Grundlagen der Informatik

Stephan Euler
FH-Giessen–Friedberg
Fachbereich MND
Version 1.15

Wintersemester 2007

Dieses Skript wurde mit LaTeX 2_ε und TeX(Version 3.141592) geschrieben. Eingesetzt wurde die integrierte Benutzeroberfläche WinEdt 5.3 zusammen mit MiKTeX Version 2.2. Die Bilder wurden mit GNUPLOT für MS Windows, Version 3.7 und jPicEdt 1.3.2 erstellt.

Vorwort

Das vorliegende Skript dient als Grundlage für ein Modul „Einführung in die Wirtschaftsinformatik“ im ersten Semester der Bachelor-Studiengänge Wirtschaftsinformatik und Wirtschaftsmathematik. Nach einem einleitenden Kapiteln zur geschichtlichen Entwicklung der Informatik und einer kurzen Darstellung einiger relevanten physikalischen Grundlagen gliedert sich das Material im wesentlichen in drei Abschnitte:

1. Informations- und Zahlendarstellung
2. Aufbau eines Rechners
3. Vernetzung von Rechnern

und spannt damit den Bogen vom einzelnen Bit bis zum weltumfassenden Internet.

Stephan Euler

Inhaltsverzeichnis

Kapitel 1

Geschichte der Informatik

1.1 Vorgeschichte

Mit der Entwicklung der Zivilisationen entstand zunehmend der Bedarf, große Mengen an Informationen - Zahlen, Namen, Texte - zu erfassen, zu bearbeiten und zu speichern. Mit der weitergehenden Technisierung entstand darüber hinaus ein spezieller Bedarf zur Steuerung von Fertigungsabläufen (z.B. Webstühle) und zur Übertragung von Texten (Telegraph). Aus diesen Bedürfnissen wurden mit den jeweils zur Verfügung stehenden Mitteln Geräte entwickelt, um die Berechnungen zu vereinfachen oder gar zu automatisieren.

1.1.1 Abakus

Die bekannteste Rechenhilfe ist der Abakus. In China wurden bereits im zweiten Jahrhundert vor Christus solche Geräte eingesetzt. Es handelt sich dabei um eine einfache Rechenhilfe für die 4 Grundrechenarten. Ein Abakus besteht aus einem Rahmen mit Stäben auf denen beweglichen Perlen stecken. Das Rechnen basiert auf einer Darstellung der Zahlen in einem Fünfer-System. Die Rechenoperationen werden dann durch Verschieben von Perlen - unter Berücksichtigung von Überträgen - durchgeführt. Geübte Benutzer erreichen damit eine sehr hohe Rechengeschwindigkeit.

1.1.2 Rechenschieber

Mit einem Rechenschieber oder Rechenstab kann man Multiplikation und Division durch Addition bzw. Subtraktion der entsprechenden Logarithmen realisieren. Der Rechenschieber wurde im 17. Jahrhundert entwickelt und beruht auf Arbeiten von Lord Napier, der 1614 ein Buch über Logarithmen veröffentlichte. Bis etwa 1975 wurde der Umgang mit Rechenschiebern noch in Schulen gelehrt.

1.1.3 Rechenmaschinen

Abakus und Rechenschieber sind lediglich Rechenhilfen. Sie vereinfachen den Rechenvorgang aber noch immer ist der Mensch der Ausführende. Im nächsten Schritt versuchte man, den Rechenvorgang selbst zu automatisieren. Der Benutzer stellt beispielsweise die beiden zu addierenden Zahlen ein. Durch Drehen einer Walze oder Zahnräder wird dann die Addition durchgeführt. Solche mechanischen Rechenwerke entwickelten u.a. Blaise Pascal (1641) und Gottfried Wilhelm Leibniz (1673). Erst ab dem 19. Jahrhundert konnte man die Teile in der benötigten Präzision herstellen. Daraus entwickelte sich ein Industriezweig zur Fertigung von mechanischen und später elektromechanischen Rechenmaschinen. Derartige Geräte wurden in kaufmännischen und technischen Büros bis etwa 1950 eingesetzt.

1.1.4 Lochkarten

Für selbständig ablaufende mechanische Geräte benötigt man einen „Programmspeicher“. Die erste Form eines solchen Speichers waren Lochkarten. Bereits im frühen 18. Jahrhundert erfunden, wurden sie von Joseph-Marie Jacquard (1752-1834) zu einer ausgereiften Technik zur Steuerung von Webstühlen entwickelt. Durch entsprechende Löcherkombinationen auf den Pappkarten wurde das Webmuster kodiert. Später wurde diese Technik auch zur Steuerung von Musikautomaten mittels Lochstreifen eingesetzt. Ein aktuelles Beispiel für dieses Verfahren ist in Bild 1.1 zu sehen.

Die Lochkarte war das erste digitale Speichermedium. Hermann Hollerith (1860-1929) benutzt dieses Konzept zur Auswertung von Daten aus der 11. amerikanischen Volkszählung (1886). Die zur Auswertung benötigte Zeit wurde auf ein Sechstel reduziert. Darauf aufbauend entwickelte er Maschinen zum Drucken, Kopieren, Sortieren und Tabellieren von Lochkarten. Derartige Systeme wurden oft für statistische und kaufmännische Aufgaben eingesetzt. Der Schwerpunkt liegt auf der großen Datenmenge, weniger auf den angewandten Operationen. Hollerith gründete 1896 die Firma „Tabulating Machine Company“, aus der 1924 die „International Business Machines Corporation (IBM)“ hervorging.

Später dienten Lochkarten und Lochstreifen als Ein- und Ausgabemedium für Computern. Bild 1.2 zeigt eine Lochkarte aus dem ersten Programmierkurs des Autors. Jede solche Karte enthielt eine Zeile Fortran-Code, wobei eine Spalte für ein Zeichen benötigt wurde.

1.1.5 Programmsteuerung

Der wesentliche Schritt zum „Computer“ ist die flexible Programmsteuerung. Als erster entwickelte der englische Mathematiker und Philosoph Charles Babbage (1792-1871) die Idee einer programmgesteuerten Rechenmaschine (analytic engine). Er verband das Konzept der Lochkarten mit den mechanischen Rechenauto-

Abbildung 1.1: Lochstreifen zur Steuerung eines Musikautomaten

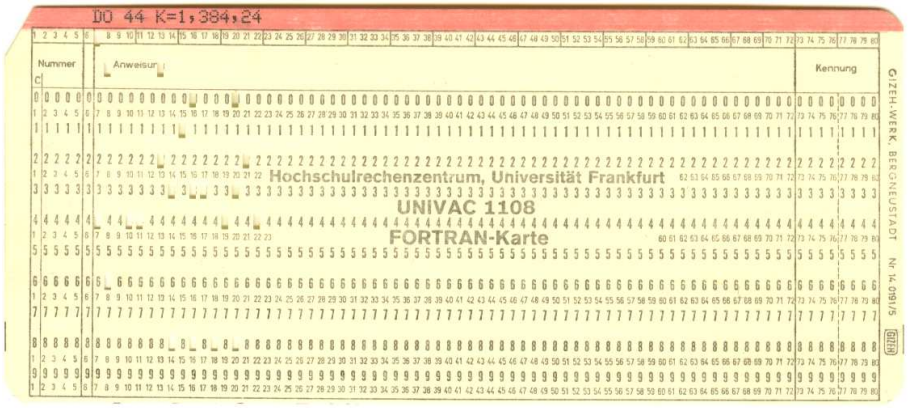


Abbildung 1.2: Beispiel einer Lochkarte mit einer Zeile eines Fortran-Programms

maten. Das Programm sollte auf Lochkarten gestanzt werden. Die Realisierung scheiterte allerdings an den unzureichenden technischen Möglichkeiten seiner Zeit. Babbage arbeitet mit Augusta Ada Byron, Countess of Lovelace (1815-1852) zusammen. Man kann sie als erste Programmiererin der Welt bezeichnen. Die Programmiersprache ADA ist nach ihr benannt.

1.2 Erste Computer

Etwa ab 1935 waren durch die Fortschritte in der Elektrotechnik die technischen Möglichkeiten zur Realisierung von Computern gegeben. Mit Relais und später Röhren standen Bauteile zum Aufbau von digitalen Schaltungen zur Verfügung. Beschleunigt wurde die Entwicklung durch Erfordernisse aus dem 2. Weltkrieg. In den USA war dies insbesondere die Entwicklung der Nukleartechnik. In Großbritannien wurden Rechner zur Entschlüsselung von Funkmeldungen entwickelt. Diese Rechner erreichten etwa 10000 Operationen pro Sekunde.

1.2.1 Konrad Zuse

„Ich bin zu faul zum Rechnen.“ Mitte der 30er Jahre des 20. Jahrhunderts plagte sich der angehende Bau-Ingenieur während seines Studiums an der damaligen Technischen Hochschule Charlottenburg mit Algorithmen für die statischen Berechnungen von Bauwerken. Aber Konrad Zuse hatte eine Vision. Er wollte die stupide Arbeit des Rechnens durch eine Maschine vornehmen lassen. Konrad Zuse hat seine erste Rechenmaschine, die Z1, von 1936 bis 1938 gebaut. Sie war eine mechanische Konstruktion und bestand aus ca. 40.000 Einzelteilen. Nach einem Zwischenexperiment mit der Rechenmaschine Z2 hat der Tüftler die Z3 vollständig mit 2500 telephonischen Relais in seiner Berliner Wohnung aufgebaut. Die Z3 war die erste programmierbare Rechenmaschine der Welt. Sie konnte die arithmetischen Grundoperationen in beliebigen Kombinationen ausführen und besaß einen Speicher für 64 Zahlen. Die Z3 war gleichzeitig die erste Maschine, der eine flexible Zahlendarstellung mit Exponenten (so genannte Gleitkommazahlen, Details der verschiedenen Zahlendarstellungen werden in Kapitel 5 behandelt) verwendete. Nach der Vorführung der Z3 im Jahr 1941 erhielt Zuse einen Auftrag für eine noch größere Maschine, die Z4, die bis 1945 fast vollendet wurde. Die Z1 und Z3 wurden 1943 im Krieg zerstört. Konrad Zuse gründete 1949 in Neukirchen/Hessen die Zuse KG, die später nach Bad Hersfeld verlagert wurde. Die Firma brachte eine Reihe von Rechnern auf den Markt, war aber mittelfristig wenig erfolgreich und wurde 1967 von Siemens übernommen.

1.2.2 Andere frühe Entwicklungen

Im Jahr 1939 baute George Stibitz (1904-1994) einen Rechner bei den Bell Laboratories in New York (BELL I) mit Relais und einem optimierten Rechenwerk zur Verarbeitung von komplexen Zahlen. Es war ein Spezialrechner zum Multiplizieren und Dividieren von komplexen Zahlen, der nicht programmierbar war. Es folgten die Modelle II bis V, die von 1943-1947 für die Landesverteidigung der USA entwickelt wurden. BELL V arbeitete in der Gleitkommadarstellung. Stibitz ist damit - wie Zuse schon 1936 - ebenfalls ein Pionier der Gleitkommarechnung im Binärsystem.

Der bis 1942 von Vincent Atanasoff gebaute Rechner ABC war ein nicht programmierbarer Spezialrechner in Röhrentechnik und basierte auf dem Binärprinzip. Der ABC kann als Prototyp des Parallelrechners angesehen werden. Zwar funktionierten die einzelnen Elemente des Rechners, aber die Maschine als ganzes wurde nicht fertig gestellt.

1944 vollendete Howard Aiken - unterstützt von der Firma IBM - die MARK I, die noch ein dezimales Rechenwerk verwendete und die Trennung von Speicher, Steuereinheit und Rechenwerk nicht kannte. Die MARK I war frei programmierbar. Es war eine gigantische Maschine mit 700.000 Einzelteilen und 35 Tonnen Gewicht.

Die 1945/46 fertiggestellte ENIAC (*Electronic Numerical Integrator And Computer*) von Eckert, Mauchly und Goldstine in den USA mit ihren ca. 18.000 Röhren benötigte eine Fläche von 150 qm und wog 30 Tonnen. Die Steuerung der Maschine (Programmierung) wurde durch das Stellen von Hunderten von Drehschaltern und das Stecken von Kabelverbindungen erreicht. Begleitend zu den Realisierungen entwickelte John von Neumann (1903-1957) das Grundkonzept moderner Rechner. Ein wesentliches Prinzip ist, dass das Programm genauso wie die Daten im Speicher abgelegt wird. Damit ist nicht mehr nur ein starrer Ablauf - wie etwa bei Lochkarten - möglich, sondern das Programm kann geändert werden und im Extremfall sogar sich selbst verändern.

Die in England von 1943-1945 gebauten COLOSSUS Rechner waren Spezialrechner mit Röhren im Binärprinzip. Sie waren nicht frei programmierbar und wurden erfolgreich zur Entschlüsselung von Funksprüchen der deutschen Wehrmacht eingesetzt. Angeblich auf Befehl Churchills wurden alle Maschinen und Unterlagen 1946 zerstört. In den Jahren 1939 bis 1945 wurde unter strengster Geheimhaltung an der Dekodierung der deutschen Funkmeldungen gearbeitet. Aufbauend auf Arbeiten des Polen Marian Rejewski konnten Verfahren zur Dechiffrierung der mit Enigma-Geräten verschlüsselten Nachrichten entwickelt werden. Allerdings war es immer noch notwendig, eine Vielzahl von Kombinationen auszuprobieren. Dazu wurden elektromechanische Rechengeräte und auch die mit Röhren aufgebauten COLOSSUS Rechner eingesetzt.

Ein wichtiges Mitglied der Gruppe war Alan Turing [1]. Der englische Ma-

[1] Alan Turing, englischer Mathematiker 1911-1952

thematiker hatte zuvor das Konzept einer universellen Maschine entwickelt, die mit primitiven Operationen alle berechenbaren Funktionen implementieren kann (1936). Die nach ihm benannte Turing-Maschine ist ein minimales Automatenmodell mit unbegrenztem Speicher. Die Turing-Maschine spielt in der theoretischen Informatik und insbesondere in der Theorie der Berechenbarkeit eine große Rolle. In einem Artikel von 1950 über künstliche Intelligenz entwickelt er weiterhin den Turing Test: Ein Computer kann „denken“, wenn man an seinen Antworten nicht mehr erkennen kann, dass sie von einem Computer stammen.

1.3 Rechnergenerationen

1.3.1 1. Computergeneration

Die Rechner mit Relais oder Röhren bezeichnet man als 1. Computergeneration. Solche Systeme waren groß, schwer, störanfällig und teuer, so dass die Einsatzmöglichkeiten eingeschränkt blieben.

1.3.2 2. Computergeneration

Etwa von 1955 bis 1965 werden Elektronenröhren zunehmend durch Transistoren ersetzt. Als Hauptspeicher dienen Magnetkernspeicher. Die ersten Magnetbänder (1952) und Magnetplatten (1956) werden angeboten.

1.3.3 3. Computergeneration (etwa bis 1970/75)

Mehrere Transistoren und Bauelemente werden zu Schaltgruppen zusammen gefasst. Die Rechner werden durch Betriebssysteme gesteuert. Rechenzentren entstehen. Einzelne Benutzer geben ihren Auftrag (job) ab und müssen auf die Aushändigung der Resultate warten. Typisch ist die Eingabe eines Programms auf Lochkarten. Später werden Terminals angeschlossen, an denen Benutzer Befehle direkt eingeben können. Da dann mehrere Programme gleichzeitig abgearbeitet werden, müssen die Betriebssysteme um *time sharing* Konzepte erweitert werden. Typisch ist, dass ein großer Rechner sehr viele Terminals gleichzeitig bedient. Die Herstellerfirmen bieten für die unterschiedlichen Ansprüche Computerfamilien (z.B IBM/360, UNIVAC 9000) von untereinander kompatiblen Geräten mit steigender Leistungsfähigkeit an.

1.3.4 4. Computergeneration

Dramatische Fortschritte in der Halbleitertechnik ermöglichen eine immer höhere Integrationsdichte. Die Miniaturisierung ermöglicht die Entwicklung von Mikrocomputern. Nach den zuerst auf dem Markt erschienenen Taschenrechnern wer-

den für Einzelpersonen auch vollwertige Heimcomputer erschwinglich. Der Markt für Computer teilt sich in

- Großrechner (main frame)
- Workstations
- PCs

Gleichzeitig beginnt die umfassende Vernetzung aller Rechner. Neue Konzepte zur Verteilung der Aufgaben - client server Konzept - lösen die zentralistischen Rechenzentrums-Strukturen teilweise ab. Wichtige Firmen für Workstations sind: Digital Equipment, sun, HP, Apollo, Silicon Graphics. Eine Workstation steht in der Regel für einen Benutzer alleine zur Verfügung. Mit der damit gewährleisteten Rechenleistung werden graphische Benutzeroberflächen möglich. Für Heimanwender werden einfache System angeboten. Als Massenspeicher dienen zunächst Kassetten und später Disketten. Einige Beispiel sind:

- Apple II mit 8 Bit Prozessor, 1977
- Commodore Pet
- Sinclair ZX81
- Commodore C64
- Atari ST

Auf den Mikrocomputern laufen spezielle Betriebssysteme. Software wird für Standardanwendungen (Textverarbeitung, Tabellenkalkulation, Graphik, etc.) angeboten, so dass solche Systeme auch vermehrt in Firmen eingesetzt werden. Daneben etablieren sie sich im Bereich von Freizeit und Unterhaltung (Spiele, Musikprogramme). Die Geräte der Firma Apple bieten frühzeitig graphische Benutzeroberflächen. In Deutschland ist der wesentlich preisgünstigere Atari ST sehr erfolgreich. Während es anfangs eine große Anzahl konkurrierender Systeme gibt, kommt es durch die Einführung des IBM PC (1981) zu einer starken Vereinheitlichung. Im Gegensatz zu den zumeist proprietären Systemen der anderen Anbieter, werden kompatible PCs von vielen Firmen angeboten.

Der IBM PC ist zum Zeitpunkt der Einführung weder in Hardware noch in Software den Konkurrenten überlegen. Aber durch das breite Angebot erobert die Kombination IBM PC mit MSDOS in den nächsten Jahren den Markt. Die Firma Microsoft (gegründet 1975) profitiert von der quasi Monopolposition als Betriebssystem-Anbieter für PCs und entwickelt sich zum Marktführer bei Anwendungsprogrammen. Andere Systeme (Apple, Amiga) werden zu Nischenprodukten. Mittlerweile sind PCs so leistungsfähig geworden, dass sie Workstations

aus vielen Bereichen verdrängen. Gleichzeitig erleben die schon totgesagten Großrechner eine gewisse Renaissance. Man musste feststellen, dass die Administration - Benutzerverwaltung, Datensicherung, Softwarepflege - für dezentrale Installationen mit vielen Einzelplatzrechnern einen sehr hohen Aufwand bedeutet. Dann ist es günstiger, zentrale Funktionen auf einem Großrechner zusammen zu fassen.

1.4 Internet

Wohl kaum eine andere Informationstechnologie hat einen vergleichbaren Einfluss auf unseren Alltag wie die Entwicklung des Internets. Ausgehend von der Vernetzung einiger weniger Rechner entwickelte sich in kurzer Zeit ein globales Netz als universelle Plattform für Information, Kommunikation und Unterhaltung. Einige Meilensteine waren:

- 1969: ARPANET ein Projekt der Advanced Research Project Agency (ARPA)
- 1971: Das ARPANET besitzt 15 Knoten. Telnet und FTP werden entwickelt.
- 1972: Ray Tomlinson entwickelt das erste E-Mail-Programm.
- etwa 1982: Umstellung auf TCP/IP
- 1987: Der Begriff Internet entsteht, es sind nun 27.000 Rechner vernetzt.
- 1989: Tim Berners-Lee entwickelt am CERN (bei Genf) das WWW
- 1994: Gründung des World Wide Web Consortium (W3C) als Gremium zur Standardisierung von Technologien

1.5 Übungen

Übung 1.1 *Welcher Mathematiker befasste sich als einer der ersten mit dem Thema* Künstliche Intelligenz *und entwickelte dazu ein nach ihm benanntes Testverfahren?*

Übung 1.2 *Welche der folgenden Programmiersprachen ist nach der ersten Programmiererin benannt?*
Ada, Algol, Java, Lisp, Pascal, Perl, Python

Kapitel 2

Physikalische Grundlagen

2.1 Atommodell

Ausgangspunkt der Betrachtungen ist das Bohrsche[1] Atommodell. Ein Atom besteht aus dem Kern – Protonen und Neutronen – und einer Hülle von Elektronen. Die Anzahl der Protonen bestimmt, um welches chemisches Element es sich handelt. Protonen tragen eine positive elektronische Ladung und Elektronen eine gleichgroße negative. Gegensätzliche Ladungen ziehen sich an. Diese Kraft hält die Elektronen auf ihren Bahnen um den Kern.

Nun lässt sich allerdings zeigen, dass nach den Regeln der klassischen Physik Elektronen bei ihrer Bewegung durch das elektrische Feld des Kerns Energie verlieren würden. Ohne weitere Annahme ist nicht zu erklären, warum die Elektronen nicht in den Kern stürzen, sondern vielmehr auf stabilen Bahnen bleiben. Niels Bohr löste das Problem durch die Annahme, dass Elektronen sich nur auf bestimmten, festen Bahnen aufhalten können. Es ist zwar möglich, unter Energieabgabe von einer höheren auf eine niedrigere Bahn zu springen oder umgekehrt durch Energieaufnahme auf eine höhere Bahn zu wechseln. Aber die Übergänge erfolgen nur zwischen den festen Bahnen, die Bereiche dazwischen sind nicht erreichbar. Die Bewegung auf einer Bahn erfolgt ohne Energieverlust. Jede Bahn kann genau ein Elektron aufnehmen.

Die verschiedenen Bahnen bilden Schalen. Die innerste Schale fasst 2 Elektronen, die nächste 8 Elektronen. Atome sind bestrebt, die Schalen möglichst aufzufüllen. Zu diesem Zweck können Atome chemische Bindungen mit anderen Atomen eingehen. Als Beispiel hat das Metall Natrium (Na) auf der äußersten Schale nur ein einzelnes Elektron. Demgegenüber fehlt dem Gas Chlor (Cl) ein Elektron. Eine Verbindung zwischen beiden entsteht, indem ein Natriumatom sein überzähliges Elektron an ein Chloratom abgibt. Beide Atome haben dann komplette Schalen. Das Resultat NaCl – besser bekannt als Kochsalz – ist eine stabile Verbindung.

[1] Niels Henrik David Bohr, dänischer Physiker, 1885-1962

2.2 Strom

Elektrischer Strom ist die Bewegung von geladenen Teilchen. In den meisten Fällen übernehmen Elektronen diese Rolle. Je nach dem, ob ein Material Strom leitet oder nicht, unterscheidet man zwischen Leitern und Nichtleitern (Isolatoren). Typische Leiter sind Metalle. Metalle haben wie das obige Beispiel Natrium nur wenige Elektronen in der äußersten Schale. Eine energetisch günstige Anordnung besteht aus einem festen Gitter der Atome, zwischen dem sich die überzähligen Elektronen als so genanntes Elektronengas frei bewegen.

Verbindet man einen Leiter – z.B. ein Stück Draht – mit einer Batterie, so passiert folgendes: Die Batterie hat an ihren Enden je einen positiven und einen negativen Pol. Der positive Pol zieht die Elektronen an und die freien Elektronen begeben sich auf den Weg zu diesem Pol. Ein elektrischer Strom fließt.

Anschaulich wirkt die Batterie wie ein Gefälle, das die Elektronen in Bewegung setzt. Die Höhe dieses Gefälles ist die Spannung U mit der Einheit Volt[2] (V). Die Stärke des Stroms ist durch die Anzahl der Elektronen, die pro Zeiteinheit einen Meßpunkt passieren, charakterisiert. So könnte man den Strom in „Anzahl von Elektronen pro Sekunde“ angeben. Tatsächlich verwendet man für die Menge an Ladung die Einheit Coulomb[3] (C). Ein Elektron trägt eine Ladung von $-1,603 \cdot 10^{-19}$ C. Die Einheit für den Strom ist Ampere[4] (A), wobei ein Ampere eine Ladungsmenge von einem Coulomb pro Sekunde bedeutet.

Übung 2.1 *Wie viele Elektronen fliessen bei einem Strom von 1,5 Ampere pro Sekunde durch den Leiter?*

Die Stärke des Strom hängt einerseits von der Verfügbarkeit der Elektronen und anderseits von der angelegten Spannung ab. Es gilt der einfache Zusammenhang des Ohmschen[5] Gesetzes

$$U = R \cdot I \tag{2.1}$$

wobei R den Widerstand (Einheit Ohm, Ω) bezeichnet. Der Widerstand selbst ist durch Materialeigenschaften sowie Länge und Querschnitt des Leiters bestimmt. Weiterhin spielt auch die Temperatur eine Rolle. Bei höhere Temperatur schwingen die Atomrümpfe im Gitter stärker. Damit behindern sie die Elektronen in ihrer freien Bewegung, der Widerstand steigt an.

2.3 Halbleiter

Zwischen Leitern und Isolatoren stehen die Halbleiter. Das Element Silizium als Beispiel verfügt auf der äußeren Schale über 4 Elektronen. Um die Schale aufzu-

[2] Allesandro Giuseppe Antonio Anastasio Graf von Volta, italienischer Pysiker 1745-1827
[3] Charles Augustin de Coulomb, französischer Wissenschaftler 1736 -1806
[4] André Marie Ampère, französischer Wissenschaftler 1775-1836
[5] Georg Simon Ohm, deutscher Mathematiker und Physiker, 1789-1854

füllen sind insgesamt 8 Elektronen notwendig. Allerdings wäre ein solcher großer Ausbau energetisch nicht besonders günstig. Als Alternative teilen sich die Atome ihre Elektronen. Jedes Atom sucht sich vier Nachbarn. Jedem Nachbarn stellt es ein Elektron zur Verfügung und darf umgekehrt eines seiner Elektronen nutzen. Zwischen zwei Nachbarn bildet sich damit ein Elektronenpaar. Ein Atom hat 4 Nachbarn und dementsprechend 4 Paare mit den insgesamt gewünschten 8 Elektronen.

Damit sind alle Elektronen gebunden und kein Strom kann transportiert werden. Allerdings ist die Bindung nicht allzu fest. So kann durch die thermische Bewegung der Atome eine Bindung aufbrechen, wodurch die beiden Elektronen als Ladungsträger frei werden. Die Bewegung nimmt mit steigender Temperatur zu und damit nimmt auch die Leitfähigkeit zu.

Bild 2.1 zeigt die Anordnung in einer vereinfachten, zweidimensionalen Darstellung. Das Siliziumkristall besteht aus ineinander geschachtelte Tetraeder. Jedes Atom hat vier gleich weit entfernte Nachbarn.

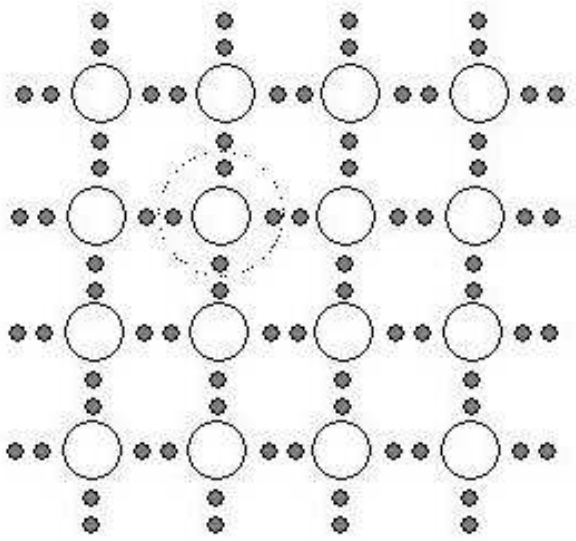


Abbildung 2.1: Anordnung von Siliziumatomen

Interessante Eigenschaften erzielt man, wenn man in ein solches Gitter eines Halbleiters gezielt Atome anderer Elemente einbaut (Dotierung). Betrachten wir als Beispiel Arsen (As). Dieses Element hat 5 äußere Elektronen. Setzt man ein Arsen-Atom in des Siliziumgitter, so werden aber nur 4 Elektronen für die Paarbildung mit den Nachbarn benötigt (Bild 2.2). Das überschüssige fünfte Elektron kann daher leicht von seinem Atom weg bewegt werden. Es steht damit für den Ladungstransport zur Verfügung. Da der Ladungsträger negativ geladen ist, spricht man von n-Leitung.

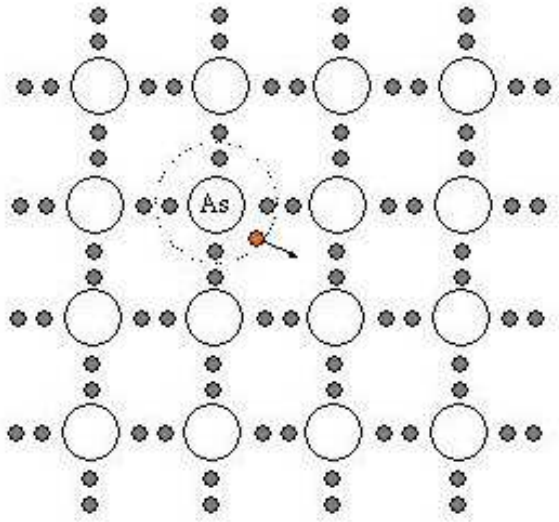


Abbildung 2.2: Dotierung mit einem Arsen-Atom

Interessanterweise funktioniert auch das umgekehrte Prinzip. Dazu dotiert man mit einem Element mit 3 äußeren Elektronen wie z.B. Gallium (Ga). Das jetzt fehlende Elektron macht sich als „Loch“ bemerkbar. Ein solches Loch kann durch ein Elektron eines Nachbarn aufgefüllt werden. Allerdings fehlt dann dem Nachbarn ein Elektron. Mit anderen Worten, das Loch ist scheinbar gewandert. Beim Anlegen einer Spannung wandern die Löcher in Richtung des negativen Pols. Formal entspricht das fehlende Elektron einer positiven Ladung. Dementsprechend wird ein entsprechendes Material als p-Leiter bezeichnet.

2.3.1 p-n-Übergang

Der große praktische Nutzen von Halbleitern ergibt sich aus den Kombinationsmöglichkeiten von p- und n-Leitern. Der einfachste Fall ist die Verbindung eines p- mit einem n-Leiter. An der Grenzschicht zwischen beiden Bereichen treffen überschüssige Elektronen auf Löcher. Die beiden ergänzen sich perfekt: das Elektron füllt das Loch aus. Damit sind beide verschwunden. Dieser so genannte Rekombinationseffekt führt dazu, dass im Grenzbereich die freien Ladungsträger verschwinden. Bild 2.3 zeigt diesen Effekt.

Was passiert nun, wenn man – beispielsweise durch Anschließen einer Batterie – an einen p-n-Übergang Spannung anlegt. Betrachten wir zuerst den in Abbildung 2.4 gezeigten Fall. Der positive Pol ist an den n-Leiter und der negative an den p-Leiter angeschlossen. Die ungleichen Ladungen ziehen sich an und die freien Ladungsträger - Elektronen und Löcher - wandern nach außen. In der Mitte weitet sich die Zone ohne freien Ladungsträger aus. Der Ladungstransport

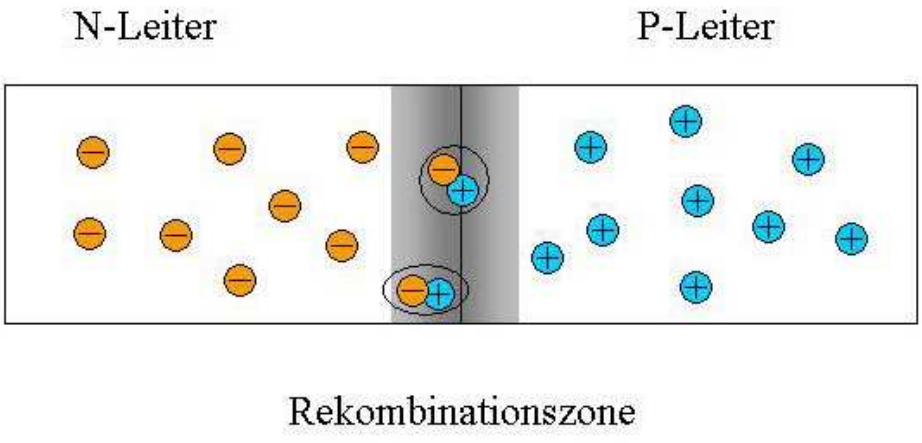


Abbildung 2.3: p-n-Übergang

kommt zum Erliegen - der p-n-Übergang wirkt als Sperre.

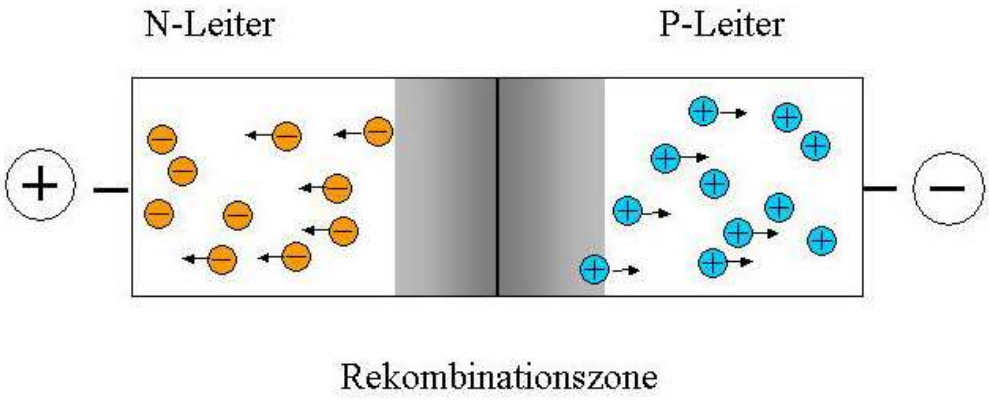


Abbildung 2.4: p-n-Übergang in Sperrrichtung

Bei umgekehrter Polung (Bild 2.5) fließen die Ladungsträger in die Mitte, wo Elektronen und Löcher sich gegenseitig aufheben. Damit können vom Rand her ständig neue Ladungsträger nachrücken - ein permanenter Ladungstransport kommt zustande. Bei dieser Polung wirkt der p-n-Übergang als Leiter.

Ein p-n-Übergang funktioniert demnach wie ein Ventil: in die eine Richtung lässt er Strom durch, in die andere blockiert er. Eine solche Diode (griech.: di zwei, doppelt; hodos Weg) funktioniert als Gleichrichter. Man verwendet Dioden beispielsweise, um die vom Elektrizitätswerk gelieferte Wechselspannung in eine Gleichspannung umzusetzen.

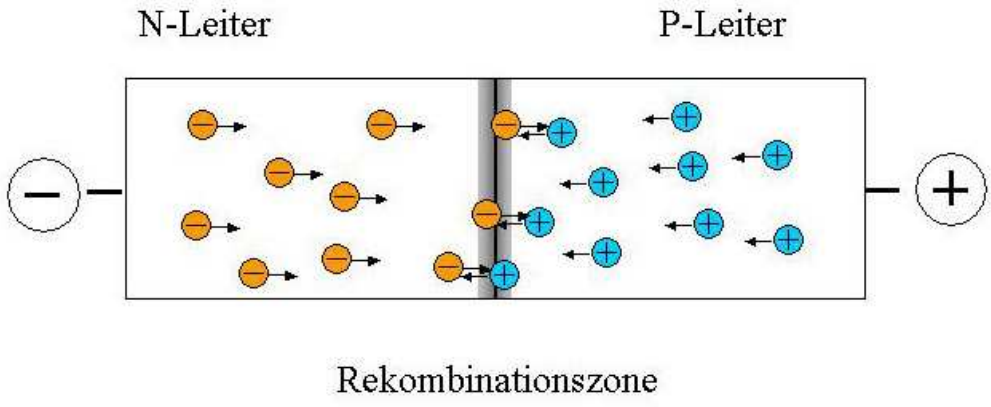


Abbildung 2.5: p-n-Übergang in Durchlassrichtung

2.3.2 Bipolartransistor

Durch Kombination zweier p-n-Übergänge kommt man zum Transistor. Genauer gesagt handelt es sich um den Bipolartransistor. Es gibt daneben andere Typen wie z.B. Feldeffekttransistoren.

Bild 2.6 zeigt den Aufbau eines Transistors als Kombination von n-, p- und n-Leiter. Dies entspricht zwei gegeneinander geschalteten Dioden. Ohne weitere Maßnahmen kann daher kein Strom über die gesamte Strecke fließen, da stets eine der Dioden als Sperre wirkt.

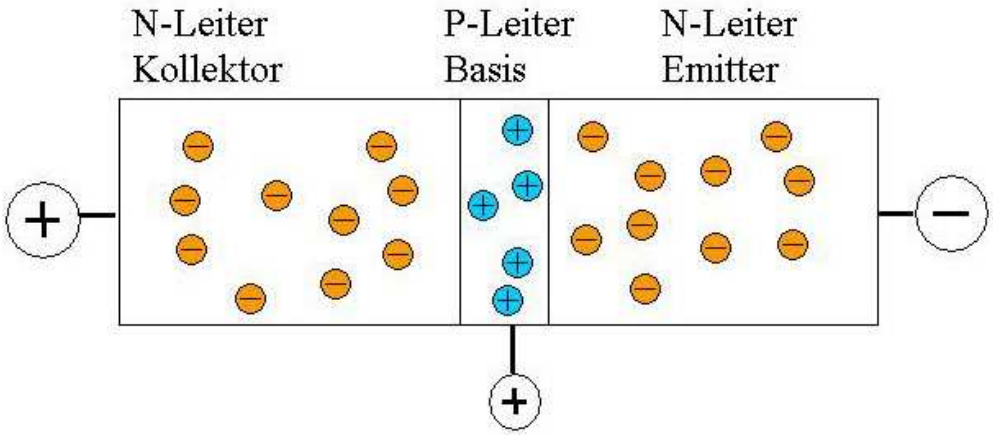


Abbildung 2.6: npn-Transistor

Legt man zwischen p-Leiter (Basis) und einem der n-Leiter (Emitter, lat. emittere: aussenden) eine Spannung in Durchlaßrichtung an, wird der Basis-Emitter-Kreis leitend. Elektronen gelangen in den Bereich der Basis. Wenn jetzt wiederum an dem zweiten n-Leiter (Kollektor lat. colligere: sammeln) eine positive

Spannung anliegt, können Elektronen auch in diese Richtung weiter gehen. Die Spannung an der Basis dient dazu, die erste Diode zu überwinden, so dass der Transistor insgesamt leitend wird. Durch geeignete Bauform sorgt man dafür, dass die Mehrzahl der Elektronen zum Kollektor gelangen und nur ein kleiner Teil an der Basis abfließen. Wie der Name sagt, sammelt der Kollektor die vom Emitter kommenden Elektronen ein.

Wichtig ist, dass mit einem kleinen Basisstrom ein großer Kollektorstrom gesteuert wird. Der Transistor arbeitet als Verstärker. Beispielsweise wird in einem Radio mittels Transistoren das schwache Signal von der Antenne für die Wiedergabe mit dem Lautsprecher verstärkt.

2.3.3 Analog- und Digitaltechnik

Die Eingangsspannung in z.B. einem HiFi-Verstärker wird um einen Faktor verstärkt. Dabei steckt die Information in der Größe einer Spannung. Entsprechend der Spannung wird auch die Lautstärke bei der Wiedergabe sein. Innerhalb des Wertebereichs ist jeder Wert möglich. Es handelt sich um ein analoges System.

Ein Beispiel für analoge Speicherung aus dem HiFi-Bereich ist die Schallplatte. Die Information über das Musiksignal steckt in der Form der Rille. Bei der Abtastung mit der Nadel wird aus der Weginformation ein entsprechendes elektrisches Signal erzeugt, das wiederum nach entsprechender Verstärkung auf dem Lautsprecher ausgegeben wird.

An diesem Beispiel erkennt auch die Schwächen der Analogtechnik. Jede kleine Veränderung an dem Datenträger wirkt sich als Störung im Signal aus. Die Qualität der Wiedergabe hängt von den Bauteilen in der Verarbeitungskette ab. Schwankungen beispielsweise infolge der Raumtemperatur beeinträchtigen die Qualität.

Daher beschreitet man bei der Digitaltechnik (lat. digitus: Finger) einen anderen Weg. Man unterschiedet im Signal nur noch zwei - oder im allgemeinen abzählbar viele - Werte. Jedem Wert ist ein Bereich zugeordnet. Beispielsweise kann man definieren:

- Spannung $<$ 2 Volt: Wert 0
- Spannung $>$ 4 Volt: Wert 1
- sonst: undefiniert

Kleine Schwankungen im Signal haben jetzt keinen Einfluss mehr auf den Wert. Auch wenn die Spannung etwas variiert, bleibt der Wert 0 oder 1 erhalten. Die Daten auf einer Musik-CD können - sehr zum Leidwesen der Musikindustrie - ohne Verlust an Genauigkeit ausgelesen und kopiert werden.

Für den Transistor spielt jetzt die Verstärkung keine Rolle mehr. Es wird nur noch unterschieden, ob eine Spannung an der Basis anliegt oder nicht. Der Transistor wird zu einem Schalter. Wir werden in den folgenden Kapiteln sehen, wie auf

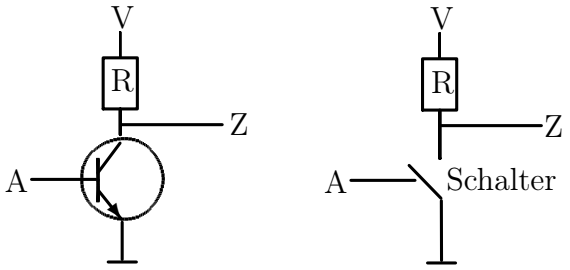


Abbildung 2.7: Transistor als Inverter und Ersatzschaltung

der Basis einer solchen einfachen Ja-Nein Entscheidung logische Verknüpfungen und Zahlendarstellungen entwickeln kann.

Bild 2.7 zeigt eine einfache Transistor-Schaltung. Dabei sind ein Widerstand und ein Transistor hintereinander (in Reihe) geschaltet. Zur Veranschaulichung ist im rechten Teilbild der Transistor durch einen Schalter ersetzt. Die Versorgungsspannung V entspricht dem Wert 1 während das andere Ende (Masse) den Wert 0 darstellt. Die Spannung am Punkt A - der Basis des Transistors - wird als Eingang und die Spannung am Punkt Z als Ausgang interpretiert.

Wenn am Punkt A keine Spannung anliegt, so ist der Schalter (Transistor) offen. Am Punkt Z ergibt sich dann die Spannung V, die dem Wert 1 entspricht. Wird an den Punkt A eine Spannung angelegt, so schließt sich der Schalter und ein Strom fließt. Durch den Widerstand wird der Strom begrenzt. Ohne eine solche Begrenzung würde der Strom zu groß werden und den Transistor zerstören. Über den geschlossenen Schalter ist der Punkt Z auf den Wert 0 gezogen. Insgesamt erhält man folgende Kombinationen:

A	Z
0	1
1	0

Die Schaltung wirkt als so genannter Inverter. Der Ausgang Z hat stets den zum Eingang A entgegengesetzten Wert. Bild 2.8 zeigt die Erweiterung um einen zweiten Transistor (Schalter). Beide Transistoren sind hintereinander - oder wie man sagt in Reihe - geschaltet.

Diese Schaltung hat zwei Eingänge A und B und einen Ausgang Z. Bei Eingängen gibt es insgesamt 4 verschiedene Kombinationen der Eingangswerte. Durch die Reihenschaltung fließt allerdings nur dann Strom, wenn beide Werte 1 sind. Nur in dieser Kombination hat der Ausgang den Wert 0. Die folgende Tabelle zeigt alle Kombinationsmöglichkeiten

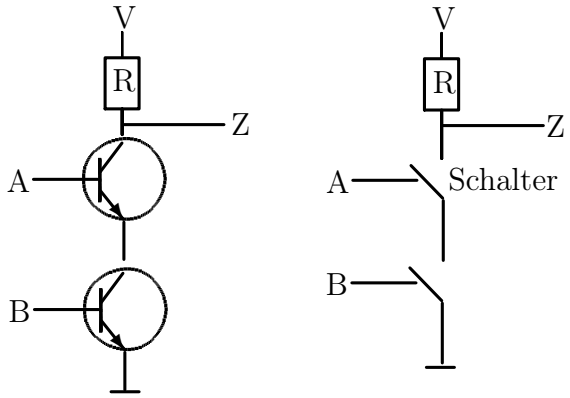
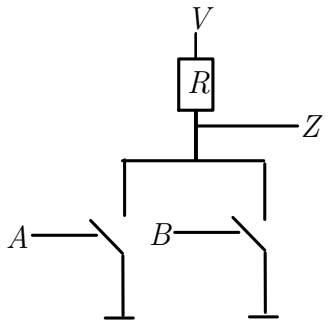


Abbildung 2.8: Reihenschaltung

A	B	Z	NICHT Z
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

Zur besseren Übersicht ist noch der invertierte Ausgang NICHT Z eingetragen. Dieser Wert ist genau dann 1, wenn beide Eingänge 1 sind. Dies ist die so genannte UND-Verknüpfung A UND B. Der nicht invertierte Ausgang Z hat die Funktionalität NICHT (A UND B).

Übung 2.2 *Parallel-Schaltung*
Welcher Zusammenhang besteht in der folgenden Schaltung zwischen Ausgang Z und den Eingängen A und B? Erstellen Sie eine Wertetabelle mit allen möglichen Belegungen. Welche anschauliche Interpretation hat der Zusammenhang?



2.3.4 Integrierte Schaltungen

Der rasante Fortschritt der Digitaltechnik beruht auf den Möglichkeiten, viele Bauteile auf kleinster Fläche zu integrieren. Eine solche integrierte Schaltung (engl. integrated circuit (IC)) besteht aus vielen Millionen Bauteilen - in erster Linie Transistoren. Das Grundprinzip ist relativ einfach. Man beginnt mit einem Träger aus Silizium (engl. wafer, Waffel). Durch gezielte Dotierungen werden dann n- und p-leitende Bereiche aufgebaut.

In einem Verarbeitungsschritt wird zunächst die ganze Oberfläche mit einer Abdeckschicht überzogen. Die zu dotierenden Bereiche werden durch Belichtung chemisch verändert. Durch anschließendes Ätzen werden nur die vorher belichteten Bereiche entfernt. Die nicht betroffenen Flächen bleiben durch den Lack abgedeckt. Bei der nachfolgenden Dotierung werden nur die freiliegenden Flächen erfasst. Das Verfahren wird dann für die nächste Dotierung wiederholt. Zusätzlich wirkt durch Oxidation gebildetes Siliziumoxid als Isolator und aufgedampfte Aluminiumbahnen verbinden die einzelnen Elemente.

Kapitel 3

Von Neumann Architektur

Wegweisend für die Entwicklung der Rechner waren die Arbeiten des Mathematikers John von Neumann[1]. Er entwarf das Konzept eines Universalrechners, dessen Struktur unabhängig von einem speziellen zu bearbeitenden Problem ist. Erforderlich dazu ist die Möglichkeit, für jedes Problem ein neues Programm in den Rechner zu laden. Indem das Programm im Speicher des Rechners abgelegt ist, ist man nicht mehr an die streng sequentielle Ausführung der einzelnen Befehle gebunden. Vielmehr kann man beispielsweise im Programm zurück springen und einen Abschnitt mit jeweils anderen Datenwerten mehrfach durchlaufen. Genauso kann man in Abhängigkeit von z. B. Zwischenergebnissen zu unterschiedlichen Programmteilen verzweigen. Diese Freiheit bedeutete einen wesentlichen Fortschritt gegenüber dem bis dahin üblichen, streng sequentiellen Abarbeiten von Programmen auf Lochstreifen.

Von Neumann ging noch einen Schritt weiter: ein Programm wird genau so wie die Daten behandelt. Die Programminformation liegt im gleichen Speicher wie die Daten. Es gibt keinerlei Trennung oder Unterscheidung zwischen beiden Kategorien. Der Inhalt einer Speicherzelle kann sowohl eine Zahl oder ein Befehl sein. Erst in Verbindung mit dem gesamten Ablauf wird entschieden, wie tatsächlich der Inhalt interpretiert wird. Damit ist es insbesondere möglich, dass ein Programm ein neues Programm erzeugt und dann dieses aufruft. Es ist sogar erlaubt, dass ein Programm sich selbst verändert.

Folgende Eigenschaften kennzeichnen das Modell:

- Ein Rechner besteht aus den Grundbestandteilen:
  - Zentraleinheit (Central Processing Unit, kurz CPU), manchmal wird noch zwischen Steuer- und Rechenprozessor unterschieden
  - Speicher
  - Ein-/Ausgabeeinheit (Input/Output, I/O)

[1] Johann von Neumann, 1903-1957

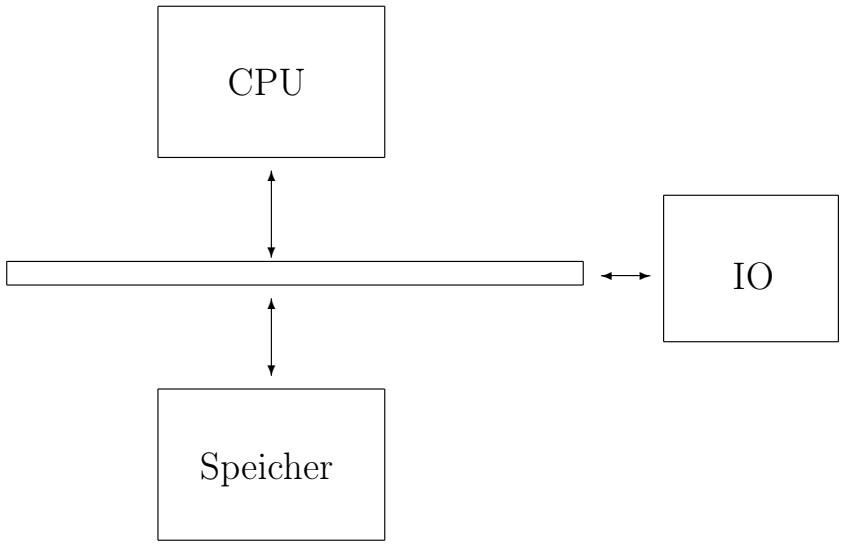


Abbildung 3.1: Von Neumann Architektur

Die Bestandteile sind untereinander durch Datenwege, so genannte Busse, verbunden (Bild 3.1).

- Die Struktur des Rechners ist unabhängig von einem speziellen zu bearbeitenden Problem. Man spricht daher von einem Universalrechner.
- Zahlen und Programme werden binär dargestellt.
- Programme sowie die Daten, die von den Programmen benötigt werden, sind in einem gemeinsamen Speicher abgelegt. Der Speicher besteht aus nummerierten Adressen, denen jeweils eine feste Anzahl von Bits (Wortlänge) zugeordnet sind.

Einige Konsequenzen:

- Alles – Daten und Programme – wird als Binärmuster abgespeichert. Man kann an dem Muster nicht erkennen, um was es sich handelt. Je nach Interpretation kann das selbe Muster einen Befehl oder eine Zahl darstellen. Auch die Speicherzelle „weiß“ nicht, welchen Typ ihr Inhalt hat.
- Programme können im laufenden Betrieb verändert werden. Dies kann gewollt sein oder durch einen unbeabsichtigten Zugriff aus Versehen erfolgen. Es besteht daher keine Garantie für die Integrität des Programms.

- Die Ausführung eines Befehls, der Speicherinhalte liest oder schreibt, benötigt mehrere sequentielle Zugriffe auf den Speicher.
- Die Architektur beinhaltet nur die unbedingt notwendigen Komponenten (*Prinzip des minimalen Hardware-Aufwands* oder *Prinzip des minimalen Speicher-Aufwands*).

Das Konzept war für die Möglichkeiten der Anfangszeit der Rechnertechnik ausgelegt. Aufwändigere Architekturen wie etwa getrennte Speicher für Programme und Daten waren mit der damaligen Technik nur schwer realisierbar und der Nutzen hätte den Mehraufwand kaum gerechtfertigt. Der Ansatz hatte weitreichende Folgen für die weitere Entwicklung. Er prägte zum einen die Hardware-Entwicklung. Zum anderen gab der durch den einheitlichen Datenbus bedingte sequentielle Ablauf auch die Randbedingungen für die Software vor. Viele Programmiersprachen folgten dem Paradigma des sequentiellen Ablaufs. Bemerkenswerterweise sind auch die Mehrzahl der modernen Rechner nach der von Neumann Architektur aufgebaut. Einige alternative Architekturen werden wir in einem späteren Kapitel untersuchen.

In den nächsten beiden Kapiteln werden wir zunächst die Grundlagen der binären Darstellung kennen lernen. Anschließend werden wir die verschiedenen Komponenten im Detail untersuchen.

Kapitel 4

Aussagelogik

4.1 Aussagen

Ein Aussage ist ein Satz, dem eindeutig ein Wahrheitswert zugeordnet werden kann.

> „Eine Aussage ist ein sprachliches Gebilde, von dem es sinnvoll ist, zu sagen, es sei wahr oder falsch.“ (Aristoteles, griechischer Gelehrter, 384-322 v. Chr.)

Beispiel 4.1 *Einige Aussagen:*
„Es regnet“
„5+7=13“

Übung 4.1 *Welche Sätze sind keine Aussagen? Können Sie Beispiele angeben?.*

Eine formale Definition ist:

Für eine Aussage A ist W(A) der Wahrheitswert mit		
W(A) :=	wahr,	falls Aussage A zutrifft
	falsch,	falls Aussage A nicht zutrifft

Mehrere Einzelaussagen können zu einer neuen Aussage verknüpft werden, für die sich wiederum ein Wahrheitswert bestimmen lässt.

Übung 4.2 *Tour de France:*
Wenn Eric Zabel bei den Zielwertungen die meisten Punkte sammelt, bekommt er dafür das grüne Trikot. Führt er allerdings auch in der Gesamtwertung, so trägt er das gelbe Trikot. Das grüne Trikot geht dann an den zweiten der Punktwertung[1]*. Wie sieht eine entsprechende Wahrheitstabelle aus?*
Aussage A1: Eric Zabel ist der Punktbeste

[1] Es gibt noch eine weitere Möglichkeit. Welche?

Aussage A2: Eric Zabel ist der gesamt Führende
Aussage A3: Eric Zabel trägt das grüne Trikot

W(A1)	*W(A2)*	*W(A3)*
falsch	*falsch*	
falsch	*wahr*	
wahr	*falsch*	
wahr	*wahr*	

Die Verknüpfung kann wie im Beispiel in einer so genannten Wahrheitstabelle dargestellt werden. Dabei trägt man alle möglichen Kombinationen der Einzelaussagen und das jeweilige Resultat in einer Tabelle ein. Ein Beispiel ist die UND-Verknüpfung. Sie besagt, dass die Gesamtaussage nur dann wahr ist, wenn jede Einzelaussage zutrifft.

Beispiel 4.2 *UND-Verknüpfung*
Zwei ist eine gerade Zahl UND drei ist eine ungerade Zahl

Die Wahrheitstabelle (auch Wertetabelle genannt) für UND mit zwei Einzelaussagen hat folgendes Aussehen:

W(A1)	W(A2)	W(A1 UND A2)
falsch	falsch	falsch
falsch	wahr	falsch
wahr	falsch	falsch
wahr	wahr	wahr

Vereinfachung der Schreibweise:
wahr wird durch 1 ausgedrückt (w, true)
falsch wird durch 0 ausgedrückt (f, false)
Wahrheitstabelle (Wertetabelle):

W(A1)	W(A2)	W(A1 UND A2)
0	0	0
0	1	0
1	0	0
1	1	1

Dies ist die UND-Verknüpfung (Konjunktion, AND), eine so genannte logische Operation. Man schreibt dafür:

$$\mathrm{W(\ A1\ und\ A2\ )} = \mathrm{W(A1)} \cdot \mathrm{W(A2)}$$

Andere Schreibweisen sind : $\mathrm{W(A1)} \cap \mathrm{W(A2)}$ und $\mathrm{W(A1)} \wedge \mathrm{W(A2)}$. Die logischen Konstanten sind 0 und 1. Eine logische Variable steht für ein Element aus der Menge $\{0, 1\}$, die Zuordnung heißt Belegung der Variablen (Beispiel: $a = 0$).

4.2 Grundverknüpfungen

Folgende Grundverknüpfungen zwischen logischen Variablen (und Konstanten) sind gebräuchlich:
Negation (NICHT, NOT): $\neg a$, $\overline{a}$ sprich: nicht a

a	$\overline{a}$
0	1
1	0

Konjunktion (UND, AND, Logisches Produkt): $a \cdot b$ sprich: a und b
Eine UND-Verknüpfung von 2 Variablen ergibt nur dann eine 1, wenn alle 2 Variablen gleich 1 sind. (Reihenschaltung von 2 Schaltern)

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

(Hinweis: Ähnlichkeit mit Zahlenmultiplikation)
Disjunktion (ODER, OR, Logische Summe): $a + b$ sprich: a oder b
Eine ODER-Verknüpfung von 2 Variablen ergibt dann eine 1, wenn eine Variable oder beide Variablen gleich 1 sind. (Andere Schreibweisen: W(A1) $\cup$ W(A2) und W(A1) $\vee$ W(A2))

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

Antivalenz (EXOR, exklusives ODER) $a \oplus b$ sprich: entweder a oder b
Wahr wenn nur eine der beiden Variablen wahr ist (nicht aber beide).

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Peircefunktion (NOR) sprich: weder a noch b
(entspricht negiertem ODER)

a	b	$\overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

Shefferfunktion (NAND) sprich: nicht beide a und b
(entspricht negiertem UND)

a	b	$\overline{a \cdot b}$
0	0	1
0	1	1
1	0	1
1	1	0

4.3 Ausdrücke

Verknüpfungen mit mehr als zwei Variablen oder Konstanten lassen sich mit den oben eingeführten Operationen darstellen. Eine solche Verknüpfung von endlich vielen Konstanten und logischen Variablen mittels Grundverknüpfungen heißt Ausdruck. Ausdrücke, in denen nur die Operationen Negation, UND und ODER verwendet werden, nennt man Boolesche Ausdrücke (George Boole, engl. Mathematiker 1815-1864, gilt als Begründer der mathematischen Logik).

Die Reihenfolge, in der die Operatoren angewendet werden, wird wie beim Rechnen mit Zahlen durch Vorrangregeln („Punktrechnung vor Strichrechnung“) und Klammern bestimmt. Es gilt insbesondere:

- Gleichrangige Operationen werden von links nach rechts berechnet
- geklammerte Verknüpfungen haben Vorrang vor
- Negation hat Vorrang vor
- Konjunktion (UND) hat Vorrang vor
- Disjunktion (ODER)

Negation wird oft durch Überstreichung dargestellt, die dann einfach auf mehrere Variablen oder Konstanten erweitert werden kann. Oft wird das Zeichen für die Boolsche Multiplikation weggelassen ($a \cdot b$ wird zu ab) Ein Ausdruck wird durch Angabe der Wertetabelle vollständig beschrieben:

Übung 4.3 *Ergänzen Sie die Wertetabelle für den Ausdruck* $a \cdot b + a \cdot \overline{c}$.

a	b	c	$a \cdot b$	$\overline{c}$	$a \cdot \overline{c}$	$a \cdot b + a \cdot \overline{c}$
0	0	0				
0	0	1				

4.4 Umformungen

Verschiedene Ausdrücke können äquivalent sein, d. h. bei gleicher Belegung der gemeinsamen Variablen nehmen sie gleiche Wahrheitswerte an. Man schreibt dann $a = b$. Es gibt in der Regel viele verschiedene Möglichkeiten, einen Zusammenhang zu formulieren.

Beispiel 4.3

$$a \cdot (b + \overline{b}) = a \cdot (c + \overline{c})$$

Übung 4.4 *Beweisen Sie durch Ausfüllen der Wahrheitstabelle diese Beziehung.*

a	b	c	$a \cdot (b + \overline{b})$	$a \cdot (c + \overline{c})$
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Für die Umformung gelten folgende Gesetze:
Negation der Negation

$$\overline{\overline{a}} = a$$

Kommutativgesetze

$$\begin{aligned} a \cdot b &= b \cdot a \\ a + b &= b + a \end{aligned}$$

Assoziativgesetze

$$\begin{aligned} a \cdot (b \cdot c) &= (a \cdot b) \cdot c \\ a + (b + c) &= (a + b) + c \end{aligned}$$

Distributivgesetze

$$\begin{aligned} a \cdot (b + c) &= (a \cdot b) + (a \cdot c) \\ a + (b \cdot c) &= (a + b) \cdot (a + c) \end{aligned}$$

Idempotenzgesetze

$$\begin{aligned} a \cdot a &= a \\ a + a &= a \end{aligned}$$

Komplementgesetze

$$\begin{aligned} a \cdot \overline{a} &= 0 \\ a + \overline{a} &= 1 \end{aligned}$$

0-1 Gesetze

$$\begin{aligned} a \cdot 1 &= a \\ a \cdot 0 &= 0 \\ a + 1 &= 1 \\ a + 0 &= a \end{aligned}$$

Absorptionsgesetze

$$\begin{aligned} a \cdot (a + b) &= a \\ a + (a \cdot b) &= a \\ (a \cdot b) + (a \cdot \overline{b}) &= a \\ (a + \overline{b}) \cdot b &= a \cdot b \\ (a \cdot \overline{b}) + b &= a + b \\ (a + b) \cdot (a + \overline{b}) &= a \end{aligned}$$

De Morgansche[2] Regel

$$\begin{aligned} \overline{(a \cdot b)} &= \overline{a} + \overline{b} \\ \overline{(a + b)} &= \overline{a} \cdot \overline{b} \end{aligned}$$

[2]Augustus De Morgan, englischer Mathematiker, 1806-1871

Beispiel 4.4 *Wir betrachten ein Fahrrad. Sei*
a : Vorderreifen platt
b : Hinterreifen platt
dann gilt für die Aussage c : „Fahrrad fährt“:
$c = \overline{a} \cdot \overline{b}$
(der Vorderreifen ist NICHT platt UND der Hinterreifen ist NICHT platt)
oder
$\overline{c} = a + b$
(Fahrrad fährt NICHT wenn Vorderreifen platt ODER Hinterreifen platt)
beziehungsweise
$c = \overline{a + b}$
(Fahrrad fährt wenn NICHT (Vorderreifen platt ODER Hinterreifen platt), (2. Morgansche Regel)
Der Beweis der Umformungsregeln kann jeweils durch Umformen oder Aufstellen der Wertetabellen erfolgen.

Beispiel 4.5 *2. Distributivgesetze* $a + (b \cdot c) = (a + b) \cdot (a + c)$

$$(a + b) \cdot (a + c) = a \cdot a + a \cdot c + b \cdot a + b \cdot c \textit{(Ausmultiplizieren)}$$

mit $a \cdot a = a = a \cdot 1$ *erhält man*

$$\begin{aligned} a \cdot 1 + a \cdot c + b \cdot a + b \cdot c &= \\ a \cdot (1 + c + b) + b \cdot c &= \textit{(Ausklammern)} \\ a \cdot 1 + b \cdot c &= \\ &= a + b \cdot c \end{aligned}$$

Häufig benutzte „Tricks“:

- $a = a \cdot 1$ um dann anschließend a ausklammern zu können
- $(1 + a + b + c + d) = 1$
- $0 \cdot a \cdot b \cdot c = 0$

Übung 4.5 *5. Absorptionsgesetz* $(a \cdot \overline{b}) + b = a + b$

a	b	$(a \cdot \overline{b})$	$(a \cdot \overline{b}) + b$	$a + b$
0	0			
0	1			
1	0			
1	1			

4.5 Boolesche Funktionen

Eine Boolesche (oder logische) Funktion ist eine Zuordnung zwischen der Belegung der Variablen und den Wahrheitswerten 0 und 1 (analog zur Funktionenlehre wie etwa $f(x) = \sin(x)$).

Definition 4.1 *Boolesche Funktion*
Eine n-stellige (vollständig definierte) Boolesche Funktion f wird definiert durch die Abbildung

$$f : \{0,1\}^n \to \{0,1\},$$

wobei $\{0,1\}^n$ die n-fache Kreuzmenge oder das n-fache kartesische Produkt der Menge $\{0,1\}$, also die Menge aller n-Tupel $(a_1, a_2, \ldots, a_n)$ mit $a_i \in \{0,1\}$ für alle $i = 1, \ldots, n$ ist. Mit anderen Worten: jeder mögliche Kombination von n binären Werten wird ein ebenfalls binärer Funktionswert zugewiesen. Die Funktion kann als logischer Ausdruck oder in Form einer 2^n-zeiligen Wertetabelle angegeben werden.

Es ist offensichtlich, wie man aus einer gegebenen Funktion die Wertetabelle aufbauen kann. Häufig stellt sich aber das umgekehrte Problem: Die Wertetabelle ist vorgegeben und dazu soll ein Ausdruck gefunden werden. Damit stellt sich die Frage: Wie kommt man von einer Wertetabelle zu einem logischen Ausdruck?

Ein Verfahren basiert auf den so genannten Mintermen. Minterme sind Terme die nur für genau einen Eingangstupel den Wert 1 ergeben. Dies wird erreicht, indem alle Argumente mittels UND verknüpft werden, wobei Argumente mit dem Wert 0 negiert werden.

Beispiel 4.6 *Minterm für $n = 4$*
Zu dem Eingang 0 0 1 1 gehört der Minterm $\overline{a_1} \cdot \overline{a_2} \cdot a_3 \cdot a_4$. Dieser Ausdruck ist für den Eingangstupel wahr und für alle anderen Tupel falsch.

Für $n = 3$ erhält man folgende Tabelle (wieder mit den Bezeichnungen a, b und c für die Eingangsvariablen):

a	b	c	Minterme = 1
0	0	0	$\overline{a}\overline{b}\overline{c}$
0	0	1	$\overline{a}\overline{b}c$
0	1	0	$\overline{a}b\overline{c}$
0	1	1	$\overline{a}bc$
1	0	0	$a\overline{b}\overline{c}$
1	0	1	$a\overline{b}c$
1	1	0	$ab\overline{c}$
1	1	1	abc

Damit kann man für eine gegebene Wertetabelle die Funktion x in der disjunktiven Normalform (DNF) als ODER-Verknüpfung aller Minterme angeben.

Beispiel 4.7 *Wertetabelle*

a	b	c	x	Minterm
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{a}bc$
1	0	0	1	$a\overline{b}\overline{c}$
1	0	1	0	
1	1	0	0	
1	1	1	1	abc

Die zugehörige disjunktiven Normalform lautet:

$$x = \overline{a}bc + a\overline{b}\overline{c} + abc$$

Die alternative Darstellung beruht auf den Maxtermen: Termen die nur für genau einen Eingangstupel den Wert 0 ergeben. Dazu werden alle Argumente ODER verknüpft, wobei Argumente mit dem Wert 1 negiert werden. Beispielsweise gehört zu dem Eingang 0 0 1 der Maxterm $a + b + \overline{c}$ Die konjunktiven Normalform (KNF) wird nach der folgenden Vorschrift konstruiert: Für jede Zeile mit x = 0 wird der Maxwert gebildet, anschließend werden alle Maxterme mit UND verknüpft.

Beispiel 4.8 *Wertetabelle*

a	b	c	x	Maxterm
0	0	0	0	$a + b + c$
0	0	1	0	$a + b + \overline{c}$
0	1	0	0	$a + \overline{b} + c$
0	1	1	1	
1	0	0	1	
1	0	1	0	$\overline{a} + b + \overline{c}$
1	1	0	0	$\overline{a} + \overline{b} + c$
1	1	1	1	

Die zugehörige konjunktive Normalform lautet:

$$x = (a + b + c)(a + b + \overline{c})(a + \overline{b} + c)(\overline{a} + b + \overline{c})(\overline{a} + \overline{b} + c)$$

Die Betrachtungen zeigen, dass man aus einer beliebigen Wertetabelle stets eine Normalform ableiten kann Anschaulich ist damit gezeigt, dass die Operationen (NICHT, UND, ODER) vollständig sind, d. h. jede Boolesche Funktion kann unter Verwendung dieser drei Operationen dargestellt werden. Weiterhin können gemäß der De Morganschen Regel UND und ODER ineinander überführt werden. Damit bilden sowohl (NICHT, UND) als auch (NICHT, ODER) bereits ein vollständiges System.

Übung 4.6 *Für welche Sonderfälle gibt es entweder nur disjunktive oder konjunktive Normalforme?*

Übung 4.7 *Zeigen Sie für (NAND) oder (NOR), dass die Operation alleine ein vollständiges System bildet.*

4.6 Vereinfachung von Schaltfunktionen

Bei der technischen Umsetzung von logischen Funktionen spricht man von Schaltfunktionen. Die logischen Operationen werden dann durch entsprechende Schaltungen realisiert.

Die Minimierung (Vereinfachung) von Schaltfunktionen hat für die Informationsverarbeitung große Bedeutung. Bei der Realisierung von Schaltfunktionen mit Gatterbausteinen sucht man einen minimalen Booleschen Ausdruck für die gegebene Funktion. Diese Vorgehensweise führt zu kostengünstigen Schaltungen, da die Kosten von der Komplexität des Booleschen Ausdrucks abhängen. Ziel der Minimierungsverfahren ist es, Schaltfunktionen mit möglichst geringen Kosten (Aufwand) zu realisieren. Dazu werden redundante Terme und Variablen entfernt. Für die Vereinfachung von Schaltfunktionen können verschiedene Verfahren verwendet werden:

- Anwendung der Axiome und Rechenregeln der Boolschen Algebra
- Graphische Verfahren, zum Beispiel das Minimierungsverfahren mittels Karnaugh-Veitch-Diagrammen (KV-Diagrammen). Ein Applet zu KV-Diagrammen wurde in der Studienarbeit [Mey98] entwickelt (http://tech-www.informatik.uni-hamburg.de/applets/kvd/).
- Tabellarische Verfahren, zum Beispiel das Minimierungsverfahren nach Quine-McCluskey

Für den praktischen Einsatz müssen allerdings auch technische Randbedingungen berücksichtigt werden. So kann es beispielsweise vorteilhaft sein, nur bestimmte Typen von Operationen zu verwenden. Früher wurden Schaltungen aus einzelnen ICs aufgebaut. Dann stellt sich das besondere Optimierungsproblem, möglichst wenige ICs zu verwenden. Ein übliches IC enthält z. B. 4 NAND Elemente. Das Optimierungsproblem besteht dann darin, die Elemente in den ICs möglichst gut auszunutzen. In diesem Fall ist beispielsweise eine Realisierung mit 4 NAND Elementen günstiger als eine mit einem NAND und einem NOR Element.

4.7 Übungen

Übung 4.8 *Wertetabelle*
Erstellen Sie die Wertetabelle für den Ausdruck

$$a \cdot \overline{b} + \overline{a} \cdot b + b \cdot \overline{c}$$

(Es können mehr Spalten als benötigt vorhanden sein.
Schreibweise: · für UND, + für ODER)

a	b	c						

Übung 4.9 *Verkaufsentscheidung*
Ein Anleger kauft Aktien der XYZ AG. Er wählt folgende Strategie: Falls sich XYZ schlechter als der DAX entwickelt, wird die Aktie verkauft, sobald sich der Kurs um mehr als 10% von dem Einkaufswert entfernt hat. Ansonsten - wenn sich XYZ gleich gut oder besser als der DAX entwickelt - wird sie verkaufen, wenn der Wert des gesamten Depots (inklusive anderer Aktien) unter 3 Millionen Euro fällt.

- *Welches sind die relevanten Einzelaussagen? Stellen Sie eine Wahrheitstabelle für die Verkaufsentscheidung auf.*
- *Geben sie die zugehörige disjunktive Normalform an.*
- *Welche Vereinfachungen bieten sich an?*

Übung 4.10 *Kraftwerküberwachung*
Ihre Firma betreibt 3 Kraftwerke, die jeweils eine Leistung von 10000 kW liefern. Jede Stunde werden die aktuellen Leistungsdaten gemeldet. Um den ordnungsgemäßen Betrieb zu überwachen, betrachten Sie zwei Kriterien:

1. *Steigt in mindestens einem Kraftwerk die Leistung über 11000 kW?*
2. *Fällt bei mindestens zwei Kraftwerken die Leistung unter 9900 kW?*

Erstellen Sie für jedes der beiden Kriterien eine Wahrheitstabelle. Verwenden Sie dazu im ersten Fall die Aussagen $L_i > 11000kW$ und im zweiten Fall $L_i < 9900kW$ wobei L_i die Leistung im Kraftwerk i bezeichnet.

1.

L_1	L_2	L_3	*Alarm*

2.

L_1	L_2	L_3	*Alarm*

Übung 4.11 *Logische Knobelei*
Man kann die Methoden der Aussagelogik auch zur Lösung von Knobelaufgaben einsetzen. Dazu zerlegt man die Aufgabe in einfache Aussagen. Die verschiedenen Kombinationen und ihre Bewertung werden in einer Wahrheitstabelle zusammen gestellt. Durch ein solches systematisches Vorgehen lässt sich ein kompliziertes Problem leichter lösen.

Betrachten Sie als Beispiel folgende Aufgabe: In einem alten Rätsel kommt ein Bauer mit Gans, Hund und einem Sack Korn an einen Fluss. In dem Fährboot kann er jeweils nur einen Teil mitnehmen. Aber er darf niemals Gans und Hund oder Gans und Korn zusammen unbeaufsichtigt lassen (und schon gar nicht alle drei). Wie kann er das Problem lösen?

- *Betrachten Sie die 4 Aussagen A1:*Bauer am linken Ufer*, A2:*Gans am linken Ufer*, A3:*Hund am linken Ufer *und A4:*Korn am linken Ufer*. A1 WAHR besagt dann, dass der Bauer am linken Ufer ist während er im Fall FALSCH sich auf der rechten Seite befindet. Die Anfangssituation ist WAHR WAHR WAHR WAHR oder kurz geschrieben 1111.*
- *Stellen Sie die Wahrheitstabelle für die erlaubten Kombinationen auf.*
- *Teilen Sie die legalen Kombinationen in zwei Gruppen gemäß der Position des Bauerns (linkes oder rechtes Ufer).*
- *Ein Überfahrt ist jetzt ein Übergang zwischen zwei Kombinationen in den beiden Gruppen. Dabei darf sich nur maximal eine Position ändern.*
- *Verfolgen Sie ausgehend von der Anfangssituation 1111 die möglichen Abfolgen bis zum Ziel 0000.*

Kapitel 5

Zahlendarstellung

5.1 Einleitung

Grundlage der Digitalrechner ist ein Element mit zwei Zuständen: Strom fließt oder Strom fließt nicht bzw Speicherzelle ist besetzt oder Speicherzelle ist nicht besetzt. Praktische Realisierungen sind

- Strom durch Röhre
- Strom durch Transistor
- Loch in Lochstreifen
- Loch in CD
- Magnetisierung auf Magnetplatte
- u.s.w.

Die Information eines solchen Elements (eine JA/NEIN Entscheidung) bezeichnet man als 1 bit (abgeleitet von binary digit). Alle Daten in einem Computer basieren auf dieser elementaren Darstellung. Einfache Zahlen oder MPEG-Videos werden intern als eine mehr oder weniger große Anzahl von Ja/Nein Entscheidungen behandelt. Im folgenden wird die Darstellung von Zahlen behandelt.

Indem wir mit unsere Fingern Zählen und Rechnen lernen, ist unsere anschauliche Zahlenvorstellung wesentlich durch das Dezimalsystem geprägt. Sprachlich ist dies noch in den Ausdrücken wie Digital oder engl. digit zu erkennen, die auf das lateinische *digital*, den Finger betreffend, zurück gehen. In den meisten Sprachen haben alle ganzen Zahlen bis 10 einschließlich und die Zehnerpotenzen einen eigenen Namen. Die Namen der übrigen Zahlen werden daraus zusammengesetzt (z.B. Achthundert Neun und Dreissig). Es gibt allerdings Ausnahmen:

- elf und zwölf haben eigene Namen

- Basis 60 für Bestimmung der Zeit und der Winkelberechnung
- Datumsangaben: 12 Monate, unterschiedliche Tage pro Monat, 12 bzw. 24 Stunden pro Tag
- 80er System in Frankreich (4 x 20)
- bei einigen Völkern gibt es Zählsysteme auf Basis 5
- Einige Völker zählen: Eins, Zwei, Zwei-Eins, Zwei-Zwei, viele

Rechnen in solchen Systemen ist schwierig:

- Der Nachtzug fährt um 22:45 ab und erreicht sein Ziel nach 8 Stunden und 33 Minuten
- Die Frist endet 50 Tage nach dem 22. August

5.2 Stellenwertsysteme

Das Rechnen wird einfacher, wenn man ein Stellenwertsystem (Positionssystem) benutzt. Dabei werden Zahlen mit Ziffern dargestellt, wobei die Wertigkeit einer Ziffer von der jeweiligen Stelle abhängt. Diese Systeme haben in der Praxis die größte Bedeutung. Ein Gegenbeispiel ist das römische Zahlensystem. Die wichtigsten Stellenwertsystem sind polyadische Systeme, bei denen Zahlen durch eine Potenzreihe dargestellt sind:

$$Z = a_{n-1}B^{n-1} + a_{n-2}B^{n-2} + \ldots + a_1B + a_0 \tag{5.1}$$

mit

Z	Zahl
a_i	Ziffer an der Stelle i aus der Menge $M = 0, 1, \ldots, B-1$
B	Basis des Stellenwertsystems

Beispiel 5.1 *Dezimalsystem:*

$$\begin{aligned} B &= 10 \\ M &= \{0,1,2,3,4,5,6,7,8,9\} \end{aligned}$$

Erweiterung der Form (5.1) auf Zahlen mit Nachkommastellen führt zu der Verallgemeinerung

$$\begin{aligned} Z &= \pm(a_{n-1}B^{n-1} + a_{n-2}B^{n-2} + \ldots + a_1B^1 + a_0B^0 + a_{-1}B^{-1} + \ldots + a_{-m}B^{-m}) \\ &= \pm(a_{n-1}a_{n-2}\ldots a_1a_0, a_{-1}\ldots a_{-m})_B \end{aligned}$$

für eine Zahl mit

n Vorkommastellen (Vorpunktstellen)
m Nachkommastellen

Das jeweilige Zahlensystem wird markiert, indem der Wert der Basis unten rechts an die Zahl angehängt wird. Ist keine Basis explizit angegeben, so handelt es sich um eine Zahl im Dezimalsystem.

Beispiel 5.2 *$Z = 105,87_{10}$:*

$$\begin{aligned} & n = 3, m = 2 \\ & a_2 = 1,\ a_1 = 0,\ a_0 = 5,\ a_{-1} = 8,\ a_{-2} = 7 \\ Z &= a_2 \cdot B^2 + a_1 \cdot B^1 + a_0 \cdot B^0 + a_{-1} \cdot B^{-1} + a_{-2} \cdot B^{-2} \\ &= 1 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0 + 8 \cdot 10^{-1} + 7 \cdot 10^{-2} \end{aligned}$$

Übung 5.1 *Nennen Sie ein Beispiel für ein nicht polyadisches Stellenwertsystem.*

Wir sind an das Dezimalsystem gewöhnt und das Rechnen darin ist uns vertraut. Demgegenüber ist für ein System mit nur zwei Zuständen das Dualsystem – das System mit der Basis 2 – passend. Im Dualsystem werden nur noch die Ziffern 0 und 1 verwendet. Jede Zahl wird dann als Summe über Zweierpotenzen ausgedrückt.

Beispiel 5.3

$$\begin{aligned} 1001.1_2 &= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} \\ &= 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 + 1 \times 0.5 \end{aligned}$$

Auf diese Art und Weise können Zahlen in einem Rechner dargestellt werden. Die Ziffern werden in die einzelnen Speicherzellen geschrieben. Bei einem Rechner mit 8 Bit Architektur kann man maximal 8 Stellen in einem Wort unterbringen. Beschränkt man sich auf ganze, positive Zahlen, so ist damit der Bereich

$$[0, 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0]$$

darstellbar. Die nächstgrößere Zahl 2^8 würde eine neunte Stelle benötigen. In Dezimaldarstellung ergibt sich der Zahlenbereich von 0 bis $2^8 - 1 = 255$.

5.3 Konvertierung zwischen Zahlensysteme

Da wir die Potenzen der Basis im Dezimalsystem angegeben haben, ist die Umwandlung der Zahl in das Dezimalzahl eine einfache Addition.

$$\begin{aligned} 1001.1_2 &= 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 + 1 \times 0.5 \\ &= 8 + 1 + 0.5 \\ &= 9.5_{10} \end{aligned}$$

Um umgekehrt eine Zahl aus dem Dezimalsystem in das Dualsystem zu wandeln, muss man sie in die Potenzen von 2 zerlegen. Eine Möglichkeit dazu ist die Potenzmethode. Das Verfahren beruht darauf, dass man die höchstwertige Ziffer durch Division mit der höchsten Potenz bestimmen kann. Für

$$Z = a_{n-1}B^{n-1} + a_{n-2}B^{n-2} + \ldots + a_1B + a_0$$

erhält man

$$\begin{aligned} Z/B^{n-1} &= a_{n-1}B^{n-1}/B^{n-1} + a_{n-2}B^{n-2}/B^{n-1} + \ldots + a_1B/B^{n-1} + a_0/B^{n-1} \\ &= a_{n-1} + a_{n-2}/B + \ldots + a_1/B^{n-2} + a_0/B^{n-1} \end{aligned}$$

Da a_i stets kleiner als B ist, sind alle Terme außer dem ersten kleiner als 1. Dadurch erhält man a_{n-1} aus Z durch ganzzahlige Division (Division ohne Rest) mit B^{n-1}. Zieht man anschließend $a_{n-1}B^{n-1}$ von Z ab, so kann man aus dem Rest die nächste Ziffer bestimmen. Für eine beliebige Basis B kann man den Algorithmus wie folgt formulieren:

1. Eingabe der zu konvertierenden Zahl Z, der gewünschten Stellenzahl n und der Basis B
2. $j = n - 1$
3. durch ganzzahlige Division mit B^j erhält man $a_j = Z/B^j$
4. von der Zahl Z wird a_jB^j subtrahiert $Z = Z - a_jB^j$
5. $j = j - 1$
6. falls j größer oder gleich 0 ist, wird ab Schritt 3) der Rechenablauf wiederholt
7. Ausgabe der konvertierten Zahl $(a_{n-1}a_{n-2}\ldots a_1a_0)_B$

Beispiel 5.4 *Konvertierung der Zahl* 21_{10} *in das Dualsystem,* $n = 8, B = 2$

j	*Division*	*Ergebnis*	*Subtraktion*	*Ziffernwert*
7	$21/2^7$	0	$21 - 0 \times 2^7 = 21$	$a_7 = 0$
6	$21/2^6$	0	$21 - 0 \times 2^6 = 21$	$a_6 = 0$
5	$21/2^5$	0	$21 - 0 \times 2^5 = 21$	$a_5 = 0$
4	$21/2^4$	1	$21 - 1 \times 2^4 = 5$	$a_4 = 1$
3	$5/2^3$	0	$5 - 0 \times 2^3 = 5$	$a_3 = 0$
2	$5/2^2$	1	$5 - 1 \times 2^2 = 1$	$a_2 = 1$
1	$1/2^1$	0	$1 - 0 \times 2^1 = 1$	$a_1 = 0$
0	$1/2^0$	1	$1 - 1 \times 2^0 = 0$	$a_1 = 1$

Ergebnis: $21_{10} = 00010101_2$

Für kleine Zahlen kann man dieses Verfahren leicht wie folgt ausführen:

Beispiel 5.5 *Bestimmung der Binärdarstellung von* 11_{10}*:*
eine 8 *Rest* $11 - 8 = 3$
keine 4
eine 2 *Rest* $3 - 2 = 1$
eine 1
also $11_{10} = 1011_2$

Während man bei der Potenzmethode zunächst die höchstwertige Ziffer bestimmt, kann man mit der so genannten Restwertmethode die Umwandlung mit der jeweils niedrigwertigste Ziffer beginnen. Dazu formt man die Zahlendarstellung zunächst nach dem Horner-Schema[1] durch fortgesetztes Ausklammern um:

$$\begin{aligned} Z &= a_{n-1}B^{n-1} + a_{n-2}B^{n-2} + \ldots + a_1B + a_0 \\ &= (a_{n-1}B^{n-2} + a_{n-2}B^{n-3} + \ldots + a_1)B + a_0 \\ &\vdots \\ &= (\ldots a_{n-1}B + a_{n-2})B^{n-2} + \ldots + a_2)B + a_1)B + a_0 \end{aligned}$$

Division durch B liefert jetzt einen ganzzahligen Anteil

$$Z/B = (\ldots (a_{n-1}B + a_{n-2})B^{n-2} + \ldots + a_2)B + a_1$$

und den Rest a_0. Durch fortgesetzte Division des jeweils verbleibenden ganzzahligen Anteils lassen sich so nacheinander die Ziffern als Restwerte bestimmen.

Beispiel 5.6 *Konvertierung der Zahl* 23_{10} *in das Dualsystem mittels Restwertmethode.*

	Division	*ganzzahliger Anteil*	*Rest*	*Ziffernwert*
0	23/2	11	1	$a_0 = 1$
1	11/2	5	1	$a_1 = 1$
2	5/2	2	1	$a_2 = 1$
3	2/2	1	0	$a_3 = 0$
4	1/2	0	1	$a_4 = 1$

Ergebnis: $23_{10} = 10111_2$

[1] William George Horner, englischer Mathematiker 1786-1837

5.4 Oktal- und Hexadezimalsystem

Das Dualsystem bietet die natürliche Darstellung für Zahlen in Computern. Ein Nachteil ist allerdings die recht große Anzahl von Stellen, durch die eine Zahl schnell unübersichtlich wird. Beispielsweise wird bei modernen Rechnern 32 oder 64 Bit (Stellen) Wortbreite benutzt. Für eine kompaktere Darstellung benutzt man daher Zahlensysteme mit der Basis 8 oder 16, den beiden Zweierpotenzen 3 und 4. Für das Oktalsystem mit der Basis 8 gilt:

$$\begin{aligned} B &= 8 \\ M &= \{0, 1, 2, 3, 4, 5, 6, 7\} \end{aligned}$$

und für das Hexadezimalsystem oder Sedezimalsystem

$$\begin{aligned} B &= 16 \\ M &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\} \end{aligned}$$

Durch die Wahl einer Zweierpotenz als Basis entspricht eine Ziffer in dem Zahlensystem immer einer festen Anzahl von Stellen im Dualsystem. So ergeben 3 Bit beziehungsweise 3 Stellen der Zahl im Dualsystem einer Stelle im Oktalsystem. Dadurch ist die Umrechnung vom Dualsystem in eines der beiden Systeme sehr einfach, da immer Gruppen von 3 oder 4 Bit zusammen gefasst (3er bzw. 4er Bündel) werden. Man braucht dann nur noch die Zuordnung zu der Gruppen zu den Ziffern zu kennen (oder nachzuschauen). Die Werte für die 3er bzw. 4er Bündel sind in Tabelle 5.1 zusammen gestellt.

Beispiel 5.7 *Konvertierung in das Oktalsystem:*

$$110110011_2 = \underbrace{110}_{6}\ \underbrace{110}_{6}\ \underbrace{011}_{3}{}_2 = 663_8,$$

Beispiel 5.8 *Konvertierung in das Hexadezimalsystem:*

$$110110011_2 = \underbrace{0001}_{1}\ \underbrace{1011}_{\mathrm{B}}\ \underbrace{0011}_{3}{}_2 = 1\mathrm{B}3_{16}$$

Das Oktalsystem ist übersichtlich und leicht zu erlernen. Andererseits ist des Hexadezimalsystem besonders für Rechner geeignet, da ein Wert gerade einem halben Byte entspricht. Die ersten einfachen Rechner hatten eine Wortbreite von einem Byte, so dass sich die darstellbaren Zahlen mit zweistelligen Hexadezimalzahlen eingeben oder ausgeben lassen konnten (4 x 4 Hex-Tastatur, 7-Segmentanzeige).

Tabelle 5.1: 3er bzw. 4er Bündel

Dual Triaden	Oktal	Dual Tetraden	Hexadezimal
000	0	0000	0
001	1	0001	1
010	2	0010	2
011	3	0011	3
100	4	0100	4
101	5	0101	5
110	6	0110	6
111	7	0111	7
		1000	8
		1001	9
		1010	A
		1011	B
		1100	C
		1101	D
		1110	E
		1111	F

Tabelle 5.2: Zahlen in verschiedenen Stellenwertsystemen

Dezimal	Dual	Oktal	Hexadezimal
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10
17	10001	21	11
18	10010	22	12

5.5 Übungen

Übung 5.2 *Umwandlung in Oktal- und Hexadezimalsystem.*

Dualsystem	*Oktalsystem*	*Hexadezimalsystem*
11111111		
11110000		
10000000		

Übung 5.3 *Ergänzen Sie folgende Tabelle:*

Dezimalsystem	*Dualsystem*	*Oktalsystem*	*Hexadezimalsystem*
12			
	100010101		
81			
668		*1234*	
61666			*F0E2*

Übung 5.4 *Häufig benutzt man hexadezimale Konstanten, wenn man einzelne Bits in einem Byte setzen möchte. Dazu wird eine bitweise ODER Verknüpfung mit der sogenannten Maske durchgeführt. Welche Werte brauchen Sie, um in einem Byte*

- *die 4 höchsten Bits*
- *die 4 niedrigsten Bits*
- *die 4 mittleren Bits*
- *alle Bits an den Positionen 1, 3, ... (ganz rechts sei die Position 0)*
- *das Bit an der höchsten Stelle (most significant bit, MSB).*

zu setzen? Geben Sie jeweils die Maske im Dual- und Hexadezimalsystem an.

5.6 Rechnen im Dualsystem

5.6.1 Addition

Die Addition ist die wesentliche Grundrechenart, aus der sich die anderen Rechenarten ableiten lassen. Betrachten wir zunächst die Addition zweier einstelliger Dualzahlen:

a	b	$a+b$
0	0	0
0	1	1
1	0	1
1	1	10

Die ersten drei Fälle sind einfach, im dritten Fall kommt es zu einem Übertrag. Dies entspricht genau dem Überlauf, der bei der uns vertrauten Addition im Dezimalsystem auftritt, wenn eine Zwischensumme größer als 10 wird (*3 im Sinn*). Damit verläuft die Addition mehrstelliger Zahlen genau wie im Dezimalsystem. An Stelle der Addition der Stellen von Einer, Zehner, Hunderter, etc tritt die Addition der 2er-Potenzen 1, 2, 4, 8, 16, etc.

Beispiel 5.9 *Addition zweier Zahlen*

	Binär								*Dezimal*
	0	1	1	0	1	1	0	1	109
	1	0	0	0	1	1	0	1	141
Übertrag				1	1		1		010
	1	1	1	1	1	0	1	0	250

Bei der Addition einer Stelle zweier Werte (einer Spalte) gehen die Ziffern an diesen Stellen sowie ein eventuell vorhandener Übertrag der vorherigen Stelle ein. Ergebnis ist die Ziffer der Summe an dieser Stelle und der Übertrag für die Addition der nächsten Stelle. Diese Zusammenhänge kann man als logische Verknüpfungen darstellen. Seien a und b die beiden Ziffern, ü der Übertrag der vorherigen Stelle, s die Summenstelle und ü2 der Übertrag für die nächste Stelle. Dann sind s und ü2 zwei logische Funktionen von a, b und ü. Zur besseren Übersicht sind in die beiden Funktionen in der folgenden Wahrheitstabelle zusammen eingetragen:

ü	a	b	s	ü2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Die entsprechende Schaltung bezeichnet man als Volladdierer (VA). Aus den drei Eingangswerten a, b und ü berechnet ein Volladdierer die Summe s und den neuen Übertrag ü2. Dies entspricht einer Stelle (Spalte) bei der Addition zweier Binärzahlen. Durch Kombination von entsprechend vielen Volladdierern lässt sich eine Schaltung zur Addition von n-stelligen Binärzahlen aufbauen.

Bild 5.1 zeigt einen Aufbau zur Addition 4-stelliger Binärzahlen. In der niedrigwertigsten Stelle braucht noch kein Übertrag berücksichtigt zu werden. In diesem Fall genügt eine einfachere Schaltung mit nur zwei Eingängen für die beiden Summanden. Dies ist ein so genannter Halbaddierer (HA). Für den höchsten Übertrag

ist im Ergebnis kein Platz mehr vorhanden. Wenn durch entsprechend große Summanden das Ergebnis mehr als 4 Stellen benötigt, kommt es zu einem Überlauf.

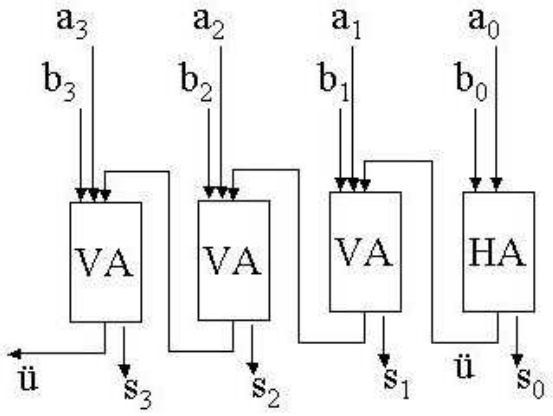


Abbildung 5.1: Addierer für 4-stellige Binärzahlen

Übung 5.5 *Geben Sie die disjunktive Normalform für den Übertrag eines Volladdierers an. Wie lässt sich der Ausdruck vereinfachen?*

Übung 5.6 *Die Summe kann durch EXOR-Operationen effizient berechnet werden. Wie sieht ein entsprechender Ausdruck aus?*

5.6.2 Subtraktion

In Hinblick auf eine möglichst einfache Struktur des Rechenwerks ist es vorteilhaft, wenn für die Subtraktion keine eigene Einheit benötigt wird, sondern die Subtraktion auf die Addition von Zahlen mit Vorzeichen zurück geführt werden kann. Dieses Ziel wird durch Einführung der 2er-Komplement Darstellung erreicht. Dabei wird eine negative Zahl wie folgt gebildet:

1. Absolutbetrag der Zahl im n-Bit Zahlensystem bilden
2. Alle Stelle invertieren (inklusive führender Nullen)
3. 1 addieren

Das damit erhaltene Bitmuster repräsentiert die negative Zahl.

Beispiel 5.10 *Zahlensystem mit 8 Bit:*

Zahl -25_{10}*:*	
Absolutbetrag	00011001_2
Alle Stelle invertieren:	11100110_2
1 addieren:	11100111_2

Die Addition der positiven Zahl und der negativen Zahl muss dann 0 ergeben:

	0	0	0	1	1	0	0	1	25_{10}
	1	1	1	0	0	1	1	1	-25_{10}
Übertrag	1	1	1	1	1	1	1		
	0	0	0	0	0	0	0	0	0

Übung 5.7 *Wandeln Sie die Zahlen 24 und 11 in ihre Binärdarstellung um und berechnen dann die Differenz 24-11.*

									24_{10}
									-11_{10}
Übertrag									

Insgesamt lassen sich die Eigenschaften von vorzeichenbehafteten ganzen Zahlen in einem n-Bit System wie folgt zusammenstellen:

Zahl	Binärmuster	Beispiel n=8
0	00 ... 0000	0000 0000
+1	00 ... 0001	0000 0001
−1	11 ... 1111	1111 1111
größte (positive) Zahl	01 ... 1111	0111 1111 = 127_{10}
kleinste (negative) Zahl	10 ... 0000	1000 0000 = -128_{10}

Das Vorzeichen einer Zahl kann man an der ersten Stelle (most significant Bit MSB) erkennen:

0 positiv

1 negativ

Vergrößert man die Anzahl der Stellen n, so muss man bei negativen Zahlen entsprechend oft 1 einfügen. Es stellt sich noch die Frage, was passiert bei Bereichsüberschreitungen? Betrachten wir ein Beispiel in der Programmiersprache C:

```
long i = 2147483647;  // groeßte 32-Bit Zahl
printf("i: %d\ni+1: %d\n", i, i + 1);
```

Die Ausgabe lautet:

```
i: 2147483647
i+1: -2147483648
```

Indem wir eine 1 zu der größten positiven Zahl addiert haben, sind wir zu der kleinsten negativen Zahl gelangt. Wie in Bild 5.2 dargestellt, kann man sich die Zahlen im Kreis angeordnet vorstellen.

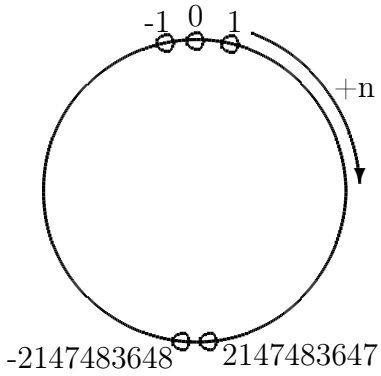


Abbildung 5.2: Anordnung von Integerzahlen

5.6.3 Multiplikation und Division

Prinzipiell können ganzzahlige Multiplikation und Division auf die Grundrechenart Addition zurück geführt werden. Eine Multiplikation wird dann durch wiederholtes Addieren realisiert. Das Vorgehen ist analog zum Rechnen im Dezimalsystem.

Beispiel 5.11 *Multiplikation von* 1011_2 *mit* 1101_2

				1	0	1	1	$*$	1	1	0	1
				1	0	1	1			$*$	1	
			0	0	0	0				$*$	0	
		1	0	1	1					$*$	1	
	1	0	1	1						$*$	1	
1	0	0	0	1	1	1	1					

Ergebnis: $1101_2 * 1011_2 = 10001111_2$

Ähnlich kann die Division durch wiederholte Subtraktion (= Addition des 2-er Komplements) realisiert werden. Damit könnten im Prinzip nur mit einem Addierwerk und einem Inverter zur Berechnung des 2-er Komplements alle Grundrechenarten für ganze Zahlen ausgeführt werden. In der tatsächlichen Realisierung werden allerdings komplexere Schaltungen eingesetzt, um eine höher Rechengeschwindigkeit zu erreichen.

5.7 Gleitkomma-Zahlen

5.7.1 Gleitkomma- Darstellung

Bisher hatten wir Zahlen in der Integerdarstellung betrachtet. Diese Darstellung von ganzen Zahlen und das Rechnen damit ist exakt solange man

- im darstellbaren Zahlenbereich bleibt
- keine Nachkommstellen betrachtet (z. B. nach Division)

In vielen praktischen Anwendungen benötigt man aber eine flexiblere Repräsentation von Zahlen. Oft ist das Rechnen mit Nachkommastellen notwendig oder zumindest natürlich. Viele Angaben enthalten Nachkommastellen (Zinssatz 3,5%, 4,7 Liter auf 100 Km). Die erste Erweiterung ist die Einführung von Nachkommastellen. Bei der Festkommadarstellung gibt man die Anzahl der Vor- und Nachkommastellen fest vor. Als Nachteil bleibt dabei die eingeschränkte Dynamik des Zahlenbereichs. Daher wird diese Darstellung nur in wenigen Spezialanwendungen verwendet.

Auch im Alltag und noch mehr in der Technik haben wir das Problem der unterschiedlichen Bereiche. Bei Längen beispielsweise kann je nach Anwendung eine Angabe in mm (Schrauben im Baumarkt) oder in km (Urlaubsreise) sinnvoll sein. Der Trick dabei ist, dass die Länge mit einer Anzahl von signifikanten Stellen und der Größenordnung angegeben wird: 3,5mm oder 650km. Vollständige Genauigkeit 650.245.789mm ist weder sinnvoll noch notwendig. Allgemein schreibt man eine Größe z als Produkt der Mantisse M und einer ganzzahligen Potenz p von 10:

$$z = M \cdot 10^p$$

etwa 3,5 10^{-3} m. Für Maßangaben gibt es Namen bzw. Vorsilben für die entsprechenden Potenzen von 10 (Kilo, Giga, Mega, Nano, etc). Zum Rechnen muss man Zahlen auf eine einheitliche Darstellung normieren:

$$3{,}5\text{mm} + 650\text{km} = 0{,}003.5\text{m} + 650.000\text{m} = 650.000{,}003.5\text{m}$$

Die Beispiele zeigen, wie man durch Verschieben des Kommas in der Mantisse den Exponenten verändert. Es gibt für eine gegebene Zahl (unendlich) viele gleichwertige Darstellungen:

$$123 = 12,3 \cdot 10 = 1,23 \cdot 10^2 = 1230 \cdot 10^{-1} = \ldots$$

Eine einheitliche Darstellung erreicht man mit der Vereinbarung, dass die Mantisse nur genau eine Vorkommastelle hat. Damit hat man eine eindeutige Abbildung zwischen der Zahl z und ihrer Darstellung durch Mantisse m und Exponent p:

$$z \Leftrightarrow (m, p)$$

Die Position des Kommas wird je nach Bedarf verschoben, man nennt daher dieses Format Gleitkommadarstellung (floating point) oder auch halblogarithmische Darstellung. Für die Verwendung in Computern geht man zum Dualsystem über, so dass p dann der Exponent zur Basis 2 ist. In der normalisierten Darstellung

wird das Komma so gesetzt, dass nur eine Vorkommastelle bleibt. Die Mantisse dann hat im Dualsystem immer die Form

$$m = 1, \ldots$$

Da die führende 1 bei jeder Zahl steht, braucht man sie in der Realisierung nicht abzuspeichern. In der Praxis sind für m und p nur endlich viele Bits verfügbar. Die Größe von m bestimmt die Genauigkeit der Zahlendarstellung und die Größe von p legt den insgesamt abgedeckten Zahlenbereich fest. Aufgrund der endlichen Größe von m und p kann es zu folgenden Fehlern in der Repräsentation kommen:

- Die Zahl ist zu groß oder zu klein ($z \geq 2^{pmax+1},\ z \leq -2^{pmax+1}$).
- Die Zahl ist betragsmäßig zu klein ($|z| < 2^{-pmax}$ bei symmetrischem Zahlenbereich des Exponenten).
- Die Mantisse ist nicht groß genug, um die erforderliche Anzahl von Stellen zu repräsentieren (Rundungsfehler).

Beispiel 5.12 *Geleitkommadarstellung*
Betrachten wir folgende Zahlendarstellung im Dezimalsystem: $\pm x.xxx \cdot 10^{\pm ee}$.
Geben Sie dazu folgende Wert an:

- *Kleinste Zahl*
- *Größte Zahl*
- *Betragsmäßig kleinste Zahl* $\neq 0$
- *Abstand zwischen den beiden größten Zahlen*
- *Abstand zwischen den beiden betragsmäßig kleinsten Zahlen*

Insbesondere die Rundungsfehler bedingen, dass im allgemeinen eine reelle Zahl nicht genau dargestellt werden kann. Berechnet man etwa 1/3 so ist das Resultat 0,33333... prinzipiell nicht exakt darstellbar. Für die Repräsentierung einer Gleitkommazahl benötigt man im Detail:

- Mantisse
- Vorzeichen der Mantisse
- Exponent
- Vorzeichen des Exponents

Weiterhin muss festgelegt werden, wie die insgesamt verfügbaren Bits auf Mantisse und Exponent aufgeteilt werden. Lange Zeit waren die Details der Implementierung von Gleitkommazahlen herstellerabhängig. Da dies zu Problemen beim Datenaustausch und auch bei der Kompatibilität von Programmen führen kann, wurde vor einigen Jahren ein Standard von Normierungsgremien des IEEE (Institute for Electrical and Electronics Engineers, www.ieee.org) verabschiedet. Dieser Standard IEEE 754 definiert 3 Formate:

short real	32 Bit	einfache Genauigkeit
long real	64 Bit	doppelte Genauigkeit
temporary real	80 Bit	erweiterte Genauigkeit

Als Beispiel betrachten wir das Format short real näher. Die 32 Bit sind wie folgt aufgeteilt:

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

mit

Vz	Vorzeichen der Mantisse (0 positiv, 1 negativ)
Charakteristik	Exponent + 127
Mantisse	Nachkommastellen

Im Gegensatz zu der bei Integer üblichen Darstellung mit dem 2er-Komplement wird die Mantisse mit Betrag und getrenntem Vorzeichen abgelegt (Vorzeichen-Betrag-Darstellung). In der Charakteristik wird der um 127 verschobene Exponent (biased exponent) eingetragen.

Beispiel 5.13 *Konvertierung in den Standard IEEE 754*
Die Zahl $-37,125_{10}$ *soll in das Format short real gebracht werden.*

1. *Konvertierung in Binärdarstellung:* $100101,001$ $(32 + 4 + 1 + 1/8)$
2. *Normalisierung:* $1,00101001 \cdot 2^5$
3. *Mantisse:* 00101001
4. *Charakteristik:* $5 + 127 = 132_{10} = 10000100_2$
5. *Vorzeichen:* 1 *für negative Zahl*

Damit erhält man die Darstellung

Bit 31		*Bit 0*
1	*1000010 0*	*0010100 10000000 00000000*
1 Bit	*8 Bit*	*23 Bit*
Vz	*Charakteristik*	*Mantisse*

beziehungsweise

$$1100.0010.0001.0100.1000.0000.0000.0000_2 = C2148000_{16}$$

Die Werte 0 und 255 für die Charakteristik sind reserviert, um den Wert Null sowie einige Spezialfälle darstellen zu können. Die Null ist ein Sonderfall, da für sie keine normalisierte Darstellung mit einer Eins vor dem Komma möglich ist. Daher wurde vereinbart, dass die Null durch ein Löschen aller Bit in Charakteristik und Mantisse dargestellt wird. Im Detail gilt:

	Vz	Charakteristik	Mantisse
Nicht normalisiert	$\pm$	0	$\neq 0$
Null	$\pm$	0	0
Unendlich (Inf)	$\pm$	255	0
Keine Zahl (NaN)	$\pm$	255	$\neq 0$

Mit den beiden Werten Inf und NaN können Fehlerfälle abgefangen werden. Bei Bereichsüberschreitung wird das Result auf Inf gesetzt, während NaN durch „unerlaubte“ Operationen wie Division von 0 durch 0 oder Wurzel aus einer negativen Zahl entsteht. Damit besteht einerseits die Möglichkeit, solche Fälle zu erkennen. So steht beispielsweise in C die Funktion `_isnan` zur Verfügung, um auf NaN zu testen. Andererseits kann man auch mit den Werten weiter rechnen. Der Standard spezifiziert das Ergebnis von Operationen wie Inf + 10 = Inf. In manchen Algorithmen kann man damit alle Abfragen auf Sonderfälle vermeiden, die sonst den linearen Programmablauf stören würden. Eine ausführliche Darstellung der Thematik enthält der Artikel von Goldberg [Gol91].

Beispiel 5.14 *`Inf` und `NaN`*
Der C-Code

```
printf( "log( 0.) = %15g\n", log( 0.));
printf( "log(-1.) = %15g\n", log(-1.));
```

liefert das Resultat

```
log( 0.) =         -1.#INF
log(-1.) =         -1.#IND
```

Bei den beiden größeren Typen long real und temporary real wird sowohl die Genauigkeit erhöht als auch der Wertebereich erweitert. Rechnen mit Gleitkommazahlen bedeutet einen wesentlich größeren Aufwand als das Rechnen mit Integerzahlen. Speziell bei der Addition müssen zunächst die Mantissen und Exponenten verschoben werden, bevor die Mantissen addiert werden können. Das Ergebnis muss anschließend gegebenenfalls wieder in die Normalform gebracht werden.

Schnelles Rechen in Gleitkommadarstellung erfordert entsprechenden zusätzlichen Schaltungsaufwand. Früher wurden dafür spezielle Baustein als Co-Prozessoren eingesetzt. Heute ist eine entsprechende Einheit bei den leistungsfähigen Prozessoren bereits integriert. Die Angabe der möglichen Gleitkommaoperationen pro Sekunde (FLOPS: floating point operations per second) ist eine wichtige Kenngröße für die Leistungsfähigkeit eines Computers.

Übung 5.8 *Gleitkommadarstellung*
Betrachten Sie folgendes einfaches Format für Gleitkommazahlen:

- *Bit 15: Vorzeichen des Exponenten (0 für positiv)*
- *Bit 14: Vorzeichen der Mantisse (0 für positiv)*
- *Bit 6-13 Betrag der normierten Mantisse*
- *Bit 0-5 Betrag des Exponenten*

Welchen Wert haben die folgenden Bitmuster:

0	*0*	*0110 0000*	*00 0011*
1	*0*	*1010 0000*	*00 0100*
0	*0*	*0000 0000*	*00 0000*

Wenn alle Bits 0 sind, erhält man trotzdem nicht den Wert 0. Gibt es ein anderes Bitmuster für die Zahl 0? Falls nicht, wie würden Sie das Problem lösen?

Übung 5.9 *Wertebereich im Standard IEEE 754*
Welches ist jeweils die

- *kleinste*
- *größte*
- *betragsmäßig kleinste*

darstellbare Zahl im short real Format?

Übung 5.10 *Konvertierung in den Standard IEEE 754*
Wie wird die Zahl $14,625_{10}$ *als Gleitkommazahl im Format real short dargestellt? Geben Sie das Resultat in Binär- und Hexadezimaldarstellung an.*

Übung 5.11 *Addition und Multiplikation*
Berechnen Sie für die beiden Zahlen $7,5 \cdot 10^4$ *und* $6,34 \cdot 10^2$ *Summe und Produkt. Geben Sie das Ergebnis jeweils in normierter Form mit einer Vorkommastelle an. Welche einzelnen Rechenschritte sind erforderlich?*

Übung 5.12 *Darstellung der Zahl 0*
In IEEE 754 gibt es zwei Bitmuster für die Zahl 0, einmal mit positiven und einmal mit negativem Vorzeichen. Diese Werte kann man als +0 und -0 interpretieren. Wo kann man diese Unterscheidung sinnvoll einsetzen? Welche Probleme können sich aus der doppelten Darstellung ergeben?

5.7.2 Verwendung von Gleitkommazahlen

Der große Vorzug von Gleitkommazahlen ist der große Wertebereich mit gleichbleibender relativer Genauigkeit. Andererseits ist die Abdeckung – im Gegensatz zu den Integerzahlen – nicht vollständig. Zwischen benachbarten Gleitkommazahlen ist eine Lücke und die Breite der Lücke hängt von der Größe der Zahlen ab. Dies kann zu Effekten führen, die der Mathematik wiedersprechen. Das folgende Fragment C-Code dient zur Veranschaulichung dieses Verhaltens. In dem Programm wird zu einer Zahl der Wert 1 addiert, wobei die Zahl schrittweise um den Faktor 10 erhöht wird.

```
double test = 1.;
double testp1;

do{
    test *= 10.;
    testp1 = test + 1.;
    printf( "%10g %10g %5g \n", test, testp1, testp1-test );
} while( testp1 > test );
```

Die Ausführung liefert die Ausgabe:

```
    10         11     1
   100        101     1
  1000       1001     1
 10000      10001     1
100000     100001     1
1e+006     1e+006     1
1e+007     1e+007     1
1e+008     1e+008     1
1e+009     1e+009     1
1e+010     1e+010     1
1e+011     1e+011     1
1e+012     1e+012     1
1e+013     1e+013     1
1e+014     1e+014     1
1e+015     1e+015     1
1e+016     1e+016     0
```

Zunächst zeigt das Programm das erwartete Verhalten und berechnet die Differenz zu 1. Aber wenn der Wert 10^{16} erreicht ist, führt die Addition nicht mehr zu einer anderen Zahl und die Differenz zwischen 10^{16} und $10^{16} + 1$ liefert den Wert 0. Dieser Einfluss der Rundungsfehler ist bei der Programmentwicklung zu berücksichtigen.

Tabelle 5.3: Vergleich zwischen Integer- und Gleitkommazahlen

	Integer	Gleitkomma
Nachkommastellen	Nein	Ja
Genauigkeit	Ergebnisse sind exakt	Rundungsfehler
Bereich	eingeschränkt	groß
Überlauf	wird nicht gemeldet	Inf
Rechenaufwand	niedrig	groß
Speicherbedarf	8-32 Bit	32-80 Bit
	Bit-Operatoren, modulo	

Beispiel 5.15 *Rundungsfehler*
Die folgende Anweisung in C

```
printf( "5. - sqrt(5)*sqrt(5) = %15g\n", 5. - sqrt(5.)*sqrt(5.));
```

ergibt

```
5. - sqrt(5)*sqrt(5) =   -8.88178e-016
```

5.7.3 Vergleich der Zahlenformate

Abschließend sind die wesentlichen Unterschiede in den Zahlendarstellungen in Tabelle 5.3 zusammen gestellt. Abhängig von der Anwendung sind die einzelnen Kriterien unterschiedlich zu gewichten. Beispielsweise spielt bei einer low-cost-Anwendung der Preis eine entscheidende Rolle, so dass auf eine eigene Einheit für Gleitkommaoperationen verzichtet wird. Dann ist es oft notwendig Berechnungen, für die Gleitkommazahlen besser geeignet wären, aus Performanzgründen trotzdem mit Integerzahlen durchzuführen.

5.8 Übungen

Übung 5.13 *Rechnen*
Welche Ausgabe liefern die folgenden Anweisungen? Inwieweit sind die Ergebnisse mathematisch korrekt?

```
System.out.println( Double.MAX_VALUE );
System.out.println( Double.MAX_VALUE + 1);
System.out.println( 1./0. );
System.out.println( 0./0. );
System.out.println( 10000000 * 10000000 );
System.out.println( 10000000.0 * 10000000  );
System.out.println( 1e100 * 1e100  );
```

```
System.out.println( Math.log( -2. ) );
System.out.println( Math.sqrt( -2. ) );
System.out.println( Math.exp( -1000. ) );
System.out.println( Math.exp( 1000. ) );
System.out.println( 1./Math.exp( 1000. ) );
System.out.println( Math.exp( -1000. ) * Math.exp( 1000. ) );
```

Übung 5.14 *Reihenfolge der Auswertung eines Ausdrucks*
Sei $x = 10^{30}$, $y = -10^{30}$ *und* $z = 1$. *Welches Resultat ergibt sich bei dem Rechnen in Gleitkomma-Arithmetik für die beiden Ausdrücke*

- $(x + y) + z$
- $x + (y + z)$

Kapitel 6

Zeichendarstellung

6.1 ASCII

Neben dem Rechnen mit Zahlen ist die Verarbeitung von Texten aller Art eine der zentralen Aufgaben von Computern. Dazu ist es erforderlich, die verwendeten Zeichen – Buchstaben, Ziffern, Satzzeichen, etc. – durch Bitmuster im Speicher zu repräsentieren. Für insgesamt N Zeichen benötigt man $\log_2 N$ Bit Wortbreite. Im Prinzip ist die Zuordnung willkürlich. Wesentlich ist nur, dass für jedes Zeichen eindeutig ein Bitmuster vereinbart wird. Allerdings kann durch geschickte Festlegung der Zuordnung das Arbeiten mit Zeichen vereinfacht werden.

Für die Zuordnung zwischen Zeichen und Zahlenwerten existieren verschiedene Systeme. Besonders bei IBM Großrechnern ist das System EBCDIC (*Extended Binary Coded Decimal Interchange Code*) gebräuchlich. Neuer und mittlerweile sehr viel verbreiteter ist der ASCII (*American Standard Code for Information Interchange*) Zeichensatz. Ursprünglich waren im ASCII Satz nur 7 Bit pro Zeichen vorgesehen. Damit können dann 128 verschiedene Zeichen dargestellt werden. Dieser Zeichensatz enthält Buchstaben, Ziffern, Sonderzeichen und spezielle Steuerzeichen, die ursprünglich für Datenkommunikation eingeführt wurden.

Für den allgemeinen Einsatz zur Textverarbeitung wurde es erforderlich, auch nationale Sonderzeichen wie etwa Umlaute für das Deutsche darstellen zu können. Daher wurde ein erweiterter 8-Bit ASCII Zeichensatz mit nationalen Sonderzeichen und einfachen Graphikelementen eingeführt. Die Erweiterung ist allerdings nicht universell, sondern es gibt eine ganze Reihe von nationalen Varianten. Daher sollte man bei Programmen diese Zeichen nur mit Vorsicht einsetzen. Tabelle 6.1 zeigt im Überblick die 7-Bit ASCII Zeichen.

Man erkennt eine Einteilung in vier Blöcke mit je 32 Zeichen. Die ersten 32 Zeichen sind die schon erwähnten Steuerzeichen. Einige der Abkürzungen bedeuten:

BEL Bell (Piepszeichen)

HT Horizontaler Tab (Tabulator)

Tabelle 6.1: 7-Bit ASCII Zeichen

		0	1	2	3	4	5	6	7
Dez.	Binär	000...	001...	010...	011...	100...	101...	110...	111...
0	...0000	NUL	DLE	SP	0	@	P	‘	p
1	...0001	SOH	DC1	!	1	A	Q	a	q
2	...0010	STX	DC2	"	2	B	R	b	r
3	...0011	ETX	DC3	#	3	C	S	c	s
4	...0100	EOT	DC4	$	4	D	T	d	t
5	...0101	ENQ	NAK	%	5	E	U	e	u
6	...0110	ACK	SYN	&	6	F	V	f	v
7	...0111	BEL	ETB	’	7	G	W	g	w
8	...1000	BS	CAN	(	8	H	X	h	x
9	...1001	HT	EM	)	9	I	Y	i	y
10	...1010	LF	SUB	*	:	J	Z	j	z
11	...1011	VT	ESC	+	;	K	[	k	{
12	...1100	FF	FS	,	<	L	\	l	|
13	...1101	CR	GS	-	=	M	]	m	}
14	...1110	SO	RS	.	>	N	^	n	~
15	...1111	SI	US	/	?	O	_	o	DEL

LF Line Feed (Zeilenvorschub)

CR Carriage return (Zeilenanfang)

Die nächsten 32 Zeichen sind die Ziffern und Sonderzeichen. SP steht für das Leerzeichen (engl. space). Darauf folgen die Groß- und Kleinbuchstaben in alphabetischer Reihenfolge zusammen mit einigen weiteren Sonderzeichen. Die Buchstaben sind so angeordnet, dass der Abstand zwischen zusammen gehörenden Groß- und Kleinbuchstaben immer 32 beträgt. Damit unterscheiden sich die jeweiligen Bitmuster nur an einer Stelle und die Umwandlung zwischen beiden Formen ist einfach auszuführen.

An dieser Stelle sei nochmals der Unterschied zwischen Ziffern und den Zahlen betont. Die Ziffer 2 wird durch das Bitmuster 110010_2 im Speicher repräsentiert. Interpretiert man dieses Bitmuster als Integerzahl, so ergibt sich der Wert 50_{10}. Viele Programmiersprachen erlauben es, eine solche Speicherzelle sowohl als Zeichen als auch als Zahl zu interpretieren. Damit kann man beispielsweise in C eine Konstruktion in der Art

```
i = ziffer - '0';
```

verwenden, um aus einer Ziffer den zugehörigen Integerwert zu berechnen. Die Subtraktion bezieht sich nicht auf den Zahlenwert, sondern auf die Position in der Zeichentabelle. Durch die Anordnung der Ziffern in der ASCII entspricht der

Abstand einer Ziffer zu der Ziffer 0 gerade ihrem Zahlenwert. Allerdings führt eine solche Konstruktion zu einer Abhängigkeit vom System. Es ist nicht garantiert, dass jede Zeichentabelle auf einem beliebigen Computersystem so aufgebaut ist, dass die Differenz den richtigen Wert ergibt.

Übung 6.1 *Die Standardbibliothek von C enthält eine Funktion* `toupper`*, die aus einem Buchstaben den zugehörigen Großbuchstaben erzeugt. Wie würden Sie diese Funktion realisieren?*

6.2 Unicode

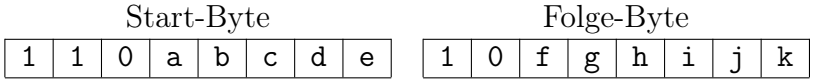
In Hinblick auf eine weltweite Nutzung von Informationssystemen ist die Beschränkung durch das 8-Bit ASCII System sehr hinderlich. Es liegt nahe, durch mehr Bits eine möglichst umfassende Darstellung der in den verschiedenen Sprachen und Kulturen verwendeten Zeichen zu erreichen. Dieses Ziel wird im Unicode System durch die Verwendung von 16 Bit (entspricht 65536 Zeichen) erreicht. Auf der Unicode Seite (`www.unicode.org`) finden man folgenden Anspruch:

> Unicode provides a unique number for every character,
> no matter what the platform,
> no matter what the program,
> no matter what the language.

Unicode enthält als Untermenge den ASCII-Zeichensatz inklusive Steuerzeichen. Darüber hinaus sind aber auch Buchstaben aus vielen Sprachen, Sonderzeichen, mathematische Symbole, etc. enthalten. Das Unicode System stellt außerdem Informationen wie etwa die Schreibrichtung für die einzelnen Zeichen zur Verfügung. Einzelne Zeichen repräsentieren Grundelemente, die dann weiter zu Gesamtzeichen kombiniert werden können, so dass insgesamt sehr viel mehr Zeichen dargestellt werden können. Es ist zu erwarten, dass sich Unicode in den nächsten Jahren weiter verbreiten wird. Die Sprachen JAVA und HTML unterstützen bereits heute Unicode.

6.2.1 Kodierung

Für eine längere Übergangszeit ist mit dem Nebeneinander von ASCII- und Unicode-Darstellung zu rechnen. Daher ist es sinnvoll, Kodierungen zu verwenden, die beide Möglichkeiten abdecken. Eine gebräuchliche Kodierung ist UTF-8 (*8-bit Unicode Transformation Format*). In UTF-8 wird eine variable Anzahl von Bytes zur Kodierung eines Zeichens verwendet. Als Untermenge sind die 7-Bit ASCII-Zeichen als Einbyte-Folgen enthalten. Solche ASCII-Zeichen werden an einer führenden Null erkannt. Ist das erste Bit demgegenüber eine Eins, so handelt es sich um das Startbyte einer mindestens zwei Byte langen Folge. Im Detail gilt folgende Vorschrift:



mit den Bedeutungen (von links)

`1`	Kennung für Mehrbyte-Zeichen
`10`	Anzahl der Folgebytes ist 1
`abcde`	Erster Teil des Unicode-Zeichens
`10`	Kennung für Folgebyte
`fghijk`	Zweiter Teil des Unicode-Zeichens

Unicodezeichen (mit Nullen aufgefüllt): `00000abc defghijk`

Abbildung 6.1: Beispiel für 2-Byte Folge in UTF-8

- Erstes Bit `0`: Einbyte-Zeichen, Wert entspricht 7-Bit ASCII
- Ersten beiden Bits `11`: Startbyte, die Anzahl der Folgebytes (mindestens eins) ist durch die Anzahl der Einsen bis zur ersten Null angegeben
- Ersten beiden Bits `10`: Eine Folgebyte

Die verbleibenden Bits definieren das Unicode-Zeichen. Falls erforderlich werden linksbündig Nullen aufgefüllt. Bild **??** zeigt ein Beispiel für eine 2-Byte Folge. Eine Datei mit 7-Bit ASCII- Zeichen ist damit automatisch auch im Format UTF-8. Da die Anzahl der Folgebytes im Startbyte angegeben ist, ist die Markierung der Folgebytes mit `01` redundant. Allerdings hilft diese Markierung bei Problemen. Geht bei der Analyse einer Byte-Folge die Synchronisation auf die Startbytes verloren, so kann man trotzdem bei jedem Byte korrekt den Typ erkennen. Folgebytes werden dann einfach ignoriert, bis wieder ein Startbyte oder ein Einbyte-Zeichen auftritt.

Kapitel 7

Informationstheorie

7.1 Einleitung

Zu Beginn der Vorlesung hatten wir Informatik als „Wissenschaft von der systematischen Verarbeitung von Informationen“ definiert. Offen blieb dabei zunächst die Definition von Information. Im Laufe der Vorlesung haben wir zahlreiche Beispiels gesehen, wie durch Programme verschiedene Arten von Zahlen, Zeichen, logische Werte oder aus diesen elementaren Typen zusammen gesetzte Strukturen verarbeitet werden.

In einfachen Fällen haben wir dabei auch schon aus den Eingangsdaten gezielte „Informationen“ gesucht. Ein Beispiel war die Suche nach Namen, Vornamen, etc. in einer Textzeile. Für die weitere Verarbeitung wurden dabei „überflüssige“ Daten wie zusätzliche Leerzeichen entfernt.

7.2 Was ist Information?

Der Informationsgehalt eines Textes, eines Bildes, einer Web-Seite, einer Audiodatei, etc. lässt sich nicht allgemein festlegen. Der Informationsgehalt ist keine Eigenschaft an sich, sondern nur aus Sicht des Lesers, Betrachters oder Zuhörers (allgemein: Empfänger) zu beurteilen. Wesentlich ist dabei:

- die verwendete Sprache bzw. Symbolik der Nachricht
- vorhandenes Vorwissen

So kann eine für den Laien unverständliche Formel für eine Chemikerin eine wichtige Information darstellen. Wenn die Symbole der Nachricht verständlich sind, hängt der Informationsgehalt davon ab, wie viel Neues sie enthält. Als klassisches Beispiel dient der Vergleich der beiden folgenden Schlagzeilen:

„Hund beisst Briefträger“
„Briefträger beisst Hund“

Die erste Nachricht bietet wenig neues oder überraschendes. Kein Redakteur würde sie als Schlagzeile verwenden. Demgegenüber berichtet die zweite Nachricht von einem ungewöhnlichen - unerwarteten - Ereignis. Es handelt sich um eine Neuigkeit, die die Leser interessieren wird. Ausgehend von diesem Gesichtspunkt - Nachrichten über unwahrscheinliche Ereignisse haben einen hohen Informationswert - entwickelte sich die statistische Informationstheorie. Sie liefert Aussagen über den Informationsgehalt von Nachrichten und damit über Möglichkeiten der effizienten Speicherung und Übertragung von Nachrichten. Im folgenden wird zunächst anhand eines Beispiels die Frage der optimalen Kodierung untersucht. Darauf aufbauend folgt die Darstellung der Grundbegriffe der statistischen Informationstheorie.

7.3 Kodierung

Zur Verarbeitung von Werten (Zahlen, Buchstaben, Ziffern) im Rechner müssen diese Werte in eine Binärdarstellung gebracht werden. Dazu wird eine eindeutige Zuordnung der Werte zu Bitmuster benötigt. Beispiele dazu sind:

- ASCII Tabelle
- Unicode Tabelle
- Zweier-Komplement Darstellung
- IEEE Format für Gleitkommazahlen

Im folgenden wird einheitlich der Begriff „Zeichen“ für die verschiedenen Arten von Werten benutzt. In diesem Sinn ist etwa auch eine Gleitkommazahl oder ein Boolescher Wert ein Zeichen. Entscheidend ist die Zuordnung zwischen den Zeichen und den Binärmustern. Die Gesamtheit der Zeichen nennt man Zeichenvorrat. So umfasst der 7-Bit ASCII-Kode einen Vorrat von 128 Zeichen. Bisher hatten wir stets eine gleichförmige Kodierung verwendet: für jedes Zeichen werden gleich viele Bits verwendet. Jedes Zeichen benötigt damit gleich viel Platz im Speicher oder Zeit zur Übertragung. Mit N Bits lassen sich 2^N verschiedene Zeichen kodieren. Umgekehrt benötigt man für M Zeichen $\log_2 M$ Bits. Betrachten wir zur Vereinfachung die Kodierung von nur 4 Zeichen. Für 4 Zeichen werden 4 verschiedene Bitmuster benötigt. Dies kann mit 2 Bits realisiert werden. Eine mögliche Zuordnung ist:

Zeichen	Kodierung	Bits
A	00	2
B	01	2
C	10	2
D	11	2

Beispiel 7.1 *Die Folge ABCDDCBA wird kodiert als* 0001101111100100

Die Zuordnung zwischen Zeichen und Bitmuster ist frei wählbar. Wesentlich ist nur, dass diese Zuordnung fest ist und beispielsweise bei der Übertragung über ein Netzwerk von Sendern und Empfängern gleichermaßen verwendet wird.

Im Allgemeinen treten nicht alle Zeichen mit gleicher Häufigkeit auf. In normalen Texten ist etwa ein E viel häufiger als ein X. Die Verteilung der Zeichen hängt dabei von der Art des Textes ab. So unterscheidet sich die Häufigkeit der einzelnen Buchstaben in verschiedenen Sprachen. Ein Beispiel für eine spezielle Verteilung sind Programmiersprachen. Hier findet man sehr ausgeprägte „Spitzen“ für die Zeichen zur Darstellung der Syntax (z. B *;* in C oder Java). Die unterschiedliche Häufigkeit kann zur effizienteren Kodierung ausgenutzt werden. Die Grundidee ist, die Anzahl der Bits pro Zeichen an die Häufigkeit anzupassen. Häufige Zeichen werden mit wenigen Bits dargestellt, während für seltene Zeichen mehr Bits verwendet werden. Insgesamt führt dies zu einer geringeren mittleren Anzahl von Bits pro Zeichen. Betrachten wir für die weitere Diskussion folgendes Beispiel mit 4 Zeichen mit deutlich unterschiedlicher Häufigkeit:

Zeichen	Häufigkeit
A	1/2
B	1/3
C	1/18
D	1/9

In diesem Fall ist im Mittel jedes zweite Zeichen ein A. Eine kompakte Kodierung sollte für dieses Zeichen einen möglichst kurzen Kode verwenden. Naheliegend ist eine Kodierung in der Form:

Zeichen	Häufigkeit	Kodierung	Bits
A	1/2	1	1
B	1/3	0	1
C	1/18	01	2
D	1/9	10	2

Aus der Folge $ABCDDCBA$ wird dann 100110100101 Allerdings hat diese Kodierung einen entscheidenden Nachteil: aus ihr lässt sich nicht mehr eindeutig die Zeichenfolge rekonstruieren. So könnte die Folge auch mit $DC\ldots$ beginnen. Ein generelles Problem bei Zeichen unterschiedlicher Länge ist die Aufteilung einer längeren Zeichenfolge in die einzelnen Zeichen. Der Einbau von zusätzlichen Trennzeichen zwischen den Kodes würde den Gewinn wieder zunichte machen. Gesucht ist daher eine Kodierung, bei der die einzelnen Kodes eindeutig erkennbar sind. Es gibt verschiedene Algorithmen, um einen solchen Kode zu erzeugen. In unserem Beispiel lässt sich dies leicht wie folgt erreichen: zunächst wird für das

Zeichen A der Kode 1 festgelegt. Alle anderen Kodes beginnen dann mit 0. Im zweiten Schritt wird für das Zeichen B zur weiteren Unterscheidung der Kode 01 gewählt. Die beiden noch fehlenden Kodes beginnen dann mit 00 und können zu 001 und 000 gewählt werden. Insgesamt ergibt sich:

Zeichen	Häufigkeit	Kodierung	Bits
A	1/2	1	1
B	1/3	01	2
C	1/18	001	3
D	1/9	000	3

Durch diese Art der Festlegung ist sichergestellt, dass kein Zeichen der Anfang eines anderen Zeichens ist. Diese Art der Kodierung bezeichnet man nach D. Huffman als Huffman-Kodierung. Für ein Musterfolge mit 18 Zeichen gemäß der Häufigkeitstabelle ergibt sich folgende Kodierung:

```
                  1 1 1 1 1 1 1 1 1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8

B A B B A C B B A A A A B D D A A A     gleichförmige
010001010010010100000000011111000000  Kodierung

B AB B AC  B B AAAAB D  D  AAA          Huffman-Kodierung
011010110010101111101000000111
```

Bei gleichmäßiger Kodierung benötigt man $18 \cdot 2 = 36$ Bits. Demgegenüber verbraucht die Huffmann-Kodierung 30 bits und im Mittel $30/18 = 5/3$ Bits pro Zeichen. Zu dem gleichen Wert kommt man durch die Summierung der Produkte von Häufigkeit und Bitlänge aller Zeichen:

$$\begin{aligned} 1/2 \cdot 1 + 1/3 \cdot 2 + 1/18 \cdot 3 + 1/9 \cdot 3 &= \\ 1/2 + 2/3 + 3/18 + 6/18 &= \\ 1/2 + 2/3 + 9/18 &= \\ 1 + 2/3 &= 5/3 = 1.6666 \end{aligned} \tag{7.1}$$

In dem betrachteten Beispiel war die Bestimmung der Kodes sehr einfach. Aber auch für aufwändigere Fälle - d. h. bei mehr Zeichen und beliebigen Häufigkeiten - kann man nach einem einfachen Algorithmus die Kodes berechnen. Im allgemeinen Verfahren unterteilt man die Menge aller Zeichen so, dass jede Teilmenge so gut wie möglich 50% Häufigkeit hat. Diese Teilmengen werden immer weiter verteilt, bis sie jeweils nur ein Zeichen enthalten. Diesen Vorgang kann man als Aufbau eines binären Entscheidungsbaumes implementieren. Zeichen mit großer Häufigkeit werden nach wenigen Entscheidungen erreicht. In dem Beispiel spart

die Huffmann-Kodierung 1/6 der Bits ein. Allerdings muss man bei korrekter Betrachtung noch Speicherplatz für die Kodierungstabelle berücksichtigen. Bei kurzen Zeichenfolgen wird dadurch der Nutzen der Huffmann-Kodierung vermindert.

7.4 Informationstheorie

Das Beispiel zeigt, dass man durch ungleichförmige Kodierung mit im Mittel weniger Bits pro Zeichen auskommt. Es stellt sich die Frage, wo die Grenze für diese Verbesserung liegt, d. h. wie viele Bits pro Zeichen mindestens benötigt werden. Diese Frage beantwortet die statistische Informationstheorie. Sie geht zurück auf die Zeit des sich entwickelnden Kommunikationstechnik. Wesentliche Impulse gab Claude Shannon[1] mit seinem Artikel *A Mathematical Theory of Communication* von 1948.

Das Grundmodell ist eine Quelle, die Zeichen aus einem endlichen Vorrat schickt. Für das Auftreten eines bestimmten Zeichens X gibt es eine feste Wahrscheinlichkeit $p(X)$. Die Information, die ein Zeichen übermittelt, hängt von seiner Häufigkeit ab. Seltene Zeichen tragen mehr Information als häufige Zeichen. Ein Beispiel sind die Buchstaben im Alphabet. Ein seltener Buchstabe wie Q oder Y liefert sehr viel mehr Information als einer der häufigen Buchstaben wie E oder A. Bei einem Kreuzworträtsel hilft in diesem Sinne ein seltener Buchstabe viel mehr beim Erraten des gesuchten Wortes. Als Maß für die Information eines Zeichens gilt der Logarithmus

$$H_e = \log 1/p(X) = -\log p(X) \tag{7.2}$$

Aus den Eigenschaften der Logarithmus-Funktion folgt:

- $H_e > 0$
- $p(X) = 1$ ergibt $H_e = 0$, das sichere Ereignis trägt keine Information
- $p(X) = 0$ ergibt $H_e = \infty$, bei extrem unwahrscheinliche Ereignisse steigt die Information in Richtung unendlich groß

Die Basis des Logarithmus ist im Prinzip frei wählbar. Üblich ist die Basis 2. Die Einheit für die Information ist dann „Bit“.

Beispiel 7.2 *Für ein Ereignis X mit der Häufigkeit 50% berechnet man*

$$H_e = -\log_2 p(X) = -\log_2 1/2 = -\log_2 2^{-1} = \log_2 2 = 1bit \tag{7.3}$$

bei $p(X) = 0.25$ gilt

$$H_e = -\log_2 p(X) = -\log_2 1/4 = -\log_2 2^{-2} = 2\log_2 2 = 2bit \tag{7.4}$$

[1] Claude Elwood Shannon, 1916-2001

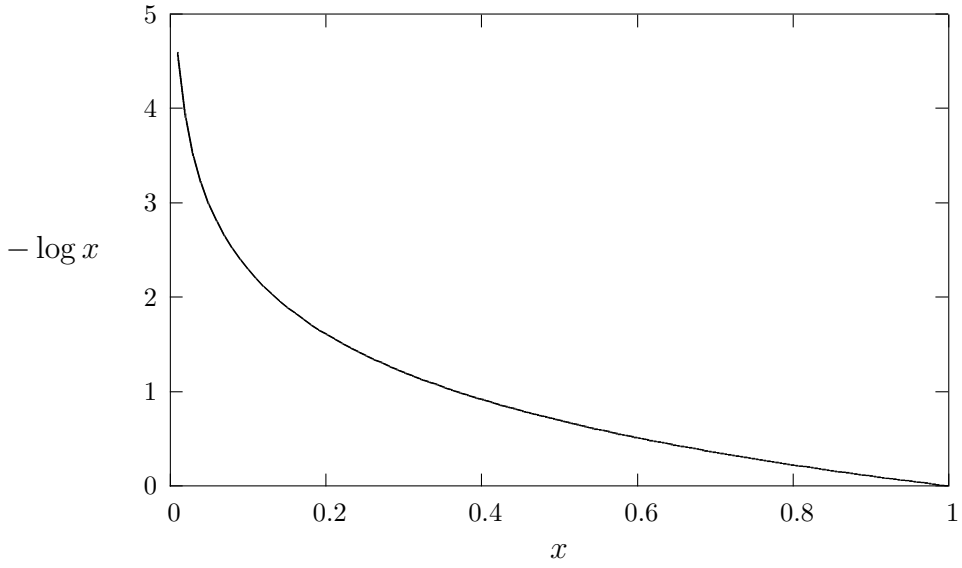


Abbildung 7.1: Verlauf der Funktion $-\log x$

Für das Beispiel mit 4 Zeichen erhält man

Zeichen	Häufigkeit	Information
A	1/2	1 bit
B	1/3	1.585 bit
C	1/18	4.170 bit
D	1/9	3.170 bit

Bei vielen Taschenrechnern fehlt die Funktion $\log_2$. Man kann sich dan mit folgender Umrechnung behelfen. Es gilt aufgrund der Definition des Logarithmus

$$x = 2^{\log_2 x}$$

und damit auch

$$\ln x = \ln 2^{\log_2 x}$$

Mit der Regel $\log x^y = y \cdot \log x$ lässt sich dies in

$$\ln 2^{\log_2 x} = \log_2 x \cdot \ln 2$$

umformen, so dass schließlich

$$\log_2 x = \ln x / \ln 2 = a \cdot lnx$$

mit dem konstanten Faktor $a = 1/\ln 2 \approx 1,442$ resultiert.

Mit der Definition (7.2) ist der Informationsgehalt eines einzelnen Zeichens gegeben. Betrachtet man die Quelle insgesamt, so stellt sich die Frage, wie viel Information ein einzelnes Zeichen im Mittel beträgt. Betrachten wir wieder das Beispiel mit 4 Zeichen A, B, C, D. In der Hälfte der Fälle erwarten wir das Zeichen A mit einer Information von 1 bit. In einem weiteren Drittel das Zeichen B mit 1.585 bit. Daher berechnet sich die mittlere Information, die wir mit einem neuen Zeichen erhalten, als

$$1/2 \cdot 1 + 1/3 \cdot 1.585 + 1/18 \cdot 4.170 + 1/9 \cdot 3.170 = 1,614 \tag{7.5}$$

Demnach trägt ein Zeichen im Mittel $1,614$ Bit Information. Allgemein kann diese Form zur Berechnung des gewichteten Mittelwertes (Erwartungswert) als

$$H = p(A) \cdot -\log_2 p(A) + p(B) \cdot -\log_2 p(B) + \ldots = \sum_i p(X_i) \cdot -\log_2 p(X_i) \tag{7.6}$$

oder

$$H = -\sum_i p(X_i) \log_2 p(Xi) \tag{7.7}$$

geschrieben werden. Die Größe H bezeichnet man als Entropie. Die Entropie ist ein Maß für den mittleren Informationsgehalt pro Zeichen einer Nachricht (Quelle). Für eine gegebene Anzahl von Zeichen hängt die Entropie von der Häufigkeit der einzelnen Zeichen ab. Die Bedeutung der Entropie liegt in der grundsätzliche Aussage für die Übertragung einer Nachricht: Um die Zeichen von einer Quelle mit einer Entropie H fehlerfrei zu übertragen, muss der Kanal eine Kapazität von mindestens H bit haben. In unserem Beispiel haben wir für die 4 Zeichen eine Entropie von $1,614$ bit berechnet. Dies ist damit die untere Grenze für die fehlerfreie Übertragung. Es ist auch bei noch so geschickter Kodierung nicht möglich, mit weniger Bits pro Zeichen fehlerfrei zu übertragen. In dem Beispiel ist die Huffman-Kodierung 1.666 bit pro Zeichen bereits recht dicht an dieser unteren Grenzen. Demgegenüber werden bei der gleichförmigen Kodierung mit 2 bit fast 0.4 bit „verschenkt“. Die Differenz zwischen Entropie und tatsächlichem Wert wird als Redundanz bezeichnet.

7.4.1 Eigenschaften der Entropie

In diesem Abschnitt soll die Frage untersucht werden, wie sich die Häufigkeitsverteilung auf die Entropie auswirkt. Der Einfachheit halber betrachten wir zunächst eine Quelle mit nur zwei möglichen Zeichen. Dann sei die Wahrscheinlichkeit für die beiden Zeichen p und $1 - p$. Die Entropie berechnet sich damit zu

$$H = -p \log_2 p - (1 - p) \log_2 (1 - p) \tag{7.8}$$

Der entsprechende Funktionsverlauf ist in Bild 7.2 dargestellt. Die Funktion hat ein Maximum von $H = 1$ bei $p = 0.5$ - d. h. bei gleichwahrscheinlichen

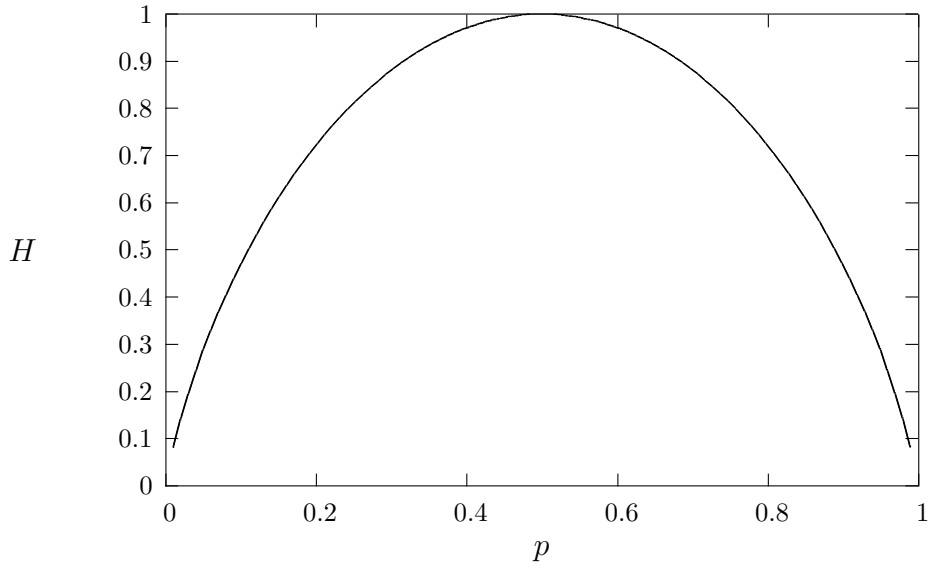


Abbildung 7.2: Verlauf der Entropie $-p \log_2 p - (1-p) \log_2 (1-p)$

Zeichen. Sobald die beiden Zeichen unterschiedliche Wahrscheinlichkeiten haben sinkt die Entropie. Der Wert wird bei zunehmender Ungleichverteilung immer stärker. In den Extremfällen $p = 1$ oder $p = 0$ tritt nur noch ein Zeichen auf und damit geht die Information gegen 0. Dieses Ergebnis lässt sich auf Quellen mit mehreren Zeichen erweitern. Im allgemeinen gilt mit N gleichwahrscheinlichen Zeichen:

$$\begin{aligned} H &= -\sum_i 1/N \log_2 1/N \\ &= \sum_i 1/N \log_2 N \\ &= N \cdot 1/N \log_2 N \\ &= \log_2 N \end{aligned} \tag{7.9}$$

Damit gilt:

- Bei gleichwahrscheinlichen Zeichen wird die Entropie maximal und hat den Wert $\log_2 N$
- Bei unterschiedlichen Häufigkeiten wird die Entropie kleiner. Dann werden weniger als $\log_2 N$ bits zur Übertragung benötigt.

Die Huffman-Kodierung nutzt diesen Effekt auf der Ebene der Binärzeichen. Aus der gleichförmigen Kodierung resultieren für die oben betrachtete Folge 11 Ein-

sen und 25 Nullen. Demgegenüber beträgt das Verhältnis nach der Huffman-Kodierung 16 : 14 - eine wesentlich bessere Annäherung an den idealen Wert von 1 : 1.

Übung 7.1 *Ihr Kurs feiert in einem Biergarten (32 Teilnehmer). Bei jeder Runde werden im Mittel folgende 6 Getränke bestellt:*
16 Mineralwasser, 8 Bier, 4 Apfelsaft, 2 Apfelwein, 1 Radler , 1 Espresso
Der Wirt schlägt vor, die Bestellung per Handzeichen direkt an die Theke zu signalisieren. Sein Vorschlag für die Zeichen:

Mineralwasser 

Bier 

Apfelsaft 

und so weiter. Damit würden Sie pro Runde $3 \cdot 32 = 96$ *Daumenzeichen benötigen. Welche Alternative schlagen Sie vor?*

7.4.2 Abhängigkeiten zwischen Zeichen

Bei der bisherigen Definition der Entropie wird die unterschiedliche Häufigkeit der Zeichen berücksichtigt. Ignoriert werden dabei Abhängigkeiten zwischen aufeinander folgende Zeichen. Beispiele:

1. *AABBDDAACCBBAADDAA*
2. *ABCDABCDABCD*
3. *Sein oder Nichtsein, das ist hier die Frage*

Im ersten Beispiel treten die Zeichen immer paarweise auf. Auch wenn jetzt die vier Zeichen die gleiche Häufigkeit haben, ist der Informationsgehalt sicherlich geringer als im allgemeinen Fall. Das zweite Beispiel zeigt eine periodische Folge. Auch in diesem Fall würde die einfache Entropie-Formel dieses Verhalten nicht erfassen. Schließlich enthalten auch Sprachen Abhängigkeiten zwischen den Buchstaben. Eine Reihe von Regeln schränkt in vielen Fällen die Anzahl der möglichen Buchstaben ein. Einige Beispiele für Deutsch:

- Nach Q folgt U
- Nach SC folgt H
- Kein Buchstabe kommt mehr als dreimal hintereinander

Bestehen derartige Abhängigkeiten, so können Sie für eine sparsamere Kodierung ausgenutzt werden. Wie groß der Gewinn ist, kann mit einer entsprechend verallgemeinerten Form der Entropie berechnet werden. Bezeichnen wir die Folge von Zeichen mit

$$\ldots, X_{t-2}, X_{t-1}, X_t \tag{7.10}$$

Dann ist die Wahrscheinlichkeit für ein Zeichen in Abhängigkeit von den Vorgängern die bedingte Wahrscheinlichkeit

$$p(X_t|X_{t-1}, X_{t-2}, \ldots) \tag{7.11}$$

Die bedingte Entropie ist dann definiert als

$$-\sum_{i,j,k,\ldots} p(X_i, X_j, X_k, \ldots) \cdot \log_2 p(X_i|X_j, X_k, \ldots) \tag{7.12}$$

Je nachdem wie viele Vorgänger man dabei berücksichtigt, bezeichnet man die Entropie dann als H_1 bei einem Vorgänger, H_2 bei 2 Vorgängern, u.s.w. . Anschaulich beschreibt die bedingte Entropie H_N den Informationsgewinn, wenn man bereits N Vorgänger kennt. Da das Wissen über die Vorgänger den Informationswert des aktuellen Zeichens einschränkt, gilt

$$H_0 \geq H_1 \geq H_2 \ldots \tag{7.13}$$

Wenn die Zeichen vollkommen voneinander unabhängig sind, gewinnt man nichts aus der Kenntnis der Vorgänger. In diesem Fall gilt das Gleichheitszeichen. Ansonsten nimmt die bedingte Entropie mit zunehmendem Vorwissen ab. Den Grenzwert H - die bedingte Entropie bei Kenntnis von allen Vorgängern - bezeichnet man dann als Entropie der Quelle. Sie ist die untere Grenze für die „sparsamste“ Kodierung der Zeichen.

Beispiel 7.3 *Für die Entropie verschiedener Schriftsprachen gilt:*

Sprache	H_0	H_1	H_3	H
Deutsch	*4,75*	*4,11*	*2,8*	*1,6*
Englisch	*4,75*	*4,03*	*3,1*	*2,0*
Russisch	*5*	*4,35*	*3,0*	

7.5 Kompressionsverfahren

Aus der Diskussion zum Informationsgehalt einer Nachricht ergeben sich zwei Ansatzpunkte zur Kompression der Daten:

- Ungleiche Häufigkeit von Zeichen
- Abhängigkeiten zwischen aufeinander folgenden Zeichen (ungleiche Häufigkeit von Zeichenpaaren)

Mit der Huffman-Kodierung haben wir bereits ein leistungsfähiges Verfahren kennen gelernt, das eventuelle Unterschiede in der Häufigkeit ausnutzt. Die in diesem Fall benötigte Häufigkeitsverteilung der Zeichen kann mit relativ geringem Aufwand gemessen werden. Die Abhängigkeiten zwischen den Zeichen sind schwerer zu erfassen. Hier spielt die Art der Nachricht eine große Rolle. So besteht bei Bilddateien eine Verwandtschaft zwischen benachbarten Pixel. Wenn die Daten zeilenweise abgelegt sind, resultiert allerdings ein großer Abstand zwischen den in Spaltenrichtung benachbarten Pixel. Die bedingte Entropie wird daher bei z. B. einer Zeilenbreite von 300 Pixel die Abhängigkeit von dem 300. Vorgänger feststellen. Solche spezifischen Strukturen innerhalb der Nachricht sind schlecht mit einem universellen Ansatz zu behandeln. Daher gibt es für verschiedene Datentypen angepasste Verfahren. Im folgenden werden beispielhaft zwei Basisverfahren zur Datenkompression vorgestellt. Die beiden Methoden können durch Verfeinerung und Kombination untereinander und mit anderen Algorithmen verbessert werden.

7.5.1 Lauflängenkodierung

In vielen Fällen treten die Zeichen nicht vereinzelt auf. So sind bei einem Schwarzweiß-Bild längere Folgen von Weiß oder Schwarz häufig. In solchen Fällen kann man durch Kodierung der Anzahl der aufeinanderfolgenden Zeichen eine deutliche Kompression erreichen. Sei

```
xxxooooooxxxxoooooooooxxxxxx
```

eine Zeile in einem Bild. Dann wird bei der Lauflängenkodierung (*run length encoding RLE*) die Information wie folgt gespeichert:

```
3x 5o 4x 8o 6x
```

Bei binären Zeichen kann diese Kodierung wesentlich verbessert werden. Jeder neue Zähler bedeutet ein Wechsel des Zeichens. Daher braucht das neue Zeichen nicht angegeben zu werden. Aus der Folge wird

```
3x 5 4 8 6
```

Diese Art der Kodierung wird bei der Übertragung von Faxen eingesetzt. Damit wird beispielsweise eine leere Zeile mit einem einzigen Zähler kodiert.

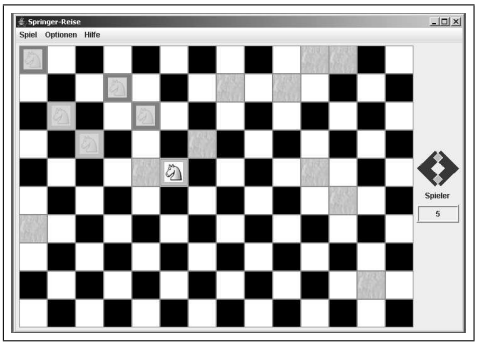
7.5.2 Komprimierung mit Tabellen

Eine umfassende Berechnung der Verbundwahrscheinlichkeiten ist sehr aufwändig und für kleine Datenmenge wenig aussagekräftig. Ein pragmatischen Ansatz ist, nach häufig vorkommenden Zeichenpaaren oder allgemein längeren Zeichenfolgen zu suchen. Diese werden dann platzsparend durch spezielle Zeichen kodiert.

Beispielsweise könnten im Deutschen sehr häufige Folgen wie *sch*, *und* oder *der* durch abkürzende Zeichen ersetzt werden. Die Folgen werden in eine Tabelle eingetragen. In der komprimierten Datei stehen dann nur noch die Verweise in die Tabelle. Auf diesem Grundgedanken beruht der LZ-Algorithmus, benannt nach seinen Erfindern Abraham Lempel und Jacob Ziv (1977). Aus dem ursprünglichen Algorithmus wurden eine ganze Reihe von Weiterentwicklungen abgeleitet. Die verschiedenen Algorithmen unterscheiden sich in der Strategie zum Aufbau der Tabelle und der möglichst effizienten Speicherung der Tabelle.

7.6 Übungen

Übung 7.2 *Entropie*
Die beiden folgenden Bilder sind im Bitmap-Format unkomprimiert gespeichert. Sie lassen mit einem Programm die Entropie in den beiden Dateien berechnen und erhalten die zwei Werte 3,38 und 7,79. Welchen Entropiewert gehört zu welcher Datei (Begründung)?



Kapitel 8

Zentraleinheit

8.1 Aufbau

Das Herzstück des von Neumann-Rechners ist die Zentraleinheit oder Central Processing Unit (CPU). Ursprünglich wurde eine CPU aus einer Reihe von Bausteinen aufgebaut. Heute ist sie zusammen mit weiteren Komponenten in einem einzigen Chip integriert. Im einzelnen enthält die CPU folgende Elemente:

- Rechenprozessor
- Steuerprozessor
- Speicherzellen (Register, Akkumulatoren)

Dabei ist der Steuerprozessor für den gesamten Ablauf zuständig. Er holt Befehle, interpretiert sie, lädt eventuell weitere Daten oder beauftragt den Rechenprozessor mit den gewünschten Operationen. Die internen Speicherzellen kann man nach ihrer Funktion in

- Adressregister
- Datenregister
- Spezialregister

unterteilen. Ein Teil der Register ist für den Benutzer zugänglich während andere Register nur für interne Abläufe verwendet werden. Der Datenaustausch zwischen den Registern untereinander sowie dem Rechenprozessor erfolgt sehr schnell über interne Busse. Anzahl, Größe und Funktion der Register bestimmen ganz wesentlich die Eigenschaften der CPU. Man spricht daher auch von dem *Programmiermodell*. Das Programmiermodell gibt die Möglichkeiten in der maschinennahen Programmierung für einen konkreten Rechner vor.

In diesem Kapitel werden allgemeine Grundprinzipien für den Aufbau der CPU behandelt. Eine konkrete Realisierung wird am Beispiel des Prozessors Intel 8086 im nächsten Kapitel vorgestellt.

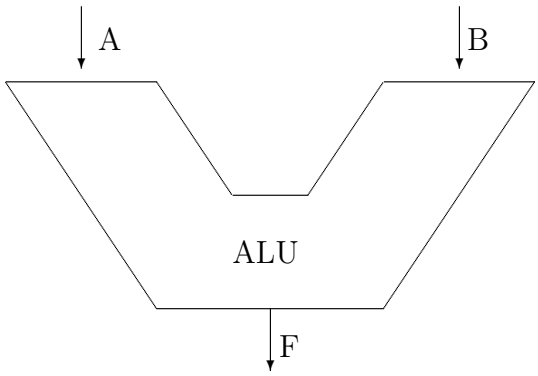


Abbildung 8.1: Aufbau einer ALU

8.2 Rechenprozessor

Kernstück des Rechenprozessors ist die so genannte arithmetisch-logische Einheit oder auf englisch *Arithmetic Logical Unit* (ALU). Die ALU kann logische Operationen wie UND, ODER und Negation sowie arithmetische Verknüpfungen wie Addition oder Subtraktion ausführen. Das Grundprinzip zeigt Bild 8.1.

Die ALU übernimmt zwei Operanden, führt die von dem Steuerprozessor angegebene Operation durch und gibt das Resultat aus. Da bei einem von Neumann Rechner nur ein sequentieller Zugriff auf den Hauptspeicher möglich ist, können die Operanden nicht direkt aus dem Hauptspeicher gelesen werden. Vielmehr müssen die Operanden zunächst in Register gebracht werden. In vielen Fällen ist die Operation nur Teil einer längeren Verarbeitungskette. Es ist daher sinnvoll, das Resultat ebenfalls zunächst in ein Register zu schreiben und nur bei Bedarf in einem weiteren Schritt von dort zum Hauptspeicher zu transportieren. Ein Befehl für eine typische Operation wie die Addition hat dann die Struktur

```
add register1,register2,register3
```

Man spricht in diesem allgemeinen Fall dem entsprechend von einer Dreiadressmaschine. Bild 8.2 zeigt den entsprechenden Aufbau. Dieser Aufbau bietet die größte Flexibilität. Allerdings wird diese Freiheit mit einem großen Aufwand erkauft. Da jeder Befehl drei Adressen enthält, steigt sowohl der notwendige Speicherplatz für die Befehle als auch der Aufwand zur Dekodierung der Befehle.

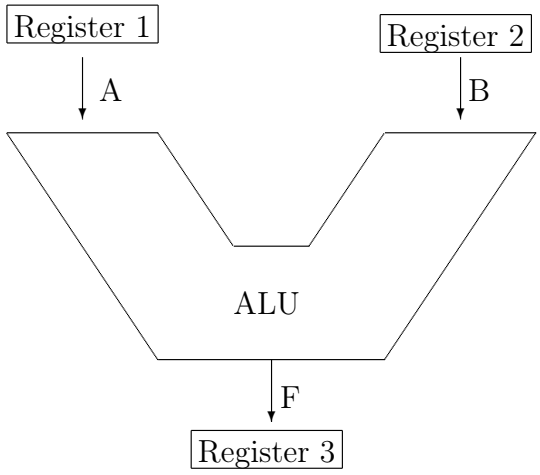


Abbildung 8.2: Dreiadressmaschine

Eine aufwandsgünstigere Lösung bieten Zweiadressmaschine (Bild 8.3). In diesem Fall wird das Ergebnis nicht in ein eigenes Register geschrieben sondern in eines der beiden Register mit den Operanden. In dem Bild wird das zweite Register als Ziel verwendet. Das Resultat steht nach Ende der Rechnung wieder als neuer Operand zur Verfügung. Ein Nachteil ist, dass der zweite Eingangswert überschrieben wurde. Eine optimierte Programmierung erfordert daher eine sorgfältige Planung, welcher Wert wann in welches Register geschrieben wird. Zweiadressmaschine bieten einen guten Kompromiss zwischen Flexibilität und Aufwand. Daher sind die meisten Prozessoren heute als Zweiadressmaschinen ausgelegt.

Noch kürzere Befehle erreicht man mit Einadressmaschinen. In diesem Fall wird der zweite Operand immer aus dem gleichen Register genommen. Sinnvollerweise wird das Resultat ebenfalls in dieses Register geschrieben. Derartige Register nennt man *Akkumulatoren*. Diese Struktur ist beispielsweise gut geeignet, die Summe aus einer Anzahl von Werten zu berechnen. Dazu wird der Akkumulator zu Beginn gelöscht. Anschließend werden nur noch Operationen in der Art

```
add wert_n
```

benötigt, um die Summe im Akkumulator zu bilden. Häufig erfordert allerdings eine Einadressmaschine einen zusätzlichen Befehl, um den Akkumulator vor einer Operation mit dem benötigten Operanden zu laden.

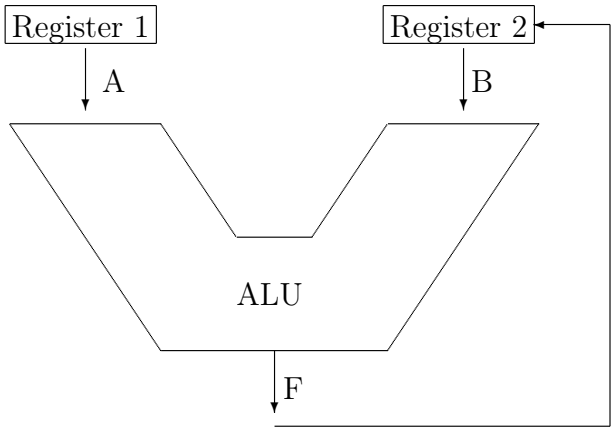


Abbildung 8.3: Zweiadressmaschine

Die kompaktesten Befehle bietet eine Nulladressmaschine. Ein Befehl spezifiziert dann nur noch die auszuführende Operation. Dazu müssen die Operanden stets an festgelegten Stellen liegen. Ein Beispiel ist das Rechnen mit der so genannten *Umgekehrt Polnischen Notation* (UPN), wie es auch in einigen Taschenrechnern realisiert wurde. Dabei werden zunächst die Operanden in einen Stapelspeicher (engl. *Stack*) geschrieben.

Ein Stapelspeicher ist ein Speicher, bei dem Werte nur am Anfang angefügt oder entnommen werden können. Anschaulich kann man sich die Werte übereinander liegend wie bei einem Stapel Spielkarten oder Mensatabletts vorstellen. Man kann jeweils eine weitere Karte bzw. ein weiteres Tablett auflegen oder vom Stapel nehmen. Diese beiden Operationen nennt man *push* (Auflegen) und *pop* (Wegnehmen). Charakteristisch für ist, dass die Elemente in umgekehrter Reihenfolge entnommen werden (*Last In First Out*, LIFO).

Bei der Ausführung einer Operation werden dann die beiden obersten Werte vom Stack genommen, verarbeitet und das Resultat wird wieder auf den Stack geschrieben. Das Result kann dann von dort abgeholt werden und steht aber auch als Zwischenergebnis für weitere Rechenschritte bereit. Durch geschicktes Belegen des Stacks lassen sich auf diese Art und Weise längere Rechnungen effizient ausführen.

Beispiel 8.1 *Berechnung von* $6 * (7 + 2)$ *in UPN.*

```
6       // Stack: 6
7       // Stack: 6 7
```

```
2       // Stack: 6 7 2
+       // Stack: 6 9
*       // Stack: 54
```

8.2.1 Zahlenbereich und Statusmeldungen

Bei den arithmetischen Operationen muss der Wertebereich des Ergebnisses berücksichtigt werden. Betrachten wir die Multiplikation zweier Zahlen A und B. Der Einfachheit halber sei die Rechnung auf positive Zahlen beschränkt. Als Binärzahlen mit n beziehungsweise m Stellen lassen sie sich als

$$A = 2^n + \ldots$$

und

$$B = 2^m + \ldots$$

schreiben. Das Produkt enthält dann als höchste Potenz von 2 den Wert $n + m$:

$$A \cdot B = (2^n + \ldots) \cdot (2^m + \ldots) = 2^{n+m} + \ldots$$

Das Resultat benötigt insgesamt $n + m$ Bit. Bei Eingangsregistern der Größe N kann das Ergebnis einer Multiplikation demnach im Extremfall $2N$ Stellen haben. Um das Resultat komplett speichern zu können, wird ein weiteres Register der Größe N benötigt. Dann steht das Resultat zumindest intern vollständig für weitere Rechnungen zur Verfügung. Bei längeren Rechnungen mit mehreren Multiplikationen müssen die Werte gegebenenfalls skaliert werden, um einen Überlauf zu verhindern.

Generell ist es häufig erforderlich, nach dem Ausführen einer Operation auf bestimmte Bedingungen zu prüfen. Einerseits können damit Problemfälle erkannt werden. Andererseits kann der weitere Ablauf des Programms von bestimmten Bedingungen abhängen. Eine typische Anwendung ist eine Zählschleife, die beendet wird sobald der Zähler den Wert 0 erreicht. Beispiele für solche Bedingungen sind:

- Überlauf
- Vorzeichen des Ergebnisses
- Resultat 0

Die jeweiligen Werte oder Zustände werden in dem Statusregister gespeichert. Das Statusregister besteht aus einer Reihe von Bits (so genannte *Flags*), die bestimmte Zustände signalisieren. Nach einer Operation setzt die CPU die zugehörigen Flags. Dort steht die Information für die weitere Ablaufsteuerung zur Verfügung. Insgesamt ergibt sich damit am Beispiel einer Zweiadressmaschine Bild 8.4.

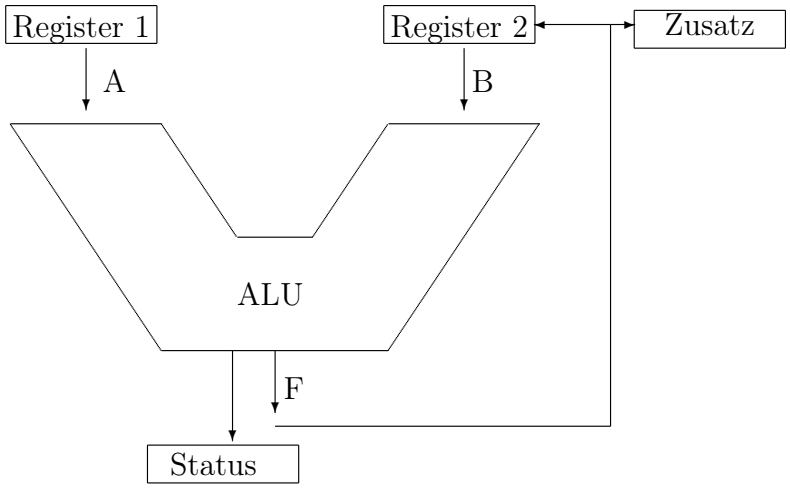


Abbildung 8.4: Zweiadressmaschine mit zusätzlichem Ergebnisregister und Statusregister

8.3 Steuerprozessor

Der Steuerprozessor ist zuständig für den gesamten Ablauf. Er stellt die Verbindung zum Speicher bereit. Von dort holt er Befehle und Daten und speichert Ergebnisse wieder zurück. Die Adresse des nächsten Befehls entnimmt der Steuerprozessor einem speziellen Register – dem Befehlszähler (*Program Counter*, PC). Er holt von der dort stehenden Adresse das Bitmuster und lädt es in ein weiteres Register. Dieses Befehlsregister (*Instruction Register*, IR) hat nur interne Bedeutung und kann vom Anwender nicht angesprochen werden.

8.3.1 Maschinenbefehle

In dem Bitmuster des Befehls ist der Befehl kodiert. Der Steuerprozessor muss umgekehrt das Bitmuster interpretieren. Dabei sind zwei Fragen zu unterscheiden:

- Welche Operation ist auszuführen?
- Welche Daten sind zu verwenden?

Dem entsprechend sind die Bitmuster in zwei Teilen organisiert: dem Befehlscode (OP-Code) und dem Operandenteil. Es ergibt sich folgendes Bild:

Befehlscode	Operanden

Die Größe des Feldes für den Befehlscode legt fest, wie viele verschiedene Befehle maximal möglich sind. Sind n Bit dafür reserviert, können maximal 2^n verschiedenen Befehle unterschieden werden. Beim Intel 8086 Prozessors bezeichnet beispielsweise das Bitmuster `1000 0011` oder in Hexadezimaldarstellung `83` eine Subtraktion. Diese Codierung der Befehle in Bitmuster bildet die Maschinensprache des jeweiligen Prozessors. Letztendlich liegt jedes ausführbare Programm in dieser Maschinensprache vor. Die CPU kann nur auf dieser niedrigsten Ebene Anweisungen verstehen.

In vielen Fällen wird der Vorrat an möglichen Befehlen nicht ausgeschöpft. Einige Bitmuster bleiben ohne Belegung. Werden dann beispielsweise durch eine fehlerhafte Adressierung Daten als Befehle interpretiert, so enthält der Befehlscode früher oder später einen ungültigen Wert. Der Prozessor meldet dann einen entsprechenden Fehler und bricht die Ausführung ab.

8.3.2 Assembler

Ein Programm in Maschinensprache zu schreiben ist extrem umständlich. Man hat dabei zwar die vollständige Kontrolle über den Programmcode und damit die weitestgehenden Möglichkeiten zur Optimierung von Laufzeit und Speicherbedarf. Aber der mit der Programmierung in Maschinensprache verbundene hohe Aufwand ist nur in wenigen Spezialfällen gerechtfertigt.

Eine Stufe höher sind die maschinenorientierten Sprachen angesiedelt. Eine solche Sprache besteht aus leichter zu merkenden Wörter für die Befehle (z. B. `SUB` statt `83`). Im nächsten Kapitel werden wir im Detail die Syntax einer solchen Sprache kennen lernen. Zur Ausführung muss ein entsprechendes Programm in Maschinensprache übersetzt werden. Diese Aufgabe übernimmt ein Assembler (engl. Monteur, „Zusammensetzer“). Die Maschinenbefehle für einen Prozessor sind von dem Hersteller fest vorgegeben. Auch die dazu gehörenden Befehlswörter der Assemblersprachen sind standardisiert. Allerdings kann es trotzdem verschiedene Assembler für einen Prozessor geben, die sich im Leistungsumfang der Sprache unterscheiden. Dies betrifft beispielsweise die Verwendung von symbolischen Namen oder die Möglichkeiten, Makros für häufig wieder kehrende Teile zu definieren.

8.3.3 Operanden

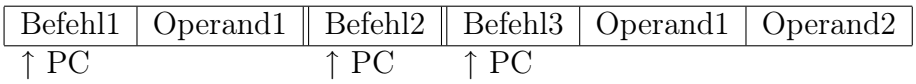
Die Anzahl der Operanden hängt vom jeweiligen Befehl ab. Einige Beispiele aus dem Befehlsvorrat des Intel 8086 Prozessors:

- `CLC` Clear Carry Flag. Dieser Befehl löscht ein bestimmtes Bit im Statusregister. Er benötigt keinen Operanden.

- `DEC A` Decrement. Der Operand `A` wird um eins erniedrigt `A = A - 1`.

- `ADD A,B` Addition. Die Summe der beiden Operanden wird berechnet und an die Stelle des ersten Operanden geschrieben: `A = A + B`.

Im einfachsten Fall ist der Befehlscode bereits ein vollständiger Befehl. Dann kann der Steuerprozessor unmittelbar nach der Interpretation des Bitmusters die entsprechende Aktion veranlassen. Andererseits gibt es Befehle, die ein oder zwei weitere Operanden enthalten. Davon hängt wiederum die Länge des Befehls ab. Bei dem 8086 Prozessor variiert die Länge der Befehle zwischen 1 Byte und 6 Byte. Der Steuerprozessor erkennt anhand des Befehlscodes die Anzahl und Größe der Operanden. Sofern erforderlich lädt er die Operanden aus dem Speicher. Anschließend setzt er den Programmzähler auf den Anfang des nächsten Befehls. Graphisch lässt sich der Aufbau eines Programms im Speicher und die entsprechende Abfolge der Werte des Programmzählers folgendermaßen darstellen:



8.3.4 Phasen der Befehlsausführung

Insgesamt führt der Steuerprozessor folgende Schritte aus:

- Lesen des Befehlscodes
- Interpretation des Codes (Decodierung)
- Gegebenenfalls Operanden aus dem Speicher holen
- Steuersignale an ALU oder andere Einheiten setzen

Diese Phase wird als Interpretations- oder *Fetch*-Phase bezeichnet. Anschließend folgt die Ausführungs- oder *Execution*-Phase. Der Ablauf hängt dann stark von der Art der Operation ab. Beispielsweise führt die ALU eine Verknüpfung durch oder ein Wert wird zwischen Registern und Hauptspeicher ausgetauscht. Sofern das Statusregister von der Operation betroffen ist, werden nach Ausführung die entsprechenden Flags gemäß dem Resultat gesetzt.

Während der Ausführungsphase wird auch der Programmzähler auf den nächsten Befehl gesetzt. Im linearen Ablauf berechnet sich die neue Adresse aus der alten plus der Länge des aktuellen Befehls. Bei Sprungbefehlen enthält der Befehl selbst die Information über die nächste Befehlsadresse.

Aus der bisherigen Betrachtung wird klar, dass die Ausführung eines Befehles in den allermeisten Fällen mehrere Speicherzugriffe erfordert. Sowohl das Laden des vollständigen Befehls als auch der Operanden führt zu Zugriffen auf den Hauptspeicher. Dabei müssen die einzelnen Speicherzellen nacheinander abgefragt werden. Bei einem von Neumann Rechner gibt es keine Trennung zwischen Programm- und Datenspeicher. Daher ist auch für Programm und Daten eine sequentielle Behandlung notwendig. Insgesamt führt dies dazu, dass bei modernen

Prozessoren die Speicherzugriffe immer mehr zum bestimmenden Faktor für die Ausführungszeit werden. Man spricht in diesem Zusammenhang vom von Neumann Flaschenhals. In einem späteren Kapitel werden wir auf diese Problematik zurück kommen und verschiedene Lösungsansätze untersuchen.

8.3.5 Mikroprogramme

Ein Maschinenbefehl wird in einer Anzahl von Einzelschritten ausgeführt. Diese Einzelschritte können direkt in Hardware durch entsprechende Schaltungen realisiert sein. Flexibler ist die Lösung mit Mikroprogrammen. Zu einem Maschinenbefehl gehört dann ein internes Programm. Das Programm enthält die Instruktionen für die einzelnen Schritte. Die Mikroprogramme stehen in einem speziellen Speicher innerhalb der CPU. Bei manchen Prozessoren besteht die Möglichkeit, eigene Mikroprogramme in diesen Speicher zu laden und damit so zu sagen neue Maschinenbefehle zu definieren.

8.4 Übungen

Übung 8.1 *X, Y und Z bezeichnen drei Zellen im Hauptspeicher. Dafür soll der Ausdruck $Z = X^3 + Y * X$ umgesetzt werden. Geben Sie Assembler-Programme für die folgenden Architekturen an:*

- *Dreiadressmaschine:*
  - *sechs Register R1 bis R6*
  - `MOV speicher,register`, `MOV register,speicher`
  - `MOV register,register`
  - `ADD Ra,Rb,Rc` *und* `MPY Ra,Rb,Rc` *wobei* `a,b` *und* `c` *einen Index zwischen 1 und 6 bezeichnet.*
- *Zweiadressmaschine: wie Dreiadressmaschine, allerdings haben ADD und MPY nur zwei Operanden, das Ergebnis überschreibt den ersten Operanden*
- *Einadressmaschine:*
  - *zusätzliches Register A*
  - *bei* `ADD` *und* `MPY` *ist automatisch A erster Operand*
- *Nulladressmaschine*
  - *Stapelspeicher S*
  - `PUSH speicher` *kopiert Wert auf den Stapel,* `POP speicher` *nimmt obersten Wert vom Stapel und transferiert ihn in den Hauptspeicher*

- *`ADD` und `MPY` entnimmt dem Stapel die beiden obersten Werten und führt die entsprechende Operation aus, das Ergebnis wird wieder auf den Stapel gelegt*

Ignorieren Sie eventuelle Probleme der Darstellungsgenauigkeit. Der Hauptspeicher soll insgesamt 256 Zellen groß sein. Wie viele Bit benötigt man in den einzelnen Fällen zur Kodierung der Befehle und Operanden?

Übung 8.2 *Betrachten Sie ein System mit 64k (64 * 1024) Hauptspeicher, wobei jede Speicherzelle 16 Bit fasst. Bei jeden Transfer können allerdings nur 8 Bit zwischen Hauptspeicher und CPU ausgetauscht werden. Der Befehlsvorrat des Prozessors umfasst 256 verschiedene Kommandos. Wie sieht der Ablauf bei einem MOV-Befehl zum Kopieren eines 16 Bit Wertes aus dem Speicher in ein CPU-Register aus? Wie viele Bustransfers sind erforderlich? Wie viele davon entfallen auf die Fetch- und auf die Execution-Phase?*

Kapitel 9

Der Intel 8086 Prozessor

9.1 Einführung

In diesem Kapitel werden wir am Beispiel des Prozessors Intel 8086 eine Architektur im Detail kennen lernen. Dieser Prozessor wurde 1978 von Intel eingeführt. Er besteht aus etwa 30000 Transistoren und arbeitet mit Taktfrequenzen von 5 bis 10 MHz. Der Prozessor ist aufgrund seiner Überschaubarkeit gut geeignet, um grundlegende Prinzipien zu veranschaulichen. Gleichzeitig ist er Teil der IA-32 Intel Architektur [Inta, Intb], und damit ein Vorläufer der modernen Prozessoren der Pentium-Gruppe.

Als praktische Beispiele werden kleine Programme dienen. Zur Entwicklung und Ausführung der Programm wird das Programm Emu8086 (www.emu8086.com) eingesetzt. Dieses Programm bietet eine komplette Entwicklungsumgebung mit Assembler, Simulator (Software Emulator) und Debugger. Der Assembler übersetzt Programme, die in der zugehörigen Assembler-Sprache geschrieben sind, in ausführbaren Maschinencode.

Der Simulator ist die software-mäßige Nachbildung des Prozessors. Es handelt sich so zu sagen um einen virtuellen Prozessor mit all seinen Komponenten. In dem Simulator können Programme ausgeführt werden und darüber hinaus Ein- und Ausgabe mit virtuellen Geräten ausgetauscht werden. Zur gezielten Fehlersuche dient ein eingebauter Debugger. Damit können beispielsweise Programme beim Eintreten von vordefinierten Ereignissen (Break points) angehalten werden.

9.2 Register

Der 8086 verfügt über 8 allgemeine Register:

- AX - Accumulator register (unterteilt in AH und AL)
- BX - Base address register (unterteilt in BH und BL)

- CX - Count register (unterteilt in CH und CL)
- DX - Data register (unterteilt in DH und DL)
- SI - Source Index register
- DI - Destination Index register
- BP - Base Pointer
- SP - Stack Pointer

Die vier 16 Bit Register AX bis DX können in je zwei 8 Bit breiten Teilen angesprochen werden (H wie High und L wie Low). Ist beispielsweise der Wert von CX 0xe45f, so hat CH den Wert 0xe4 und CL den Wert 0x5f. Eine Änderung in einem der Teilregister betrifft genauso das gesamte Register. Löscht man CH, so hätte in dem Beispiel CX anschließend den Wert 0x005f. Wie die Namen zeigen, sind die 8 Register jeweils für bestimmte Zwecke vorgesehen. Welche Verwendung im einzelnen durch entsprechende Befehle unterstützt wird, werden wir später im Detail sehen. Allerdings ist man relativ frei in der Nutzung. Zur erweiterten Adressierung stehen vier Segmentregister zu Verfügung:

- CS Code Segment
- DS Daten Segment
- ES Extra Segment
- SS Stack Segment

Die Adressierung wird später im Abschnitt 9.5 im Detail beschrieben. Schließlich gibt es noch zwei Spezialregister:

- IP - Instruction Pointer
- Flags Register

IP weist auf den aktuellen Befehl. Das Flags Register enthält Statusinformationen und wird daher auch als Statusregister bezeichnet.

9.3 Erstes Beispiel

Das folgende Programm lädt zunächst zwei Zahlen in die beiden Register AX und BX. Dazu wird der MOV-Befehl[1] verwendet. Der erste Operand bezeichnet

[1]Groß- und Kleinschreibung spielen bei dem Assembler keine Rolle. Die Schreibweisen MOV und mov sind gleichwertig

das Ziel, der zweite enthält direkt den Zahlenwert. Zahlenwerte können in der Assemblersprache in verschiedenen Systemen angegeben werden. Die Endungen b, o und h kennzeichnen Werte als Binär-, Oktal- oder Hexadezimalzahl. Im Programm werden dann beide Registerinhalte mit dem Befehl ADD addiert. Das Ergebnis 322_{10} oder 142_{16} steht dann in AX.

Die Befehlszeilen bestehen jeweils aus dem Befehlswort (Mnemonic) und den eventuellen Operanden. Ein Semikolon markiert den Rest der Zeile als Kommentar. Neben den eigentlichen Assemblerbefehlen können Direktiven an den Assembler eingefügt werden. Entsprechende Anweisungen beginnen mit dem #-Zeichen.

```
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h    ; Programm an Adresse 100h beginnen

mov AX,67   ; Kommentare beginnen mit ;
mov BX,0ffh ; Hex-Konstanten werden mit h gekennzeichnet und
            ; müssen mit einer Ziffer beginnen
add AX,BX   ; AX = AX + BX
ret         ; Ende des Programms
```

Das Beispiel ist als COM-Typ angelegt. Dies ist eine einfache Form von ausführbaren Programmen. Mit der Anweisung `ORG` (*origin*, engl. Ursprung) wird die Anfangsadresse des Programms festgelegt. Bei Programmen vom Typ COM sind die ersten 100_{16} Zellen für Systeminformationen reserviert.

Der Assembler generiert aus diesem Programm die entsprechenden Binärdarstellung. Dieser Code muss dann noch in den Hauptspeicher geladen werden. Dann kann die Ausführung ab der Startadresse beginnen. Der Simulator ersetzt für Testzwecke eine realen Maschine. Im Speicher liegt das Programm in folgender Form vor:

100	101	102	103	104	105	106	107	108	Adresse
b8	43	00	bb	ff	00	03	c3	c3	Maschinencode
mov AX,67			mov BX,0ffh			add AX,BX		ret	Befehl

Die beiden MOV-Befehle benötigen jeweils nur ein Byte Maschinencode. Dabei enthält dieses Byte bereits die Information über das Ziel-Register. In den beiden darauf folgenden Bytes steht der Zahlenwert, wobei das niedrigwertige Byte zuerst kommt.

9.4 Befehle

In diesem Abschnitt wird der Befehlssatz des 8086 vorgestellt. Zunächst werden die wesentlichen Befehlsarten erläutert. Anschließend werden einige Befehle im Detail untersucht. Intel unterscheidet zwischen folgenden 6 Befehlsarten:

1. Transportbefehle: Befehle, mit denen Werte zwischen verschiedenen Einheiten kopiert werden können. Dazu gehört der wahrscheinlich am häufigsten verwendete Befehl MOV, PUSH und POP für den Stack sowie IN und OUT für periphere Ein- und Ausgabe.

2. Arithmetische Befehle: Die Grundoperationen für ganzzahlige Operanden ADD, SUB, MUL und DIV und weitere Befehle wie INC, DEC, NEG, ...

3. Logische Befehle: Bitoperatoren AND, OR, XOR, NOT sowie diverse Shift-Operatoren.

4. Verarbeitung von Zeichenketten (Strings): Einige Befehle zur Bearbeitung von Zeichenketten. Die Befehle verwenden die beiden Register SI und DI zur vereinfachten Adressierung aufeinander folgender Zeichen (Bytes).

5. Sprungbefehle: Sprünge innerhalb eines Moduls, entweder unbedingt (JMP) oder nur falls eine bestimmte Bedingung erfüllt ist (z. B. JNZ jump not zero). Weiterhin gehören zu dieser Gruppe Befehle zum Aufruf eines Unterprogramms (CALL) und zur Rückkehr (RET).

6. Prozessorkontrolle: Befehle zum Setzen und Löschen von Bits im Statusregister (STC set carry flag, CLC clear carry flag) und allgemeine Anweisung wie HLT zum Anhalten des Prozessors.

9.4.1 Addition

Der typische Vertreter der arithmetischen Befehle ist der Befehl ADD zur Addition. Der Befehl enthält zwei Operanden und das Ergebnis überschreibt den ersten Operanden. Für die Operanden sind folgende Kombinationen möglich:

Register	Speicher
Speicher	Register
Register	Register
Speicher	Konstante
Register	Konstante

Einer der Operanden kann eine Speicherzelle sein. Speicherzellen können in der Form [*Adresse*] angegeben werden. Die Anweisung

```
add [107h],2    ; Addiere 2 zu Speicherzelle 107h
```

addiert den Wert 2 zu dem Inhalt der Speicherzelle mit der Adresse 107_{16}. Jedoch ist es bei dem 8086 nicht möglich, zwei Speicherzellen direkt zu addieren. Vielmehr muss zunächst einer der Werte in ein Register geladen werden. Die Register können sowohl in voller als auch halber Breite benutzt werden. Dabei müssen allerdings die Operanden gleiche Größe haben. Dementsprechend wird die Addition mit 8 oder 16 Bit ausgeführt.

Beispiel 9.1 *Beispiele für Addition:*

```
add ax,-1       ; Register und Konstante
add ah,bl       ; zwei Halbregister
;  add al,bx  Halb- und Ganzregister geht nicht!
```

Mit der Ausführung werden im Statusregister die betroffenen Bits gesetzt oder gelöscht. Im wesentlichen werden eingetragen:

Carry Flag	Übertrag
Overflow Flag	Überlauf
Zero Flag	Ergebnis gleich 0
Sign Flag	Vorzeichen des Resultats

Übung 9.1 *Addition von Halbregistern: Welcher Wert wird nach folgendem Programm im Register AX stehen?*

```
mov ax,0103h
add ah,al
```

9.4.2 Multiplikation und Division

Die Befehle für vorzeichenlose Multiplikation und Division sind MUL und DIV. Die Varianten IMUL und IDIV berücksichtigen die Vorzeichen. In Abschnitt 8.2.1 hatten wir gesehen, dass für die Multiplikation im allgemeinen zusätzlicher Platz für das Ergebnis benötigt wird. Gleichzeitig sind Multiplikation und Division besonders aufwändige Operationen. Daher nehmen die Befehle eine gewisse Sonderstellung ein.

Anders als etwa bei ADD wird ein Operand stets aus dem Akkumulator genommen. Als zweiter Operand ist ein Register oder eine Speicherzelle möglich. Konstanten können nicht direkt verwendet werden. Der genaue Ablauf hängt von der Größe des zweiten Operanden ab. Handelt es sich um einen 8 Bit Wert, so wird er mit AL multipliziert und das Ergebnis in AX geschrieben. Sowohl AH und AL sind daran beteiligt. Selbst wenn das Resultat in AL passt, wird AH trotzdem gelöscht.

Beispiel 9.2 *Multiplikation mit 8 Bit Wert*
Nach

```
mov al,3
mov bl,2
mul bl          ; AX = AL * BL
```

enthält AX den Wert 6*, ein eventueller Wert in AH geht verloren.*

Bei einem 16 Bit großen Operanden wird mit dem gesamten AX multipliziert. Der obere Teil das 32 Bit Resultats wird in DX abgelegt.

Beispiel 9.3 *Multiplikation mit 16 Bit Wert*

```
mov ax,4000
mul ax          ; AX quadrieren
```

Ergibt in AX 2400_{16} *und in DX* $00F4_{16}$*, insgesamt also* $00F42400_{16}$ *entsprechend* 16000000_{10}

Die Division verläuft umgekehrt. Bei einem 8 Bit Operanden wird AX durch diesen Wert dividiert. Das Resultat kommt in AL. Zusätzlich wird der verbleibende Rest (AX % Operand) in AH geschrieben.

Beispiel 9.4 *Division durch 8 Bit Wert*
Nach

```
mov ax,17
mov bl,5
div bl
```

führt zu AL=3 (17/5) und AH=2 (17%5).

Die Division bei 16 Bit Operanden verwendet wiederum AX und DX als Ausgangswert und berechnet DX:AX/Operand. Das Ergebnis steht anschließend in AX, der Rest in DX. Falls das Ergebnis der Division zu groß für das Zielregister AX ist, so wird dies als Fehler signalisiert. Den dabei benutzen Interrupt-Mechanismus werden wir in Abschnitt 9.8 kennen lernen.

9.4.3 Sprungbefehle

Häufig ist es notwendig, vom linearen Programmablauf abzuweichen. So sind Programmabschnitte mehrfach zu durchlaufen oder in Abhängigkeit von Zwischenergebnissen soll zu unterschiedlichen Fortsetzungen verzweigt werden. In modernen Hochsprachen stehen dafür entsprechende übersichtliche Konstruktionen wie if ... then ... else oder while ... zur Verfügung. Damit sind Sprungbefehle nicht mehr notwendig und sollten - sofern die Sprachen sie überhaupt noch anbieten - vermieden werden.

Anders ist die Situation auf Assembler-Ebene. Der Prozessor kennt im wesentlichen nur Sprungbefehle. Die Umsetzung der Konstruktionen der Hochsprache in Sprünge übernimmt ein Compiler. Bei der Programmierung in Assembler ist der Programmierer gefordert, die Ablaufsteuerung durch entsprechende Sprungbefehle selbst zu realisieren.

Der 8086 verfügt über mehrere Sprungbefehle sowie einen speziellen Befehl zur einfachen Implementierung von Zählschleifen. Mit dem absoluten Sprungbefehl JMP wird die Ausführung an der angegebenen Adresse fortgesetzt. Die Adressen können als Absolutwerte oder – gekennzeichnet durch ein `$` vor der Zahl – relativ zur momentanen Adresse angegeben werden. Um die Eingabe der Adressen zu vereinfachen, unterstützt der Assembler Sprungmarken (Label). Eine solche Marke wird in der Form *Name:* vor einer Anweisung eingetragen. Bei JMP *Name* ersetzt der Assembler den Namen durch die Adresse.

Beispiel 9.5 *Sprungbefehl*

```
l1: mov cx,10     ; Sprungmarke l1
    inc ax
again:
    add ax,cx     ; Sprungmarke again
    ...           ; weiterer Code
    jmp again
    ...
    jmp l1
    ...
    jmp $-10      ; um zehn Adressen zurück springen
```

Bei bedingten Sprüngen wird die entsprechende Bedingung geprüft, und nur wenn sie erfüllt ist, erfolgt der Sprung. Ansonsten folgt die nächste Anweisung in der linearen Abfolge. Die Bedingungsprüfung beruht auf dem Statusregister. Bei der Ausführung des Sprungbefehls wird der Wert des entsprechenden Flags geprüft. So erfolgt der Sprung JZ (jump if zero), wenn das Zero Flag gesetzt ist. Das bedeutet aber, dass zuvor das Statusregister entsprechend gesetzt werden musste. Daher muss unmittelbar vorher eine entsprechende Operation ausgeführt werden. Die Abfolge ist demnach

Operation → Statusregister wird gesetzt → Bedingung prüfen → Sprung

In den meisten Fällen ist ohnehin ein Befehl notwendig, dessen Auswirkung dann getestet werden kann. Ansonsten kann der Befehl CMP verwendet werden. Er vergleicht zwei Operanden indem er die Differenz bildet. Das Statusregister wird entsprechend gesetzt, aber das Resultat nirgends abgelegt. Zwischen dem relevanten Befehl und dem Sprung können „neutrale“ Befehle, die die entsprechenden Statusbits nicht beeinflussen, eingefügt werden.

Die bedingten Sprünge testen auf die verschiedensten Bedingungen und berücksichtigen dabei teilweise auch mehr als nur ein Statusbit. So erfolgt JGE (jump if greater or equal) wenn das Zero Flag gesetzt ist oder das Sign Flag auf positiv steht. Von den Befehle ist auch die negierte Form JNxxx wie z. B. JNZ (jump if not zero) verfügbar. Das folgende Beispiel zeigt die Verwendung eines

bedingten Sprung zur Programmierung einer Schleife, in der Zahlen aufaddiert werden. Das Register BX übernimmt die Rolle des Zählers.

Beispiel 9.6 *Addition der Zahlen* $10, 9, \ldots, 1$

```
    mov bx,10     ; Zähler nach BX
    mov ax,0      ; vorsorglich AX löschen
l:  add ax,bx     ; Aufsummieren
    dec bx        ; Herunterzählen
    jnz l         ; noch nicht 0 ?
```

Solange der Befehl `DEC BX` noch nicht 0 ergibt, wird durch den Sprung die nächste Addition ausgeführt. Am Ende steht die Summe 55_{10} bzw. 37_{16} im Akkumulator AX. Da diese Art von Schleifen häufig vorkommt, gibt es dafür einen eigenen Befehl LOOP. Dabei wird das Register CX (Count register) eingesetzt. Zu Beginn wird die gewünschte Anzahl von Wiederholung in CX geschrieben. Dann führt der LOOP-Befehle folgende Aktionen aus:

- Er dekrementiert CX: CX = CX - 1
- Falls CX ungleich 0 ist, springt er an die angegebene Stelle.

Aus dem Beispiel wird mit dieser Vereinfachung:

Beispiel 9.7 *Addition der Zahlen* $10, 9, \ldots, 1$ *mittels LOOP Befehl*

```
    mov cx,10     ; Zähler muss nach CX
    mov ax,0      ; vorsorglich AX löschen
l1: add ax,cx     ; Aufsummieren
    loop l1
```

Ähnlich wie bei dem Befehl MUL bedingt die Vereinfachung eine Festlegung. Der Zähler steht immer in CX und in jeder Iteration wird er um den Wert 1 erniedrigt. Damit lassen sich ein Großteil der vorkommenden Schleifen effizient realisieren. Lediglich in Spezialfällen – z. B. mit einem Inkrement ungleich 1 – ist eine aufwändigere Lösung erforderlich.

Übung 9.2 *Verkehrsampel*
Mit dem Befehl OUT Port*,AX wird der Wert in AX in den I/O-Port mit der angegebenen Nummer geschrieben. Damit können externe Geräte (Device) angesprochen werden. Unter der Portnummer 4 wird im Simulator eine Steuerung für 4 Verkehrsampeln angesprochen. Über die einzelnen Bits können die Farben geschaltet werden (0 aus, 1 ein). Es gilt:*

- *Bit 0,1,2 für Grün, Gelb, Rot bei Ampel 1*

- *Bit 3,4,5 für Grün, Gelb, Rot bei Ampel 2*
- *Bit 6,7,8 für Grün, Gelb, Rot bei Ampel 3*
- *Bit 9,10,11 für Grün, Gelb, Rot bei Ampel 4*

1. *Schreiben Sie ein Programm, das alle Kombinationen durchläuft.*
2. *In einer zweiten Variante sollen alle Lampen einzeln getestet werden. (Hinweis: der Befehl SHL Register,Konstante schiebt den Inhalt eines Registers um die angegebene Anzahl von Stellen nach links).*
3. *Implementieren Sie einen realistischen Ablauf, bei dem z. B. die Ampeln 1 und 3 sowie 2 und 4 jeweils den gleichen Zustand haben.*

9.5 Speicheradressierung

9.5.1 Adressbereich

Der 8086 verwendet eine Wortbreite von 16 Bit. Damit lassen sich $2^{16} = 65536 = 64k$ Adressen ansprechen. Um den Bereich zu vergrößern, werden die Adressen aus zwei 16 Bit Werten gebildet. Dazu werden die Segmentregister CS, DS, ES und SS verwendet. Eine vollständige Adresse besteht dann aus der Segmentadresse und dem so genannten Offset. Mit den vier Segmentregister sind vier einzelne Bereiche vorgegeben. Im einzelnen sind dies wie die Namen der Register anzeigen

- Programmcode (CS)
- Daten (DS)
- Stack (SS)
- Extra (2.Datensegment) (ES)

Für die tatsächliche Lage dieser Segmente gibt es keine zwingenden Vorgaben. Sie können überlappen oder komplett getrennt voneinander liegen. Bei Programmen vom Typ COM liegen alle vier Segmente übereinander.

Intern wird bei jedem Speicherzugriff das zueinander passende Paar von Segmentregister und Adressregister verknüpft. Die Adresse eines Befehls wird beispielsweise aus den Werten in den Registern CS und IP gebildet. Im Laufe der Zeit wurden zwei verschiedene Strategien zur Berechnung verwendet. Zunächst wurde unter dem Betriebssystem DOS der Real-Mode eingesetzt. Das modernere Verfahren ist der Protected-Mode.

Im Real-Mode wird aus den beiden 16 Bit Werten eine 20 Bit große Adresse berechnet, so dass sich insgesamt ein Adressbereich von $2^{20} = 1M$ ergibt. Die Vorschrift dazu ist einfach:

- Segmentadresse: Segmentregister wird mit 16 (10_{16}) multipliziert. Dies entspricht einer Verschiebung um 4 Stellen nach links.
- Gesamtadresse: Segmentadresse + Offset

Man schreibt dies in der Form *Segmentregister:Adressregister*. Die Adressberechnung läuft automatisch ab. Beispielsweise wird beim Laden des nächsten Befehls der Wert in IP mit dem Wert in CS verknüpft.

Beispiel 9.8 *Adressberechnung in Real-Mode*
Sei CS=2F53h und IP=2E95h. Dann berechnet sich für CS:IP oder 2F53:2E95 die Adresse wie folgt:

- *Segmentadresse:* $2F53h * 10h = 2F530h$
- *Gesamtadresse:* $2F530h + 2E95h = 323C5h$

Als tatsächliche Adresse wird der Wert CS:IP=$323C5h$ *genommen.*

Solange man innerhalb eines Segments bleibt, braucht man lediglich zu Beginn des Programms einmal die Segmentregister zu laden. Das Programm verhält sich dann so, als ob nur 64k Speicher vorhanden wären. Mit diesem Konzept können auch leicht mehrere Programme gleichzeitig im Speicher gehalten werden. Das System muss dann lediglich die jeweiligen Segmentadressen verwalten.

Die Segmentgrenzen lassen sich in Schritten von 16 verschieben. Dies erlaubt bei knappem Speicherplatz eine sehr feine Aufteilung praktisch ohne Lücken. Allerdings leidet die Transparenz, da die Zuordnung nicht eindeutig ist. Mehrere Kombinationen von Segmentadresse und Offset führen zu einer gegebenen 20 Bit Adresse. So könnte in unserem Beispiel die Adresse $323C5h$ auch durch $2F54$:$2E85h$ angesprochen werden.

Im Protected-Mode (ab 80286 verfügbar) werden die Werte in den Segmentregistern nicht mehr direkt zur Adressgenerierung herangezogen, sondern stellen Selektoren dar. Ein solcher Selektor bezeichnet einen Index in einer Deskriptortabelle. Der Deskriptor wiederum enthält dann u.a. die Basisadresse und die Größe eines Segmentes. In dem Deskriptor sind außerdem Zugriffsrechte festgelegt. Damit ist es als Beispiel möglich, das Programmsegment als „read-only“ zu markieren und damit gegen unbeabsichtigte Veränderungen zu schützen.

Übung 9.3 *Adressberechnung in Real-Mode*
Geben Sie mindestens zwei verschiedene Kombinationen von Segmentadresse und Offset an, über die im Real-Mode die Adresse $4a36fh$ *erreicht werden kann.*

9.5.2 Adressierungsmöglichkeiten

Wesentlich für die Programmierung sind die Möglichkeiten, Speicherzellen anzusprechen. In diesem Abschnitt werden die verschiedenen Arten der Adressierung des 8086 vorgestellt.

Registeradressierung

Besonders einfach ist die Adressierung, wenn die zu verarbeitenden Daten in Registern stehen. Dann wird im Befehl direkt der Name der Register angegeben. Je nach Bedarf kann ein volles Register mit 16 Bit oder ein Halbregister mit 8 Bit spezifiziert werden. Die Operation erfolgt dann in der entsprechenden Bitbreite. Man spricht dann von Wort- und Byteoperationen.

Beispiel 9.9 *Registeradressierung*

```
mov bx,ax    ; Wortoperation
mov al,bh    ; Byteoperation
inc cx       ; Wortoperation
```

Unmittelbare Adressierung

Bei Befehlen mit zwei Operanden kann einer davon eine Konstante sein. Der Wert ist dann im Befehl enthalten und steht im Speicher nach dem Maschinencode für den Befehl.

Beispiel 9.10 *Unmittelbare Adressierung*

```
mov bx,655
add al,'a'
```

Zwei Anmerkungen:

1. Eine Konstante kann kein Ziel sein.
2. Eine Konstante ist nicht gegen Veränderungen geschützt. Es ist durchaus möglich, die entsprechende Speicherzelle zu verändern.

Direkte Adressierung

Die Adresse im Speicher kann direkt als Konstante angegeben werden. Zur Unterscheidung werden die Adressen in eckige Klammern gesetzt. Die Zählung beruht auf Bytes. Bei einer Wortoperation sind dann die beiden aufeinander folgenden Zellen betroffen.

Beispiel 9.11 *Direkte Adressierung*

```
mov bx,[2000]  ; Inhalt von 2000 und 2001 nach BX
add [2efh],bl  ; Inhalt von BL nach 2EFh
```

Indirekte Adressierung

Häufig ist es erforderlich, die Adressen variabel zu halten. Bei der Verarbeitung von Feldern oder Zeichenketten ist es beispielsweise erforderlich, ausgehend von einer Anfangsadresse eine ganze Reihe von Zellen nacheinander zu bearbeiten. Für derartige Anwendung besteht die Möglichkeit der indirekten Adressierung. Hierbei befindet sich die Adresse, die verwendet werden soll, selbst in einem Register. Im Befehl schreibt man den Namen des Registers in eckigen Klammern. Die Anweisungen

```
    mov bx,200
    add ax,[bx]  ; AX + Inhalt von 200
```

verwenden BX zur indirekten Adressierung. Zunächst wird mit einem MOV-Befehl die Adresse nach BX kopiert. Bei der Addition wird dann nicht etwa der Inhalt von BX (in diesem Fall 200) zu AX addiert, sondern BX wird als Verweis oder Zeiger verwendet. Dieser Zeiger deutet auf die Speicherzelle 200. Von dort wird der zweite Wert für die Addition geholt. Das Beispiel entspricht demnach der Anweisung

```
    add al,[200]  ; AL + Inhalt von 200
```

Der Vorteil der indirekten Adressierung zeigt sich erst, wenn mit der Adresse selbst gerechnet werden soll. Die folgende Erweiterung des obigen Codes bildet die Summe aus den Werten in 10 aufeinander folgenden Adressen:

```
    mov bx,200
    mov cx,10
l:
    add al,[bx]  ; AL + Inhalt von 200,201,...
    inc bx
    loop l
```

Hier wird zunächst die Anfangsadresse geladen. Nach jeder Addition wird die Adresse erhöht. Veränderungen an BX bewirken, dass der Zeiger anschließend auf eine andere Speicherzelle zeigt.

Der Mechanismus der indirekten Adressierung über Index oder Zeiger wird immer dann verwendet, wenn ein flexibler Zugriff erforderlich ist. Ein Anwendungsfeld ist die Verarbeitung von Feldern von Speicherzellen sei es als Zeichenketten oder als Vektoren oder Matrizen von Werten. Auch die Übergabe größerer Objekte an Funktionen erfolgt über einen Verweis auf die Speicherposition.

Implizit verwenden auch höhere Programmiersprachen diesen Mechanismus. Die Zeiger in C sind eine direkte Umsetzung. Die Sprache C erlaubt es, Zeiger auf Speicherstellen zu setzen und mit den Adressen zu rechnen (Zeigerarithmetik). Damit kann sehr effizienter Code geschrieben werden.

Allerdings ist der direkte Einsatz von Zeigern sehr fehlerträchtig. Ein Fehler in der Adressberechnung kann dazu führen, dass vollkommen falsche Zellen benutzt und eventuell geändert werden. Derartige Fehler führen zu schwer nachvollziehbaren Seiteneffekten. Aus diesem Grund ist bei vielen Sprachen keine direkte Manipulation von Zeigern möglich. So verzichteten die Designer der Sprache Java darauf, die Zeiger aus C zu übernehmen. Intern finden allerdings viele Zugriffe über Verweise (Referenzen) statt. Die Adressberechnung wird vom Compiler eingefügt. Es ist aber trotzdem für den Anwender wichtig, den Unterschied zwischen einem direkten Zugriff auf ein Objekt und dem indirekten Zugriff über eine Referenz zu kennen.

Die indirekte Adressierung bedingt einen zusätzlichen Aufwand bei der Ausführung. Zunächst muss der Inhalt des Indexregisters geholt werden und dann erfolgt der Zugriff auf die Speicherzelle mit dieser Adresse. Daher ist im 8086 diese Möglichkeit auf die vier Register BX, SI, DI und BP beschränkt.

Beispiel 9.12 *Indirekte Adressierung*

```
mov bx,[si]
add [bx],bl
inc [di]
```

Erweiterte indirekte Adressierung

Die Zeiger der indirekte Adressierung können beim Einsatz noch auf mehrere Arten erweitert werden. Zunächst ist es möglich, eine konstante Verschiebung (Displacement) zu verwenden. Dieser Wert `D` wird in der Form `[Reg]+D` angegeben. Die gleichwertigen Formen `[Reg+D]`, `[D+Reg]` und `D+[Reg]` sind ebenfalls erlaubt. Bei der Befehlsausführung werden die Adresse im Register und die Verschiebung addiert. Die Summe bezeichnet dann die Zelle, die tatsächlich verwendet werden soll. Je nach Anwendung kann man die Gesamtadresse in zwei Arten interpretieren:

1. variable Basisadresse plus einer konstanten Verschiebung
2. konstante Basisadresse plus einer variablen Verschiebung

Beispiel 9.13 *Indirekte Adressierung mit Displacement*
Kopieren eines Speicherinhaltes in zweite Zelle mit dem Abstand 10:

```
mov si,200
mov bx,[si]       ; hole Wert von Adresse 200 nach BX
mov [si]+9,bx     ; Kopiere Wert aus BX nach 209
```

Weiterhin kann die Adresse als Summe über zwei Register gebildet werden. Damit können sowohl Basis als auch Verschiebung flexibel gehalten werden.

Beispiel 9.14 *Indirekte Adressierung mit zwei Registern*
Kopieren eines Feldes mit 10 Bytes:

```
    mov bx,130      ; Adresse von Quelle
    mov bp,140      ; Adresse von Ziel
    mov si,10       ; Anzahl der Bytes
loop:
    mov al,[bx+si]  ; ein Byte holen
    mov [bp+si],al  ; und wieder schreiben
    dec si          ; herunter zählen
    jnz loop
```

Schließlich ist es noch möglich, zwei Register plus einer konstanten Verschiebung zu verwenden.

Beispiel 9.15 *Indirekte Adressierung mit zwei Registern und Displacement*

```
mov cl,[bx+si]+5   ; Adresse: Inhalt von BX + Inhalt von SI + 5
mov cl,[bx+si+5]   ; kann man auch so schreiben
mov cl,[bx][si]+5  ; auch erlaubt
                   ; weitere Möglichkeiten siehe Handbücher
```

Allerdings sind dabei nicht alle Kombinationen der vier Register erlaubt. Die möglichen Kombinationen zeigt folgende Darstellung:

BX	SI	Displacement
BP	DI	

Die Regel dazu ist, dass aus jeder Spalte ein Wert gewählt werden kann. Also ist beispielsweise BP+SI+Displacement möglich. Andere Kombinationen wie BX+BP werden vom Assembler als Fehler abgelehnt.

Übung 9.4 *Speicheradressierung*
Ab der Adresse B8000 liegt der Speicher für das CGA-Display (Color Graphics Adapter). Für jedes Zeichen des Bildschirms sind zwei Bytes reserviert: Der ASCII-Code des Zeichens und die Festlegung der Farben.

1. *Schreiben Sie ein Programm, das alle ASCII-Zeichen ausgibt.*
2. *Geben Sie mehrere Zeichen nebeneinander aus.*
3. *Variieren Sie die Farbinformation.*

9.6 Variablen

Im Assembler ist es möglich, Speicherplatz zu reservieren, mit einem Startwert zu füllen und mit einem symbolischen Namen zu versehen. Es stehen dazu zwei Befehle zur Verfügung:

1. DB (Define Byte): Reserviert Bytes
2. DW (Define Word): Reserviert Wörter (je 16 Bit)

Die Syntax ist

name DB|DW werte

Über den Namen kann später auf den Speicherplatz zugegriffen werden. Ein einfaches Beispiel dazu:

Beispiel 9.16 *Variablen*

```
    mov al,x        ; lade Inhalt von x nach AL
    ret
x   db    2         ; Variable x, 1 Byte groß, Anfangswert 2
```

Mit dem Befehl DB wird Speicherplatz reserviert. Der Speicherplatz schließt sich direkt an den letzten Befehl an. In dem Beispiel wurde er hinter die letzte Anweisung RET gesetzt. Dies ist zwar sinnvoll, aber keineswegs zwingend notwendig. Variablen und Programm können beliebig gemischt werden. Syntaktisch ist auch die Folge

```
    mov al,x        ; lade Inhalt von x nach AL
x   db    2         ; Variable mitten im Programm
    ret
```

korrekt. Der Assembler wird einfach hinter der MOV-Anweisung ein Byte mit dem Wert 2 einfügen. Bei der Programmausführung würde dies als Befehl interpretiert werden. Die CPU würde die 2 als Maschinencode lesen, eventuell weitere Operanden holen und den Befehl ausführen. Es liegt in der Verantwortung des Anwenders dafür zu sorgen, dass Programm und Daten klar getrennt bleiben.

Über den Namen x kann im Programm die Speicherzelle angesprochen werden. Der Assembler verwaltet die Variablennamen und ersetzt sie im Code durch die tatsächlichen Adressen. Die Variablen erleichtern die Programmierung. Aber im späteren ausführbaren Code sind sie verschwunden.

Beispiel 9.17 *Variablendefinitionen*

```
feld  dw 2,3,4,5     ; Ein Feld mit 4 Werten
text  db 'Hallo',0   ; 5 Zeichen plus abschließende Null
      db 6fh         ; anonyme Variablen sind auch möglich
f5    db 4 dup(5)    ; mit n dup(x) werden n Kopien von x angelegt
ex    db 4+5*7       ; Ausdrücke werden vom Assembler berechnet
```

Letztlich sind die Variablen nur eine symbolische Darstellung einer Speicheradresse. Es ist keinerlei Typ- oder Größeninformation mit ihnen verbunden. Daher können sie genauso wie konstante Adressen verwendet werden. Insbesondere können sie mit den Indexregistern für indirekte Adressierung kombiniert werden.

Beispiel 9.18 *Verwendung von Variablen*

```
inc x+5         ; Inhalt von (Adresse von x) + 5 erhöhen
mov al,x[di]    ; Inhalt von (Adresse von x) + (Inhalt von DI) laden
mov al,[di]+x   ; das selbe in anderer Schreibweise
```

9.7 Unterprogramme

Wesentlich für die Programmierung größerer Projekte ist die Aufteilung in kleinere, überschaubare Einheiten. Sinnvollerweise ist eine solche Einheit für eine konkrete, in sich abgeschlossen Aufgabe zuständig. Sie erhält Werte, führt die entsprechende Aufgabe aus und gibt das Resultat zurück.

Je nach Programmiersprache heißen die entsprechenden Einheiten Funktionen, Prozeduren oder Methoden. Allgemein kann man von Unterprogrammen sprechen. Der Ablauf ist folgender:

- Das Hauptprogramm unterbricht die weitere Ausführung.
- Es ruft das Unterprogramm auf und spezifiziert die Eingangswerte (Argumente, Parameter).
- Das Unterprogramm führt seine Aufgabe aus.
- Das Unterprogramm meldet das Ergebnis (Rückgabewert) und gibt die Kontrolle an das Hauptprogramm zurück.
- Das Hauptprogramm setzt die weitere Ausführung fort.

Ein Unterprogramm selbst kann weitere Unterprogramme aufrufen, eventuell sogar eine weitere Instanz von sich selbst (Rekursion). Bei der Umsetzung dieses Konzeptes stellen sich die Fragen:

- Wie übergibt das Hauptprogramm die Kontrolle an das Unterprogramm?

- Wie kann nach Ende des Unterprogramms das Hauptprogramm fortgesetzt werden?
- Wie erfolgt die Übergabe von Werten?

Der Aufruf eines Unterprogramms bedeutet, dass die Ausführung mit der entsprechenden Anweisung weiter geführt wird. Intern muss dazu der Programmzähler auf die Anfangsadresse gesetzt werden. Dies könnte durch einen Sprungbefehl geschehen. Allerdings fehlt nach einem Sprung die Möglichkeit der Rückkehr. Irgendwo muss also noch vermerkt werden, wo es nach Ende des Unterprogramms weiter gehen soll. Hier kommt der Stack ins Spiel. Beim Aufruf eines Unterprogramms (Befehl CALL) wird die Adresse des nächsten Befehls im Hauptprogramm auf den Stack gelegt. Dann beginnt die Ausführung des Unterprogramms. Das Ende des Unterprogramms ist durch einen Befehl RET (Return) markiert. Mit dem Erreichen dieses Befehls wird die Adresse vom Stack geholt und die Ausführung dort fortgesetzt.

Welchen Vorteil bietet nun gerade der Stack zum Speichern der Adresse? Betrachten wir dazu den Ablauf bei einer Verkettung von Unterprogrammaufrufen.

Aktion	Stack
Hauptprogramm legt Rückkehradresse AHa auf Stack und ruft U1 auf	AHa
U1 legt Rückkehradresse AU1 auf Stack und ruft U2 auf	AU1 AHa
U2 legt Rückkehradresse AU2 auf Stack und ruft U3 auf	AU2 AU1 AHa
U3 endet und nimmt Rückkehradresse AU2 vom Stack	AU1 AHa
U2 endet und nimmt Rückkehradresse AU1 vom Stack	AHa
U1 endet und nimmt Rückkehradresse AHa vom Stack	

Bei jedem weiteren Aufruf wird die Rücksprungadresse über die bereits abgespeicherten gelegt. Am Ende eines Unterprogramms wird die zuletzt abgelegt Adresse wieder entnommen. Durch die Last In First Out Eigenschaft des Stacks werden die Adressen automatisch in der passenden Reihenfolge abgelegt und entnommen. Mit jedem weiteren Unterprogrammaufruf wächst der Stack um eine Adresse. Umgekehrt schrumpft er bei einem Rücksprung wieder um eine Adresse.

Damit ist im wesentlichen auch schon die Realisierung von Unterprogrammen im Assembler beschrieben. Unterprogramme selbst werden im Programmcode mit *Name* PROC eingeleitet und mit *Name* ENDP beendet. Der dabei angegebene Name wird dann beim Aufruf CALL Name verwendet.

Beispiel 9.19 *Unterprogramm*

```
    MOV    AL,7
    CALL   u1       ; Aufruf
    RET

u1  PROC            ; Beginn des Unterprogramms
    ADD    AL,3     ; AL=AL+3
    RET             ; Rücksprung
u1  ENDP            ; Ende des Unterprogramms
```

Es existiert kein Mechanismus, der die Übergabe von Werten zwischen Haupt- und Unterprogramm automatisch übernimmt. Vielmehr beruht die Übergabe auf „Verabredungen“. Eine Möglichkeit ist, Register für die Übergabe zu benutzen. So könnte die Festlegung für ein Unterprogramm sein:

- Argument 1 steht in AX, Argument 2 steht in BX
- Ergebnis steht in CX

Der folgende Code zeigt den Ablauf am Beispiel eines Unterprogramms zur Bestimmung des Maximums aus zwei Zahlen.

Beispiel 9.20 *Unterprogramm max*

```
    MOV    AX,34     ; Argumente laden
    MOV    BX,22
    CALL   max       ; Aufruf
                     ; Resultat steht in CX
    RET

max PROC            ; Beginn des Unterprogramms
;  max erwartet zwei Werte in AX und BX, bestimmt das Maximum
;  und speichert das Ergebnis in CX
    CMP AX,BX       ; Werte vergleichen
    JLE fall2       ; BX <= AX ?
    MOV CX,AX       ; max = AX
    RET
```

```
fall2:
    MOV CX,BX       ; max = BX
    RET             ; Rücksprung
max ENDP            ; Ende des Unterprogramms
```

Wenn ein Unterprogramm Register benutzt, werden die darin enthaltenen Werte überschrieben. Ein Hauptprogramm kann daher nicht ohne weiteres davon ausgehen, dass nach der Ausführung eines Unterprogramms alle Register noch ihre alten Inhalte enthalten. Wenn die ursprünglichen Werte anschließend weiter benötigt werden, so müssen sie zu Beginn des Unterprogramms geeignet gespeichert und am Ende wieder geladen werden. Dazu sind zwei Strategien möglich:

- Die Verantwortung liegt beim Hauptprogramm. Es kümmert sich darum, dass Register mit wichtigem Inhalt gerettet und nach Rücksprung wieder gesetzt werden.
- Die Verantwortung liegt beim Unterprogramm. Das Unterprogramm speichert alle Register, die es verändert wird, ab und lädt sie vor dem Rücksprung wieder.

Beide Vorgehensweise haben Vor- und Nachteile. Am einfachsten und übersichtlichsten ist es wohl, wenn das Unterprogramm alle Register, die es verändern wird, vorsorglich sichert. Allerdings entsteht dadurch ein häufig unnötiger Mehraufwand, indem Register kopiert werden, die vom Hauptprogramm gar nicht verwendet werden.

Als Zwischenspeicher bietet sich wieder der Stack an. Das Unterprogramm kann in diesem Fall zu Beginn mittels PUSH-Operationen die betroffenen Register retten. Am Ende werden sie über POP-Operationen wieder geladen. Der Vorteil des Stacks erweist sich bei verschachtelten, mehrfachen Aufrufen. Mehrere Kopien der Register liegen dann in unterschiedlicher Tiefe im Stack. Hätte man demgegenüber einen festen Bereich als Zwischenspeicher festgelegt, so würden die Werte bei jedem weiteren Aufruf überschrieben. Mit dem gleichen Argument sollte ein Unterprogramm benötigten Speicherplatz für temporäre Variablen ebenfalls in dem Stack anlegen.

9.8 Interrupt

Eine spezielle Art von Unterprogrammen stellen Interrupts (engl. Unterbrechungen) dar. Sie werden über einen eigenen Befehl INT *Nummer* aufgerufen. Der Wert von Nummer liegt zwischen 0 und 255, so dass 256 verschiedene Interrupts möglich sind. Zusätzlich werden Untergruppen von Interrupts durch den Wert in AH spezifiziert. Insgesamt ergeben sich damit $256 \cdot 256$ verschiedene Interrupts. Andere Register (vornehmlich AL) dienen zur Übergabe von Parametern und Resultaten. Interrupts stellen vor allem Schnittstelle zum System dar. Über

spezielle Interrupts können Geräte wie Bildschirm, Tastatur oder Festplatten angesprochen werden.

Beispiel 9.21 *Ausgabe der Zeichenfolge* FB *über Interrupt 10H/0E.*

```
MOV    AH, 0Eh    ; sub-function festlegen
                  ; AH wird durch Interrupt nicht verändert

MOV    AL, 'F'
INT    10h        ; Ausgabe

MOV    AL, 'B'
INT    10h        ; Ausgabe
```

Beispiel 9.22 *Interrupt zum Lesen von Tastatur.*

```
l:
mov ah,0
int 16h     ; Interrupt um 1 Zeichen von Tastatur zu lesen
mov bl,al   ; Ergebnis (Zeichen) steht in AL, nach BL kopieren
jmp l       ; Endlos-Schleife
```

Beispiel 9.23 *Einfügen von einer Wartezeit von einer Sekunde.*

```
; Interrupt 15 / AH=86h ist eine Wartefunktion. In den kombinierten
; Registern CX und DX wird die Anzahl von Mikrosekunden spezifiziert.
; 1 Sekunde entspricht 1 Million Mikrosekunden
; also nach CX:DX 1000000 bzw. F4240h
MOV     CX, 0Fh      ; oberer 16 Bit
MOV     DX, 4240h    ; untere 16 Bit
MOV     AH, 86h
INT     15h
```

Wo der Code zu den einzelnen Interrupts zu finden ist, ist in einer Tabelle (Interrupt table) festgelegt. Im Prinzip wird bei einem Aufruf `INT` *n* die n-te Zeile dieser Tabelle verwendet, um die Startadresse des Unterprogramms (Interrupt Service Routine) auszuwählen. Interrupts sind eine einfache und standardisierte Form von Unterprogrammen. Sie sind daher besonders geeignet, um Systemfunktionen zu implementieren. Diese Funktionen brauchen dann nur einmal in den Speicher geladen zu werden. Jedes Programm hat über den Interruptmechanismus Zugriff auf sie.

Neben den beschriebenen Aufruf durch Programme (Software-Interrupts) können Interrupts auch durch die Hardware ausgelöst werden (Hardware-Interrupts). Verschiedenste Ereignisse innerhalb oder außerhalb der CPU können Ursache für Hardware-Interrupts sein. Einige Möglichkeiten sind:

- Fehler bei arithmetischen Operationen (z. B. Divisionsergebnis passt nicht in Zielregister).
- Der Anwender betätigt eine Taste.
- Die Netzwerkkarte meldet den Empfang eines Paketes.
- Die Versorgungsspannung fällt aus.

Die meisten dieser Fälle können zu beliebigen Zeiten auftreten. Das gerade laufende Programm wird dann unterbrochen und über die Nummer des Interrupts die zugehörige Service Routine ausgeführt. Die Routine endet mit dem Befehl IRET und das angehaltene Programm läuft weiter.

Anders als ein normales Unterprogramm ist ein Interrupt nicht vorhersehbar. Beispielsweise könnte ein Interrupt unmittelbar vor einem bedingten Sprung erfolgen. Daher ist es notwendig, dass am Ende des Interrupts auch das Statusregister wieder auf den ursprünglichen Wert gesetzt wird. Dazu wird bei einer Interrupt Service Routine beim Aufruf zusätzlich das Statusregister auf den Stack gelegt und am Ende wieder von dort geholt.

Die Ursachen für einen Interrupt können von unterschiedlicher Dringlichkeit sein. Auf einen Netzausfall beispielsweise muss sehr viel schneller reagiert werden, als auf einem Tastendruck. Aus diesem Grund werden Interrupts mit einer Priorität versehen. Interrupts mit einer höheren Priorität unterbrechen solche mit niedrigerer Priorität. Die Behandlung der Interrupts übernimmt ein spezieller Interrupt-Controller. Er entlastet die CPU von der Behandlung der verschiedenen Interrupts (Interrupt Request IRQ). Weiterhin bietet er die Möglichkeit, einzelne Interrupts frei zu geben oder zu unterdrücken (maskieren). Die sehr wichtigen Interrupts sind allerdings davon ausgenommen. Sie können nicht per Programm unterdrückt werden (nicht maskierbare Interrupts).

Die Reaktionszeit auf einen Interrupt bestimmt die Echtzeitfähigkeit eines Systems. Systeme, bei denen Echtzeitfähigkeit eine wichtige Rolle spielt, sind dementsprechend für eine schnelle Bearbeitung von Interrupts ausgelegt. Für den Prozessor in einem Handy als Beispiel ist es wichtig, dass die Sprachdaten schnell genug abgeholt, bearbeitet und wieder ausgegeben werden.

9.9 Vom C-Programm zum Assemblercode

Nur in seltenen Fällen wird eine Anwendung direkt in der Assembler-Sprache geschrieben. Vielmehr wird in der Regel eine Hochsprache verwendet. Die Sprache

selbst ist idealerweise unabhängig von dem Zielsystem. Für die Umsetzung des Programmcodes in Assembler beziehungsweise Maschinencode ist der Compiler zuständig.

Im Normalfall läuft diese Umsetzung vollautomatisch ab. Über entsprechende Optionen kann der entstehende Assemblercode und Maschinencode ausgegeben werden. Betrachten wir als Beispiel das folgende, kurze C-Programm.

```
void ausgabe( short a, short b ) {
    printf( "%d %d", a, b );
}

short main(short argc, char* argv[])
{
    short a=2, b=5, i;

    for( i=0; i<10; i++ ) a += b;

    ausgabe( a, b );

    return 0;
}
```

Mit der Option `/FAs` erzeugt der Compiler von Microsoft Visual C++ eine Datei mit der Endung `.asm`, die sowohl den C-Code als auch die generierten Assemblerbefehle enthält. Als Konfiguration wurde *Debug* gewählt. Anders als in der Konfiguration *Release* wird der Code nicht allzu stark durch Optimierungen verändert. Der Teil `main` wird dann in folgenden Code für einen Pentium-Prozessor übersetzt[2]:

```
_TEXT   SEGMENT
_a$ = -4
_b$ = -8
_i$ = -12
_main   PROC NEAR     ; Beginn der Prozedur main

; 11   : {
    push    ebp       ; Register auf Stack retten
    mov ebp, esp
    sub esp, 76
    push    ebx
    push    esi
    push    edi
```

[2]Der Assemblercode ist etwas vereinfacht wiedergegeben. Der Zusatz `e` vor den Registernamen bezieht sich auf die auf 32 Bit erweiterten Register

```
...  ; ********* hier sind einige Befehle ausgelassen *********

; 12   :    short a=2, b=5, i;
    mov  _a$[ebp], 2
    mov  _b$[ebp], 5

; 14   :    for( i=0; i<10; i++ ) a += b;
    mov _i$[ebp], 0              ; Initialisierung i = 0
    jmp $L538
$L539:
    mov ax, _i$[ebp]             ; Hochzählen
    add ax, 1
    mov _i$[ebp], ax
$L538:
    movsx   ecx, _i$[ebp]
    cmp ecx, 10                 ; Abbruchbedingung
    jge  $L540
    mov dx, _a$[ebp]            ; Schleifenkern
    add dx, _b$[ebp]
    mov _a$[ebp], dx
    jmp  $L539
$L540:

; 16   :    ausgabe( a, b );
    mov ax, _b$[ebp]
    push    eax
    mov cx, _a$[ebp]
    push    ecx
    call    ?ausgabe@@YAXFF@Z           ; ausgabe
...  ; ********* hier sind einige Befehle ausgelassen *********
_main   ENDP
```

Im Assemblercode findet man die Struktur des C-Programms gut wieder. Zur Übergabe der beiden Argumente an die Funktion `ausgabe` wird der Stack verwendet. Die Zählschleife wird mit mehreren Sprüngen realisiert, wobei die Sprungmarken fortlaufend nummeriert werden. Die Umsetzung ist allerdings nicht optimal. Insbesondere das ständige Transferieren der Werte zwischen Hauptspeicher und Registern bedeutet einen vermeidbaren Mehraufwand.

9.10 Übungen

Übung 9.5 *Anweisungen*
Welche der folgenden Move-Anweisungen sind nicht erlaubt? Geben Sie jeweils

eine kurze Begründung.

```
mov ax,123
mov ax,[256]
mov ax,bh
mov 345,ax
mov ax,ax
mov ah,bl
```

Übung 9.6 *In manchen Assembler-Programmen findet man die Anweisung XOR für ein exklusives Oder mit identischen Operanden, also XOR AX,AX oder XOR DS,DS. Was bewirkt dieser Befehl? Welche Gründe könnten dafür sprechen, diesen „Trick“ einzusetzen?*

Übung 9.7 *Ampel*
Erweitern Sie unser Programm für die Ampelsteuerung zu einem realistischen Ablauf. Die Dauer der einzelnen Phasen soll durch Warteschleifen realisiert werden. Um den Ablauf flexibel zu gestalten, sollen diese Zeiten in Variablen gespeichert werden.

Übung 9.8 *Fibonacci-Zahlen*
Die Fibonacci[3]*-Zahlen sind durch die Startbedingung* $i_1 = 1$, $i_2 = 1$ *und die Rekursion* $i_n = i_{n-2} + i_{n-1}$ *definiert. Damit gilt*

$$\begin{aligned} i_3 &= i_1 + i_2 = 1 + 1 = 2 \\ i_4 &= i_2 + i_3 = 1 + 2 = 3 \\ i_5 &= i_3 + i_4 = 2 + 3 = 5 \\ i_6 &= i_4 + i_5 = 3 + 5 = 8 \end{aligned}$$

und so weiter. Schreiben Sie ein Assembler-Programm, das ausgehend von den Startwerten 1,1 weitere Fibonacci-Zahlen berechnet.

Übung 9.9 *Sortier-Algorithmus*
Implementieren Sie einen Algorithmus, um ein Feld mit Werten zu sortieren.

Übung 9.10 *Wurm*
Realisieren Sie einen „Wurm“, d. h. ein Programm, das sich selbst ständig im Speicher kopiert.

[3]Leonardo Pisano genannt Fibonacci, italienischer Mathematiker, ca. 1170-1250

Kapitel 10

Schnittstellen zum Betriebssystem

10.1 Schichtenmodell

Ausgehend von den Fähigkeiten eines Prozessors kann eine Software-Architektur entworfen werden. In einfachen Anwendungen wie etwa die Steuerung von Geräten, wird nur eine spezielle Anwendung in dem Prozessor ausgeführt. In diesem Fall ist es ausreichend, wenn eine Möglichkeit zum Laden und Starten dieser Anwendung besteht. Die entsprechende Funktionalität stellen so genannte Monitore zur Verfügung. Komplexer ist die Situation bei allgemeinen Computern, auf denen Benutzer interaktiv verschiedenste Anwendungen ausführen. Dann wird ein Betriebssystem zur Verwaltung der Ressourcen benötigt.

Die gesamte Architektur umfasst dann mehrere Ebenen von der Hardware bis zur Benutzerschnittstelle. Ein entsprechendes Schichtenmodell für das Betriebssystem DOS zeigt Tabelle 10.1. Die erste Software-Komponente direkt über der Hardware ist das BIOS (*Basic Input Output System*) [FKS03]. Das BIOS übernimmt die Kommunikation mit den verschiedenen Geräten wie Tastatur, Bildschirm, Laufwerke, und so weiter. Diese Kommunikation erfolgt mit den Befehlen IN und OUT über Ports oder auch direkt über bestimmte Speicherbereiche. Nach oben stellt das BIOS durch Interrupts Standardfunktionen bereit. Damit sind die Abhängigkeiten von den speziellen Geräten (z. B. Typ von Festplatte) weitgehend im BIOS gekapselt.

In früheren Versionen mussten Informationen über die Hardware zum Teil manuell in das BIOS eingetragen werden. Modernere Systeme verwenden das

Tabelle 10.1: Schichtenmodell für das Betriebssystem DOS

Ebene 1	COMMAND.COM, Anwenderprogramme
Ebene 2	DOS
Ebene 3	BIOS
Ebene 4	Hardware

Plug and Play Prinzip (PnP, etwa „Einstecken und funktioniert“), bei dem die Geräte automatisch erkannt werden.

Wenn ein Programm auf die BIOS-Funktionalität aufsetzt und nur über die BIOS-Interrupts Geräte anspricht, so ist die Portabilität gewährleistet. Der Anwender braucht sich nicht um Hardware-Details zu kümmern. Der Preis für diesen Komfort ist ein gewisser Mehraufwand bei der Ausführung. Wenn eine Anwendung direkt über Ports mit einem Gerät kommuniziert, entfällt der zusätzlich Aufwand durch den Interrupt-Aufruf. Ein weiterer Nachteil liegt in mitunter mangelnden Aktualität des BIOS. Gerade wegen der gewünschten Kompatibilität werden neueste Möglichkeiten oft nicht ausgeschöpft. Das Betriebssystem Linux verwendet aus diesen Gründen nicht die BIOS-Aufrufe, sondern ersetzt sie durch eigene Treiber.

Das eigentliche Betriebssystem ist DOS (*Disk Operating System*), sei es das System der Firma Microsoft (MS-DOS) oder eine der Varianten (DR-DOS von Digital-Research, IBM-DOS, FreeDOS). Seine Aufgaben umfassen:

- Verwaltung von Geräten und Dateien
- Verwaltung von Programmen (Prozessen)
- Verwaltung von Speicher

Es greift dabei auf die Systemaufrufe des BIOS zurück. Umgekehrt werden auch die DOS-Funktionalitäten über Interrupts angesprochen. Im wesentlichen handelt es sich dabei um den Interrupt 21h. Über den Wert im Register AH können mehr als 200 Funktionen über diesen einen Interrupt ausgeführt werden.

Das Betriebssystem ist für die Ausführung der Anwendungsprogramme zuständig. Eines davon ist COMMAND.COM – der Befehlsinterpreter. Über diese Schnittstelle kann der Benutzer Befehle eingeben. Typische Beispiele sind die Befehle

- TYPE: Anzeigen einer Datei
- COPY: Kopieren einer Datei
- MD: Anlegen eines Verzeichnisses
- FORMAT: Formatieren eines Datenträgers

Auch Anwenderprogramme werden aus der Befehlszeile von COMMAND.COM heraus gestartet. Die ersten graphischen Benutzeroberflächen der Firma Microsoft wie z. B. Windows 3.x waren DOS-Anwendungen. Erst die neueren Systeme Windows NT oder Windows 2000 sind eigene Betriebssysteme.

Die hierarchische Strukturierung in mehreren Schichten vereinfacht die Entwicklung von Applikationen. Wenn das Programm nur auf Systemaufrufe von

DOS basiert, so ist ein hohes Maß an Portabilität erreicht. Allerdings beeinträchtigt die Abwicklung über mehrere Hierarchien die Ausführungsgeschwindigkeit. Insbesondere Grafik-intensive Anwendungen wie CAD-Programme oder Spiele greifen daher oft direkt auf die Graphikkarte zu.

10.2 Booten

Bisher haben wir die Frage offen gelassen, wie überhaupt nach dem Einschalten ein Computer startet. Der Hauptspeicher ist üblicherweise aus Bausteinen aufgebaut, die ohne Stromversorgung ihren Inhalt verlieren. Nach dem Einschalten enthält der Hauptspeicher keine sinnvollen Daten. Gleichzeitig wird ein Stück Software benötigt, um das System in Gang zu setzen.

Aufgrund dieser paradoxen Ausgangssituation bezeichnet man den Vorgang als *bootstrapping* oder kurz *booten*. Der Name bezieht sich auf die Vorstellung, wie jemand ohne fremde Hilfe sich an den Stiefel hochzieht. Im Deutschen gebräuchlicher ist das Bild vom Baron Münchhausen, der sich am eigenen Schopf aus dem Sumpf zieht[1].

Allerdings geht es nicht ganz ohne Hilfe. Benötigt wird ein kleiner Speicherteil, der die Daten dauerhaft behält. Dies können spezielle Bausteine wie z. B. ROM-Speicher (*Read Only Memory*) sein. Eine andere Möglichkeit ist, einen entsprechenden Speicher über eine Batterie permanent mit Spannung zu versorgen.

In diesem Speicher befindet sich ein Startprogramm, der *boot loader* oder Urlader. Nach dem Einschalten erhält der Prozessor ein Reset Signal. Darauf setzt er den Programmzähler an eine fest definierte Stelle, an der sich der boot loader befindet. In der Regel enthält dieses Programm nur rudimentäre Funktionen, die aber ausreichen, um weitere Programme zu laden. Als Quelle für das vollständige Betriebssystem kommen Plattenlaufwerke oder auch das Netzwerk in Frage.

Bei IBM-kompatiblen PCs ist der Ablauf wie folgt:

- Das BIOS inklusive boot loader steht am oberen Ende des adressierbaren Speichers.
- Bei einem Reset wird die oberste Instruktion im Speicher bei FFFF0h ausgeführt. Dort steht ein Sprung an den eigentlichen Beginn des boot loader.
- Der boot loader führt einen Test der Hardware aus (*Power On Self Test*, POST) und initialisiert die Geräte.
- Er sucht gemäß einer Liste nach dem ersten bootfähigen Gerät (Laufwerk).

[1] *Aber auch beim zweiten Anlauf sprangen wir zu kurz und sanken, nicht weit vom anderen Ufer, bis an den Hals in den Morast! Und wir wären rettungslos umgekommen, wenn ich mich nicht, ohne mich lange zu besinnen, mit der eignen Hand am eignen Haarzopf aus dem Sumpf herausgezogen hätte!*

- Von dem Gerät wird der 512 Byte große *master boot record* (MBR) in den Hauptspeicher geladen.
- Dann wird die Ausführung mit dem geladen Programm weitergeführt.
- Diese Startprogramm lädt die weiteren Teile nach. Bei Laufwerken mit mehreren installierten Betriebssystemen entscheidet sich erst an dieser Stelle, welches davon geladen wird.

Bei einfache Systeme mit nur einem geringen Speicherbedarf kann auch das gesamte Betriebssystem in Bausteinen mit dauerhafter Speicherung abgelegt werden. Dann muss nur dafür gesorgt werden, dass bei einem Reset der Programmzähler auf eine geeignete Anfangsadresse gesetzt wird. Ein Beispiel aus der Anfangszeit der PCs ist der Commodore C64. Bei diesem Computer war das gesamte Betriebssystem in einem 8 kByte großen ROM-Baustein gespeichert. Ein zweiter Baustein mit 8 kByte enthielt einen Interpreter für die Sprache BASIC. Damit war das System ohne externe Laufwerke voll funktionsfähig.

Kapitel 11

Optimierung

11.1 Einführung

Die Architektur nach von Neumann hat sich als Grundlage für die Rechnerentwicklung bewährt. Das Prinzip des minimalen Hardware- und Speicher-Aufwands führt zu kostengünstigen Realisierungen. Mit der fortschreitenden Entwicklung der Halbleiter-Technologien wurde es allerdings möglich, von diesem Prinzip abzuweichen und die Architektur in Hinblick auf höhere Performanz zu erweitern. In diesem Kapitel werden die wichtigsten Ansätze vorgestellt. Ausgangspunkt ist die in Kapitel 3 beschriebene Grundstruktur. Wir hatten bei der Diskussion bereits die schwer wiegenden Defizite, die unmittelbar aus der Architektur resultieren, kennen gelernt. Die zwei wesentlichen Punkte sind:

- Der einheitliche Bus stellt einen Flaschenhals dar.
- Bei der Bearbeitung der Befehle wechseln sich Hol- und Ausführungsphasen ab. Dadurch sind in der meisten Zeit nur einige Komponenten aktiv.

In den folgenden Abschnitten werden verschiedene Ansätze zur Behebung dieser Schwachstellen vorgestellt.

11.2 Harvard-Architektur

Eine prinzipielle Alternativ besteht in der Auftrennung des Bussystems. In der Harvard-Architektur werden getrennte Speicherbereiche für Daten und Programme eingeführt. Jeder Speicherbereich ist über ein eigenes Bussystem mit der CPU verbunden. Das Konzept wird vornehmlich bei Digitalen Signalprozessoren (DSP) realisiert. Diese auf hohe Geschwindigkeit optimierten Prozessoren werden zur Verarbeitung von Audio- oder Videodaten eingesetzt. Dabei wird durch die Harvard-Architektur ein schneller Datendurchsatz erzielt. Mit einer Instruktion kann dann der nächste Befehl und gleichzeitig Daten geladen werden. Ein Beispiel für den Motorola DSP56300 aus dem Manual[Mot00] ist:

```
mac y0,x1,b     x:(r4)-,x1   y:(r0)+,y0
; Signed Multiply Accumulate b = b + y0 * x1
;               Lade aus Speicherbereich X, Adresse in Register R4
;               nach Register X1, anschließend noch R4 = R4 -1
;                           Aus Bereich Y, Adresse in R0 nach Y0
;                           R0 = R0 +1
```

Zeitgleich mit einer arithmetischen Operation werden neue Werte aus zwei verschiedenen Speicherbereichen geladen. Die Adressierung erfolgt indirekt über Register. Zusätzlich werden auch die Inhalte dieser Register aktualisiert. In der arithmetischen Operation werden noch die alten Werte in den Registern verwendet. Die neu geladenen Werte stehen dann für die nächste Instruktion bereit.

Einen Schritt weiter ging die Firma Analog Devices. Bei dem 32-Bit Signalprozessoren SHARC® wurde das Konzept noch um einen zusätzlichen Bus für die Ein- und Ausgabe erweitert. Dafür wurde die Bezeichnung *Super Harvard Architektur* eingeführt.

11.3 DMA

In der ursprünglichen von Neumann Architektur werden alle Datentransfers über die CPU abgewickelt. Das Prinzip zeigt Bild 11.1 am Beispiel eines Transfers von einem Gerät in den Speicher. Um einen Wert zu kopieren ist folgender Ablauf notwendig:

1. Wert von Quelle in CPU laden.
   (a) MOV-Befehl laden.
   (b) Wert laden.
2. Wert von CPU zu Ziel speichern.
   (a) MOV-Befehl laden.
   (b) Wert speichern.

Wenn man weiterhin bedenkt, dass auch der MOV-Befehl beispielsweise beim 8086 mehrere Bustransfers für Befehlscode und Adressen bedingt, wird klar wie aufwändig ein solcher Transfer wird. Andererseits kommen solche Transfers häufig vor. In vielen Fällen wird nicht nur ein einzelner Transfer benötigt, sondern ganze Blöcke sollen kopiert werden.

Eine wesentliche Verbesserung bietet in solchen Situationen der *Direct Memory Access* (DMA). Die Grundidee ist, die Aufgabe des Datentransfers an eine eigene Einheit (DMA-Controller) zu delegieren. Die CPU muss sich dann nicht mehr selbst um den Transfer kümmern, sondern beauftragt den (oder einen von mehreren) DMA-Controller. Sie gibt Quelle, Ziel und Umfang der Daten vor. Der

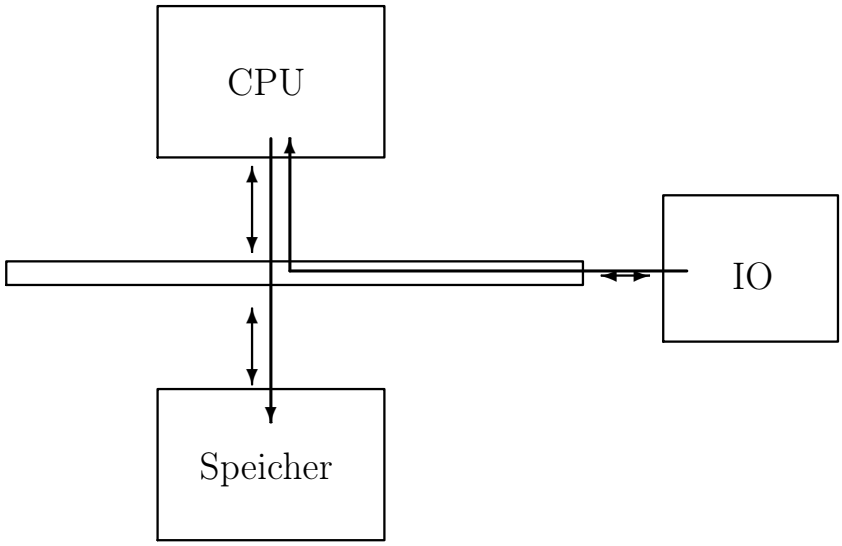


Abbildung 11.1: Ablauf beim Datentransfer

DMA-Controller führt dann den Transfer selbständig durch und informiert die CPU nach erfolgreichem Abschluss. Bild 11.2 illustriert dieses Konzept.

Während der DMA-Transfer erfolgt, kann die CPU andere Aufgaben erledigen. Dafür wird sie auch weiterhin selbst auf den Bus zugreifen. Durch eine entsprechende Steuerlogik wird dafür gesorgt, dass es dabei zu keinen Konflikten kommt. Die Kontrolle behält dabei die CPU. Sie gibt über ein spezielles Signal den Bus für den DMA frei. Dann wird entweder ein einzelner Wert oder ein ganzer Block (*Block mode*) übertragen. Anschließend erhält die CPU die Kontrolle zurück.

Dieser Vorgang wiederholt sich, bis alle angegebenen Daten transferiert sind. Die Zählung der Daten übernimmt der DMA-Controller. Nach Abschluss informiert er – beispielsweise per Interrupt – die CPU.

Aus Sicht des Anwenders ist der DMA relativ einfach. Man muss nur durch entsprechende Befehle an den DMA-Controller den Vorgang starten. Alles andere läuft dann automatisch im Hintergrund ab. Man muss lediglich beachten, dass der Zustand des Zielspeichers während des DMA-Vorgangs nicht definiert ist. Würde man während dieser Zeit eine Zelle aus diesem Bereich lesen, könnte man entweder bereits den neuen oder noch den alten Wert erhalten, je nach dem ob der DMA diese Zelle bereits erreicht hat oder nicht. Das Resultat hängt von den Details im Timing ab. Daher sollte sinnvollerweise die Anwendung während eines laufenden DMA-Transfers den Zielspeicher nicht verwenden.

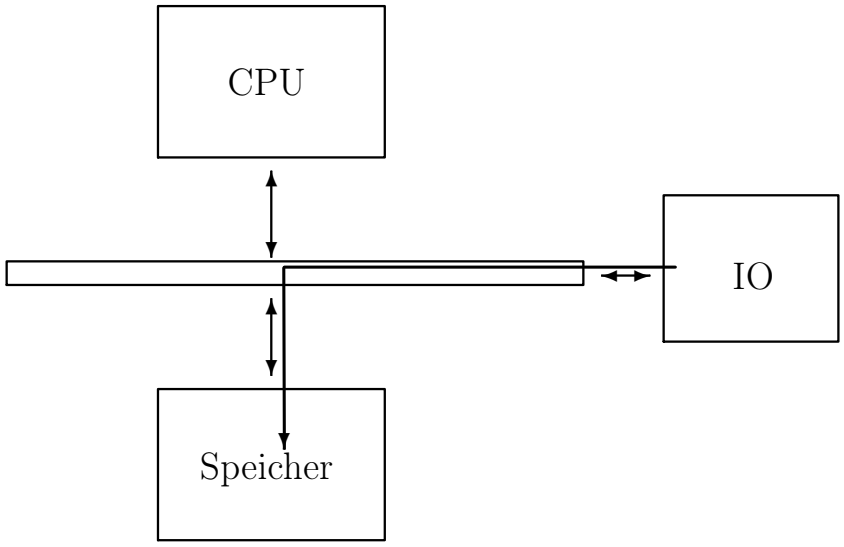


Abbildung 11.2: Ablauf beim Datentransfer mit DMA

11.4 Pipelining

In Kapitel 8 hatten wir gesehen, wie die Ausführung eines Befehls in mehreren Phasen abläuft. Dabei kann man unterscheiden zwischen Phasen mit und ohne Buszugriff. Nehmen wir als vereinfachtes Modell an, dass ein Befehl nur zwei Phasen umfasst:

1. Holen des Befehls aus dem Hauptspeicher.
2. Ausführen der Operation innerhalb der CPU.

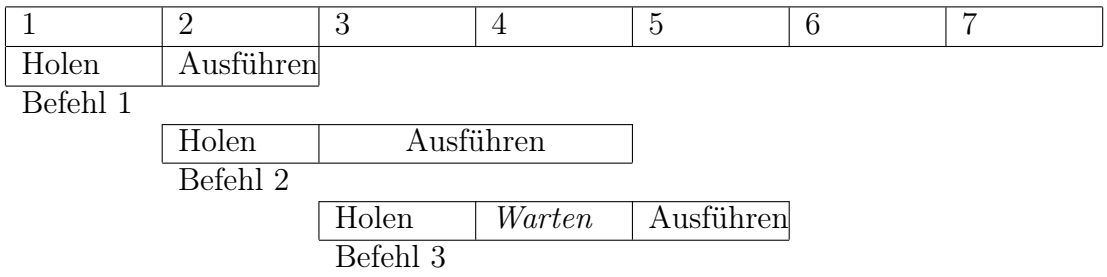
Dann ergibt sich folgendes Bild für die Abfolge von drei Befehlen:

1	2	3	4	5	6	7
Holen	Ausführen	Holen	Ausführen		Holen	Ausführen
Befehl 1		Befehl 2			Befehl 3	

Der Ablauf zeigt den Wechsel zwischen den beiden Phasen. In jeder Phase ist aber nur ein Teil des Systems aktiv, oder – negativ formuliert – der andere Teil ist untätig und wartet. Es bietet sich als Alternative an, die Befehle nicht streng sequentiell sondern überlappend zu bearbeiten. Man kann die Situation gut mit der Fertigung von z. B. Autos vergleichen. Die Arbeiten am nächsten Auto beginnen nicht erst, wenn ein Auto komplett fertig gestellt ist. Vielmehr wird der

Fertigungsprozess in viele einzelne Schritte unterteilt. Sobald ein Schritt abgeschlossen ist, wird das entsprechende Auto einen Schritt weiter gegeben und das nächste rückt nach. Zu jedem Zeitpunkt befinden sich mehrere Autos in der Fertigungskette. Sie durchlaufen den Prozess in Form eines Fließbandes.

Übertragen auf die Befehlsverarbeitung werden die Instruktionen ebenfalls durch ein Fließband oder einen Schlauch (engl. *Pipeline*) geschoben. Aus dem obigen Beispiel wird dann folgender Ablauf:



Während der erste Befehl noch intern ausgeführt wird, kann bereits der zweite Befehl aus dem Speicher geholt werden. Bei dem zweiten Befehl dauert die Ausführung länger als das Holen des dritten Befehls. Der dritte Befehl kann dementsprechend erst nach einem Wartetakt weiter geschoben werden.

Dieses Prinzip wird als *Pipelining* oder Fließband-Prinzip bezeichnet. In dem Beispiel verkürzt sich die Verarbeitungszeit für drei Befehle um 2 Einheiten. Demgegenüber steht ein gewisser Mehraufwand, um die parallele Verarbeitung der einzelnen Stufen zu ermöglichen. Insbesondere werden zusätzliche Register benötigt, um die in den einzelnen Stufen benötigten Werte getrennt zu halten.

Wie groß die Ersparnis tatsächlich wird, hängt von den einzelnen Verarbeitungszeiten und der Struktur der Befehle ab. Lassen sich mehr Verarbeitungsstufen separieren, ist ein größerer Gewinn durch die gleichzeitige Bearbeitung aufeinander folgender Befehle erzielen. Der größte Gewinn wird erzielt, wenn die Pipeline permanent gefüllt ist. Dann sind alle Einheiten gleichzeitig aktiv und der Durchsatz wird maximal. Am Ende jeder Pipeline-Phase ist ein Befehl fertig ausgeführt.

Dieser optimale Betrieb lässt sich nicht mehr aufrecht erhalten, sobald Abhängigkeiten zwischen den Befehlen zu berücksichtigen sind. Besonders problematisch sind Sprünge. Von welcher Adresse der nächste Befehle zu holen ist, steht erst nach Analyse des Sprungbefehls fest. Daher kann die nächste Instruktion erst mit einer Verzögerung geladen werden.

Ohne besondere Maßnahmen würden in der Zwischenzeit die nächsten Befehle in der linearen Folge geladen und mit ihrer Ausführung begonnen werden. Die Bearbeitung dieser „falschen“ Befehle ist abzubrechen. Außerdem müssen gegebenenfalls bereits vorgenommene Veränderungen an Registern o. ä. zurück genommen werden. Um diesen Mehraufwand zu vermeiden, kann man nach jedem Sprungbefehl entsprechend viele Instruktionen einfügen, die keine weiteren

Auswirkungen haben. Die meisten Assembler kennen für derartige Zwecke eine Instruktion NOP (*No Operation*).

Noch schwieriger ist die Situation bei bedingten Sprüngen. Hier dauert es oft noch länger bis entschieden ist, ob der Sprung ausgeführt werden soll oder nicht. Moderne Prozessoren verwenden ausgeklügelte Strategie für solche Fälle. Eine Möglichkeit ist, die Wahrscheinlichkeit für den Sprung abzuschätzen (*Branch prediction*). Eine plausible Annahme ist, dass das gleiche Ergebnis wie bei dem letzten Durchgang eintrifft. Damit liegt man z. B. bei Zählschleifen immer außer beim letzten Durchgang richtig. In der Mehrzahl der Fälle kann daher die Ausführung nahtlos fortgesetzt werden. Wurde der falsche Pfad gewählt, müssen die angefangenen Operationen abgebrochen und eventuelle Veränderungen rückgängig gemacht werden.

11.5 Cache-Speicher

11.5.1 Funktionsweise

Mit zunehmender Geschwindigkeit moderner Prozessoren erweist sich immer mehr der Speicherzugriff als bestimmend für die erzielbare Leistungsfähigkeit. Sehr schnelle Speicherbausteine sind teuer. Gleichzeitig benötigen Betriebssystem und Anwendungen immer mehr Speicherkapazität. Preisgünstigere Speicherbausteine benötigen länger für einen Zugriff als die CPU für die Anforderung. Nach einer Leseanforderung muss die CPU eine gewisse Zeit auf das Ergebnis warten. Anstatt weiter zu arbeiten, fügt sie Wartezyklen (*wait states*) ein.

Einen Ausweg bietet die Einführung eines so genannten Cache-Speichers (Cache, engl. geheimes Lager). Ausgangspunkt ist die Beobachtung, dass in der Regel auf bestimmte Speicherzellen viel häufiger zugegriffen wird als auf andere. Ein typisches Beispiel sind Schleifen. In dem Programmfragment

```
for( i=0; i<anzahlAktien; i++ ) {
    depotWert += aktie[i].wert;
}
```

wird einerseits der Programmcode für jeden Schleifendurchgang wiederholt. Andererseits werden immer wieder die Variablen `i`, `anzahlAktien` und `depotWert` verwendet. Wenn man diese Befehlscodes und Daten in einem schnellen Zwischenspeicher legt, kann die Ausführung weitgehend von dem langsameren Zugriff auf den Hauptspeicher abgekoppelt werden. Bild 11.3 illustriert diese Idee.

Das Konzept lässt sich leicht an Alltagssituationen erläutern. Wenn man ein Buch zur Arbeit benötigt, wird man es nicht nach jedem Nachschlagen zurück in das Regal stellen. Vielmehr wird man es griffbereit auf den Schreibtisch legen. Im Laufe der Zeit wird sich ein kleiner Stapel von aktuell verwendeten Büchern

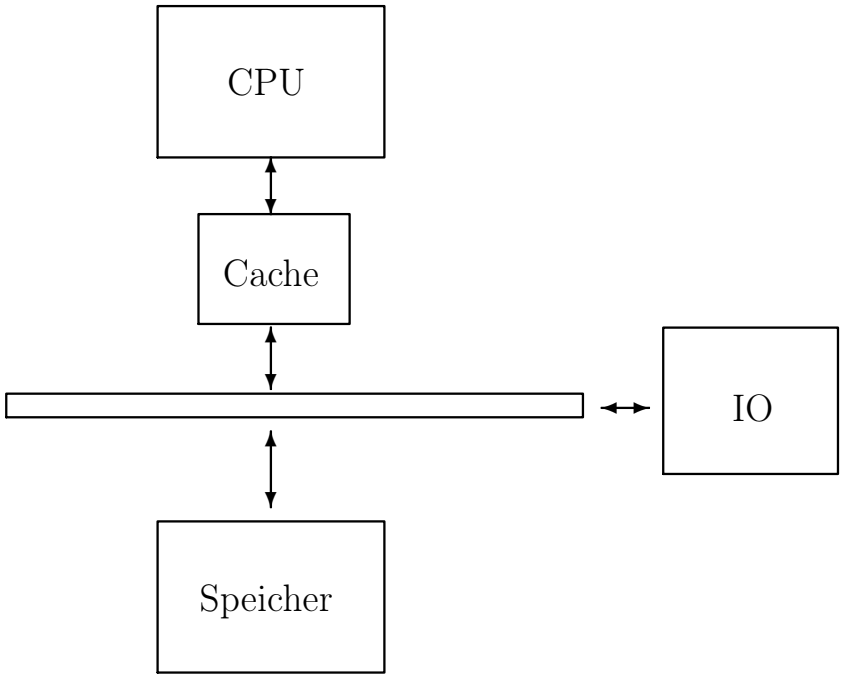


Abbildung 11.3: Schneller Cache-Speicher zwischen Hauptspeicher und CPU

auf dem Schreibtisch aufbauen. Erst wenn dort der Platz nicht mehr ausreicht, werden Bücher wieder zurück ins Regal gestellt.

Wieder bezogen auf einen Rechner gestaltet sich ein Lesezugriff wie folgt:

- Es wird geprüft, ob von der entsprechenden Zelle bereits eine Kopie im Cache vorliegt.
- Wenn ja (*Cache hit*), wird diese Kopie verwendet.
- Wenn nein (*Cache miss*), muss der Wert vom Hauptspeicher geholt werden. Gleichzeitig wird eine Kopie für die weitere Arbeit in den Cache gelegt.

Sofern die Anzahl der Treffer im Cache groß ist, ergibt sich ein deutlicher Gewinn an Performanz. Wie hoch der Gewinn tatsächlich ausfällt, hängt von der Anwendung ab.

11.5.2 Cache-Organisation

Bei dem ersten Lesezugriff auf eine Zelle, wird sie in den Cache aufgenommen. Dabei genügt es nicht, den Inhalt zu zu kopieren. Vielmehr muss auch die Adresse vermerkt werden. Ein Eintrag oder eine Zeile im Cache hat dann die Form

Adresse	Wert

In der Praxis benötigt man noch weiter Bits, um z. B. die Gültigkeit der Daten anzuzeigen. Festzustellen, ob eine Adresse bereits im Cache liegt, ist in diesem einfachen Konzept recht aufwändig. Da eine Zelle in ein beliebige Zeile des Caches liegen kann, sind alle Zeilen im Cache auf diese Adresse hin zu vergleichen.

Einfacher ist die Situation, wenn man anhand der ursprünglichen Adresse auf den Speicherplatz im Cache schließen kann. Diese Art von Speicher, bei denen eine Verbindung zwischen Adresse und Inhalt besteht, nennt man Assoziativ-Speicher. Betrachten wir als Beispiel folgende Situation:

- Der Hauptspeicher ist 64 k groß (16-Bit Adressen).
- Der Cache-Speicher hat 256 Zeilen (8-Bit Adressen).

Eine einfache Strategie ist, die letzten 8 Bit der Speicheradresse als Index für den Cache zu verwenden. Angenommen die Speicherzelle $4f3e$ im Hauptspeicher ist zum ersten Mal angesprochen. Dann wird davon das zweite Byte $3e$ oder 62_{10} abgetrennt und als Zeilenindex für den Cache verwendet. In diese Zeile wird Adresse und Werte eingetragen. Es ist allerdings nicht erforderlich, die komplette Adresse abzulegen. Vielmehr liegt der zweite Teil bereits durch die Zeilennummer fest. Es genügt also, das erste Byte zu vermerken. Damit erhält man die Darstellung:

62. (3e.) Zeile	Rest der Adresse: $4f$	Wert

Um umgekehrt festzustellen, ob die Adresse $4f3e$ bereits im Cache liegt, muss man nur nachschauen, ob in der $3e$. Zeile der Adressteil $4f$ steht und die Zeile als gültig markiert ist.

Mit diesem Prinzip kann der Zugriff einfach realisiert werden. Der Nachteil ist, dass es nur eine mögliche Ablagestelle für eine Adresse gibt. Entsprechend viele Adressen teilen sich diese eine Zeile. In unserem Beispiel würde auch die Adresse $b23e$ in die gleiche Zeile gehören. Dazu müsste der alte Eintrag für $4f3e$ überschrieben werden. Auch wenn im Cache noch andere freie Zeilen sind, kann keine andere Zeile genutzt werden. Unter ungünstigen Umständen kann eine schlechte Auslastung des Cache-Speichers resultieren. In der Praxis benutzt man daher Anordnungen, bei denen für eine gegebene Adresse zwei oder vier Ablagestellen in Frage kommen. Derartige Speicher werden dementsprechend als 2-fach oder 4-fach assoziativ bezeichnet. Sie bieten einen guten Kompromiss zwischen Vergleichsaufwand und guter Auslastung.

Wenn alle Ablageplätze belegt sind, muss einer der alten Werte überschrieben werden. Eine häufig verwendete Strategie besteht darin, den am längsten nicht mehr benutzen Wert zu verdrängen (*Least-Recently-Used* LRU). Dazu werden entsprechende Zähler in den Cache integriert, um die Zugriffsstatistik zu erfassen.

11.5.3 Realisierungen

Je nach Lage des Cache-Speichers unterscheidet man.

- Level 1 (L1): Der Cache ist in den Prozessor integriert.
- Level 2 (L2): Der Cache befindet sich in einem eigenen Baustein, der über eine schnelle Verbindung an den Prozessor angeschlossen ist.

Neben diesen beiden Hauptklassen werden auch die Bezeichnungen Level 0 bei Entkopplung der Komponenten innerhalb der CPU und Level 3 bei Systemen mit mehreren Prozessoren verwendet. Weiterhin werden oft getrennte Cache-Speicher für Programme und Daten verwendet.

Caching ist ein sehr wirksames Verfahren, um Zugriffe zu beschleunigen. Daher findet es auf verschiedensten Ebenen Anwendung. Durch einen vorgeschalteten Cache-Speicher werden Daten von der Festplatte schneller verfügbar. Das HyperText Transport Protocol (HTTP) unterstützt das Caching von Web-Seiten. Dabei werden z. B. auf dem lokalen Rechner Kopien der zuletzt besuchten Seiten abgelegt.

11.6 RISC-Rechner

Der 8086 ist ein Beispiel für einen Prozessor mit einem recht umfangreichen Befehlssatz. Sein Befehlssatz umfasst sowohl vergleichsweise einfache Operationen (`INC AX`) als auch komplexe Operationen (z. B. `LOOP`-Befehl). Die aufwändigen Befehle werden intern durch eine Folge von Mikrobefehlen realisiert. Ein derartiger Befehlssatz bedingt unterschiedliche Längen und Ausführungszeiten für die verschiedenen Befehle.

Mit der zunehmenden Integration wuchs der Befehlsatz immer mehr an. Andererseits musste man feststellen, dass ein so mächtiger Befehlssatz kaum vollständig ausgenutzt wurde. Untersuchungen ergaben:

- 80% der Zeit wird mit 20% der Befehlen ausgefüllt.
- 90% aller Befehle sind Load/Store oder einfache ALU-Befehle.

Komplexe Befehle kommen demnach relativ selten vor und ihr Beitrag zur Ausführungszeit ist relativ gering. Dabei spielt sicherlich auch eine Rolle, dass weder ein Programmierer noch ein Compiler Befehlsätze mit teilweise mehr als 400 verschiedenen Instruktionen optimal benutzen kann.

Ausgehend von diesen Überlegungen entstand das Konzept der *Reduced Instruction Set Computer* (RISC) als Gegenentwurf zu den *Complex Instruction Set Computer* (CISC). Wie der Name sagt, wird bei diesem Ansatz ein kleiner Befehlsvorrat verwendet. Gleichzeitig sollen die Befehle möglichst einfach gestaltet sein. Durch diese Beschränkungen ist eine Vereinheitlichung im Befehlsformat möglich, so dass die Dekodierung der Befehle erleichtert wird. Weiterhin können die Befehle direkt als Schaltungen realisiert werden. Die Notwendigkeit von Mikroprogrammen entfällt. Im einzelnen gelten für RISC-Architekturen folgende Punkte.

- Speicherzugriffe sind nur über LOAD und STORE Befehle möglich. Die ALU-Befehle können nur auf Register arbeiten. Soll ein Wert im Speicher verändert werden, so muss er zunächst in ein Register geladen werden. Dann kann er verändert und anschließend wieder in den Speicher zurück geschrieben werden. Ein CISC-Befehl wie `INC [200]` muss in drei RISC -Befehle aufgeteilt werden.

- RISC-Prozessoren haben viele Register. Da die Bearbeitung von Werten im Speicher aufwändig ist, sind viele Register zur internen Zwischenspeicherung sinnvoll. Üblich sind 50 bis 200 Register.

- Die Befehle haben ein einheitliches Format (gleiche Länge, gleicher Aufbau). Damit ist die Dekodierung einfach.

- Durch den überschaubaren und einheitlichen Befehlssatz ist eine Pipeline mit mehreren Stufen (z. B. Sun Sparc 4 Stufen) möglich. Wird in jedem Zeittakt ein Befehl in der Pipeline fertig, so bezeichnet man den Prozessor als skalar. Pro Zeittakt wird dann ein Befehl ausgeführt, unabhängig davon wie lange die Bearbeitung in der Pipeline dauert.

In der RISC-Architektur wird das Fehlen von einigen leistungsfähigen Befehlen oder Adressierverfahren durch die schnelle Ausführung und viele interne Register kompensiert. Um die Leistungsfähigkeit voll auszuschöpfen, muss ein Compiler diese Eigenschaften berücksichtigen. Beispielsweise ist es oft möglich, durch geschicktes Umordnen von Anweisungen Konflikte in der Bearbeitung der Pipeline zu vermeiden oder zumindest zu verringern.

11.7 Parallele Strukturen

Die Ausführungsgeschwindigkeit eines Rechners lässt sich nicht beliebig steigern. Bei allen Fortschritte der Halbleitertechnik mit beeindruckenden Steigerungen der Taktraten existieren doch grundsätzliche, physikalische Grenzen. Außerdem sind Systeme, die an der Grenze des zur Zeit gerade machbaren liegen, erfahrungsgemäß sehr teuer. Als Alternative bietet es sich an, anstelle einer sehr aufwändigen Einheit mehrere einfache Einheiten parallel zu schalten.

Wenn es gelingt, ohne übermäßigen Verwaltungsaufwand die Arbeit auf mehrere Prozessoren zu verteilen, so ist eine preisgünstige Lösung möglich. Der Gewinn durch Parallelisierung hängt sehr stark von der Aufgabe ab. Viele rechenintensive Probleme lassen sich gut in unabhängig voneinander zu bearbeitenden Teilprobleme aufteilen. Allerdings gibt es stets eine Grenze für diese Aufteilung. Gerne wird folgendes Beispiel aus dem Alltagsleben benutzt, um diese Problematik zu veranschaulichen:

- Ein Maurer braucht 8 Stunden für eine Wand.

- Zwei Maurer brauchen 4 Stunden für die selbe Wand.
- Wie lange brauchen 100 Maurer für die gleiche Wand?

Der Idealfall, dass N Einheiten die Aufgabe in der Zeit T/N erledigen, wird nur selten erreicht. Wie hoch die Leistungssteigerung (*Speedup*) ausfällt, hängt vom Problem und der Lösungsstrategie ab. Häufig sind für sequentielle und parallele Bearbeitung jeweils unterschiedliche Lösungswege optimal.

Die Parallelisierung kann auf verschiedenen Ebenen erfolgen. Auf oberster Ebene werden vollständige Rechner parallel betrieben. Mit diesem Ansatz wurden beispielsweise durch koordinieren Einsatz vieler, über das Internet verbundener Rechner einige Verschlüsselungsverfahren geknackt. Ein anderes Beispiel ist die Suche nach außerirdischen Signalen in Aufnahmen von Radioteleskopen. Im Projekt SETI@home (SETI steht für *Search for Extraterrestrial Intelligence*) kann jeder Anwender auf seinem Rechner sonst nicht benötigte Rechenleistung für die Suche bereit stellen [ACK+02].

Auf der untersten Ebene sind in einem Prozessor mehrere parallel arbeitende Einheiten integriert. Im einfachsten Fall führen diese Einheiten alle die gleiche Operation an mehreren Daten aus. Typische Anwendungen findet man in der Bildbearbeitung, wenn die gleiche Operation (z. B. Umwandlung von Farbwerten in Graustufen) auf alle Punkte eines Bildes angewendet werden soll. Allgemein lassen sich Verfahren, bei denen Vektoren oder Matrizen verwendet werden, leicht aufteilen. Die Parallelität kann räumlich oder zeitlich sein. Je nachdem unterscheidet man:

- Feldrechner: viele gleichartige Einheiten führen gleichzeitig eine Instruktion auf unterschiedlichen Daten aus. Die Einheiten können von unterschiedlicher Komplexität sein (vollständige Prozessoren oder spezialisierte Schaltungen) und über eigenen Speicher verfügen.
- Vektorrechner: eine Einheit ist für die mehrfache, schnelle Ausführung der gleichen Operation ausgelegt. Dazu dient eine spezielle Pipeline-Struktur. Bei langen Vektoren sind nach dem Füllen der Pipeline alle Stufen gleichmäßig ausgelastet.

Komplexer ist die Situation, wenn mehrere Einheiten jeweils eigene Befehle ausführen. Die einzelnen Befehle sind dann zu einem sehr langen Befehlsword (*very long instruction word* VLIW) zusammen gefasst. Aus diesem Befehlswort entnimmt jede Einheit ihre Instruktion und Information über Operanden.

Diese Anordnung ist theoretisch sehr leistungsfähig. Mit N parallelen Einheiten können N voneinander unabhängige Befehle angestoßen werden. Das Problem liegt in der Aufteilung der Arbeit. Einerseits sollen möglichst viele Einheiten gleichzeitig aktiv sein. Andererseits muss sichergestellt sein, dass es zu keinen Konflikten bezüglich Ablauf oder Daten kommt. In dieser Allgemeinheit wird der Compiler vor eine schwierige Aufgabe gestellt. Daher findet man in der Praxis nur vereinfachte Versionen dieses Konzeptes.

Tabelle 11.1: Klassifikation nach Flynn

	Ein Befehl	Mehrere Befehle
Ein Datenwert	SISD	MISD
Mehrere Datenwerte	SIMD	MIMD

11.8 Klassifikation von Rechnern

Eine systematische Einteilung von Rechnern in Bezug auf Parallelität stammt von M. J. Flynn [Fly72]. Er unterscheidet (Tabelle 11.1):

- SISD *Single Instruction, Single Data*: Der klassische von-Neumann Rechner. Zu einem Zeitpunkt wird eine Instruktion ausgeführt, die auf einem Datenwert oder einem zusammen gehörenden Satz von Datenwerten (z. B. zwei Eingangswerte, ein Ausgangswerte) wirken.

- SIMD *Single Instruction, Multiple Data*: Die gleiche Instruktion wird auf mehrere Datensätzen angewandt. Zu dieser Gruppe gehören Feld- und Vektorrechner.

- MISD *Multiple Instructions, Single Data*: Keine sinnvolle Kombination. Hierbei würden mehrere Instruktionen gleichzeitig auf einen Datensatz angewandt.

- MIMD *Multiple Instructions, Multiple Data*: Allgemeinster Fall mit unterschiedlichen Befehlen für unterschiedliche Datensätze. Realisierungen sind Multiprozessorsysteme.

In der Praxis sind die Grenzen zwischen den Kategorien fließend. Ähnliches gilt für die Unterscheidung zwischen RISC und CISC. Der Trend ist, das Beste aus verschiedenen Konzepten zu kombinieren.

Kapitel 12

Prozessoren

Unterscheidung von Rechnern nach Größe:

- Supercomputer: auf höchste Leistungsfähigkeit optimierte Rechner, in der Regel mit parallelen Strukturen. Heute oft durch Kombination von sehr vielen Standardprozessoren realisiert. Einsatzgebiete sind besonders rechenintensive Spezialanwendungen wie z.B. Simulationsrechnungen (Wettervorhersage, Klimamodellierung, etc.)
- Großrechner (Main frame): Zentralrechner für die gemeinsame Nutzung von Anwendungssystemen sowie die Bereitstellung zentraler Datenbestände.
- Minicomputer (Midrange): so zu sagen ein kleiner Großrechner. Charakteristisch ist die Fähigkeit, viele Prozesse gleichzeitig zu bearbeiten. Entsprechend können viele Benutzer gleichzeitig den Rechner verwenden.
- Arbeitsplatzrechner (Workstation): anspruchsvolle Einsatzgebiete wie CAD, 3-D Graphik, Video, Software-Entwicklung, wissenschaftliche Anwendungen.
- Personal Computer PC: persönlicher Rechner für Textverarbeitung, Tabellenkalkulation, Terminplanung, Email, Zugang zu Internet bzw. Intranet
- Personal Digital Assistant PDA: kleine tragbarer Rechner mit Funktionen wie Adressbuch, Terminplan, Taschenrechner und Spiele.
- Controller: einfache Rechner zur Steuerung von Geräten (Drucker, Kühlschrank, Auto, etc). In diesem Bereich spielen neben der Rechenleistung andere Kriterien wie Preis, Stromverbrauch, Platzbedarf, Robustheit gegenüber Umweltbedingungen eine große Rolle.

Neben diesen Universalrechnern gibt es Systeme optimiert für spezialisierte Anwendungen. Ein Beispiel sind Spielkonsolen. Hier werden schnelle Rechner mit guten Grafikfähigkeiten benötigt.

12.1 Intel 80x86-Architektur

Mit den Prozessoren aus der 80x86 Familie schuf die Firma Intel eine der erfolgreichsten Linien. Die folgende Zusammenstellung zeigt die wichtigsten Entwicklungsstufen.

1971 4004: 2300 Transistoren, 400-800 KHz, 640 Byte Adressraum

1972 8008: 500-800 KHz, 16 KByte Adressraum

1974 8080: 2 MHz, 64 KB

1978 8086: 29.000 Transistoren, 1 MByte Adressraum

1982 80286: Protected-Mode

1985 80386: 32-Bit Prozessor

1989 486-DX: Pipeline, integrierte Fließkomma-Einheit und integrierter Cache

1993 Pentium: zweite Pipeline, damit zwei Befehle pro Takt (superskalarer Prozessor), Branch Prediction, getrennter L1 Cache für Daten und Programm, 64-Bit breiter Datenbus,

1995 Pentium Pro: der erste Prozessor mit der P6 Mikro-Architektur. Dabei wird die Pipeline durch einen flexiblen Verbund von Einheiten ersetzt. Drei Dekodierer zerlegen die einkommenden Befehle in kleinere, so genannte Mikro-Ops. Diese wandern in einen gemeinsamen Pool. Sofern keine Konflikte vorliegen, werden sie dann in optimierter Reihenfolge aus diesem Pool auf 5 Verarbeitungseinheiten verteilt (*Out of order execution*).

1997 Pentium II: MMX-Befehlserweiterungen (*Multi Media Extension*) für Multimedia-Anwendungen: Gleichzeitige Bearbeitung nach dem SIMD-Schema von mehreren Integerzahlen, die in 64 Bit Registern gepackt stehen.

1999 Pentium III: 70 neue Instruktionen – Streaming SIMD extensions (SSE) – für parallele Gleitkomma-Operationen (128 Bit Register). Damit werden Anwendungen wie 2-D oder 3-D Grafik, Bild- oder Video-Bearbeitung, Spracherkennung, etc. beschleunigt.

2000 Pentium 4: Intel NetBurst Mikro-Architektur, 144 neue Befehle zur Verarbeitung multimedialer Daten (SSE2)

Hyper-Threading: ein Prozessor kann zwei Threads gleichzeitig bearbeiten.

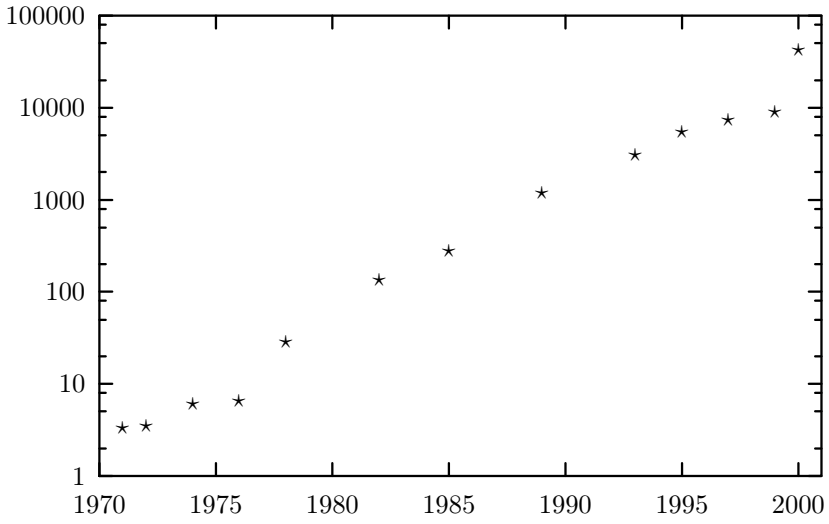


Abbildung 12.1: Anzahl der Transistoren in Intel-Prozessoren (Angaben in Vielfachen von 1000, ohne Cache).

12.1.1 Gesetz von Moore

Gordon E. Moore, damals Leiter der Forschung und Entwicklung von Fairchild Semiconductor und später Mitbegründer der Firma Intel sagte 1965 voraus, dass sich die Anzahl der Transistoren auf einem Chip von Jahr zu Jahr verdoppeln werden. In der Tat wuchs die Anzahl der Transistoren exponentiell an. Allerdings erwiesen sich die angenommenen 12 Monate als zu optimistisch. Derzeit liegt die Periode für eine Verdopplung bei circa 20 bis 24 Monate. In Bild 12.1 ist die Anzahl der Transistoren in den Intel-Prozessoren dargestellt. Die Werte belegen ein näherungsweise exponentielles Wachstum.

Es bleibt die spannende Frage, wie lange das Gesetz von Moore noch gilt. Bisher konnten physikalische Grenzen stets durch den Wechsel zu einer neuen Technologie überwunden werden.

Übung 12.1 *Gesetz von Moore*
Angenommen, das Mooresche Gesetzt gilt ungebrochen weiter. Welche Leistung ergibt sich, ausgehend von einem aktuellen Rechner, für ein Gerät in 10 Jahren? Rechnen Sie mit einer Periode von 20 Monaten und wenden Sie den berechneten Faktor auf alle Kenngrößen wie Rechenleistung und Speichergröße an. Welche neuen Anwendungen werden damit möglich sein?

12.2 Controller

Als Beispiel für Controller betrachten wir die Familie Motorola M68HC05 [Mot]. Es handelt sich dabei um low-cost 8 Bit Mikroprozessoren (Einzelpreis je nach Ausstattung zwischen 1,5 und 10 Euro). Der Prozessor ist in von Neumann Architektur aufgebaut. Er verfügt über 5 Register:

- Accumulator
- Index register
- Stack pointer
- Program counter
- Condition code register

Je nach Ausbaustufe enthält der Prozessor unterschiedlich große Speicherbereiche. Ein Teil davon kann die Daten permanent speichern. In diesen Bereich wird das Programm gelegt. Weiterhin können diverse Schnittstellen integriert sein. Zum Beispiel enthalten einige Modelle eine serielle Schnittstelle (RS-232), die zur Kommunikation mit anderen Rechnern genutzt werden kann. Die Schnittstellen werden über Speicheradressen angesprochen (memory mapped I/O).

Viele Anwendungen aus dem Bereich der Gerätesteuerung benötigen analoge Ein- oder Ausgangssignale. Ein Controller zur Steuerung einer Waschmaschine als Beispiel erhält Temperatur- und Druckinformationen von entsprechenden Sensoren. Die Umwandlung der analogen Signale in digitale Form übernehmen Analog-Digital-Converter (ADC). Derartige Bausteine sind in einige Prozessoren aus der M68HC05 Reihe enthalten. Dadurch können solche Applikationen mit einem Prozessor und einem Minimum an weiteren Bauteilen kostengünstig realisiert werden.

Kapitel 13

Speicher

13.1 Einleitung

Rechner benötigen Speicher um Programme und Daten abzulegen. Das Ideal möglichst viel von möglichst schnellem Speicher ist in der Praxis nicht zu erfüllen und auch gar nicht unbedingt notwendig. Vielmehr kann man durch eine Kombination unterschiedlicher Speicherarten leistungsfähige Systeme aufwandsgünstig realisieren. In Tabelle 13.1 sind verschiedenen Speicherformen als Hierarchie gemäß der zunehmenden Entfernung von der CPU zusammen gestellt. Den schnellsten Zugriff erlauben die innerhalb der CPU liegenden Register. Am anderen Ende stehen Wechselmedien, bei denen gegebenenfalls das entsprechende Band oder die benötigte CD eingelegt werden muss.

In der Diskussion des Cache-Konzeptes hatten wir gesehen, wie durch einen schnellen Zwischenspeicher für häufig benötigte Daten die Anzahl der zeitaufwändigen Zugriffe auf den Hauptspeicher reduziert wird. Dieser Ansatz der Verlagerung von Daten zwischen den einzelnen Speichern findet sich immer wieder. So werden wichtige Daten von der Festplatte in einen Cache im Hauptspeicher geladen. Umgekehrt werden bei Speicherknappheit Daten aus dem Hauptspeicher auf die Festplatte ausgelagert.

Bei Intel-Prozessoren wird dabei der Speicher in Seiten (*Pages*) von jeweils 4 kByte Größe unterteilt. Sofern erforderlich, wird dann immer eine komplette Seite aus- oder eingelagert (Swapping). Die Festplatte wirkt wie eine Erweiterung des Hauptspeichers. Dadurch steht insgesamt ein größerer Adressraum zur Verfügung (virtuelle Adressierung). Die Verwaltung übernimmt die so genannte *Memory Management Unit* (MMU).

Ähnlich ist die Situation bei Festplatten. Reicht der Platz nicht mehr aus, so kann man aktuell nicht mehr benötigte Dateien auf Wechselmedien wie CD, DVD oder Bänder auslagern. Derartige Speichermedien werden auch zur Archivierung und Datensicherung (Backup) oder zum Austausch mit anderen Rechnern eingesetzt.

Tabelle 13.1: Speicher-Hierarchie

Typ	Größe	Zugriffszeit
Register	5–200 Worte	
L1 Cache		5-10 ns
L2 Cache		15 ns
Hauptspeicher		60 ns
Festplatte	bis 100 GByte	10-20 ms
Backup-Medien (Bänder, DVD)		
Wechselmedien (CD, Diskette, Flash-Card, Memory-Stick)		

13.2 Halbleiter-Speicher

13.2.1 Speichertypen

Generell lassen sich Speicher nach zwei Kriterien unterscheiden:

- Den Möglichkeiten zum Schreiben.
- Der Fähigkeit, Daten auch ohne Spannungsversorgung zu speichern (nicht flüchtige Speicher).

Im folgenden werden die Grundtypen kurz vorgestellt.

Random Access Memory (RAM) RAM ist eine Speicherform, auf die von der CPU sowohl schreibend als auch lesend zugegriffen werden kann. Sowohl der Hauptspeicher als auch die Cache-Speicher sind mit RAM aufgebaut. Mit dem Ausschalten verliert ein RAM seinen Inhalt.

Read-Only Memory (ROM) ROM kann von der CPU nicht beschrieben werden. Die Inhalte werden im Fertigungsprozess eingetragen und können später nicht mehr verändert werden. Genutzt wird es beispielsweise im Bereich von Gerätesteuerungen, um feste Programme und Parameter abzulegen (firm ware). ROM ist relativ preiswert. Allerdings lohnt sich der Aufwand nur bei großen Stückzahlen. Nachteilig ist, dass kein Update der Software mehr möglich ist.

Programmable ROM (PROM) Unter dieser Bezeichnung fasst man Speichertypen zusammen, die im normalen Betrieb nur gelesen werden können. Allerdings können – im Gegensatz zu einem ROM – die Inhalte mit speziellen Maßnahmen neu geschrieben werden. Einige Formen sind: EPROM, OTP und EEPROM.

Erasable PROM (EPROM) Der Inhalt eines EPROM kann durch starke UV-Strahlung gelöscht werden. Chips mit EPROM kann man gut an einem kleinen Fenster aus Quarzglas erkennen. Durch dieses Fenster erreicht die UV-Strahlung das PROM. Zum Beschreiben benötigt man höhere Spannungen. Daher verwendet man spezielle Programmiergeräte zum Beschreiben von EPROM. Man bezeichnet den Schreibvorgang auch als Programmieren des EPROM.

One-time-programmable PROM (OTP) Verzichtet man in dem EPROM-Chip auf das teure Quarzfenster, so entfällt die Möglichkeit der erneuten Programmierung. Der Speicher kann nur einmal beschrieben werden. Diese Lösung ist bei kleinen Stückzahlen eine preiswerte Alternativen zu ROM-Speicher.

Electrical EPROM (EEPROM) Eine Weiterentwicklung von EPROMs sind EEPROMs. Hier benötigt man keine UV-Lampe mehr zum Löschen. Vielmehr erfolgt der Löschvorgang auf elektrischem Weg mit einem hohen Spannungsimpuls. Der Prozessor kann damit den Speicher mehrfach beschreiben. Allerdings ist der Zeitaufwand zum Löschen und Neuschreiben sehr viel größer als bei RAM. Außerdem ist die Anzahl der Schreibzyklen begrenzt (typische Werte liegen bei 10000 Zyklen). Vorteilhaft ist, dass die Daten dauerhaft erhalten bleiben.

Flash Flash-Speicher sind eine moderne Form von EEPROM. Die Daten können nur blockweise geschrieben werden. Das Beschreiben ist eine relativ aufwändige Prozedur. Etwa 100,000 Lösch-Schreibvorgänge sind möglich. Aufgrund des günstigen Preises finden Flash-Speicher in Form von Karten oder Sticks immer mehr Verbreitung als Speichermedium. Einsatzgebiete sind unter anderem digitale Kameras und MPEG-Player.

13.2.2 RAM-Bausteine

Statisches RAM (SRAM) In SRAM-Bausteinen sind die Speicherzellen mit elektronische Schaltungen (Flip-Flop) aufgebaut. Einmal beschrieben behält eine Zelle ihren Wert, solange die Versorgungsspannung anliegt (oder der Wert neu geschrieben wird). SRAM hat sehr kurze Zugriffszeiten. Der komplexe Aufbau beschränkt allerdings die Integrationsdichte und hält den Preis hoch. Daher wird SRAM nur eingeschränkt verwendet (z.B. für Cache).

Dynamisches RAM (DRAM) In einem DRAM wird ein Kondensator zur Speicherung verwendet. Die beiden Zustände 0 und 1 werden dann durch entladene oder geladene Kondensatoren dargestellt. Problematisch ist die Tatsache, dass sich die Kondensatoren im Laufe der Zeit entladen. Daher

muss im regelmäßigen Zeitintervallen die Information gelesen und wieder neu geschrieben werden (*Refresh*). DRAM sind preiswerter aber nicht so schnell wie SRAM.

Synchronous DRAM (SDRAM) Bei SDRAM-Bausteinen wird ein Teil der Steuerlogik in den Speicherbaustein integriert. Diese Bausteine können nicht nur einzelne Wert zurück geben, sondern „verstehen“ einfache Befehle. So können mit einer Anforderung mehrere aufeinander folgende Speicherzellen gelesen werden. Dadurch werden die häufig vorkommenden Blocktransfers beschleunigt.

Double data rate (DDR) SDRAM Eine Verbesserung der SDRAM sind die DDR SDRAM. Indem sowohl auf die steigende als auch die fallende Flanke des Taktsignals reagiert wird, wird die Datenrate verdoppelt.

Rambus DRAM (RDRAM) Eine Weiterentwicklung mit noch höheren Übertragungsraten bietet die von der Firma Rambus entwickelte Technologie. Laut Hersteller werden damit Übertragungsraten von 4-5 GByte pro Sekunde erreicht.

Synchronous graphics RAM (SGRAM) Speziell für Grafikkarten ausgelegt sind SGRAM-Bausteine. Sie unterstützen schnelle Blocktransfers sowie spezielle Maskierungsoperationen.

13.3 Diskettenlaufwerke

Disketten gehören zu den magnetischen Datenträgern. Es handelt sich dabei um eine dünne Kunststoff-Scheibe. Wegen ihrer Flexibilität wird sie auch als Floppy oder Floppy disk bezeichnet. Zum besseren Schutz sind die Datenträger in mehr oder weniger feste Plastikgehäuse eingebaut.

Auf dem Datenträger befindet sich eine dünne, magnetisierbare Schicht. Die Information ist im Zustand kleiner Bereiche gespeichert. Vereinfacht gesagt: je nach dem, ob ein Bereich magnetisiert ist oder nicht, enthält er eine 1 oder 0. Die Magnetisierung kann durch Anlegen eines äußeren Magnetfeldes verändert werden. Dazu dient ein Elektromagnet, der sich im so genannten Schreibkopf befindet.

Die Aufteilung einer Diskette zeigt Bild 13.1. Die gesamte Fläche wird in konzentrische Spuren (engl. *tracks*) unterteilt. Jede Spur enthält mehrere Sektoren. Die gebräuchlichsten Disketten mit 3,5 Zoll Radius verfügen pro Seite über 80 Spuren mit jeweils 18 Sektoren. Jeder Sektor ist 512 Byte groß. Insgesamt beträgt damit die Kapazität 1,44 MByte.

Als kleinste Einheit wird stets ein Sektor gelesen oder geschrieben (Interrupt 13H). Der Zugriff erfolgt über Köpfe. Die Positionierung enthält zwei Komponenten:

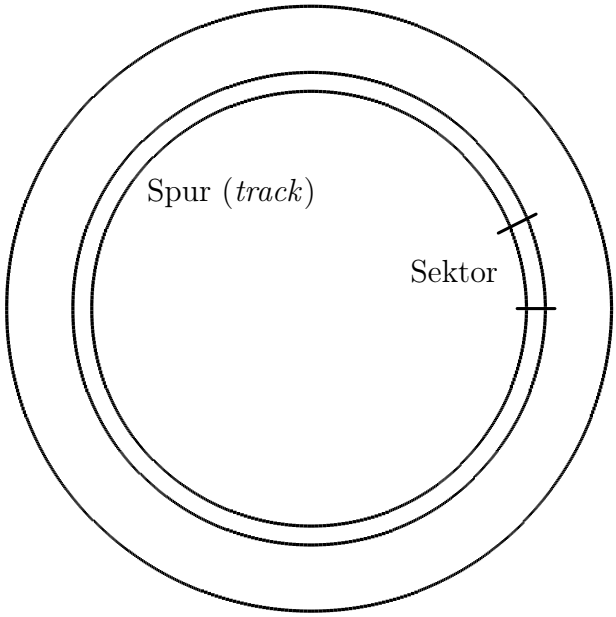


Abbildung 13.1: Aufteilung einer Diskette

1. Der Kopf wird zu der entsprechenden Spur bewegt.

2. Durch die Rotation (300 U/min) kommt der Sektor regelmäßig am Kopf vorbei.

Für die Zugriffszeit lässt sich kein fester Wert angeben. Allgemein gilt, dass der erste Zugriff relativ lange dauert. Eventuell muss der Motor erst anlaufen und der Kopf muss positioniert werden. Weiter Zugriffe in der gleichen Spur oder in benachbarten Spuren laufen sehr viel schneller ab.

Neben den Standard-Disketten gibt es eine Reihe von Neuentwicklung mit größerer Kapazität. Wohl die größte Verbreitung erreichten die ZIP-Laufwerke der Firma Iomega mit etwa 100 MByte Kapazität. Mehrere andere Produkte konnten sich trotz großer Kapazität (120 MByte) und gleichzeitiger Kompatibilität mit dem 3,5-Zoll Standard nicht am Markt durchsetzen.

Die ersten PCs wurden nur mit Disketten-Laufwerken als einzige Form von Massenspeicher ausgeliefert. Demgegenüber hat heute die Bedeutung sehr stark abgenommen und Disketten werden immer mehr von moderneren Speichermedien wie USB-Sticks verdrängt.

13.4 Festplatten

Festplatten funktionieren nach dem gleichen Prinzip wie Disketten. Allerdings bestehen die Datenträger aus festem Trägermaterial (Hard disk). Eine Festplatte enthält mehrere auf einer gemeinsamen Achse montierte Scheiben. Dieser Stapel rotiert mit hoher Geschwindigkeit (10000 U/min). Die Köpfe gleiten auf einem Luftpolster in geringem Abstand über den Scheiben.

Jede Scheibe ist wieder in Spuren und Sektoren aufgeteilt. Alle auf den verschiedenen Scheiben übereinander liegende Sektoren bilden einen Zylinder. Da die Köpfe auf einem gemeinsamen Zugriffskamm montiert sind, können nur Daten aus einem Zylinder gleichzeitig gelesen werden. Durch geschicktes Anordnen von zusammen gehörigen Daten sowie geeignete Optimierung der Kopfbewegungen wird die Performanz verbessert.

13.4.1 RAID-Systeme

Unter der Bezeichnung RAID für Redundant Array of Independent (oder Inexpensive) Disks fasst man verschiedene Möglichkeiten zur Kombination mehrerer Plattenlaufwerke zusammen. Neben der günstigeren Gesamtkosten verfolgt man dabei die beiden Ziele

- schnellerer Zugriff durch Parallelisierung
- Ausfallssicherheit durch Redundanz.

Forscher der University of California, Berkeley schlugen 1988 5 Alternativen vor und verglichen die Leistungsfähigkeit und Ausfallssicherheit [PGK88]. Für die Alternativen verwendeten sie die Bezeichnungen RAID Level 1 bis 5. Mittlerweile wurden weitere Typen definiert. Die wichtigsten werden im folgenden vorgestellt.

Level 0

Der Level 0 gehörte nicht zu den ursprünglichen Vorschlägen und beinhaltet keine Redundanz. Vielmehr handelt es sich um eine Reihe von parallelen Laufwerken. Dabei wird eine Datei für die Speicherung in Blöcke aufgeteilt. Diese Blöcke werden dann nacheinander auf den Laufwerken gespeichert (data striping). Hat eine Datei A die Größe von 5 Blöcken A1 bis A5, so ergibt sich bei zwei Laufwerken folgendes Bild:

Platte 1	Platte 2
A1	A2
A3	A4
A5	

In dieser Anordnung unterscheidet sich ein RAID Level 0 von einer einfachen Hintereinanderschaltung mehrerer Laufwerke (Just a Bunch of Disks, JBOD). Allerdings ist keine Redundanz vorhanden. Fällte ein Laufwerk aus, so sind alle Daten verloren.

Level 1

Redundanz durch Spiegelung (Mirroring):

Platte 1	Mirror 1
A1	A1
A2	A2
A3	A3
A4	A4
A5	A5

Vor- und Nachteile:

- Schnelleres Lesen durch parallelen Zugriff auf mehrere Platten
- Daten können bei Ausfall eines Laufwerks rekonstruiert werden
- Einfach zu implementieren
- Kostenintensiv

Level 10

Kombination von Level 0 und 1

Platte 1	Platte 2	Mirror 1	Mirror 2
A1	A2	A1	A2
A3	A4	A3	A4
A5		A5	

Level 4

Paritätsinformation auf zusätzlicher Platte:

Platte 1	Platte 2	Platte 3	Paritäts-Platte 2
A1	A2	A3	PA
B1	B2	B3	PB
C1	C2	C3	PC

Vor- und Nachteile:

- Daten können bei Ausfall eines Laufwerks aus Paritätsinformation rekonstruiert werden
- Nachteil: bei jedem Schreibvorgang ist auch Paritäts-Platte betroffen.

Level 5

Paritätsinformation wird auf alle Platten verteilt:

Platte 1	Platte 2	Platte 3	Platte 4
A1	A2	A3	PA
B1	B2	PB	B3
C1	PC	C2	C3

Je nach Realisierung unterscheidet man zwischen Software- und Hardware-RAID:

- Software-RAID: die Steuerung erfolgt durch eine Software auf dem Host. Gängige Betriebssystem beinhalten bereits entsprechende Komponenten.
- Hardware-RAID: ein eigener Controller

13.5 Optische Speicher

13.5.1 Compact Disc CD

Bei den optischen Speichern erfolgt der Zugriff auf das Speichermedium mit einem Laserstrahl. Zur Kodierung werden Bereiche mit unterschiedlichen Reflexionseigenschaften verwendet. Zum Lesen wird ein Laserstrahl auf einen Bereich gerichtet und die Menge des reflektierten Lichtes gemessen. Die Grundform der Compact Disc ist die CD-ROM, wie sie als Audio-CD oder Daten-CD verwendet wird. Dabei wird die Information im Fertigungsprozess aufgebracht.

Zur Kodierung dienen kleine Vertiefungen (*Pits*), die in die Oberfläche des Trägermaterials gepresst werden. Aus dem Wechsel von Vertiefungen und stehen gebliebenen Zwischenräumen (*Lands*) ergibt sich die Bit-Folge. Genauer gesagt gilt:

- 1 : Wechsel zwischen Land und Pit oder umgekehrt
- 0 : kein Wechsel

Sowohl zu schnelle Wechsel als auch zu lange Folgen von Land oder Pit erschweren die Synchronisation. Daher werden nur „günstige“ Folgen mit einem gut zu erkennenden Muster zugelassen. Zur Kodierung eines Bytes werden 14 Bit verwendet. Aus den $2^{14} = 16384$ möglichen Folgen werden nur 256 verwendet (*Eight-to-Fourteen-Modulation* EFM). Weitere Bits dienen zur Trennung aufeinander folgender Bytes und zur Fehlerkorrektur. Die Netto-Kapazität beträgt 840 MByte. Im Gegensatz zu den konzentrischen Spuren einer Festplatte enthält eine CD nur eine einzige, spiralförmige Spur (Gesamtlänge ca. 5 km).

Bei einmal beschreibbaren CDs (CD-R) wird das Muster durch einen Laser in eine Farbschicht eingebrannt. Der Effekt ist der gleiche wie bei einer CD-ROM. Es

entsteht eine Folge von Bereichen mit unterschiedlichen Reflektionseigenschaften. Die Rolle der Vertiefungen einer CD-ROM übernehmen „verbrannte“ Stellen.

Wieder beschreibbare CDs (CD-RW) enthalten eine Schreibschicht aus einer speziellen Legierung von Silber, Indium, Antimon und Tellur. Diese Legierung kann in zwei Formen vorliegen (*Phase Change Metal*):

- kristallin = hohe Reflektivität
- amorph = niedrige Reflektivität

Durch Erhitzen auf unterschiedliche Temperaturen (200 bzw. 600 Grad) wird gezielt das Material in einen der beiden Zustände gebracht. Im Ergebnis entsteht wieder ein Muster von Bereichen mit unterschiedlichen Reflektionseigenschaften. Laut Spezifikation kann eine CD-RW bis zu 1000 Mal neu beschrieben werden.

13.5.2 Digital Versatile Disc DVD

DVDs sind eine Weiterentwicklung der CD. Die Erhöhung der Kapazität wird im wesentlichen durch kleinere Pits und geringeren Spurabstand erreicht.

13.6 Bandlaufwerke

Speicherung auf Magnetband:

- Verwendung als preiswertes Backup-Medium bei großen Datenmengen
- Verschiedene Bandformate und Schreibverfahren (z. B. Exabyte, DAT)
- Kapazitäten bis etwa 400 GByte pro Band (komprimiert), Tendenz steigend
- Automatisches Laden von Bändern mit Robotersystemen

Kapitel 14

Grundlagen

14.1 Vermittlungsverfahren

Lange Zeit waren die beiden wohl wichtigsten Kommunikationsnetze Post und Telefon. Diese beiden Netze sind gleichzeitig Repräsentanten für zwei Netzwerktypen. Bei dem Postdienst werden einzelne Sendungen (Briefe, Pakete, o.ä.) vermittelt. Jedes Paket trägt die Zieladresse. Innerhalb des Netzes wird anhand dieser Zieladresse der Laufweg festgelegt: Briefkasten – örtliches Postamt – Postverteilzentrum – . . . – Empfänger. Aber selbst wenn man mehrere Briefe gleichzeitig an den gleichen Adressaten aufgibt ist nicht garantiert, dass die Briefe den gleichen Weg nehmen und gleichzeitig ankommen. Typischerweise wird an jedem Punkt nur über den nächsten Schritt entschieden. Diese Art der Vermittlung nennt man „**Paketvermittlung**“. Charakteristisch sind:

- diskrete Einheiten (Pakete)
- jedes Paket trägt die Adresse
- jedes Paket wird einzeln weitergeleitet
- im Allgemeinen keine Garantien bezüglich Laufweg und Laufzeit
- Robustheit gegenüber Unterbrechung einzelner Verbindungen

Demgegenüber wird beim klassischen Telefonnetz eine feste Verbindung zwischen den beiden Teilnehmern aufgebaut. Der Anrufer meldet seinen Verbindungswunsch an. Daraufhin wird im System eine Leitung zu dem Ziel aufgebaut. Wenn die Gesprächsverbindung zustande kommt, wird für die Dauer des Gesprächs eine feste Übertragungsrate exklusiv reserviert. Man spricht daher von „**Leitungsvermittlung**“. Wesentliche Eigenschaften sind:

- Aufbau einer Verbindung (Leitung) zwischen Quelle (Anrufer) und Ziel (Angerufenem)

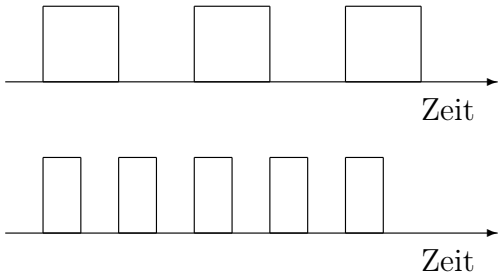


Abbildung 14.1: Einzelne Bits als zeitliche Impulse

- Erreichbarkeit des Zieles wird vor Beginn der Kommunikation sichergestellt
- Kommunikation erfolgt unter exklusiver Nutzung der Verbindung
- Zum Abschluss wird die Verbindung abgemeldet (Verbindungsabbau)
- Vorteil: konstante und garantierte Qualität
- Nachteil: schlechte Ausnutzung bei variabler Übertragungsrate
- Nachteil: empfindlich gegenüber Störung im Verbindungspfad

14.2 Leistungsfähigkeit

Für die Realisierung verschiedener Dienste sind unterschiedlich leistungsfähige Netzwerke erforderlich. Primäre Kriterien sind dabei Durchsatz und Verzögerung. Durchsatz oder Bandbreite gibt an, wie viele Daten in einer Zeiteinheit übertragen werden können. Das übliche Maß ist Bit pro Sekunde beziehungsweise die daraus abgeleiteten Einheiten wie etwa Mbit/s oder Gbit/s. Die Angabe 10Mbit/s beispielsweise bedeutet, dass das Netzwerk 10 Millionen Bits in einer Sekunde übertragen kann. Umgekehrt kann man aus dieser Angabe berechnen, wie viel Zeit die Übertragung eines einzelnen Bits benötigt. Für das Beispiel des 10Mbit/s Netzwerkes ergibt sich der Wert von 0,1μs (Mikrosekunden). Dies ist allerdings nur die Zeit, bis der Sender das nächste Bit abschicken kann. Man darf diesen Wert nicht mit der Laufzeit bis zum Empfänger verwechseln. Wenn man sich die einzelnen Datenbits als ein Folge von Impulsen auf der Zeitachse vorstellt, kann man sagen, dass ein Bit (Impuls plus Pause) 0,1μs breit ist.

Die Bandbreite wird durch die technische Realisierung der Leitung bestimmt. Eine Bandbreite von 10 Mbit/s bedeutet, dass irgendwo im System ein Takt von 1 MHz vorgegeben wird. Die Angabe von Frequenzen folgt der üblichen Konvention von Zehnerpotenzen, d. h. 1 MHz bedeutet 10^6 Hz. Daraus abgeleitet versteht man unter 1 Mbit/s auch 10^6 bit/s. Demgegenüber erfolgt die Größenangabe von Computerspeicher in Potenzen von 2. Ein kbit umfasst $2^{10} = 1024$ bit und

ein Mbit $2^{20} = 1048576$ bit. Die unterschiedliche Interpretation muss im Prinzip immer berücksichtigt werden. So dauert etwa die Übertragung einer Datenmenge von 1 kbit über eine 1 kbit/s Verbindung nicht exakt eine Sekunde. Vielmehr berechnet sich der exakte Wert zu

$$\frac{1024\text{bit}}{1000\text{bit/1s}} = 1.024\text{s}$$

Glücklicherweise ist der Unterschied recht gering (1.024 im Vergleich zu 1.000). Da in den allermeisten Fällen nur die ungefähre Größenordnung benötigt wird, kann man diese Feinheit zunächst ignorieren und den kleinen Fehler in Kauf nehmen.

Die zweite wichtige Charakteristik einer Verbindung ist die Verzögerung oder Latenz (von lat. lateō verborgen, versteckt, unbekannt sein). Latenz bezeichnet die Zeit, die eine Nachricht für den Weg vom Sender bis zum Empfänger benötigt. Die Latenz beinhaltet drei Komponenten:

1. Ausbreitungsverzögerung
2. Übertragungsverzögerung
3. Wartezeit

Die Ausbreitungsverzögerung t_A resultiert aus der endlichen Geschwindigkeit mit der sich Signale ausbreiten. Kein Signal kann schneller als mit Lichtgeschwindigkeit übertragen werden. Die Lichtgeschwindigkeit hängt vom Übertragungsmedium ab. Als Richtwert kann der Wert $c = 3.0 \times 10^8$ m/s für die Ausbreitung im Vakuum dienen. Damit kann man näherungsweise für die Ausbreitungsverzögerung bei einer Leitungslänge l den Wert $t_A = l/c$ ansetzen. Bei der Strecke Friedberg – Berlin (ca. 500km) erhält man

$$t_A = (500 \times 10^3)/(3 \times 10^8) = 166.7/10^5 = 1.667 \times 10^{-3} = 1.667ms \qquad (14.1)$$

Die Übertragungsverzögerung t_U ist durch die bereits oben diskutierte Dauer eines Bits bestimmt. Wir hatten gesehen, dass bei einer Bandbreite von B bit/s ein einzelnes Bit $1/B$ s an Übertragungszeit benötigt. Für ein Objekt der Größe K ergibt sich dann

$$t_U = K/B \qquad (14.2)$$

Die Definition bezieht sich auf größere Objekte, da in der Regel nicht ein einzelnes Bit sondern immer größere Datenblöcke als eine Einheit gesendet werden.

Die dritte Komponente – Wartezeit t_W – spielt dann eine Rolle, wenn die Übertragungsstrecke nicht nur aus einer einzelnen Leitung sondern aus mehreren Teilstücken besteht. Dann entsteht in den dazwischen liegenden Vermittlungsknoten eine Wartezeit. Die gesamte Wartezeit hängt von der Verarbeitungsgeschwindigkeit der einzelnen Knoten, der Größe der Datenpakete sowie der Anzahl der

Knoten ab. In praktischen Fällen ist die Wartezeit oft der größte Beitrag zur Latenz. Bei Paketvermittlung wartet ein Vermittlungsknoten häufig, bis er ein Paket vollständig erhalten hat, bevor er es weiter schickt. Bei hohem Verkehrsaufkommen muss das Paket zusätzlich auf eine freie Lücke warten. Typisch für beispielsweise einen transkontinentalen Kanal ist eine Latenz in der Größenordnung von 50 ms.

Die Latenz ist die Verzögerung vom Sender zum Empfänger. Häufig findet man auch die Angabe, wie lange ein Paket vom Sender zum Empfänger und wieder zurück benötigt. Man spricht dann von Roundtrip–Zeit (engl. Round–Trip Time, RTT). Zur besseren Unterscheidung benutzt man in Fällen, in denen eine Verwechslung auftreten könnte, für die einfache Verzögerung den Ausdruck Einweglatenz.

14.2.1 Bedeutung von Bandbreite und Latenz

Prinzipiell kann man sagen, je größer die Bandbreite und je kleiner die Verzögerung desto „besser“ ist die Verbindung. Allerdings hängen die konkreten Anforderungen von der jeweiligen Anwendung ab. Bei kleinen Objekten – z. B. der Übermittlung eines Tastendrucks – dominiert die Verzögerungszeit. Bei einem Kanal mit 1 Mbit/s beträgt die Übertragungsverzögerung für 1 Byte

$$t_U = 8/(1 \times 10^6) = 8\mu s = 0.008ms \tag{14.3}$$

Dieser Wert ist im Vergleich zur gesamten Latenz nahezu vernachlässigbar. Bei größeren Objekten erfolgt die Übertragung in vielen kleineren Paketen. Die gesamte Übertragungsdauer – die vom Benutzer wahrgenommene Latenz – berechnet sich dann aus der Latenz eines einzelnen Pakets und der benötigten Zeit, um entsprechend viele Pakete zu übertragen. Für die Übertragung einer Videodatei mit 20 MByte berechnete man bei dem 1 Mbit/s Kanal für die reine Übertragungsverzögerung

$$t_U = 8 \cdot 20 \times 2^{20}/(1 \times 10^6) = 168s \tag{14.4}$$

In diesem Fall spielt die Laufzeit eines einzelnen Pakets keine Rolle. Die Übertragung lässt sich nur durch Erhöhen der Bandbreite signifikant verbessern. In Tabelle 14.1 sind für verschiedene Bandbreiten und RTT-Werte die resultierenden Werte der wahrgenommenen Latenz zusammen gestellt. Die Werte zeigen, dass die höhere Bandbreite erst bei größeren Datenmengen zu spürbaren Verbesserungen führt.

14.2.2 Verzögerung–Bandbreite–Produkt

Während die ersten Daten auf dem Weg vom Sender zum Empfänger sind, kann der Empfänger bereits weitere Daten schicken. Ansonsten, wenn der Sender beispielsweise auf eine Empfangsbestätigung wartet, ist der Kanal nur schlecht ausgenutzt. Die Anzahl der Daten, die gleichzeitig auf dem Weg sind wenn der Sender

Tabelle 14.1: Wahrgenommene Latenz in ms für verschiedene Verzögerungszeiten und Objektgrößen

	1ms RTT		100ms RTT	
Größe	1 Mbit/s	10 Mbit/s	1 Mbit/s	10 Mbit/s
1Byte	1.008	1.0008	100.008	100.0008
1kByte	9.192	1.8192	108.192	100.8192
1MByte	8389.60	839.86	8488.61	938.86

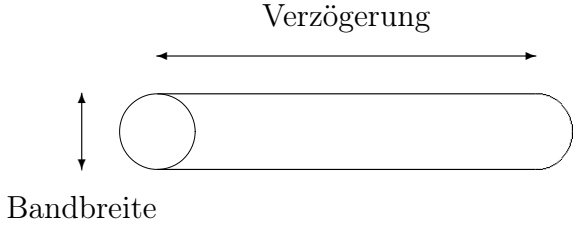


Abbildung 14.2: Verzögerung–Bandbreite–Produkt als Pipeline

permanent Pakete schickt, berechnet sich als Produkt aus Latenz und Bandbreite. In Bild 14.2 ist das so genannte Verzögerung–Bandbreite–Produkt grafisch als Röhre beziehungsweise Pipeline dargestellt. Ein Kanal mit einer Bandbreite von 10Mbit/s und einer Latenz von 50ms nimmt ein Volumen von

$$50 \times 10^{-3}s \cdot 10 \times 10^{6}Bit/s = 500 \times 10^{3}Bit \sim 62kByte \tag{14.5}$$

auf. Wenn der Empfänger das erste Bit empfängt sind bereits etwa 62 kByte Daten unterwegs. Signalisiert der Empfänger dem Sender, dass er keine weiteren Daten mehr aufnehmen kann, so benötigt der Sender wiederum die Latenzzeit von 50ms um reagieren zu können. Der Empfänger muss daher ausreichend Speicher vorhalten, um die Leitung noch leeren zu können. Andernfalls – wenn die Daten verloren gehen – müssen sie später erneut gesendet werden.

14.2.3 Anwendungen

In der Praxis sind die Anforderungen von Anwendungen an die Leistungsfähigkeit des Übertragungskanals sehr unterschiedlich. Häufig schwankt die Anforderung auch sehr stark während der Verbindungsdauer. So benötigt man während des Surfens im Internet während der meisten Zeit beim Lesen und Anschauen nur geringe Bandbreite. Demgegenüber wird beim Wechsel zu einer neuen Seite kurzfristig viel Bandbreite benötigt, um aufwändige Bilder oder Videos schnell zu laden. Am anspruchsvollsten im Hinblick auf die Netzwerksanforderungen sind

Tabelle 14.2: Datenraten für einige Anwendungen nach [Ste01]

Anwendung	Datenrate
Sprachübertragung (Telefonqualität)	64 kbit/s
Audiosignale (hohe Qualität)	1–2 Mbit/s
Videosignale (komprimiert)	2–10 Mbit/s
Video	1–2 Gbits/s

Anwendungen, die einen kontinuierlichen Datenstrom mit engen Randbedingungen für die Verzögerung erfordern. Ein klassisches Beispiel ist das Telefonnetz. Die erforderliche Datenrate ist zwar gering (64 kbit/s bei ISDN, 13 kbit/s bei GSM), aber die Daten müssen ohne grosse Verzögerung und für die Dauer der Verbindung ohne erkennbare Pausen übertragen werden. Aus diesem Grund werden Telefonverbindungen bis heute in großen Teilen mit leitungsvermittelnden Netzwerken realisiert. In Tabelle 14.2 sind für einige Anwendungen die erforderlichen Datenraten angegeben.

Neben der absoluten Verzögerung kann auch die relative Schwankung eine Rolle spielen. Bei einer Videoanwendung ist beispielsweise die absolute Verzögerung nicht entscheidend. Ob das Video nach 50 ms oder 200 ms beginnt ist nur ein geringer Unterschied. Aber nach dem Starten des Videos müssen die einzelnen Bilder immer rechtzeitig im erforderlichen Takt (z. B. 30 Bilder pro Sekunde) geliefert werden. Bei einem paketvermittelnden Netzwerk kann es zu Schwankungen der Laufzeit – so genanntem Jitter – kommen. Im Extremfall überholt ein Paket das vor ihm gestartet Paket. Beim Empfänger kommt in solchen Fällen das Abspielen des Videos ins Stocken. Wenn die absolute Verzögerung keine große Rolle spielt, kann der Empfänger das Problem umgehen, indem er zunächst einige Bilder speichert und das Video erst mit einer entsprechenden Verzögerung abspielt. Dann können bei eventuellen Verzögerungen vermehrt Bilder aus dem Speicher wieder gegeben werden, bis der Sender neue Bilder schickt.

14.3 Netzwerktypen und –topologien

14.3.1 Größe

Aus der Größe eines Netzes ergeben sich wichtige Randbedingungen und Konsequenzen für den Betrieb. Größe umfasst dabei sowohl die geographische Ausdehnung als auch die Anzahl der angeschlossenen Teilnehmer. Zwei wichtige Kategorien sind lokale Netze (LAN, *local area network*) und Fernnetze (WAN, *wide area network*). Ein LAN ist ein geschlossenes Netz innerhalb eines Gebäudes oder Firmengeländes. Die Reichweite ist damit auf wenige Kilometer beschränkt. Die Anzahl der angeschlossenen Teilnehmer ist bekannt. Weiterhin kann der Betreiber die eingesetzte Technik festlegen. Diese Faktoren vereinfachen das Netzwerkma-

nagment erheblich.

Ein WAN umfasst demgegenüber einen großen Bereich – beispielsweise ein Land oder einen Kontinent. Die Signallaufzeiten können daher unter Umständen sehr groß werden. In der Regel ist das Netz offen. Ständig werden neue Teilnehmer angeschlossen oder auch vorhandene Teilnehmer wieder abgemeldet. Für die Benutzung des Netzes müssen die Teilnehmer Gebühren bezahlen. Ein öffentliches Netz ist gerade dadurch gekennzeichnet, dass jeder der die technischen Voraussetzungen für einen Netzanschluss erfüllt und die Gebühren bezahlt das Netz auch benutzen darf. Der Betrieb eines solchen Netzes erfordert daher eine klare Regelung der technischen und rechtlichen Fragen. Durch Standards wird sicher gestellt, dass die in aller Regel sehr heterogenen Endgeräte von verschiedenen Herstellern mit dem Netz zusammen funktionieren.

Neben diesen beiden Grundtypen unterscheidet man noch zwischen verschiedenen Mischformen. Häufig benutzt man die Kategorie Stadtnetz (MAN, *metropolitan area network*). Das weltweite Netz kann man als Netzverbund mehrerer WANs oder als ein GAN (*global area network*) sehen. In [Ste01] findet man weiterhin die Definitionen

- PAN (*personal area network*) oder Piconetz für die Vernetzung eines Arbeitsplatzes
- SAN (*storage / system area network*) für die Vernetzung eines Raumes – typischerweise ein Rechenzentrum

Bezüglich der Offenheit kann man weiterhin unterscheiden zwischen

- Internet: das öffentliche und für alle offene Netz.
- Intranet: ein Netzwerk basierend auf der gleichen Technik wie das Internet, aber nur für einen geschlossenen Benutzerkreis (z. B. Mitarbeiter einer Firma, Angehörige einer Hochschule) zugänglich.
- Extranet: ein Intranet, das ausgewählten, externen Benutzern (z. B. Kunden) einen Zugang ermöglicht.

14.3.2 Topologien

Bei dem Entwurf eines Netzes gibt es zahlreiche Möglichkeiten, die einzelnen Rechner zu verbinden. Dies betrifft sowohl die physikalische Verkabelung als auch die logische Struktur. So kann es beispielsweise sinnvoll sein, bestimmte Teile des Netzes gegen andere abzuschirmen um etwa den Zugriff auf sensible Daten zu kontrollieren. Für die Verkabelung spielen Fragen wie die erforderlichen Bandbreiten und einzuhaltenden Verzögerungen aber auch wirtschaftliche Überlegungen eine Rolle. Innerhalb eines Netzwerkes gibt es einige Standardstrukturen oder Topologien für Netzwerke. Im folgenden werden diese Standardtopologien

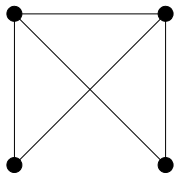


Abbildung 14.3: Teilweise vermaschtes Netz

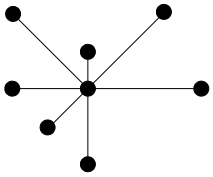


Abbildung 14.4: Sternnetz

kurz vorgestellt. In der Praxis wird man selten die reinen Formen sondern durch Kombination verschiedener Typen Mischformen realisieren.

Punkt-zu-Punkt Topologie

Beim Maschennetz sind je zwei Endpunkten direkt verbunden. Dadurch ist eine sehr schnelle Kommunikation zwischen Hosts möglich. Darüber hinaus können einzelne Verbindungen an betreffende Geräte angepasst werden. Insofern wäre eine komplette Vermaschung aller Rechner innerhalb eines Netzes die leistungsfähigste Lösung. Allerdings ist dies bei wachsender Anzahl von Endgeräten nicht mehr praktikabel. Bei N Endgeräten braucht jedes Endgerät $N - 1$ Anschlüsse und insgesamt hätte man $N * (N - 1)$ Leitungen. Daher wird meist nur eine teilweise Vermaschung durchgeführt. Nicht direkt verbundene Endgeräte kommunizieren dann über Zwischenstationen.

Stern–Topologie

Eine Topologie mit sehr einfacher Struktur erhält man durch sternförmige Anordnung der Endgeräte um einen zentralen Knoten. Jedes Endgerät kommuniziert dann direkt nur noch mit dem Zentralknoten. Der Zentralknoten muss entsprechend leistungsfähig sein. Sinnvoll kann es sein an den Zentralknoten Komponenten wie Fileserver oder Backup-Geräte anzuschließen.

Baum–Topologie

Verwandt mit der Stern–Topologie ist die Baum–Topologie. Im Unterschied zur Stern–Topologie sind hierbei nicht alle Endgeräte an einen Zentralknoten ange-

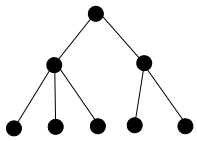


Abbildung 14.5: Baum

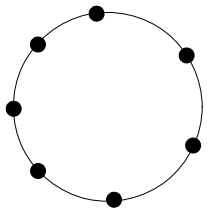


Abbildung 14.6: Ring

schlossen sondern es gibt Zwischenknoten, die jeweils mehrere Endgeräte bedienen.

Ring–Topologie

Eine Topologie mit gleichmäßiger Arbeitsteilung – zumindest bezüglich der Kommunikation – ist die Ring–Topologie. Dabei ist jeder Knoten nur mit zwei anderen Knoten verbunden. Jedes Datenpaket wird dabei in jedem Knoten gelesen und – falls es nicht für diesen Knoten bestimmt ist – weiter gereicht. Da auf jeder Leitung nur jeweils ein Knoten sendet und empfängt, hat man gute Kontrolle über die Leitung und kann einen hohen Durchsatz erzielen. Nachteilig ist die hohe Abhängigkeit des Gesamtsystems von jeder einzelnen Leitung.

Bus–Topologie

In den bisher besprochenen Topologien verbindet eine Leitung stets zwei Knoten. Eine grundsätzliche Alternative ist die Nutzung einer gemeinsamen Leitung – dann auch als Bus bezeichnet. Jeder Knoten wird direkt mit diesem Bus verbunden. Um einen anderen Knoten zu erreichen, schickt ein Knoten eine Nachricht auf den gemeinsamen Bus. Alle Knoten müssen ständig die Daten auf dem Bus beobachten und die an sie adressierten Nachrichten aufnehmen. Eine Stärke dieser Anordnung ist die effiziente Realisierung von Broadcast Nachrichten, d. h. Nachrichten die an mehrere oder sogar alle Knoten im Netz gerichtet sind. Ein derartiges System stellt eine Reihe von besonderen Anforderungen. So muss man mit entsprechenden Protokollen sicher stellen, dass nicht zwei Knoten gleichzeitig senden oder ein Knoten den Bus dauerhaft belegt.

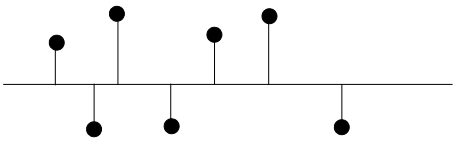


Abbildung 14.7: Bus

14.4 Referenzmodelle

14.4.1 Protokollstapel

In vielen Fällen kommunizieren Sender und Empfänger nicht direkt sondern über Zwischenstufen. Betrachten wir folgendes Beispiel: Herr A. möchte 12 Flaschen Rotwein von Chateau Rothschild in Bordeaux kaufen. Seine Nachricht kann man wie folgt darstellen:

Herr A.	12 Flaschen Rotwein	Chateau Rothschild

Herr A. spricht weder Französisch noch kennt er die Anschrift von Chateau Rothschild. Daher wendet er sich an seinen Weinhändler als Vermittler. Der Weinhändler kennt einen Großhändler in Bordeaux, spricht selbst allerdings auch kein Französisch. Er übersetzt daher die Bestellung in Englisch und schickt folgende Nachricht an den Großhändler:

Herr A.	12 bottles red wine	Chateau Rothschild	Großhändler in Bordeaux

Der französische Großhändler entfernt seine eigene Adresse, übersetzt in Französisch und leitet die Bestellung weiter.

Herr A.	12 bouteilles vin rouge	Chateau Rothschild

Der gesamte Ablauf der Bestellung ist in Bild 14.8 dargestellt. Dabei ist als zusätzliches Element noch die Übertragung der Nachricht als Brief per Post eingetragen. Das Bild zeigt, wie die eigentliche Kommunikation über mehrere Zwischenstufen abläuft. Auf diesem Weg wird die Nachricht zweimal übersetzt. Wichtig ist dabei, dass die einzelnen Schritte unabhängig voneinander erfolgen. Für die Bestellung ist es unwesentlich, ob die beiden Händler sich in Englisch oder irgend einer anderen Sprache verständigen, solange sie sich auf eine gemeinsame Sprache verständigen können. Entsprechend brauchen die beiden Endbenutzer die Übertragungssprache nicht zu verstehen.

Für jede Schnittstelle muss nur zwischen den beiden beteiligten Partnern ein Protokoll verabredet werden. Die gesamte Übertragung beinhaltet eine Kette von Protokollen - den so genannte Protokollstapel Protokollstapel(*protocol stack*). Auf der Sendeseite wird die Protokollkette von oben nach unten abgearbeitet bis die Nachricht physikalisch verschickt wird. Beim Empfänger durchläuft die Nachricht die Kette dann in umgekehrter Reihenfolge. Typisch ist dabei, dass

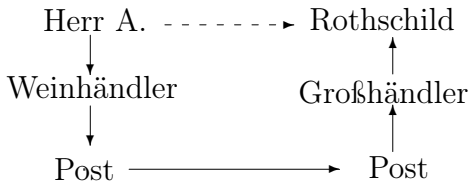


Abbildung 14.8: Ablauf Rotwein–Bestellung

auf der Sendeseite die unteren Ebenen zusätzliche Informationen anfügen (im Beispiel die Adresse des französischen Großhändlers), die dann beim Empfänger auf der gleichen Ebene wieder entfernt werden.

Diese Situation – die beiden Kommunikationspartner verständigen sich indirekt über eine Reihe von Zwischenschichten – stellt die große Herausforderung für heterogen Netzwerke dar. Ein typisches Beispiel ist die Kommunikation zwischen einem Web-Browser, der auf einem PC unter Windows2000 läuft, und einem Web-Server auf einem Großrechner mit dem Betriebssystem Linux. Trotz aller Unterschiede in Hard- und Software können die beiden Anwendungen sich verständigen.

14.4.2 OSI Referenzmodell

Zum Verständnis der Abläufe in Netzwerken ist eine standardisierte Darstellung der verschiedenen Prokollebenen hilfreich. Eine klare Definition funktionaler Ebenen erlaubt eine starke Kapselung. Die Schnittstelle zwischen zwei Ebenen kann dann leicht ausgetauscht werden, ohne dass es zu Rückwirkungen auf die anderen Protokolle kommt. Eine formale Definition der verschiedenen Schichten entwickelte die ISO (*International Standards Organization*) ab etwa 1977. Resultat ist das ISO/OSI 7 Schichten Modell (OSI: *Open Systems Interconnection*). Bild 14.9 zeigt die 7 Schichten des Referenzmodells. In diesem Modell sind weiterhin die Schnittstellen zwischen den Schichten festgelegt. Man unterscheidet zwischen Protokollen und Diensten. Protokolle definieren die Schnittstelle zwischen zwei Systemen auf einer Ebene. Demgegenüber sind in der OSI–Terminologie Dienste Funktionen, die eine Schicht der nächsthöheren Schicht zur Verfügung stellt.

Bitübertragung

Die Bitübertragungschicht (*physical layer*) stellt einen Bitstrom zwischen Sender und Empfänger bereit. Auf dieser Ebene spielt die Bedeutung der Bits keine Rolle. Vielmehr geht es darum, wie die einzelnen Bits übertragen werden. Darunter fallen Fragen wie Übertragungsmedien, Stecker, Darstellung einzelner Bits, Aufbau einer Verbindung, etc.

7 Anwendung		7 Anwendung
6 Darstellung		6 Darstellung
5 Sitzung		5 Sitzung
4 Transport		4 Transport
3 Vermittlung		3 Vermittlung
2 Sicherung		2 Sicherung
1 Bitübertragung	⇔ Netzwerk ⇔	1 Bitübertragung

Abbildung 14.9: ISO/OSI 7 Schichten Referenzmodell

Sicherung

Bei der Bitübertragung können Fehler auftreten. Aufgabe der Sicherungsschicht (*data link layer*) ist es, solche Fehler zu erkennen und durch entsprechende Mechanismen zu korrigieren. Dazu werden Blöcke von Datenbits zusammen gefasst. Die so gebildeten Datenrahmen (*data frames*) werden mit zusätzlichen Bits zur Kennung von Anfang und Ende versehen. Weitere Kontrollbits erlauben dem Empfänger, die korrekte Übertragung des Rahmens zu überprüfen. Erkennt der Empfänger Fehler, kann er sie bis zu einer gewissen Grenze korrigieren. Ist der Rahmen so gestört, dass eine Korrektur nicht mehr möglich ist, fordert der Empfänger den Sender auf, den Rahmen erneut zu senden.

Vermittlung

Der Weg den die Daten nehmen sollen wird in der Vermittlungsschicht (*network layer*) festgelegt. Diese Festlegung kann beispielsweise zu Beginn einer Verbindung erfolgen. Alternativ kann auch für jedes Paket neu die gerade optimale Route bestimmt werden. Man spricht auf der Vermittlungsschicht nicht mehr von Rahmen sondern von Paketen. Die Vermittlungsschicht enthält in der Regel auch Abrechnungsfunktionen, um die entstandenen Gebühren zu erfassen.

Wenn auf dem Weg vom Sender zum Empfänger mehrere Knoten liegen, werden in jedem der Knoten die drei untersten Schichten benötigt.

Transport

Die Transportschicht (*transport layer*) ist die erste Ende–zu–Ende Schicht. D. h. das Programm in der Transportschicht der Sendeseite kommuniziert mit einem passenden Gegenstück auf Empfängerseite. Aufgabe der Transportschicht ist, größere Datenmengen von den oberen Schichten zu übernehmen, falls notwendig in kleinere Einheiten zu zerlegen und dann an die Vermittlungsschicht weiter zu

geben. Die Transportschicht ist auch zuständig für den Aufbau der Verbindung.

Sitzung

Die Sitzungsschicht (*session layer*) bietet eine übergeordnete Sicht der Verbindung. Eine Sitzung kann verschiedene Verbindungen beinhalten, z. B. für Hin- und Rückkanal oder in zeitlicher Folge wenn die Verbindung nicht permanent gehalten wird.

Darstellung

In der Darstellungsschicht (*presentation layer*) werden die von unten gelieferten Daten interpretiert und in die für das System richtige Darstellung gebracht. Beispielsweise werden für die Übertragung Standardrepräsentationen für Daten wie Zeichen, Festkommazahlen, etc. definiert. In der Darstellungsschicht erfolgt dann die Umsetzung aus der allgemeinen Repräsentation in das maschinenspezifische Format. Zu den weiteren Aufgaben in dieser Schicht gehören Verschlüsselung und Datenkompression.

Anwendung

Die Anwendungs– oder Verarbeitungsschicht (*application layer*) stellt dem Benutzer bestimmte Dienste zur Verfügung. Dazu gehören der Dateitransfer, das Anmelden auf anderen Rechnern oder die Kontrolle von Prozessen.

14.4.3 Bedeutung des OSI Referenzmodells

Das ISO/OSI 7 Schichten Modell bietet eine gute Grundlage zum Verständnis sowie zur vergleichenden Beurteilung von eingesetzten Technologien. Die gewählte Einteilung in 7 Schichten sollte allerdings nicht zu starr gesehen werden. In vielen Fällen erweist sich die Einteilung als teilweise zu fein oder zu grob. Die höheren Schichten sind oft in weniger Schichten realisiert. Beispielsweise hat die im Internet verwendete TCP/IP Architektur insgesamt nur 4 Schichten. Tanenbaum benutzt in seinem Standardwerk [Tan00] ein vereinfachtes Modell mit 5 Schichten. Umgekehrt ist es sinnvoll, bei genauerer Untersuchung der unteren Schichten weitere Zwischenschichten einzuführen.

In der Praxis hat das Modell nie die von den Initiatoren erhoffte universelle Bedeutung erlangt. Durch den aufwändigen und zeitraubenden Standardisierungsprozess setzten sich oft am Markt mehr pragmatisch entstandene aber frühzeitig verfügbare Systeme durch. Universitäten als Vorreiter neuer Techniken haben eine andere Kultur als die ITU/ISO Gremien. Hier gilt mehr das Interesse an funktionierenden Programmen als an detailgenauen Spezifikationen (*„Grober Konsensus und laufender Code“*).

Die Vorlesung orientiert sich daher nicht streng an dem ISO/OSI 7 Schichten Modell. Vielmehr liegt der Schwerpunkt auf einzelnen Themen, die sich teilweise über mehrere Schichten ziehen. Im Zweifelsfall wird die inhaltliche Nähe stärker berücksichtigt als die Zuordnung zu den Schichten. Die folgenden vier Kapitel behandeln Rechnernetze zunehmender Komplexität. Im einzelnen lassen sich die Stufen

1. direkte Verbindung zweier Rechner
2. Kommunikation in einem lokalen Netz
3. Verbindung mehrerer lokaler Netze

unterscheiden. Anhand der Verbindung zweier Rechner werden Grundlagen wie Bildung von Rahmen oder Einfügen von Sicherungsinformationen dargestellt. Die Anforderungen, die sich aus dem Zusammenschluss mehrerer Rechner ergeben, werden zunächst für lokale Netze diskutiert. Schließlich wird die Verbindung vieler einzelner lokalen Netze unterschiedlichster Art zu einem großen Netzverbund – dem Internet – beschrieben.

14.5 Übungen

Übung 14.1

Welche Bandbreite ist für eine Echtzeitübertragung in den folgenden Fällen erforderlich:

1. *Musik von CD-ROM (75 Minuten Musik auf 650 MByte)*
2. *Video mit Auflösung 320 x 240 Pixel, 1 Byte Farbinformation pro Pixel, 10 Bilder pro Sekunde*
3. *Video mit Auflösung 640 x 480 Pixel, 3 Byte Farbinformation pro Pixel, 30 Bilder pro Sekunde*
4. *Wie stark müssen die Daten jeweils komprimiert werden, damit sie in Echtzeit über eine ISDN-Leitung (64 kbit/s) übertragen werden können.*

Übung 14.2 *In den Fernsehnachrichten wird ein Live-Interview mit dem Korrespondenten in New York gezeigt. Die Übertragung von Bild- und Sprachdaten erfolgt über getrennte Wege:*

Sprache: *eine Satellitenverbindung über einen geostationären Satelliten, Gesamtstrecke 70000 km.*

Bild: *über ein Unterseekabel, Gesamtstrecke 5000 km.*

Berechnen Sie die Ausbreitungsverzögerung für beide Strecken. Welche Laufzeitdifferenz ergibt sich?

Übung 14.3 *Eine Verbindung soll nicht mehr als 10 ms Einwegverzögerung haben.*

1. *Wie ist die maximale Länge, wenn nur die Ausbreitungsgeschwindigkeit berücksichtigt wird?*

2. *Wie ist der entsprechende Wert, wenn man zusätzlich die Übertragungsverzögerung bei einer Bandbreite von 1 Mbit/s und einer Paketgröße von 1024 Bytes einrechnet?*

Übung 14.4 *Sie sollen die Kommunikation mit einem Erkundungsfahrzeug auf dem Mond planen. Die Entfernung beträgt 385000 km und die Übertragung soll über eine Bandbreite von 1 Mbit/s erfolgen.*

1. *Wie groß ist die Ausbreitungsverzögerung? Wie groß ist damit die minimale Antwortzeit auf eine Anfrage?*

2. *Wie sind die Werte für die Verbindung mit einen Roboter auf dem Mars (Kleinster Abstand zur Erde ungefähr 54 Millionen Kilometer)?*

3. *Berechnen Sie auf der Basis der Ausbreitungsverzögerung das Verzögerungs-Bandbreiten Produkt für die Verbindung zwischen Erde und Mond.*

4. *Eine Webcam auf dem Mond nimmt Bilder in guter Qualität auf. Ein Bild benötigt 2 MByte Speicherplatz. Wie lange dauert es mindestens bis nach der Anforderung von der Erde aus ein Bild vollständig übertragen wurde.*

Übung 14.5 *Entwerfen Sie ein Netz für 10 Knoten in Form von zwei verbundenen Sternen.*

Übung 14.6 *Sie erstellen mit einem Textverarbeitungsprogramm eine Hotelreservierung, die Sie dann ausdrucken und per Fax an ein Hotel in Miami schicken.*

1. *Verfolgen Sie den Übertragungsweg. Welche Umwandlungen werden vorgenommen, wie sind die Schnittstellen?*

2. *Vergleichen Sie den Ablauf mit der Alternative, den Text als Email an das Hotel zu schicken. Wo liegen Vor- und Nachteile der beiden Möglichkeiten?*

Übung 14.7 *Unter Betriebssystemen wie Windows oder UNIX gibt es eine Reihe von Anwendungen zum Testen und Konfigurieren von Netzwerken:*

- `hostname`

- `ipconfig`
- `ping`
- `tracert`
- `nslookup`
- `net`
- `netsh`
- `netstat`

(Namen der Windows-Versionen). Informieren Sie sich an Hand der Hilfe-Funktionen über die Möglichkeiten dieser Anwendungen. Benutzen Sie die Anwendungen um die folgenden Fragen zu beantworten bzw. Aufgaben zu lösen:

- *Welchen Namen und welche IP Adresse hat Ihr Rechner?*
- *Testen Sie die Verbindung zu einem anderen Rechner.*
- *Welche Route wird zwischen den beiden Rechnern genommen?*
- *Eröffnen Sie auf einem zweiten Rechner eine Session mit* `telnet` *(sofern der andere Rechner als Server für* `telnet` *arbeitet).*

Übung 14.8 *Schreiben Sie ein Programm, um eine Liste von Adressen auf Erreichbarkeit zu testen. Sie können dazu aus dem Programm heraus wiederholt die Anwendung* `ping` *starten. In vielen Programmiersprachen wie z. B. C oder Perl steht für solche Zwecke eine Funktion* `system` *zur Verfügung. In Java wird mit*

```
Process p = Runtime.getRuntime().exec(cmd);
```

der Befehl im String `cmd` *ausführen. Die Ausgaben des Prozesse kann man nach folgendem Muster aus seinem Ausgabestrom in die Konsole kopieren:*

```
InputStream is = p.getInputStream();
int i;
while( (i = is.read() ) > -1 ) System.out.print( (char) i );
```

Hinweis: Für einen besseren Ablauf sollte der `InputStream` *in einen* `BufferedInputStream` *eingebettet werden.*

Kapitel 15

Rechner zu Rechner Verbindung

15.1 Übertragung von Bits

Das einfachste Modell für eine Übertragung ist eine Leitung, auf der ein Signal zwei Zuständen annehmen kann. Dieses Signal bewegt sich mit der Ausbreitungsgeschwindigkeit vom Sender zum Empfänger. Die Dauer eines einzelnen Zustandes bestimmt die erreichbare Übertragungsrate. Bei einer Dauer Δt kann man pro Sekunde $1/\Delta t$ Zustände übermitteln.

Bei einer elektrischen Leitung werden die beiden Zustände beispielsweise durch zwei verschiedene Spannungen realisiert. Die beiden Zustände seien als low und high bezeichnet. Über diese Leitung sollen Daten übertragen werden. Dazu muss eine Zuordnung der Bits zu den Zuständen getroffen werden - die **Kodierung**. Die nahe liegende Vereinbarung ist, für den Wert 1 das Signal high und für 0 low zu übertragen. Bild 15.1 zeigt für eine Folge von 0–1 Werten die Zustände in zeitlicher Folge. Aus im folgenden klar werdenden Gründen nennt man diese Art der Kodierung Non-Return to Zero (NRZ).

Einen etwas allgemeineren Fall zeigt Bild 15.2. Folgen mehrere gleiche Werte aufeinander, bleibt das Signal auf einem festen Pegel und es findet kein Wechsel mehr statt. Wenn beispielsweise der Sender eine lange Folge von Nullen schickt, sieht der Empfänger für die entsprechende Zeitdauer einen konstanten Pegel.

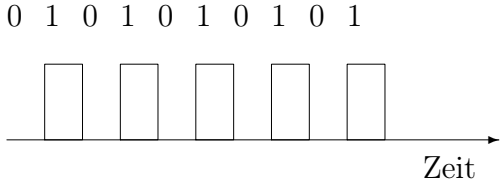


Abbildung 15.1: Folge von Bits mit Werte 0 und 1

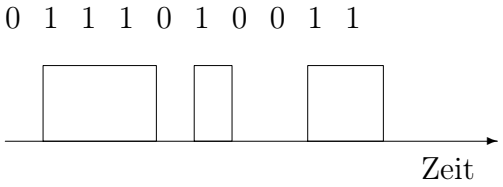


Abbildung 15.2: Allgemeine Folge von Bits

Dann kann es schwierig werden, zu zählen exakt wie viele Nullen geschickt wurden. Falls Sender und Empfänger keine gemeinsame Zeitbasis haben, können kleine Abweichungen in der Zeitmessung zwischen beiden dazu führen, dass der Empfänger zu wenige oder zu viele Nullen zählt. Im Normalfall – d. h. wenn keine zu langen Folgen mit konstanten Werten auftreten – kann der Empfänger kleine Abweichungen kompensieren, indem er die Signalwechseln nutzt, um sich immer wieder auf den Sender zu synchronisieren (Taktregenerierung).

Andererseits können Taktfehler weitreichende Folgen haben. Betrachtet man als Beispiel die Übertragung eines Stroms von Bytes. Sobald ein Bit verloren geht, verschiebt sich für alle nachfolgenden Werte die Zuordnung der Bits zu den Bytes, so dass sich komplett andere Bytewerte ergeben.

Es ist daher sinnvoll dafür zu sorgen, dass auch bei konstanten Werten Signalwechsel statt finden. Ein Ansatz dazu ist die Non-Return to Zero Inverted (NRZI) Kodierung. Dabei gilt die Regel, dass bei einer 1 der Pegel sich ändert, d. h. von low auf high oder von high auf low springt. Dadurch wird aus der Folge 0 1 1 1 1 die Signalfolge low high low high low. NRZI löst das Problem von vielen aufeinander folgenden Einsen, hilft aber nicht bei langen Folgen von Nullen.

Ein weitergehender Ansatz ist die Manchester–Kodierung. Man kann sich dabei die Übertragung mit doppelter Rate vorstellen, d. h. für jeden Wert stehen zwei Zeiteinheiten zur Verfügung. Dann überträgt man die Folge low–high für 0 und high–low für 1. So ist gewährleistet, dass für jeden übertragenen Wert mindestens ein Signalwechsel statt findet. Nachteil ist, dass jetzt für jeden Wert zwei Zeiteinheiten gebraucht werden, oder – umgekehrt gesprochen – in der gleichen Zeit nur halb so viele Werte übertragen werden können. Von den vier möglichen Signalfolgen werden nur noch zwei verwendet.

Es gibt eine ganze Reihe weiterer Schemata für die Kodierung, die auf diesen Ideen beruhen. Der verbreitete 4B5B-Code überträgt 4 bit mit 5 Signalwerten. Die Zuordnung ist dabei so gewählt, dass nie mehr als 3 Nullen hintereinander kommen. Zusammen mit einer NRZI Kodierung ermöglicht der Code bei nur geringem Mehraufwand (20%) eine deutliche Verbesserung der Taktregenerierung.

In diesem Zusammenhang wird oft das Maß Baud (Bd) benutzt. Die Einheit

ist benannt nach dem französischer Ingenieur Baudot[1], der den Baudot-Code – den Vorläufer des ASCII-Code – entwickelte. Die Anzahl der pro Sekunde erfolgten Übertragungen bezeichnet man als Baudrate. Wenn mit jeder Übertragung genau ein Bit dargestellt wird, entspricht die Baudrate dem Maß bit/s. Anderseits sind bei der Manchester–Kodierung zwei Übertragungen pro Bit notwendig. Damit ist die Baudrate doppelt so hoch wie die Anzahl der Bits pro Sekunde. Können bei der Übertragung mehr als nur zwei Zustände unterschieden werden, ist umgekehrt die Baudrate kleiner als die Zahl der Bits pro Sekunde.

15.2 Rahmenerstellung

Die Übertragung von Bits ist stets auch bis zu einem gewissen Grad fehleranfällig. Neben den bereits besprochenen Synchronisationsfehlern können auch einzelne Bits falsch dekodiert werden. Solche Fehler sind in einem kontinuierlichen Datenstrom nicht zu erkennen. Es ist daher wichtig, den Datenstrom zu strukturieren und durch Zusatzinformationen unempfindlich gegen Störungen zu machen. Weiterhin ist bei Rechnernetzen die kontinuierliche Übertragung eher die Ausnahme. Bei typischen Anwendungen (z. B. surfen im Internet) wechseln sich Phasen von Datenübertragungen mit – aus Sicht des Kommunikationskanals – Ruhephasen ab. Daher muss der Beginn einer neuen Übertragung kenntlich gemacht werden.

Aus diesen Gründen werden mehrere Bits oder Bytes zusammengefasst, um entsprechenden Informationen ergänzt und als Einheit übertragen. Man spricht dann von Rahmen oder Frames. Dabei unterscheidet man zwischen Byte-orientierten und Bit-orientierten Protokollen, je nach dem ob Bytes oder Bits als kleinste Bausteine für einen Frame benutzt werden.

Bei klassischen Kommunikationsanwendungen wie Telefon hat man eine feste Übertragungsrate, so dass man den Datenstrom gut in eine Reihe gleich grosser Rahmen aufteilen kann. Wenn eventuell am Ende der Übertragung ein Rahmen nicht mehr komplett gefüllt wird, kann problemlos mit Nullen aufgefüllt werden. Die dadurch entstehende kleine Pause wird nicht stören. Demgegenüber ist die Rechnerkommunikation durch starke Schwankungen in der Übertragungsrate gekennzeichnet. Außerdem ist das Auffüllen des letzten Rahmens beispielsweise bei dem Kopieren einer Datei nicht akzeptabel. Daher werden in der Regel Rahmenformate für eine variable Anzahl von Daten eingesetzt.

Dazu muss der Sender dem Empfänger mitteilen, wie viele Daten im aktuellen Rahmen enthalten sind. Zwei Verfahren werden dazu benutzt: Verwendung einer Endemarkierung oder explizite Angabe der Anzahl der Daten. Zusammengefasst kann man daher folgende Methoden der Rahmenbildung unterscheiden:

- Feste Anzahl von Nutzdaten
- Variable Anzahl von Nutzdaten

[1] Jean-Maurice-Émile Baudot (1845-1903)

- Markierung des Endes der Nutzdaten
- Angabe der Anzahl der Nutzdaten im aktuellen Frame

Im folgenden werden am Beispiel von Byte-orientierten Protokollen die beiden Verfahren zur Erstellung von Rahmen mit einer variablen Anzahl von Datenbytes diskutiert.

15.2.1 Markierungszeichen

Bei der Übertragung von Texten verwendet man die im ASCII–Standard definierten Steuerzeichen zum Aufbau von Rahmen. Ein einfacher Rahmen könnte dann folgende Form haben (Beispiel nach [Tan00]):

DLE	STX	Daten	DLE	ETX

Die zeitliche Abfolge ist bei dieser Darstellung von links nach rechts, d. h. das am weitesten links stehende Element wird zuerst gesendet. Das Zeichen DLE (Data Link Escape, ASCII Code 127) markiert den Beginn einer Steuersequenz. Anfang und Ende werden durch STX (Start of TeXt, ASCII Code 2) beziehungsweise ETX (End of TeXt, ASCII Code 3) gekennzeichnet. Eingebettet zwischen diesen beiden Steuersequenzen liegen die Nutzdaten.

Dieses Verfahren funktioniert bei der Übertragung von Textdaten problemlos. Bei beliebigen Binärdaten – Gleitkommazahlen, Bilder, oder ähnliches – besteht allerdings die Möglichkeit, dass die Endsequenz DLE ETX auch in den Nutzdaten auftritt:

DLE	STX	...	DLE	ETX	...	DLE	ETX

Der Empfänger würde dann vorzeitig auf Ende des Rahmens erkennen. Ein ähnliches Problem ist bei der Programmierung in C und verwandten Sprachen die Darstellung des Zeichens " als Teil einer Zeichenkette. Eine Lösung besteht in der Einfügung eines zusätzlichen Zeichens. Man kann etwa in den Daten vor jedem DLE ein zweites DLE einfügen:

DLE	STX	...	DLE	DLE	ETX	...	DLE	ETX

Durch dieses so genannte Zeichenstopfen (engl. character stuffing) wird sicher gestellt, dass innerhalb des Datenfeldes nie ein einzelnes DLE erscheint. Vielmehr folgen stets 2 oder allgemein betrachtet eine gerade Anzahl von DLE Zeichen aufeinander. Der Empfänger muss dann nur aus jedem Paar von DLE Zeichen eines entfernen, um die gesendete Bytefolge zu rekonstruieren.

15.2.2 Zeichenzähler

Die zweite Methode zur Rahmenerzeugung beruht auf der Angabe der Anzahl der nachfolgenden Daten. Im einfachsten Fall wird ein Feld mit einem Zähler vor den Daten eingefügt:

In dem Anzahlfeld steht, wie viele Bytes Daten in diesem Rahmen noch kommen werden. Die Größe des Nutzfeldes ist dann durch die Größe des Anzahlfeldes beschränkt. Steht nur ein Byte für die Größenangabe zur Verfügung, so kann das Nutzfeld maximal 256 Bytes lang sein. In dieser einfachsten Form ist die Übertragung recht fehleranfällig. Tritt ein Fehler im Anzahlfeld auf, berechnet der Empfänger die Rahmengröße falsch. Es kann unter Umständen lange dauern, bis der Empfänger sich wieder auf einen korrekten Rahmenanfang synchronisiert. Man verwendetet daher in der Regel den Zeichenzähler nur in Kombination mit anderen Methoden.

15.3 Fehlererkennung

Im Allgemeinen ist die Übertragung über einen Kanal nicht fehlerfrei. Ein Rahmen kann daher fehlerhafte Daten enthalten. Wie oben beschrieben, können Fehler im Extremfall sogar dazu führen, dass das Rahmenende nicht richtig erkannt wird und damit auch der Empfang der nachfolgenden Rahmen gestört wird. Aus diesen Gründen ist es wichtig, dass der Empfänger durch Analyse des Rahmens fest stellen kann, ob Übertragungsfehler auftraten.

Bei optimaler Ausnutzung der Übertragungskapazität entspricht jedem Bitmuster ein gültiges Zeichen. Durch Übertragungsfehler werden dann aus gültigen Zeichen auch wieder gültige Zeichen. Der Empfänger kann dann nicht erkennen ob Übertragungsfehler vorliegen. Um eine Fehlererkennung zu ermöglichen, muss die Übertragung so erweitert werden, dass Fehler zu ungültigen Zeichen (oder Rahmen) führen. Zu diesem Zweck fügt der Sender zusätzliche Informationen (Redundanz) in den Rahmen ein.

Eine Anwendung dieses Prinzips sind die Prüfzeichen bei wichtige oder leicht zu verwechselnden Informationen wie Kontonummer, Kreditkarten-Nummern oder die Internationale Standard Buchnummer (ISBN). Diese Nummern enthalten ein zusätzliches Zeichen, das nach einem vorgegebenen Algorithmus aus den anderen Stellen berechnet wird.

Auch in der sprachlichen Kommunikation setzten wir in schwierigen Fällen Redundanz ein. Ein Beispiel ist das Buchstabieren bei der Übermittlung von Namen. Noch mehr Redundanz kann man durch die Verwendung von Buchstabier–Alphabeten *A wie Anton, B wie Berta ...* erreichen. Die Eigennamen sind als längere Einheiten besser zu erkennen als die einzelnen Buchstaben.

Oft können nicht erkannte Fehler in der unteren Schicht auf einer höheren Schicht noch erkannt werden. Bei der Übermittlung einer Textdatei führen Übertragungsfehler mit einiger Wahrscheinlichkeit dazu, dass erkennbar falsche Wörter resultieren. Diese Fehlererkennung ist aber nur aufgrund der Redundanz der Sprache möglich. Nicht jede Buchstabenfolge ergibt ein Wort in der jeweiligen Sprache, so dass Sprache auch ein entsprechendes Maß an Redundanz enthält.

Die Zusatzinformation als Sicherung gegen Fehler verringert die Netto-Übertragungsrate. Daraus ergibt sich die Herausforderung, mit möglichst wenig Zusatzinformation eine zuverlässige Fehlererkennung zu ermöglichen. Man unterscheidet dabei:

- Fehlererkennung: Der Empfänger kann erkennen, dass ein Rahmen fehlerhaft übertragen wurde.

- Fehlerkorrektur: Der Empfänger kann – bis zu einem gewissen Grad – erkannte Übertragungsfehler auch korrigieren.

Verwendet man nur eine Fehlererkennung, muss der Empfänger im Fehlerfall den Sender auffordern, den Rahmen erneut zu schicken. Will man eine Fehlerkorrektur realisieren, muss mehr Sicherheitsinformation in den Rahmen eingebaut werden. Trotzdem wird man auch damit nur einen Teil der Fehler reparieren können. Enthält ein Rahmen zuviele Fehler, kann die ursprüngliche Information nicht wieder hergestellt werden.

Welches Verfahren effizienter in Hinblick auf die Sicherheit und die insgesamt erreichbare Netto-Datenrate ist hängt von dem Übertragungskanal ab. Dabei spielen sowohl die Auftrittswahrscheinlichkeit für einen Bitfehler als auch die statistische Verteilung der Fehler eine Rolle. Bei vielen Kanälen treten Fehler nicht gleichmäßig verteilt auf sondern in Gruppen (Bursts). Wenn der Kanal in einem schlechten Zustand ist – z. B. bei einer Funkverbindung in einem Schatten hinter einem Hochhaus – kommt es zu sehr viel mehr Fehlern als im Normalzustand. Dann scheitert die Fehlerkorrektur und die entsprechenden Rahmen müssen verworfen werden.

Insgesamt ist die Fehlererkennung eine komplexe Aufgabe. Die eingesetzten Methoden erfordern ein tiefes Verständnis von Statistik und Informationstheorie. Die Informationstheorie erlaubt auch Aussagen zu den theoretischen Grenzen für die Nachrichtenübertragung über einen gegebenen Kanal. Im folgenden werden ohne Anspruch auf mathematische Tiefe anhand von zwei wichtigen Beispielen einige grundsätzlichen Prinzipien der Fehlererkennung dargestellt. Bei diesen beiden Verfahren bleiben die Daten unverändert und werden um zusätzliche Sicherungsdaten ergänzt.

15.3.1 Parität

Eine weit verbreitete Methode zur Fehlererkennung ist das Einfügen von Paritätsbits. Dazu zählt man die Bits mit Wert 1 in dem Datenwort. Dann fügt man ein weiteres Bit hinzu, so dass bei der so genannten geraden Parität (*even parity*) insgesamt eine gerade Anzahl von Einsen entsteht. Bei ungerader Parität (*odd parity*) ist die Anzahl der gesetzten Bits ungerade. Für das ASCII Zeichen C als Beispiel erhält man bei gerader Parität:

Zeichen	Bitmuster	Paritätsbit
H (0x48)	100 1000	0
A (0x41)	100 0001	0
L (0x4C)	100 1100	1
L (0x4C)	100 1100	1
O (0x4F)	100 1111	1
Längsparität	100 0110	1

Abbildung 15.3: Zweidimensionale Parität

Zeichen	Bitmuster	Paritätsbit
C (0x43)	100 0011	1

Das zusätzliche Paritätsbit eignet sich gut für ASCII–Zeichen. Wenn man sich auf den 7-Bit Zeichensatz beschränkt, kann das 8. Bit als Paritätsbit verwendet werden. Die Parität hat weiterhin den Vorteil, dass die Prüfung sich in Hardware aufwandsgünstig mit hinter einander geschalteten XOR-Elementen realisieren lässt.

Von den mit 8 Bit möglichen 256 Zeichen bleiben nach bei der Paritätsprüfung nur noch 128 gültige Zeichen übrig. Dabei gilt, dass sich zwei gültige Zeichen in mindestens zwei Bitpositionen unterscheiden. Ändert man nur ein Bit, so resultiert ein ungültiges Zeichen. Man sagt, der Hamming[2]-Abstand beträgt 2. Allgemein bedeutet ein Hamming-Abstand n, dass sich zwei Zeichen in mindestens n Positionen unterscheiden.

Mit einem Paritätsbit wird ein Fehler (oder allgemein eine ungerade Anzahl von Fehlern) innerhalb eines Bytes detektiert. Zwei Fehler kompensieren sich gegenseitig und führen wieder zu einem korrekten Paritätsbit. Betrachtet man einen Block – d. h. eine Anzahl aufeinander folgender Zeichen – gemeinsam, kann man die Sicherheit deutlich erhöhen. Dazu führt man zu der beschriebenen Parität pro Zeichen eine zweite Parität pro Position in Zeitrichtung ein. Zur Unterscheidung zur einfachen (eindimensionalen) Parität spricht man auch von zweidimensionaler Parität. Abbildung 15.3 zeigt die zweidimensionale Parität am Beispiel der Zeichenfolge `HALLO`. Die Parität in Zeilen– und Spaltenrichtung nennt man Quer– und Längsparität.

Ein einzelner Fehler macht sich sowohl in einer Zeile als auch Spalte bemerkbar und kann damit korrigiert werden. Zwei Fehler – selbst in unterschiedlichen Zeilen und Spalten – können nicht mehr korrigiert werden. Allerdings lassen sich die Fehler relativ gut lokalisieren und es gibt nur zwei Möglichkeiten, wo die Fehler sein können. Liegen die beiden Fehler zusätzlich in einer Zeile stimmen alle Querparitäten. Die Fehler werden zwar durch die Längsparität detektiert, können aber nicht mehr ausgebessert werden. Allgemein lassen sich alle 3 Bit Fehler erkennen und auch die meisten 4 Bit Fehler. Nur wenn die 4 Fehler ge-

[2]Richard Wesley Hamming, amerikanischer Mathematiker, 1915 - 1998

nau an den passenden Kreuzungspunkten liegen, stimmen wieder alle Quer– und Längsparitäten.

Die Paritätsprüfung zählt die Einsen und prüft, dass sich in Summe eine gerade Anzahl ergibt. Die Übertragung umfasst einen Block Daten und eine Anzahl von Paritätsinformationen:

Datenbits	Paritätsbits

Der Empfänger kann dann durch die Prüfoperation auf dem Gesamtblock die Integrität testen. Dieses allgemeine Schema Daten – Prüfinformation – Prüfoperation lässt sich leicht verallgemeinern, indem man andere Prüfoperationen nutzt. In Internet Protokollen wird ein Verfahren mit einer Prüfsumme angewandt. Die Prüfoperation ist dabei eine Addition im Einerkomplement. Als Prüfinformation wird das Ergebnis dieser Addition an die Daten angehängt. Der Empfänger führt dann ebenfalls die Addition aus und vergleicht das Ergebnis mit der Prüfsumme. Dieses Verfahren ist einfach zu realisieren, bietet aber nicht sehr viel Schutz. Wesentlich mächtiger ist das Verfahren der zyklische Redundanzprüfung.

15.3.2 Zyklische Redundanzprüfung

Bei dem Verfahren der zyklische Redundanzprüfung (*Cyclic Redundancy Check*, CRC) ist im Prinzip die Prüfoperation eine Division. Betrachten wir zunächst ein Beispiel mit Dezimalzahlen. Übertragen werden soll eine große Dezimalzahl wie etwa 35628828297292. Weiterhin wählt man einen Divisor D aus. Nimmt man dafür beispielsweise den Wert 17 und führt die Division aus so erhält man bei ganzzahliger Division (Modulo–Operator % in C) einen Rest von 8. Der Sender kann damit zu der großen Zahl als Prüfwert die 8 senden. Der Empfänger rechnet dann zur Kontrolle $(35628828297292 - 8)\%17$. Bleibt bei der Division ein Rest übrig, so liegt ein Übertragungsfehler vor.

An diesem einfachen Beispiel werden einige Grundeigenenschaften deutlich. Zunächst sind nicht alle Divisoren gleich geeignet. Beispielsweise wäre 10 ein schlechter Divisor, da dann nur Fehler an der letzten Stelle sich auswirken würden. Weiterhin wird ein größerer Divisor einen besseren Schutz bieten. Allerdings muss dann entsprechend mehr Platz für den Rest vorgehalten werden. Der Rest kann maximal den Wert $D - 1$ annehmen. Der Divisor selbst hat sinnvollerweise einen festen Wert und braucht dann nicht mehr übertragen zu werden.

Die Übertragung dieses Ansatzes auf Binärdaten beruht auf der Interpretation von Bitfolgen als Polynomen. Für eine Bitfolge $abcdef$ schreibt man

$$M(x) = a \times x^5 + b \times x^4 + c \times x^3 + d \times x^2 + e \times x^1 + f \times x^0 \tag{15.1}$$

Dabei genügt es, die Terme mit Einsen anzugeben. Für zum Beispiel die Bitfolge 100101 erhält man dann

$$M(x) = x^5 + x^2 + x^0 \tag{15.2}$$

Die Prüfbits werden durch eine Division von solchen Polynomen bestimmt. CRC benutzt eine polynomiale Modulo-2 Arithmetik. In dieser Arithmetik reduziert sich die Subtraktion auf eine XOR–Operation. Die Division lässt sich – wie bei der üblichen Division – durch fortgesetzte Subtraktion ausführen.

Sei als Beispiel x^3+x+1 das Divisor-Polynom und damit 1011 das zugehörige Bitmuster. Übertragen werden soll die Nachricht 110101101. Zunächst wird die Nachricht entsprechende der höchsten Potenz im Divisor um 3 Bits erweitert: 110101101*000*. Genau in diese zusätzlichen Bits kommen später die Prüfbits. Der erste Schritt in der Division ist dann

1	1	0	1	0	1	1	0	1	0	0	0
1	0	1	1								
	1	1	0								

Das Ergebnis der Division wird nicht benötigt und daher nicht notiert. Dann wird die nächste Stelle von oben übernommen und der Divisor erneut subtrahiert:

1	1	0	1	0	1	1	0	1	0	0	0
1	0	1	1	↓							
	1	1	0	0							
	1	0	1	1							
		1	1	1							

Insgesamt ergibt sich

1	1	0	1	0	1	1	0	1	0	0	0
1	0	1	1	↓							
	1	1	0	0							
	1	0	1	1	↓						
		1	1	1	1						
		1	0	1	1	↓					
			1	0	0	1					
			1	0	1	1	↓	↓			
				0	1	0	0	1			
					1	0	1	1	↓	↓	
							1	0	0	0	
							1	0	1	1	↓
									1	1	0

Bei der Division bleibt ein Rest von 110. Zieht man diesen Rest von der erweiterten Zahl ab, so ist die neue Zahl ohne Rest durch den Divisor teilbar. Da Subtraktion einer XOR–Operation entspricht, erhält man 110101101*110*. Diese um die 3 Prüfbits erweiterte Bitfolge wird gesendet. Der Empfänger führt dann ebenfalls die Division durch. Falls dabei ein Rest übrig bleibt, liegt ein Übertragungsfehler vor.

Wie bei dem Beispiel mit den Dezimalzahlen gibt es auch für die Polynom-

division mehr oder weniger gut geeignete Divisor–Polynome. In der Literatur – beziehungsweise in den entsprechenden Spezifikationen – findet man geeignete Polynome. Man bezeichnet die Polynome in diesem Zusammenhang auch als Generator–Polynome. Einige davon sind in Tabelle 15.1 angegeben. Die Zahl nach dem Namen gibt an, wie viele Prüfbits jeweils genutzt werden. Details der Implementierung zusammen mit einer Realisierung in der Programmiersprache C findet man beispielsweise in [PTVF92].

CRC	Divisor
CRC-8	$x^8 + x^2 + x^1 + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$

Tabelle 15.1: Beispiele für gebräuchliche CRC–Polynome

Die Prüfverfahren schützen in allen Fällen, in den die Anzahl der Bitfehler kleiner ist als die Anzahl der Prüfbits. Darüber hinaus werden die „meisten“ anderen Fehler erkannt.

15.4 Sichere Übertragung von Rahmen

Wenn der Sender einen Rahmen abgeschickt hat, dauert es eine Weile bis der Empfänger den Rahmen erhalten hat und die Korrektheit festgestellt bzw. wieder hergestellt hat. Der Empfänger kann dann den Empfang quittieren indem er dem Sender eine Bestätigung schickt. Insgesamt ergibt sich folgender Ablauf:

1. Sender schickt Rahmen
2. Empfänger wartet bis der Rahmen vollständig angekommen ist
3. Empfänger prüft Korrektheit
4. Empfänger schickt Bestätigung
5. Sender erhält Bestätigung

Dieser Zusammenhang ist in Bild 15.4 mit einem Zeitstrahl graphisch dargestellt. R1 bezeichnet den ersten Rahmen und ACK die Bestätigung (Acknowledgment). In diesem Bild ist beim Empfänger eine kleine Zeitspanne nach Erhalt des Rahmens zur Prüfung eingetragen. Diese Zeit wird zwar im Prinzip immer benötigt. Im folgenden wird aber zur Vereinfachung der Darstellung diese Zeit nicht mehr explizit eingetragen.

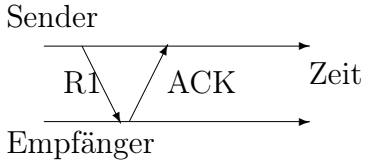


Abbildung 15.4: Zeitlicher Ablauf bei Bestätigung eines Rahmens R1

15.4.1 Stop–and–Wait Algorithmus

Eine direkte Implementierung des beschriebenen Verfahrens ist der Stop–and–Wait Algorithmus. Der Sender schickt einen Rahmen und wartet auf die Bestätigung. Erhält er innerhalb einer gewissen Wartezeit keine Bestätigung, so geht er davon aus, dass der Rahmen verloren gegangen ist und schickt ihn erneut. Die Wartezeit bezeichnet man als *Timeout*. Damit ergibt sich Bild 15.5.

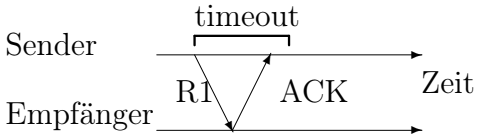


Abbildung 15.5: Zeitlicher Ablauf bei Stop–and–Wait Algorithmus

Der Timeout soll dem Empfänger genügend Zeit zur Bestätigung lassen. Andererseits soll er aber so kurz wie möglich sein, um die Wartezeit bei einem verloren Rahmen zu minimieren. Ein Problem tritt auf, wenn die Bestätigung zu spät – oder gar nicht – ankommt. Beim Sender ist mittlerweile der Timeout abgelaufen und in der Annahme, dass der Rahmen nicht korrekt empfangen wurde, schickt der Sender den Rahmen erneut. Der Empfänger hat andererseits den Rahmen bestätigt und erwartet nun bereits den nächsten Rahmen. Ohne weiter Maßnahmen würde er den zum zweiten Mal geschickten Rahmen bereits als nächsten Rahmen interpretieren.

Man benötigt daher eine Kennung für die einzelnen Rahmen. Beim Stop–and–Wait Algorithmus verwendet man in der Regel die einfachste Nummerierung mit 0 und 1. Falls ein Rahmen fälschlicherweise erneut geschickt wird, erkennt dies der Sender an der Nummer des Rahmens. Er kann daher den Rahmen ignorieren, muss aber natürlich eine Bestätigung schicken. Den vollständigen Ablauf zeigt Bild 15.6. In diesem Beispiel geht die Bestätigung für den Rahmen 0 verloren. Der Sender schickt daher nochmals den Rahmen 0. Diesmal wird die Bestätigung ordnungsgemäß zugestellt und der Sender geht zum nächsten Rahmen, der die Kennung 1 erhält.

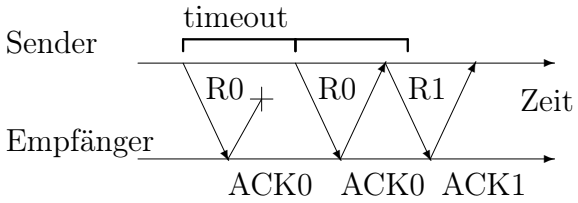


Abbildung 15.6: Stop–and–Wait Algorithmus mit Rahmennummer, Verlust einer Bestätigung.

15.4.2 Sliding–Window Algorithmus

Im Stop–and–Wait Algorithmus gibt es lange Pausen, in denen der Sender auf eine Bestätigung wartet. Daher wird der Kanal nur schlecht ausgenutzt. Eine bessere Auslastung wird erreicht, indem der Empfänger bereits während der Wartezeit weitere Rahmen schickt. Zu jedem Zeitpunkt sind dann mehrere Rahmen auf der Leitung unterwegs und der Sender wartet auf entsprechend viele ausstehende Bestätigungen.

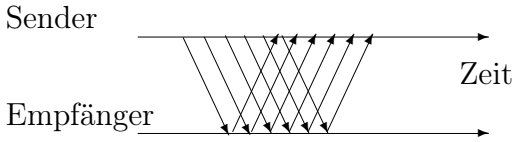


Abbildung 15.7: Sliding–Window Algorithmus

Der Sender nummeriert die Rahmen, so dass sie individuell bestätigt werden können. Die maximale Anzahl von gleichzeitig auf dem Weg befindlichen Rahmen – das Sendefenster (*Send Window Size*, SWS) – kann bei bekannter Leitungscharakteristik an die RTT der Leitung angepasst werden. Im Idealfall wird das Fenster so gewählt, dass der Timeout für den ersten Rahmen unmittelbar nach Abschicken des letzten Rahmens im Sendefenster abläuft. Der Sender muss alle Rahmen im Sendefenster vorrätig halten, um sie bei Bedarf nochmals senden zu können. Kommt die Bestätigung für den ersten Rahmen an, kann der Empfänger diesen Rahmen löschen und das Sendefenster weiter schieben. Anderenfalls sendet er diesen Rahmen erneut.

Im allgemeinen wird es vorkommen, dass zwar ein Rahmen verloren geht, aber die nachfolgenden Rahmen trotzdem ankommen. Eine gute Kanalauslastung erreicht man, wenn jeder angekommene Rahmen bestätigt wird (selektive Bestätigung, engl. *selective acknowledgements*). Allerdings wird dann der Aufwand in der Verwaltung des Sendefensters und des Empfangsfensters kompliziert. Geht beispielsweise ein bestimmter Rahmen mehrfach verloren, dann muss der Empfänger sehr viele neue Rahmen speichern, um diesen alten Rahmen an der richtigen Stelle

einfügen zu können. Wenn erforderlich, kann der Empfänger den Sender bremsen, indem er Rahmen, die zu weit vor dem letzten fehlenden Rahmen liegen, nicht bestätigt oder zumindest die Bestätigung verzögern.

Die Arbeit des Senders wird wesentlich vereinfacht, wenn der Empfänger die aus der Reihe angekommenen Rahmen nicht bestätigt. Angenommen Rahmen 4 geht verloren oder wird stark verzögert. Der Empfänger erhält statt dessen bereits die folgenden Rahmen 5 und 6. Er bestätigt diese Rahmen aber nicht sondern wartet auf den Rahmen 4 – entweder das verspätete Original oder die wieder gesendete zweite Version. Sobald der Rahmen 4 ankommt, bestätigt der Empfänger den Rahmen 6. Bei entsprechendem Protokoll erkennt der Sender daraus, dass alle Rahmen bis zur Nummer 6 gut angekommen sind (kumulative Bestätigung). Er kann dann das Sendefenster auf die Position 7 weiter schieben.

Wenn tatsächlich Rahmen verloren gehen, wird bei diesem Verfahren der Kanal nicht optimal ausgenutzt. Der Sender schickt während des Wartens nach dem wiederholten Rahmen unnötigerweise auch die nachfolgenden Rahmen zum zweiten Mal. Andererseits ist in dieser Form die Verwaltung der Rahmen in Sende– und Empfangsfenster recht einfach. Die Nummer eines Rahmen (Sequenznummer) ist eine zusätzlich zu übertragende Information. Aus Effizienzgründen sollten möglichst wenige Bits für dies Nummer verbraucht werden. Die Sequenznummer wird daher auf einen relativ kleinen Bereich beschränkt. Sobald das Maximum erreicht ist, springt der Zähler wieder auf 0 zurück. Es muss allerdings gewährleistet sein, dass die Rahmen noch eindeutig identifiziert werden können.

15.5 Übungen

Übung 15.1 *Die Zeichenkette* `Friedberg` *soll übertragen werden. Zur Sicherung gegen Übertragungsfehler werden Paritätsinformationen eingebaut. Ergänzen Sie in der Tabelle die Bits für Querparität sowie das zusätzliche Byte für die Längsparität. Verwenden Sie in beiden Fällen gerade Parität.*

Querparität	*Binär-Darstellung*	*Zeichen*
	100 0110	*F*
	111 0010	*r*
	110 1001	*i*
	110 0101	*e*
	110 0100	*d*
	110 0010	*b*
	110 0101	*e*
	111 0010	*r*
	110 0111	*g*
		Längsparität

Übung 15.2 *Wie groß ist der Hamming-Abstand bei zweidimensionaler Parität?*

Übung 15.3 *Berechnen Sie für das Divisor-Polynom x^3+x^2+1 die Prüfzeichen zu der Nachricht 101011011.*

Übung 15.4 *Die Prüfziffer der Internationalen Standard Buchnummer ISBN wird nach folgendem Algorithmus berechnet: Zunächst wird aus den 9 Ziffern der eigentlichen Kennung eine gewichtete Summe bestimmt. Dazu wird die erste Ziffer der ISBN mit 10 multipliziert, die zweite mit 9, die dritte mit 8 usw. Die Produkte werden addiert. Für die Summe wird der Rest bei Division durch 11 bestimmt (Modulus 11). Dieser Rest wird als Prüfziffer angehängt. Der Sonderfall Rest 10 wird durch ein X als Prüfziffer markiert. Implementieren Sie diesen Prüfalgorithmus beispielsweise als C-Programm oder in Excel. Testen die Korrektheit an Hand einiger Beispiele.*

Übung 15.5 *Das nachfolgende C-Programm simuliert einen Kanal mit Bit-Fehlern. Dazu werden aus einer Textdatei* `test.txt` *nacheinander alle Zeilen gelesen. Aus jeder Zeile wird ein Rahmen erstellt. In der Grundversion wird dabei lediglich die Zeichenkette kopiert. Der Rahmen wird dann an die Kanalsimulation übergeben. Dort werden nach einer vorgegebenen Bitfehler-Wahrscheinlichkeit Übertragungsfehler in den Rahmen eingefügt. Aus dem Rahmen wird dann wieder der Text extrahiert und in eine Datei mit dem Namen* `uebertragen.txt` *geschrieben.*

1. *Untersuchen Sie mit der Grundversion den Einfluss der Bitfehler auf die Rahmenfehler, d. h. die Anzahl der fehlerhaften Rahmen (Zeilen). Erstellen Sie eine graphische Darstellung der Abhängigkeit zwischen der Häufigkeit von Bitfehlern und Rahmenfehlern.*

2. *Fügen Sie in den vorgesehenen Funktionen eine Paritätsprüfung ein. Wie wirken sich Quer- und Längsparität aus? Wie oft müssen Rahmen wiederholt werden?*

3. *Integrieren Sie eine CRC-Prüfung. (Hinweis: eine Implementierung ist auf der Seite* `www.w3.org/TR/PNG-CRCAppendix.html` *des World Wide Web Consortiums angegeben.)*

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <math.h>

#define MAX_ZEICHEN 200

/* Kanalsimulation
 * Verarbeite len Daten aus buffer
```

```
 * Mit der Wahrscheinlichkeit prob wird ein Bit verfälscht.
 * Rückgabewert: Anzahl der veränderten Bits
 */
int kanal( char *buffer, int len, double prob ){
    int i, j, limit;
    unsigned int pos;
    int count = 0;

    /* Um auch kleine Wahrscheinlichkeiten behandeln zu können,
     * werden zwei Zufallszahlen erzeugt. Nur wenn beide kleiner
     * als Wurzel(prob) sind wird das jeweilige Bit gedreht. */
    limit = (int) (sqrt(prob) * RAND_MAX);
    for( i=0; i<len; i++ ) {    /* über alle Zeichen */
        pos = 1;                /* Bit-Maske */
        for( j=0; j<8; j++ ) {  /* über alle 8 bits */
            if( rand() < limit && rand() < limit) {
                /* Bit mittels XOR umdrehen */
                buffer[i] ^= pos;
                ++count;
            }
            pos <<= 1;  /* Maske für nächstes Bit */
        }
    }
    return count;
}

void bildeRahmen( char *rahmen, char *zeile ) {
    /* Hier wird ein Rahmen gebildet */
    strcpy( rahmen, zeile );
}

void leseRahmen( char *erhalten, char *rahmen ) {
    /* Hier wird der Text aus dem Rahmen geholt */
    strcpy( erhalten, rahmen );
}

int rahmenFalsch( char *rahmen ) {
    /* Bis jetzt sind keine Rahmen falsch */
    return 0;
}


void main(int argc, char* argv[])
```

```
{
    double prob = 0.025;   /* Bitfehler-Wahrscheinlichkeit */
    int len;
    int rahmenFehler = 0;
    int c_char = 0, c_fehler = 0;
    char zeile[MAX_ZEICHEN],
        rahmen[MAX_ZEICHEN],
        erhalten[MAX_ZEICHEN];
    FILE *fp_in, *fp_out;

    /* Initialisierung des Zufallszahlengenerators */
    srand( (unsigned)time( NULL ) );

    fp_in  = fopen( "test.txt", "r" );
    fp_out = fopen( "uebertragen.txt", "w" );
    while( fgets( zeile, sizeof zeile, fp_in ) ) {
        len = strlen( zeile );
        do {
            bildeRahmen( rahmen, zeile );
            c_fehler += kanal( rahmen, len, prob );
            c_char   += len * 8;
        } while( rahmenFalsch( rahmen ) );

        leseRahmen( erhalten, rahmen);
        if( strcmp( erhalten, zeile ) ) ++rahmenFehler;
        fprintf( fp_out, "%s", rahmen );
    }
    printf( "Anzahl von Bitfehlern:    %d, relativ %f%%\n",
        c_fehler, 100.*(double) c_fehler / c_char );
    printf( "Anzahl von Rahmenfehlern: %d\n",
        rahmenFehler );

}
```

Übung 15.6 *Sie wollen eine Datei von 3 MByte von Frankfurt nach New York (8000 km inklusive Umwegen) übertragen. Berechnen Sie die Dauer vom Beginn der Übertragung (Senden des ersten Zeichens) bis zum Ende (Empfang des letzten Zeichens) für eine Leitung mit 1 Mbit/s. Auf der Strecke befinde sich in der Mitte (bei 4000 km) ein Knoten, der die korrekte Übertragung prüft. Dazu nimmt er Pakete von jeweils 1024 Bytes an. Dann benötigt er 1 ms zur Paritätsprüfung jedes Pakets. Anschließend verschickt er das Paket. Sobald er das letzte Zeichen verschickt hat, ist er zum Empfang des nächsten Pakets bereit. Dann schickt er eine Bestätigung (8 Bytes) an den Sender, damit dieser das nächste Paket*

absenden kann. Welche Zeit benötigt nun die Übertragung? Hinweis: Zeichnen Sie ein Zeitstrahldiagramm mit Sender, Knoten und Empfänger.

*Verwenden sie die Lichtgeschwindigkeit c=300000 km/s als Ausbreitungsgeschwindigkeit. Rechnen Sie bei der Dateigröße mit 1 MByte = 1024*1024 Byte.*

Kapitel 16

Ethernet & Co

In diesem Kapitel wird die Funktionsweise von lokalen Netzen beschrieben. Unter den LAN–Technologien ist Ethernet derzeit am weitesten verbreitet. Eine aktuelle Technologie mit wachsender Bedeutung sind LANs auf der Basis von Funktnetzen. Daneben gibt es für Spezialanwendungen eine große Anzahl von weiteren Technologien.

16.1 Ethernet

Unter der Bezeichnung Ethernet fasst man eine Familie von Netztechnologien zusammen. Den verschiedenen Technologien gemeinsam ist die Adressierung, das Rahmenformat und die Zugriffskontrolle auf das Übertragungsmedium. Ethernet wurde in den siebziger Jahren im PARC (Palo Alto Research Center), dem Forschungslabor der Firma XEROX, entwickelt. Später wurde daraus in Zusammenarbeit mit den Firmen DEC und Intel ein offener Standard entwickelt. Dieser wiederum bildete die Grundlage für den offiziellen IEEE-802.3 Standard.

IEEE (sprich *Eye-triple-E*) ist die Abkürzung für Institute of Electrical and Electronics Engineers. Von dieser Organisation (www.ieee.org) werden neben vielen anderen Aktivitäten Standards entwickelt. Die dreistellige Zahl im Namen eines Standards bezeichnet das allgemeine Thema. Die Nummer 802 umfasst diverse Standards für *Local and Metropolitan Area Networks (LAN/MAN)*. Durch die Zahl hinter dem Punkt (*Part*) wird der Standard genauer bezeichnet. Die Gruppe 802.3 trägt den Titel *CSMA/CD Access Method*. Ein anderes Beispiel ist die Gruppe 802.11 *Wireless LANs*. Einzelne Standards innerhalb einer Gruppe werden durch angehängte Buchstaben oder Ziffern sowie eine Jahreszahl unterschieden (z. B. 802.11g-2003, 802.15.3-2003).

Der IEEE-802.3 Standard umfasst wie gesagt eine ganze Familie von Techniken. Die jeweiligen Namen werden nach dem Schema

<Datenrate><Übertragungsverfahren><Segmentlänge oder Kabeltyp>

gebildet. So bezeichnet 10BASE5 eine Variante mit 10Mbit/s Basisband-Übertragung und einer maximalen Segmentlänge von 500m.

16.1.1 Adressen

Die Adressen für Ethernet bestehen aus 6 Byte. Üblich ist die Angabe der einzelnen Bytes mit je 2 Hex-Zeichen, getrennt durch Doppelpunkte. Führende Nullen werden oft weggelassen. Das Programm `ipconfig` gibt die Adresse mit allen Nullen und Bindestrichen aus.
Beispiele:
8:34:3e:0:55:a2
00-D1-59-6B-88-63

Jeder Rechner oder genauer gesagt jede Netzwerkkarte in der Welt hat eine eindeutige Ethernet-Adresse. Die Adresse enthält einen Anfangsteil, der den Hersteller kennzeichnet. Das erste Bit hat eine besondere Bedeutung. Ist es gesetzt, handelt es sich nicht um eine individuelle Adresse sondern die Adresse einer ganzen Gruppe von Rechnern (*Multicast*). Der Extremfall ist eine Adresse, die nur Einsen enthält. Mit dieser Adresse (Broadcast) werden alle Rechner gleichzeitig angesprochen.

16.1.2 Rahmenformat

Die Rahmen im Standard IEEE-802.3 haben folgendes Format:

8 Byte	6	6	2	46-1500	4
Präambel	Zieladresse	Quelladresse	Länge	Daten	CRC

Die Präambel enthält im wesentlichen eine Folgen von abwechselnden Nullen und Einsen und erlaubt den Empfängern die Synchronisation auf den Rahmen. Anschließend folgen die Adressen von Ziel(en) und Sender. Bei IEEE-802.3 gibt das nächste Feld die Größe der darauf folgenden Daten an. Dabei gilt eine Mindestgröße von 46 Byte und eine maximale Größe von 1500 Byte. Im ursprünglichen Ethernet–Standard enthält dieses Feld eine Kennung für den Rahmentyp. Wenn man sich für die Typkennzeichnung auf Werte größer als 1500 beschränkt, können beide Rahmentypen ohne Verwechslungsgefahr parallel verwendet werden. Nach den Daten folgen noch die Prüfbits einer 32 Bit CRC.

Diese Darstellung ist etwas vereinfacht. Die Realität ist etwas komplizierter und es gibt mehrere Ausprägungen von 802.3-Rahmen. Auf die Details soll hier nicht näher eingegangen werden. Es sei lediglich erwähnt, dass die ersten drei Bytes der Nutzlast als Information für die so genannte *Logical Link Control* (LLC) gemäß IEEE-802.2 dienen.

16.1.3 Medienzugriff

Ethernet wurde für die gemeinsame Nutzung eines Mediums durch viele Stationen (Rechner) entwickelt (Mehrfachzugriffsnetz). Der erste Vorläufer war ein Funknetz zwischen den Hawaii–Inseln mit dem Namen Aloha. In diesem Fall war die Atmosphäre das gemeinsam genutzte Medium. Bei Ethernet war es ursprünglich ein Kabel, über das die Rechner in einer Bus-Topologie verbunden waren.

Der gemeinsame Zugriff (Medienzugriffssteuerung, engl. *Media Access Control* MAC) wird über die Methode CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) geregelt. Das Grundprinzip ist einfach: jede Station kann feststellen, ob die Leitung momentan frei ist (Carrier Sense). Wenn eine Station einen Rahmen senden möchte und die Leitung frei ist, beginnt sie sofort mit der Übertragung. Die Übertragungsdauer ist durch die maximale Datenmenge von 1500 Bits begrenzt. Alle Stationen hören permanent die Leitung ab. Entdecken sie einen Rahmen, der für sie bestimmt ist, übernehmen sie ihn.

Jede Station entscheidet selbständig, wann sie senden will. Damit kann es passieren, dass zwei Stationen in der Annahme, dass die Leitung frei ist, gleichzeitig anfangen zu senden. Die beiden Rahmen kollidieren auf der Leitung und werden damit unbrauchbar. Die Stationen erkennen die Kollision (*Collision Detection*), brechen die Übertragung ab und schicken ein 32 Bit langes Stausignal.

Nach einiger Zeit versuchen die Stationen erneut, ihre Rahmen zu schicken. Um zu verhindern, dass wieder beide Stationen gleichzeitig senden, wird die Wartezeit zufällig variiert. Im ersten Wiederholungsversuch beträgt die Wartezeit 51,2 μs und jede Station entscheidet zufällig, ob sie 0 μs oder 51,2 μs wartet. Falls es erneut zu einer Kollision kommt, verdoppelt sich die maximale Wartezeit und jede Station wartet zufällig für 0, 51,2, 102,4 oder 153,6 μs. Diese Verfahren der Verdopplung der Wartezeit (*exponential backoff*) wird bis zu einer gegebenen Maximalzahl von Versuchen angewandt. Im Versuch i wählt jede Station zufällig eine Wartezeit aus der Menge $0, 51,2, \ldots, (2^i - 1) * 51,2$ aus. In der Praxis wird allerdings i nicht größer als 10 und nach 16 erfolglosen Versuchen bricht der Netzwerkadapter mit einer Fehlermeldung ab. Mit dieser Strategie werden einerseits Sendekonflikte zwischen zwei oder einigen wenigen Stationen schnell geklärt. Andererseits löst sich auch in dem Fall, dass viele Stationen gleichzeitig senden wollen, nach einigen Verdoppelungen der Gesamtwartezeit der Stau rasch auf.

Aus der CSMA/CD Strategie ergeben sich Grenzwerte für die erlaubte maximale Laufzeit und die Rahmengröße. Betrachten wir dazu den extremen Fall zweier Stationen A und B an den entgegengesetzten Enden der Leitung. Station A sendet einen Rahmen. Der Rahmen erreicht nach der Latenz τ die Station B. Für B war bis dahin die Leitung frei. Im ungünstigsten Fall hat B daher unmittelbar bevor dieser Rahmen bei ihm eintrifft selbst mit der Übermittlung begonnen. B entdeckt sofort den Konflikt und bricht die Übertragung ab. Allerdings dauert es wiederum die Zeit τ bis der von B begonnene Rahmen bei A eintrifft. Damit

Tabelle 16.1: Kabeltypen für 10Mbit/s Ethernet

Bezeichnung	Typ	Maximale Länge	Bemerkung
10Base5	Dickes Koaxialkabel (ThickWire)	500m	Kabel wird für Anschlüsse (Transceiver) angebohrt
10Base2	Dünnes Koaxialkabel (ThinWire)	200m	Anschluss über T-Stück
10Base-T	Verdrilltes Paar (Twisted pair)	100m	Punkt–zu–Punkt Verbindungen
10Base-F	Glasfaser	2000m	

A seinen Rahmen noch zurück nehmen kann, darf A zu diesem Zeitpunkt noch nicht mit der Übertragung fertig sein. In Summe muss daher die Übertragung eines Rahmens stets mindestens die längste RTT 2τ innerhalb des Netzes dauern. Die minimale Übertragungsverzögerung muss die Beziehung

$$t_U = K/B \geq 2\tau \tag{16.1}$$

erfüllen. Bei der Übertragungsrate 10 Mbit/s und einer minimalen Rahmengröße von 64 Bytes beziehungsweise 512 bits ergibt sich als maximale RTT 51,2μs. Erhöht man die Übertragungsrate beispielsweise auf 100 Mbit/s so reduziert sich die maximale RTT bei gleichen Rahmen auf 5,12μs.

16.1.4 Physikalische Eigenschaften

In der ursprünglichen Form wird Ethernet über ein Koaxialkabel (10Base5) in Bus–Topologie betrieben. Das Kabel hat eine maximale Länge von 500 m. Direkt an den Kabeln werden Transceiver angebracht (Mindestabstant 2,5 m), von denen aus Kabeln zu den Rechnern führen. Als preisgünstiger Alternative wird dünneres Koaxialkabel (10Base2) verwendet, bei dem eine Abzweigung über ein T–Stück realisiert wird. Zwei Kabelsegmente können über einen Repeater verbunden werden.

Die moderne Form 10Base-T basiert auf verdrillten Kabelpaaren (*twisted pair*). Damit sind nur noch Punkt-zu-Punkt-Verbindungen möglich. Mehrere Stationen werden dann auf einen mehrwegigen Repeater (Sternkoppler oder *Hub* (engl. Nabe, Mittelpunkt)) geführt. Die einzelnen Kabeltypen sind in Tabelle 16.1.4 zusammen gestellt. Repeater und Hubs arbeiten auf Bitebene und leiten die Signale ohne Analyse der Adressen weiter. Dann bleibt – unabhängig von der technischen Realisierung – das Netz eine logische Einheit. Jede Nachricht wird über das gesamte Netz verteilt, so dass Kollisionen an beliebiger Stelle auftreten können. Das Netz bildet eine Kollisionsdomäne (collision domain).

Einen höheren Datedurchsatz erreicht man durch Trennung eines Netzes in mehrere Teilnetze, die durch vermittelnde Geräte wie Switches oder Router verbunden sind. Diese Geräte leiten Pakete nur an die Teilnetze weiter, in denen sich die Zielknoten befinden. Dadurch werden die anderen Teilnetze von unnötigem Verkehr frei gehalten. Der Durchsatz lässt sich dann durch geeignete Topologien weiter optimieren. So ist es sinnvoll, Stationen die viel miteinander kommunizieren, an ein gemeinsames Teilnetz anzuschließen. Weiterhin können Stationen mit sehr hohem Datenaufkommen einem eigenen Port am Switch erhalten, so dass ihnen die Bandbreite exklusiv zur Verfügung steht. Ein Leistungssteigerung erreicht man durch Full Duplex Betrieb mit getrennten Kanälen für die beiden Richtungen.

Aufbauend auf dem Standard für 10 Mbit/s wurden mit 100 Mbit/s Fast Ethernet und 1000 Mbit/s Gigabit Ethernet zwei Nachfolgetechnologie mit höheren Bandbreiten entwickelt. Die beiden Verfahren benötigen Punkt zu Punkt Verbindungen, wobei sowohl Twisted Pair als auch Lichtleiterkabel eingesetzt werden können. Die weitgehende Kompatibilität erleichtert die Migration bestehender Ethernet-Netze zu den höheren Datenraten.

16.1.5 Bewertung

Ethernet ist eine überaus erfolgreiche Technologie. Die Bandbreite wird zwar nicht optimal ausgenutzt, aber solange die Belastung nicht zu hoch ist (kleiner 30%), ist der Verlust durch die Kollisionen nicht störend. Vorteilhaft ist die Einfachheit von Installation und Betrieb. Es ist kein Problem, Stationen einzufügen oder aus dem Netz zu nehmen. Auch der Ausfall einzelner Stationen hat in der Regel keine negativen Auswirkungen auf den Netzbetrieb. Durch das robuste Protokoll gibt es auch kaum Situationen, in denen eine einzelne Station durch Fehlfunktionen das ganze Netz in Mitleidenschaft zieht.

Für normale Computer-Anwendungen wie etwa Fileservice ist Ethernet gut geeignet. Schwierig sind zeitkritische Anwendungen, die enge Anforderungen an die Reaktionszeit stellen. Darunter fallen auch Applikationen wie etwa Telefonverbindungen, bei denen Schwankungen in der Laufzeit sich störend bemerkbar machen. Durch den zufälligen Charakter des Medienzugriffs gibt es keine Garantie, dass ein Rahmen innerhalb einer vorgegebenen Zeit sein Ziel erreicht. In der Praxis kann man diese Probleme weitgehend vermeiden, indem man das Netzwerk nicht zu stark belastet.

Ethernet ist die bei weitem am häufigsten eingesetzte LAN Technologie. Die Tendenz geht zu immer höheren Datenraten, wobei aber stets noch die Kompatibilität mit den älteren Techniken gewahrt bleibt.

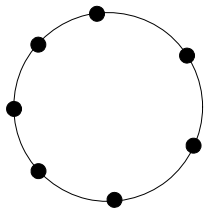


Abbildung 16.1: Netzwerk mit Ring-Topologie

16.2 Token Ring

Im Zugriffsverfahren CSMA/CD können die Rechner zu jeder Zeit versuchen, eine Nachricht zu senden sofern die Leitung frei ist. Die dadurch immer wieder auftretenden Kollisionen reduzieren den Durchsatz. Eine grundsätzliche Alternative ist ein geregelter Zugriff auf das Medium, bei dem zu jedem Zeitpunkt nur jeweils eine Station das Senderecht hat. Eine flexible Regelung basiert auf der Verwendung eines „Tokens“ (engl. für Zeichen, Erkennungsmarke). Nur die Station, die jeweils den Token besitzt, darf senden. Die Details des Medienzugriffs werden dann durch den Umlauf des Tokens sowie die Regeln zur Verwendung des Tokens bestimmt. Eine vergleichende Analyse der Leistungsfähigkeit von CSMA/CD und Token Ring findet man in [Sch88].

Die erfolgreichste Technik auf dieser Basis sind Token-Ring-Netze. Weit verbreitet ist der IBM–Token-Ring, der weitgehend identisch ist mit dem IEEE–Standard 802.5. Ein neuerer Standard ist FDDI (*Fiber Distributed Data Interface*).

Bei dem Token–Ring sind alle Stationen – zumindest logisch – in Form eines Ringes angeordnet (Bild 16.1). Die Daten werden stets nur in eine Richtung weiter gegeben. Jede Station hat genau eine Vorgängerin und eine Nachfolgerin. Die Stationen werden über spezielle Anschaltvorrichtungen an den Ring angeschlossen. Fällt eine Station aus oder wird sie vom Netz genommen, schließt ein Relais wieder den Ring.

16.2.1 Medienzugriff

Der Token ist eine spezielle Nachricht, die zunächst frei im Ring kreist. Möchte eine Station senden, so nimmt sie den Token aus dem Ring und schickt statt dessen ihre Nachricht in einem oder mehreren Rahmen. Ähnlich wie bei Ethernet enthält jeder Rahmen die Zieladresse. Die Rahmen laufen dann im Netz von Station zu Station. Die Zielstation kopiert die Nachricht, nimmt die Rahmen aber nicht aus dem Netz sondern lässt sie weiter kreisen. Erst die sendenden Station selbst nimmt die Nachricht wieder aus dem Netz. Anschließend gibt sie den Token weiter, so dass die nachfolgende Station die Gelegenheit zum Senden hat.

Für eine sichere Übertragung ist ein Protokoll mit zwei Bits im Rahmen vorgesehen. Die beiden Bits mit der Bezeichnung A und C werden vom Sender auf 0 gesetzt. Erkennt eine Station einen an sie gerichteten Rahmen, setzt sie das A–Bit. Bei erfolgreicher Kopie setzt sie zusätzlich auch das C–Bit. Der Sender kann auf diese Art erkennen, ob

1. der Empfänger aktiv ist
2. der Empfänger den Rahmen übernehmen konnte

Wenn eine Station den Token übernommen hat, kann sie ihre Daten verschicken. Optimal im Sinne der Netznutzung wäre es, diese Station ihre gesamten Daten direkt hintereinander schicken zu lassen. Insbesondere bei ansonsten geringen Aktivitäten könnte man die Wartezeit, bis der Token jeweils umgelaufen ist, einsparen. Dann hätten allerdings die anderen Stationen keine Chance, in der Zwischenzeit eigene Nachrichten zu versenden. Ein umfangreicher Transfer würde dann auch viele kleine Nachrichten blockieren. Daher wird die Zeit, für die eine Station den Token behalten darf (THT, *Token Hold Time*), beschränkt. Vor jedem weiteren Rahmen prüft die Station, ob sie diesen Rahmen noch innerhalb der THT schicken kann. Anderenfalls hält sie die Nachricht zurück und gibt den Token weiter. Im Standard 802.5 beträgt die THT 10 ms.

Damit kann man die obere Grenze für die Wartezeit, bis eine Station wieder an die Reihe kommt, angeben. Im ungünstigsten Fall (*worst case*) übernimmt jede Station den Token und nutzt ihre THT voll aus. Für den Umlauf des Tokens (TRT, *Token Rotation Time*) gilt damit

$$TRT \leq AnzahlStationen \times THT + RingLatenz \tag{16.2}$$

Das Protokoll gibt allen Stationen eine gleiche Chance zum Senden. Die maximale Dauer für eine Übertragung (ohne Wiederholung von Rahmen aufgrund von Fehlern) kann aus der oberen Abschätzung 16.2 für TRT bestimmt werden.

16.2.2 Netz–Überwachung

Der Betrieb eines Token-Ring-Netzes erfordert eine Überwachung oder Wartung während des Betriebs. Einige mögliche Fehlerfälle sind:

- Der Token geht aufgrund von Bitfehlern verloren
- Die Station im Besitz des Tokens stürzt ab
- Rahmen werden verfälscht und kreisen ewig im Ring
- Ein Rahmen „verwaist“ weil die Sendestation abstürzt, bevor sie ihn wieder aus dem Ring nimmt

Tabelle 16.2: Standards für WLAN

802.11	erster Standard von 1997	2 bis 3 Mbit/s, 2,4 GHz-Band
802.11b	Nachfolger von 802.11	bis 11 Mbit/s, 2,4 GHz-Band
802.11g	Nachfolger von 802.11b	bis 54 Mbit/s, 2,4 GHz-Band
802.11a	Nachfolger von 802.11b	bis 54 Mbit/s, 5 GHz-Band

Daher übernimmt eine der Stationen im Netz die Funktion eines Monitors. Jede Station kann Monitorstation werden. Es gibt ein wohl definiertes Protokoll, nach dem die Stationen bei der Inbetriebnahme des Rings oder nach Ausfall der Monitorstation aushandeln, welche Station diese Rolle übernimmt. Entdeckt die Monitorstation, dass nach der maximalen Umlaufszeit gemäß Gleichung 16.2 der Token immer noch nicht zurück gekommen ist, erzeugt sie einen neuen. Weiterhin kann sie anhand eines speziellen Monitorbits Rahmen erkennen, die mehrfach durch den Ring gereist sind. Diese Rahmen nimmt sie aus dem Ring.

16.3 Drahtlose LAN

Drahtlose Netze (*wireless LAN*, WLAN) gewinnen immer mehr an Bedeutung. Nur durch drahtlose Netze ist freie Beweglichkeit möglich. Dadurch können auch bewegliche Teilnehmer – sowohl Rechner (Laptops) als auch Geräte wie Scanner, Barcode-Leser, etc. – in ein Netz eingebunden werden. Der besondere Vorteil liegt im schnellen Netzaufbau. Dieser Vorteil kommt in Fällen zu tragen, in denen die Teilnehmer häufig wechseln oder einen schnellen, unkomplizierten Zugriff auf ein Netz brauchen. Typische ist der Einsatz als Netzzugang in öffentlichen Bereichen wie Flughäfen, Bahnhöfen oder Hotels (Hotspots).

Die wichtigste Standards für drahtlose Netze sind IEEE 802.11 und die daraus abgeleiteten Varianten. Eine Übersicht über die Standards dieser Familie gibt Tabelle 16.3. Derzeit aktuell ist 802.11b auch bekannt als Wi-Fi für *wireless fidelity*. Der Standard 802.11g ist kompatibel mit 802.11b. Erste Produkte dafür werden für Mitte 2003 erwartet. Als potentieller Nachfolger erscheint der Anfang 2003 verabschiedete Standard IEEE 802.16 aussichtsreich. Bei Entfernungen bis 50 km und Übertragungsleistungen bis zu 134 Mbit/s soll er eine großflächige Bereitstellung von drahtlosen Netzen ermöglichen. In diesem Zusammenhang spricht man bereits von Wireless Metropolitan Area Networks (WMAN).

IEEE 802.11 wurde für zwei Methoden der Funkübertragung und eine Art der Infrarot-Übertragung ausgelegt. Die Kollisionsvermeidung erfolgt ähnlich wie bei Ethernet. Allerdings ergeben sich aus der endlichen Reichweite der Signale spezielle Probleme. Anders als bei Ethernet erreicht ein Rahmen nicht alle Stationen. Abbildung 16.2 zeigt zur Veranschaulichung ein Netz mit 4 Stationen A, B, C und D. Die jeweilige Reichweite ist durch einen Kreis dargestellt.

In einer solchen Konstellation kann ein Sender nicht mehr alle Kollisionen

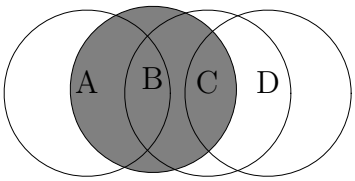


Abbildung 16.2: Drahtloses Netz mit 4 Stationen

feststellen. Angenommen A und C senden gleichzeitig an B. Da A und C weit auseinander liegen, empfangen sie nicht das Signal der jeweils anderen Station, so dass die Kollision unbemerkt bleibt. 802.11 sieht verschiedene Zugriffsverfahren vor, um solche Probleme zu umgehen.

Ein Konzept beruht auf dem Austausch von kurzen Protokollrahmen (MACA, *Multiple Access with Collision Avoidance*). Der Sender schickt zunächst eine Benachrichtigung (RTS, *Request To Send*) an den Empfänger. Ist der Empfänger bereit, antwortet er mit einer Bestätigung (CTS, *Clear To Send*). Der CTS-Rahmen dient als Bestätigung, dass der Sender mit der Übertragung beginnen kann. Bleibt das CTS aus, schließt der Sender – nach einem timeout – dass das RTS nicht angekommen ist, weil es z. B. zu einer Kollision kam. Beide Protokoll-Rahmen enthalten die Information über den Umfang der zu schickenden Daten. Daraus können benachbarte Stationen berechnen, wie lange der Transfer dauern wird und warten entsprechend lange mit eigenen Übertragungen.

Ähnlich wie bei Mobiltelefonsystemen mit Funkzellen organisiert man Netze nach 802.11 in Regionen. Die Rolle der Basisstationen übernehmen dann so genannte Anschlußpunkte (AP, *Access Point*). Die Anschlußpunkte können untereinander beispielsweise über Ethernet verbunden sein. Jeder mobile Knoten meldet sich bei einem Anschlußpunkt an und kommuniziert bis auf weiteres nur über ihn. Der Standard definiert die Protokolle, nach denen sich Rechner anmelden. Ein Knoten kann einerseits selbst nach APs suchen (aktives scanning). Andererseits schicken auch die APs Rahmen, mit denen sie ihre Dienste anbieten (passives scanning). Bei mobilen Knoten kann es allerdings durchaus vorkommen, dass ein Knoten den Bereich seines AP verlässt. In einem solchen Fall findet er wiederum mit einer der Scanning-Methoden einen besser geeigneten AP. Idealerweise erfolgt dieser Wechsel zwischen zwei APs ohne Unterbrechung der Verbindung.

Neben der Kommunikation über mindestens einen gemeinsamen AP (*infrastructure mode*) können auch mehrere Stationen direkt miteinander Verbindung aufnehmen (*ad hoc* oder *peer-to-peer mode*). Eine Sonderform sind so genannte Manets (*mobile ad hoc networks*), bei denen mehrere mobile Knoten untereinander ein gemeinsames ad hoc Netz betreiben.

Der Aufbau und Betrieb von drahtlosen Netzen erfordert deutlich höheren Aufwand. Die beschränkte Reichweite der Knoten und ihre Beweglichkeit stellen

neue Herausforderungen an die Kontrolle des Medienzugriffs. Auch die Anforderungen an die Sicherheit der Daten gegen Verfälschungen oder Abhören wachsen. Im Gegensatz zu einem leitungsgebundenen Netz ist es sehr einfach, Pakete abzuhören oder eigene Pakete einzuschleusen. Die Inhalte sollten daher durch Verschlüsselung geschützt werden. Bei Netzen mit geschlossenem Benutzerkreis ist es sinnvoll, nur Geräten mit einer registrierten Netzwerkadresse den Zugang zu erlauben. Der derzeit noch nicht verabschiedete Standard 802.11i bietet Erweiterungen bezüglich Sicherheit und Authentifizierung.

16.4 Andere Netztechnologien

Neben den beschriebenen Technologien gibt es eine ganze Reihe von Alternativen. Für viele Spezialfälle gibt es angepasste Lösungen. So bedingen manche Anwendungen hohe Anforderungen an Robustheit und Zeitverhalten. Häufig sind mehrere Alternativen verfügbar. Nicht in allen Fällen entwickelte sich ein einheitlicher Standard.

- LAN
  - Token Bus IEEE 802.4: in diesem Protokoll wird ein Token-basiertes Verfahren auf einer Bus-Architektur realisiert. Der Vorteil liegt in dem deterministischen Zeitverhalten. Diese Technik hat nur in speziellen Anwendungsbereichen Verbreitung gefunden. So basiert das von der Firma General Motors für die industrielle Fertigung entwickelte Manufacturing Automation Protocol (MAP) auf diesem Ansatz.
  - Fiber Distributed Data Interface (FDDI): eine ursprünglich für die Übertragung über Glasfaser entwickelte Technik basierend auf einem doppelten Token-Ring. Mit einer Übertragungsrate von 100 Mbit/s bei Ringlängen bis maximal 100 km wurde es häufig als *Backbone* (engl. Rückgrat) zur Vernetzung größerer Bereiche (Firmen, Uni-Campus) eingesetzt. Einzelne Rechner werden in der Regel nicht direkt, sondern über Konzentratoren an den Ring angeschlossen. Auch Ethernet-Netze können mittels FDDI/Ethernet Bridges mit dem Ring verbunden werden. Durch die Entwicklung von Fast Ethernet und Gigabit Ethernet hat FDDI kaum noch Bedeutung.
- Drahtlose Netze
  - Bluetooth[1]: ein Standard für die drahtlose Vernetzung von (kleinen) Geräten mit geringer Reichweite. Typisch ist die Vernetzung von mobilen Kleingeräte wie Mobiltelefone und PDAs. Ein solches Netzwerk wird auch als *Wireless Personal Area Network* (WPAN) bezeichnet.

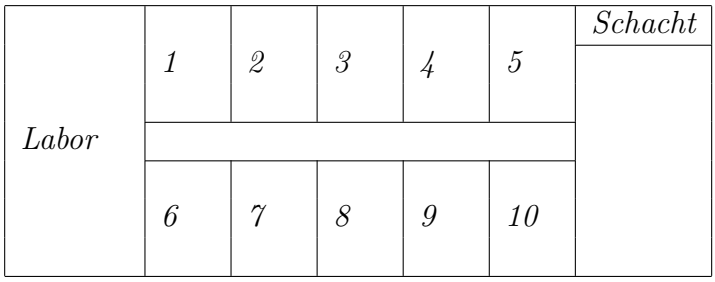
[1]Benannt nach dem Wikinger-König Harald Blauzahn, um 910-986

- Netze für Peripheriegeräte
  - SCSI Small Computer System Interface: Ein paralleler Bus für Peripheriegeräten wie Festplatten oder Scanner. Über einen SCSI-Bus können bis zu 15 Geräte angeschlossen werden. Der SCSI-Bus wird vornehmlich im High-End Bereich eingesetzt.
  - USB Universal Serial Bus: Ein schneller, serieller Bus zum Anschluß von Peripheriegeräten (Drucker, Scanner, Maus, etc.) an einen Rechner. In den Versionen 1.0 und 1.1 beträgt die Übertragungsrate bis zu 12 Mbit/s, bei der aktuellen Version 2.0 bis zu 480 Mbit/s. Durch Einsatz von USB-Hubs können an einen USB-Port bis zu 127 Geräte angeschlossen werden.
  - Firewire IEEE 1394: Eine weitere schnelle serielle Schnittstelle mit Übertragungsraten bis zu 400 Mbit/s bei maximal 63 anschließbaren Geräten. Firewire wird beispielsweise als Verbindung zwischen Rechner und externer Festplatte oder Videokamera verwendet.
- Feldbusse: Feldbusse sind für den Einsatz in der Automatisierungstechnik ausgelegt. Sie verbinden Steuergeräte mit Sensoren und Aktoren Wichtig ist die Robustheit gegenüber Störungen. Die meisten garantierten eine Zeitspanne, innerhalb der Daten immer übermittelt werden (Echtzeitverhalten).
  - PROFIBUS: Profibus ist ein offenes, flexibles System für Anwendungen in den Bereichen Prozeßleittechnik, Gebäudeautomation, etc.. Es unterstützt die Kombinationen von Geräten in unterschiedlichen Konfigurationen (z. B. Mono-Master, Multi-Master, Master-Slave). Ein Token-basiertes Protokoll regelt den Buszugriff.
  - CAN Controller Area Network: CAN benutzt eine Bus-Topologie mit Vielfachzugriffsverfahren und Prioritätsregeln. CAN wird auch zur Vernetzung von Komponenten in Kraftfahrzeugen eingesetzt. Neuere Entwicklungen für zeitkritische Anwendungen im Kfz-Bereich sind TTP (Time Triggered Protocol) und, basierend auf dem von BMW entwickelten ByteFlight Konzept, FlexRay.
- Gebäudevernetzung Darunter versteht man die Vernetzung der verschiedensten Sensoren und Geräte innerhalb eines Gebäudes. Die Vision ist eine einheitlich Steuerung von Geräten zur Klimasteuerung, Beleuchtung, Sicherheit, Kommunikation, Unterhaltung und so weiter. Derzeit wird die Technik überwiegend in gewerblichen Gebäuden eingesetzt, soll aber zunehmend auch in Privathaushalten Einzug halten.
  - EIB Europäischer Installationsbus: EIB ist ein europaweit einheitlicher Standard zur Vernetzung von Komponenten der Gebäudesystemtechnik.

– Local Operating Network LON: auch LONWORKS® genannt ist ebenfalls ein universelles Automatisierungsnetzwerk, entwickelt von der Echelon Corporation [Ech99].

16.5 Übungen

Übung 16.1 *Planen Sie das Netzwerk für Ihre Firma. Die Firma befindet sich in einem dreistöckigen Gebäude. In jedem Stockwerk sind die Räume wie folgt verteilt:*



Jeder Raum ist 5 m breit und 10 m lang. Die Labore sind 10 m breit und 23 m lang und die Geschosshöhe beträgt 4 m. Die normalen Räume benötigen je 4 Netzanschlüsse, die Labore je 10. Die Server werden in Raum 5, 1. Stockwerk untergebracht. Von dort sollen Kabel zu allen Anschlüssen im gleichen Stockwerk gelegt werden. Über den Kabelschacht können Verteiler in den beiden anderen Stockwerken erreicht werden.

- *Skizzieren Sie die notwendige Verkabelung.*
- *Informieren Sie sich im Internet über die entsprechenden Kosten der benötigten Teile (z. B. www.transtec.de). Welche Gesamtkosten ergeben sich inklusive eines großen Servers?*
- *Wie groß sind die Preisunterschiede bei verschiedenen Alternativen (unterschiedliche Übertragungskapazitäten je nach Anforderung wie z.B Büroräume oder CAD-Arbeitsplätze)?*
- *Wie sieht eine Alternative mit WLAN aus?*

Kapitel 17

Internet Protokoll IP

17.1 Einleitung

Der Erfolg des Internets zeigt, dass eine weltweite Vernetzung unterschiedlichster Rechner und Systeme möglich ist und damit funktionierende Anwendungen angeboten werden können. Das Internet ist kein einheitliches Netz sondern ein Verbund vieler Netze mit unterschiedlicher Technologie. Insgesamt entstand ein riesiger Verbund von Knoten mit weiterhin starkem Wachstum [LCC+97]. Bei dem Aufbau des bisherigen Netzes und seinem weiterer Ausbau sind Kernprobleme:

- Heterogenität: wie können die verschiedensten Netze zusammen funktionieren?
- Adressierung: wie werden eindeutige und aussagefähige Adressen vergeben?
- Routing: wie findet eine Nachricht den Weg durch das Netz?
- Skalierbarkeit: wie kann das Netz das zu erwartenden starke Wachstum aufnehmen?

Grundlage des Internets ist IP – das Internet Protokoll. IP beinhaltet die Adressvergabe und einen verbindungslosen, unzuverlässigen Datagramm-Dienst. Im OSI Modell stellt IP die Schicht 3 (Vermittlungsschicht) dar. Unzuverlässig bedeutet, dass keine Garantie bezüglich der richtigen Zustellung von Paketen gegeben wird. Pakete können verfälscht werden, verloren gehen und mehrfach oder in der falschen Reihenfolge ankommen. Das Netzwerk bemüht sich die Pakete zuzustellen, aber um eventuelle Fehler müssen sich Sender und Empfänger selbst kümmern. Man sagt, das Netz arbeitet nach dem Prinzip Best-Effort.

Tabelle 17.1: Aufbau der IP-Adressen in verschiedene Klassen

8				16				24				32				Klasse
0	Netz-ID							Host-ID								A
1	0	Netz-ID								Host-ID						B
1	1	0	Netz-ID										Host-ID			C
1	1	1	0	Multicast-Adresse												D

17.2 Adressen

Eine IP-Adresse (in Version 4, IPv4) besteht aus 32 Bit. Üblicherweise schreibt man die Adressen als 4 durch Punkte getrennte Dezimalzahlen, also z. B. 120.56.222.94, wobei jede Zahl 8 Bit repräsentiert. Jede Zahl kann damit maximal den Wert 255 annehmen. Insgesamt sind 2^{32} verschiedene Adressen zwischen 0.0.0.0 und 255.255.255.255 möglich.

Im Gegensatz zu den Ethernet-Adressen sind die IP-Adressen hierarchisch aufgebaut (Ethernet-Adressen sind hierarchisch bezüglich des Herstellers der Netzwerkkarte, eine Information die allerdings beim Routing nicht hilft). Der erste Teil der IP-Adresse bezeichnet ein Netzwerk und der zweite Teil einen Knoten. Dadurch vereinfacht sich das Routing, da zunächst nur die Netzwerk-Adresse ausgewertet werden muss.

Netzwerke sind unterschiedlich groß. Beispielsweise haben Firmen sehr unterschiedlich große Netze. So unterscheidet sich der Bedarf eines weltweiten Großkonzerns sehr von dem einer kleinen Software-Firma. Daher wurde ein flexibles Schema mit mehreren Klassen eingeführt. Die Position der ersten Null in der Adresse legt die Klasse fest. Ist das erste Bit Null (d. h. in der Hälfte der Fälle), so handelt es sich um eine Adresse der Klasse A. In diesem Fall sind die nächsten 7 Bit die Netzwerk-Adresse und die restlichen 24 Bit bezeichnen den Knoten. Berücksichtigt man noch zwei Sonderfälle verbleiben 126 Adressen für Netze der Klasse A. Diese 126 Netze belegen bereits die Hälfte aller zur Verfügung stehenden Adressen. In Tabelle 17.1 ist die Zuordnung der Bits zu den Feldern für die verschiedene Klassen zusammen gestellt.

Wie gesagt erkennt man die Klasse einer IP-Adresse an der Position der ersten Null. Betrachten wir als Beispiel die Adresse 212.201.24.18. Das erste Byte hat den Dezimalwert 212. Umwandlung in die Darstellung als Binärzahl ergibt 1101 0100. Die erste Null ist an der dritten Stelle von links. Es handelt sich also um eine Adresse aus der Klasse C.

Die Grundidee bei der Festlegung der Klassen war, dass es nur wenige sehr grosse Netze mit vielen Knoten, aber sehr viele kleine Netze mit wenigen Knoten gibt. Die jeweilige Anzahl der Netze und darin enthaltenen Knoten ist in Tabelle 17.2 eingetragen. Dabei ist berücksichtigt, dass einige Adressen eine besondere Bedeutung haben. Adressen mit dem Wert 0 beziehen sich auf das eigene Netz

Tabelle 17.2: Anzahl und Größe der Netze in den einzelnen Klassen

Klasse	Anzahl Netze	Anzahl Knoten
A	126	16777214
B	16384	65534
C	2097152	254

oder den eigenen Knoten. Sind andererseits alle Bits im Knotennamen gesetzt, so handelt es sich um eine Broadcast-Nachricht. Die Adresse 127 in der Klasse A wird für Tests benutzt, bei denen ein Knoten sein gesendetes Paket zurück erhält (Loop-Back).

Die Netznummern werden vom Network Information Center (NIC) bzw. jetzt durch Internet Assigned Numbers Authority (IANA) (`www.iana.org`) verwaltet. In der Praxis hat sich die Einteilung als nicht optimal erwiesen. Es gibt einerseits zu wenige Netze der Klasse B für große Firmen oder Institutionen. Andererseits ist die Klasse B für die meisten Firmen überdimensioniert während die Klasse C zu klein ist oder zumindest zu wenig Reserven hat. Eine Lösung, die die starre Einteilung überwindet, ist Classless InterDomain Routing CIDR. Wie der Name zeigt, ersetzt CIDR die feste Klasseneinteilung durch eine flexible Aufteilung der Bits zwischen Netzkennung und Knotennummer. Gleichzeitig wurden die Adressen den 4 Weltzonen Europe, Nordamerika, Mittel- und Südamerika sowie Asien und Pazifik zugeordnet, um das Routing zu vereinfachen.

Eine andere Verfeinerung des Adressraums erreicht man durch Subnetz-Adressierung. Angenommen eine Firma betreibt mehrere LANs, die aber unter einer einheitlichen Netzadresse angesprochen werden sollen. Um die interne Weiterleitung zu vereinfachen, werden die einzelnen LANs als Subnetze behandelt. Dazu wird ein Teil der Bits der Host-ID als Subnetz-ID verwendet. Wie viele Bits für diese weitere Hierarchie zur Verfügung stehen, wird in einer Subnetzmaske spezifiziert. Angeschlossen Knoten können dann an der Adresse erkennen, ob das Paket in ihrem eigenen LAN bleibt oder über einen Switch in ein anderes LAN geschickt wird.

Die Subnetzmaske hat die Form einer IP-Adresse. Gesetzte Bits kennzeichnen den Netzwerkteil inklusive Subnetzadresse. Beispielsweise bedeutet eine Subnetzmaske 255.255.255.0, dass nur die letzten 8 Bit die Knotennummer beinhaltet. Alle anderen Bits, bei denen die Subnetzmaske den Wert Eins hat, gehören zur Netzwerksadresse. Die genaue Interpretation – Netzadresse und Subnetzadresse – hängt von der Klasse der Adresse ab. Bei einer Adresse der Klasse B enthalten die ersten 16 Bit die Netzwerksadresse. Bei der Subnetzmaske 255.255.255.0 bilden dann die nächsten 8 Bit, die ursprünglich zur Hostadresse gehörten, die Subnetzadresse.

Der Netzwerkteil einer Adresse lässt sich leicht durch bitweise UND-Verknüpfung mit der Subnetzmaske berechnen. Als Beispiel berechnet man für die Adresse

212.201.25.18 und die Subnetzmaske 255.255.252.0 als Adresse des Netzwerkteils:

IP-Adresse	212.201.25.18	11010100.11001001.00011001.00010010
Subnetzmaske	255.255.252.0	11111111.11111111.11111100.00000000
bitweise UND		11010100.11001001.00011000.00000000
Netzwerkteil	212.201.24.0	

Um das prinzipielle Problem von zu wenigen Adressen zu lösen, wird in der Version IPv6 die Größe der Adressen auf 16 Byte erhöht. Damit stehen selbst bei schlechter Ausnutzung des Adressraums rechnerisch mehr als 1000 IP-Adressen pro Quadratmeter der Erdoberfläche zur Verfügung.

Die IP-Adressen der einzelnen Knoten können – beispielsweise vom Systemadministrator – fest vergeben werden. Eine Alternative sind dynamische Adressen. Damit können insbesondere für temporäre Verbindungen zeitlich befristete Adressen vergeben werden. Über das Protokoll DHCP (Dynamic Host Configuration Protocol) erhält ein Rechner von einem DHCP-Server eine IP-Adresse. Der DHCP-Server verwaltet die Nummern und sorgt für eine eindeutige Vergabe der Nummern. Der Client erhält die Adresse für einen bestimmten Zeitraum (Lease). Vor dem Ablauf der Gültigkeitsdauer muss der Client diese verlängern oder eine neue Lease beziehen. Ein Beispiel für meinen Laptop am Netz der FH:

```
ipconfig /all

...
Ethernetadapter "LAN-Verbindung":

  Verbindungsspezifisches DNS-Suffix: fh-friedberg.de
  Beschreibung. . . . . . . . . . . : Accton EN2242 Series MiniPCI Fast
                                            Ethernet Adapter
  Physikalische Adresse . . . . . . : 00-D0-59-6A-88-63
  DHCP-aktiviert. . . . . . . . . . : Ja
  Autokonfiguration aktiviert . . . : Ja
  IP-Adresse. . . . . . . . . . . . : 212.201.26.252
  Subnetzmaske. . . . . . . . . . . : 255.255.255.0
  Standardgateway . . . . . . . . . : 212.201.26.1
  DHCP-Server . . . . . . . . . . . : 212.201.26.1
...
  Lease erhalten. . . . . . . . . . : Dienstag, 22. Juni 2004 14:21:28
  Lease läuft ab. . . . . . . . . . : Dienstag, 22. Juni 2004 14:31:28
```

Tabelle 17.3: IPv4 Paket-Format

0	4	8	16	19 31 bit
Version	IHL	TOS	Gesamtlänge	
Ident			Flags	Offset
TTL		Protocol	Checksum	
Quellenadresse				
Zieladresse				
Optionen + Pad				
Daten				

17.3 Paketformat

Tabelle 17.3 zeigt das Format der IP-Pakete (Datagramme) in der Version IPv4. Die Darstellung ist in Vielfachen von 32 bit organisiert. Die Bedeutung der einzelnen Felder ist:

- **Version:** IP Version (hier 4)
- **IHL:** Internet Header Length. Die Länge des Headers in Vielfachen von 32 bit.
- **TOS:** Type of Service. In diesem Feld können Eigenschaften für die Übertragung spezifiziert werden.
- **Gesamtlänge:** Datagrammlänge (Header plus Daten) in Bytes. Die maximale Größe ist 65535 Byte.
- **Ident:** Identifikation des Datagramm. Notwendig falls das Datagramm für die unteren Übertragungsschichten in kleinere Pakete – so genannte Fragmente – (z. B. bei Ethernet maximal 1500 Byte) zerlegt werden muss. Jedes Fragment enthält den vollständigen IP-Header zusammen mit einem Anteil der Daten.
- **Flags:** Informationen über die Fragmentierung.
- **Offset:** Die laufende Nummer des ersten Bytes im Datenteil relativ zum ersten Byte des gesamten Datagramms.
- **TTL:** Time To Live. Zähler für die maximale Lebensdauer eines Datagramms. Der Sender setzt TTL auf einen Startwert (z.Z. 64). Jeder Router auf dem Weg dekrementiert TTL. Ist der Wert Null erreicht, wird das Datagramm vernichtet.

- **Protocol:** Spezifikation für das höhere Protokoll (z. B. 6 für TCP und 17 für UDP).
- **Checksum:** Prüfsumme für den Header (Summe in Einerkomplement)
- **Quellenadresse und Zieladresse:** Die vollständigen IP-Adressen von Sender und Empfänger.
- **Optionen + Pad:** Mögliche zusätzliche Optionen und eventuelle Füllbits. Die Anzahl der Optionen berechnet sich aus der angegebenen Header Länge.

17.4 Weiterleitung

Wenn ein Knoten – ein Rechner oder ein Router – ein Datagramm verschicken oder weiter leiten will, muss er die IP-Adresse analysieren. Zunächst prüft er die Netzwerkadresse. Stimmt die Netzwerkadresse mit seiner eigenen überein, erkennt der Knoten, dass der Zielknoten sich in seinen eigenen physikalischen Netzwerk befindet. Rechner sind normalerweise nur an einem physikalischen Netzwerk angeschlossen, während Router zwei oder mehr Netzschnittstellen haben. Router überprüfen daher, ob die Netzwerkadresse zu einem ihrer Netzanschlüsse passt.

Befindet sich das Ziel im gleichen physikalischen Netzwerk, so kann der Knoten die zugehörige interne Adresse (z. B. eine Ethernet-Adresse) ermitteln und das Datagramm direkt zustellen. Wie dies im Detail geschieht werden wir im nächsten Abschnitt sehen.

Ist das Ziel außerhalb des eigenen Netzwerkes, wird das Datagramm an einen Router weiter geleitet. Der Router wirkt als Schnittstelle in die externen Netzwerke. Im allgemeinen wird der Router selbst auch keine direkte Verbindung zu dem Ziel haben, sondern das Paket an einen weiteren Router weiter reichen bis irgendwann ein Router mit Zugang zu dem Zielnetz erreicht wird.

Die Weiterleitung basiert auf Tabellen. Ähnlich den Weiterleitungstabellen in Switches gibt es Weiterleitungstabellen für die IP-Adressen. Im einfachsten Fall enthält die Weiterleitungstabelle eine paarweise Zuordnung von Netzwerksadressen und nächsten Routern. Für die Weiterleitung des Datagramms schlägt der Knoten in seiner Weiterleitungstabelle nach, an welchen nächsten Knoten er Datagramme für die gegebene Netzwerksadresse schicken soll. Den ausgewählten nächsten Router nennt man Next-Hop-Router.

Die Weiterleitungstabelle wird nicht vollständig sein. Daher gibt es noch einen Eintrag für einen Default-Router, an den alle nicht direkt zuordnenden Datagramme geschickt werden. Ein Rechner hat häufig nur Zugang zu einem Router. In diesem Fall ist die Weiterleitungstabelle leer beziehungsweise enthält nur den Eintrag für diesen Default-Router.

Insgesamt gesehen erfolgt die Zustellung in zwei Schritten. Zunächst wird das Datagramm in das richtige Netzwerk geschickt. Anschließend wird es dort an

den Zielknoten zugestellt. Dadurch wird der Verwaltungsaufwand beträchtlich reduziert. Die Router müssen nicht für jeden Rechner in der Welt Einträge in ihre Weiterleitungstabellen aufnehmen, sondern nur noch für die Netze. Bleibt die Frage, wie die Router zu ihren Weiterleitungstabellen kommen. Dies wird Thema des übernächsten Abschnittes sein.

17.5 Zuordnung IP-Adresse zu Ethernet-Adresse

Wenn ein Knoten feststellt, dass der Zielknoten sich im gleichen Netzwerk befindet, so kann er das Datagramm direkt an dessen Adresse schicken. Sind beispielsweise die beiden Knoten über Ethernet verbunden, so benötigt er dazu die Ethernet-Adresse des Zielknotens. Die Zuordnung von IP-Adressen zu Ethernet-Adressen erfolgt über das Adreßauflösungsprotokoll (Address Resolution Protocol ARP).

Wieder wird die Information in einer Tabelle gespeichert. Jeder Knoten führt eine Zuordnungstabelle mit IP-Adressen und Ethernet-Adressen, die ARP-Tabelle oder auch ARP-Cache. Da die Adresszuordnung sich jederzeit ändern kann, wird die Tabelle dynamisch verwaltet.

Angenommen ein Knoten S möchte zum ersten Mal ein Datagramm an einen bestimmten Zielknoten Z schicken. Zu diesem Zeitpunkt gibt es noch keinen Eintrag für Z in der ARP-Tabelle. Daher schickt S eine Anfrage (Rundruf) an alle Knoten im eigenen Netz. Bei Ethernet wird diese Anfrage als Broadcast gleichzeitig an alle geschickt. Die Anfrage enthält die Zieladresse und die eigene Adresse von S. Jeder Knoten empfängt die ARP-Anfrage. Knoten Z erkennt seine eigene Adresse und reagiert, indem er eine Antwortnachricht mit seiner eigenen Ethernet-Adresse zurück schickt. Gleichzeitig fügt Z die Adresse von S in seine eigene ARP-Tabelle ein, in der Annahme, dass er bald Datagramme an S schicken wird. S erhält die Antwort, erweitert seine ARP-Tabelle und schickt das Datagramm. Bei weiteren Datagrammen nutzen S und Z die Einträge in ihren ARP-Tabellen.

Um die ARP-Tabelle aktuell zu halten, wird jeder Eintrag nach 15 Minuten automatisch wieder entfernt. Findet in der Zwischenzeit eine weitere Kommunikation mit dem Knoten statt, wird die Lebensdauer des Eintrags entsprechend verlängert. Genau so wird die Lebensdauer verlängert, wenn ein Knoten eine ARP-Abfrage von einem anderen Knoten sieht. Diese Aktualisierung nehmen alle Knoten vor, die einen Eintrag für den Sender haben. Nicht betroffene Knoten ignorieren die ARP-Abfrage und erzeugen keinen neuen Eintrag.

Mit dem Programm `arp` kann man die Übersetzungstabellen anschauen und manuell ändern. Man kann Einträge löschen oder einfügen. Manuell eingefügte Einträge sind permanent und werden auch bei Zeitüberschreitung nicht aus dem Cache gelöscht. Auf meinem Laptop ergibt sich:

```
arp -a
```

```
Schnittstelle: 212.201.26.252 on Interface 0x1000003
  Internetadresse       Physikal. Adresse     Typ
  212.201.26.1          00-20-18-58-b8-1f     dynamisch
```

Das Gegenstück zu ARP ist RARP (Reverse Address Resolution Protocol). Damit können Rechner ohne eigenen Massenspeicher von einem RARP-Server ihre IP-Adresse erfragen.

17.6 Internet Control Message Protocol

Das Internet Protokoll wird ergänzt durch das Internet Control Message Protocol (ICMP). Über ICMP tauschen Hosts und Router Nachrichten aus. Dazu gehören unter anderem Fehlermeldungen, Informationen zur Steuerung der Übertragung sowie Echo-Anfragen und Antworten. Um welche Art von Nachricht es sich handelt, wird in einem Typ-Feld im Kopf des Paketes spezifiziert. Abhängig vom konkreten Typ folgen dann die dazu gehörenden Daten. Eine ICMP-Nachricht wird im Nutzdatenfeld eines IP-Paketes übertragen.

Eine einfache Anwendung auf der Basis von ICMP ist `ping`[1]. Ping steht für Packet InterNet Groper (das Verb *grope* bedeutet greifen oder suchend nach etwas tasten). Gleichzeitig ist Ping auch die Bezeichnung für einen kurzen Impuls, wie er bei Echolot oder Radarsuchgeräten ausgesendet wird. Die gleiche Funktion hat `ping` im Netzwerk: es schickt eine ICMP-Nachricht mit einer Echo-Anforderung an einen Host und wartet auf das Echo. Ein Beispiel:

```
>Ping www.w3.org [128.30.52.25] mit 32 Bytes Daten:

Antwort von 128.30.52.25: Bytes=32 Zeit=160ms TTL=55
Antwort von 128.30.52.25: Bytes=32 Zeit=150ms TTL=55
Antwort von 128.30.52.25: Bytes=32 Zeit=150ms TTL=55
Antwort von 128.30.52.25: Bytes=32 Zeit=150ms TTL=55

Ping-Statistik für 128.30.52.25:
    Pakete: Gesendet = 4, Empfangen = 4, Verloren = 0 (0% Verlust),
Ca. Zeitangaben in Millisek.:
    Minimum = 150ms, Maximum =  160ms, Mittelwert =  152ms
```

In diesem Fall wurden 4 Pakete geschickt. Bei allen kam eine Antwort. Das Programm gibt die jeweils benötigte Zeit aus und berechnet aus den einzelnen Werten eine kleine Statistik. Eine solche erfolgreiche Ausführung zeigt, dass die Netzwerkverbindung zu dem angegebenen Rechner funktioniert. Damit ist `ping` ein

[1] Der Quellcode und vielen anderen Informationen zu dem Programm sind auf folgender Seite zu finden `www.ping127001.com/pingpage/ping.html`

nützliches Prüfwerkzeug. Falls Fehler auftreten, kann man mittels `ping` schnell prüfen, ob die Kommunikation bis zur Ebene von IP zustande kommt. Damit lassen sich elementare Fehler wie z. B. ein ausgeschalteter Zielknoten oder ein falscher Knotenname aufdecken.

17.7 Routing – eine kurze Einleitung

In der bisherigen Diskussion hatten wir behandelt, wie ein Router Pakete weiter leitet. Grundlage sind Weiterleitungstabellen, die die benötigte Information beinhalten. Wir hatten gesehen, wie damit das Problem der Weiterleitung effizient gelöst wird. Weitgehend ausgeblendet hatten wir bisher die Frage, wie die Weiterleitungstabellen gefüllt werden. Die zugrunde liegende Frage ist: wie wird die Verbindung zwischen zwei Knoten S und E festgelegt?

Wenn man das Netzwerk als Graphen sieht – eine Menge durch Kanten verbundener Knoten – ist die entsprechende Frage nach der kürzesten Verbindung zwischen zwei Knoten. In Bild 17.1 ist ein Netzwerk als Graph dargestellt. Der Graph repräsentiert allerdings nur die Verbindungen zwischen den Routern. In der Realität gehört dann zu einem Router wiederum ein lokales Netz, in dem Pakete mit den beschriebenen Methoden weiter verteilt werden.

Der Einfachheit halber wird nur zwischen verbundenen und nicht verbundenen Knoten unterschieden. Im allgemeinen werden die Kosten (Latenz, Bandbreite, Gebühren) für verschiedene Verbindungen (Kanten) unterschiedlich sein. Man kann dies berücksichtigen, indem man entsprechende Maßzahlen an die Kanten anbringt. Die Suche nach der optimalen Verbindung muss dann die entstehenden Kosten berücksichtigen.

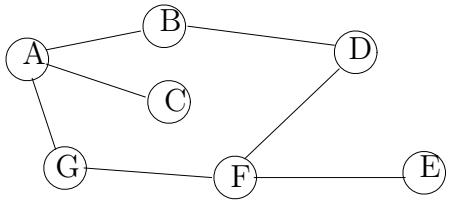


Abbildung 17.1: Netzwerk mit Knoten A,. . . ,G als Graph

Die Suche nach dem kürzesten Pfad in einem Graphen ist ein Standardproblem der Informatik. Eine klassische Anwendung ist die Bestimmung einer optimalen Route z. B. für ein Auto-Navigationssystem. Die Kosten der Kanten sind in diesem Fall die entsprechenden Fahrzeiten bestimmt durch Entfernung und Straßentyp. Zur effizienten Lösung des Problems existieren Standardlösungen. Grundvoraussetzung ist allerdings die Kenntnis des Graphs.

Die Kenntnis des Graphs – d. h. das Wissen welche Knoten aktiv sind, welche Kanten es gibt sowie die eventuellen Kosten einer Verbindungsstrecke – ist bei

einem großen Netz mit ständigen Veränderungen kaum zentral verfügbar. Daher beruhen die Routingverfahren auf verteilten Algorithmen. Jeder Knoten teilt anderen Knoten sein Wissen über das Netz mit. Aus diesen Informationen erstellt jeder Knoten für sich eine Sicht des Netzes. Im Idealfall konvergieren die Algorithmen, so dass nach einer gewissen Zeit alle Knoten eine einheitliche Sicht haben.

Die Aufgabe des Routings hängt sehr von dem betrachten Netzwerk ab. Solange es sich um ein weitgehend einheitliches Netz unter gemeinsamer Verwaltung handelt, ist die Information über den Graph relativ leicht zu erhalten. Betrachtet man demgegenüber das weltweite Internet, so ist eine einheitliche Sicht nicht mehr möglich. Verschiedene Provider haben unterschiedliche Sichtweisen der Kosten für bestimmte Verbindungen. So wird z. B. ein Provider bestrebt sein, möglichst häufig eigene Verbindungen zu benutzen, selbst wenn der resultierende Pfad nicht der kürzeste ist.

Im folgenden wird daher zwischen Intradomain- und Interdomain-Routing unterschieden. Man betrachtet ein abgeschlossenes System mit einheitlicher Administration als autonomes System (AS). Beispiele für autonome Systeme sind das interne Netzwerk eines Großunternehmens oder das Netzwerk eines Service-Providers. Dementsprechend unterscheidet man zwischen internem (intradomain) und externem (interdomain) Routing. Die Vorgehensweise bei Intradomain-Routing wird im folgenden im wesentlichen am Beispiel des Distanzvektor-Routings beschrieben. Bei der Betrachtung des Interdomain-Routings beschränkt sich die Darstellung auf einige allgemeine Eigenschaften des im Internet eingesetzten Protokolls.

17.8 Distanzvektor-Routing

Der Algorithmus des Distanzvektor-Routings basiert auf den Informationen der Knoten über ihre unmittelbaren Nachbarn. Betrachten wir nur die Anzahl der Kanten bis zu einem anderen Knoten (Hops), so erstellt ein Knoten zunächst eine Tabelle, in der alle direkten Nachbarn den Abstand 1 und alle anderen Knoten den Abstand ∞ haben. In dem obigen Beispiel hat Knoten A die Entfernungstabelle 17.4.

Tabelle 17.4: Entfernungstabelle von Knoten A

	Entfernung zu Knoten						
Knoten	A	B	C	D	E	F	G
A	-	1	1	∞	∞	∞	1

In gleicher Weise erstellen auch alle anderen Knoten ihre initiale Entfernungstabelle. Insgesamt erhält man die Tabelle 17.5.

Tabelle 17.5: Anfangswerte der Entfernungstabellen aller Knoten

	Entfernung zu Knoten						
Knoten	A	B	C	D	E	F	G
A	–	1	1	∞	∞	∞	1
B	1	–	∞	1	∞	∞	∞
C	1	∞	–	∞	∞	∞	∞
D	∞	1	∞	–	∞	1	∞
E	∞	∞	∞	∞	–	1	∞
F	∞	∞	∞	1	1	–	1
G	1	∞	∞	∞	∞	1	–

Die Tabelle existiert allerdings nur in verteilter Form. Jeder Knoten sieht nur seine eigene Zeile. Im nächsten Schritt schickt jeder Knoten seine Information an seine Nachbarn. Knoten A erfährt beispielsweise von B, dass B mit einem Hop D erreichen kann. Damit weiß A, dass er D über B mit 2 Hops erreichen kann. Knoten A ergänzt seine Tabelle und trägt den jeweils nächsten Hop (Port) für jeden Knoten ein (Tabelle 17.6).

Tabelle 17.6: Entfernungstabelle von Knoten A, nach einer Iteration

	Entfernung zu Knoten						
Knoten	A	B	C	D	E	F	G
A	-	1	1	2	∞	2	1
Port	-	B	C	B	-	G	G

Wenn alle Knoten die Informationen von ihren Nachbarn eingetragen haben, ergeben sich die Werte in Tabelle 17.7. Nach der ersten Iteration kennt Knoten G bereits die Entfernungen und die nächstgelegenen Knoten zu allen anderen Knoten im Netzwerk. Knoten E fehlen demgegenüber noch Informationen über A, B und C.

Im nächsten Schritt verschicken die Knoten diese erweiterten Tabellen – genauer gesagt jeweils ihre eigene Zeile – an ihre Nachbarn. Die Knoten werten die Informationen aus und ergänzen ihre eigenen Tabellen. Dieses Verfahren – Versenden des aktuellen Kenntnisstandes und Aktualisieren der Tabellen – wird kontinuierlich fortgesetzt. Die Knoten prüfen dabei, ob sie zu einem Knoten eine kürzere Verbindung als bisher in der Tabelle eingetragen finden. Für das Beispielnetz ergibt sich Tabelle 17.8.

In dieser Art und Weise bauen die Knoten die Information für ihre Weiterleitungstabellen auf. Kein Knoten hat die vollständige Sicht über das komplette Netz, aber jeder Knoten weiß genug, um die Pakete an den jeweils richtigen Folgeknoten zu schicken.

Tabelle 17.7: Entfernungstabellen aller Knoten nach einer Iteration

	Entfernung zu Knoten						
Knoten	A	B	C	D	E	F	G
A	–	1	1	2	∞	2	1
B	1	–	2	1	∞	2	2
C	1	2	–	∞	∞	∞	2
D	2	1	∞	–	2	1	2
E	∞	∞	∞	2	–	1	2
F	2	2	∞	1	1	–	1
G	1	2	2	2	2	1	–

Tabelle 17.8: Entfernungstabellen aller Knoten nach 4 Iterationen

	Entfernung zu Knoten						
Knoten	A	B	C	D	E	F	G
A	–	1	1	2	3	2	1
B	1	–	2	1	3	2	2
C	1	2	–	3	4	3	2
D	2	1	3	–	2	1	2
E	3	3	4	2	–	1	2
F	2	2	3	1	1	–	1
G	1	2	2	2	2	1	–

Die Knoten senden ihre Informationen periodisch oder wenn sie eine Änderung festgestellt haben. Zum Beispiel kann ein Knoten feststellen, dass einer seiner Nachbarn keine periodischen Aktualisierungen mehr schickt. Er wird dann davon ausgehen, dass die Verbindung zu diesem Knoten gestört ist und die Einträge in seiner Tabelle, die über diesen Knoten laufen, mit der Distanz unendlich belegen. Diese Information schickt er an seine anderen Nachbarn, die daraufhin ebenfalls ihre Tabellen aktualisieren.

Im Normalfall konvergiert der Algorithmus relativ schnell wieder zu einem stabilen Zustand. Allerdings kann es zu Problemen kommen, wenn im Netz Schleifen vorhanden sind. Es existieren Verfeinerungen des Verfahrens, die das Verhalten in solchen Fällen verbessern.

Ein verbreitetes Protokoll, mit dem Router ihre Weiterleitungsinformationen austauschen ist das Routing Information Protocol RIP. RIP implementiert den beschriebenen Distanzvektor-Algorithmus. Die Informationen werden durch Pakete im RIP-Format verschickt. Die Aktualisierung erfolgt nach jeweils 30 Sekunden.

Ein alternatives Verfahren ist das Link-State-Routing. Im grundsätzliche Unterschied zum Distanzvektor-Routing verschickt dabei ein Knoten nur die Infor-

mationen über seine Nachbarn, nicht sein gelerntes Wissen über weiter entfernte Knoten. Diese eingeschränkte Information schickt jeder Router allerdings an alle anderen Router. Er setzt dazu ein als *zuverlässiges Fluten* bezeichnetes Verfahren ein.

17.9 Interdomain Routing

Wie in der Einleitung beschrieben besteht das Internet aus einer Anzahl von autonomen Systemen. Dabei gibt es verschiedene Klassen von autonomen Systemen:

- AS mit nur einer Verbindung zu einem anderen AS: z. B. eine kleine Firma mit einem internen Netz und nur einem Router (Border-Gateway) als Schnittstelle nach extern.

- AS mit mehreren externen Verbindungen: z. B. eine große Firma mit mehreren Standorten. Innerhalb des AS findet aber nur lokaler Verkehr statt.

- AS mit mehreren externen Verbindungen und der Bereitschaft, Pakete für andere AS zu übermitteln: z. B. ein Backbone-Provider.

Aufgabe des Interdomain-Routings ist es, die Erreichbarkeit zwischen den AS zu gewährleisten. Die Ansprüche sind dabei wesentlich geringer als bei dem Intradomain-Routing. Wesentliches Ziel ist es, irgendeinen Weg ohne Schleifen zu finden. Die Festlegung eines optimalen Pfads ist kaum möglich, da dann beispielsweise alle AS die gleichen Kostenmetrik verwenden müssten. Außerdem sind Kriterien wie etwa „möglichst lange im eigenen Netz“ oder „nur durch vertrauenswürdige Netze“ nur schwer zu formalisieren.

Im Internet wird das Border Gateway Protocol BGP eingesetzt. BGP ist ausgelegt, um das Routing in einen Verbund von AS zu ermöglichen. Jedes AS erhält eine eindeutige Nummer (16 bit). Jedes AS benennt mindestens einen BGP-Sprecher. Die BGP-Sprecher tauschen untereinander die Routing-Information in Form von vollständigen Weglisten aus. Ein BGP-Sprecher könnte beispielsweise mitteilen, dass er das Netzwerk 178.53.66 auf dem Weg <AS15, AS356, AS27> erreichen kann. Mit entsprechenden Aktualisierungspaketen verschickt ein BGP-Sprecher die ihm bekannten Wege oder melde Wege auch wieder ab.

Durch die Kombination von Intradomain- und Interdomain-Routing wird der große, heterogene Verbund des Internets überschaubar. Ein Router innerhalb eines AS einer Firma braucht beispielsweise nur das Border-Gateway zu kennen. Er schickt dann alle nicht intern zustellbaren Pakete an diesen Router. Umgekehrt benötigen auch der Router eines Backbone-Providers nicht alle Detailkenntnisse. Es ist ausreichend, wenn er die Netze den einzelnen AS zuordnen kann.

17.10 Domain Name System DNS

Zur besseren Übersichtlichkeit verwendet man statt der numerischen IP-Adressen Namen für Knoten. Die Auflösung der Namen erfolgt durch das Domain Name System DNS. Die Namen sind hierarchisch organisiert, wobei die einzelnen Namensteile durch Punkte getrennt sind. Die Reihenfolge ist nach zunehmender Größe geordnet. Ein Beispiel ist `monet.fh-friedberg.de`. Der Knoten `monet` gehört zur Domäne `fh-friedberg`, die wiederum Teil von `de` ist. Die Top Level Domains sind entweder Länderkürzel oder Zuordnungen wie `edu` (Education) oder `mil` (Military).

Die gesamte Hierarchie ist in so genannte Zonen eingeteilt. Für jede Zone gibt es eine verantwortliche Instanz. Jede Zone stellt mindestens zwei Name-Server bereit, die auf Anfrage Informationen über die Adressen erteilen.

Wenn eine Anwendung die IP-Adresse eines Knoten benötigt, schickt sie eine Anfrage mit dem Namen an einen lokalen Name-Server. Wenn der lokale Name-Server den Namen nicht kennt, leitet er eine Abfrage an den Name-Server in seiner Zone weiter. Von dort erhält er entweder die gewünschte Zieladresse oder die Adresse eines weiteren Name-Servers. Im zweiten Fall fragt der lokale Name-Server bei dem weiteren Name-Server. Dieser Prozess wird fortgesetzt, bis ein Server gefunden wird, der die gewünschte Adresse angeben kann.

Zur Verringerung des Abfrageverkehrs merken sich die Name-Server in einem Cache für eine gewisse Zeit die erfragten Adressen. So kann eine neue Frage nach der Adresse direkt beantwortet werden.

Basis von DNS ist eine einfache Datenbank mit Ressourcendatensätzen. Ein Datensatz enthält ein Typfeld, mit dem verschiedene Arten von Informationen spezifiziert werden. DNS bietet dadurch Möglichkeiten wie die Definition von Alias-Namen oder Festlegung des Mail-Servers.

17.11 Internet-Standards

Das Internet ist ein loser Verbund vieler einzelner Rechner. Für den Endbenutzer ist in der Regel nur sein Internet-Provider relevant. Für die Gebühren an den Provider erhält er den Zugang und die Möglichkeit zur Datenübertragung. Die Infrastruktur des Internets kann er kostenfrei benutzen. Ob eine angefragte Adresse in der Nachbarschaft oder am anderen Ende der Welt liegt, spielt keine Rolle.

Gleichzeitig ist das Internet ein wesentlicher Teil der globalen Infrastruktur der Informationsgesellschaft. Die Ausgestaltung und die Weiterentwicklung des Internets wirft damit eine Vielzahl von technischen, wirtschaftlichen, politischen und kulturellen Fragen auf. Das Zusammenspiel verschiedenster Systeme setzt klare Standards voraus. Diese Standards bedürfen einer ständigen Anpassung an die technische Weiterentwicklung. Gleichberechtigter Zugang zu neuen Stan-

dards ist eine wesentliche Voraussetzung für einen fairen Wettbewerb. Fragen nach erlaubten Inhalten und die Garantie eines freien und sicheren Fluss von Informationen berühren unmittelbar unsere Lebenssituation.

Im Vergleich zu anderen Bereichen ist die Entwicklung des Internets geprägt von einem Geist der Offenheit und Freiwilligkeit. Weder staatliche Einrichtungen noch einzelne Firmen mit marktbeherrschender Stellung bestimmen das Internet. Vielmehr entwickelte sich eine Reihe von Mechanismen zur selbstbestimmten Regulation und Weiterentwicklung. Die Dachorganisation für die verschiedenen Aktivitäten zur Koordination des Internets ist die Internet Society (ISOC, `www.isoc.org`). Ihr Ziel ist zusammen gefasst in dem Mission Statement „*To assure the open development, evolution and use of the Internet for the benefit of all people throughout the world.*“ Die Mitgliedschaft in der ISOC ist kostenfrei und steht jedem Interessenten offen. Neben der internationalen ISOC gibt es nationale Vereine. So wurde 1995 aus der 1992 gegründeten Deutsche Interessengemeinschaft Internet (DIGI e.V.) unter dem Namen ISOC.DE das „German Chapter“ der Internet Society.

Technische Themen werden in speziellen Gruppen behandelt. Allgemeine grundlegende Fragen zur Architektur des Internets sind Thema des Internet Architecture Board (IAB). Spezielle Fragen wie z. B. Protokolle werden in den verschiedenen Working Groups der Internet Engineering Task Force (IETF) behandelt. Es gibt keine formale Mitgliedschaft in der IEFT. Wer in einer der zahlreichen Arbeitsgruppen mitwirken möchte, lässt sich einfach in den entsprechenden Email-Verteiler aufnehmen. Erste Versionen von technischen Dokumente haben den Status von „Internet Drafts“. Sie werden in den Arbeitsgruppen zur Diskussion gestellt. Hat ein Internet Draft einen stabilen Stand erreicht, so kann er als Request for Comments (RFC) publiziert werden. Die einzelnen RFC werden fortlaufend nummeriert. Der derzeit (Juni 2004) aktuellste ist RFC 3846 „Mobile IPv4 Extension for Carrying Network Access Identifiers“. Über die Seite `www.ietf.org/rfc.html` hat man Zugang zu allen RFCs. Wichtig ist auch dieser Stelle die Idee des freien Zugriffs auf die Dokumente sowie die Möglichkeit, die beschriebenen Protokolle lizenzfrei nutzen zu können.

Neben technischen Dokumenten wie Spezifikationen oder ähnlichem sind auch mehr beschreibende Dokumente unter den RFC. Ein Beispiel ist RFC 1118 „Hitchhikers guide to the Internet“ mit einer allgemeinen Einführung in die Internet-Technologie. RFC 2828 „Internet Security Glossary“ ist ein umfangreiches Glossar von mehr als 200 Seiten.

Aus einigen RFC werden Internet Standards. Der Ablauf der Standardisierung ist im Detail in RFC 2026 beschrieben. Durchgeführt wird dieser Prozess von der Internet Engineering Steering Group (IESG). IESG wirkt außerdem als Schnittstelle zu anderen Standardisierungsgremien wie zum Beispiel ITU.

Einige Namen und Nummern im Internet müssen fest und eindeutig vergeben werden. Dazu zählen etwa die IP-Netzwerkadressen oder die Endungen für die Top Level Domains. Derartige Werte wurden ursprünglich von der Internet Assigned

Numbers Authority (IANA) zentral vergeben und verwaltet. Mittlerweile hat Internet Corporation for Assigned Names and Numbers (ICANN) diese Aufgabe übernommen. Dabei wurde die Zuständigkeit auf vier Regional Internet Registries (RIRs) verteilt [KRWN01]. Das RIR für den europäischen Bereich ist das RIPE Network Coordination Centre (RIPE NCC) (www.ripe.net).

17.12 Übungen

1. Füllen Sie für Ihren Rechner die Felder im folgenden Formblatt aus. Benutzen Sie die Programme

   (a) `ipconfig`

   (b) `nslookup`

   (c) `ping`

   (d) `arp`

   (e) `tracert`

   mit entsprechenden Optionen, um die Informationen zu erhalten.

Hostname	
IP-Adresse	
Subnet-Mask	
Adresse ihres Netzwerks	
Netzwerk-Klasse (A, B oder C?)	
Ethernet-Adresse	
DNS-Server	
Anzahl der Hops bis zu `www.ibm.com`	

Kapitel 18

UDP und TCP

18.1 Einleitung

Bisher haben wir betrachtet, wie Pakete von einem Rechner zu einem anderen Rechner geschickt werden. Die eigentliche Kommunikation erfolgt zwischen Anwendungen, d. h. Prozessen auf den jeweiligen Betriebssystemen. Beispiele für Anwendungen sind:

- Terminalprogramme wie `telnet`
- Datentransfer `ftp`
- email Programme
- Browser

Diese Anwendungen kommunizieren mit einem entsprechenden Partner auf einem entfernten Rechner. Man spricht daher von Ende-zu-Ende-Protokollen und der Transportschicht. Diese Protokolle sind Mittler zwischen der Vermittlungsschicht und den Anwendungen. Sie müssen mit den Möglichkeiten der Vermittlungsschicht ein den Anforderungen der Anwendungen genügendes Protokoll realisieren. So muss beispielsweise für den Dateitransfer eine zuverlässige Verbindung über das unzuverlässige, best-effort IP bewerkstelligt werden.

Allerdings stellen verschiedene Anwendungen unterschiedliche Anforderungen. So benötigt `ftp` eine zuverlässigen Verbindung zur effizienten Übertragung großer Datenmengen. Ein Terminalprogramm liefert demgegenüber nur kleine Datenmengen (ein Zeichen für einen Tastendruck), die aber möglichst sofort geschickt werden sollen. Einige Anwendungen wie etwa eine Videokonferenz können gelegentliche Datenverluste akzeptieren, benötigen aber eine große Bandbreite mit einer über die Zeit relativ konstanten Qualität.

Erst auf der Ebene der Ende-zu-Ende-Protokollen können derartige Anforderung konkretisiert werden. Durch diese Arbeitsteilung wird das Protokoll der Vermittlungsebene IP entlastet. IP stellt nur die notwendige Basisfunktionalität zur

Tabelle 18.1: UDP Paket-Format

0	16 31
Quell-Port	Ziel-Port
Prüfsumme	Länge
Daten	

Verfügung und kann damit als Basis für unterschiedlichste Anwendungen dienen. Würde umgekehrt IP beispielsweise einen Schutz gegen Paketverluste gewährleisten, müssten verloren gegangene Pakete erneut geschickt werden. Durch die daraus resultierende Verzögerung wäre das Protokoll für Echtzeit-Anwendungen weniger geeignet.

Im folgenden werden die beiden Transport-Protokolle UDP und TCP vorgestellt, die die beiden Prototypen Datagrammdienst und stromorientierten Dienst über IP darstellen. Im späteren Kapitel **??** wird ein Beispiel für ein Anfrage/Antwort-Dienst vorgestellt.

18.2 UDP

Das UDP – User Datagram Protocol – ist ein einfaches Datagramm-Protokoll. Im Prinzip erweitert es lediglich den Host-zu-Host Dienst um die Adressierung einer Anwendung. Auf einen Rechner können gleichzeitig mehrere Anwendungen oder mehrere Instanzen einer Anwendung aktiv sein. Daher wird eine eindeutig Kennung für jede Anwendung benötigt.

Das Betriebssystem stellt zu diesem Zweck Kommunikationspunkte – die Ports – zur Verfügung. Eine Anwendung meldet sich an einem Port an und erhält von da an vom Betriebssystem alle an diesen Port geschickten Nachrichten. Der Port ist eine Abstraktion eines Kommunikationspunktes. Wie er konkret realisiert wird, bleibt dem Betriebssystem überlassen. Wesentlich ist, dass die UDP-Pakete die entsprechende Port-Adresse enthalten. Für die Port-Adressen stehen im UDP-Header 16 Bit zur Verfügung, wobei der Header sowohl Quell- als auch Ziel-Port enthält. Zur Vollständigkeit ist der Aufbau der UDP-Pakete in Tabelle 18.1 dargestellt. Neben den Port-Adressen enthält der Header nur noch Felder für ein optionale Prüfsumme sowie die Längeninformation.

Ein UDP-Paket wird dann für die Übermittlung in ein IP-Paket gepackt. Das IP-Paket trägt die IP-Adresse des Zielrechners. Beim Zielrechner angekommen, wird das IP-Paket entpackt und die Port-Adressen werden dem UDP-Header entnommen. Insgesamt bildet das Paar <IP-Adresse, Port-Adresse> die vollständige Adresse der Anwendung. Das IP-Paket enthält sowohl für Sender als auch Empfänger diese beiden Informationen.

Tabelle 18.2: Reservierte Portnummern

FTP-Daten	20
FTP-Steuerung	21
Telnet	23
DNS	53
WWW	80

Jeder der beiden Prozesse muss in irgendeiner Form die Port-Adressen des jeweiligen Partners kennen. Einige Dienste verwenden daher feste, öffentlich bekannte Portnummern. In Tabelle 18.2 sind für einige Anwendungen die Portnummern aufgelistet. Eine umfangreiche Liste findet man auf der Seite

`www.iana.org/assignments/port-numbers`.

In manchen Fällen benutzen Anwendungen derartige wohlbekannte Portnummern nur, um den Kontakt aufzunehmen und für die weitere Kommunikation einen eigenen Port auszuhandeln.

18.3 TCP

UDP ist letztlich nur ein Multiplexer zwischen verschiedenen Anwendungen und dem IP-Protokoll. Es bietet keine funktionale Erweiterungen gegenüber dem unzuverlässigen Paketdienst. Viele Anwendungen benötigen aber eine zuverlässige Verbindung. Das wichtigste Protokoll für diese Fälle ist TCP Transmission Control Protocol. TCP realisiert einen zuverlässigen, verbindungsorientierten Byte-Strom. Anwendungen, die auf TCP aufsetzen, werden dadurch von vielen Implementierungsdetails entlastet. Wesentliche Eigenschaften von TCP sind:

- Verbindungsaufbau
- zuverlässige Übertragung
- Vollduplex Byte-Strom
- Flußkontrolle (Schutz des **Empfängers** vor zu vielen Daten)
- Überlastkontrolle (Schutz des **Netzes** vor zu vielen Daten)
- Multiplex für mehrere Anwendungen (analog zu UDP)

Die große Herausforderung für TCP ist, über einen unzuverlässigen Paketdienst und ein komplexes, sich ständig veränderndes Netz eine zuverlässige Verbindung mit möglichst hohem Datendurchsatz und geringer Verzögerung zu realisieren. TCP benutzt den uns aus Kapitel 15 bekannten Sliding-Window-Algorithmus. Der Basisalgorithmus wird allerdings an mehreren Punkten erweitert, um der wesentlich schwierigeren Situation Rechnung zu tragen.

18.3.1 Segmentierung

Aus Sicht der Anwendung stellt TCP einen Byte-Strom dar. D. h. die Anwendung kann einzelne Bytes schreiben oder lesen. Andererseits überträgt IP Pakete. Es ist wenig effizient, jedes Byte stets in einem eigenen IP-Paket zu verschicken. Das Verhältnis Nutzdaten zu Headerdaten ist dann sehr ungünstig. Daher setzt TCP den Byte-Strom in eine Folge von so genannten Segmenten um. Den gesamten Ablauf für eine Richtung zeigt Bild 18.1

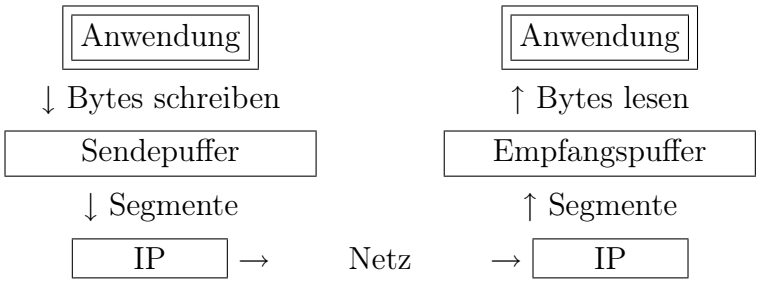


Abbildung 18.1: Segmentierung in TCP

Auf der Sendeseite sammelt TCP zunächst die geschriebenen Bytes in einem Puffer. Aus dem Puffer werden dann Bytes zu Segmenten zusammen gefasst. Im Sinne der Netzwerknutzung sind möglichst große Segmente wünschenswert, um ein gutes Verhältnis Nutzdaten zu Header-Daten zu erzielen. Anderseits führen große Segmente zu längeren Wartezeiten. Außerdem gibt es im zugrunde liegenden Netz eine Beschränkung der Paketgröße. Insgesamt verwendet TCP folgende Kriterien für das Versenden des nächsten Segments:

- Die maximale Segmentgröße (Maximum Segment Size, MSS) ist erreicht. Zweckmäßigerweise wird MSS so gewählt, dass IP das Segment nicht fragmentieren muss.
- Die Anwendung wünscht ausdrücklich den sofortigen Versand (Push-Operation). Terminalanwendungen wie `telnet` erfordern die – wenig effiziente – sofortige Weitergabe jedes Tastendrucks.
- Der Timer für die Zeit zum Aufsammeln ist abgelaufen.

Das Format der TCP-Segmente zeigt Tabelle 18.3. Die Information über die Segmentierung steckt im Feld Sequence Number. Diese Zahl bezeichnet den Index des ersten Bytes im Datenfeld. Der Header enthält keine Information zu der Länge des Datenfelds. Die Anzahl der Daten berechnet der Empfänger aus der Gesamtlänge des IP-Paketes abzüglich der beiden Header.

Wie bei UDP enthält der Header die beiden Port-Adressen, die Länge des Headers sowie eine Prüfsumme. In dem Feld Flags stehen Steuerinfomationen. Hier kann beispielsweise das Flag URG gesetzt werden, um den ersten Teil der

Tabelle 18.3: TCP Segment-Format

0	4	10	16 31
Source Port			Destination Port
Sequence Number			
Acknowledgement Number			
HdrLen	0	Flags	Advertised Window
Checksum			Urgent Pointer
Optionen + Pad			
Daten			

Daten als dringend zu kennzeichnen. Der Zeiger Urgent Pointer deutet dann auf das erste Byte, das nicht mehr zu dem dringenden Teil gehört. Die Bedeutung der anderen Felder wird im weiteren behandelt.

18.3.2 Verbindungsaufbau

TCP ist ein verbindungsorientierter Dienst. Die Kommunikation beginnt mit einer Phase zum Aufbau einer virtuellen Verbindung. Die Verbindung existiert nur auf der Ebene von TCP und nicht im physikalischen Netz. Sie geht von einer Anwendung auf einem Rechner zu einer anderen Anwendung auf einem zweiten Rechner. Die beiden Endpunkte sind jeweils durch Portadresse und IP-Adresse gegeben. Insgesamt ist die Verbindung damit durch die 4 Werte <IP-Adresse1, Port-Adresse1, IP-Adresse2, Port-Adresse2> definiert.

Eine wesentliche Aufgabe der beiden Partner ist es, sich über die Segmentnummern zu einigen. Die Designer von TCP hatten entschieden, dass für eine neue Verbindung zufällige Startwerte gewählt werden. Damit soll verhindert werden, dass eventuelle Irrläufer aus einer vorherigen Verbindung der beiden Anwendungen die neue Verbindung stören. Durch die zufälligen Startwerte können diese alten Pakete mit großer Wahrscheinlichkeit an ihren nicht mehr passenden Segmentnummern erkannt werden.

Zum Verbindungsaufbau wird ein Drei-Wege-Handshake benutzt. Dabei werden die Flags im Header zum Informationsaustausch genutzt. Der Ablauf ist wie folgt:

1. Anwendung A schickt ein Segment mit dem Flag SYN und seiner Segmentnummer N_A.

2. Anwendung B bestätigt den Eingang durch ein Segment mit den Flags SYN und ACK. Gleichzeitig setzt B die Acknowledge Number auf $N_A + 1$ um

anzugeben, welches Byte als nächstes erwartet wird. Schließlich wird die eigene Segmentnummer N_B zufällig ausgewählt und eingetragen.

3. Anwendung A erhält die Bestätigung und schickt daraufhin ebenfalls eine Bestätigung mit ACK und $N_B + 1$ als Acknowledge Number.

18.3.3 Sliding Window

Um eine zuverlässige Verbindung darzustellen, verlangt der Sender für jedes geschickte Segment eine Bestätigung. Die Bestätigung kann in einem eigenen Segment stehen oder Teil eines Segments mit Daten sein. Wir hatten bereits in der Diskussion zum einfachen Stop–and–Wait Algorithmus gesehen, dass das Netzwerk schlecht ausgelastet wird, wenn erst nach Erhalt der Bestätigung das nächste Paket geschickt wird. Diese Situation wird bei einer Verbindung im Internet noch verschärft. Die Antwortzeiten sind hier recht groß und insbesondere kann ein Paket sich z. B. durch Überlastung einer Teilstrecke deutlich verzögern. Daher muss der Sender beim Ausbleiben einer Bestätigung lange warten, bis er sicher ist, dass das Segment verloren gegangen ist.

TCP nutzt daher den Sliding Window Algorithmus und ein Sender schickt mehrere Segmente direkt nacheinander. An Hand der eingehenden Bestätigungen kann er verfolgen, welche Segmente angekommen sind. Bleibt eine Bestätigung aus, wiederholt er die alten Segmente ab dem letzten bestätigten Segment.

Schätzung der RTT

Im Internet mit den unterschiedlichsten Arten von Verbindungen, die noch dazu zeitlichen Schwankungen unterworfen sind, ist die Wahl der Wartezeit auf Bestätigungen schwierig. Ein fester Wert für den Timeout müsste an Verbindungen mit großer RTT ausgerichtet sein und wäre damit in vielen Fällen zu pessimistisch.

Daher wird bei TCP die Laufzeit adaptiv geschätzt. Der Sender misst dazu die Zeit vom Versand bis zum Erhalt der Bestätigung und benutzt diesen Wert als aktuelle Schätzung RTT_a. Diese aktuelle Schätzung kann sehr stark schwanken. Um diese Schwankungen auszugleichen, wird der Schätzwert für RTT als gleitender Mittelwert zwischen dem alten Wert und dem aktuellen Wert in der Form

$$\mathrm{RTT} = \alpha \times \mathrm{RTT} + (1 - \alpha) \times \mathrm{RTT}_a$$

gemittelt. Der Koeffizient α bestimmt, wie schnell sich die Schätzung an Änderungen adaptiert. Bei einem kleinen Wert von α geht der alte Wert nur zu einem geringen Anteil in die Berechnung ein, so dass sich Änderungen sehr schnell auswirken. Umgekehrt bewirkt ein großer Wert, d. h. α nahe an 1, dass die Änderungen langsam erfolgen.

Typische Werte liegen im Bereich von 0,8 bis 0,9. TCP benutzt dann als Wert für den Timeout das Doppelte der geschätzten RTT. Der beschriebene Algorithmus stammt aus der ursprünglichen TCP-Spezifikation. Mittlerweile werden Verfeinerungen des Algorithmus eingesetzt, die u.a. auch die Schwankung der Messwerte als Maß für die Zuverlässigkeit der Schätzung benutzen.

Flußkontrolle

Auf Seiten des Empfängers besteht die Gefahr, dass die ankommenden Daten nicht schnell genug von der Anwendung aus dem Empfangspuffer gelesen werden. Dann füllt sich der Puffer immer mehr und irgendwann ist kein Platz mehr für weitere Daten. Der Empfänger kann den Empfang neuer Segmenten nicht mehr bestätigen und der Sender muss sie später nochmals schicken. Ohne besondere Vorkehrungen würde in einem solchen Fall viel unnötiger Netzverkehr erzeugt werden. Daher verfügt TCP über einen Mechanismus, mit dem der Empfänger die Übertragungsgeschwindigkeit des Senders steuern kann.

Der Empfänger benutzt dazu das Feld Advertised Window im TCP-Header. In seinen Bestätigungsmeldungen (oder eigenen Datensegmenten) trägt er in diesem Feld ein, wie viele Bytes er zur Zeit noch aufnehmen kann. Der Sender schickt dann bis zur nächsten Bestätigung nur noch maximal so viele Bytes. Wenn beim Empfänger der Platz abnimmt weil die lokale Anwendung die Daten momentan nur sehr langsam oder gar nicht mehr liest, so reduziert er die Größe im Feld Advertised Window und bremst damit den Sender.

Im Extremfall – wenn der Puffer vollständig gefüllt ist – meldet er den Wert 0 und signalisiert damit, dass er zur Zeit keine Daten mehr aufnehmen kann. In diesem Fall schickt der Sender trotzdem weiterhin in regelmäßigen Abständen Segmente mit nur einem Datenbyte. Solange der Empfänger keine Daten aufnehmen kann, wird er die Segmente ignorieren. Hat er allerdings wieder Platz , kann er das Byte übernehmen und in der Bestätigung wieder ein größeres Advertised Window angeben. Auf diese Art und Weise wird der Transfer wieder aufgenommen, ohne dass beim Empfänger eine spezielle Vorgehensweise implementiert sein muss. Dieses Konzept, bei dem die Intelligenz beim Sender liegt, wird mit dem allgemeinen Ausdruck *smart sender/ dumb receiver rule* bezeichnet.

18.3.4 Überlastkontrolle

Ohne weitere Maßnahmen kann TCP leicht zu einer Überlastung des Netzes führen. Besonders kritisch ist, dass TCP dann wenn das Netz überlastet ist und Segmente verloren gehen mit erhöhter Aktivität reagiert indem es Segment erneut sendet. Damit verschlimmert sich eine Überlast im Netz weiter und es kann zum Kollaps kommen. Um die Situation zu verbessern, wurde Ende der achtziger Jahre eine Überlastkontrolle in TCP eingeführt. Die Überlastkontrolle basiert auf der Schätzung der Netzbelastung durch einen Sender. Auf der Basis der beob-

achteten Paketübertragungen und -verluste steuert ein Sender selbständig seine Übertragungsgeschwindigkeit.

Ein prinzipielles Problem dabei ist, dass ein Sender nur indirekt Informationen über das Netz erhält. Um das Optimum zu finden, muss er die Grenzen herausfinden. Die Grenzen findet er nur, indem er sie überschreitet. Er muss zunächst die Übertragungsrate möglichst weit erhöhen, bis er irgendwann an ein Limit stößt. Dann nimmt er die Rate zurück und nähert sich sozusagen vorsichtig von unten wieder dem Optimum. Durch diese gezielte Überschreitung des Optimums wird zusätzlich Verkehr erzeugt. Daher ist es wichtig, nach einer Überschreitung schnell wieder auf einen sicheren Wert zurück zu gehen. Umgekehrt kann das Herantasten an das Optimum langsam erfolgen.

Auf diesen Gedanken beruht das Konzept Additive Increase / Multiplicative Decrease (AIMD). Der Sender führt ein Überlastfenster, das festlegt wie viele Daten maximal bis zur nächsten Bestätigung gesendet werden dürfen. Diese Größe wird nur ausgeschöpft, wenn der Empfänger entsprechende Aufnahmekapazität signalisiert hat. Geht ein Segment verloren, vermutet der Sender eine Überlastung im Netz. In diesem Fall reduziert er sein Überlastfenster auf die Hälfte.

Umgekehrt zeigt ihm ein ACK an, dass ein Segment gut angekommen ist. Wenn alle Segmente in einem Sliding-Window Interval bestätigt wurden, erhöht er das Überlastfenster um einen kleinen Betrag. Das Fenster wird dann langsam anwachsen, bis irgendwann wieder das Limit erreicht ist. Dann wird es wieder auf die halbe Größe reduziert. Dieser Zyklus des langsamen Anwachsens und schnellen Schrumpfens wiederholt sich immer wieder. Die Priorität liegt auf dem Schutz des Netzes vor Überlast.

Das typische Verhalten des Überlastfensters zeigt Abbildung 18.2. Die Werte wurden mittels eines einfachen Simulationsprogramms erzeugt (Details dazu finden sich in der Übungsaufgabe 18.4). In der Simulation wird eine Kapazität vorgegeben. Das Überlastfenster wächst bis zum Erreichen der Kapazitätsgrenze linear an. Nach jedem Überschreiten wird das Fenster wieder auf die Hälfte des letzten Wertes reduziert. Der Wert für die Kapazitätsgrenze – in der Abbildung jeweils als Punkt eingetragen – wird in der Simulation nach jedem Zeitschritt um einen zufälligen Wert verändert.

Eine besondere Situation besteht am Anfang der Verbindung. Hier hat der Sender noch keinerlei Informationen über das Netz. Daher befindet er sich in einem Dilemma: Beginnt er mit der vollen Fenstergröße riskiert er ein Überlastung. Andererseits ist ein langsames, lineares Steigern u.U. zu pessimistisch. Daher verwendet der Sender einen Slow-Start Ansatz (im Gegensatz zu Full-Start mit der vollen Fenstergröße). In diesem Fall erhöht er das Fenster bereits nach jedem ACK. Wenn er vor dem Timeout n Segmente geschickt hat, wird im Erfolgsfall das Fenster auch n mal erhöht und nicht nur einmal wie im oben beschriebenen Additive Increase Ansatz. In Summe verdoppelt sich dadurch die Anzahl der gesendeten Segmente pro RTT. Das Verfahren ist sozusagen das Gegenstück zum Multiplicative Decrease mit umgekehrter Tendenz.

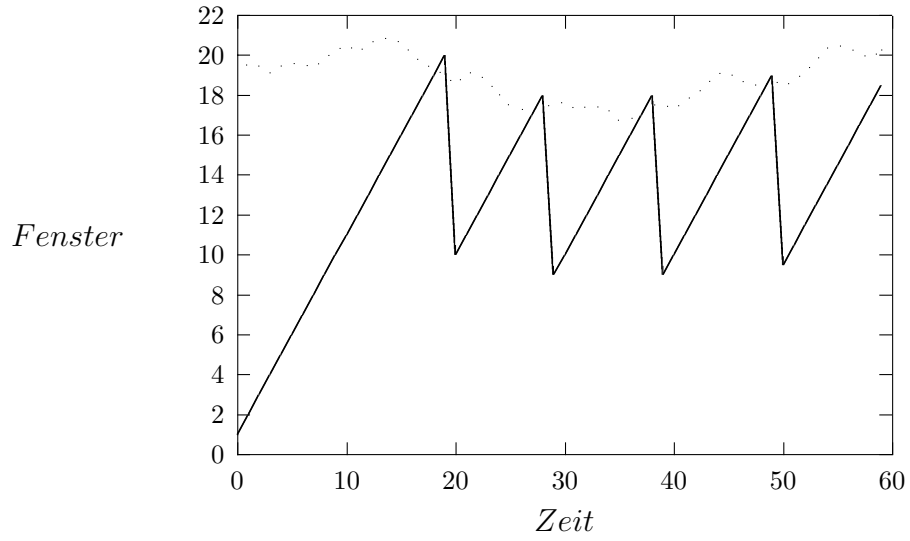


Abbildung 18.2: Regelung der Fenstergröße durch Lastkontrolle bei simulierter Kapazitätsgrenze ($\cdots$).

18.4 Übungen

Übung 18.1 *Betrachten Sie eine TCP-Verbindung mit 100Mbit/s. Wie lange dauert es bei maximaler Ausnutzung der Bandbreite, bis sich die Segmentnummer wiederholt?*

Übung 18.2 *Die Werte für die Abbildung zur Überlastkontrolle in 18.3.4 wurden mit folgendem C-Programm erzeugt.*

```
#include "stdio.h"
#include "stdlib.h"
int main(int argc, char* argv[])
{
    int t;
    double fenster    = 1;
    double kapazitaet = 20;

    for( t=0; t<60; t++ ) {
       printf("%d %.2f %.2f\n", t, fenster, kapazitaet);
       if( fenster < kapazitaet ) ++fenster;
       else fenster /= 2.;
       /* Kapazitaet um einen Wert aus [-0.6,0.6] ändern */
       kapazitaet += 1.2 * (rand()/RAND_MAX - 0.5) ;
    }
```

```
    return 0;
}
```

Ausgehend von diesem Programm:

1. *Testen Sie den Einfluss der verschiedenen Parameter auf den Verlauf.*
2. *Integrieren Sie einen Zähler für die Häufigkeit der Kapazitätsüberschreitung.*
3. *Wie lässt sich das Modell für die aktuell zur Verfügung stehende Kapazität verfeinern? Wie könnte beispielsweise ein Verhalten mit zwei Zuständen – frei und stark belastet – modelliert werden?*

Kapitel 19

Anwendungen

19.1 Einleitung

Auf die von Ende-zu-Ende Protokollen wie TCP und UDP bereit gestellten Dienste bauen Anwendungen wie email oder WWW für die Benutzer auf. Einige Anwendungen wie `telenet` nutzen direkt den Transportdienst während andere zusätzliche eigene Kommunikationsprotokolle einführen. In diesem Kapitel wird am Beispiel einiger Anwendungen diskutiert, wie diese Anwendungen die Funktionalität der Transportschicht nutzen.

19.2 WWW

Die wohl derzeit am meisten genutzte Anwendung über Rechnernetze ist das World Wide Web WWW. An diesem Beispiel lassen sich gut einige grundlegende Konzepte darstellen.

Auf Seite des Benutzers steht ein Anwendungsprogramm (User Interface), das Informationen in textueller und graphischer Form darstellt und Eingaben des Benutzers annimmt. Dieses Anwendungsprogramm ist der Web-Client oder Web-Browser wie beispielsweise Opera, Netscape, Firefox oder Microsoft Internet Explorer. Auf der anderen Seite – in der Regel über ein Netzwerk zu erreichen – steht ein Web-Server (häufig *Microsoft Internet Information Services* IIS oder Apache). Der Web-Server reagiert auf Anfragen des Clients und schickt Bestätigungen und angeforderte Informationen.

Die Kommunikation zwischen Server und Client erfolgt mittels TCP. TCP stellt einen zuverlässigen, bidirektionalen Byte-Strom bereit. Die Kommunikation hat aber die Form von Anfragen und Antworten. Daher wird ein Protokoll benötigt, das über den TCP Byte-Strom einen geeigneten Anfragen und Antwort Mechanismus realisiert. Dieses Protokoll ist HTTP: das HyperText Transport Protocol.

HTTP ist ein einheitliches Anwendungsprotokoll mit dem verschiedene Clients – d. h. Clients von unterschiedlichen Herstellern – mit unterschiedlichen Servern kommunizieren können. Die Clients und Server können auf unterschiedlichen Hard- und Software-Plattformen laufen und die Darstellung der Information durch den Client kann sich nach den jeweiligen Gegebenheiten stark unterscheiden. Aber das gemeinsame Anwendungsprotokoll ermöglicht über diese Grenzen die Interoperabilität zwischen Server und Client.

HTTP legt in seinem Anwendungsprotokoll fest, welche Anfragen möglich sind und wie die Antworten darauf aussehen. Der Inhalt der ausgetauschten Informationen ist für HTTP irrelevant. HTTP ist lediglich für den Transport zuständig. Die Informationsdarstellung ist in dem Begleitprotokoll HTML *HyperText Markup Language* geregelt. In HTML ist der Aufbau der Seite mit Formatierungsinformationen wie Textart, Textgrößen, Farben, etc. sowie die Einbindung von Elemente wie Bilder oder Videoclips beschrieben. Weiterhin kann ein Dokument Verweise (Links) auf andere Dokumente enthalten. Erweiterungen wie Java-Applets, Javascript oder Flash können in die Dokumente integriert werden.

Diese Unterteilung in Client-Anwendung, Server-Anwendung, Anwendungsprotokoll und eventuelles Begleitprotokoll findet man bei vielen Anwendungen wieder. In manchen Fällen ist die Trennung weniger deutlich sichtbar, wenn wie bei `ftp` die Anwendung den gleichen Namen wie das Anwendungsprotokoll trägt.

19.2.1 HTTP

Das Protokoll HTTP besteht aus Anfragenachrichten und Antwortnachrichten. Die Nachrichten werden in lesbarer Form als ASCII-Texte ausgetauscht. Es ist daher ohne weiteres möglich, über eine `telnet`-Verbindung selbst Anfragenachrichten an einen Server zu schicken.

Betrachten wir den Ablauf für die elementare Operation zum Lesen einer HTML-Datei. Der Standort der Datei ist als Universal Resource Locator URL spezifiziert. Ein solcher URL ist `http://www.fh-friedberg.de/index.html`. Der Ablauf ist dann wie folgt:

1. Der Client extrahiert aus dem URL den Knotennamen `www.fh-friedberg.de` des Servers.
2. Über DNS ermittelt der Client die IP-Adresse.
3. Er baut eine TCP-Verbindung zu Port 80 des Server auf.
4. Der Client schickt die Abfrage nach der Datei `index.html`.
5. Der Server antwortet und überträgt die Datei.
6. Die TCP-Verbindung wird abgebaut. (Ab der Version 1.1 besteht die Möglichkeit, die TCP-Verbindung aufrecht zu erhalten, um weitere Elemente über die gleiche Verbindung laden zu können.)

Tabelle 19.1: Anfrageoptionen in HTTP

`GET`	Lesen eines Dokuments
`HEAD`	Lesen des Headers eines Dokuments
`PUT`	Schreiben eines Dokuments
`POST`	Anhängen von Daten an bestehendes Dokument
`DELETE`	Löschen eines Dokuments
`OPTIONS`	Abfrage von verfügbaren Optionen
`TRACE`	Zu Testzwecken wird die Anfrage wie erhalten an den Client zurück geschickt.

7. Der Browser analysiert den HTML-Code der Seite und zeigt den Text an. Gegebenenfalls holt er weitere benötigte Elemente wie z. B. Bilder.

Die Anfragenachricht hat die Form:

```
GET http://www.fh-friedberg.de/index.html HTTP/1.1
```

Sie besteht aus drei Teilen:

1. dem Befehl `GET`
2. dem URL
3. einer Kennung der verwendeten Version des Protokolls.

Im allgemeinen können auf die erste Zeile der Abfrage noch mehrere weitere Zeilen mit Optionen folgen. Das Ende der Nachricht wird durch eine Leerzeile markiert. Tabelle 19.1 enthält wichtige Anfrageoptionen in HTTP. Die Optionen ermöglichen im Wesentlichen das Lesen, Schreiben und Löschen von Dokumenten auf dem Server.

In der ersten Zeile der Antwortnachricht sendet der Server seine Versionsnummer für HTTP und einen Ergebniscode mit einem erläuternden Text. Daran anschließend können optionale Informationszeilen folgen. Das Format ist *Informationsart : Informationstext*. Typische Informationen sind Beschreibung des Inhaltes oder Datum der letzten Änderung. Bei der Abfrage `GET` folgt dann der Text des angeforderten Dokuments. Als Beispiel erhält man mit der oben angegebenen Abfrage die Antwort

```
HTTP/1.0 200 OK
Server: WEBULA/1.2.3
Date: Wed, 19 Jun 2002 08:51:23 CEST
Last-Modified: Wed, 05 Jun 2002 11:00:27 CEST
MIME-Version: 1.0
Content-Type: text/html
```

Tabelle 19.2: Typen der Statuscodes in HTTP mit einigen Beispielen

Code	Typ	Erklärung
2xx:	Erfolg	Aktion empfangen, verstanden und ausgeführt
200		OK
204		No Content: Methode war erfolgreich, jedoch folgt keine Antwort im Rest der Nachricht.
3xx:	Redirection	Weitere Aktionen sind erforderlich.
301		Moved Permanently: Die angeforderte Seite ist dauerhaft umgezogen.
4xx:	Client-Fehler	Anfrage enthält fehlerhafter Syntax oder ist nicht ausführbar.
401		Unauthorized: keine Berechtigung für die angeforderte Seite.
5xx:	Server-Fehler	Eine gültige Anfrage führt zu einem Fehler im Server.
500		Internal Server Error.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
...
</HTML>
```

Die erste Zeile enthält in diesem Beispiel den Code 200 mit der Erläuterung `OK` als Bestätigung für eine erfolgreiche Transaktion. Eine Übersicht über die Statuscodes in HTTP gibt Tabelle 19.2. Die nächsten Zeilen enthalten Informationen über den Server, Datumsangaben sowie die Spezifikation des Inhalts. Getrennt durch eine Leerzeile folgt dann der HTML-Code der Seite.

19.2.2 Universal Resource Identifier

Dokumente werden über eine Adresse in Form eines Universal Resource Locators URL – eine der beiden Unterkategorien von Universal Resource Identifier URI – angesprochen. Die zweite Unterkategorie sind die Universal Resource Names URN. Dazu gehören Email oder News Adressen. URI ist ein allgemeines Schema um Ressourcen zu adressieren. Die Spezifikation ist in RFC 2396: „Uniform Resource Identifiers (URI): Generic Syntax“ beschrieben. Die allgemeine Form ist *Schema:schemaspezifischer Teil*. Im Schema-Teil wird der Dienst angegebenen. Im obigen Beispiel war dies `http`. Daneben sind u.a. noch `ftp`, `news`, `file`, `telnet` oder `mailto` möglich. Browser erlauben in der Regel auch die Eingabe

von anderen Diensten als HTTP und starten dann die entsprechende Anwendung.

Der zweite Teil hat bei HTTP die allgemeine Form *//user:password@host:port/pfad*. In den meisten Fällen werden nur ein Teil der Angaben benötigt. Weiterhin gelten Vereinbarungen wie etwa die Verwendung von `index.html` als Defaultwert für den Dateinamen. Die Abkürzung *∼username* für das Verzeichnis eines Benutzers ist aus UNIX übernommen.

Neben dem eigentlichen Namen kann man weitere Optionen an den URI anhängen. Dies ist eine einfache Möglichkeit für den Client Informationen an den Server zu schicken. Der Server entnimmt dann die Optionen dem erweiterten URI und erzeugt eine Seite mit der angeforderten Information. Als Beispiel ist der folgende URI die Anfrage an ebay nach Artikeln mit dem Schlüsselwort *Friedberg* (Zeilenwechsel und Leerzeichen sind nur zur besseren Übersicht eingefügt)[1]:

```
http://search.ebay.de/search/search.dll?MfcISAPICommand=GetResult
 &ht=1
 &shortcut=4
 &SortProperty=MetaEndSort
 &maxRecordsPerPage=50
 &st=2
 &ebaytag1code=77
 &query=friedberg
```

Der Teil bis zu dem Fragezeichen spezifiziert die Anwendung. Der Rest wird dann beim Aufruf als Argument an diese Anwendung übergeben. Im Beispiel besteht das Argument aus einer Liste mit Eigenschaften und den dazu gehörigen Werten. Die Anwendung wird vom Browser aktiviert. Sie liest die Argumente, führt eine entsprechende Aktion aus und gibt das Resultat als HTML-Text an den Browser zurück.

19.2.3 Cache

Wenn für jedes Lesen einer Web-Seite der Inhalt neu über das Netz geladen wird, entsteht sehr viel Datenverkehr. In vielen Fällen ist das neue Lesen gar nicht erforderlich, da sich der Inhalt nicht geändert hat. Daher ist es sinnvoll, die gelesenen Seiten in einem temporären Speicher – dem Cache (engl. geheimes Lager) – abzulegen. Bei einem erneuten Lesen der Seite kann dann die Kopie aus dem Zwischenspeicher verwendet werden. Dadurch wird nicht nur das Netz entlastet, sondern auch der Seitenaufbau beschleunigt.

Die Zwischenspeicherung kann an vielen Stellen erfolgen. Zunächst kann der Browser Daten lokal auf der Festplatten speichern. Weiterhin kann in einem lokalen Netz ein Knoten einen gemeinsamen Cache anbieten. Dann können mehrere

[1] Das Beispiel stammt aus einer älteren Version der ebay-Software.

Benutzer die lokale Kopie gemeinsam benutzen. Knoten mit dieser Funktionalität nennt man Proxy (engl. Stellvertreter). Neben dem Caching übernehmen Proxies weitere Aufgaben wie Regelung von Zugriffsrechten oder Umsetzung von Protokollen. So kann ein Proxy für einen Client, der nur HTTP unterstützt, eine Umsetzung auf FTP vornehmen, um damit einen FTP-Server abfragen zu können. Schließlich kann auch ein Internet Service Provider (ISP) in seinem Netz einen oder mehrere Knoten mit Cache installieren.

Das Design von HTTP unterstützt Caching durch mehrere Elemente. Bevor eine Seite aus dem Cache verwendet wird, muss sicher gestellt werden, dass die Seite noch aktuell ist. Dazu weist der Server beim Verschicken den Seiten in einem Optionsfeld `Expires` ein Gültigkeitsdatum zu. Bis zu diesem Datum kann der Cache davon ausgehen, dass die Seite noch aktuell ist. Bei späteren Zugriffen kann er über die Abfrageoption `HEAD` beziehungsweise eine spezielle Variante von `GET` prüfen, ob die Seite in der Zwischenzeit geändert wurde. Andererseits werden Seiten nach einer gewissen Zeit ohne Zugriff auch wieder aus dem Cache entfernt.

Ein Seiteneffekt des Caching-Mechanismus betrifft die häufig benutzten Zähler für Zugriffe auf Seiten. Wenn noch eine aktuelle Version in dem Cache eines Proxies liegt, wird der Zugriff eines zweiten Benutzers lokal bedient, ohne dass der Server dies zählen kann. Daher kann der Zugriffszähler einen zu niedrigen Wert enthalten. Durch das Caching werden Kopien von Webseiten an verschiedenen Stellen im Internet vorgehalten. Damit verliert der Eigner einer Seite zu einem gewissen Grad die Kontrolle über die Verbreitung seiner Seite. Wenn er seine Seite ändert - im Extremfall um Inhalt aus rechtlichen Gründen aus dem Netz zu nehmen - können immer noch Kopien mit dem mittlerweile unerwünschten Inhalt im Netz verbleiben.

19.3 Web-Anwendungen

Dem Protokoll HTTP funktioniert nach dem *Request-Response-Paradigma*. Der Client sendet eine Anforderung (*Request*), auf die der Server mit einer Antwort (*Response*) reagiert. Das Protokoll ist zustandslos. Jede Anfrage wird für sich alleine behandelt. Der Server behält keine Informationen aus früheren Anfragen. Im einfachsten Fall fordert der Client eine Datei an, der Server sendet diese Datei und der Client zeigt den Inhalt an.

Mittlerweile sind Anwendungen im Internet wesentlich komplexer. Moderne Anwendungen sind interaktiv und beinhalten eine Benutzerverwaltung. Gleichzeitig verschwindet die Trennung zwischen lokalen und zentralen Komponenten und Daten immer mehr. Für den Endanwender ist häufig gar nicht mehr ohne weiteres sichtbar, ob Daten lokal oder zentral abgelegt werden. Möglich wurde dies durch eine Reihe von Technologien sowohl für die Client als auch die Server-Seite.

Zunächst ist möglich, dass der Server Programme schickt, die dann im Client ausgeführt werden. Beispielsweise kann JavaScript-Code in den HTML-Text ein-

gebettet werden. Eine typische Anwendung ist, die Benutzereingaben in Formularfeldern auf Plausibilität zu prüfen. Fehlerhafte oder fehlende Eingaben können dann schon lokal erkannt und behandelt werden. Das Netzwerk und der Server werden entlastet und - positiv für den Benutzer - die Reaktion erfolgt sofort. Komplexe Animationen können als Java Applets oder Adobe Flash Dateien übertragen werden. In beiden Fällen benötigt der Browser zur Ausführung eine entsprechende Erweiterung (*Browser plugin*): die Java Laufzeitumgebung beziehungsweise den Adobe Flash Player. Verwendet werden diese Technologien unter anderem für interaktive Tutorials oder Spiele. Da auf diese Art und Weise sehr viel mehr Möglichkeiten als mit HTML zur Verfügung stehen, spricht man von *Rich Internet Applications* (RIA).

Aus Sicherheitsgründen können JavaScript-Programme und Java Applets nicht auf Dateien zugreifen. Es besteht allerdings die Möglichkeit, kleine Textdateien (Cookies) mit Informationen wie Benutzername oder Spielstand anzulegen und später wieder einzulesen. Diese Dateien werden vom Browser verwaltet. Ein vorsichtiger Benutzer kann diese Möglichkeit durch eine entsprechende Einstellung im Browser unterbinden.

Anders als bei den clientseitigen Verfahren wird bei serverseitigen Lösungen die angeforderte Seite auf dem Serversystem dynamisch erzeugt. Dazu stehen vielfältige Möglichkeiten zur Verfügung. Zunächst können die Seiten mit Skriptsprachen wie Perl oder PHP erzeugt werden, wobei PHP derzeit wohl am meisten verbreitet ist. Der Client schickt beispielsweise eine GET-Anfrage nach der Datei `datei.php`. Der Server ruft den PHP-Interpreter mit der angegebenen Datei auf und gibt die dabei resultierende Ausgabe an den Client zurück. Mit PHP sind neben vielen anderen Anwendungen Wikipedia und die Content-Management-Systeme TYPO3 und Mambo realisiert. Eine besonders bequeme Installation für solche Fälle bietet XAMPP. Das für verschiedene Betriebssysteme (daher das *X*) vorkonfektionierte Paket beinhaltet den Webserver Apache, die Datenbank MySQL bzw. SQLite und die Skriptsprachen Perl und PHP.

Eine komplexe, mehrschichtige Architekturen für große Anwendungen ist Java EE (*Java Platform, Enterprise Edition*). Neben anderen Komponenten werden dabei Java Servlets und Java Server Pages (JSP) eingesetzt. Zur Vereinfachung der Arbeit wurden Frameworks wie Spring oder Struts entwickelt. Eine andere Möglichkeit als serverseitige Technologie für Webanwendungen ist ASP.NET (*Active Server Pages .NET*) von Microsoft auf Basis des .NET-Frameworks.

Bei großen Dateien macht sich die Übertragungszeit negativ bemerkbar. Um die Antwortzeit zu verbessern, wird bei der Technologie Ajax (Asynchronous JavaScript and XML) die Übertragung auf das tatsächlich notwendige beschränkt und über das XMLHttpRequest API asynchron abgewickelt. Eine Anwendung, die sich dieser Technologie bedient, ist die online-Verwaltung für Bilder Flickr (www.flickr.com).

19.4 Email

Elektronische Post – email – ist eine einfache aber wirkungsvolle Anwendung. Die Aufgabe ist es, ein Dokument – z. B. ein einfacher Text, ein Bild, ein Video oder eine Kombination mehrere Elemente – von einem Sender zu einem Empfänger zu schicken. Auch hier finden wir wieder die Unterteilung in Client-Anwendung, Server-Anwendung, Anwendungsprotokoll und Begleitprotokoll. Wir betrachten das Anwendungsprotokoll *Simple Mail Transfer Protocol* (SMTP) und das Protokoll *Multi-Purpose Internet Mail Extension* (MIME) zum Format der Nachricht.

Auf Seiten des Clients ist die Situation etwas komplizierter. Anders als bei WWW ist email eine asynchrone Anwendung. Neue mails können zu beliebigen Zeiten eintreffen. Man benötigt daher noch einen Mechanismus, um die Anwendung für das Benutzerinterface (Benutzeragent, Mail-Reader) von der Zustellung zu entkoppeln. Dabei gibt es zwei unterschiedliche Strategien.

Falls eine permanente Verbindung zum Mail-Server besteht (z. B. ein Rechner in einem lokalen Netz), dann kann ein im Hintergrund laufender Prozess als Postamt arbeiten. Dieser Mail-Daemon nimmt Mails an und legt sie im Postfach des Benutzers ab. Gleichzeitig kann er den Benutzer über die Ankunft einer neuen Mail informieren. Der Benutzer kann dann mit seinem Mail-Reader die Nachricht lesen. Umgekehrt erhält der Mail-Daemon ausgehende Nachrichten und schickt sie an den Mail-Server.

Wenn andererseits nur temporär eine Verbindung zu dem Mail-Server vorhanden ist (z. B. durch Einwahl zum ISP), benötigt man ein Protokoll, um gezielt Nachrichten abzuholen und abzugeben. Ein solches Mail-Fetching Protokoll ist das *Post Office Protocol Version 3* (POP3).

Bei der Email-Anwendung ist die Trennung zwischen Client und Server weniger deutlich als bei www. Letztlich stellen beide Seiten die gleichen Funktionalität – Empfangen und Senden von Emails – bereit. Insofern ergibt sich die Unterscheidung eher aus der jeweiligen Rolle aus Sicht des Benutzers.

19.4.1 SMTP

Das Protokoll zum Austausch von Emails im Internet ist das *Simple Mail Transfer Protocol* SMTP. Wie HTTP ist es ein textbasiertes Protokoll auf der Basis von TCP mit Anfragen und Antworten. Ein kleines Beispiel für das Versenden einer Mail sieht wie folgt aus (Eingaben sind zur besseren Lesbarkeit mit `>>` gekennzeichnet):

```
>> telnet monet 25
Trying 212.201.24.18...
Connected to monet.
Escape character is '^]'.
220 monet.fh-friedberg.de ESMTP
```

```
>> HELO monet.fh-friedberg.de
250 monet.fh-friedberg.de
>> MAIL FROM <stephan.euler@mnd.fh-friedberg.de>
250 ok
>> RCPT TO <stephan.euler@t-online.de>
553 sorry, that domain isn't in my list of allowed rcpthosts (#5.7.1)
>> RCPT TO <stephan.euler@mnd.fh-friedberg.de>
250 ok
>> DATA
354 go ahead
>> Hallo
>> Hier ist eine kleine Testmail.
>>
>> .
250 ok 1024507580 qp 27898
>> QUIT
221 monet.fh-friedberg.de
```

Der Server bestätigt zunächst den erfolgreichen Verbindungsaufbau. Anschließend schickt der Client Anfragen mit Optionen. Der Server bestätigt die Anfragen mit einem dreistelligen Code und einem erläuternden Text. In dem Beispiel ist die erste Adresse nicht erlaubt und die Eingabe führt zu einem Fehler. Nach der Option `DATA` folgt der Nachrichtentext, der durch eine Zeile mit nur einem Punkt beendet wird.

19.4.2 Nachrichtenformat und MIME

Zu Beginn der Entwicklung konnte man über elektronische Post lediglich einfache Textnachrichten verschicken. Eine Nachricht besteht aus zwei Teilen: einem Kopf (Header) und einem Rumpf (Body). Jede Kopfzeile enthält einen Typ und einen dazu gehörenden Wert, getrennt durch einen Doppelpunkt. Kopfzeilen werden mit der Kombination der Zeichen für Zeilenende CR und LF (kurz <CRLF>) abgeschlossen. Eine Leerzeile trennt den Kopf vom Rumpf. Beispiele für Kopfzeilen sind:

```
Return-Path: <Manfred.Merkel@mnd.fh-friedberg.de>
Message-Id: <3C57CD5A.3020609@mnd.fh-friedberg.de>
Date: Wed, 30 Jan 2002 11:39:22 +0100
From: merkel <Manfred.Merkel@mnd.fh-friedberg.de>
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; de-DE;  ...
To: Stephan.Euler@mnd.fh-friedberg.de
Subject: [Fwd: Termin morgen]
```

Die Kopfzeilen enthalten Informationen über Sender und Empfänger, Thema, Datum, etc.. Weiterhin fügen Gateways Informationen über die Weiterleitung ein wie z.B:

```
Received: from mx0.gmx.net by merkur.hrz.uni-giessen.de for
Stephan.Euler@mnd.fh-friedberg.de; Mon, 4 Mar 2002 11:34:21 +0100
```

In der Regel steht die eigentliche Nachricht im Rumpf. Lediglich in Spezialfällen kann die gesamte Information bereits im Betreff enthalten sein (z. B. `subscribe` um sich für eine Mailing-Liste anzumelden).

Moderne Email-Systeme erlauben den Versand von nahezu beliebigen Objekten. Der Text kann formatiert sein (z. B. als HTML-Code) und Bilder, Musikstücke oder ähnliches können als Anhang (*Attachment*) mitgeschickt werden. Als Standard zur Darstellung der unterschiedlichen Objekten wird *Multi-Purpose Internet Mail Extension* (MIME) eingesetzt. MIME umfasst

- Weitere Typen für die Kopfzeile wie `MIME-Version`.
- Definition von Inhaltstypen und -untertypen; Zwei Beispiele sind: `image/gif`, `text/richtext`.
- Einem Typ `Multipart` um mehrere Objekte zu einer gemeinsamen Email zu verbinden.

Das folgende Beispiel zeigt eine Email mit einem kurzen Text und einer MSWord-Datei.

```
MIME-Version: 1.0
Content-Type: Multipart/Mixed;
 Boundary="__Next_1024570278_Part4__"

--__Next_1024570278_Part4__
Content-Type: Text/Plain;
 Charset="ISO-8859-1"
Content-Transfer-Encoding: Quoted-Printable

Text
--__Next_1024570278_Part4__
Content-Disposition: Attachment; filename="ueb1.doc"
Content-Type: application/msword;
 Name="ueb1.doc"
Content-Transfer-Encoding: Base64

0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAAABAAAALgAAAAA
AAAAEAAAMAAAAAEAAAD+////AAAAAC0AAAD/////////////////////////////
////////////////////////////////////////////////////////////////
```

Die Datei wird nicht binär übertragen, um eventuelle Probleme mit der Darstellung von Binärdaten auf den diversen Gateways zu vermeiden. In dem Beispiel wird eine Base64 Kodierung benutzt. Je 3 Bytes werden zu einer 24 Bit Einheit zusammen gefasst, aus der dann 4 ASCII Zeichen mit je 6 Bit generiert werden. Die folgende Darstellung zeigt die Zuordnung der einzelnen Bit.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1. Byte								2. Byte								3. Byte							
1. Zeichen				2. Zeichen				3. Zeichen				4. Zeichen											

Die Darstellung beschränkt sich damit auf nur 64 Zeichen: Buchstaben, Ziffern und die beiden Sonderzeichen + und /. In den Kopfzeilen des Teils stehen alle Informationen, um die Form der ursprünglichen Datei wieder herzustellen.

19.4.3 Adressierung

Aus der Email-Adresse extrahiert der Mail-Daemon die Zieladresse. Mittlerweile haben die allermeisten Adressen ein klares und leicht verständliches Format in der Art *Vorname.Name@Institut*. Der Teil nach dem @ enthält den Namen des zuständigen Mail-Servers. Die zugehörige IP-Adresse kann dann über DNS ermittelt werden. Häufig werden Emails jedoch nicht direkt, sondern über spezielle Zwischenstationen (Gateways) geleitet. Diese Gateways übernehmen auch, falls erforderlich, eine Zwischenspeicherung. Anders als IP-Pakete sollen Emails nicht ohne weiteres verworfen werden, wenn die Weiterleitung momentan nicht möglich ist. Eine weitere Aufgabe von Gateways kann die Überprüfung von Emails auf verdächtigen Inhalt wie z. B. Viren sein.

Der eigentliche Zielknoten – d. h. der Rechner auf dem der Adressat seine Emails anschauen wird – ist in der Regel nicht bekannt und kann sich darüber hinaus von Fall zu Fall ändern. Daher wird die Email zunächst nur bis zum Mail-Server geschickt. Ist der Benutzer angemeldet und hat eine aktiven Mail-Daemon, so kann die Nachricht von dort aus direkt zugestellt werden. Ansonsten wird bei der nächsten Anmeldung des Benutzers an einem Rechner durch den dann gestarteten Mail-Daemon der Server informiert. Alternativ kann der Benutzer selbst aktiv die Emails abfragen. Unter Umständen (z. B. Urlaub) muss der Mail-Server die Nachrichten über entsprechend längere Zeit vorhalten.

19.5 Usenet

Neben der direkten Kommunikation über Email besteht auch die Möglichkeit, Informationen in Diskussionsforen auszutauschen. Der Ablauf ähnelt der Veröffentlichung von Zeitungsartikeln:

- Die Teilnehmer können Beiträge (Artikel, engl. *news*) schreiben und in dem Forum veröffentlichen.
- Andere Teilnehmer können auf diese Artikel antworten. Diese Antworten werden als neue Beiträge veröffentlicht, auf die wiederum reagiert werden kann. Zur besseren Übersicht werden alle Antworten auf einen Artikel als gemeinsamer Baum dargestellt. Man spricht dann von *Threads* – dem englischen Wort für Faden oder Zwirn. Dementsprechend markiert der Newsreader von Mozzila solche Diskussionen mit dem Symbol einer Zwirnrolle.

Diese Diskussionsforen (newsgroups) sind auf speziellen Newsservern abgelegt. Ein Verbund vieler Newsserver bildet das USENET (urspr. Unix User Network). Die einzelnen Server tauschen untereinander ständig neue Artikel aus (Replikation), so dass ein einheitlicher Stand gewährleistet ist. Allerdings wird ein Server nicht alle Newsgroups bereit stellen. Daneben ist es auch möglich, mit der gleichen Technologie private oder interne Newsserver zu betreiben.

Die Vielzahl von Diskussionsgruppen ist hierarchisch unterteilt. Die einzelnen Namensbestandteile sind durch Punkte getrennt und es gilt die Regel „Vom allgemeinen zum speziellen“. Die erste Kürzel spezifiziert das Hauptthema oder ein Land. Einige Beispiele sind:

- `comp.speech.research` Ein Forum zur Forschung in der Sprachverarbeitung.
- `de.sci.informatik.ki` Eine deutsche Gruppe zum Thema *Künstliche Intelligenz*.
- `rec.games.majong` Alles über das Spiel Majong (Die Kürzel *rec* steht für *recreation*, d. h. Erholung).

Newsgroups findet man über spezielle Suchdienste wie z. B. findolin (http://www.findolin.com/). Die allermeisten Newsgroups sind unmoderiert. Allerdings wird an die Teilnehmer appelliert, bestimmte Regeln einzuhalten. Man sollte sich klarmachen, dass ein eigener Beitrag von Millionen von Menschen gelesen werden könnte. Daher ist Klarheit und Sorgfalt notwendig, um eventuellen Missverständnisse vorzubeugen. Allzu schnell wird ein Beitrag falsch verstanden und führt dann zu einem erbitterten Austausch von News. In RFC 1855 „*Netiquette Guidelines*“ sind allgemeine Regeln für Newsgroups und andere Kommunikationsdienste zusammen gestellt.

Um an Newsgroups teilzunehmen, benötigt man einen entsprechen *Newsreader*. Dazu stehen spezielle Programme zur Verfügung. Alternativ bieten die meisten Email-Programme die entsprechende Funktionalität. Man muss dann nur noch einen News-Server auswählen. Das kann der Server eines Netzanbieters sein (z. B. `news.t-online.de` bei T-Online). Daneben gibt es kommerzielle oder frei verfügbare Server wie z. B. `www.individual.de` der Freien Universität Berlin. Der komplette Verweis auf eine Gruppe hat dann die Form

`news://news.t-online.de/de.comm.internet.misc`

Der Zugang ist ebenfalls über die Seiten der Firma Google (`groups.google.de`) möglich. Der Austausch von Artikeln erfolgt über das *Network News Transfer Protocol* (NNTP). Dieses Protokoll spezifiziert eine Anzahl von Befehlen, um über eine strom-orientierte Verbindung Informationen und die Artikel zu übertragen. Es umfasst sowohl die Kommunikation zwischen Newsserver und Newsreader als auch den Abgleich zwischen zwei Newsservern. Für die Nachrichten wird die gleiche Formatierung wie für Emails verwendet. Über entsprechende Kopfeinträge werden die erforderlichen Informationen übermittelt. So wird im Eintrag `Newsgroups:` festgelegt, zu welcher Gruppe oder zu welchen Gruppen ein Artikel gehört. Der Rumpf der Nachricht wird mittels MIME kodiert.

19.6 Netzwerkmanagement

In der Diskussion der verschiedenen Protokollebenen hatten wir gesehen, dass die Designer Wert auf ein weitgehend selbständiges, unüberwachtes Funktionieren des Netzes gelegt hatten. Die Protokolle enthalten Mechanismen, um mit auftretenden Fehler umgehen zu können und die Knoten passen teilweise ihr Verhalten an die aktuelle Situation an. Trotzdem benötigt man Möglichkeiten, das Verhalten eines Netzes im Detail untersuchen zu können.

Ein dazu häufig verwendetes Protokoll ist *Simple Netwerk Management Protocol* (SNMP). SNMP erlaubt es, zahlreiche Parameter von anderen Knoten abzufragen. Jeder Knoten, der an diesem System teilnimmt, stellt einen SNMP-Server bereit. Dann kann man von anderen Systemen aus mit einem Client Informationen abfragen SNMP realisiert dazu ein Abfrage-Antwort-Protokoll auf der Basis von UDP. Die beiden wesentlichen Operationen sind `GET` um Werte zu lesen und `SET` um Werte zu setzen.

Die Informationen werden im laufenden Betrieb gesammelt und in einer speziellen Datenbank – *Management Information Base* (MIB) – abgelegt. Über ein spezielles Identifizierungssystem kann man jede Variable im MIB abfragen. SNMP verwendet dann eine standardisierte Darstellung zur Übermittlung von Datentypen wie Integer-Zahlen oder Gleitkommazahlen.

19.7 Multimedia-Kommunikation

Mit seiner ständig wachsenden Ausdehnung und Verfügbarkeit ist das Internet auch als Plattform für Multimedia-Anwendungen interessant. Unter Multimedia im eigentlichen Sinne wird die Integration verschiedener Medien in einer Anwendung oder in einem Dokument verstanden. Dabei können die Medien zeitunabhängig (Text, Grafik, Bild) oder zeitabhängig (Audio, Video) sein. Weiterhin kann man die Anwendungen in zwei Klassen aufteilen:

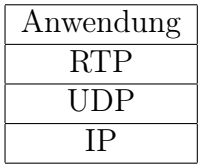


Abbildung 19.1: Protokollstack für RTP

1. Anwendungen zwischen zwei oder mehr Benutzern (Konferenz-Anwendungen)

2. Anwendungen zwischen einem Server und einem Client (Streaming-Anwendungen)

Multimedia-Anwendungen stellen hohe Anforderungen an die Übertragung. So benötigt etwa eine Video-Konferenz eine große Bandbreite bei gleichzeitig geringer Latenz. Die Synchronisation verschiedener Datenströme (z. B. getrennte Audio- und Videokanäle) erfordert zusätzliche Maßnahmen.

19.7.1 Real-Time Transport Protocol

Ein universelles Transport Protokoll für Multimedia-Daten ist das *Real-Time Transport Protocol* (RTP). Das Protokoll soll die notwendigen Eigenschaften für eine Echtzeit-Kommunikation bereit stellen, dabei aber so wenig Festlegungen wie möglich treffen. Vielmehr bleibt es der Anwendung selbst überlassen, wie sie beispielsweise mit Paketverlusten umgeht. RTP setzt auf UDP auf. UDP bietet die notwendige Grundfunktionalität – Zustellung von Paketen – ohne großen Overhead. RTP ergänzt diesen Paketdienst um einige für die Echtzeit-Kommunikation notwendige Elemente. Bild 19.1 zeigt den Protokollstack.

Der Header für RTP-Pakete ist mindestens 12 Byte lang. Er enthält eine 16 bit große Sequenznummer für das Paket. Diese Nummer wird bei jedem Paket um Eins erhöht. Anhand dieser Nummern kann der Empfänger erkennen, ob er alle Pakete in der richtigen Reihenfolge erhält. Die zeitliche Beziehung wird über ein Feld mit einem Zeitstempel hergestellt. Dabei ist das exakte Format des Zeitstempels anwendungsabhängig. Die Quelle des Datenstroms wird über eine 32 bit Zahl im Feld *Synchronization Source* (SSRC) mitgeteilt. Über ein 7 bit großes Feld wird der Typ der Nutzdaten angeben. Damit könnte beispielsweise ein Wechsel der Kodierungsmethode angezeigt werden. Schließlich kann ein RTP-Strom Beiträge von mehreren Quellen enthalten (z. B. mehrere Mikrophonkanäle).

RTP ist optimiert auf die schnelle Übertragung. Das Protokoll enthält keinerlei Elemente zum Umgang mit verlorenen Paketen oder etwa zur Überlastkontrolle. Allerdings ist es durchaus wünschenswert, dass der Empfänger über den aktuellen Zustand der Verbindung informiert wird. Beispielsweise könnte ein

Sender bei Knappheit an Bandbreite zu einer Kodierung mit niedriger Datenrate wechseln.

Zur Überwachung und Steuerung einer RTP-Sitzung dient das Protokoll *RTP Control Protocol* (RTCP). Eine RTP-Sitzung besteht aus einem RTP- und einem RTCP-Kanal. Für eine Sitzung wird für RTP eine gerade Portnummer und für RTCP die darauf folgende ungerade Portnummern vergeben. Über RTCP werden parallel zu RTP Pakete – in der Regel ebenfalls über UDP – ausgetauscht. Die Pakete beinhalten Sende- oder Empfangs-Berichte mit Übertragungsstatistiken. Diese Berichte werden periodisch verschickt. Informationen über den Sender werden mit Quellenbeschreibungspaketen übermittelt. Als Kennung dient der so genannte kanonische Name (CNAME). Das übliche Format ist *user@host*. Die Quellenbeschreibungspakete enthalten die Zuordnung zwischen SSRC und CNAME. Mit dieser Information kann der Empfänger die Daten dem Absender zuordnen. Damit können auch mehrere Quellen (z. B. Audio, Video, Grafik) von einem gemeinsamen Sender erkannt werden. Schließlich bietet RTCP die Möglichkeit, ergänzende Informationen parallel zu dem Datenstrom über RTP zu senden. Ein Beispiel hierzu sind Untertiteln für einen Videostrom. Mit verschiedenen Mechanismen wird während einer Sitzung angestrebt, den Verkehr über RTCP auf etwa 5% des RTP-Verkehrs zu beschränken.

19.7.2 Verbindungsaufbau

Vor einer Kommunikation über RTP muss eine Sitzung eingeleitet werden. Der Schwierigkeitsgrad der Aufgabe reicht dabei von der einfachen Ankündigung einer Übertragung bis zum aufwändigen Aufbau einer Videokonferenz mit Suchen der Gesprächspartner und Aushandeln der Kodierungsverfahren. Eine Arbeitsgruppe der IETF hat dafür eine Reihe von Protokollen entwickelt. Um beispielsweise eine Telefonverbindung über Internet herzustellen, werden die Protokolle *Session Initiation Protocol* (SIP) und *Session Description Protocol* (SDP) verwendet. Alternativ entwickelte ITU die Empfehlung H.323 für den Gesprächsaufbau. Die Eigenschaften des Gesprächs werden über das Call-Control Protokoll H.245 bestimmt. Geräte, die über H.323 Verbindungen aufbauen, werden als H.323-Terminals bezeichnet.

19.7.3 Sprachübertragung über IP

Eine Anwendung von großer kommerzieller Bedeutung ist die Sprachübertragung für Telefongespräche. Daher begannen verschiedene Firmen frühzeitig, Lösungen zur Sprachübertragung über IP (*Voice over IP*, VoIP) zu entwickeln. Die Technik verspricht einen deutlichen Kostenvorteil gegenüber dem herkömmlichen Telefonsystem. Zum einen entfällt die Installation eines eigenen Netzwerkes und zum anderen ist die Übertragung innerhalb des Internets zumindest bei Fernverbindungen wesentlich preisgünstiger. Auf der anderen Seite erwartet der Kunde einen

vergleichbaren Stand bezüglich

- Einfachheit der Bedienung
- gute Sprachqualität
- hohe Zuverlässigkeit

Im folgenden werden die Grundprinzipien als Beispiel für einen Echtzeitkommunikation über das Internet beschrieben. Eine ausführliche Darstellung findet man beispielsweise in [Kö02] und [Goo02].

19.7.4 Sprachcodierung

Im analogen Telefonnetz wurden die Sprachsignale in dem Frequenzbereich von 300 Hz bis 3300 Hz übertragen (Telefonbandbreite). Ausgehend von dieser Qualität wurde bei der Digitalisierung eine Abtastrate von 8000 Hz gewählt. Gemäß dem Nyquist-Kriterium, demzufolge Frequenzen bis zur halben Abtastrate dargestellt werden können, sind damit Frequenzen bis maximal 4000 Hz enthalten. Bei jedem Abtastwert wird die Amplitude als Zahlenwert kodiert (Pulscodemodulation, PCM). Die Auflösung der Zahlen bestimmt die Qualität dieser Darstellung. Für die Übertragung im Telefonnetzen nutzt man aus, dass nicht über den gesamten Amplitudenbereich eine gleichmäßige Quantisierung notwendig ist. Für große Amplituden kann man eine gröbere Rasterung verwenden, ohne dass dadurch eine hörbare Verschlechterung der Sprachqualität wahrzunehmen ist. Auf diese Weise ist es möglich, mit nur 8 bit Auflösung eine gute Qualität zu erreichen. Diese Kombination – 8 kHz Abtastrate und 8 bit Auflösung entsprechend 64 kbit/s – ist der Standard in ISDN.

Die Datenrate lässt sich verringern, wenn man spezielle Eigenschaften der Signale ausnutzt. Ein Ansatz beruht auf der Annahme, dass die Amplituden aufeinander folgender Abtastwerte in der Regel nicht beliebig weit auseinander liegen. Betrachtet man anstelle der Abtastwerte deren Differenzen, so kann der dann kleinere Wertebereich mit weniger Bit quantisiert werden. Bei adaptiven Verfahren wird der Wertebereich zusätzlich an die aktuelle Lautstärke angepasst, um eine möglichst gute Übereinstimmung zwischen Wertebereich und auftretenden Amplituden zu erreichen. Ein Standard nach diesem Prinzip ist *Adaptive Differentielle PCM* (ADPCM) mit 32 kbit/s gemäß ITU G.726. Gegenüber dem PCM-Standard wird bei nahezu gleicher Sprachqualität nur noch die halbe Datenrate benötigt. Ein Beispiel für den Einsatz dieses Standards ist DECT (*Digital European Cordless Telephony*) für schnurlose Telefone .

Sprachsignale enthalten noch sehr viel mehr Strukturen. Eine Reihe von Verfahren, die diese Strukturen weitgehend ausnutzen, beruhen auf dem Basisprinzip von *Code Excited Linear Predictive Coding* (CELP). Diese Verfahren benutzen spezielle adaptive Filter – so genannte lineare Prädiktoren – zur Vorhersage der

nächsten Abtastwerte. Das Anregungssignal wird durch systematisches Probieren von Codes aus einem Vorrat ermittelt [Nol00]. Vertreter dieser Coderfamilie werden in GSM-Netzen verwendet. Der so genannte *Full Rate Coder* arbeitet mit 12,2 kbit/s. Die neue Nachfolgetechnik Half Rate benötigt bei annähernd gleicher Sprachqualität nur noch 5,6 kbit/s. Für VoIP weit verbreitet sind die Standards G.729A (CompuServe ACELP) mit 8 kbit/s und G.723.1 MultiRate Coder mit 5.3 und 6.3 kbit/s. Ein neuerer Standard speziell für das Internet ist der *Internet Low Bit Rate Codec* (iLBC). Der Coder arbeitet bei 13.3 kbit/s oder 15.2 kbit/s. Er wurde speziell für Situationen, in denen Pakete verloren gehen, ausgelegt. In solchen Fällen wird mittels einer so genannten *Packet Loss Concealment* Einheit für eine Überbrückung der fehlenden Blöcke gesorgt.

Eine weitere Datenreduktion ist möglich, indem Sprachpausen nicht übertragen werden. Dialoge enthalten einen Anteil von bis zu 50% Pausen. Erkennt der Koder, dass ein Teilnehmer im Moment nicht spricht, so werden die Blöcke mit dem Hintergrundgeräusch nicht übertragen. Um den Eindruck einer „toten Leitung“ zu vermeiden, kann an der Gegenseite ein künstliches Rauschen eingespielt werden. Damit ist eine deutliche Reduktion der Datenkommunikation zu erreichen. Allerdings können Fehlentscheidung in der Pausedetektion oder ein zu spätes Ansprechen auf beginnende Sprachaktivität zu einem sehr unnatürlichen Spracheindruck führen.

Zu den Sprachdaten müssen bei der Berechnung des Bedarfs an Bandbreite die verschieden Paketheader hinzu addiert werden. In Tabelle 19.3 ist für einen Koder mit 6 kbit/s und einer Dauer von 30 ms für einen Sprachblock die Größe des resultierenden IP-Paketes berechnet. Bei der Übertragung über Ethernet kommen weitere 29 Byte (inklusive 3 Byte LLC Header) pro Paket hinzu. Insgesamt ergeben sich somit 92 Byte pro Paket. Mit 33,33 Paketen pro Sekunde resultiert eine Bandbreite von 24,5 kbit/s pro Sprachkanal beziehungsweise 49 kbit/s pro Telefonverbindung.

Tabelle 19.3: Größe eines IP-Paketes mit Sprachdaten

Kodierrate	6	kbit/s
Blocklänge	30	ms
Bit pro Block	180	bit
RTP-Nutzlast	23	Byte
RTP Header	12	Byte
UDP Header	8	Byte
IP Header	20	Byte
Summe	63	Byte

Die Zusammenstellung zeigt, dass die verschiedenen Header in Summe gegenüber der Nutzlast dominieren. Daher wurden Protokoll-Erweiterungen entwickelt, um die Länge der Header zu reduzieren. Ein Ansatzpunkt dazu ist die Tatsache,

dass ein Teil der Headerinformationen nur zu Beginn der Verbindung benötigt wird. Während der eigentlichen Datenübertragung werden diese Informationen entweder gar nicht mehr oder nur geringfügig verändert. Indem nur noch solche Veränderungen übertragen werden, kann die Länge der Header deutlich reduziert werden. Ein Problem besteht allerdings in der Möglichkeit, dass einzelne Pakete verloren gehen können. Ohne weitere Maßnahmen würde dann unter Umständen ein Teil der Veränderungen fehlen, so dass der Empfänger fehlerhafte Werte rekonstruiert. Speziell für dieses Einsatzgebiet wurde das Protokoll *RObust Header Compression* (ROHC), RFC 3095, definiert. Nach [RFR03] lässt sich mit ROHC eine Reduktion der Header von RTP, UDP und IP von zusammen 20 Byte auf im Mittel nur noch etwa 6 Byte erzielen, ohne dass die Sprachqualität beeinträchtigt wird.

Ein zweites wichtiges Qualititätskriterium ist die Verzögerungszeit. Ab etwa 100 ms machen sich Verzögerungen deutlich störend bemerkbar. Laut ITU-T Empfehlung gelten Verzögerungen bis 200 ms noch als gut und bis 400 ms noch als gerade akzeptabel. Die Sprachkodierung führt bei den niedrigen Datenraten zu etwa 10 bis 30 ms Verzögerung. Hinzu kommt die Latenz der Verbindung. Innerhalb eines Firmennetzes ist die resultierende Gesamtverzögerung in der Regel unterhalb der kritischen Grenzen. Bei einer Übertragung im Internet können sich allerdings durch zu große Verzögerungen und Paketverlusten deutliche Qualitätseinbußen ergeben.

19.7.5 Telefon-Anwendungen

Mit den beschriebenen Mitteln kann ein Rechner die Funktion eines Telefons übernehmen. Dazu genügen ein PC, eine handelsüblichen Soundkarte, ein Mikrophon und ein Lautsprecher sowie eine entsprechende Software. Eine Anwendung ist das Telefonieren von Rechner zu Rechner. Die entsprechenden Programme ähneln Chat-Programmen. Über einen Server kann man Verbindung zu den angemeldeten Personen aufnehmen. In der Regel können sich weitere Personen an dem Gespräch beteiligen, so dass eine Konferenz entsteht. Viele Programme unterstützen die parallele Kommunikation über mehrere Medien. So sind in das Programm `NetMeeting` der Firma Microsoft neben Audio unter anderem auch Video und Whiteboard – ein gemeinsames Zeichentablett – integriert.

Neben der Kommunikation zwischen Rechnern ist auch die Verbindung zu dem öffentlichen Telefonnetz möglich. So bieten diverse Anbieter Gateways zwischen Internet und Telefonnetz an. Eine kostengünstige Verbindung besteht dann aus einer Internet-Verbindung bis zu einem Gateway in der Nähe des Ziels und einer Telefonverbindung vom Gateway bis zum Endteilnehmer. Das Gateway übernimmt die Umsetzung der Daten sowie der Signalisierungsinformationen. Bei H.323 ist ein Gatekeeper bei der Suche nach dem günstigsten Gateway behilflich.

Das Konzept lässt sich leicht zum Telefonieren zwischen zwei konventionellen Telefonen erweitern. Ein Teilnehmer ruft bei seinem lokalen Gateway an. Von

dort wird eine RTP-Sitzung zu dem passenden Gateway in der Nähe des Ziels aufgebaut.

19.8 Übungen

Übung 19.1

Verwenden Sie `telnet` *um mit SMTP-Befehlen eine email zu schreiben und zu versenden. Testen Sie, welche Adressen als Absender akzeptiert werden.*

Übung 19.2 *Erweitern Sie den Socket-Client aus Übung* **??** *so, dass er von einem www-Server mittels HTTP die Seite index.html liest. Suchen Sie in dem HTML-Text nach Links in der Form* `<a href="Name des Links" ...>` *und geben Sie diese Links aus.*

Übung 19.3 *Über welchen URI kann man den selbst entwickelten Server ansprechen? Wie lässt sich der Server zu einem Besucherzähler ausbauen?*

Übung 19.4 *Geben Sie den Verweis auf eine Newsgroup in Ihren Browser ein. Wie zeigt er die entsprechende Gruppe an? Unter Windows können Sie den Verweis auch direkt auf dem Desktop anlegen. Was passiert dann bei Aktivieren?*

Kapitel 20

Java

20.1 Einleitung

In den Bibliotheken von Java finden sich eine ganze Reihe von Klassen zur Unterstützung von Anwendungen über Netzwerke und insbesondere im Internet. Die Klassen bieten ein unterschiedliches Einstiegsniveau. Einerseits besteht die Möglichkeit, mittels Sockets auf einer relativ niedrigen Ebene zu arbeiten. Anderseits stehen auch Klassen wie URL zur Verfügung, die bereits einen Großteil der Netzwerkfunktionalität beinhalten. Im folgenden werden exemplarisch einige Möglichkeiten der beiden Klassen Socket und URL vorgestellt.

20.2 Socket

20.2.1 Client-Socket

In der Klassenbibliothek von Java stehen mehrere nach Funktionalitäten getrennte Klassen für Sockets zur Verfügung. Die Klasse `Socket` realisiert einen Client-Socket. Verschiedene Konstruktoren ermöglichen die Angabe des Servers über den Namen oder die IP-Adresse. Als Beispiel sei die Form

```
Socket s = new Socket( "www.fh-friedberg.de", 80 );
```

angegeben. An den so erzeugten Socket können mit den Methoden

```
getInputStream()
getOutputStream()
```

Ein- und Ausgangsströme angehängt werden. Die Anwendung kann dann diese Ströme in der gleichen Art und Weise wie z. B. Ströme von und zu Dateien benutzen. Der Socket übernimmt die Details der Netzwerkkommunikation.

Als Beispiel folgt ein Programm, das über eine Socket-Verbindung eine Email mit einer Klausurnote verschickt. Zunächst öffnet das Programm eine Socket-Verbindung zu einem Mail-Server an dessen Port 25. Anschließend werden die

Befehle gemäß SMTP gesendet. Zur besseren Übersicht wird im Beispiel auf eine Erfolgskontrolle verzichtet. Dazu müssten über einen `InputStream` die zurück geschickten SMTP-Statuscodes gelesen und ausgewertet werden.

```
import java.net.*;
import java.io.*;

public class TestEmail {

    public static void main( String args[]) {
       String host   = "mailto.t-online.de";
       String from   = "stephan.euler@t-online.de";
       String to     = "stephan.euler@t-online.de";
       String betreff= "Note der Informatikklausur III";
       String inhalt = "Sie haben die Note 3 \n";
       try
       {
          Socket mailSocket = new Socket(host, 25 );
          PrintWriter out = new PrintWriter(
          mailSocket.getOutputStream(), true );

          out.println( "HELO " + host);
          out.println( "MAIL FROM: <" + from + ">");
          out.println( "RCPT TO: <" + to + ">");
          out.println( "DATA" );
          out.println( "SUBJECT: " + betreff );
          out.println( inhalt );
          out.println( ".");
          out.println( "QUIT" );

          out.close();
          mailSocket.close();
       }
       catch( IOException e ) { System.err.println( e ); }
   }
}
```

20.2.2 Server-Sockets

Ein Server für verbindungsorientierte Sockets wird mit der Klasse `ServerSocket` realisiert. Die Klasse enthält eine Reihe von Konstruktoren und Methoden, um einen Server-Socket zu erzeugen und anzumelden. Eine kompakte Form ist

```
ServerSocket socket = new ServerSocket(port, backlog);
```

mit der Angabe der Portnummer `port` und der Größe `backlog` der Warteschlange für anstehende Clients. Nach dem Aufruf der Methode `accept()` wartet der Socket auf einen Client. Meldet sich ein Client an, so kehrt die Methode mit einem „normalen“ Socket als Rückgabewert zurück. Mit diesen Elementen ist in der Klasse `SocketServer` ein einfacher Server realisiert. Der Server wartet in einer Endlosschleife auf Anfragen. Ein Client erhält bei der Kontaktaufnahme eine kurze Begrüßung. Anschließend wird die Verbindung wieder abgebaut.

```
import java.net.*;
import java.io.*;

public class SocketServer {
   static int backlog = 10;  // Laenge der Warteschlange
   static int port    = 1234;

   public static void main( String args[]) {

      try
      {
       ServerSocket socket =
                  new ServerSocket(port, backlog);
       for( ;; ) {
          System.out.println( "Warte auf Verbindung... " );
          Socket sockConnected = socket.accept();
          System.out.println( "Verbunden mit "
                        + sockConnected);
          PrintStream ps = new PrintStream(
                 sockConnected.getOutputStream() );
          ps.println( "Hallo" );
          sockConnected.close();
       }

   }
   catch( IOException e ) { System.err.println( e ); }
   }
}
```

20.2.3 UDP-Socket

Der beschrieben Client-Socket verwendet eine stromorientierte Verbindung (TCP). Paketorientierte Verbindungen (UDP) werden mit der Klasse `DatagramSocket` realisiert. So erzeugt die Zeile

```
DatagramSocket s = new DatagramSocket(8888);
```

einen neuen Socket für Datagramm-Kommunikation über Port 8888. Der Datenaustausch erfolgt dann mit Objekten der Klasse `DatagramPacket`. Zum Versenden werden in diese Pakete die Daten und die Zieladressen eingetragen. Das Zusammenspiel der Klassen soll am Beispiel eines Chat-Servers dargestellt werden. Der Server hat folgende Funktionalitäten:

- Anmeldung von Clients
- Die Nachricht eines Clients wird an alle andere Clients weiter geleitet
- Abmeldung von Clients

An- und Abmeldung erfolgen über Pakete, die eine entsprechende Kennung enthalten. Der Server führt eine Liste mit allen angemeldeten Clients. Erhält er von einem der Clients eine Nachricht, so wird diese an alle anderen Clients kopiert. Der folgende Code realisiert einen solchen Server. Die angemeldeten Clients werden in einem Objekt der Klasse `Vector` gespeichert. Die Fehlerprüfung beschränkt sich auf die Ausgabe von eventuell auftretenden Ausnahmefehlern (Exceptions).

```
import java.net.*;
import java.io.*;
import java.util.*;

public class DatagrammServer {
   // Kennungen für An- und Abmeldung
   static final String  ANMELDUNG ="ANMELDUNG";
   static final String  ENDE      ="ENDE";
   static int port    = 1234;
   static int length  = 256;   // Länge eines Pakets

   public static void main( String args[]) {

      DatagramPacket paket =
           new DatagramPacket( new byte[length], length);
      Vector clients = new Vector();   // Liste der Clients

      try {
        DatagramSocket socket = new DatagramSocket(port);
        for( ;; ) {

           // Warten auf nächstes Paket
           socket.receive( paket );
           InetSocketAddress add =
               (InetSocketAddress)paket.getSocketAddress();
```

```
            // Text aus Paket extrahieren
            String text =
               new String(paket.getData(), 0, paket.getLength());
            System.out.println( add +">" + text);

            // Paket auswerten
            if( text.equals( ANMELDUNG ) ) {
               clients.add( add );
               System.out.println( "Anzahl Clients: "
                                   + clients.size() );
            }  else if( text.equals( ENDE ) ){
               clients.remove( add );
               System.out.println( "Anzahl Clients: "
                                   + clients.size() );
            }  else {
               // Versenden von Kopien an alle anderen Clients
               for( int i=0; i<clients.size(); i++ ) {
                 InetSocketAddress dest =
                     (InetSocketAddress) clients.get(i);
                 if( ! dest.equals( add ) ) {
                    paket.setSocketAddress( dest );
                    socket.send( paket );
                    System.out.println( "Kopie an " + dest );
                 }
               }
            }
          }
       }
       catch( IOException e ) {
          System.err.println( "Ausnahmefehler: " + e );
       }
    }
}
```

Als passendes Gegenstück kann der folgende Client benutzt werden. Bei dem Aufruf kann der Name des Servers als Argument übergeben werden. Fehlt diese Angabe, sucht der Client auf dem eigenen Rechner nach dem Server. Der Client meldet sich dann mit einem entsprechenden Paket beim Server an. In einer Schleife wartet er auf Eingaben. Jede Eingabe wird als Zeichenkette (String) gelesen, in ein Feld von Bytes umgewandelt und dann als Paket verschickt. Besteht die Eingabe aus dem Text `ENDE`, so wird dies als Abmeldenachricht an den Server geschickt. Anschließend wird die Schleife verlassen und das Programm beendet.

Anders als bei dem Server erfolgt der Nachrichtenaustausch asynchron. Der Server wartet auf eine Nachricht und reagiert darauf. Demgegenüber kann beim Client zu jedem Zeitpunkt vom Server eine neue Nachricht eintreffen. Es gibt keine zeitliche Koppelung zwischen Eingaben und Ausgaben.

In der Programmiersprache Java bzw. in der virtuellen Maschine zur Ausführung von Java-Anwendungen sind Möglichkeiten für parallele Verarbeitung integriert. Eine Anwendung kann in mehrere eigenständige Programmfragmente – so genannte Threads – aufgeteilt werden. Im vorliegenden Fall wird die Aufgabe auf zwei solche Threads aufgeteilt. Auch die Hauptanwendung mit der Schleife für Benutzereingaben läuft in einem Thread. Die einkommende Nachrichten werden in einem eigenen Verarbeitungsstrang in einem zweiten Thread behandelt. Die beiden Threads laufen parallel.

Programmtechnisch wird der zweite Thread in einer eigenen Klasse realisiert. Diese Klasse implementiert das Interface `Runnable`. Damit verpflichtet sie sich, eine Methode `run` zu realisieren. Bei der Instanzierung eines Objektes der Klasse wird dann automatisch diese Methode aufgerufen und in einem eigenen Thread ausgeführt. Die Hauptanwendung braucht daher nur ein Objekt der Klasse anzulegen um den zweiten Thread zu starten. Beim Anlegen wird im Konstruktor der Socket übergeben.

```
import java.net.*;
import java.io.*;

public class DatagrammClient {
   static final String  ANMELDUNG = "ANMELDUNG";
   static final String  ENDE      = "ENDE";
   static int port    = 1234;
   static int length  = 256;    // Länge eines Pakets

   public static void main( String args[]) {
      String servername = "localhost";
      String text = null;
      DatagramPacket packet;
      byte[] ba = ANMELDUNG.getBytes();

      // Namen des Servers von Kommandozeile übernehmen
      if( args.length > 0 ) servername = args[0];

      try {
        DatagramSocket socket = new DatagramSocket();
        InetAddress ia =  InetAddress.getByName( servername );
        packet = new DatagramPacket( ba, ba.length, ia, port);
        // sende Anmeldung
```

```
        socket.send( packet );

        // Lesen der empfangenen Pakete erfolgt in eigenem Thread
        LeseThread lt = new LeseThread( socket );

        // Eingaben von Tastatur an Server schicken
        BufferedReader br = new BufferedReader(
             new InputStreamReader( System.in )  );
        do {
           text = br.readLine();
           ba = text.getBytes();
           packet.setData( ba, 0, ba.length );
           socket.send( packet );
        } while( ! text.equals("ENDE") );

        // alles beenden
        System.exit(0);

      }
      catch( IOException e ) {
        System.err.println("Ausnahmefehler: " +  e );
      }
   }
}
```

Dazu gehört die Klasse `LeseThread` zum Empfang von Nachrichten.

```
class LeseThread implements Runnable {
   static int length  = 256;
   DatagramSocket socket;

   LeseThread(DatagramSocket socket ) {
      this.socket = socket;
      Thread t = new Thread(this,"Lesen");
      t.start();
   }

   public void run()  {
      DatagramPacket packet =
         new DatagramPacket( new byte[length], length);
      while( true ) {
        try {
           socket.receive( packet );
```

```
            InetSocketAddress add =
               (InetSocketAddress)packet.getSocketAddress();
            String text =
              new String(packet.getData(), 0, packet.getLength());
            System.out.println( add +">" + text);
            //System.out.println( ">" + text);
         }
         catch( IOException e ) {
            System.err.println("Ausnahmefehler: " +  e );
         }
      }
   }
}
```

20.3 Die Klasse URL

Während einerseits die Klassen zu Sockets Netzwerkkommunikation auf einer relativ niedrigen Ebene behandeln, gibt es in Java auch zahlreiche Möglichkeiten auf höherer Ebene anzusetzen. Eine solche Möglichkeit besteht über die Klasse `URL`. Ein URL-Objekt wird im einfachsten Fall durch den Konstruktor mit dem Namen des Links als Parameter erzeugt:

```
URL url = new URL("http://www.fh-friedberg.de");
```

Das URL-Objekt beinhaltet zunächst nur die Information über den Link. In einen weiteren Schritt kann darüber eine Verbindung aufgebaut werden. Eine Möglichkeit dazu bietet die Methode `openConnection()`. Sie liefert ein Objekt der Klasse `URLConnection`, über das die weitere Kommunikation läuft. Im folgenden Beispiel wird dieses Schema benutzt, um Informationen über einen Link abzufragen. Insbesondere verwendete die Anwendung die Methode `getContentType()` um den Typ des Dokumentes zu erfragen. Der Rückgabewert ist eine Zeichenkette mit der MIME-Angabe des Typs. Die Methode `getContent()` liefert ein passendes Objekt, um den Inhalt des Links zu lesen. Bei HTML-Dateien sind dies Varianten von `InputStream`. Für gif-Dateien als anderes Beispiel erhält man ein Objekt der Klasse `URLImageSource`.

```
import java.net.*;
import java.io.*;

public class ExamineURL {
   static int keyWidth = 20;

   public static void main( String args[]) {
```

```
        // Kopieren des Argumentes beim Aufruf
       String link = args[0];

       try {
          System.out.println("Trying URL " + link );
          URL url = new URL( link  );
          URLConnection conn = url.openConnection();
          int n = 1;

          // Ausgabe der Elemente des http-Headers
          String key;
          System.out.println("Header fields:");
          while((key=conn.getHeaderFieldKey(n) ) != null ) {
             key += ": ";
             while( key.length() < keyWidth ) key += " ";
             System.out.println( n + " " + key
                         + conn.getHeaderField(n) );
             ++n;
          }

          // Objekt für Zugriff auf Inhalt
          Object o =  conn.getContent();
          System.out.println( "Content:     " +  o );
          System.out.println( "ContentType: "
                              + conn.getContentType() );
          System.out.println( "Permission:  "
                              + conn.getPermission() );

       } catch( Exception ex ) {
             System.out.println("Cannot create URL "
                              + link + " Excption:" + ex);
       }
    }
}
```

Bei der Anwendung auf die Homepage der FH Friedberg erhält man:

```
java ExamineURL http://www.fh-friedberg.de
Trying URL http://www.fh-friedberg.de
Header fields:
1 Date:               Mon, 16 Dec 2002 08:56:52 GMT
2 MIME-Version:       1.0
3 Content-Type:       text/html
Content:     java.io.PushbackInputStream@3169f8
```

```
ContentType: text/html
Permission:  (java.net.SocketPermission
    www.fh-friedberg.de:80 connect,resolve)
```

20.3.1 Mailto-Links

Ein URL-Objekt kann nicht nur auf Dateien verweisen sondern auch andere Dienste ansprechen. Ein Beispiel sind Verweise mit dem Protokoll `mailto`. Damit können über ein URL-Objekt auch Emails verschickt werden. Die Klasse `URLemail` verwendet diesen Mechanismus, um eine Notenemail zu verschicken. Zu beachten ist, dass vor dem Verbinden der Java-VM der zu verwendende Mail-Server bekannt gemacht werden muss. Dies erfolgt durch Setzen der entsprechenden Systemeigenschaft mittels

```
System.setProperty("mail.host", "mailto.t-online.de");
```

In entsprechender Art und Weise können auch andere Eigenschaften wie z. B. der Name eines eventuell vorhandenen Proxy-Servers festgelegt werden.

```
import java.io.*;
import java.net.*;

public class URLemail
{
   public static void main(String args[])throws Exception {
      String from      = "stephan.euler@t-online.de";
      String to        = "stephan.euler@t-online.de";
      String subject   = "Klausurergebnis IN3";
      String nachricht = "Sie haben die Note 2.";

      System.setProperty("mail.host",
                    "mailto.t-online.de");
      URL u = new URL("mailto:" + to);
      URLConnection c = u.openConnection();
      c.setDoOutput(true);
      System.out.println("Connecting...");
      c.connect();

      PrintWriter out = new PrintWriter(
             new OutputStreamWriter(c.getOutputStream()));
      out.println("Subject: " + subject);
      out.println(nachricht);
      out.close();
      System.out.println("Nachricht versendet.");
```

```
    }
}
```

20.4 Übungen

Übung 20.1 *Ergänzen Sie die Socket-Anwendung zum Email-Versand um eine Kontrolle der zurück gegebenen Statusmeldungen.*

Übung 20.2 *Erweitern Sie das Beispiel eines Chat-Servers:*

1. *Ein Client wird automatisch mit der ersten Nachricht angemeldet.*
2. *Wenn ein Client nach einer vorgegebenen Anzahl von Nachrichten selbst keine geschickt hat, wird er aus der Liste der aktiven Clients gelöscht. Ergänzen Sie das Protokoll, so dass Clients ansonsten leere Nachrichten als „Lebenszeichen“ schicken können.*
3. *Was passiert, wenn ein eingegebener Text länger als die Paketlänge 256 ist? Erweitern Sie die Programm, um auch solche langen Texte behandeln zu können.*
4. *Wenn sich der Client hinter einem Firewall befindet, kann der Sendeport von Paket zu Paket variieren. Verändern Sie das Protokoll dahingehend, dass ein Client bei der Anmeldung eine eindeutig Kennung erhält. Mit dieser Meldung markiert er dann alle Nachrichten, so dass sie vom Client wieder eindeutig zugeordnet werden können.*

Übung 20.3 *Testen Sie die Anwendung* `ExamineURL` *mit verschiedenen Dateitypen als Ziel. (Hinweis: über das Protokoll* `file:` *können auch lokale Dateien angesprochen werden.)*

Kapitel 21

Datensicherheit und Verschlüsselung

21.1 Einleitung

Mit der zunehmenden Bedeutung vernetzter Rechnersysteme wächst der Bedarf an Sicherheit. Insbesondere die Verlagerung von kritischen Abläufen wie

- Einkaufen und Verkaufen
- Abwickeln von Bank- und Börsengeschäften
- Bewerbungen auf Ausschreibungen

hin zu elektronischen Formen verlangt eine sichere und vertrauliche technische Grundlage. Darüber hinaus gilt es, die direkt oder indirekt an das universelle Netz angeschlossenen Systeme gegen unberechtigte Zugriffe zu sichern.

21.2 Firewall

Wie bei der namengebenden Anwendung als Schutz gegen die Ausbreitung von Bränden, hat ein Firewall die Aufgabe, ein privates Netz vor unkontrollierten Zugriffen aus anderen Netzen zu schützen. Oder – um ein anderes Bild zu gebrauchen – so wie eine mittelalterliche Burg mit Mauer, Tor und Zugbrücke wird ein Rechnernetz durch einen kontrollierten Zugang vor Angriffen von außen geschützt. Insofern versteht man unter Firewall das allgemeines Konzept einer strikten Trennung zwischen internen und externen Netzen. Nach vorgegebenen Regeln wird entschieden, welche Daten durchgelassen werden.

Eine Firewall kann durch ein eigenständiges Gerät (Rechner mit entsprechender Software oder spezielle Hardware-Firewall) realisiert werden. Diese Firewall wird in der Regel als Absicherung zwischen einem lokalen Netz und dem Internet

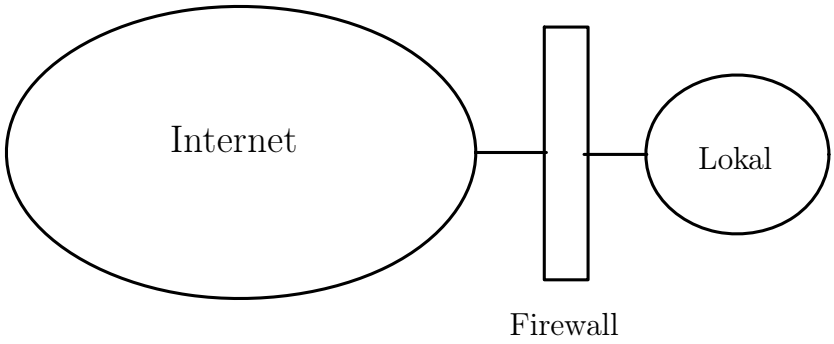


Abbildung 21.1: Firewall zwischen Internet und lokalem Netz

geschaltet. Man kann dies auch als einen Router zwischen den beiden Netzen sehen, der allerdings zusätzlich den Datenverkehr kontrolliert.

Am einfachsten aufgebaut sind filterbasierte Firewalls (Paketfilter). Die Regeln beziehen sich dann auf bestimmte Adressen. Eine Möglichkeit ist, eine Tabelle mit erlaubten Quell- und Zieladressen zu führen. Als Prinzip gilt dann: *Alles, was nicht ausdrücklich erlaubt ist, ist verboten* . Dann würde eine Zeile

169.254.107.238, 123, 212.125.100.86, 80

bedeuten, dass alle Pakete vom Rechner 169.254.107.238 und Port 123 zu Rechner 212.125.100.86, Port 80 passieren dürfen. Als Alternative kann man auch das liberale Konzept: *Alles, was nicht ausdrücklich verboten ist, ist erlaubt* einsetzen. In diesem Fall würde der Eintrag genau diese Kombination verbieten.

Ein Paketfilter arbeitet auf den unteren Protokollebenen. In dem beschriebenen Beispiel werden Pakete bis zur Schicht 4 analysiert, um dann anhand der IP-Adresse und der Port-Nummer über die Weiterleitung zu entscheiden. Daher werden sie auch als Network Level Firewall bezeichnet. Die Wirksamkeit einer Firewall lässt sich erhöhen, indem nicht nur auf Basis eines einzelnen Paketes entschieden wird, sondern eine Verbindung als ganzes betrachtet wird. Am weitesten gehen in diesem Sinne Application Level Firewalls (Proxy-basierte Firewalls).

In diesem Fall kommuniziert eine Anwendung – z. B. ein Browser – nicht direkt mit dem Internet. Vielmehr wird der gesamt Verkehr über einen Stellvertreter – den Proxy – abgewickelt. Nur der Proxy hat die Berechtigung, Daten mit dem Internet auszutauschen. Dieses Konzept erlaubt weit gehende Kontroll- und Protokollfunktionen und gilt als sehr sicher. Negativ ist eine gewisse Einschränkung der Flexibilität und verringerte Transfergeschwindigkeit. Als Alternative werden daher auch Mischformen zwischen den Filtern auf Paketebene und den Proxys auf Anwendungsebene eingesetzt.

Von außen ist der Firewall der einzige Angriffspunkt auf das lokale Netz. Daher wird bei der Administration besondere Sorgfalt auf die Sicherheit dieses Systems gelegt. Ist allerdings diese Hürde doch genommen, so befindet sich der

Eindringling bereits im relativ ungeschützten internen Netz. Um diese Gefahr zu verringern, kann das interne Netz durch eine zweite Firewall geschützt werden. Hinter der ersten Firewall sind dann nur Rechner angeordnet, die einen externen Zugang benötigen. Typischerweise werden hier Webserver und ähnliches zu finden sein. Der Zugang aus diesem Netz zu dem internen Netz mit den kritischen Daten wird dann durch eine zweite Firewall geschützt. Die Zone zwischen den beiden Firewalls bildet ein Grenznetz (auch De-Militarized Zone, DMZ) .

Zur Absicherung von einzelnen Rechnern werden so genannte *Personal* oder *Desktop Firewall* Lösungen eingesetzt. Solche Systeme werden häufig von Privatanwendern verwendet, um ihren direkt am Internet angeschlossenen Rechner zu schützen. Die Personal Firewall wirkt dann als Paketfilter, der unerwünschten Datenverkehr zwischen dem Rechner und dem Internet blockiert.

Allerdings darf man den Gewinn an Sicherheit nicht überschätzen. Man muss leider davon ausgehen, dass ein z. B. über Email eingedrungener Virus intelligent genug ist, um die Firewall von innen heraus zu überwinden. Beispielsweise könnte er eine Regel für eine weitere Verbindung eintragen und diese dann für eigene Zwecke nutzen. Eine Personal Firewall ist allerdings gut geeignet, um einen Überblick über die Netzwerk-Aktivitäten zu gewinnen.

21.3 Virtuelle private Netze

Firewalls dienen dazu, das eigene Netz gegenüber dem Internet abzuschotten. Nun gibt es allerdings Situationen, in denen gerade ein externer Zugang benötigt wird. Ein typischer Fall ist ein Mitarbeiter auf Dienstreise, der Zugriff auf seine internen Daten benötigt. Ein recht sicherer Weg besteht in der Verwendung von Wählleitungen. Die Verbindung geht dann über ein Modem und eine Telefonleitung zu einem entsprechenden Firmenanschluss. Dieser Anschluss wird durch ein Passwort geschützt.

Wesentlich eleganter ist es, auf die Infrastruktur des Internets zurück zu greifen. Der Mitarbeiter nutzt dann beispielsweise den WLAN-Zugang in seinem Hotel, um sich mit dem Internet zu verbinden. Umgekehrt muss die Firma einen Zugang vom Internet zu den internen Daten bereit stellen. Für den Weg dazwischen wird das Internet genutzt.

Das Problem ist nun, wie diese Verbindung sicher gestaltet werden kann. Der Weg durch das Internet ist nicht geschützt und es besteht die Gefahr, dass der Inhalt der IP-Pakete von Unberechtigten eingesehen oder gar manipuliert wird. Betrachten wir ein Szenario, bei dem ein externer Mitarbeiter (Client) auf einen Rechner X im internen Netz zugreifen möchte. Dazu muss er Daten an die Adresse X schicken. Da es sich um vertrauliche Informationen handelt, müssen die Daten verschlüsselt werden. Aber auch die Adresse X soll nicht öffentlich bekannt werden.

Eine in diesem Sinne sichere Verbindung wird durch einen so genannten *Tun-*

nel durch das Internet realisiert. Dazu wird das eigentliche Paket – Kopf und Rumpf – als Einheit verschlüsselt. Die entstehenden Daten werden dann als Nutzlast in einem IP-Paket an einen entsprechenden Server (Remote-Tunnel-Server) im Firmennetz geschickt. Aus Sicht des Internets handelt es sich also um eine IP-Verbindung zwischen Client und diesem Server. Die Daten in dem IP-Paket sind durch Verschlüsselung geschützt. Der Server nimmt das Paket entgegen und entschlüsselt die Daten. Damit rekonstruiert er das eigentliche Paket, das er dann in das interne Netz einspeisen kann.

Durch diese Technik kann eine Endanwendung sich so verhalten, als wäre der Rechner direkt an das Firmennetz angeschlossen (Virtuelles privates Netz, VPN). Die Pakete durchqueren das dazwischen liegende Internet eingebettet in ein IP-Paket. Auf diese Art und Weise kann eine sichere Kommunikation über das Internet erreicht werden. Dies ist nicht der einzige Vorteil. Der Inhalt des Paketes spielt für den Transport durch den IP-Tunnel keine Rolle. Es ist daher durchaus möglich, dass im firmeninternen Netz ein anderes Protokoll oder private IP-Adressen verwendet werden.

Eine komplette Lösung für den Aufbau eines Tunnels besteht aus den Komponenten für die einzelnen Funktionen wie z. B. Benutzer-Authentifikation oder Daten-Verschlüsselung. Einige verbreitete Tunneling-Protokoll sind:

- Point to Point Tunneling Protocol (PPTP)
- Layer 2 Tunneling Protocol (L2TP), RFC 2661
- IP Security Protokoll (IPSec)

Sie wurden für verschiedene Anwendungsmöglichkeiten entwickelt und unterscheiden sich dem entsprechend in ihren Möglichkeiten.

21.4 Verschlüsselung

21.4.1 Einleitung

Der Inhalt der Pakete im Internet ist relativ leicht einsehbar. Mit wenig Aufwand kann man auf einem Rechner beispielsweise alle ankommenden und abgehende IP-Pakete lesen. Ohne weitere Maßnahmen sind daher kritische Daten wie Kennwörter oder die Nummern von Kreditkarten nicht vor Unberechtigten geschützt. Im Internet hat man als Anwender keinen Einfluss auf den Übertragungsweg. Man muss vielmehr im Allgemeinen davon ausgehen, dass die Übertragung nicht abgesichert ist und der Inhalt der Pakete auch Fremden zugänglich ist.

Um trotzdem sensible Daten übermitteln zu können, muss man diese Daten durch eine Verschlüsselung schützen. Im Prinzip handelt es sich dabei um eine Art von Kodierung ähnlich wie der Einbau von Paritätsinformationen. Durch einen Verschlüsselungsalgorithmus wird die Nachricht – der Klartext – in eine

andere Form gebracht. Diese kodierte Nachricht wird im Netz übertragen. Der Empfänger kann den Verschlüsselungsalgorithmus umkehren und den Klartext rekonstruieren.

Die Besonderheit bei diesem Vorgehen ist allerdings, dass der Algorithmus geheim gehalten wird. Nur Sender und Empfänger kennen den Algorithmus (oder zumindest die verwendeten Parameter). Dritte können zwar während der Übertragung die Nachricht abfangen, aber ohne Kenntnis des Verschlüsselungsalgorithmus können sie die Nachricht nicht verstehen oder gar manipulieren. Anschaulich kann man sich die Nachricht wie ein Paket mit einem Schloss vorstellen. Der Sender verschliesst das Paket und nur mit dem richtigen Schlüssel kann es wieder geöffnet werden. Die Methoden der Verschlüsselung sind Gegenstand der Kryptographie. Demgegenüber bezeichnet Kryptoanalyse die Wissenschaft von der Entschlüsselung ohne Kenntnis des Schlüssels.

Im folgenden werden zunächst anhand eines einfachen Beispiels die Grundregeln eingeführt. Anschließend folgt der Übergang zu digitalen System mit einer Beschreibung einiger wesentlichen Verfahren. Darauf aufbauen werden einige Anwendungsbeispiele vorgestellt. In der Literatur zur Kryptographie werden häufig die Beteiligten als Personen mit den Namen *Bob* und *Alice* behandelt. Im Rahmen dieses Textes werden aber weiterhin die Bezeichnungen *Sender* und *Empfänger* verwendet. In symmetrischen Fällen, bei denen diese Unterscheidung nicht sinnvoll ist, wird durch die Abkürzungen A und B ersetzt.

21.4.2 Monoalphabetisch Verschlüsselung

Betrachten wir ein einfaches Beispiel für einen Algorithmus zur Verschlüsselung von Texten:

1. Die Buchstaben des Schlüsselwortes werden in umgekehrter Reihenfolge aufgeschrieben. Eventuelle doppelt vorkommende Buchstaben werden dabei nur einmal berücksichtigt.

2. Alle anderen Buchstaben des Alphabets werden ebenfalls in umgekehrter Reihenfolge angehängt.

Nach dieser Vorschrift wird aus dem Schlüsselwort HAMLET folgende Tabelle zur Verschlüsselung erzeugt:

A	B	C	D	E	F	G	H	I	J	K	L	M
T	**E**	**L**	**M**	**A**	**H**	Z	Y	X	W	V	U	S
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
R	Q	P	O	N	K	J	I	G	F	D	C	B

Anhand dieser Tabelle wird der zu übermittelnde Text umgesetzt. Jeder einzelne Buchstabe wird gemäß dieser Zuordnung durch einen anderen ersetzt. Als Beispiel ergibt sich folgende Verschlüsselung:

S	E	I	N		O	D	E	R		N	I	C	H	T	S	E	I	N
K	A	X	R		Q	M	A	N		R	X	L	Y	J	K	A	X	R

Die Leerzeichen verraten noch relativ viel über die Satzstruktur. Es ist daher eine gute Idee, entweder das Leerzeichen als weiteres Zeichen in die Tabelle aufzunehmen oder einfach alle Leerzeichen zu entfernen. In aller Regel wird der Empfänger nach der Entschlüsselung aller Buchstaben die Wortgrenzen leicht rekonstruieren können. Ohne Leerzeichen erhält man in dem Beispiel die verschlüsselte Nachricht

KAXRQMANRXLYJKAXR

Für den Uneingeweihten handelt es sich hier um eine sinnlose Buchstabenfolge. Der Empfänger kann demgegenüber - sofern er das Schlüsselwort kennt - den Prozess der Verschlüsselung umkehren. Dazu konstruiert er zuerst nach der obigen Vorschrift ebenfalls die Zuordnungstabelle. Dann ersetzt er die empfangenen Zeichen wieder durch die ursprünglichen Zeichen. Aus der resultierenden Buchstabenfolge wird er leicht durch Einfügen der Leerzeichen den ursprünglichen Text rekonstruieren können.

Da nur ein Alphabet zur Verschlüsselung benutzt wird, bezeichnet man derartige Verfahren als monoalphabetisch. Weiterhin handelt es sich um ein symmetrisches Verfahren: Es gibt genau einen Schlüssel, der sowohl zum Verschlüsseln als auch zum Entschlüsseln benötigt wird. Sender und Empfänger benötigen die gleichen Information. Kennt ein Unbefugter den Schlüssel, so kann er ebenfalls jede Nachricht entschlüsseln.

Solche Verfahren waren in unterschiedlichen Varianten jahrhundertelang in Gebrauch. Allerdings bieten sie nur einen geringen Schutz. Die Verschlüsselung kann relativ leicht durch den Einsatz der Häufigkeitsanalyse gebrochen werden. Dabei nutzt man die unterschiedliche Häufigkeit für das Auftreten der einzelnen Buchstaben aus. So ist beispielsweise in deutschen Texten im Mittel E der häufigste Buchstabe. Damit kann man davon ausgehen, dass der häufigste Buchstabe in einem derart verschlüsselten Text mit recht hoher Wahrscheinlichkeit eigentlich das E darstellt. Weiterhin kann man nach häufig vorkommenden Buchstabenpaaren suchen. So sind die Kombinationen LL, MM, NN, EE oder TT gute Kandidaten für häufig doppelt vorkommende Zeichen.

Auch immer wiederkehrende längere Folgen wie SCH bieten Ansatzpunkte für die Entschlüsselung. Auf diese Art und Weise kann durch systematisches Raten unter Berücksichtigung der bekannten Häufigkeiten von Buchstaben und Buchstabenfolgen eine monoalphabetische Verschlüsselung relativ leicht entschlüsselt werden. Eine besondere Hilfe ist dabei, wenn man einige der Wörter im Klartext kennt. Handelt es sich beispielsweise um einen Wetterbericht, so kommt darin mit einiger Wahrscheinlichkeit auch das Wort WETTER vor. Man kann dann gezielt nach der Folge ETTE beziehungsweise einem Muster XYYX suchen. Findet man ein Muster XYYX, so kann man die Zuordnung X=E und Y=T ausprobieren.

Derartige einfache Verschlüsselungsverfahren sind damit nicht vollkommen nutzlos geworden – aber sie bieten nur einen geringen Schutz. Ob dieser Schutz

als ausreichend erachtet werden kann, hängt von der Anwendung ab. Soll nur eine sehr kurze Nachricht geschützt werden oder ist der Schutz nur für eine kurze Zeit erforderlich, so kann eine solche einfache Verschlüsselung durchaus ausreichen. Die verschlüsselte Nachricht ist zunächst gegenüber Unbefugten geschützt. Jemand, der zufällig oder absichtlich die Nachricht liest, erkennt nicht direkt den Inhalt.

21.4.3 Digitale Verschlüsselung

Mit dem Aufkommen von Rechenautomaten und Computern änderte sich zunächst nichts am Prinzip. Allerdings können sowohl die Verschlüsselung als die Versuche zum Aufbrechen des Codes durch den Einsatz von Rechnern dramatisch beschleunigt werden. Damit sind einerseits aufwändigere Verfahren möglich und andererseits verkürzt sich die Zeitspanne, für die ein Code Sicherheit bietet. Die ersten Rechner wurden während des zweiten Weltkrieges in den britischen Geheimlaboren als Hilfsmittel zur Entschlüsselung des deutschen Funkverkehrs eingesetzt [Sin00]. Genauer gesagt, wurde diese Rechner speziell für diesen Zweck entwickelt.

Mit der beginnenden Digitalisierung wurden entsprechende neuartige Verschlüsselungsverfahren benötigt. Diese Verfahren arbeiten nicht mehr mit Zeichen sondern mit einzelnen Bits. Die Nachricht wird unabhängig von ihrem Inhalt als ein Bitstrom behandelt. Bei den im folgenden vorgestellten Verfahren handelt es sich um so genannte Blockcodes. Dabei wird die Nachricht in entsprechende viele einzelne Blöcke mit jeweils einer gegebenen Anzahl von Bit unterteilt. Jeder Block wird dann durch Bit-Manipulationen verschlüsselt. Ein solches Verfahren ist der Data Encryption Standard (DES). Der Algorithmus wurde Mitte der 70er Jahre entwickelt und später vom amerikanischen National Institute of Standards and Technology (NIST) als Standard eingeführt [Nat80].

Der Algorithmus beruht auf der Verwürfelung von Gruppen von jeweils 64 Bit. In einem mehrstufigen Verfahren werden die einzelnen Bit miteinander verknüpft und mit den Ergebnissen überschrieben. In DES werden insgesamt 16 Iterationen – auch als Runden bezeichnet – durchgeführt. Die Details der Verknüpfungsfunktion sind durch den Schlüssel festgelegt. Die Länge eines Schlüssels beträgt 56 Bit. In jeder Runde wird aus diesem Hauptschlüssel ein spezifischer Schlüssel berechnet.

Im Grunde ist DES in dieser Basisversion ein monoalphabetisches Verfahren. Es bildet Wörter der Länge 64 auf andere Wörter der gleichen Länge ab. Kennt man die Struktur der ursprünglichen Daten, so eröffnet diese Eigenschaft die Möglichkeit zur Manipulation. Handelt es sich beispielsweise um Datensätze für Mitarbeiter, so kann man einen Block von 64 Bit einfach durch einen anderen Block ersetzen. Auf diese Art und Weise könnte man beispielsweise das eigene Gehalt durch das eines anderen - besser verdienenden - Kollegen ersetzen, indem man den Block mit den entsprechenden Feldern aus dem fremden Datensatz kopiert. Um derartige Manipulationen auszuschließen, besteht die Möglichkeit, in

die Verschlüsselung eine Abhängigkeit von vergangenen Blöcken einzubauen. Damit ergibt ein bestimmter Datenblock in Abhängigkeit vom Kontext unterschiedliche Codes und die Daten sind gegen einfachen Austausch geschützt. Generell führt die Verkettung zu einem besseren Schutz gegen Kryptoanalyse.

Durch den Schlüssel der Länge 56 sind insgesamt 2^{56} verschiedene Abbildungen möglich. Kennt man ein kurzes Stück Klartext, so kann man durch erschöpfende Suche alle 2^{56} möglichen Schlüssel ausprobieren. Geht man davon aus, dass ein handelsüblicher PC in der Größenordnung eine Million Schlüssel pro Sekunde ausprobieren kann, so berechnet sich eine Zeit von etwa 2300 Jahren für die Suche über alle möglichen Schlüssel. Im Mittel wird der richtige Schlüssel dann nach 1250 Jahren gefunden.

Diese Zeit klingt zunächst beruhigend. Allerdings lässt sich die Aufgabe in einfachster Art und Weise parallelisieren und auf viele Rechner aufteilen. Mit entsprechend vielen Rechnern lässt sich somit die Verschlüsselung in überschaubarer Zeit aufbrechen. Weiterhin kann durch spezielle Hardware-Bausteine die Suche wesentlich beschleunigt werden. Insgesamt kann man daher DES zumindest für kritische Anwendungen nicht mehr als hinreichend sicher betrachten.

Eine ausreichende Sicherheit kann durch einen längeren Schlüssel oder die dreifache Anwendung von DES mit 2 oder 3 verschiedenen Schlüsseln wieder hergestellt werden. Daneben wurden eine Reihe anderen Verfahren vorgeschlagen. Das wohl wichtigste darunter ist IDEA (International Data Encryption Algorithm), entwickelt von Xuejia Lai und James L. Massey von der ETH Zürich [LM90]. IDEA basiert auf einem Schlüssel der Länge 128 Bit. Die erschöpfende Suche auf einem einzelnen PC würde etwa 10^{25} Jahre benötigen. Damit besteht ausreichend Sicherheit auch gegen Angriffe mit optimierten Hardware-Bausteinen und parallelen Einsatz vieler Rechner.

21.4.4 Gemeinsame Schlüsselvereinbarung

Grundsätzlich bieten die beschriebenen Verfahren eine ausreichende Sicherheit. Sind die verwendeten Schlüssel groß genug, so können nach derzeitigem Stand die Verschlüsselungen nicht aufgebrochen werden. Das Problem dieser Verfahren liegt in der Verwaltung der Schlüssel. Will ein Sender eine verschlüsselte Nachricht verschicken, muss er dem Empfänger seinen Schlüssel mitteilen oder einen vom Empfänger zuvor erhaltenen Schlüssel verwenden. Neben der eigentlichen Nachricht muss also auch der Schlüssel übertragen werden. Diese Übertragung ist besonders sensibel. Sowie ein Dritter in Besitz des Schlüssels gelangt, kann er ebenfalls die Nachricht lesen. Daher muss der Schlüssel auf einem besonders sicheren Weg verschickt werden. Beispielsweise kann eine Bank die Schlüssel mit einem zuverlässigen Boten an ihre Kunden ausliefern.

Mit dem zunehmenden Nachrichtenverkehr stellte die Schlüsselverteilung ein immer größeres logistisches Problem dar. Um so wichtiger war die Entdeckung, dass eine Schlüsselvereinbarung auch über einen unsicheren Kanal möglich ist.

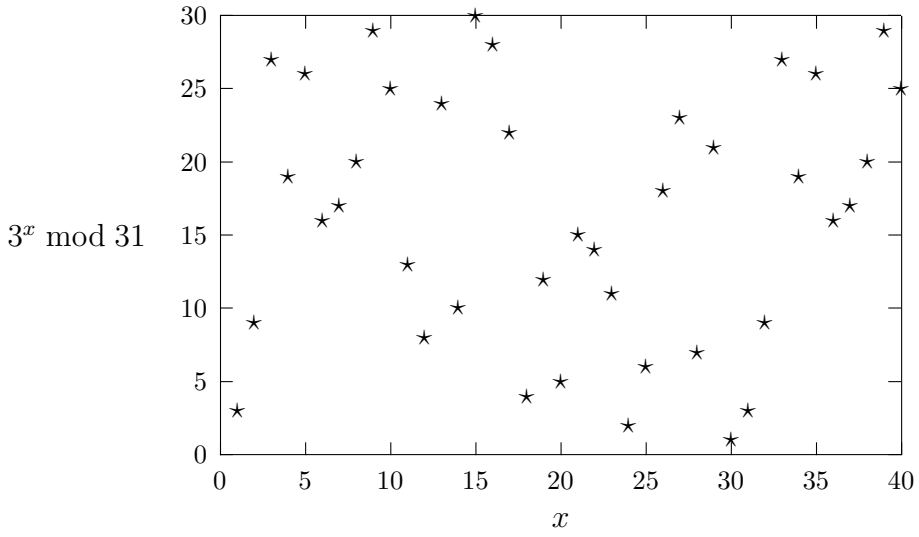


Abbildung 21.2: Beispiel für eine Modulo-Funktion.

Sender und Empfänger vereinbaren dabei einen Schlüssel, der dann wie gehabt für eine symmetrische Verschlüsselung verwendet wird. Die Vereinbarung kann über einen offenen Kanal erfolgen. Ein Dritter kann aus den übertragen Informationen nicht den Schlüssel rekonstruieren. Diese zunächst verblüffende Möglichkeit wurde zuerst von Whitfield Diffie und Martin Hellmann publiziert [DH76]. Der Kerngedanke ist, eine mathematische Funktion zu verwenden, die praktisch nicht umkehrbar ist. Eine solche „Einwegfunktion“ kann zwar für ein Argument leicht ausgewertet werden. Aber die Umkehrung – die Berechnung des Argumentes aus dem Funktionswert – ist nicht in vernünftiger Zeit möglich. In der Schlüsselvereinbarung nach Diffie und Hellmann (manchmal auch als Diffie-Hellmann-Merkle-Verfahren bezeichnet) und in vielen weiteren Ansätzen wird die Modulo-Funktion verwendet, um den Einweg-Charakter zu erzielen.

Die Modulo-Funktion berechnet den Rest nach ganzzahliger Division. Der Ausdruck $17 \bmod 5$ als Beispiel ergibt den Wert 2, da $17 = 3 * 5 + 2$ gilt. Bei arithmetischen Ausdrücken werden die Operanden zunächst wie gewohnt verknüpft und auf das Ergebnis wird die Modulo-Funktion angewandt. So ergibt beispielsweise $3 * 3 \bmod 5 = 4$. Das Resultat der Exponentialfunktion $3^x \bmod 31$ für verschiedene Werte von x zeigt Bild 21.2. Anders als bei herkömmlichen Funktionen ist hier keine klare Struktur zu erkennen. Aufgrund der Modulo-Funktion schwanken die Ergebnisse selbst für direkt aufeinander folgende Werte von x stark.

Der Einweg-Charakter der Modulo-Funktion resultiert aus der Schwierigkeit, die Funktion zu invertieren. Es ist zwar relativ leicht, für ein gegebenes x die Exponentialfunktion zu berechnen und die Modulo-Operation anzuwenden. Aber

Tabelle 21.1: Schlüsselvereinbarung nach Diffie und Hellmann zwischen Partner A und Partner B

A	öffentlich	B
	$Y = 7$, $P = 11$	
$A = 3$		$B = 6$
$Y^A \bmod P$ $7^3 \bmod 11 = 2$		$Y^B \bmod P$ $7^6 \bmod 11 = 4$
	$\alpha = 2$, $\beta = 4$	
$\beta^A \bmod P$ $4^3 \bmod 11 = 9$		$\alpha^B \bmod P$ $2^6 \bmod 11 = 9$

für die Umkehrung ist kein schnelles Verfahren bekannt. Die Aufgabe, für einen Wert a ein x zu finden, so dass $3^x \bmod 31 = a$ erfüllt wird, erfordert im Allgemeinen einen sehr viel höheren Rechenaufwand. Bei großen Zahlen mit mehreren hundert Stellen lässt sich mit geschickten Algorithmen die Exponentialfunktion immer noch in überschaubarer Zeit auswerten. Aber die Umkehrung erfordert dann selbst bei leistungsfähigen Systemen so viel Zeit, dass sie als praktisch unmöglich anzusehen ist.

Betrachten wir die Schlüsselvereinbarung nach Diffie und Hellmann an einem einfachen Beispiel. In Tabelle 21.1 sind die einzelnen Schritte dargestellt. Zunächst vereinbaren die beiden Partner A und B zwei Zahlen Y und P. Diese Zahlen müssen bestimmte Bedingungen erfüllen. So müssen Y, $(Y-1)/2$ und P Primzahlen sein. Diese Zahlen können öffentlich bekannt sein. Dann wählt jeder Partner ein geheime Zahl aus. Jeder Partner berechnet dann aus seiner Zahl den Wert der Funktion $Y^x \bmod P$. Das Resultat α beziehungsweise β schickt er an sein Gegenüber. Schließlich berechnet er aus dem gerade erhaltenen Wert und seiner Geheimzahl den Schlüssel. Durch die geschickte Konstruktion der Funktion erhalten beide Partner den gleichen Wert – im Beispiel die Zahl 9. Diese Zahl können sie als symmetrischen Schlüssel verwenden. In der Praxis werden die Zahlenwerte sehr viel größer sein, um einen Schlüssel ausreichender Länge zu erhalten.

Die Wirkungsweise dieses Verfahrens lässt sich leicht nachvollziehen. Ausgeschrieben ergeben die beiden Berechnungen $(Y^B \bmod P)^A \bmod P$ und $(Y^A \bmod P)^B \bmod P$. Nach den Regeln der modularen Arithmetik kann der zweite Exponent in den ersten Ausdruck gezogen werden. Damit sind beide Ausdrücke gleichwertig mit $Y^{AB} \bmod P$.

Ein Dritter kann über die öffentliche Leitung die Zahlen Y, P, α und β erfahren. Um den Schlüssel zu berechnen, fehlt ihm die Zahl A (oder gleichwertig B). Daher müsste die Funktion

$$Y^A \bmod P = \alpha \tag{21.1}$$

nach A aufgelöst werden. Dafür ist wie gesagt kein effizientes Lösungsverfahren

bekannt. In dem Beispiel mit den kleinen Zahlen könnte aus der Gleichung

$$7^A \bmod 11 = 2 \tag{21.2}$$

der gesuchte Wert $A = 3$ noch relativ leicht durch Probieren bestimmt werden. Aber für große Zahlenwerte führt dies zu so langen Zeiten, dass die Umkehrung als praktisch nicht möglich angesehen werden kann.

Das Verfahren von Diffie und Hellmann ermöglicht es, über eine Leitung einen gemeinsamen Schlüssel zu vereinbaren ohne dass die übertragenen Informationen von Dritten verwendet werden können. Allerdings ist es noch nicht sicher genug für unsichere Kanäle. Wenn ein Dritter X die Möglichkeit hat, Nachrichten zu verändern, kann er sich in das Austausch-Protokoll einschalten. Er wählt eine eigene Zufallszahl und führt damit das Protokoll mit den Partnern A und B durch. Dann entstehen sichere Verbindungen zwischen A und X sowie B und X. A und B sind im besten Glauben, ihre Nachrichten direkt aneinander zu senden. In Wirklichkeit erreichen die Nachrichten immer zuerst X, der sie dann nach Belieben manipulieren kann bevor er sie weiter gibt. Eine Möglichkeit sich gegen solche aktiven Angriffe zu schützen, ist die Verwendung fester Schlüssel, die bei einer zuverlässigen Stelle hinterlegt werden.

21.4.5 Öffentliche Schlüssel

Ein praktischer Nachteil bleibt bei dem Verfahren der Schlüsselvereinbarung: beide Partner müssen gleichzeitig aktiv sein, um das Protokoll durchführen zu können. Damit ist das Verfahren beispielsweise zum Versand einer Email nur schlecht einsetzbar. Einfacher zu verwenden sind Verfahren, bei denen der Absender seine Nachricht verschlüsseln kann, ohne immer vorher mit dem Empfänger direkt Informationen austauschen zu müssen. Dies ermöglichen Verfahren mit öffentlichen Schlüssel (*public key*) [Dif88]. Hierbei stellt der Empfänger einen Schlüssel bereit, der allgemein bekannt gemacht wird. Der Sender verwendet diesen Schlüssel, um seine Nachricht zu verschlüsseln. Der öffentliche Schlüssel hilft nicht zu Entschlüsselung. Benötigt wird vielmehr ein zweiter Schlüssel. Diesen Schlüssel kennt nur der Empfänger, so dass auch nur er die Verschlüsselung aufheben kann. Solange er ihn geheim hält, kann niemand sonst die Nachricht verstehen.

Das bekannteste derartige Verfahren ist RSA, so benannt nach den Entwicklern Ronald Rivest, Adi Shamir und Leonhard Adleman. RSA basiert ebenfalls auf der Modulo-Arithmetik. Der öffentliche Schlüssel besteht aus zwei Zahlen, die wie folgt gewählt werden:

- Der potentielle Empfänger A wählt zwei Primzahlen p und q.
- Aus den beiden Primzahlen berechnet er das Produkt $n = p \cdot q$.
- Weiterhin wählt er eine Zahl $e < n$.

- e und n bilden den öffentlichen Schlüssel und werden entsprechend publiziert.

Damit kann ein Sender eine Nachricht an A nach der folgenden Vorschrift verschlüsseln

$$C = M^e \bmod n \tag{21.3}$$

Die Zahl $M < n$ repräsentiert einen Block mit der Länge $\ln_2 n$ Bit. Eine längere Nachricht muss in entsprechen viele Teile aufgespalten werden. Die resultierende Zahl C ist der geheime Text für diesen Block. Aufgrund der Eigenschaften der Modulo-Funktion ist es sehr schwer, aus C und e wieder auf M zu schließen. Bei hinreichend großen Zahlen n und e kann man die Umkehrung als praktisch unmöglich betrachten.

Damit ist die Verschlüsselung sicher, aber es stellt sich die Frage, wie der Empfänger die Nachricht lesen kann. Hier kommen die beiden wohlweislich geheim gehaltenen Zahlen p und q ins Spiel. Mit diesen beiden Zahlen kann eine weiter Zahl d mit der Eigenschaft

$$e \cdot d = 1 \bmod ((p-1) \cdot (q-1)) \tag{21.4}$$

berechnet werden. Details dazu sind im Anhang **??** beschrieben. An dieser Stelle genügt es festzustellen, dass die Bestimmung von d relativ einfach ist. Bei dieser Wahl des Produktes $e \cdot d$ gilt weiterhin die Beziehung

$$C^d = M^{e \cdot d} \bmod n = M \tag{21.5}$$

Auch hier sei für Details auf den Anhang **??** verwiesen. Mit anderen Worten: der Empfänger braucht lediglich einen Exponenten zu berechnen und auf das Ergebnis die Modulo-Funktion anzuwenden, um gemäß

$$M = C^d \bmod n \tag{21.6}$$

den ursprünglich Wert M zu erhalten. Die Sicherheit von RSA beruht auf den beiden Zahlen p und q. Kennt man sie oder zumindest das Produkt $(p-1)\cdot(q-1)$, so ist es leicht daraus d zu bestimmen. Öffentlich bekannt ist aber nur der Wert $n = p \cdot q$. Der Rückschluss von n auf die beiden Faktoren – die so genannte Faktorzerlegung – ist ein enorm zeitaufwändiges Verfahren. Solange niemand eine schnelle Methode zur Suche nach den Primfaktoren findet – ein Problem an dem die Mathematiker seit mehr als 2000 Jahren arbeiten – ist RSA ein sicheres Verfahren. Für sicherheitskritische Anwendungen werden Zahlen größer als 10^{300} eingesetzt. Die weiteren Fortschritte in der Rechengeschwindigkeit können durch Erhöhung von n leicht kompensiert werden.

Der Ausdruck (21.5) weist eine interessante Symmetrie bezüglich e und d auf. In dem Produkt spielt die Reihenfolge keine Rolle. Man kann also durchaus auch mit dem geheimen Schlüssel d verschlüsseln und dann mit dem öffentlichen

Schlüssel e den Vorgang umkehren. Damit ist keine Vertraulichkeit möglich, da jeder Zugang zu dem öffentlichen Schlüssel hat. Aber diese Eigenschaft kann zum Signieren von Nachrichten genutzt werden. Wenn eine Nachricht zu dem öffentlichen Schlüssel passt, muss sie mit dem zugehörigen privaten Schlüssel erzeugt worden sein. Damit kann nur der Besitzers des Schlüssels der Absender sein.

21.5 Anwendungen

Ausgehend von den Algorithmen lassen sich Anwendungen realisieren. Für die Praxis spielen neben den theoretischen Eigenschaften auch Fragen wie der benötigte Rechenaufwand oder die sichere Verwaltung der Schlüssel eine entscheidende Rolle. So hat RSA bei allen Vorzügen den Nachteil eines relativ hohen Rechenbedarfs. Damit ist es nicht praktikabel, alle Nachrichten mit RSA zu schützen. Weiterhin sind legale Aspekte wie z. B. Patentsituation oder Ausfuhrgesetze zu beachten.

21.5.1 PGP

Eine freie, weit verbreitete Software für die Verschlüsselung von Emails ist PGP (*Pretty Good Privacy*) [ZCB99]. Phil Zimmermann entwickelte mit PGP ein einfach zu bedienendes Programm, in dem mehrere Verschlüsselungsverfahren kombiniert werden. Die eigentliche Nachricht wird mit einem aufwandsgünstigen symmetrischen Verfahren wie DES kodiert. Dazu verwendet wird ein einmaliger Schlüssel, der wiederum mit dem öffentlichen Schlüssel des Empfängers geschützt wird. Der Empfänger – und nur der Empfänger – kann den Einmalschlüssel lesen und damit die gesamte Nachricht entschlüsseln.

Der Sender fügt noch eine mit seinem eigenen privaten Schlüssel kodierte Signatur an, die eine Prüfsumme über die Nachricht enthält. Damit kann der Empfänger sicher stellen, dass die Nachricht tatsächlich vom angegebenen Sender stammt und der Text nicht verändert wurde. In PGP integriert ist die Verwaltung der öffentlichen Schlüssel in einem so genannten *Key-Ring* (Schlüsselbund), wobei die Schlüssel nach ihrer Vertrauenswürdigkeit bewertet werden.

21.5.2 Sichere Transportschicht

Eine flexible Möglichkeit für verschiedenste sicherheitskritische Anwendungen wird durch eine sichere Transportschicht bereit gestellt. Bild 21.3 zeigt diesen Ansatz am Beispiel von HTTP. Ein zusätzliches Protokoll wird zwischen TCP und Anwendungsprotokoll eingeschoben. Das Protokoll verwendet die Funktionalitäten von TCP und ergänzt sie um eine Sicherung. Aus Sicht der Anwendung ändert sich praktisch nichts, außer dass jetzt die Transportschicht sicher

<table>
<tr><td>HTTPS</td><td>Anwendung</td></tr>
<tr><td>Sichere Transportschicht</td><td rowspan="2">Transport</td></tr>
<tr><td>TCP</td></tr>
<tr><td>IP</td><td>Netzwerk</td></tr>
<tr><td>...</td><td></td></tr>
</table>

Abbildung 21.3: Sichere Transportschicht im Protokollstapel

ist. Zur Kennzeichnung hängt man an die Abkürzung des Anwendungsprotokoll den Buchstaben S für *Secure* an (z. B. HTTPS für HTTP Secure).

Der IETF-Standard dazu ist TLS (*Transport Layer Security*), eine Weiterentwicklung des ursprünglich von der Firma Netscape Communications entwickelten Protokolls SSL (*Secure Sockets Layer*). Der Ablauf entspricht wieder den bereits besprochenen Grundprinzipien. Der Client benutzt den öffentlichen Schlüssel des Servers, um den einen so genannten session-key zu übermitteln. Dieser session-key ist ein einmaliger Schlüssel, gültig nur für die aktuelle Verbindung, für eine symmetrische Verschlüsselung.

21.6 Übungen

Übung 21.1 *Der folgende Text wurde mit dem beschriebenen, einfachen monoalphabetischen Verfahren verschlüsselt. Können Sie mit der Häufigkeitsanalyse die Verschlüsselung aufheben? Wie lautet das Schlüsselwort?*

```
SRNAZNKERTPNRLMZWZYDMNZYMXNLAVSNBKXPWZCP
NWJVWNRWEZCANWJNOMCAYKNMMNYKWTNWAZLMVTZO
NRWTZWTRWSRNYRLNOZLKOTNEKWSNWSNOZXNORPZW
RMCANMCAORELMLNYYNONSTZOZYYNWUVNHNMCAONR
HLRWSNONOBZNAYKWTSNOTVYSPZNENOGRNSKOCASR
NMNXNLAVSNNRWTNANRXNOUYZWEKNONRWNWTVYSMC
AZLBNWLMCAYKNMMNYLGROS
```

Anhang A

Einheiten

Die kleinste Informationseinheit ist eine einzelne Ja/Nein-Entscheidung. Dieser Wert – Ja oder Nein beziehungsweise 1 oder 0 – ist ein Bit. Eine Gruppe von 8 Bit bilden ein Byte. Die Verarbeitungsbreite eines Prozessors bezeichnet man als Wort. Ein Wort hat demnach bei dem einfachen 8086 Prozessor 16 Bit und bei dem Pentium 32 Bit. Üblich ist die Abfolge von Zweierpotenzen 8, 16, 32, 64 und 128 Bit. Eine Ausnahme bilden die Serie 56000 Signalprozessoren der Firma Motorola mit 24 Wortbreite.

Die Angaben bei größeren Speicherbereichen werden mit entsprechenden Präfixen gebildet. Bei anderen Größen wie z.B. Längen werden Potenzen von 10 verwendet und es gilt z.B. die Umrechnung 1 km = 10^3m. Im Gegensatz dazu basieren die Angaben bei Speichergrößen auf Potenzen von 2. Der Präfix *kilo* entspricht dann $2^{10} = 1024$. Für die gängigen Größenordnungen ergibt sich:

Präfix		Potenz	Wert
K	Kilo	2^{10}	1024
M	Mega	2^{20}	1.048.576
G	Giga	2^{30}	1.073.741.824
T	Tera	2^{40}	1.099.511.627.776

Der Unterschied zwischen den beiden Systemen ist nicht sehr groß und kann bei vielen Betrachtungen vernachlässigt werden.

Anhang B

Aufgaben

Für den Schwierigkeitsgrad der Aufgaben gilt:
- $\star$ Leicht
- $\star\star$ Mittel
- $\star\star\star$ Schwer

Übung B.1 $\star$ *Die in England unter Mitarbeit von Alan Turing entwickelten COLOSSUS Rechner gehörten zu den ersten* echten *Computern. Für welches Einsatzgebiet wurden sie gebaut?*

Übung B.2 $\star$ *Ordnen Sie die folgenden Ereignis nach ihrer zeitlichen Reihenfolge (frühestes zuerst):*

A *Der erste IBM PC*

B *Mit der Verbindung von vier Großrechner in Kalifornien und Utah beginnt das ARPANET, der Vorläufer des Internets*

C *Erste Auswertung einer Volkszählung mit Lochkarten*

D *Steve Wozniak und Steve Jobs gründen mit 1300 $ Startkapital die Firma Apple*

Reihenfolge:

Übung B.3 $\star$ *Wandeln Sie die Binärzahlen in Oktal- und Hexadezimalsystem (ohne Taschenrechner).*

Dualsystem	*Oktalsystem*	*Hexadezimalsystem*
10100101		
00100011		
10001001		

Übung B.4 ⋆ *Ergänzen Sie die Wertetabelle für den Ausdruck*

$$a \cdot \overline{b} + b \cdot \overline{c}$$

(Es können mehr Spalten als benötigt vorhanden sein.)

a	b	c						

Übung B.5 *Woran erkennt man bei der Zahlendarstellung im 2er-Komplement, ob es sich um eine positive oder negative Zahl handelt?*

Übung B.6 *Welche Kompressionsfaktoren (qualitative Angaben wie* gar nicht, niedrig, mittel, hoch*) erwarten Sie bei der Kompression mit zip oder einem ähnlichen Packer für folgende Dateitypen:*

- *MPEG Videos*
- *Grafiken als Bitmap abgespeichert*
- *Java Quellcode*

(Begründung angeben).

Übung B.7 ⋆ *Wo „merkt" sich der Intel 8086-Prozessor die Rücksprungadresse eines Unterprogramms?*

Übung B.8 ⋆ *Was ist ein Register?*

Übung B.9 ⋆⋆ *Was bedeutet* ***Indirekte Adressierung****?*

Übung B.10 ⋆ *Worin unterscheiden sich Level 1 und Level 2 Cache?*

Übung B.11 ⋆⋆ *Erläutern Sie das Grundprinzip von RISC-Architekturen.*

Übung B.12 ⋆⋆⋆ *Wie wirken sich häufige Interrupts auf die Wirksamkeit einer mehrstufigen Pipeline aus?*

Übung B.13 ⋆⋆ *Welchen Inhalt hat das Register AX nach jedem Schritt des folgenden Programms? Geben Sie den Wert als Hexadezimalzahl an.*

```
mov ax,4        ; Wert in AX:
add ax,0ah      ; Wert in AX:
inc ah          ; Wert in AX:
dec al          ; Wert in AX:
```

Übung B.14 ⋆ *Welche Adresse berechnet sich im Real-Mode aus der Segmentadresse* $4F53h$ *und dem Offset* 4*?*

Übung B.15 ⋆⋆ *Was ist ein* master boot record*?*

Übung B.16 ⋆ *Wie nennt man einfache Prozessoren zur Steuerung von Geräten wie z. B. Kühlschrank, Waschmaschine oder Drucker?*

Übung B.17 ⋆ *Nennen Sie ein wichtiges Kriterium für einen Prozessor, der in einer Spielekonsole eingesetzt werden soll?*

Übung B.18 ⋆ *Was besagt das Gesetz von Moore?*

Übung B.19 ⋆⋆ *Worin unterscheidet sich der Ablauf von Unterprogramm und Software-Interrupt?*

Übung B.20 ⋆⋆ *Geben Sie zwei Beispiele für die Auslösung von Hardware-Interrupts an.*

Übung B.21 ⋆⋆ *Nennen Sie je ein Beispiel für Gleitkomma-Ausdrücke, bei denen als Resultat der Sonderfall* `NaN` *beziehungsweise* `Inf` *auftritt.*

Übung B.22 ⋆⋆ *Welchen Wert als Dezimalzahl hat* 11.101_2*?*

Übung B.23 ⋆ *Skizzieren Sie den Aufbau eines von Neumann-Rechners.*

Übung B.24 ⋆ *Was kennzeichnet einen Prozessor mit Harvard-Architektur?*

Übung B.25 ⋆⋆⋆ *Untersuchen Sie das folgende Assembler-Programm:*

```
        mov di,0

next:   mov al,text[di]
        cmp al,0
        jz fertig
        or  al,00100000b
        mov text[di],al
        inc di
        jmp next
fertig: ret

text  db 'AbCd',0
```

- *Wie wird der Text verändert? Welche allgemeine Aufgabe führt das Programm aus?*
- *Was bedeutet der Ausdruck* `text[di]`*?*
- *Wann wird die Verarbeitung beendet?*

Übung B.26 ⋆ *Eine Null-Adressmaschine führt das nachstehende Assembler-Programm aus:*

```
push 3  ; 1.
push 4  ; 2.
mpy     ; 3. Multiplikation
push 5  ; 4.
push 2  ; 5.
mpy     ; 6. Multiplikation
add     ; 7. Addition
```

Welchen Inhalt hat der Stack nach jeder Instruktion?

1.	*2.*	*3.*	*4.*	*5.*	*6.*	*7.*

Übung B.27 ⋆⋆ *In der allgemeinen Betrachtung hatten wir die Unterscheidung von Dreiadressmaschine bis Nulladressmaschine kennen gelernt. Wo würden Sie den Intel 8086 Prozessor einordnen? Begründen Sie Ihre Entscheidung. Ist die Zuordnung eindeutig?*

Übung B.28 ⋆⋆ *Nach welchem Prinzip ist ein Cache-Speicher organisiert, damit schnell festgestellt werden kann, ob ein Wert schon abgespeichert wurde? Welcher Nachteil ist damit verbunden?*

Übung B.29 ⋆
Die Datei mit den Folien zur Vorlesung ist 2,15 MByte groß.

1. *Wie viele Sekunden beträgt die Übertragungsverzögerung der Datei bei:*
   (a) *einer 100 Mbit/s Leitung*
   (b) *einer ADSL Verbindung mit 2000 kbit/s*
   (c) *einer ISDN Verbindung mit 128 kbit/s*

*Rechnen Sie bei der Dateigröße mit 1 kByte = 1024 Byte und 1 MByte = 1024*1024 Byte.*

2. *Sie haben die Datei als Anhang in einer Email erhalten und laden sie jetzt vom Server. Nennen Sie zwei Gründe, warum die Übertragung länger als oben berechnet dauert – selbst wenn die volle Bandbreite zur Verfügung steht und keine anderen Übertragungen gleichzeitig stattfinden.*

Übung B.30 ⋆ *Wofür wird bei TCP eine Schätzung der Round Trip Time RTT benötigt?*

Übung B.31 *Bitte markieren Sie durch Ankreuzen bei den folgenden Adresse*

a	*b*	*c*	*Adresse*
			22-B2-FF-GG-21-34
			123.200.20.2
			00-0A-E4-A2-4B-88
			255.255.255.0
			124.346.122.122
			254.255.255.0

falls es sich jeweils um

a *eine gültige IP-Adresse*

b *eine Subnetz-Maske*

c *eine MAC-Adresse*

handelt (Mehrfach-Nennungen sind möglich).

Übung B.32 ⋆ *Gegeben ist die IP-Adresse 145.254.137.125 und die Subnetzmaske 255.255.254.0.*

1. *Zu welcher Netzwerk-Klasse gehört diese Adresse?*
2. *Wie lautet die Hostadresse innerhalb des Subnetzes?*

Übung B.33 ⋆ *Gegeben ist der URL*

```
http://beckenbauer:bayern@www.fifa.de/eintrittskarten.html
```

Welche Funktion haben die Teile `beckenbauer` *und* `bayern`*?*

Übung B.34 ⋆ *Warum muss man in der spannenden Endphase einer Auktion bei ebay immer wieder selbst den Browser aktualisieren? Wieso übernimmt das nicht der Server automatisch bei einem neuen Gebot? Wie könnte man so etwas realisieren?*

Übung B.35 ⋆⋆ *Ist es technisch möglich, dass in einem Netz IPv4 und IPv6 gleichzeitig verwendet werden? Begründen Sie Ihre Entscheidung.*

Übung B.36 ⋆ *Wie kann ein Browser Information wie z. B. den Suchtext für Google oder Ebay mittels HTTP an den Server übermitteln?*

Übung B.37 ⋆ *In welchen Gremien werden Details der Internet-Protokolle diskutiert und festgelegt?*

Übung B.38 ⋆ *TCP wurde eingeführt, um auf Basis von IP zuverlässige, verbindungsorientierte Kommunikation zu ermöglichen. Wozu ist UDP notwendig? Könnte man nicht stattdessen direkt IP-Pakete verwenden?*

Übung B.39 ⋆ *Sie öffnen mit Ihrem Browser die Seite*
`http://www.google.com/search?q=giessen+friedberg&ie=utf-8`

1. *Welche Rolle spielen bei diesem Vorgang HTTP und HTML?*
2. *Erläutern Sie die Struktur der Adresse.*

Anhang C

Ausgewählte Lösungen

Lösung 4.9 Die relevanten Aussagen sind:

a XYZ schlechter DAX

b Kurs mehr als 10% von Einkaufswert entfernt

c Depot unter 3 Millionen Euro

a	b	c	$Verkaufen$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

disjunktive Normalform:

$$\overline{a}\overline{b}c + \overline{a}bc + ab\overline{c} + abc$$

Vereinfachung:

$$\overline{a}c(\overline{b} + b) + ab(\overline{c} + c) = \overline{a}c + ab$$

Lösung 4.10

1.

L_1	L_2	L_3	Alarm
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

2.

L_1	L_2	L_3	Alarm
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Lösung 4.11

a Bauer am linken Ufer

b Gans am linken Ufer

c Korn am linken Ufer

d Hund am linken Ufer

a	b	c	d	*Erlaubt*
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Erlaubte Kombinationen

a	b	c	d		a	b	c	d
1	0	1	1		0	0	0	0
1	1	0	0		0	0	0	1
1	1	0	1		0	0	1	0
1	1	1	0		0	0	1	1
1	1	1	1		0	1	0	0

Jede Fahrt entspricht einem Wechsel von einer Kombination im linken Teil zu einer aus dem rechten Teil der Tabelle. Dabei darf sich außer a maximal eine weitere Stelle ändern. Beginnen wir mit der Kombination 1111. Mögliche Übergänge sind 0111, 0011, 0101 und 0110. Davon ist allerdings nur 0011 erlaubt. Also ist die erste Fahrt

$$1111 \rightarrow 0011$$

Mögliche Übergänge von 0011 sind: 1011, 1111, 1001 und 1010. Da 1111 wieder zum Ausgangspunkt zurückführt und 1001 sowie 1010 nicht erlaubt sind, bleibt nur 1011 (eine Leerfahrt). Nach diesem Prinzip lässt sich leicht verfolgen, welche Fahrten möglich sind. Als Lösung findet man dann die Abfolge

1. **1** 1 1 1
2. **0** 0 1 1
3. **1** 0 1 1
4. **0** 0 1 0
5. **1** 1 1 0
6. **0** 1 0 0
7. **1** 1 0 0
8. **0** 0 0 0

Lösung 5.9

- kleinste Zahl:

```
Vorzeichen:    1
Charkteristik: 11111110
Exponent:      127 (dez)
Mantisse:      11111111111111111111
insgesamt:
1111 1111 0111 1111 1111 1111 1111 1111
ff7fffff (hex)
Float Zahl:    -3.4028235 E38
```

- größte Zahl:

```
Vorzeichen:    0
Charkteristik: 11111110
Exponent:      127 (dez)
Mantisse:      11111111111111111111111
0111 1111 0111 1111 1111 1111 1111 1111
7f7fffff (hex)
Float Zahl:    3.4028235 E38
```

- betragsmäßig kleinste (normalisierte) Zahl:

```
Vorzeichen:    0
Charkteristik: 00000001
Exponent:      -126 (dez)
Mantisse:      00000000000000000000000
0000 0000 1000 0000 0000 0000 0000 0000
800000 (hex)
Float Zahl: 1.17549435 E-38
```

Lösung 9.2 Verkehrsampeln
Alle Lampen:

```
    mov cx,111111111111b  ; alle Lampen ein
    mov ax,0
l1: mov ax,cx
    out 4,ax
    loop l1
```

Jede Lampe einzeln:

```
    mov cx,12             ; Zaehler fuer 12 Lampen
    mov ax,1              ; erste einschalten
l2: out 4,ax
    shl ax,1              ; zur naechsten Lampe
    loop l2
```

Realistischer Ablauf:

```
la:
    mov ax,100001100001b
    out 4,ax
    mov ax,100010100010b
    out 4,ax
    mov ax,100100100100b
    out 4,ax
    mov ax,110100110100b
```

```
    out 4,ax
    mov ax,001100001100b
    out 4,ax
    jmp la
```

Lösung 9.4 CGA-Darstellung
Schleife, um alle ASCII-Zeichen ab `@` auszugeben:

```
    ORG 100h

    mov ax,0b800h       ; zunaechst Segmentadresse fuer Daten setzen
    mov ds,ax
    mov cx,64           ; Schleifenzaehler setzen
    mov al,64           ; erstes Zeichen nach AL
    mov [0],ax          ; Zeichen darstellen
schleife:
    inc [0]             ; naechstes Zeichen
    loop schleife
    ret
```

Lösung 9.7

```
    ORG 100h

la:
    mov ax,100001100001b
    out 4,ax
    mov cx,t1
l1:
    loop l1

    mov ax,100010100010b
    out 4,ax
    mov ax,100100100100b
    out 4,ax
    mov ax,110100110100b
    out 4,ax
    mov ax,001100001100b
    out 4,ax
    jmp la
    ret

t1  dw 5
```

Lösung 9.9 Nicht-optimale Implementierung des Bubble-Sorts.

```
org  100h   ; set location counter to 100h

    mov di,0
    mov cx,9           ; Anzahl der Werte - 1
    mov dx,0
anfang:
    mov al,feld[di]    ; erster Wert nach AL
schleife:
    mov bl,feld[di+1]  ; naechster Wert nach BL

; Vergleichswerte stehen in AL und BL
; der kleinere soll abgespeichert werden
; der groessere fuer den naechsten Vergleich
; in AL bleiben
    cmp al,bl          ; AL < BL ?
    jle  fall2         ; ja, springe zu Fall2
    mov feld[di],bl    ; nein, also BL ist kleiner
                       ; dann BL in Speicher, AL bleibt
    jmp weiter         ; zu naechstem Wert
fall2:
    mov feld[di],al    ; AL in Speicher
    mov al,bl          ; BL nach AL umkopieren
                       ; fuer naechsten Vergleich
weiter:
    add di,1           ; Zaehler erhoehen
    cmp di,cx          ; Noch Werte?
    jl  schleife       ; ja
    mov feld[di],al    ; nein, also letzten
                       ; Wert in AL speichern
    inc dx
    cmp dx,cx          ; Weiterer durchgang?
    jg  ende
    mov di,0
    jmp anfang

ende:
    ret

; die zu sortierenden Zahlen
feld db 5,3,9,8,1,4,7,2,6,0   ; Zahlen 0 bis 9
     db 0ffh,0ffh,0ffh        ; Markierung zur besseren Darstellung
```

Lösung B.25

- Das Programm wandelt die Zeichen in Kleinbuchstaben um.
- `text[di]` kann man als das `di`-te Element des Feldes `text` interpretieren. Intern bedeutet dies, dass zu der Adresse der Variablen `text` der Wert im Register `di` addiert wird.
- Das Programm testet ob ein Zeichen den Zahlenwert 0 hat. Dann wird durch den bedingten Sprung die Bearbeitung beendet. Dies entspricht der Darstellung von Zeichenketten in der Programmiersprache C.

Lösung B.27

Der 8086-Prozessor ist im wesentlichen eine Zweiadressmaschine. Typisch sind Anweisungen in der Art `add op1,op2`. Allerdings gibt es einige Sonderfälle. So wird bei der Multiplikation nur ein Operand angegeben. Dieser Fall entspricht einer Einadressmaschine.

Lösung B.28

Der Cache ist als Assoziativ-Speicher organisiert. Aus dem Inhalt (in diesem Fall die Adresse im Hauptspeicher) errechnet sich direkt die Adresse im Cache. Der Nachteil ist, dass mehrere Hauptspeicheradressen sich die selbe Cache-Adresse teilen. In ungünstigen Fällen – wenn gerade solche Adressen nacheinander verwendet werden – wird dann der vorhandene Wert überschrieben, selbst wenn der Cache nur wenig gefüllt ist. In der Praxis benutzt man daher Anordnungen, bei denen für eine gegebene Adresse zwei oder vier Ablagestellen in Frage kommen.

Anhang D

Abkürzungen

3GPP 3rd Generation Partnership Project

ACF Application Configuration File

ACK Acknowledge. ASCII-Zeichen 6.

ADPCM Adaptive Differentielle Pulscodemodulation

ADSL Asymmetric Digital Subscriber Line

AGP Advanced Graphics Port

AIMD Additive Increase Multiplicative Decrease

ALU Arithmetic Logical Unit

AP Access Point

APNIC Asia Pacific Network Information Centre

ARIN American Registry for Internet Numbers

ARP Address Resolution Protocol

ARPANET Advanced Research Projects Agency Network

AS Autonomes System. In sich abgeschlossenes System mit einheitlichem Routing.

ASCII American Standard Code for Information Interchange

ATM Asynchronous Transfer Mode

BACnet Building Automation Control Network

BGP Border Gateway Protocol. Protokoll für das Routing zwischen autonomen Systemen.

CAN Controller Area Network

CCITT Comité Consultatif International Téléphonique et Télégraphique

CELP Code Excited Linear Predictive Coding

CEPT Conférence Européenne des Administrations des Postes et des Télécommunications

CGA Color Graphics Adapter

CGI Common Gateway Interface

CIDR Classless InterDomain Routing

CISC Complex Instruction Set Computer

CORBA Common Object Request Broker Architecture

CPU Central Processing Unit

CRC Cyclic Redundancy Check

CSMA/CD Carrier Sense Multiple Access with Collision Detection

CTS Clear To Send

DEC Digital Equipment Corporation. Computerfirma, entwicklete u.a. PDP und VAX Systeme und das Betriebssystem VMS, später von Compaq übernommen.

DECT Digital European Cordless Telephony. Standard für digitales schnurloses Telefon.

DES Data Encryption Standard

DHCP Dynamic Host Configuration Protocol

DLE Data Link Escape

DMA Direct Memory Access

DMZ De-Militarized Zone

DNF Disjunktive Normalform

DNS Domain Name System.

DRAM Dynamisches RAM

DSL Digital Subscriber Line

DSP Digitaler Signalprozessor

DVD Digital Versatile Disc

ECP Extended Capability Port

EEPROM Electrical EPROM

EIB Europäischer Installationsbus

EIGRP Enhanced Interior Gateway Routing Protocol

ENIAC Electronic Numerical Integrator And Computer

EPROM Erasable PROM

ETSI European Telecommunications Standards Institute

ETX End of TeXt. ASCII-Zeichen 3.

FDDI Fiber Distributed Data Interface

FIFO First In First Out

FSB Front side bus

FTP File Transport (Transfer) Protocol

GAN Global Area Network

GPRS General Packet Radio Service

GSM Global System for Mobile Communications. Standard für digitales Funkttelefon.

HDLC High Level Data Link Control

HTML HyperText Markup Language

HTTP HyperText Transport Protocol

IAB Internet Architecture Board

IANA Internet Assigned Numbers Authority

IC integrated circuit

ICANN Internet Corporation for Assigned Names and Numbers

ICMP Internet Control Message Protocol

IDE Integrated Disc Electronic

IDEA International Data Encryption Algorithm

IDL Interface Definition Language.

IEEE Institute of Electrical and Electronics Engineers

IESG Internet Engineering Steering Group

IETF Internet Engineering Task Force

IGRP Interior Gateway Routing Protocol

IHL Internet Header Length

IKE Internet Key Exchange

iLBC Internet Low Bit Rate Codec

IM Instant Messenger

IMAP Internet Message Access Protocol

IP Internet Protocol

IPsec IP Security Protokoll

IPTV IP Television

IRC Internet Relay Chat

IRTF Internet Research Task Force

ISDN Integrated Services Digital Network

ISO International Standards Organization

ISOC Internet SOCiety

ISO-IEC/JTC1 Joint Technical Committee of the International Organization for Standardization and International Electrotechnical Commission

ISP Internet Service Provider

ITU International Telecommunications Union

JBOD Just a Bunch of Disks

JPEG Joint Photographic Experts Group

KNF Konjunktive Normalform

L2F Layer 2 Forwarding

L2TP Layer 2 Tunneling Protocol

LACNIC Latin American and Caribbean Internet Addresses Registry

LAN Local Area Network

LCP Link Control Protocol. Protokoll zur Verhandlung der Parameter einer Verbindung mit PPP.

LDAP Lightweight Directory Access Protocol

LLC Logical Link Control

LIFO Last In First Out

LON Local Operating Network

MACA Multiple Access with Collision Avoidance

MAN Metropolitan Area Network

MAP Manufacturing Automation Protocol

MIB Management Information Base

MIME Multi-Purpose Internet Mail Extension

MMU Memory Management Unit

MPEG Motion Picture Experts Group

MSS Maximum Segment Size. Die maximale Größe eines TCP-Segments.

NAK Negative Acknowledge. ASCII-Zeichen 21.

NAT Network Address Translation

NetBEUI NetBIOS Extended User Inferface

NetBIOS Network Basic Input Output System

NIC Network Information Center

NIST National Institute of Standards and Technology

NNI Network node interface. Netzwerk-Interface bei ATM.

NNTP Network News Transfer Protocol

NRZ Non-Return to Zero

NRZI Non-Return to Zero Inverted

NTP Network Time Protocol

OSI Open Systems Interconnection

OTP One-time-programmable PROM

PAN Personal Area Network

PARC Palo Alto Research Center. Das Forschungslabor der Firma XEROX in Palo Alto.

PAT Port and Address Translation

PCI Peripheral Component Interconnect

PCM Pulscodemodulation

PCMCIA Personal Computer Memory Card International Association

PGP Pretty Good Privacy

POP3 Post Office Protocol Version 3

PPP Point–to–Point Protocol

PPTP Point–to–Point Tunneling Protocol

PTI Payload Type Identifier. Kennung für Zellentyp bei ATM.

PVC Permanent Virtual Circuit. Permanente virtuelle Leitung über Paketvermittlung.

QoS Quality of Service

RAID Redundant Array of Independent (Inexpensive) Disks

RAM Random Access Memory

RARP Reverse Address Resolution Protocol

RFC Request For Comment

RIP Routing Information Protocol

RIPE NCC RIPE Network Coordination Centre

RIPE Réseaux IP Européens

RISC Reduced Instruction Set Computer

RLE Run Length Encoding

RMI Remote Methode Invocation

ROHC RObust Header Compression

ROM Read-Only Memory

RPC Remote Procedure Call.

RTCP RTP Control Protocol

RTP Real-Time Transport Protocol

RTS Request To Send

RTT Round-Trip Time

SAN Storage / System Area Network

SAP Session Announcement Protocol

SDP Session Description Protocol

SDRAM Synchronous DRAM

SEAL Simple Efficient Adaptation Layer. Bezeichnung für AAL-5.

SGRAM Synchronous graphics RAM

SIP Session Initiation Protocol

SMTP Simple Mail Transfer Protocol

SNMP Simple Network Management Protocol

SOAP Simple Object Access Protocol oder Service Oriented Access Protocol

SRAM Statisches RAM

SSL Secure Sockets Layer

SSRC Synchronization Source. Quelle eines RTP-Stroms.

STP Spanning Tree Protocol

STX Start of TeXt. ASCII-Zeichen 2.

SVC Switched Virtual Circuit. Vermittelte virtuelle Leitung über Paketvermittlung.

SWS Send Window Size. Die maximale Anzahl von ausstehenden Rahmen beim Sliding Window Algorithmus.

TCP Transmission Control Protocol

TFRC TCP-Friendly Rate Control

TFTP Trivial File Transport Protocol

THT Token Hold Time. Die Zeit, für die ein Knoten beim Token-Ring den Token behalten darf.

TLS Transport Layer Security

TOS Type of Service. Spezifikation im IP-Header.

TRT Token Rotation Time. Die Umlaufzeit für den Token bei Token-Ring-Netzen.

TTL Time To Live. Zähler für die maximale Lebensdauer eines IP-Datagramms.

TTP Time Triggered Protocol. Echtzeitfähiger Bus zur Vernetzung im Kfz-Bereich.

UBE Unsolicited Bulk Email. Unverlangte Massen-E-Mails.

UCE Unsolicited Commercial Email. Unverlangte kommerzielle E-Mails.

UDDI Universal Description, Discovery and Integration

UDP User Datagram Protocol

UPN Umgekehrt Polnische Notation

URI Universal Resource Identifier

URL Universal Resource Locator

URN Universal Resource Name

USB Universal Serial Bus

UUID Universally Unique IDentifier.

VC Virtual Circuit. Virtuelle Leitung über Paketvermittlung.

VCI Virtual Channel Identifier. Kennung für virtuellen ATM-Kanal.

VCI Virtual Circuit Identifier

VLIW Very long instruction word

VLSI Very Large Scale Integration

VPN Virtual Private Network

VRRP Virtual Router Redundancy Protocol

W3C World Wide Web Consortium

WAN Wide Area Network

WEP Wireless Equivalent Privacy. Ursprünglicher Sicherheitsmechanismus für WLAN.

Wi-Fi Wireless Fidelity.

WLAN Wireless LAN

WMAN Wireless Metropolitan Area Networks

WPA Wireless Protected Access. Verbesserter Sicherheitsmechanismus für WLAN.

WPAN Wireless Personal Area Network

WSDL Web Service Description Language

WWW World Wide Web

XDR eXchange Date Representation

XML EXtensible Markup Language

Literaturverzeichnis

[ACK+02] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, November 2002.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, November 1976.

[Dif88] W. Diffie. The first ten years of public-key cryptography. *Proc. IEEE*, 76(5):560–577, May 1988.

[Ech99] Echelon. *Introduction to the LONWORKS® System 1.0*, 1999.

[FKS03] Björn Fröhlecke, Martin Koch, and Hans-Peter Schulz. *Das große Buch BIOS*. Data Becker, 2003.

[Fly72] M. J. Flynn. Some computer organizations and their effectiveness. In *IEEE Trans. on Computers*, volume C-21, pages 948–960, September 1972.

[Gol91] David Goldberg. What every computer scientist should know about floating-point arithmetic. *Computing Surveys*, 1991.

[Goo02] B. Goode. Voive over Internet Protocol (VoIP). *Proc. IEEE*, Vol. 90:1495–1517, 2002.

[Her02] Paul Herrmann. *Rechnerarchitektur*. Vieweg, 2002.

[Inta] Intel. *IA-32 Intel© Architecture Software Developer's Manual Volume 1: Basic Architecture*.

[Intb] Intel. *IA-32 Intel© Architecture Software Developer's Manual Volume 2: Instruction Set Reference*.

[Kö02] Rolf-Dieter Köhler. *Voice over IP*. mitp-Verlag, Bonn, 2002.

[KRWN01] Daniel Karrenberg, Gerard Ross, Paul Wilson, and Leslie Nobile. Development of the regional internet registry system. *The Internet Protocol Journal,* `www.cisco.com/warp/public/759/ipj_4-4/ipj_4-4_regional.html`, 2001.

[LCC+97] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jonathan B. Postel, Lawrence G. Roberts, and Stephen S. Wolff. The past and future history of the internet. *Communications of the ACM*, 40(2):102–108, 1997.

[LM90] X. Lai and J. Massey. A proposal for a new block encryption standard. In I. B. Damgård, editor, *Proc. EUROCRYPT 90*, pages 389–404. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 473.

[Mal01] H. Malz. *Rechnerarchitektur.* Vieweg, 2001.

[Mey98] Matthias Meyer. Konzeption und Umsetzung eines Java-Applets zur Logikminimierung mit KV-Diagrammen. Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1998.

[Mot] Motorola. *M68HC05 Family — Understanding Small Microcontrollers—Rev. 2.0.*

[Mot00] Motorola. *DSP56300 FAMILY MANUAL*, 2000.

[Nat80] National Institute of Standards and Technology (NIST). *FIPS Publication 81: DES Modes of Operation*, December 2, 1980.

[Nol00] P. Noll. Speech and audio coding for multimedia communications. In *Proceedings International Cost 254 Workshop on Intelligent Communication Technologies and Applications*, pages 253–263, Neuchatel, 2000.

[PGK88] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the International Conference on Management of Data (SIGMOD)*, June 1988.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C.* Cambridge University Press, 1992.

[RFR03] Stephan Rein, Frank Fitzek, and Martin Reisslein. Voice quality evaluation for wireless transmission with ROHC. In *Proceedings of the 7th IASTED International Conference on Internet and Multimedia*

Systems and Applications (IMSA), pages 461–466, Honolulu, August 2003.

[Sch88] M. Schwartz. *Telecommunication Networks: Protocols, Modeling and Analysis*. Addison-Wesley, 1988.

[Sin00] S. Singh. *Geheime Botschaften*. Carl Hanser Verlag, München, 2000.

[Ste01] Erich Stein. *Taschenbuch Rechnernetze und Internet*. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2001.

[Tan00] Andrew S. Tanenbaum. *Computernetzwerke*. Prentice Hall, 2000.

[TG01] Andrew S. Tanenbaum and James Goodman. *Computerarchitektur. Strukturen, Konzepte, Grundlagen*. Pearson Studium, 2001.

[ZCB99] Philip Zimmermann, Christopher Creutzig, and Andreas Buhl. *PGP - Pretty Good Privacy - Der Briefumschlag für Ihre elektronische Post*. Art d'Ameublement, 1999.

Index