größe des Level-2-Cache ist sehr wichtig für die Arbeitsgeschwindigkeit des Rechners. Neuerdings wird zunehmend auch noch ein Level-3-Cache verwendet.

2.2.5 Externe Schnittstellen und Bus-Systeme

Klassische Schnittstellen des PC sind die serielle und parallele Schnittstelle, die immer mehr an Bedeutung verlieren und standardmäßig beim Kauf eines neuen Computers oft gar nicht mehr vorhanden sind. Die PS/2-Schnittstelle dient zum Anschluss von Maus und Tastatur und büßt ebenfalls an Bedeutung ein.

Als moderne Schnittstellen zum Anschluss externer Medien wie Memory-Stick, externe Festplatte oder Digitalkamera konkurrieren zwei Bus-Standards für die PC-Welt: USB und Firewire. Sie erlauben Datenraten bis zu 480 Mbit/s (nicht zu verwechseln mit MByte!).

USB

Der Universal Serial Bus unterstützt bis zu 127 Geräte. Er arbeitet nach dem Hot-Plugging-System, d. h. Geräte können während des laufenden Betriebs ein- und ausgesteckt werden. Beides wird vom Betriebssystem erkannt. Zurzeit gibt es zwei Versionen mit folgenden Kenngrößen:

USB-Version	Daten-transfer-Rate	Maximale Anzahl der Geräte
1.1	1,5 und 12 Mbit/s	127
2.0	1,5, 12 und 480 Mbit/s	127

Geräte beider Versionen sind kreuzkompatibel, allerdings nur mit der USB 1.1-Datenrate. Die hohe USB 2.0-Datenrate wird nur erreicht, wenn ausschließlich USB 2.0-Geräte am Hub angeschlossen sind. Bei Geräten mit hohem Stromverbrauch muss evtl. ein USB-Hub mit eigener Spannungsversorgung verwendet werden.

Firewire

Firewire ist im Prinzip die Konkurrenz von USB. Häufiger Einsatzbereich ist die Verbindung spezieller CCD-Kameras mit dem PC im Einsatzfeld digitale Bildverarbeitung. Der Firewire-Bus ist im Gegensatz zu USB international genormt. Die wesentlichen Kenngrößen sind:

Firewire	Datentransfer-Rate	Maximale Anzahl der Geräte
IEEE 1394	bis 480 Mbit/s	63 (kein Hub nötig, da Geräte beliebig kaskadierbar)

Randbemerkung: Club der toten Medien

Bei der Langzeit-Archivierung von Daten trifft man auf zwei Probleme: Auch digitale Datenträgen sind nicht unbegrenzt haltbar und Speichermedien veralten und können von Computern neuerer Generationen nicht mehr gelesen werden. Viele Magnetbänder, *das* Speichermedium bis Ende der siebziger Jahre, sind heute nicht mehr lesbar. Ähnliches gilt für die ersten CDs der achtziger, die aufgrund der damals verwendeten aggressiven Lacke heute schon häufig Datenfehler aufweisen. Bei neueren CDs hofft man bei guter Lagerung auf 50 bis 100 Jahre Haltbarkeit, aber auch das ist keine Ewigkeit. Doch man muss gar nicht so lange zurückgehen: 5¼-Zoll-(Floppy-)Disketten waren das wichtigste Datenaustausch-Medium bei PC bis in die neunziger Jahre. Wer kann Sie noch lesen? Und selbst die 3½-Zoll-Disketten suchen zunehmend vergeblich nach einem passenden Laufwerk. Wann trifft es den USB-Memory-Stick? Kein Wunder also, das sich mittlerweile Unternehmen auf die Langzeit-Archivierung von Daten spezialisieren. Die amerikanische Firma Iron Mountain (wegen ihres Stollensystems in Pennsylvania) beschäftigt weltweit 15.000 Mitarbeiter und freut sich über 2 Milliarden Dollar Jahresumsatz.

2.2.6 Aufgaben

1) Ein Mikroprozessor verfüge über 22 Adressleitungen. Wie groß ist sein physikalischer Adressraum?

   a) 640 KByte b) 1 MByte c) 2 MByte
   d) 4 MByte e) 8 MByte f) 16 MByte

2) Welche der folgenden Eigenschaften hat keinen Einfluss auf die Arbeitsgeschwindigkeit eines Rechners?

   a) Taktrate b) Anzahl der Datenregister c) Breite des Adressbus
   d) verfügbarer Hauptspeicher e) Plattenzugriffszeit f) Breite des Datenbus

3) Welche der folgenden Adressen entsprechen absolut der SEGMENT:OFFSET – Adresse `0040:0010 im Real Mode`?

   a) `0050:0000` b) `0000:0050` c) `0410:0000`
   d) `0000:0410` e) `0140:0000` f) keine

4) Welcher der folgenden Analog/Digital-Wandler-Auflösungen wird mindestens benötigt, um ein Signal mit einer Dynamik von 1 : 2000 noch genügend genau zu digitalisieren?

a) 8 Bit b) 10 Bit c) 12 Bit
d) 16 Bit e) 18 Bit f) alle sind zu ungenau

5) Der fortschrittliche Mikrocomputer ROBONIX O.K. (O.K. = ohne Kurbel) besitzt eine Register-Wortbreite von 11 Bit. Negative ganze Zahlen stellt er im 2er-Komplement dar. Welcher Dezimalzahl entspricht der Registerinhalt 4FE hexadezimal?

a) 770 b) -770 c) -771
d) -769 e) 769 f) 771

6) Das Premiummodell ROBONIX Turbo G.O.K. (Garantiert ohne Kurbel) verfügt sogar über 13 Bit. Auch es arbeitet mit der 2er-Komplementdarstellung. Welches Ergebnis ergibt bei ihm die Rechnung `AD2` + `AD2` (hex)?

a) `05A4` b) `15A4` c) `25A4`
d) Fehler e) `1234` f) `AD2`

7) Ein 12-Bit-Register enthalte nacheinander folgende Werte in 2er-Komplementdarstellung (hexadezimal). Welche der folgenden Zahlen ist die kleinste?

a) `000` b) `777` c) `FFF`
d) `7FF` e) `807` f) `EDF`

8) Welche Baueinheit eines Mikrorechners ist auf die Registerbreite abgestimmt?

a) Adressbus b) Datenbus c) Steuerbus
d) RAM e) Taktgenerator

9) Tattam, der schreckliche digitale Herrscher des fernen Planeten Irakus verfügt über 25 Bit, um alle Untertanen ständig unter Kontrolle zu halten. Welche der folgenden Einwohnerzahlen seines Reiches ist die größte (von den genannten), die er noch kontrollieren kann?

a) 16733322 b) 16888888 c) 64000
d) 100000000 e) 31123456 f) 33666666

10) Ein 10-Bit-A/D-Wandler habe einen Messbereich von 0 bis 10 Volt. Die Messung erfordert eine Genauigkeit von 9 Bit. Welcher der folgenden Rauschpegel (alle in Millivolt) ist gerade noch tolerabel ?

a) 1000 b) 500 c) 100 d) 10
e) 1 f) 19 g) 2000 h) 255

11) Ein 16-Bit-Register enthalte nacheinander folgende Werte in 2er-Komplementdarstellung (hex). Welche der folgenden Zahlen ist die kleinste?

a) `8008` b) `0` c) `80` d) `FFFF` e) `7FFF` f) `AAAA`

12) Ein 8-Bit-Register enthalte nacheinander folgende Werte in 2er-Komplementdarstellung (hex). Welche der folgenden Zahlen ist die kleinste?

a) `00` b) `87` c) `FF` d) `7F` e) `88` f) `83`

13) Ein Mikroprozessor verfüge über 20 Adressleitungen. Wie groß ist sein physikalischer Adressraum?

a) 64 KByte b) 640 KByte c) 1 MByte d) 16 MByte e) 1 GByte

14) Wie viele unterschiedliche Speicherstellen (absolute Adressen beim INTEL-Prozessor) sprechen die folgenden SEGMENT:OFFSET - Adressen im Real Mode an?

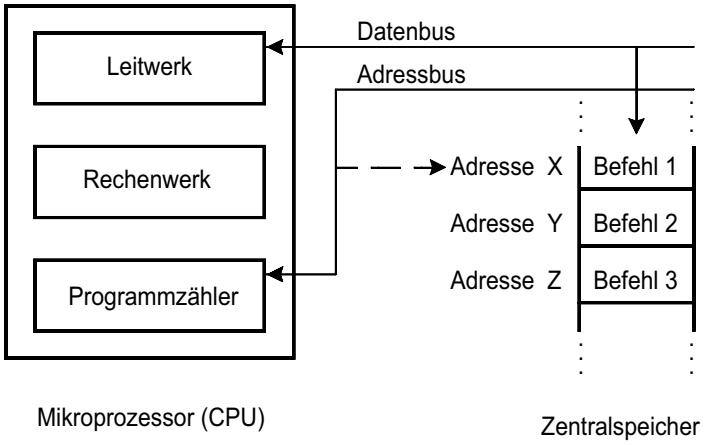
`1111:0222 AF07:2932 AA46:6542 B010:1B82 B0F6:0D32 AC56:4732`

a) 1 b) 2 c) 3 d) 4 e) 5 f) 6

3 Einführung in die Maschinensprache

Programme müssen vor ihrer Ausführung in binär kodierter Form in den Zentralspeicher des Rechners geladen werden. Programme bestehen aus einer endlichen Anzahl von *Maschinenbefehlen*. Für diese Maschinenbefehle existiert keine Norm, lediglich die Befehle innerhalb einer Prozessorfamilie, z. B. INTEL 80(X)86 oder MOTOROLA 680X0, sind kompatibel. Allerdings folgen alle *Maschinensprachen* einem ähnlichen Schema, so dass man sehr leicht eine neue erlernt, wenn man bereits eine beherrscht. Jede Maschinensprache besteht aus einer begrenzten Anzahl von Befehlen, meist in der Größenordnung von 100, die jedoch in vielen Varianten auftreten können.

Beim Programmstart wird der Befehlszeiger (Programmschrittzähler, beim INTEL 80(X)86 IP-Register) auf den ersten Befehl des Programms gesetzt, d. h. die Speicheradresse des ersten Befehls (nicht notwendigerweise 0!) wird in den Befehlszeiger geladen. Diese Aufgabe übernimmt das Betriebssystem.



Der so adressierte Befehl wird über den Datenbus in die CPU transportiert. Anschließend wird er vom Leitwerk decodiert und, ggf. mit Hilfe des Rechenwerks, ausgeführt. Im Anschluss daran wird der nächste Befehl (im Bild bei Adresse Y) geladen, ausgeführt, usw.

Einzelne Maschinenbefehle können ungleich lang sein. Sie können aus einem oder mehreren Bytes bestehen. Der Befehlsdecoder erkennt jedoch aus den Informationen des ersten Bytes die Länge des aktuellen Befehls. Stets wird vor der Ausführung der komplette Befehl geladen. Der Befehlszähler wird um die entsprechende Anzahl von Bytes weiter gezählt. Moderne Prozessoren, wie der Pentium, sind in der Lage, mehrere Befehle gleichzeitig, nur um einige Prozessortakte versetzt, zu verarbeiten (Parallelverarbeitung). Größere Abschnitte des Programmcodes werden vor der Ausführung in den Cache (→ s. Kap. 2.2.4) geladen, um die Zugriffszeit für das Leitwerk zu verkürzen.

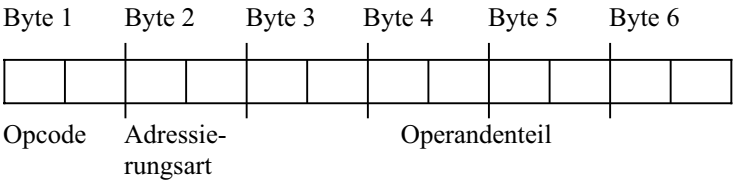
Um die Arbeitsweise eines Mikroprozessors kennen zu lernen, nutzen wir die Kompatibilität aller späteren Intel-Prozessoren zum 8086 und verwenden unseren PC als hätte er noch einen 16-Bit-Prozessor. Dabei hilft uns die Eigenschaft der Konsole, zwischen 16- und 32-Bit-Programmen zu unterscheiden. 16-Bit-Programm werden unter Windows in der Multitasking Umgebung im Virtual Mode (→ s. Kap. 2.2.2) ausgeführt. Für uns als Benutzer stellt sich dies

wie ein Programm unter DOS im Real Mode dar. Erst im weiteren Verlauf (Inline-Assembler mit C++) nutzen wir die 32-Bit-Fähigkeit moderner Prozessoren. Hier geht es uns jedoch in erster Linie um die prinzipielle Arbeitsweise von Prozessoren und dazu genügen uns die Fähigkeiten des 8086-Prozessors.

3.1 Maschinenbefehle des 8086

Je nach Befehl und Adressierungsart (→ s. Kap. 3.4 und 3.5) kann ein Maschinenbefehl eines 8086-Prozessors 1 bis 6 Byte lang sein.

Der generelle Aufbau ist wie folgt:



Der Opcode (Anweisungsteil) besagt, *was* getan werden soll, der Operandenteil *womit*. Auf die Adressierungsart gehen wir später ein (→ s. Kap. 3.5).

Maschinenbefehle werden in der Regel in hexadezimaler Schreibweise angegeben, obwohl sie selbstverständlich in Wahrheit binär kodiert sind.

Ein Beispiel: `BA3740` ist ein 3-Byte-Befehl, er bedeutet:
„Lade die Zahl `4037` hexadezimal in das DX-Register“
`BA` ist der Opcode, `3740` der Operandenteil.

Obwohl die hexadezimale Schreibweise handlicher ist als die binäre, ist es kaum möglich, auch nur die wichtigsten Maschinenbefehle „im Kopf“ zu behalten. Aus diesem Grund wurde die *Assemblersprache* geschaffen, die mit *Mnemonics* (gedächtnisstützenden Abkürzungen) arbeitet. Für jeden Maschinenbefehl gibt es genau einen mnemonischen Code. Es existiert also, anders als bei Hochsprachen, eine 1 : 1 Zuordnung zwischen Assemblerbefehl und Maschinenbefehl. Die Mnemonics sind vom Prozessorhersteller vorgeschrieben.

In der Assemblersprache lautet unser obiger Befehl:

```
MOV DX,4037
```

Assemblerbefehle sind nicht unmittelbar vom Prozessor interpretierbar. Sie müssen zunächst von einem Programm namens *Assembler* in Maschinenbefehle übersetzt werden, so wie ein C-Programm von einem *Compiler* übersetzt wird.

`MOV DX,4037` → ASSEMBLER → `BA3740`

Zu einem Prozessor gibt es durchaus mehrere Assembler von unterschiedlichen Softwareherstellern, z. B. MASM von Microsoft und TASM (T wie Turbo) von Borland. Während

Mnemonics und Registerbenennungen, wie gesagt, vorgeschrieben sind, kann die Syntax der übrigen Operanden (Zahlen, Adressen, Labels, Symbolische Namen) geringfügig differieren. Da wir nicht davon ausgehen können, dass Sie einen kommerziellen Assembler besitzen, arbeiten wir im Folgenden mit der einfachen Syntax des 16-Bit-Hilfsprogramms Debug, das Bestandteil jedes Windows-Betriebssystems ist.

Vielleicht ist Ihnen beim obigen Maschinenbefehl eine Ungereimtheit aufgefallen: `MOV DX,4037` wird übersetzt mit `BA3740`. High und Low Byte des Operanden wurden demnach vertauscht.

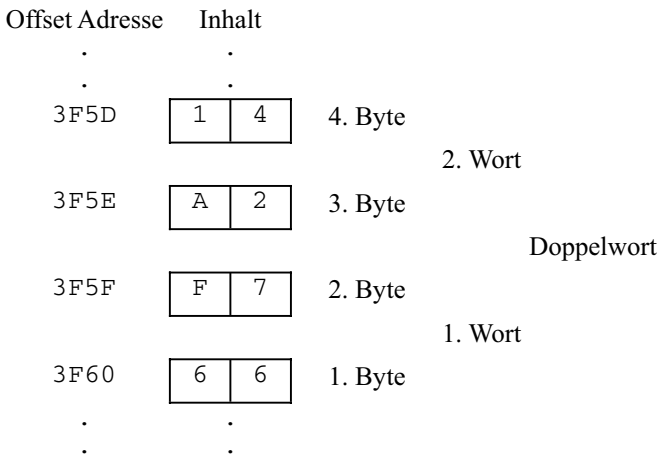
Das hängt mit der *INTEL-Konvention* zur Datenspeicherung zusammen. Anders als bei den meisten anderen Prozessoren (z. B. MOTOROLA 680X0) gilt:

INTEL-Konvention der Datenspeicherung
High Byte auf High-Adresse
Low Byte auf Low-Adresse

Das gilt auch für Doppelworte:
High Word auf High-Adresse
Low Word auf Low-Adresse

■ **Ein Beispiel**

Das Doppelwort (32 Bit Wert) `66F7A214` (hexadezimal) wird von den 80(X)86-Prozessoren folgendermaßen abgespeichert:



■

Wir betrachten einige weitere Beispiele für Maschinenbefehle, doch von nun an stets in der Assemblerschreibweise. Wir benutzen dabei, wie gesagt, die Syntax des einfachen DEBUG-Assemblers.

■ **Beispiel**

`MOV CX,AX` bedeutet: kopiere den Inhalt von Register `AX` nach Register `CX`

`ADD AX,BX` bedeutet: `AX` = `AX` + `BX`

Die INTEL-Konvention für die Reihenfolge der Operanden lautet:

Hat ein Befehl zwei Operanden (Quelle und Ziel), so gilt:

1. Operand ist das Ziel
2. Operand ist die Quelle

Beispiel: `MOV BX,DX`

Hat ein Befehl zwei Operanden (zwei Quellen und ein Ziel), so gilt:

1. Operand ist Quelle und Ziel
2. Operand ist Quelle

Beispiel: `ADD DX,CX`

■

Wir werden mutiger und wagen uns an eine kleine Befehlsfolge heran:

```
MOV AX,1234
MOV CX,2345
SUB CX,AX          Subtraktion: CX - AX ⇒ CX
```

Welche Inhalte haben `AX` und `CX` nach dem `SUB`-Befehl? Natürlich enthält `CX` dann `1111` (hexadezimal), während `AX` unverändert bleibt.

> Beim DEBUG-Assembler sind alle numerischen Konstanten automatisch *hexadezimal*

Assembler wie MASM oder TASM sind nicht nur bezüglich der Zahlenschreibweise komfortabler als unser DEBUG-Assembler. Dennoch eignet sich dieser sehr gut zum Erlernen der Maschinensprache, zumal er unmittelbar verfügbar ist. Außerdem ist der Umgang mit DEBUG leicht zu erlernen.

3.2 Das Hilfsprogramm DEBUG

DEBUG stammt noch aus der DOS-Welt und wird entsprechend durch Eingabe seines Namens in der Konsole (Eingabeaufforderung) gestartet. Eigentlicher Zweck dieses umfangreichen interaktiven Kommandos ist die Fehlersuche (*debugging*) auf Maschinenbefehlsebene bei der Programmentwicklung im Real Mode. Auf der Webseite zu diesem Buch (URL siehe Vorwort oder Rückseite des Einbands) finden Sie eine ausführliche Beschreibung der wichtigsten DEBUG-Befehle mit Anwendungsbeispielen. Mit zunehmendem Lernfortschritt können Sie sich dort bei Bedarf nach und nach mit den spezielleren Möglichkeiten von DEBUG vertraut machen. An dieser Stelle genügt eine Kurzreferenz.

DEBUG ermöglicht das „Herumstochern“ im Zentralspeicher des Rechners im Real Mode. Das klingt zunächst gefährlich und ist es im Prinzip auch. Unbedachte Eingaben können zum Absturz führen. Unter Windows 98 sogar zum Absturz des kompletten Rechners (ein Neustart repariert alles). Unter Windows XP wirkenden alle Zugriffe auf Speicher und Hardware nur

„lokal“, also innerhalb der 16-Bit-Umgebung, die Ihnen das Betriebssystem beim Start von DEBUG zur Verfügung stellt. „Gefährliche“ Aktionen, etwas das Abschalten des Tastatur-Interrupts, gelten nur innerhalb Ihrer Konsole, die sonstige Windows XP Umgebung funktioniert weiterhin problemlos.

Nach Aufruf meldet sich DEBUG mit einem „-“ als Prompt und wartet auf die Eingabe von Ein-Buchstaben-Befehlen, evtl. gefolgt von Parametern. Der Befehl „q“ beendet DEBUG und führt Sie auf die Shellebene (Eingabeaufforderung „>“ der Konsole) zurück. Die Konsole starten Sie unter Windows mit <Start> ⇒ <Ausführen> ⇒ cmd.

Hier nun eine Kurzbeschreibung der wichtigsten Befehle von DEBUG anhand von Beispielen:

Aufruf:
```
>DEBUG
>DEBUG F:UEB1.COM
```
(Einlesen der Datei UEB1.COM von Laufwerk F zur Bearbeitung mit DEBUG)

Hex-Arithmetik:
```
-H 3 2
-H 2 3
-H 3D5C 2A10
-H 5 FFFF
```
(zeigt Summe und Differenz der jeweiligen Hexadezimalzahlen)

Register anzeigen und ändern:
```
-R
```
(zeigt alle 16-Bit-Register und Flags)
```
-R AX
```
(zeigt nur AX)
```
-R IP
```
(u.s.w.)
```
-R SP
```

Speicherplätze ansehen und ändern:
```
-E 100
```
(nächste Speicherstelle: jeweils <SPACE> drücken)
Die Ausgabe des E-Kommandos sieht etwa so aus: 25A6:100 00. Hinter dem „.“ können Sie einen neuen Inhalt eingeben,
z. B. 25A6:0100 00.01 00.d8

Single Step Modus (*Trace*):
```
-T
```
(Ausführung eines einzelnen Maschinen-Befehls mit anschließender Anzeige aller Register)

Programm ausführen (*Go*):
```
-G
-G 1FE
```
(Programmausführung bis incl. Adresse CS:01FE)

Programm ausführen (*Proceed*):
```
-P
```
(wie T, jedoch werden komplette Unterprogramme wie ein Step behandelt)

Programme eingeben (*Assemble*):
```
-A 100
```
(nächster Assemblerbefehl:
jeweils<CR> drücken,
Ende: nochmals <CR> drücken)

Programm ausgeben (*Unassemble*):	`-U`	(16 Befehle ab `IP`)
	`-U 200`	(16 ab Adresse `200`)
	`-U 100 200`	(Adr. `100` bis `200`)
Speicherinhalte ausgeben (*Dump*):	`-D`	(8 Zeilen ab IP)
	`-D 200`	(8 ab Adresse `200`)
	`-D 100 200`	(Adr. `100` bis `200`)
Programme benennen (*Name*):	`-N WRITESTR.COM`	
Programm auf Platte schreiben (*Write*):	`-W`	(vorher die ungefähre Länge des Programms in Byte in das Registerpaar `BX:CX` schreiben!)
DEBUG verlassen:	`-Q`	

Groß- oder Kleinschreibung spielt keine Rolle!

Symbole für die Flags in Abhängigkeit des Zustandes des Flagregisters:

Flagname	gesetzt	gelöscht
-Overflow(yes/no)	`OV`	`NV`
-Direction (decrement/increment)	`DN`	`UP`
-Interrupt (enable/disable)	`EI`	`DI`
-Sign (negative/positive)	`NG`	`PL`
-Zero (yes/no)	`ZR`	`NZ`
-Auxiliary carry (yes/no)	`AC`	`NA`
-Parity (yes/no)	`PE`	`PO`
-Carry (yes/no)	`CY`	`NC`

Diese Flagzustände erhalten Sie zusammen mit den Registerinhalten nach Eingabe des R-Kommandos.

Seien Sie nicht enttäuscht, wenn Ihnen diese Kurzbeschreibung allzu kurz vorkam. Das Verständnis kommt nach und nach mit den Übungen. Ggf. steht Ihnen die ausführliche Anleitung auf unserer Buch-Webseite zur Verfügung.

Wenn Sie DEBUG starten, stellt Ihnen das Programm ein volles 64 KByte-Segment zur Verfügung. Alle Segmentregister zeigen auf die gleiche Adresse. Das entspricht dem Real-Mode-Speichermodell COM (→ s. Kap. 2.2.2 und 4.2). Diese Einstellung sollten Sie nicht verändern. Damit sind die Segmentadressen fest, und Sie müssen sich lediglich um die Offsetadressen kümmern. Aus einem später zu erläuternden Grund beachten Sie bitte folgende Regel:

> Beginnen Sie jedes DEBUG-Maschinenprogramm bei der Offset-Adresse 100(hex) oder höher.

DEBUG bietet Ihnen die Startadresse `100` standardmäßig an, indem es den Programmschrittzähler (IP) zu Beginn automatisch auf `100` setzt. Prüfen Sie das mit dem R-Kommando nach.

3.3 Aufgaben

Benutzen Sie DEBUG zur Lösung der folgenden Aufgaben.

0) Starten Sie DEBUG und machen Sie sich mit der wichtigsten DEBUG-Befehlen vertraut:

`R`	alle Register ansehen
`R AX`	das AX-Register verändern
`R IP`	den Programmschrittpointer setzen
`E 100`	Speicherplatz `100` ansehen und ggf. verändern

1) Berechnen Sie die hexadezimalen Werte der folgenden Dezimalzahlen unter Verwendung des H-Befehls (Achtung: - bedeutet bei den folgenden Zahlen „minus“, die Beträge der folgenden negativen Dezimalzahlen müssen Sie zunächst „auf dem Papier“ oder per Taschenrechner nach hexadezimal umrechnen):

```
-1
-100
-200
-9999
-32000
```

Beispiel für -100: `-H 0 64`
Ausgabe: `64 FF9C` (Summe und Differenz)

Die Differenz (zweite Hex-Zahl) gibt den Wert für -100 (dezimal) in der hexadezimalen Zweierkomplementdarstellung (= FF9C) aus.

2) Addieren Sie die Register `AX` und `BX` mit `ADD AX,BX`.

Der Maschinenbefehl lautet: `D801`

Hinweis: Geben Sie den Befehl byteweise ab Adresse 100 mit dem E-Kommando ein, (erst `01`, dann `D8` eingeben, Intel-Konvention!).
Führen Sie ihn mit dem `T`-Kommando aus.

Verändern Sie mehrmals die Inhalte von `AX` und `BX` (`R`-Kommando) und wiederholen Sie jeweils die Befehlsausführung, indem Sie das `IP`-Register immer wieder auf `100` setzen, dann erneut `T`-Kommando.

3) Subtrahieren Sie die Register `AX` und `BX` mit `SUB AX,BX`.

Der Maschinenbefehl lautet: `D829`

Hinweis: Geben Sie den Befehl byteweise mit dem `E`-Kommando ab Adresse 100 ein. Beachten Sie die Intel-Konvention. Führen Sie ihn mit dem `T`-Kommando aus. Dabei muss `IP` auf `100` stehen, sorgen Sie ggf. mit dem `R`-Befehl dafür).

Verändern Sie mehrmals die Inhalte von `AX` und `BX` (`R`-Kommando) und wiederholen Sie jeweils die Befehlsausführung, so dass auch negative Ergebnisse und Null resultieren (`IP` immer wieder auf `100` setzen).

Beachten Sie die Wirkung auf die Flags, die durch jeweils zwei Buchstaben symbolisiert sind.

4) Einfacher ist die Anwendung des `A`-Befehls (Assembler). Setzen Sie den `IP` wie gewohnt auf Adresse `100` und geben Sie ein:

`-A 100`
anschließend: `SUB AX,BX`
dann zweimal <CR> drücken

Mit
`-U 100`
können Sie überprüfen, ob der DEBUG Assembler (`A`-Befehl) die korrekte Übersetzung in die Maschinensprache vorgenommen hat.

5) Lesen Sie die ausführliche Anleitung zum Arbeiten mit DEBUG (Download von unserer Buch-Webseite) und probieren Sie möglichst viel aus.

3.4 Befehlsarten

Die Menge aller Maschinenbefehle eines Prozessors nennt man *Befehlssatz*.

Die verschiedenen Befehle lassen sich zu Gruppen, den *Befehlsarten* zusammenfassen. Beim 8086/88-Prozessor unterscheiden wir 6 Befehlsarten:

1. Transportbefehle		**2. Arithmetische Befehle**	
MOV	übertrage Operanden	ADD	addiere
IN	periphere Eingabe	SUB	subtrahiere
OUT	periphere Ausgabe	INC	erhöhe um 1
PUSH	schreibe Register auf Stack	DEC	erniedrige um 1
POP	lese Register von Stack	CMP	vergleiche
PUSHF	schreibe Flagregister auf Stack	NEG	negiere (2er- Komplement)
POPF	lese Flagregister vom Stack	MUL	multipliziere
…		DIV	dividiere
		...	
3. Logische Befehle		**4. Sprungbefehle**	
AND	logisches UND	JMP	unbedingter Sprung
OR	logisches ODER	JG	springe, wenn größer
XOR	exclusives ODER	JNZ	springe, wenn ungleich 0
NOT	1er-Komplement	LOOP	springe, wenn CX ≠ 0
SHR	logischer Rightshift	CALL	Unterprogrammsprung
SHL	logischer Leftshift	RET	Rücksprung aus U.P.
ROR	arithm. Rightshift	INT	Softwareinterrupt
ROL	arithm. Leftshift	IRET	Rücksprung aus ISR
...		...	
5. Prozessorkontrolle		**6. Stringbefehle**	
HLT	halte Prozessor an	werden im Rahmen dieses Buches nicht behandelt	
STC	setze Carry Flag		
CLC	lösche Carry Flag		
...			

Es ist nicht notwendig, alle Befehle eines Prozessors vollständig zu behandeln. Wegen der Gemeinsamkeiten genügt es, wichtige und typische Vertreter der jeweiligen Art vorzustellen.

Eine Beschreibung des vollständige Befehlssatz finden Sie auf unserer Buch-Webseite (→ s. Vorwort) zum Download. Es sei nochmals erwähnt, dass wir im Folgenden Assembler-Befehle in der DEBUG-Syntax darstellen. Damit bewegen wir uns in der 16-Bit-Welt.

3.4.1 Transportbefehle

Der wichtigste Transportbefehl ist zweifellos der bereits oben erwähnte MOV-Befehl. Die folgende Tabelle zeigt die zahlreichen Variationsmöglichkeiten dieses Befehls. Hier spielen allerdings schon die *Adressierungsarten*, die wir systematisch erst im folgenden Abschnitt behandeln wollen, eine Rolle. Beispielsweise bedeuten:

■ **Beispiel**

`MOV AX,DX`	transportiere den Inhalt von Register `DX` nach `AX`
`MOV CL,AH`	transportiere den Inhalt von Halbregister `AH` nach `CL`
`MOV BX,1000`	lade die Zahl `1000`(hex) ins `BX`-Register
`MOV BL,C4`	lade die Zahl `C4`(hex) ins `BL`-Halbregister
`MOV [1000],CH`	transportiere den Inhalt von Halbregister `CH` zum Speicher 1000
`MOV [1000],CX`	transportiere den Inhalt von Register `CX` zum Speicher `1000` und `1001` (warum auch `1001`? Weil eine Adresse nur ein Byte aufnehmen kann, ein Vollregister wie `CX` aber zwei Bytes enthält! Es gilt die INTEL-Konvention High Byte auf High-Adresse) Diese Adressierungsart nennt man „Register zu Speicher“ oder „direkt“.

■

Als Operanden kommen prinzipiell Registerinhalte, Speicherinhalte und Konstanten in Frage. `MOV` kann als Prototyp der Zwei-Operand-Befehle gelten. Ein Operand ist in der Regel ein Register. Dieses bestimmt die Wortbreite der Operation (16 Bit bei Vollregister, 8 Bit bei Halbregister). Gemischte Operationen (8- und 16-Bit-Operanden in einem Befehl) sind unzulässig!

`MOV AX,BL` unzulässig, da unterschiedliche Registerbreite

Der `MOV`-Befehl ist der statistisch häufigste in einem Programm. Ihm kommt die Funktion eines „Spediteurs“ zwischen den „Produktionsstätten“ wie `ADD`, `SUB`, `AND`, ... zu. `MOV`, oder andere Transportbefehle, versorgen die Register mit den gewünschten Inhalten. Das ist notwendig, weil die INTEL-Prozessorfamilie maximal Speicher-zu-Register-Befehle zulässt. Andere Prozessoren, wie der TM 9900 von Texas Instruments, besitzen Speicher-zu-Speicher-Architektur. Bei solchen Prozessoren spielen Transportbefehle eine geringere Rolle.

`MOV [1000],[2000]` unzulässig, da Speicher-zu-Speicher-Zuweisung

Weitere wichtige Transportbefehle sind `PUSH` und `POP`, mit denen Registerinhalte auf den Stack gesichert, bzw. wieder zurückgeholt werden. Wir werden in Zusammenhang mit *Unterprogrammen* (→ s. Kap. 5.1) darauf zurückkommen.

Mit `IN` und `OUT` lassen sich Daten zwischen Prozessor und I/O-Ports austauschen. Auch diesem Punkt ist ein besonderes Kapitel gewidmet (→ s. Kap. 5.2).

3.4.2 Arithmetische Befehle

Alle arithmetischen Befehle arbeiten mit Ganzzahl-Operanden (8 Bit, 16 Bit oder gar 32 Bit). Für Gleitkomma-Operationen bedient man sich des mathematischen Coprozessors oder einer Softwareemulation desselben.

Eindeutig stehen die beiden uns bekannten arithmetischen Anweisungen `ADD` und `SUB` in der Hitliste ganz oben:

`ADD AX,DX`	Addiere den Registerinhalt von `DX` zu `AX`
`ADD BL,CH`	Addiere den Halbregisterinhalt von `CH` zu `BL`
`ADD [1000],BX`	Addiere den Registerinhalt von `BX` zum Inhalt des Speichers `1000` und `1001`
`SUB DH,4F`	Subtrahiere `4F`(hex) vom Inhalt des Halbregisters `DH`
`SUB AX,[2000]`	Subtrahiere den Inhalt des Speichers `2000` und `2001` vom Inhalt des Registers `AX` (die Registerbreite bestimmt die Wortbreite der Operation)

Ungewöhnlich sind die Befehle für die vorzeichenlose `Integer`-Multiplikation und -Division `MUL` und `DIV`. Beide Befehle fallen insofern aus dem bisher gesteckten Rahmen, weil sie *implizit* von bestimmten Registern Gebrauch machen.

■ **Beispiel MUL**

Gegebene Registerinhalte:

DX	AX	BX
0000	5DE7	0100

die Operation: `MUL BX`

führt zu folgendem Ergebnis:

DX	AX	BX
005D	E700	0100

denn MUL BX bedeutet: `AX` · `BX` ⇒ (`DX`,`AX`)

Erklärung: Implizit wird das Register `AX` mit dem angegebenen Register (hier `BX`) multipliziert. Das Ergebnis landet ebenfalls implizit im 32-Bit-Registerpaar (`DX`,`AX`), wobei `DX` das High-Word aufnimmt. ■

■ **Beispiel DIV**

Gegebene Registerinhalte:

DX	AX	BX
003E	F9D7	0100

Operation: `DIV BX`

führt zu folgendem Ergebnis:

DX	AX	BX
00D7	3EF9	0100

denn `DIV BX` bedeutet: (`DX`,`AX`) / `BX` ⇒ `AX`,
Rest ⇒ `DX`

Erklärung: Implizit wird das Registerpaar (`DX`,`AX`) durch das angegebene Register (hier `BX`) geteilt. Das Ergebnis landet, ebenfalls implizit, im Register `AX`, wobei `DX` den Rest aufnimmt. `MUL` und `DIV` sind somit kompatibel. ■

MUL und DIV erlauben also 16/32-Bit-„Punktrechnung“; die 8/16-Bit-Variante arbeitet nur mit AX als implizitem Register:

■ **Beispiel MUL (8/16 Bit)**

Gegebene Registerinhalte:

AH	AL	BL
08	80	04

die Operation: `MUL BL`

führt zu folgendem Ergebnis:

AH	AL	BL
02	00	04

denn MUL BL bedeutet: AL · BL ⇒ (AX)

Erklärung: Implizit wird das Halbregister AL mit dem angegebenen Halbregister (hier BL) multipliziert. Das Ergebnis landet ebenfalls implizit im Register AX. ■

■ **Beispiel DIV (8/16 Bit)**

Gegebene Registerinhalte:

AH	AL	BL
02	03	04

Operation: `DIV BL`

führt zu folgendem Ergebnis:

AH	AL	BL
03	80	04

denn DIV BL bedeutet: AX / BL ⇒ AL, Rest nach AH

Erklärung: Implizit wird das Register AX durch das angegebene Halbregister (hier BL) geteilt. Das Ergebnis landet, ebenfalls implizit, im Register AL, wobei AH den Rest aufnimmt. MUL und DIV sind somit kompatibel. ■

Der Befehl INC incrementiert das angegebene Register um 1.
Beispiel: `INC AX` bedeutet: `AX = AX + 1`

Entsprechend decrementiert DEC um 1.
Beispiel: DEC DX bedeutet: DX = DX - 1

Achtung: INC und DEC lassen das Carry Flag unbeeinflusst!

NEG negiert, bildet also das 2er-Komplement.
Beispiel:
```
MOV AH,1
NEG AH     macht aus 1 -1 (FFh)
```

CMP (*Compare*) arbeitet im Prinzip wie SUB, ändert jedoch keine Operandeninhalte, sondern vergleicht nur und setzt die betroffenen Flags entsprechend.
Beispiel: CMP DX,1000 bedeutet: setze Flags entsprechend der Operation DX – 1000h (Ergebnis positiv, negativ oder 0)

CMP wird meist vor bedingten Sprungbefehlen (→ s. Kap. 3.4.4) eingesetzt.

3.4.3 Logische Befehle

Die folgenden logischen Befehle führen Boolesche Operationen durch:
AND, OR, XOR, NOT.

Und so sehen die entsprechenden Wahrheitstafeln der *bitweisen* Verknüpfungen aus:

AND (und)

Wahrheitstafel

Op.1	0	1	0	1
Op.2	0	0	1	1
Erg.	0	0	0	1

Beispiel:

```
      0101 1100
AND   1111 0000
      0101 0000
```

OR (oder)

Wahrheitstafel

Op.1	0	1	0	1
Op.2	0	0	1	1
Erg.	0	1	1	1

Beispiel:

```
      0101 1100
 OR   1010 1010
      1111 1110
```

XOR (exklusives oder)

Wahrheitstafel

Op.1	0	1	0	1
Op.2	0	0	1	1
Erg.	0	1	1	0

Beispiel:

```
      1101 0010
XOR   0000 1111
      1101 1101
```

NOT (Komplement)

Wahrheitstafel

Op.	0	1
Erg.	1	0

Beispiel:

```
NOT   0011 1011
      1100 0100
```

Angewendet auf unsere Maschinenbefehle ließe sich folgendes Beispiel konstruieren:

■ **Beispiel**

```
MOV AH,4D      AH   0100 1101   (4D hex)
AND AH,F0      AND  1111 0000   (F0 hex)
               AH   0100 0000   (40 hex)
```

■

Eine solche Operation nennt man „Maskierung“: Ein Registerinhalt wird mit einer „Bitmaske“ verknüpft. Hier dient sie dazu, das zweite Nibble von AH auszublenden, d. h. auf Null zu setzen.

Mit einer „Oder-Maske“ lassen sich gezielt bestimmte Bits, z. B. jedes zweite, setzen. Der Zustand der anderen Bits richtet sich nach dem ursprünglichen Registerinhalt:

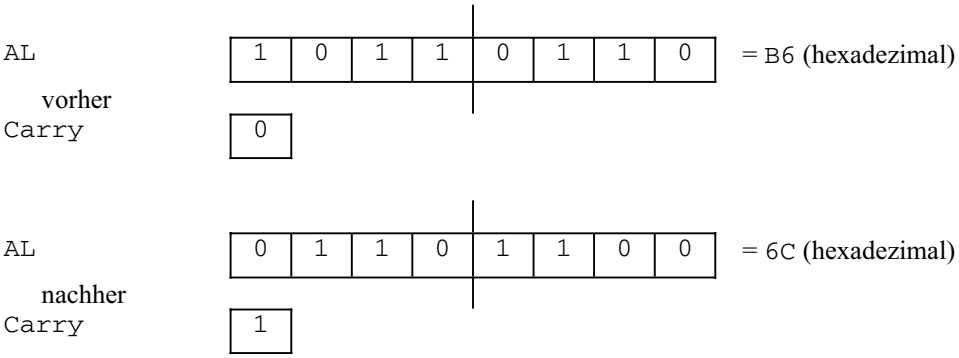
■ **Beispiel**

```
MOV AH,4D     AH  0100 1101  (4D hex)
OR  AH,55     OR  0101 0101  (55 hex)
              AH  0101 1101  (5D hex)
```
■

Sehr wichtig sind auch die Bit-Shift-Operationen. Stellvertretend werden zwei in Beispielen vorgestellt:

`RCL AL,1` bedeutet *Rotate Carry Left* (hier: AL-Register)

die Wirkung:

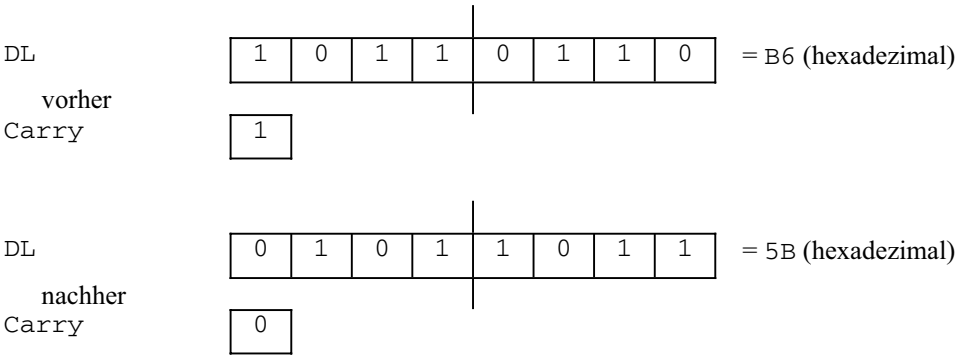


Also: das MSB geht ins Carry Flag, das alte Carry Flag wird neues LSB, daher „Rotation“.

Anwendung: Lichtorgel

`SHR DL,1` bedeutet *Shift Right* (hier: DL-Register)

die Wirkung:



Also: das LSB geht ins Carry Flag, im MSB wird eine 0 nachgeschoben.

Anwendung: schnelle Integer-Division durch 2.

3.4.4 Sprungbefehle

Die Sprungbefehle stellen eine weitere Befehlsart dar, die jeder Prozessor in leicht abgewandelter Form aufzuweisen hat. Auch hier sollen Beispiele den Einsatz erläutern:

Wir nehmen an, dass `2000` eine Offset-Adresse innerhalb eines kleinen Maschinenprogramms ist.

Unbedingter Sprung:

`JMP 2000`	Sprung zur Offset-Adresse `2000` (jump)

Bedingte Sprünge hängen vom Zustand bestimmter Flags ab, z. B.:

Bedingter Sprung:

`JNZ 2000`	springe zur Adresse `2000`, wenn das Zero Flag nicht gesetzt ist (*jump if not zero*)
`JGE 2000`	springe zur Adresse `2000`, wenn das Zero Flag gesetzt ist *oder* wenn das Sign Flag auf positiv steht (*jump if greater or zero*)

Aus dem Anhang der kompletten Maschinenbefehle auf unserer Buch-Webseite zu diesem Kapitel ist die Vielzahl der Sprungbefehle ersichtlich.

Wir starten DEBUG und geben ein kleines Programm ein:

`-A 100`	Assemblerprogramm ab Offset-Adresse `100`
`XXXX:0100 MOV AL,3`	`AL = 3`
`XXXX:0102 DEC AL`	`AL = AL - 1`
`XXXX:0103 JNZ 102`	Sprung nach Adresse `102` wenn `AL` ungleich `0`
`XXXX:0106 MOV AH,4C`	Function Code für „Programm beenden“
`XXXX:0108 INT 21`	Software-Interrupt (wird später erklärt)
`XXXX:010A`	erneut ENTER-Taste drücken
`-`	

Einige Erläuterungen:

`XXXX:YYYY` gibt jeweils der DEBUG Assembler aus, `XXXX` steht für eine (nicht vorhersehbare aber feste) Segment-Adresse, die DEBUG von sich aus anbietet, `YYYY` ist die Offset-Adresse, die nacheinander die oben stehenden Werte (`0100`, `0102`, ...) annimmt. Diese Adressen hängen von der Länge des jeweiligen Maschinenbefehls ab: so benötigt `MOV AL,3` zwei Byte (Adressen `100` und `101`), `DEC AL` dagegen nur eins (Adresse `102`). Zum Glück berechnet der DEBUG Assembler selbständig die Befehlslänge. Nach Eingabe eines Assemblerbefehls drücken Sie einmal die ENTER-Taste, nach dem letzten Befehl zweimal.

Das Programm lädt die Zahl 3 ins `AL`-Register und zählt dann `AL` um eins herunter. Solange `AL` ungleich (hier: größer) Null ist, erfolgt ein bedingter Sprung nach Adresse `102`, wo der `DEC`-Befehl steht. Anschließend werden die beiden Befehle `MOV AH,4C` und `INT 21` ausgeführt. Diese sorgen dafür, dass unser Programm ordnungsgemäß beendet wird. `INT 21` ist ein Softwareinterrupt, der eine System-Routine aufruft. Wir werden später hierauf zurückkommen (→ s. Kap. 4.1), zunächst betrachten wir die beiden Befehle als „Blackbox“.

Sie sollten das Programm im Trace Modus (T) bei Adresse `100` starten und die Register `AL` und `IP` beobachten. Nach Ausführung des Befehls `MOV AH,4C` sollten Sie entweder den Assemblermodus beenden oder ein G-Kommando eingeben. Hinter `INT 21` steht nämlich eine komplette DOS-Routine, deren Durchlaufen im Trace Modus zur abendfüllenden Beschäftigung werden kann. Eine Alternative stellt der Proceed Modus (P) dar, der dem T-Modus entspricht, jedoch INTs „überspringt".

Der Maschinenbefehl `LOOP` ist ebenfalls ein Sprungbefehl, jedoch ein komplexer. `LOOP` zählt das `CX`-Register um eins herunter (wie `DEC CX`). Solange `CX` ungleich 0 ist, erfolgt ein Sprung zu der angegebenen Adresse. Damit realisiert man eine Zählschleife, vergleichbar **for . . .** in C/C++. `LOOP` ist also ein Befehl, der wieder, wie schon `MUL` und `DIV`, *implizit* auf ein Register, das Zählregister `CX`, zugreift. Zur Illustration wandeln wir unser obiges Programmbeispiel geringfügig um:

■ **Beispiel**

```
XXXX:0100  MOV CX,3      Lade 3 nach CX
XXXX:0103  LOOP 103      Erniedrige CX um 1 und springe solange ungleich 0
XXXX:0105  MOV AH,4C     Function Code 4C
XXXX:0107  INT 21        Softwareinterrupt                                ■
```

Dieses kleine Programm kann auf analoge Weise ausprobiert werden.

`CALL` ruft ein Unterprogramm auf, mit `RET` wird das Unterprogramm wieder verlassen. Ein eigenes Kapitel behandelt diese Thematik. Die entsprechenden Befehle für Interrupt-Service-Routinen (statt „normaler" Unterprogramme) heißen `INT` und `IRET`. Auch diese werden uns später beschäftigen (→ s. Kap. 5).

3.4.5 Befehle zur Prozessorsteuerung

Meist sind es 1-Byte-Befehle, die verschiedene Prozessorzustände beeinflussen.

■ **Beispiele**

`STC` setzt *explizit* das Carry Flag (auf 1),

`CLC` löscht *explizit* das Carry Flag (auf 0). ■

Probieren Sie die beiden Befehle mit DEBUG aus! DEBUG zeigt alle Register *und* alle Flags mit ihrem aktuellen Zustand an. Das Carry Flag steht als letztes, NC bedeutet: Carry Flag = 0, CY bedeutet: Carry Flag = 1.

3.4.6 Aufgaben

1) Multiplizieren Sie die Register `AX` und `BX` mit `MUL BX`. `MUL` multipliziert den Inhalt von `AX` mit dem des angegebenen Registers (hier `BX`). Das Ergebnis steht im Registerpaar `DX:AX`.

   Hinweis: Geben Sie den Befehl mit dem A-Kommando unter DEBUG ein, führen Sie ihn mit dem T-Kommando aus.

   `AX` und `BX` sollen zuvor die Werte `8A33` bzw. `100` enthalten (R-Kommando).

   Das Resultat muss lauten: `DX = 008A`, `AX = 3300`, `BX = 0100`.

   Wiederholen Sie die Multiplikation mit anderen Registerinhalten.

2) Dividieren Sie die Register DX:AX durch BX mit DIV BX. DIV teilt immer das Registerpaar DX:AX durch das angegebene Register (hier BX). Das Ergebnis der Division steht in AX, der ganzzahlige Rest in DX.

   Hinweis: Geben Sie den Befehl mit dem A-Kommando ein, führen Sie ihn mit dem T-Kommando aus.

   Die Anfangswerte sollen lauten: DX = 008A, AX = 3321, BX = 0100.

   Das Resultat muss lauten: DX = 0021, AX = 8A33, BX = 0100.

   Wiederholen Sie die Division mit anderen Registerinhalten.

3) Überprüfen Sie die Wirkung des Befehls RCL (*Rotate Carry Left*).
   Setzen Sie BX auf den Wert 00AA.
   Setzen Sie IP auf den Wert 100.
   Setzen Sie zunächst das Carry Flag auf 0, d.h. Sie starten Ihr kleines Programm bei Adresse 100 mit CLC. Es folgt RCL BL,1.
   Führen Sie die Befehle im Trace Modus mit T aus.
   BL hat nun den Inhalt 54. Das Carry Flag ist gesetzt (CY).
   Wiederholen Sie die Ausführung (Achtung: IP wieder auf 101, usw.).
   BL hat nun den Inhalt A9. Das Carry Flag ist nicht gesetzt (NC).
   Wiederholen Sie die Ausführung (Achtung: IP wieder auf 101, usw.).
   BL hat nun den Inhalt 52. Das Carry Flag ist wieder gesetzt (CY).
   .
   Nach 8 Rotationen muss BL wieder der Wert AA besitzen.

   Sie können sich die Ausführung des Programms mit dem zusätzlichen Maschinenbefehl JMP 101 erleichtern.

4) Jeder Prozessor verfügt auch über logische Befehle.
   Einer davon bewirkt eine UND-Verknüpfung.
   Geben Sie bei Adresse 100 ein: AND DL,0F.
   Setzen Sie DX auf den Wert 00FF (mit dem R-Kommando).
   Setzen Sie IP auf den Wert 100.
   Führen Sie den Befehl im Trace Modus mit T aus.
   Wirkung: Der Inhalt des Registers DL (=FF) wird mit der Maske 0F UND-verknüpft.
   Damit werden die oberen 4 Bit „ausgeblendet“, d.h. auf „0“ gesetzt.
   Probieren Sie einige andere Masken, z. B. 88.

5) Einer der wichtigsten Maschinenbefehle ist der Befehl MOV.
   Mit seiner Hilfe lassen sich Registern Werte zuweisen.
   Geben Sie bei Adresse 100 ein: MOV AH,44.
   Führen Sie den Befehl mit T aus.
   Schauen Sie sich das Register AH (bzw. AX) an.
   Setzen Sie IP wieder auf Adresse 100.
   Wiederholen Sie das ganze mit AA statt 44. Welche Flags ändern sich?

6) Geben Sie ab Adresse 100 byteweise ein (E 100): B2 73 80 E2 F0.
   Sie haben ein kleines Maschinenprogramm, bestehend aus 2 Befehlen (welchen?), geschrieben.

Führen Sie es aus, indem Sie zweimal T eingeben.
Schauen Sie sich nach jedem Schritt das DL-Register an.
Wo steht der IP nach Ende ihres Miniprogramms? Warum?

7) `MOV CL,F1` (DEBUG Assembler-Syntax)
`MOV BH,12`
`ADD BH,CL`

Welchen Inhalt hat BH nach Ausführung des Befehls: `ADD BH,CL`?

a) F3 b) 13 c) 3 d) DE e) 0 f) alle falsch

(alle Werte hexadezimal)

8) `MOV AH,AA` (DEBUG Assembler-Syntax)
`MOV AL,BB`
`SUB AH,AL`

Welchen Inhalt hat AX nach Ausführung des Befehls: `SUB AH,AL`?

AX: a) BBEF b) BB11 c) 11BB d) EFBB e) 11AA f) AA11 g) alle falsch

(alle Werte hexadezimal)

9) `MOV DX,AAAA` (DEBUG Assembler-Syntax)
`MOV CX,1`
`ADD CX,DX`

Welchen Inhalt haben CX und DX nach Ausführung des Befehls: `ADD CX,DX`?

	a)	b)	c)	d)	e)	
CX:	AAAA	AAAB	AAAA	AAA9	keine Lösung richtig	
DX:	AAAB	AAAA	AAA9	AAAA		

(alle Werte hexadezimal)

10) `MOV AH,EE` (DEBUG Assembler-Syntax)
`MOV DL,FE`
`SUB AH,DL`

Welchen Inhalt hat AH nach Ausführung des letzten Befehls ?

a) EE b) FE c) 10 d) F5 e) F0 f) alle falsch

(alle Werte hexadezimal)

11) Wie sehen Carry Flag und DX nach Ausführung der folgenden Befehlsfolge aus?

```
STC              setze Carry Flag
MOV DX,F8CE
RCL DX,1         Rotate Carry Left
RCL DX,1
RCL DX,1
RCL DX,1
```

	a)	b)	c)	d)	e)	f)
DX:	8CEFh	8CEFh	C677h	8CE7h	8CE7h	alle
CARRY:	1	0	1	1	0	falsch

12) Inhalt von DX = 03FFh (h = hexadezimal)
AX = AEC4h
CX = 1000h

Welchen Inhalt haben AX und DX nach Ausführung des Befehls: DIV CX?

	a)	b)	c)	d)	e)	f)
AX:	000Ah	3FFAh	03FFh	0EC4h	003Fh	alle
DX:	0EC4h	0EC4h	AEC4h	3FFAh	FAECh	falsch

13) Mit dem E-Kommando des DEBUG verändern Sie den Inhalt der Adresse FFFF:FFFF (Segment:Offset) bei einem 8086-INTEL-Prozessor.

Welche der angegebenen Adressen wird tatsächlich angesprochen? (mit dem E-Kommando ausprobieren! Sie können durchaus volle Segment:Offset-Adressen ansprechen: geben Sie mit E FFFF:FFFF einen prägnanten Wert ein, z. B. „AB“ und schauen Sie dann unter den angegebenen Adressen nach, wo Sie Ihren Wert wieder finden).

a)	b)	c)	d)	e) ???	f)
0000:FFFF	0000:0000	FFFF:0000	0000:FFDF	0000:FFEF	FEFF:0011

14) Inhalt von AX = 17h (h = hexadezimal)
BX = FFF3h

Welchen Inhalt hat AX nach Ausführung des Befehls: ADD AX,BX?

a) 1010h b) FFDCh c) 0h d) 10h e) Ah f) alle falsch

15) Gegeben sei folgende Ausgangssituation:
Carry Flag = 1
DX = 1234h

Wie sehen Carry Flag und DX nach zweifacher Ausführung des Befehls RCL DX,1 aus?

	a)	b)	c)	d)	e)
DX:	48D0h	2469h	2469h	48D2h	91A4h
CARRY:	0	1	0	0	0

16) Unabhängig vom aktuellen Wert des DL-Registers soll das LSB (niederwertiges Bit) und das MSB (höchstwertiges Bit) ausgeblendet, d. h. auf Null gesetzt, werden. Die übrigen Bits sollen unverändert bleiben. Welche der folgenden Befehle in der DEBUG Assembler-Syntax löst diese Aufgabe korrekt?

a)	b)	c)	d)	e)	f)
AND DL,70	OR DL,7E	XOR DL,7E	AND DL,7E	MOV DL,7E	keiner

3.5 Adressierungsarten

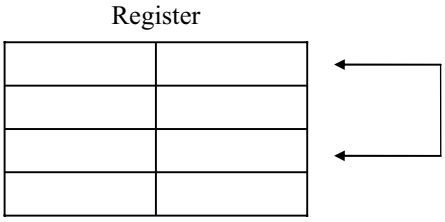
Bei den Befehlsarten wurde gezeigt, *was* die einzelnen Maschinenbefehle leisten. Bei den *Adressierungsarten* geht es um das „*wie komme ich an die Daten heran*“.

Die Adressierungsarten sind Hardwareeigenschaften des Prozessors. Jeder Prozessor, selbst der allereinfachste, kennt verschiedene Adressierungsarten. Je komplizierter der Prozessor, desto

mehr weist er auf. Die INTEL 80(X)86-Familie weist sechs Adressierungsarten auf, die nun vorgestellt werden. Bei Zwei-Operand-Befehlen ist ein Operand in der Regel ein Register. Der zweite bestimmt die Adressierungsart. Bei Ein-Operand-Befehlen bestimmt der Operand die Adressierungsart. Befehle ohne Operanden, wie `STC` oder `HLT`, kennen keine Adressierungsarten.

3.5.1 Registeradressierung

Alle Operanden stehen in Registern. Der Maschinencode ist kompakt und besonders schnell ausführbar.



■ **Beispiele**

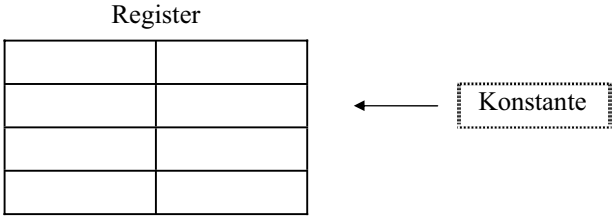
Zwei Operanden	`ADD CX,DX`	Wortoperation
	`MOV BX,AX`	
	`SUB AL,CH`	Byteoperation
Ein Operand	`INC CX`	Wortoperation
	`NEG BH`	Byteoperation
	`MUL BX`	Variante: inhärente Adressierung

nicht erlaubt (wegen Datentypenmix 8/16 Bit) sind:

```
AND AL,BX
MOV CX,DH
```
■

3.5.2 Unmittelbare Adressierung

Ein Operand ist eine Zahl oder genauer gesagt eine Konstante. Hier sind nur Zwei-Operand-Befehle möglich, wobei der erste (Zieloperand) ein Register ist.



■ **Beispiele**

`ADD BL,A4`	Byteoperation
`MOV AH,34`	
`MOV AX,DDDD`	Wortoperation
`SUB DX,12AD`	

Man beachte, dass Byteoperationen häufig etwas mit ASCII-Zeichen zu tun haben. So kann die '34' im zweiten Beispiel die Hexadezimalzahl 34, aber auch das ASCII-Zeichen '4' bedeuten. Erst die Verwendung in einem Programm gibt Aufschluss über die aktuelle Bedeutung. Codes beruhen eben auf Vereinbarung!

Selbstverständlich kann eine Konstante nur als Quelloperand, niemals aber als Ziel, auftreten.

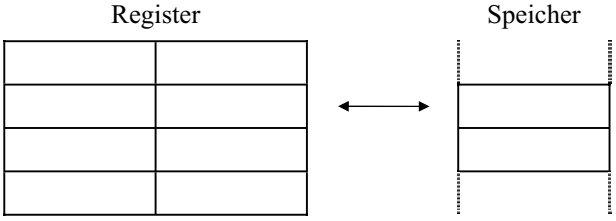
Nicht erlaubt sind also Operationen wie die folgenden:

```
OR  00FF,AX
MOV 1000,CL
```

■

3.5.3 Direkte Adressierung

Ein Operand ist eine Speicheradresse. Diese wird zur Unterscheidung von einer Konstanten (s. unmittelbare Adressierung) in eckige Klammern [] gesetzt (Syntax des DEBUG- Assemblers).



■ **Beispiele**

`MOV AX,[1000]` Wortoperation: die Inhalte der Speicherplätze `1000` und `1001` werden in das `AX`-Register transportiert. Nach der INTEL-Konvention beinhaltet `1001` das High Byte.

`ADD [AF00],DX` Der Inhalt des Registers `DX` wird zu den Inhalten der Speicher `AF00`, `AF01` addiert.

`SUB [2000],BL` Byteoperation: der Inhalt des Halbregisters `BL` wird vom Inhalt der Adresse `2000` subtrahiert.

`AND BL,[1000]` Der Inhalt des Registers `BL` wird mit dem des Speichers `1000` UND-verknüpft. ■

Wie Sie bereits wissen, sind Speicher-zu-Speicher-Operationen leider nicht möglich, so dass folgender Befehl **unzulässig** ist:

```
MOV [1000],[2000]
```

3.5.4 Indirekte Adressierung

Auch hier ist einer der Operanden ein Speicherplatz, allerdings wird dieser indirekt über bestimmte Register angesprochen. Dies geschieht in der Form

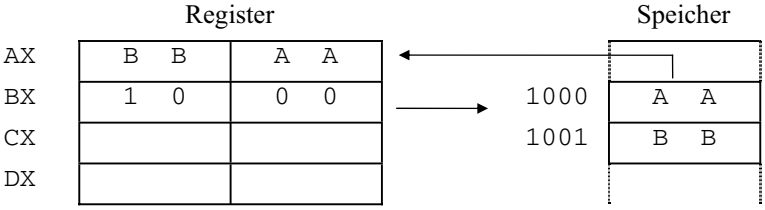
Opcode Register,[Indexregister] bzw. Opcode [Indexregister],Register

Indexregister im engeren Sinne sind `SI` und `DI`. Insgesamt dürfen jedoch folgende Register verwendet werden:

`SI`	Indexregister
`DI`	Indexregister
`BP`	Basepointer
`BX`	Basisregister

z. B. `MOV AX,[BX].`

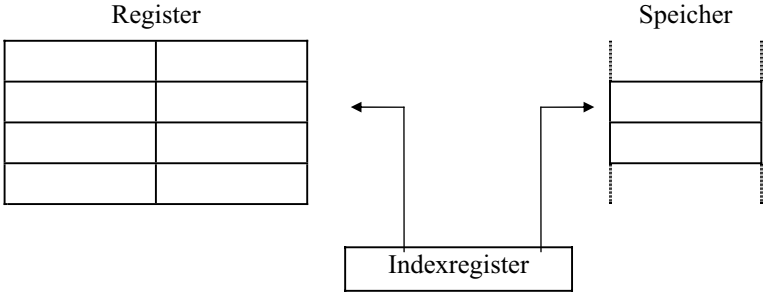
Zur Unterscheidung von der Registeradressierung wird das Indexregister in eckige Klammer [] gesetzt. Im obigen Beispiel wird nicht etwa der Inhalt von BX nach `AX` zugewiesen, sondern vielmehr der Inhalt des Speicherplatzes, auf den `BX` *zeigt*. Möge `BX` den Wert `1000` enthalten, der Speicher `1000` den Wert `AA` und Speicher `1001` den Wert `BB`. Bezogen auf dieses Beispiel haben wir nach Ausführung des Befehls `MOV AX,[BX]` folgende Situation:



In Worten: Register `BX` zeigt auf die Adresse `1000`. Da es sich um eine Wortoperation handelt, wird der Inhalt der Adressen `1000`, `1001` dem `AX`-Register zugewiesen. Natürlich muss in der Praxis dem Indexregister zuvor die Adresse des Speicherplatzes zugewiesen worden sein, in der Regel mit `MOV`, z. B. direkt `MOV BX,1000`.

Ein Indexregister *zeigt auf* einen Speicherplatz, wenn es dessen Adresse enthält.

Allgemein:



Der Pfeil zeigt in beide Richtungen. Dies soll andeuten, dass z. B. sowohl

`MOV AX,[BX]` als auch
`MOV [BX],AX`

möglich ist.

Achtung: denken Sie daran, dass nur die folgenden Register als Indexregister verwendet werden dürfen:

`BX`	`BP`	`SI`	`DI`

■ **Beispiele**

`MOV AX,[SI]` Der Inhalt des Speichers, auf den `SI` zeigt, wird `AX` zugewiesen.

`ADD DX,[BX]` Der Inhalt der Adresse, auf die `BX` zeigt, wird zu `DX` addiert.

`AND [DI],CL` UND-Verknüpfung zwischen `CL` und der Adresse, auf die `DI` zeigt. Hier liegt eine Byte-Operation vor, weil `CL` ein Halbregister ist.

Nicht erlaubt sind:

`MOV [DX],AX` `DX` ist kein Indexregister.

`ADD [200],[BX]` Hier liegt eine verdeckte Speicher-zu-Speicher-Operation vor, die ebenfalls verboten ist. ■

Auch registerindirekte Sprünge sind möglich. Hier sind *alle* Register zulässig. Achtung: das Segment ist dabei *nicht* der Inhalt des `DS`-Registers, sondern der des `CS`-Registers! Dies ist aber nur bei solchen Programmen von Belang, bei denen sich Code- und Datensegment unterscheiden, was jedoch bei unseren kleinen DEBUG Assembler-Programmen nicht der Fall ist.

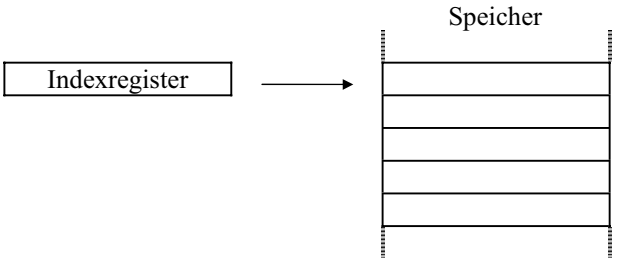
■ **Beispiele**

`JMP BX` Springe zur Speicheradresse im Code-Segment, die im `BX`-Register steht.

`CALL DI` Unterprogrammverzweigung zu der Offset-Adresse im Code-Segment, die im `DI`-Register steht ■

Im Zusammenhang mit Sprüngen werden die Register nicht in eckige Klammern gesetzt.

Bleibt noch die Frage nach dem Sinn der indirekten Adressierung zu beantworten. Sie erleichtert die Bearbeitung von Tabellen. Zunächst lässt man das Indexregister auf den Kopf der Tabelle (im Speicher) zeigen. Man führt dann die gewünschte Operation aus (z. B. Ausgabe eines ASCII-Zeichens auf den Bildschirm), erhöht anschließend den Inhalt des Indexregisters um 1 und wiederholt in einer Schleife den Vorgang bis zum Erreichen einer Abbruchbedingung.



In Assemblercode könnte das beispielsweise so aussehen:

```
XXXX:0100 MOV BX,1000    Adresse des Tabellenkopfs
XXXX:0103 MOV CX,A       10 ins Zählregister CX laden
XXXX:0106 MOV DL,[BX]    erstes ASCII-Zeichen nach DL
XXXX:0108 CALL 200       Unterprogramm zur Zeichenausgabe rufen
XXXX:010B INC BX         BX auf nächsten Speicher zeigen lassen
XXXX:010C LOOP 106       Schleife (10 mal ausführen)
XXXX:010E MOV AH,4C      Function Code für
XXXX:0110 INT 21         Programm beenden
```

Dieses kleine Programm, das an der Adresse `XXXX:0100` beginnt, gibt 10 ASCII-Zeichen, die im Speicher ab Adresse `XXXX:1000` hintereinander stehen, auf den Bildschirm. Dazu dient ein fiktives Unterprogramm bei Adresse `XXXX:0200`, das mit `CALL 200` aufgerufen wird. Da dieses Unterprogramm nicht existiert, kann das Beispiel nicht praktisch erprobt werden. ■

Die indirekte Adressierung wird in Hochsprachen wie C/C++ für die Feldverarbeitung genutzt. Sie entspricht eindimensionalen Feldern (*Arrays*).

Die beiden verbleibenden Adressierungsarten sind nur Erweiterungen der indirekten Adressierung.

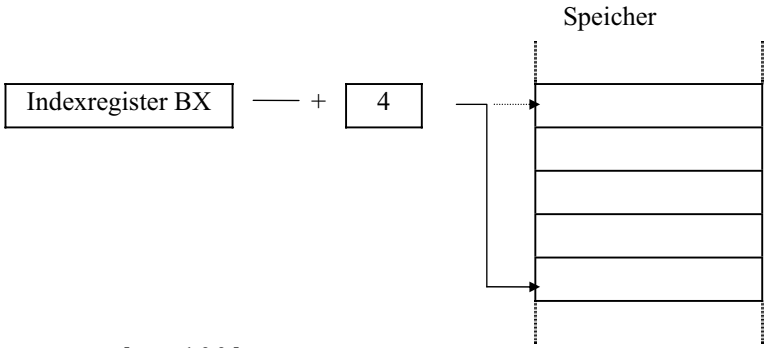
3.5.5 Basisregister plus Displacement

Angesprochen wird die Adresse, auf die das Indexregister zeigt plus einer konstanten Verschiebung (*displacement*). Die Feldverarbeitung wird damit noch komfortabler, wenn man das konstante Displacement auf den Feldanfang zeigen lässt. In diesem Fall kann man mit dem Anfangsindex 0 arbeiten, wie man es von den Hochsprachen her gewohnt ist.

■ **Beispiele**

```
MOV CX,[BX+4]
      oder
MOV CX,[4+BX]
```

Der Inhalt der Adresse im Datensegment, auf die `BX` zeigt + 4 (sowie + 5, da Wortoperation), wird zugewiesen.




```
ADD AX,[SI+100]
```

Der Inhalt der Adresse, auf die `SI` zeigt + `100`h (sowie + `101`h), wird zu `AX` hinzuaddiert.

```
MOV [DI+6],BL
```

Der Inhalt von `BL` wird der Speicheradresse `[DI] + 6` zugewiesen (Byteoperation).

```
MOV [6+DI],BL
MOV [3+DI+3],BL
```

funktioniert ebenfalls, genau wie: ■

3.5.6 Basisregister plus Indexregister plus Displacement

Angesprochen wird die Adresse, auf die das erste Indexregister zeigt plus dem Inhalt des zweiten Indexregister plus einer konstanten Verschiebung (*displacement*)

Anwendung: Abarbeitung von Matrizen (in Hochsprachen zweidimensionale Felder).

■ **Beispiele**

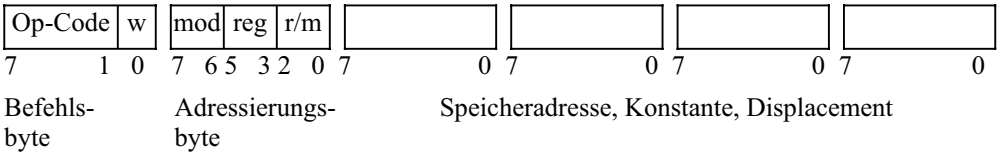
`MOV CL,[BP+DI+100]` Der Inhalt der Adresse im Datensegment, auf die `BP` zeigt + dem Inhalt von `DI` + `100`h, wird `CL` zugewiesen.

`MOV [BX][SI],AX` Auch diese Schreibweise ist erlaubt. Angesprochen wird der Inhalt von `BX` plus dem von `SI`. Man kann sich `BX` als Zeilen- und `SI` als Spaltendimension vorstellen.

■

3.5.7 Detaillierter Aufbau eines Maschinencodes

Natürlich muss auch die Adressierungsart in den Maschinencode einfließen. Wir schauen uns zum Schluss dieses Kapitels den Detailaufbau eines 80(X)86-Maschinenbefehls an.



Es bedeutet:

* `Op-Code` bestimmt den Befehl (was ist zu tun)
* `w-Bit` Befehlsweite: `w = 1` → 16-Bit-Operation `w = 0` → 8-Bit-Operation
* `mod` Art des Displacements (→ Kap. 3.5.6)
* `reg` bestimmt den Registeroperanden (zus. mit w-Bit)
* `r/m` welche Register werden zur Adressbildung des anderen Operanden benutzt?

Wie man sieht, kann ein Befehl maximal 6 Byte lang sein, minimal ist er 1 Byte lang, denn nur das Befehlsbyte ist in jedem Fall notwendig.

3.5.8 Übungen

1) Gegeben sei folgender Speicherauszug (alle Werte hexadezimal):

Adresse	Inhalt
100	4400
200	3300
3300	200
4400	100

Wie verändert die Befehlsfolge (Syntax des DEBUG Assemblers)

```
MOV DX,200
MOV BX,[4400]
MOV [BX],DX
ADD BX,100
MOV [3300],BX
```

`[4400]` bedeutet: Inhalt von Adresse `4400`

`[BX]` bedeutet: Inhalt der Adresse, auf die Register BX zeigt

die Speicherinhalte?

Adressen	a)	b)	c)	d)	e)	f)
	veränderte Inhalte					
100h	200h	3300h	4400h	200h	200h	alle
200h	200h	3300h	3300h	200h	3300h	falsch
3300h	200h	3400h	200h	200h	200h	
4400h	200h	100h	200h	100h	100h	

2) Welcher der folgenden Befehle führt zu einer Fehlermeldung des DEBUG Assemblers?

a) `SUB DH,127`

b) `MOV [BX],CX`

c) `ADD BX,[BX]`

d) `LOOP 1000`

e) `ADD CX,CX`

f) `INT 21`

3) Welcher der folgenden Assemblerbefehle ist falsch bzw. nicht erlaubt?

a) `MOV BX,AX` b) `MOV CX,[DI]` c) `MOV [BX],2000`

d) `MOV DX,EF89` e) `MOV DL,[BX]` f) `JMP BX`

4) Gegeben sei folgender Speicherauszug (alle Werte hexadezimal):

Adresse	Inhalt
1000	2000
2000	1000
3000	4000
4000	3000

Wie verändert die Befehlsfolge (Syntax des DEBUG Assemblers)

```
MOV AX,1000
MOV BX,[3000]
MOV [BX],AX
```

`[3000]` bedeutet: Inhalt von Adresse 3000

`[BX]` bedeutet: Inhalt der Adresse, auf die Register BX zeigt

die Speicherinhalte (alle Werte hexadezimal)?

Adressen	a)	b)	c)	d)	e)
	veränderte Inhalte				
1000	1000	2000	2000	2000	2000
2000	2000	1000	1000	1000	1000
3000	1000	4000	1000	4000	4000
4000	2000	3000	3000	1000	2000

5) Auf welchem Wert steht bei dem Befehl

```
ADD [BX],DL
```

das w-Bit im Befehlsbyte?

a) 0 b) 1 c) beeinflusst das w-Bit nicht

6) Schreiben Sie mit Hilfe des DEBUG Assemblers ein Programm, das die Speicherplätze ab `1000` mit den Werten `0` bis `FE`(hex) belegt. Kontrollieren Sie anschließend mit dem E-Kommando des DEBUG den Erfolg.

7) Schreiben Sie mit Hilfe des DEBUG Assemblers ein Programm, das die Speicherplätze ab `1000` mit den Werten `0` bis `1000`(hexl) belegt. Kontrollieren Sie anschließend mit dem E-Kommando des DEBUG den Erfolg.

Hinweis zu Aufg. 6 und 7: Beenden Sie die Programme mit dem Befehl „INT 20“ und starten Sie die Programme mit dem G-Kommando.

4 Schnittstellen zum Betriebssystem

Die effektive Programmierung eines modernen Mikrocomputers ist ohne die Unterstützung durch ein leistungsfähiges Betriebssystem (BS) nicht denkbar (Ausnahme: low cost Mikrocontroller, doch selbst die haben meist ein Mini-BS, *Monitor* genannt). Beispielsweise verwaltet das BS den Zentralspeicher, u. a. sorgt es dafür, dass ein ausführbares Programm korrekt geladen wird. Außerdem versieht es die ausführbaren Programm häufig mit einem Programmkopf (→ s. Kap 4.2).

Vor allem aber bieten moderne BS den Zugang zu zahllosen Standarddiensten wie Tastatureingabe, Bildschirmausgabe, Plattenzugriffe, Beeinflussung des Videomodus. Der Zugriff auf derartige Dienste erfolgt über *Systemaufrufe*. In der 16-Bit-Welt des 8086, auf modernen Prozessoren wie dem Pentium emuliert durch den Virtual Mode, gelten noch die Regeln des Single Tasking BS DOS, das uns hier als Lehrbeispiel für ein einfaches Mikrocomputer-BS dienen soll. Unter DOS heißen die Systemaufrufe *Softwareinterrupts*, weil sie, wie wir später sehen werden, formal wie Interrupts funktionieren. Man unterscheidet zwischen DOS- und BIOS-Interrupts. Diese Softwareinterrupts sind für den Systemprogrammierer von ungeheurer Bedeutung, denn ohne sie wäre eine simple Bildschirmausgabe ein ernsthaftes Problem. Er müsste sich sehr intensiv mit der Hardware auseinandersetzen. Wahrscheinlich ist Ihnen aufgefallen, das wir das Ein-/Ausgabeproblem bei unserer Assembler-Einführung bisher umgangen haben.

Anmerkung zu den nachfolgenden Kapiteln:

Um unsere Assemblerprogramme auf unserem PC zu starten, benötigen wir ein möglichst einfaches BS, das nicht durch seine Komplexität vom Wesentlichen ablenkt. Hier bietet sich das bei allen Windows-Versionen enthaltene DOS an, das wir im Virtual Mode in einem Fenster starten können. Obwohl die Ära von DOS als Standardbetriebssystem natürlich vorüber ist, ist es für unsere Zwecke sehr gut geeignet, da alle prinzipiellen Vorgänge bei den Systemaufrufen und Hardware-Zugriffen noch gut verfolgt werden können, was bei Windows viel komplizierter wäre!

4.1 BIOS und DOS

Mit BIOS (*Basic Input Output System*) wird eine Gruppe von Programmen bezeichnet, die sich teils im RAM-Speicher, teils auf dem Festwertspeicher (EPROM) des PC befindet. Eines dieser Programme übernimmt die Kontrolle beim Einschalten des PCs (Kaltstart), auch unter Windows. Es beginnt bei der absoluten Real-Mode-Adresse `FFFF0`h. Daneben enthält das BIOS eine Programm-Sammlung zum elementaren Ansprechen der Peripherie, beispielsweise des Bildschirms.

Über das BIOS wird die Kompatibilität aller PC gewährleistet, auch wenn die Hardware unterschiedlicher Hersteller unterschiedlich arbeitet. Greift man über die vorgeschriebenen BIOS-Aufrufe auf die Hardware zu, bleiben die Hardwareunterschiede verdeckt. Direktzugriffe auf die Hardware über I/O-Ports sind zwar oft erheblich schneller, bergen aber die Gefahr der Inkompatibilität, wenn das Programm auf dem Rechner eines anderen Herstellers laufen soll.

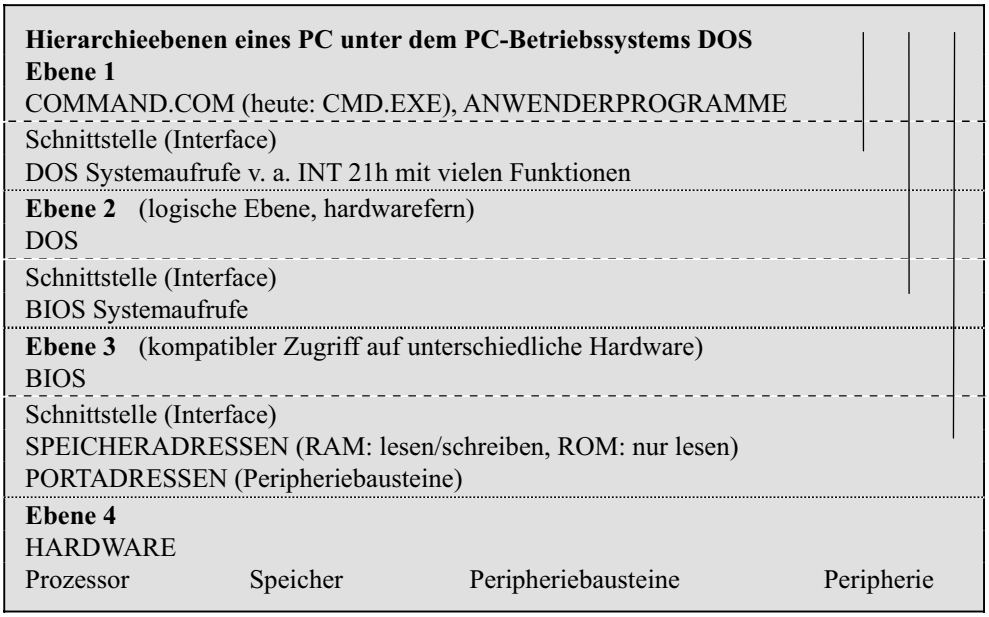
Das Betriebssystem DOS dagegen, welches erst im Laufe des Systemstarts von der Festplatte geladen wird, ist für „höhere“ Aufgaben zuständig. Eine wichtige Teilaufgabe ist die Speicherverwaltung. Auch DOS (*Disk Operating System*) bietet dem Benutzer eine Vielzahl von Programmen zur Vereinfachung der systemnahen Programmierung an. Sie sind jedoch im Gegen-

satz zu den BIOS-Programmen nicht *reentrant*, d. h. sie dürfen nicht vom BS unterbrochen werden.

Die Schnittstellen sowohl zum BIOS als auch zum DOS sind die Systemaufrufe, Softwareinterrupts genannt. Sie lassen sich auf einfache Weise mit dem Maschinenbefehl

```
INT <interrupt-nummer>
```

ausführen. Den dahinter stehenden Mechanismus werden wir später untersuchen. An dieser Stelle interessieren wir uns für den Nutzen, den die Softwareinterrupts uns bieten. Die Interrupt-Nummer bezeichnet die BIOS- oder DOS-Funktion, die gestartet werden soll. Bei der Vielzahl der Funktionen können wir nur einige Beispiele betrachten, es geht uns hier ja ohnehin hauptsächlich ums Prinzip. Die folgende schematische Darstellung zeigt die Hierarchieebenen eines PC unter dem „einfachen“ Betriebsystem DOS.



Als PC-Anwender kennen Sie vielleicht einige Dienstprogramme, die sich von der Konsole (Eingabeaufforderung) heraus starten lassen. Diese Kommandos (`COPY`, `DIR`, `TYPE`, ...) wurden unter DOS von dem Programm `COMMAND.COM` ausgeführt und können auch heute noch recht nützlich sein. Erstens arbeiten sie schneller als die entsprechenden „Klick-Programme“ der grafischen Oberfläche und zweitens können sie in Prozeduren (→ s. Kap. 9.2) eingebaut werden. `COMMAND.COM war` der Befehlsinterpreter des BS DOS. Unter Windows heißt der Kommandointerpreter CMD.EXE. Er kann zwischen 16- und 32-Bit-Programmen unterscheiden und lässt sie entsprechend im Virtual- oder Protected Mode laufen. Was er kann, verrät er Ihnen, wenn sie „CMD /?“ eingeben. Wie man der schematischen Abbildung entnimmt, steht die Benutzeroberfläche im Prinzip auf einer Ebene mit den Anwendungsprogrammen. Von Letzteren aus hat man, wie die senkrechten Linien links in der Abbildung andeuten, Zugriff auf alle Ebenen von DOS, ja sogar auf die Hardware des Rechners direkt. Hält man sich dagegen an die Hierarchie (Zugriff nur auf die nächste Ebene), so erhält man stets kompatible und portable Programme (wenn auch etwas langsamer, so ist das nun mal mit Hierarchien).

Kleinere Mikrocomputersysteme, v. a. Einplatinen-Mikrocontroller-Boards, besitzen in der Regel kein Betriebssystem, sondern lediglich einen *Monitor*. Darunter versteht man eine Sys-

temsoftware, die vollständig auf einem EPROM untergebracht ist. Sie enthält ebenfalls eine RESET-Routine sowie einen kleinen Befehlsinterpreter für einige wenige Primitiv-Kommandos. Systemaufrufe stehen nur selten zur Verfügung. Der Benutzer ist gezwungen, auch elementarste Grundfunktionen selbst zu programmieren. Monitorprogramme haben typischerweise einen Umfang zwischen 1 und 32 KByte.

4.1.1 BIOS-Systemaufrufe

Die folgende Tabelle zeigt beispielhaft einige BIOS-Interrupts:

Assembler-Aufruf	Funktion
`INT 10`	Steuerung des Bildschirms
`INT 14`	Steuerung der seriellen Schnittstelle
`INT 16`	Tastatureingaben
`INT 17`	Steuerung des Druckers

Viele Systemaufrufe (Softwareinterrupts) haben mehrere Unterfunktionen. Beispielsweise kann man mit Hilfe des `INT 10(hex)` den Video-Modus des Bildschirms verändern, aber auch den Cursor positionieren oder ein Zeichen auf den Bildschirm geben. Die (Unter-)Funktionsnummer (*function code*) wird immer im Register `AH` übergeben. Eigentlich ist das keine Neuigkeit für Sie, wenn Sie an den Function Code `4C(hex)` des `INT 21(hex)` aus unserer Assembler-Einführung denken.

Um die konkrete Nutzung der BIOS-Routinen zu erlernen, betrachten wir beispielhaft die Bildschirm- und Tastaturfunktionen näher:

`INT 10(hex)`	**Bildschirmsteuerung (ausgewählte Funktionen)**		
function code (AH)	Beschreibung der Tätigkeit	Übergabe	Rückgabe Parameter
`00`(hex)	Einstellung des Videomodus `AL` = 0: 40 x 25 sw, Text `AL` = 1: 40 x 25 Farbe, Text `AL` = 2: 80 x 25 sw, Text (Standard) `AL` = 3: 80 x 25 Farbe, Text	`AL` = Modus	--
`0E`(hex)	Zeichenausgabe	`BX` = 0 `AL` = Zeichen	--

`INT 16(hex)`	**Tastatureingabe (ausgewählte Funktionen)**		
function code (AH)	Beschreibung der Tätigkeit	Übergabe	Rückgabe Parameter
`00`(hex)	Lies ein Zeichen nach `AL`	--	`AL` = Zeichen
`01`(hex)	Setze Zero Flag = 0, wenn beliebige Taste gedrückt, sonst Zero Flag = 1.	--	Zero Flag

Aus der Tabelle gehen die Function Codes (nicht vollständig!) ebenso hervor wie die sonstigen Übergabeparameter, die ebenfalls in Registern übergeben werden. Unter Umständen liefert die aufgerufene Systemroutine Werte zurück (Rückgabeparameter). Diese Werte stehen in den „Rückgaberegistern“ der Tabelle. Rückgaberegister werden also von der gerufenen Systemroutine geändert, um etwas mitzuteilen. Beachten Sie ggf., dass die Softwareinterrupts oft auch solche Register ändern, in denen sie nichts Wesentliches zurückliefern. So ändert beispielsweise der `INT 10(hex)` immer die Register `AX`, `SI`, `DI` und `BP`.

Das folgende Beispielprogramm zeigt, wie mit Hilfe des BIOS-Video-Interrupts 10(hex) (s. Tabelle) der Bildschirmmodus auf 40 x 25 Zeichen schwarz-weiß (s/w) verändert wird. Der BIOS-Tastatur-Interrupt 16(hex), Function Code 0 wird benutzt, um ein Zeichen von der Tastatur ins AL-Register einzulesen. Mit `INT 10(hex)`, Function Code E wird es auf dem Bildschirm ausgegeben. Nach Eingabe eines weiteren Zeichens, das nicht ausgegeben wird, erfolgt die Umschaltung in den Standard-Textmodus 80 x 25 sw sowie die Rückkehr nach DOS.

```
XXXX:0100 MOV AH,0     Video Modus auf 40 x 25 sw.
XXXX:0102 MOV AL,0
XXXX:0104 INT 10
XXXX:0106 MOV AH,0     Eingabe eines Zeichens von der Tastatur nach AL
XXXX:0108 INT 16
XXXX:010A MOV AH,0E    Ausgabe des Zeichens in AL auf Bildschirm
XXXX:010C MOV BX,0
XXXX:010F INT 10
XXXX:0111 MOV AH,0     Warten auf erneute Zeicheneingabe
XXXX:0113 INT 16
XXXX:0115 MOV AH,0     Video Modus wieder auf 80 x 25 sw
XXXX:0117 MOV AL,2
XXXX:0119 INT 10
XXXX:011B MOV AH,4C    Rückkehr zur Eingabeaufforderung
XXXX:011D INT 21
```

Geben Sie das Programm in der Eingabeaufforderung mit DEBUG ein und probieren Sie es aus.

4.1.2 DOS-Systemaufrufe

Während das BIOS die Kompatibilität der Rechner trotz unterschiedlicher Hardware garantiert, stellt DOS eine Sammlung „höherer“ Funktionen zu Verfügung, die sich nicht selten ihrerseits des BIOS bedienen.

Mit DOS-Funktionen lassen sich praktisch alle Betriebsmittel des Rechners recht komfortabel zugänglich machen. Man realisiert damit u. a.

- Tastatureingaben
- Bildschirmausgaben
- Dateizugriffe
- Speicherverwaltung
- Zeitfunktionen
- Interruptsteuerungen

Während es viele wichtige BIOS-Interrupts gibt, sind die meisten DOS-Aufrufe in einem einzigen Interrupt, dem INT 21(hex), zusammengefasst. Entsprechend gibt es an die 200 Functions. Der Function Code wird stets im AH-Register übergeben.

Die Dokumentation des gesamten INT 21(hex) nimmt sehr viele Seiten in Anspruch. Wir wollen uns an dieser Stelle wiederum nur aufs Prinzip beschränken und beispielhaft einige Funktionen aufführen:

function code (AH)	Beschreibung der Tätigkeit	Übergabe Parameter	Rückgabe Parameter
01(hex)	Zeicheneingabe mit Ausgabe	--	AL = Zeichen
02(hex)	Ausgabe eines Zeichens	DL = Zeichen	--
07(hex)	Zeicheneingabe ohne Echo	--	AL = Zeichen
09(hex)	Ausgabe einer Zeichenkette, die mit „$“ endet	DS = Segment DX = Offset-Adresse der Zeichenkette	--
0A(hex)	Eingabe einer Zeichenkette, die mit <CR> endet Achtung: Das BS schreibt die Anz. der gelesenen Zeichen in das erste Wort des Puffers bei DS:DX	DS = Segment DX = Offset	--
4C(hex)	Programm beenden: Rückkehr zur Eingabeaufforderung	AL = Ende-Code	--

Die Ausgabe eines Zeichens (z. B. ‘*’) auf den Bildschirm beispielsweise funktioniert mit Hilfe des DOS-Interrupts 21(hex) so:

■ Beispiele:

```
MOV AH,02       Function Code
MOV DL,2A       ASCII-Code für '*'
INT 21          DOS-Interrupt
```

Abschließend ein komplettes Programmbeispiel:

```
MOV AH,2
MOV CX,8
MOV DL,0
RCL BL,1
ADC DL,30
INT 21
LOOP 105
INT 20
```

Zwei Befehle sind für Sie neu: `ADC` steht für *Add with Carry*. Im obigen Beispiel wird nicht nur `30h`, sondern auch noch zusätzlich der aktuelle Wert des Carry Flags (entweder 0 oder 1) zu `DL` hinzu addiert. `INT 20` (hex) ist ebenfalls ein DOS-Interrupt, der aber nur eine einzige Funktion besitzt (daher kein Function Code): er beendet ein Programm, ähnlich wie die Funktion `4Ch` des `INT 21` (hex), jedoch erfolgt die Rückkehr nach DEBUG und nicht zum Kommandointerpreter, was für uns manchmal praktischer ist.

Geben Sie das Programm zunächst ein. Schreiben Sie es unter dem Namen „`WRITEBIN.COM`“ auf die Festplatte in Ihr Arbeitsverzeichnis und laden Sie es anschließend mit „`DEBUG WRITEBIN.COM`“. Die Anleitung dazu finden Sie in Kap. 3.2.

Schauen Sie sich das Programm mit `-U 100` an und versuchen Sie, seine Funktion festzustellen (das Programm umfasst die Speicherplätze 100 -111). Schreiben Sie „`1234`“ in das `BX`-Register. Führen Sie das Programm mit dem G-Kommando aus. Schreiben Sie einige andere Hexadezimalwerte in das `BX`-Register und führen Sie das Programm jeweils erneut aus. Spätestens jetzt wird klar, dass das Programm den Inhalt des `BL`-Registers binär ausgibt. ■

Als weiteres Beispiel benutzen wir den Function Code `09h`, um eine Zeichenkette, die mit „`$`“ enden muss, auszugeben. Die Zeichenkette soll aus den Buchstaben „`ABC`“, einem Zeilenvorschub (*line feed*), einem Wagenrücklauf (*carriage return*) und dem „`$`“ bestehen. Die Zeichenkette soll ab Offset-Adresse `200` stehen. Im DEBUG geben Sie zunächst mit dem E-Kommando die Zeichenkette ein, indem Sie die entsprechenden ASCII-Codes eintippen: `E 200` und dann `41 42 43 0A 0D 24`. Das Programm ist denkbar kurz:

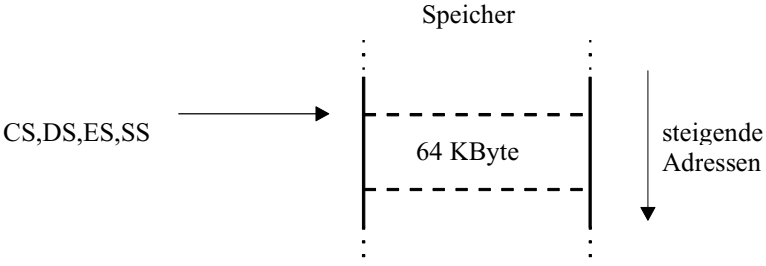
```
MOV AH,9   Ausgabe der Zeichenkette
MOV DX,200      ab Adresse DS:0200
INT 21
INT 20          Rückkehr zu DEBUG
```

4.2 Die Speichermodelle COM und EXE

Der Real Mode kennt zwei verschiedene Typen von ausführbaren Maschinenprogammen:

EXE und COM. Obwohl EXE der Standardtyp ist, werden aus 16-Bit-Assembler-Quellprogrammen meist COM-Module.

Ein COM-Modul ist v. a. dadurch gekennzeichnet, dass alle Segmente deckungsgleich „übereinander“ liegen. Anders ausgedrückt, alle Segmentregister zeigen auf die gleiche Segmentadresse:



Speichermodell COM

Im Prinzip existiert somit nur ein einziges gemeinsames Segment für Programmcode, Daten und Stack. COM-Module können also insgesamt nicht größer sein als 64 KByte. DEBUG ist ganz auf COM-Programme ausgelegt. Bei Assemblerprogrammen benötigt man selten das

größere EXE-Modell, bei dem bis zu vier volle Segmente, also insgesamt 256 KByte, gleichzeitig zu Verfügung stehen. Im Allgemeinen machen nur Hochsprachenprogramme von diesem Angebot Gebrauch. Im Protected Mode gibt es nur 32-Bit-EXE-Programme ohne die obige Speicherbeschränkung.

Beim Start eines 16-Bit-Programms wird dieses vom Betriebssystem mit einem „Kopf“, dem so genannten *Program Segment Prefix* (PSP) bei COM-Programmen bzw. dem Dateikopf bei EXE-Programmen, versehen. Beide Vorspannblöcke enthalten Informationen, die man eventuell zur systemnahen Programmierung benötigt.

Der PSP der COM-Programme beginnt immer bei der Offset-Adresse `0` und reicht bis `FF`. Aus diesem Grund beginnt ein COM-Programm bei der Adresse `100` (hex). Beispielsweise enthält die Adresse `0` und `1` des PSP den Befehl, der eine ordnungsgemäße Rückkehr zu DOS garantiert, wenn man das Programm mit einem Sprung auf die (Offset-)Adresse `0` abschließt.

Interessanter ist folgende Eigenschaft: Ab der Adresse `81` (hex) stehen die Kommandozeilenparameter, die eine professionelle Kommandostruktur eines Programms ermöglichen. Die meisten DOS-Kommandos selbst nutzen diesen Mechanismus.

Beispiel: `COPY DATEI1 DATEI2`

„`DATEI1 DATEI2`“ ist der Kommandozeilenstring, auf den das Programm „`COPY`“ zugreifen kann. Das folgende Beispielprogramm zeigt, wie man an die Übergabeparameter herankommt. Dabei ist es wichtig zu wissen, dass in der Adresse `80` (hex) (1 Byte!) die Länge des Übergabestrings steht.

```
XXXX:0100 MOV CH,0          Anzahl der Zeichen nach CX laden
XXXX:0102 MOV CL,[80]       Byteoperation, nicht MOV CX,[80]!
XXXX:0106 MOV BX,81         BX auf Adresse 81h zeigen lassen
XXXX:0109 MOV DL,[BX]       Zeichen nach DL laden
XXXX:010B MOV AH,2          Ausgabe eines Zeichens
XXXX:010D INT 21            mit DOS-Interrupt
XXXX:010F INC BX            Indexregister erhöhen
XXXX:0110 LOOP 109          nächstes Zeichen, solange CX nicht 0
XXXX:0112 INT 20            Programmabschluss
```

Geben Sie das Programm mit DEBUG ein, führen Sie es aber nicht aus, sondern schreiben Sie es zunächst auf Festplatte, z. B.:

- `N PARAM.COM`

dann `BX` auf `0`, `CX` auf `20` setzen (das Programm ist kürzer als 20 Byte)

- `W`

Anschließend starten Sie das Programm von der Kommando-Ebene aus, z. B. so:

```
PARAM Das ist ein Parameterstring
```

Wenn das Programm korrekt arbeitet, müsste es

„`Das ist ein Parameterstring`“

ausgeben.

Natürlich müsste ein „ernsthaftes“ Programm den String nun in seine einzelnen Bestandteile (Worte) zerlegen und deren Bedeutung (z. B. Dateinamen) beachten. Wir geben uns auch hier mit dem Prinzip zufrieden.

4.3 Aufgaben

Alle folgenden Übungen erfolgen über die konsole (Eingabeaufforderung) mit Hilfe des Dienstprogramms DEBUG (→ s. Kap 3.2).

1) Geben Sie mit Hilfe des A-Kommandos folgendes kleine Assemblerprogramm ein:

```
MOV AH,02
MOV DL,2A
INT 21
INT 20
```

Führen Sie das Programm mit dem G-Kommando aus.
Geben Sie dem Programm den Namen `WRTSTAR.COM`.
Schreiben Sie das Programm auf Ihre Festplatte.
Verlassen Sie DEBUG.
Rufen Sie DEBUG so auf, dass `WRTSTAR.COM` geladen wird (mit U-Kommando nachprüfen).

2) Schreiben Sie ein Programm, das zunächst ein Zeichen von der Tastatur einliest und es anschließend auf dem Bildschirm ausgibt:

```
MOV AH,1
INT 21
MOV DL,AL
MOV AH,2
INT 21
INT 20
```

Führen Sie das Programm mit dem G-Kommando aus. Es erwartet die Eingabe eines Zeichens von Tastatur.

Ändern Sie nun den ersten Befehl `MOV AH,1` in `MOV AH,7`. Wo liegt der Unterschied?

3) Das folgende Programm soll 10 '*' auf den Bildschirm drucken.

Welchen Wert muss das Argument (?) von LOOP haben, damit das Programm korrekt und möglichst schnell ausgeführt werden kann?

Adresse	**Befehl in DEBUG Assembler-Syntax**
100	MOV CX,A
103	MOV AH,2
105	MOV DL,2A
107	INT 21
109	LOOP ?
10b	INT 20

	a)	b)	c)	d)	e)	f)
? =:	100	103	105	107	109	10b

4) Schreiben Sie ein Programm, das in einer „Dauerschleife“ abprüft, ob eine beliebige Taste gedrückt ist. Wird tatsächlich eine Taste gedrückt, soll ein akustisches Zeichen (ASCII-Code 07 = BELL) ausgegeben und das Programm beendet werden.

   Tip: Verwenden Sie den BIOS Interrupt 16 (s. Kap. 4.1.1).

5) Ändern Sie das Programm `WRITEBIN.COM` so ab, dass es das komplette Wort im `BX`-Register binär ausgibt.

6) Schreiben Sie ein Programm, das von der Tastatur eine Zeichenkette einliest, z. B. in die Speicherplätze `200` folgende. Geben Sie diese Zeichenkette anschließend aus. Orientieren Sie sich dabei an unserem Beispielprogramm `PARAM.COM`.

5 Unterprogramme und Programmunterbrechungen

Wie von höheren Programmiersprachen bekannt, lassen sich mehrfach wiederkehrende oder in sich abgeschlossene Teilprobleme am besten durch Anwendung der Unterprogrammtechnik lösen. Vor allem in der Systemprogrammierung treten neben den herkömmlichen Prozeduren häufig Interrupt-Service-Routinen auf, die auf Programmunterbrechungen, asynchron zum „normalen“ Programmablauf, reagieren. Um das Prinzip dieser Mechanismen kennen zu lernen, bleiben wir in der 16-Bit-Welt.

5.1 Call-Unterprogramme

Mit dem `CALL`-Befehl wird das laufende Programm (z. B. das Hauptprogramm) verlassen. Die Syntax lautet:

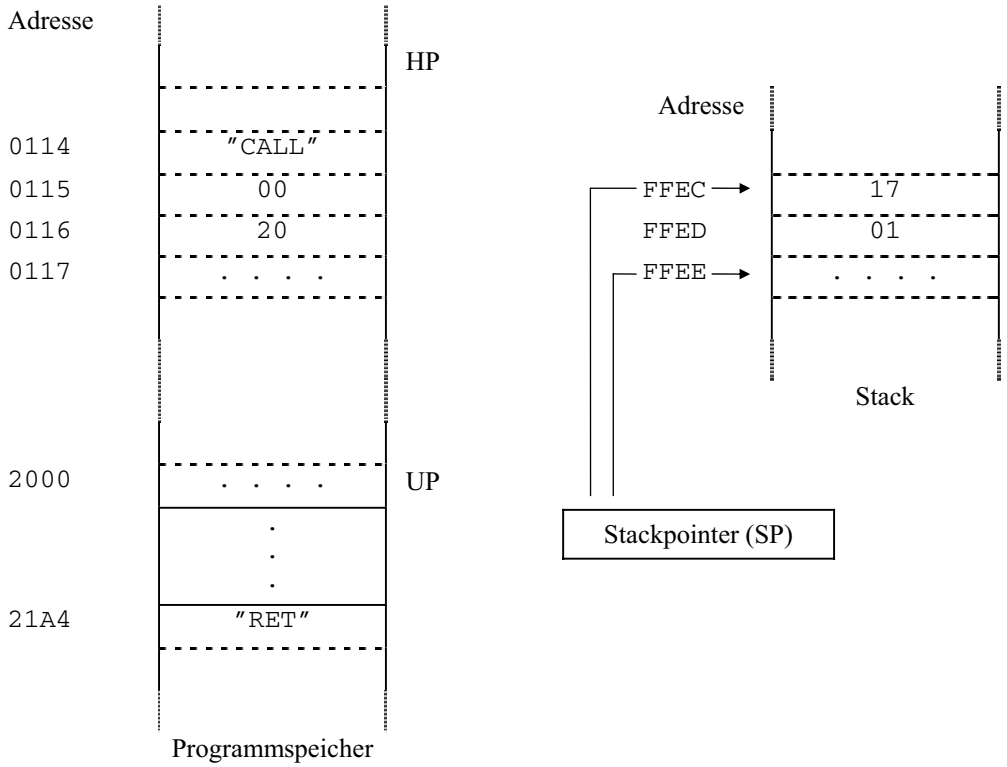
```
CALL <adresse>
```

Anschließend wird das Unterprogramm komplett ausgeführt. Es endet stets mit dem Befehl

```
RET
```

Dieser bewirkt einen Rücksprung auf den Befehl im rufenden Programm, der dem `CALL`-Befehl folgt.

Die Rücksprungadresse wird auf dem Stack abgelegt, wie in der folgenden Abbildung zu sehen ist. Von dort holt sie sich später der `RET`-Befehl. Befindet sich das Unterprogramm im gleichen Segment, handelt es sich um eine Intrasegment-Prozedur (Typ `NEAR`) und es wird nur die Offsetadresse auf dem Stack gespeichert. Bei Intersegmentprozeduren (Typ `FAR`) wird zusätzlich die Segmentadresse nach der Offsetadresse gespeichert. Wir betrachten hier lediglich den ersten Fall, so wie er in COM-Programmen vorkommt. Wir machen uns diesen Mechanismus an einem Beispiel klar.



Erläuterungen:

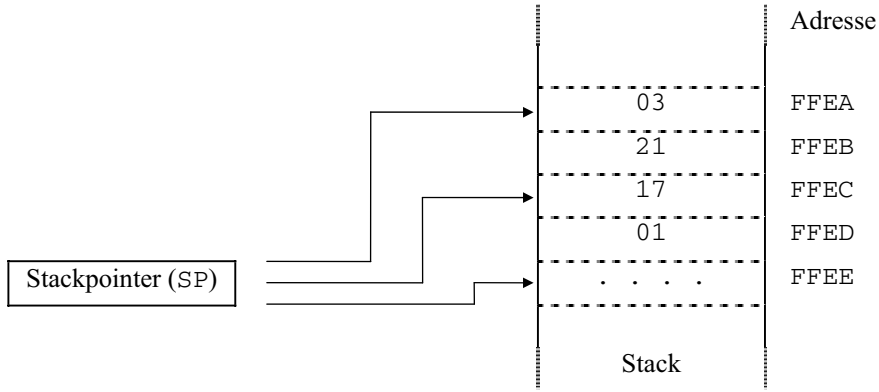
Bei Offest-Adresse `0114(hex)` im Hauptprogramm (HP) beginnt der 3-Byte-Befehl `CALL 2000`. Dieser bewirkt folgendes:

1. Die Rücksprungadresse `0117(hex)` wird auf den Stack geschrieben. Nach der INTEL-Konvention gelangt das High Byte auf die höhere Adresse. Der Stackpointer, ein spezielles Register, zeigt immer auf den letzten Stackeintrag. Vor dem `CALL`-Befehl war das `FFEE`(hex), danach `FFEC(hex)`, weil zwei Bytes auf den Stack gelangten.
2. Es erfolgt der Sprung zum Unterprogramm (UP), das bei Adresse `2000(hex)` beginnt.

Das UP endet mit dem `RET`-Befehl. Dieser bewirkt folgendes:

1. Die Rücksprungadresse (`0117(hex)`) wird vom Stack geholt. Danach zeigt der Stackpointer wieder auf die Adresse `FFEE`h.
2. Es erfolgt der Rücksprung ins HP, das bei Adresse `0117(hex)` fortgesetzt wird. Der Stackpointer zeigt wieder auf die Ausgangsadresse.

Natürlich sind – Sie kennen das von C/C++ – Unterprogramm-Schachtelungen möglich: Das HP ruft UP1 auf, UP1 ruft UP2, usw. Die Rückkehr erfolgt in umgekehrter Reihenfolge: von UPn nach UPn-1 bis zum HP. Erweitern wir einfach das obige Beispiel, indem wir annehmen, dass unser UP bei Adresse `2100(hex)` ein weiteres UP, beginnend bei Adresse `3000(hex)` ruft. Der Befehl dazu lautet `CALL 3000`. Dieser Befehl ist, wie Sie wissen, drei Byte lang. Folglich lautet die Rücksprungadresse `2103(hex)`. Was geschieht mit dem Stack?



Erläuterungen:

Zu Beginn zeigt der Stackpointer auf die Adresse FFEEh, nach CALL 2000 (hex) zeigt er auf die Adresse FFEC (hex), nach CALL 3000 (hex) zeigt er auf die Adresse FFEA (hex), nach dem RET in UP2 (das bei Adresse 3000 (hex) beginnt) zeigt er wieder auf Adresse FFECh, nach dem RET in UP1 (das bei Adresse 2000 (hex) beginnt) zeigt er wieder auf die Ausgangsadresse FFEE (hex).

Aus dem Beispiel erkennen wir:

1. Der Stack ist ein LIFO-Speicher (*Last In First Out*). Der mit dem letzten CALL getätigte Eintrag wird mit dem ersten RET entfernt. Dies ermöglicht auf einfache Weise Unterprogramm-Schachtelungen.
2. Der Stack wächst nach unten, d. h. zu niederen Adressen hin. Der Grund liegt im COM-Modell: dort gibt es kein separates Stacksegment. Das Betriebssystem setzt beim Programmstart den Stackpointer auf eine relativ hohe Adresse, z. B. FFEE (hex). Der Programm- und Datenbereich beginnt bekanntlich bei 0100 (hex) und wächst nach oben. Damit wird ein *Crash* mit dem Stack bis zum letzten Moment hinausgeschoben.

Das folgende Programm sollten Sie im Trace Modus des DEBUG (→ s. Kap. 3.2) ausführen, um den Stackpointer zu beobachten:

```
Hauptprogramm:
XXXX:0100 CALL 200        UP bei Adresse 200 rufen
XXXX:0103 INT 20          Programm beenden (im G-Modus ausführen)
Unterprogramm 1:
XXXX:0200 CALL 300        UP bei Adresse 300 rufen
XXXX:0203 RET             Return zum HP
Unterprogramm 2:
XXXX:0300 CALL 400        UP bei Adresse 300 rufen
XXXX:0303 RET             Return zum UP1
Unterprogramm 3:
XXXX:0400 RET             Return zum UP2
```

Die Befehle PUSH und POP

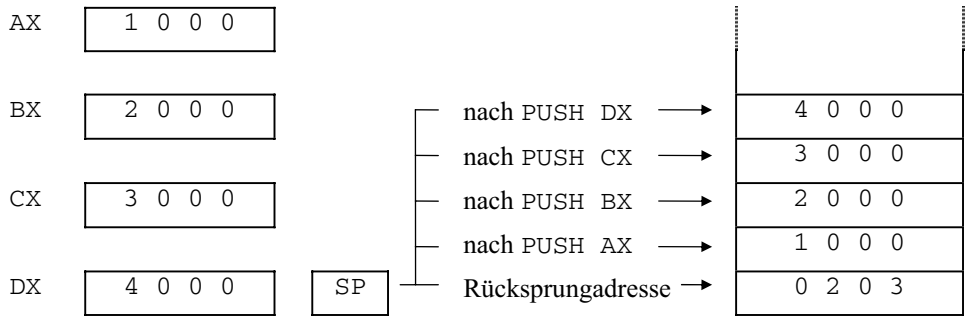
Der Stack dient nicht nur zur Verwaltung von Rücksprungadressen. Mit den beiden Maschinenbefehlen PUSH und POP lassen sich die Inhalte der 16-Bit-Register auf den Stack „retten“ bzw. zurückholen. Auf dem Stack gespeichert (PUSH) werden sie meist zu Beginn eines Unterprogramms, zurückgeholt (POP) am Ende. Werden auf diese Weise alle Register behandelt,

die das Unterprogramm verändert, findet das rufende Programm, z. B. das Hauptprogramm, nach Beendigung des UP alle Registerinhalte unverändert vor. Anderenfalls kann es böse Überraschungen geben. Gute Unterprogramme sollten die Register sichern, es sei denn, ein Register wird benutzt, um einen Wert zurückzuliefern, wie das häufig bei den Softwareinterrupts der Fall ist.

Betrachten wir die Wirkung der PUSH- und POP-Befehle an einem Beispiel. Zu Beginn eines Unterprogramms sollen die vier Datenregister AX, BX, CX und DX auf den Stack geschrieben werden.

■ **Beispiel:**

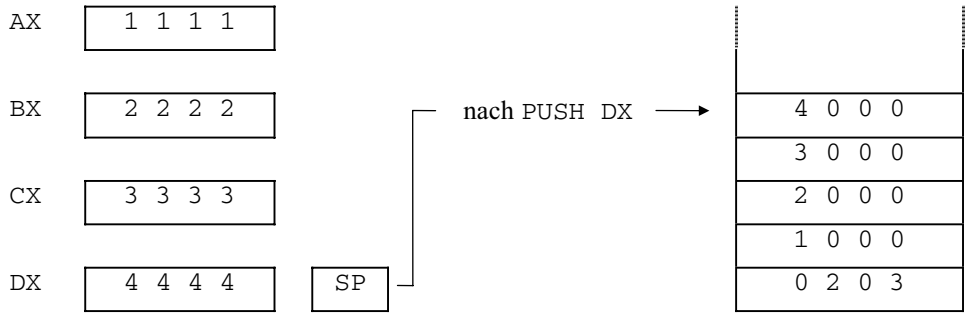
Befehlsfolge:
```
PUSH AX
PUSH BX
PUSH CX
PUSH DX
```



Register zu Beginn — Stack nach den PUSH-Befehlen

Danach werden im UP die Registerinhalte verändert.

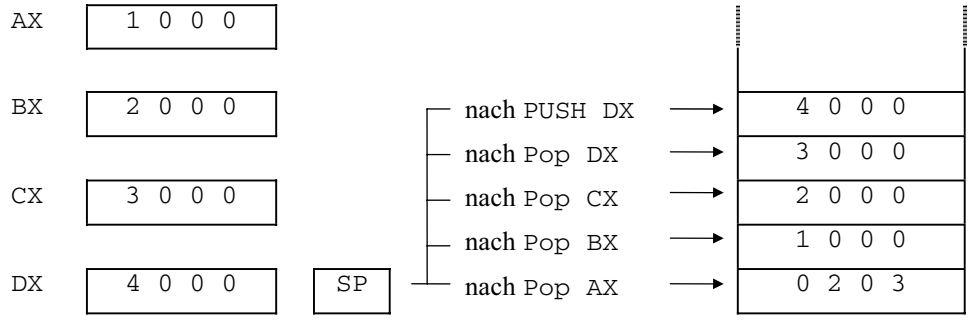
Befehlsfolge:
```
MOV AX,1111
MOV BX,2222
MOV CX,3333
MOV DX,4444
```



Register nach den MOV-Befehlen — Stackpointer unverändert

Zum Ende des UP, vor dem RET-Befehl sollen die alten Registerinhalte wieder hergestellt werden.

Befehlsfolge:
```
POP DX
POP CX
POP BX
POP AX
```



Register nach den POP-Befehlen

Stackpointer vor dem RET-Befehl

Wir beobachten

1. PUSH erniedrigt den Stackpointer (weil der Stack zu niederen Adressen wächst!). Damit wirkt PUSH aus Sicht des Stacks wie CALL.
2. POP erhöht den Stackpointer. Damit wirkt POP aus Sicht des Stacks wie RET.
3. Register, die mit PUSH auf den Stack geschrieben wurden, müssen in umgekehrter Reihenfolge mit POP zurückgeholt werden (wegen der LIFO-Eigenschaft des Stacks).
4. Beachtet man die Reihenfolge CALL ... PUSH ... POP ... RET, verwaltet sich der Stack „von selbst".

■

Folgende Regeln sollten beachtet werden

Alle Daten, die auf den Stack abgelegt wurden, müssen auch wieder entfernt werden: RET „entsorgt" CALL und POP „entsorgt" PUSH.

Verändere nie SP, denn es tut dir selber weh! Verwenden Sie BP, wenn Sie explizit per MOV-Befehl auf den Stack zugreifen möchten (nur für Profis zu empfehlen!).

Da PUSH und POP nur auf 16-Bit-Register angewendet werden dürfen, sind Befehle wie:

```
PUSH AL und
POP DH
```

verboten. Mit den speziellen Befehlen PUSHF und POPF lässt sich das Flagregister auf den Stack sichern und wieder zurückholen.

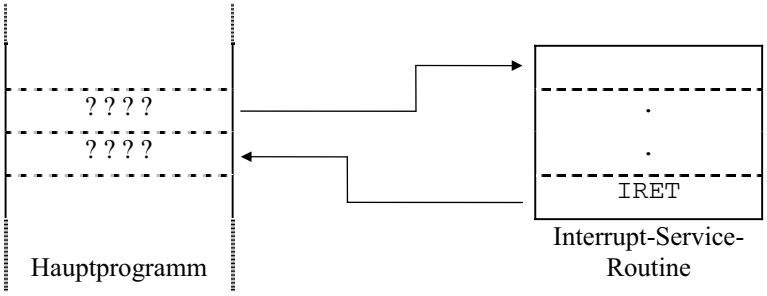
Das nachstehende Beispiel zeigt den generellen Aufbau eines UP, das die vier Datenregister sichert:

```
PUSH AX
PUSH BX
PUSH CD
PUSH DX
   .
   .        die eigentliche Prozedur
   .
POP DX
POP CX
POP BX
POP AX
RET
```

Achtung: In der 32-Bit-Welt beziehen sich alle PUSH- und POP-Befehle auf 32-Bit-Register (EAX, …, EF).

5.2 Interrupts

Ein Interrupt (Programmunterbrechung) unterbricht das gerade laufende Maschinenprogramm, um augenblicklich in die *Interrupt-Service-Routine* (ISR) einzutreten. Diese ist ähnlich wie ein Unterprogramm aufgebaut. Nach Beendigung der ISR mit dem Befehl `IRET` wird das unterbrochene Programm wieder fortgesetzt. Die folgende Abbildung verdeutlicht diese Situation.



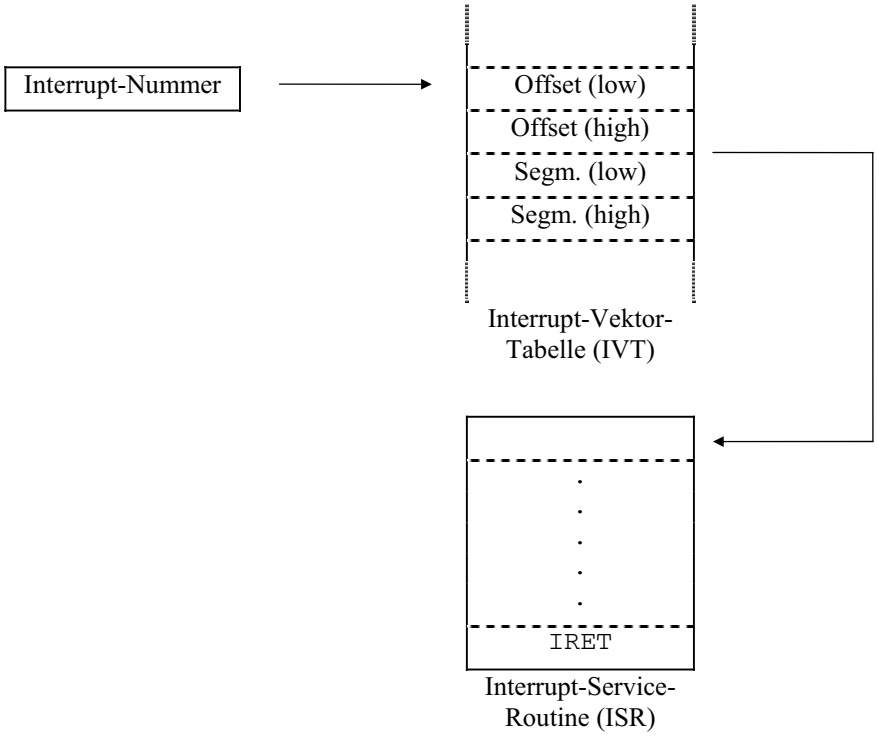
Der Unterschied zu gewöhnlichen Unterprogrammen besteht vor allem darin, dass Interrupts spontan, z. B. durch Tastendruck, auftreten können. Aus diesem Grund wird stets, vor der Rücksprungadresse, das Flagregister automatisch auf den Stack gerettet. Die Interrupt-Service-Routine muss daher mit dem Befehl `IRET` verlassen werden. `IRET` stellt zunächst die „alten" Flagzustände wieder her. Niemals darf der `RET`-Befehl zum Abschluss verwendet werden! Tritt ein Interrupt auf, so wird der gerade aktuelle Maschinenbefehl noch komplett abgeschlossen, dann erfolgt die Verzweigung in die ISR. Der aktuelle Befehl könnte z. B. ein `COMPARE` (CMP) sein. Als nächstes möge ein bedingter Sprung folgen. Dazwischen läge die ISR. Würde diese aus Sicht des unterbrochenen Programms die Flags verändern, wäre das für den weiteren Programmverlauf fatal.

Einen Sonderfall stellt der `INT`-Befehl (Softwareinterrupt) dar, mit dem man an determinierter Stelle einen Interrupt auslösen kann. Der weitere Ablauf unterscheidet sich nicht von dem eines Hardwareinterrupts. Mit diesem „Trick“ realisiert der Real-Mode die Systemaufrufe.

5.2.1 Die Interrupt-Vektor-Tabelle

Vom `INT`-Befehl ist Ihnen geläufig, dass diesem nicht etwa eine Startadresse wie bei `CALL`, z. B. `CALL 2000`, sondern eine Nummer, z. B. `INT 21`, beigefügt ist. Diese Nummer ist ein Verweis auf den entsprechenden Eintrag in der Interrupt-Vektor-Tabelle. Die „`21`“ stellt also einen Verweis auf den (hexadezimal!) 21. Eintrag dar.

Die Startadressen aller Interrupt-Service-Routinen (ISR) findet der Prozessor in der Interrupt-Vektor-Tabelle (IVT), ganz zu Beginn des Speichers ab Adresse `0000:0000`. Jedes Element der IVT besteht aus vier Bytes, die die Segment:Offset-Adresse der entsprechen ISR enthalten. Die Adressfindung erfolgt gewissermaßen „einmal um die Ecke“, also indirekt:



Der Prozessor bildet intern aus der Interrupt-Nummer die zugehörige Einsprungadresse in die IVT.

Insgesamt besitzt die Interrupt-Vektor-Tabelle im Real-Mode folgenden Aufbau:

0000:0000	Offset-Adresse (Low Byte) der ISR 0
0000:0001	Offset-Adresse (High Byte) der ISR 0
0000:0002	Segm.-Adresse (Low Byte) der ISR 0
0000:0003	Segm.-Adresse (High Byte) der ISR 0
0000:0004	Offset-Adresse (Low Byte) der ISR 1
0000:0005	Offset-Adresse (High Byte) der ISR 1
0000:0006	Segm.-Adresse (Low Byte) der ISR 1
0000:0007	Segm.-Adresse (High Byte) der ISR 1
0000:0008	Offset-Adresse (Low Byte) der ISR 2
0000:0009	Offset-Adresse (High Byte) der ISR 2
0000:000A	Segm.-Adresse (Low Byte) der ISR 2
0000:000B	Segm.-Adresse (High Byte) der ISR 2
0000:000C	Offset-Adresse (Low Byte) der ISR 3
0000:000D	Offset-Adresse (High Byte) der ISR 3
0000:000E	Segm.-Adresse (Low Byte) der ISR 3
0000:000F	Segm.-Adresse (High Byte) der ISR 3
u.s.w.	

Die Startadresse des `INT 0` steht in den Speicheradressen `0000:0000` bis `0000:0003`, entsprechend beginnt der `INT-1`-Zeiger bei `0000:0004`. Die Anfangsadresse des IVT-Elements eines Interrupts ermittelt man also nach der Formel

Interrupt-Nummer * 4

(Vorsicht: Nummerierung hexadezimal).

Insgesamt enthält die IVT 100(hex) (=256 dez.) Einträge, sie ist also 1024 Byte lang (0 - 3FF(hex)).

Die Tabelle wird beim *system boot* (der PC wird immer im Real Mode gestartet!) mit Standardwerten besetzt.

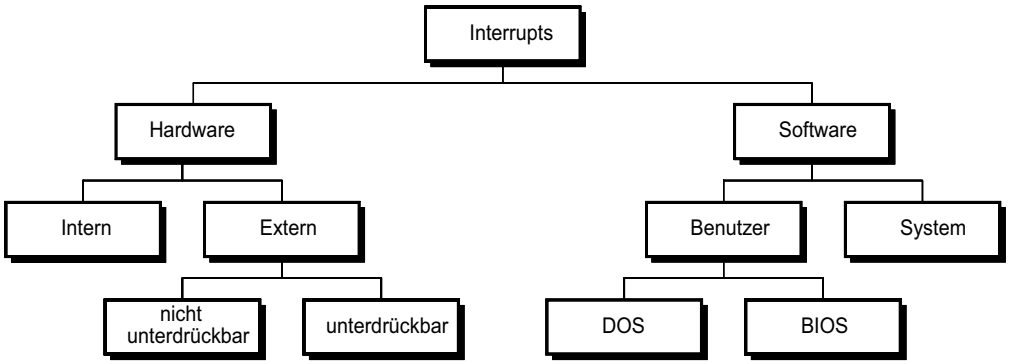
Die folgende Übersicht zeigt einige wichtige Einträge in der Interrupt-Vektor-Tabelle (IVT) im 16-Bit-Real-Mode:

INT-Nr.	Adressenbereich	Standardbelegung
00	0000 - 0003	CPU: Division durch Null
01	0004 - 0007	CPU: Single Step
02	0008 - 000B	RAM-Fehler (NMI)
03	000C - 000F	CPU: Breakpoint erreicht
04	0010 - 0013	CPU: numerischer Überlauf
05	0014 - 0017	Print Screen (Bildschirm Hardcopy)
..	.....	
08	0020 - 0023	IRQ0: Timer-Baustein
09	0024 - 0027	IRQ1: Tastatur
0A	0028 - 002B	IRQ2: 2. Interruptcontroller
0B	002C - 002F	IRQ3: 2. serielle Schnittstelle
0C	0030 - 0033	IRQ4: 1. serielle Schnittstelle
0D	0034 - 0037	IRQ5: 2. Drucker
0E	0038 - 003B	IRQ6: Diskettenlaufwerk
0F	003C - 003F	IRQ7: 1. Drucker
10	0040 - 0043	BIOS: Video-Funktionen
..	.....	
16	0058 - 005B	BIOS: Tastaturabfragen
..	.....	
19	0064 - 0067	BIOS: Warmstart <Ctrl><Alt><Del>
..	.....	
20	0080 - 0083	DOS: Programm beenden
21	0084 - 0087	DOS: Diverse DOS-Funktionen
..	.....	
F1	03C4 -	nicht benutzt: können vom Anwender frei belegt
FF	- 03FF	werden

Interrupt-Vektor-Tabelle (Auszug)

5.2.2 Die Interruptarten

Die verschiedenen Interruptarten der 80(X)86-Familie im Real Mode lassen sich wie folgt systematisieren:



Softwareinterrupts haben wir bereits kennen gelernt, z. B. den DOS-Interrupt 21(hex) oder den BIOS-Interrupt 10(hex) (Video-Interrupt). Software-Interrupts erfolgen programmgesteuert wie Unterprogramm-Calls. Der entsprechende Maschinenbefehl heißt `INT <int-nr.>`, z. B. ruft der Befehl „`INT 16`“ den Tastatur-Interrupt auf. Der wichtigste Unterschied zu `CALL` besteht darin, dass die IVT zur Ermittlung der Startadresse genutzt wird.

Jeder Benutzer kann sich eigene Service-Routinen schreiben. Er muss jedoch dafür Sorge tragen, dass die Startadresse korrekt in das entsprechende Element der IVT eingetragen wird. Wie das geschieht, wird später behandelt. Man sollte nach Möglichkeit einen „freien“ Interrupt verwenden, damit es nicht zu Kollisionen mit System-Interrupts kommt. Wie Sie bereits wissen, unterscheidet man DOS- und BIOS-Interrupts.

Hardwareinterrupts unterbrechen das laufende Programm spontan, also *nicht* softwaregesteuert. Auch hier muss unbedingt eine entsprechende Interrupt-Service-Routine zur Verfügung stehen, die mindestens aus einem Befehl (`IRET`) besteht. Anderenfalls resultiert ein Absturz des Programms (unter Windows 98 sogar des gesamten Rechners!).

Standard-ISRs, z. B. zur Behandlung von Tastatureingaben, beinhaltet das Betriebssystem. Auch hier können eigene Routinen die Standardroutinen ersetzen. Dazu muss der entsprechende Interruptvektor so „verbogen“ werden, dass er auf die neue ISR zeigt.

Interne Hardwareinterrupts sind in der Tabelle oben mit CPU gekennzeichnet. Sie werden vom Prozessor selbst ausgelöst und können nicht per Software unterdrückt werden.

Externe Hardwareinterrupts werden von sonstigen Komponenten des Rechners (Controller, Tastatur, ...) oder von Fremdgeräten (z. B. externes Messgerät, das über eine I/O-Karte mit dem Rechner verbunden ist) ausgelöst. Sie können per Software unterdrückt (maskiert) werden.

Unterdrückbare Interrupts können per Software verboten werden. Der Befehl dafür lautet: `CLI`. Wirkung: Löschen des Interrupt Flags.

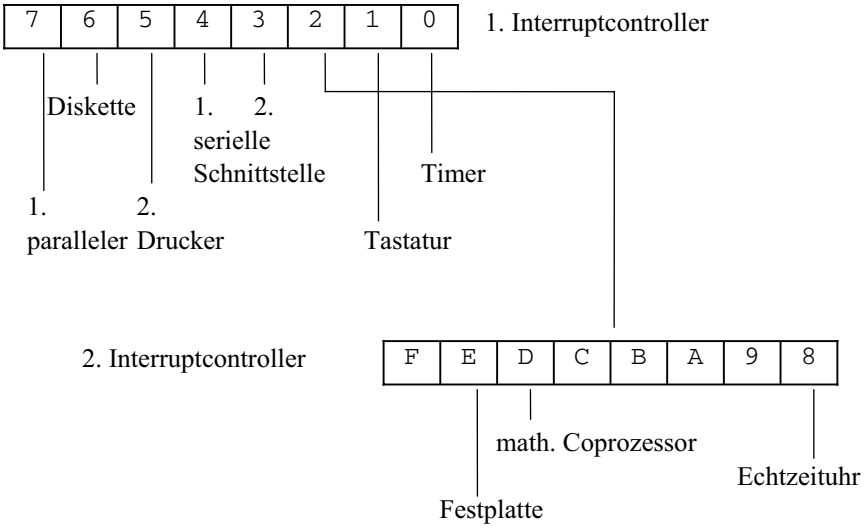
Beispiel: `INT 09` (Tastatur)

Mit dem Befehl: `STI` können die unterdrückbaren Interrupts wieder freigegeben werden.

Nichtunterdrückbare Interrupts zeigen Hardwarefehler an. Auf sie hat der `CLI`-Befehl keine Wirkung.

5.2.3 Der Interruptcontroller

Der Unterbrecherbaustein 8259 entlastet den Mikroprozessor von der Hardwareinterrupt-Verwaltung. Dieser Baustein verwaltet acht Interruptquellen (*Interrupt Requests, IRQ*). Ab der AT-Klasse (80286) sind alle PC mit zwei Interruptcontrollern ausgestattet, wobei der zweite auf dem IRQ2-Eingang des ersten zusammengefasst wurde (Kaskadierung).



Die Zahlen entsprechen den IRQ-Nummern (siehe IVT). Einige IRQ-Eingänge des zweiten Controllers sind frei für Anwenderbelegungen.

Sollten mehrere Interrupt-Anforderungen gleichzeitig anliegen, erhält der mit der niedrigsten IRQ-Nummer den Vorrang (Interrupt-Prioritäten). Die Controller-Interrupts sind *maskierbar*, d. h. sie können selektiv an- und abgeschaltet werden. Hierzu dient das Interrupt-Mask-Register (IMR) des 8259. Die Controller-Register werden über I/O-Ports angesprochen (→ s. Kap. 6.1). In diesem 8-Bit-IMR-Register steht eine „0“ für Freigabe, eine „1“ für Sperren des entsprechenden Interrupts.

5.3 Aufgaben

1) Erstellen Sie mit dem DEBUG folgendes Assemblerprogramm:

```
Adresse  100:   CALL 200
                INT 20
         200:   CALL 300
                RET
         300:   CALL 400
                RET
         400:   CALL 500
                RET
         500:   RET
```

Führen Sie das Programm im Single Step Modus aus. Notieren Sie das Verhalten der Register IP (Programmcounter) und SP (Stackpointer).

2) Welcher der folgenden Begriffe passt nicht zu dem Begriff „Stack“?

a) Stapelspeicher b) LIFO c) EPROM d) SP-Register

e) `CALL` f) `IRET` g) `RET`

3) Gegeben seien folgende Registerinhalte:

`AX = 0003(hex) BX = 0004(hex) CX = 0001(hex) DX = 0002(hex)`

Es werden nun folgende Stackoperationen ausgeführt:

`PUSH AX`		`POP BX`	(zur Klarstellung:
`PUSH BX`	und	`POP DX`	erst alle `PUSH`-Befehle,
`PUSH CX`	anschließend	`POP CX`	dann alle `POP`-Befehle)
`PUSH DX`		`POP AX`	

Wie lauten die Registerinhalte jetzt (h = hex)?

	a)	b)	c)	d)	e)	f)
AX	0004h	0003h	0003h	0004h	0002h	
BX	0003h	0002h	0002h	0001h	0004h	alle
CX	0002h	0004h	0001h	0003h	0003h	falsch
DX	0001h	0001h	0004h	0002h	0001h	

4) Welche der folgenden Begriffe passt nicht zu dem Begriff „Stack“?

a) Stapelspeicher b) FIFO c) RAM

d) SS-Register e) ISR f) Unterprogramm

5)

Adresse	**Befehl (DEBUG Assembler-Syntax)**
100	MOV AX,1000
103	MOV BX,2000
106	PUSH AX
107	PUSH BX
108	CALL 200
10b	INT 20
.	.
.	.
200	POP BX
201	RET

Der Stack-Pointer (SP-Register) enthalte zu Anfang des Programms die Adresse `FFEE` (hex). Welchen Inhalt haben die Register IP und SP unmittelbar nach Ausführung des `RET`-Befehls?

	a)	b)	c)	d)	e)	f)	g)
IP:	10B	10B	1000	1000	1000	2000	2000
SP:	FFEE	FFEC	FFEC	FFEA	FFEE	FFEC	FFEE

6) Der Stackpointer zeige zu Beginn des folgenden Programms auf die Adresse FFEE mit dem Inhalt 0000 (alle Werte hexadezimal).

Das Programm lautet in der Syntax des DEBUG Assemblers:

Adresse:	Befehl
100:	CALL 104
103:	RET
104:	CALL 108
107:	RET
108:	RET

Das Programm startet bei Adresse 100(hex). In welcher Reihenfolge werden die einzelnen Befehle ausgeführt?

a)	b)	c)	d)	e)	f)	g)	h)
100:	100:	100:	100:	100:	100:	100:	alle falsch
108:	103:	104:	104:	103:	104:	104:	
104:	104:	108:	108:	Absturz	108:	108:	
107:	107:	103:	Absturz		107:	107:	
103:	108:	107:			104:	103:	

7) Ein Programm laute in der Syntax des DEBUG Assemblers:

Adresse:	Befehl
1000:	MOV AX,1000
1003:	CALL 1007
1006:	RET
1007:	PUSH AX
1008:	CALL 100C
100B	RET
100C	RET

In welcher Reihenfolge (Adressen alle hex) werden die Befehle ausgeführt?

a)	b)	c)	d)	e)	f)	g)
1000	1000	1000	1000	1000	1000	alle
1003	1003	1003	1003	1003	1003	falsch
1007	1006	1007	1006	1007	1007	
1008	1007	1008	1007	1008	1008	
100C	1008	100B	100B	Absturz	100C	
100B	100B	100C	100C		100B	
1006	100C	1000	1000		1000	
Absturz	Absturz	usw.	usw.	usw.	usw.	

8) Der Stackpointer zeige zu Beginn des folgenden Programms auf die Adresse FFEE(mit dem Inhalt 0000 (alle Werte hexadezimal).

Das Programm lautet (xxx: = hexadezimale Adresse des dahinter stehenden Befehls):

```
100:  CALL 110
103:  RET
110:  CALL 100
113:  RET
```

Das Programm startet bei Adresse `100(hex)`. Wie sieht der Stack nach Ausführung von genau 6 Programmschritten (z. B. mit dem T-Kommando des DEBUG) aus?

-> = Stackpointer

	a)	b)	c)	d)
Stackadressen	Inhalte der Adressen und Stackpointer			
FFEE	0000	0000	-> 0000	0000
FFEC	0103	0103		-> 0103
FFEA	0113	0103		
FFE8	0103	0103		
FFE6	0113	0113		
FFE4	0103	0113		
FFE2	-> 0113	-> 0113		

9) Schreiben Sie mit Hilfe von DEBUG ein Programm, das

1. ein PRINT-SCREEN auslöst und
2. normal beendet wird.

Hinweis: nur zwei Befehle sind notwendig.

10) Die Interrupt-Vektor-Tabelle eines PCs beginne wie folgt:

Adresse	Inhalt
0000:0000	EA
1	56
2	8C
3	02
4	74
5	07
6	70
7	00

(alle Werte hexadezimal)

Wie lautet die vollständige Anfangsadresse (Segment:Offset) der Interrupt-Service-Routine für den Interrupt Nummer 0 (Divisionsfehler)?

a) `8C02:EA56` b) `028C:56EA` c) `EA56:8C02` d) `56EA:028C` e) `0000:028C`

f) `028C:0000`

6 Controller-Bausteine und Ports

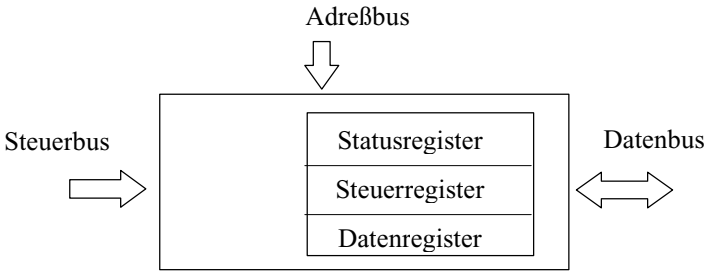
Zahlreich „intelligente“ Controller-Bausteine auf oder außerhalb der CPU-Platine befreien den Mikroprozessor von vielfältigen Spezialaufgaben. Einer wurden bereits angesprochen: der Interruptcontroller.

Es folgt eine Übersicht über die wichtigsten Controller-Bausteine:

INTEL-Nummer	Bezeichnung	Einsatz bzw. Tätigkeit
8237	DMA-Controller	schneller Datenaustausch zwischen RAM und Plattenspeicher
8248	Taktgenerator	Taktgeber zur Synchronisation der Signale
8254	Timer	löst standardmäßig 18,2 mal pro Sekunde einen Interrupt (`INT 08`) aus
8255	Peripherie-Interface	stellt die Verbindung zwischen CPU und einigen peripheren Geräten, wie Tastatur und Lautsprechen, her
8259	Interrupt-Controller	übernimmt die Verwaltung der maskierbaren Hardware-Interrupts
8087 80287 80387	Mathematischer Coprozessor	ermöglicht hardwaremäßige Fließpunktoperationen incl. der wichtigsten mathematischen Funktionen (`SIN`, `COS`, `SQRT`...) Arithmetikprozessoren sind nicht zwingend notwendig! Ab dem 80486 sind Coprozessoren auf dem Prozessorchip integriert.

Externe Controller befinden sich nicht auf der CPU-Platine (*motherboard*). Als Beispiele seien genannt: der Bildschirmcontroller 6845, PCI- und Festplattencontroller.

Prinzipiell sind Controller-Bausteine nach folgendem Schema aufgebaut:



Statusregister: nur lesen
Steuer- oder Controlregister: schreiben und lesen
Datenregister: schreiben oder lesen

Die Register des Controllers sind über Portadressen mit Hilfe der Maschinenbefehle `IN` und `OUT` ansprechbar. In den Kapiteln 6.2 und 6.3 gehen wir konkret auf die Programmierung von Controller-Bausteinen ein.

6.1 Die Befehle „IN“ und „OUT“

Alle Controller-Bausteine können über *Ports* angesprochen bzw. programmiert werden. Die INTEL-80(X)86-Familie verfügt im Real Mode über einen I/O-Adressraum von 64KByte, der *nicht* mit dem Speicheradressraum zusammenfällt. Portadressen werden wie Speicheradressen über den Adressbus angesprochen. Zur Unterscheidung aktiviert der Prozessor zusätzlich eine Steuerleitung. Die I/O-Adressen sind 16 Bit lang. Die meisten entsprechen 8-Bit-Ports, einige 16-Bit-Ports.

Einige Portadressen sind vom System belegt, die folgende (nicht vollständige!) Tabelle zeigt einige Standardeinträge und freie Ports:

Portadresse von	bis	**Funktion**
0000	000F	DMA-Controller 8237
....	....	
0020	003F	1. Interrupt-Controller 8259
0040	005F	Timer-Baustein 8254
0060	006F	Tastatur-Controller 8042
....	....	
00A0	00BF	2. Interrupt-Controller 8259
00C0	00CF	2. DMA-Controller 8237
....	....	
00F0	00FF	Math. Coprozessor 80287
....	....	
0220	024F	frei für I/O-Erweiterungskarten
....	....	
02F0	02F7	frei für I/O-Erweiterungskarten
....	....	
02F8	02FF	2. Serielle Schnittstelle (COM2)
....	....	
0360	036F	frei für I/O-Erweiterungskarten
....	....	
0390	0393	Netzwerkadapter
....	....	
03CD	03CF	frei für I/O-Erweiterungskarten
....	....	
03F8	03FF	1. Serielle Schnittstelle (COM1)

Die höheren Adressen stehen zur freien Verfügung.

Die Hauptfunktionen einiger Controller lassen sich in vielen Fällen recht komfortabel über BIOS-Aufrufe (Software-Interrupts, → s. Kap. 5.2) ansprechen.

Möchte man mehr als nur die Standardfunktionen oder ganz einfach schneller sein, bleibt nur die direkte Programmierung mittels der Maschinenbefehle

`IN` und `OUT`.

`IN` liest von der entsprechenden Peripherie über deren Portadresse in das Register `AL` (8-Bit-Port) oder `AX` (16-Bit-Port). Die Portadresse (I/O-Kanal) steht im `DX`-Register (> 8 Bit) oder wird unmittelbar angegeben (8 Bit).

OUT gibt den Inhalt von AL (bei Ausgabe auf einen 8-Bit-Port) oder AX (bei Ausgabe auf einen 16-Bit-Port) auf das angewählte Peripheriegerät aus. Dessen Adresse steht entweder im DX-Register oder sie wird unmittelbar angegeben. Achtung: wird die Adresse direkt angegeben, darf sie maximal 8 Bit (= FF) lang sein!

■ **Beispiele**

`IN AL,C0`	Eingabe vom 8-Bit-I/O-Port C0 nach AL
`IN AX,C2`	Eingabe vom 16-Bit-Port C2 nach AX
`IN AL,DX`	Eingabe von einem 8-Bit-Port, dessen Adresse im DX-Register steht, nach AL
`IN AX,DX`	Eingabe von einem 16 Bit-Port, dessen Adresse im DX-Register steht, nach AX
`OUT DX,AL`	Ausgabe des AL-Inhalts auf das 8-Bit-Port, dessen Adresse im DX-Register steht
`OUT DX,AX`	Ausgabe des AX-Inhalts auf das 16-Bit-Port, dessen Adresse im DX-Register steht
`OUT C6,AL`	Ausgabe des AL-Inhalts auf das 8-Bit-Port C6
`OUT CA,AX`	Ausgabe des AX-Inhalts auf das 16-Bit-Port CA

nicht erlaubt sind:

`IN AX,100`	Die direkt angegebene Portadresse ist länger als 8 Bit.
`OUT 200,AL`	

■

6.2 Beispiel: Programmierung des Interrupt-Controllers

In Kapitel 5.2.3 wurde der Interrupt-Controller 8259 kurz vorgestellt. Wir erwähnten dabei bereits das Interrupt-Masken-Register (IMR), das über die Portadresse 21 ansprechbar ist. Mit seiner Hilfe lassen sich die nachstehenden IRQs selektiv abschalten.

1. Contr.	2. Contr.	Funktion
IRQ0		Timer-Interrupt, Standard: 18.2 Hz
IRQ1		Tastatur-Interrupt
IRQ2		Interrupts von 2. Controller
	IRQ8	Echtzeituhr
	IRQ9	Software Umleitung von IRQ2 auf INT A
	IRQA	frei nutzbar für den Anwender
	IRQB	frei nutzbar für den Anwender
	IRQC	frei nutzbar für den Anwender
	IRQD	Math. Coprozessor
	IRQE	Festplatten-Controller
	IRQF	frei nutzbar für den Anwender
IRQ3		2. Serielle Schnittstelle (COM2)
IRQ4		1. Serielle Schnittstelle (COM2)
IRQ5		2. Parallel-Schnittstelle (LPT2)
IRQ6		Disketten-Controller
IRQ7		1. Parallel-Schnittstelle (LPT1)

Die Interruptquellen 8 bis F werden auf dem IRQ2-Eingang des ersten Controllers zusammengefasst.

Möchte man nun beispielsweise nur den Tastatur-Interrupt erlauben, alle anderen aber verbieten, so muss auf Bit 1 des IMR eine „0“ , auf alle anderen eine „1“, gegeben werden:

```
MOV AL,FD       Bitmuster 1111 1101, alle Interrupts außer
OUT 21,AL       Tastatur-Interrupts sind gesperrt (→ s. Kap. 5.2.3)
                Port 21(hex): IMR des ersten Controllers
```

Außerdem verfügt der Interrupt-Controller über ein Steuerwort-Register. Beim ersten 8259 ist es über die

Port-Adresse 20h

zugreifbar.

Jeder Controller-Interrupt verbietet weitere Controller-Interrupts automatisch. Erst das *End-Of-Interrupt*-Kommando (EOI) gibt den Controller wieder frei. Das EOI-Kommando wird in der Regel am Ende einer Interrupt-Service-Routine (ISR) abgesetzt. Es hat folgende Form:

```
MOV  AL,20      EOI-Kommando
OUT  20,AL
```

Die Gleichheit zwischen Adresse und Kontrollwort ist zufällig, bzw. kommt daher, weil Bit 5 im Steuerwort-Register für die Freigabe zuständig ist. Andere Kontrollwörter sind in diesem Zusammenhang ohne Belang.

Wichtig für die Interrupt-Programmierung ist darüber hinaus das Interrupt Flag (IF) im Statusregister des 80(X)86, das durch zwei Befehle beeinflusst werden kann:

```
CLI             sperrt alle Interrupts (IF = 0)
STI             gibt alle Interrupts frei (IF = 1)
```

Erfolgt ein Interrupt, sind in der Service Routine hardwaremäßig alle weiteren Interrupts gesperrt. Mit dem `STI`-Befehl kann man sie jederzeit wieder zulassen. Der `IRET`-Befehl sorgt am Ende der ISR in jedem Fall automatisch für eine Freigabe.

Zum Schluss dieses Kapitels folgt ein ausführliches Programmbeispiel, dass den Umgang mit Hardware-Interrupts illustriert. Es benutzt den Timer-Interrupt 08, der standardmäßig 18,2 mal pro Sekunde erfolgt, um zyklisch drei Prozeduren aufzurufen, die jeweils die Buchstaben „A“, „B“ und „C“ auf den Bildschirm schreiben. Zunächst biegt das Programm den Vektor des Systemtimer-Interrupts (`INT 08`) auf eine eigene Service-Routine um. Diese Interrupt-Service-Routine verteilt die Rechenzeit gleichmäßig auf die drei Unterprogramme.

Achten Sie bei der Eingabe mit DEBUG (ohne die Kommentare!) auf die unterschiedlichen Anfangsadressen der verschiedenen Programmteile. Ehrlich gesagt empfehlen wir Ihnen, das Programm nicht einzutippen, sondern von unserer Buch-Webseite (**multi.com**) zu laden.

```
-------------------------------------------------------------
*************** VORBEREITUNGEN ************
-------------------------------------------------------------
XXXX:0100 MOV BL,01           Unterprogramm-Nummer-Flag
XXXX:0102 MOV [200],BL        auf 1 setzen
XXXX:0106 MOV DX,210          Text bei Adresse 210
XXXX:0109 MOV AH,9            mit INT 21, Function 9
XXXX:010B INT 21              ausgeben
XXXX:010D MOV DX,240          Text bei Adresse 240
XXXX:0110 MOV AH,9            mit INT 21, Function 9
XXXX:0112 INT 21              ausgeben
XXXX:0114 MOV AX,50           äußere Zeitschleife
XXXX:0117 MOV CX,7FFF         innere Zeitschleife
XXXX:011A LOOP 11A            innere Zeitschleife Ende
XXXX:011C DEC AX
XXXX:011D JNE 117             äußere Zeitschleife Ende
XXXX:011F MOV AH,0C           Tastaturpuffer löschen
XXXX:0121 MOV AL,0            mit INT 21,
XXXX:0123 INT 21              Function C
XXXX:0125 MOV AH,35           INT 08 Vektor ermitteln
XXXX:0127 MOV AL,8            mit INT 21,
XXXX:0129 INT 21              Function 35
XXXX:012B MOV [280],BX        Originaladressen sichern:
XXXX:012F MOV BX,ES           BX = Offset-Adresse
XXXX:0131 MOV [282],BX        ES = Segment-Adresse
XXXX:0135 CLI                 alle Interrupts verbieten
XXXX:0136 MOV AX,CS           INT 08 Vektor umbiegen:
XXXX:0138 MOV DX,180          Offset-Adresse nach DX
XXXX:013B MOV DS,AX           Segment-Adresse nach DS
XXXX:013D MOV AH,25           Function Code 25
XXXX:013F MOV AL,8            Interrupt-Nummer nach AL
XXXX:0141 INT 21              Umbiegen per DOS-Interrupt
XXXX:0143 STI                 alle Interrupts wieder erlauben
-------------------------------------------------------------
*************** „HAUPTPROGRAMM“ ************
-------------------------------------------------------------
XXXX:0144 CMP AH,AH           Zero Flag setzen
XXXX:0146 MOV AH,01           Tastaturpuffer prüfen:
XXXX:0148 INT 16              wurde eine beliebige Taste gedrückt?
XXXX:014A JNZ 14E             wenn ja, Ende
XXXX:014C JMP 144             wenn nein, weiter
```

```
-------------------------------------------------------------
****************** Programm-Ende ***************
-------------------------------------------------------------
XXXX:014E CLI                   alle Interrupts verbieten
XXXX:014F MOV AX,[282]          Original-Interrupt 8 Routine
XXXX:0152 MOV DX,[280]          wiederherstellen
XXXX:0156 MOV DS,AX             mit DOS-Interrupt,
XXXX:0158 MOV AH,25             Function Code 25
XXXX:015A MOV AL,8
XXXX:015C INT 21
XXXX:015E MOV AX,CS
XXXX:0160 MOV DS,AX
XXXX:0162 STI                   Interrupts wieder freigeben
XXXX:0163 INT 20                Programmende
-------------------------------------------------------------
************ eigene ISR zu INT 08 *************
-------------------------------------------------------------
XXXX:0180 PUSH AX               AX retten
XXXX:0181 MOV AX,CS             DS auf CS
XXXX:0183 MOV DS,AX             setzen
XXXX:0185 NOP
XXXX:0186 MOV AL,[0200]         den aktuellen Wert des Unterprogramm-
XXXX:0189 CMP AL,01             Nummer-Flags nacheinander auf 1, 2
XXXX:018B JZ 0199               oder 3 überprüfen
XXXX:018D CMP AL,02
XXXX:018F JZ 01A4
XXXX:0191 CMP AL,03
XXXX:0193 JZ 01AF
XXXX:0195 NOP
XXXX:0196 JMP 01B8              Fehlerausgang
XXXX:0198 NOP
XXXX:0199 INC AL                AL=1: UP-Nummer erhöhen
XXXX:019B MOV [0200],AL         und abspeichern
XXXX:019E CALL    0300          UP1 aufrufen
XXXX:01A1 JMP 01B8              zum Ausgang springen
XXXX:01A3 NOP
XXXX:01A4 INC AL                AL=2: UP-Nummer erhöhen
XXXX:01A6 MOV [0200],AL         und abspeichern
XXXX:01A9 CALL    0350          UP2 aufrufen
XXXX:01AC JMP 01B8              zum Ausgang springen
```

```
XXXX:01AE NOP
XXXX:01AF MOV AL,01          AL=3: UP-Nummer auf 1 setzen
XXXX:01B1 MOV [0200],AL      und abspeichern
XXXX:01B4 CALL    03A0       UP3 aufrufen
XXXX:01B7 NOP
XXXX:01B8 MOV AL,20          Ausgang: Interrupt-Controller
XXXX:01BA OUT 20,AL          Interrupts freigeben (Port 20)
XXXX:01BC POP AX             AX wiederherstellen
XXXX:01BD IRET               ISR verlassen
-------------------------------------------------------------
************** Unterprogramm 1 **************
-------------------------------------------------------------
XXXX:0300 PUSH AX            verwendete Register
XXXX:0301 PUSH DS            auf den Stack sichern
XXXX:0302 PUSH DX
XXXX:0303 PUSH BX
XXXX:0304 MOV AH,E           Function Code E
XXXX:0306 MOV AL,41          ASCII-Code für „A“
XXXX:0308 MOV BX,0
XXXX:030B INT 10             BIOS-Video-Interrupt
XXXX:030D POP BX             verwendete Register
XXXX:030E POP DX             wiederherstellen
XXXX:030F POP DS
XXXX:0310 POP AX
XXXX:0311 RET                UP1 verlassen
-------------------------------------------------------------
************** Unterprogramm 2 **************
-------------------------------------------------------------
XXXX:0350 PUSH AX            verwendete Register
XXXX:0351 PUSH DS            auf den Stack sichern
XXXX:0352 PUSH DX
XXXX:0353 PUSH BX
XXXX:0354 MOV AH,E           Function Code E
XXXX:0356 MOV AL,42          ASCII-Code für „B“
XXXX:0358 MOV BX,0
XXXX:035B INT 10             BIOS-Video-Interrupt
XXXX:035D POP BX             verwendete Register
XXXX:035E POP DX             wiederherstellen
XXXX:035F POP DS
XXXX:0360 POP AX
XXXX:0361 RET                UP2 verlassen
```

```
-------------------------------------------------------------
************** Unterprogramm 3 **************
-------------------------------------------------------------
XXXX:03A0 PUSH AX              verwendete Register
XXXX:03A1 PUSH DS              auf den Stack sichern
XXXX:03A2 PUSH DX
XXXX:03A3 PUSH BX
XXXX:03A4 MOV AH,E             Function Code E
XXXX:03A6 MOV AL,43            ASCII-Code für „C“
XXXX:03A8 MOV BX,0
XXXX:03AB INT 10               BIOS-Video-Interrupt
XXXX:03AD POP BX               verwendete Register
XXXX:03AE POP DX               wiederherstellen
XXXX:03AF POP DS
XXXX:03B0 POP AX
XXXX:03B1 RET                  UP3 verlassen
-------------------------------------------------------------
*************** Datenbereich ****************
-------------------------------------------------------------
XXXX:0200 ?                    Unterprogramm-Nummer-Flag
XXXX:0210 0D 0A 0A 5A          1. Ausgabestring:
          55 4D 20 41          ZUM ABBRUCH$
          42 42 52 55
          43 48 24
XXXX:0240 0D 0A 0A 42          2. Ausgabestring:
          45 4C 49 45          BELIEBIGE TASTE DRUECKEN$
          42 49 47 45
          20 54 41 53
          54 45 20 44
          52 55 45 43
          4B 45 4E 0A
          0D 24
XXXX:0280 ?                    Speicher für Offset-Adresse der
                               Original-ISR 08
XXXX:0282 ?                    Speicher für Segment-Adresse der
                               Original-ISR 08
```

Falls Sie das Programm trotz unserer Warnung per Hand eingetippt haben, müssen Sie die Datenwerte im Datenbereich mit dem E-Kommando des DEBUG eingeben. „?“ bedeutet: Anfangswert ist unerheblich und muss nicht eingegeben werden.

In diesem Fall nennen Sie das Programm `MULTI.COM`:

– `N MULTI.COM`

Geben Sie die ungefähre Programmlänge in das Registerpaar `BX:CX` (`BX` = 0, `CX` = 400, mit R-Kommando).

Schreiben Sie das Programm auf die Festplatte:

– `W`

Erläuterungen zum Programm `MULTI`

Das Programm trifft zunächst einige Vorbereitungen:

- das UP-Nummer-Flag (Adresse `200(hex)`) wird auf den Anfangswert 1 gesetzt,
- zwei Nachrichten werden auf den Bildschirm gegeben,
- die Anfangsadresse der Original-ISR des Timer-Interrupts 08 wird ermittelt und gesichert, dies geschieht mit Hilfe eines DOS-Aufrufs (Funktion 35h).

Interrupt-Adresse ermitteln		
`MOV`	`AH,35`	Function Code
`MOV`	`AL,xx`	Interrupt-Nummer (konkret einsetzen)
`INT`	`21`	DOS-Interrupt
`MOV`	`adr1,BX`	`BX` enthält die Offsetadresse
`MOV`	`BX,ES`	`ES` enthält die Segmentadresse
`MOV`	`adr2,BX`	`adr1` und `adr2` sind Wortadressen, für die konkrete Adressen einzusetzen sind.

- Die Anfangsadresse der neuen ISR wird in die Interrupt-Vektor-Tabelle geschrieben. Die Funktion 25(hex) des DOS-Interrupts erleichtert dies.

Interrupt-Adresse „verbiegen“		
`CLI`		alle Interrupts verbieten
`MOV`	`AX,CS`	Codesegment der neuen ISR
`MOV`	`DX,adr`	Offsetadresse der neuen ISR, (für „`adr`“ ist die konkrete Adresse einzusetzen)
`MOV`	`DS,AX`	Segmentadresse der neuen ISR
`MOV`	`AH,25`	Function Code
`MOV`	`AL,08`	Nummer des zu verbiegenden Int.
`INT`	`21`	DOS-Interrupt
`STI`		Interrupts wieder zulassen

- Der Befehl `NOP` (*No Operation*) in der ISR bewirkt, wie sein Name andeutet, nichts. Der von ihm erzeugte Code (90(hex)) an der Einsprungadresse der ISR kann jedoch von anderen Programmen abgefragt werden. Diese können somit feststellen, dass die Timer-ISR verändert wurde. Allgemein ist auf diese Weise erkennbar, ob sich bestimmte Programme im Speicher befinden.
- `MULTI.COM` geht in den Wartezustand. Es beendet sich, wenn ein Tastendruck erfolgt. Dann wird der alte Interrupt-Vektor wieder restauriert. Auch dies geschieht mit der Funktion 25(hex) des `INT 21(hex)`.

Die 18,2-mal pro Sekunde auftretenden Timer-Interrupts führen dazu, dass abwechselnd die Prozeduren UP1, UP2 und UP3 (je nach Wert des Flags) aufgerufen werden.

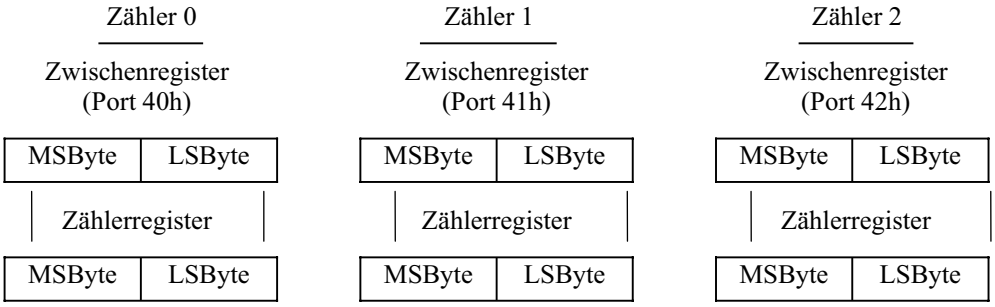
Weitere Details entnehmen Sie bitte den Kommentierungen zu den einzelnen Befehlen.

6.3 Aufgabe

Der Timerbaustein 8254 befindet sich auf der CPU-Platine. Er wird über die vier 8-Bit-Port-Adressen 40h bis 43h angesprochen. An seinem Beispiel wollen wir unsere Kenntnisse zur Programmierung eines Controllers vertiefen.

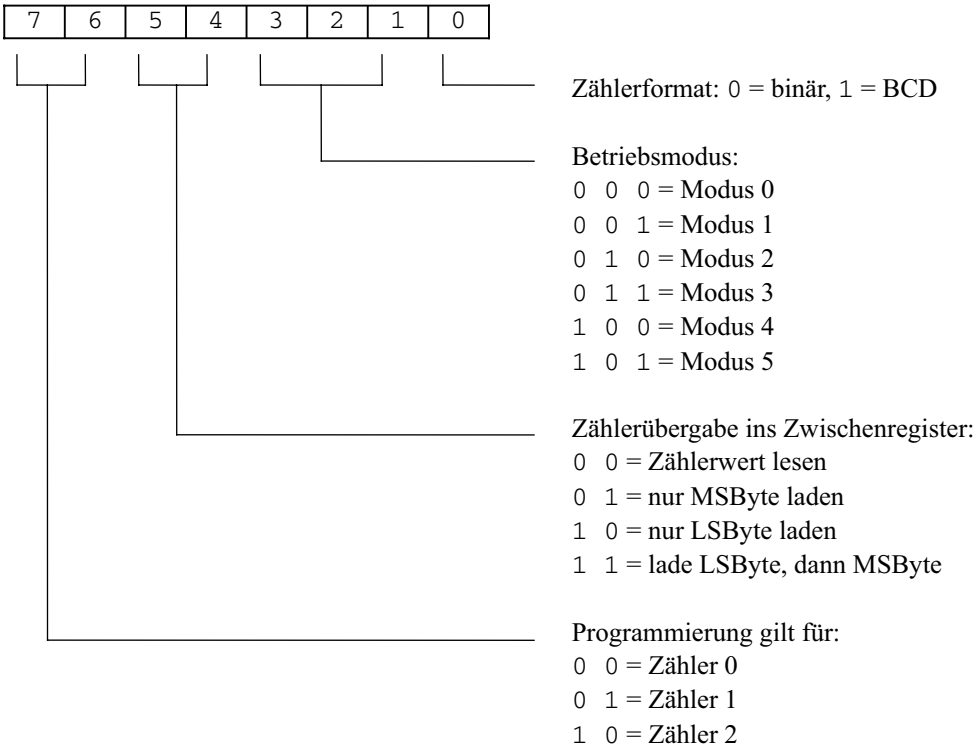
Der Baustein verfügt über insgesamt drei Zähler, die unabhängig voneinander arbeiten können. Da die Zähler 1 und 2 für interne Zwecke des PC benötigt werden, ist es ratsam, nur den Zähler 0 für Übungszwecke zu benutzen. Der Zähler 0 ist derjenige, der standardmäßig 18,2-mal pro Sekunde einen Interrupt 08 auslöst, welcher allerdings nichts bewirkt, außer einem kurzen Eintauchen in die Interrupt-Service-Routine mit sofortiger Rückkehr zum unterbrochenen Programm.

Alle drei Zähler sind 16 Bit breit. Sie sind mit jeweils einem Zwischenregister verbunden, das über eine Portadresse vom Programmierer mit einem Zählwert belegt werden kann.



Zusätzlich existiert ein Steuerwort-Register (Port 43(hex)), dort wird der gewünschte Betriebsmodus eingestellt.

Steuerwort-Register (Port 43h) des Timerbausteins:



Die Ports 40h bis 42h nehmen die Initialisierungswerte (Zählerwerte) des jeweiligen Timerkanals (Zähler 0 bis 2) auf. Am Takteingang des Zählerbausteins liegt eine Grundfrequenz von 1,19318 MHz an. Das ergibt einen Taktzyklus von 1/1,19318 MHz = 838 ns. Soll der Timer nun mit einer bestimmten Frequenz ein Signal erzeugen, muss der entsprechende Wert zunächst in das Zählerregister geladen werden:

Wert = 1,19318 / gewünschte Frequenz [MHz]

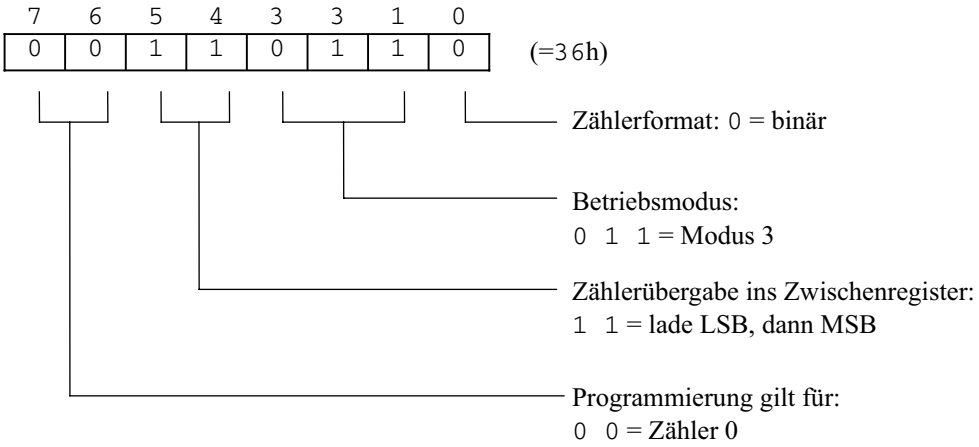
■ **Beispiele**

gewünschte Frequenz = 1,19318 MHz	Wert = 1 (0001 (hex))
gewünschte Frequenz = 18,2 Hz	Wert = 65535 (FFFF (hex))
gewünschte Frequenz = 100 Hz	Wert =11932 (2E9C (hex))
gewünschte Frequenz = 200 Hz	Wert = 5966 (174E (hex)) ■

Am einfachsten ist es nun, den so errechneten Wert binär zu übergeben (Bit 0 des Steuerwortregisters auf 0). Von den fünf möglichen Betriebsmodi des Timers ist der Modus 3 der gebräuchlichste: Er erzeugt am Timerausgang mit jedem Takt ein Rechtecksignal und löst bei Freigabe des entsprechenden Bits des Interrupt-Controllers einen Interrupt aus. Zum Laden (Beschreiben) des Zwischenregisters bestehen zwei prinzipielle Möglichkeiten: entweder

LSByte und MSByte getrennt laden (Bit 5 und 4 zunächst auf `0 1`, danach auf `1 0`) oder beide auf `1` setzen. Dann wird automatisch erst das LSByte, anschließend das MSByte übernommen.

Dieser Modus ist der bequemere. Programmiert werden soll schließlich der Zähler 0 (Bit 7 und 6 auf `0 0`). Insgesamt ergibt sich daraus folgendes Steuerwort:



Also muss an Port 43h der Wert `36h` übergeben werden. Soll nun beispielsweise der Timer-Kanal 0 200 Signale pro Sekunde generieren, müssen nacheinander die Werte `4E(hex)` (LSByte) und `17(hex)` (MSByte) an den Port 40(hex) übergeben werden, damit das Zwischenregister 0 mit dem Wert `174E(hex)` geladen wird.

Aufgabe

Schreiben Sie mit Hilfe des DEBUG ein kleines Programm, das den Timer (Kanal 0) auf eine Interrupt-Frequenz von 100 Hz bringt. Vor der Programmierung des Timer-Bausteins sollten Sie alle Interrupts verbieten, anschließend wieder erlauben.

Nennen Sie das Programm `TINT.COM` und schreiben Sie es auf die Festplatte.

Starten Sie zunächst `MULTI.COM`, dann `TINT.COM` und anschließend wieder `MULTI.COM`. Was beobachten Sie? Achtung: Die Starts müssen aus der gleichen Konsole (Eingabeaufforderung) erfolgen, weil sonst die Umstellung der Timer-Interrupt-Frequenz nicht mehr mehr wirksam ist. Auch ein Start per Doppel-Mausklick von der Desktop-Oberfläche zeigt nicht den gewünchten Effekt!

Wiederholen Sie den Vorgang mit 200 Hz und evtl. noch schneller.

7 Symbolische Assembler

Das Beispielprogramm `MULTI.COM` macht deutlich, dass die Erstellung größerer Programme mit dem DEBUG-Assembler eine mühsame Angelegenheit ist. Vor allem die direkte Angabe von Adressen bei Daten und v. a. Sprungbefehlen bereitet große Probleme, denn möchte man das Programm auch nur geringfügig verändern, ändern sich die Adressen, so dass man in vielen Fällen das Programm neu eingeben muss.

Für die professionelle Assemblerprogrammierung wird man daher auf einen kommerziellen Assembler, wie den TASM von Borland oder den MASM von Microsoft, zurückgreifen. Diese Assembler arbeiten wie Hochsprachen mit symbolischen Adressen, mit Sprungmarken und Variablen. Allerdings ist auch ein gewisser „Wasserkopf“ notwendig: Segmente müssen definiert werden, absolute Anfangsadressen müssen festgelegt, Variablen und Konstanten definiert werden. Dazu dienen spezielle Anweisungen, die keinem Maschinenbefehl entsprechen, die so genannten Assemblerdirektiven. Ein Beispiel:

```
ORG 100h
```

bedeutet, dass die folgenden Assemblerbefehle oder Variablen ab der Offset-Adresse 0100(hex) abzulegen sind. Der MASM von Microsoft ist ein 16-Bit-Assembler für den Real-Mode und mittlerweile frei verfügbar, z. B. auf unserer Buch-Webseite. Dort finden Sie außerdem eine die Batch-Datei „ALC.BAT“ zum Übersetzen und Linken von MASM-Assembler-Programmen. Das Hilfsprogramm „EASY.ASM“ erleichtert Ihnen den Einstieg in die symbolsche Assembler-Programmierung (s. auch die Anleitung dazu auf der Buch-Webseite).

Maschinenprogramme werden mit Hilfe eines Assemblers in ähnlicher Weise entwickelt wie Hochsprachenprogramme. Der Entwicklungsweg lautet:

EDITOR ⇒ ASSEMBLER ⇒ LINKER ⇒ PROGRAMMSTART

Die einzelnen Schritte sind entweder zu einer integrierten Entwicklungsumgebung zusammengefasst, wie Sie es von C/C++ her kennen, oder verlaufen getrennt wie beim „alten“ MASM.

Entwicklungsschritt	Werkzeug	Funktion
Editieren	Editor	Das sog. Quellprogramm `NAME.ASM` wird mit einem beliebigen Editor erstellt.
Assemblieren	Assembler	Das Quellprogramm in der Syntax des verwendeten Assemblers wird in die Maschinensprache übersetzt. Allerdings werden die Speicheradressen noch nicht endgültig festgelegt (meist vorläufige Anfangsadresse 0), sie sind verschiebbar (-> `NAME.OBJ`).
Binden	Linker	Eine oder mehrerer Obj-Dateien werden zu einem unter dem jeweiligen Betriebsystem ablauffähigen Programm mit in sich konsistenten Adressen zusammengebunden. Als Ganzes kann das Programm allerdings in beliebige Speichersegmente geladen werden (-> `NAME.EXE` oder `NAME.COM`).
Laden	Kommando-interpreter	Unter Windows in der Regel `CMD.COM`. Lädt das Programm von der Festplatte und weist ihm freie Speichersegmente zu. Startet das Programm.

Außerdem existieren Hilfen (Tools) zur Fehlerbeseitigung, die sog. Debugger.

Kernstück der Programmentwicklung ist jedoch der Assembler. Er führt vor der Übersetzung eine Syntaxanalyse durch und gibt ggf. Fehlermeldungen aus.

Wir wollen hier nicht näher auf symbolische Assembler eingehen, verweisen jedoch nochmals auf unsere Buch-Webseite, mit vielen Dokumenten und Programmen zur Assembler-Programmierung in der 16-Bit-Welt im Real Mode. Moderne Assembler wie der MASM32 ermöglichen Assembler-Programmierung in der 32-Bit-Welt des Protected Modes. Diesen verwenden wir hier nicht, sondern nutzen stattdessen die Fähigkeit von C++-Compilern, 32-Bit-Assembler-Befehle auf sehr komfortable Weise abzusetzen (Inline Assembler).

Mehr zum Thema symbolische Assembler finden Sie auf unserer Buch-Webseite.

8 PC-Technik in der 32-Bit-Welt

Wir wechseln nun wieder in die 32-Bit-Welt des Protected Mode, d. h. wir benutzen wieder unseren C/C++-Compiler. Während im Band 1 typische Anwender-Programme im Vordergrund stehen, bewegen wir uns hier etwas näher am System, soweit die neueren Windows-Versionen das erlauben.

8.1 Die Programmierung von Kommandos in C/C++

Kommandos sind Programme, die, im Gegensatz zu interaktiven Programmen, ihre Parameter (Eingabewerte) direkt beim Start, meist über die Shell (Eingabeaufforderung), erhalten.

■ **Beispiel:**

*calculate 13.4 * 24.89*

Das Kommando(-Programm) heißt *calculate* und erhält die Parameter *13.4*, * und *24.89*, jeweils durch mindestens ein Leerzeichen (blanks) getrennt.

■

In Kap. 4.2 wurde der prinzipielle Mechanismus der Parameterübergabe bei Kommandos behandelt. In C gestaltet sich der Vorgang sehr einfach. Die Funktion *main()* wird wie ein Unterprogramm behandelt, d. h. die Parameter werden „von oben" über die Parameter-Klammer übergeben. „Oben" ist meistens die Shell, kann aber auch ein anderes Programm sein, dass Sie evtl. selbst in C geschrieben haben (→ s. Kap. 8.2). Als Formal-Parameter definieren wir zunächst eine **int**-Variable (***int** argc*), die automatisch die Anzahl der Parameter übergeben bekommt. Zweiter Parameter ist ein zweidimensionaler String (***char** *argv[]* oder ***char** **argv*). Dort finden wir alle Parameter als Strings, auch wenn es sich um Zahlen handelt. Diese müssen dann im „Inneren" des Programms in den gewünschten Datentyp umgewandelt werden. Dazu stehen Standardfunktionen wie *atoi()* und *atof()* zur Verfügung. Übrigens ist die Anzahl der Parameter immer um eins höher als gedacht, denn der erste Parameter ist immer der Pfadname des Programms selbst. Also hätte *argc* in unserem obigen Beispiel den Wert 4. Die Bezeichnungen *argc* und *argv* sind nicht zwingend aber üblich.

Die „Überschrift" der *main*-Funktion sollte also folgendes Aussehen haben:

int main(int argc, char *argv[]) // *main*-Funktion mit Parameterliste

Wir machen uns diesen Mechanismus wieder an einem Beispiel klar. Nachstehend also ein kurzes C/C++-Programm, das die wichtigsten Regeln der Kommando-Erstellung enthält:

■ **Beispiel:**

```
// Programm parameter
// Dieses Programm hat Kommandostuktur
// Es gibt den Programmnamen und die beiden
// Uebergabeparameter aus
// Start aus der Konsole (Eingabeaufforderung)
// Falls nicht genau 2 Parameter uebergeben werden,
// erfolgt eine Fehlermeldung
#include <iostream.h>
#include <stdio.h>
```

```
using namespace std;
int main(int argc,  char *argv[])
{
 if(argc !=  3)
 {
   cerr << wrong number of parameters << endl;
   return  -1;
 }
  cout << Programmname:  << argv[0] << endl;
  cout << 1. Parameter:  << argv[1] << endl;
  cout << 2. Parameter:  << argv[2] << endl;
  getchar();
  return 1;
}
```

■

So wandelt man ggf. Parameter-Strings in numerische Datentypen um:

■ **Beispiel:**

```
int intzahl;
float floatwert;
intzahl = atoi(argv[1]);
floatwert = atof(argv[2]);
```

■

Diese Art der Parameterübergabe an Progamme ist unabhängig vom Betriebssystem und funktioniert auch mit 16-Bit-C-Compilern. Kommandos haben gegenüber interaktiven Programmen den Vorteil, dass sie Prozedur-fähig sind (→ s. Kap 9).

8.2 Parallele Prozesse

C/C++ bietet die Möglichkeit, aus einem Programm heraus ein anderes oder mehrere andere zu starten, die wiederum weitere Programme starten können.. Programm, die gerade ausgeführt werden, nennt man *Prozesse.* Es kann also eine regelrechte Prozesslawine erzeugt werden. Man spricht von Eltern- und Kindprozessen. Ermöglicht wird diese Technik duch die *spawn*-Funktionsfamilie. Es gibt verschiedene Versionen von *spawn()*, wobei die Version *spawnlp()* die gebräuchlichste ist. Der Prototyp liegt in der *process.h* und sieht so aus :

int spawnlp(int mode, char *fname, char *arg0, .., char *argN, NULL)

Der Rückgabewert von *spawnlp()* ist im Fehlerfall -1, sonst 0.

Als Übergabeparameter werden erwartet:

mode: P_WAIT (Der Elternprozess wartet auf die Beendigung des Kindprozesses)
P_NOWAIT (Eltern- und Kindprozess werden parallel ausgeführt)
P_OVERLAY (Der Elternprozess wird beendet)

***fname**: Name des zu startenden Programms (Kindprozess). Die Endung „lp“ bei der *spawn*-Funktion bewirkt, das das Programm zunächst im aktuellen Verzeichnis, dann im Pfad (PATH) gesucht wird.

***arg0**: Dummy-String (beliebig), der später mit dem Pfadnamen des Elternprogramms überschrieben wird.

***arg1 … *argN**: Die Übergabeparameter an den Kindprozess (alles Strings).

NULL: Parameter-Abschluss-Kennung. In jedem Fall zwingend.

■ **Beispiel:**

Das folgende Beispiel demonstriert den Aufruf des Kindprogramms *dispatcher.exe* durch das Elternprogramm *command.exe*. Übergeben werden die (String-)Parameter "abcdefghij" und "25". Gestartet wird der Kindprozess in der *NO_WAIT*-Variante, d. h. *command* und *dispatcher* laufen parallel, sie teilen sich den Prozessor. Der Kindprozess gibt alle Parameter, inclusive des Programmnamens (Parameter 0), aus. Der Zahlenstring (Parameter 2) wird zuvor mittels der *atoi*-Funktion in eine Integerzahl umgewandelt. Die „Überschrift" (Kopfzeile) von *dispatcher.exe* entspricht dem, was wir im vorherigen Kapitel (→ s. Kap. 8.1) über die Progammierung von Kommandos gelernt haben.

```
//------------------------------------------------------------//
Elternprogramm "command" fuer Windows
//------------------------------------------------------------
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
using namespace std;
//------------------------------------------------------------

int main(void)
{
  int i, retval;
  char p1[256] = "abcdefghij";
  char p2[2] = "25";
  clrscr(); // nur unter Windows
  cout << "Elternprozess command" << endl;
  cout << "---------------------" << endl;
  retval = spawnlp(P_NOWAIT, "dispatcher.exe", "dummy", p1,
           p2, NULL);
  if(retval != -1)
    cout << endl << "+++ Aufruf von dispatcher.exe ok +++"
         << endl;
  else
    cout << endl
         << "+++ Aufruf von dispatcher.exe fehlerhaft +++"
         << endl;
  cout << endl << "hit <Enter>" << endl;
  getchar();
  cout << endl << endl << "Elternprozess command Ende"
       << endl << endl;
  return 0;
}

//------------------------------------------------------------
```

```
//-------------------------------------------------------------
// Kindprogramm "dispatcher" fuer Windows
//-------------------------------------------------------------
#include <iostream.h>
#include <stdio.h>
#include <process.h>
using namespace std;
//-------------------------------------------------------------

int main(int argc, char* argv[])
{
  int i, p2;
  char *p0, *p1;
  cout << "Kindprozess dispatcher" << endl;
  cout << "----------------------" << endl;
  if(argc != 3)
  {
    cout << endl << "*** Falsche Anzahl der Parameter ***"
         << endl;
    return 0; // Fehlerausgang
  }
  p0 = argv[0];
  p1 = argv[1];
  p2 = atoi(argv[2]);
  cout << endl << "Parameter";
  cout << endl << "---------" << endl;
  cout << "0. Parameter (Programmname des Elternprozesses: "
       << endl << p0 << endl;
  cout << "1. Parameter: " << p1 << endl;
  cout << "2. Parameter: " << p2 << endl;
  cout << endl << endl << "Kindprozess dispatcher Ende"
       << endl << endl;
  return 0; // alles ok
}
//-------------------------------------------------------------
```

Die Ausgabe der beiden Prozesse in der gleichen Konsole erscheint etwas wirr:

```
Elternprozess command
---------------------

+++ Aufruf von dispatcher.exe ok +++

hit <Enter>
Kindprozess dispatcher
----------------------

Parameter
---------
0. Parameter (Programmname des Elternprozesses:
D:\spektro\Testprogramme\dispatcher.exe
```

```
1. Parameter: abcdefghij
2. Parameter: 25

Kindprozess dispatcher Ende

Elternprozess command Ende
```

Der Grund für die etwas wirre Ausgabe-Reihenfolge liegt in der gleichzeitigen Ausführung der beiden Prozesse. Ändern Sie im Programm *command.exe* den Parameter *P_NOWAIT* in *P_WAIT* um, so erscheint alles geordnet.

■

In der Praxis bestimmt das zu lösende Problem die Wahl zwischen *P_NOWAIT*, *P_WAIT* oder *P_OVERLAY*. Die Eigenschaft von *P_NOWAIT* verdanken wir der Multitasking-Fähigkeit unseres Betriebssystems. Man kann unter Windows sogar 16-Bit-Programme, z. B. mit DEBUG erzeugte COM-Assembler-Programme, aus einer 32-Bit-Umgebung heraus starten. Diese laufen dann in einer Multitasking-Umgebung im Virtual-Mode, der den Real-Mode emuliert. Sogar die Übergabe von Parametern ist möglich, wenn wir auf den *Program Segment Prefix* (PSP) (→ s. Kap 4.2) zugreifen.

■ **Beispiel:**

Das 32-Bit-Elternprogramm *call16bit.exe* ruft die drei 16-Bit-Kindprogramme *stars.com*, t*int.com* und *multi.com* auf, die Sie alle auf unserer Buch-Webseite finden. Diesmal erfolgt der Aufruf im *P_WAIT*-Modus (also keine parallelen sondern sequentielle Prozesse), was aber keineswegs zwingend ist. s*tars* liest den Parameterstring „Hallo 16-Bit-Welt“ aus dem PSP (→ s. Kap 4.2) und gibt ihn zeichenweise aus. Das umfangreiche Programm *multi* ist uns noch von früher bekannt (→ s. Kap 6.2). Es nutzt sogar einen Hardwareinterrupt (Timer) um Zeichen auszugeben. Der Timer wird zuvor von *tint* auf 100 Hz gesetzt.

```
//-------------------------------------------------------------//
Elternprogramm "call16bit" fuer Windows
//-------------------------------------------------------------
#include <iostream.h>
#include <conio.h> //
#include <process.h>
using namespace std;
//-------------------------------------------------------------

int main(void)
{
  int err;
  clrscr();
  cout << "call16bit" << endl;
  cout << "---------" << endl;
  cout << endl << "Zum Beenden <Enter> druecken"
       << endl << endl;
  err = spawnlp(P_WAIT, "stars.com", "dummy",
                "Hallo 16-Bit-Welt", NULL);
```

```
  cout << endl << "Error-Code von stars = " << err
       << endl << endl;
  err = spawnlp(P_WAIT, "tint.com", "dummy", NULL);
  err = spawnlp(P_WAIT, "multi.com", "dummy", NULL);
  return err;
}
//---------------------------------------------------------
```

■

Offenbar ist es mit dem Trick, ein 16-Bit-Programm zu starten, möglich, sogar unter Windows XP, auf Hard- und Software-Interrupts und sogar auf die Hardware selbst zu zugreifen. Auf diese etwas exotische Weise lassen sich also einfache Hardware-Treiber schreiben. Leider können Parameter nur von „oben" nach „unten" übergeben werden, nicht umgekehrt. Das ist eine starke Einschränkung. Windows besitzt im Gegensatz zu UNIX/Linux (→ s. Kap. 10) keine allgemein üblichen standardmäßigen Interprozess-Kommunikations-Mechanismen wie *Pipes*, *Semaphore,* und *memory mapped files*. Trotzdem lohnt es sich, bei Bedarf solche Begriffe in Zusammenhang mit Windows zu „googeln", denn einige kompetente Software-Entwickler haben Lösungen gefunden, die sie selbst- und kostenlos Interessenten zum Download zur Verfügung stellen. Mehr zu anderen *spawn*-Varianten und genaueren Fehlerhinweisen finden Sie in der weiterführenden Literatur über C/C++ oder im Internet.

8.3 Dynamic Link Libaries (DLL)

Unter dem Betriebssystem Windows (alle Varianten) wird das DLL-Konzept benutzt, um Treiber zu erstellen. Ein Zugriff auf die Hardware ist allerdings ohne weiteres nur unter den Varianten 95, 98 möglich. Das DLL-Konzept kann jedoch auch benutzt werden, um allgemein zugängliche Funktions-Bibliotheken zu erstellen. DLL werden in der Regel, im Gegensatz zu „normalen" Bibliotheken erst während der Laufzeit eines Programms, bei der ersten Verwendung, eingebunden. Ein größer Vorteil besteht darin, dass DLL sehr leicht von Programmen, die in einer anderen Sprache (Pascal, LabVIEW, …) geschrieben sind, eingebunden werden können. Oft erhalten Sie bei Zusatz-Hardware den Treiber in Form einer DLL mitgeliefert.

Eine DLL wird beim Aufruf der Reihe nach in folgenden Verzeichnissen gesucht:
1. im Verzeichnis der Anwendung (i. A. .EXE) selbst
2. im aktuellen Verzeichnis (working directory)
3. im Windows-System-Verzeichnis (Befehl: GetSystemDirectory)
4. im Windows-Verzeichnis (Befehl: GetWindowsDirectory)
5. in den Verzeichnissen, die in der Umgebungs-Variablen *PATH* aufgeführt sind

Eine DLL ist eine Sammlung von Funktionen (functions) von denen einige nach außen hin „sichtbar" sind (Exportfunktionen), einige nur intern verwendet werden. Letztere sind in der Regel Unterfunktionen der Exportfunktionen.

Leider gibt es beim Erstellen von Dynamic Link Libraries (DLL) Unterschiede zwischen den verschiedenen C++-Compilern. Wir zeigen im Folgenden die DLL-Erstellung bei Verwendung des Borland C++-Builders.

■ **Beispiel: Grundgerüst einer DLL:**

So sieht der Borland-C++-Overhead beim Erstellen einer DLL mit einer Export-Funktion (hier: **double** fourmax()) aus:

```
//---------------------------------------------------------
#include <windows.h>
#pragma hdrstop
//---------------------------------------------------------
int WINAPI DllEntryPoint(HINSTANCE hinst,
                         unsigned long reason, void*)
{
        return 1;
}
//---------------------------------------------------------
extern "C" __declspec(dllexport) double fourmax(short n,
                                 double *y, double noise)
  {
    .
    .
    .
 }
```

■

■ **Beispiel: Erstellen eines ausführbaren Programms mit DLL:**

So sieht der Borland-C++-Overhead beim Erstellen einer EXE mit einer DLL-Import-Funktion (hier: **double** fourmax()) aus:

```
// Import einer DLL
extern "C" __declspec(dllimport) double fourmax(short n,
                                 double * vek, double noise);

int main(void)
{
   .
   .
   .
}
```

■

Erstellen eines DLL-Projekts mit dem Borland-C++-Builder:

Builder staten

Datei → neu

DLL-Experte auswählen → <ok>

Entweder Quelltext eingeben

oder

Alles löschen (Bearbeiten → alles markieren → löschen)

Datei → öffnen (Quelltext aus anderer Datei holen)

Bearbeiten → alles markieren

<Strg> + <C>

auf Reiter von „Unit1.cpp“ klicken

<Strg> + <V>

auf 2. Reiter gehen, rechte Maustaste drücken → Seite schließen

<F9> (Programm übersetzen) → <ok>
Es entsteht eine .DLL- und eine .LIB-Datei.

Eigenen Dateinamen festlegen:
Datei → Projekt speichern unter
Neues Verzeichnis anlegen
Quelltextnamen festlegen
Projektnamen festlegen
(Die beiden letzten Schritte erfolgen automatisch in dieser Reihenfolge)

EXE-Programm als Projekt erzeugen:
Projekt anlegen analog zur DLL
Zusätzlich (falls das Programm eine DLL – Funktion aufruft):
Projekt → dem Projekt hinzufügen
.LIB-Datei der zu importierenden DLL suchen → <öffnen>

Andere Compiler mit einer integrierten Oberfläche erwarten ähnliche Schritte.

■ **Beispiel für eine DLL und deren Import in ein Testprogramm:**
Die DLL-Funktion *fourmax()* berechnet den Schwerpunkt einer Reihe von Messwerten.

```
//-------------------------------------------------------------
// Datei: dpsd_pos.cpp
// Zieldatei: DLL
// Ermittelt den Profilschwerpunkt einer Zeile nach der
// Fourier-Methode (Shift Theorem).
// Uergabeparameter der Aufruffunktion fourmax():
// short n      : Anzahl der Werte pro Zeile
// double *y    : Zeiger auf die Intensitaetswerte der Zeile
// double noise : Rauschlevel, wird von jedem Wert abgezogen,
// ggf. = 0.0
// Return-Wert  : Position des peaks bzw. Schwerpunkts
// (double)
//-------------------------------------------------------------
#include <math.h>
#include <stdlib.h>
#include <windows.h>
using namespace std;
//-------------------------------------------------------------
int WINAPI DllEntryPoint(HINSTANCE hinst,
                         unsigned long reason, void*)
{
        return 1;
}
```

```
//-------------------------------------------------------------
extern "C" __declspec(dllexport) double fourmax(short n,
                                 double *y, double noise)
  {
    short data, i;
    double suma, sumb;
    double pos;
    double con, fak, c;

    // Variableninitialisierungen
    data = n;
    con = 2. * 3.14159265358979 / data; // oder M_PI
    suma = sumb = 0.0;

    // Berechnung der 1. Fourier-Koeffizienten
    for(i = 0; i < data; i++)
    {
      *y = *y - noise;
      if(*y <= 0.0) *y = 0.0;
      suma = suma + *y * cos(i * con);
      sumb = sumb + *y * sin(i * con);
      y++;
    }
    fak = data / (2.0 * 3.14159265358979); // oder M_PI

    // Phasenkorrektur
    if((sumb > 0.0) && (suma > 0.0)) c = 0.0;
    if(suma < 0.0) c = data / 2.;
    if((sumb < 0.0) && (suma > 0.0)) c = (double) data;
    if(suma == 0.0)
    {
      if(sumb > 0.0) pos = data / 4.0;
      if(sumb < 0.0) pos = 3.0 * data / 4.0;
    }
    else
      pos = fak * atan(sumb / suma) + c;

    // Rueckgabe des Ergebnis
    return pos;
  }
```

Eine mathematische Ableitung des Verfahrens finden Sie in:

Weißhaar, E., Küveler, G., Bianda, M.: Schnelle und genaue Methode zur Schwerpunktfindung in Messreihen. Photonik 4-2003 (auch als pdf-Datei auf unserer Buch-Webseite).

Das nachstehende Programm, testet die obige DLL-Funktion *fourmax()*. Diese Funktion ermittelt, wie gesagt, nach einer speziellen, auf der Fourier-Theorie beruhenden, Methode den Schwerpunkt einer Messreihe. Wir erzeugen Sie dazu eine halbe positive Sinus-Welle von 0 bis Π in 100 diskreten Werten. Diese stellen wir in ein entsprechendes double-Feld und rufen die importierte DLL-Funktion *fourmax()* mit den entsprechenden Übergabe-Parametern auf. Die Variable *noise* setzen wir auf 0.0. *fourmax()* liefert den ermittelten Schwerpunkt (X-Wert)

zurück. Dieser Wert wird ausgegeben. Welchen Wert erwarten wir, falls die getestete Funktion korrekt arbeitet?

```
//-------------------------------------------------------------
// Programm dll_test.cpp
// Importiert die Funktion fourmax() aus einer DLL
// Erzeugt eine halbe Sinus-Welle von 0 bis PI (180 Grad)
// Uebergibt diese an die Schwerpunkt-Such-Funktion fourmax()
// Diese liefert den Schwerpunkt der Kurve zurück
//-------------------------------------------------------------
#include <math.h>
#include <iostream.h>
#include <stdlib.h>
using namespace std;
//-------------------------------------------------------------
// Import einer DLL
extern "C" __declspec(dllimport) double fourmax(short n,
                                 double * vek, double noise);

int main(void)
{
   double pi, step, next, vek[100], sp, noise;
   short i;

   pi = 3.14159265358979; // oder M_PI
   step = pi / 99.;
   for(i = 0; i <= 99; i++)
     {
       next = i * step;
       vek[i] = sin(next);
     }
   noise = 0.;
   sp = fourmax(100, vek, noise); // Aufruf der DLL
   cout << "Schwerpunkt = " << sp;
   getchar();
   return 0;
}
//-------------------------------------------------------------
```

■

8.4 Assembler und C

Zunehmend werden einige Hochsprachen zu einer echten Assembler-Alternative. Vor allem Basic, Pascal und C lassen hardwarenahe Programmierung zu. Kaum etwas, was in Assembler machbar ist, ließe sich nicht auch in diesen Hochsprachen realisieren. Für Assembler spricht noch der Geschwindigkeitsvorteil und der kompaktere Maschinencode. Ein Kompromiss kann darin bestehen, Assemblercode in Hochsprachenprogramme einzufügen. Am Beispiel von C++ soll diese Möglichkeit kurz aufgezeigt werden. Dazu dient das Schlüsselwort **asm**, das vor jedem Assemblerbefehl einzufügen ist. Der C++-Compiler leitet die entsprechend gekennzeichneten Befehle an den Assembler weiter, der Bestandteil der Entwicklungsumgebung ist (Inline-Assembler). Achten Sie darauf, Ihre Entwicklungsumgebung (Borland C++-Builder,

Microsoft Visual C++, ...) vollständig zu installieren. **asm** ist ein Schlüsselwort für C++, für reine C-Compiler ist es optional.

■ **Beispiel 1:** Zusammenwirken von C++ und Assembler

```
// Programm c_asm_1.cpp
#include <iostream.h>
#include <stdio.h>
using namespace std;

int main(void)
{
  short m1 = 0x1000, m2 = 0x2000, m3;
  asm  mov ax,m1;
  asm  mov bx,m2;
  asm  add ax,bx;
  asm  mov m3,ax;
  cout << hex << (int)m3 << endl;
  getchar();
  return 0;
}
```

Ausgabe:
3000 ■

Die Probleme des Overheads, die symbolisch Assembler bereiten, entfallen. Die Variablen-Deklaration folgt einfach den üblichen C/C++-Regeln. Man muss jedoch, wie auch sonst bei Assembler, auf die strenge Datentyp-Verträglichkeit achten:

Datentyp **char** passt zu 8-Bit-Halbregistern (**AL, AH, BL, BH, CL, CH, DL, DH**),

Datentyp **short** passt zu 16-Bit-Registern (**AX, BX, CX, DX**),

Datentyp **long** passt zu 32-Bit-Registern (**EAX, EBX, ECX, EDX**).

Letzteres ist natürlich nur möglich, wenn man mit einem entsprechenden Compiler unter einem 32-Bit-Betriebssystem (Windows 2000, Windows XP, ...) entwickelt, so dass der Prozessor im Protected Mode betrieben wird. Somit ist es uns erstmals möglich, auf die 32-Bit-Register moderner PC-Prozessoren, wie dem Pentium, zu zugreifen.

Statt **asm** vor jeden Assembler-Befehl zu setzen, ist es bei längeren Assembler-Befehlssequenzen praktischer, **_asm** zu verwenden. Die Sequenz muss dann allerdings mit {} geklammert werden.

■ **Beispiel 2:** Zugriff auf 8- und 32-Bit-Register

```
// Programm c_asm_2.cpp
#include <iostream.h>
#include <stdio.h>
using namespace std;
```

```
int main(void)
{
  char m1 = 0x10, m2 = 0x20, m3;
  long m4 = 500000, m5 = 300000, m6;
  _asm
     {
       mov al,m1;         // 8 Bit
       mov ah,m2;
       add al,ah;
       mov m3,al;
       mov ecx,m4;        // 32 Bit
       mov edx,m5;
       sub ecx,edx;
       mov m6,ecx;
     }
  cout << hex << (int)m3 << endl;
  cout << dec << m6 << endl;
  getchar();
  return 0;
}
```

Ausgabe:
30
200000 ■

Wir müssen noch beachten, dass das Flagregister im Protected Mode ein 32-Bit-Register ist. Folglich sprechen die Stack-Befehle *push* und *pop* sowie *pushf* und *popf* (→ s. Kap. 5.1.1) die Register EAX bis EDX sowie EF an. Das folgende Beispielprogramm gibt die ersten sechzehn Bit des Flagregisters nach der Operation SUB AX, BX (AX = AX – BX) aus.

■ **Beispiel:**

```
//-----------------------------------------------------------
// Programm asm_flag_1.cpp
// Ausgabe des Flagregisters nach einer Subtraktion AX - BX
// Eingabe: Dezimale Werte fuer AX und BX
// Laeuft unter jeder Windows-Version
//-----------------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <iostream.h>
using namespace std;
//-----------------------------------------------------------

void main(void)
{
 short a, b, n;
 long flag;
 const short maske = 1;
 char wort[16][12];
```

```
 strcpy(wort[0], "Carry.....");
 strcpy(wort[1], "Flag 1....");
 strcpy(wort[2], "Parity....");
 strcpy(wort[3], "Flag 3....");
 strcpy(wort[4], "Auxillary.");
 strcpy(wort[5], "Flag 5....");
 strcpy(wort[6], "Zero......");
 strcpy(wort[7], "Sign......");
 strcpy(wort[8], "T.........");
 strcpy(wort[9], "I.........");
 strcpy(wort[10], "D.........");
 strcpy(wort[11], "Overflow..");
 strcpy(wort[12], "PL........");
 strcpy(wort[13], "IO........");
 strcpy(wort[14], "NT........");
 strcpy(wort[15], "Flag 15...");

 cout << "Rechenoperation: AX - BX" << endl;
 cout << "AX [dezimal] >"; cin >> a;
 cout << "BX [dezimal] >"; cin >> b;
 cout << endl << endl;
 cout << "Inhalt des Flag-Registers" << endl;
 cout << "-------------------------" << endl;

 // Rechenoperation ausfuehren, Flag-Register speichern
 _asm
 {
   mov ax,a;
   mov bx,b;
   sub ax,bx;
   pushf;      // Mit diesem Trick kann man das
   pop eax;    // Stack-Register auslesen
   mov flag,eax;
 }

 // flag bitweise untersuchen (0 oder 1) und ausgeben
 for(n = 0; n < 16; n++)
 {
   cout << wort[n] << (flag & maske) << endl;
   flag = flag >> 1;
 }
 cout << endl << "<Enter> zum Beenden" << endl;
 getchar();
}
```

■

Selbstverständlich können alle Assembler-Befehle verwendet werden. Es gelten jedoch die üblichen Einschränkungen unter Multitasking-BS: Windows NT, 2000 und XP kontrollieren den Zugriff auf Speicher und Ports und unterbinden ihn im Falle von Schutzverletzungen. Unter Windows 98 hat man vollen Zugriff auf die Hardware. Natürlich mit der Konsequenz, Rechnerabstürze produzieren zu können

8.5 Aufgaben

Aufgabe 1

Schreiben Sie ein Programm *int2hex*, das eine übergebene int-Zahl (32 Bit) in der internen hexadezimalen 2er-Komplement-Darstellung ausgibt. Gestalten Sie das Programm wiederum als Kommando.

Aufgabe 2

Schreiben Sie ein Kommando-Programm *hex2int*, das eine übergebene 8-stellige hexadezimale 2er-Komplement-Zahl als dezimale int-Zahl (32 Bit) ausgibt.

Aufgabe 3

Schreiben Sie ein Kommando-Programm *float2hex*, das eine übergebene 8-stellige hexadezimale Zahl als dezimale float-Zahl ausgibt.

Tipp: Analysieren Sie zunächst das Programm *real_Hex* im Buch, Kap. 1.3

Aufgabe 4

Schreiben Sie ein kleines Programm, dass eine DLL-Funktion aufruft.
Legen Sie für beide Quelltexte nach dem obigen Schema Projekte an.
Inhalt der DLL: eine Funktion, die ein übergebenes Integer-Feld aufsteigend sortiert. Übergabe-Parameter sind:

1. ein Zeiger auf das int-Feld (int *)
2. die Feldlänge

Inhalt des aufrufenden Testprogramms (EXE): Per Zufallsgenerator (s*rand ()*, *rand ()*) sollen N Integer-Werte zwischen 0 und 99 erzeugt werden, die in einem entsprechenden Feld abzulegen sind. Die Anzahl der Werte N ist vom Benutzer einzugeben. N legt gleichzeitig die Länge des Feldes fest, das dynamisch (mit *new* oder *malloc()*) zu erzeugen ist. Ein Zeiger auf das Feld und dessen Länge ist der zu importierenden DLL zu übergeben. Das zurück erhaltene sortierte Feld ist in sinnvoller Weise, z. B. 10 Werte pro Zeile, auszugeben.

9 Technische Anwendungen mit dem PC

In Produktionsstätten und Laboratorien hat der PC den klassischen Prozessrechner mehr und mehr verdrängt. Moderne Vernetzungsmöglichkeiten (→ s. Teil II) ermöglichen weltweit verteilte Automatisierungs-Systeme.

9.1 Hardware Modelle

Die meisten technischen Anwendungen wie Messdaten-Erfassung, Steuerungen und Regelungen arbeiten heute in unterschiedlicher Weise unter Einbeziehung des PC. Aus Sicht des PC lassen sich zwei grundsätzliche Konzepte unterscheiden:

– das PC-interne Konzept

– das PC-externe Konzept

Das PC-interne Konzept

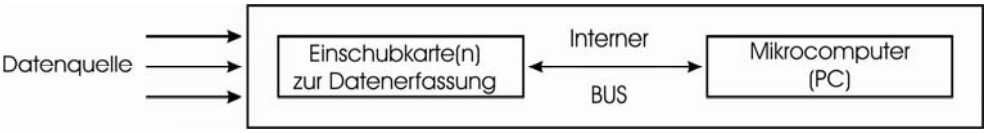
Dieses Konzept entspricht einer Einschubkarten-Lösung, d. h. für die diversen Aufgaben der Messdatenerfassung und Steuerung werden spezielle I/O-Karten mit dem internen Bus des PC verbunden. Dies geschieht durch Einschieben der Karte in einen freien Steckplatz (Slots) der Zentraleinheit.

Nachstehend finden Sie eine Übersicht über die Einschubkarten-Typen für die verschiedenen parallelen internen Bussysteme des PC, die sich im Laufe der Zeit vom 8-Bit-PC-Bus über den 16-Bit-ISA-Bus zum heute aktuellen 32- und 64-Bit-PCI-Bus entwickelt haben. Der interne Bus verbindet die Peripherie mit dem Chip-Satz des PC. Für viele technische Anwendungen genügen durchaus alte, ansonsten ausgemusterte PC, insbesondere dann, wenn Geld eine Rolle spielt. Daher seien auch die älteren Typen erwähnt:

– **PC**	
8-Bit-Karten:	nur noch ältere Modelle, heute meist „Bastelvorschlag“, evtl. Bausatz für älter PC.
– **ISA**	
ehemaliger Standard:	16 Bit, Systemressourcen wie Adress- und Speicherbereich, Interrupt-Nr. und DMA-Kanal über DIP-Schalter oder Jumper, bei neueren Karten z. T. per Installationsprogramm.
– **VLB**	
″Low-cost″ Erweiterung des ISA-Busses:	PC besaßen außer ISA-Steckplätzen typischerweise 3 VLB-Slots. Da nur als Übergangslösung gedacht, existieren keine Messdatenerfassungskarten für den VLB.
– **MCA**	
nicht ISA-kompatibel:	die größeren ISA-Karten passen nicht in einen MCA-Slot. Einschubkarten werden per Software automatisch konfiguriert.
– **EISA**	
32 Bit Erweiterung:	ISA bleibt kompatibel. Die Konfiguration von Einschubkarten erfolgt per Software (EISA Configuration Utility, ECU). Achtung: Die Konfiguration ist Slot-abhängig.

– **PCI**

Stand der Technik: seit 1992, 32 Bit, maximale Taktrate 33 MHz, relativ teuer, für viele Anwendungen würden 16-Bit-Karten ausreichen, jedoch haben heutige PC standardmäßig keine 16-Bit-Steckplätze mehr. Seit 2002: PCI 2.3, 64 Bit, maximale Taktrate 66 MHz.



PC-interne Lösung mit Einschubkarten

Am häufigsten findet man so genannte Multi-I/O-Karten, die typischerweise folgende Bausteine enthalten:

– parallele digitale Eingänge
– parallele digitale Ausgänge
– A/D-Wandler für analoge Eingabedaten
– D/A-Wandler für analoge Ausgabedaten
– Timerbausteine zur Signalerzeugung (Sägezahn, Rechteck) und zur Interrupterzeugung mit quarzgenauer Zeitbasis

Bekannte Firmen wie National Instruments, Burr-Brown, Keithley oder Meilhaus bieten eine immer breiter werdende Palette an Karten, auch für spezielle Probleme, an. Jede Einschubkarte wird in den I/O-Adressraum des PC eingeblendet, indem die Basis-Portadresse festgelegt wird. Dies geschah früher mittels DIP-Switches („Mäuseklavier“) oder Jumper. Bei neueren Karten erfolgt die Festlegung per Software. In der Regel hat der Hersteller bereits eine Adresse voreingestellt, die von den Standardbauteilen eines PC nicht genutzt wird. Bei mehreren Einschubkarten muss jede Karte einen freien Bereich im Adressraum zugewiesen bekommen, damit Buskonflikte vermieden werden. Jede Karte verfügt über eine Schnittstelle, die die Bussignale dekodiert und nur diejenigen akzeptiert, die einer Adresse im Adressbereich der Karte entspricht.

Prinzipiell kann einen Einschubkarte in jeden beliebigen (passenden) Slot eingesteckt werden. Allerdings sollte man darauf achten, dass sie möglichst weit vom Schaltnetzteil des PC entfernt sitzt, weil dieses typischerweise im kHz-Bereich Störsignale verbreitet. Auch zur Grafikkarte, die Synchronisationssignale für den Bildschirm erzeugt, sollte man Distanz halten. Dennoch lassen sich Störungen innerhalb eines PC-Gehäuses nie ganz vermeiden, so dass etwa A/D-Wandlungen mit höherer Genauigkeit als 12 Bit nur mit hohem Abschirmaufwand möglich sind.

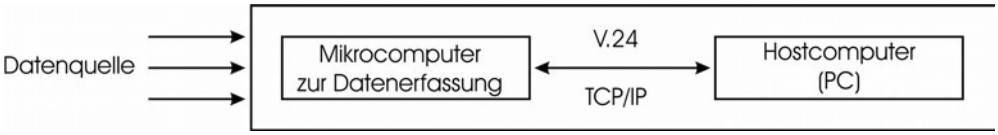
Hinter den einzelnen Portadressen stehen die Status-, Steuer- und Datenregister der diversen Bausteine der Karte. Die genaue Belegung entnimmt man dem Handbuch zu der jeweiligen Karte. In der Maschinen bzw. Assemblersprache des PC werden die Ports, je nach Zugriffrichtung, mit IN- oder OUT-Befehlen angesprochen. Auch von höheren Sprachen (BASIC, Pascal, C/C++, LabVIEW) aus ist ein Zugriff möglich. Ein direkter Portzugriff aus einem normalen Anwenderprogramm heraus wird jedoch nur noch von Windows 98 zugelassen. Bei NT, 2000 und XP benötigt man Treiber, welche die Restriktionen dieser Betriebssysteme berücksichti-

gen. Sie werden heute in aller Regel mit der erworbenen Einschubkarte, meist in Form von DLL (→ s. Kap. 8.3), mitgeliefert.

Die Einschubkarten-Lösung eignet sich besonders für kleine Messaufgaben. In diesem Fall ist das System sehr schnell, da die Datenerfassungs-Hardware direkt mit dem internen Bus des PC verbunden ist. Benötigt man mehrere Einschubkarten, stößt man schnell an Grenzen, weil zum einen die Anzahl der Steckplätze begrenzt ist, zum anderen, bei mehreren schnellen Prozessen, evtl. die Leistungsfähigkeit des PC nicht mehr ausreicht. Eine Lösung mit mehreren PCs verliert auf jeden Fall den (eventuellen) Vorteil der Preisgünstigkeit.

Das PC-externe Konzept

Beim externen Konzept wird die Feldebene (Sensoren, Geräte) völlig von der Leitebene getrennt. Die Daten werden von einem oder mehreren Mikrorechnern, oft 8- oder 16-Bit-Systeme weit unterhalb der PC-Klasse, erfasst, ggf. zwischengespeichert und auf Anforderung an den Hostrechner, meist ein PC, weitergegeben.

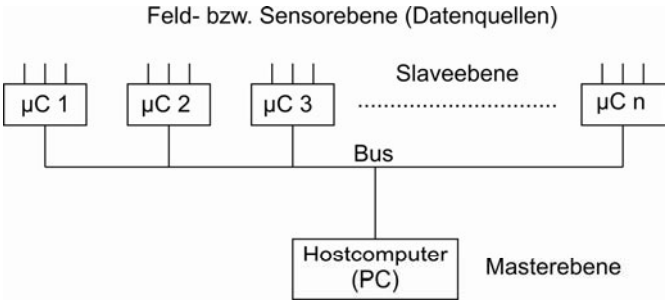


PC-externe Lösung mit separatem Feldrechner

Die Verbindung zwischen Erfassungscomputer und Host kann im einfachsten Fall über eine serielle V.24-Leitung realisiert werden. Zunehmend werden Ethernet-Verbindungen mit TCP/IP-Protokoll verwendet. Eine Variante stellen USB-Module dar. Sie sind sehr preisgünstig und eignen sich für kleinere Projekte.

Da ein „Feld"-Mikrocomputer häufig über sehr weitgehende Eigenintelligenz verfügt (er ist evtl. voll programmierbar), wird der Hostrechner (Host = „Gastgeber", in unserem Fall ein PC) stark entlastet. Damit ist es möglich, mehrere Erfassungsrechner eines Labors an nur einen PC zu koppeln.

Bei größeren Systemen ist ein Bus notwendig, der, falls gewünscht, auch die Kommunikation der Feldrechner untereinander ermöglicht.



PC-externe Lösung mit vielen Feldrechnern und Bus-Sytem

Bei Bedarf ist es leicht möglich, mehrere Hostcomputer in das System zu integrieren. Der Bus kann ein industrieller Feldbus (CAN-, Profi-, Interbus, …) oder, zunehmend populär weil schnell und preiswert, Ethernet mit TCP/IP sein, das sogar eine weltweite Verteilung der einzelnen Rechner zulässt.

Für beide Konzepte gibt es passende Anwendung. Eine sehr sorgfältige Planung unter Abwägung aller Kriterien ist notwendig, um die im Einzelfall günstigste Lösung zu finden:

PC-interne Lösung	PC-externe Lösung
Vorteile • Leicht zu realisieren. • Hohe Geschwindigkeit bei der Datenübertragung. • Kompakte Lösung. • Breites Angebot an Einschubkarten für nahezu jede Anwendung	**Vorteile** • Saubere Trennung der Verschiedenen Ebenen. • Nahezu unbegrenzt erweiterbar. • Erfassungscomputer können in der Nähe der Datenquelle platziert werden, der Hostcomputer ggf. weit entfernt. • Hostcomputer kann beliebig ausgetauscht werden, ohne die Erfassungsebene zu verändern. • Mehrere Hostcomputer sind möglich.
Nachteile • Begrenzte Kapazität bzw. Erweiterbarkeit. • Starke Hardwareabhängigkeit wegen fehlender Trennung der Ebenen.	**Nachteile** • Aufwendigere Planung. • Schwierigere Realisierung.

9.2 Prozeduren und Skripsprachen

Skriptsprachen (Prozedur-, Batchsprachen) eignen sich zur Automatisierung von bestimmten Abläufen im Rechner wie z. B. periodischen Daten-Sicherungen. Dabei werden häufig verschiedene Programme, die Kommando-Struktur (→ s. Kap. 8.1) besitzen, in der gewünschten Weise kombiniert aufgerufen. Es kann sich um Betriebssystem-Kommandos oder auch um selbst geschriebene Kommandos handeln. Skriptsprachen-Programme werden nicht kompiliert sondern zeilenweise oder bis zum nächsten Trennzeichen, meist Semikolon oder Zeilenende; interpretiert. Dazu bedarf es eines Interpreters, der Bestandteil des Betriebssystems sein kann (Batch-Prozeduren unter DOS oder die Shell-Skript-Sprache unter UNIX/Linux (→ s. Kap. 10)) oder der für einen speziellen Zweck erstellt wurde. Wie alle Systemprogramme werden Interpreter meist in C/C++ geschrieben. Ist der Interpreter Bestandteil des Betriebssystems, genügt zum Start einer Prozedur meist die Eingabe ihres Namens über die Text-Konsole (Eingabeaufforderung). Mitunter wird eine bestimmte Extension zum Dateinamen gefordert, z. B. *.BAT* im Falle von DOS. Gehört die Skriptsprache nicht zum Betriebssystem, wird der Interpreter meist durch Eingabe seines Namens, gefolgt von der auszuführenden Prozedur selbst, gestartet.

Da es in diesem Kapitel um technische Anwendungen geht, stellen wir eine Interpretersprache vor, die an unserer Fachhochschule im Rahmen einiger Diplomarbeiten entwickelt wurde. Sie läuft unter Windows NT, XP und Vista und ist die Steuerung von flexiblen Abläufen in Automatisierungs-Prozessen spezialisiert. Sie heißt *AMI* und auch die Prozeduren, die der Interpre-

ter ausführen soll, müssen die Extension *.AMI* besitzen. Nehmen wir an, eine Prozedur heiße *BSP01.AMI*, dann wird sie von der Konsole aus folgendermaßen gestartet:

>ami bsp01

Der Interpreter *AMI.EXE* ist zusammen mit allen Hilfsdaten, Beispielprozeduren und Dokumenten auf unserer Buch-Webseite zu finden. Eine Kurzdokumentation aller Sprachelemente bietet die Datei *AMI.HLP* Aufruf: *ami –h*).

AMI (Advanced Macro Interpreter) ist eine C-ähnliche Interpretersprache. Daneben orientiert sie sich an der UNIX-Shell (→ s. Kap. 10) soweit es die Übergabe von Parametern beim Aufruf ($0… $9) und einige Kommandos (*test*, *set*) angeht. Wie bei Batch-Prozeduren üblich, besteht die Möglichkeit, externe Programme zu starten. Bei AMI ist der Mechanismus des Parameter-Austauschs gegenüber den meisten Skriptsprachen sehr viel einfacher.

Grundlage jeder Programmiersprache sind die Datentypen und die Ablaufstrukturen. Wir stellen Sie in den folgenden Tabellen kurz vor. Alle Variablen müssen vor ihrer Verwendung deklariert werden. Dabei können Anfangswerte oder externe Parameter zugewiesen werden.

Datentyp	**Wertebereich**	**Deklarations-Beispiele**
int	-2147483648 bis +2147483647	int ia, ib=2, ic=$1;
float	-1.7e308 bis +1.7e308	float fa, fb=123.456, fc=$2;
time	00:00:00 bis 23:59:59	time ta=12:34:56, tb=now;
`date`	15:10:1582 bis 31:12:9999	date da, db = 21:12:1991;
string	Zeichenkette: max. 200 Zeichen	string sa="example", sb="$5";
complex	-1.7e308 bis +1.7e308	complex ca, cb=10 + i20;

AMI Datentypen

Die Verschachtelung der C-ähnlichen Strukturen ist beliebig. Gegenüber C fehlt nur case. Jede Anweisung wird mit einem Semikolon abgeschlossen. Zeilenumbrüche spielen keine Rolle. Kommentare stehen zwischen Rauten (# … #).

Struktur	**Art**	**Bemerkung**
Schleifen	while, do...while, for	Alle wie in C/C++
`Abbruch`	break	sofortiger Schleifenabbruch
Selektion	if … else if … else	Wie in C/C++
Programmausstieg	quit	vorzeitige Prozedurbeendigung

AMI Ablauf-Strukturen

Die Vielzahl von Standard-Funktionen kann hier nicht aufgezählt werden. Die folgende Tabelle liefert jedoch einen Überblick über die Funktionsarten.

Funktionsart	**Bemerkung**
Konsol-Ein/Ausgabe	*read* und *printf* (C-ähnlich) *idump* zur Ausgabe aller Variablen
Datei-Ein/Ausgabe	*fread* und *fwrite*: nur ASCII-Dateien
Mathematische Funktionen	"alle" vorhanden
Datentyp-Umwandlung	alle wichtigen vorhanden, v.a. string <> numerisch
Zeit-Funktion	*now* liefert die aktuelle Zeit, *today* das aktuelle Datum
Parameter-Management	*set* und *rot* arbeiten UNIX-ähnlich mit $0 … $9
Parameter- und Datei-Test	*test* prüft UNIX-ähnlich Parameter- und Datei-Eigenschaften

AMI Funktionsarten (Überblick)

Als mathematische Operatoren stehen +, -, * , /, % (Modulo) und ^ (Power) zur Verfügung. Bei zusammengesetzten Ausdrücken geht Punktrechnung vor Strichrechnung, ansonsten kann beliebig geklammert werden. Beim Datentyp *complex* sind nur einfache Ausdrücke erlaubt, bei *date*, *time* und *string* nur „sinnvolle" Operationen.

Auf Verstöße gegen die Syntax wird bei Ausführung oder einem vorherigen Syntax-Test mit einer ausführlichen Fehlermeldung (Fehler-Nummer, Zeile des Fehlers und Kommentar) reagiert.

AMI verfügt über eine Hilfedatei, eine Art Kurzhandbuch (Aufruf: *ami –h*). Es existieren diverse weitere Optionen. Eine AMI-Prozedur muss die Endung *.ami* besitzen. Der allgemeine Aufruf einer AMI-Prozedur von der Konsole aus lautet:

ami [-option] [macroname] [parameter1] [parameter2] ...

Auch wenn AMI fast (Felder sind nicht implementiert) wie eine „normale" Programmiersprache verwendet werden kann, liegt der Hauptnutzen in der vollen Automatisierung komplexer Abläufe, an deren Bearbeitung mehrere EXE- Programme beteiligt sind. Wie diese erzeugt wurden (C/C++, LabVIEW, …) spielt dabei keine prinzipielle Rolle. Verwendet man jedoch fremd erzeugte Programme, muss man sich natürlich deren Einschränkungen bezüglich der Parameter-Übergabe und der Interprozess-Kommunikation unterwerfen. Stößt der AMI-Interpreter auf ihm unbekannte Kommandos, z. B. DOS-Befehle, reicht er sie an die Shell zur Ausführung weiter. Wird ein Programm auf die folgende Weise mit der vorangestellten vorinitialisierten int-Variablen *ret_int* aufgerufen,

ret_int = childprocess [parameter1 ... parameter(n)];

so wird dieses in einer separaten Shell gestartet. Gleichzeitig wird ein Pipe-Client aktiviert, der es dem Tochter-Prozess ermöglicht, Ergebnisse, getrennt durch Leerzeichen, in einer Zeichenkette zurückzuliefern. Auf diesen kann die AMI-Prozedur über die vorinitialisierte String-Variable *ret_env* zugreifen. Dazu muss der Tochter-Prozess die C-Funktion *pipewrite()* aufrufen, um ihr die entsprechende Zeichenkette zu übergeben. Dieser Mechanismus funktioniert also nur mit selbst geschriebenen C/C++-Programmen, unter Einbindung der AMI-Beigabe *pipe.h.* Die Variable *ret_int* nimmt den Returnwert des Tochter-Prozesses auf. Da z. Zt. nur eine Pipe implementiert ist, können derartige Prozesse nur sequentiell ausgefüht werden. Entsprechend wartet der AMI-Interpreter auf die Beendigung, bevor er weitere Schritte ausführt.

Parameter können an jedes Programm übergeben werden, das zur Aufnahme bereit ist. AMI erleichtert die Übergabe durch den Operator „@“, z. B.

ret_int = sun_coord @long @lat;

mit *long* und *lat* als float-Variablen. Aufgrund des vorangestellten @-Operators werden nicht die Zeichenketten ″long″ und ″lat″ sondern die Inhalte der entsprechenden Variablen, automatisch in Zeichenketten umgewandelt, an das Programm *sun_coord* übergeben.

Bei vielen Automatisierungsaufgaben ist es sinnvoll oder sogar notwendig, parallele Prozesse zu starten. Wird ein Programm mit dem vorangestellten Schlüsselwort *par* gestartet, z. B. *par catch_sun*, so wartet AMI nicht auf die Beendigung des Prozesses sondern fährt mit der Abarbeitung der folgenden Befehle fort. Da in solchen Fällen häufig eine komplexe Interprozess-Kommunikation notwendig ist, wurde hier ein völlig anderer Mechanismus gewählt. Die Verständigung erfolgt mittels des Winsock-Clients *tcp_client*. Voraussetzung ist natürlich, dass der parallele Tochterprozess einen entsprechenden Server startet. Dies ist heute mit gängigen Programmiersprachen wie C/C++ oder LabVIEW recht einfach zu realisieren. *tcp_client* ist als EXE-Programm AMI beigegeben und unterstützt auch den oben beschriebenen Pipe-Mechanismus.

Jeder Server wird über seine IP- und Port-Adresse identifiziert und angesprochen. *tcp_client* prüft zunächst, ob der Server bereits gestartet wurde und wartet gegebenenfalls. Die Verständigung über TCP/IP hat den Vorteil, dass sie weitgehend Hardware- und Betriebssystemunabhängig ist. Auf diese Weise lassen sich auch verteilte Systeme problemlos steuern. Umgekehrt funktioniert es genauso gut zwischen verschiedenen Prozessen auf dem „eigenen“ Rechner, wenn man die „local-host-IP-Adresse“ 127.0.0.1 verwendet. AMI hat somit die Möglichkeit, parallel laufende Prozesse mehrfach im Verlauf einer Prozedur anzusprechen und mit immer neuen Aufträgen zu versorgen.

Der parallele Prozess sendet das jeweilige Ergebnis oder „ok“ an *tcp_client*, welcher es seinerseits in die Pipe schreibt, auf die schließlich AMI über die *ret_env*-Variable zugreifen kann. Mit Hilfe eines zwischen den Programmen vereinbarten Befehls kann AMI den parallelen Prozess zur Beendigung auffordern, wenn seine Dienste nicht mehr benötigt werden.

■ Beispiel für eine AMI-Prozedur:

Um uns auch praktisch mit AMI vertraut zu machen, betrachten wir die kleine Prozedur *kalender.ami*, die einen Datumskalender mit Wochentag ausgibt. Anfangs- und Enddatum werden als Parameter übergeben:

z. B. *kalender 01:12:2006 31:01:2007*

Die Ausgabe erfolgt über die Konsole mit 20 Zeilen, dann wird auf die Eingabe von <Enter> gewartet, das Konsolenfenster gelöscht und die nächsten 20 Datumsausgaben getätigt, usw. Beispiel:

09:09:2005 Freitag

10:09:2005 Samstag

….

Weiter mit <Enter>

Zur Errechnung des Wochentags steht das In C++ geschriebene Kommando *wochtag* zur Verfügung. Ihm müssen die Parameter *tag*, *monat* und *jahr* übergeben werden. Testen Sie das Programm durch Eingabe von

>wochtag 9 9 2005

Die Ausgabe muss *Freitag* lauten.
So sieht die AMI-Prozedur *kalender.ami* aus:

```
#-------------------------------------------------------#
# Gibt einen Kalender mit Datum und Wochentag aus       #
# Anfangs- und Enddatum werden vom Benutzer uebergegeben #
# Die Ausgabe erfolgt mit 20 Zeilen pro Konsolenfenster  #
#-------------------------------------------------------#

# Variablendeklarationen                                #
date day_anf, day_end, day;
int zz = 0;
int dd, mm, jjjj, daynr;
string enter;
# Beginn des Ablaufteils                                #
# Uebergabeparameter entgegen nehmen                    #
day_anf = $1;
day_end = $2;
cls;  # Konsole loeschen                                 #
# zum besseren Verständnis einiger Funktionen: ami -h    #
day = day_anf;
cls;
do
{
   datetoi(day, dd, mm, jjjj);
   printf("%y", day);
   wochtag @dd @mm @jjjj;  # EXE-Programm aufrufen       #
   printf("\n");
   day = day + 1;
   zz = zz + 1;
   if((zz % 20) == 0)
   {
      printf("\n \n");
      read("Weiter mit <Enter>", enter);
    cls;
  }
}
while(day <= day_end);
```

Hier noch der C++-Quelltext des Kommandos *wochtag*:

```
//-------------------------------------------------------
// Kommando "wochtag"
// Berechnet zu einem eingegebenen Datum ab 1583 den
// Wochentag und gibt diesen aus.
// Tag, Monat und Jahr werden als Parameter beim Aufruf
// uebergeben
// Bsp.: wochtag 9 9 2005, Ausgabe: Freitag
//-------------------------------------------------------
#include <stdlib.h>
#include <iostream.h>
using namespace std;
//-------------------------------------------------------
```

```
int main(int argc, char *argv[])
{
 int j, m, t, h, n, tag;
 if(argc != 4)
   cerr << "Falsche Parameterzahl" << endl;
 t = atoi(argv[1]);
 m = atoi(argv[2]);
 j = atoi(argv[3]);
 if(j > 1582)
 {
   if(m == 1 || m == 2)
   {
     m = m + 12;
     j--;
   }
   h = t + 2 * m + ((3 * m + 3) / 5) + j + (j / 4)
       - (j / 100) + (j / 400) + 2;
   tag = h % 7 + 1;
   if(tag == 1) cout << " Samstag";
   if(tag == 2) cout << " Sonntag";
   if(tag == 3) cout << " Montag";
   if(tag == 4) cout << " Dienstag";
   if(tag == 5) cout << " Mittwoch";
   if(tag == 6) cout << " Donnerstag";
   if(tag == 7) cout << " Freitag";
 }
 else cerr << "Falsche Jahreszahl" << endl;
 return 0;
}
//-------------------------------------------------------
```

Das Programm prüft keine falschen Datumsbestandteile ab. Diese führen zu einem falschen Ergebnis. AMI-Prozedur und C-Programm finden Sie auf unserer Buch-Webseite. ■

Das obige Beispiel ist noch keine absolut überzeugende Demonstration für den sinnvollen Einsatz von Skriptsprachen. Der ist dann gegeben, wenn z. B. mehrere Programme mit einander kombiniert werden, wie das in unserer Prozedur *pardemo.ami* der Fall ist. Schauen Sie sich dazu die Aufgabe 2 der folgenden Übungen an.

9.3 Aufgaben

Aufgabe 1

Machen Sie sich mit der Skriptsprache AMI vertraut, indem Sie die auf der Buch-Webseite befindlichen Beispiele anschauen und diese dann ausführen.

Aufgabe 2

Ebenfalls auf der Buch-Webseiten befindet sich die die AMI-Prozedur *pardemo.ami*, die eine komplexere Automationsaufgabe simuliert. Neben der genannten Prozedur benötigen Sie noch die vier LabVIEW-Programme *catch_sun.exe*, *tel_control.exe*, *spec_control.exe* sowie *ccd_control.exe* sowie die LabVIEW-Runtime-Engine (alle auf der Buch-Webseite).

Starten Sie die Prozedur:

>ami pardemo

Simuliert wird eine Messprozedur an einem Sonnenobservatorium unter Verwendung des Teleskops, des Spektrografen und einer CCD-Kamera zur Aufnahme von Sonnenspektren.

Zunächst wird das Teleskop mit Hilfe des Programms *catch_sun.exe* auf die Sonne gefahren. Danach werden die drei anderen Programme als parallele Prozesse gestartet (AMI-Befehl *par*) und jeweils in einer Schleife nacheinander mehrfach mit Parametern versorgt. Die Kommunikation erfolgt mittels des Kommandos *tcp_client* über TCP/IP. Der folgende kleine Auszug aus der Prozedur *pardemo.ami* demonstriert den Gebrauch von *tcp_client*:

```
# Teleskop nach (B, L) fahren #
ret_int = tcp_client.exe @ip @port2 @b @l; ex = ret_env;
if(ex != "ok") quit;
```

tcp_client adressiert zunächst einen Server. Das ist in diesem Fall ein Programm, das Parameter entgegen nehmen möchte. Der Parameter *ip* enthält die IP-Adresse des Servers, *port2* ist die Portadresse, auf der der Server horcht und sendet. Die beiden restlichen Parameter *b* und *l* enthalten die eigentlichen Nutzinformationen (Breite und Länge auf der Sonne) für den Server, in diesem Beispiel das LabVIEW-Programm *tel_control.exe*, welches das Sonnen-Teleskop steuert. Der Server ist in der Lage, ein Erfolgssignal an *tcp_client* zurückzusenden. In diesem Fall wurde der String "ok" vereinbart. Da *AMI* selbst keine TCP-Schnittstelle hat, kommunizieren *tcp_client* und *AMI* über die *AMI*-eigene Pipe. Der Rückgabestring der Pipe landet immer in der vordefinierten Variablen *ret_env*. Im obigen Beispiel wird einfach der String ″ok″ weitergereicht. Unter einer Pipe versteht man eine gepufferte Datenverbindung zwischen zwei Prozessen nach dem First In - First Out-Prinzip. Pipes sind in verschiedenen Betriebssystemen realisiert, z. B. in UNIX (→ s. Kap. 10.2), nicht jedoch in Windows. Hier musste die Pipe speziell für *AMI* implementiert werden.

Wegen der beliebigen IP-Adresse könnten die verschiedenen über *tcp_client* bedienten Programme auch auf verschiedenen, im Prinzip weltweit, vernetzten Rechnern liegen. Möchte man ein Programm auf dem „eigenen“ Rechner ansprechen, wählt man immer die IP-Adresse

```
127.0.0.1
```

Verschiedene Programme auf dem gleichen Rechner werden über verschiedene Ports angesprochen.

Wegen der vielen Fenster gestaltet sich der Ablauf etwas unübersichtlich. Führen Sie die Prozedur daher mehrfach aus und versuchen Sie dann, den Ablauf anhand des nachstehenden Skripts (Prozedur-Quelltext) nachzuvollziehen.

Eine kleine Einführung in die grafische Programmiersprache LabVIEW von National Instruments, die sich vornehmlich zur Lösung von Automatisierungsproblemen eignet, finden Sie auf unserer Buch-Webseite.

10 UNIX und Linux

Besonders bei Hochschulen und Forschungsinstituten erfreut sich UNIX einer großen Beliebtheit (und Windows einer besonderen Abneigung). Aus diesem Grund sollte ein Naturwissenschaftler oder Ingenieur zumindest grob mit UNIX vertraut sein.

UNIX und Linux sind Hersteller- und Hardware-unabhängige Betriebssysteme (BS). Mit UNIX wurde erstmals ein BS in einer Hochsprache, nämlich C, verfasst. C wurde sogar extra erfunden, um UNIX zu implementieren (Ken Thompson und Dennis Ritchie, 1973). Somit kann UNIX auf jede Maschine portiert werden, für die ein C-Compiler existiert. Lediglich ein Teil des UNIX-Kerns muss jeweils neu formuliert und in Assembler geschrieben werden. Daraus ergibt sich zwanglos eine enge Verwandtschaft zwischen UNIX und der Sprache C. Systemaufrufe (→ s. Kap. 4 und Kap. 10.2) können aus C-Programmen heraus abgesetzt werden.

Linux geht auf Linus B. Torvald (ab 1990) zurück. Es fußt auf einem eigenen Kern, alle Quellprogramme wurden neu geschrieben. So gleicht es UNIX, unterliegt aber nicht dessen Lizenzbestimmungen und ist frei verfügbar.

Einführungen in UNIX und Linux existieren in großer Zahl. Wir befassen uns deshalb hier nur mit den Aspekten „Shell als Programmiersprache“ und „Prozesskonzept“, die für technische Anwendungen von besonderem Interesse sind. Im Weiteren werden wir nur noch von UNIX sprechen, meinen aber stets UNIX und Linux. Als Windows-Benutzer empfehlen wir Ihnen zum Ausprobieren die Verwendung einer bootfähigen UNIX-Live-CD. Wir haben die folgenden Beispiele mit Knoppix 5.0 getestet. Es ist frei im Internet verfügbar und enthält auch den GNU-C-Compiler *gcc*.

10.1 Die Shell als Programmiersprache

Der UNIX-Befehlsinterpreter heißt SHELL, weil er den Betriebssystemkern schalenartig umgibt. Der Kern selbst besitzt im Gegensatz zu manchen anderen BS keinen eigenen Interpreter. Die SHELL ist ein normales Anwenderprogramm. Jeder Benutzer kann seine eigene SHELL schreiben.

Tatsächlich existieren mehrere allgemein verwendete SHELLs. Natürlich gibt es auch grafische Benutzeroberflächen, die meist auf dem X-Windows-System beruhen. Die standardisierte UNIX-Oberfläche heißt CDE (Common Desktop Environment). Unter Linux ist zusätzlich KDE (K Desktop Environment) von Bedeutung. Die Erläuterungen des nachstehenden Textes beziehen sich auf die Bourne-SHELL, die einen zeilenorientierten Befehlsinterpreter bereitstellt. Die zu interpretierenden Befehle sind Programmnamen oder Namen von ausführbaren Dateien. UNIX beinhaltet mehr als 200 Dienstprogramme, die unter ihrem Namen als Befehl eingegeben werden können. Sie dienen der Dateimanipulation und -verarbeitung, der Programmausführung, der Textverarbeitung, der Kommunikation, der Programmentwicklung, usw. Das durchgängige Konzept von UNIX lautet ″small is beautiful″. Entsprechend kurz und kryptisch sind die Befehle. Nachstehend seien einige Beispiele genannt:

`login`	Sitzung beim Sytem anmelden = einloggen (z. B. login meier)
`cp`	Kopieren von Daten (copy)
`ls`	Verzeichnis-Inhalt ausgeben (wie *dir* unter DOS bzw. Eingabeaufforderung)

`pwd` Arbeitsverzeichnis (working directory) anzeigen

`cc` Aufruf des C-Compilers (oder gcc: Aufruf des GNU-C-Compilers)

`rm` Löschen einer Datei (remove)

`cat` Datei ansehen (concatenate)

`who` Aktive Benutzer anzeigen

`ps` Laufende Prozesse anzeigen

`wc` Zählt die Zeichen (-c), Wörter (-w), oder Zeilen (-l) einer Datei (wordcount)

`echo` Gibt das nachfolgende Argument aus (eine Art Print-Befehl)

Mit dem Befehle *man <Kommando>* erhält man genau die Beschreibung eines Kommandos.

Nachdem Sie sich beim System angemeldet haben, übernimmt die SHELL die Funktion eines Mittlers zwischen Ihnen und dem UNIX-Kern, ähnlich der *cmd.exe* unter Windows. Kommandos werden in der aktuellen Zeile hinter dem Prompt (i.a. ″$″) eingegeben. Sie bestehen aus einem Wort oder mehreren Wörtern. Ein Wort ist eine zusammenhängende Zeichenkette bis zum nächsten Trennzeichen (blank = Leerzeichen). Die vollständige Syntax eines Kommandos lautet:

Kommandoname [-option ...] [argument ...]

Alle Eingaben werden mit Carriage Return <CR > abgeschlossen. UNIX-Kommandos werden kleingeschrieben, Großschreibung führt zu einem Fehler!

Filter, Pipelines und Wildcards

Programme, die von der Standardeingabe lesen, die gelesenen Daten in irgendeiner Weise behandeln und anschließend auf die Standardausgabe schreiben, werden als Filter bezeichnet. Filter bieten die Möglichkeit der Ein-/Ausgabe-Umlenkung von bzw. auf Dateien auf SHELL-Ebene. Die Programme selbst müssen nicht geändert werden. '<' ist das Eingabe-, '>' das Ausgabe-Umlenkzeichen.

■ **Beispiele:**

`ls -l > dirdat`	(Umlenkung der Ausgabe von *ls –l* in die Datei *dirdat*)
`cal < year`	(liest den Parameter <yyyy> aus der Datei *year* und gibt einen Jahreskalender aus) ■

Eine Verkettung von Kommandos (Programmen) in einer Ausführungszeile wird als Pipeline bezeichnet. Diese Kommandos werden von der SHELL simultan gestartet und laufen parallel. Die Verkettung erfolgt über namenlose temporäre Dateien, die so genannten Pipes. Diese leiten die Ausgabe eines Prozesses an die Eingabe des nächststehenden Prozesses weiter. Das Pipezeichen ist '|'.

■ **Beispiele:**

`who | wc -l`	(Gibt die Anzahl der aktiven Benutzer aus)
`ls /bin | more`	(Bildschirm-gerechte Ausgabe des Inhalts des Verzeichnisses /bin) ■

Die Shell löst außerdem Wildcards ('?' = ein beliebiges Zeichen , '*' = beliebige Zeichenkette von 0 bis 14 Zeichen) auf.

■ **Beispiel:**

`rm ?` (löscht die Dateien a, b, c, …)

`rm a*b` (löscht alle Dateien, deren Namen mit ’a ’ anfangen und mit ’ b’ aufhören)

■

Drei weitere Eigenschaften werden im Folgenden behandelt:

– **SHELL-Variablen:**

Der Benutzer kann das Verhalten der SHELL und auch andere Programme und Dienstprogramme durch Daten, die in Variablen abgespeichert sind, steuern.

– **SHELL-Prozeduren (scripts):**

Eine häufig benutzte Folge von SHELL-Befehlen kann in einer Datei gespeichert werden. Der Name der Datei kann später dazu benutzt werden, um diese abgespeicherte Sequenz mit einem einzigen Befehl auszuführen.

– **Programmiersprachenkonstruktionen:**

Die SHELL beinhaltet Eigenschaften, die es erlauben, sie wie eine Programmiersprache zu benutzen. Diese Eigenschaften können dazu verwendet werden, um SHELL-Prozeduren zu erzeugen, die komplexe Operationen ausführen. Dazu stehen moderne Kontrollstrukturen (if-then-else, while, …) zur Verfügung. Der Sinn von Prozeduren besteht in einer Vereinfachung des Umgangs mit dem Betriebssystem. Man ist in der Lage, selbst definierte ″Superkommandos″ zu schreiben, die auf die eigenen spezifischen Bedürfnisse abgestimmt sind. Eine wichtige Spezialanwendung ist die Systemverwaltung (automatisches Sichern von Dateien nach bestimmten Kriterien, Betriebsstatistik usw.).

Prozesse

Um verschiedene Eigenschaften von SHELL-Prozeduren verstehen zu können, müssen wir zunächst einige etwas theoretische Überlegungen zum Prozesskonzept von UNIX anstellen. Die Prozesshierarchie ist baumartig wie das Dateiensystem. Ein Elternprozess kann Kinderprozesse hervorbringen, usw. Ein Prozess ist die Ausführung eines Programms, genauer gesagt eines ″image″. Ein ″image″ umfasst alles was ein Programm benötigt, um ablaufen zu können. Es enthält:

– Abbild des Speichers (mit Programmcode und Daten)
– Werte der Register
– Zustand der geöffneten Dateien
– Aktuelles Dateiverzeichnis
– Befehlszähler
– Statusregister
– SHELL-Variablen

Während der Ausführung muss das ″image″ im Hauptspeicher sein.

Nach dem Login ist die SHELL der einzige (benutzerspezifische) aktive Prozess (″login SHELL″). Sie können dies mit dem ps-Kommando verifizieren. Soll zum Beispiel das Kommando ls ausgeführt werden, dupliziert sich die SHELL. Die Kopie überlagert sich mit dem ls-Programm, während die Original-SHELL auf die Beendigung der Kopie wartet. Die Kopie einer SHELL nennt man ″subSHELL″.

Hintergrundprozesse

Beim Abschluss eines Kommandos mit <CR> startet die SHELL einen Prozess und wartet auf dessen Beendigung. Während der Ausführdauer können Sie keine weiteren Kommandos eingeben. Sie haben jedoch die Möglichkeit, einen Prozess in den ″Hintergrund″ zu schicken. Anders ausgedrückt: Sie nutzen die Mehrprogrammfähigkeit von UNIX aus.

Hintergrundprozesse sind immer dann sinnvoll, wenn ein Kommando oder auch ein Anwendungsprogramm als Batchjob ablaufen kann, d. h. wenn keine Interaktion (kein Dialog) zwischen Programm und Benutzer notwendig ist. Die meisten der gängigen UNIX-Kommandos sind derartige Programme, z. B.: *who, wc, cp, cd, pwd*. Dennoch lohnt es sich nicht, sie als Batchjob im Hintergrund ablaufen zu lassen, weil ihre Ausführzeit so gering ist, dass man sie genauso gut im Dialog starten kann. Reine Dialogkommandos wie *passwd* (zum Ändern des Passworts) sind für die Hintergrundverarbeitung natürlich erst recht ungeeignet.

In Frage kommen also solche Kommandos, die möglichst gar keine Bildschirmausgabe erzeugen und deren Laufzeit so lang ist, dass sie nicht innerhalb weniger Sekunden beendet sind, z. B. Drucker-Befehle sowie das find-Kommando, das den Dateienbaum nach bestimmten Kriterien durchsucht (*man find*). Während das Kommando im Hintergrund abläuft, können Sie im Dialog mit der SHELL wie gewohnt weiterarbeiten. Tragen Sie aber Vorsorge, dass Ihr Hintergrundkommando nicht mit einer Bildschirmausgabe in Ihren Dialog platzt. Unangenehm ist dies vor allem dann, wenn Sie gerade mit einem Editor oder einem Textverarbeitungs-Programm arbeiten. Die einfachste Art, derartiges zu verhindern, ist die Ausgabeumlenkung auf eine Datei.

Hintergrundprozesse werden mit Hilfe des &-Zeichens gestartet. Die allgemeine Syntax lautet:

Kommando [optionen] [argumente]&

Nach dem Start des Hintergrundprozesses gibt die SHELL gibt die Prozessnummer (PID) aus und ist bereit, weitere Kommandos auch für weitere Prozesses entgegen zunehmen. Die Beendigung eines Hintergrundprozesse wird nicht angezeigt (es sei denn vom Prozess selbst).

Informationen ermöglicht das

ps-Kommando (Infos über Prozesse)

Es liefert vor allem die PID, falls Sie diese vergessen haben. Möchten Sie einen Prozess vorzeitig beenden, geben Sie ein:

`kill -9 pid` (pid ist die Prozessnummer)

Das kill-Kommando soll hier nicht ausführlich behandelt werden. Die Syntax lautet

kill [-signal] pid.

Das Signal ″9″ bewirkt einen sicheren Abbruch des Programms. Sie können auch mehrere Prozesse gleichzeitig abbrechen.

Beispiel:

```
kill -9 1024 1036 946
```

Die entsprechenden drei Prozesse werden abgebrochen. Natürlich lassen sich nur eigene Prozesse beenden. Mit

```
kill -9 0
```

beenden Sie Ihre sämtlichen aktiven Prozesse. Der Superuser (Systemadministrator) darf die Prozesse aller Benutzer ″killen″. Er sollte ein vertrauenswürdiger Mensch sein.

■ **Beispiele:**

```
pr -h "Beispiel 1" prdat | lpr&
```

Die Datei *prdat* wird mit der Überschrift ″Beispiel 1″ ausgedruckt.

```
find . -name "c*" -print > finddat&
```

Alle Dateien im aktuellen Teil des Dateienbaums, deren Namen mit ″c″ anfangen, werden gesucht und in der Datei *finddat* aufgelistet. Beide Beispiele benötigen zur Ausführung möglicherweise längere Zeit und sollen deshalb im Hintergrund ablaufen. ■

Natürlich können Sie auch selbst geschriebene Programme im Hintergrund ablaufen lassen.

SHELL-Variablen

Jede Programmiersprache enthält Variablen, so auch die SHELL. Zwar existieren, wie wir noch sehen werden, verschiedene Arten von SHELL-Variablen, jedoch enthalten sie alle einzelne Zeichen oder Zeichenketten. Numerische Variablen gibt es nicht. Dennoch kann man mit SHELL-Variablen auch rechnen, doch davon später mehr.

Die Namen dieser frei definierbaren SHELL-Variablen bestehen aus Buchstaben und Ziffern. Das erste Zeichen muss ein Buchstabe oder das Unterstreichungszeichen (underscore) sein. Die Zuweisung erfolgt durch das Gleichheitszeichen.

Beispiele:

```
a=Ciao                  (kein Leerzeichen links und rechts vom ″=″)
b=Ciao Petra            (ist nicht erlaubt, da 2 Zeichenketten!)
c="Ciao Petra"
d=51
```

Steht vor einer SHELL-Variablen das Zeichen ″$″, erkennt die SHELL, dass eine Variable gemeint ist und setzt deren Werte ein.

Beispiel: `echo $a $b $c $d` (Ausgabe der Inhalte der SHELL-Variablen a, b, c, d)
SHELL-Variablen gelten nur in der SHELL, in der sie erzeugt wurden. Die Werte von SHELL-Variablen sind nur dann in einer subSHELL bekannt, wenn sie vorher ″exportiert″ worden sind.

■ **Beispiel:**

Geben Sie ein:

```
a=a
b=b
c=c
d=d
echo $a $b $c $d
export a b
sh                          (-oder- bsh,  startet eine subSHELL)
echo $a $b $c $d            (nur a und b haben definierte Werte)
```

■

Mit <Strg>+<D> beendet man die aktuelle SHELL, z. B. obige subSHELL.

Der Export gilt für alle subSHELLS. Wenn Sie jedoch eine exportierte SHELL-Variable in einer subSHELL verändern (einen anderen Inhalt zuweisen), müssen Sie diese erneut exportieren (*export*-Kommando), wenn die veränderte Variable untergeordneten subSHELLS bekannt gemacht werden soll.

SHELL Variablen stehen im Zentralspeicher. Nachdem Sie sich beim System abgemeldet haben, sind alle SHELL-Variablen vergessen. Es sei denn, es sind noch Hintergrundprozesse (s. o.) aktiv, die Variablen verwenden.

Vordefinierte Variablen der SHELL

Einige SHELL-Variablen haben für die SHELL eine besondere Bedeutung. Sie sind mit Werten vorbesetzt. Die Namen und Werte dieser sog. *Environment*-Variablen können mit dem Kommando *set* (ohne Argument) ausgegeben werden. Vordefinierte SHELL-Variablen müssen exportiert werden, wenn ihre Werte verändert wurden und diese in einer subSHELL benutzt werden sollen. Wichtige Environment-Variablen sind:

HOME Name des Home-Dateiverzeichnisses. Diesen Namen verwendet die SHELL, wenn beim Kommando *cd* (change directory) kein Dateiverzeichnis angegeben wird.

PATH Suchpfad für Kommandos. Namen von Dateiverzeichnissen werden getrennt durch **':'**. Mit diesen Namen wird von links nach rechts ein relativer Pfadname vor Ausführung eines Kommandos ergänzt (Standard: PATH=.:/bin:/usr/bin).

PS1 Erstes Promptzeichen der SHELL (Standard PS1=$, Superuser: PS1=#)

PS2 Zweites Promptzeichen der SHELL

MAIL Pfadname der Mail-Datei.

Achtung: Die Großschreibung ist signifikant! Es existieren weitere vordefinierte SHELL Variablen.

Quoting

Mit Hilfe von Metazeichen; wird der SHELL mitgeteilt, welche Ersetzungen vor Ausführung eines Kommandos vorgenommen werden sollen. Solche Metazeichen sind zum Beispiel:

? * [...] <> $ | >>

Im Zusammenhang mit den SHELL-Variablen interessieren uns u. a. die Ersetzungsmechanismen. Die Ersetzungsmechanismen werden durch Apostrophe gesteuert.

'string' Rechtsgerichtete („normale“) Apostrophe: In "string" sollen keine Ersetzungen vorgenommen werden.

Geben Sie ein: `echo $HOME '$HOME'`

"string" Doppelte Apostrophe: In ″string″ sollen nur SHELL Variablen ersetzt werden. Es findet keine Dateinamengenerierung statt.

Geben Sie ein:
```
echo * $PS2
echo "* $PS2"
```

`string` Linksgerichtete Apostrophe: ″string″ wird als Kommando interpretiert und anstelle von "string" wird das Ergebnis des Kommandos eingesetzt.

Geben Sie ein:
```
a=pwd
echo `$a`
```

\ ″Backslash″: Das nachfolgende Metazeichen verliert seine Bedeutung.

Geben Sie ein: `echo \$"HOME\"`

■ **Beispiel:**

```
a=date
echo $a              liefert ″date″
echo `$a`            liefert das aktuelle Datum
echo 'Datum $a'      liefert ″Datum $a″
echo "Datum $a"      liefert "Datum date"
```

■

SHELL-Variablen in Prozeduren

Es gibt spezielle SHELL-Variablen, die nur in Prozeduren Verwendung finden. Auch sie werden durch ein vorangestelltes ″$″ ausgewertet. Die Werte dieser Variablen können Sie nicht durch bloße Zuweisung mittels "=" Zeichen verändern. Von besonderer praktischer Bedeutung sind die *Stellungsvariablen*.

Mit ihrer Hilfe lassen sich Parameter an eine SHELL-Prozedur übergeben. Die Stellungsvariablen haben folgende Bedeutung:

0 =Name der aufrufenden SHELL-Prozedur

1 =1. Parameter (Argument) des Aufrufs

…

9 =9. Parameter (Argument) des Aufrufs

Stellungsvariablen können nicht exportiert werden.

Auswertung bzw. Ausgabe: echo $0, usw.

■ **Beispiel:**

Die folgende ″Mini″-Prozedur soll *xprint* heißen. Die Eingabe erfolgt mit Hilfe eines Editors (z.B.des Editors *joe*: `joe xprint`). Die Prozedur dient zur druckergerechten Ausgabe einer Datei.

```
pr -h $2 $1 | lpr
```

Als Parameter (Argument) wird ihr beim Aufruf der Name der auszudruckenden Datei sowie die Drucküberschrift mitgegeben. Innerhalb der Prozedur findet sich dieser Name als Inhalt derVariablen 1 wieder, der mit $1 ausgewertet werden kann. Die Variable 2 enthält die Überschrift. So rufen Sie die Prozedur *xprint* auf:

`sh xprint datei ″Ueberschrift″` (oder bsh …)

■

Im Zusammenhang mit den Stellungsvariablen sind noch einige andere vordefinierte Variablen zu erwähnen, die Informationen über Prozesse enthalten und entsprechend gelesen und ausgewertet werden können.

* Alle Argumente des Prozedur-Aufrufs als eine Zeichenkette

@ Alle Argumente des Aufrufs als einzelne Zeichenketten

- # Anzahl der Argumente
- - Optionen der aktuellen SHELL
- ? Exit-Status des zuletzt ausgeführten Kommandos
- $ Prozessnummer (PID) der aktuellen SHELL
- ! Prozessnummer des letzten Hintergrundprozesses

Möchten Sie beispielsweise die PID der aktuellen SHELL sehen, geben Sie ein

```
echo $$
```

Übrigens: Sie können *xprint* und jede andere SHELL-Prozedur auch ohne das sh-Kommando ausführen. In einem solchen Fall müssen Sie sich für die entsprechende Datei das Ausführrecht erteilen, z. B.

`chmod u+x xprint` (*chmod* steuert die Zugriffsrechte auf eine Datei: *man chmod*),

sofern dieses noch nicht besteht. Der Aufruf verkürzt sich nun auf:

```
xprint datei "Ueberschrift"
```

Beachten Sie, dass *xprint* ein relativer Dateiname ist. *xprint* muss entweder im Working-Directory oder in einem der in der SHELL-Variablen PATH definierten Directories verzeichnet sein.

Alle SHELL-Variablen, die nicht Stellungsvariablen sind, nennt man Kenn- oder Schlüsselwortvariablen.

Neben der Übergabe durch Parameter (Argumente) im Aufruf (s. Beispiel *xprint*) können Stellungsvariablen mit Hilfe des *set*-Kommandos Werte zugewiesen werden.

■ **Beispiel:**

Das Kommando *date* liefert eine Reihe von Datums- und Zeitinformationen, meist in der Form:

Wochentag Monat Tag Tageszeit MEZ Jahr.

Das sind insgesamt 6 verschiedene Parameter. Prüfen Sie die Wirkung von *date* auf Ihrem Rechner. Sie kann, je nach Systemvariante, geringfügig verschieden sein.

Schreiben Sie die folgende Prozedur *set_dat*:

```
set `date`
echo $1 $2 $3 $4 $5 $6
echo $*
echo $#
echo $$
```

Ausführen mit sh set_dat oder set_dat bzw. /set_dat (x-Recht muss bestehen!) ■

Zur Vergabe des x-Rechts bietet sich folgend Ein-Zeilen-Prozedur *cx* an:

```
chmod u+x $1
```

Die allgemeine Syntax des *set*-Kommandos lautet:

set [optionen] [parameter1 ....]

set ohne Optionen und Parameter gibt alle definierten SHELL-Variablen mit ihren Werten aus. Die Optionen von *set* heißen SHELL-Optionen. Die wichtigste Option des *set*-Kommandos lautet:

-k: Kennwortvariablen dürfen auch hinter dem Kommandonamen (Name d. Prozedur) stehen.

Die Übergabe von Variablen kann folgendermaßen geschehen:

1. Export von SHELL-Variablen
2. im Prozeduraufruf
   a) als Kennwortvariablen
   b) als Stellungsvariablen

Mit der Übergabe der Kennwortvariablen; haben wir uns noch nicht befasst. Das soll nun nachgeholt werden. Die Prozedur heiße *test1*, die zu übergehende Kennwortvariable *name*. Der entsprechende Aufruf hat folgendes Aussehen:

```
name=Erna sh test1
```

Die Kennwortvariable(n) müssen vor dem Prozedurnamen (hier: *test1*) stehen. Es sei denn, die -k Option ist gesetzt:

set -k

```
sh test1 name=Erna
```
(was sicherlich eingängiger ist)

Alle automatisch zur Verfügung stehenden Variablen können Sie sich mit dem Kommando

```
env
```

anzeigen lassen.

Mit dem Kommando

```
readonly  [variable1 ...]
```

können Sie SHELL-Variablen zu Konstanten erklären (bis zur Beendigung der SHELL). Auflistung aller readonly-Variablen:

```
readonly
```

Löschen von Variablen:

```
unset VARNAME
```

SHELL-Programmierung

SHELL-Prozeduren müssen keinesfalls als Sequenz von unbedingten Befehlen ablaufen. Vielmehr erlauben SHELL-Prozeduren Ablauf-Strukturen wie if... then... else (Auswahlen) und while... do... done (Schleifen). Im Rahmen dieser kleinen Einführung können wir nur verhältnismäßig kurz auf die SHELL-Programmierung mit Hilfe solcher Strukturen eingehen. Insgesamt weist die SHELL alle Eigenschaften einer höheren Programmiersprache auf. Dennoch ist es nicht sinnvoll, alle Probleme auf SHELL-Ebene zu lösen. Da die Befehle nicht kompiliert, sondern nur interpretiert werden, laufen die Programme vergleichsweise langsam. Umfangreichere Anwendungsprobleme sollten unbedingt in einer Compilersprache, vorzugsweise C oder Java, gelöst werden.

Die SHELL-Programmierung bietet eine sehr komfortable Möglichkeit, wiederkehrende Aufgaben im Umgang mit UNIX effektiv und flexibel zu lösen. Eigenschaften von Dateien, die mit speziellen Kommandos, u. a. *test*, geprüft werden, spielen dabei sehr oft eine Rolle.

Innerhalb von SHELL-Programmen dürfen Sie alle UNIX-Kommandos verwenden. Es gibt sogar eine Reihe von Kommandos wie *test*, *sleep*, *eval* und *expr*, die nur in SHELL-Prozeduren sinnvoll sind. Einige dieser Kommandos, die bestimmte Aussagen auf ihren Wahrheitswert überprüfen, liefern einen Exit-Status, der folgende Werte annehmen kann:

Exit-Status = 0 wahr (true)
Exit-Status = !0 falsch (false)

Gerade umgekehrt wie bei C/C++. Das am häufigsten verwendete Kommando dieser Art ist *test*. Wir wollen es deshalb ausführlich behandeln.

test

Syntax: test Ausdruck oder
[Ausdruck]

Das Kommando *test* dient dazu:

– Eigenschaften von Dateien festzustellen,
– Zeichenketten zu vergleichen,
– Ganze Zahlen algebraisch zu vergleichen.

test liefert als Ergebnis den Wert ″wahr″ (Exit-Status 0) oder den Wert ″falsch″ (Exit-Status ungleich 0).

Optionen:

-b name	name existiert und ist ein blockorientiertes Gerät.
-c name	name existiert und ist ein zeichenorientiertes Gerät.
-d name	name existiert und ist ein Dateiverzeichnis.
-f name	name existiert und ist eine Datei (file).
-r name	name existiert und Prozess hat Leseerlaubnis.
-s name	name existiert und ist nicht leer.
-n zeichenfolge	zeichenfolge ist nicht leer (Länge > 0).
-w name	name existiert und Prozess hat Schreiberlaubnis.
-x name	name existiert und ist ausführbar.
-z zeichenfolge	zeichenfolge ist leer (Länge = 0)
-z1 = z2	Die beiden Zeichenfolgen z1 und z2 sind gleich
-z1 != z2	Die beiden Zeichenfolgen z1 und z2 sind ungleich
wert1 -op wert2	Die Zahlenwerte wert1 und wert2 werden anhand des Operators *op* verglichen. Für op kann eq, ne, gt, ge, lt oder le eingesetzt werden.

Weitere Ausdrücke existieren.
Bedingungen können mit ″ (″ und ″)″ geklammert werden (Achtung: mit ″\″für die SHELL entwerten), mit ″!″ negiert werden, mit ″-o″ logisch ODER verknüpft und mit ″-a″ logisch UND verknüpft werden.

Beispiele:

`test -f name`	Wahr, falls die Datei ″name″ existiert und eine normale Datei ist.
`test -d name`	Wahr, falls die Datei ″name″ existiert und ein Dateiverzeichnis ist.
`test -f name -a`	Wahr, falls die Datei ″name″ existiert und -s name nicht leer ist (and).
`test -z "$a"`	Wahr, falls die SHELL-Variable ″a″ eine leere Zeichenkette enthält.
`test -n "$a"`	Wahr, falls die SHELL-Variable ″a″ eine nicht leere Zeichenkette enthält.
`test "$a" = "oma"`	Wahr, falls die SHELL-Variable ″a″ die Zeichenkette ″oma″ enthält.
`test "$a" -eq 1 -o "$a" -eq 2`	Wahr, falls die SHELL-Variable ″a″ 1 oder 2 ist.
`test ! $# -eq 0`	Wahr, falls die Anzahl der Parameter einer SHELL-Prozedur ungleich 0 ist.

Vorsicht: Eigene Datei "test" vermeiden.

Natürlich können auch Wildcards verwendet werden.

Einige weitere Prozedur-Kommandos wollen wir nur kurz streifen.

read

liest eine komplette Zeile von stdin (datei) und weist die Variablen zu.
exit status = 0 gültige Zeile gelesen
exit status = 1 EOF (Dateiende)

Beispiel:

```
read a b c d
```

der gelesene Satz soll lauten: „Welche Variable bekommt welchen Wert?“

Kriterium ist das Trennzeichen. Existieren mehr Worte als Variablen, bekommt die letzte den Rest als Zeichenkette " .... " zugewiesen, im obigen Beispiel: d = "welchen Wert?"

expr

dient zum arithmetischen Rechnen mit SHELL-Variablen, auch Vergleiche sind möglich. Beim Rechnen sind nur ganze Zahlen erlaubt.

Syntax: expr a op b

Folgende logische Operationen *op* sind erlaubt:

= gleich
< kleiner
> größer
<= kleiner gleich
>= größer gleich
!= ungleich

Folgende arithmetische Operationen sind erlaubt:

+ plus
- minus
* mal
/ geteilt
% Modulo (ganzzahliger Rest)

Verifizieren Sie das *expr*-Kommando durch folgende Eingaben:

```
a=6
b=5
c= `expr $a = $b`
echo $c  (c = 1  falls Vergleich erfolgreich, c = 0  falls
          Vergleich nicht erfolgreich)
expr $a + 1
echo $a
echo `expr $a \< $b`
a= `expr $a + 1`
b= `expr $a \* $b`
echo $a $b
```

Achtung: Die in den Beispielen vorkommenden Operatoren "<" und "*" sind gleichzeitig Sonderzeichen der SHELL. Sie müssen deshalb mit einem Backslash "\" entwertet werden!

#, true, false, sleep

Die Kommandos #, true, false und sleep erzeugen keine Ausgaben.

Der Rest der Kommadozeile wird von der SHELL überlesen (Kommentarzeichen).

true liefert immer den Wert "wahr" (Exit-Status 0).

false liefert immer den Wert "falsch" (Exit-Status !=0).

■ **Beispiel:**

Schreiben Sie eine Prozedur mit folgendem Inhalt:

```
while true # immer wahr
do
  date
  sleep 5
done
```

Das Kommando *sleep <zeit>* veranlasst die SHELL, die angegebene Zeit (in Sekunden) zu warten. Dies ist Ihre erste SHELL-Prozedur, die eine Struktur, in diesem Fall while... do... done, verwendet. Sie gibt alle 5 Sekunden das Datum auf dem Bildschirm aus. Da die Bedingung immer "wahr" bleibt (while true), läuft das Programm in einer Endlosschleife (Abbruch mit <Strg>+<c>). Wie muss die Prozedur verändert werden, damit die Wartezeit variabel gehalten werden kann? ■

Wir wollen nun die wichtigsten Strukturen kennen lernen.

if then else fi

```
if [Bedingung]
then
   [Kommandos]
else                  -oder- elif
   [Kommandos]                 then
fi                               [Kommanods]
                               else
                                 [Kommandos]
                               fi
```

Der *else*-Teil ist optional.

Jedes Kommando liefert einen Exit-Status. Der Exit-Status wird von dem jeweiligen Programm gesetzt und ist
- 0 für "true"
- ungleich 0 für "false"

Mit *elif* kann die if-Konstruktion weiter geschachtelt werden.

■ **Beispiele:**

Die folgenden zwei Beispiele stellen fest, ob eine Datei eine normale Datei oder ein Dateiverzeichnis ist. Es wird erwartet, dass der Dateiname als Argument übergeben wird:

Beispiel 1:

```
if test -f $1
then
  echo File $1 ist eine normale Datei
elif test -d $1
then
  echo File $1 ist eine directory
else
  echo File $1 existiert nicht
fi
```

Beispiel 2:

```
if test -z "$1"  # falls Zeichenkette leer (kein Argument ue
                 # bergeben) Fehlermeldung
then
  echo Syntaxfehler
elif test -f $1
then
  echo File $1 ist eine normale Datei
elif test -d $1
then
  echo File $1 ist eine directory
else
  echo File $1 existiert nicht
fi
```

■

while do done

```
while [Bedingung]
do
  [Kommandos]
done
```

Die Kommandos werden solange ausgeführt, bis der Exit-Status von Bedingung "false" wird.

■ **Beispiele:**

Es sollen alle Argumente einer Kommandozeile ausgegeben werden:

```
while test -n "$1"  # bei beliebigen Zeichenketten empfehlen
                    # sich Anfuehrungszeichen
do
  echo $1
  shift
done
```

Das Kommando *shift* bewirkt, dass alle Argumente um eine Position nach links verschoben werden. Damit ist es möglich, auch mehr als 9 Argumente zu adressieren.

Das folgende Beispiel stellt ebenfalls fest, ob eine Datei eine normale Datei oder ein Dateiverzeichnis ist. Es wird erwartet, dass ein oder mehrere Dateinamen als Argument übergeben werden:

```
while test -n "$1"
do
  if test -f $1
  then
    echo File $1 ist eine normale Datei
  elif test -d $1
  then
    echo File $1 ist eine directory
  else
    echo File $1 existiert nicht
  fi
shift
done
```

Durch Aufruf dieser SHELL-Prozedur mit
listdat * (*listdat* sei der Name der Prozedur)
erhält man eine Liste aller Dateien des aktuellen Dateiverzeichnisses. ■

until do done

until [Bedingung]
do
  [Kommandos]
done

Die until-Schleife ist die Umkehrung der while-Schleife. Die Kommandos werden solange ausgeführt, bis der Exit-Status von Bedingung "wahr" wird. Z. B. kann until in Kommandoprozeduren benutzt werden, um sicherzustellen, dass der Benutzer eine richtige Aktion vornimmt.

■ **Beispiel:**

Die folgende Prozedur läuft solange, bis ein existierender Dateiname über die Tastatur eingegeben wird.

```
until test -f "$name"
do
  echo Eingabe:
  read name
done
echo "Datei $name existiert"
```

Die erste Zeile kann auch *until [-f "$name"]* geschrieben werden.
Vorsicht: Je ein Blank hinter "[" und vor "]". ■

for in do done

for wort in string1 string2 ...
do
  [Kommandos]
done

Der SHELL-Variablen "wort" wird nacheinander "stringl", "string2",... zugewiesen. Anschließend wird der Schleifenrumpf ausgeführt. In der Praxis wird für "string1 string2 ... " gewöhnlich die Ausgabe eines Kommandos verwendet, z. B. `cat file`. Die *for*-Konstruktion ist also keine Zählschleife wie in vielen höheren Programmiersprachen.

■ **Beispiel:**

```
for FILE in `find . -atime +90 -print`
do
  rm $FILE  # als Uebung besser cat statt rm verwenden
done
```

Mit dem *find*-Kommando werden die Dateien des aktuellen Dateiverzeichnisses ermittelt, auf die seit 90 Tagen nicht mehr zugegriffen wurde. Diese Dateien werden nacheinander der SHELL-Variablen FILE zugewiesen und anschließend gelöscht. ■

10.2 C und UNIX

C ist eng mit UNIX verknüpft, das überwiegend selbst in C geschrieben ist. C ist gewissermaßen die "Haussprache" von UNIX. Viele positive Systemeigenschaften können durch Programmierung in C genutzt werden, beispielsweise die Ein-, Ausgabeumlenkung (Filtereigenschaften) und der Pipe-Mechanismus. Systemaufrufe sind aus C-Programmen heraus möglich. *fork*, *exec*, *wait* und *exit* sind Beispiele für solche Systemaufrufe. Unix besitzt noch weitaus mehr, doch diese vier regeln im Wesentlichen die Prozessgenerierung und -synchronisation. Darüber hinaus sind sie als Kommandos, die v. a. in SHELL-Prozeduren von Bedeutung sein können, realisiert.

Im Einzelnen haben die wichtigsten Systemaufrufe folgende Wirkung:

int getpid(void)
int getppid(void)
Erfragt eigene PID und die des Elternprozesses (parent).

int fork(void)
Erzeugt eine Kopie eines Prozesses mit gleichem image (Multithreading). Das Original wird als Elternprozess (parent) bezeichnet, die Kopie als Kindprozess (child). Beide Prozesse laufen mit gleicher Umgebung und mit dem gleichen Befehlszähler weiter. Eltern- und Kindprozess identifizieren sich anhand des Wertes, den *fork* zurückliefert:

0	Kindprozess
>0	Elternprozess (Prozessnummer (PID) des Kindes)
-1	Kindprozess konnte nicht erzeugt werden (mehr als 14 Kindprozesse?, abhängig von Variable MAXCMD im Systemkern)

int system(const char *programmname)
Startet eine neue Shell (/bin/sh oder /bin/bsh) und ruft dort in eigener Umgebung das angegebene Programm auf (Multitasking). Keine Parameterübergabe beim Aufruf möglich.

int execlp(char *programmname, char *arg[0], ...)
Ähnlich wie *spawnlp()* (→ s. Kap. 8.2). Es können Parameter
übergeben werden. Der letzte Parameter muss NULL sein. Im Gegensatz zu *system()* gibt es keine Rückkehr zum Eltenprozess (wie P_OVERLAY bei *spawnlp()*). Trick: erst Kindprozess mit *fork()* starten und von dort aus *execlp()* aufrufen. Es gibt noch andere *exec()*-Varianten.

int wait(int *status)
Veranlasst den Elternprozess zu warten, bis ein Kindprozess beendet ist. *wait() l*iefert die Prozessnummer des Kindes zurück (evtl. mit Fehlercode in der Pointer-Variablen *status*). Variante: *waitpid()* wartet auf einen bestimmten Kindprozess.

int exit(int returnwert)
Beendet einen Prozess und aktiviert einen evtl. wartenden Elternprozess (wenn der beendete Prozess ein Kindprozess ist).

```
#include <signal.h>
```
int kill(int pid, int signal)

Beendet den Kindprozess mit der angegebenen PID. Als Signal ist in der Regel die symbolische Konstante SIGKILL zu verwenden (#include <signal.h>).

Der UNIX-Befehlssatz kann durch eigene, meist kleine C-Programme, erweitert werden. Der C-Compiler ist ein Unix-Dienstprogramm (nicht zwingend C++, daher verwenden wir im Folgenden nur C-Syntax) und wird mittels des Kommandos *cc* gestartet. In der Praxis wird heute gerne der frei verfügbare GNU-C-Compiler *gcc* verwendet. Er ist auch unter Knoppix verfügbar. C verwendet ein anderes Konsolen-I/O-System als C++. Statt *cin* müssen wir die etwas umständlichere *printf*-Funktion für Bildschirm-Ausgaben verwenden (→ s. Band 1, Kap. 8.4).

■ **Beispiel:**

Ein einfaches C-Programm:

```
#include <stdio.h>
int main(void)
{
   printf("Mein erstes C-Programm unter UNIX \n");
   getchar();
   return 0;
}
```
■

Das Programm soll in der Datei *prog1.c* im aktuellen Verzeichnis (working directory) stehen. Zur Quelltext-Eingabe empfehlen wir, falls verfügbar, den Editor *joe* (z. B. joe prog1.c), der intuitiv zu bedienen ist, leider ohne Maus-Unterstützung. Beendet, incl. Speichern des eingegebenen Textes, wird er mit <Strg>+<k> und dann <x>.

Zur Compilierung dient das UNIX-Kommando

cc –oder- **gcc**

Es bewirkt gleichzeitig das Binden, so dass sofort ein ausführbares Programm entsteht:

```
gcc prog1.c
```

Das ausführbare Programm gelangt automatisch auf die Datei

a.out

im gleichen Verzeichnis. Sie können nun das Programm durch Eingabe von

```
a.out
```

ausführen. Natürlich wird *a.out* durch Compilierung neuer Programme immer wieder überschrieben. Wollen Sie dies verhindern, so können Sie dem ausführbaren Programm auch einen frei definierten Namen zuweisen, indem Sie die -o-Option des cc- oder gcc-Kommandos anwenden:

```
gcc -o prog1 prog1.c
```

Das ausführbare Programm steht nun in

prog1

und kann durch entsprechende Eingabe gestartet werden.

Ausführbare Programme benötigen unter Unix keine besondere Extension (z. B. exe) wie unter Windows. Sie benötigen jedoch das Ausführrecht, z. B. mittels des Befehls

`chmod u+x prog1` (user erhält zusätzlich Ausführrecht auf die Datei *prog1*).

Bei komplexeren Anwendungen ist es oft sinnvoll, mehrere Programme, genauer gesagt Prozesse, an ein und derselben Aufgabe arbeiten zu lassen. Z. B. horcht ein Prozess die TCP/IP-

Schnittstelle ab während ein anderer die von dort empfangenen Befehle ausführt. Nachstehend einige einfache und kurze C-Beispiele zur Prozesssteuerung und Synchronisation mit Systemaufrufen zum Ausprobieren:

■ **Beispiele:**

```
/*************************************************/
/*  Systemaufrufe fork(), wait() und exit()      */
/*************************************************/
#include <stdio.h>
int main(void)
{
  int x = 1, status, pid;
  printf("Elternprozess: x = %d\n", x);
  if((pid = fork()) == 0) /* Kindprozess erzeugen */
  {
    printf("Kindprozess: x = %d\n mit pid %d\n ", x, pid);
    /* Ausgabe 1 */
    x = 2; /* betrifft nur die x-Variable
              des Kindprozesses*/
    printf("Kindprozess: x = %d\n", x); /* Ausgabe 2 */
    exit(0); /* Kindprozess beenden */
  }
  wait(&status); /* Elternprozess wartet auf Ende
                    des Kindprozess */
  printf("Elternprozess: Status Kind = %d\n", status);
  printf("Elternprozess: x = %d\n", x); /* Ausgabe 1 */
  return 0;
}
```

```
/*************************************************/
/*  Systemaufruf kill()                          */
/*************************************************/
#include <signal.h>
#include <stdio.h>
int main(void)
{
  int child_pid;
  long x = 1;
  if((child_pid = fork())== 0)
    /* Endlosschleife des Child-Prozesses mit Ausgabe */
    for(;;)
      printf("Kindprozess: %d\n", x++);
  /* Parent wartet 10 Sekunden */
```

```
  sleep(10);
  /* Parent beendet child durch Senden des
     Signals SIGKILL */
  kill(child_pid, SIGKILL);
  return 0;
}
```

Eine Pipe ist ein FIFO-Puffer (Fist In First Out) zur Datenübergabe (**char** *) von einem Prozess zum anderen und damit ein wichtiges Interprozess-Kommunikations-Medium. Den Umgang mit Pipes, genauer gesagt „unnamed pipes“, demonstriert das folgende Beispiel.

```
/*********************************************************/
/* Interprozess-Kommunikation unter UNIX mit Pipes       */
/*********************************************************/
#include <stdio.h>
int main(void)
{
  char puffer[81];  /* Puffer zum Datenempfang           */
  int pd[2];  /* Deskriptoren fuer Leseende (pd [0])
                 und Schreibende (pd [1]) der Pipe       */
 /* Erzeugung einer unnamed pipe                         */
 printf("Parent erzeugt die unnamed pipe\n");
 pipe(pd);
 if(fork()==0)
 {
    /* Childprozess schreibt*/
    /* nicht benoetigten Lesedeskriptor schliessen       */
    close(pd[0]);
    /* Text in die Pipe schreiben                        */
    printf("Child schreibt nun einen Text in die Pipe\n");
    write(pd[1], "Kind gruesst seine Eltern", 81);
    exit(0); /* Child beenden                            */
  }

  /* Parentprozess liest                                 */
  /* nicht benoetigten Schreibdeskriptor schliessen      */
  close(pd[1]);
  /* Pipe auslesen                                       */
  read(pd[0], puffer ,81);
  printf("\nParent liest aus der Pipe:\n %s\n", buffer);
  return 0;
}
```

■

Übersicht: Systemaufrufe zur Erzeugung von Kindprozessen (childs)

fork(): Kindprozess läuft mit gleichem Speicher-„Image“ (Multithreading)

system(): Kindprozess läuft in eigener Speicherumgebung (keine Parameterübergabe beim Aufruf möglich; ggf. *pipes* nutzen! (Multitasking)

exec(): Kindprozess läuft in eigener Speicherumgebung, Parameterübergabe beim Aufruf ist möglich. Es gibt jedoch keine Rückkehr zum Elternprozess, da dieser beim Aufruf beendet wird.

Weitere wichtige Interprozess-Kommunikations-Mittel sind *named pipes* (memory mapped files) und *Semaphore*, die, ähnlich einer Ampelanlage, den Zugriff auf Ressourcen koordinieren. All diese Möglichkeiten sind in der C-Schnittstelle von UNIX implementiert. Unter Windows muss man sie sich bei Bedarf mühsam selber erzeugen oder sehr spezielle Windowslösungen verwenden. In jedem Fall lohnt ein Blick ins Internet unter den entsprechenden Stichwörtern.

10.3 Aufgaben

Aufgabe 1:

Schreiben Sie mit Hilfe der `if-then-else`-Struktur eine Prozedur, mit der entweder der Inhalt der im Aufruf angegebenen Datei, der Inhalt des Verzeichnisses oder die Meldung "`Datei ist eine b-Datei`", "`Datei ist eine c-Datei`" bzw. "`Datei existiert nicht`" ausgegeben wird (auf `stdout` = Bildschirm).

Hinweis: In Bedingungen kann der Ausdruck `test` fehlen. Es genügt die Angabe der Option, z.B. `if test -f $1` lässt sich auch schreiben als `if [ -f $1 ]`, allerdings nicht bei Berkeley-Systemen. Achtung: die Leerzeichen hinter „[“ und vor „]“ sind zwingend. c- und b-Dateien sind Gerätedateien (s. Verzeichnis `/dev`).

Aufgabe 2:

Schreiben Sie eine SHELL-Prozedur, die beim Aufruf mit `aufg_2 a b` *f*olgendes ausgibt:

```
Aufruf: aufg_2
Anzahl Argumente: 2
Prozessnummer: ...
1. Argument: a
2. Argument: b
```

u.s.w.

Halten Sie die Anzahl der Argumente flexibel. Dazu benötigen Sie das Kommando `shift`. Es bewirkt in einer SHELL-Prozedur, dass alle Übergabe-Parameter um eine Position nach links geschoben werden. So können Sie beispielsweise alle Übergabe-Parameter nacheinander in einer Schleife der Stellungsvariablen 1 zuweisen. Damit ist die Übergabe beliebig vieler Parameter möglich.

Beispiel:

```
until test -z $1 # Ausgabe aller uebergebenen Zeichenketten
do
  cat $1
  shift
done
```

Aufgabe 3:

Schreiben Sie eine Prozedur, die Datei_1 in die Datei_2 kopiert. Die Dateinamen sollen als Stellungsparameter übergeben werden. Die Prozedur soll auch mehrere Dateien kopieren können (durch Angabe mehrerer Dateinamen, der erste ist Zieldatei).

Hinweis: Auch hier benötigen Sie das `shift`-Kommando. Benutzen Sie `cat` zum kopieren. Das Ausgabe-Umlenkzeichen für Anhängen ist „>>“.

Aufgabe 4:

Schreiben Sie eine Prozedur, die ein Verzeichnis nach leeren Dateien durchsucht, die Namen in eine Kontrolldatei schreibt und die gefundenen leeren Dateien interaktiv löscht.

Aufgabe 5:

Schreiben Sie eine SHELL-Prozedur, die die Quadratzahlen von `n` bis `m` berechnet und ausgibt. `n` und `m` sollen Eingabeparameter sein.

Aufgabe 6:

Schreiben Sie ein C-Programm, das mit `system()` einen parallelen Prozess startet (Multitasking, zweites C-Programm), der ohne Ausgaben zu tätigen in einer Dauerschleife läuft. Auch der Erzeugerprozess soll nach Start des zweiten Programms in eine Endlosschleife ohne Ausgabe gehen. Starten Sie das Programm im Hintergrund durch Anhängen des `&`-Zeichens (z. B. `aufg6&`). Ermitteln Sie von der SHELL aus die PIDs (`ps`-Kommando) und killen Sie die Prozesse mit dem UNIX-Kommando `kill`.

Teil II RECHNERNETZE

Innerhalb der letzten 20 Jahre hat sich in der Datenverarbeitung ein grundlegender Wandel vollzogen, der durch die beiden Schlagworte „PC“ und „Internet“ gekennzeichnet ist. Mit den immer leistungsfähigeren Arbeitsplatzrechnern (PCs und Workstations) ist man zunehmend weniger auf Rechenleistungen zentraler Anlagen angewiesen. Die kurzen Innovationszyklen der Hardware-Entwicklungen und der damit einhergehende Preisverfall von PCs ermöglichen eine Rechenkapazität vor Ort am Arbeitsplatz des Nutzers, die bisher nur von Großrechnern erbracht werden konnte.

Damit die Einzelplatzsysteme nicht isoliert sind und weiterhin auch auf zentrale Datenbestände, wie z. B. Lagerbestände oder Kundendateien zugreifen können, werden sie in Netzwerke eingebunden. Nachdem zunächst die Bürowelt die Vorteile der Vernetzung nutzte, ist sie nunmehr auch in den komplexeren Bereich von Wissenschaft und Technik eingedrungen. Durch die Einbindung der Arbeitsplätze in ein „Lokales Netz“ (LAN, *Local Area Network*) können zentrale „Dienste“, die von „Servern“ im Netz angeboten werden, wie z. B. Fileserver-Dienste oder Printserver-Dienste, genutzt werden. Durch die Entwicklung des Internet ist eine Plattform für eine weltweite, globale Kommunikation entstanden. Die räumliche Entfernung zwischen kommunizierenden Rechnern ist unwichtig geworden! Parallel zu den Hardware-Entwicklungen hat die Qualität der Netz-Anwendungen ständig zugenommen. Die praktische Nutzung von Rechnernetzen ist heute für alle Schichten der Gesellschaft unverzichtbar geworden.

Die durch das Netzwerk erreichte Rechner-Rechner-Kommunikation kann unterschiedlich stark in die Prozesse der Arbeitsplatzrechner eingreifen:

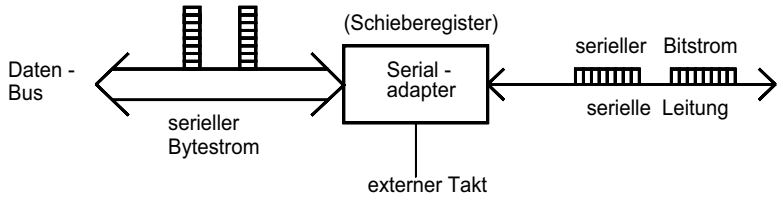
Formen der Rechnerkommunikation in Netzen

1. schwache Kopplung:
   – Server-Dienste (z. B. File- und Print-Dienste) werden nur gelegentlich in Anspruch genommen.
2. mittlere Kopplung:
   – Client/Server-Anwendungen:
     Die auf den Arbeitsstationen laufende Client-Software ist auf einen im Server laufenden Server-Prozess abgestimmt und kommuniziert mit diesem.
3. starke Kopplung:
   – verteilte Anwendungen:
     Eine Anwendung wird auf mehrere Rechner im Netz verteilt.

In den nachfolgenden Kapiteln sollen die beiden zentralen Aspekte von Rechnernetzen behandelt werden: Die Technik des Netzaufbaus und der Mechanismus des Datenaustausches innerhalb und zwischen Netzwerken.

11 Die Serielle Datenübertragung

Rechner können über serielle Interfaces mit anderen Rechnern oder Geräten kommunizieren. Kernkomponente eines seriellen Interfaces ist ein extern getaktetes Schieberegister, das die parallel/serielle-Wandlung vornimmt.



Prinzip der seriellen Datenübertragung

Wir unterscheiden zwischen der asynchronen und der synchronen Datenübertragung.

11.1 Die asynchrone Datenübertragung

Ein Taktgenerator (Baudratengenerator) bestimmt den Schritt-Takt, mit dem gesendet bzw. empfangen wird. Ist T die Dauer eines Schritttaktes, so wird die

Schrittgeschwindigkeit: $$v_s = \frac{1}{T} \quad [1/s] \quad \text{oder} \, [\text{Baud}].$$

Da je Schritt genau 1 Bit übertragen wird (in der Übertragungstechnik können durch mehrstufige Codes auch mehr als 1 Bit je Schritt vorkommen!), ergibt die

Übertragungsgeschwindigkeit: $$v_{ü} = \frac{1 \text{ bit}}{T} [\text{bit/s}].$$

Die asynchrone Übertragung ist eine Einzelzeichenübertragung, die durch die folgenden Merkmale gekennzeichnet ist:

- die Übertragung ist zeichengebunden;
- *asynchron* bedeutet, dass jedes Zeichen zeitlich unabhängig von anderen übertragen wird;
- Der Empfänger synchronisiert sich auf jedes eintreffende Zeichen neu;
- Jedes Zeichen ist durch Start- und Stoppbits gerahmt;
- Das Datenformat ist:

1 Startbit (LOW)	St
5-8 Datenbits	D
[optional: 1 Paritätsbit even oder odd	P]
1 oder 2 Stoppbits (HIGH)	Sp

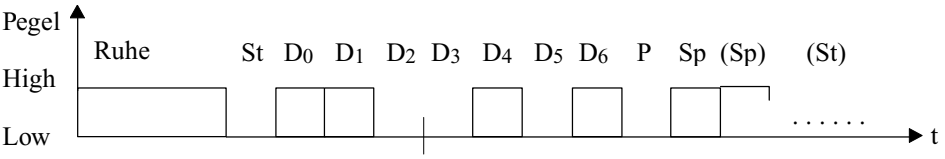
Die Signalzustände auf der Leitung werden an einem Beispiel deutlich:

■ **Beispiel: Übertragung des ASCII-Zeichens „S“**

ASCII „S“ :53H 1 0 1 0 0 1 1

D_6 D_0

Parität even : ➔ P = 0



■

Der Empfänger synchronisiert sich auf die fallende Flanke des Startbits. Durch die Stoppbits wird die HIGH-Ausgangslage wieder eingenommen und ein neues Startbit erkennbar. Wegen des Start/Stopp-Bit-Rahmens nennt man die asynchrone Übertragung auch „Start-Stopp-Betrieb“. Vor Beginn einer Datenübertragung müssen sich Sender und Empfänger auf die gleichen „Protokollparameter“ einstellen. Die Übertragungsgeschwindigkeiten sind in genormten Stufen wählbar.

Protokollparameter der asynchronen Datenübertragung

Codelänge:	5-8 Bit
Paritätsbit:	even, odd, none
Stoppbit(s):	1 oder 2 Bit
Übertragungsgeschwindigkeit:	19200 Bit/s
	9600 Bit/s
	4800 Bit/s
	2400 Bit/s
	1200 Bit/s
	600 Bit/s
	300 Bit/s
	150 Bit/s
	110 Bit/s
zusätzlich im PC-Bereich:	115200 Bit/s
	57600 Bit/s
	38400 Bit/s

Die relativ niedrigen Übertragungsgeschwindigkeiten und der für jedes Einzelzeichen erforderliche „Overhead“ durch Start- und Stopp-Bit führen zu einer schlechten Leitungsausnutzung.

Das asynchrone Übertragungsverfahren wird bei Datenübertragungen über die bekannte RS232C- (V.24-) Schnittstelle angewendet.

11.2 Die synchrone Datenübertragung

Im Gegensatz zur asynchronen Datenübertragung (z. B. über eine RS232C/ V24-Schnittstelle, s. u.) benutzen digitale Netze das synchrone Datenübertragungsverfahren. Hier entfallen Start- und Stoppbits, die Daten werden in Paketen zu meist mehreren hundert Byte zusammengefasst und Paketweise übertragen. Die Übertragungsgeschwindigkeit liegt meistens erheblich über der asynchroner Verfahren und erstreckt sich bis zu einigen hundert MBit/s.

Die Übertragung kann entweder mit dem (älteren) zeichenorientierten oder dem (moderneren) bitorientierten Verfahren erfolgen.

Synchrone Übertragungsverfahren

Zeichenorientiertes Verfahren:

- Die Übertragung ist codegebunden.
- Die Steuerung der Übertragung wird durch spezielle Codeworte vorgenommen (z. B. ASCII Steuerzeichen).
- Zur Bytesynchronisation dient ein spezielles Synchronisationsbyte, das mehrmals vor einem Datenpaket gesendet wird.

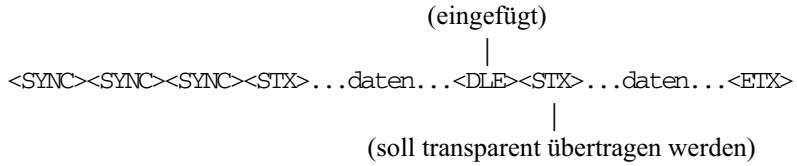
Bitorientiertes Verfahren:

- Die Übertragung ist codetransparent.
- Steuerinformation im Datenpaket ist durch die **Lage** innerhalb des Paketrahmens identifizierbar.
- Zur Bitsynchronisation dient ein charakteristisches FLAG-Bitmuster zu Beginn und am Ende des Pakets.

Ein entscheidender Vorteil der bitorientierten Verfahren liegt in ihrer Transparenz: Im Datenteil können beliebige Bitmuster „transparent“ übertragen werden, eine Missinterpretation als Steuercode ist ausgeschlossen.
Um Transparenz beim zeichenorientierten Verfahren zu erreichen, müssen besondere Vorkehrungen getroffen werden: Tritt innerhalb des Datenteils der Code eines Steuerwortes auf, wird das entsprechende Byte durch Vorschalten eines Ausweichzeichens (<DLE>, ASCII #16) kenntlich gemacht. Der Empfänger entfernt dieses Ausweichzeichen anschließend wieder.

Synchrone Datenübertragungsverfahren

a) Zeichenorientiertes Verfahren:
Beispiel eines Paketaufbaus:



<SYNC> : Synchronisationsbyte, ASCII #22
<STX> : Start of Text, ASCII #2
...daten... : Nutzdaten
<DLE> : Data Link Escape, Ausweichzeichen, ASCII #16
<ETX> : End of Text, ASCII #3

b) Bitorientiertes Verfahren:

Beispiel eines Rahmenaufbaus:

Flag	Adressfeld	Steuerfeld	Datenfeld	Blockprüf-zeichen	Flag

z. B. 128 Byte

Flag: `01111110`
Adressfeld: Absende- und Empfängeradresse
Steuerfeld: Länge des nachfolgenden Datenfeldes, Art des Datenpakets
Datenfeld: Nutzdaten
Blockprüfzeichen: Fehlerprüfcode (→ s. Kap. 11.6)

Beispiel:
Datenübertragungen über das ISDN-Netz erfolgen bittransparent und synchron.

11.3 Datenübertragung über die RS232C/V.24-Schnittstelle

11.3.1 Die RS232C/V.24-Schnittstelle

Fast jeder Rechner und auch Mikrocontrollerboards verfügen über eine oder mehrere RS232C- bzw. V.24-Schnittstellen, über die Daten nach dem asynchronen Verfahren übertragen werden können. Die Schnittstelle wird zur Unterscheidung von der Parallel-Schnittstelle auch einfach „die serielle Schnittstelle“ genannt und als „com-Port“ (Windows) oder „tty-Port“ (Linux) bezeichnet.

Die beiden Standards RS232C und V.24 unterscheiden sich praktisch nur durch die Leitungsbezeichnungen. Hier benutzen wir die Bezeichnungen nach RS232C.

Funktionale Eigenschaften:

Der Standard definiert 25 Signale, von denen aber nur wenige in der Praxis tatsächlich genutzt werden.

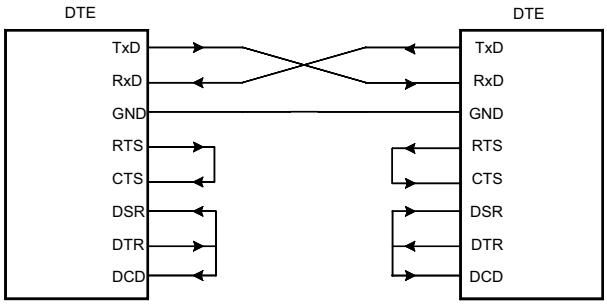
Die wichtigsten RS232C-Signale

DTE (*Data Terminal Equipment*), Endgerät z. B. Rechner

TxD	(*Transmit Data*) Sendedaten
RxD	(*Receive Data*) Empfangsdaten
GND	(*Ground*) Erde
RTS	(*Request To Send*) Sendeteil einschalten
CTS	(*Clear To Send*) Sendebereitschaft
DSR	(*Data Set Ready*) Betriebsbereitschaft
DTR	(*Data Terminal Ready*) Endgerät betriebsbereit
DCD	(*Data Carrier Detect*) Empfangssignalpegel

Die getrennten Sende- und Empfangsleitungen TxD und RxD ermöglichen einen bidirektionalen Datenverkehr. Die Signalpaare RTS/CTS und DSR/DTR sind HW-„Handshake“-Leitungen (Anforderung/Bestätigung), über die der Datenfluss gesteuert oder die Betriebsbereitschaft gegenseitig mitgeteilt werden kann. DCD zeigt bei Modemverbindung den empfangenen Signalpegel an.

Für Rechner-Rechner-Verbindungen (allgemeiner: DTE-DTE-Verbindungen) benötigt man sog. Nullmodemkabel. Das am Häufigsten eingesetzte Nullmodemkabel besitzt nur 3 Leitungen:



Nullmodemkabel als 3-Draht-Verbindung

Typisch sind die gekreuzten Adern in Nullmodemkabeln. Nicht in jedem Fall sind die gezeigten Steckerbrücken alle unbedingt erforderlich. Dies hängt davon ab, ob die verwendete Kommunikationssoftware diese Leitungen tatsächlich abprüft. Durch die Brücken wird ein von der Gegenseite erwartetes Signal durch die eigene DTE „vorgetäuscht“.

Das mit einem Nullmodemkabel angeschlossene Endgerät (DTE) muss nicht unbedingt ein zweiter Rechner sein. Endgerät könnte auch ein peripheres Gerät sein, wie z. B. Drucker, Scanner oder Plotter, die neben ihrer parallelen Schnittstelle oft auch eine RD232C/V.24-Schnittstelle besitzen, so dass die Endgeräte bis zu 15 m entfernt voneinander betrieben werden können.

Da bei dieser Verbindung keine Leitungen für ein HW-Handshake zur Datenflusskontrolle vorhanden sind, benutzt man eine Software-Methode:

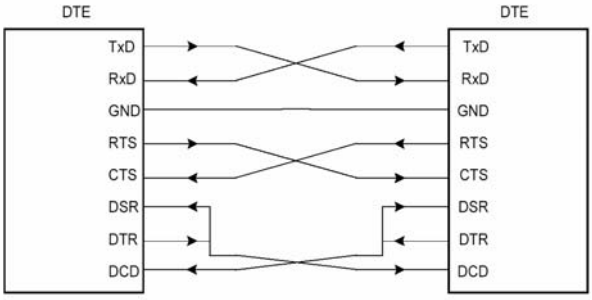
Das XON/XOFF - Protokoll

Software-Methode zur Datenflusssteuerung;
Die Datenleitungen werden zur Steuerung mitbenutzt;
Die Steuerung erfolgt über 2 ASCII-Kontrollzeichen:

- ASCII #19: XOFF
- ASCII #17: XON

Kann der Empfänger die eintreffenden Daten nicht mehr verarbeiten, sendet er „XOFF“. Der Sender „hört“ während seiner Sendung seine RxD-Leitung ab. Empfängt er das Zeichen „XOFF“, stoppt er seine Sendung solange, bis der Empfänger mit „XON“ signalisiert, dass er zur weiteren Datenaufnahme bereit ist.

Ein Beispiel für eine Mehrdraht-Verbindung zeigt das folgende Bild:



Nullmodemkabel als Mehrdraht-Verbindung

Hier kann RTS/CTS zur HW-Flusssteuerung benutzt werden. Zusätzlich lässt sich mit DCD eine allgemeine Betriebsbereitschaft abprüfen.

Elektrische Eigenschaften:

RS232C/V.24-Schnittstellen arbeiten bipolar mit +12V/-12V-Pegeln. Da Rechner oder Controller intern meistens mit TTL-Pegeln arbeiten, ist ein Pegelumsetzer erforderlich. Es ist festgelegt:

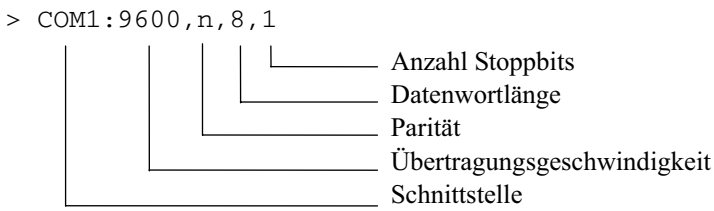
-12 V ... – 3 V : Logisch „Eins“
- 3 V ... + 3 V : undefiniert
+ 3 V ... +12 V : Logisch „Null“

Mechanische Eigenschaften:
Der Standard beschränkt die maximale Leitungslänge von Verbindungskabeln auf 15m. Die mechanische Verbindung kann entweder als 25-polige (immer seltener) oder als 9-polige D-Sub-Steckverbindung (meistens) ausgebildet sein. Die DTE hat Stifte (Unterschied zur Parallel-Schnittstelle!), ein Nullmodemkabel hat also an beiden Enden Buchsen. Die Pinbelegung ist wie folgt festgelegt:

Pinbelegung

25polig Pin-Nr.	Signal	9polig Pin-Nr.
2	TxD	3
3	RxD	2
4	RTS	7
5	CTS	8
6	DSR	6
1,7	GND	5
8	DCD	1
20	DTR	4
22	RI	9

Die Initialisierung der seriellen Schnittstelle kann direkt durch das Betriebssystem oder indirekt über Kommunikationsprogramme vorgenommen werden. Die Schnittstellenparameter werden häufig in der folgenden Kurzform dargestellt:



11.3.2 Terminalemulation und Kommunikationsprogramme

Die einfachste Rechner-Rechner-Kopplung besteht in einer Punkt-zu-Punkt Verbindung zweier Rechner, die über die serielle Schnittstelle verbunden sind. Auf beiden Rechnern wird eine spezielle Anwendung, ein Terminalprogramm (Terminalemulation) gestartet, das die jeweilige Schnittstelle anspricht. Die verbundenen Endgeräte verhalten sich dann wie zwei miteinander verbundene Terminals. Werden dabei durch die Software spezielle Eigenschaften eines bestimmten Terminals „xyz“ nachgebildet, so spricht man von einer Terminalemulation des Gerätes xyz.

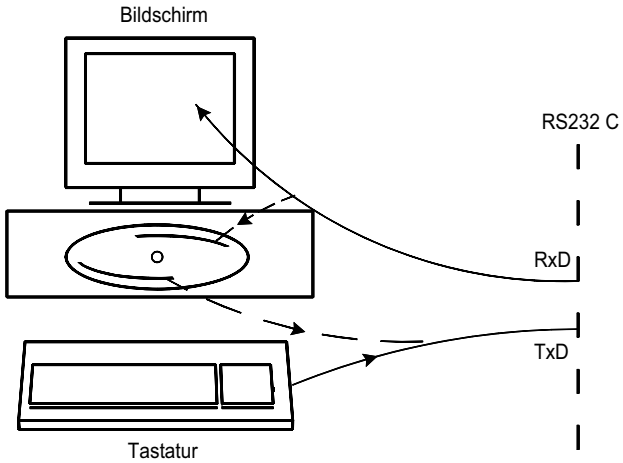
Die Begriffe „Emulation“ und „Simulation“ in der DV

Emulation:	Vollständige Nachbildung eines Gerätes durch Software. Der emulierende Rechner steht leistungsmäßig über dem Gerät, das emuliert wird.
Simulation:	Approximative Nachbildung eines Gerätes oder eines Prozesses durch Software. Der simulierende Rechner kann nicht alle Eigenschaften des simulierten Originals darstellen.

Verbreitete Emulationen sind „VT100“, eine Emulation der DEC VT100-Terminalfamilie, oder „ANSI“ (*American National Standards Institut*), ein genormtes Standard-Terminal.

Emulationsprogramme nennt man häufig auch einfach „Terminalprogramme“.

Die Grundfunktion eines Terminalprogramms macht die folgende Skizze deutlich:



Terminalemulation

Bei jedem Tastendruck wird das entsprechende ASCII-Zeichen über die TxD-Leitung der RS232-Schnittstelle gesendet. Über RxD eintreffende Zeichen werden am Bildschirm dargestellt. Alternativ können Daten auch von einem Datenträger (Platte) gesendet und empfangen werden.

Läuft auf einem Arbeitsplatzrechner eine Terminalemulation, so ist für diese Anwendung die Intelligenz des lokalen Rechners praktisch ausgeschaltet, der Nutzer arbeitet mit der CPU des Partnerrechners.

Obwohl Terminalprogramme die Grafikeigenschaften bestimmter Grafikterminals emulieren können, werden sie in der Praxis hauptsächlich für textorientierte Kommunikation eingesetzt. Einige Haupteinsatzbereiche sind:

- Lokale PC <-> (Multiuser-) Host-Kommunikation
  (z. B. PC<->Unix-System)
- Fernzugriffe über das Telefonnetz auf entfernte Hosts oder Netz-Einwähl-Rechner (Remote Access)
- Fernsteuerung von PCs zu Wartungszwecken (Remote Control: nur Bildschirm und Tastatur werden übertragen)
- Lokaler „out-of-band“ Zugriff (Zugriff nicht über das Netzmedium) auf Netzkoppelelemente zur Konfigurierung und Wartung (Router, Bridges, Switches usw.)

Während bei reinen Terminalprogrammen der Dialog im Vordergrund steht, besitzen allgemeinere „Kommunikationsprogramme“ darüber hinaus Möglichkeiten eines gesicherten Filetransfers. Beide Kommunikationspartner benutzen das gleiche „Übertragungsprotokoll“, einen Satz von Regeln, die vorschreiben, wie die Datenpakete zu konstruieren sind und welche Fehlersicherungsmaßnahmen getroffen werden. Verbreitete Kommunikationsprotokolle für die serielle Übertragung sind *Kermit*, *Xmodem* und *Zmodem*.

Eine über die RS232C/V.24-Schnittstelle laufende Rechner-Rechner-Verbindung arbeitet zeichenorientiert. D. h. ein bestimmtes Übertragungsprotokoll benutzt zur Steuerung des Datentransfers Steuerzeichen, die in den ersten 32 ASCII-Tabellenpositionen definiert sind (z. B. XON/XOFF-Protokoll). Wegen dieser Abhängigkeit von Steuerzeichen spricht man von einer „nicht transparenten“ Datenübertragung.

Ein Problem tritt nun aber auf, wenn nicht nur reine Textdateien, sondern auch Binärdateien, z. B. vom Typ *.exe übertragen werden sollen. Hier ist es durchaus möglich, dass einzelne Bytes gerade den Bitkombinationen von Steuerzeichen entsprechen und als solche eine ungewollte Steuerwirkung auslösen. Die Folge wäre eine Störung oder der Abbruch des Filetransfers. Binärdateien müssen daher vor dem Senden so umcodiert werden, dass keine Steuerzeichen auftreten. Eine andere Methode setzt vor Bytes mit Steuerwirkung ein spezielles „Flagbyte“, das im Übertragungsprotokoll vereinbart ist und das dem Empfänger signalisiert, dass das nachfolgende Byte „transparent“ übernommen werden soll. Kommunikationsprogramme haben aus diesem Grund eine Einstellmöglichkeit für Text- (ASCII-) oder Binärdateien.

Dateitransfer über die RS232C/V.24-Schnittstelle

Die Übertragung ist zeichenorientiert

Die Übertragung ist nicht transparent

Bei Übertragung von Binärdateien müssen besondere Vorkehrungen getroffen werden

Kommunikationsprogramme werden auch dazu eingesetzt, eine Punkt-zu-Punkt Verbindung über Modemstrecken aufzubauen und zu steuern (→ s. nächstes Kapitel). Sie übernehmen die Modeminitialisierung und die Verwaltung anwählbarer Gegenstellen („Telefonbuch"). *Telix* (DOS, Windows), *SCOM32* (Windows), *Hyperterm* (Windows) oder *minicom* (Linux) sind häufig eingesetzte Kommunikationsprogramme.

USB-Schnittstellen werden von Terminalprogrammen über emulierte virtuelle com-Ports angesprochen. Die USB-Schnittstelle lässt sich so wie eine Standard-RS232C-Schnittstelle konfigurieren und nutzen. Einige Kommunikationsprogramme können statt mit den seriellen Schnittstellen auch mit einer ISDN-Karte (CAPI-Schnittstelle, → s. Kap. 11.5) oder einem Netzwerkadapter zusammenarbeiten. Im letzteren Fall ergibt sich eine Terminalemulation über das Lokale Netz oder sogar das Internet.

11.4 Datenübertragung mit Modems

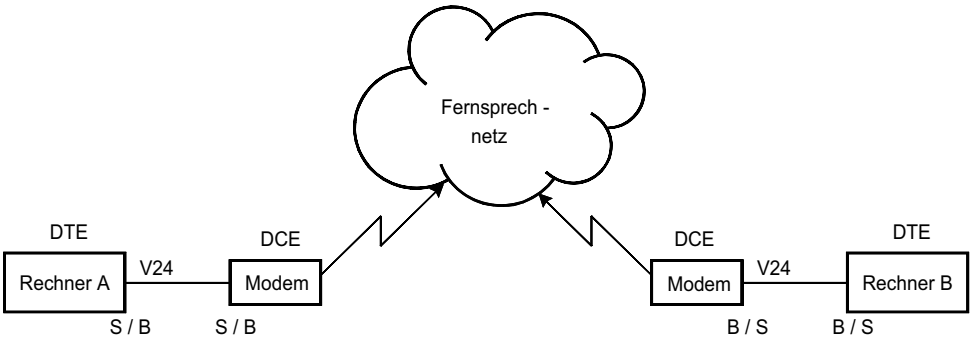
Modems (MODulator – DEModulator) sind Geräte, die eine Übertragung digitaler Informationen über einen analogen Fernsprechanschluss ermöglichen. Als Geräte für die „Datenfernübertragung" (*DCE Data Communication Equipment*) werden Modems in zwei Anwendungsumgebungen eingesetzt:

- Internet-Zugang zu einem ISP*(Internet Service Provider*) über das PPP-Protokoll.
- „Verlängerung" eines seriellen Kabels durch eine Modem-Modem-Kopplung, die durch Kommunikationsprogramme gesteuert wird.

Internet-Zugänge über Modems werden im Kap. 18.4 behandelt. Beispiele für Modemeinsätze im „Nicht-Internet-Umfeld" sind:

- Rechnergestützte, automatisch arbeitende Messstationen übertragen in regelmäßigen Zeitabständen ihre Messdaten per Modemverbindung an einen Auswerterechner.
- Ein mit Modem ausgestatteter Hochschul- oder Firmenrechner ist per Modem anwählbar.
- Ein normalerweise nur für die Internetanbindung genutztes Modem kann auch dazu eingesetzt werden, einen ebenfalls mit Modem ausgestatteten Rechner eines Bekannten anzuwählen und mit ihm –unabhängig vom Internet und dessen Infrastruktur- Daten auszutauschen über Terminalprogramme.
- Öffentlich anwählbare Mailboxen zum Informations- und Softwaredownload.
- Fernwartung von Rechnern.
- Modem-gekoppelte Rechner verhalten sich nach dem Verbindungsaufbau wie mit Nullmodemkabel direkt verbundene Endgeräte.

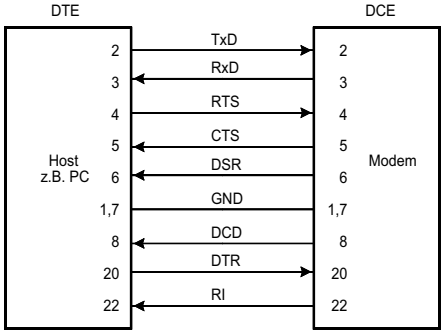
Ein externes Modem wird über ein Modemkabel mit der RS232C/V.24-Schnittstelle (com Port) des Rechners verbunden. Modems mit USB-Schnittstelle oder interne Modems emulieren einen virtuellen com-Port, so dass auch sie wie externe Modems konfiguriert werden.



Rechnerkopplung über Modems S: Stecker, z. B: Buchse

Modemkabel enthalten im Gegensatz zu Nullmodemkabel keine gekreuzten Adern (es liegt zwischen DTE und DCE keine Symmetrie vor!); sie sehen aus wie „Verlängerungskabel“ mit Stecker und Buchse.

Ein Modemkabel sollte die folgenden 9 Schnittstellensignale führen:



Verbindung Host ↔ Modem, 25polige Steckverbindung

Gegenüber DTE-DTE Verbindungen ist für den Modemanschluss ein zusätzliches Signal aufgenommen: RI (Ring Indicator), Signalisierung eines ankommenden Rufs („klingeln“). Dieses Signal kann von der Host-Software zur Rufannahme benutzt werden.

Die wichtigsten Eigenschaften von Modems sind in Standards festgelegt:

Wichtige Standards und Normen für Modems

AT-Steuerung	HAYES-kompatibler AT-Befehlssatz zur Steuerung von Modems (entspricht CCITT V.25, aber nicht kompatibel)
CCITT V.32	bis 9600 Bit/s duplex
CCITT V.32bis	bis 14400 Bit/s duplex
ITU V.34	bis 28800 Bit/s duplex
ITU V.34+v.34bis	bis 33600 Bit/s duplex
ITU V.90	Internetzugang: 56kBit/s downstream / 33,6kBit/s upstream
CCITT V.100	automatische Baudratenerkennung
CCITT V.42	Fehlerkorrekturverfahren LAPM (Link Access Procedure for Modems) es wird auch MNP2-4 unterstützt
CCITT V.42bis	Kompressionsverfahren für V.42
MNP1-4	Fehlerkorrekturverfahren (Microcom Networking Protocol)
MNP5	Kompressionsverfahren (für MNP4)

Die Übertragungsraten der Hochgeschwindigkeitsmodems beruhen auf dem „Multi-Level-Encoding"-Verfahren, eine Kombination von Amplituden- und Phasenmodulation. Dabei werden je Übertragungsschritt (Baud) mehrere Bits in dem nur 2700 Hz breiten Fernsprechfrequenzband übertragen (vgl. Kap. 11.1 zum Unterschied Schrittgeschwindigkeit ↔ Übertragungsgeschwindigkeit). Übertragungen mit 56kbit/s setzen eine absolut störungsfreie Fernsprechverbindung voraus. Da dies jedoch in der Praxis häufig nicht der Fall ist, lassen sich die Modems so konfigurieren, dass sie bei Leitungsstörungen einen automatischen „Fallback" zu tieferen Übertragungsraten durchführen.

Der ITU V.90-Standard (*International Telecommunications Union*) ist speziell für die Internetanbindung entwickelt worden. Die hohe downstream Übertragungsrate von 56Kbit/s wird durch sog. „V.90-Hosts" des Internet-Dienstanbieters (*ISP Internet Service Provider*) erreicht. Diese Datenrate ergibt sich durch die Ausnutzung des heute praktisch vollständig in digitaler Technik ausgelegten Fernsprechnetzes und der Einsparung der Analogwandlung zwischen V.90-Host und der digitalen Vermittlungsstelle. Lediglich auf der Nutzerseite bleibt der analoge Zugang per Modem bestehen. Die höchste Übertragungsgeschwindigkeit einer echten Modem-Modemverbindung bleibt dagegen auf 33.6 Kbit/s beschränkt.

Daten-Kompressionsverfahren erhöhen den effektiven Datendurchsatz und verringern dadurch die Verbindungskosten. Kompressionsverfahren dürfen nur für fehlerkorrigierte Verbindungen (V.42/MNP4) eingesetzt werden, weil sich anderenfalls die Fehler vervielfachen! Damit sich eine Kompression auch tatsächlich auswirken kann, muss die Übertragungsgeschwindigkeit PC ↔ Modem höher sein als die, die auf der Telefonseite benutzt wird.

Zur Programmierung und Initialisierung von Modems hat sich in der Praxis der HAYES-Befehlssatz gegen die entsprechende CCITT Norm (V.25) durchgesetzt (de-facto-Standard). Diese Befehle können vom PC aus interaktiv an das Modem gesendet werden. Kommunikationsprogramme benutzen häufig einen „Modem-Initialisierungs-String", um das Modem zu

konfigurieren. Die nachfolgenden AT-Befehle können in der Form *AT<befehl>* von jedem Terminalprogramm aus an das Modem gesandt werden:

Beispiele einiger wichtiger Hayes-AT-Befehle (Auswahl)

Alle Kommandos werden mit „AT“ (Attention) eingeleitet

AT	Modem antwortet mit „OK“ (Verbindungstest)
A	Modem nimmt ab
D	Dial, wählt die nachfolgenden Zeichen und Ziffern
H0	Verbindung bricht ab, Modem legt auf
Ln	Lautsprecher Lautstärke n=0: aus oder sehr leise n=1: leise n=2: mittel n=3: laut
Mn	Lautsprecher n=0: immer aus n=1: ein bis Verbindung aufgebaut ist n=2: immer ein n=3: ein nachdem Verbindung aufgebaut ist
O	zurückschalten vom Kommando-Mode in den On-line-Mode (eingesetzt nach Escape-Kommando)
P	Impulswahl
Sn	Setzen und Lesen der internen Modem-Register n: Registernummer Sn=x: Beschreiben des Register n mit Wert x Sn? : Ausgeben des Registers n
T	Frequenzwahl
Vn	Form der Ergebnismeldungen an den Host n=0: Rückmeldungen als Zahl n=1: Rückmeldungen als Text
Zn	Soft-Reset und Konfigurationsprofil laden n=0: Profil 0 n=1: Profil 1

Die Anzahl der S-Register und ihre Bedeutung muss man der jeweiligen Gerätebeschreibung entnehmen.

Neben diesem Basis-Befehlssatz gibt es erweiterte Befehlssätze, die nicht genormt sind und daher stark vom Modemtyp abhängen ($-Befehle, %-Befehle, &-Befehle oder \-Befehle).

Der größte Teil der Einstellungen lässt sich in einem nichtflüchtigen Speicher in den Modems als abrufbares „Profil“ speichern und mit dem ATZ-Befehl aktivieren.

Sie können Telefon und Modem an der gleichen Fernsprechleitung (TAE-Dose mit F-N Zugängen) anschließen. Bei eingestecktem Modem hat dies stets Vorrang vor dem Telefon. Kommt ein Ruf an, wird zunächst das S0-Register geprüft. Falls S0 <> 0, nimmt das Modem den Ruf an, anderenfalls klingelt das Telefon. Praktisch gilt bei allen Modems:

S0=0: Modem nimmt keine ankommenden Rufe an!
Notwendig, falls an dem gleichen Fernsprechanschluss Telefongespräche ankommen.

S0=*n*: (*n*<>0) Modem nimmt Ruf nach *n* Klingelzeichen an. Notwendig, falls ein ankommender Ruf eine Verbindung zum Rechner aufbauen soll.

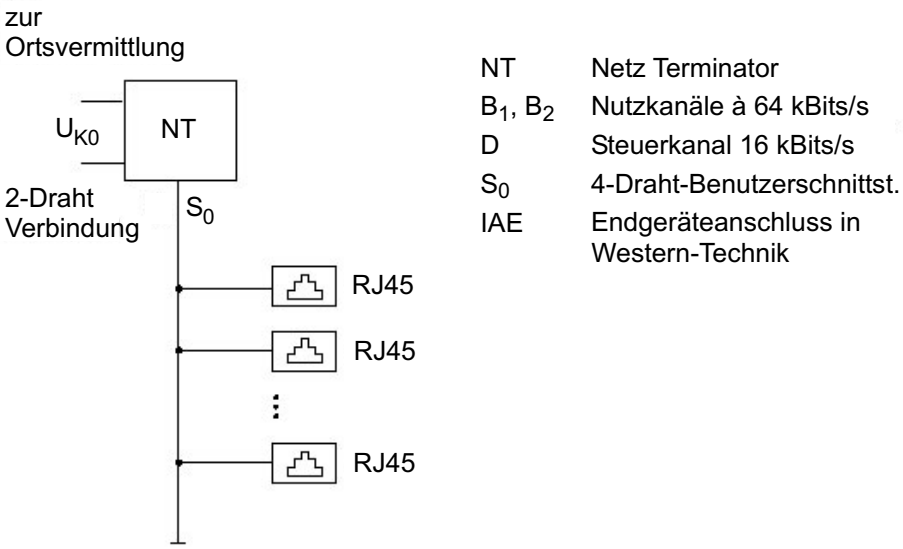
11.5 Datenübertragung mit ISDN

Für eine Rechnerkopplung über Weitverkehrsnetze ist ein rein digitales Netz natürlich erheblich besser geeignet als modemgestützte Verbindungen über analoge Telefonanschlüsse. Das ISDN (*Integrated Service Digital Network*) ist ein leitungsvermittelndes digitales Netz, das sich bis zum Teilnehmeranschluss erstreckt. Die Vorteile von ISDN,

- eine hohe Übertragungsgeschwindigkeit von 64Kbit/s oder 128Kbit/s
- eine hohe Störungsfreiheit,
- ein schneller Verbindungsaufbau,

wirken sich vor allem bei der Datenübertragung aus. Für Internetnutzer sind Dienstanbieter (ISP, *Internet Service Provider*) mit ISDN-Zugängen attraktiver als V.90 Zugänge.

ISDN-Anschlüsse gibt es in unterschiedlichen Konfigurationen. Die am Häufigsten genutzte Installation eines ISDN Basisanschluss (BRI, *Basic Rate Interface*) als Mehrgeräteanschluss ist in dem nachfolgendem Bild dargestellt:



ISDN Basisanschluss als Mehrgeräteanschluss

An dem bis zu 150m langen S0-Bus können bis zu 12 Steckdosen liegen, die mit maximal 8 Geräten bestückt sein dürfen. Jeweils 2 Endgeräte können unabhängig voneinander gleichzeitig über die beiden B-Kanäle betrieben werden. Andererseits ist es möglich, durch Kanalbündelung für eine Anwendung beide B-Kanäle parallel zu nutzen und damit eine Rate von 128 Kbit/s zu erzielen.

Der D-Kanal ist ein 16Kbit/s-Steuerkanal, über den die Informationen für den Aufbau, die Steuerung und den Abbau der Verbindung in Form des „D-Kanalprotokolls“ übertragen werden. Dies ist das „EURO-ISDN“, E-DSS1 (*Digital Subscriber System No1*) Protokoll, das inzwischen in über 30 Ländern benutzt wird.

Um an einem Mehrgeräteanschluss gezielt ein Endgerät anwählen zu können, werden mehrere Rufnummern (MSN, *Multiple Subscriber Number*) für einen ISDN-Anschluss von der Telekom eingerichtet.

ISDN bietet eine synchrone bittransparente Datenübertragung nach den folgenden Standards, die mit der Gegenstelle abgestimmt sein müssen:

Wichtige Standards für ISDN-Übertragungsverfahren

ITU X.75	gesicherte Übertragung von Daten im HDLC-Format im ISDN-B-Kanal mit einer festen Bitrate von 64000 Bit/s
ITU V.110	Anpassung asynchroner und synchroner serieller Datenströme an die ISDN-Bitrate von 64000 Bit/s im B-Kanal. Bei diesem als „Bitratenadaption“ bezeichneten Verfahren werden Füllbits in den Bitstrom eingefügt, wenn die digitale Gegenstelle nicht die volle ISDN-Bitrate unterstützt.
ITU V.120	(hauptsächlich in USA genutzt) Paketierung asynchroner und synchroner Daten in HDLC-Rahmen und Übertragung mit 64000 Bit/s oder 56000 Bit/s im B-Kanal

Es gibt zwei Möglichkeiten für den Anschluss eines PC an das ISDN:

- mit einer ISDN-Einsteckkarte (ISDN-Controller),
- über einen externen Terminaladapter (TA) mit serieller Schnittstelle.

Einsteckkarten besitzen eine S0-Schnittstelle mit RJ45-Buchse, über die direkt eine Verbindung mit dem ISDN hergestellt wird. SW-Anwendungen sprechen über die CAPI-Schnittstelle (*Common ISDN Application Interface*) den ISDN-Controller an. Die CAPI-Schnittstelle entspricht dabei der seriellen Schnittstelle bei Modem-Kommunikationsprogrammen. CAPI 1.0 und 1.1 sind PC-basierende Normen, CAPI 2.0 unterstützt dagegen beliebige Plattformen und hat sich deshalb auch im Unix-Umfeld durchgesetzt.

Externe Terminaladapter werden wie Modems an der seriellen Schnittstelle des Rechners betrieben. Netzseitig besitzen sie einen RJ45-Stecker. Der TA nimmt die asynchron/synchron Wandlung vor und wird mit einem der oben genannten Übertragungsverfahren konfiguriert. Dies geschieht wie bei Modems mit einem erweiterten AT-Befehlssatz. Als Kommunikations-Software kann jedes Terminalprogramm benutzt werden, das die serielle Schnittstelle unterstützt, nun aber die ISDN-Geschwindigkeit von 64 Kbit/s, bei Kanalbündelung sogar 128 Kbit/s nutzt. In jedem Fall sollte die Übertragungsrate zwischen PC und TA auf den maximal möglichen Wert (115 Kbit/s) eingestellt sein. Für den (meist seltenen) Fall einer Kanalbündelung sind TAs allerdings nachteilig: Hier wirkt die serielle Übertragung PC<->TA als Bremse.

Während zwischen analogen und ISDN-Telefonen problemlos kommuniziert werden kann, gilt dies nicht für die Datenübertragung. Rechner mit reinem ISDN-Interface können nicht mit analogen Gegenstellen, die mit Modems arbeiten, Daten austauschen! Aus diesem Grund besitzen TAs oft zusätzlich eine integrierte Modememulation, die auch eine Verbindung zu Modems gestattet. Der TA lässt sich so konfigurieren, dass zunächst versucht wird, eine ISDN-Verbindung aufzubauen. Ist das nicht möglich, wird der Modemteil aktiviert und eine Verbindung zu einem Modem als Gegenstelle aufgebaut.

Externe TAs haben gegenüber eingebauten PC-Einsteckkarten den Vorteil, dass an den Frontlämpchen des Geräts der Verbindungszustand jederzeit beobachtet werden kann. Vermeintlich beendete Verbindungen, die tatsächlich aber noch aktiv sind, können teuer werden. Ein Alptraum für die, die ihre Rechner „durchlaufen“ lassen!

11.6 Fehlersicherung

Um Fehler bei der Datenübertragung erkennen und korrigieren zu können, gibt es eine Vielzahl von Verfahren, von denen wir hier 3 unterschiedlich aufwendige vorstellen:

a) Parität

Jedes Byte kann durch ein Paritätsbit abgesichert sein. Die auf die physikalische Schnittstelle zugreifende Software erkennt 1-Bit-Fehler und kann veranlassen, dass das fehlerhafte Byte erneut übertragen wird. 2-Bit-Fehler bleiben unerkannt.

b) einfache Blockabsicherung

Ein Block hintereinander liegender Byte ist durch ein „Block Check Character“ (BCC) gesichert. Typische Blocklängen sind 10..1024 Byte. Das Check-Byte entsteht durch Addition oder durch eine logische XOR-Verknüpfung aller Bytes des Blocks und wird als zusätzliches „Zeichen“ an den Block angehängt.

■ **Beispiel**

Byte-Block (HEX-Notation):	BCC (ADD)	BCC (XOR)
`03 12 07 4F 30 A4 F1 7C 2B 65 ->`	`3C`	`0E`

■

Falls das beim Empfänger generierte Check-Byte nicht mit dem gesendeten übereinstimmt, fordert die Software eine Wiederholung der Blocksendung an.

c) CRC-Verfahren

Der „Cylic Redundancy Check“ (CRC) ist eine sehr verbreitete Methode der Fehlersicherung, die nicht nur bei Datenübertragungen zwischen verschiedenen Rechnern eine überragende Rolle spielt, sondern auch innerhalb eines Rechners, z. B. bei Datentransfers mit Festplatten und Disketten, intensiv eingesetzt wird. Im Gegensatz zu den oben genannten Verfahren ist die CRC-Methode nicht Byte-gebunden. Es können beliebige serielle Bitströme abgesichert werden. Die an den Bitstrom angefügten CRC-Bits bilden die „Frame Check Sequence“ (FCS) eines Übertragungsblocks.

Da das mathematische Konstruktionsverfahren nicht ganz einfach verständlich ist, beschränken wir uns hier auf eine pragmatische Darstellung der Schritte zur Generierung der CRC-Bitfolge.

Das Prinzip der Fehlererkennung/Fehlerkorrektur zunächst in der Übersicht:

> Bitströme werden als binäre Polynome geschrieben. Die CRC-Bits werden so konstruiert, dass das abgesicherte Polynom, welches aus Informationsbits und CRC-Bits entstanden ist, ohne Rest durch ein vorher festgelegtes „Generatorpolynom“ teilbar ist. Der Empfänger prüft den eintreffenden Bitstrom, indem er das zugehörige Polynom erneut durch das Generatorpolynom dividiert. Falls sich bei dieser Division ein Rest ≠0 ergibt, ist ein Fehler aufgetreten. Aus dem Restpolynom können Rückschlüsse auf die verfälschten Bits gezogen werden, so dass nicht nur Fehler erkannt, sondern diese auch automatisch korrigierbar sind.

Die Form des Generatorpolynoms bestimmt das Ausmaß der Fehlererkennung und -Korrektur. Generatorpolynome sind nach bestimmten Regeln aufgebaut und liegen tabelliert vor.

■ **Beispiel für die Erzeugung eines binären Polynoms:**

Bitstrom: `1 1 0 1 0 1`

zugehöriges Polynom *I*(x):

$$I(x) = 1 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$$

$$= x^5 + x^4 + x^2 + 1$$

■

Schritte zur CRC-Generierung

gegeben: Generatorpolynom $G(x) = x^r + ... + 1$ (Grad r)

1. Bilde binäres Polynom $I(x)$ aus gegebenen Infobits

2. $I(x)\, x^r$ dadurch werden Infobits in den Wertigkeiten um r Bits „nach oben“ geschoben, Platz geschaffen für r CRC-Bits

Infobits	000...0

3. Polynom-Division

$$\frac{I(x)\, x^r}{G(x)} = Q(x) + \frac{R(x)}{G(x)} \quad \Rightarrow \text{ Rest } R(x)$$

4. Divisionsrest $R(x)$ entspricht den CRC-Bits.
   Rückwandlung von $R(x)$ in einen Bitstrom und an die Infobits anhängen

Der Empfänger prüft, ob das Polynom des eintreffenden Bitstroms ohne Rest durch $G(x)$ teilbar ist.
Falls bei dieser Division ein Rest auftritt: $\Rightarrow$ Fehler!
Aus der Form des Restpolynoms: $\Rightarrow$ Korrektur der Fehler

Aus der Gleichung bei Schritt 3 folgt für die Polynomdarstellung des gesendeten Bitstroms (incl. CRC):

$$I(x)\, x^r + R(x) = Q(x)\, G(x)$$

D. h. es werden nur solche Pakete verschickt, die (Polynom-)Vielfache von G(x) sind. Daher darf bei der erneuten Division beim Empfänger kein Rest auftreten!

Beim Rechnen im „binären System“ (modulo-2-System) müssen wir die Regeln beachten:

$0 \pm 0 = 0 \quad 0 \pm 1 = 1$

$1 \pm 0 = 1 \quad 1 \pm 1 = 0$

„−“ entspricht „+“

Wir wollen die CRC-Erzeugung durch ein Beispiel verdeutlichen:

■ **Beispiel**

vorgegebenes Generatorpolynom: $G(x) = x^3 + x + 1$

Infobits: 1 0 1 0 (also ein sehr kleiner Block!)

Schritt 1: $I(x) = x^3 + x$

Schritt 2: $I(x) \cdot x^3 = x^6 + x^4$

Schritt 3: $(x^6 + x^4) / (x^3 + x + 1) = x^3 + 1$

$$\underline{x^6 + x^4 + x^3}$$

$$x^3$$

$$\underline{x^3 + x + 1}$$

$$x + 1 \quad \text{(Rest)}$$

Schritt 4: $R(x) = x + 1$

Rückwandlung in Bits:

$\underline{x^2\ x^1\ x^0}$ (← mögliche Potenzen)

0 1 1 (= CRC)

damit wird das gesamte „Paket“:

1 0 1 0 0 1 1 ■

Wir können feststellen:

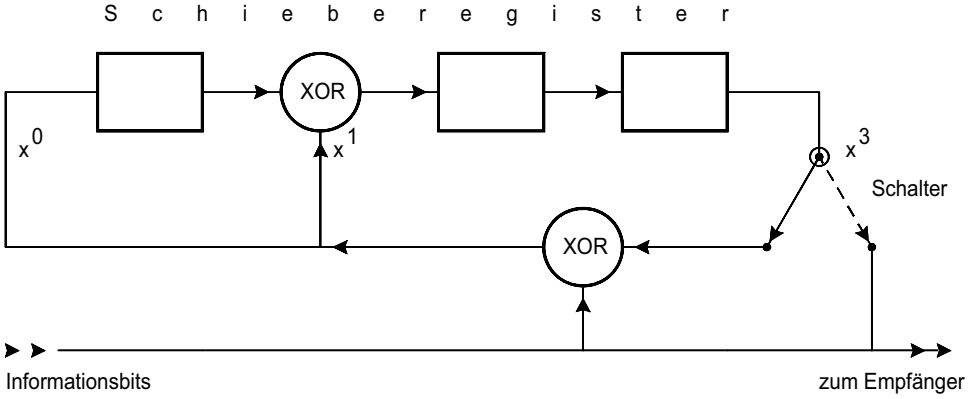
> Der Grad des Generatorpolynoms bestimmt die Anzahl der CRC-Bits.
> Je höher der Grad des Generatorpolynoms, desto besser ist ein Byte-Block abgesichert.

Im Gegensatz zu dieser etwas kompliziert wirkenden theoretischen Betrachtung ist die technische Generierung der CRC-Bits sehr einfach. Ihre Erzeugung kann hardwaremäßig durch rückgekoppelte Schieberegister realisiert werden. Die Rückkopplung erfolgt über XOR-Glieder an den vom Generatorpolynom bestimmten Stellen:

■ **Beispiel**

$G(x) = x^3 + x + 1$

zugehörige Schaltung:



■

Bei einem Generatorpolynom vom Grad r besteht das Schieberegister aus r Bitzellen. Für jede im Generatorpolynom auftretende Potenz gibt es einen XOR-gekoppelten Rückkopplungspfad. Die getaktet eintreffenden Informationsbits gelangen direkt zum Empfänger. Parallel dazu entstehen im Schieberegister die CRC-Bits, die am Ende des Byteblocks durch Umlegen des Schalters ausgetaktet, d. h. „nachgeschoben“ werden.

Überzeugen Sie sich am nachfolgenden Beispiel über die Wirkungsweise:

■ **Beispiel**

Generatorpolynom $G(x) = x^3 + x + 1$

Informationsbits: `1 0 1 0`

Inhalt Bitzellen

	eintreffende Infobit				zum Empfänger
Ruhe	---	0	0	0	---
1.Takt	1	1	1	0	1
2.Takt	0	0	1	1	0
3.Takt	1	0	0	1	1
4.Takt	0	1	1	0	0
*** Schalter umlegen, CRC austakten ***					
5.Takt	---	0	1	1	0
6.Takt	---	0	0	1	1
7.Takt	---	0	0	0	1
*** Schalter zurück ***					

Die CRC-Bits `011` stimmen mit den oben berechneten überein. ■

CRC-Generatoren für die Datenfernübertragung bestehen meistens aus 8, 16 oder 32 Bit (Plattensysteme nutzen auch 56 Bit und darüber).

Einige Generatorpolynome sind genormt:

„CRC-12“: $x^{12} + x^{11} + x^3 + x^2 + x + 1$

„CRC-CCITT“: $x^{16} + x^{12} + x^5 + 1$

„CRC-32“: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Letzteres Polynom dient zur Absicherung der Ethernetpakete in Lokalen Netzen. CRC-CCITT wird beim öffentlichen DATEX-P Netz benutzt, wodurch die Bitfehlerrate von 10^{-5} auf 10^{-12} abgesenkt wird.

11.7 Aufgaben

1) Was bedeutet der Begriff „code-transparent“ bei der seriellen Datenübertragung.

2) Wie hoch liegt die Leitungsausnutzung (Verhältnis Nutzdatenbits/Gesamtzahl übertragener Bits) bei der asynchronen Datenübertragung, wenn die Parameter 19200,8,E,1 eingestellt sind.

3) Ist es möglich, Binärdaten zwischen Rechnern zu übertragen, die mit Nullmodemkabel direkt verbunden sind?

4) 2 Rechner seien über ein Nullmodemkabel verbunden. Die seriellen Schnittstellen seien konfiguriert mit den Parametern 115200,8,E,1. Wie lange dauert die Übertragung einer CD-ROM, auf der sich eine Datenbank von 654 Mbyte Umfang befindet.

5) Welcher Unterschied besteht zwischen einem Modem- und einem Nullmodemkabel.

6) Wie hoch liegen die derzeit maximal erreichbaren Datenraten zwischen
   - einer Modem-Modem Verbindung,
   - einer ISDN-ISDN Verbindung,
   - einer Modem-ISDN (mit Modemchip) Verbindung.

7) Warum eignen sich die logischen Operatoren „AND“ und „OR“ nicht als BCC (Block Check Character) zur Absicherung eines Byteblocks?

8) Der Bitstrom 10101110 werde nach dem CRC-Verfahren mit dem Generator-Polynom $G(x) = x^3 + x + 1$ abgesichert. Welche Bitfolge wird übertragen?

9) Schreiben Sie ein C/C++-Programm, das zur Absicherung eines Bitstroms die Polynome CRC-3 ($x^3 + x + 1$), CRC-CITT und CRC-32 anbietet. Das Programm soll nach Auswahl eines dieser Polynome die Prüfbits einer eingegebenen Bitfolge berechnen. Die abzusichernde Bitfolge soll bis zu 64 Bit lang sein können.

   Anleitung: Benutzen Sie zur Ermittlung der Prüfbits eine Simulation der Schaltung mit rückgekoppelten Schieberegistern. Zeichnen Sie die Schaltung und geben. Sie die Bits des Schieberegisters nach jedem Takt aus.

12 Entwicklung und Organisation des Internet

Während noch vor wenigen Jahren das Internet fast ausschließlich als eine akademische Einrichtung galt, ist heute nach seiner Kommerzialisierung das „Netz der Netze“ jedermann zugänglich als allgemeine Informationsplattform. Die stürmische Entwicklung der globalen Vernetzung wirkt sich auf alle gesellschaftlichen Bereiche aus und verändert sie. Informationen jeglicher Art sind innerhalb weniger Sekunden weltweit verteil- und abrufbar. Begriffe wie „Globalisierung“ und „Informationsgesellschaft“ verdeutlichen die gegenwärtige Entwicklung. Jeder Kleinbetrieb präsentiert sich heute bereits im Internet und große Erwartungen werden an eine weltweite Rechnervernetzung geknüpft.

Bevor wir auf die Technik des Internet in den nächsten Kapiteln eingehen wollen, soll hier die Entwicklung und Organisation kurz vorgestellt werden.

Die Wurzeln des Internet gehen zurück auf das US Verteidigungsministerium (DoD, *Department of Defense*), das Ende der 60er Jahre für Experimente zur Vernetzung von Rechnern mit der Entwicklung der TCP/IP-Protokolle begann.

Entwicklung des Internet

USA:	
1972:	Aufbau des ARPANET (*Advanced Research Project Agency*) Zugang für Wissenschaftler zu Großrechnern; Entwicklung von Hardware-unabhängiger Kommunikationssoftware; Entstehung der TCP/IP-Protokolle („DoD-Protokolle“);
1978:	TCP/IP wird staatlich als Kommunikationssoftware vorgeschrieben;
Anfang der 80er:	zivile Nutzung des ARPANET für Forschung, Lehre und Entwicklung; TCP/IP wird Bestandteil von Unix; Entstehung des Begriffs „Internet“;
1985:	Aufbau des NFSNET (*National Science Foundation*), das als Backbone die Rechenzentren aller US-Unis verbindet;
Anfang der 90er:	das Internet wird zunehmend kommerziell genutzt;
Europa:	
1986:	RARE (*Réseaux Associés pour la Recherche Européenne*) Organisation zur Koordinierung von Aktivitäten der Vernetzung von Forschungs- und Wissenschaftsprojekten in Europa mit OSI-Protokollen; Paneuropäisches Netzwerk auf X.25-Basis (DATEX-P);
1993:	EuropaNet; Multiprotokoll-Backbone; zunehmende Verdrängung von OSI-Protokollen durch TCP/IP;
Deutschland:	
1990:	Aufbau des staatlich geförderten Wissenschaftsnetzes (WIN) durch den DFN-Verein (Verein zur Förderung eines Deutschen Forschungsnetzes) für wissenschaftliche Einrichtungen zunächst auf X.25-Basis, später zunehmend mit TCP/IP; das WIN wird Teil des Internet;
1996:	Umbau zum B-WIN (Breitband-WIN) mit Übertragungsraten bis 155MBit/
seit 1999:	Aufbau eines Gigabit-Weitverkehrs-Kernnetzes in Glasfasertechnologie mit Anschlusskapazitäten bis zu 10 Gbit/s

Das Internet besteht aus einer internationalen Solidargemeinschaft seiner Nutzer. Es wird nicht durch eine zentrale Autorität reglementiert, sondern lediglich durch verschiedene hierarchisch angeordnete Organisationen reguliert. Das oberste Gremium bildet die „Internet Gemeinschaft“ ISOC (*Internet Society*), die internationale Organisation zur Förderung des Internet.

Wichtige Organisationen des Internet:		
ISOC	*Internet Society*	Oberstes Internet Gremium
|		
IAB	*Internet Architecture Board*	Oberstes Steuerorgan
|		
IETF	*Internet Engineering Task Force*	Technik des Internet
|		
ICANN (früher IANA)	*Internet Corporation for Assigned Names and Numbers*	Koordination von Adressen, Port-Nummern, Top-Level Domains
|		
Regionale Zonen, Europa:		
RIPE NCC	*Réseaux IP Européens Network Coordination Center*	Koordination für den Pan-Europäischen Raum
CENTR	*Council of European National Top-Level Domain Registries*	Vertretung der Europäischen Interessen gegenüber ICANN
|		
Nationale Autorität für Deutschland:		
DE-NIC	Deutsches *Network Information Center*, Frankfurt	Zuständig für die deutschen *ISPs (Internet Service Provider)*

Die Hauptaufgaben des nationalen DENIC-Zentrums bestehen in dem Betrieb der Nameserver für die Top-Level Domain (TLD) „ .de “, der Verwaltung und Registrierung der ersten Domains unterhalb der TLD, wie z. B. „ .fh-wiesbaden.de “ und der Kooperation mit den übergeordneten Organisationen CENTR, ICANN und IEFT. Die DENIC hat ca. 230 Mitglieder, die als Internet-Diensteanbieter (ISP*, Internet Service Provider*) Kontingente von Internet-Adressen bezogen haben und diese an die „Endkunden“ weitervergeben. Unter den Mitgliedern befinden sich bekannte ISPs wie z. B. die Deutsche Telekom AG, die Arcor AG oder AOL Deutschland.

Sowohl die DENIC als auch das RIPE NCC bieten Zugang zu einer verteilten „whois“-Datenbank, von der man interessante Informationen zu IP-Adressbereichen und Domains abrufen kann.

Informationen zur Organisation und Aufbau des Internet:	
Allgemein zur Internet-Organisation:	http://www.denic.de/
Wem gehört eine bestimmte Domain?	http://www.denic.de/whois
Ist die Domain „xyz“ noch frei?	
Wem gehört eine bestimmte IP-Adresse?	http://www.ripe.net/pearl/whois
Wie groß ist der zugeteilte Adressraum?	

Anwendungen und Verfahren im Internet sind in sog. RFC (*Request for Comment*)-Dokumenten definiert. Diese Dokumente haben den Status eines Standards. Sie sind öffentlich zugänglich und bei den meisten Informationsservern des Internet abrufbar. Jeder kann nach einer Vorschrift, die in einem RFC-Dokument festgelegt ist, neue Internet-Anwendungen einbringen.

13 Das ISO/OSI-Schichtenmodell der Datenkommunikation

13.1 Probleme der Rechner-Rechner-Kommunikation

Schon Ende der 60er Jahre hatte man die Notwendigkeit erkannt, Daten über Netze auszutauschen. Die Rechnerhersteller haben daher für ihre Systeme eigene Netztechniken entwickelt. Die Netzwerke der großen Computerhersteller arbeiten sehr effizient, da sie optimal an die jeweiligen Rechnerarchitekturen und speziellen Betriebssysteme angepasst sind. Es entstanden „proprietäre“ (herstellerabhängige) Netzwerke, z. B.:

IBM : SNA (*System Network Architecture*)
DEC : DNA (*Digital Network Architecture*)
SIEMENS : TRANSDATA
HP : DSN (*Distributed Systems Network*)
– und viele andere –

Probleme treten jedoch auf, wenn Fremdprodukte in diese proprietären Systeme integriert werden sollen oder wenn eine Kommunikation zwischen unterschiedlichen Netzwerken aufgebaut werden soll. Die Systeme sind nicht miteinander kompatibel: die Stecker passen nicht, Signalpegel und Signalfunktionen sind unterschiedlich, die Übertragungsverfahren stimmen nicht überein, die Adressierung ist anders oder die Daten sind unterschiedlich codiert. Ein Informationsaustausch zwischen derartigen Systemen ist nur möglich durch teure und uneffizient arbeitende Anpassgeräte.

13.2 Das 7-Schichtenmodell

Die Vorteile eines „offenen Systems“ sind dem Anwender im PC-Bereich deutlich geworden. Hier sind praktisch alle Hard- und Software-Produkte kompatibel einsetzbar, die „Offenheit“ hat zu einer enormen Belebung des Marktes und damit auch zu einer für uns Anwender erfreulichen Preisentwicklung geführt. Entsprechende Forderungen nach „offenen Systemen“ im Netzwerkbereich führten zu einem ISO-Standard bei der Rechner-Rechner-Kommunikation: Das ISO/OSI – Referenzmodell der Datenkommunikation (ISO: *International Standardization Organisation*, OSI: *Open System Interconnection*).

Das ISO/OSI-Referenzmodell
Das Modell unterteilt das Problem der digitalen Kommunikation in sieben Teilaufgaben, die als „Schichten“ (*Layers*) hierarchisch angeordnet sind. Jede Schicht erledigt eine fest definierte Aufgabe.

Mit diesem Modell sollen die Voraussetzungen für eine Kommunikation in einer heterogenen Rechnerlandschaft geschaffen werden, ohne auf einen Hersteller festgelegt zu sein.

Ziel dieses Standards ist:
Jeder Nutzer (Rechner, Programm) kann mit jedem anderen in Verbindung treten und Daten austauschen. Durch die Anerkennung und Befolgung festgelegter Regeln (Normen) ist dies möglich, ohne die technische Realisierung (Implementierung) des Kommunikationspartners zu kennen. Der Informationsaustausch ist damit herstellerunabhängig.

Die Aufgaben der 7 Kommunikationsschichten

Schicht 7: *Application Layer* — Anwendungsschicht
Anwenderprogramm

Schicht 6: *Presentation Layer* — Darstellungsschicht
Festlegung der Datenstruktur und Datenformate

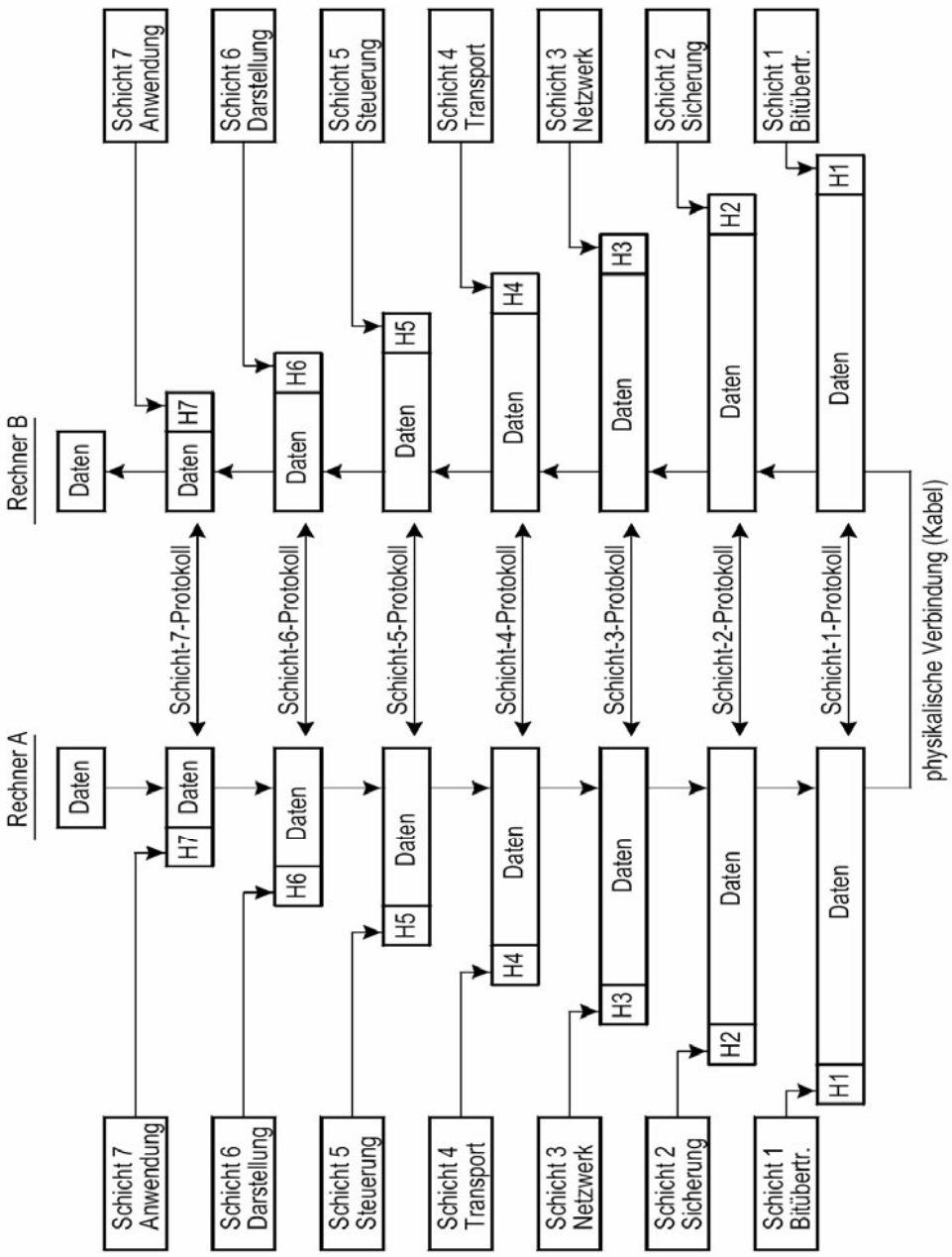
Schicht 5: *Session Layer* — Kommunikationssteuerung
Steuerung des Teilnehmerdialogs
Zuordnung logische Namen ⇔ phys. Adressen

Schicht 4: *Transport Layer* — Transportschicht
Bereitstellung einer gesicherten Prozess-zu-Prozess Verbindung auf den Endknoten
Auf- und Abbau einer Verbindung

Schicht 3: *Network Layer* — Netzwerk/Vermittlung
Adressierung, Wegewahl (Routing) durch das Netz

Schicht 2: *Data Link Layer* — Sicherungsschicht
Steuerung des Zugriffs auf das Netzmedium
Fehlererkennung und Fehlerkorrektur

Schicht 1: *Physical Layer* — Bitübertragung
transparenter, ungesicherter Bitstrom
Kabeltyp, Stecker, Signalpegel



Kommunikation im 7-Schichten-Modell: H: Header, Rechner A: Sender, Rechner B: Empfänger

Die OSI-Türme werden jeweils vertikal durchlaufen, beginnend bei der Anwendung in Schicht 7 des Rechners A bis hinunter zur physikalischen Verbindung in Schicht 1 und wieder hinauf zur Schicht 7 des Rechners B. Eine Schicht erbringt einen „Dienst“ für die darüberliegende Schicht und nutzt dabei den „Dienst“ der darunterliegenden Schicht. Jede Schicht versieht die ihr jeweils angelieferten Daten mit einer schichtspezifischen Zusatzinformation (*Header*), die für die entsprechende Schicht des Zielrechners vorgesehen ist. Durch das wiederholte „Einpacken“ der Daten entsteht von Schicht zu Schicht eine immer umfangreichere „Verpackung“ (*Overhead*) der Ursprungsdaten. Beim Kommunikationspartner durchlaufen die Datenpakete die Schichten in umgekehrter Reihenfolge, sie werden von Schicht zu Schicht wieder „ausgepackt“, bis wieder die reinen Ursprungsdaten beim Verbindungspartner vorliegen.

Jede Schicht „kommuniziert“ nur mit der entsprechenden Partnerschicht, indem sie die für sie spezifische Information auswertet. Damit sich bei dieser „horizontalen Kommunikation“ keine Missverständnisse ergeben, müssen Regeln befolgt werden, die das „Protokoll“ einer Schicht ausmachen. Das Protokoll bestimmt vollständig die Funktionalität einer Schicht.

Schichten-Protokoll

Satz von Regeln für eine horizontale Kommunikation zweier äquivalenter Schichten

Die Reihenfolge der Schichtenabarbeitung bei einer Verbindung erinnert an die Arbeitsweise des Stack (*last in, first out*). Die Schichtenstruktur wird daher häufig auch als „Protokollstack“ bezeichnet.

Es sollte uns bewusst sein, dass allein die Befolgung des OSI-Modells keine Garantie dafür bietet, dass eine Kommunikation möglich ist. Denn das Modell legt lediglich fest, *was* von einer Schicht an prinzipiellen Aufgaben zu behandeln ist, nicht *wie* die technische Implementierung vorzunehmen ist. Es bedarf daher noch eines zweiten Schritts: Die Standardisierung der Schichtenprotokolle selbst.

Für das Zustandekommen einer Kommunikation ist es nicht zwingend erforderlich, dass auch jede Schicht des OSI-Modells tatsächlich implementiert wurde. Bei Lokalen Netzen ist es sogar häufig der Fall, dass einzelne Schichten übersprungen werden.

Das ISO/OSI-Schichtenmodell ist ein theoretisches Modell, das sich nicht – wie sonst häufig bei Standards – von einem erprobten System aus der Praxis ableitet. Obwohl alle großen EDV-Hersteller sich eindeutig zu OSI bekannt haben, gibt es nur wenige reine OSI-Produkte, die nicht stark verbreitet sind:

Einige genormte OSI-„Dienste“ der Anwendungsschicht

X.400	Message Handling Service (Electronic Mail)
X.500	Directory Service (Zertifikate, „Adressbuch für Electronic-Mail“)
FTAM	File Transfer, Access and Management (Dateiübertragung)
VT	Virtual Terminal Service (Dialog)

Der Grund für die mangelnde Akzeptanz des OSI-Modells liegt an der Komplexität und Schwerfälligkeit dieser Produkte. „Alleskönner“ haben viel Overhead!

Herstellerübergreifend – und damit auch „offen“ – hat sich dagegen infolge der Ausdehnung des Internet der Protokollstack auf Basis der TCP/IP-Protokollfamilie etabliert. TCP/IP-Anwendungen sind jedoch nicht OSI-konform.

TCP/IP war schon immer fester Bestandteil des Betriebssystems Unix, das in den Anfängen des Internet praktisch ausschließlich als Netzwerkbetriebssystem genutzt wurde. Der erfolgreiche Einsatz dieser Protokollfamilie im sich explosionsartig ausbreitenden Internet hat OSI-Protokollen keine Chance gegeben. Heute unterstützen daher alle gängigen Netzwerkbetriebssysteme nicht den OSI-, sondern den TCP/IP-Protokollstack, evtl. neben anderen proprietären Protokollen.

Trotzdem hat es sich als zweckmäßig erwiesen, auch nicht OSI-konforme Protokolle in Schichten darzustellen und dem OSI-Modell gegenüberzustellen. Insofern hat das OSI-Modell für den Aufbau von Netzwerkanwendungen seine Bedeutung behalten. Auch die Forderungen des ISO-Gremiums nach „Offenheit“ und „Herstellerunabhängigkeit“ sind für TCP/IP-Anwendungen in der Praxis vorzüglich erfüllt: Sämtliche im Internet eingesetzten Kommunikationsprotokolle werden vom IEFT (*Internet Engineering Task Force,* → s. Kap. 12) in sog. RFC-Standards weltweit veröffentlicht. Diese Dokumente sind von jedermann abrufbar und jeder kann diesem Gremium neue Entwicklungen vorschlagen und zur Prüfung einreichen.

Die im Kap. 11 vorgestellten öffentlichen Weitverkehrsnetze decken im OSI-Modell die Schichten 1–3 ab. Die darauf aufsetzende Kommunikationssoftware übernimmt die Aufgabe der höheren Schichten.

Lokale Netze sind meistens proprietäre Systeme und lassen sich häufig nur in den unteren Schichten auf das OSI-Schichtenmodell abbilden.

13.3 Aufgabe

Ordnen Sie nachfolgende Aufgaben bei einer Netzkommunikation den zuständigen Schichten des OSI-Modells zu:

a) Regelung, ob bei der Darstellung eines Bytes das höchstwertige Bit in dem Byte mit der höchsten Adresse zugeordnet ist (*Big-endian*-Format, typisch bei Motorola-Prozessoren) oder das höchstwertige Bit in dem Byte der niedrigsten Adresse zugeordnet ist (*Little-endian*-Format, typisch bei Intel-Prozessoren).

b) Die Bit-Übertragungsgeschwindigkeit auf dem Netzmedium.

c) Auf einem E-Mail-Server eine eingetroffene E-Mail an den E-Mail-Server-Prozess weiterleiten.

d) In einem vermaschten Netzwerk ein eingetroffenes Datenpaket auf einem Netzwerkknoten mit mehreren Netzwerkinterfaces zum richtigen Netzwerkinterface weiterleiten in Richtung Zielknoten.

e) Steuerung des Netzmediumzugriffs, damit nicht zwei Rechner gleichzeitig senden und die Datenpakete sich nicht auf dem gemeinsam genutzten Netzwerkkabel überlagern und zerstören.

f) Sortieren eingetroffener Datenpakete in die richtige Reihenfolge.

g) Festlegung der Frequenz der Lichtimpulse bei Lichtleiterübertragungen.

14 Basiskomponenten von Lokalen Netzen

Befindet sich die Netzwerkinstallation vollständig im „privaten“ Bereich einer Institution, so sprechen wir von einem „Lokalen Netzwerk“, einem „LAN“ (*Local Area Network).* Wird hingegen die Kommunikation über öffentliche Datennetze geführt, so ist das Netz nicht mehr lokal, sondern ein Weitverkehrsnetz, ein „WAN“ (*Wide Area Network).* LANs haben häufig einen oder mehrere WAN-Zugänge, um darüber einzelne Arbeitsstationen oder LANs zu koppeln oder um mit dem Internet zu kommunizieren. Regionale Stadtnetze nehmen als „MAN“ (*Metropolitan Area Network*) eine Zwischenstellung ein.

LANs basieren überwiegend auf dem Ethernet-Standard und werden mit den Netzbetriebssystemen Windows (XP, 2000, NT, ...), NetWare oder Unix (Linux) betrieben.

14.1 Funktionseinheiten und Grundaufbau

Die in einem Netzwerk eingebundenen Rechner und Netzkoppelelemente bezeichnet man als „Knoten“ eines Netzes.
Ein LAN ist aus folgenden Grundbausteinen aufgebaut:

Grundbausteine eines LAN

– Clients:	Arbeitsstationen des LAN (PC, Workstation), z. B. DOS-Rechner, Windows Workstation, Unix-Rechner
– Server:	leistungsstarker Rechner, der den Clients verschiedene „Dienste“ (*Services)* anbietet, multiuserfähig z. B. XP-Server, NetWare Server, Unix-Server, Windows 2000 Server
– Verkabelungssystem:	(gemeinsam genutztes) Übertragungsmedium (Netzkabel)
– Netzkoppelelemente	Netzverteiler, LAN-LAN-Koppler, Internet-Zugänge, z. B. Hub, Bridge, Switch, Router

In ein LAN eingebundene Time-Sharing Systeme, z. B. Unix-Rechner, an denen man sich über das Netz anmelden und Sitzungen durchführen kann, werden auch als *Hosts* („Gastgeber“) bezeichnet.

Die Rechner werden durch den Einbau eines Netzwerk-Interfaces mit Netzanschlussstecker, z. B. einen Ethernetadapter oder Token Ring Adapter, netzwerkfähig.

Die Arbeitsstationen (Clients) arbeiten grundsätzlich dezentral unter der Kontrolle ihrer lokalen Betriebssysteme. Darüber hinaus können sie von einem Server spezielle Netzdienste anfordern.

Netze können in unterschiedlichen Organisationsformen auftreten:

– Client / Server-Netzwerk: Clients und Server sind unterschiedliche Rechner, die fest vorgegeben sind.
typische Server: Fileserver, Kommunikationsserver, Printserver,

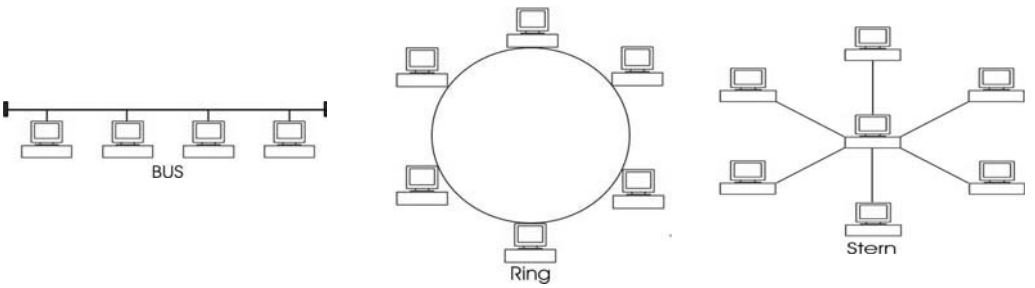
Beispiel: NetWare-Netze

- Peer-to-Peer -Netzwerk: Das Netz besteht aus gleichwertigen Arbeitsstationen. Jede Arbeitsstation enthält auch Serverfunktionen und ist Client und Server zugleich.

  Beispiel: Unix-Netze, Arbeitsgruppen unter Windows

Wenn ein Server eines Client / Server-Netzwerks neben seiner Serverfunktion auch noch als Arbeitsstation genutzt werden kann, arbeitet er in der Betriebsart *non-dedicated.* Dies ist wegen der damit verbundenen Leistungseinbußen nur für selten genutzte Serverdienste sinnvoll. In der Regel besitzen diese Netze *dedicated* Server, die ausschließlich Serverdienste erbringen.

Die Topologie (räumliche Anordnung) eines Netzes ist eine Folge des benutzen Netzmediums.

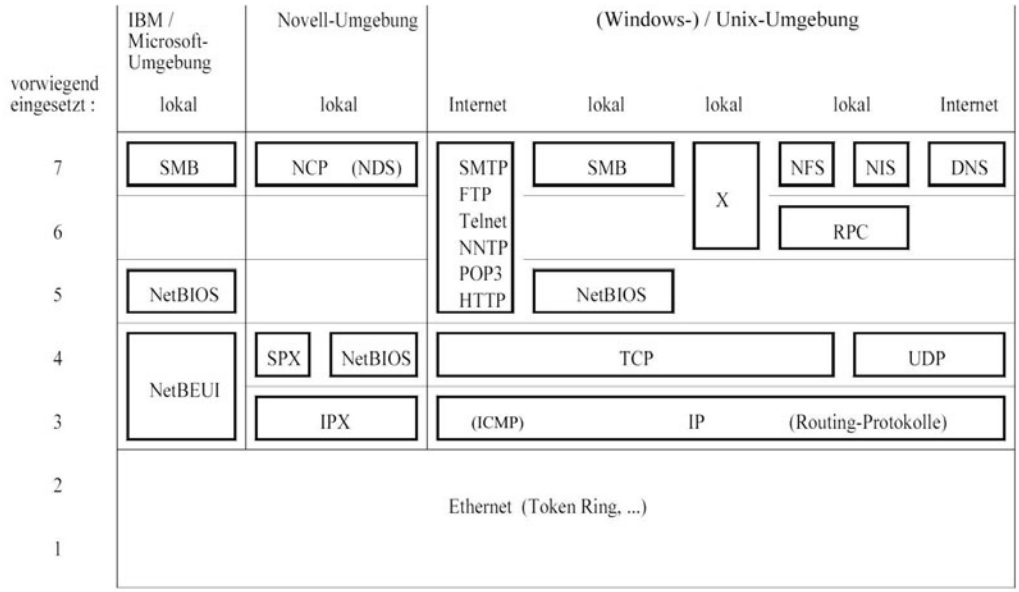


Basistopologien: Bus, Ring, Stern

In der Praxis sind diese Topologien in LANs häufig gemischt. Auch ein von der Kabelführung physikalisch als Stern erscheinender Aufbau kann logisch, d. h. beim Mechanismus des Mediumzugriffs, als Bus-System wirken. Dies ist z. B. der Fall in Hub-gekoppelten Ethernet-Netzen.

14.2 Bezug zum OSI-Modell

Lokale Netze folgen als proprietäre Systeme meistens nur in den unteren vier Protokollschichten dem OSI-Modell. Eine Übersicht der in LANs am Häufigsten eingesetzten Protokolle ist in dem nachfolgenden Bild dargestellt:



NetBEUI:	NetBIOS Extended User Interface
NetBIOS:	Network Basic Input/Output System
SMB:	Server Message Block
IPX:	Internet Packet Exchange
SPX:	Sequential Packet Exchange
NCP:	NetWare Core Protocol
NDS:	NetWare Directory Services
IP:	Internet Protocol
TCP:	Transmission Control Protocol
UDP:	User Datagram Protocol
HTTP:	HyperText Transfer Protocol
POP3:	Post Office Protocol Vers. 3
NNTP:	Network News Transfer Protocol
Telnet:	Login auf Remote Host
FTP:	File Transfer Protocol
SMTP:	Simple Mail Transfer Protocol
X-Windows:	graphische Benutzeroberfläche
RPC:	Remote Procedure Call
NIS:	Network Information System
DNS:	Domain Name Service
ICMP:	Internet Control Message Protocol

Grob lässt sich eine Einteilung in drei Bereiche vornehmen: Anwendungen, Transportprotokolle und Mediumzugriff.

OSI-Schicht:	*Aufgabe im LAN:*	*realisiert durch:*
7. Anwendung 6. Darstellung 5. Steuerung	**Netzanwendungen**	WWW, E-Mail, FTP,..... Software der Netzwerkbetriebssysteme z. B.: Windows XP, Windows 2000, Unix, NetWare
4.Transport 3. Netzwerk	**Transport**	
2. Sicherung 1. Bitübertragung	**Mediumzugriff** (Topologie)	Hardware, Firmware und HW-Treiber-Software: Netzwerkadapter und Verkabelungssystem

Die Protokolle der Schicht 3 und 4 werden bei Lokalen Netzen wegen ihrer zentralen Bedeutung häufig auch einfach mit „das benutzte Protokoll ist ...“ angesprochen.

Die Transportprotokolle und ein großer Teil der Anwendungen sind in den Netzwerkbetriebssystemen enthalten. Bei Betriebssystemen ohne eigene Netzfunktionalität, z. B. DOS, muss der Protokollstack als zusätzliche Software geladen werden.

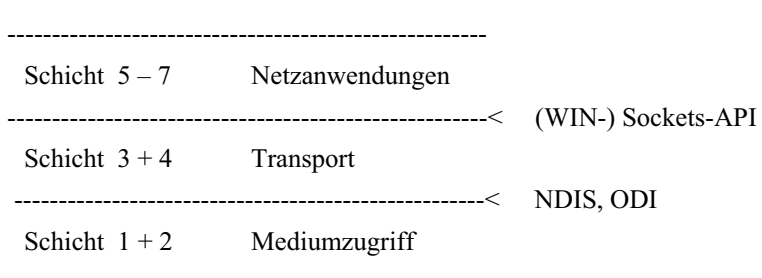
Die Normierung der Schnittstellen erlaubt, dass eine bestimmte Netzwerkanwendung wie in einem Baukastensystem mit unterschiedlichen Transportprotokollen und/oder Netzmedien zusammenarbeiten kann. Die Schnittstelle zwischen den unteren beiden Schichten und den darüber liegenden höheren Schichten ist in LANs besonders deutlich ausgeprägt. So können z. B. in heterogenen Netzen verschiedene Netzwerk-Betriebssysteme mit unterschiedlichen Transportprotokollen eine gemeinsame Ethernet-Basis nutzen.

Beispiel:

NetWare	Windows (XP, 2000, NT,...)	Unix (Linux)
Ethernet (oder Token Ring,..)		

Trotz des gleichen „Unterbaus“ (z. B. Ethernetpakete) können sich aber nur diejenigen Pakete gegenseitig „verstehen“, die in sich die gleichen höheren Protokolle verpackt haben. Andrerseits folgt aus dem Schichtenmodell, dass ein und dasselbe Betriebssystem mit unterschiedlichen Netzwerkadaptern, d. h. unterschiedlichen Mediumzugriffsverfahren, Verkabelungssystemen und Topologien zusammenarbeiten kann.

Für Netzwerkprogrammierer bilden die Nahtstellen an den Schichtübergängen 2-3 und 4-5 „Einstiegspunkte“ (Interface-Spezifikationen und Netzwerk-API, *Application Programming Interface*) zur Entwicklung eigener Netzanwendungen.



Die NDIS- (*Network Device Interface Specification*, Windows-Netze) und ODI- (*Open Data Link Interface,* NetWare-Netze) Spezifikationen beschreiben eine einheitliche Software-Schnittstelle, über die die Netzwerktreiber unterschiedlicher Netzwerkadapter mit den höheren Protokollen in geräteunabhängiger Form kommunizieren. Besonders interessant ist die auf höherer Ebene liegende Socket-Schnittstelle, da sie als standardisierte C/C++-Bibliotheken für Unix und Windows vorliegt. Die Socket-API gestattet die schnelle Entwicklung eigener Client/Server-Anwendungen, bei denen das Transportsystem – und damit die Dienste aller tieferen Schichten – durch Funktionsaufrufe eingebunden werden (→ s. Kap. 11).

In der Praxis klassifiziert man Netze häufig nach Zugriffsverfahren und Betriebssystemen und bezeichnet die LANs als

„Ethernet-Netz“, „Token-Ring-Netz“, ...

bzw.

„Windows-Netz“, „NetWare-Netz“, „Unix-Netz“, ...

In den nächsten Abschnitten konzentrieren wir uns im Wesentlichen auf Ethernet und das TCP/IP-Protokoll.

15 Ethernet-Netze

Ethernet hat sich in den letzten 25 Jahren zur erfolgreichsten LAN-Technologie entwickelt. Die stetige Weiterentwicklung zu schnelleren Übertragungsraten hat andere Technologien wie Token Ring, FDDI oder ATM aus dem LAN Bereich fast vollständig verdrängt.

Ethernet deckt die untersten beiden hardwarenahen Schichten des OSI-Modells ab.

Kenndaten von Ethernet-Netzen:

Bitcodierung: Manchester-Verfahren, B4/B5, ...

Zugriffsverfahren bei *shared media:* CSMA/CD

Rahmenaufbau (*Frame*): Ethernet II, 802.2 oder 802.3

Übertragungsgeschwindigkeit:

Standard: 10 Mbit/s
Fast-Ethernet: 100 Mbit/s
Gigabit-Ethernet: 1000 Mbit/s (10 Gbit/s)

Bezeichnung: Medium xx BASE y

- y: Segmentlänge/Segmenttyp
- BASE: Basisband
- xx: Übertragungsgeschwindigkeit in MBit/s

15.1 Bitcodierung

Die Festlegung des Bit-Codierungsverfahrens ist eine Aufgabe der Schicht 1. Übliche Taktfrequenzen zur Übertragung der Bits auf dem Netzwerkkabel liegen zwischen 10 Mbit/s und 1Gbit/s (10 Gbit/s in Vorbereitung). Damit die am Empfänger eintreffenden Bits fehlerfrei vom Netzmedium „abgetastet“ werden können, ist eine Synchronisation der Taktfrequenz zwischen Sender und Empfänger erforderlich. Statt nun den Takt durch eine zusätzliche Leitung im Netzwerkkabel mitzuführen, werden Bitcodierungsverfahren benutzt, aus denen der Takt zurückgewonnen werden kann. Voraussetzung für solche selbsttaktierenden Verfahren ist, dass bei beliebigen Bitströmen genügend häufig high/low-Pegelwechseln in dem übertragenen Bitstroms auftreten.

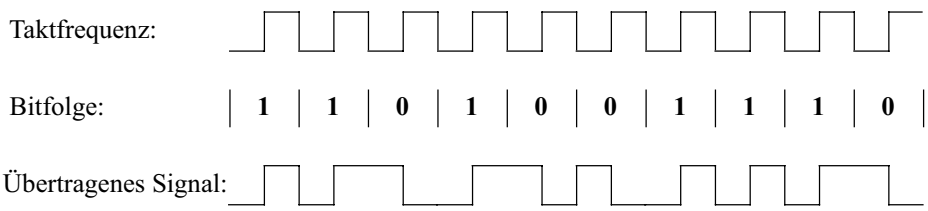
Das einfachste Codierungsverfahren für die beiden Bitzustände 0 und 1 würde sich ergeben durch eine Pegelzuordnung: 0: low-Pegel, 1: high-Pegel.

Dieses Verfahren eignet sich jedoch nicht, weil für längere 0-Bitfolgen oder 1-Bitfolgen keine Pegelwechsel auftreten und deshalb eine Taktrückgewinnung aus dem übertragenen Signal nicht sicher gewährleistet wäre. LANs nutzen daher je nach eingesetzter Netzwerktechnik spezielle Codierungsverfahren, die in jedem Fall eine Taktrückgewinnung garantieren.

Selbsttaktierende Bit-Codierungsverfahren	
Verfahren:	*angewendet bei:*
Manchester-Codierung	10Mbit-Ethernet
4B/5B-Codierung	100Mbit-Ethernet

Manchester Codierung:

Bitwert:	Signalpegel:
0	Pegelübergang von high nach low
1	Pegelübergang von low nach high



Ein Pegelwechsel tritt spätestens nach einer Periodendauer der Taktfrequenz auf. Die Bitabtastung erfolgt in der 2. Halbperiode, da hier stets der Pegel dem jeweiligen Bitwert entspricht.

Andere Bitcodierungsverfahren lösen das Problem mangelnder Pegelwechsel durch Umcodierungen kleiner Bit-Gruppen derart, dass durch Einfügen eines zusätzlichen Bits ein Pegelwechsel erzwungen wird. Ein Beispiel ist das 4B/5B-Verfahren des Fast-Ethernet (100Mbit):

4B/5B-Codierung:

Hier werden jeweils 4-Bit in eine 5-Bit-Kombinationen umcodiert. Unter den 32 möglichen 5-Bit-Worten werden die erforderlichen 16 Bitmuster so ausgewählt, dass ein Pegelwechsel garantiert ist. Der Empfänger dekodiert wiederum auf die ursprünglichen 4-Bit-Kombinationen und entfernt damit das zusätzliche Bit. Wegen des 4/5-Verhältnisses sinkt die Effektivität gegenüber dem Manchesterverfahren auf 80%.

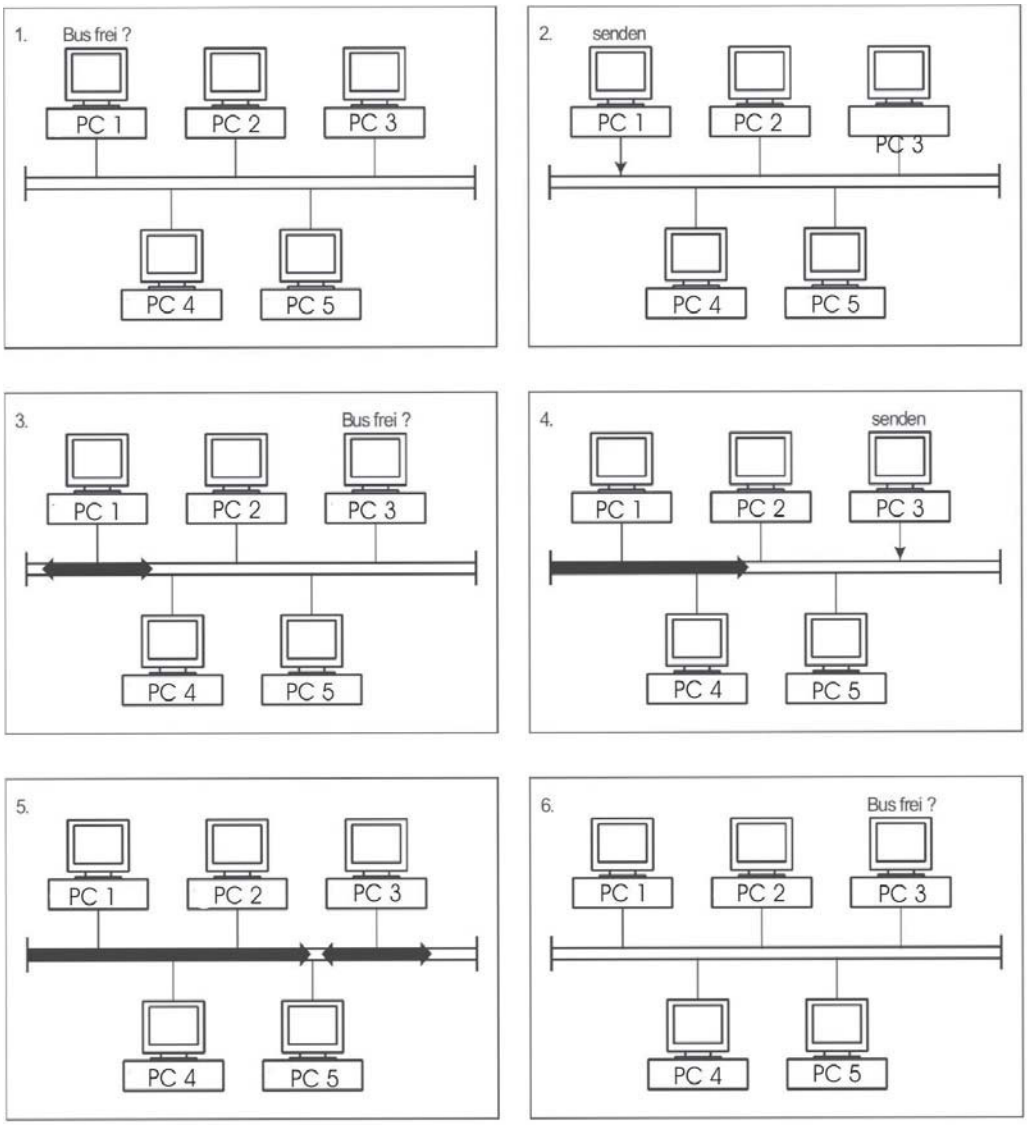
15.2 Das Netz-Zugriffsverfahren CSMA/CD

Die Berechtigung auf das Netzmedium zu zugreifen, ist eine Aufgabe der Schicht 2. Die Zugriffsverfahren für Lokale Netze sind in einer Reihe von IEEE 802-Standards (*Institute of Electrical and Electronic Engineers*) festgelegt.

Benutzen Rechner das Netzwerkkabel als ein gemeinsames Netzmedium (*shared medium)*, kommt im Ethernet-Umfeld das CSMA/CD-Netz-Zugriffsverfahren zum Einsatz. Das Medium kann jeweils nur von *einem* Rechner belegt werden, d. h. es darf jeweils nur eine Station senden.

Das CSMA/CD (*Carrier Sense Multiple Access/Collision Detect*) -Verfahren ist das am Häufigsten eingesetzte Zugangsverfahren in LANs. Es eignet sich besonders für heterogene Netze und UNIX-Umgebungen. Häufig werden diese Netze als „Ethernet“-Netze bezeichnet.

CSMA/CD ist ein Bus-Zugangsverfahren. Alle Nutzer (Knoten) greifen auf das gemeinsame Medium zu. Dabei kann es beim gleichzeitigen Senden mehrerer Stationen zu „Kollisionen“ auf dem Medium kommen. Wie dann vorzugehen ist, geht aus der folgenden Abbildung hervor:



CSMA/CD-Verfahren

1. PC1 will senden und hört den Bus ab (*Carrier sense*):
2. Da der Bus frei ist, beginnt PC1 zu senden, hört dabei weiterhin den Bus ab.
3. PC3 möchte ebenfalls senden. PC3 hört den Bus ab, findet ihn frei, da sich die Sendung von PC1 noch nicht bis PC3 ausgebreitet hat.
4. PC3 beginnt ebenfalls zu senden (*Multiple Access*), hört dabei weiterhin den Bus ab.

5. Die Sendungen kollidieren. Die Signale überlagern sich und sind dadurch verfälscht. Beide Sender erfahren von der Kollision (*Collision Detect*).
6. Beide Stationen stoppen ihre Sendungen und senden ein spezielles Kollisionssignal (*Jam*) aus. Somit registrieren alle Datenendgeräte die Kollision. Die sendewilligen Stationen unternehmen nach einer unterschiedlichen Wartezeit einen neuen Sendeversuch.

Damit nach einer Kollision nicht beide Stationen wieder gleichzeitig einen neuen Sendeversuch starten, ermittelt jede Station ihre Wartezeit nach einem Zufallsverfahren statistisch. Nach 16 erfolglosen Versuchen wird die Sendung mit einer Fehlermeldung an die Netzwerkschicht endgültig abgebrochen.

Damit CSMA/CD auch in ausgedehnten Netzen funktioniert, muss sichergestellt sein, dass alle Stationen noch während der Sendezeit von PC1 erreicht werden; denn anderenfalls hätte der Sender PC1 gar nicht den Verlust seiner Sendung erfahren! Es gibt daher Zusammenhänge zwischen Mediumausbreitungsgeschwindigkeit, Bitübertragungsgeschwindigkeit, minimale Nachrichtenlänge und maximaler Ausdehnung des Netzes. Wir kommen auf diese Zusammenhänge im Kap. 15.4 zurück.

Da man beim CSMA/CD-Verfahren nicht sicher voraussagen kann, wann eine Station erfolgreich senden darf, sind mit CSMA/CD arbeitende Ethernet-Netzwerke „nicht-deterministisch". Dies widerspricht strengen Echtzeitanforderungen (garantierte Reaktion innerhalb einer definierten Zeit). Die früher häufig gefundene Aussage „Ethernet ist nicht echtzeitfähig" stimmt heute aber generell nicht mehr, da man durch den Einsatz von modernen Switching-Technologien das CSMA/CD-Verfahren in Ethernet-Netzen vermeiden kann (→ s. Kap. 15.6.3).

15.3 MAC-Adressen

Jedes Netzwerk-Interface eines Ethernet-Netzes besitzt eine von den Interface-Herstellern vorgegebene, weltweit eindeutige physikaliche Adresse, die als Ethernet-Adresse oder MAC-Adresse (*Medium Access Control*) bezeichnet wird. Die über das Netzsegment (Teilnetz) gesendeten Datenpakete, auch „Rahmen" (*Frames)* genannt, enthalten in ihren Headern diese Hardware-Adressen zur Identifikation von Absender und Empfänger. Ethernet-Adressen sind 48 Bit lang und werden in der Regel in hexadezimaler Form angegeben (12 Hex-Stellen):

Beispiel einer Ethernet-Adresse: 00-01-02-AB-65-9C

MAC-Adressen sind fest an die Hardware des jeweiligen Netzwerk-Interface gebunden, sie können nicht verändert werden. Besitzt ein Rechner z. B. zwei Ethernet-Interfaces, so gibt es auch zwei MAC-Adressen, d. h. aus der Sicht des Netzmediums zwei verschiedene Netzknoten.

Die MAC-Adressen sind dem Benutzer einer Station meistens gar nicht bekannt, da Netzanwendungen die Adressierungssysteme der höheren Protokolle benutzen. Jede „höhere" Adressform einer Anwendung muss letztlich von den Schichtenprotokollen in diese Hardware-Adresse umgesetzt werden. Diese Umsetzung geschieht durch die Protokollsoftware, die entsprechende Adresstabellen auf den Stationen des Netzwerks anlegt.

MAC-Adressen:
Die Stationen im LAN sind immer nur über die physikalischen MAC-Adressen (Ethernet-Adressen) erreichbar. Sie werden in den Headern der übertragenen Frames als Quell- und Zieladresse eingetragen.

Sollte (z. B. wegen eines Defektes) die Ethernet-Adresse nicht verfügbar sein, ist keinerlei Kommunikation mit diesem Netzinterface möglich!

Die ersten 3 Byte einer Ethernet-Adresse enthalten einen Code, der den Hersteller des Netzwerkinterface identifiziert. Man kann also aus den MAC-Adressen den Adapter-Hersteller ermitteln, z. B.:

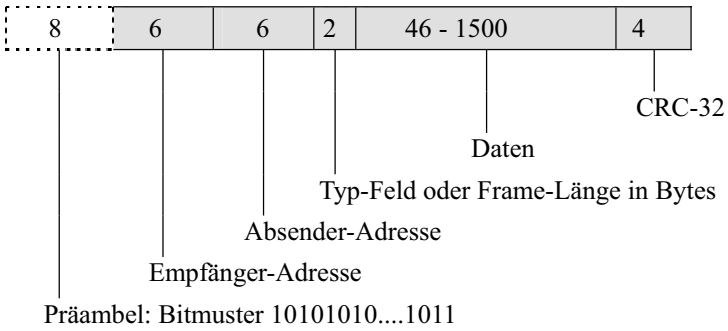
Byte 1-3 der Ethernet.-A.	*Hersteller*
00-01-02	Bolt, Beronek & Newman
00-20-AF	3COM
00-60-08	3COM
00-00-1B	Novell
08-00-09	Hewlett Packard
08-00-20	Sun

Eine vollständige Liste findet man im Internet unter
http://www.standards.ieee.org/regauth/oui/oui.txt.

Datenpakete, die an alle Stationen des Netzsegmentes gerichtet sind, enthalten als Empfänger die *Broadcast*-Adresse FF-FF-FF-FF-FF-FF. Diese Datenpakete werden von allen Stationen empfangen und ausgewertet.

15.4 Ethernet-Frames

Ein Ethernet-Datenpaket (*Frame)* besitzt folgenden prinzipiellen Aufbau (Zahlen in Bytes):



Die Übertragung beginnt mit der Präambel, die zur Synchronisation dient. Es gibt verschiedene Frame-Typen für Ethernet. Sie unterscheiden sich hauptsächlich in der Interpretation des 2 Byte Feldes nach der Adressierung entweder als Frame-Länge (802.3-Frame, IEEE 802.2-Frame) oder als Typfeld (Ethernet II-Frame) zur Kennzeichnung des übergeordneten Protokolls, an das das Paket übergeben werden soll. Da sich die unterschiedlichen Frame-Typen nicht gegenseitig verstehen, muss manchmal bei der Konfiguration der Netzwerktreiber der Frame-Typ explizit vorgegeben werden. Heterogene, mit dem Internet verbundene Netze benutzen meist den aus der Unix-Welt stammenden Ethernet II-Frame.

Die Forderung einer Mindestlänge der Ethernet-Frames ergibt sich durch das CSMA/CD-Verfahren. Dies soll durch die folgende Beispielrechnung an einem 10-Mbit-Ethernetnetz deutlich werden:

Damit ein Sender eine Kollision mit einer möglichen Sendung von der entferntesten Station des Segmentes noch während seiner Sendezeit erkennt, darf es nicht vorkommen, dass ein Paket losgelöst von Sender und Empfänger isoliert auf dem ausgedehnten Netzmedium „schwimmt“. Die Ausbreitungsgeschwindigkeit auf dem Netzkabel liegt bei ca. 0.7*c, mit $c=3*10^8$ m/s (Lichtgeschwindigkeit). Bei der typischen Übertragungsgeschwindigkeit von 10 Mbit/s für Ethernet beträgt die „Bitlänge“ somit ca. 20 m auf dem Netzmedium. Wegen Leitungsdämpfungen darf ein Ethernetsegment höchstens 500 m lang sein, bevor das Signal durch einen Repeater (→ s. Kap. 15.6.1) wieder verstärkt wird. Jeder Repeater benötigt ca. 3 µs zur Übertragung des Signals (*Latenz)*. Es wurde festgelegt, dass maximal 5 Segmente über 4 Repeater gekoppelt werden dürfen. Damit ergibt sich die Abschätzung des kleinsten Pakets zu:

Übertragungsdauer des Minimalpakets		> 2 * Laufzeit zur entferntesten Station
	Bitanzahl * 10^{-7} s	> 2 * (5 * 500 m / 0.7 * 3 * 10^8 m/s + 4 * 3 µs)
		> 48 µs
===>	Bitanzahl	> 480

Die Mindestlänge für Ethernetpakete wurde auf 64 Bytes (= 512 Bit) festgelegt. Liegen weniger Daten vor, wird das Paket durch „Füllbits“ auf die vorgeschriebene Mindestlänge ergänzt.

Der MTU-Wert:

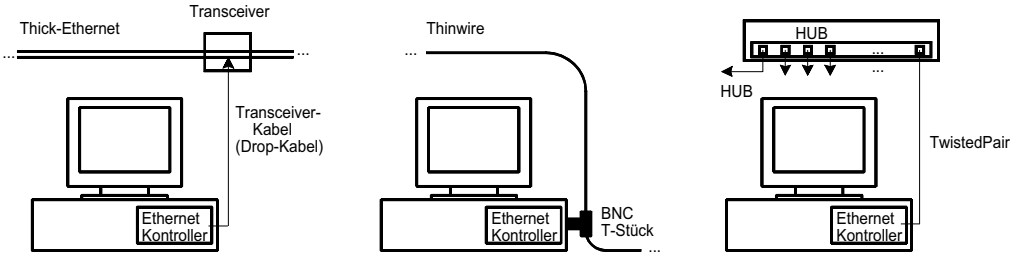
Die maximale Anzahl der Datenbytes, die ein Frame aufnehmen kann, heißt *MTU (Maximum Transmission Unit)*. Die MTU-Werte unterscheiden sich geringfügig für die o. g. Ethernet-Varianten; sie liegen zwischen 1492-1500 Bytes.

15.5 Verkabelungssysteme und Hubs

Je nach eingesetztem Verkabelungssystem und der Netzwerk-Interfaces beträgt die Übertragungsgeschwindigkeit 10 Mbit/s (Standard), 100 Mbit/s (Fast-Ethernet) oder 1Gbit/s (Gigabit Ethernet), demnächst sogar 10 Gbit/s. Das Verkabelungssystem besteht aus „Twisted Pair"-Kabeln, Koax-Kabeln (veraltet) oder Glasfaserkabeln zur Verbindung von Segmenten.

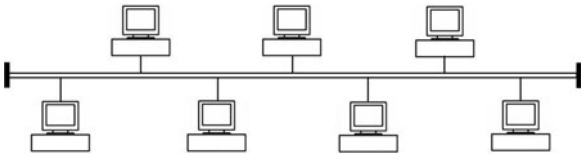
Physikalische Medien:

10 Base 5:	ThickWire (Yellow Cable), 10 MBit/s, steifes Koax-Kabel, max. Segmentlänge 500 m, Stations-Anschluss über externe Tranceiver, Stecker: AUI (*Attachment Unit Interface*), aufwendig, veraltet
10 BASE 2:	ThinWire (Cheapernet), 10 MBit/s, flexibles Koax-Kabel mit höherer Dämpfung, max. Segmentlänge: 185 m, Tranceiver integriert im Stationsadapter, Stecker BNC, billig, wird mehr und mehr durch 10 BASE-T ersetzt
10 BASE T, 100 BASE T:	TwistedPair (verdrillte Doppeladern),10 MBit/s bzw. 100 MBit/s (Fast Ethernet), ungeschirmte Doppeladern (UTP, Unshielded TwistedPair), geschirmte Doppeladern (SUTP, shielded UTP), Punkt-zu-Punkt - Verkabelung: jede Station wird mit einem eigenen Segment an einem HUB (Repeaterfunktion) angeschlossen, max. Kabellänge: 100 m, Stecker: RJ45, „strukturierte Verkabelung", billig, sehr gut wartbar! einfache Migration 10 MBit/s --> 100 MBit/s, am Häufigsten eingesetzt
10 BASE F, 100 BASE F:	Lichtleiter (Fiber), 10 MBit/s, 100 MBit/s Punkt-zu-Punkt-Verkabelung zwischen Netzkoppelelementen, Kabellängen bis zu 2 km, Backbone-Verkabelung, störsicher
1000 BASE T, 1000 BASE F:	Gigabit Ethernet, 1000 MBit/s, UTP oder Lichtleiter, Punkt-zu-Punkt-Verkabelung



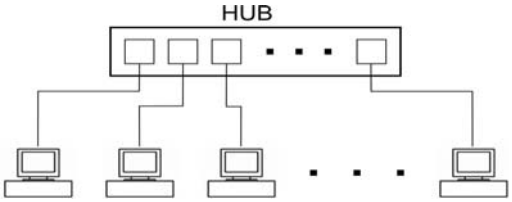
Ethernet Anschluss-Varianten

Das Verkabelungsystem bestimmt das äußere Erscheinungsbild des Netzwerks. Ältere Systeme benutzen Koaxialkabel als *shared medium*, an die die einzelnen Stationen angeschlossen sind. Die logische Busstruktur eines Ethernet-Netzes ist hier auch im Netzaufbau deutlich sichtbar. Das Netzwerkkabel wird an beiden Enden zur Vermeidung von Reflektionen mit dem Wellenwiderstand (50 Ohm) abgeschlossen. Koax-Systeme sind auf eine Übertragungsgeschwindigkeit von 10 MHz beschränkt. Sie werden heute nicht mehr als Neuinstallationen benutzt. Das hier erforderliche CSMA/CD-Verfahren gestattet nur eine Halbduplex-Kommunikation, d. h. eine Station kann nur zeitversetzt senden und empfangen.



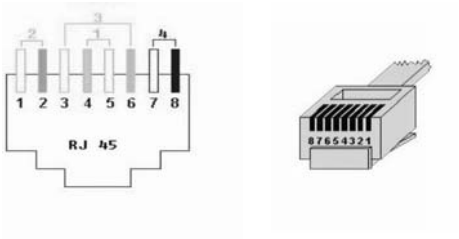
Koax-Verkabelung

TwistedPair-Kabel bestehen aus abgeschirmten (STP, *Shielded Twisted Pair)* oder nichtabgeschirmten (*UTP, Unschielded Twisted Pair*) verdrillten Doppeladern. Mit ihnen kann grundsätzlich nur eine Punkt-zu-Punkt-Verbindung eingerichtet werden. D. h. jede Station ist mit einem eigenen TP-Kabel an einen *Hub* als „Verteilzentrum“ angeschlossen. Der Hub ersetzt das gemeinsame Medium. Je nach Segmentgröße kann ein Hub 4..32 TP-*Ports* (RJ45 Anschlußbuchsen) besitzen. Für größere Netze lassen sich Hubs über einen *Uplink-Port* miteinander verbinden. Hubs für größere Segmente ersetzt man heute allerdings durch effizientere *Swiches* (→ s. Kap. 15.6.3).



Stationen am Hub

In der Praxis benutzt man meistens genormte UTP-Kabel der Kategorie 5 („cat5-Kabel“), die bis zu 1000 Mbit/s ausgelegt sind. Das Kabel besteht aus 2-4 Doppeladerpaaren. Zum Senden und Empfangen steht jeweils ein Aderpaar zur Verfügung. Die Signale werden erdfrei geführt. Twisted Pair Kabel besitzen an ihren Enden genormte 8-polige RJ45-Stecker.



RJ45 Stecker

RJ45 Steckerbelegung für 10BASE-T/100BASE-T Kabel:

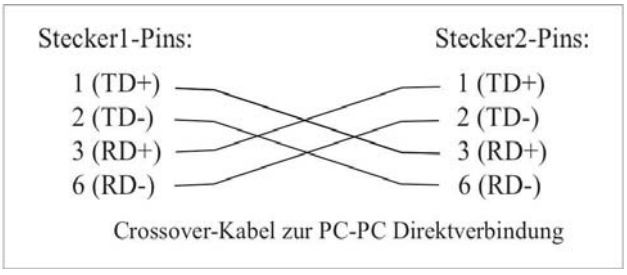
Stecker-Pin:	Signal:
1	TD+ (Transmit)
2	TD- (Transmit)
3	RD+ (Receive)
4	---
5	---
6	RD- (Receive)
7	---
8	---

Der Hub verbindet die Transmit-Pins eines Ports mit allen Receive-Pins der übrigen Ports. Er stellt das „shared medium“ dar: Jedweder Datenverkehr gelangt zu jedem Netzwerk-Interface. Gleichzeitiges Senden mehrerer Stationen kann zu Kollisionen am Netzsegment führen.

Voll-Duplex-Betrieb:
Die getrennte Signalführung der Sende- und Empfangsdaten in jeweils einem eigenen Twisted-Pair-Paar ergibt einen weiteren Vorteil gegenüber Koax-Systemen: es ist ein Voll-Duplex-Betrieb möglich, also gleichzeitiges Senden und Empfangen. Die meisten Hubs und Netzwerk-Interfaces unterstützen den Voll-Duplex-Betrieb, was zu einer Verdoppelung der Bandbreite führt (20Mb/s, 200Mb/s, 2000Mb/s).

Hubs sind keine Datenendgeräte. Sie besitzen keine (HW-)Adresse und man kann sie deshalb nicht im LAN adressieren.

Crossover-Verbindungen:
Besteht ein „entartetes“ Netzwerk nur aus zwei Stationen, ist ein Hub nicht erforderlich. Möchte man z. B. zwei PCs direkt miteinander verbinden, benötigt man aber ein sog. *Crossover*-Kabel. In ihm sind die Adern – ähnlich wie bei Nullmodemkabeln – paarweise vertauscht, so dass eine symmetrische Anordnung entsteht:



Crossover-Kabel zur PC-PC Direktverbindung

Auch Hub-Hub-Verbindungen sind „symmetrisch“ und benötigen daher ein Crossover-Kabel, wenn sie nicht einen speziellen Uplink-Port zur Verbindung zu einem weiteren Hub (oder Switch) besitzen. Am Uplink-Port hingegen übernimmt der Hub selber die Tauschung, so dass normale Kabel verwendet werden können. (Aber nicht Uplink-zu-Uplink verbinden! Dann ist alles zurückvertauscht!)

Neuere Geräte verfügen über eine *Auto MDI/MDI-x Funktion (Medium-Dependent Interface):* Ports, die mit dieser Sensorik ausgestattet sind, prüfen das an ihnen angeschlossene Kabel und tauschen selbständig im Bedarfsfall. Das ist sehr bequem, man kann bei diesen Geräten beide Kabeltypen einsetzen.

15.6 Ethernet – Netzkoppler

Es gibt unterschiedliche Gründe für den Einsatz von Netzkoppelelementen, z. B. Vergrößerung der räumlichen Ausdehnung, Lasttrennung, Anschluss unterschiedlicher Topologien und Verkabelungssysteme, Zusammenschluss einzelner LANs oder Übergänge in andere Netze (proprietärer EDV-Systeme).

Koppelelemente lassen sich klassifizieren nach der Ebene der Protokollschicht, in der sie arbeiten:

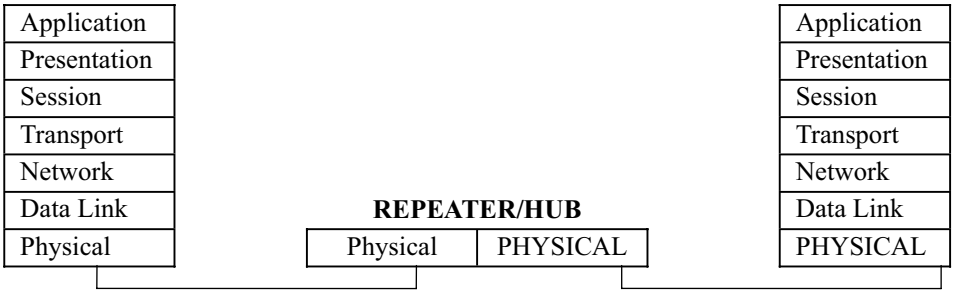
Netzwerk-Koppler		
Repeater /Hub	Schicht-1-Koppler	(Ethernet)
Bridge /Switch	Schicht-2-Koppler	(Ethernet)
Router	Schicht-3-Koppler	(IP)
Gateway-Rechner	Schicht-7-Koppler	(Anwendung)

Grundsätzlich ist mit jedem Koppelelement natürlich auch eine Verzögerungszeit (Latenz) verbunden, die den Datendurchsatz herabsetzt. Diese Verzögerungen sind umso länger, je mehr Schichten im Koppler „umgeschnürt“, d. h. umgesetzt werden.

In diesem Kapitel befassen wir uns zunächst nur mit Netzkopplern in Ethernet-Netzen.

15.6.1 Repeater und Hubs

Repeater und Hubs sind die einfachsten Koppler. Sie sind „dumme“ Signalverstärker, die Segmente gleicher Netztechnik, z. B. Ethernetsegmente, miteinander verbinden. Die Geräte wirken nur auf die Schicht 1, alle höheren Protokollschichten bleiben unbeeinflusst.



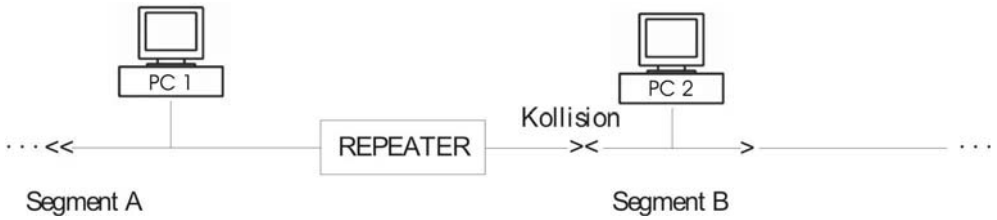
Netzkopplung mit Repeater oder Hubs

Ein Repeater kann eine bestimmte Realisierung der Schicht 1 „Physical“, z. B. Thinwire-Kabel, durch eine äquivalente Realisierung „PHYSICAL“, z. B. TwistedPair-Kabel ersetzen. Das in Schicht 2 festgelegte Medium-Zugriffsverfahren bleibt im Netz unverändert, z. B. CSMA/CD, ebenso alle höheren Schichten. Da Repeater keine eigene Netzwerkadresse besitzen, können sie nicht über das Netz gezielt angesprochen werden, so wie das bei den „intelligenteren“ Netz-Kopplern möglich ist.

Repeater verwendet man, um die Längenbeschränkungen eines Ethernetnetzes aufzuheben und um es durch Ankoppeln eines weiteren Segmentes physikalisch auszudehnen. Größere Distanzen lassen sich durch Remote-Repeater Paare überbrücken, ein durch zwei Repeater begrenztes

Lichtleitersegment, das bis zu 1000 m lang sein kann. Am Lichtleitersegment befinden sich dabei keine Endgeräte.

> Repeater und Hubs verhindern nicht Kollisionen und ihre Verbreitung bei CSMA/CD! Sie können Netzüberlasten nicht reduzieren.



Repeater verhindern Kollisionen nicht

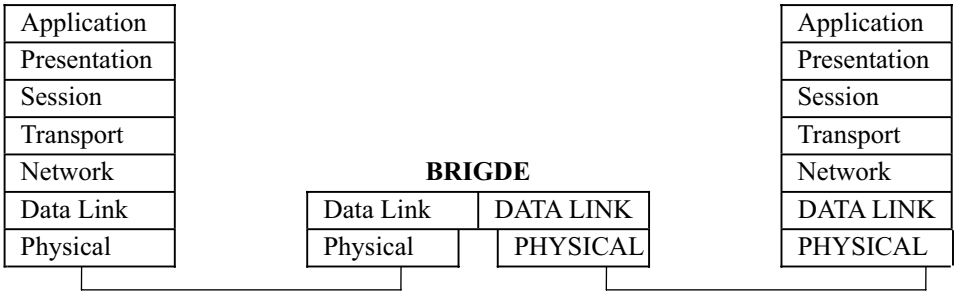
Eine besondere technische Ausführung ist der Multiport-Repeater: Er koppelt nicht nur zwei, sondern mehrere Segmente, wodurch sich dann eine sternförmige Verkabelung ergibt. Multiport-Repeater für BASE-T Verkabelungssysteme werden „Hub“ genannt. Sie wurden schon im Kap. 15.5 als die zentralen Geräte eines TwistedPair Ethernetnetzes vorgestellt. Sie verfügen meistens über 4, 8, 16 oder mehr RJ45-Ports, über die jeweils *nur ein* Gerät angeschlossen ist.

Wie im Kap. 15.5 beschrieben, lassen sich mehrere Hubs mit *Crossover*-Kabeln oder über *Uplink-Ports* zu einem größeren Netzwerk verbinden. Nach der Ethernet-Spezifikation dürfen aber auf der Verbindungsstrecke zweier Kommunikationspartner nicht mehr als 4 Repeater oder Hubs durchlaufen werden (sog. „Repeater-Regel“), wobei ein Remote-Repeater-Paar als *ein* Repeater zählt. Abhilfe schafft der Einsatz von Switches.

> **Kollisions-Domäne:**
> Alle in einem Netzwerk über Hubs und Repeater verbundene Endgeräte liegen in derselben „Kollisions-Domäne“, d. h. sie liegen logisch am gleichen *shared medium:* Jeder sendende Host belegt das gesamte Medium und jeder auftretende Ethernet-Frame gelangt an jedes Netzwerk-Interface.

15.6.2 Bridges

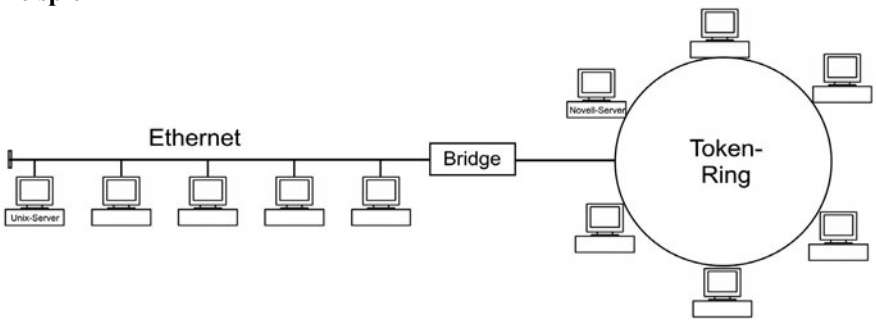
Mit einer Bridge können Netzwerk-Segmente eines LANs gekoppelt werden. Bridges sind „intelligente“ Geräte, sie arbeiten auf der Schicht 2:



Segmentkopplung mit Bridge

Da hier auch das Medium-Zugriffsverfahren umgesetzt werden kann, lassen sich verschiedene Netzwerke, z. B. ein Token-Ring-Netzwerk mit einem Ethernet-Netzwerk (CSMA/CD) miteinander verbinden. „Umsetzen“ heißt, dass die Nachrichtenpakete bis zur Schicht 2 „aufgeschnürt“ werden (und nur bis dahin!) und gegebenenfalls nach einer anderen Schicht-2-Implementierung neu paketiert werden.

■ **Beispiel**

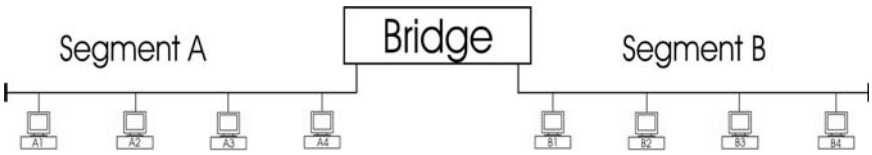


Kommunikation Ethernetstation ↔ Token Ring Station ■

Da die Bridge die höheren Schichten transparent überträgt, spricht man von der „Protokoll-Transparenz“ von Brücken (wobei man die Protokolle ab Schicht 3 im Auge hat). Eine Brücke weiß also nicht, ob die übertragenen Datenpakete z. B. zu einem NetWare-Netz (IPX/SPX) oder zu einem Unix-Netz (TCP/IP) gehören.

Bridges verfolgen den Datenverkehr auf den beiden angeschlossenen Segmenten. Jeder eintreffende Ethernet-Frame wird auf seine Quell- und Ziel-MAC-Adresse untersucht. Anhand der Quelladressen „lernen“ Bridges selbständig, welche Stationen auf welcher Segmentseite liegen. Sie bauen im Laufe des Betriebs zu jedem Port Adresstabellen auf und leiten nur diejenigen Frames weiter, deren Zieladresse im anderen Segment liegt. Stellt die Bridge fest, dass Absender und Empfänger im gleichen Segment liegen, wird das betreffende Datenpaket nicht auf das andere Segment übertragen. Wegen dieser Filterwirkung benutzt man Bridges zur „Segmentierung“ von Netzen, d. h. es wird eine Trennung des Datenverkehrs in Teilnetze erreicht. Durch diese Lasttrennung treten weniger Kollisionen auf, wodurch die *Performance* des gesamten Netzes steigt.

■ **Beispiel**



Segmentierung durch eine Bridge

Kommunikation innerhalb des Segments A wird nicht zum Segment B gekoppelt und somit das Medium von Segment B nicht belegt. Es kann also gleichzeitig z. B. A1 ↔ A3 und B2 ↔ B4 kommunizieren. Ebenso werden Kollisionen innerhalb der Segmente nicht übertragen. Nur bei der Kommunikation z. B. zwischen Ax ↔ By überträgt die Bridge, wodurch das gesamte Medium belegt ist.

■

Bridges:
Aufteilung bzw. Kopplung von zwei (Ethernet-)Segmenten.
Lasttrennung durch Erzeugung von zwei getrennten Kollisions-Domänen.
Broadcasts werden immer übertragen.

Während man früher Bridges in „gewachsenen“ Netzen einsetzte, um Überlastungen zu vermeiden, benutzt man heute dafür leistungsfähigere Switches.

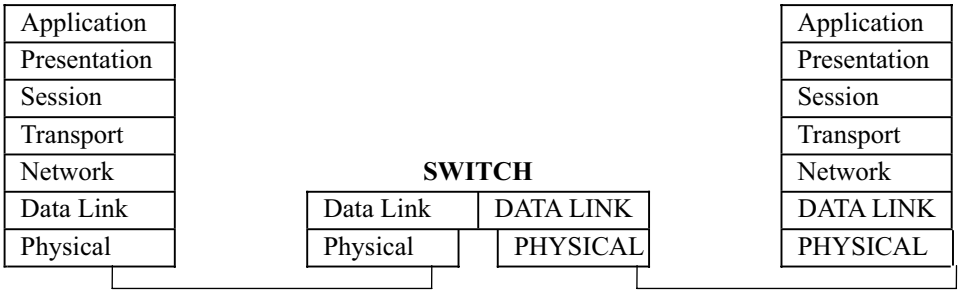
15.6.3 Switches

Es gibt zwei Ansätze, um der ständigen Forderung nach einem höheren Datendurchsatz im Netzwerk nachzugehen:

– Erhöhung der Übertragungsrate, z. B. von 100 Mb/s auf 1 Gb/s oder höher;
– Segmentierung des Netzes in mehrere „Kollisions-Domänen“ durch Switching-Technologien.

Während die Umstellung auf eine höhere Übertragungsrate meistens mit erheblichen Kosten verbunden ist, bietet die Switching-Technologie eine Möglichkeit, den Datendurchsatz eines **bestehenden** Netzwerks Kosten sparend zu verbessern. Für höchste Ansprüche werden beide Möglichkeiten kombiniert.

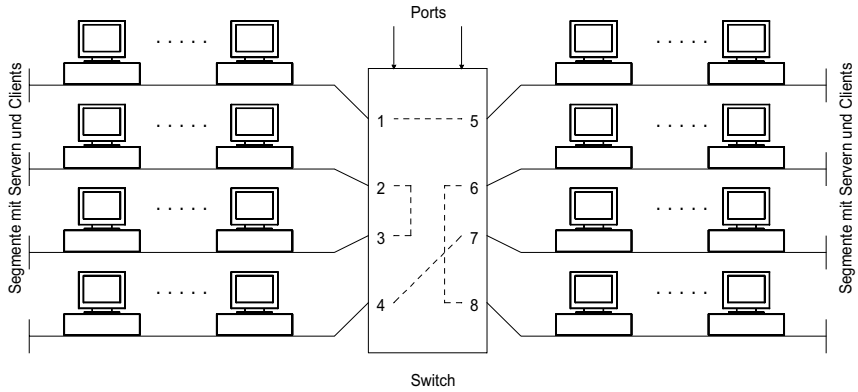
Neben dem Standard-Switch, der auch als „Layer-2-Switch“ bezeichnet wird, gibt es „Layer-3-Switches“, die nicht hier, sondern im Zusammenhang mit Routern vorgestellt werden.



Segmentkopplung mit Switch

Ein Switch ist eine Multiport-Bridge mit einem schnellen internen Bus als *backplane.* Er arbeitet wie eine Bridge auf Schicht 2. Ebenso wie Bridges lernen Switches, welche Geräte an welchem Port angeschlossen sind. Beim Eintreffen eines Datenpakets an einem Port entscheidet der Switch anhand der MAC-Adresse, zu welchem Ausgangsport durchgeschaltet werden muss (Bridge-Funktion). Ein besonderes Merkmal von Switches liegt in ihrer Fähigkeit, mehrere Übertragungen zwischen unterschiedlichen Segmentpaaren gleichzeitig und unabhängig voneinander durchzuführen. Dadurch erhöht sich die effektive Bandbreite des Netzes entsprechend.

■ **Beispiel:**

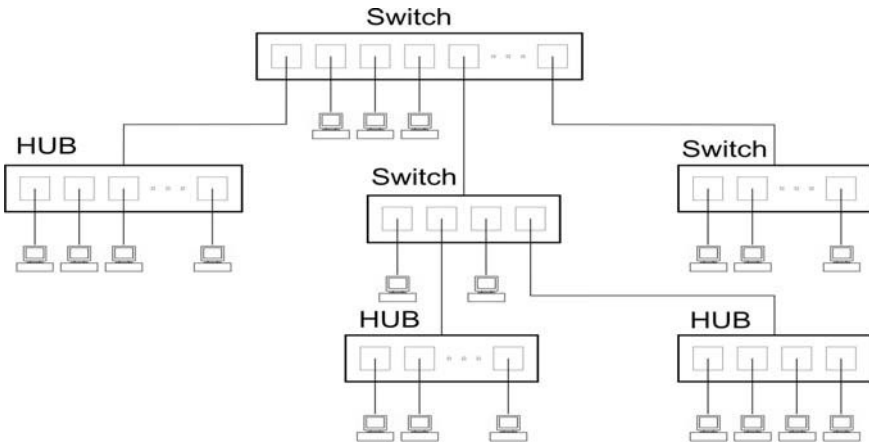


„Direktvermittlung“ eines 8-Port-Switches. Zur gleichen Zeit können 4 Segmentverbindungen parallel mit z. B. je 100 Mb/s hergestellt werden. ■

Damit ein n-Port-Switch nicht die Kommunikation bremst, muss die *backplane* in der Lage sein, die n-fache LAN-Basisgeschwindigkeit zu verarbeiten, was sich bei Voll-Duplex-Übertragungen sogar noch einmal verdoppelt. Die *backplane* Bandbreite ist daher neben Port-Anzahl und der LAN-Geschwindigkeit ein wichtiger Geräteparameter. Er beträgt meist einige -zig Gbit/s.

Switches:
Aufteilung in verschiedene „Kollisions-Domänen“ durch Segmentierung.
Dadurch starke Verkehrsentkopplung.
Eintreffende Broadcasts werden auf alle Ports „geflutet“.

Um die Vorteile eines Switches wirklich auszunutzen, muss seine Position innerhalb des LANs sorgfältig überlegt werden. Optimal sind Netzwerk-Organisationen, bei denen ein Switch autonome Arbeitsgruppen (d. h. Server und Clients) eines LANs miteinander verbindet.



Beispiel: LAN-Aufbau mit Switches und Hubs

Äußerlich unterscheidet sich ein Switch nicht von einem Hub. Wie dieser besitzen die häufigsten Switches 4 bis 32 RJ45-Ports für Twisted-Pair Verkabelung. Die Ports arbeiten in der Regel mit „Auto-Sensing“ für 10/100/1000 Mbit/s und übertragen im Voll-Duplex-Verfahren, sofern der Verbindungspartner dies unterstützt. Switch ↔ Switch oder Switch ↔ Hub-Verbindungen sind möglich mit *Crossover*-Kabeln oder mit 1:1-Kabeln über Uplink-Ports. Unterstützt das Gerät die *MDI/MDI-x-Funktion*, lassen sich beide Kabeltypen nutzen (→ s. Kap. 15.5). Switches sind wie Bridges *Plug-and-Play* Geräte, eine Konfiguration ist nicht erforderlich.

„Echtzeit“-Ethernet:
Setzt man die Segmentierung weiter fort, gelangt man zur Mikrosegmentierung: hier besteht das *shared medium* nur noch aus einem einzigen Endgerät, ist also nicht mehr „*shared“.* Formal wurde ein Hub durch ein Switch ersetzt. Nun besteht zwischen je zwei Endgeräten eine Voll-Duplex Punkt-zu-Punkt-Verbindung. Kollisionen können nicht auftreten, das CSMA/CD-Verfahren ist überflüssig und wird nicht eingesetzt. Damit wird Ethernet im Zeitverhalten „deterministisch“, was eine Voraussetzung für die Echtzeitverarbeitung ist. Wie bei jeder Datenverbindung ist jedoch eine Flusskontrolle erforderlich, die sich bei Verarbeitungsengpässen nicht wie bei *shared media* über „künstliche“ Kollisionen steuert, sondern dazu ein Handshake-Verfahren nutzt: Ähnlich dem XON/XOFF-Protokoll steuern die Endgeräte den Datenfluss mit speziellen Pausepaketen. Sowohl der Switch als auch das Endgerät, also z. B. der Ethernet-Adapter des PC, müssen diese Flusskontrolle (IEEE 802.3x-Norm) unterstützen.

15.7 Kommandos zur Ethernet-Konfiguration

Ethernet-Adresse:
Mit den Konsolen-Kommandos

```
> ipconfig /all       (unter Windows)
```

oder

```
> ifconfig            (unter Unix)
```

kann man sich die eigene Ethernet-Adresse (neben vielen weiteren Infos) ausgeben lassen.

MTU:
Die von den höheren Protokollen an die Ethernet-Schichten übergebenen Datenpakete können größer sein als die MTU des Netzwerkadapters. In diesen Fällen wird das Paket in eine Folge kleinerer Frames zerlegt. Diese Aufteilung heißt *Fragmentierung*. Die für eine bestimmte Verbindung benutze MTU lässt sich mit dem vielseitigen ping-Kommando experimentell ermitteln (auch bei Internet-Verbindungen):

```
>ping  -l <Anzahl der gesendeten Bytes>  -f   <hostadresse>
    (Flag –f:  Fragmentierung nicht zulassen!)
z. B.   >ping  -l 1500  -f   195.72.108.12
```

Gibt man für die Bytes-Anzahl l einen Wert > 1500 an, muss auf jeden Fall fragmentiert werden. Verbietet man dieses aber durch ein gesetztes f-Flag, erscheint auf jeden Fall eine Fehlermeldung. Erhält man eine Fehlermeldung für 1500, so muss man die Bytes-Anzahl verkleinern. Durch Variation der gesendeten Bytes-Anzahl kann man die MTU ermitteln. Beim Ausprobieren werden Sie feststellen, dass je nach benutztem Zielhost die MTU unterschiedlich sein kann

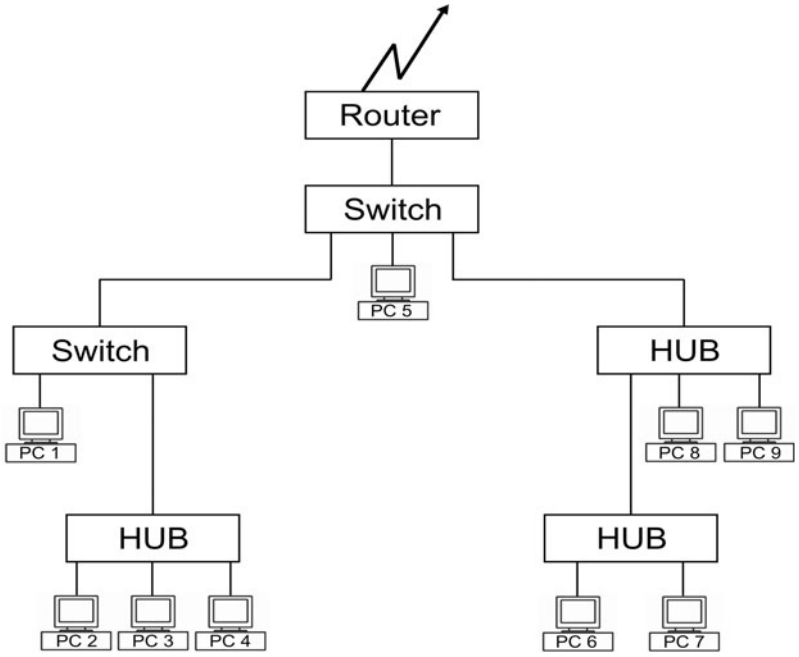
(im Bereich 1400-1500). Es gilt das „schwächste Glied der „Kette“, d. h. der kleinste MTU-Wert aller durchlaufener Ethernetsegmente auf dem Weg zum Ziel.

Paket-Sniffer:
Netzwerkadministratoren benutzen zur Netzwerk-Analyse und Fehlererkennung sog. Ethernet-Paket-„Sniffer“, die *alle* am Netzsegment auftretenden Frames empfangen und darstellen können. Dabei wird das Netzwerk-Interface der Analyse-Station im „Promiscous Mode“ betrieben, der bewirkt, dass die Empfangsadressen-Überprüfung ausgeschaltet ist, d. h. es werden nicht nur die an die Station andressierten Pakete empfangen, sondern alle Pakete, die sich auf dem Netzsegment befinden. Die meisten Ethernet-Interfaces beherrschen diesen Mode. Sniffer-Software ist im Internet z. B. unter dem Programm-Namen *ethereal* sowohl für Windows als auch für Linux frei verfügbar.

15.8 Aufgaben

1) Wie viel „Platz“ belegt ein Bit auf dem Netzwerkkabel von Fast Ethernet?
2) Welche Auswirkungen hätte es, wenn man in einem Ethernet-Netz die vorgeschriebene Mindest-Paketlänge herabsetzen würde?
3) Wie hoch ist der maximal erreichbare Paketdurchsatz (Pakete/s) bei einem Standard-Ethernetnetz (10 Mbit/s), wenn ein vorgeschriebener Mindest-Paketabstand von 9,6 µs (*inter frame gap*) eingehalten wird?
4) Ethernet-Adressen sind weltweit eindeutig. Warum eignen sie sich trotzdem nicht zur LAN-übergreifenden Kommunikation?
5) Überprüfen Sie unter der „erweiterten“ Konfiguration Ihres Ethernet-Adapters, welcher *Media Type* eingestellt ist. Hier findet man meistens: *Auto Negotiation*, also das selbständige Aushandeln der schnellsten Übertragungsart, die der Partner (Hub, Switch) unterstützt. Setzen Sie hier explizit unterschiedliche Werte aus dem angebotenen Katalog ein und überprüfen Sie die Kommunikation. Unterstützt Ihr System z. B. 100 Mbit/s Voll-Duplex?

6) Ein Netzwerk habe den folgenden Aufbau:



Es gebe die folgenden Kommunikationen: k1: PC1 <---> Internet;
k2: PC2 <---> PC6;

Auf PC8 läuft ein Paket-Sniffer (z. B. ethereal).

a) Kann es zu Kollisionen kommen zwischen k1 und k2?

b) Welche Datenpakete von k1 und k2 sind am Sniffer (PC8) sichtbar?

16 Netzverbindungen mit TCP/IP

Um eine Kommunikation zu einem Verbindungspartner (Host) aufzunehmen, der entweder innerhalb des eigenen LANs oder außerhalb in einem anderen LAN oder Internet liegt, wird eine zweite, logische Adressierung in der Schicht 3 des OSI-Modells eingeführt. Die Schicht 3 ist für die Host-Adressierung und die Wegewahl (*Routing*) zum Verbindungspartner zuständig. Die Schicht 4 stellt einen *gesicherten* Ende-zu-Ende-Kommunikationskanal zwischen zwei Hosts bereit. Die Schichten 3/4 sind die Transportprotokolle des Netzwerks.

Bei der Installation eines Netzwerks vergibt der Netzwerkadministrator für jeden Netzknoten eine eindeutige Netzknotenadresse, deren Form von den eingesetzten Transportprotokollen abhängt. Anwendungen sprechen grundsätzlich diese Adresse an.

Beispiele für Host-Adressen und Transportprotokolle:

Transportprotokoll:	*Beispiel einer Knotenadresse:*
TCP/IP: Schicht 3: IP *(Internet Protocol)* Schicht 4: TCP *(Transmission Control Protocol)* -oder- Schicht 4: UDP (*User Datagramm Protocol)*	193.175.39.112
IPX/SPX: (NetWare) Schicht 3: IPX *(Internet Packet Exchange)* Schicht 4: SPX *(Sequential Packet Exchange)*	0002.0060B031A81A
NetBEUI : (ältere Windows-Systeme) Schicht 3/4: *(NetBIOS Extended User Interface)*	MOMO1

Das auch im Internet benutzte Transportsystem TCP/IP ist heute Standard in allen Lokalen Netzen. Auf der Basis dieser Protokolle (TCP/IP Protokoll-Suite) werden ständig neue Anwendungen entwickelt. LANs auf TCP/IP-Basis können Internet-Anwendungen ohne Protokollumsetzungen direkt ansprechen.

Das sehr effiziente IPX/SPX wurde von NOVELL für NetWare-Netze entwickelt. Es wird jedoch auch dort zunehmend von TCP/IP verdrängt. Sowohl Windows als auch Linux unterstützen IPX, so dass es häufig in heterogenen Netzwerken neben TCP/IP zusätzlich aktiviert wird.

Routbare Protokolle:

Grundsätzlich nennt man ein Transportprotokoll „routbar“, wenn aus der Adresse neben der Endgeräteadresse (Hostkennung, Host-ID) auch eine Netzwerkkennung (Netz-ID) ersichtlich ist, die Adresse also aus zwei Teilen, der Netzwerkkennung **und** der Hostkennung besteht. Adressen mit unterschiedlichen Netzkennungen liegen nicht im gleichen LAN und müssen (besser: „können“) über Router vermittelt werden. IP und IPX arbeiten mit numerischen Netzadressen der Form *Netzwerk-ID + Host-ID*, die sich gut für Algorithmen zur Wegewahl eignen. Router werten nur die Netzwerkkennungen von Adressen aus und vermitteln zwischen unterschiedlichen Netzwerken. Bei IPX ist die Netzkennung deutlich durch einen Punkt vom Host-

teil abgetrennt. Der Hostteil besteht aus der MAC-Adresse. Etwas weniger offensichtlich ist die Netzwerkkennung bei IP-Adressen im vorderen Adressteil (s. unten). IP und IPX sind routbare Protokolle, eine Kommunikation ist LAN-übergreifend über Router möglich.

NetBEUI mit der NetBIOS-Schnittstelle dient in älteren Windows-Versionen als Transportsystem. Es besitzt keine ausgeprägte Schicht 3, enthält keine Netzkennung und ist deshalb nicht routebar. Als Hostadressen dienen bis zu 15 Zeichen lange (NetBIOS-Namen). NetBEUI wird nur noch in kleinen Netzen eingesetzt, wenn TCP/IP nicht verfügbar ist und die Kommunikation auf ein einzelnes LAN beschränkt bleibt.

Routbare Protokolle:

Ermöglichen LAN-übergreifende Kommunikation;

Adressaufbau: Netzkennung + Hostkennung;

Beispiele: TCP/IP , IPX/SPX

Beispiel für ein NICHT-routbares Protokoll: NetBEUI

16.1 IP-Adressen

Eine IP-Adresse ist 32 Bit lang und wird meistens in der 4-Byte-Schreibweise angegeben:

xxx.xxx.xxx.xxx („dotted“ Schreibweise)

z. B.: 193.178.12.56

Jedes Byte xxx kann den Wertebereich 0…255 annehmen. Zur Trennung zwischen Netzkennung und Hostkennung dient die *Netzwerkmaske*, die bei der Konfiguration eines Netzknotens neben der IP-Adresse immer mit angegeben werden muss. Die Netzwerkmaske besteht ebenfalls aus 32 Bit. Sie *maskiert* in der Binärebene denjenigen Teil der IP-Adresse, der zur Netzkennung gehört.

IP-Adresse und Netzwerkmaske:

Die Netzwerkmaske bestimmt denjenigen Anteil einer IP-Adresse, der zur Netzkennung gehört.
Alle Knoten eines Netzes, die zum gleichen Netzwerk gehören, besitzen die gleiche Netzwerkmaske.

■ **Beispiel:**

Es sei konfiguriert:

IP-Adresse:	193.178.12.56	binär: 11000001.10110010.00001100.00111000
Netzwerkmaske:	255.255.255.0	binär: 11111111.11111111.11111111.00000000

Die Maske markiert durch „1“ den Adressteil für die Netzkennung.

➠ Netzwerk: 193.178.12.0

➠ Host: 56

Alternative Schreibweise: 193.178.12.56/24 ■

Möchte z. B. ein PC ein Datenpaket an eine bestimmte IP-Adresse (anderen Rechner) absenden, prüft er die Zieladresse gegen seine Netzwerkmaske und findet so heraus, ob der Zielrechner im eigenen Netzwerk liegt. Falls dies der Fall ist, kann das Datenpaket direkt an den Empfänger abgesandt werden. Liegt der Zielrechner jedoch nicht im eigenen Netz, muss das Datenpaket an die IP-Adresse des Routers gesandt werden, der die weitere Vermittlung übernimmt.

Das Arp-Protokoll:

Nun ist aber weder der Zielhost im eigenen Netz noch der Router des Netzes über die IP-Adresse direkt ansprechbar, da – wie wir in Kap. 15.3 festgestellt hatten – auf unterster Ebene aller Datenaustausch auf MAC-Adressen beruht. Es ist also eine Adressübersetzung

IP-Adresse ↔ MAC-Adresse (Ethernet-Adresse)

vor dem Absenden erforderlich. Diese Umsetzung übernimmt das *Arp-Protokoll (Address Resolution Protokoll)*. Jeder Netzknoten speichert in einem sog. *Arp-Cache* die Umsetzungstabelle.

■ **Beispiel:**

Ein Benutzer auf Rechner A möchte mit Rechner B kommunizieren. Der Netzwerkadministrator hat für die beiden Rechner die IP-Adressen 130.43.115.4/24 und 130.43.115.7/24 vergeben. Die beiden Rechner besitzen Ethernet-Interfaces mit den HW-Adressen 08-00-2B-93-15-3A und 00-80-A3-04-4E-D2.

Schicht		Adresstyp	Rechner A	Rechner B
4 3	Trans- port	IP-Netz- knoten:	130.43.115.4/24	130.43.115.7/24
2 1	HW- Adapter	Ethernet HW-Adr.:	08-00-2B-93-15-3A	00-80-A3-04-4E-D2

Der Benutzer auf Rechner A kennt nur die IP-Adresse seines Partners und gibt diese an.

Der Rechner A erkennt anhand der Netzwerkmaske, dass Rechner B im eigenen Netzwerk liegt. Er durchsucht seinen Cache, um die zugehörige Ethernet-Adresse zu ermitteln. Enthält der Cache keinen zutreffenden Eintrag, sendet er eine Broadcast-Nachricht (*ARP-Request*) an alle Stationen des Netzes mit der Ethernet-Zieladresse FF-FF-FF-FF-FF-FF: „Wer kennt die Ethernet-Adresse von 130.43.115.7?“ Der Rechner B antwortet mit seiner Ethernet-Adresse. Erst jetzt kann Rechner A Schicht-2-Datenpakete (Frames) mit der erforderlichen Ethernet-Empfängeradresse bilden und absenden. Rechner A speichert die IP ↔ Ethernet-Adresszuordnung als Tabelleneintrag in seinem Arp-Cache für eine eventuelle spätere Nutzungen.

■

Der Arp-Cache arbeitet dynamisch: Nach dem Einschalten des Rechners ist er leer und füllt sich im Laufe der Zeit. Arp-Einträge verbleiben nur einige Minuten im Cache. Wird ein bestimmter Eintrag in dieser Zeit nicht gebraucht, wird er wieder gelöscht.

Physikalische und logische Adressen:

Hier stellt sich nun die Frage: Warum sind in Netzwerken überhaupt zwei Adressierungen erforderlich, die physikalische MAC-Adresse (Ethernet-Adresse) der Schicht 2 und eine zusätzliche logische Schicht-3-Adresse, z. B. die IP-Adresse?

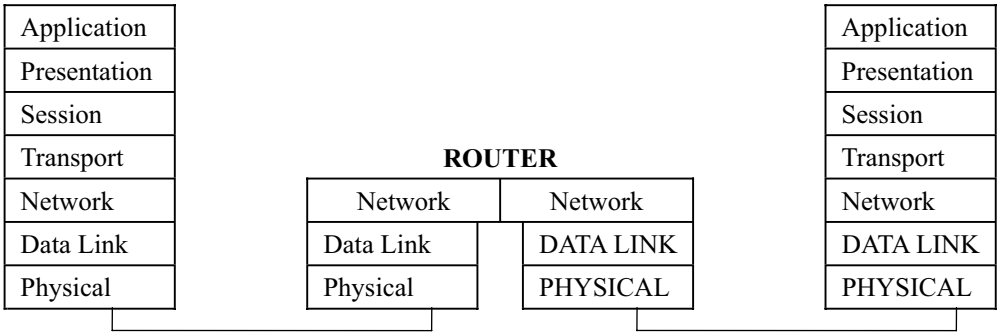
Wir erinnern uns: Die physikalische Adressierung ist nötig, um hardwarenah vom Netzmedium die Adressbits abzutasten und mit der eigenen physikalischen Adresse zu vergleichen. Jegliche Kommunikation in Netzwerken beruht letzlich auf dieser Adresse in den übertragenen Frames. Eine MAC-Adresse (Ethernet-Adresse) ist also zwingend erforderlich, unabhängig davon, ob darüberliegende Schichten TCP/IP, IPX/SPX oder andere Transportsysteme nutzen. Ethernet-Adressen haben jedoch einen großen Nachteil, was sie für die Verwaltung eines Netzes untauglich macht: sie sind „kryptisch“ ohne eine Systematik aufgebaut, z. B. 08-00-CA-9F-45-5B. Die Adressen sind so zu nutzen, wie sie die Hersteller in den Netzwerkadaptern „eingebrannt“ haben. Der Adressraum ist zwar eindeutig, jedoch „flach“ ohne jede Hierarchie. Ein Netzwerkadministrator müsste akribisch Listen mit Ethernet-Adressen verwalten, um zu wissen, wie ein bestimmter Rechner anzusprechen wäre. Der Hauptnachteil ist jedoch: Ethernet ist nicht routbar! Es gibt keine Netzkennung und somit keine Kommunikation über das LAN hinaus.

Diese Nachteile werden durch die Einführung eines frei wählbaren logischen, von der Hardware unabhängigen Adressierungsschemas vermieden. Ein 10-Rechner-LAN könnte z. B. mit den IP-Adressen 195.18.16.1, 195.18.16.2,....,195.18.16.10 konfiguriert werden. Der vordere Teil der Adresse 195.18.16 kennzeichnet das Lokale Netz. Adressen mit anderer Netzkennung können über Router erreicht werden.

16.2 Router

Am Router "endet" ein einzelnes LAN, er bildet das „Tor zur Außenwelt“ und wird deshalb im „Internetjargon“ auch *Gateway* genannt (Nicht zu verwechseln mit dem Schicht-7-Koppler, der ebenfalls als *Gateway* bezeichnet wird!). Liegt der Verbindungspartner nicht im eigenen Netz, so wird der Router angesprochen, der die Datenpakete zu einem anderen Netz vermittelt. Die Adressräume auf beiden Seiten eines Routers gehören also zu unterschiedlichen Netzwerken.

Router sind Schicht-3-Koppler:



Netz-Kopplung über Router

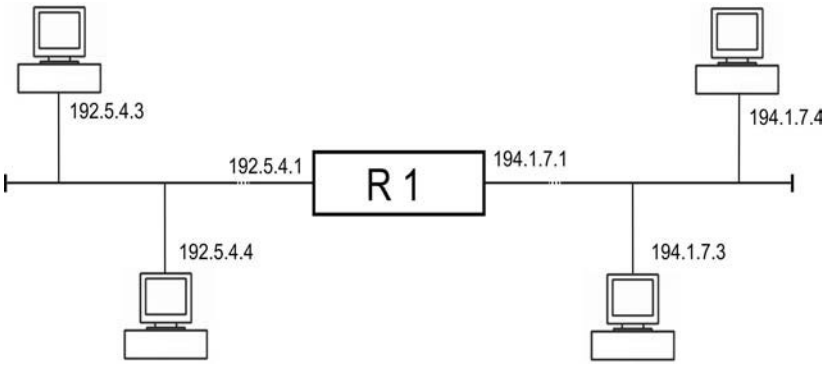
Router sind protokollabhängig, da sie die Informationen Schicht-3-protokollspezifisch umsetzen. Übliche Router beherrschen jedoch meistens mehrere Protokolle, d. h. sie bestehen eigentlich aus mehreren voneinander unabhängigen Geräten im gleichen Gehäuse z. B. für die routbaren Protokolle IP, IPX oder DECnet (DECnet ist ein proprietäres Transportprotokoll von DEC). In der Praxis wird oft nur das IP-Routing aktiviert.

Router sind keine „Plug & Play"-Geräte wie Repeater, Bridges oder Switches, ihre Konfigurierung in größeren Netzen ist recht mühsam.

Router:
Koppeln eigenständige Netze mit unterschiedlicher Netzwerkkennungen.
Jedes Router-Interface erhält eine IP-Adresse des jeweils angeschlossenen Netzes.
Broadcasts sind immer netzspezifisch. Sie werden durch Router nicht weitergeleitet.

■ **Beispiel:**

Die Netze 192.5.4.0/24 und 194.1.7.0/24 sind über einen Router R1 gekoppelt:



■

Jedes Router-Interface erhält eine eigene IP-Adresse aus dem zugehörigen IP-Adressraum des jeweils angeschlossenen Netzes. Stellt eine Station anhand der Netzwerkmaske fest, dass sein Verbindungspartner nicht im eigenen Netz liegt, addressiert er den Router. Jede Station eines LAN muss daher die IP-Adresse „seines" Routers kennen. Zu der Konfiguration einer LAN-Station gehört daher neben der eigenen IP-Adresse und der Netzwerkmaske die IP-Adresse des *Default*-Routers, der die Weiterleitung übernimmt.

Liegt der Verbindungspartner nicht im angrenzenden Netz, müssen weitere Router die Weiterleitung zum Zielhost übernehmen. In großen Router-vermaschten Netzen wie dem Internet (→ s. Kap. 18) werden oft 10-20 „Zwischennetze" durchlaufen, bis das Zielnetz am letzten Router erreicht wird. Jeder Router auf dem Weg zum Ziel muss entscheiden, welcher Router, genauer: welche IP-Adresse eines Routers als nächstes „Etappenziel" angesprochen werden soll. Dabei kann es auch vorkommen, dass aufeinander folgende Datenpakete auf unterschiedlichen Wegen zum Ziel geroutet werden, weil vielleicht auf dem einen Weg eine temporäre Störung oder Netzüberlastung vorliegt.

Woher hat ein Router diese Information?

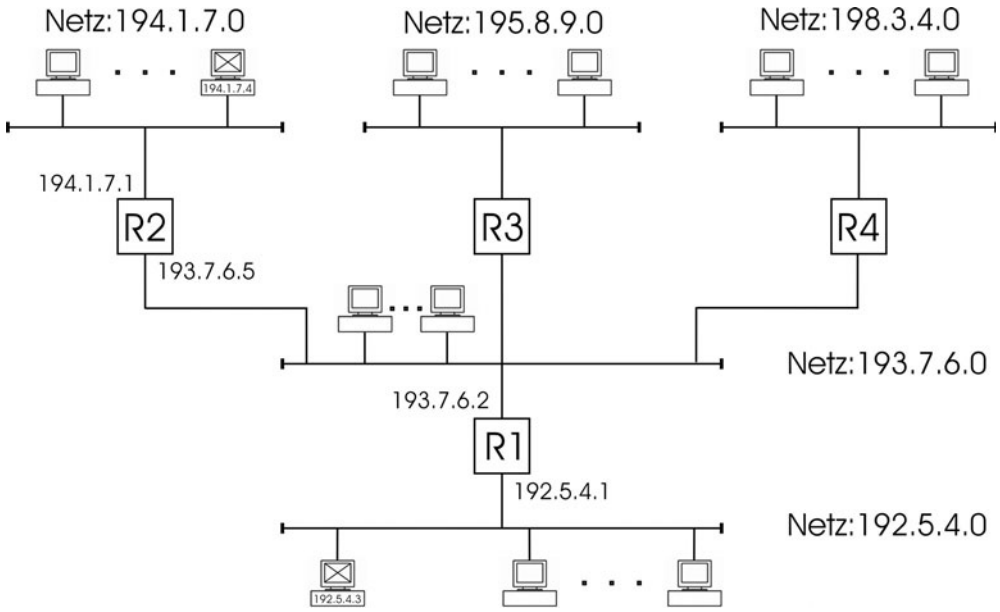
Es gibt „statische" und „dynamische" Informationen:

In kleineren Netzen mit wenigen Routern konfiguriert der Netzwerk-Administrator einen Router durch statische Routingtabellen darüber, welche Netze über welchen Router erreichbar sind. Unbekannte Netze gelangen anhand eines „default"-Tabelleneintrags zum übergeordneten Router, der mehr Informationen besitzt. Den weitaus größeren Anteil der Routingtabellen wird jedoch dynamisch aufgebaut: Router kommunizieren selbständig über eigene Routingprotokolle und informieren sich gegenseitig, welche Netze über sie erreichbar sind. Der Netzwerkadministrator muss hier nur eines der verfügbaren Routingprotokolle am Router aktivieren, die im entsprechenden Netz genutzt werden. Die Router bauen so ihre Routingtabellen selbständig auf und verändern sie auch dynamisch.

Beispiele für Routing-Protokolle sind *RIP (Routing Information Protocol)* und *OSPF (Open Shortest Path First)*. Obwohl sie direkt auf IP aufsetzen, steckt in ihnen eine große Intelligenz! Routing-Kommunikation ist für Endgeräte unsichtbar, macht jedoch in *backbone*-Netzen wie dem Internet einen erheblichen Anteil der Gesamtkommunikation aus, da Router in Abständen von wenigen Sekunden ihre Informationen ins Netz „broadcasten".

■ **Beispiel:** Routing

Der Rechner 192.5.4.3 möchte ein Datenpaket an den Rechner 194.1.7.4 senden.
Quell- und Ziel-IP werden im IP-Paketkopf (IP-Header) eingetragen.



Ablauf:

1) Der Absender erkennt an seiner Netzwerkmaske, dass der Zielrechner nicht im eigenen LAN liegt. Er sendet daher das Datenpaket an den lokalen (Default-) Router R1 (192.5.4.1), in dem er das IP-Paket in ein „Ethernet-Umschlag" verpackt und mit der Ethernetadresse (MAC-Adresse) des Routers R1 versieht. Die Ethernetadresse des Routers wird über eine ARP-Anfrage ermittelt („Wer kennt die Ethernetadresse von 192.5.4.1?).

2) Der Router R1 verwirft beim Eintreffen des Datenpakets den „Ethernet-Umschlag“ und gelagt an den IP-Header und die IP-Zieladresse. Anhand seiner gespeicherten Routing-Tabellen erkennt er, dass das Datenpaket an den Router R2 mit 193.7.6.5 weitergeleitet werden muss. Er erfragt per ARP-Anfrage die Ethernet-Adresse von R2, verpackt das IP-Paket in einen neuen „Ethernet-Umschlag“ mit der MAC-Adresse von R2 und sendet das Paket ab.

3) Der Router R2 entfernt beim Eintreffen des Datenpakets den „Ethernet-Umschlag“ und gelangt an den IP-Header und die IP-Zieladresse. Er erkennt, dass das Zielnetzwerk 194.1.7.0 direkt an seinem Netzwerkinterface abgeschlossen ist. Er erfragt die Ethernetadresse des Zielrechners mit IP 194.1.7.4 (ARP), bildet einen neuen „Ethernet-Umschlag“ mit dieser Adresse und sendet das Datenpaket zum Zielrechner ab.

■

In großen vermaschten Netzen (z. B. Internet) kann es vorkommen, dass ein Router fehlerhaft arbeitet oder falsch konfiguriert wurde und eintreffende IP-Datenpakete falsch geroutet werden. z. B. könnte Router A den Router B adressieren, dieser wiederum den Router A, wodurch eine „Paketfalle“ entstehen würde. Ebenso könnten mehrere Router einen Kreisverkehr aufbauen, aus denen die Datenpakete nie wieder herauskämen, das Netz jedoch im Laufe der Zeit wegen Überlastung blockiert wäre. Um solchen Fallen vorzubeugen, enthält jedes IP-Paket den *TTL* (*Time To Live)*-Wert, der in einem 1-Byte großen Feld in den IP-Headern eingebaut ist. Die Protokoll-Software des Absenders trägt TTL-Werte von z. B. 128 oder 255 ein. Jeder durchlaufener Router dekrementiert diesen Wert um eins. Ist der TTL-Wert auf 0 abgesunken, vernichtet der betreffende Router das Datenpaket.

Sowohl LANs oder auch private Einzelrechner werden über Router an das Internet gekoppelt.

Layer-3-Switch:
Ein Layer-3-Switch ist ein „Multi-Port-Router“. Er vermittelt auf der Schicht 3 zu verschiedenen IP-Netzen und vermeidet so das Durchlaufen mehrerer hintereinander liegender Router, was zu längere Latenzzeiten führen würde. Ihr Einsatz ist immer dort sinnvoll, wo mehrere Netze örtlich dicht beieinander betrieben werden. Dies ist z. B. der Fall, wenn ein Unternehmensnetz durch Subnetzbildung in kleinere Netze aufgeteilt wird, um den Datenverkehr, insbesondere auch den Broadcast-Verkehr zu entkoppeln. Subnetze behandeln wir im nächsten Kapitel.

16.3 Verbindungsorientierte oder verbindungslose Kommunikation

Die Transportschichten legen auch die Art einer Verbindung fest. Grundsätzlich kann eine Kommunikation entweder *verbindungsorientiert* oder *verbindungslos* sein, abhängig davon, ob IP mit dem Schicht-4-Protokoll TCP oder UDP zusammenarbeitet.

Verbindungsarten der TCP/IP-Protokoll-Suite:

TCP : verbindungsorientiert

UDP: verbindungslos

TCP-Verbindungen:
TCP arbeitet verbindungsorientiert. Vor dem Austausch der Anwenderdaten wird in einem *Handshake*-Verfahren (Verbindungsanforderung/Verbindungsbestätigung) eine logische Verbindung (virtuelle Verbindung) zwischen den Partnern aufgebaut, ähnlich dem Verfahren, wie wir es beim Telefonieren gewöhnt sind. Ist der Partner, genauer, der angesprochene Prozess auf dem Partnerrechner, nicht verfügbar, kommt keine Kommunikation zustande. Jedes beim Empfänger eingetroffene Datenpaket wird bestätigt. Die Verbindung arbeitet mit einer Flusssteuerung: Bleiben mehr als eine vereinbarte Anzahl von Datenpaketen unbestätigt, unterbricht der Absender solange seine Sendungen, bis die ausstehenden Bestätigungen eingegangen sind. Die Durchnummerierung der übertragenen Datenpakete ermöglicht eine Sortierung der beim Empfänger eingetroffenen Sendungen, falls die Reihenfolge durch unterschiedliche Übertragungswege durcheinander geraten ist. Verlorengegangene Pakete werden vom Empfänger erneut angefordert und erneut übertragen. Am Ende der Kommunikation muss die Verbindung wieder explizit abgebaut werden.

Als Schnittstelle zu den anwendungsorientierten Protokollen stellt TCP eine gesicherte Prozess-zu-Prozess Verbindung her. Es ist das am Häufigsten eingesetzte Transportprotokoll. Wegen seiner Zuverlässigkeit wird es besonders im Internet viel angewendet. (Mails sind also nicht einfach „verloren“, wenn der Empfänger oder Mailserver nicht erreichbar ist. Der Absender erfährt davon!)

Beispiele für Anwendungen, die TCP nutzen: SMTP(Mail), FTP, POP3, HTTP

UDP-Verbindungen:

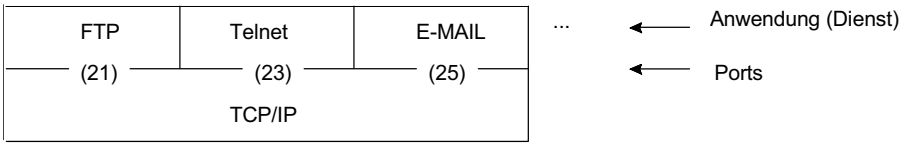
Der *Overhead* eines expliziten Verbindungsaufbaus wird beim UDP-Protokoll vermieden: UDP arbeitet verbindungslos. Bei der verbindungslosen Methode sendet ein Knoten seine Nachricht als „Datagramm“ ohne weitere Vorbereitung spontan an den Empfänger ab. Jedes Datagramm enthält vollständige Angaben über Ziel- und Quelladresse. Der Sender erhält keine Bestätigung vom Empfänger über Erfolg oder Misserfolg seiner Sendung. Der Absender kann lediglich am Ausbleiben einer erwarteten Reaktion darauf schließen, dass der Empfänger die Nachricht nicht verarbeitet hat. Nach mehreren vergeblichen Sendeversuchen wird die Kommunikation mit einem „Time out“-Fehler abgebrochen. Die einfache verbindungslose Kommunikation ist zwar schneller, aber weniger zuverlässig als eine verbindungsorientierte Kommunikation. UDP wird daher für weniger kritische Anwendungen eingesetzt, ebenso auch im LAN-Bereich, der technisch überschaubarer und sicherer ist als das Internet.

Beispiele für Anwendungen, die UDP nutzen: DNS, NFS, LAN-Multimedia-Anwendungen.

16.4 Ports und Sockets

Nachdem die Transportdienste eine Rechner-Rechner-Verbindung bereitgestellt haben, muss auf dem Zielrechner das eingetroffene Datenpaket an den richtigen Anwendungsprozess, d. h. an den gewünschten Netzdienst übergeben werden. Wie wird die „richtige“ Anwendung gefunden, da Server i. a. mehrere Netzdienste anbieten oder auch unterschiedliche Clienten den gleichen Serverdienst anfordern können? Die Eindeutigkeit bei der Auswahl des Prozesses geschieht über das *Port*konzept. Ports sind numerische 16-Bit-Anwendungsadressen, die im Header der Transportprotokolle neben der Netzknotenadresse abgelegt sind und nicht mit den ebenfalls als Ports bezeichneten Anschlüssen eines Hardware-Interfaces zu verwechseln sind. Die Ports im Bereich 1-1024 sind vordefiniert und standardisiert (*well known ports).* Jeder neuen Entwicklung einer Anwendung muss auch ein neuer Port zugeordnet werden, über den sie angesprochen werden soll.

Beispiel für Portnummern:



***Well-Known-Ports* (Auszug)**	
Port-nummer:	*Anwendung:*
20	File Trandfer Protocol (FTP-data)
21	File Transfer Protocol (FTP-control)
23	Telnet
25	E-Mail: Simple Mail Transfer Protocol (SMTP)
53	Domain Name Service (DNS)
80	World Wide Web (WWW-http)
110	Post Office Protocol V3 (POP3)
135	DCE Endpoint Resolution (epmap)
137	NetBIOS Name Service (netbios-ns)
138	NetBIOS Datagramm Service (netbios-dgm)
139	NetBIOS Session Service (netbios-ssn)
143	EMAIL: Internet Message Access Protocol (IMAP)
443	HTTP over TLS/SSL (https)
445	Microsoft Directory Service (ms-ds)

Damit der angesprochene Dienst auch wieder dem „richtigen“ Prozess des Absenders antworten kann, enthält das Datenpaket auch die vollständige Absenderinformation, d. h. Netzknotenadresse und Portnummer des Absenders. Netzknotenadresse und Portangabe zusammengenommen werden als *Socket* bezeichnet. Jede Verbindung auf Anwenderebene ist eindeutig durch sein Socketpaar definiert. In einem Netzwerk gibt es nicht zwei gleiche Socketverbindungen. (Das gilt auch für das weltweite Internet!)

Sockets:
Netzknotenadresse + Portnummer, z. B. 192.168.17.9:23
Ein Socketpaar bildet eine Prozess-zu- Prozess-Verbindung

Die oben gezeigten „well known ports“ gelten für TCP- oder UDP-Anwendungen auf der Serverseite; es sind also Zielports, auf denen ein Server „lauscht“, d. h. empfangsbereit ist. Der Quellport muss nicht „well known“ sein, er kann vom Client frei gewählt werden und liegt meistens im Bereich > 6000. Der Server antwortet diesem Port, den er im ersten Paket im

Quellsocket erkennt. Weiterhin teilt der Server nun dem Client mit, dass die weitere Kommunikation über einen anderen, von ihm frei gewählten Port stattfinden soll, damit die „Lauschports“ wieder frei sind für weitere Anfragen. Die „well known ports“ gelten also nur für erste Kontaktaufnahmen. Das Portkonzept spielt eine wichtige Rolle bei Firewalls (→ s. Kap. 18.4.5).

16.5 Client-Server-Kommunikation und Broadcasts

Eine Kommunikation im Netz kann entweder als Dialog zwischen zwei festen Stationen (*point-to-point*) oder als „Rundsendungen“ an alle (*broadcast)* ablaufen.

Bei der Kommunikation nach dem Client/Server-Modell fordert ein Client gezielt von einem Server einen bestimmten Dienst an. Der Server reagiert, indem er dem Clienten den angeforderten Dienst zur Verfügung stellt. Diese Form der Kommunikation ist client-initiiert.

Typische Beispiele: Ein Client fordert eine Datei an vom Fileserver;
Ein Mail-Client fragt auf einem Mail-Server nach, ob Nachrichten für ihn eingetroffen sind.

Andererseits kann eine Kommunikation auch durch periodische, allgemeine Bekanntgaben oder Anfragen in das Netz hinein entstehen, die als Broadcasts mehrere oder alle Knoten des Netzes erreichen sollen.
Typische Beispiele:

Ein Host fragt:	„Wer kennt die MAC-Adresse zu der IP-Adresse 194.78.45.12 ?“
Ein NetWare-Server gibt bekannt:	„Mein Name ist ...; ich bin ein Fileserver“;
ein NetBIOS-Knoten gibt bekannt:	„Mein NetBIOS-Name ist ...“;
ein Router gibt bekannt:	„Ich kenne folgende Netze, die über mich erreichbar sind:... ... ...“ (Router-Protokolle).

Andere Netzknoten speichern diese Bekanntgaben in speziellen *Cache*-Speichern und greifen bei Bedarf darauf zurück, ohne erst das Netz abfragen zu müssen. NetWare-Server geben z. B. ihre Dienste auf diese Art bekannt. Derartige Broadcasts werden in zeitlichen Abständen von oft nur wenigen Sekunden periodisch gesendet und können damit einen erheblichen Anteil der Netzkommunikation in größeren Netzen ausmachen! Für den Nutzer bleiben diese Aktivitäten unsichtbar, im ungünstigsten Fall verspürt er lediglich lange Antwortzeiten bei Broadcast-überlasteten Netzen.

16.6 Kommandos zum TCP/IP-Protokoll

arp (*Address Resolution Protocoll*):
Das Konsolenkommando

>arp –a (Windows und Unix)

greift auf den arp-Cache der Station zu und zeigt die aktuelle Zuordnungstabelle IP-Adresse ↔ Ethernetadresse an. Die Tabelle ist unmittelbar nach dem Einschalten des Rechners noch leer. Sie wird im Laufe der Zeit, genauer: im Laufe stattgefundener Kommunikation dynamisch aufgebaut. Findet einige Minuten lang kein Datenaustausch mit einer bestimmten Station statt, wird der betreffende Eintrag wieder gelöscht. Man kann sich also die in der Regel unbekannten Ethernetadressen aller am Netzsegment angeschlossenen Stationen „besorgen“, wenn man zu jeder einen Befehl *>ping <IP-Adresse>* absetzt und anschließend seinen arp-Cache ausgeben lässt.

netstat:
Die Liste der aktiven Netzverbindungen mit den aktuell benutzten Portnummern kann man sich mit dem Konsolenkommando

> netstat (Windows und Unix)

anzeigen lassen. Mit

> netstat –a

werden alle empfangsbereiten Prozesse mit ihren Portnummern einer Station ausgegeben (*listening ports)*. Hacker nutzen diese Informationen, indem sie sog. Portscanner einsetzen um sich über ansprechbare Ports illegal Zugang zu Rechnern zu verschaffen.

>netstat –s

zeigt ausführliche statistische Angaben über die Transportprotokolle und kann bei Netzproblemen hilfreich sein.

16.7 Aufgaben

1) In einem TCP/IP-Netzwerk seien Hubs, Repeater, Switches und Router eingesetzt. Welche dieser Netzkoppler lassen sich über eine IP-Adresse ansprechen?
2) Wie verhält sich ein Ethernet-Switch beim Eintreffen eines Datenpakets mit der MAC-Zieladresse FF-FF-FF-FF-FF-FF?
3) Welche Bedeutung haben die Begriffe *Port* und *Socket*?
4) Wie beschaffen Sie sich die Ethernetadresse Ihres Internet-Routers ohne einen ping-Befehl direkt an den Router abzusetzen?
5) Gibt es Beschränkungen, wie viele Router maximal durchlaufen werden können?
6) Ordnen Sie die nachfolgenden Begriffe einander zu: Hub, Switch, Router, Kollisions-Domäne, Broadcast-Domäne.

17 Netzwerkbetriebssysteme

Die von einem Netzwerkbetriebssystem angebotenen Netzwerk-Anwendungen (Netzwerkdienste) lassen sich in der Praxis in zwei Gruppen einteilen:

- Basisdienste für (überwiegend) LAN-interne Kommunikation
  z. B. zentrale Benutzerverwaltung, File- und Print-Services;
- (überwiegend) LAN-übergreifende Kommunikation: Internet-Dienste
  z. B. E-Mail, Telnet, File-Transfer (FTP), World Wide Web (WWW), ...

Die Konzeption für die Basisdienste hängt stark ab vom jeweiligen Betriebssystem. Einige typische Eigenschaften sollen hier für die drei am Häufigsten anzutreffenden Systeme Windows, NetWare und Unix (Linux) aufgezeigt werden. Die Internet-Dienste dagegen sind praktisch Betriebssystem-unabhängig. Sie werden im Kapitel über das Internet vorgestellt.

17.1 Spezielle Netzwerkeigenschaften von Windows

17.1.1 Zentrale Protokolle

Die Weiterentwicklung der Microsoft-Betriebssysteme von DOS, Windows 3.1, Windows NT, ... Windows 2000, Windows XP ... brachten jeweils auch Veränderungen in den Netzwerkkonzepten. DOS selbst besaß keinerlei Netzwerkfunktionalität. Diese musste erst durch Betriebssystem-externe Client-Software, z. B. NetWare-Clients auf der Basis des IPX/SPX Transportprotokolls, nachinstalliert werden. Mit Windows 3.1 wurde Windows netzwerkfähig durch die Einführung des NetBEUI/NetBIOS-Protokolls, das seitdem in allen Windows-Versionen enthalten ist, aber wegen seiner nicht-Routbarkeit zunehmend an Bedeutung verliert.

Das NetBIOS Protokoll spielt in lokalen Windows-Netzen eine dominierende Rolle. Je nach eingesetztem Transportsystem tritt es in verschiedenen Versionen an unterschiedlichen Positionen im Schichtenmodell auf:

Protokoll:	*Installationsname:*	*Einsatz:*
NetBEUI/NetBIOS	NBF (*NetBIOS Frame*)	kleine LANs
NetBIOS über TCP/IP	NBT	Standard
NetBIOS über IPX	NWLink	

Ferner wird unterstützt:

TCP/IP-Protokoll-Suite für die „Internetprotokolle“(HTTP, FTP,...)

IPX/SPX-Protokollfamilie für Kommunikation mit NetWare-Servern.

Alle diese Protokolle können parallel aktiviert und betrieben werden.

Die zentrale Aufgabe des NetBIOS-Protokolls besteht in der Verwaltung des „NetBIOS-Namenraums“, der zur Adressierung der Rechner und Dienste und zur Kennzeichnung von Gruppen genutzt wird. NetBIOS-Namen dürfen bis zu 15 Zeichen lang sein und enthalten nur alphanumerische Zeichen. Der Namensraum ist „flach“, eine Hierarchie ist nicht vorgesehen. Wenn innerhalb eines Windows-LANs ein Rechner angesprochen wird, geschieht dies in der Regel über seinen NetBIOS-Namen. Die Eindeutigkeit der Namen wird über einen Broadcast-

Mechanismus sichergestellt: Ein neu im Netz eingeschalteter Rechner gibt in mehreren Broadcasts seinen Namen im Netz bekannt. Falls der neu eingeführte Name bereits vergeben wurde, antwortet der entsprechende Rechner mit einer „Name-bereits-in-Nutzung!"-Nachricht. Damit ist der Name abgelehnt und es muss ein anderer Name gewählt werden. Gibt es nach mehreren Broadcasten keine Proteste, gilt der neue Name als angenommen (registriert) und alle Netzrechner speichern ihn in ihrem lokalen NetBIOS-Namen-Caches.

Die gespeicherten Namenstabellen kann man sich mit dem Konsolenkommando *nbtstat* anzeigen lassen, z. B.

>nbtstat –n	: Ausgabe der lokalen Namenstabelle
>nbtstat –A <ip-Adresse>	: Ausgabe der Namenstabelle eines anderen Rechners (funktioniert auch über das Internet!)

NetBIOS-Broadcasts in Windows-Netzen können einen erheblichen Anteil der Netzwerkslast ausmachen. NetBIOS gilt daher als ein sehr „geschwätziges" Protokoll!

Die Namensauflösung bei Windows-Rechnern kann durch Anlegen einer statischen Datei LMHOSTS (%systemroot%\system32\drivers\etc\lmhost, wobei *%systemroot%* das Rootverzeichnus der Windows-Installation ist) auf den einzelnen Arbeitsstationen unterstützt werden. Diese Datei enthält eine Tabelle der NetBIOS-Namen und der zugehörigen IP-Adressen.

Broadcasts breiten sich nur innerhalb eines Netzwerksegmentes aus. Soll ein gemeinsamer NetBIOS-Namensraum auch in Netzsegmenten gelten, die durch Router gekoppelt sind, ist die Namensverbreitung per Broadcast nicht möglich. Für derart große Netze muss eine zentrale Instanz die Namensverwaltung für alle Segmente übernehmen, ähnlich dem Domain Name Service des Internet. Diese Instanz ist für Windows-Netze ein WINS-Server. Ein WINS-Server führt die NetBIOS-Namen und die zugehörigen IP-Adressen der Windows-Rechner aller Netzsegmente der Institution. Jede Workstation derartiger Netze enthält in ihrer Netzwerk-Konfiguration die IP-Adresse des WINS-Servers und befragt diesen, wenn die lokale Namenstabelle oder die LMHOSTS-Datei einen Namen nicht auflösen können.

NetBIOS-Namen sind unabhängig von dem im Internet vergebenen Domainnamen, so dass ein Rechner je nach benutzten Protokollen mit unterschiedlichen Namen angesprochen werden kann.

NetBIOS arbeitet mit dem darauf aufsetzenden höheren Anwenderprotokoll SMB (*Server Message Block*) eng zusammen, das Datei- und Druckerdienste bereitstellt. Spezielle Anwendungen des SMB-Protokolls sind die für Windows-Netze typischen „Browser-Dienste". Der Browsing-Dienst hat nichts zu tun mit Web-Browsing oder dem http-Protokoll. Er ist unabhängig vom jeweiligen Transportsystem, um SMB zu befördern und kann auf NetBEUI/NetBIOS, NetBIOS über TCP/IP oder NetBIOS über IPX aufsetzen. Die Aufgabe des Browser-Dienstes ist es, NetBIOS-Namen für Server-Systeme und deren Dienste dynamisch zu registrieren und diese dynamische Liste den Rechnern des Netzwerks zur Verfügung zu stellen. Typische Anwendungen des Browser-Dienstes sind die Ausgaben der „Netzwerkumgebung".

Der Browser-Dienst ist eine Client/Server-Anwendung. Jede Workstation, die Dateien zum Netzwerkzugriff freigibt, kann sowohl als Server als auch als Client arbeiten. Bei (kleinen) Peer-to-peer-Netzwerken handeln die potentiellen Server unter sich aus, wer „Master-Browser" wird und auf Client-Anfragen der anderen Rechner antwortet. In Domänen-organisierten Netzen übernimmt der Domänenserver auch die Funktion des Master-Browsers.

17.1.2 Organisation von Windows-Netzen

Arbeitsgruppen:

Kleinere Windows-Netze werden meistens als Peer-To-Peer-Netz in Form einer „Arbeitsgruppe“ angelegt, bei dem sich die Arbeitsstationen gegenseitige Zugriffsmöglichkeiten einräumen. Auf allen Rechnern ist eine Workstation-Version des Betriebssystems installiert. Alle Stationen sind Mitglied der gleichen Arbeitsgruppe, d. h. müssen bei der Netzwerkkonfiguration den gleichen Arbeitsgruppen-Namen enthalten.

Windows-Domänen:

Mittleren und größeren Windows-Netze liegt eine Client/Server-Architektur zugrunde; statt einer Arbeitsgruppe wird eine Domäne eingerichtet. Das Server-Betriebssystem (Windows NT Server, Windows 2000 Server, Windows XP Server, ...) unterscheidet sich nicht grundlegend von den Workstation-Versionen, sondern lediglich durch einige zusätzliche Dienste. Der Server kann auch *non dedicated* als Arbeitsstation dienen, senkt dadurch aber u. U. die Leistungsfähigkeit (*Performance)* der Serverfunktionen.

Die Verwaltung von Windows-Netzwerken beruht auf dem Domänen-Konzept. Kleinere Netzwerke bestehen aus *einer* Domäne, größere können auch mehrere, voneinander unabhängige Domänen enthalten. Jede Domäne verwaltet ihre Benutzer und Netzwerkressourcen eigenständig. Eine Kommunikation zwischen unterschiedlichen Domänen ist erst nach Einrichten einer Vertrauensstellung unter den Domänen möglich (*Trusted Domain).* Zu einer Domäne gehört der Domänen-Server (PDC, *Primary Domain Controller),* evtl. Sicherungs-Domänen-Server (*BDC, Backup Domain Controller)* und die der Domäne zugeordneten Arbeitsstationen. Der Domänen-Server ist für die Sicherheit einer Domäne zuständig. Auf ihm sind die Benutzerkonten aller der Domäne angehörigen Benutzer zusammen mit den Zugriffsrechten für die Netzwerkobjekte (Dateien, Verzeichnisse, Drucker) zentral gespeichert. Eine Domäne kann neben dem PDC weitere „normale“ Server enthalten, deren Netzressourcen über den Domänen-Server verwaltet werden.

Grundsätzlich gibt es zwei Möglichkeiten eines Logins an den Arbeitsstationen eines Windows-Netzes:

– lokale Anmeldung: Der Benutzer muss einen lokalen Account besitzen. (beim Login ist bei „Domäne“ der lokale Rechnername anzugeben):
  Zugriff nur auf das lokale System;
  Die *lokale* Benutzer-Datenbank muss vom Administrator der *lokalen* Station gepflegt werden;

– Anmeldung an der Domäne: Der Benutzer muss einen Account auf dem PDC besitzen. (beim Login ist bei „Domäne“ der Domänenname anzugeben):
  Zugriff auf das lokale System;
  Zugriff auf freigegebene Netzressourcen der Domäne;
  Die zentrale Benutzer-Datenbank wird vom Administrator der Domäne gepflegt;

Eine zentrale Administrierung der Benutzerkonten einer Domäne reduziert den Verwaltungsaufwand. Sie ist besonders für Benutzer wichtig, die sich von unterschiedlichen Arbeitsstationen am Netz anmelden und vom Domänen-Server jeweils ihre individuelle Arbeitsumgebung zugewiesen erhalten („wandernde User-Profile“). Das lokale System unterliegt nicht der Sicherheit des Domänen-Servers.

Der Administrator der Domäne pflegt die Benutzer-Datenbank und regelt die Zugriffsrechte auf die Netzwerkressourcen. Die Organisation der Benutzer in Gruppen, denen pauschal Rechte

eingeräumt werden können, senkt den Verwaltungsaufwand. Für jeden Benutzer legt der Domänen-Administrator fest:

- Angaben zum Login (Name, Erstpasswort)
- Gruppenzugehörigkeit
- Pfad für Benutzerprofil
- Pfad für (evtl. vorhandenes) Anmeldeskript
- Basisverzeichnis

Das Server-Betriebssystem legt für jeden Workstation-Benutzer ein persönliches Benutzerprofil lokal an. Das Benutzerprofil besteht aus einer einige MByte umfassenden Verzeichnisstruktur, die die benutzerspezifischen Einstellungen der Arbeitsumgebung (Bildschirmfarben, Desktop-Anordnungen, Laufwerkszuordnungen usw.) enthält. Ein Server-basierendes Benutzerprofil wird während des Login-Prozesses auf die lokale Arbeitsstation übertragen und am Sitzungsende wieder auf dem Server aktualisiert, sofern Profiländerungen stattgefunden haben.

Vorhandene Anmeldeskripte (Typ .BAT, .CMD oder .EXE) werden bei jeder Anmeldung ausgeführt und gestatten den automatischen Start von Anwendungen und Prozessen. Das Basisverzeichnis ist das persönliche „Heimatverzeichnis“ (Home-Directory) des Benutzers. Es wird als Voreinstellung stets dann benutzt, wenn keine expliziten Pfadangaben gemacht werden. Eine Server-basierende Speicherung aller Nutzerdaten ist die Voraussetzung für ein Workstation-ungebundenes Arbeiten und erleichtert ferner eine Datensicherung, die zentral für alle Benutzer am Domänen-Server durchgeführt werden kann.

Eine Domäne stellt Netzwerkressourcen in Form von „Freigaben“ (*Shares)* zur Verfügung. Der Domänen-Administrator bestimmt beim Einrichten von Freigaben, welche Benutzer oder Benutzergruppen in welcher Art und Weise darauf zugreifen dürfen, d. h. jede Freigabe führt eine Liste, in der die Zugriffsberechtigten und die Zugriffsarten festgelegt sind. Typische Freigaben sind Verzeichnisse, Dateien und Drucker. Die Zugriffsrechte wie z. B. Lese- oder Schreibrechte an einer Datei, lassen sich über die „Eigenschaften“ des freigegebenen Objektes differenziert vergeben, auch vom Benutzer, wenn er „Eigentümer“ des betreffenden Objektes ist.

Netzwerkbenutzer können Freigaben über den vom Domänen-Administrator vergebenen Freigabenamen in UNC (*Universal Naming Convention*)-Notation ansprechen:

\\servername\freigabename

Durch das Anklicken der „Netzwerkumgebung“ auf der Arbeitsstation wird der *Browser*-Dienst gestartet, der die Netzfreigaben auf den Servern der Domäne anzeigt. Ein Benutzer kann freigegebenen Verzeichnissen Laufwerksbuchstaben zuordnen und diese nutzen wie lokale Festplatten. Solche Laufwerkszuordnungen können in den jeweiligen Benutzerprofilen gespeichert werden.

Das Server-Betriebssystem gestattet weiterhin noch die Vergabe eines festen Satzes von vordefinierten Systemrechten, die an vordefinierte oder selbstdefinierte Benutzergruppen und individuelle Benutzer vergeben werden können. Beispiele für Systemrechte sind die Berechtigung zur Änderung der Systemzeit oder die Berechtigung zum Herunterfahren des Systems.

Windows bietet mit seinem „NET“-Befehlen eine gute Möglichkeit, Informationen über das Netzwerk abzufragen bzw. Setzungen vorzunehmen. NET-Befehle sind textorientiert und müssen im Konsolmodus „Eingabeaufforderung“ abgesetzt werden. Dies bietet jedoch den besonderen Vorteil, dass sie in Batch-Programmen oder Anmeldeskripten eingefügt werden können. Die nachfolgende Liste enthält einige wichtige NET-Kommandos, die für den Workstation-Nutzer besonders interessant sind:

Auswahl einiger NET-Befehle	
NET ACCOUNTS	Angaben zum Benutzerkonto
NET CONFIG	Angaben zur Konfiguration, u. a. Ausgabe der Ethernet-Adresse
NET NAME	Anzeige des aktuellen Rechnernamens
NET PRINT \\Rechner\Freigabename	Anzeige von Druckaufträgen, Druckerwarteschlangen und Status
NET SEND Nachricht	Versenden von Nachrichten im LAN
NET SHARE Freigabename	Anzeige der Freigaben, die das lokale System zur Verfügung stellt
NET START	Liste aller aktiver Dienste
NET STATISTICS WORKSTATION	Statistiken zu den aktiven Diensten
NET USE	Liste der Netzwerkverbindungen und deren Status
NET USER	Liste der auf dem Rechner eingerichteten Benutzerkonten
NET VIEW	Liste der aktiven Rechner der Domäne
NET VIEW \\Rechner	Liste der Freigaben auf „Rechner“

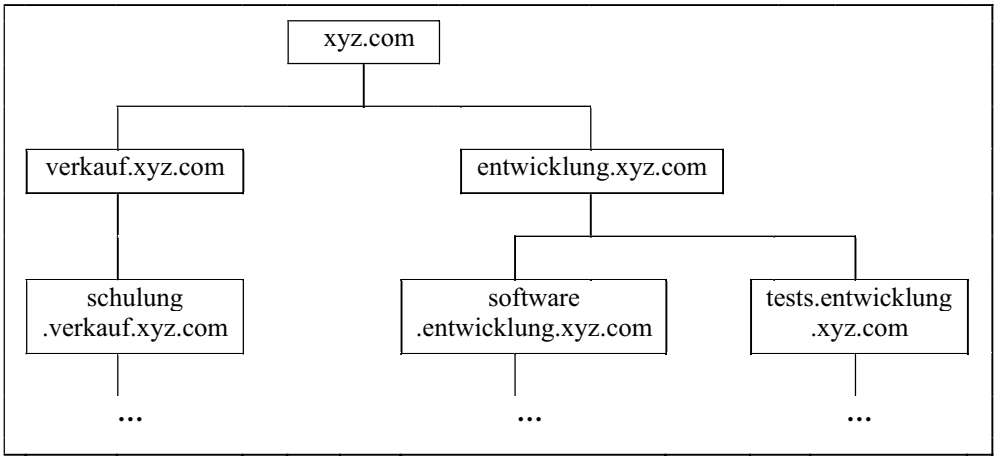
Mit NET HELP kann man sich alle NET-Befehle mit ihren vielfältigen Optionen anzeigen lassen.

Der Active Directory Verzeichnisdienst:

Die neueren Server-Betriebssysteme (ab Windows 2000) bieten mit der Einführung des „Verzeichnisdienstes“ (*Active Directory, AD*) eine Möglichkeit, eine Mehr-Domänen-struktur eines gesamten Unternehmens in eine einzige logische Struktur in Form eines Verzeichnisbaums abzubilden. Microsoft folgte damit der seit Jahren erfolgreich eingeführten Netzwerkorganisation von Novell-Netzen durch die *Novell-Directory-Services* (NDS). Durch die Schaffung einer einzigen Netzstruktur werden die isoliert nebeneinander existierenden Domänen größerer Unternehmen in *eine* hierarchische Domänenstruktur integriert. Damit entfallen die lästigen Einrichtungen von Vertrauensstellungen zwischen den Domänen.

Ähnlich, wie auf einer 100 GByte Festplatte eine Verzeichnisstruktur Ordnung und Übersichtlichkeit schafft, ermöglicht der Verzeichnisdienst *Active Directory* eine logische hierarchische Anordnung aller Netzobjekte. Typische Netzobjekte sind Benutzer, Gruppen, Drucker, Rechner, Server und Domänen.

Auf oberster Ebene bilden die Domänen eines Unternehmens eine Baumstruktur (*Domain Tree*):

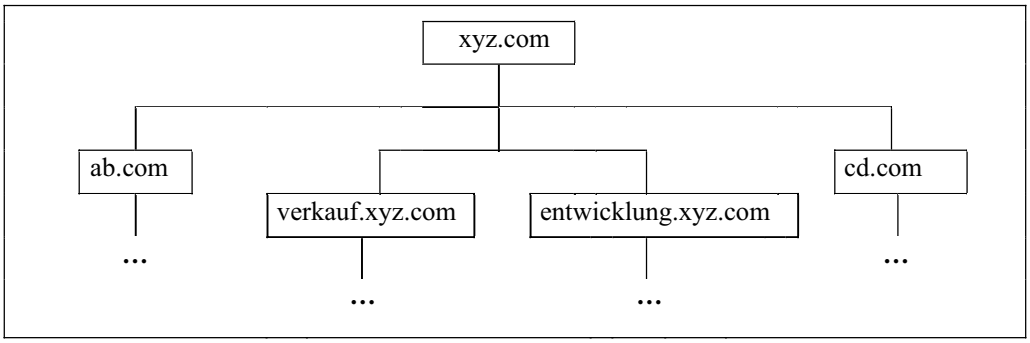


Beispiel: Domänen-Baum der Firma xyz

An der Spitze der Hierarchie befindet sich die Stammdomäne (*root domain*); sie ist als die zuerst angelegte Domäne der betreffenden Institution den anderen Domänen (*child domains*) übergeordnet.

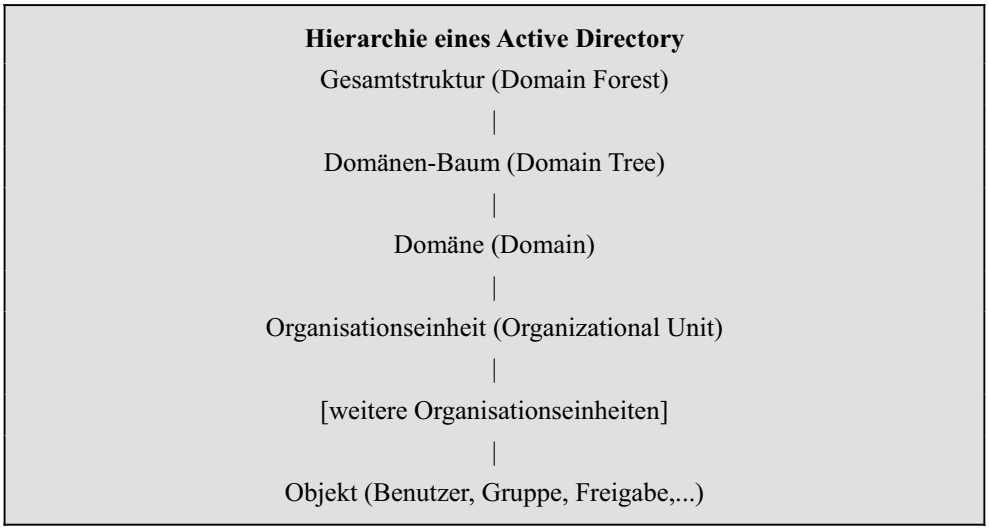
Auch innerhalb der einzelnen Domänen entsteht durch Anlegen von Organisations-Einheiten (*OrganizationalUnits,OU*) als C*ontainer* Objekt eine hierarchische Substruktur. OU-Objekte können andere OU-Objekte oder „End"-Objekte wie Benutzer, Gruppen, Drucker oder Freigaben enthalten.

Mehrere Domänen-Bäume lassen sich zu einer Gesamtstruktur (*domain forest*) zusammenfassen, die Stammdomänen der einzelnen Domänen-Bäume werden dabei miteinander verbunden:



Beispiel: Gesamtstruktur der Firma xyz nach Eingliederung der Domänen-Bäume ab.com und cd.com.

Eine derartige Gesamtstruktur entsteht z. B. nach einer Übernahme der beiden Firmen ab.com und cd.com, die ihre Netzwerke in jeweils eigenständigen Domain Trees verwaltet hatten. xyz.com wird zur Stammdomäne der Gesamtstruktur. Die Namensräume der eingegliederten Domänen-Bäume bleiben jedoch erhalten.



In Windows-Netzen werden Netzobjekte üblicherweise mit ihrem NetBIOS-Namen angesprochen. NetBIOS nutzt einen flachen Namensraum. Mit der Einführung der Verzeichnisdienste ist auch ein neues Namensvergabesystem verbunden: Der Namensraum des Verzeichnisdienstes zeigt eine Hierarchie wie die des Internet-Namensraumes und kann mit diesem identisch gewählt werden. Damit entfällt die übliche doppelte Namensvergabe (NetBIOS-Name, DNS-Name) für mit dem Internet verbundene und den DNS-Dienst nutzende Windows-Netze. Die Namen des Verzeichnisdienstes stützen sich auf den DNS-Namensdienst ab. Damit kann erstmals (ab Windows 2000 Server) ganz auf das „geschwätzige“ und Broadcast-intensive NetBIOS-Protokoll und WINS verzichtet werden.

17.2 Spezielle Netzwerkeigenschaften von NetWare

Novell hat mit seinem Netzwerkbetriebssystem NetWare das erste Produkt zur Vernetzung von PCs auf dem Markt eingeführt. Es ist auch heute noch in viele Unternehmen und Instituten eingesetzt. NetWare ist ein reines Client/Server-Betriebssystem. Jede Kommunikation im LAN ist auf den/die NetWare-Server ausgerichtet, Peer-to-Peer-Verbindungen sind nicht möglich. Die Workstations eines NetWare-Netzes werden meistens mit dem lokalen Betriebssystem Windows (evtl. DOS) betrieben, NetWare unterstützt jedoch auch die Betriebssysteme Unix, Apple Macintosh und OS/2.

Die Kommunikation in einem NetWare-Netzwerk basiert primär auf den IPX/SPX oder IPX/NETBIOS-Protokollen, auf die das Schicht-7-Protokoll *NetWare Core Protocol* (NCP) aufsetzt. Daneben ist zunehmend auch der Datentransport über TCP/IP alternativ oder zusätzlich möglich.

Netzwerk-Anbindung der Clients:

Werden die Arbeitsstationen mit dem Betriebssystem Windows betrieben, genügt es, den Microsoft *Client Service für NetWare* bei der Einrichtung des Netzwerkes zu aktivieren. Allerdings sollte man eher die bei NetWare mitgelieferte Client-Software installieren, da deren Funktionalität umfassender ist. Die NetWare-Clients fügen sich vollkommen ein in die Netzwerkkonfigurationen der Windows-Betriebssysteme. Für die Netzanbindung von DOS- und Windows 3.1x-Arbeitsstationen liefert Novell spezielle Client-Software mit aus.

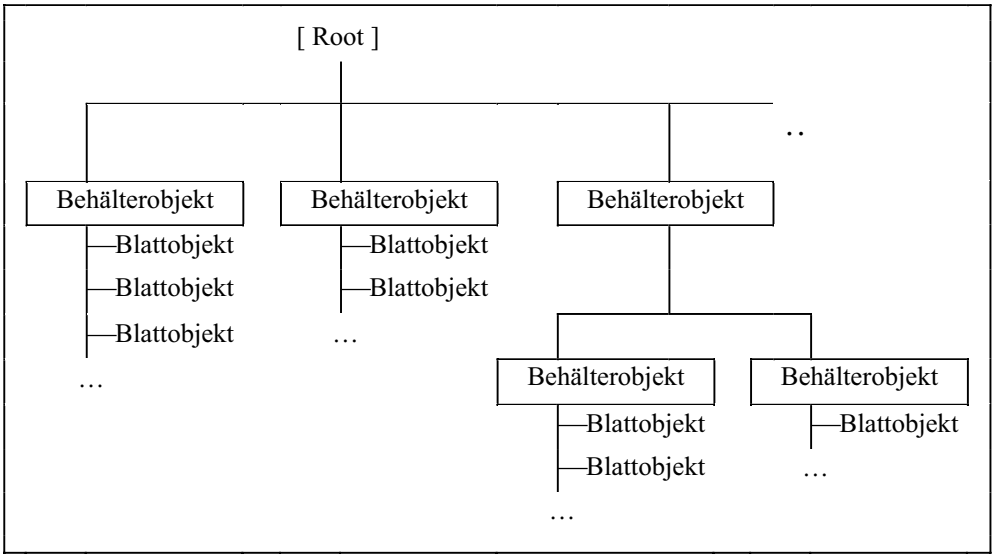
17.2.1 Der NDS-Verzeichnisdienst

Den Kern des NetWare-Betriebssystems bilden die NDS-Verzeichnisdienste (*Novell Directory Services*), die in einer Datenbank alle Objekte eines NetWare-Netzes enthalten. Objekte sind die Benutzer und die Ressourcen des Netzwerks, wie z. B. Drucker, Warteschlangen, Laufwerke, Verzeichnisse. Im Gegensatz zur Server-bezogenen Verwaltung in der *Bindery* (lokale Systemdatenbank eines 3.x-NetWare-Servers) oder der Domänen-bezogenen Verwaltung von Microsoft-Netzen zielt die NDS ab auf eine unternehmensweite Verwaltung *aller* Netzressourcen („*Enterprise Networking“)* in *einer* streng hierarchischen Baumstruktur. Der Benutzer muss sich nur einmal „am Netzwerk“ (nicht an einem Server!) anmelden, um auf alle Netzressourcen zugreifen zu können, für die der Systemverwalter die Rechte für einen bestimmten Benutzer eingerichtet hat.

In großen NetWare-Netzen kann die NDS einen so großen Umfang annehmen, dass es übersichtlicher ist, sie auf mehreren Servern in Form von einzelnen *Partitionen* (Teilbäume der NDS) abzulegen. Kopien von Partitionen auf unterschiedlichen Servern (*Replikationen)* erhöhen die Sicherheit und beschleunigen den Zugriff auf die NDS-Daten. Für den Benutzer ist die Verteilung von Replikationen auf unterschiedliche Server vollkommen transparent.

Der NDS-Baum ist objektorientiert und gleicht in seiner streng hierarchischen Struktur dem Aufbau eines Dateisystems auf einem Datenträger. Er enthält zwei Typen von Objekten:

- Behälterobjekte (C*ontainer Objects)*
- Blattobjekte (*Leaf Objects).*



Ein NDS-Baum besteht aus Behälter- und Blattobjekten

Behälterobjekte können weitere Objekte enthalten und entsprechen einem Verzeichnis eines Dateisystems, Blattobjekte liegen an den Enden eines Zweiges und können keine weiteren Objekte enthalten. Blattobjekte sind z. B.: ein Benutzer, ein Rechner im Netzwerk, ein Drucker, eine Druckerwarteschlange. Jedes Blattobjekt wird im Verzeichnisbaum mit seinem logischen Namen angezeigt. Das Stammobjekt *[Root]* ist in jedem NDS-Baum fest vorgegeben.

Der „obere Teil“ eines NDS-Baums (die Wurzel ist ganz oben!) besteht aus einer fest vorgegebenen Hierarchie von Behälterobjekten:

	[Root]	Stammobjekt	
	|		
optional:	Country(C)	z. B.	C=DE, C=DK
	|		
mindestens 1x:	Organisation(O)	z. B.	O=FH-Wiesbaden
	|		
i. a. mehrfach:	Organisations-Einheit(OU)	z. B.	OU=Umwelttechnik, OU=E-Technik
	|		
optional:	Organisations-Einheit(OU)	z. B.	OU=Haus_1, OU=Haus_2
	|		
	<Blattobjekte>		

Für die Behälterobjekte *Country* dürfen nur die standardisierten Länderkennungen benutzt werden (s. Anhang). Diese Objekte werden praktisch nur in länderübergreifenden Unternehmensnetzen eingesetzt (*Enterprise Networks),* die meisten NetWare-Netze verzichten auf diesen Eintrag.

Die wichtigsten Klassen der Blattobjekte sind:

- User (Benutzer)
- Group (Benutzergruppe)
- NCP-Server (NetWare-Server)
- Computer (Arbeitsstation im Netzwerk)
- Volume (Datenträger)
- Directory Map (Zeiger auf ein Verzeichnis)
- Profile (Gruppen Login-Skript)
- Print Queue (Druckerwarteschlange)
- Print-Server (Druck-Server)
- Printer (Drucker)
- Alias (allgemeiner Verweis)

Über die Objektklasse *Volume* wird der direkte Zugang zum Dateisystem eines NetWare-Servers geschaffen.

Die Objekte der NDS werden durch die Angabe ihres *Context* bei NetWare-Befehlen und Utilities angesprochen. Der Context ist der „Pfad innerhalb der NDS-Struktur“ zu dem übergeordneten Objektbehälter, der das angesprochene Objekt enthält. Dabei enthalten Context-Angaben jedoch nie das angesprochene Objekt selber. Die Context-Schreibweise ist aus der ISO-Norm X.500 abgeleitet. Für den Benutzer *Schmidt* (Blattobjekt CN=Schmidt) in dem vorgestellten NDS-Baum ist der Context z. B.:

OU=Elektronik.OU=Entwicklung.O=Berlin.C=DE

oder kürzer:

Elektronik.Entwicklung.Berlin.DE

Benutzer eines NetWare-Netzes melden sich an der NDS mit ihrem Namen an. Der Benutzer ist als Objekt in der NDS aufgeführt. Beim Login-Prozess muss der Context des Benutzers, also die Position innerhalb des NDS-Baums, angegeben werden. Login-Namen, Erstpasswort und Context werden vom Netzwerkadministrator mitgeteilt. Context-Angaben lassen sich in der Client-Software der Arbeitsstation speichern.

Nach erfolgreichem Login wird der angegebene Context zum *aktuellen Context* für den Benutzer. Mit dem NetWare-Befehl *CX* kann man den aktuellen Context anzeigen lassen oder auch wechseln. Angaben über die Contexte anderer Objekte liefert die NetWare-Utility *NLIST.* In kleineren Netzwerken befinden sich häufig alle für einen Nutzer wichtigen Objekte im gleichen Context. In diesem Fall kann auf die Contextangabe verzichtet werden (relativer NDS-Pfad).

17.2.2 Die Arbeitsumgebung der Benutzer

Benutzer können vom Netzadministrator in Benutzergruppen organisiert werden, um Verwaltungsaufgaben zu vereinfachen. Ein Benutzer kann Mitglied (*Member of ...*) mehrerer Benutzergruppen sein. Die Rechte eines Benutzers ergeben sich aus den Rechtezuweisungen (*Trustee Assignments*) an ihn persönlich und den Rechten aller Gruppen, denen er angehört. Ein typisches persönliches Recht ist z. B. der Schreibzugriff auf das HOME-Verzeichnis des Benutzers auf dem Server. Hingegen sind die Zugriffsberechtigungen auf die verschiedenen Netzwerkdrucker typische Gruppenzuweisung. Benutzergruppen werden als Blattobjekte in der NDS aufgeführt.

Das Rechtesystem von NetWare-Netzen ist weitaus differenzierter ausgelegt als das von Windows oder Unix. Es erlaubt eine sehr feingestufte Zugriffssteuerung.

Um den Benutzern eine jeweils angepasste Arbeitsumgebung zu ermöglichen, werden bei jeder Anmeldung mehrere Login-Skripts durchlaufen (*Container-Login-Script, Profile-Login-Script, User-Login-Script*). Login-Skripte sind ein sehr mächtiges Mittel, um Einstellungen sowohl auf Benutzerebene oder auf übergeordneten Ebenen vorzunehmen.

Das zentrale Verwaltungs-Programm – nicht nur für den Systemadministrator – ist der *Netware Administrator*, auf das jeder NetWare-Benutzer standardmäßig Zugriff hat. Während der Microsoft *Windows Explorer* lediglich diejenigen Teile des NDS-Baums darstellt, die Dateisystem-bezogen sind, kann mit dem NetWare Administrator auf den gesamten NDS-Baum zugegriffen werden, Informationen abgefragt und Setzungen vorgenommen werden.

NetWare bietet ein umfassendes Hilfesystem an für alle Befehle und Utilities. Es ist über das Kommando *HELP* bzw. *HELP <befehl>* im Verzeichnis SYS:PUBLIC auf jedem NetWare-Server erreichbar.

17.3 Spezielle Netzwerkeigenschaften von Unix-Netzen

Unix-Netzwerke sind Peer-to-Peer Netzwerke. Im Gegensatz zu Windows oder NetWare-Netzen sind sie nicht an die PC-Hardware-Plattform gebunden: Die einzelnen Netzrechner können von sehr unterschiedlicher Leistungsfähigkeit sein, von PC-basierten *Linux*-Systemen bis zu Höchstleistungsrechnern. Unix ist ein Multiuser Time-Sharing Betriebssystem, d. h. mehrere Benutzer können gleichzeitig an einem System arbeiten. So ist es z. B. typisch für leistungsfähige Unix-Systeme, dass einige Benutzer lokal, andere über das Netzwerk an einem Rechner angemeldet sind. Wegen dieser Multiuserfähigkeit zählen die in den vorangegangenen Netzwerkbetriebssystemen vorgestellten Mechanismen der Benutzerverwaltung, Zugriffssteue-

rung und Datensicherheit bei Unix nicht zu den Netzwerkeigenschaften. Sie müssen auch bei *stand-alone-Systemen* (nicht vernetztes System) vorhanden sein. Die Ressourcen in Unix-Netzen ergeben sich aus den Serverdiensten, die der jeweilige lokale Unix-Administrator für die Benutzer der anderen Unix-Rechner eingerichtet hat.

Die Kommunikation in Unix-Netzen basiert auf der TCP/IP-Protokollfamilie, die in jeder Unix-Installation standardmäßig enthalten ist. Der Datentransport findet entweder über das verbindungsorientierte Transportprotokoll TCP oder über das verbindungslose Transportprotokoll UDP statt.

Jede Unix-Station erhält vom Netzwerkadministrator eine eindeutige IP-Adresse und einen symbolischen Namen, wahlweise wird über die Adresse bzw. den Namen ein Netzknoten angesprochen, z. B.:

IP-Adresse: 193.175.39.112
(Domain-)Name: ux1.mnd.fh-wiesbaden.de

Den Namen des eigenen Rechners wird mit dem Unix-Kommando

```
$ hostname
```

ausgegeben. ("$" sei der Unix-Prompt.) Eine Tabelle der im LAN befindlichen Hosts mit Namen und IP-Adresse sind in der Datei */etc/hosts* abgelegt, die man sich mit

```
$ cat  /etc/hosts
```
oder
```
$ more  /etc/hosts
```

anzeigen lassen kann. Dieser Tabelle kann man auch die lokale IP-Adresse entnehmen. (Anmerkung: Bei einer Internet-Anbindung übernimmt der Nameserver die Zuordnung von IP-Adressen <--> Domain-Namen, s. u.).

Die Kommunikationsdienste in einem Unix-Netz arbeiten nach dem Client/Server-Modell, wobei jeder Netzknoten sowohl als Server als auch als Client fungieren kann. Die im Hintergrund laufenden Serverprozesse heißen *Daemons.* Sie sind in der Prozessliste an dem abschließenden „....d“ in ihrem Namen erkennbar, z. B. telnetd, ftpd, nfsd, ...

Kennt man den Namen oder die IP-Adresse eines anderen Netzrechners, so lässt sich mit

```
$ telnet <hostname  -oder-  IP-Adresse>
```

eine Sitzung auf dem Remote System durchführen (Terminal-Emulation über TCP/IP), sofern man einen Account auf dem Remote System besitzt.

Das *File Transfer Protocol* (FTP) gestattet einen Dateitransfer zu (Upload) oder von (Download) einem Rechner des Netzes:

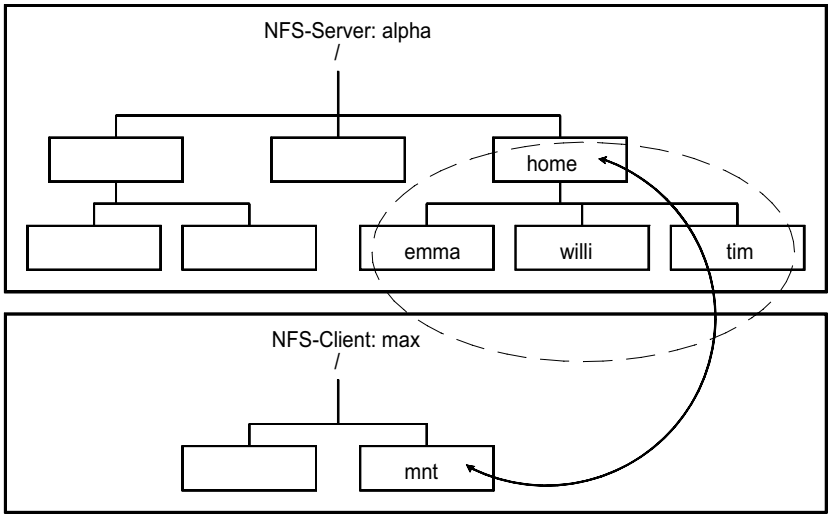
```
$ ftp <hostname  -oder-  IP-Adresse>
```

Nach erfolgreichem Login hat man Zugang zu dem Teil des Remote Dateisystems, zu dem man auch bei einem lokalen Login zugriffsberechtigt wäre.

TCP/IP ist auch das Transportprotokoll des Internet. Alle auf TCP/IP basierenden Internet-Anwendungen sind damit auch Anwendungen von Unix-Netzen. Es ist eigentlich jedoch umgekehrt: das Internet hat sich aus Unix-Netzen entwickelt, es bestand zunächst ausschließlich aus Unix-Hosts. Erst nach und nach haben andere Betriebssysteme (z. B. Windows, NetWare, OpenVMS,...) TCP/IP in ihr System integriert und sind damit „Internetfähig“ geworden. Telnet und FTP sind klassische Internet-Anwendungen. Da wir das Internet und seine Anwendungen in einem separaten Kapitel behandeln, wollen wir hier nur auf zwei typische im *lokalen* Bereich eingesetzten Netzdienste von Unix eingehen: NFS und X-Windows.

17.3.1 NFS

NFS (*Network File System)* wurde von der Firma Sun entwickelt und ist heute fester Bestandteil des Betriebssystems Unix. Es ermöglicht die gemeinsame Nutzung von Dateisystemen (Filesystemen) an verschiedenen Netzrechnern. Der als Server arbeitende Rechner (NFS-Server) stellt einen Teil seines Filesystems anderen Rechnern zur Verfügung, indem er einen Teilbaum seiner Directory-Struktur in das Netz „exportiert". Der NFS-Client muss dieses exportierte Filesystem in sein eigenes Filesystem integrieren. Da es in Unix keine einzelnen „Laufwerke" gibt und das Filesystem grundsätzlich nur aus einem einzigen Dateibaum besteht, muss das exportierte Filesystems (– ebenso wie das Filesystems einer Diskette –) an einer Stelle des Client-Dateibaums „einmontiert" werden. Dieser Vorgang heißt *mounten.*



In dem Beispiel lautet der Mount-Befehl des Client:

```
$ mount  -t  nfs  alpha:/home  /mnt
```

Hiermit startet der Client einen Dialog mit dem Server. Der Server prüft in seiner lokal vorliegenden Datei */etc/exports*, welche Teile seines Filesystems zum Export freigegeben sind, welche Clients zugriffsberechtigt sind (IP-Adressen) und auf welche Art der Zugriff stattfinden darf: *read-only* oder *read/write.* Die Zugriffserlaubnis auf die einzelnen Dateien richtet sich nach den mitübertragenen Standard-Dateiattributen eines jeden Unix-Filesystems.

Typisch ist der Export von Benutzerverzeichnissen in einem Unix-Cluster. Den Benutzern wird so von jedem Rechner aus der gleiche Zugriff auf ihren Datenbestand ermöglicht. Dabei müssen die Benutzer jeweils auf dem Server und auf dem Client mit gleichem Benutzernamen (UID, User IDentification) und Gruppenname (GID, Group IDentification) geführt werden. Ferner sollten natürlich Benutzerverzeichnisse mit der Option *read/write* exportiert werden. Das praktische Arbeiten mit importierten NFS-Dateien unterscheidet sich nicht von lokal abgelegten Daten. Durch NFS wird eine zentrale Datensicherung der Benutzerdaten ermöglicht.

Das „Mounten" von Filesystemen ist eine typische Aufgabe des Unix-Systemadministrators (UID: root), der meistens durch einen Eintrag im Bootskript des Client-Rechners das Mounten des NFS-Filesysteme beim Systemstart veranlasst. Ein Benutzer sieht *ein* Filesystem und weiß oft gar nicht, auf welchem Rechner welche Daten physikalisch abgelegt sind. Er kann sich mit dem mount-Befehl (ohne Optionen) aber die eingebundenen Filesysteme anzeigen lassen.

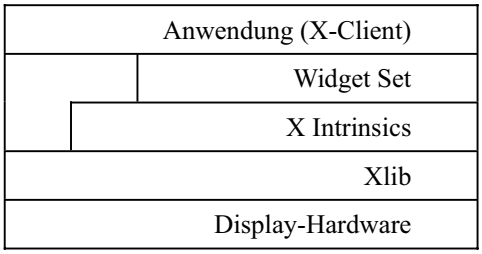
Die Autorisierungskontrolle des mountd-Dämon des Servers besteht lediglich in einer Überprüfung der Clientadresse (IP-Adresse) und entspricht nicht den heutigen Sicherheitsansprüchen des Internet. NFS gilt daher als unsicher und wird daher kaum über das Internet eingesetzt. Eine Möglichkeit, Angriffe vom Internet aus auf ein privates Unix-Cluster mit aktiviertem NFS abzublocken, besteht darin, den verbindenden Router so zu konfigurieren, dass die für NFS verwendeten Ports gesperrt werden.

17.3.2 X-Windows

Die graphische Benutzeroberfläche (*GUI, Graphical User Interface)* von Unix-Systemen ist *X-Windows.* Im Gegensatz zu den Windows-Betriebssystemen von Microsoft ist X-Windows ein internationaler Standard (aktuelle Version: X11R6), der weder an eine bestimmte Rechner-Hardware noch an ein bestimmtes Betriebssystem gebunden ist. X-Windows kommt auch auf nicht-Unix-Betriebssystemen zum Einsatz.

Die Grundidee von X ist, durch Standardisierung eines Grafik-Befehlssatzes Anwendungsprogramme von der speziellen Hardware abzutrennen und den Programmentwickler davon zu entlasten, sich mit den speziellen Hardware-Eigenschaften des gerade vorliegenden Displays beschäftigen zu müssen. Durch die Zwischenschaltung einer Pufferschicht „X“ zwischen Anwendungsprogramm und Display-Hardware mit standardisierter Anwenderschnittstelle wird die gewünschte Plattform- und Herstellerunabhängigkeit erreicht. Eine ähnliche Forderung nach „offenen“ Systemen hatten wir beim ISO/OSI-Schichtenmodell kennen gelernt.

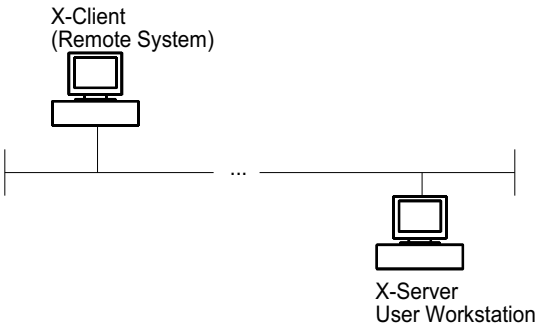
Der X-Standard umfasst mehrere Bibliotheken, die Anwenderschnittstellen mit unterschiedlichen Abstraktionsebenen bereitstellen.



Xlib:	grundlegende C-Bibliothek für das Zeichnen und Verwalten von Fenstern. „Der Assembler für Grafikprogrammierung“
X Intrinsics:	Toolkit; Objektorientierte Modul-Schnittstelle
Widget Set:	Sammlung von wiederverwendbaren Komponenten (z. B. Screenbuttons, Scrollbars,...)

Aufgrund der klaren Trennung zwischen der X-Anwendung und der Bildschirmdarstellung gibt es in X-Window-Systemen zwei getrennte Prozesse, die miteinander kommunizieren: *X-Client* und *X-Server*. Der X-Client ist die eigentliche Anwendung, der X-Server stellt die Möglichkeiten einer fensterorientierten GUI zur Verfügung. X-Window-Systeme können daher nur in einer Multitasking-Umgebung eingesetzt werden.

Was hat das nun mit Netzwerken zu tun? X-Client und X-Server müssen nicht auf der gleichen Workstation laufen, sie können sich auf verschiedenen Netzknoten befinden. Die Kommunikation im Netz zwischen Client und Server übernimmt das *X-Netzwerk-Protokoll*. Es basiert auf den Transportprotokollen TCP/IP.



ACHTUNG: Bei X-Anwendungen im Netzwerk ist der X-Server das Gerät, das das Display verwaltet. An diesem Rechner arbeitet in der Regel der Anwender. Der X-Client ist das Anwenderprogramm, das auf einem Remote Rechner ablaufen kann.

Der X-Server verwaltet die grafischen Ressourcen wie Display, Tastatur, Maus und mehrere Bildschirmfenster und stellt diese anderen Anwendungen (X-Clients) im Netz zur Verfügung.

Eine spezielle Anwendung ist der *Window-Manager*. Nachdem der X-Server lokal gestartet wurde, übernimmt der Window-Manager die Verwaltung der dargestellten Fenster und ermöglicht Einstellungen zu den Fensterpositionen, Größen und Farben nach den Wünschen des Benutzers. Unix-Systeme bieten oft mehrere Window-Manager an, die sich durch die Gestaltungsmöglichkeiten der GUI unterscheiden. Bekannte GUIs sind *Motif, OPEN LOOK* oder *CDE (Common Desktop Environment,* in Linux: *KDE).*

Nachdem der Window-Manager aktiv ist, können weitere lokale oder remote X-Clients gestartet werden, die jeweils in einem separaten Bildschirmfenster ablaufen.

X-Anwendungen erkennt man an dem vorgestellten „x...“, z. B. xterm, xeyes, xclock, xedit. Unter diesen ist *xterm* eine besonders wichtige Anwendung. Sie stellt in einem Fenster eine Terminalemulation (Konsole) zur Verfügung und gestattet so den weiteren Dialog mit dem Rechner zur Eingabe von Unix-Kommandos oder den Aufruf anderer X-Anwendungen. Unter X-Windows können mehrere xterm-Fenster gleichzeitig gestartet werden. So könnte z. B. in einem Fenster das lokale Directory angezeigt werden, in einem anderen Fenster eine Telnet-Sitzung zu einem Internet-Host durchgeführt werden und in einem dritten Fenster eine Datenbankanwendung mit einem Host des LAN ablaufen. Der Benutzer bestimmt mit der Maus, welches Fenster aktuell Eingaben entgegennehmen soll (*Fokussierung).*

Um remote Clients auf dem lokalen X-Server darzustellen, muss auf dem remote Rechner die entsprechende Anwendung gestartet werden. Dabei muss dieser Anwendung aber noch mit der *-display* -Option mitgeteilt werden, welcher X-Server benutzt werden soll. Diese Information ist sehr wichtig für den Client, da er sonst seinen lokalen X-Server benutzen würde! Der Start eines remote Client benötigt also zwei Schritte:

1. Login auf dem remote Rechner
2. Start des Client auf dem remote Rechner mit
   >X-Anwendung> *-display <eigene Hostadresse>:<screen>*

Beispiel:

der eigene Rechner ist r2, der remote Rechner r5:

1. r2> telnet r5
   Login-Prozess
2. r5> /usr/bin/X11/xclock -display r2:0 &

==> Es öffnet sich ein Fenster, in dem die Uhr dargestellt wird.
(nachgestelltes '&': Programm läuft im Hintergrund)

Damit remote Clients auf den lokalen X-Server zugreifen dürfen, müssen sie auf dem lokalen System mit *xhost* in eine Zugriffsberechtigungsliste eingetragen werden:

xhost +<remote Rechner>,

z. B.: xhost +r5
xhost +r6
...

„xhost +" erlaubt den Zugriff für alle Rechner, „- <host>" entzieht einem Rechner wieder das Zugriffsrecht, „xhost –" lässt von keinem Rechner einen Zugriff zu.

Der Start des X-Window-Systems erfolgt im Konsolenmodus meistens durch die Shell-Skripte *startx* oder *initx.* In ihnen wird der X-Server, der Window-Manager und mindestens ein xterm-Fenster aktiviert. Auf manchen UNIX-Workstations wird dagegen schon beim Booten der X-Server gemeinsam mit einer Anwendung, dem *X Display Manager, xdm,* gestartet. Hier erscheint schon vor der Anmeldung eines Benutzers eine grafische Benutzeroberfläche mit einer Login-Box. Nach erfolgreichem Login startet der lokale Window-Manager.

Eine sehr interessante Möglichkeit für die Benutzung von X-Windows besteht darin, beim Starten des X-Servers nicht den lokalen, sondern einen remote Display Manager xdm aufzurufen. Das bewirkt, dass auch die Login-Box des remote-Systems erscheint und nach dem Login auch die GUI (Window-Manager) des remote Systems aktiv wird. Fortan arbeitet man also vollkommen mit dem remote-Rechner, der eigene Rechner ist zu einem *X-Terminal* geworden. Diese Arbeitsweise ist immer dann sinnvoll, wenn der remote-Host wesentlich leistungsfähiger ist als die eigene Workstation.

xdm kommuniziert mit einem eigenen Netzwerkprotokoll *XDMCP (XDM Control Protocol)* mit dem X-Server. Es benutzt TCP/IP als Transportprotokolle.

Beim Aufbau einer Verbindung zu einem entfernten xdm-Host gibt es drei mögliche Optionen, die beim Start des X-Servers aus dem Konsolmodus angegeben werden können:

1. Option *-query:* Verbinden mit einem bestimmten Host:

> /usr/bin/X11/X -query <host> -once
===> Login Box des Rechners <host>

2. Option *-indirect:* Anzeige einer „Chooser-Box":

> /usr/bin/X11/X -indirect <host> -once
===> Anzeige aller im Netz verfügbaren xdm-Hosts;
Auswahl eines Hosts und Login dort

3. Option *-broadcast:* generelle Netzanfrage

> /usr/bin/X11/X -broadcast -once
===> Anzeige aller Hosts, auf denen xdm läuft

Die Option *-once* stellt sicher, dass nach dem Sitzungsende (Logout) auch der X-Server beendet wird und der lokale Konsolenprompt erscheint. Ohne diese Option würde nach einem Log-

out an dem remote Host wieder dessen Login-Box erscheinen und eine Rückkehr zum lokalen System nicht möglich sein! (Abbruch dann nur durch kill-Kommando möglich).

Der Administrator eines Unix-Hosts kann XDMCP-Verbindungen durch eine Reihe von Konfigurationsdateien steuern. Eine der wichtigsten ist die Datei */usr/lib/X11/xdm/Xaccess,* in der festgelegt ist, welcher Server bei XDMCP-Anfragen ein Login-Window erhält.

X-Windows wird praktisch nur in LANs benutzt, da die Netzbelastung von X-Anwendungen sehr hoch ist, so dass die Weitverkehrsverbindungen stark belastet würden (Jede Mausbewegung erzeugt Datenverkehr!).

17.4 Aufgaben

1) In einem gemischten Windows/Linux – Netzwerk soll eine Datei von einer Windows-Workstation auf ein Unix-System übertragen werden. Sie haben auf beiden Rechnern ein Account und physikalischen Zugang zu beiden Rechnern. Wie gehen Sie vor?

2) Warum sind in einem gemischten Windows/Unix – LAN die Unix-Rechner nicht sichtbar in der „Netzwerkumgebung“ des Windows-Explorer?

3) Wie kann ein Linux-Benutzer feststellen, welche Teile des Dateisystems nicht lokal auf der Festplatte liegen sondern über NFS importiert sind?

4) Wie lässt sich in einem Windows-Netzwerk feststellen:

a) Alle aktiven Workstations der Domäne,

b) Alle für die Domäne zugelassenen Benutzer.

5) Sie arbeiten an einem Linux-System unter X. Wie können Sie Ihren X-Server so einstellen, dass nur der Rechner 196.30.56.112 Ausgaben auf Ihrem Display machen darf, alle anderen Rechner sollen abgewiesen werden.

6) Untersuchen Sie die Zugriffsrechte auf Dateien in Ihrem Home Directory auf einem

a) Windows-Server
(Hilfe: *Explorer → Eigenschaften → Sicherheit → Berechtigungen)*

b) NetWare Server
(Hilfe: *Explorer → Eigenschaften → NetWare Rights)*

c) Unix (Linux) Server
(Hilfe: *ls -l* und *chmod* , *man chmod*)

Legen Sie mit dem Editor eine neue Datei *test1.txt* an, die von allen Nutzern gelesen und von Gruppenmitgliedern auch verändert werden kann.

Legen Sie mit dem Editor eine neue Datei *test2.txt* an, die nur von Gruppenmitgliedern gelesen aber nicht verändert werden kann; andere sollen keinen Zugriff haben.

Legen Sie mit dem Editor eine neue Datei *test3.txt* an, auf die nur Sie selbst zugreifen können.

18 Internet-Verbindungen

Das Internet besteht aus einer großen Zahl von Einzelnetzen, die über Router verbunden sind und TCP/IP als Transportsystem nutzen. Damit gilt im Internet die IP-Adressierung, deren Grundlage wir bereits im Kap. 16 vorgestellt haben.

18.1 IP-Adressierung im Internet

Jeder Rechner (Host) im Internet ist über seine weltweit eindeutige Adresse ansprechbar. Die Adressierung kann auf zwei Weisen vorgenommen werden:

- numerische 4 Byte IP-Adresse (Schicht-3 Adresse),
  z. B. 193.175.41.16
- Host- und Domainname (Schicht-7 Adresse),
  z. B. ux1.mnd.fh-wiesbaden.de

Die Zuordnung der Namen zu den numerischen IP-Adressen erfolgt über eine spezielle Anwendung: den *Domain-Name-Service (DNS)* (→ s. Kap. 18.2).

Der Datentransport über das Netz geschieht grundsätzlich anhand der numerischen IP-Adresse, da nur sie von den Routern ausgewertet werden kann. Wird eine Domain-Adresse angegeben, muss sie erst durch das DNS in die numerische Form gewandelt werden, bevor eine Kommunikation aufgebaut werden kann.

18.1.1 Adressklassen

IP-Adressen für das Internet werden meistens Netzweise vergeben. Wie wir schon im Kap. 16.1 festgestellt haben, enthält eine IP-Adresse sowohl die Netzkennung als auch die Hostkennung:

IP-Adressaufbau: Netzwerk-ID + Host-ID

Je nach Lage der Netz/Host-Schnittstelle innerhalb der Adresse unterscheidet man die drei Netzklassen A, B und C. Zwei weitere Adressklassen D und E gelten für spezielle Zwecke:

IP Adressklassen

bbbbbbbb.bbbbbbbb.bbbbbbbb.bbbbbbbb	32 Bit binärer Adressaufbau mit, b = n oder h n: Netzbit, h: Hostbit
xxx.xxx.xxx.xxx	Dezimalschreibweise, mit x = N oder H N: Netzwerk-ID, H: Host-ID

Klasse A:

Binär: **0**nnnnnnn.hhhhhhhh.hhhhhhhh.hhhhhhhh

	Adressbereich	*Default-Netzwerkmaske*	*Anzahl Netze*	*Anzahl Hosts*
Dezimal:	1.HHH.HHH.HHH 2.HHH.HHH.HHH ... 126. HHH.HHH.HHH	255.0.0.0	126	16 777 214

Klasse B:

Binär: **10**nnnnnn.nnnnnnnn.hhhhhhhh.hhhhhhhh

	Adressbereich	*Default-Netzwerkmaske*	*Anzahl Netze*	*Anzahl Hosts*
Dezimal:	128.NNN.HHH.HHH 129.NNN.HHH.HHH ... 191.NNN.HHH.HHH	255.255.0.0	16 384	65 534

Klasse C:

Binär: **110**nnnnn.nnnnnnnn.hhhhhhhh.hhhhhhhh

	Adressbereich	*Default-Netzwerkmaske*	*Anzahl Netze*	*Anzahl Hosts*
Dezimal:	192.NNN.NNN.HHH 193.NNN.NNN.HHH ... 223.NNN.NNN.HHH	255.255.255.0	2 097 152	254

Klasse D:

Binär: **1110**bbbb.bbbbbbbb.bbbbbbbb.bbbbbbbb

Dezimal: 224. ..... Multitaskadressen an eine Gruppe von Hosts
...
239. .....

Klasse E:

Binär: **11110**bbb.bbbbbbbb.bbbbbbbb.bbbbbbbb

Dezimal: 240. ..... reservierter Bereich
...
255. .....

Weitere Regeln:

Eine Hostkennung darf nicht nur aus binären „0“ bestehen.
Eine Hostkennung darf nicht nur aus binären „1“ bestehen.
Die Adresse 127. .... ist für das eigene Netzinterface *(loopback device)* reserviert.

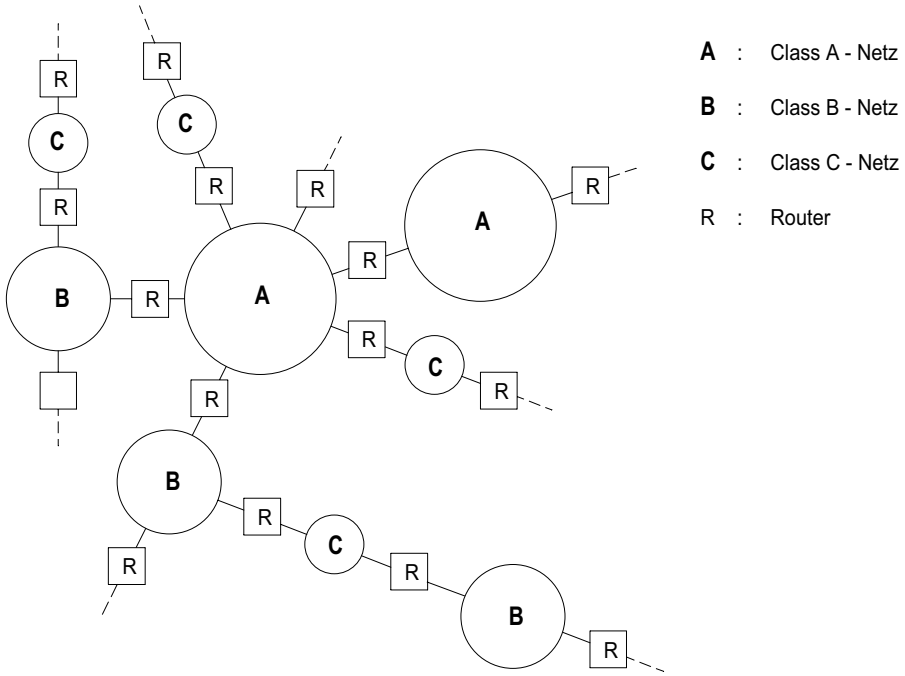
Die Zugehörigkeit zum Klassentyp ist also aus der ersten Zahl zu entnehmen. Nur die Adressklassen A, B und C sind für die Hostadressierung verwendbar; nur sie werden von den Service-Providern an Unternehmen und Organisationen vergeben.

■ **Beispiele:**

IP:	10.101.12.76	→	Knoten gehört zu einem Class-A-Netz, Netzkennung: „10“;
	135.83.250.124	→	Knoten eines Class-B-Netzes, Netzkennung: „135.83“;
	193.175.39.12	→	Knoten eines Class-C-Netzes, Netzkennung: „193.175.39“;

■

Die IP-Router des Internet speichern Tabellen mit den Netzkennungen derjenigen Netze, die über sie erreichbar sind. Das Internet kann man sich demnach als den Zusammenschluss von Class-A-, -B- und -C-Netzen vorstellen, die einen Teil ihres Adressraumes für Router „opfern“ müssen.



Prinzipieller Aufbau des Internet

Bei der Installation des TCP/IP-Stacks auf einem Rechner wird neben der vom Netzwerkadministrator vergebenen individuellen IP-Adresse (*Unicast address)* zusätzlich die feste *loopback*-Adresse 127.0.0.1 zugeordnet. Mit ihr wird grundsätzlich der eigene Rechner *(localhost*) angesprochen. Dies wird häufig zu Tests für die lokale Installation benutzt.

Mit dem Konsolenkommando

> traceroute <internet-adresse> (für Unix)

bzw.

> tracert <internet-adresse> (für Windows)

kann man sich alle Router anzeigen lassen, die auf dem Weg vom Absender zu dem Ziel <internet-adresse> durchlaufen werden. Mit Hilfe dieser Anwendung können z. B. falsch konfigurierte Rechner entdeckt werden, wenn ein anderer als der vorgesehene Weg zum Zielrechner benutzt wird. Die ausgegebenen Laufzeiten zu den einzelnen Routern geben Auskunft über besonders belastete Abschnitte im Internet und zeigen evtl. durch einen "*" an, dass ein Knoten gar nicht erreichbar ist. *traceroute* ist ein nützliches Werkzeug für die Analyse von Verbindungsproblemen im Internet.

18.1.2 Broadcast- und Multicast-Adressen

In jedem IP-Netz ist jeweils die höchste Host-Adresse als Broadcast-Adresse reserviert, sie darf also nicht an einen Host vergeben werden.

Beispiel: Netzkennung: 195.72.101.0

Broadcast-Adresse: 195.72.101.255

Broadcasts breiten sich nur innerhalb eines Netzes aus.

Multicast-Nachrichten sind an Gruppen gerichtet. Als Zieladresse wird eine IP-Adresse der Klasse D eingesetzt.

Ein typisches Beispiel für Multicast-Datenverkehr bilden die „Bekanntmachungen" eines Routers an „alle Router", d. h. an die Gruppe der Router eines Netzes:

Multicastadresse an Router: 224.xxx.xxx.xxx

Broadcasts und Multicasts
Verbreiten sich nur innerhalb eines Netzes, werden also durch Router nicht weitergeleitet.

18.1.3 „Private" Adressen

Die ICANN hat drei IP-Adressbereiche für die Internet-Adressierung ausgespart, die nicht im Internet vorkommen dürfen und nicht im Internet geroutet werden. Diese „unregistrierten" Adressen kann jedes Unternehmen frei benutzen, wenn die betreffenden Geräte nicht direkt mit dem Internet kommunizieren.

Private Adressbereiche

Class A:	10.0.0.0 -- 10.255.255.255
Class B:	172.16.0.0 -- 172.31.255.255
Class C:	192.168.0.0 -- 192.168.255.255

Diese Möglichkeit wird in der Praxis stark für Netze genutzt, die nicht mit dem Internet verbunden sind oder das Internet nur indirekt über „Proxy-Server" oder Firewalls erreichen können. Diese Geräte gestatten eine „Adress-Umsetzung" (*NAT Network Address Translation* oder *Masquerading,* → s. Kap. 18.4.4) auf private Adressen des hinter ihnen „verborgenen", d. h. im Internet nicht sichtbaren Netzes. Diese Techniken helfen, den sprunghaft angestiegenen „Adressverbrauch" im Internet einzudämmen.

18.1.4 Subnetze

Netzwerkverwalter können zugeteilte Class-A-, -B-. oder -C-Netze weiter segmentieren, in dem sie die Netz/Host-Schnittstelle auf Bitebene weiter nach rechts verschieben. Damit lässt sich z. B. ein Class-B-Netz in weitere Subnetze (die nun nur über zusätzliche Router zugänglich sind!) einteilen. Von diesem *Subnetting* macht man in der Praxis intensiven Gebrauch, um eine Verkehrstrennung zu erreichen und große Netze besser administrieren zu können.

Einer IP-Adresse kann man nicht ansehen, in welchem Subnetz sie liegt oder ob überhaupt Subnetze eingerichtet wurden. Diese Information ist aber für das Versenden von Nachrichten wichtig: liegt der Empfänger im gleichen Netz, wird er direkt adressiert, liegt er aber in einem anderen Subnetz, muss der vermittelnde Router angesprochen werden. Die Netzzugehörigkeit einer IP-Adresse geht aus der Netzwerkmaske hervor. Der Netzwerkadministrator kann von der Default-Netzwerkmaske abweichen, in dem er z zusätzliche „1“sen in die Maske aufnimmt und so 2^z zusätzliche Netze einrichtet. So könnte z. B. ein Class-B Netz weiter unterteilt sein:

IP-Adresse:	130.83.241.44	binär: 10000010. 01010011. 11110001. 00101100
Subnetzmaske:	255.255.255.0	binär: 11111111. 11111111. 11111111. 00000000
		< Netzkennung > < Hostk. >

Hier gibt es statt *eines* riesigen Netzes mit 65534 Hosts 255 Subnetze mit jeweils nur 254 Hosts, was die Netzverwaltung erheblich vereinfacht und wegen der Entkopplung des Datenverkehrs die *Performance* verbessert.

Konfiguration: „130.83.241.44 / 24“ (24 Netzbits).

Alle Rechner des Class-B-Netzes besitzen die gleiche Subnetzmaske „/24“, auch wenn sie verschiedenen Subnetzen angehören.

Die Netzwerk/Host-Schnittstelle muss nicht auf eine Bytegrenze fallen, auch Class-C-Netze werden häufig in Subnetze unterteilt.

Subnetz-Bildung
Unterteilung von Class-A- oder Class-B- oder Class-C- Netzen in 2^z Subnetze, in dem der Default-Netzwerkmaske z zusätzliche “1” angefügt werden.

■ **Beispiele:**

1) *dezimal:* *binär:*

IP-Adresse: 130.83.250.131

← N →← H →

Netzwerkmaske: 255.255.0.0 11111111.11111111.00000000.00000000

→ Netz-ID 130.83.0.0, keine Subnetze in dem Class-B-Netz,
Host-ID 250.131
Broadcastadresse 130.83.255.255

2) dezimal: binär:

IP-Adresse: 131.84.165.11

← N →← H →

Netzwerkmaske: 255.255.255.0 11111111.11111111.11111111.00000000

→ Netz-ID 131.84.165.0, Class-B-Netz ist in $2^8 = 256$ Subnetze unterteilt,
Host-ID 11, Subnetz-ID 165,
Broadcastadresse 131.84.165.255

3) dezimal: binär:

IP-Adresse: 193.174.36.109 ..... .01101101

← N →←H →

Netzwerkmaske: 255.255.255.192 11111111.11111111.11111111.11000000

→ Class-C-Netz ist in $2^2 = 4$ Subnetze unterteilt:

Subnetz 1: .....00..., Netz-ID 193.174.36.0,
Hostbereich: 1 .. 62,
Broadcastadresse: 193.174.36.63

Subnetz 2: .....01..., Netz-ID 193.174.36.64,
Hostbereich: 65 .. 126,
Broadcastadresse: 193.174.36.127

Subnetz 3: .....10..., Netz-ID 193.174.36.128,
Hostbereich: 129 .. 190,
Broadcastadresse: 193.174.36.191

Subnetz 4: .....11..., Netz-ID 193.174.36.192,
Hostbereich: 193 .. 254,
Broadcastadresse: 193.174.36.255

Somit liegt der Host im Subnetz 2 und hat die Host-ID 45.

■

Je feiner ein Netz in Subnetze unterteilt wird, desto mehr Adressen-„Verschnitt“ ergibt sich durch Netz-IDs und Broadcast-Adressen, die nicht an Host vergeben werden können.

18.1.5 Klassenlose Adressierung

Der IP-Adressraum des Internet ist sehr knapp geworden. Eine der Maßnahmen, die Situation zu entschärfen, ist die Einführung der „klassenlosen“ Adressvergabe: CIDR (*Classless Inter-Domain Routing*).

Die Praxis hat gezeigt, dass die starre klassenweise IP-Adressvergabe an Institutionen (Unternehmen, Hochschulen, usw.) nicht immer flexibel genug ist, um den Bedürfnissen der Institution zu begegnen. So besitzt z. B. eine Universität ein Class-B-Netz, benötigt tatsächlich aber vielleicht nur 15000 IP-Adressen statt der zugewiesenen 65534 IP-Adressen eines Class-B-Netzes. Ein großer Teil bleibt ungenutzt. Eine FH kommt dagegen nicht mit einer Class-C Netzadresse aus und beantragt z. B. 16 Class-C Netze. Alle diese 16 Netz-IDs müssen im Internet separat zu der FH geroutet werden.

CIDR löst die starre Klasseneinteilung auf, in dem den Institutionen Adressräume durch frei definierte Netzwerkmasken zugewiesen werden. Eine Universität kann z. B. ein Adressraum erhalten, der durch eine Netzwerkmaske „/18“ definiert ist, der 16384 Host-Ids enthält. Eine FH erhält durch „Supernetting“ einen zusammenhängenden Adressraum durch Zuweisung einer Netzwerkmaske „/20“, so dass 4096 IP-Adressen verfügbar sind.

Im Gegensatz zur Subnetz-Bildung, die ja jede Institution „für sich“ durchführt, muss CIDR von den Internet-Routern unterstützt und entsprechend konfiguriert werden. Der Zusammenschluss von Netzen durch Supernetting (mehrere Class-C-Netze durch *eine* Netz-ID ersetzen) trägt darüber hinaus zur Entlastung der Routertabellen im Internet bei.

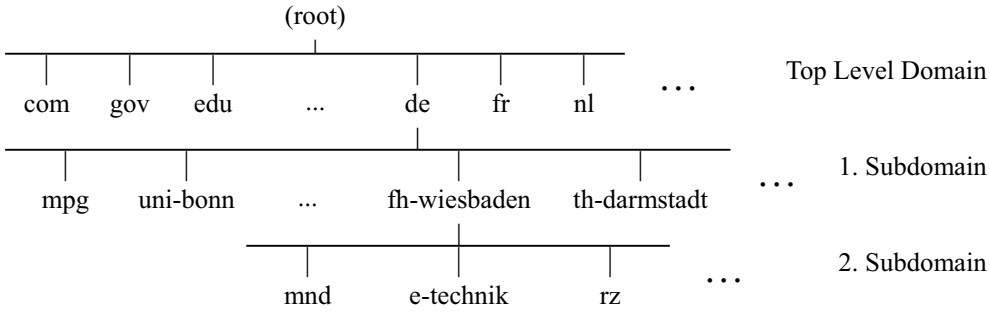
18.2 Der Domain-Name-Service

Die Adressierung mit numerischen IP-Adressen ist in der Praxis unbequem, da Zahlenkombinationen schwer zu merken sind und aus ihnen keinerlei Information über den Adressaten zu entnehmen ist. Daher wurde eine neue Adressierung (die dritte!) über „Domain“-Namen eingeführt. Der Domain-Name-Service (DNS) ist eine Anwendung der Schicht 7 des OSI-Modells und ist über Port 53 ansprechbar. In der Praxis wird ein Internetknoten meistens statt mit seiner IP-Adresse mit seinem Domainnamen angesprochen, z. B. „r4.mndu.fh-wiesbaden.de“.

Der Aufbau einer Domain-Adresse hat die allgemeine Form:

hostmame**.**subdomain_n**.**...subdomain_2**.**subdomain_1**.**Top_Level_Domain

Der Namensraum im Internet ist ähnlich einer Directory-Struktur hierarchisch angelegt:



Die Top Level Domains (TDL) enthalten entweder den ISO-Ländercodes wie z. B. de, fr, uk (s. Anhang auf unserer Buch-Webseite) oder die „generische“ Netzkennungen:

.com (commercial) Firmen
.edu (education) Forschungsinstitute, Hochschulen
.gov (government) US-Regierungsstellen
.org nicht-kommerzielle Einrichtungen
.mil militärische Einrichtungen
.net Organisationen für Netzwerk-Management

sowie die seit dem Jahr 2000 von der ICANN neu zugelassenen TDLs:

.biz uneingeschränkte geschäftliche Nutzung
.info Informationsdienste
.name nur persönliche Nutzung, nicht kommerziell
.pro für Freiberufler wie z. B.Rechtsanwälte, Ärzte, Steuerberater usw.
.museum Museen und Ausstellungen
.aero Unternehmen, die mit Flugverkehr zu tun haben
.coop Verbände und Vereinigungen

Seit 2006 gibt es die TLD „eu“ für die Europäische Union. Sie ist stark nachgefragt.

Damit auch die Namen weltweit eindeutig bleiben, muss jede Institution die 1. Subdomain bei den zuständigen Internet-Netzwerkzentren (*NIC*-Zentren) anmelden. Für eine weitere Einteilung in darunter angesiedelten Subdomains ist jede Institution selber verantwortlich. Bei deutschen Hochschulen finden wir meistens eine Adressstruktur der Form:

<rechnername>.<fachbereich>.<hochschule>.de,

also z. B. ux1.umwelttechnik.fh-wiesbaden.de.

Wie wir bereits oben festgestellt haben, ist zum Aufbau einer Netzverbindung jedoch die numerische IP-Adresse erforderlich. Die ein-eindeutige Zuordnung von Domainnamen in IP-Adressen erfolgt durch den Domain-Name-Service. Jede am Internet angeschlossene Institution muss einen oder auch mehrere DNS-Server betreiben, in denen für die Rechner der betreffenden Institution diese Adressumsetzungen in Form von Tabellen gespeichert sind. DNS-Server werden zur Namensauflösung von Anwendungen oder von anderen DNS-Servern abgefragt.

DNS-Informationen bilden eine weltweit verteilte Datenbank, der alle DNS-Server des Internet angehören. Die Namensauflösung erfolgt über verschiedene DNS-Server der hierarchisch angeordneten Domainebenen bis hin zu den Root-Name-Servern, die als oberste Instanz alle DNS-Server der Top Level Domains kennen.
Diese wiederum führen Listen aller DNS-Server der 1.Subdomain usw.

■ **Beispiel:**

Ein Benutzer an einem Rechner der Domain „fh-wiesbaden.de“ gibt das Kommando:
>telnet sun1.informatik.uni-hamburg.de

Schritt 1: Die Benutzer-Arbeitsstation befragt den lokalen DNS-Server der FH Wiesbaden. Dieser kennt die gesuchte IP-Adresse nicht.

Schritt 2: Der lokale DNS-Server wendet sich an „seinen Vorgesetzten“, den Nameserver von Deutschland „ns.nic.de“. Dieser kennt die Domain-Server der Domain „uni-hamburg“ und leitet die Anfrage dorthin weiter.

Schritt 3: Der DNS-Server der Uni Hamburg kennt die Subdomain „informatik“ und alle darin befindlichen Hosts. Er findet die IP-Adresse von Host „sun1“: 134.100.9.6.

Schritt 4: Die IP-Adresse wird über die beteiligten Name-Server an den Ausgangsrechner zurückübertragen.

Schritt 5: Das Kommando heißt nun: >telnet 134.100.9.6

Schritt 6: Die Protokoll-Software erkennt, dass die IP-Adresse nicht im eigenen Netz liegt. Deshalb wird das Telnet-Kommando zum lokalen Internet-Router („default Gateway“) geschickt.

weitere Schritte: Aufbau der Verbindung über die Internet-IP-Router zum Zielhost.

■

Da Nameserver die über sie laufenden Informationen zur Namensauflösung „cachen“, laufen nachfolgende Anfragen von anderen Rechnern eventuell erheblich schneller ab.

nslookup

Die normalerweise im Hintergrund stattfindende Abfrage des DNS-Systems kann man mit der Konsolenanwendung *nslookup* auch interaktiv durchführen und am Bildschirm ausgeben. *nslookup* liefert die IP-Adresse von Domain-Adressen, z. B.

```
>nslookup
...Ausgabe des auskunftgebenden  DNS-Servers...
> r5.umwelttechnik.fh-wiesbaden.de
⇒      193.175.39.115
```

Auch umgekehrt lässt sich der Domainname zu einer IP-Adresse ermitteln. *nslookup* spricht grundsätzlich zunächst den lokalen DNS-Server an. Kann dieser keine Auskunft geben, leitet dieser die Anfrage weiter an den übergeordneten DNS-Server usw. bis hin zu den Root-Name-Servern. Mit *nslookup* kann man Informationen zu den Name-Servern von anderen Institutionen und Listen aller von dieser Institution betriebenen Internet-Rechner erhalten, z. B.

> set query=ns		(Abfrage nach Name-Servern)
>uni-hamburg.de.	⇒	Liste der DNS-Server der Uni Hamburg
>de.	⇒	Liste der DNS-Server der Domain „de“
>.	⇒	Liste der Root-Name-Server

Aus Sicherheitsgründen ist die Möglichkeit zur Auflistung aller angeschlossenen Rechner (Option *-ls* <domainname>) einer Institution manchmal deaktiviert.

18.3 IPv6

Der Adressraum der 32 Bit IP-Adressen (IPv4) ist nahezu erschöpft. Die IETF hat ein neues Adressierungsschema erarbeitet, das mit *IPv6* (Version 6) bezeichnet wird und IP-Adressen auf 128 Bit erweitert. Der neue Adressraum ist theoretisch ausreichend, um jeden Quadratmeter der Erdoberfläche mit $6 \cdot 10^{23}$ Adressen auszustatten!

Die neue Adressierung ist kompatibel zu den IPv4-Adressen, da sie den IPv4-Adressraum als Untermenge enthält. Die Unterstützung der alten 4-Byte-Adressen war eine wichtige Forderung der Entwickler des neuen Adressierungssystem, damit beide Adressierungen noch für längere Zeit (Jahre!) nebeneinander betrieben werden können und so ein allmählicher Übergang zu IPv6 möglich wird. Einige Institutionen haben bereits auf IPv6 umgestellt.

Mit der Einführung der neuen Adressierung hat sich das IP-Protokoll insgesamt verändert. Wichtige Änderungen bestehen in der Unterstützung von „Mobilen Diensten“ (*Mobile IP*) und Echtzeitanwendungen sowie in der Integration von Sicherheitsaspekten im Transportsystem, die nun nicht länger durch die höheren Protokollschichten für jede Anwendung einzeln „nachgeflickt“ werden müssen: Der IPv6-Protokollstack unterstützt standardmäßig eine Datenverschlüsselung (*IPSec, Secure IP,* → s. Kap. 19.2.4).

IPv6-Adressen werden hexadezimal geschrieben, jeweils 16 Bit bilden eine 4-Hex-Zahlen Kombination, die durch „**:** “ abgetrennt sind, z. B.:

fe80**::**4b0**:**e3ff**:**dea2**:**e0d8

In diesem Beispiel ist berücksichtigt, dass Bereiche mit aufeinander folgenden Nullen durch „**::**“ abgekürzt werden können und führende Nullen nicht geschrieben werden müssen. D. h. die gezeigt Adresse lautet ausgeschrieben:

fe80**:**0000**:**0000**:**0000**:**04b0**:**e3ff**:**dea2**:**e0d8

Wie bei den alten IP-Adressen befindet sich im vorderen Teil die Netzkennung und im hinteren Teil die Hostkennung, beide Teile enthalten jeweils 64 Bit.

IPv6-Adressstruktur

n: 48 Bit Netzkennung vom Internet Provider zugeteilt

s: 16 Bit fester Raum für eigene eigene Subnetzbildung

x: 64 Bit Hostkennung, aus MAC- (Ethernet-) Adresse xx-xx-xx-xx-xx-xx ableitbar

64 Bit zur Hostadressierung erscheinen zunächst weit überzogen, denn ein Netz mit 2^{64} Rechnern wäre wohl kaum administrierbar! Die Idee dahinter ist, in diesem Raum die MAC-Adresse des Interfaces aufzunehmen zu können. Lautet z. B. die Ethernet-Adresse eines Rechners 00-10-4b-b9-f9-02, so ergibt sich die IP-Host-Adresse daraus durch eine Aufspaltung der Ethernet-Adresse in der Mitte und Einfügen der festen hex-Bit Kombination „fffe“ zu: 0010:4bff:feb9:f902.

Damit wird erreicht, dass sich IPv6-Rechner selber konfigurieren können (*autoconfiguration)*, was natürlich den Netzwerkadministrator entlastet. Die Eindeutigkeit der IP-Adressen ist automatisch gegeben, der Fehler doppelt vergebener IP-Adressen kann nicht vorkommen. Weiterhin entfallen innerhalb des LANs Broadcast-Anfragen zur HW-Adressermittlung (*ARP*-Dialoge), da diese ja aus den IP-Adressen zurückgewonnen werden können. Allerdings wird der direkte Umgang mit den IP-Adressen mühsamer, da die MAC-Adressen in ihren numerischen Werten keine Systematik aufweisen. Die gelegentliche Nutzung numerischer IP-Adressen wird dann zu Gunsten der DNS-Adressierung weiter sinken.

Für die Netzkennungen führt eine Hierarchie auf Provider-Basis zu einem effizienteren Routing-Mechanismus durch eine Reduktion der immer länger gewordenen Routingtabellen der Backbone-Internet-Router.

Zwei Adress-Formate unterstützen den Übergang von IPv4 auf IPv6:

a) (*IPv4-compatible IPv6 address*) **::** <IPv4-Adresse>, also z. B. **::**195.72.101.117 zur Übertragung (*tunneling*) einer IPv6-Adresse über eine IPv4-Router-Infrastruktur;

b) (*IPv4-mapped IPv6 address*) **::** ffff **:** <IPv4-Adresse>,
z. B. **::** ffff **:** 193.175.39.12 zur Adressierung von Hosts, die nicht IPv6 unterstützen in einer IPv6-Infrastruktur.

18.4 Internet-Zugangstechniken

Ein am Internet angeschlossene Rechner muss mindestens über folgende Informationen verfügen:

Minimale Netzkonfiguration eines am Internat angeschlossenen Rechners

Eigener Domainname und eigene IP-Adresse

IP-Netzwerkmaske

IP-Adresse des lokalen DNS-Servers

IP-Adresse des „Default Gateway“, d. h. des anzusteuernden Routers für Zieladressen, die nicht im eigenen Netz liegen.

Große Unternehmen, Forschungsanstalten und Hochschulen sind über feste Standleitungen permanent mit dem Internet verbunden. Hier muss der Netzwerkadministrator die Rechner seiner LANs mit den oben dargestellten Daten konfigurieren.

Ein privater Internet-Nutzer oder kleinere Institutionen sind nicht permanent mit dem Internet verbunden. Sie nehmen bei Bedarf über Modems, ISDN oder DSL mit dem Internet-Service-Provider (ISP) eine Punkt-zu-Punkt-Verbindung über das *PPP*-Protocol auf und werden vom ISP mit den erforderlichen Konfigurationsdaten versorgt.

Das „Einloggen in das Internet" verläuft in zwei Schritten:

Schritt 1: Aufbau einer seriellen Verbindung zum Server des Providers. Da noch keine IP-Adresse verfügbar ist, kann dafür TCP/IP nicht eingesetzt werden. Es wird das Protokoll PPP benutzt: es erfolgt eine Nutzer-Authentisierung.

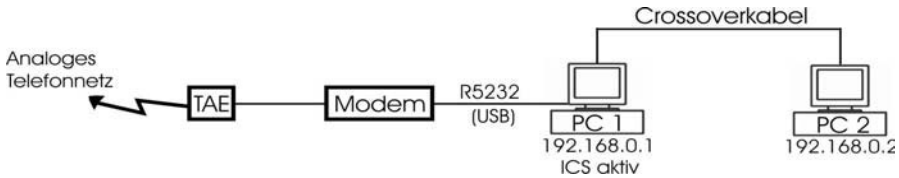
Schritt 2: Zuweisung der IP-Konfigurationsdaten.

Nun ist der Rechner Teil des Internet, d. h. er kann andere Internet-Rechner erreichen und ist selber aus dem Internet erreichbar (sofern eine *Firewall* dies nicht verhindert!).

18.4.1 Typische Anschluss-Konfigurationen

In nachfolgenden Konfigurationen ist lediglich nur *eine* registrierte IP-Adresse vom ISP erforderlich. Weitere Endgeräte erhalten „private" IP-Adressen. Der Internet-Host arbeitet als „Adress-Translation"-Server (*NAT*, → s. Kap. 18.4.4). Folgende Konfigurationen sind in der Praxis gebräuchlich, um eine Verbindung zum ISP aufzunehmen:

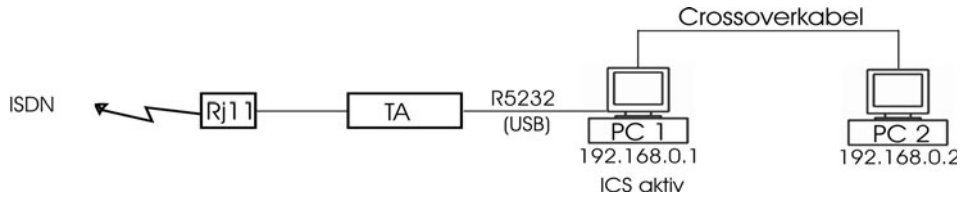
Modems:
Ein Modem ermöglicht den Internet-Zugang über einen analogen Telefonanschluss:



Beispiel: Anschluss von 2 Rechnern an das analoge Telefonnetz über Modem und ICS

Für interne Modems oder Modems mit USB-Anschluss wird ein virtueller serieller Port konfiguriert, wie es im Kap. 11.4 vorgestellt wurde. Das Modem wird im V.90 Mode betrieben, der speziell für die Internetanbindung entwickelt wurde. Die maximale Übertragungsgeschwindigkeiten sind: Downstream 56 Kbit/s, Upstream 33,6 Kbit/s zum ISP. Weitere PCs lassen sich bei Windows durch Aktivierung von *ICS (Internet Connection Sharing,* keine Bridge-Funktion!*)* am primären Internet-Host an das Internet anschließen. Hier arbeitet der Internet-Host als DHCP-Server (s.u.) für weitere Endgeräte, die über Ethernet-Einsteckkarten mit dem Internet-Host verbunden sind. Der Host vergibt eine „private" IP-Adresse mit DHCP an die zusätzlichen Rechner. Wird Linux auf dem Internet-Host eingesetzt, lässt sich dieser ebenfalls als DHCP-Server konfigurieren.

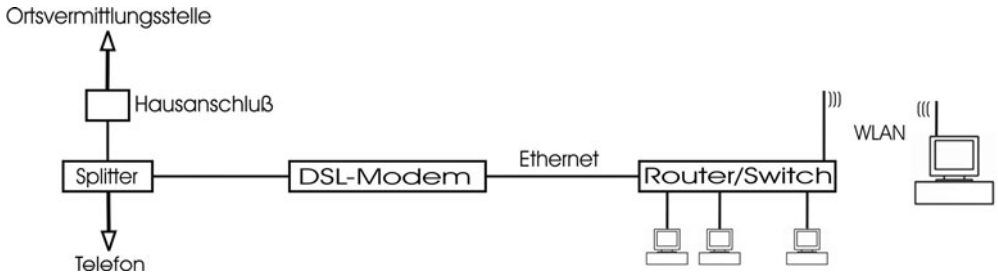
ISDN:
Ein rein digitaler Internet-Anschluß ist über einen Terminaladapter (TA) als ISDN-Einteckkarte oder als externes „ISDN-Modem" (TA mit Modememulation) über RS232C- oder USB-Kabel möglich (→ s. Kap. 11.5).



Beispiel: Anschluss von 2 Rechnern an ISDN über Terminaladapter und ICS

Die Übertragungsgeschwindigkeit beträgt entweder 64 Kbit/s oder 128 Kbit/s, wenn Kanalbündelung konfiguriert ist (Geschwindigkeitsgrenze der RS232C-Schnittstelle beachten!). Der TA arbeitet in der Regel mit dem ITU X.75 Standard. Der Internet-Zugang von mehr als einem Rechner lässt sich über ICS und DHCP-Funktionalität des primären Internet-Hosts erreichen.

DSL:
Seit der fast flächendeckend verfügbaren ADSL (*Asymmetric Digital Subcriber Line*)-Technik hat deren Nutzung sprunghaft zugenommen. DSL Internet-Zugänge sind keine Wählverbindungen, sie sind fest verschaltet mit der Vermittlungsstelle, man kann sie lediglich aktivieren oder deaktivieren. Es werden die normalen Telefonleitungen in den betreffenden Haushalten der Kunden benutzt. Dort führen sie zum „Splitter", der den breitbandigen Frequenzbereich der DSL-Datenübertragung von dem Frequenzbereich der Telefon-Datenübertragung abtrennt.



Typischer Internet-Zugang mit DSL

Die hohen Datenübertragungsgeschwindigkeiten im Mbit/s-Bereich (z. Zt. 1Mbit/s....16Mbit/s) werden durch spezielle Modulationstechniken auf konventionellen Telefon-Kupferkabeln erreicht. Die auf einer Leitung maximal realisierbare Übertragungsgeschwindigkeit ist von der Entfernung zur Vermittlungsstelle abhängig. Hierüber kann man für seinen Wohnort vom DSL-Anbieter (z. B. Telekom) Auskunft erhalten. Den Bedürfnissen der Internet-„Surver" entsprechend ist die Übertragungsgeschwindigkeit *asymmerisch*: Die Upstream-Geschwindigkeit beträgt in der Regel nur ca. 1/8 der Downstream-Geschwindigkeit. Der verbreitetste DSL-Anbieter ist die Deutsche Telekom mit ihrem Produkt T-DSL.

Das nicht-ausschaltbare DSL-Modem steuert die Übertragung zur Vermittlungsstelle. Es schließt den DSL-Zugang mit einer Ethernet-Schnittstelle (RJ45) ab. Das Ethernet-Kabel führt (eher selten) direkt zum Internet-Host oder (meistens, s. Abbildung) zu einem „Internet-Router", an dem ein LAN aus mehreren Rechnern angeschlossen ist. Die Ethernet-Verbindung zum DSL-Modem wird speziell für die PPP-Nutzung mit dem *PPP over Ethernet* (PPPoE) - Protokoll konfiguriert. Alle am Router angeschlossenen Rechner können den Internetzugang über eine integrierte Switch-Funktion gleichberechtigt nutzen. Der Router übernimmt den Verbindungsaufbau zum ISP. Er enthält meistens auch Firewall-Funktionen, einen NAT- und DHCP-Server (s. u.), der für das interne LAN „private" Adressen, z. B. aus dem Bereich 192.168..... vergibt.

18.4.1 PPP

Die Kommunikation vom privaten Internet-Host bzw. Router zum ISP verläuft über das PPP-(*Point-to-Point Protocol*) Protokoll. Es arbeitet auf den unteren beiden ISO-Schichten und ersetzt eine Ethernet-Verbindung durch eine Punkt-zu-Punkt-Verbindung.

Die wichtigsten Aufgaben des PPP-Protokolls sind:

- Verbindungsauf- und Abbau zum ISP über das übergeordnete *Link Control Protocol (LCP).*
- Authentifizierung der beiden Kommunikationspartner über die standardisierten Verfahren CHAP *(Challenge Handshake Authentication Protocol)* oder PAP (*Password Authentication Protocol).*
- Bereitstellung der IP-Konfigurationsdaten nach dem DHCP-Verfahren (s.u.).
- „Einpacken" (*Encapsulation*, „Tunnelung") der Schicht-3 IP-Datenpakete für die Internet-Kommunikation.

PPP ist Bestandteil der Netzwerkkonfiguration der Betriebssysteme Windows und Unix (Linux). Durch die Hinterlegung von *username* und *password* in Skripten lässt sich der Login-Prozess für die Authentifizierungs-Verfahren automatisieren.

Nach der Zuteilung der IP-Adresse verhält sich der Rechner wie ein gewöhnlicher Knoten in einem IP-Netzwerk. Nun kann der Benutzer jeden lokal installierten Internet Client- oder Server-Dienst starten. Beachten Sie, dass während der Verbindungszeit auch Ihr Rechner aus dem Internet ansprechbar ist und das Ziel eines unberechtigten Angriffs sein kann!

Internet-Login über PPP
PPP ermöglicht einen **Netz-Zugang**, indem es IP-Konfigurationsdaten zuweist und einen Datenweg zum Internet-Router des ISP bereitstellt. Es ist kein Login auf einem Remote Rechner, auf dem man anschließend arbeitet. D. h. man arbeitet nach einem Login „ganz normal" an seinem Rechner. Es besteht jedoch die Möglichkeit, eine Verbindung mit dem Internet aufzunehmen oder vom Internet her „angesprochen" zu werden.
Nach einem erfolgreichen PPP-Login ist man Teil des Internet.

RADIUS:
Die als ISP auftretende Institution, z. B. eine Hochschule, die ihren Studenten Zugriff zum Internet ermöglicht, betreibt in der Regel am Studienort oft mehrere Access-Server, über die man sich ins Internet einwählen kann. Für jeden „Einwähler" muss der Systemadministrator einen PPP-Account auf den Access-Servern mit Namen und Passwort anlegen. Damit ein Systemadministrator nun nicht auf den verschiedenen Access-Servern für jeden Einwähler Accounts einrichten muss, kann RADIUS (*Remote Access Dial In User Service*) eingesetzt werden. RADIUS ist eine Client-Server-Anwendung zur zentralen Verwaltung von Einwählnutzern. Der RADIUS-Server führt zentral die Accounts aller PPP-Einwähler. Die RADIUS-Client-Software der Access-Server leitet Login-Prozesse zur Benutzer-Authentification auf den zentralen RADIUS-Server um. Als RADIUS-Server können Windows Server oder Unix (Linux)-Systeme dienen, die ohnehin für Mehrbenutzerbetrieb eingerichtet sind und RADIUS unterstützen.

Eine besonders interessante Anwendung von RADIUS ergibt sich durch das *RADIUS-Roaming*: Mehrere räumlich verteilte, am Internet angeschlossene Institutionen (z. B. Hochschulen) vereinbaren, ihre PPP-Zugangseinrichtungen gegenseitig mitbenutzen zu dürfen.

■ **Beispiel:**

Ein Student der FH Wiesbaden wohnt in Darmstadt. Für einen Internet-Zugang benutzt er den Einwählserver der TU Darmstadt in Darmstadt zum Ortstarif. Durch eine Erweiterung seines Login-Namens durch eine „Zonenangabe“

<loginname>@<zone>, also z. B. weisalles@fb08.fhw,

erkennt der RADIUS-Server der TU, dass der Benuzter nicht zur TU, sondern zum Fachbereich 08 der FH Wiesbaden gehört. Der Server leitet zur Authentification des Benutzers die Anmeldung über das Internet zum entsprechenden RADIUS-Server der FH Wiesbaden weiter. Hier besitzt er einen Account. Nach positiver Rückantwort des FH-Servers erhält der Benutzer eine IP-Adresse aus dem Adress-Pool der TU Darmstadt und nutzt auch für seine weitere Internet-Kommunikation die örtlichen Einrichtungen der TU.

■

18.4.3 DHCP

DHCP (*Dynamic Host Configuration Protocol)* ist eine Client-Server-Anwendung um Clients, d. h. Rechner – meistens zeitlich begrenzt – mit IP-Konfigurationsdaten zu versorgen. Es dient nicht nur speziell für Internet-Zugänge, sondern ist auch in Lokalen Netzen einsetzbar.

Der DHCP-Server verfügt über einen Pool von IP-Adressen, die von den Clients für eine Zeitperiode „geleased“ werden können. Am Ende der Leasing-Periode fällt die IP-Adresse wieder zurück in den Pool der verfügbaren IP-Adressen, sie kann nun einem anderen Client angeboten werden.

Client-Rechner konfigurieren ihre Netzwerkinterface nicht mit einer bestimmten IP-Adresse, sondern mit der Einstellung „DHCP-Client“. Eine weitere Konfiguration ist nicht erforderlich. Befinden sich Client und Server im gleichen Lokalen Netz, sendet der Client beim Booten ein Ethernet-Broadcast (*DHCP-Request)* aus, um die IP-Daten vom DHCP-Server anzufordern. Bei temporären Verbindungen über Modems, ISDN oder DSL übernimmt das PPP-Protokoll die Kommunikation mit dem DHCP-Server des Internet-Providers. Der DHCP-Server antwortet mit den IP-Konfigurationsdaten: IP-Adresse, IP-Netzmaske, IP-Adresse des Default-Routers (Internet-Router), IP-Adresse des DNS-Servers.

Ein Vorteil von DHCP liegt in dem ökonomischen Umgang mit registierten IP-Adressen. Sie werden nur aktiviert, wenn tatsächlich eine Kommunikation stattfinden soll. Ein Internet-Service-Provider hat in der Regel mehr Kunden als IP-Adressen in seinem DHCP-Pool verfügbar sind. Aber nicht alle Clients sind gleichzeitig im Internet!

Auch in LANs, die nicht notwendigerweise mit dem Internet verbunden sind, kann man DHCP einsetzen: Durch die „automatische“ IP-Konfiguration vermeidet man Probleme, die durch von Nutzern selbst konfigurierte Netzwerkparametern entstehen können, z. B. doppelte IP-Adressvergabe.

Typisch für den Einsatz von DHCP ist, dass nach jedem neuen Booten oder Verbindungsaufbau zum ISP eine andere IP-Adresse aus dem Pool zugewiesen wird. Das kann in LANs auch von Nachteil sein, da die IP-Adresse nicht mehr einem bestimmten Host zuzuordnen ist. In speziellen Fällen lässt sich DHCP auch so betreiben, dass einem bestimmten (Ethernet-) Interface immer die gleiche IP-Adresse zugewiesen wird.

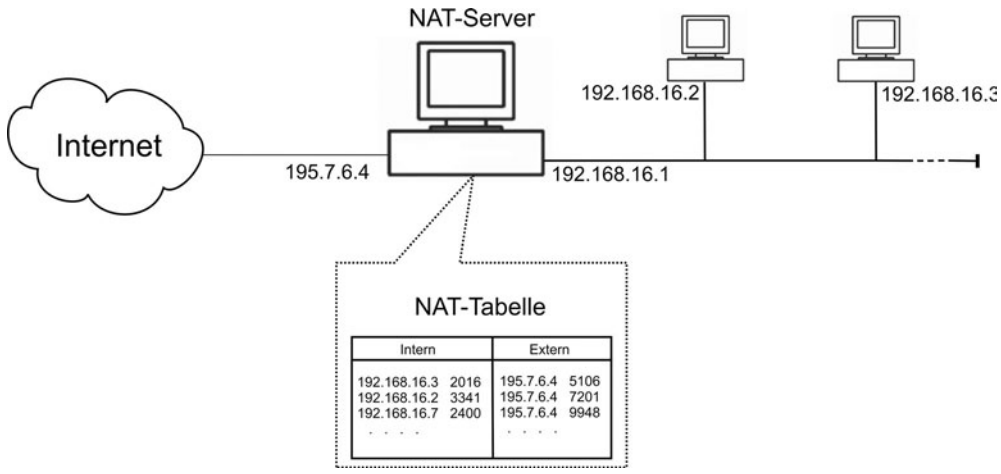
18.4.4 NAT

Die „*Network Address Translation*“-(NAT-)Technik wurde mit dem Ziel entwickelt, den Adressmangel des Internet zu entschärfen. Sie hat sich heute zu einem wichtigen Standard des Internet-Zugangs für LANs entwickelt.

Prinzipiell benötigt jeder mit dem Internet kommunizierender Rechner eine registrierte IP-Adresse.

Ein NAT-Gateway (NAT-Server) verbindet ein internes LAN, welches mit privaten Adressen konfiguriert ist, mit dem Internet. Sein Netzinterface zum Internet besitzt eine (einzige) offizielle IP-Adresse, das interne Netzinterface gehört zum Adressraum des privaten Netzes. Die Verbindung zum Internet geschieht **nicht** durch eine Routerfunktionalität (das wäre unzulässig!), sondern durch eine Socket-Substitution, also Ersetzung von IP-Adresse und Portnummer. Möchte ein interner Rechner einen Internet-Host erreichen, ersetzt das NAT-Gateway im IP-Paket vor dem Weiterleiten die Quelladresse durch seine eigene, offizielle IP-Adresse. Auch der Quellport wird durch einen neuen Port aus einem speziellen Bereich, meist oberhalb 5000, ersetzt. Der Internet-Host antwortet nun nicht direkt dem Quellrechner, sondern dem NAT-Gateway auf diesem neuen Port. Der NAT-Gateway erkennt an der Zielportnummer, dass das eintreffende IP-Paket nicht für ihn, sondern für den ursprünglichen Quellrechner aus dem internen Netz bestimmt ist. Es ersetzt daher die Ziel-IP (seine eigene IP) und den Zielport durch die ursprüngliche IP-Adresse und Portnummer des internen Rechners und leitet das Paket weiter ins interne Netz. Der interne Rechner bemerkt diese Socketumsetzung gar nicht, er hat den Eindruck, direkt mit dem Internet-Host zu kommunizieren. Der Internet-Host weiß nichts von der Existenz eines internen LANs. Aus seiner Sicht läuft die Kommunikation nur mit dem NAT-Server ab.

Ein NAT-Server kann mehrere Verbindungen aus dem internen Netz zum Internet verwalten. Er legt interne Tabellen an für die Umsetzung: interne IP, interne Portnummer ↔ externe IP des NAT-Servers, neue externe Portnummer. Dabei handelt es sich jeweils um die Quellports (Port des Absenders). Diese Umsetzung wird auch als *Masquarading* bezeichnet.



Beispiel: Internet-Verbindung über NAT-Server

Somit kann ein ganzes internes Netz mit z. B. 200 Rechnern über eine einzige registrierte IP-Adresse mit dem Internet kommunizieren. Voraussetzung ist jedoch, dass diese Kommunikation vom internen Netz aus initiiert wird! Ein Internet-Host kann keine Verbindung zu einem

internen Rechner beginnen, da das NAT-Gateway keinen Tabelleneintrag angelegt hat. Gerade diese Eigenschaft der NAT-Technik, die Nicht-Erreichbarkeit und das „Verbergen“ ganzer Netze vor dem Internet, ist oft aus Sicherheitsgründen erwünscht. NAT bzw. Masquarading wird daher häufig in Firewalls benutzt.

Port-Forwarding:
Sollen einzelne Rechner im privaten Netz als Server arbeiten, müssen sie vom Internet aus erreichbar sein, genauer: eine Kommunikation mit ihnen muss auch vom Internet aus initiiert werden können. Um dies zu ermöglichen, sind die Internet-Zugangsrouter (DSL-Router) neben NAT zusätzlich mit der *Port-Forwarding* („Port-Weiterleitung“)-Technik ausgestattet. Bei einem aus dem Internet eintreffenden IP-Paket erkennt der NAT-Server an der Port-Nummer, z. B. Port 80, dass nicht er selbst, sondern der (Web-)Server im privaten Netz angesprochen werden soll. Er ersetzt daher die Zieladresse (bisher die des Routers) durch die IP-Adresse des Servers und leitet das Paket an den Server in das interne Netz weiter. Weiterzuleitende IP-Pakete erkennt der NAT-Server anhand einer konfigurierten Tabelle: Port-Nummer ↔ private IP-Adresse. Typischer Anwendungsfälle für Port-Forwarding sind E-Mail- und Web-Server im privaten Netz, aber auch Computerspiele über das Internet, da diese oft eine Erreichbarkeit vom Spieleserver aus erfordern.

18.4.5 Firewalls

Ein Firewall („Brandschutzmauer“) ist eine Anordnung, die ähnlich wie ein Router oder Gateway eingesetzt wird, um ein privates Netz vor unberechtigten Zugriffen aus dem öffentlichen unsicheren Internet zu schützen. Es besitzt zwei Netzwerk-Interfaces und kontrolliert den Datenverkehr zwischen diesen Netzen.

Ein Firewall-System ist kein „plug-and-play“-Gerät, es ist je nach den geforderten allgemeinen Sicherheitsrichtlinien (*policies*) und speziellen Filterregeln (*rules*) mehr oder weniger aufwendig zu konfigurieren. Firewalls können auch aus mehreren Geräten bestehen.

Die NAT- bzw. Masquarading-Technik bietet bereits gute Firewall-Funktionalität, da das private Netz vom Internet aus nicht erreichbar ist und sein innerer Aufbau daher verborgen bleibt. In der Praxis hat die Firewall meistens jedoch „Löcher“ durch Port-Forwarding, was zu einem hohen Sicherheitsrisiko führt! Deshalb werden heute in der Regel flexible, den individuellen Bedürfnissen des Internet-Benutzers anpassbare Firewall-Systeme eingesetzt.

Es gibt zwei grundsätzlich unterschiedliche Arten von Firewalls:
- IP-Paketfilter Firewall
- Application Firewall (*Dual-Homed Gateway*)

IP-Filter Firewalls arbeiten auf Paketebene. Sie kontrollieren den Datenverkehr auf Grund von IP-Absenderadresse, -Zieladresse, Ports und Pakettyp (TCP, UDP, ICMP). Zu ihrer Realisierung dienen gewöhnliche IP-Router (Internet-Zugangsrouter), bei denen die Regeln, nach denen sie Pakete in und aus dem privaten Netz weiterleiten, in Form von *Access-Listen* eingegeben werden können. Beim Eintreffen eines IP-Pakets werden diese Regeln sequentiell abgearbeitet. Trifft eine Regel zu, wird diese ausgeführt und die weitere Prüfung abgebrochen. Es kommt also auf die Reihenfolge der konfigurierten Regeln an!

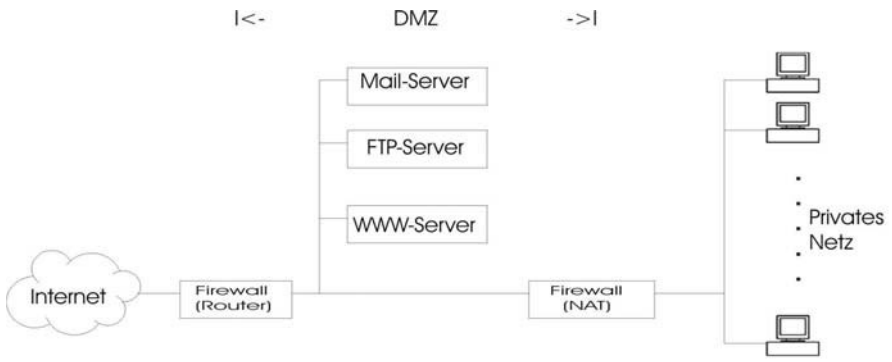
TCP-Verbindungen, die ja verbindungsorientiert arbeiten, lassen sich durch *statische* Paketfilter gut kontrollieren. Das erste TCP-Paket, das eine neue Verbindung initiiert, ist in seinem Header durch ein spezielles Bit, das sog. *SYN*-Bit (*Synchronize Bit)* gekennzeichnet. Folge- und Antwortpakete enthalten statt des SYN-Bit das *ACK*-Bit (*Acknowledge Bit*). Filterregeln beziehen diese Bits mit ein in ihre Entscheidungen und können so neue Verbindungsaufnahmen aus dem Internet erkennen und evtl. blockieren. UDP-Verbindungen arbeiten dagegen nicht verbin-

dungsorientiert und bieten nicht diese Entscheidungsmöglichkeit. Für sie sind die *dynamischen* Paketfilter (*stateful inspection*) geeigneter. Diese Firewalls speichern den Kontext und die Historie mit ab. So lässt sich kontrollieren, ob ein eintreffendes UDP-Paket die Antwort einer vorausgegangenen Kommunikation ist und ob das eingetroffene UDP-Paket auch den gewünschten Ziel-Socket anspricht. UDP-Pakete für im Kontextspeicher unbekannte Sockets können abgelehnt werden. Auch Regeln zur Behandlung von ICMP-Paketen (→ s. nächstes Kap.) sind möglich. Häufig bleiben „ping"-Anfragen deshalb erfolglos, weil Firewalls das ICMP-Protokoll blockieren.

Filter-Firewalls sind relativ einfach zu installieren, besitzen aber einige Nachteile: Bei komplexeren Installationen wird der Umfang der Regeln schnell unübersichtlich. Ferner sind die Protokollierungsmöglichkeiten von Routern stark eingeschränkt, so dass im Fall eines Einbruchs oder Einbruchsversuchs in das private Netz wenig Information über den Eindringling ermittelbar ist. Da Filter Firewalls immer noch einzelne Pfade über das Firewall zulassen, kann durch Manipulation der Datenpakete, z. B. durch Verändern der IP-Absenderadresse in eine zugelassene Adresse (*spoofing*), ein Eindringen in das private Netz nicht vollständig verhindert werden.

Einen höheren Schutz bieten Application Firewalls. Sie trennen das private Netz vollständig ab vom Internet. Grundsätzlich gibt es hier kein Routing über die Firewall. Benutzer des privaten Netzes, die mit dem Internet kommunizieren möchten, müssen sich zunächst auf dem Firewall einloggen und von dort aus ihre Internetsitzungen starten. Für die Außenwelt bleibt das private Netz vollständig verborgen. Durch *Proxy*-Dienste („Stellvertreter"-Dienste), die als Applikationen auf dem Firewall eingerichtet werden können, lässt sich für die Nutzer des privaten Netzes Transparenz bei Internetnutzungen erreichen: Der jeweilige Proxy-Server nimmt stellvertretend für den Nutzer automatisch die Verbindung auf zum Internet und reicht seinerseits die Daten weiter zum privaten Netzrechner. Jeder genutzte Dienst erfordert einen eigenen Proxy-Server. Da Application Firewalls ihre Kommunikation duplizieren, können sie alles mitprotokollieren.

Eine häufig benutzte Variante stellt ein Firewall mit *Demilitarisierter Zone* (DMZ) dar. Hier befinden sich die vom Internet erreichbaren Server in einer Pufferzone, der DMZ, die sowohl zum Internet als auch zum privaten Netz durch Paketfilter abgegrenzt sind. Die DMZ ist ein eigenes Subnetz und enthält die Server, die sowohl vom Internet als auch vom privaten Netz über Filter-Firewalls erreichbar sind. Die Ausgliederung der Server in eine DMZ hat neben dem zweifach zu überwindenden Paketfiltern den Vorteil, dass der vom Internet kommende Datenverkehr bei Serverzugriffen gar nicht in das private Netz vordringt, was die Sicherheit erhöht und den internen Datenverkehr reduziert.



Firewall-System mit DMZ

Firewalls lassen sich einstellen, ob sie abgewiesene Pakete einfach vernichten (*drop)* oder ob sie eine (ICMP-)Fehlermeldung wegen Unerreichbarkeit an den Absender zurücksenden *(reject)*. Die erste Möglichkeit ist die sicherere und übliche Variante, da hier der Absender nicht einmal erfährt, ob es überhaupt einen Verbindungspartner gibt!

18.5 Test- und Diagnoseprogramme

Gelegentliche Störungen bei der Netzwerk-Kommunikation sind leider unvermeidbar. Allgemeine Aussagen wie „es funktioniert einfach nicht!“ oder „das Netz spinnt!“ sind jedoch für Ingenieure und Naturwissenschaftler völlig unakzeptabel, da ein Bündel von Test- und Diagnoseprogrammen in den Betriebssystemen enthalten sind, mit denen die Fehlerursache eingegrenzt, analysiert und das Problem oft dann auch schnell behoben werden kann. Ferner liefern diese *Tools* auch viele Informationen über Aufbau und Zustand des Netzes. Sie sind daher nicht nur für den Netzwerkadministrator wichtig sondern auch für den „gewöhnlichen“ Benutzer interessant.

18.5.1 Das ICMP-Protocol

Das *Internet Control Message Protocol* (ICMP) ist speziell für Netzwerkdiagnosen und Fehlermeldungen ausgelegt. Es arbeitet als verbindungsloser Dienst auf der ISO-Schicht 3 direkt mit dem IP-Protocol zusammen. Da es relativ „tief“ im TCP/IP-Stack liegt, liefert es unabhängig von den auf höheren Schichten angesiedelten Anwendungen wichtige Informationen über die Erreichbarkeit eines IP-Knotens oder eines Dienstes im Netz.

ICMP-Meldungen zeigen dem Absender Informationen an über den Zustand des Netzes oder Probleme bei der Zustellung von IP-Paketen. Neben diesen bei Problemen automatisch generierten ICMP-Meldungen kann man auch gezielt eine ICMP-Anfrage (*ICMP-Request)* starten. Auf eine ausgesandte ICMP-Anfrage reagiert der Remote IP-Knoten mit einer entspechenden ICMP-Antwort (*ICMP-Reply).*

Die wichtigsten ICMP-Pakettypen:	
Protolltyp:	Bedeutung:
0	*Echo Reply* Als Antwort auf einen *Echo Request* (ping)
3	*Destination Unreachable* Netz ist nicht erreichbar Host nicht erreichbar Protokoll ist nicht erreichbar Port ist nicht erreichbar vorgeschriebene Route ist nicht erreichbar Paket müsste fragmentiert werden, *„Don't Fragment“*-Flag ist aber gesetzt
8	*Echo Request* ping-Anfrage
11	*Time-to-live Exeeded* Von Routern: TTL-Wert ist auf 0 abgesunken (→ s. Kap. 16.2), Paket wurde vernichtet

Es ist die Aufgabe der absendenden Anwendung, die Rückmeldungen auszuwerten und dem Nutzer entsprechende Meldungen auszugeben.

Die wichtigen Netzwerk-Diagnose-*Tools* „ping“ und „traceroute“ beruhen auf ICMP.

18.5.2 Testprogramme

Mit den nachfolgend vorgestellten Konsolenanwendungen erhält man Informationen über den Netzaufbau und seinen Zustand. Diese Programme werden hauptsächlich vom Netzwerkadministrator eingesetzt. Sie sind jedoch häufig auch für den „gewöhnlichen“ Benutzer zugänglich, der damit interessante Netzwerkinformationen abfragen kann.

Auf den Betriebssystemen verfügbare Netzwerk-Testprogramme

Anwendung:	*Windows:*	*Unix:*
Verbindungstest	ping	[/bin/]ping
Internet-Route	tracert	/usr/sbin/traceroute
Konfiguration	ipconfig	/sbin/ifconfig –oder- ethtool
ARP	arp	/sbin/arp
Routingtabelle	rout	/bin/route
DNS	nslookup	[/usr/bin/]nslookup
Netzzustand	netstat	[/bin/]netstat
User-Liste	finger	[/usr/bin/]finger

ping <hostadresse>

Verbindungstest auf Schicht 3; Die ausgegebene Laufzeiten zeigen, wie gut der angesprochene Host erreichbar ist (schnelle oder langsame Verbindung).

Wichtige Optionen:

Windows:	Unix(Linux):	
-t	-	:sendet fortlaufende ping-Signale; Abbruch mit CTRL/C
-l <Anzahl>	-s >Anzahl>	:Anzahl der gesendeten Bytes (default: 32 Bytes)
-i <TTL>	-t <TTL>	:TTL-Wert
-f	-M dont	:setzt das Flag *„don't fragment“*

Bei Verbindungsproblemen gibt ein „ping“ Auskunft darüber, ob der angesprochene IP-Knoten (Rechner, Router) eingeschaltet und am Netz ist.

ACHTUNG: Aus Sicherheitsgründen lassen Firewalls manchmal ICMP-Meldungen nicht zu, so dass ein erfolgloser Ping-Test zu faschen Schlüssen führt!

traceroute <hostadresse> bzw. **tracert** <hostadresse>

Es werden alle Router mit Namen und IP-Adresse angezeigt, die auf dem Weg vom eigenen Rechner bis zum Host durchlaufen werden. Bei Verbindungsproblemen erkennt man, an welchem Router die Verbindung abbricht oder wo das Netz überlastet ist.

traceroute ist eigentlich eine Folge von ping-Befehlen mit der Option „ping -i <TTL>“ bzw. „ping –t <TTL>“. Für TTL werden nacheinander die Werte 1,2,3 ... eingesetzt. Damit provoziert man ICMP-Fehlermeldungen vom Typ *Time-to-live Exceeded.* Der erste ping mit TTL=1 löst eine Rückmeldung vom eigenen Internet-Router aus, weil bei ihm TTL=0 gesetzt wird, die zweite Meldung kommt vom übergeordneten Router, usw.

Da auch traceroute auf ICMP beruht, kann ein Ausgabe-Abbruch auch von einem Firewall verursacht sein, der das ICMP-Protokoll nicht zulässt!

ipconfig bzw. **ifconfig**

Windows: Option */all.*

Konfiguration des lokal Host, Ethernetadresse, IP-Adresse, Netzwerkmaske;

Linux auch: ethtool <interface>, z. B. *ethtool eth0*

ethtool gestattet auch Setzungen des *network media,* z. B. 100 Mbit/s FULL. Bei Windows ist dies bei der „Erweiterten Konfiguration" des Ethernet-Interfaces möglich.

route

Windows*: route print*.

Ausgabe der lokalen Routing-Tabelle; Ausgabe der aktiven Routen und der zugehörigen Gateways (= Router).

Es lassen sich auch statische Routen eintragen, z. B. wenn das LAN neben dem Internet-Default-Router noch andere Router zu anderen LANs enthält. So wird anhand der Zieladresse am lokalen Rechner bereits entschieden, welcher Router angesprochen wird.

Wichtige Optionen:

add	<ziel>	<gateway>	:neue Route eintragen
delete	<ziel>	<gateway>	:Route löschen
change	<ziel>	<gateway>	:Route ändern

nslookup

Interaktive Abfrage von DNS-Servern; nslookup liefert die IP-Adresse von Domain-Adressen oder umgekehrt die Domain-Adressen zu IP-Adressen. Umfangreiche Setzungen werden mit *help* angezeigt. Interessante Setzungen vor der Abfrage sind z. B. (→ s. auch Kap. 18.2):

nslookup> *server* <serveradresse>	: primär auskunftgebenden Server wechseln;
nslookup> *set query=ns*	: Suche nach Name-Servern für eine Domain
nslookup> *set query=mx*	: Suche nach Mail-Servern für eine Domain
nslookup> *set query=any*	: Suche nach allen Rechnern einer Domain
nslookup> *-ls <domainname>*	: Auflistung aller angeschlossener Rechner der Domain (aus Sicherheitsgründen oft nicht zugelassen!)

netstat

Zeigt offene Sockets, Statistiken zu Protokollen und aktuellen Netzverbindungen an.

Wichtige Optionen:

Windows:	Unix(Linux):	
-	-	: Ausgabe der verbundenen Sockets
-r	-r	: Ausgabe der Routing-Tabelle
-a	-a	: Ausgabe aller „offener" Ports des Rechners (Serverports)
-s	-s	: Ausgabe einer ausführlichen Statistik für jedes Protokoll
-o	-p	: zu den Sockets wird auch die PID ausgegeben
	-e	: ausführliche Ethernet-Statistik

arp

Anzeige und Editieren des lokalen ARP-Cache, d. h. der Zuordnungstabelle IP-Adressen ↔ Ethernetadressen im lokalen Netzsegment. (→ s. auch Kap. 16.1).

Wichtige Optionen:

-a	: Anzeige des aktuellen ARP-Cache
-s <IP-Adr.> <Eth.-Adr.>	: Hinzufügen eines statischen(!) Tabelleneintrags, z. B.: *arp –s 192.168.18.12 00-04-76-da-a5-54*
-d <IP.-Adr.>	: Löschen des entsprechenden Tabelleneintrags

finger @<hostadresse> | <benutzername>@<hostadresse>
Gibt Informationen zu aktuell angemeldeten Benutzern auf dem Internet-Host aus. Wird „<benutzername>“ nicht angegeben, erhält man Informationen zu allen angemeldeten Benutzern. Aus Sicherheitsgründen ist der Finger-Server für Abfragen über das Netz häufig deaktiviert. Dann erhält der Client die Meldung: „*Connection refused“*.

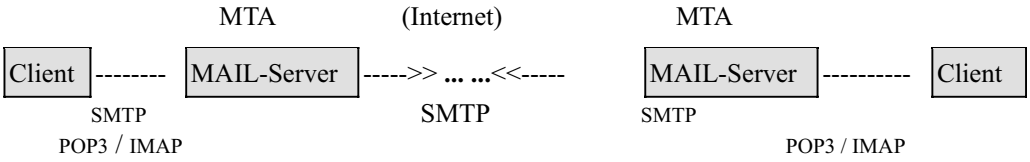
18.6 Wichtige Internet-Anwendungen

Da in TCP/IP-LANs und im Internet die gleichen Protokolle eingesetzt werden, sind natürlich die nachfolgenden „Internet-Anwendungen“ auch lokal, nur innerhalb eines LANs nutzbar. Konkret heißt das, dass der Kommunikationspartner sowohl ein Rechner des LAN als auch das Internet sein kann. Die meisten Anwendungen sind jedoch im Hinblick auf eine weltweite Kommunikation entwickelt worden.

18.6.1 E-Mail

Electronic Mail ist neben dem World Wide Web die am meisten genutzte Internet-Anwendung. Während noch vor ca. 10 Jahren die echte OSI-Anwendung *X.400 MHS (Message Handling System)* eine gewisse Rolle spielte, sind heute diese Systeme fast vollständig durch das Internet Mail-System *SMTP (Simple Mail Transfer Protocol)* auf TCP/IP-Basis ersetzt worden.

E-Mail ist eine „off-line“-Anwendung, d. h. der Absender und der Empfänger müssen nicht gleichzeitig im Netz aktiv sein, die Nachrichten werden auf Mailservern solange zwischengespeichert, bis der Empfänger die Nachricht abholt.



Aufbau eines E-Mail Systems

SMTP-Adressen haben die Form:

[vorname.]name@<domainname des mailservers>

z. B. fritz.willnich@umwelttechnik.fh-wiesbaden.de

Der Transport einer E-Mail über das Internet erfolgt mit dem SMTP-Protokoll von *MTA (Message Transfer Agent)* zu *MTA*, bis der Mail-Server des Empfängers erreicht ist.

Der Client setzt für das Absenden und Empfangen von Mails unterschiedliche Protokolle ein und muss entsprechend konfiguriert sein:

Beim *Absenden* einer E-Mail benutzt der Client das SMTP-Protokoll und kommuniziert mit dem SMTP Server-Prozess (Port 25) des Mail-Servers seines Unternehmens oder seines Internet Service Providers (ISP). Eine erfolgreiche Absendung einer Nachricht bestätigt zunächst nur wegen des verbindungsorientierten Protokolls eine erfolgreiche Entgegennahme der Mail vom Server, nicht aber die Existenz des Adressaten! Kann die Nachricht bei dem weiteren Transport nicht dem Empfänger zugestellt werden (Empfänger, Domäne oder Mail-Server gibt es nicht), generiert SMTP eine Fehlermeldung, die den Absender nach wenigen Sekunden in einer separaten Mail darüber informiert (*Mail delivery failed)*. Man tut also gut daran, kurz nach dem Absenden an unsichere Empfängeradressen den Eingang von Mails zu prüfen!

Der empfangene Mail-Server speichert eintreffende Nachrichten solange, bis der Client diese mit dem *POP3 (Post Office Protokoll V.3)*-Protokoll oder dem neueren *IMAP* (*Internet Message Access Protocol*) abholt, d. h. auf seine Arbeitsstation kopiert. Das Abholen von Mail-Servern ist Passwort-geschützt, d. h. der betreffende Benutzer muss einen E-Mail-Account auf dem Server besitzen. Mail-Clients bieten meistens ein Ordner-System an, um die Nachrichten übersichtlich zu speichern.

Häufig benutzte E-Mail Clients sind Mozilla Mail (Windows, Unix), Microsoft Outlook (Windows) oder elm (Unix).

Konfiguration von Mail-Clients	
SMTP-Server:	Mail-Server zum **Absenden** von Mails; Hier ist stets der ISP-Server einzutragen, dessen IP-Adresse benutzt wird. Die eigene E-Mail-Adresse ist unabhängig vom Provider!
POP3 oder IMAP-Server:	Mail-Server zum **Empfang** von Mails. Entspricht der eigenen E-Mail-Adresse; unabhängig vom ISP; Passwort-geschützt.
SMTP-Server und POP3/IMAP-Server können unterschiedliche Hosts des Internet sein!	

Das Absenden einer Mail ist nicht Passwort-geschützt.

Den im Klartext geführten SMTP-Dialog zwischen einer sendenden Station und einem Mail-Server kann man mit Hilfe des Telnet-Protokolls sichtbar machen, wenn man (statt des üblichen Telnet-Ports 23) den SMTP-Port 25 mit in die Kommandozeile explizit aufnimmt:

>telnet <mailserver> 25

<mailserver> muss entweder der eigene Mailserver oder der zuständige Mailservers des Empfängers sein.

Beispiel (Antworten des Mailservers **fett** dargestellt):

>telnet nt3.rz.fh-wiesbaden.de 25
220 nt3.rz.fh-wiesbaden.de Microsoft ESMTP Mail Service, Version: 5.0.2195.2966

ready at Mon, 10 Apr 2006 14:16:34 +0200
HELO pluto.hokuspokus.de
250 pluto.hokuspokus.de Hello [217.80.77.74]
MAIL FROM:<diddy@hokuspokus.de>
250 2.1.0 diddy@hokuspokus.de....Sender OK
RCPT TO:<schwoch@r5.mnd.fh-wiesbaden.de>
250 2.1.5 schwoch@r5.mnd.fh-wiesbaden.de
DATA
354 Start mail input; end with <CRLF>.<CRLF>

Diese Mail wurde ohne Mail-Client-SW geschrieben.
Erstaunlicherweise ist das möglich!

250 2.6.0 NT3XfyY0QUUBPcZi71m0000000@nt3.rz.fh-wiesbaden.de Queued mail for delivery
QUIT
221 2.0.0 nt3.rz.fh-wiesbaden.de Service closing transmission channel

Der MIME-Standard

Ursprünglich durfte eine E-Mail nur aus 7-Bit-ASCII-Zeichen bestehen, d. h. es konnten weder Texte mit Sonderzeichen (z. B. Umlaute) noch binäre Dateien in Form von „Anhängen“ (*Attachements*) an die Nachricht angehängt werden. Beides wird heute jedoch intensiv in der Praxis genutzt. Ermöglicht wird dies durch den MIME Standard (*Multipurpose Internet Mail Extension*), der die Daten speziell behandelt und den Header einer Mail um einige zusätzliche Zeilen erweitert.

Die MIME-Konvention wird ebenfalls bei der WEB-Kommunikation angewandt und hat sich generell zu einem Standard entwickelt, nach dem im Internet Daten ausgetauscht werden.

Die MIME Header-Zeilen kann man sich bei einer Mail mit anzeigen lassen. Die wichtigste Angabe ist der *Content-Type*, also die Beschreibung des Mail-Inhalts:

MIME Content-Type	*Mail besteht aus ...*
text	reinem Text, z. B. text/plain (ASCII) oder text/html
multipart	mehreren Teilen
application	Daten einer Applikation, z. B. application/msword
image	Bilddaten, z. B. image/gif oder image/jpeg
audio	Audiodaten, z. B. audio/wav
video	Videosequenzen, z. B. video/mpeg

Die Mail-Daten werden nach einem Verfahren behandelt, das in der Header-Zeile *Content-Transfer-Encoding* angegeben wird:

MIME Content-Transfer-Encoding	*Daten bestehen aus ...*
7Bit	7 Bit ASCII, keine Behandlung, Standard
quoted-printable	8 Bit Text mit Sonderzeichen, Angabe der Codierungstabelle (Charset); Codierung der Sonderzeichen mit „=“<Codeposition>, z. B. µ: =B5
base64	Umcodierung mit base64-Code
8Bit	8 Bit Text
binary	Binärdatei

Base64-Umcodierung:
Binärdateien als Anhänge werden meistens mit dem base64-Verfahren umcodiert, um eine 7 Bit-Textübertragung zu ermöglichen. Dabei werden jeweils die 24 Bits drei aufeinander folgender Bytes in vier Gruppen zu je 6 Bits neu organisiert. Jede 6-Bit-Gruppe wird durch ein ASCII-Zeichen repräsentiert, das dann als ASCII-Zeichen (zwar in acht Bit, jedoch mit MSB=0) übertragen wird.

Base64-Codierung							
6Bit-Wert	**Zeichen**	**6Bit-Wert**	**Zeichen**	**6Bit-Wert**	**Zeichen**	**6Bit-Wert**	**Zeichen**
000000	A	010000	Q	100000	g	110000	w
000001	B	010001	R	100001	h	110001	x
000010	C	010010	S	100010	i	110010	y
000011	D	010011	T	100011	j	110011	z
000100	E	010100	U	100100	k	110100	0
000101	F	010101	V	100101	l	110101	1
000110	G	010110	W	100110	m	110110	2
000111	H	010111	X	100111	n	110111	3
001000	I	011000	Y	101000	o	111000	4
001001	J	011001	Z	101001	p	111001	5
001010	K	011010	a	101010	q	111010	6
001011	L	011011	b	101011	r	111011	7
001100	M	011100	c	101100	s	111100	8
001101	N	011101	d	101101	t	111101	9
001110	O	011110	e	101110	u	111110	+
001111	P	011111	F	101111	V	111111	/

■ **Beispiel:**

Binärdaten: .... 01100011 01011100 01111010 00010100 11101110 10010101.....

6-Bit-Gruppen: ... 011000 110101 110001 111010 000101 001110 111010 010101...

übertragen wird: ... Y 1 x 6 F O 6 V ....

Durch die Umcodierung verlängert sich die Datei im Verhältnis 3:4.

Mailing-Listen und Listserver: E-Mail-Abonnements

Trotz boomender WWW-Euphorie sind Mailing-Listen immer noch stark nachgefragt. Das mag daran liegen, dass der Benutzer hier nicht die Informationen des Internet abfragen muss, sondern er erhält sie per E-Mail-Abonnement automatisch zugesandt.
Mailing-Listserver speichern Informationen zu bestimmten Themen, die automatisch an die Abonnenten der Liste verteilt werden. Wer sich in eine Liste zu einem bestimmten Thema eintragen möchte, muss sich auf dem Listserver „subscriben", d. h. er sendet eine Mail an

<servername>@<domainname>

mit der E-Mail Nachricht

subscribe <listenname> <eigener Name>

Nachdem man in die Liste aufgenommen wurde, erhält man Mails regelmäßig solange zugesandt, bis man sich mit einer weiteren Mail wieder „unsubscribed". ■

18.6.2 WWW

Das World Wide Web (WWW) ist die dominierende Anwendung des Internet und verantwortlich für die explosionsartige Zunahme der Netzbenutzung der letzten Jahre. Es wird sowohl von wissenschaftlichen Institutionen als auch – und in zunehmendem Maße – von kommerziellen Einrichtungen zur Präsentation und Werbezwecken genutzt. Für private Nutzer ist das WWW zu einer unverzichtbaren Informationsquelle geworden. Jedermann hat heute über seinen Internet Dienste-Anbieter (ISP) sogar die Möglichkeit, „der Welt" eine eigene „Home-Page" anzubieten.

WWW ist ein hyperlink-basiertes, multimediales Informationssystem in Client/Server-Architektur. Sowohl Server als auch Clients („WWW-Browser") sind für alle gängigen Betriebssysteme verfügbar. Die am Häufigsten anzutreffenden Browser sind Mozilla und Microsoft Internet Explorer. WWW-Browser sind fensterorientiert und erfordern eine Windows- oder X-Oberfläche.

WWW kommuniziert mit dem eigenen höheren Protokoll HTTP (*HyperText Transfer Protocol)*, das 1989 am CERN, Genf entwickelt wurde. Die Ausgangsidee geht zurück auf Dokumenten-Recherchen im Internet: Während in gedruckten wissenschaftlichen Texten über Fußnoten oder angefügte Glossare zusätzliche Erklärungen angeboten werden, ermöglicht die EDV einen eleganteren Weg, indem Spezialbegriffe im Text durch hervorgehobene Darstellung kenntlich gemacht und angeklickt werden können. Damit wird ein „Link" zu dem „unterlegten" Dokument aufgebaut und dieses dargestellt. Auch dieses Dokument kann wieder besondere Terme als „hot spots" enthalten, so dass beim Anklicken weitere Links zu Hintergrunddokumenten aufgebauten werden. Es entsteht ein hierarchisch aufgebautes Informationssystem, dessen oberste Ebene die Startseite („Home Page", Portal) des WWW-Servers bildet. Die *HyperLinks* können zu Dateien des gleichen Rechners, aber ebenso zu allen anderen Web-Servern des weltweiten Internet führen, zu dem automatisch eine Verbindung hergestellt wird. Ferner muss es sich bei der angewählten Datei nicht um ein Text-Dokument handeln. Hinter den Links können auch *Hypermedia*-Komponenten wie Grafik-, Audio- oder Videodateien stehen, die im MIME-Standard übertragen und durch die Web-Browser „live" verarbeiten werden können.

Web-Dokumente werden im HTML-(*HyperText Markup Language*)Format erstellt. HTML ist eine Seitenbeschreibungssprache. Der im Klartext aufgebaute Quelltext eines Web-Dokumentes enthält Formatierungsanweisungen in Form von „Tags“, die bei der Darstellung in den Browsern zu der gewünschten Darstellung führen, selbst aber nicht sichtbar sind. Ein Tag steht in Klammern <...>, meistens gibt es ein „Start-Tag“ <*tag*> und ein „End-Tag“ </ *tag*>.

Beispiele für häufig vorkommende Tags in HTML-Dokumenten:

<HTML> ... </HTML>	Start und Ende der gesamten HTML-Datei
<HEAD> ... </HEAD>	klammert „Meta-Informationen“, wie Dokumententitel, Autor, usw.
<BODY> ... </BODY>	klammert den Text, der vom Browser dargestellt wird
<Hn> ... </Hn>	Angaben zur Schriftgröße
<P> ... </P>	klammert den Text, der vom Browser automatisch umgebrochen wird
<A HREF=”*URL”*> … </A>	Link auf ein weiteres HTML-Dokument
<A HREF=”*#Marke”*> … </A	Link auf eine Stelle im selben Dokument
<IMG SRC=“*URL“*>	Einfügen eines .gif oder .jpg-Bildes
<! *Text*>	Kommentar

Eine HTML-Datei ist in ihrer Grundform sehr einfach aufgebaut und kann mit jedem Texteditor geschrieben werden.

Beispiel einer einfachen HTML-Datei:

```
<HTML>
<HEAD>
<TITLE> Testseite </TITLE>
</HEAD>

<BODY>
<H2> <CENTER>Überschrift der Testseite </CENTER> </H2>
Dieser Text erscheint in Standardgrösse
<BR>  <! Umbruch>
dieser Text ist <B>fett</B>  <BR>
Mit diesem Link kommen Sie auf meine
<A HREF=“http://r5.mnd.fh-wiesbaden.de/home_max/“>Homepage
</BODY>
</HTML>
```

Ein Web-Browser stellt diese Seite so dar:

Überschrift der Testseite

Dieser Text erscheint in Standardgröße

Dieser Text ist **fett**

Mit diesem Link kommen Sie auf meine Homepage

Für komplexere Anwendungen nutzt man besser einen der vielen speziellen HTML-Editoren, die auf dem Markt (zum Teil kostenlos) verfügbar sind.

Ein Web-Server verwaltet die HTML-Dateien in Verzeichnissen unterhalb seines „*Document-Root-Directory“*. Ein Browser adressiert ein Dokument relativ zu diesem Verzeichnis als „URL“, einer standardisierten Form zur Adressierung im Internet:

Adressierung im WWW

URL (*Uniform Recource Locator*)

URL - Aufbau: <protokoll>://<hostadresse>[<lokaler pfad>/[dateiname]]

Beispiele:

http://www.tud-darmstadt.de/
http://r2.umwelttechnik.fh-wiesbaden.de/home_max/testseite.html
ftp://ames.arc.nasa.gov/pub/SPACE/
file://usr/home/emil/scratch/MASQUE.TXT
file://E:/supermodels/cindy.gif

Wenn die URL-Angabe nicht direkt ein HTML-Dokument anspricht (d. h. nicht die Form hat: http://.....html), sucht der Server nach einer Datei „index.html“ im entsprechenden Verzeichnis und überträgt diese als Einstiegsseite zum Browser. Liegt auch diese nicht vor, wird das Directory-Listing des betreffenden Verzeichnisses dargestellt.

Die letzten Beispiele zeigen, wie ein Browser auch dazu benutzt werden kann, mit anderen Protokollen umzugehen Mit „file://...“ kann man das lokale Dateisystem „durchbrowsen“. Da Browser einen eigenen Cache anlegen, ermöglicht dies auch „offline“ Dokumente früherer Sitzungen darzustellen.

Ebenso wie Mail ist eine WWW-Kommunikation textorientiert. HTML ist eine „*Content-based Markup Language“,* sie wird durch den MIME-Standard unterstützt. Der Dialog lässt sich mit Telnet sichtbar machen, wenn man den für HTTP zuständigen Port „80“ in der Kommandozeile mit angibt, z. B. (Antworten des Web-Servers **fett**):

```
>telnet  r5.mnd.fh-wiesbaden.de  80
Trying 195.72.101.117...
Connected to r5.mnd.fh-wiesbaden.de.
Escape character is '^]'.
```

```
GET /newpage/index.html/
<!-- written by Philipp Schneider ICQ# 104811385 -->
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
<META NAME="GENERATOR" CONTENT="Namo WebEditor v4.0">
<TITLE>MND: Allgemeine Informationen</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#000000">
<font face="Arial"><br>
… … …
<font face="Arial"> </font>
<p><font face="Arial">Zu den Aufgaben des übergreifenden Fachbereichs
</font><B><font face="Arial">MND Umwelttechnik</font></B><font
face="Arial">
<BR></font><B><font face="Arial">(M</font></B><font
face="Arial">athematik,</font><B><font face="Arial"> N</font></B><font
face="Arial">aturwissenschaften, </font><B><font
face="Arial">D</font></B><font face="Arial">atenverarbeitung,
… … …
</HTML>
```

Nach der Antwort des Servers ist die Verbindung beendet. Sie wird durch jede weitere Clientanfrage wieder neu aufgebaut.

Der dargestellte Dialog zeigt, dass die „Intelligenz“ einer Web-Kommunikation vor allem auf der Clientseite in den Browsern steckt.

Um das Internet als Informationsquelle sinnvoll nutzen zu können, muss man wissen, wo welche Daten zu finden sind. Da es keine „ordnende“ Instanz im Internet gibt, kann das Aufspüren der interessanten URLs zu einer sehr zeitraubenden Beschäftigung werden. Hier bieten die ständig verbesserten „Suchmaschinen“ gute Dienste. Sie durchsuchen das WWW, häufig auch FTP- und News-Server nach Schlagworten und speichern diese gemeinsam mit den URLs in Datenbanken ab. Der so entstehende Index kann von Browsern durch die Eingabe von Suchworten abgefragt werden. Der Benutzer erhält als Suchergebnis direkt anklickbare Links zu den URLs, die den Suchbegriff enthalten.

Für die Weiterentwicklung und Standardisierung von *Markup*-Sprachen ist das *World Wide Web Consortium* (W3C) zuständig. Erweiterte Versionen von HTML sind XML (*Extensible Markup Language*) und XHTML (*Extensible Hypertext Markup Language*). Speziell für den Einsatz auf Handys ist WML (*Wireless Markup Language*) entwickelt worden.

CGI-Scripte

Das passive Browsen in Hypertexten und die statische Darstellung von HTML-Dokumenten erhält durch die Einbindung von ablauffähigen Programmen ein dynamisches Verhalten. *CGI-Skript*e sind Programme, die auf dem WWW-Server ablaufen. Sie werden wie die HTML-

Dateien über URLs angesprochen und damit auf dem Server gestartet. Die Ein-Ausgaben dieser Programme werden auf die CGI-Schnittstelle (CGI: *Common Gateway Interface)* des WWW-Servers umgeleitet und zum WWW-Client übertragen. Die Eigenschaft von CGI-Skripten, in Abhängigkeit von Benutzereingaben „Seiten“ zur Laufzeit zu generieren, kann z. B. auch für Datenbankabfragen genutzt werden. Der Ausdruck „Skript“ rührt von den anfangs eingesetzten Perl-Skripten auf Unix-Rechnern her, als CGI-Skript kann jedoch jedes compilierte Programm (Pascal, C, C++, Java,...) oder zu interpretierendes Skript (Shell-Skripte, Batch-Programme,...) eingesetzt werden, das auf dem WWW-Server ablauffähig ist.

Ein WWW-Server besitzt ein (oder mehrere) Verzeichnisse, dessen Dateien als CGI-Scripte interpretiert und bei Aufruf dem CGI-Interface zugeführt werden. Meistens heißt dieses Verzeichnis *Document-Root-Directory*/cgi-bin/. Die Scripte werden wie gewöhnliche html-Dokumente in den URLs angegeben.

Demonstration eines sehr einfachen CGI-Scripts „serverzeit“ auf einem WWW-Unix-Server:

```
#!/bin/sh
echo  'Content-type: text/html'
echo
echo  '<html>'
echo  '<body>'
echo  '<center> <h1>'
echo  'Die aktuelle Uhrzeit auf dem WWW-Server ist:'
echo  '<br>'
date
echo  '</h1> </center>'
echo  '</body>'
echo  '</html>'
```

Ein Aufruf im Browser mit der URL „http://r5.mnd.fh-wiesbaden.de/cbi-bin/serverzeit“ ergibt das Bild:

Die aktuelle Uhrzeit auf dem WWW-Server ist:
Fri Apr 28 19:45:39 CEST 2006

Natürlich wird bei jedem Anklicken von „Neu laden“ die Uhrzeit aktualisiert.

Applets
Im Gegensatz zu CGI-Skripten laufen *Applets* nicht auf dem Server, sondern auf dem WWW-Client ab. Dies hat den Vorteil, dass der von vielen Anwendern angesprochene Server entlastet wird. Java-Applets sind kleine Programme, die in „Java“ geschrieben werden und nach ihrer Übersetzung wie binäre Bilddateien in HTML-Dokumente eingebettet werden. Java ist eine objektorientierte, C++-ähnliche Programmiersprache, die von Sun Microsystems entwickelt wurde und die auf den Client-Server-Betrieb im Internet zugeschnitten ist. Die besondere Eigenschaft von Java besteht darin, plattformunabhängigen Code zu erzeugen: Der Compiler

generiert nicht wie üblich direkten Maschinencode, sondern einen plattformunabhängigen Zwischencode, den sog. Bytecode, der zum Client übertragen und von den Browsern der jeweiligen Client-Plattformen interpretiert wird. Browser müssen auf die Verarbeitung von Java-Applets eingerichtet sein.

Durch „Applet-Tags“ wird eine Java-Anwendung in den HTML-Quelltext eingefügt:

■ **Beispiel:**

```
<html>
.....
<body>
<applet code="MeinApp.class" width=200 height=100>
</applet>
....
</html>
```

Hier ist *MeinApp.class* das Compilat des Java-Compilers auf dem Server. ■

Für Applets gelten besondere Sicherheitsregeln: Sie dürfen nicht auf das Dateisystem des Client zugreifen, keine Betriebssystemfunktionen aktivieren und ihrerseits keine Netzverbindungen (außer der zu ihrem Server) herstellen. Trotzdem dauert die Diskussion um die Sicherheit und den Ausschluss von Missbräuchen beim Einsatz von Applets an. So ist z. B. nicht auszuschließen, dass ein Browser ein Applet aktiviert hat, ohne dass der Benutzer überhaupt davon Kenntnis erhält!

Eine Alternative zu Java-Applets ist die von Microsoft im Internet-Explorer eingeführte System-Schnittstelle *Active X*. Im Gegensatz zu Java ist dieses System jedoch nicht plattformunabhängig.

Servlets
Java *Servlets* und *Java Server Pages* sind Java-Anwendungen, die mit WWW angesprochen werden und wie CGI-Skripte serverseitig ablaufen. Gegenüber CGI-Skripten bieten sie den Vorteil, dass sie erheblich schneller sind und erweiterte Funktionalität aufweisen. Zu ihrer Entwicklung sind spezielle Software-Pakete wie z. B. das „Java Servlet Developers Kit“ (kostenlos im Internet) erforderlich.

18.6.3 „Klassische“ Internet-Anwendungen

Zu dieser Gruppe gehören die ältesten Internet-Anwendungen:

- E-Mail (SMTP-Mail, Mailing-Listen) (→ s. Kap. 18.6.1)
- Telnet
- FTP (Anonymous FTP)
- News

Telnet
Syntax: telnet <hostadresse>
Beispiel: telnet r5.mnd.fh-wiesbaden.de
Telnet ermöglicht einen Dialogzugriff auf andere Rechner (Remote Login). Nach erfolgreicher Login-Prozedur arbeitet man am Remote System so, als wenn man lokal dort angemeldet wäre.

Man benutzt die CPU des Remote Rechners. Telnet arbeitet textorientiert. Auf Windows-Rechnern und NetWare-Servern kann man sich nicht mit Telnet einloggen, sie verfügen nicht über einen Telnet-Server-Prozess (Ausnahme: Windows XP). Typisch ist ein Telnet-Login an einem Unix-Host. Telnet ist eine sicherheitskritische Anwendung, da Passworte uncodiert übertragen werden. Es wird zunehmend ersetzt durch *SSH* (*Secure Shell* → s. Kap. 19.2.1).

Telnet wird häufig im lokalen Bereich benutzt, um sich „Wege zu ersparen".

FTP

Syntax: ftp <hostadresse>
Beispiel: ftp ux5.umwelttechnik.fh-wiesbaden.de

FTP (File Transfer Protocol) ist eine Anwendung, mit der Dateien zwischen verschiedenen Rechnern des Internet übertragen, d. h. kopiert werden können.

Nach erfolgreichem Verbindungsaufbau erwartet der FTP-Server-Prozess ein Login mit User/Passwort. Anschließend kann man sich mit „dir" oder „ls" Verzeichnisse am Remote Sy-stem ansehen oder mit „cd" das Verzeichnis wechseln. Mit

get <filename>

wird die genannte Datei in das aktuelle Verzeichnis des lokalen Rechners übertragen *(download).* FTP gestattet sowohl die Übertragung von ASCII-Dateien (Mode „ascii, Befehl: ascii) als auch von Binärdateien (Mode „bin", Befehl: bin). Es können auch Dateien zum Server übertragen werden (*upload*, Befehl: put <filename>). Die vielfältigen Optionen kann man sich am Besten mit den Hilfesystemen der Betriebssysteme anzeigen lassen (Windows: FTP>help; Unix: man ftp).

Da FTP einen Login-Prozess erfordert, müssen Sie am Remote System ein Account besitzen. Dies ist jedoch nicht immer der Fall. Häufig wird das „anonymous FTP" genutzt:

Anonymous FTP:

Viele Institutionen und praktisch alle Hochschulen bieten Informationen und Dateien über Anonymous FTP an: Der Benutzer meldet sich am Server mit dem Namen *Anonymous* an, als Passwort wird die eigene Mail-Adresse akzeptiert. Nach dem Login landet man meistens in einem Verzeichnis „pub", von dem aus in untergeordnete Verzeichnisse verschiedener Sachgebiete gewechselt werden kann. Die verfügbaren Dateibestände liegen meistens in komprimierter Form vor.

Hier finden Sie riesige Sammlungen von Public Domain Software und Informationen aller Art, z. B. Anti-Virus-Programme für alle Betriebssysteme, RFC-Dokumente oder „Tools" zur Datenarchivierung.

Achtung: Die meisten FTP-Server sind Unix-Maschinen, die zwischen Groß- und Kleinschreibung unterscheiden!

News:

Die Syntax ist Client-abhängig, meistens sind sie in den E-Mail-Systemen der Clienten integriert.

News -eine Client/Server-Anwendung auf Basis des *Network News Transfer Protocol* (NNTP)- ist ein weltweites Diskussionsmedium. Die News-Server werden in der Regel von den Rechenzentren der Hochschulen oder Instituten betrieben. Sie stehen in einem weltweiten Verbund. Auf News-Servern sind Artikel zu mehr als 15000 Themenbereichen hierarchisch gespeichert und als News-Gruppen abrufbar. Die Information einer News-Gruppe besteht aus den Diskussionsbeiträgen in Form von E-Mails über ein bestimmtes Thema. Jeder kann sich beteiligen, d. h. zu den aufgeworfenen Problemen Stellung nehmen (*posten)* oder aber sich damit begnügen, eine Diskussion nur „mitzulesen". News-Clients nennt man auch *News Reader.*

News sind ein außerordentlich nützliches Mittel, um bei speziellen Problemen schnell Kontakt zu Experten auf der ganzen Welt aufzubauen. Sie tragen wesentlich dazu bei, Fachwissen überall und jedermann verfügbar zu machen und fördern eine internationale wissenschaftliche Zusammenarbeit auf allen Sachgebieten.

Beispiele einiger News-Gruppen:

alt.books.stephen-king
de.comp.os.linux
petrus.cs.uni-magdeburg.demisc.health.diabetes
comp.protocols.tcp-ip.domains
suncom.rz.hu-berlin.defido.ger.kochen
paperboy.osi.comz-netz.rechtswesen.urteile.steuerrecht
rec.arts.startrek.info
comp.virus

Die Diskussionsgruppen besitzen ein sehr unterschiedliches Niveau, manche sind – wie wir es ja auch vom Medium Zeitungen/Zeitschriften kennen –, ausgesprochen „unflätig“ oder enthalten nur „Geschwafel“. Es kostet daher einige Zeit, für sich nützliche Gruppen herauszufinden. Einen ersten Einstieg für interessante Gruppen findet man oft über WWW-Suchanfragen zu Kernbegriffen.

Empfehlung: Verfolgen Sie einige News-Gruppen einige Zeit und studieren sie den Jargon, bevor Sie eigene Beiträge „posten“ oder gar selber eine Gruppe aufmachen. Die *Netiquette* spielt eine große Rolle bei der internationalen Kommunikation. Bevor Sie ein Problem zur Diskussion einbringen, suchen Sie in den oft vorhandenen FAQ-Dokumenten (*Frequently Asked Questions*), ob das Problem dort bereits behandelt und gelöst wurde. Wiederholte Anfragen zu ein und demselben Thema sind „nervig“.

News stellen eine Alternative zum „Subcriben in Listen“ dar. News sind aber passiv, d. h. der Benutzer muss die Informationen von dem News-Server explizit herbeiholen, er erhält sie nicht automatisch als Mails zugesandt.

18.7 Aufgaben

1) Ermitteln Sie die Ethernet-Adresse und die IP-Konfiguration Ihres Rechners.
2) Sind die Rechner *195.72.101.115, altavista.digital.com, 192.168.43.10* am Netz?
3) Ermitteln Sie von Ihrem Rechner aus die Ethernet-Adresse des default-Routers.
4) Ein Rechner sei konfiguriert mit der IP-Adresse 195.67.121.159 und der Netzwerkmaske 255.255.255.224.
   In wie viele Subnetze wurde das Class-C-Netz aufgeteilt? Nennen Sie eine IP-Adresse aus dem gleichen Class-C-Bereich, die jedoch NICHT zum gleichen Subnetz gehört.
5) Gibt es im Internet einen Rechner *www.hampelmann.de* ?
6) Ermitteln Sie die IP-Adresse des Hosts *r5.mnd.fh-wiesbaden.de.*
7) Betreibt die FH Wiesbaden einen (Anonymous) FTP-Server (ftp.fh-wiesbaden.de)? Die Uni Heidelberg?

8) Zu welchen Domains gehören die IP-Adressen *195.72.99.123, 62.153.159.123, 130.83.200.104* ?

9) Finden Sie die Nameserver zu folgenden Institutionen:
mnd.fh-wiesbaden.de, tu-muenchen.de, t-online.de
Welche Nameserver gibt es für Deutschland, Spanien, Italien?
Betreiben die Arabischen Emirate einen Nameserver?

   Gibt es Nameserver von Kiribati, Kokos Inseln, Weihnachtsinseln? (Länderkennungen im Anhang nachschauen.)
Ermitteln Sie die Root-Name-Server des Internet.

10) Wie heißen die Mail-Server von der Uni Bonn, FH Hamburg, T-Online, Microsoft? (Hilfe: *set q = mx* (Mail-Exchanger))

11) Welche Rechner werden von den nachfolgenden Institutionen am Internet betrieben? Versuchen Sie ein Listing auszugeben:

   Uni Würzburg, Uni Flensburg, FH Wiesbaden, Uni Hannover.

12) Lassen Sie sich die Verbindungswege zu den folgenden Internet Hosts ausgeben: *www.fh-wiesbaden.de, www.paris.fr, julian.uwo.ca, freebsd.cdrom.com.*

19 Sicherheit in Netzen

Der allgemeine Begriff „Sicherheit“ hat im EDV-Bereich unterschiedliche Bedeutungen. Sicherheit im Sinne von Zuverlässigkeit technischer Systeme lässt sich durch Redundanz, z. B. durch RAID-Festplattensysteme erhöhen. Sicherheit im Sinne von Schutz vor versehentlichem Löschen von Dateien kann z. B. durch regelmäßige Software-Backups erreicht werden. Sicherheit vor Virenbefall liefern Anti-Virenprogramme. Firewall-Systeme gewähren Sicherheit vor dem unberechtigten Eindringen in private Rechner. In diesem Kapitel geht es um Sicherheit im Sinne von

- Integrität (Unversehrtheit und Unverfälschtheit von Nachrichten)
- Vertraulichkeit (Geheimhaltung von Nachrichten)
- Authenzität (Identität der Kommunikationspartner)

Grundlage hierfür sind die Verschlüsselungstechniken (Kryptografie).

19.1 Grundlagen der Kryptografie

Der Übermittlung geheimer Nachrichten galt schon immer, auch weit vor der Erfindung von Computern, ein besonderes Interesse. In der heutigen Zeit der weltweiten Kommunikation über unsichere weltweite Rechnernetze kommt diesem Thema spezielle Bedeutung zu. Ein großer Teil unserer Kommunikation in Netzen, wie z. B Online-Shopping und -Banking (*E-Commerce*) wird erst durch die Verschlüsselung von Nachrichten ermöglicht.

Als versierte C++-Programmierer fällt es uns leicht, eine Nachricht selber so zu verschlüsseln, dass kein „Unberechtigter“ sie lesen kann. Wir könnten z. B. jedes in der Nachricht enthaltene Zeichen durch ein anderes ersetzen. Diese Umcodierungstabelle müssten wir dem Empfänger „zukommen lassen“, so dass er die Nachricht wieder entschlüsseln kann.

■ Ein einführendes Beispiel:

Unverschlüsselte Nachricht: *>>Dies ist eine geheime Nachricht!<<*

Das Verschlüsselungsprogramm:

```
char nachricht[]="Dies ist eine geheime Nachricht!";
// Verschlüsselung
for(int i=0; i<strlen(nachricht); i++)
   nachricht[i] += i%3 +1;
cout << nachricht << endl;
// Entschlüsselung
for(int i=0; i<strlen(nachricht); i++)
   nachricht[i] -= i%3 +1;
cout << nachricht << endl;
.....
```

Verschlüsselte Nachricht: *>>Ekht"ltv#fkqf"jfjhjoh!Pddjujeku#<<*

Entschlüsselte Nachricht: *>>Dies ist eine geheime Nachricht!<<*

Das Verschlüsselungsverfahren besteht in diesem Beispiel darin, jeweils 3 Zeichen nacheinander um 1, 2 oder 3 Positionen in der ASCII-Tabelle nach oben zu verschieben. Der Schlüssel besteht also aus der Kombination „123“. Er dient sowohl zum Verschlüsseln als auch zum Entschlüsseln. ■

Verschlüsselungen, die auf reiner Zeichensubstitution beruhen, wie im obigen Beispiel, sind nicht sehr sicher, da sie durch Analyse-Programme schnell entziffert werden können. In der Praxis nutzt man komplizierte mathematische Verfahren, die auf Primzahlenzerlegungen großer Zahlen beruhen und zu Schlüssellängen zwischen 32 Bit und 2048 Bit führen.

Für die im folgendem vorgestellten Verschlüsselungsverfahren gehen wir sehr allgemein davon aus, dass eine „Nachricht“ oder ein „Dokument“ sicher über das Netz vom Absender zum Empfänger transportiert werden soll. Hierbei kann es sich z. B. um eine E-Mail handeln, um einen Online-Dienst mit einem WEB-Server oder einen abgesicherten (Telnet-ähnlichen) Dialog mit einem Server.

19.1.1 Erweiterte Prüfsummen: Fingerprints

In Kap. 11.6 haben wir Methoden zur Absicherung von Datenblöcken gegen Übertragungsfehler durch BCC(*Block Check Character*)- oder FCS(*Frame Check Sequenz*)-Bytes vorgestellt. Diese Verfahren dienen hauptsächlich dazu, Fehler einzelner Bits zu entdecken und zu korrigieren, Mehrfachfehler bleiben oft unerkannt. Zur Integritätsprüfung (Unversehrtheit) ganzer Dokumente oder Dateien gegen Textmanipulationen sind diese Methoden daher nicht geeignet. Zu ihrer Absicherung werden sog. *Fingerprints* benutzt. Während BCC-Methoden einen Datenbestand auf nur ein oder zwei Byte „abbilden“, reduzieren Fingerprints die Daten weniger stark auf eine typische Länge von ca. 20 Byte. Fingerprints sind somit „erweiterte“ Prüfsummen.

Aus der Sicht der Kryptografie handelt es sich bei der Bildung von Fingerprints um eine Einweg-Verschlüsselung mit starker Kompression auf der Basis von mathematischen *Hash*-Funktionen. Aufgrund der Kompression ist eine Rückgewinnung der Originaldaten aus dem Fingerprint nicht mehr möglich. Allerdings erlaubt ein Vergleich eines 20-Byte-Fingerprints eines Dokumentes vor und nach einem Netztransport sehr zuverlässig, ob die Daten auf dem Transportweg verändert wurden.

Für den praktischen Einsatz müssen die Methoden zu Bildung von Fingerprints standardisiert sein. Ein Fingerprint eines bestimmten Hash-Algorithmus hat immer die gleiche Länge, unabhängig von dem Umfang der Ausgangsdaten. Sie werden stets in hexadezimaler Form dargestellt. Es kommen hauptsächlich zwei Methoden zum Einsatz:

Standardisierte Hash-Algorithmen zur Bildung von Fingerprints:	
SHA1 (*Secure Hash Algorithm No.1*):	feste Länge 20 Byte
MD5 (*Message Digest No.5*):	feste Länge 16 Byte

Diese Funktionen sind z. B. in den Web-Browsern integriert. Mit ihnen lassen sich kritische Daten wie Zertifikate und öffentliche Schlüssel des *Public Key* Verfahrens beim Aufbau abgesicherter Internet-Verbindungen verifizieren (s. u.). Banken veröffentlichen z. B. die Fingerprints zu ihren *Public Keys* auf ihren Homepages und ermöglichen so eine Überprüfung der Schlüsselübertragung. Auch beim Absichern vertraulicher Dokumente über digitale Signaturen werden Fingerprints eingesetzt.

Programme zur Berechnung der Prüfsummen einer Datei sind sowohl für SHA1 als auch für MD5 im Internet als EXE-Dateien frei verfügbar. So könnten Sie z. B. – unabhängig von unserem aktuellen Thema „Sicherheit in Netzwerken“ – ihren Datenbestand auf der Festplatte auf Integrität überprüfen, indem Sie regelmäßig Fingerprints berechnen lassen und mit früher erzeugten vergleichen.

■ **Beispiel:**

Berechnung von Fingerprints der Datei „KLAUSURNOTEN.DAT“:

```
sha1  KLAUSURNOTEN.DAT
Hash: 2271ba255093bd78a0736afdd5e3532d720c107c

Md5sum KLAUSURNOTEN.DAT
MD5 checksum of "H:\MD5\KLAUSURNOTEN.DAT" :
bf 11 8b 5e fa 8b cc 84 27 04 b1 3d 29 bb 9e e8
```

■

19.1.2 Symmetrische Verschlüsselungsverfahren

Ein Verschlüsselungsverfahren nennt man „symmetrisch“, wenn zum Verschlüsseln und Entschlüsseln der gleiche Schlüssel benutzt wird. Dies erscheint uns eigentlich als selbstverständlich, wenn wir an den praktischen Umgang mit einem Schlüssel denken. Dies ist jedoch nicht so beim *Public-Key*-Verfahren, das wir im nächsten Kapitel kennen lernen (und deshalb auch „asymmetrisch“ genannt wird). Bei symmetrischen Verfahren gibt es also nur *einen* Schlüssel.

Die mathematischen Algorithmen zur Erzeugung der Schlüssel sind genormt:

Die wichtigsten Symmetrischen Verschlüsselungsverfahren	
Mathematischer Algorithmus:	*Schlüssellänge:*
DES (*Data Encryption Standard*)	56 Bit
3DES (3fach DES)	168 Bit
RC2, RC4 (*Ron Rivest's Cipher No 2/4*)	128 Bit–1024 Bit
Blowfish	32 Bit–448 Bit
IDEA (*International Data Encryption Standard*)	128 Bit
AES (*Advanced Encryption Standard)*	128Bit–256Bit

Je länger ein Schlüssel ist, desto sicherer ist er, desto langsamer ist jedoch auch das Ver- und Entschlüsselungsverfahren!

Solange beide Kommunikationspartner im Besitz des Schlüssels sind und diesen geheim halten, können sie sicher über das unsichere Netz (Internet) kommunizieren.

Beim praktischen Einsatz symmetrischer Verfahren ergibt sich ein generelles Problem: Wie erhält mein Kommunikationspartner den von mir erzeugten Schlüssel? Er muss auf sicherem Weg zum Partner gelangen. Sicher wäre z. B. den Schlüsselaustausch bei einem persönlichen Treffen vorzunehmen. Da benutzte Schlüssel aus Sicherheitsgründen häufig geändert werden sollten, wäre ein regelmäßiges Treffen aber sehr aufwendig, wenn nicht sogar unmöglich!

Wegen dieser Problematik des Schlüsselaustausches werden symmetrische Verfahren kaum alleine eingesetzt. Das Problem des Schlüsselaustausches wird durch das *Public-Key*-Verfahren gelöst! Die symmetrische Verschlüsselung wird daher hauptsächlich in Kombination mit dem Verfahren der asymmetrischen Verschlüsselung in Hybridverfahren eingesetzt (s. u.).

19.1.3 Asymmetrische Verfahren: Das Public Key Verfahren

Es klingt zunächst unglaubwürdig: Zwei sich unbekannte Kommunikationspartner können mit Hilfe des asymmetrischen Verschlüsselungsverfahren sicher ihre Schlüssel austauschen und sicher über das Netz kommunizieren! Das nutzen wir praktisch (fast täglich) aus, wenn wir mit unseren WEB-Browsern Online-Banking durchführen oder im Internet „shoppen".

Die asymmetrische Verschlüsselung beruht auf den genialen Ideen von *Diffie* und *Hellmann*, nicht nur *einen*, sondern ein Schlüssel*paar* zu erzeugen und zu benutzen. Der eine Schlüssel dient nur zum Verschlüsseln. Er kann nicht wieder entschlüsseln, was er selber verschlüsselt hat! Dazu dient der zweite Schlüssel: nur er kann entschlüsseln, was vom anderen Schlüssel verschlüsselt wurde. Das Verfahren ist also „asymmetrisch".

Das asymmetrische Verfahren wurde von *Rivest, Shamir* und *Adelmann* zu dem nach ihnen benannten RSA-Verfahren weiterentwickelt. Die Sicherheit dieses mathematischen Verfahrens beruht auf der Tatsache, dass es bis heute nicht möglich ist, schnell sehr große Zahlen in ihre Primfaktoren zu zerlegen. Die beiden bei der Schlüsselerzeugung entstehenden Schlüssel sind mathematisch zwar voneinander abhängig, jedoch nicht voneinander herleitbar.

Für eine sichere Kommunikation werden die Schlüsselpaare beider Partner benötigt. Jeder Partner erzeugt zunächst sein eigenes Schlüsselpaar und veröffentlicht den einen, den anderen hält er bei sich geheim. Der veröffentlichte Schlüssel ist der *Public Key*, der private, geheime ist der *Private Key.* Möchte jemand dem (ihm nicht notwendig bekannten) Partner „X" eine vertrauliche Nachricht senden, verschlüssel er diese mit dem Public Key des Partrners, hier also mit „PUBLIC_X". Da „X" seinen Public Key veröffentlicht hat, z. B. per E-Mail, im WEB oder auf speziellen Public Key Servern, kann jeder diesen „öffentlichen" Schlüssel ohne Schwierigkeiten erhalten. Die übertragene Nachricht ist nur mit dem Private Key, dem nicht-öffentlichen, geheimen Schlüssel des Empfängers „PRIVATE_X" zu entschlüsseln. Niemand sonst als der Inhaber des Private Key von „X" kann die Nachricht entschlüsseln.

Das folgende Beispiel soll den grundsätzlichen Ablauf einer vertraulichen Datenübertragung nach dem Public Key Verfahren verdeutlichen:

■ **Beispiel:**

Tom möchte Susi eine geheime Nachricht (z. B. E-Mail) senden:

Tom		Susi	
<Schlüsselpaar generieren>		<Schüsselpaar generieren>	
T_PRIVAT	T_PUBLIC	S_PUBLIC	S_PRIVAT

Tom besorgt sich S_PUBLIC — Susi besorgt sich T_PUBIC

Nachrichten-Klartext:

Hallo Susi!
Es soll niemand wissen, dass....

Verschlüsselung mit S_PUBLIC:

ATZ'89fsd fgjgfhf57ghjj.....

Übertragung

---> ---> ---> eingetroffene Nachricht:

ATZ'89fsd fgjgfhf57ghjj.....

Entschlüsseln mit S_PRIVAT:

Hallo Susi!
Es soll niemand wissen, dass....

Ebenso kann Susi Tom eine geheime Nachricht senden, die sie mit T_PUBLIC verschlüsselt, usw.

Da nur Susi Inhaber von S_PRIVAT ist, kann auch nur sie die Nachricht decodieren.

■

Das RSA-Verfahren ist solange sicher, wie die nicht-öffentlichen Private Keys geheim gehalten werden!

Das Public Key Verschlüsselungsverfahren (RSA-Verfahren)

löst das Problem des Schlüsselaustausches! Absender: verschlüsseln mit dem veröffentlichten Public Key des Partners; Empfänger: entschlüsseln mit dem eigenen geheimen Private Key.

Das Verfahren hat jedoch noch Schwachstellen:

- Wie kann sichergestellt werden, dass die Nachricht nicht manipuliert wurde auf dem Transportweg vom Absender zum Empfänger?
- Jeder beliebige Nutzer könnte sich als „Tom“ ausgeben und die Nachricht mit dem öffentlichen Schlüssel des Partners versandt haben!

Die Antwort auf diese beiden Fragen sind digitale Signaturen.

19.1.4 Digitale Signaturen

Eine „Unterschrift“ unter einem Dokument gilt schlechthin als Nachweis dafür, dass das Dokument auch wirklich vom Unterzeichnenden stammt; denn Unterschriften gelten als einzigartig. Unter einem *digitalen* Dokument (Nachricht) ist auch nur eine *digitale* Unterschrift möglich: die digitale Signatur. Was ist am Absender einzigartig? Sein geheimer Schlüssel!

Eine digitale Signatur besteht aus einem mit dem geheimen Schlüssel des Absenders verschlüsselten Text.

Es macht nun wenig Sinn, die gesamte Nachricht mit dem eigenen geheimen Schlüssel zu verschlüsseln. Denn jeder könnte mit dem veröffentlichten öffentlichen Schlüssel die Nachricht entschlüsseln. Gelingt dies, ist man zwar sicher, dass die Nachricht auch wirklich vom Inhaber des geheimen Schlüssels stammt, aber die Vertraulichkeit wäre nicht gegeben!

Statt die gesamte Nachricht mit dem geheimen Schlüssel zu verschlüsseln, (was außerdem sehr zeitaufwendig wäre!) bildet der Absender einen Fingerprint seiner Nachricht, verschlüsselt nur diesen mit seinem geheimen Schlüssel, „signiert“ („unterschreibt“) damit sein nach Kap. 19.1.2 verschlüsseltes Dokument und sendet es ab.

Der Empfänger dekodiert die Nachricht mit seinem geheimen Schlüssel und dekodiert die Signatur mit dem öffentlichen Schlüssel des Absenders. Er bildet seinerseits den Fingerprint aus dem dekodierten Nachrichten-Klartext (gemäß SHA1 oder MD5) und vergleicht. Ist der neu erzeugte Fingerprint mit dem aus der Signatur extrahierten gleich, wurde die Nachricht seit der Signierung nicht verändert und die Nachricht stammt vom angegebenen Absender.

Digitale Signaturen

gewährleisten die Integrität der Nachricht und die Authentizität des Absenders. Signatur: Fingerprint der Nachricht mit eigenem geheimen Private Key verschlüsseln und an Nachricht anhängen.

19.1.5 Zertifikate

Das Public-Key-Verfahren beruht auf der Annahme, dass der veröffentlichte Public Key auch wirklich derjenigen Person oder demjenigen (Bank-)Server gehört, die bzw. der bei der Veröffentlichung des Keys angegeben wird. Dies ist aber in unseren bisherigen Betrachtungen nicht sichergestellt. Die *vertrauenswürdige* Verteilung und Verwaltung öffentlicher Schlüssel erweist sich als nicht so einfach!

Das Problem der Kopplung eines öffentlichen Schlüssels an eine bestimmte Person (oder Rechner) wird durch Zertifikate gelöst.

Zertifikate

garantieren, dass ein bestimmter veröffentlichter Public Key auch wirklich zu einer bestimmten Person, einer bestimmten Institution oder einem bestimmten Rechner gehört.

Sofern nur eine überschaubare abgeschlossene Nutzergemeinschaft wie z. B. kleinere Unternehmen oder Hochschulen „sicher“ untereinander kommunizieren wollen, genügt es, dass alle Benutzer des Unternehmens oder Hochschule eine Stelle als eine vertrauenswürdige Einrichtung akzeptieren. Diese Stelle arbeitet als Zertifizierungsstelle (*Certification Authority:* CA oder *Tust Center:* TC). Die Benutzer hinterlegen bei der CA gegen Vorlage ihres Personalausweises ihren eigenen öffentlichen Schlüssel und erhalten den öffentlichen Schlüssel der CA

persönlich ausgehändigt. Die CA erstellt ein Zertifikat: Es enthält die persönlichen Daten des Antragstellers und dessen öffentlichen Schlüssel. Das Zertifikat wird mit dem geheimen Schlüssel der CA signiert und veröffentlicht.

Zertifikat

Persönliche Daten:	- oder -	*Rechnername:*
Name:.....		Domain-Name:.....
Vorname:.....		Institution/Abteilung:.....
Ausweis-Nr:.....		.....
.....		.....

Public Key des Antragstellers:

1f:7e:ab:51:96 . . . . . .

Getungsdauer des Zertifukats bis:. . . . . .

Ausstellende CA:. . . . .

Signatur der ausstellenden CA:

1F:68:E9:a7 . . . . . .

Die Kommunikationspartner tauschen nun nicht untereinander ihre öffentlichen Schlüssel aus, sondern erhalten von dritter Stelle, der CA das Zertifikat. Beide Kommunikationspartner vertrauen ja dem öffentlichen Schlüssel dieser CA, so dass jeder die Signatur dekodieren und somit Integrität des Zertifikats und Identität des (evtl. auch unbekannten) Partners feststellen kann. Nun sind sie im Besitz des öffentlichen Schlüssels des Partners und können sicher mit dem RSA-Verfahren kommunizieren.

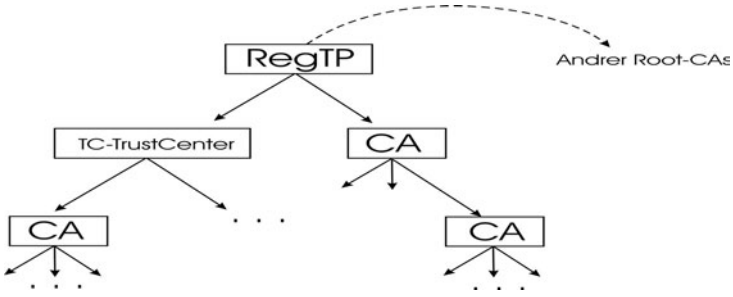
Einer Certification Authority (Trust Center) zu vertrauen bedeutet, deren öffentlichen Schlüssel zu akzeptieren!

Soll die „sichere“ Kommunikation ausgedehnt werden auf z. B. Online-Dienste oder E-Mail-Verkehr mit fremden Partnern, so ist die einzelne Zertifizierungsstelle der eigenen Firma oder Hochschule nicht mehr „zuständig“, es müssen andere CAs und Trust Center mit einbezogen werden. Unser Browser oder E-Mail-System wird aber nicht ohne weiteres einer fremden CA vertrauen, es sei denn, diese fremde Zertifizierungsstelle hat sich bei einer übergeordneten Zertifizierungsstelle „zertifiziert“, der wir vertrauen. Es entsteht so eine *Public-Key-Infrastruktur* (PKI), die aus einer Hierarchie von CAs aufgebaut ist und an deren Spitze eine „Wurzel-Zertifizierunsstelle“ (*Root Certification Authority*) alle untergeordneten Trust Center überwacht. Dies ist in Deutschland die Bundesnetzagentur (früher: Regulierungsbehörde für Telekommunikation und Post), „RegTP“. Sie überwacht die in Deutschland zugelassenen Trust Center.

Eine Liste der in Deutschland akkreditierten Zertifizierungsdiensteanbieter findet man unter *www.bundesnetzagentur.de*.

Letzlich müssen wir einer Root-CA vertrauen, z. B. indem wir den im Web veröffentlichten Fingerprint ihres öffentlichen Schlüssels mit dem Fingerprint des auf unserem Rechner gespeicherten vergleichen! Manche Browser klettern die Hierarchie bis zum Wurzelzertifikat hoch,

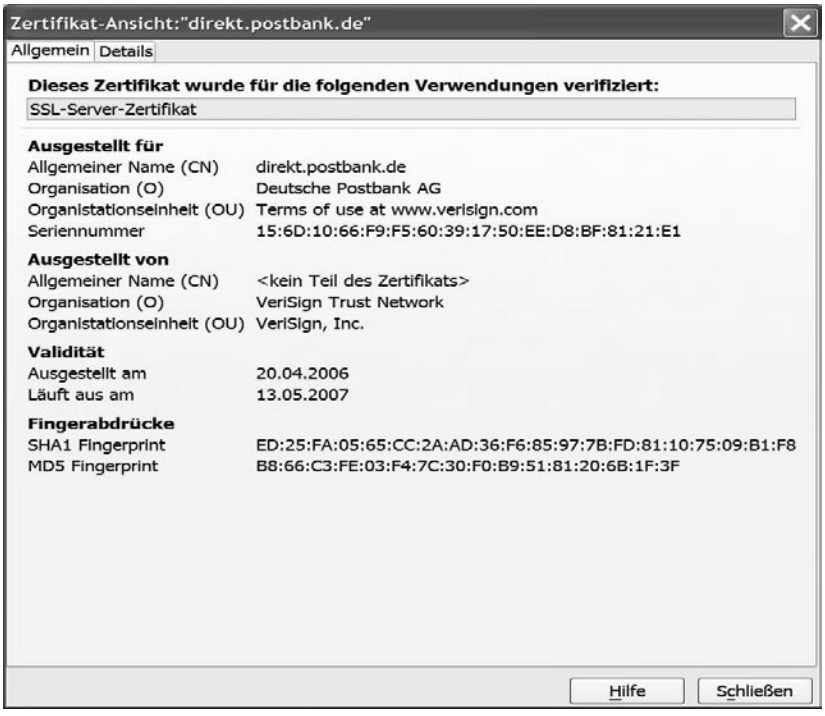
bevor sie eine gesicherte Kommunikation zulassen. Die meisten Internet-Browser und E-Mail-Systeme haben eine Reihe von vertrauenswürdigen (Root-)CAs und Trust Center vorinstalliert (z. B. TC TrustCenter, VeriSign...).



Trust Center Hierarchie

Ist erst einmal ein Partner als „sicher“ eingestuft, wird sein Zertifikat im Zertifikatsspeicher des PC gespeichert, er muss nicht jedes Mal zu Beginn einer Kommunikation neu geprüft werden. Genauer: Bis zum Ende der zeitlich begrenzten Gültigkeit des Zertifikats kann der öffentliche Schlüssel des Partners ungeprüft benutzt werden. Damit ein Zertifikat von allen Browsern und Mailsystemen verstanden wird, muss ihre Form normiert sein. Zertifikate benutzen das standardisierte X.509-Format der ITU in der Version 3 (X.509v3). Die Internet-Browser verwalten die Zertifikate unter den Einstellungen *Sicherheit und Zertifikate.* Dort kann man sich die gespeicherten Zertifikate von Verbindungspartnern und ihre signierenden CAs anzeigen lassen.

■ **Beispiel eines gespeicherten X.509 Zertifikats (Mozilla):**



Unter „Details“ kann man sich u. a. explizit den öffentlichen Schlüssel des Inhabers anzeigen lassen. ■

Die Zertifikatüberprüfung verläuft im Hintergrund und wird in der Regel vom Benutzer nicht bemerkt. Soll jedoch ein bisher unbekanntes Zertifikat installiert werden, öffnen die Browser oder Mailsysteme ein Dialogfenster und fragen nach, ob diesem neuen Zertifikat vertraut werden soll. Grundsätzlich sollte man sich darüber im Klaren sein, dass ein Zertifikat die Glaubwürdigkeit des Schlüssels besiegelt, nicht die des Besitzers!

Die Erstellung eines von einer CA herausgegebenen Zertifikats für eine Person oder Institution ist kostenpflichtig.

19.1.6 Hybride Verschlüsselungsverfahren

Das Problem des Schlüsselaustausches wird durch das oben beschriebene asymmetrische Verfahren (RSA-Verschlüsselung) gelöst. Allerdings haben asymmetrische Verschlüsselungsverfahren einen großen Nachteil: sie sind sehr rechenaufwendig und benötigen mehr als 1000 mal längere Rechenzeiten als symmetrische Verfahren! Während dieser Zeitbedarf bei E-Mail Anwendungen wenig ins Gewicht fällt, ist er bei Dialoganwendungen, z. B. Online-Banking, unzumutbar störend. Dialoganwendungen nutzen daher nicht reine RSA-Verfahren, sondern eine Kombination aus asymmetrischer und symmetrischer Verschlüsselung, sog. Hybridverfahren:

Die Kommunikation startet mit dem RSA-Verfahren, das zunächst einen sicheren Kommunikationskanal zum Partner aufbaut. Bevor nun die eigentlichen Nutzdaten ausgetauscht werden, generiert ein Verbindungspartner einen symmetrischen Schlüssel und der Dialog wird anschließend nach dem schnelleren symmetrischen Verfahren fortgesetzt. Welches der oben vorgestellten symmetrischen Verfahren eingesetzt wird, wird mit dem Verbindungspartner ausgehandelt. Der neue symmetrische „Sitzungsschlüssel“ (*Session Key*) wird asymmetrisch verschlüsselt an den Partner übermittelt. Die weitere Kommunikation nutzt ab diesem Zeitpunkt die symmetrische schnellere Verschlüsselung für den Dialog.

Hybridverfahren

nutzen das RSA-Verfahren zum Austausch eines symmetrischen Schlüssels. Nutzdatenverschlüsselung erfolgt mit einem symmetrischen „Einmal“-Sitzungsschlüssel.

Hybridverfahren erfüllen automatisch die Forderung nach häufigen Wechseln der symmetrischen Schlüssel, da der Sitzungsschlüssel am Dialogende wieder gelöscht wird.

19.2 Sicherheits-Dienste und -Anwendungen

Zusammenfassung: Die wichtigsten Sicherungsdienste/-anwendungen

Verfahren:	*hauptsächlich eingesetzt für:*
SSH	Dialog, Remote Login
SSL/TLS	Web-Browser, https
PGP	E-Mail
S/MIME	E-Mail
IPSec	VPN (*Virtual Private Network*)

19.2.1 SSH

SSH (*Secure Shell*) ist eine Schicht-7-Anwendung auf Port 22 und stammt aus der Unix-Welt. Es wird wie Telnet für Remote Logins genutzt, stellt aber eine abgesicherte Verbindung bereit. Während Telnet Benutzernamen und Passworte unverschlüsselt überträgt und deshalb leicht abgehört werden kann, werden bei SSH sowohl Anmeldedaten als auch die anschließenden Nutzerdaten nach dem Hybrid-Verschlüsselungsverfahren verschlüsselt. Man sollte deshalb grundsätzlich alle Telnet-Verbindungen durch SSH-Verbindungen ersetzen!

Kommando:

>ssh –l <login_name> <host_id>

Anschließend wird nach dem Passwort des am Server zugelassenen Benutzers <login_name> gefragt.

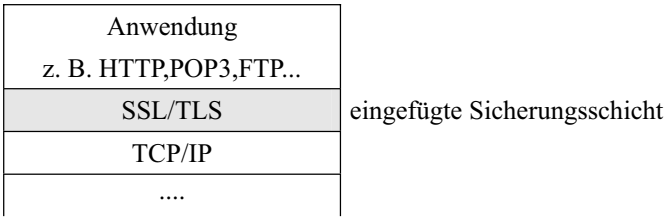
Bei der ersten Verbindungsaufnahme zwischen SSH-Client und SSH-Server erfolgt die Generierung der RSA-Schlüsselpaare und der Austausch der Public Keys. Der Client erhält eine Warnmeldung mit der Frage, ob er den Public Key des Servers akzeptieren will. Zusätzlich wird der Fingerprint des Public Key des Servers angezeigt. Will der Client ganz sicher gehen, so könnte er nun den Administrator des Servers anrufen und sich den Fingerprint des Servers vorlesen lassen und vergleichen. Akzeptiert der Client den Schlüssel, wird er unter den vertrauenswürdigen SSH-Hosts abgespeichert für alle weiteren Kontaktaufnahmen mit diesem Server, die dann ohne Warnung aufgenommen werden. Nun erfolgt der verschlüsselte Austausch eines Sitzungsschlüssels, mit dem dann alle weiteren Dialogdaten verschlüsselt werden.

Nach dem gleichen Verfahren arbeiten die Dateitransfer-Programme SFTP (*Secure* FTP) und SCP (*Secure* Copy) als Ersatz für die unsicheren Anwendungen FTP und RCP (*Remote Copy).*

SSH ist in Unix (Linux) im Lieferumfang enthalten, muss für Windows jedoch nach installiert werden. Kostenlose SSH-Clients für Windows sind im Internet verfügbar.

19.2.2 SSL/TLS und HTTPS

Internet-Browser arbeiten standardmäßig mit dem http-Protokoll. Dieses Protokoll überträgt die Anwenderdaten unverschlüsselt im Klartext. Niemand würde wohl unverschlüsselt seine Kreditkartennummer oder andere sensible Daten über das Netz übertragen! Netscape entwickelte deshalb ein zusätzliches Protokoll, das als eigene Schicht *Secure Socket Layer* (SSL) in den Protokollstack eingefügt wurde und kryptografische Verfahren einbezog. Die Weiterentwicklung zur SSL Version 3 ist mit dem international standardisierten Protokoll *Transport Layer Security* (TLS) praktisch identisch. Die zusätzlich eingefügte Sicherungsschicht liegt oberhalb der Transportprotokolle unter den Anwendungen:



Hauptaufgaben von SSL/TSL sind die Authentifizierung der Kommunikationspartner und die Zertifikat-basierende Verschlüsselung nach dem Hybridverfahren. Eine unter Anwendung von SSL/TLS aufgebaute http-Verbindung ist im Browser an dem angehängten „s“ erkennbar:

https (*http Secure*). Eine SSL-abgesicherte Verbindung wird zusätzlich im Browserfenster durch ein geschlossenes Schlosssymbol gekennzeichnet.

■ **Beispiel Online-Banking:**

Folgende Schritte laufen ab, wenn ein Web-Browser eine abgesicherte Verbindung zu einer Online-Bank aufnimmt:

- Eingabe der URL der Bank
- Web-Server der Bank sendet ein Serverzertifikat an den Web-Browser
- Web-Browser prüft die Authentizität, indem er die Zertifizierungsstelle, die das Zertifikat ausgestellt hat, als vertrauenswürdige CA in seinem Zertifikatsspeicher findet
- Web-Browser schlägt ein (oder mehr) symmetrische Verschlüsselungsverfahren vor
- Web-Server akzeptiert ein Verfahren
- Web-Browser generiert einen symmetrischen Sitzungsschlüssel, verschlüsselt diesen mit dem öffentlichen Schlüssel des Servers (aus Zertifikat extrahiert) und sendet ihn an den Server
- Nur die Bank kann mit ihrem geheimen Schlüssel den Sitzungsschlüssel dekodieren
- Wechsel von „http“ → „https“ und Anzeige eines geschlossenes Schlosssymbol im Web-Browser
- ID, PIN, TAN und Nutzdatenaustausch mit Sitzungsschlüssel verschlüsselt ■

Da SSL/TLS ein Schichten-Dienst ist und damit von der darüber angesiedelten Anwendung unabhängig ist, lassen sich neben HTTP auch andere Anwendungen wie z. B. SMTP, POP3, IMAP Telnet oder FTP über SSL absichern. Weiterhin wird es dazu genutzt, die Echtheit von über das Internet bezogener Software zu bestätigen.

19.2.3 Sichere E-Mail-Systeme

E-Mails werden im Internet offen als ASCII-Texte übertragen, so dass jeder mit geeigneter Netzsoftware (z. B. Etherreal) die Nachrichten im Netz mitlesen kann. Das Bedürfnis nach Vertraulichkeit und Authentizität der Kommunikationspartner führte zur Anwendung von kryptografischen Methoden. Die verbreitetsten E-Mail-Verschlüsselungssysteme sind *PGP* und *S/MIME*.

PGP (*Pretty Good Privacy*):

PGP ist ein E-Mail Software-Produkt, das die oben beschriebenen Verschlüsselungsalgorithmen (RSA, IDEA, MD5) und digitale Signaturen für einen abgesicherten Nachrichtenaustausch nutzt. Im Gegensatz zur hierarchischen CA-Struktur für X.509-Zertifikate können PGP-Nutzer ihre öffentlichen Schlüssel selber gegenseitig beglaubigen. Unternehmen, die PGP als Standard Mail-System einsetzen, betreiben oft sog. *PGP Key-Server* zur Verteilung der öffentlichen Schlüssel. Wegen der „flachen Vertrauensstruktur“ spricht man auch von einem *Web of Trust* (Geflecht gegenseitiger Vertrauensbeziehungen).

PGP ist für Windows und Linux für einen nicht-kommerziellen Einsatz kostenlos im Internet erhältlich.

S/MIME (*Secure MIME*):

Als Erweiterung des in Kap. 18.6.1 vorgestellten MIME-Standards ist S/MIME in den gängigen E-Mailsystemen enthalten. Die Verschlüsselung und Signierung von Nachrichten durch S/MIME greift grundsätzlich auf Zertifikate zurück. Die Kommunmikationspartner müssen

also erst bei einer von beiden Seiten als vertrauenswürdig angesehenen Zertifizierungsstelle ein persönliches Zertifikat erwerben und dies jeweils in ihren Mailsystemen installieren. Anschließend senden sich die Mail-Partner ihre öffentlichen Schlüssel gegenseitig zu. Die Umstellung von einer ungesicherten auf eine abgesicherte E-Mail-Kommunikation erfordert somit folgende Schritte bei beiden Partnern:

- Bei einer CA (kommerzielles TrustCenter oder Unternehmens-CA) ein persönliches Zertifikat erwerben/kaufen und in das E-Mailsystem importieren; der eigene private (geheime) Schlüssel wird mit einem „Krypto-Passwort“ geschützt;
- Im Browser der ausstellenden CA das Vertrauen erklären (Einstellung: „Zertifikate verwalten“);
- Eine signierte Mail an den Partner senden; an diese Mail wird vom System automatisch der eigene öffentliche Schlüssel angefügt;
- Bei Erhalt einer signierten Mail wird das Zertifikat des Absenders im Zertifikat-Speicher abgelegt; das Zertifikat ist vertrauenswürdig, da die ausstellende CA vertrauenswürdig ist.

Nun können diese Partner verschlüsselte und signierte Mails austauschen, solange die Zertifikate gültig sind.

Der eigentliche „Aufwand“ bei einer solchen Umstellung besteht darin, sein eigenes Zertifikat zu erwerben. Dies kann ein persönliches Erscheinen bei der CA erfordern oder z. B. über das PostIdent-Verfahren durch die Post abgewickelt werden.

Möchte man eine Nachricht an einen Partner verschlüsseln, dessen Zertifikat nicht im Zertifikats-Speicher vorliegt, erhält man eine Fehlermeldung: es liegt ja nicht der öffentliche Schlüssel des Partners vor, mit dem die Nachricht zu verschlüsseln wäre! Digital signieren kann man dagegen jede seiner ausgehenden Mails (stets mit dem privaten geheimen Schlüssel). Die Signatur kann jedoch nur von demjenigen überprüft werden, der den öffentlichen Schlüssel, also ein Zertifikat des Absenders besitzt.

19.2.4 IPSec

Im Gegensatz zu den oben vorgestellten Sicherheits-Verfahren zielt IPSec (*IP Secure*) nicht auf bestimmte Anwendungen. IPSec stellt einen Schichten-Dienst bereit, der allgemein eine Absicherung der Datenpakete auf der IP-Vermittlungsschicht, also noch unterhalb der Transportprotokolle TCP oder UDP ermöglicht. IPSec ist eine Erweiterung von IPv4 und ist fester Bestandteil des neuen IPv6-Standards.

Die Sicherheits-Architektur von IPSec besteht aus drei zusätzlichen Sicherungsprotokollen, in denen unterschiedliche kryptografischen Verfahren angewendet werden können:

- ISAKMP (*Internet Security Association Key Management Protocol*)
- AH (*Authentication Header*)
- ESP (*Encapsulating Security Payload*)

Die erste Phase eines Verbindungsaufbaus beginnt mit dem ISAKMP-Protokoll in der Aushandlung eines IKE-(*Internet Key Exchange*)Verfahrens und dem Austausch der öffentlichen Schlüssel. Dabei kommen vorwiegend Zertifikate zum Einsatz; es ist jedoch auch möglich, auf einen vorab übergebenen gemeinsamen Schlüssel (PSK-Verfahren: *pre shared key*) zurückzugreifen.

Das AH-Protokoll sichert die Authentizität von Absender- und Zieladressen auf der Basis von Hash-Prüfsummen, die in einem zusätzlichen „AH-Header“ enthalten sind. Die eigentliche Verschlüsselung der Nutzdaten geschieht durch das ESP-Protokoll. Beide Protokolle können gemeinsam oder einzeln eingesetzt werden.

IPSec unterstützt zwei Verschlüsselungsmodes:

IPSec-Modes

Mode	*Verschlüsselung*	*Praktischer Einsatz*
Transport-Mode:	Inhalte der TCP- oder UDP-Pakete verschlüsseln und mit original IP-Header absenden;	Host-zu-Host Verbindung
Tunnel-Mode:	Gesamtes IP-Paket verschlüsseln incl. Original IP-Header; neuen IP-Header voranstellen;	Gateway-zu-Gateway Verbindung; Gateway-zu-Client Verbindung; VPN

Da beim Transport-Mode der original IP-Header (die originale IP-Adresse) erhalten bleibt, verarbeiten die Router das Datenpaket so, als wenn es nicht verschlüsselt wäre. Die miteinander kommunizierenden Endgeräte müssen beide über IPSec-Installationen verfügen.

Beim Tunnel-Mode ist das gesamte IP-Paket „eingepackt“ als verschlüsselte Nutzdaten in einem neuen IP-Paket mit einem neuem IP-Header. Der neue IP-Header enthält nicht notwenigerweise die IP-Adressen der beteiligten Endgeräte, sondern lediglich die IP-Adressen der Router, die an den Tunnel-Enden als *Security Gateway* arbeiten. Diese packen das originale IP-Paket wieder aus und routen es im sicheren Firmennetz unverschlüsselt an den eigentlichen Adressaten weiter. Der Vorteil des Tunnel-Modes liegt darin, dass Manipulationen an den IP-Adressen der Endgeräte verhindert werden, da die Adressen auf dem Weg durch das Internet verborgen bleiben. Weiterhin muss nicht jedes Endgerät mit IPSec arbeiten, da die Gateways dies für alle LAN-Rechner übernehmen.

Der Haupteinsatz von IPSec liegt im Aufbau von *Virtual Private Networks* (VPN), die im nächsten Kapitel behandelt werden.

IPSec ist in den Betriebssystemen Windows und Linux enthalten.

19.3 Aufgaben

1) Rufen Sie in Ihrem Browser eine geschützte WebSite auf (*https*, z. B. Bankverbindung). Klicken Sie auf das Schlosssymbol und prüfen Sie die Sicherheitseinstellungen. Von wem wurde das Zertifikat ausgestellt? Welches Verschlüsselungsverfahren wird eingesetzt? Wie lange ist das Zertifikat noch gültig?

2) Das TC TrustCenter, Hamburg erstellt zu Demonstrationszwecken kostenlos „Demo Zertifikate“, die online bezogen werden können. Die Zertifikate gelten leider nur für kurze Zeit, etwa 30-90 Tage. Besorgen Sie sich vom TC TrustCenter unter *www.trustcenter.de* ein „class 0 Demo Zertifikat“ und installieren Sie dies in Ihrem Browser. Benutzen Sie das Zertifikat, um sich selber eine verschlüsselte und signierte E-Mail zu senden.

   Überreden Sie eine Freundin/einen Freund das Gleiche zu tun. Senden Sie sich verschlüsselte und signierte Mails zu.

3) Wie können Sie einer (Root-) CA das Vertrauen entziehen? Finden Sie unter „Sicherheitseinstellungen“ in Ihrem Browser die entsprechenden Einstellungen heraus.

4) Wie können Sie überprüfen, ob die vorinstallierten (Wurzel-)Zertifikate ihres Internet-Browsers nicht gefälscht sind?

20 Spezielle Netzwerkkonfigurationen

20.1 Intranets

Im Gegensatz zum offenen Internet bezeichnet man die privaten Netze von Unternehmen/Institutionen als *Intranet*, wenn sie nicht direkt aus dem Internet erreichbar sind. Intranets sind entweder gar nicht mit dem Internet verbunden oder durch Firewall-Systeme vom Internet abgetrennt. Sie basieren auf der Adaption und Inegration von im Internet verwendeten und erprobten Technologien und Kommunikationsprotokollen. Intranets verfügen häufig über einen internen Web-Server, der unternehmensinterne Informationen für alle Angehörige des Unternehmens anbietet, aber nicht von ausserhalb des Intranets zugänglich ist.

Intranets werden in der Regel mit privaten IP-Adressen betrieben.

20.2 Virtual Private Networks (VPN)

Das Internet bietet eine äußerst kostengünstige Plattform für eine weltweite Kommunikation von verteilten Unternehmens-Standorten. Während man früher teure Standleitungen zur Verbindung von Filialen und Außenstellen mieten musste, lassen sich heute Netzanbindungen durch Internetverbindungen erheblich billiger und flexibler realisieren.

Das Internet ist jedoch ein „unsicheres“ Netz. Folglich müssen kryptografische Verfahren eingesetzt werden. Die Technologie hierfür sind *Virtual Private Networks* (VPN).

VPNs nutzen überwiegend das IPSec-Sicherungsprotokoll zur Wahrung von Integrität, Vertraulichkeit und Authentizität (→ s. Kap. 19.2.4).

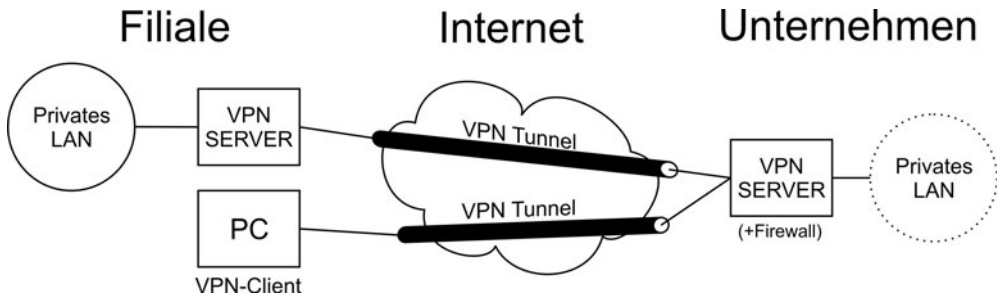
VPN

Nutzung von IPSec im Tunnel-Mode, zertifikat-basierend:
Aufbau eines sicheren Kommunikationskanals („Tunnel“) durch das unsichere Internet;
Einbindung von Außen-Rechnern (LANs, mobile Außendienst-Mitarbeiter, Heimarbeitsplätze) in das private Netz (Intranet) des Unternehmens,

Ein VPN-Tunnel kann aufgebaut werden zwischen

- zwei LANs, die je über einen VPN-Server miteinander verbunden sind;
- einem Einzelrechner (VPN-Client) und einem LAN mit VPN-Server;
- zwei Einzelrechnern (eher selten).

Als VPN-Server dienen meistens Firewall-Systeme oder Internet-Zugangsrouter mit zusätzlicher IPSec/VPN Funktionalität *(Security Gateway*). Die in das Intranet zu integrierende Rechner können entweder feste oder vom ISP temporär zugeteilte IP-Adressen besitzen.



VPN-Anbindung von LAN oder Einzelrechner

Auch der Einsatz von NAT für die Nutzung von privaten IP-Adressen ist mit VPN-Verbindungen möglich. Je nach der Lage des Tunnelendes sind hier 2 Konfigurationen zu unterscheiden:

a) VPN-Server ⇔ *Internet* ⇔ VPN-Server——NAT-Server——Privates Netz

b) VPN-Server ⇔ *Internet* ⇔ NAT-Server——Privates Netz
(NAT-T) |
VPN-Client (NAT-T)

Fall a) ist unproblematisch, da der VPN-Tunnel ohne NAT Beteiligung aufgebaut wird. Der Tunnel kann von allen Rechnern des privaten Netzes genutzt werden. Die VPN-Software läuft nur auf dem VPN-Server.

Im Fall b) ist die Situation komplizierter: Hier liegt der NAT-Server innerhalb des Tunnels, das Tunnelende also hinter einem NAT-Server. Jedes vom VPN-Client ausgehende IPSec-Paket erhält im NAT-Server eine neue IP-Quelladresse und einen neuen Quellport. IPSec enthält jedoch in seiner verschlüsselten Nutzlast die originale Quell-Adresse (→ s. Kap. 19.2.4, IPSec im Tunnel-Mode). Trifft dieses Paket nun beim VPN-Server ein, wird es verworfen, da die verschlüsselte Adresse nicht mit dem äußeren IP-Header übereinstimmt! Um dennoch eine Kommunikation zu ermöglichen, muss auf den VPN-Tunnelrechnern ein zusätzliches Protokoll aktiviert werden: *NAT-Traversal* (NAT-T). NAT-T verpackt die ausgehenden IPSec-Pakete des VPN-Client in UDP-Pakete (*UPD Encapsulation*) und sendet diese Pakete mit dem Zielport 500 an den VPN-Server ab. Auch dieses Paket erhält durch den NAT-Server eine neue Absender-Adresse, die jedoch nicht stört. Trifft dieses Paket beim VPN-Server auf Port 500 ein, wird es aus seiner UDP-Verpackung befreit und liegt nun als originales IPSec-Paket vor und kann normal weiter verarbeitet werden. Durch NAT wurde nur die „UDP-Verpackung“ modifiziert, nicht das originale IPSec-Paket!

NAT-T ist Bestandteil moderner VPN-Software.

Neben IPSec kann auch das von Microsoft entwickelte und in Windows enthaltene Protokoll PPTP (*Point To Point Tunnel Protocol*) für eine VPN-Verbindung benutzt werden. PPTP ist eine Ergänzung des oben vorgestellten PPP-Protokolls. Es baut eine verschlüsselte Verbindung zwischen VPN-Client und dem Einwählserver des ISP auf, der als VPN-Server arbeitet. PPTP ist damit weniger flexibel als IPSec, gilt auch als weniger sicher und wird hauptsächlich in reinen Windows-Umgebungen genutzt, bei denen sich einzelne Außenstationen direkt bei einem Windows-RAS-Server (*Remote Access Server*) einwählen.

20.3 Funk-Netze (WLAN)

Die große Verbreitung mobil einsetzbarer Notebooks hat zunehmend zum Aufbau von Funk-LANs, sog. „Wireless LANs“ (WLAN) geführt. Durch sie ist ein Notebook ohne zusätzliche Verkabelung problemlos einerseits in das Firmen- oder Hochschulnetz integrierbar, andererseits lässt sich zuhause im ganzen Haus oder von der Terrasse aus bequem das Internet erreichen. Öffentlichen Einrichtungen wie z. B. Flughäfen oder Hotels stellen in sog. „Hotspots“ WLAN-Funkzellen bereit, über die kostenlos oder gebührenpflichtig (per Accounting) Internet-Zugänge bereitgestellt werden.

WLAN-Interfaces sind in neueren Notebooks meistens bereits integriert oder mit USB-WLAN-Stick oder PCMCIA-Einsteckkarte aufrüstbar. Aber auch fest installierte Desktop-PCs oder Server lassen sich mit entsprechenden Einsteckkarten einfach WLAN-fähig machen.

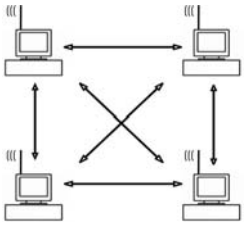
Im OSI-Schichtenmodell sind beim WLAN die unteren beiden Schichten neu organisiert:

Schichten 5-7	...
Schicht 4	TCP/UDP
Schicht 3	IP
Schicht 2	WLAN-Netzzugriff
Schicht 1	Luftschnittstelle

20.3.1 WLAN Architekturen

WLANs können in zwei Betriebsmodi arbeiten:

- Ad-hoc-Modus (Peer-to-Peer-Modus)
- Infrastruktur-Modus

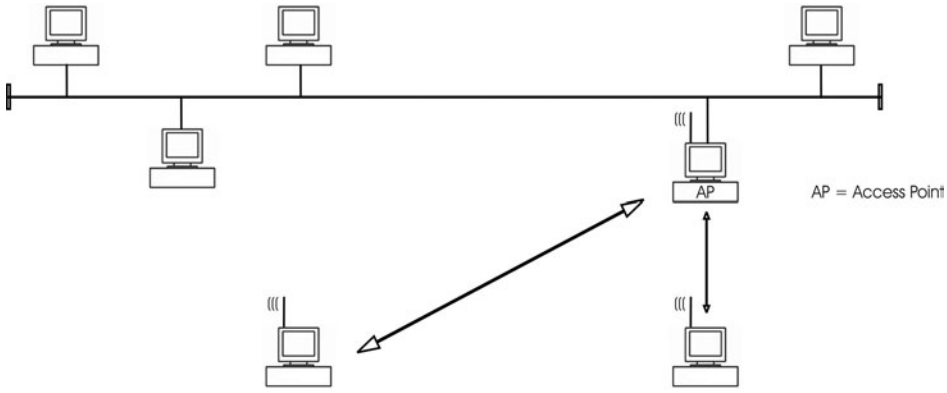


Ad-hoc-Modus

Im Ad-hoc-Modus sind alle Stationen gleichberechtigt, jeder kann mit jedem eine Verbindung aufnehmen. Es gibt keinen Zugang zu drahtgebundenen Netzsegmenten oder Internet. Diese Betriebsart wird nur selten eingesetzt und dient bei temporären Ereignissen wie Messen oder Konferenzen zum schnellen Datenaustausch.

In der Regel werden WLANs im Infrastruktur-Modus betrieben. Die WLAN-Clients nehmen über einen *Access-Point* (AP) Verbindung zum kabelgebundenen LAN und/oder Internet auf. Ein Zugang über APs ermöglicht einen direkten Zugriff auf das angeschlossene private Netzwerk unter Umgehung evtl. eingerichteter Firewall-Systeme. Ein AP verfügt daher über eigene Sicherheitsfunktionen (s.u.).

Während größere Unternehmen und Hochschulen separate Geräte für WLAN-APs einsetzen, sind im privaten Bereich die APs häufig zusätzlich in den Internet-Zugangsroutern integriert. Der AP liegt praktisch auf einem Port des integrierten Switch auf der privaten Netzseite. Über ihn sind sowohl das Internet als auch die Rechner des internen Netzes ansprechbar.



Infrastruktur-Modus

Mehrere APs können über überlappende Funkzellen unterbrechungsfreie Funkverbindungen beim Übergang von einer zu einer anderen Funkzelle bereitstellen („Roaming"). WLAN-Richtfunk-Systeme lassen sich als Funkbrücken zwischen nicht zu weit voneinander entfernte Liegenschaften nutzen.

20.3.2 Die Luftschnittstelle

Die Technik von WLANs ist in einer Reihe von IEEE802.11-Standards festgelegt. Nachfolgende Tabelle zeigt die Standards in der Reihenfolge ihrer Entwicklung:

WLAN Standards				
IEEE-Standard	*Frequenzband*	*max. Übertragungsrate*	*Anzahl Kanäle (D)*	*max. Sendeleistung*
802.11	2.4GHz	2 Mbit/s	13	100 mW
802.11b		11 Mbit/s		
802.11b+		22 Mbit/s		
802.11g		54 Mbit/s		
802.11a	5.2GHz	54 Mbit/s	19	bis 1000 mW

Die neueren 2.4 GHz Standards sind abwärtskompatibel.

Der 2.4-GHz-Frequenzereich ist Teil des lizenzfreien ISM-(*Industrial Scientific Medical*) Bandes, das von jedermann frei genutzt werden kann. In diesem Bereich arbeiten auch andere Geräte mit Funkübertragungen wie Garagenöffner, schnurlose Telefone oder Bluetooth-Geräte (Handys!), deren Störungen oft die effektive WLAN-Datenrate reduzieren und zu einem automatisches Fallback zu niedrigeren Datenraten führen. Weiterhin beträgt die Kanalbandbreite ca. 22 MHz, der Kanalabstand jedoch nur 5 MHz! Wegen dieser starken Kanalüberlappung können von den 13 verfügbaren Kanälen nur 3 parallel genutzt werden!

WLAN 802.11b/g:

Die große Kanalüberlappung kann zu Störungen und Reduktion der Übertragungsrate führen, falls andere WLAN-Systeme in die Funkzelle hineinreichen!

Praktisch parallel nutzbar sind Kanäle mit einem 5er-Kanalabstand, also z. B. die Kanäle 1,6,11 oder 2,7,12.

Die auf 100mW beschränkte Sendeleistung führt je nach Umgebungsbedingungen zu Reichweiten von ca. 100 m bei Sichtverbindung oder ca. 20 m innerhalb von Gebäuden.

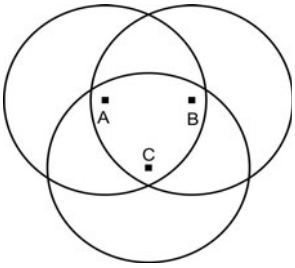
Die ständig anwachsende Anzahl installierter 802.11b/g-Systeme und die o.g. Störanfälligkeit im 2.4-GHz-Bereich führte zur Einführung des neuen Standards 802.11a im 5.2-GHz-Bereich. Dieser neue Frequenzbereich ist gegenüber dem schon seit langer Zeit etablierten offenen 2.4-GHz-Bereich weniger belegt und deshalb störsicherer. Ferner wurde hier durch ein 20-MHz-Kanalraster eine Kanalüberlappung vermieden. Mit der höheren Sendeleistung sind größere Reichweiten erreichbar.

Bei den oben angegebenen Datenraten handelt es sich um Brutto-Werte. Wegen der bei Funkübertragungen erhöhten Anteile an Handshakes und Zusatzheadern bei den gesendeten Datenblöcken beträgt die Netto-Nutzlastrate nur ca. 50% des Brutto-Datendurchsatzes.

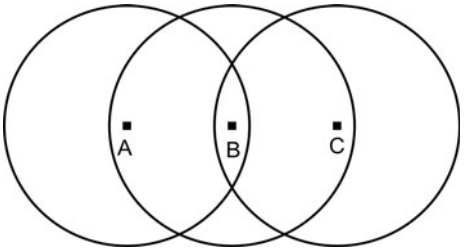
Die IEEE Arbeitsgruppe arbeitet an weiteren WLAN-Standards, die zu erheblich größeren Reichweiten führen sollen.

20.3.3 Der Netzzugriff

Funkübertragungen nutzen naturgemäß ein *shared medium.* Der Netzzugriff bei WLANs basiert wie bei drahtgebundenen Ethernetnetzen auf dem CSMA-Verfahren. Es ist jedoch nicht möglich, Kollisionen während der eigenen Sendung zu erkennen (*/CD :collision detect)*, da der eigene Funkempfänger andere Sendungen „überstrahlen“ würde und immer nur die eigene Funksendung wahrnehmen könnte. Es kommen daher andere Verfahren zum Einsatz, die Kollisionen nicht ganz verhindern, aber weitgehend vermeiden: *CSMA/CA (/CA :collision avoidance).* Das Prinzip ist für zwei Fälle dargestellt:



a) Alle Stationen liegen im gemeinsamen Überlappungsbereich



b) Station A und C haben keine Verbindung

Fall a: Idealfall, alle Stationen liegen im überlappenden Funkbereich und können die Sendungen aller Stationen mitverfolgen. Jede Station erkennt durch Abhören des Funkkanals, ob der Kanal frei ist. Stellt eine sendewillige Station fest, dass der Kanal belegt ist, wartet sie nach dem Ende der Sendung noch eine zufällig lange Zeitspanne ab, bevor sie mit der Sendung beginnt.

Fall b: „*Hidden Station Problem"*: Station C liegt außerhalb der Funkreichweite der Station A. C ist gegenüber A „versteckt" (und umgekehrt). In diesem Fall muss das *RTS/CTS*-Handshake-Protokoll aktiviert werden:

- A möchte an B senden und hört den Funkkanal ab. Sobald er frei ist, sendet A ein RTS-(*Request To Send)* Kanal-Reservierungspaket an B. Das RTS-Paket enthält die Länge – und damit den erforderlichen Zeitbedarf – des Datenpakets, das anschließend übertragen werden soll.
- Station B antwortet mit einem CTS-(*Clear To Send*)Paket, das die Reservierungslänge aus dem RTS-Paket enthält. Das CTS-Paket wird von allen Stationen des Sendebereichs von B mitgelesen, auch von der Station C. C wartet die reservierte Sendezeit ab, bevor es seinerseits einen Sendeversuch unternimmt.
- Nach Empfang des CTS-Pakets sendet A das Datenpaket ab an B.

In beiden Fällen kann trotzdem ein zufälliger gleichzeitiger Sendebeginn zweier Stationen nicht ganz ausgeschlossen werden, z. B. das gleichzeitige Senden von RTS-Paketen von A und C. Folglich sind Kollisionen und Verlust von Datenpaketen nicht ganz unmöglich. Um Verluste von Paketen zu erkennen, quittieren die Empfänger eingetroffene Sendungen mit einem *ACK-(Acknowledge)* Paket. Das Ausbleiben eines ACK-Pakets zeigt dem Absender an, dass seine Sendung den Empfänger nicht erreicht hat. Er wiederholt seine Sendung.

In gemischten 802.11b/11g-Umgebungen kann eine 11b-Station nicht die Kommunikation der 11g-Datenpakete verstehen (umgekehrt ist das möglich!), was zusätzlich zu unvermeidbaren Kollisionen und damit zu Absenkungen des Datendurchsatzes führt.

Trotz des großen Vorteils kabelloser Netze sollte jedoch auch der prinzipiell geringere Datendurchsatz in WLANs bedacht werden (insbesondere von „Zockern"!):

WLAN Datendurchsatz

Zusätzliche Handshakes und mögliche Kollisionen machen eine Funk-LAN-Verbindung langsamer als eine leitungsgebundene Verbindung.
Ping-Zeiten in WLANs sind in der Praxis mindestens 4x länger als in drahtgebundenen Netzen!

20.3.4 Sicherheit von WLANs

Als „War-Driving" beteichnet man das Vorgehen, mit dem Auto und Notebook durch die Gegend zu fahren und in ungesicherte Funknetze einzudringen. Untersuchungen zeigen immer wieder, dass ein grosser Teil der WLAN-Betreiber ihre Funknetze gar nicht oder nur unzureichend gegen Eindringlinge absichern. Dabei ist eine kostenlose Mitbenutzung des Internetzugangs „über den Nachbarn" noch vergleichsweise harmlos, das Ausspähen und evtl. sogar Manipulieren der Daten privater Netze weit schädlicher!

Vertraulichkeit und Integrität der übertragenen Daten werden durch Verschlüsselung und Authentifizierung geschützt. Die bei WLANs eingesetzten Verfahren sind:

- WEP (*Wired Equivalent Privacy*)
- WPA (*WiFi Protected Access)*

WEP ist Bestandteil des 802.11-Standards. Es beruht auf dem RC4 symmetrischen Verschlüsselungsalgoritmus mit statischen 64 Bit oder 128 Bit langen Schlüsseln. Das Verfahren lässt jedoch nur eine Schlüsseleingabe von 40 Bit (5 ASCII-Zeichen) oder 104 Bit (13 ASCII-Zeichen) zu, da die restlichen 24 Bit nach einem fest vorgegebenen Algoritmus intern berech-

net werden. Der Schlüssel muss als PSK (*Pre Shared Key*) allen Stationen des WLAN mitgeteilt werden.

WEP gilt nicht als sehr sicher, da statische Schlüssel oft monatelang unverändert bleiben und während dieser langen Zeit durch systematisches Abhören und Analysieren ermittelt werden können. Im Internet sind „WEP-Knacker"-Werkzeuge offen zugänglich!

Das WPA-Verfahren stammt von der WiFi-(*Wireless Fidelity*) Allianz, eine Herstellervereinigung zur Vermarktung kompatibler WLAN-Produkte. Es ersetzt die statischen WEP-Codes durch dynamische Schlüssel und führt deshalb zu einem erheblich bessseren Schutz.

Für die Authentifizierung der WLAN-Clienten stehen in der Regel die folgenden Alternativen zur Verfügung:

- „Open System": keine Prüfung
- „Shared Key" : Der Client sendet eine Authentifizierungsanforderung an den AP.
  Der AP antwortet mit einem Zufallstext (*Challenge*).
  Der Client verschlüsselt diesen mit dem PSK und sendet ihn an den AP zurück.
  Falls der AP den Text entschlüsseln kann, sind beide Schlüssel gleich und der Client wird zugelassen.
- nach 802.1x : Authentifizierung durch einen RADIUS-Server (→ s. Kap. 18.4.2).

Für jedes WLAN vergibt der Administrator bei der Konfiguration des AP einen eindeutigen Netzwerknamen, die sog. *SSID* (*Service Set Identity*). Der AP sendet periodisch Broadcasts in seine Funkzelle mit der SSID aus. Clienten scannen den WLAN-Frequenzbereich ab und zeigen an, welche WLANs (SSIDs) erreichbar sind. Anschließend kann eine SSID ausgewählt und eine Verbindung zu einem bestimmten AP aufgenommen werden. Die SSID-Meldungen sind für einen WLAN-Client jedoch nicht zwingend erforderlich. Im Hinblick auf eine erhöhte Sicherheit sollte man die SSID-Broadcasts deaktivieren, da alleine schon Namen wie z. B. „PRAXIS" oder „SCHMIDTNET" Rückschlüsse auf das Netz zulassen und einen Einbruchsversuch erleichtern!

Sicherheits-Empfehlungen zur Konfiguration von WLANs	
Verschlüsselung und Authentifikation:	WPA-PSK *oder* WPA-802.1x
SSID-Broadcasts:	deaktivieren
DHCP-Server des AP:	deaktivieren, feste IP an Client vergeben
MAC Zugriffslisten beim AP:	einrichten; es können nur MAC-Adressen (Ethernet-Adressen) eingetragener Clients eine Verbindung zum AP aufbauen.

20.4 Virtuelle LANs (VLAN)

Größere Unternehmen/Hochschulen betreiben oft hunderte von Rechnern. Aufteilungen in Subnetze und Segmentierungen durch Switches sind erforderlich, um den Datenverkehr zu reduzieren und verwaltbare Einheiten zu schaffen. Die so erreichten „Netzwerk-Inseln" (Verwaltungs-Einheiten) sind durch den Aufstellungsort der Koppelelemente und Endgeräte örtlich fest vorgegeben, d. h. z. B. ein bestimmtes Subnetz der Projektgruppe/Abteilung *x* ist durch die

Position des Routers im Gebäude *y* in der Etage *z* starr vorgegeben. Veränderungen sind nur mit großem Aufwand an Hard- und Software-Umrüstungen (Verkabelung, Gerätekonfiguration) durchführbar.

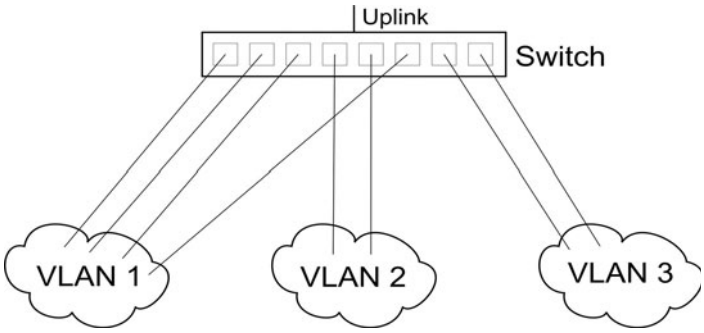
In der Praxis zeigt sich jedoch, dass Firmen sehr häufig umstrukturieren, indem sie z. B. Mitarbeiter neuen Projektgruppen zuordnen, Räumlichkeiten wechseln und interne Arbeitsabläufe neu organisieren. Die damit verbundene Umstellung der Netzwerkorganisation wird durch die Installation von *Virtuellen Netzen,* sog. VLANs, erheblich vereinfacht.

VLAN-Struktur

Ein physikalisches Unternehmens-Netzwerk, das aus örtlich aufgestellten Switches und Subnetzroutern besteht, wird auf ein virtuelles logisches Netzwerk (VLAN-Struktur) abgebildet, das einfach verwaltbar und sehr flexibel veränderbar ist.

VLANs benutzen den gleichen vorliegenden Netzwerk-Adressraum (IP-Subnetzstruktur), verteilen diesen jedoch – unabhängig von örtlichen Gegebenheiten – über das gesamte Unternehmensnetz.

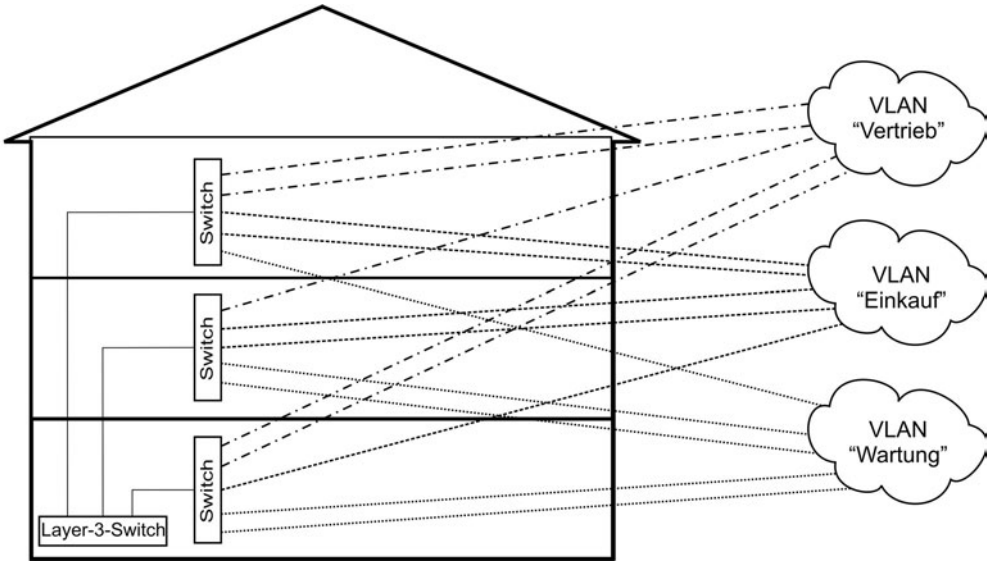
VLANs werden an VLAN-fähigen Switches eingerichtet (nicht jeder Switch unterstützt VLANs!), indem die Switch-Ports verschiedenen virtuellen LANs zugeodnet werden.



VLANs am Switch

Verschiedene Endgeräte, die am gleichen Switch(-Segment) angeschlossen sind, können unterschiedlichen VLANs angehören und bleiben verkehrsentkoppelt. Jedes VLAN bildet nicht nur eine eigene Kollisionsdomäne, sondern auch eine eigene Broadcast-Domäne (also ein eigenes Netz!). Broadcasts verbleiben so in demjenigen VLAN, in dem sie entstanden sind und werden nicht – wie bei „normalen“ Switches – auf alle Ports geflutet. Da Broadcast-Verkehr den Datendurchsatz in großen geswitchten Netzen erheblich beeinträchtigen kann (NETBIOS von Windows-Rechnern, DHCP, ARP usw.), trägt die VLAN-Bildung auch zur Effizienzsteigerung und erhöhten Sicherheit bei (Broadcasts nur an die Stationen, die es angeht!). Eine Verbindung zwischen unterschiedlichen VLANs ist nur über Router zu erreichen.

Der echte Vorteil von virtuellen Netzen kommt erst bei der Verteilung von VLANs über mehrere gekoppelte Switches voll zur Geltung. Ein und dasselbe VLAN kann an den Ports verschiedener Switches fortgesetzt werden. Der Layer-3-Switch ermöglicht als „Multiport-Router“ die Inter-VLAN-Kommunikation.



VLANs über mehrere Switches und Layer-3-Switch

Switches arbeiten auf der Schicht 2, d. h. ein Switch kann nur mit Ethernet-Frames umgehen. Wie kann auf dieser Schicht eine Aufteilung in verschiedene virtuelle Netze geschehen? Dies ist nur möglich durch Veränderungen am Frame-Aufbau durch den Switch. Das Verfahren ist im IEEE 802.1Q-Standard festgelegt: Vor dem Typen-/Längenfeld eines Ethernet-Pakets (→ s. Kap. 15.4) wird in einem Zusatzfeld eine 12-Bit-lange VLAN-ID (VLAN-Nummer) eingefügt, womit ein *Tagged Frame* entsteht. VLAN-fähige Switches tauschen Tagged Frames aus und schalten das entsprechende Datenpaket auf den zugehörigen Port eines bestimmten VLAN durch.

Trifft ein unge*tag*tes Datenpaket an einem Port eines Switches ein, so fügt der Switch die diesem Port entsprechende VLAN-ID in den Frame ein. Bevor ein ge*tag*tes Datenpaket den letzten Switch in Richtung Endgerät verlässt, entfernt der Switch wieder den *Tag*, da Endgeräte meistens nicht mit ge*tag*ten Frames umgehen können und das Paket verwerfen würden.

Die Aufteilung eines Netzes in VLANs kann Hardware-gebunden (nicht flexibel!) oder logisch, Adress-gebunden, erfolgen:

Die wichtigsten VLAN-Typen		
Typ	*VLAN-Zugehörigkeit*	*Anmerkungen*
Schicht-1-VLAN Port-basierend	Switch-Port, an dem das Endgerät liegt ↔VLAN-ID.	Statisch und wenig flexibel: Bei Verlagerung eines Endgerätes muss der Switch umkonfiguriert werden.
Schicht-2-VLAN MAC-basierend	MAC- (Ethernet-) Adresse des Endgerätes ↔VLAN-ID	Dynamisch: Bei Verlagerung eines Endgerätes konfiguriert sich Switch selber.
Schicht-3-VLAN IP-Subnetz-basierend	1:1 Zuordnung IP-Subnetz ↔VLAN-ID durch Layer-3-Switch	Dynamisch: Bei Verlagerung eines Endgerätes konfiguriert sich Switch selber.

Da eine Aufteilung in VLANs eine Inter-VLAN-Kommunikation nicht ganz verhindern soll, wird meistens die letzte Variante in der Praxis eingesetzt. Sie erfordert einen Layer-3-Switch, der die Zuordnung der Subnetze zu den VLANs bereitstellt.

Die frei konfigurierbare logische Einteilung von VLANs liefert die große Flexibilität bei örtlichen Veränderungen innerhalb eines Unternehmens: Ein umziehender Mitarbeiter nimmt seine VLAN-Zugehörigkeit mit! Die Switche erkennen den Umzug.

VLANs

ermöglichen die „Fortsetzung“ von Subnetzen an beliebigen Switch-Ports des Unternehmens; sie unterstützen damit eine flexible, nicht ortsgebundene Arbeitsgruppenorganisation. Jedes VLAN bildet eine eigene Broadcast-Domäne und fördert dadurch Datendurchsatz und Sicherheit.

21 Netzwerkprogrammierung mit Sockets

Das Socket-API liegt zwischen den transportorientierten und den anwendungsorientierten Netzwerkschichten und tritt in der Praxis in 2 Formen auf: Ursprünglich wurde die Schnittstelle für BSD-Unix (*Berkeley Software Distribution*) entworfen. Sie ist heute als „BSD-Sockets“ in allen Unix-Versionen enthalten. Microsoft hat die Schnittstelle als „Windows Sockets“ (kurz: WINSOCK) für Windows-Systeme mit geringen Erweiterungen adaptiert.

Die Socket-Schnittstelle liegt als C-Programmbibliothek vor und kann mit der Anweisung „#include <socket.h>“ bzw. „#include <winsock.h>“ in ein C/C++ - Programm eingebunden werden. Das Interface stellt eine Reihe von Funktionen, Strukturen und Typdefinitionen für das Transportsystem zur Verfügung und bietet damit die erforderlichen Dienste der Schichten 1-4 für eine Ende-zu Ende-Kommunikation (Prozess-zu-Prozess-Kommunikation) in Netzwerken. Es können sowohl verbindungsorientierte (z. B. TCP-basierend) Transportdienste als auch verbindungslose (z. B. UDP-basierend) Transportdienste genutzt werden. Auf diesem Transportsystem aufbauend, erlaubt die Socket-Schnittstelle somit die Entwicklung eigener Client/Server-Anwendungen.

Die Schnittstelle ist „offen“ und für beide Betriebssysteme gut dokumentiert, die entsprechenden ca. 120-seitigen Dokumente sind im Internet frei erhältlich. Einen guten Einblick in die Funktionalität gewinnt man auch durch die Include-Datei (z. B. winsock.h) selbst, die ja als Textdatei vorliegt.

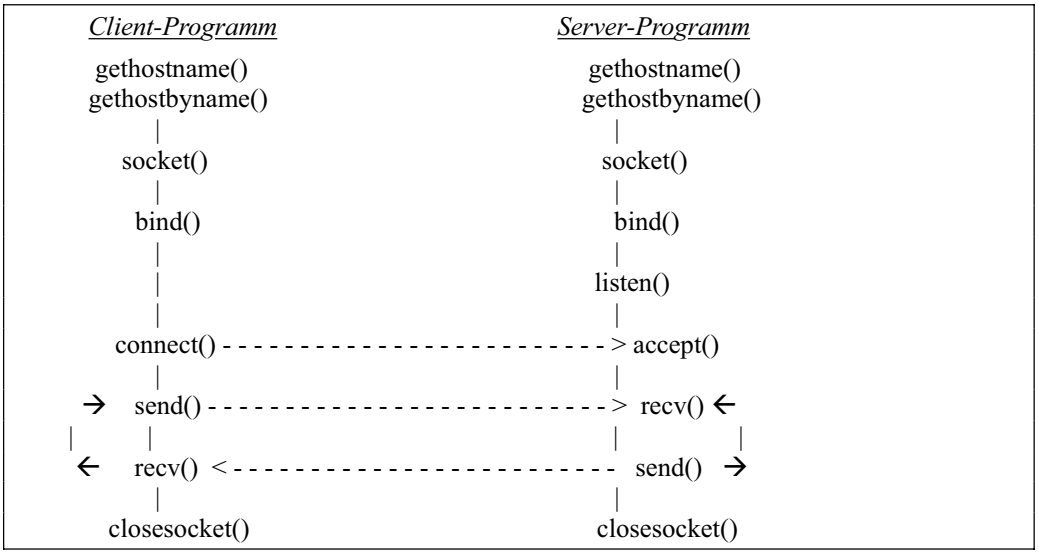
Im Folgenden soll ein einfaches Beispiel für eine verbindungsorientierte Client/Server-Kommunikation vorgestellt werden. Das Programmierbeispiel zeigt dabei auch, wie die in der Netzwerktechnik eingeführten Begriffe konkret und praktisch in die Softwareentwicklung eingehen (gemäß der häufig geäußerten Bemerkung: „erst wenn man ein funktionierendes Programm entwickelt hat, hat man die Sache richtig verstanden!“)

21.1 Socket -Funktionen und -Datenstrukturen

Die Kommunikation über Sockets erfolgt über einen Kommunikationskanal, der charakterisiert ist durch die Parameter:

> Transportprotokoll; lokale Adresse; lokaler (Prozess-)Port;
> Partner-Adresse; Partner-(Prozess-)Port.

Der Kommunikationskanal wird durch eine Reihe von Socket-Funktionen eingerichtet und am Ende wieder abgebaut. Eine TCP-Kommunikation hat den folgenden prinzipiellen Ablauf:



Zunächst wird über die Funktion

***int** gethostname(**char** *name,* out: Hostname
***int** namelen)* out: Namenslänge

der lokale Hostname erfragt. Dieser wird dann benutzt, um mit der Funktion *gethostbyname()* Informationen über den lokalen Host zu erhalten:

***struct** hostent *gethostbyname(**char** *name)* in: Hostname

Die Funktion liefert einen Pointer auf die vordefinierte Struktur:

***struct** hostent {*
***char** *h_name;* Hostname
***char** *h-alias* ; Alias Liste
***short** *h_addrtype;* Host Adress-Typ
***char** **h_addr_list;}* Adress-Liste, **h_addr_list[0]* enthält IP-Adresse.

Die so erhaltene Adresse wird in die vordefinierte Socket-Adress-Struktur für einen Internet-„Endpunkt“ übernommen:

***struct** SOCKADDR_IN {* Internet-Endpoint-Adresse:
***unsigned short** sin_family;* Adress-Familie *AF_INET;*
***unsigned short** sin_port;* Portnummer der Anwendung;
***unsigned long** sin_addr;* IP-Adresse;
***char** sin_zero[8];* } Füllbyte für 14 Byte Gesamtlänge;

Nun wird durch die Funktion

*SOCKET socket (**int** af,* in:Adressformat, für Internet stets *AF_INET,*
***int** type,* in: Typ des erzeugten Socket,
SOCK_STREAM für TCP (verbindungsorientiert),
SOCK_DGRAM für UDP (verbindungslos);
***int** protocol)* in: Protokoll, Wert „0“ wählt jeweils default zu *type*;

ein ganzzahliger positiver Socket-Deskriptor erzeugt. Hierbei ist *SOCKET* vordefiniert mit ***typedef unsigned long int*** *SOCKET*. Über den Ganzzahlenwert wird der Socket von nun an angesprochen.

Mit *bind()* wird der neu erzeugte Socket an die Adress-Struktur gebunden:

int *bind (SOCKET s,*	in: Socket-Deskriptor,
const struct *SOCKADDR *name,*	in: Zeiger auf Adress-Struktur,
int *namelen)*	in: Grösse der Adress-Struktur.

Die hier erwartete Socket-Adresse ist nicht die des Internet-Sockets (ist nicht an das Internet-Protokoll gebunden) und besitzt die allgemeine Struktur

struct *SOCKADDR {*	allgemeine Endpoint-Adresse:
unsigned short *sa_family;*	Adress-Familie;
char *sa_data[14];}*	Adresse.

Durch Casten wird die Internet-Adress-Struktur auf die allgemeine Adress-Struktur abgebildet: *bind(SOCKET s, (SOCKADDR *)&LocalAddr,* ***sizeof****(LocalAddr)),* wobei *LocalAddr* vom Typ *SOCKADDR_IN* ist.

Nun sind die jeweiligen Endpunkte der Verbindung für Server und Client aufgebaut.

Der **Server** wird durch die Funktion

int *listen (SOCKET ServerSocket*	in: Socket-Deskriptor,
int *quelen)*	in: Länge der Warteschlange,

empfangsbereit gemacht. Für den Fall mehrerer eintreffender Client-Anfragen wird eine Warteschlange eingerichtet.

Der **Client** baut mit

int *connect (SOCKET ClientSocket,*	in: Socket-Deskriptor,
const struct *SOCKADDR *r_name,*	in: Zeiger auf Adess-Struktur des Remote Partners, zu dem die Verbindung auf genommen wird,
int *namelen)*	in: Grösse der Adress Struktur;

die Verbindung zum Server auf.

Der Verbindungswunsch wird vom Server durch *accept()* angenommen. Die Funktion erzeugt serverseitig einen neuen Socket-Deskriptor *ClientSocket,* der zum Empfang und Senden von Daten angesprochen wird:

SOCKET accept SOCKET ServerSocket,	in: Socket-Deskriptor,
struct *SOCKADDR *r_name*	out: Zeiger auf Adress-Struktur des akzeptierten Partners,
int **addrlen)*	out: Zeiger auf ein Integer, der die Adress-Länge enthält.

Damit ist der Verbindungsaufbau abgeschlossen, der Kommunikationskanal ist eingerichtet.

Nun können mit *send()* und *recv()* Daten ausgetauscht werden, die auf dem Server und auf dem Client jeweils in Puffern bereitgestellt bzw. von dort abgeholt werden. Die Funktionen liefern die Anzahl der abgesandten bzw. empfangenen Zeichen zurück:

int *recv (SOCKET ClientSocket,*	in: Socket-Deskriptor,
char **buf,*	in: Zeiger auf Empfangspuffer,
int *len,*	in: Pufferlänge,

***int** flags)*	in: spezifiziert weitere Optionen, meistens: *0*.
***int** send (SOCKET ClientSocket,*	in: Socket-Deskriptor,
***char** *buf,*	in: Zeiger auf Sendepuffer,
***int** len,*	in: Pufferlänge,
***int** flags)*	in: spezifiziert weitere Optionen, meistens: *0*.

Nach Beendigung des Datenaustausches werden die Sockets auf Server und Client wieder geschlossen mit

***int** closesocket (SOCKET s)*	in: Socket-Deskriptor,

und der Kommunikationskanal ist abgebaut.

21.2 Beispiel einer Client/Server-Anwendung

Das nachfolgende Beispiel zeigt eine einfache Client/Server-Kommunikation auf der Basis von Windows Sockets unter Windows mit Einbindung von <winsock.h>. Der Client sendet Daten an den Server, der Server bestätigt das Eintreffen der Daten, indem er einen Antworttext an den Client zurücksendet. Eingabe von „##“ am Client beendet das Clientprogramm, der Server bleibt weiterhin aktiv und kann weitere Clientanfragen bedienen.

Das Server-Programm:

```
// Demoprogram Server
#include <iostream.h>
#include <winsock.h>
using namespace std;
const short int ServerTCP_Port = 10035;
const int MAXQUE = 2; //Client-Warteschlange bei listen
void WSAInit(void);
int comm(void);
//--------------------------------------------------------------------------
int main()
{
   while(true)
     comm();
   return 0;
}
void WSAInit(void)
{
   WORD wVersionReg;
   WSADATA wsaData;
   // Windows-Version
   // high-byte: Revisionsnummer, low-byte: Versionsnummer
   wVersionReg = MAKEWORD(1,1);
   if(WSAStartup(wVersionReg,&wsaData) != 0)
   {
         cout << "Fehler bei WSAStartup()" << endl;
         getchar();
         exit(1);
   }
   cout << "WSAStartup() fehlerfrei beendet" << endl << endl;
}
```

```
int comm()
{
    int addrlen, n=0;
    bool weiter = true;
    char rBuffer[250];
    char okBuffer[] ="Sendung am Server erfolgreich eingetroffen";
    char rechnername[50];
    struct hostent *hostinfo;

    SOCKET ServerSocket = INVALID_SOCKET; //0xFFFFFFFF=4294967295
    SOCKET ClientSocket = INVALID_SOCKET;
    SOCKADDR_IN SockIN, Client_addr;

    memset(&SockIN, 0x00, sizeof(SockIN)); //erst komplett löschen
    memset(&Client_addr,0x00, sizeof(Client_addr));
    cout << "Server gestartet!" << endl << endl;

    // Socket-Dienste initialisieren
      WSAInit();

    //   Host-Informationen:
    gethostname(rechnername,20);  //Rechnername holen
    hostinfo = gethostbyname(rechnername);// Struktur für "rechnername"
                                          // anlegen
    memcpy((char*)&SockIN.sin_addr, hostinfo->h_addr_list[0],
           hostinfo->h_length);
    cout << "***********************************" << endl
         << "Host-Information:" << endl
         << "Mein Rechnername ist: " << rechnername << endl
         << "Meine IP-Adresse ist: " << inet_ntoa(SockIN.sin_addr) // IP
         << endl << "***********************************" << endl;

    // Aufbau des Endpoints
    ServerSocket = socket(AF_INET, SOCK_STREAM, 0);
    if(ServerSocket == INVALID_SOCKET)
    {
        cout << "Fehler bei socket()" << endl;
        getchar();
        exit(1);
    }
    cout << "...ServerSocket erfolgreich erzeugt" << endl;
    // Adressenfamilie
    SockIN.sin_family = AF_INET; //  =2
    // eigene Anwendung (Nicht bereits bestehender Service):
    SockIN.sin_port = htons(ServerTCP_Port);   // = 0x3327  Port

    // verbinden des Sockets mit der Adresse
    if( bind(ServerSocket, (SOCKADDR*)&SockIN, sizeof(SockIN) ) <0 )
    {
        cout << "Fehler bei bind()" << endl;
        getchar();
        exit(1);
    }
    cout <<"...bind() erfolgreich" << endl;
```

```
    if( listen(ServerSocket, MAXQUE) == SOCKET_ERROR)
    {
       cout << "Fehler bei listen()" << endl;
       getchar();
       exit(1);
    }
    cout << "...listen() erfolgreich" << endl
         << "...warte auf Client" << endl;
    addrlen = sizeof(Client_addr);
    ClientSocket = accept(ServerSocket,(SOCKADDR*)&Client_addr, &addrlen);
    if(ClientSocket == INVALID_SOCKET)
    {
       cout << "Fehler bei accept()" << endl;
       getchar();
       exit(1);
    }
    cout << "Verbindung hergestellt mit Client: "
         << " IP: "<<inet_ntoa(Client_addr.sin_addr)
         << " Port: " << dec << ntohs(Client_addr.sin_port)
         << endl;

    // Client-Verbindung eingerichtet
    while(weiter)
    {
       cout << "...warte bei recv()" << endl;
       n = recv(ClientSocket, rBuffer, sizeof(rBuffer), 0);
       if(n>0)
       {
          cout << "..." << n << " Zeichen empfangen" << endl
               << rBuffer << endl;
          n = send(ClientSocket, okBuffer, sizeof(okBuffer),0);
        }
        if((strcmp(rBuffer, "##") == 0))
        {
            cout << "Client hat Kommunikation beendet" << endl;
            weiter = false;
        }
        if(n<0)
        {
            cout << "Kommunikation wurde abgebrochen!" << endl;
            weiter = false;
        }
    }

    //Socket nach der Arbeit immer schliessen
    closesocket(ClientSocket);
    closesocket(ServerSocket);
    cout << "...Sockets geschlossen" << endl;
    // Socket-Dienste terminieren
    WSACleanup();
    cout << "...WSACleanup() beendet" << endl << endl << endl;
    return 0;
}
```

Anmerkungen zum Programm:

Das Programm ist als Dauerschleife angelegt.

Windows verlangt eine Initialisierung von WINSOCK durch *WSAStartup().* Die Funktion prüft, ob die Datei „winsock.dll“ vorliegt.

Der Server-Port ist hier auf 10035 festgelegt.

Die C-Funktion *memset()* beschreibt einen Speicherbereich mit einem vorgegebenen Bitmuster. Sie dient hier zum initialen Löschen der Adress-Strukturen.

Die C-Funktion *memcpy()* kopiert Speicherbereiche und wird hier dazu benutzt, Daten aus verschiedenen Strukturen zu übertragen.

Netzwerke arbeiten grundsätzlich bei der Speicherung von Mehrbyte-Datentypen nach der „Network Byte Order“ im „Big-Endian“-Format, d. h. das höherwertige Byte liegt auf der tieferen Speicheradresse. Die meisten Hosts -insbesondere Intel- benutzen als „Host Byte Order“ jedoch das „Little-Endian-Format“, bei dem das höherwertige Byte auf der höheren Speicheradresse abgelegt ist. Integer-Datentypen für Portnummern und IP-Adressen müssen daher entsprechend konvertiert werden, bevor sie den Socket-Funktionen übergeben oder nach Übernahme vom Netzwerk am Bildschirm ausgegeben werden können. Die Socketbibliotheken bieten hierfür vier Konversions-Funktionen an:

short int *htons (* ***short int*** *k)* : Konversion Host-to-Network short,

short int *ntohs (****short int*** *k)* : Konversion Network-to-Host short,

long int *htonl (* ***long int*** *k)* : Konversion Host-to-Network long,

long int *ntohl (****long int*** *k)* : Konversion Network-to-Host long.

Zwei weitere Konvertierungen beziehen sich auf die „dotted“ (mit Punkt geschriebenen) IP-Adressen:

char ****inet_ntoa(*long int*** <IP-Adresse in Network-Order>) : Rückgabe der „dotted“ IP-Adresse als ASCII-String, und die Umkehrung:

long int *inet_addr* ***(char ****<dotted IP-Adresse als ACII-String>*)* : Rückgabe der IP-Adresse in Network-Order.

WINSOCK verlangt nach dem Schließen der Sockets noch den Aufruf von *WSACleanup(),* der die Socket-Dienste terminiert.

Das Client-Programm:

```
#include <iostream>
#include <winsock.h>
using namespace std;
const short int ServerTCP_Port =10035;
void WSAInit(void)
{
   WORD wVersionReg;
   WSADATA wsaData;
   // Windows-Version
   // high-byte: Revisionsnummer, low-byte: major Versionsnummer
   wVersionReg = MAKEWORD(1,1);
   if(WSAStartup(wVersionReg,&wsaData) != 0)
   {
           cout << "Fehler bei WSAStartup()" << endl;
```

```
         getchar();
         exit(1);
   }
   cout << "...WSAStartup() gestartet" << endl;
}
//-----------------------------------------------------------------------
int main()      // Cient
{
    int n=0;
    bool weiter = true;
    char sBuffer[256];
    char rBuffer[256];
    int rn=0;
    SOCKET ClientSocket = INVALID_SOCKET;
    SOCKADDR_IN SockIN, RemoteAddr;
    char rechnername[50], server_ip_addr[15];
    struct hostent *hostinfo;

    memset(&SockIN, 0x00, sizeof(SockIN)); //erst komplett löschen
    cout << "Client gestartet" << endl;

    // Socket-Dienste initialisieren
    WSAInit();

    //   Host-Informationen:
    gethostname(rechnername,20);  //Rechnername holen
    hostinfo = gethostbyname(rechnername);//Struktur für host
                                          //"rechnername" anlegen
    memcpy((char*)&SockIN.sin_addr, hostinfo->h_addr_list[0],
           hostinfo->h_length);
    cout << "************************************" << endl
         << "Host-Information:" << endl
         << "Mein Rechnername ist: " << rechnername << endl
         << "Meine IP-Adresse ist: " << inet_ntoa(SockIN.sin_addr) // IP
         << endl << "************************************" << endl;

    // Aufbau des Endpoints
    ClientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if(ClientSocket == INVALID_SOCKET)
    {
        cout << "Fehler bei socket()" << endl;
        getchar();
        exit(1);
    }
    //----- Ermittlung des Clientport, kann entfallen!-----------//-
    SockIN.sin_family = AF_INET;                                  //-
    if(bind(ClientSocket, (SOCKADDR*)&SockIN,sizeof(SockIN))<0)  //-
    {                                                             //-
        cout << "Fehler bei bind()" << endl;                      //-
        getchar();                                                //-
        exit(1);                                                  //-
    }                                                             //-
    int laenge;                                                   //-
     if(getsockname(ClientSocket, (SOCKADDR*)&SockIN,&laenge)<0)  //-
 getchar();                                                 //-
        exit(1);                                                  //-
```

```
    }                                                                  //-
    cout << "Clientport: " << ntohs(SockIN.sin_port) << endl;    //-
    //-------------------------------------------------------------//-

    // Adresse des angeforderten Servers
    RemoteAddr.sin_family = AF_INET;
    cout << "IP-Adresse des Servers: ";
    cin >> server_ip_addr;
    RemoteAddr.sin_addr.s_addr = inet_addr(server_ip_addr);
    RemoteAddr.sin_port = htons(ServerTCP_Port); // vorgegeben

    cout << "...warte bei connect()" << endl;
    if( connect(ClientSocket, (SOCKADDR*)&RemoteAddr,
                sizeof(RemoteAddr)) == SOCKET_ERROR)
    {
        cout << "Fehler bei connect()" << endl;
        getchar();
        exit(1);
    }
    while(weiter)
    {
       cout << "Text eingeben, Ende mit \"##\":" << endl;
       gets( sBuffer);
       n = send(ClientSocket, sBuffer, strlen(sBuffer)+1, 0);
       if(n>0)
          cout << "..." << n << " Zeichen gesendet!" << endl;
       if(strcmp(sBuffer, "##") == 0)
          weiter = false;
       rn=recv(ClientSocket,rBuffer, sizeof(rBuffer)+1,0);
       if(rn>0)
       {
          cout << "..." << rn << " Zeichen empfangen" << endl;
          puts(rBuffer);
       }
       else
       {
          cout << "Kommunikation unterbrochen!" << endl;
          weiter = false;
       }
       cout << endl;
    }

    //Socket nach der Arbeit immer schliessen
    closesocket(ClientSocket);
    cout << "...Socket geschlossen" << endl;

    // Socket-Dienste terminieren

   {                                                                   //-
       cout << "Fehler bei getsocket()" << endl;                       //-
    WSACleanup();
    cout << "Programmende" << endl;
    getchar();
    return 0;
}
```

Allgemeine Anmerkungen:

Die Demonstrationsprogramme sind ohne Einschränkung im Internet einsetzbar.

In dem Beispiel ist kein spezieller „Login“ mit Authentifizierung für den Client erforderlich! Jeder, der das Client-Programm nutzt, hat Zugang zum Server!

In dem Beispiel wurde der Server-„Dienst“ selbst entwickelt, wir benutzten einen selbstgewählten Port. Der Client hätte auch so entworfen werden können, dass eine Verbindung zu einem „well known“ Port des Servers, d. h. zu einem bestehenden Dienst wie E-Mail (Port 25) oder WWW (Port 80) aufgenommen wird. Somit erlaubt die Programmierung mit Sockets auch, direkt mit bestehenden Serverdiensten zu kommunizieren.

Im Gegensatz zu Windows-spezifischen Ansätzen wie z. B. *Active Directory* ist die Programmentwicklung mit Sockets plattformunabhängig. Mit nur sehr wenigen Änderungen sind die gezeigten Programme auch auf Unix (Linux-)-Sytemen einsetzbar oder machen eine Kommunikation zwischen Windows und Unix möglich.

21.3 Übungen

1) Verändern Sie die Programme so, dass ein „Chatten“ möglich ist. Da der Server in der Regel im Hintergrund läuft, soll das Serverprogramm einen Nutzer mit einem akustischen Signal darauf aufmerksam machen, dass ein Client eine Verbindung hergestellt hat.
2) Verändern Sie das Server-Programm derart, dass nur Clients mit bestimmten IP-Adressen zugelassen sind.
3) Erweitern Sie die Programme so, dass zu Beginn ein „Login“ mit Name und Passwort stattfindet.
4) Entwickeln Sie eine Client/Server-Anwendung, die auf Anforderung des Client eine Datei vom Server zum Client überträgt.
5) Entwickeln Sie ein Client-Programm, das den SMTP-Port eines E-Mail-Servers anspricht und eine E-Mail an eine bestimmte E-Mail-Adresse absetzt.

Anhang A: Erweiterte ASCII-Tabelle

ASCII-Tabelle (0-127 sowie erweitert 128-255)

Hex-code	2.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1.																	
0		NUL 0	SOH 1	STX 2	EXT 3	EOT 4	ENQ 5	ACK 6	BEL 7	BS 8	HT 9	LF 10	VT 11	FF 12	CR 13	SO 14	SI 15
1		DLE 16	DC1 17	DC2 18	DC3 19	DC4 20	NAK 21	SYN 22	ETB 23	CAN 24	EM 25	SUB 26	ESC 27	FS 28	GS 29	RS 30	US 31
2		 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
3		0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
4		@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
5		P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
6		` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
7		p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	▓ 127
8		Ç 128	ü 129	é 130	â 131	ä 132	à 133	å 134	ç 135	ê 136	ë 137	è 138	ï 139	î 140	ì 141	Ä 142	Å 143
9		É 144	æ 145	Æ 146	ô 147	ö 148	ò 149	û 150	ù 151	ÿ 152	Ö 153	Ü 154	¢ 155	£ 156	& 157	● 158	ƒ 159
A		á 160	í 161	ó 162	ú 163	ñ 164	Ñ 165	ª 166	º 167	¿ 168	⌐ 169	¬ 170	½ 171	¼ 172	¡ 173	« 174	» 175
B		 176	▒ 177	▓ 178	│ 179	┤ 180	╡ 181	╢ 182	╖ 183	╕ 184	╣ 185	║ 186	╗ 187	╝ 188	╜ 189	╛ 190	┐ 191
C		└ 192	┴ 193	┬ 194	├ 195	─ 196	┼ 197	╞ 198	╟ 199	╚ 200	╔ 201	╩ 202	╦ 203	╠ 204	═ 205	╬ 206	╧ 207
D		╨ 208	╤ 209	╥ 210	╙ 211	╘ 212	╒ 213	╓ 214	╫ 215	╪ 216	┘ 217	┌ 218	▉ 219	▄ 220	▌ 221	▐ 222	▀ 223
E		α 224	ß 225	Γ 226	π 227	Σ 228	σ 229	µ 230	τ 231	Φ 232	Θ 233	Ω 234	δ 235	∞ 236	φ 237	ε 238	∩ 239
F		≡ 240	± 241	≥ 242	≤ 243	⌠ 244	⌡ 245	÷ 246	≈ 247	° 248	∙ 249	· 250	√ 251	ⁿ 252	² 253	■ 254	 255

Anhang B: Steuerzeichen im 7-Bit-ASCII-Code

Dez	Hex	Binär	Strg	Kommando	Bedeutung
0	00	00000000	^@	NUL	null (end of C string)
1	01	00000001	^A	SOH	start of heading
2	02	00000010	^B	STX	start of text
3	03	00000011	^C	ETX	end of text
4	04	00000100	^D	EOT	end of transmission
5	05	00000101	^E	ENQ	enquiry
6	06	00000110	^F	ACK	acknowledge
7	07	00000111	^G	BEL	bell
8	08	00001000	^H	BS	backspace
9	09	00001001	^I	TAB	horizontal tab
10	0A	00001010	^J	LF	line feed
11	0B	00001011	^K	VT	vertical tab
12	0C	00001100	^L	FF	form feed
13	0D	00001101	^M	CR	carrige return
14	0E	00001110	^N	SO	shift out
15	0F	00001111	^O	SI	shift in
16	10	00010000	^P	DLE	data line escape
17	11	00010001	^Q	DC1	device control 1 (X-ON)
18	12	00010010	^R	DC2	device control 2
19	13	00010011	^S	DC3	device control 3 (X-OFF)
20	14	00010100	^T	DC4	device control 4
21	15	00010101	^U	NAK	negative acknowledge
22	16	00010110	^V	SYN	synchronous idle
23	17	00010111	^W	ETB	end transmission block
24	18	00011000	^X	CAN	cancel
25	19	00011001	^Y	EN	end of medium
26	1A	00011010	^Z	SUB	substitute (end of text file)
27	1B	00011011	^[	ESC	escape
28	1C	00011100	^\	FS	file seperator
29	1D	00011101	^]	GS	group seperator
30	1E	00011110	^^	RS	record seperator
31	1F	00011111	^_	US	unit seperator

Anhang C: Länderkennungen

Standardisierte Länderkennungen im Internet und in IntraNetWare Country-Objekten:

AD	Andorra
AE	Vereinigte Arabische Emirate
AF	Afghanistan
AG	Antigua
AI	Anguilla
AL	Albanien
AN	Niederländische Antillen
AO	Angola
AQ	Antarktis
AR	Argentinien
AS	Amerikanisch-Samoa
AT	Österreich
AU	Australien
AW	Aruba
BB	Barbados
BD	Bangladesch
BE	Belgien
BH	Bahrain
BJ	Benin
BS	Bahamas
BY	Weißrußland
BZ	Belize
CA	Kanada
CC	Kokos-Inseln
CF	Zentralafrikanische Republik
CG	Kongo
CH	Schweiz
CM	Kamerun
CO	Kolumbien
CU	Kuba
CX	Weihnachtinseln
CY	Zypern
CZ	Tschechische Republik
DZ	Algerien
EG	Ägypten
EH	Westsahara
ES	Spanien
ET	Äthiopien
EU	Europäische Union
FM	Mikronesien
GQ	Äquatorialguinea
GS	Südgeorgien
HR	Kroatien
HU	Ungarn
IL	Israel
IT	Italien
JE	Jemen
JM	Jamaika
JO	Jordanien
JP	Japan
KE	Kenia
KH	Kambodscha
KI	Kiribati
KM	Komoren
KP	Volksrepublik Korea
KR	Republik Korea
KW	Kuweit
KZ	Kasachstan
LA	Volksrepublik Laos
LB	Libanon
LC	St.Lucia
LI	Lichtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LU	Luxembourg
LV	Latvia
LY	Lybisch-Arabisch Dschamahirija
MA	Macao
MC	Monaco
MD	Moldavien
MG	Madagaskar
MH	Marshall-Inseln
ML	Mali
MM	Myanmar
MN	Mongolei
MO	Marokko
MP	Nördliche Mariannen
MQ	Martinique
MR	Mauretanien
MS	Montserrat
MT	Malta
MU	Mauritius
MV	Malediven
MW	Malawi
MX	Mexiko
MY	Malaysia
MZ	Mazedonien
NA	Nambia
NC	Neukaledonien
NE	Niger
NF	Norfolk-Inseln
NG	Nigeria
NI	Nicaragua
NL	Niederlande
NO	Norwegen
NP	Nepal
NR	Nauru
NU	Niue
NZ	Neuseeland
OM	Oman
PA	Panama
PE	Peru
PG	Papua-Neuguinea
PH	Philippinen
PK	Pakistan
PL	Polen
PM	Saint-Pierre-Miquelon
PN	Pitcairn
PR	Puerto Rico
PT	Portugal
PW	Palau
PY	Paraguay
QA	Katar
RE	Réunion
RO	Rumänien
RU	Russische Föderation
RW	Ruanda
SA	Saudi-Arabien
SB	Salomonen
SC	Sechellen
SD	Sudan
SE	Schweden
SG	Singapur
SH	St.Helena
SI	Slowenien
SJ	Svalbard- und Jan-Mayen
SK	Slowakei
SL	Sierra Leone
SM	San Marino
SN	Senegal
SO	Somalia
SR	Suriname
ST	Sao Tome und Principe
SY	Syrien
SZ	Swasiland

TC	Turks- und Caicos-Inseln	TV	Tuvalu	VI	Amerikanische Jungfern-Inseln
TD	Tschad	TW	Taiwan	VN	Vietnam
TG	Togo	TZ	Tansania	VU	Vanuatu
TI	Thailand	UA	Ukraine	WF	Wallis- und Futuna
TJ	Tadschikistan	UG	Uganda	WK	Wake-Inseln
TK	Tokelau-Inseln	US	Vereinigte Staaten von Amerika	WS	Samoa
TM	Turkmenistan	UY	Uruguay	YT	Mayotte
TN	Tunesien	UZ	Usbekistan	YU	Jugoslavien
TO	Tonga	VA	Vatikanstaat	ZA	Südafrika
TP	Ost-Timor	VC	St.Vincent	ZM	Sambia
TR	Türkei	VE	Venezuela	ZR	Zaire
TT	Trinidad und Tobago			ZW	Zimbabwe

Sachwortverzeichnis