
IT Grundlagen

MANNHEIM, 12. JANUAR 2021

Prof. Dr. A. Wiedemann

DHBW Mannheim

Coblitzallee 1 – 9

68163 Mannheim

wiedemann@dhbw-mannheim.de

Inhaltsverzeichnis

1	Grundkonzepte der Datenverarbeitung	1
1.1	Daten und Information	2
1.1.1	Information	2
1.1.2	Zeichen	3
1.1.3	Zeichenvorrat	4
1.1.4	Daten	4
1.1.5	Nachrichten	6
1.2	Informatik	7
1.2.1	Technische Informatik	8
1.2.2	Praktische Informatik	8
1.2.3	Theoretische Informatik	8
1.2.4	Angewandte Informatik	9
1.2.5	Ingenieur-Informatik	10
1.2.6	Rechts-Informatik	10
1.2.7	Verwaltungs-Informatik	10
1.2.8	Medizinische Informatik	11
1.2.9	Bio-Informatik	11
1.2.10	Medien-Informatik	12
1.2.11	Wirtschaftsinformatik	13
1.3	Betriebswirtschaftliche Anwendungsprogramme	16
1.3.1	ERP-Systeme	16
1.3.2	eCommerce Lösungen	19
1.3.3	Data-Warehouse	20
1.3.4	Data-Mining	22
1.3.5	CRM-Systeme	24
1.3.6	CAS-Systeme	24
1.3.7	SCM — Supply Chain Management	24
1.3.8	Dokumentenmanagementsysteme	24
1.3.9	Business Intelligence	24
1.3.10	Application Service Providing	25
1.4	Künstliche Intelligenz (KI)	26
1.5	Datenverarbeitung	28
1.6	Datenverarbeitungssystem	30
1.7	Who is who in der Informatik	33

1.8	Standards in der Informationstechnologie	35
1.8.1	Die International Telecommunication Union	36
1.8.2	International Organization for Standardization	39
1.8.3	Das IEEE	41
1.8.4	Weitere Organisationen	44
1.8.5	Internet Organisationen	47
1.8.6	Request for Comments	49
1.8.7	Die Internet Society	51
1.8.8	Das WWW Consortium	52
1.8.9	Das Computer Emergency Response Team	53
1.8.10	Das Software Engineering Institute (SEI)	53
1.9	Computer und Computerklassen	56
1.9.1	Mikro-Computer	56
1.9.2	Mini-Computer bzw. Midrange Rechner	59
1.9.3	Großrechner	62
1.9.4	Supercomputer	65
2	Die von Neumann Architektur	71
2.1	Aufbau des von-Neumann Rechners	71
2.2	Der Prozessor	76
2.2.1	Das Rechenwerk	77
2.3	Das Steuerwerk	80
2.3.1	Aufbau des Steuerwerks	80
2.3.2	Der Fetch-Decode-Execute Zyklus	84
2.4	Ausführungsmodelle	90
2.4.1	Allzweckregister Architekturen	90
2.4.2	Akkumulator-Register Architektur	91
2.4.3	Keller Architektur	92
2.4.4	Speicher-Speicher Architektur	99
2.5	Klassifikation von Rechnerarchitekturen	100
3	RISC und CISC	103
3.1	CISC-Architekturen bei Mikroprozessoren	104
3.2	Das RISC-Konzept	108
3.2.1	Anwendung für RISC-Prozessoren	111
3.2.2	RISC oder CISC aus anwendungstechnischer Sicht	112
3.2.3	Kommerzielle RISC - Plattformen	113
4	Repräsentation der Informationen	117
4.1	Zeichen	117
4.2	Alphabete und Daten	118
4.3	Codierungen	121
4.4	8-Bit Zeichencodes	126
4.4.1	Der erweiterte ASCII-Code	127
4.4.2	Der ECBDIC-Zeichensatz	133
4.5	Unicode	135

4.6	Weitere Codes	137
4.6.1	Barcodes	137
4.6.2	ISBN	139
4.6.3	QR-Codes	142
5	Zahlen und Stellenwertsysteme	145
5.1	Einführung	145
5.1.1	Additionssysteme	146
5.1.2	Hybridsysteme	147
5.1.3	Stellenwertsysteme	147
5.2	Das Binärsystem	151
5.2.1	Umrechnungen Dezimalsystem \iff Binärsystem	151
5.2.2	Arithmetik	157
5.3	Komplementärarithmetik	167
5.3.1	Motivation:	167
5.3.2	Zweierkomplementdarstellung	169
5.3.3	Addition und Subtraktion von Zweierkomplementzahlen	176
5.3.4	Multiplikation zweier Zweierkomplementzahlen	180
5.4	Oktalsystem	189
5.5	Hexadezimalsystem	189
5.6	Übungen	192
6	Codierung von Zahlen	195
6.1	Codierung natürlicher Zahlen	195
6.2	Gleitkommazahlen	199
6.2.1	Das IEEE 754 Gleitkommaformat	199
6.2.2	Genauigkeit	207
6.2.3	Gleitkomma Arithmetik	208
7	Digitale Schaltungen	215
7.1	Gatter	215
7.2	Schaltnetze	220
7.2.1	Halbaddierer	221
7.2.2	Volladdierer	225
7.2.3	Ripple Carry Adder	231
7.2.4	Lichtschalter	232
7.2.5	Das Sieben-Segment-Display	234
7.2.6	Codierer	241
7.2.7	Decoder	242
7.2.8	Multiplexer	243
7.2.9	Komparator	248
7.3	Übungen und Wiederholungsfragen	251
A	Glossar	255
B	Einheiten	259

C	Akronyme	263
D	Literaturhinweise	271
D.1	Historische Entwicklung	271
D.2	Hindergründiges	271
D.3	Informatik Einführung	272
D.4	Theoretische Informatik	272
D.5	Betriebssysteme	272
D.6	Rechnerarchitektur	272
D.7	Programmiersprachen	272
D.8	Netzwerke	272
	Literaturverzeichnis	273

Kapitel 1

Grundkonzepte der Datenverarbeitung

Das Ziel dieses Kapitels ist, eine Antwort auf die Fragen zu finden

Was ist Datenverarbeitung ?

Womit beschäftigt sich die Datenverarbeitung ?

Heutzutage spricht man von Datenverarbeitung, wenn Daten in einen Computer eingegeben werden, vom Computer be- bzw. verarbeitet werden und anschließend vom Computer ausgegeben werden. Grob gesagt versteht man unter Daten alle Arten von Informationen. Dies können wohlvertraute Buchstaben, Zahlen und Worte sein, aber auch Meßwerte oder andere Signale. Neuerdings werden mittels Computer aber auch Bilder, Sprache, Musik und Videos verarbeitet. Aus diesem Grund geht der Trend gegenwärtig dahin, nicht mehr von Datenverarbeitung, sondern allgemeiner von Informationsverarbeitung zu sprechen.

Mit der immer unproblematisch werdenden Möglichkeit, über das Telefonnetz Rechner miteinander zu verbinden und Daten auszutauschen, wächst auch die Tendenz, von der sogenannten *Informationstechnologie* (IT) zu sprechen. Ein Indiz dafür mag sein, daß 1997 das klassische Berufsbild des EDV-Kaufmanns offiziell (von der IHK) durch den IT Kaufmann ersetzt wurde.

Wir wollen uns zunächst einige Begriffe ansehen, die in dem Umfeld der Datenverarbeitung eine grundlegende Rolle spielen.

1.1 Daten und Information

1.1.1 Information

Der Begriff der **Information** hat zahlreiche Bedeutungen in den unterschiedlichsten Bereichen.¹

In der Informationstechnologie ist die folgende Definition zweckmäßig:

Definition 1.1:

Unter dem Begriff **Information** versteht man im Rahmen der Datenverarbeitung ein zweckorientiertes und zielgerichtetes Wissen.

Eine etwas präzisere Definition, die heute als operationaler Standard in der Informationstechnologie angesehen ist, ist die folgende [32]

Definition 1.2:

σ ist eine Instanz von Information, genau dann, wenn

- σ besteht aus einem oder mehreren *Daten*,
- die Daten in σ sind *wohlgeformt*,
- die wohlgeformten Daten in σ haben eine *Bedeutung*.

Offensichtlich erfordert dieser Ansatz die Erklärung was Daten sind.² Die erste Klausel drückt also aus, dass Information aus Daten bestehen. Die zweite Aussage besagt, dass diese Daten *wohlgeformt* sind, das bedeutet, die Daten sind nach bestimmten Regeln zusammengefügt, die dem gewählten System zugrunde liegen, die Regeln nennt man auch die **Syntax**. In der Informatik sagt man, die Daten sind syntaktisch korrekt.

In der dritten Klausel in der Definition [1.2] kommt schließlich der Begriff *Bedeutung* ins Spiel. Der Fachbegriff hierfür ist die **Semantik**.

Beispiel 1.1:

Diese Konzepte lassen sich gut an einem alltäglichen Beispiel erläutern [32].

Montag morgen, man dreht den Zündschlüssel seines Wagens an, es passiert aber nichts. Der Motor rührt sich überhaupt nicht, der Anlasser gibt keinen Laut von sich.. Das Schweigen des Motors läßt gewisse Befürchtungen aufkommen. Es überrascht einen dann auch nicht weiter, dass das rote Kontrollämpchen für niedrigen Batteriestand leuchtet. Nach einigen weiteren Versuchen gibt man die

¹Siehe dazu zum Beispiel den Überblick von FLORIDI [31], der Beitrag in der *Stanford Encyclopedia of Philosophy*, [32], oder das Buch von J. GLEICK [39] zum gleichen Thema. Sehr empfehlenswert ist auch das Buch von VON BAEYER, [106].

²Dies betrachten wir im Abschnitt [1.1.4].

Versuche auf und ruft die Werkstatt an. Man erklärt sein Dilemma, der Mechaniker hilft einem durch den Hinweis weiter, dass die Bedienungsanleitung des Wagens erklärt, wie man mit einem Überbrückungskabel den Motor zum Laufen bringt. Der Nachbar hat glücklicherweise alles was man benötigt, man liest die Bedienungsanleitung, sieht sich die Beschreibungen an, folgt den Anweisungen, löst das Problem und fährt schließlich wohlgelaunt ins Büro.

Daraus läßt sich schließen:

Es besteht immer eine Korrelation zwischen der Information einerseits und zweckorientiertem bzw. zielorientiertem Handeln einer Person andererseits, die auf die jeweilige Information zugreift.

Zielorientiert bedeutet in unseren Beispielen Gewinn maximieren, das nächste Fußballspiel gewinnen oder möglichst schnell durch den Verkehr zu kommen. Wichtig in diesem Zusammenhang ist die Beziehung zwischen der Information einerseits und der Person (oder Personengruppe) andererseits, die die Information auswertet. In den obigen Beispielen wird den Kfz - Händler für seine ökonomische Zwecke weniger die Aufstellung einer Fußballmannschaft interessieren oder die optimale Fahrstrecke von Neustadt/W. nach Mannheim Friedrichsfeld. Ihn interessieren die aktuellen Marktpreise für Gebrauchte PKW, um mit Hilfe dieser Information einen möglichst hohen Gewinn zu erzielen.

1.1.2 Zeichen

Nicht nur im Rahmen der Datenverarbeitung dienen Zeichen dazu, bestimmte Informationen darzustellen, also, **Zeichen** sind die Elemente zur Darstellung von Informationen.

Beispiel 1.2:

Ampel	⇒	Zeichen rot für Halt
Marktpreis	⇒	Zeichen Dezimalzahlen für Preise
Wegbeschreibung	⇒	Zeichen z. B. einfache Linie für Bundesstraße, doppelte Linie für Autobahn

- Verschiedene Informationen werden durch die unterschiedlichsten Zeichen dargestellt.
- Die Zeichen müssen vereinbart sein, damit der Benutzer die Informationen überhaupt versteht. Wenn z. B. unbekannt ist, dass eine schwarze Linie

in Karten Bahnlinien bedeuten, fährt der EDV-Dozent mit dem Auto auf Bahnlinien.

1.1.3 Zeichenvorrat

Um bestimmte Informationen darzustellen, darf die Menge der verwendeten Zeichen natürlich nicht unbeschränkt sein. Die zur Übermittlung der Informationen zulässigen Zeichen bezeichnet man daher als den **Zeichenvorrat**.

Beispiel 1.3:

Ampel	⇒	Farben Rot, Gelb und Grün
Marktpreis	⇒	Dezimalzahlen von ... bis...
Wegbeschreibung	⇒	Zeichen, die in der Legende einer Karte beschrieben sind + Buchstaben für Ortsnamen

Es ist anzumerken, dass der Zeichenvorrat zur Darstellung einer bestimmten Information ebenfalls vereinbart sein muß. Man stelle sich vor, ein Autofahrer — er weiß, wie er sich bei grünem, gelbem und rotem Licht zu verhalten hat — steht vor einer Ampel, die plötzlich auf Blau springt. Da es für dieses Zeichen keine Konvention gibt, kann er mit dieser Information nichts anfangen.

1.1.4 Daten

Nach der Terminologienorm DIN ISO/IEC 2382 sind Daten

Definition 1.3:

Gebilde aus Zeichen oder kontinuierliche Funktionen, die aufgrund bekannter oder unterstellter Abmachungen Informationen darstellen, vorrangig zum Zweck der Verarbeitung und als deren Ergebnis.

Unter **Daten** versteht man eine also eine Menge von Zeichen, die zum Zweck der *Verarbeitung* zusammengefaßt sind.

Beispiel 1.4:

Gegeben ist die Information der möglichen Fahrtstrecken von Mannheim nach Kaiserslautern.

Die Verarbeitung besteht darin, dass sich der Dozent (oder wer auch immer) die bequemste, schnellste oder kürzeste Wegstrecke aussucht.

Mögliche Marktpreise für VW-Golf II TD in unterschiedlichen Regionen. Verarbeitung: Kfz-Händler wählt aus diesen Informationen den für seine Zwecke optimalen Verkaufspreis.

Man klassifiziert in der Informationstechnologie Daten in fünf Kategorien, die u.U. jedoch nicht klar gegeneinander abgegrenzt sind:

- **Primäre Daten**
Darunter versteht man die Daten, die beispielsweise in einer Datenbank abgelegt sind, wie eine Folge von Zahlen. Das sind die Daten, die ein Informationssystem dem Benutzer in Form von Informationen zur Verfügung stellt. Im Bild des Beispiels [1.1] kann man den Indikator für eine schwache Batterie — also die aufleuchtende rote Lampe — als Instanz für die Primärdaten zählen.
- **Sekundäre Daten**
Sekundäre Daten sind das Gegenteil von Primärdaten, diese sind durch ihre Abwesenheit festgelegt. Im Beispiel [1.1] ist die Abwesenheit des Anlassergeräuschs bereits ein Indiz dafür, dass die Batterie nicht genügend Strom liefern kann. Dies ist daher ein typisches sekundäres Datum.
- **Metadaten**
Metadaten sind Angaben über andere Daten — üblicherweise Primärdaten. In Datenbanken beschreiben Metadaten Eigenschaften wie Speicherplatz, Format, Zugriffsbeschränkungen usw. Dementsprechend beschreibt *Meta-information* Informationen über Informationen. Ein einfaches Beispiel für Metainformation ist

 ’’Die Batterie ist leer’ ist in deutscher Sprache geschrieben’’.
- **Operative Daten**
Operative Daten sind Daten, die sich auf die Steuerung und die Performanz eines Informationssystems beziehen. Im Beispiel [1.1] wäre das ein zusätzliches gelbes Lämpchen im Wagen, welches anzeigt, dass das Elektrik-Kontrollsystem nicht richtig funktioniert. Die Tatsache, dass diese Kontrollleuchte aufleuchtet kann ein Hinweis darauf sein, dass die Batterieanzeige nicht richtig funktioniert, dies wiederum untergräbt die ursprüngliche Hypothese, dass die Batterie schwach ist.
- **Abgeleitete Daten**
Schließlich gibt es noch abgeleitete Daten, die aus anderen Quelldaten (meist primäre Daten) erhalten werden können. Mit diesem Typ von Daten hat man es in der Regel bei dem sogenannten *Data Mining* zu tun.³

³Näheres zu Data Mining findet man in Abschnitt [1.3].

1.1.5 Nachrichten

Nachrichten sind — ähnlich wie Informationen — eine Menge von Zeichen. Nur dienen diese nicht zur direkten Verarbeitung, sondern der primäre Zweck von Nachrichten ist der der Weitergabe.

Gegenüber der Information ist die Nachricht also der Träger, in der die Information übertragen wird oder die Form, in der eine Information festgehalten wird. Das Gewinnen einer Information aus einer Nachricht ist mitunter ein sehr komplexer Prozeß, an dessen Ende immer (irgendwo) der Mensch steht.

Beispielsweise ist ein in einer natürlichen Sprache verfaßter Text eine Nachricht als Träger einer Information, die dem Empfänger übermittelt werden soll. Aus ein und derselben Nachricht können die verschiedensten Informationen gewonnen werden, abhängig von unterschiedlichen Vorkenntnissen des Empfängers und/oder den Umständen, unter denen die Nachricht empfangen wird.

Beispiel:

Der EDV-Dozent beauftragt eine Sekretärin, die möglichen Fahrtstrecken von Mannheim nach Kaiserslautern auszusuchen und ihm per FAX zu übermitteln. Was der Dozent dann mit dieser Nachricht anfängt, hängt davon ab, zu welchem Zweck die Information benötigt wird.

- Daten im engeren Sinn, das heißt im Rahmen der EDV, stellen Informationen aufgrund bekannter Konventionen dar, die in maschinell lesbare Form gebracht sind. Man spricht dabei von sogenannter formatierter Information.
- Im Gegensatz dazu gibt es nicht-formatierte Informationen. Darunter versteht man unstrukturierte Informationen wie zum Beispiel Texte. Ein Computer ist in der Lage, die Buchstaben des gerade geschriebenen Textes zu verarbeiten, also man kann die Buchstaben mittels einer Tastatur in den Computer eingeben und mit einem Drucker den Text dann Schwarz auf Weiß ausdrucken lassen. Den Inhalt des Textes, was er also bedeutet, bleibt dem Computer aber völlig im Verborgenen. Er kann damit nichts anfangen !

1.2 Informatik

Die Wissenschaft der maschinellen Informationsverarbeitung heißt **Informatik**. Dabei leitet sich der Begriff Informatik von dem Begriff der Information ab. Die englische Bezeichnung für Informatik ist *Computer Science*, sinngemäß also die Wissenschaft, die sich mit Rechner beschäftigt. Die Informatik ist eng mit der Mathematik und Elektrotechnik verbunden und wird als Studienfach in Deutschland erst seit 1971 (Uni Darmstadt/TH Karlsruhe) angeboten.

Technische	Praktische	Theoretische	Angewandte
Hardware-komponenten	Algorithmen, Datenstrukturen Programmiermethoden	Automatentheorie	Computergrafik
Schaltnetze, Schaltwerke, Prozessoren	Programmiersprachen, Übersetzer	Formale Sprachen	Datenbanken
Mikro-programmierung	Betriebssysteme	Theorie der Berechenbarkeit	Künstliche Intelligenz
Rechner-organisation, -architektur	Softwaretechnik	Komplexitätstheorie	Digitale Signalverarbeitung
Rechnernetze	Mensch-Maschine Kommunikation	Formale Semantik	Simulation und Modellierung
	Verteilte Systeme	Automatische Programmsynthese	Textverarbeitung, Bürokommunikation
	Virtualisierung Cloud Computing	Quantum Computing Kryptographie	Spezifische Anwendungen

Tabelle 1.1: Eine grobe Einteilung der Informatik.

Die Spannweite der Disziplin Informatik ist sehr breit gefächert und beschäftigt sich unter anderem mit:

- Schaltwerken und Rechnerarchitektur
- Automatentheorie und formale Sprachen
- Aufbau von Betriebssystemen
- Programmiersprachen und Übersetzungsprogramme (Compilerbau)
- Algorithmen
- Hardwareentwicklung
-

Dementsprechend gliedert sich die Informatik in eine Reihe von zum Teil überlappenden Teilgebiete auf. Eine grobe Untergliederung der Informatik ist in der Tabelle 1.1 dargestellt. Generell werden vier große Teilgebiete unterschieden: *Technische*, *Praktische*, *Theoretische* und *Angewandte Informatik*.

1.2.1 Technische Informatik

Die Technische Informatik beschäftigt sich in erster Linie mit dem Bau und der Konstruktion von Rechnern, Speicherbausteinen, schnellen Prozessoren oder Parallelprozessoren und alle damit zusammenhängenden Problemen. Die Konstruktion von Peripheriegeräten wie Drucker, Bildschirme oder Festplatten fallen ebenfalls in diesen Bereich der Informatik. Naturgemäß sind die Grenzen zwischen Technischer Informatik und Elektrotechnik/Nachrichtentechnik fließend.

Im Mittelpunkt der technischen Informatik steht weniger der einzelne Schaltkreis sondern eher die Vielfalt, mit der man Schaltkreise und Baugruppen zusammenfügen kann, um daraus ein funktionierendes Rechnersystem zu bauen. Man nennt dieses Gebiet auch **Rechnerarchitektur**, da die Aufgabe des Technischen Informatikers sehr dem eines Architekten ähnelt.

1.2.2 Praktische Informatik

Die Praktische Informatik befaßt sich im weitesten Sinne mit den Programmen, die einen Rechner steuern. Die Brücke zwischen der Hardware und der Anwendungssoftware zu schlagen ist die eigentliche Aufgabe der Praktischen Informatik.

Zu den klassischen Gebieten der Praktischen Informatik zählt der **Compilerbau**. Ein Compiler übersetzt Programme, die in einer Hochsprache (diese heißen auch Programmiersprachen) formuliert sind, in die sehr stark von den technischen Merkmalen der Maschine geprägten Maschinsprache.

Ein weiteres zentrales Thema der Praktischen Informatik ist die **Softwaretechnik**. Die Softwaretechnik beschäftigt sich mit der Entwicklung systematischer Methoden und Verfahren, um große, komplexe Softwareentwicklungen strukturiert durchführen zu können (siehe *e.g.* [7]).

1.2.3 Theoretische Informatik

Nomen est omen: die Theoretische Informatik befaßt sich natürlich mit den theoretischen Grundlagen der Informatik. Zu den angestammten Gebieten dieser Teildisziplin zählen Automatentheorie, formale Sprachen, Komplexitätstheorie und theoretische Strukturen, die bei der Programmentwicklung eingesetzt werden. Hier sind natürlich Berührungspunkte mit der Mathematik und der Logik. Siehe dazu [13] bis [109].

In dem Teilgebiet *Automatentheorie*⁴ wird zum Beispiel danach gefragt, wel-

⁴Die Automatentheorie ist die Theorie der informationsverarbeitenden Systeme, die sich

che einfachsten mathematischen Modelle einem Rechner zu Grunde liegen. In der *Theorie der Berechenbarkeit* wird untersucht, wie man das Berechenbare von dem Nicht-Berechenbaren abgrenzen kann. Dazu versucht man Probleme zu finden, die ein Computer unter keinen Umständen lösen kann. Die *Komplexitätstheorie* fragt danach, welchen rechnerischen Aufwand die Lösung eines gegebenen Problems erfordert. In dem Teilgebiet *Formale Sprachen* wird der strukturelle Aufbau von Programmiersprachen untersucht. Die *Formale Semantik* versucht, die Bedeutung der Konstruktionen von Programmiersprachen mathematisch exakt zu erfassen. Letztendlich versucht man auch, mathematische Methoden direkt auf Programme anzuwenden um damit ihre Korrektheit streng zu beweisen.

1.2.4 Angewandte Informatik

Die Technische, Praktische und Theoretische Informatik bilden die Informatik im engeren Sinn und werden deshalb oft auch als *Kerninformatik* bezeichnet. Diesen Disziplinen steht die Angewandte Informatik gegenüber, in der die Anwendungsmöglichkeiten des Computers erforscht werden. Praktische und Angewandte Informatik sind oft schwer gegeneinander abzugrenzen, weil in beiden Teildisziplinen die Programmierung im Mittelpunkt steht. Gegenstand der Praktischen Informatik ist das Programmieren an sich, während in der Angewandten Informatik der Rechner als Werkzeug eingesetzt wird, um bestimmte Probleme zu lösen.

Die Angewandte Informatik befaßt sich also mit dem Einsatz von Rechnern in den unterschiedlichsten Bereichen. Da in den letzten Jahren die Kosten für Hardware immer weiter gefallen sind, gibt es so gut wie keinen Bereich, der der Computeranwendung verschlossen ist. Einerseits müssen Programme und Konzepte entwickelt werden, die in vielfältigen Bereichen einsetzbar sein sollen, andererseits braucht's natürlich auch spezialisierte Programme zur Bewältigung ganz bestimmter Aufgaben.

Es fängt an mit der Datenverarbeitung in der Wirtschaft und öffentlichen Verwaltung, setzt sich fort mit Computern in der Automatisierungstechnik, beispielsweise in Robotern, Maschinensteuerungen, Verkehrsleitsystemen und endet bei den Mikroprozessoren in Autos, Waschmaschinen oder Armbanduhren. Ein weiteres Feld der Angewandten Informatik sind große Datenbanksysteme zur Speicherung von Informationen großen Umfangs. Unter dem Schlagwort *Künstliche Intelligenz* versucht man Computer so zu programmieren, dass sie

unter Rückgriff auf mathematische und logische Begriffsbildungen und Methoden mit den theoretischen Grundlagen für den Aufbau und die Analyse von Automaten befaßt. Im Vordergrund stehen dabei die digital arbeitenden Automaten. Die analog arbeitenden Automaten (z.B. Regelmechanismen) werden der Kybernetik zugeordnet. In der abstrakten Automaten-theorie wird eine rein mathematische Charakterisierung von Automaten gegeben, die Frage nach einer möglichen Realisierung ist dabei zweitrangig.

sich wie intelligente Wesen verhalten.

Neben diesen vier Hauptbereichen der Informatik haben sich in den letzten Jahren eine Reihe von anwendungsbezogenen Bindestrich-Informatiken entwickelt, die zum großen Teil zur Angewandten Informatik zählen.

1.2.5 Ingenieur–Informatik

Die Aufgaben der Ingenieur–Informatik umfassen unter anderem:

- Statik
- Vermessungstechnik
- Verkehrswesen
- Luft– und Raumfahrttechnik
- computergestütztes Konstruieren (CAD = Computer Aided Design)
- computergestützte Fertigung (CAM = Computer Aided Manufacturing)
- Prozeßautomatisierung
- Robotik

1.2.6 Rechts–Informatik

Spezielle Themen der Rechts–Informatik umfassen:

- Juristische Informations- und Dokumentationssysteme,
- Computerkriminalität
- Urheberschutz für Software.

1.2.7 Verwaltungs–Informatik

Die **Verwaltungs–Informatik** ist eigentlich ein Teilgebiet der Wirtschaftsinformatik und befaßt sich mit dem Einsatz der Informationstechnologie in der öffentlichen Verwaltung. Teilgebiete der Verwaltungs–Informatik sind beispielsweise:

- Einwohnermeldewesen,

- Finanzverwaltung,
- Polizei,
- kommunales Haushaltswesen,
- Statistik.

Unter dem mit der Verwaltungs-Informatik zusammenhängenden Begriff **E-Government** versteht man heute die Vereinfachung und Durchführung von Prozessen zur Information, Kommunikation und Transaktion innerhalb und zwischen staatlichen, kommunalen und sonstigen behördlichen Institutionen sowie zwischen diesen Institutionen und Bürgern oder Unternehmen durch den Einsatz von digitalen Informations- und Kommunikationssystemen.

1.2.8 Medizinische Informatik

Diagnostik, Therapieplanung, Computer Tomographie, EKG-Analyse, ein neuer Begriff in diesem Kontext ist **E-Health**, darunter versteht man Anwendungen elektronischer Medien aller Art im Rahmen der medizinischen Versorgung und anderer Gesundheitsdienstleistungen. Die derzeitige inhaltliche Schwerpunkte der medizinischen Informatik umfassen:

- Archivierung von Krankenunterlagen
- Informationsverarbeitung in der Pflege
- Informationssysteme im Gesundheitswesen
- Mobiles Computing
- Qualitätsmanagement in der Medizin
- Standards zur Kommunikation und Interoperabilität
- Medizinische Bildverarbeitung und Telemedizin

1.2.9 Bio-Informatik

Die Bio-Informatik⁵ ist eine interdisziplinäre Wissenschaft, die Probleme aus den Lebenswissenschaften mit theoretischen computergestützten Methoden löst. Sie hat zu grundlegenden Erkenntnissen der modernen Biologie und Medizin beigetragen. Bekanntheit in den Medien erreichte die Bioinformatik in erster Linie 2001 mit ihrem wesentlichen Beitrag zur Sequenzierung des menschlichen Genoms.

⁵Englisch *bioinformatics*, auch *computational biology*.

In der Bioinformatik — einer noch recht jungen Disziplin — geht es um die computergestützte Speicherung, Organisation und Analyse molekularbiologischer, genetischer, biotechnologischer, biochemischer, pharmazeutischer und medizinischer Daten. Ihre Anfänge reichen bis in das Jahr 1965 zurück, als der *Atlas über Proteinsequenzen und -strukturen* erschien. Dies stellt die erste (bescheidene) Proteindatenbank dar.

Einen spektakulären Erfolg erlebte die Bioinformatik mit der in den Jahren 2000 bis 2003 realisierten vollständigen Sequenzierung beziehungsweise Entschlüsselung des menschlichen Erbgutes, an der sie maßgeblichen Anteil hatte. Aktuell bestehen die Aufgaben der Bioinformatik vor allem darin, mathematische Modelle, Algorithmen und Software zur Simulation biochemischer Prozesse und zur Analyse molekularbiologischer Daten zu entwickeln. So können beispielsweise die Wechselwirkungen zwischen Substanzen in der Pharmazie untersucht werden.

1.2.10 Medien-Informatik

Medieninformatik ist ein Teilgebiet der Informatik, das Anfang der 1990er-Jahre vor dem Hintergrund der Digitalisierung von Text, Bild, Audio und Video entstanden ist. Durch neue Technologien und die Konvergenz der Medien sind neue Technologien, Märkte, Anwendungen, Tätigkeitsfelder und Berufsbilder entstanden. In den 1990er-Jahren beherrschte der Begriff **Multimedia** die Diskussion.

Die Medieninformatik ist stark interdisziplinär angelegt und hat insbesondere Berührungspunkte zur angewandten Informatik mit dem Schwerpunkt Multimedia, zur Medientheorie, Medienökonomie, Mediengestaltung, Psychologie oder Mediendidaktik.

Je nach Hochschule und Abschluss werden den einzelnen Teilgebieten der Medieninformatik unterschiedlich starke Bedeutung zugemessen. An manchen Hochschulen liegen die Schwerpunkte auf den Informatik-Grundlagen und deren praktischer Anwendung im Bereich der Medientechnik. An anderen Hochschulen werden hingegen eher Kompetenzen vermittelt, die dazu befähigen sollen, sich auf gestalterische Weise mit jenen Mediensystemen auseinanderzusetzen, welche durch die Digitalisierung von Printmedien, audiovisuellen Medien und digitalen Kommunikationstechnologien hervorgegangen sind. Diese digitalisierten Medien werden häufig unter dem Schlagwort *Neue Medien* subsumiert.

1.2.11 Wirtschaftsinformatik

Den Einsatz der Informatik in Wirtschaftsbetrieben oder Verwaltung und in Behörden wird als **Wirtschaftsinformatik** bezeichnet. Die Wirtschaftsinformatik hat sich unter der Angewandten Informatik als eigenständige *Realwissenschaft* etabliert. Realwissenschaft bedeutet, der Untersuchungsgegenstand der Wirtschaftsinformatik sind reale Dinge, nämlich die Abläufe und Prozesse in Wirtschaftsunternehmen.

Inhaltlich befaßt sich die Wirtschaftsinformatik in erster Linie mit dem Überlappungsbereich von Betriebswirtschaft und Informatik. Viele andere Fachgebiete spielen in diesem Umfeld jedoch ebenfalls durchaus gewichtige Rollen, zum Beispiel:

- ✦ statistische Methoden aus der Mathematik wie Trends, Prognosen
- ✦ Optimierungsverfahren aus der mathematischen Disziplin Operations Research, beispielsweise Wegoptimierung, Lineare Optimierung.
- ✦ Mathematische Methoden zur Verschlüsselung
- ✦ spezielle Rechtsfragen aus dem Bereich Jura – z.B. Urheberrecht im Internet, Datenschutz, digitale Signatur, Arbeitsrecht
- ✦ Themen aus dem Gebiet der Volkswirtschaft
- ✦ Themen aus der Psychologie, *e.g.* Bildschirmgestaltung, Ergonomie, usw.

Diese Überlappung mit anderen Fachgebieten untermauert den interdisziplinären Charakter der Wirtschaftsinformatik.

Die Wirtschaftsinformatik beschäftigt sich mit betriebswirtschaftlichen **Informationssystemen**, *i.e.* mit Systemen zur Beschaffung, Verarbeitung, Speicherung, Übertragung und Bereitstellung von Informationen. Solche Informationssysteme kommen in unterschiedlichen Formen und Ausprägungen vor:

- ✎ Systeme basierend auf isoliert betriebenen Computern, *i.e.* Einzelplatzrechnern, mit Peripheriegeräten für die Dateneingabe (Tastatur, Maus, Scanner) und Datenausgabe (Bildschirm, Drucker). Außerdem verfügen solche Systeme adäquate Software zur Bearbeitung von Informationen. Beispiele sind
 - Lohnabrechnung
 - Fakturierung
 - Steuerung von Produktionsanlagen
- ✎ Systeme auf der Basis hochleistungsfähiger Midrange- und Mainframe Rechner.

- ☞ Systeme unter Einsatz mobiler Rechner wie Notebooks, Handhelds und Smartphones. Solche Systeme sind für den mobilen Einsatz konzipiert, beispielsweise für Versicherungsvertreter, Handelsvertreter.
- ☞ Systeme, die vorwiegend Datenübertragungsinfrastrukturen nutzen und in erster Linie der Kommunikation zwischen den angeschlossenen Endstationen dienen. Beispiele sind Reservierungssysteme oder Bankautomaten.
- ☞ Die heute verbreitetste Form besteht aus Systemen auf der Basis vernetzter Computer, die bei der Abwicklung der Aufgaben zusammenarbeiten. Diese vernetzten Systeme sind in Form sogenannter **Client/Server Architekturen** aufgebaut, die bei der Aufgabenabwicklung — teilweise zentral über die Server — zusammenarbeiten. Typisches Anwendungsbeispiel sind integrierte Standardsoftwaresysteme, die modular die Funktionsbereiche eines Unternehmens abdecken.

Die Tatsache, dass der zentrale Gegenstand der Wirtschaftsinformatik der Einsatz von Informationssystemen ist, impliziert eine Reihe verschiedener Einzelbereiche, die in die Thematik der Wirtschaftsinformatik einfließen:

- ☞ Geräte zur Speicherung und Verarbeitung von Informationen, also insbesondere
 - ☛ Computer in unterschiedlichen Ausprägungen und unterschiedlichen Leistungsklassen,
 - ☛ Peripheriegeräte zur Ein- und Ausgabe von Daten,
 - ☛ Speichermedien.
- ☞ Infrastrukturen zur Übertragung von Informationen.
- ☞ Software in allen Erscheinungsformen.
- ☞ Methoden und Werkzeuge, die für die Entwicklung von Software eingesetzt werden.
- ☞ Datenaufbereitung
- ☞ Organisation und Projektmanagement
- ☞ Präsentationstechniken

Zusammenfassend ist also festzuhalten:

Gegenstand der Wirtschaftsinformatik

Wirtschaftsinformatik befaßt sich mit

- ✎ Konzeption und Planung
- ✎ Implementierung
- ✎ Einführung
- ✎ Wartung
- ✎ Betrieb und Nutzung

von Informationssystemen (IS) in Wirtschaft und Verwaltung unter Berücksichtigung ökonomischer Gesichtspunkte.

Ausgehend von Informationssysteme als Gegenstand der Wirtschaftsinformatik, ist das primäre Ziel der Wirtschaftsinformatik die für den jeweiligen Anwendungszweck optimale Gestaltung von Informationssystemen nach vorgegebenen Kriterien wie:

- ✎ Verarbeitungsgeschwindigkeit
- ✎ Verfügbarkeit
- ✎ Sicherheit
- ✎ Funktionsabdeckung usw.

Ein gleichgelagertes Ziel ergibt sich hinsichtlich der Bereitstellung der Informationsinfrastruktur, *i.e.* die optimale Gestaltung und der optimale Betrieb der Infrastruktur (*e.g.* Rechnernetzwerk) in einem Unternehmen oder einer Behörde. Neben diesen Hauptzielen lassen sich weitere Ziele der Wirtschaftsinformatik formulieren, beispielsweise:

- ✎ Entwicklung von Methoden und Werkzeugen zur Unterstützung des Entwurfs und der Gestaltung der Informationssysteme.
- ✎ Entwicklung von Ansätzen zur Strukturierung und Modellierung von Daten.
- ✎ Entwicklung von Organisationsmodellen.

Die **Aufgabenfelder der Wirtschaftsinformatik** folgen direkt aus den formulierten Zielen. Kernaufgabe ist die Planung, der Entwurf, die Konzeption, die Realisierung und der Betrieb

- von *optimalen* Informationssystemen,

- einer optimaler Informationsinfrastruktur

sowie alle damit zusammenhängenden Fragen.

Daraus folgen die typischen Aufgabengebiete der Wirtschaftsinformatik:

- ① Erstellung und Wartung integrierter betriebswirtschaftlicher Administrations-, Dispositions-, Planungs- und Informationssysteme in allen betriebswirtschaftlichen Funktionsbereichen (Beschaffung, Produktion, Vertrieb, Verwaltung) und in allen Branchen (Banken, Fertigung, Handel).
- ② Entwicklung von Computerprogrammen für diese speziellen Anwendungsbereiche. Genutzt werden sollen die Prinzipien, Methoden, Verfahren und Werkzeuge des Software-Engineerings und die Vorgaben des Projekt - Managements.
- ③ Erstellen von Kriterien zur Auswahl geeigneter Soft- und Hardware für DV-Anwendungen im betriebswirtschaftlichen Bereich.
- ④ Die Auswahl und Einführung computergestützter Bürokommunikationssysteme.
- ⑤ Analyse und Konzeption von Nutzen und Rentabilität des DV - Einsatzes.
- ⑥ Informationsmanagement - Bereitstellung aller relevanten Informationen an jeder Stelle im Unternehmen.

1.3 Betriebswirtschaftliche Anwendungsprogramme

1.3.1 ERP-Systeme

Die Klasse von Anwendungsprogrammen zur Abwicklung der Geschäftsprozesse eines Unternehmens nennt man heute:

ERP-Systeme

oder

Enterprise Resource Planning Programme

Unter ERP Anwendungen versteht man also die Klasse der betriebswirtschaftlichen Standardsoftware zur Unterstützung sämtlicher relevanter Geschäftsprozesse eines Unternehmens.

Beispiele von ERP Systemen sind:⁶

⁶Eine umfassende Liste von ERP Lösungen findet man unter:

<http://www.softguide.de/software/erp.htm>.

Siehe auch die Gesellschaft zur Prüfung von Software unter

<http://www.gps-ulm.de>.

- R/3 von SAP bzw. SAP Business Suite
- Navision bzw. Dynamics von MicroSoft
- JD Edwards von ORACLE
- Peoplesoft von ORACLE
- ProAlpha
- PC-Kaufmann oder KHK von SAGE für Kleinbetriebe

Unter **Standardsoftware** im engeren Sinn, d.h. im Rahmen der Wirtschaftsinformatik versteht man Programme — bzw. Programmpakete — mit folgenden Eigenschaften:

- das Programmpaket deckt *ein* eindeutig definiertes betriebliches Anwendungsgebiet ab; z.B.
 - Fakturierung
 - Anlagenbuchhaltung
 - Kostenrechnung
 - Auftragsabwicklung
 - Bestellabwicklung
 - Vertriebsaktivitäten
 - usw.
- das Programmpaket bzw. die einzelnen Programme haben einen Festpreis für die Grundversion. Die Anpassung an die individuellen betrieblichen Spezifikationen erfolgt nach Aufwand.

Es gibt im wesentlichen zwei Arten, wie Standardsoftware angeboten wird:

- ⇒ in Form von Programmen für einzelne betriebliche Anwendungen
- ⇒ in Form geschlossener Programmpakete für die Gesamtheit aller betriebswirtschaftlichen Abläufe. Diese Art von Standardsoftware firmiert heute unter dem Begriff ERP System, es gibt Synonyme wie
 - *integrierte Standardprogramme*
 - *integrierte Standardsoftware*
 - *integriertes betriebswirtschaftliches Informationssystem*
 - *Business Systeme*
 - *integrierte kaufmännische Software*
 - ...

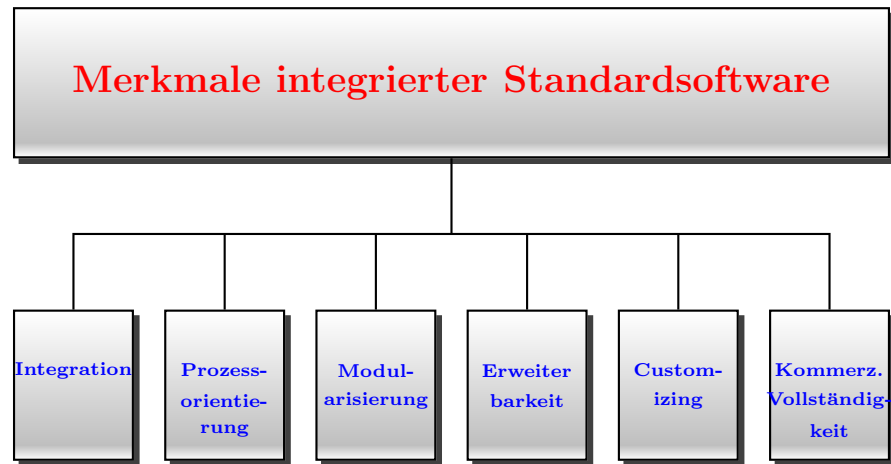


Abbildung 1.1: Verschiedene Charakteristika von betriebswirtschaftlicher Standardsoftware.

Die wesentlichen Merkmale der integrierten Standardprogramme sind in der Abbildung [1.1] schematisch abgebildet.

❶ **Kommerzielle Vollständigkeit**

Zumindest die wichtigsten oder sogar alle betriebswirtschaftlichen bzw. kaufmännischen Funktionen werden von der Software abgedeckt. Dadurch kann der gesamte kommerzielle Bereich eines Unternehmens mit einem einzigen Softwaresystem abgedeckt werden.

❷ **Integration**

Unter der Eigenschaft 'Integration' versteht man einmal, dass sämtliche Teilsysteme der Software miteinander vernetzt sind. Dadurch wirkt sich die Bearbeitung in einem Teilsystem auf das Gesamtsystem aus.

Die zweite Bedeutung von 'Integration' bezieht sich darauf, dass sich ein ERP System möglichst nahtlos — d.h. ohne Medienbruch — in eine bestehende Bürokommunikationslandschaft einpasst. Dazu sind eine Reihe von Schnittstellen sehr vorteilhaft, die solch ein System bieten sollt. Beispielsweise ermöglicht ein ODBC-Treiber den Datenaustausch mit MS-Office.⁷

❸ **Prozeßorientierung**

⁷ODBC steht für *Open Database Connectivity*.

Eine wichtige Eigenschaft einer integrierten Standardsoftware ist ihre Ausrichtung auf *organisatorische Prozeßketten*. Die Prozeßorientierung stellt den zentralen Aspekt der Bearbeitung dar. Ihr untergeordnet sind zum Beispiel Funktionen, Abteilungen, oder Arbeitsplätze.

4 Modularität

Integrierte Standardsoftwaresysteme sind modular aufgebaute Programme. Dadurch ist es möglich, die Software Baustein für Baustein in einem Unternehmen einzuführen. Üblicherweise reicht die Zeitspanne für die Einführung integrierter Software von einigen Monaten (z.B. mittelständige Unternehmen) bis zu mehreren Jahren (z.B. Großunternehmen). Den Prozeß der Softwareeinführung nennt man auch *Implementierung*.

5 Customizingfähigkeit

Diese Art von Software kann und muß in größerem oder geringerem Umfang an die tatsächlichen betrieblichen Belange anpaßbar sein. Dies nennt man **Customizing**. Solche Anpassungen können zum Beispiel sein:

- Betriebsspezifika
- Branchengegebenheiten
- Länderspezifikationen

6 Erweiterbarkeit

Die Erweiterbarkeit und die Änderbarkeit integrierter Softwaresysteme durch das Unternehmen, das diese Art von Software einsetzt, ist ein weiteres kritisches Merkmal. So verfügen die meisten integrierte Softwarepakete über eine eigene Entwicklungsumgebung mit eigener Programmiersprache.

Standardsoftware ist definitionsgemäß weitestgehend **branchenneutral**.

1.3.2 eCommerce Lösungen

Neuerdings auch **eBusiness** genannt, bezeichnet den elektronischen Kauf und Verkauf von Waren und Dienstleistungen mit Hilfe von computergestützten Geschäftstransaktionen, die über das Internet, Netzwerke und andere elektronische Techniken abgewickelt werden.⁸ Bei den vielfältigen Ausprägungen des eCommerces unterscheidet man:

- **B2B, Business-to-Business Lösungen** umfassen den Vertrieb von Produkten und Dienstleistungen zwischen Unternehmen. Hauptanwendungsbereich ist die elektronische Beschaffung von B- und C-Gütern. Zur elektronischen Kopplung von Unternehmen werden unterschiedliche Techniken und Standards eingesetzt, beispielsweise virtuelle Marktplätze oder EDIFACT.⁹

⁸Siehe [70], Seite 68 und Kap. 4.2, oder [75]

⁹EDIFACT ist ein Akronym für *electronic data interchange for administration, commerce and transport* und bezeichnet einen branchenübergreifenden internationalen Standard für das Format elektronischer Daten im Geschäftsverkehr. Näheres siehe *e.g.* [75], Kap. 17.3.

- **B2A, Business-to-Administration** bezeichnet den Verkauf von Produkten und Dienstleistungen von Unternehmen an öffentliche Behörden und den Staat. Diese Form des eCommerce klassifiziert den Staat ebenfalls als einen Kunden gegenüber den Unternehmen und gewinnt im Kontext des eGovernment immer mehr an Bedeutung.
- **B2C, Business-to-Consumer Lösungen** bezeichnen den Verkauf von Produkten und Dienstleistungen von Unternehmen an einzelne Verbraucher. Die Website von **amazon** ist ein typisches Beispiel einer B2C-Lösung.
- **C2C, Consumer-to-Consumer-Lösungen** bezeichnen den Handel von Produkten zwischen Verbrauchern. Dazu zählen beispielsweise Auktionen (eBay).
- **C2B, Consumer-to-Business**. Dies bezeichnet Kommunikationsbeziehungen zwischen Privatpersonen und (mehreren) Firmen. Typisch für C2B Szenarien sind Internetseiten, auf denen Privatpersonen ihre Wunschprodukte angeben und die Händler sich mit ihren Angeboten unterbieten. Ein Beispiel einer C2B Plattform stellt die Website **myhammer.de** dar, über die Privatpersonen von Handwerkern Angebote einholen können.

1.3.3 Data-Warehouse

Ein **Data-Warehouse** ist eine zentrale Datensammlung, deren Inhalt sich aus Daten unterschiedlicher Quellen zusammensetzt. Solche Quellen sind operative Systeme (ERP-Systeme), weitere transaktionsbasierte Datenbanken, unstrukturierte Daten wie Word-Dokumente, pdf-Dateien, Excel-Sheets usw. Die Daten werden von den Datenquellen bereitgestellt und einen Ladeprozess — diesen Schritt nennt man ETL für *Extract, Transform, Load* — in das Data-Warehouse geladen. Dieser Prozess kann turnusgemäß durchgeführt werden, so dass im Data-Warehouse nicht nur Daten nach inhaltlichen Aspekten, sondern auch nach dem Aspekt Zeit — also langfristig — vorgehalten werden, was auch Analysen über die Zeit ermöglicht. In dem Data-Warehouse werden diese zusammengeführten Daten vor allem für die Datenanalyse und zur betriebswirtschaftlichen Entscheidungshilfe in Unternehmen sowie zum Data-Mining langfristig gespeichert. (Siehe Abb. [1.2])

Die Erstellung eines Data-Warehouses verfolgt zwei grundsätzliche Ziele:

1. Die Integration von Daten aus verteilten und unterschiedlich strukturierten Datenbeständen, um im Data-Warehouse eine globale Sicht auf die Quelldaten und damit übergreifende Auswertungen zu ermöglichen.
2. Die Separation der Daten, die für das operative Geschäft genutzt werden, von solchen Daten, die im Data-Warehouse etwa für Aufgaben des Berichtswesens, der Entscheidungsunterstützung, der Geschäftsanalyse sowie des Controllings und der Unternehmensführung verwendet werden.

Datenquellen

Data-Warehouse

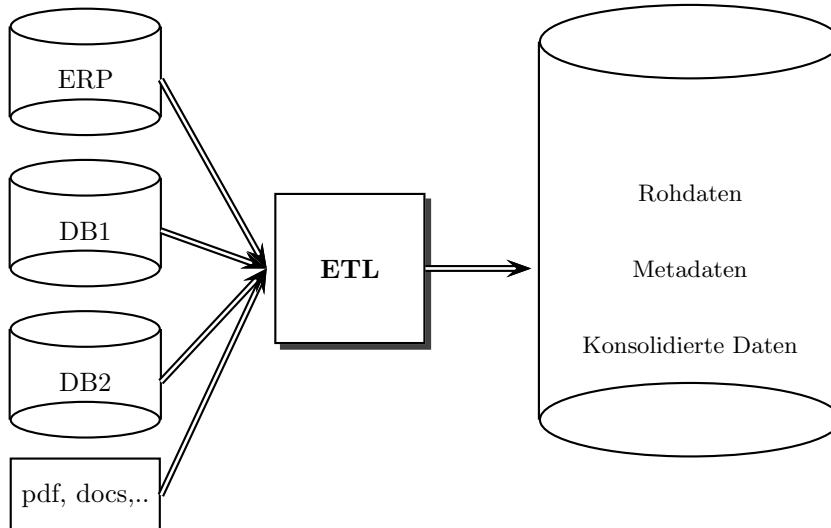


Abbildung 1.2: Der strukturelle Aufbau eines Data-Warehouse-Systems.

Zur Zeit existiert keine einheitliche Definition für den Begriff Data-Warehouse. Weitgehend gilt jedoch:

- Ein Data-Warehouse ermöglicht eine globale Sicht auf heterogene und verteilte Datenbestände, indem die für die globale Sicht relevanten Daten aus den Datenquellen zu einem gemeinsamen konsistenten Datenbestand zusammengeführt werden.
- Somit entsteht der Inhalt eines Data-Warehouse durch Kopieren und Aufbereiten von Daten aus unterschiedlichen Quellen.
- Meist ist ein Data-Warehouse die Basis für die Aggregation von betrieblichen Kennzahlen und Analysen innerhalb mehrdimensionaler Matrizen, dem sogenannten *Online Analytical Processing* (OLAP).
- Ein Data-Warehouse ist häufig Ausgangsbasis für *Data-Mining*.
- In der Regel arbeiten die Anwendungen mit anwendungsspezifisch erstellten Auszügen aus dem Data-Warehouse, den sogenannten **Data-Marts**.

1.3.4 Data-Mining

Unter **Data-Mining**¹⁰ versteht man die systematische Anwendung von mathematischen und statistischen Methoden auf einen Datenbestand mit dem Ziel, neue Muster zu erkennen. Hierbei geht es auch um die Verarbeitung sehr großer Datenbestände, die in der Regel nicht mehr manuell verarbeitet werden können. Daher werden effiziente Methoden benötigt, deren Zeitkomplexität sie für solche Datenmengen geeignet macht. In der Praxis, vor allem im deutschen Sprachgebrauch, etablierte sich der Begriff Data-Mining für den gesamten Prozess der so genannten **Knowledge Discovery in Databases** (Wissensentdeckung in Datenbanken; KDD), der auch Schritte wie die Vorverarbeitung beinhaltet, während Data-Mining eigentlich nur den Analyseschritt des Prozesses bezeichnet.

Eng verwandt mit dem Data-Mining ist das Thema *maschinelles Lernen*, wobei beim Data-Mining der Fokus auf dem Finden neuer Muster liegt, während im maschinellen Lernen primär bekannte Muster vom Computer automatisch in neuen Daten wiedererkannt werden sollen. Eine einfache Trennung ist hier jedoch nicht immer möglich: werden beispielsweise Assoziationsregeln aus den Daten extrahiert, so ist das ein Prozess der den typischen Data-Mining Aufgaben entspricht, die extrahierten Regeln erfüllen aber auch die Ziele des maschinellen Lernens. Umgekehrt ist der Teilbereich des unüberwachten Lernens aus dem maschinellen Lernen sehr eng mit Data-Mining verwandt. Verfahren aus dem maschinellen Lernen finden oft im Data-Mining Anwendung und umgekehrt.

Das **Information Retrieval** (IR) ist ein weiteres Fachgebiet, das von Erkenntnissen des Data-Mining profitiert. Hierbei geht es um die computergestützte Suche nach komplexen Inhalten, aber auch um die Präsentation für den Nutzer. Data-Mining Verfahren wie die **Clusteranalyse** finden hier Anwendung um die Suchergebnisse und ihre Präsentation für den Nutzer zu verbessern, beispielsweise indem man ähnliche Suchergebnisse gruppiert. Mit dem Data-Mining verwandt sind die Begriffe **Textmining**¹¹ und **Webmining**.¹²

Typische Aufgabenstellungen des Data-Mining sind:

- **Ausreißererkennung:**

Darunter versteht man die Identifikation von ungewöhnlichen Datensätzen: Ausreißern, Fehlern, Änderungen. Bei der Ausreißererkennung werden Datenobjekte gesucht, die inkonsistent zu dem Rest der Daten sind, beispielsweise indem sie ungewöhnliche Attributswerte haben oder von einem generellen Trend abweichen.

¹⁰Einen Überblick findet man in dem Artikel von FAYYAD *et al.* [29].

¹¹Im Textmining geht es um die Analyse von großen textuellen Datenbeständen. Dies kann beispielsweise der Plagiats-Erkennung dienen oder um den Textbestand zu klassifizieren.

¹²Beim Webmining geht es um die Analyse von verteilten Daten, wie es Internetseiten darstellen. Für die Erkennung von Clustern und Ausreißern werden hier aber nicht nur die Seiten selbst, sondern insbesondere auch die Beziehungen (Hyperlinks) der Seiten zueinander betrachtet.

- **Clusteranalyse:**

Die Clusteranalyse ist eine Gruppierung von Objekten aufgrund von Ähnlichkeiten. Bei der Clusteranalyse geht es darum, Gruppen von Objekten zu identifizieren, die sich auf eine gewisse Art ähnlicher sind als andere Gruppen. Oft handelt es sich dabei um Häufungen im Datenraum, woraus sich der Begriff Cluster ableitet.

- **Klassifikation:**

Bei der Klassifikation geht es — ähnlich der Clusteranalyse — darum, Objekte Gruppen (hier als Klassen bezeichnet) zuzuordnen. Im Gegensatz zur Clusteranalyse sind hier aber in der Regel die Klassen vordefiniert, beispielsweise: Fahrräder, PKW, LKW, und es werden Verfahren aus dem maschinellen Lernen eingesetzt um bisher nicht zugeordnete Objekte diesen Klassen zuzuordnen.

- **Assoziationsanalyse:** Die Assoziationsanalyse identifiziert Zusammenhänge und Abhängigkeiten in den Daten in Form von Regeln wie *Aus A und B folgt normalerweise C*. In der Assoziationsanalyse werden häufige Zusammenhänge in den Datensätzen gesucht und meist als Schlussregeln formuliert. Ein beliebtes Beispiel: Bei der Warenkorbanalyse wurde festgestellt, dass die Produktkategorien 'Windeln' und 'Bier' überdurchschnittlich oft zusammen gekauft werden, meist dargestellt in Form einer Schlussregel 'Kunde kauft Windeln \implies Kunde kauft Bier'. Die Interpretation dieses Ergebnisses war, dass Männer, wenn sie von ihren Ehefrauen Windeln kaufen geschickt werden, sich gerne noch ein Bier mitnehmen. Durch Platzierung des Bierregals auf dem Weg von den Windeln zur Kasse konnte angeblich der Bierverkauf weiter gesteigert werden.

- **Regressionsanalyse:**

Hierbei geht es um die Identifikation von Beziehungen zwischen (mehreren) abhängigen und unabhängigen Variablen. Bei der Regressionsanalyse wird der statistische Zusammenhang zwischen unterschiedlichen Attributen modelliert. Dies erlaubt unter anderem die Prognose von fehlenden Attributswerten, aber auch die Analyse der Abweichung analog zur Ausreißererkennung. Verwendet man Erkenntnisse aus der Clusteranalyse und berechnet separate Modelle für jeden Cluster so können typischerweise bessere Prognosen erstellt werden. Wird ein starker Zusammenhang festgestellt, so kann dieses Wissen auch gut für die Zusammenfassung genutzt werden.

- **Zusammenfassung:**

Ziel ist die Reduktion der selektierten Datenmenge in eine kompaktere Beschreibung ohne wesentlichen Informationsverlust. Da Data-Mining oft auf sehr große und komplexe Datenmengen angewendet wird, ist eine wichtige Aufgabe auch die Reduktion dieser Daten auf eine für den Nutzer handhabbare Menge wichtig. Insbesondere die Ausreißererkennung identifiziert hierzu einzelne Objekte die wichtig sein können; die Clusteranalyse

identifiziert Gruppen von Objekten bei denen es oft reicht, sie nur anhand einer Stichprobe zu untersuchen, was die Anzahl der zu untersuchenden Datenobjekte deutlich reduziert. Die Regressionsanalyse erlaubt es, redundante Informationen zu entfernen und reduziert so die Komplexität der Daten. Klassifikation, Assoziationsanalyse und Regressionsanalyse (zum Teil auch die Clusteranalyse) liefern zudem abstraktere Modelle der Daten.

1.3.5 CRM–Systeme

CRM ist ein Kürzel, das für *Customer Relationship Management* steht und bezeichnet im engeren Sinn Softwarelösungen, zur Unterstützung und Optimierung aller kundenbezogenen Prozesse eines Unternehmens. Beispiel: Siebel.

1.3.6 CAS–Systeme

CAS steht für *Computer Aided Selling* und bezeichnet Systeme, mit deren Hilfe die Verkaufsmitarbeiter eines Unternehmens bei der Abwicklung der Verkaufsprozesse unterstützt werden. Solche Systeme erlauben den Zugriff auf Kunden- und Kontaktdaten und ermöglichen dadurch eine gezielte Steuerung des Verkaufs.

1.3.7 SCM — Supply Chain Management

Unter einer Supply Chain versteht man alle Geschäftsprozesse einer Wertschöpfungs- bzw. Lieferkette, die zur Erstellung und Lieferung von Produkten sowie Dienstleistungen erforderlich sind. Sie beginnt mit dem Auftrag, umfaßt den Bedarf an Rohstoffen und endet mit der Lieferung an den Endverbraucher. SCM - Systemen sollen Unternehmen ein effektives EDV-gestütztes System an die Hand geben, um die Supply Chain zu managen [12].

1.3.8 Dokumentenmanagementsysteme

Ein **Dokumentenmanagementsystem**, kurz DMS, unterstützt das Einfügen, Aktualisieren und die Archivierung nicht strukturierter Dokumente in einem Repository. Wesentliche Eigenschaften von DMS sind Versionskontrolle der Dokumente, Verwaltung der Zugriffsrechte auf die Dokumente, elektronische Unterschriften, Unterstützung von Volltextsuche in den Dokumenten (und Archiv), sowie die Möglichkeit der Integration in Workflow–Systeme.

1.3.9 Business Intelligence

Der Begriff **Business Intelligence**, kurz BI, umfasst sämtliche analytische Konzepte, Prozesse und Tools, um Unternehmens- und Wettbewerbsdaten in

verfügbares Wissen¹³ zu transformieren. Dabei werden sowohl unternehmensinterne als auch externe Datenquellen herangezogen.

1.3.10 Application Service Providing

ASP steht für **Application Service Providing** (siehe *e.g.* [24]) Der Begriff *Application Service Providing*, kurz ASP, kursierte bereits in den 1990er Jahren, und steht für Mietsoftware. Neuerdings erfährt diese Art der zur Verfügung Stellung von Software ein Revival unter dem Schlagwort *Software as a Service*, kurz SaaS.

IT-Investitionen sind häufig sehr kostenintensiv, Programme und Hardware veralten sehr schnell, die Unterhaltung einer unternehmenseigenen IT-Infrastruktur erfordert immer geeignet qualifiziertes Personal. Dies führte zu der Idee, die IT komplett auszulagern. Wesentlicher Aspekt dabei ist, dass ein sogenannter Application Service Provider den Kunden Dienste in Form von Bereitstellung von Anwendungssoftware (*e.g.* ERP-Systeme) anbietet. Die Software wird dabei auf den zentralen Rechnern des Anbieters zur Verfügung gestellt und dort gewartet, so dass stets die aktuellste Version verfügbar ist. Die vertraglichen Vereinbarungen zwischen Anbieter und Nutzer sind in den sogenannten *Service Level Agreements*, kurz SLA, geregelt. Zugriff haben die Kunden des Service Providers in der Regel über das Internet, so dass die Software betriebssystemunabhängig läuft, auf Clientseite genügt oft ein Browser.

Die Akzeptanz dieses Mietmodells war anfangs eher zurückhaltend aufgrund Befürchtungen hinsichtlich der Datensicherheit. Ein weiteres Problem stellte die mangelnde Mandantenfähigkeit¹⁴ damaliger ERP Systeme dar. Diese Probleme gelten heute als gelöst.

¹³Darauf bezieht sich der Begriff *intelligence*.

¹⁴Informationstechnik bzw. Software bezeichnet man als mandantenfähig, wenn über den selben Server mehrere Kunden (*i.e.* Mandanten) bedient werden können, ohne dass diese gegenseitig Einblick in Daten, Benutzerverwaltung und ähnliches haben. Ein IT-System, das diese Eigenschaft hat, bietet die Möglichkeit der disjunkten, mandantenorientierten Datenhaltung, Präsentation der Daten (GUI) und Konfiguration (Customizing). Jeder Kunde hat dann ledeiglich Zugriff auf seine Daten.

1.4 Künstliche Intelligenz (KI)

Der etwas esoterisch anmutende und an Arthur Clarkes HAL erinnernde Begriff Künstliche Intelligenz (engl. AI = artificial intelligence) wurde schon Mitte der fünfziger Jahre geprägt. Heute versteht man darunter eine Teildisziplin der Informatik, bei der es grundsätzlich darum geht, die menschliche Intelligenz (falls vorhanden) zu verstehen, dies mittels Computer zu simulieren um damit Computerprogramme zu ersinnen, die in der Lage sind, Probleme auf intelligente Art und Weise zu lösen [19, 26, 55, 60, 71, 79].

Zur Künstlichen Intelligenz werden üblicherweise die folgenden Anwendungsgebiete gezählt:

- **Sprachverarbeitung**

Sprachverarbeitung bedeutet das Verstehen von natürlicher Sprache (Umgangssprache, wohlgemerkt), die zur Verarbeitung als geschriebener Text via Tastatur oder als gesprochener Text über Spracheingabesysteme eingegeben werden. Das fundamentale KI Problem besteht in der Sprachanalyse, also der syntaktischen und semantischen Textinterpretation. Die Umgangssprache ist eigentlich ein völlig unzureichendes Werkzeug, um dem Computer mitzuteilen, was er tun oder lassen soll. Denn die Umgangssprache ist unpräzise, oft unvollständig und noch öfters mehrdeutig, ganz zu schweigen von doppelsinnigen Ausdrucksweisen. Schließlich hängen viele Worte der Umgangssprache vom Zusammenhang ab.

Ein kleines Beispiel mag da hilfreich sein. Jedem ist das Wort *Staubecken* geläufig, es hängt jedoch vom Zusammenhang ab, ob damit monatealte Schmutzsedimente in Arbeitszimmerecken gemeint sind oder ein großer Behälter zum Auffangen von Wasser. Jeder halbwegs gebildete Mitteleuropäer wird wohl in der Lage sein, den Sinn des Wortes Staubecken im Zusammenhang mit dem vorher Gesagten zu verstehen, ein Computer kann das ohne weiteres nicht!

- **Verarbeitung optischer Informationen (Bilderkennung)**

Bilderkennung oder -interpretation bedeutet die inhaltliche Auswertung von Bildinformationen, zum Beispiel von Röntgen- und Ultraschallaufnahmen zur Diagnose von medizinischen Anomalien, von Satellitenaufnahmen zur Erkennung von Waldschäden oder von Analysen von Blasenkammeraufnahmen in der Hochenergiephysik usw. Die grundlegende Voraussetzung, daß Bilderkennung funktioniert ist die sogenannte *Mustererkennung* (pattern recognition), die man einem Computer erst mühsam beibringen muß.

De facto zeigt die Bilderkennung die ganze Problematik der KI. Jeder kennt das Phänomen, wenn man ein Gesicht sieht, dann weiß man sofort (irgendwie...), daß man den- oder diejenige schon mal gesehen hat. Das

klappt im menschlichen Gehirn sofort (man weiß sofort, daß man dieses Gesicht schon einmal gesehen hat) ohne die zeitaufwendige Prozedur eine Datenbank mit gespeicherten Bildern durchkämmen zu müssen (so würde ein Computer an die Sache 'rangehen). Das "Muster" des Gesichtes ist bekannt. Genau diese Mustererkennung einem Computer beizubringen ist das Problem.

- **Robotik**

Roboter sind frei programmierbare Maschinen, die ihre Aktionen aufgrund von Meldungen von Sensoren der unterschiedlichsten Art ausüben. Ziel dieser Teildisziplin der KI ist die Entwicklung neuer Generationen von Maschinen, die in der Lage sind, in immer stärkerem Ausmaß aus den Umgebungseinflüssen selbständig sinnvolle Aktionen auszuführen.

- **logische Deduktion und Problemdeduktion**

Deduktionssysteme sind Programme, die in der Lage sind, Behauptungen, die sich in Form mathematischer Sätze formulieren lassen, zu beweisen (siehe [83]). Anwendungen sind unter anderem die Verifikation von DV Programmen hinsichtlich ihrer Spezifikationen oder die Verifizierung von IC Schaltkreisentwürfen.

- **Expertensysteme**

Expertensysteme befassen sich mit der Erfassung und Speicherung des Wissen von Experten und darauf basierenden Mechanismen zur automatischen Lösung von Problemen. Gegenüber konventionellen EDV Programmen unterscheiden sich Expertensysteme im wesentlichen in zwei Punkten:

1. die Fähigkeit, Schlüsse zu ziehen, d.h. Expertensysteme sind in der Lage auch dann einen Lösungsweg zu finden, wenn dieser nicht klar durch Algorithmen strukturiert ist (Schlußfolgerungsfähigkeit).
2. die Fähigkeit, den Schlußfolgerungsprozeß auch selbständig zu erklären (Erklärungsfähigkeit).

In der Praxis werden Expertensysteme dazu eingesetzt, auf speziellen Fachgebieten das Wissen menschlicher Experten zu verwerten und allgemein nutzbar zu machen. Dies wird dadurch erreicht, dass aus gespeichertem Expertenwissen (Wissensbasis) in Verbindung mit anwendungsspezifischen Fakten Schlußfolgerungen gezogen werden.

Expertensysteme werden bei der medizinischen Diagnose, bei der Fehlerdiagnose oder auch nach der Suche nach Bodenschätzen eingesetzt.

1.5 Datenverarbeitung

Derjenige Vorgang, der sich auf die

- Erfassung
- Speicherung
- Übertragung
- Transformation

von Daten bezieht, nennt man **Datenverarbeitung**. Dies kann sich zunächst sowohl auf die manuelle als auch auf die maschinelle Verarbeitung beziehen.

Den Prozeß der Datenverarbeitung wollen wir uns anhand eines einfachen Beispiels klarmachen nämlich eine einfach, manuelle Rechnungserstellung für einen Kunden.

Ohne Anspruch auf Vollständigkeit läßt sich diese Tätigkeit in folgende Schritte zerlegen:

- a) Datenerfassung
 - ⇒ Lieferscheine
 - ⇒ Kundendaten
 - ⇒ Artikeldaten
- b) Datenabspeicherung
 - ⇒ Erfasste Daten werden zusammengestellt, zusammengefaßt und aufgeschrieben, z.B. auf einen Notizzettel
- c) Datenbearbeitung
 - ⇒ Ermittlung des Rechnungsbetrages, Menge x Einzelpreis, Skonto, Mehrwertsteuer, Frachtkosten. etc.
- d) Datenabgabe
 - ⇒ Vorliegen der fertigen Rechnung
- e) Datenausgabe
 - ⇒ Versendfertige Rechnung
- f) Datenübertragung
 - ⇒ Versenden der Rechnung mittels Post an den Kunden

Im Rahmen der EDV betrachten wir hier ausschließlich: Die maschinelle Verarbeitung von Daten mit dem Computer.

Der Oberbegriff dazu ist:

EDV – elektronische Datenverarbeitung

Dies ist der Sammelbegriff für die Datenverarbeitung und Datenverarbeitungssysteme, die nicht mechanisch, sondern elektronisch arbeiten, also in der Regel sind dies Computer.

1.6 Datenverarbeitungssystem

Unter einem Datenverarbeitungssystem im weitesten Sinne versteht man eine elektronisch arbeitende Einheit, die mittels gespeicherter Programme Daten automatisch verarbeitet, also

- mathematische
- umformende
- übertragende
- speichernde

Operationen ausführt. Zu Datenverarbeitungssystem gibt es eine Reihe synonyme Begriffe, die wir ebenfalls häufig verwenden werden:

- * Computer
- * EDV-System
- * DV-System (Datenverarbeitungssystem)
- * Rechenanlage
- * Rechner

Merke:

Ein Datenverarbeitungssystem führt die im letzten Beispiel aus Abschnitt 1.5 aufgeführten Vorgänge automatisch und viel schneller durch, als dies manuell möglich ist.

Ein Computer ist eine

speicherprogrammierbare Rechenanlage

Dies erfordert eine nähere Erklärung.

Dazu betrachten wir ein Auto. Der Zweck eines Autos ist der eines Fortbewegungsmittels, um auf dem Landweg von einem Punkt A zu einem Punkt B zu gelangen. Es ist wohl einleuchtend, daß es einer völligen Umkonstruktion des Autos bedarf, um daraus ein Fortbewegungsmittel zu machen, sodaß man mit dem gleichen Gerät auf dem Luftweg von A nach B gelangt. Eine Maschine

wie das Auto dient also (in der Regel) nur einem einzigen Zweck und damit diese Maschine einen anderen Zweck erfüllen kann, muß das Ding völlig umgebaut werden. Mit einem Computer verhält sich dies etwas anders. O.k., fliegen kann man damit auch nicht so ohne weiteres, dennoch erfüllt diese Maschine die unterschiedlichsten Aufgaben. Mit Hilfe eines Computers kann man Texte erstellen, Spiele spielen, Tabellen berechnen, Kundenabrechnungen ermitteln, Statistiken erstellen, Pläne grafisch darstellen und jede Menge andere Dinge tun. Damit ein Computer diese verschiedenen Dinge tun kann, muß man ihn nicht jedesmal komplett umbauen (das war in den Kindertagen des Computers so), es genügt dazu, jeweils adäquate Programme einzusetzen. So wird erst bei der Verwendung eines Textverarbeitungsprogramms aus einem Computer eine Textverarbeitungsmaschine. Damit ein Computer eine Statistik berechnen kann, muß ein Statistikprogramm eingesetzt werden. Die verschiedenen Programme werden in einen Speicher - den sogenannten Arbeitsspeicher - geladen. Dieser Arbeitsspeicher stellt gewissermaßen die Werkstatt dar, in der die Verarbeitung der Daten stattfindet. Daher nennt man den Computer auch speicherprogrammierbare Rechenanlage.

Damit ein Computer ein Computer ist, **muß** er

- ⇒ frei programmierbar sein
- ⇒ über einen Arbeitsspeicher zur Aufnahme von Programmen und Daten verfügen
- ⇒ die Möglichkeit besitzen, Geräte zur Ein- und Ausgabe von Daten anzuschließen

Technisch realisiert wird dieses Konzept der speicherprogrammierbaren Rechenanlage (man erkennt hier die VON NEUMANN Architektur¹⁵) durch eine Gliederung in drei Komponenten:

Jedes EDV-System besteht prinzipiell aus drei Komponenten

HARDWARE SYSTEMSOFTWARE ANWENDUNGSSOFTWARE

Unter diesen drei Komponenten versteht man folgendes:

Hardware	⇒	physikalische Geräte, die die Anlage bilden
-----------------	---	---

¹⁵Detailliertes dazu wird in Kapitel [2] diskutiert.

Systemsoftware	\Rightarrow	Programme, die es ermöglichen, die Hardware zu nutzen. Diese Programme heißen auch Betriebssysteme
Anwendungssoftware	\Rightarrow	Programme für spezielle Datenverarbeitungsanwendungen

1.7 Who is who in der Informatik

Bei dem Einsatz und der Benutzung der Informatik unterscheidet man verschiedene Personengruppen. Dies geschieht allerdings in keiner einheitlichen Weise, dies liegt wohl mit daran, dass es im sich rasant entwickelnden Umfeld der Informatik keine einheitlichen Tätigkeits- und Berufsbezeichnungen¹⁶ gibt. Eine grobe Einteilung kann hinsichtlich der ausgeübten Tätigkeit durchgeführt werden, wobei sich diese natürlich nicht gegenseitig ausschließen. So gibt es

- **Anwender oder User**

Diese Personenkreis ist reiner *Endbenutzer* der Anwendungen. Der typische Anwender benutzt die Hard- und Software um die ihm/ihr gestellte Aufgabe zu effektiv und schnell zu lösen. Haupttätigkeit ist die Eingabe von Daten in eine Datenbank, das Erstellen von Texten im Bürokommunikationsumfeld oder die Bearbeitung von Grafiken.

- **Systemverwalter, System-Administrator, System-Operator oder Web-Master**

Die Tätigkeiten dieses Personenkreises hat typischerweise administrativen Charakter und beinhaltet zum Beispiel

- die Wartung und Betreuung von Betriebssystemen und Betriebssystem-Komponenten
- die Verwaltung der am System arbeitenden Benutzer
- Netzwerk-Verwaltung
- Administration von Servern (Web Server, Mail Server, Datenbank Server)
- Analyse betriebsrelevanter Log-Dateien (Monitoring) und systemseitige Fehlerbeseitigung
- Backup-Administration (Datensicherungen)
- Sicherstellung der Verfügbarkeit von Produktivumgebungen

- **Help-Desk Support**

Beim Einsatz von Rechnern in Unternehmen ist es unabdingbar, daß der Anwender auf Probleme bei der Arbeit mit der Software oder Hardware stößt. Um diesen leidgeprüften Mitmenschen zu helfen richtet man einen sogenannten *Help Desk* ein. Dies ist eine zentrale Stelle für Anwender, um bei auftretenden Problemen Unterstützung zu bekommen.

- **System-Entwickler**

Dieser Personenkreis ist in erster Linie mit der Entwicklung von Anwendungen beschäftigt. Für dieses Tätigkeitsfeld gibt es auch die Bezeichnung

¹⁶Davon kann man sich leicht durch einen Blick in eine überregionale Wochendendausgabe einer Zeitung in die Stellenausschreibungen überzeugen.

Software-Ingenieur oder **Systems-Engineer**. Die grundlegende Vorgehensweisen bei der Erstellung von Anwendungen unterliegen den Prinzipien des **Software-Engineerings**. Hierunter versteht man die systematische Anwendung ineinandergreifender, formaler Techniken für

- die Planung
- die Analyse
- den Entwurf
- die Implementierung (Programmierung)
- das Testen
- die Wartung und Pflege

von Informationssystemen auf einer unternehmensweiten Basis.

Aufgrund der Vielfalt der vorhandenen informationstechnischen Infrastrukturen, wird dieses Tätigkeitsfeld weiter differenziert in:

- Software-Entwickler
- Entwickler mit Know-How in einer bestimmten Programmiersprache, z.B. Java-Entwickler, C/C++ Entwickler, Delphi-Entwickler, etc.
- Datenbank-Entwickler, dessen Aufgabe die Erstellung von Datenbank-Applikationen ist, diese heißen auch Application Designer
- Web-Applikations Entwickler; diese Programmierer sind auf die Erstellung von Internet-Anwendungen spezialisiert
- Systemprogrammierer; diese Programmierer entwickeln hardwarenahe Programme, z.B. Chipkarten, Maschinensteuerungen, Hardware-treiber

- **IT-Projekt-Manager und Projektleiter**

Die heutigen komplexen Software-Entwicklungen unterliegen den Vorgaben des Projektmanagements. Die Aufgaben des IT-Projekt Managers besteht in der Erstellung und Umsetzung

- der Projektplanung
- der Projektüberwachung
- der Projektsteuerung

- **Security-Engineer.**

Diese Personengruppe ist auf Sicherheitsaspekte der Datenverarbeitung spezialisiert. Aufgabe ist es unter anderem, Sicherheitskonzepte für eine bestehende IT-Infrastruktur zu erarbeiten, zu planen und umzusetzen.

- Dann gibt es noch den großen Kreis der **IT-Berater, IT-Consultants** oder wie immer die sich nennen¹⁷. IT-Berater operieren typischerweise kundenorientiert, i.e. sie bilden die 'Schnittstelle' zwischen Auftraggeber und Auftragnehmer eines Software-Projektes.

¹⁷... denken Sie sich auch einen Namen aus!

1.8 Standards in der Informationstechnologie

Standards lassen sich in zwei Kategorien aufteilen:

- **de-facto Standards**

Darunter versteht man Standardisierungen, die einfach ohne konzeptionelle Vorarbeiten und formalen Plan als solche akzeptiert sind. Beispielsweise sind die IBM-PC (und Nachfolger) der *de-facto* Standard der Büro-Computer, denn eine Vielzahl von Herstellern haben IBM's ursprünglichen PC als Vorlage benutzt und weitestgehend kopiert. Ein weiterer *de-facto* Standard bildet das UNIX Betriebssystem an naturwissenschaftlichen Fakultäten.

- **de-jure Standards**

Darunter versteht man formale, legale Standardisierungen, die von autorisierten Gremien erarbeitet wurden. Die internationalen Gremien unterteilt man dabei in zwei Klassen:

- Gremien, die durch Verträge und/oder Vereinbarungen zwischen nationalen Regierungen eingerichtet werden.
- Freiwillige Gremien, die nicht durch Regierungsvereinbarungen eingerichtet sind.

Wer den legalen Stand der Dinge in der nationalen Telekommunikationswelt festlegt, variiert stark von Land zu Land. Bis vor wenigen Jahren gehörte die BRD zu jener Kategorie von Ländern, in denen eine Regierungsbehörde¹⁸ monopolistisch alle Standards im Kommunikationsbereich verwaltete und festlegte. Dazu gehörten u.a. Postdienst und Fernmeldedienste (*e.g.* Telegraphie, Telefon). In den meisten Ländern der Erde ist die Situation nach wie vor sehr ähnlich. Nach dem Vorbild der amerikanischen Telefongesellschaft AT&T, die 1984 in mehrere unabhängige Gesellschaften aufgelöst wurde, wurde Mitte der 90er Jahre die Deutsche Bundespost in mehrere eigenständige und unabhängige Gesellschaften aufgeteilt (Post AG, Telekom, Postbank). Denn man erkannte selbst hier, daß Konkurrenz das Geschäft belebt.

Weltweit beobachtet man, dass die Monopolstrukturen der Regierungsbehörden aufgebrochen werden, die Tendenz geht eindeutig hin zur Privatisierung und Liberalisierung des Kommunikationsmarktes.

Da es unterschiedliche Anbieter von Telekommunikationsdienste sowohl auf nationaler als auch internationaler Ebene gibt, bedarf es einer gewissen Kompatibilität auf globaler Ebene. Dies allein zum Beispiel zu dem banalen Zweck, daß Menschen (und Computer) mit ihren Artgenossen in anderen Ländern telefonieren (bzw. Daten austauschen) können.

¹⁸Das war das Bundespostministerium.

1.8.1 Die International Telecommunication Union

Bereits im Jahre 1865 trafen sich eine Reihe von europäischen Regierungsvertretern, um den Vorläufer der heutigen ITU (International Telecommunication Union) zu gründen. Die Aufgabe der ITU war die Standardisierung der internationalen Telekommunikation, was seinerzeit natürlich die Telegraphie betraf. Bereits damals erkannte man, dass man mit einem Problem konfrontiert wird, wenn die Hälfte der Länder den Morse-Code für die telegraphische Übermittlung benutzt, die andere Hälfte der Länder irgend einen anderen Code.

Im Jahre 1947 wurde die ITU der UNO zugeordnet mit Sitz in Genf (Schweiz)¹⁹.



Die ITU baut sich aus drei Hauptabteilungen auf:

- 1 der Radio-Kommunikationsbereich (ITU-R)
- 2 der Telekommunikations-Standardisierungsbereich (ITU-T)
- 3 der Entwicklungsbereich (ITU-D)

Der Radio-Kommunikationssektor ITU-R befaßt sich mit der weltweiten Zuteilung der Radiofrequenzen für die rivalisierenden Interessengruppen.

Die ITU-T war von 1956 bis 1993 unter dem Namen CCITT²⁰ bekannt und wurde 1993 aus verwaltungstechnischen Gründen reorganisiert und firmiert seitdem als ITU-T. Dennoch begegnet man heute noch den CCITT Empfehlungen, z.B. die CCITT-X.25, obwohl diese Standards ab 1995 die ITU Bezeichnung führen.

Die ITU-T hat fünf Klassen von Mitgliedern:

- 1 Regierungsbehörden (e.g. nationale Postbehörden)
- 2 Anerkannte private Organisationen (e.g. AT&T, British Telecom, Deutsche Telekom)
- 3 Regionale Telekommunikationsverbände (europ. ETST)
- 4 Anbieter von Telekommunikationsprodukten und wissenschaftliche Organisationen

¹⁹Siehe auch die URL:

<http://www.itu.int/home/index.html>

²⁰Dieses Kürzel steht für *Comité Consultatif International Télégraphique et Téléphonique*.

5 sonstige Organisationen (*e.g.* Banken, Fluggesellschaften)

Die Aufgabe der ITU-T besteht darin, technische Empfehlungen für Schnittstellen im Telekommunikationsbereich zu erstellen. Diese Empfehlungen bilden oftmals die Grundlage international anerkannter Standards.

Die Bereiche der ITU-T sind:

- A** Organization of the work of ITU-T
- B** Means of expression: definitions, symbols, classification
- C** General telecommunication statistics
- D** General tariff principles
- E** Overall network operation, telephone service, service operation and human factors
- F** Non-telephone telecommunication services
- G** Transmission systems and media, digital systems and networks
- H** Audiovisual and multimedia systems
- I** Integrated services digital network
- J** Cable networks and transmission of television, sound programme and other multimedia signals
- K** Protection against interference
- L** Construction, installation and protection of cables and other elements of outside plant
- M** TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
- N** Maintenance: international sound programme and television transmission circuits
- O** Specifications of measuring equipment
- P** Telephone transmission quality, telephone installations, local line networks
- Q** Switching and signalling
- R** Telegraph transmission
- S** Telegraph services terminal equipment
- T** Terminals for telematic services
- U** Telegraph switching

- V Data communication over the telephone network
- X Data networks and open system communications
- Y Global information infrastructure and Internet protocol aspects
- Z Languages and general software aspects for telecommunication systems

Die Standards der ITU-T/CCITT werden durch einen Kennbuchstaben — dieser entspricht den obigen Bereichen, einen Punkt und eine Seriennummer benannt, *e.g.* *X.500*. International anerkannte Standards der ITU-T bzw. CCITT für *Datenübertragung* sind in der V-Reihe geregelt, diejenigen Standards für Netzwerke sind in der X-Reihe festgelegt, also

- X – Standards für Datenübertragung in digitalen Netzen
- V – Standards für Datenübertragung über analoge Telefonnetze

Beispiele von Standards der ITU-T:

- Die V.24 Schnittstelle (vergleichbar mit der seriellen RS-232 Schnittstelle), die unter anderem die Verteilung und Bedeutung der unterschiedlichen Pins der seriellen Schnittstelle am Computer festlegt.
- V.90 bzw. V.92 bezeichnet den aktuellen Standard für Datenübertragung mit einem Modem mit einer Datenübertragung von 56 kbps (kilobit pro Sekunde) downstream und 33.6 kbps upstream.
- X.25 regelt den Datenaustausch zwischen einem Endgerät und einer Übertragungseinrichtung für ein paketvermittelndes Datennetz. Dieser Standard wird heute universell eingesetzt um über einen Rechner auf paketvermittelnde Netze zugreifen zu können und wird beispielsweise bei Datenübertragung über eine ISDN Verbindung verwendet²¹
- X.400 regelt die Funktionalität und Übertragungsprotokolle für E-Mail Systeme.
- X.500 regelt den sogenannten *Directory Service* in einem Netzwerk. Ein Directory ist im wesentlichen ein Server, oder mehrere verteilte Server, die eine Datenbank bereitstellen, welche Informationen über Benutzer enthalten.
- X.509 beschreibt einen Authentifizierungsdienst durch den X.500 Standard. X.509 ist ein verbreiteter Standard, da die Struktur der Zertifikate und die Authentifizierungsprotokolle, die in X.509 definiert sind, in einer Reihe von Netzwerkprotokollen wie S/MIME, IPSec oder SSL/TLS eingesetzt werden

²¹Siehe dazu [96], Chapter 10.3.

- In der 1980er Jahren wurde von der ITU-T zusammen mit der ISO die *Joint Photographic Experts Group* etabliert, diese Gruppe entwickelte den *jpeg*-Standard. Die JPEG-Norm beschreibt lediglich sogenannte Bildkompressionsverfahren, legt aber nicht fest, wie die so entstandenen Daten gespeichert werden sollen. Gemeinhin werden mit *JPEG-Dateien* oder *JPG-Dateien* Dateien im Grafikformat JPEG File Interchange Format (JFIF) bezeichnet.

Seit Ende der 1990er Jahren ist die *Bundesnetzagentur*²² mit Sitz in Bonn für die Einhaltung und Zuteilung von Standards im Telekommunikationsbereich in Deutschland zuständig, *e.g.* Vergabe für Funkfrequenzen für zellulare Netze, UMTS, LTE.²³

1.8.2 International Organization for Standardization



International anerkannte Standards werden von der ISO (International Organization for Standardization) verfasst und verabschiedet. Die ISO ist eine freiwillige, nicht-staatliche Organisation, die im Jahre 1946 gegründet wurde mit Sitz in Genf (Schweiz)²⁴. Die Mitglieder der ISO sind die nationalen Standardisierungsgremien der 89 Mitgliedsstaaten, z.B. ANSI (USA), BST (UK), AFNOR (FR), DIN (BRD) usw.



Abbildung 1.3: Der leibhaftige Sitz der ISO in Genf.

²²Siehe z.B. unter der Homepage www.bundesnetzagentur.de.

²³UMTS steht für Universal Mobile Telecommunication System und ist die Mobilfunktechnologie der dritten Generation, LTE ist die Nachfolge von UMTS und steht für Long Term Evolution.

²⁴Detailinformationen über die ISO kann man sich im Internet auf der Page www.iso.ch besorgen.

Die ISO erstellt Standards für so ziemlich alles, was man sich denken kann, das fängt bei Schraubengrößen an, geht über Bolzennormen zu Größen von Batterien ("Babyzellen") zum Sieben-Schichten-Modell für Netzwerke oder Farbmarkierungen von Telefonkabeln. Die ISO hat mittlerweile über 5.000 international gültige Standards in die Welt gesetzt. Diese Organisation besteht aus etwa 200 technischen Komitees, die gemäß der Reihenfolge ihrer Entstehung nummeriert sind. So befaßt sich TC1 mit Schrauben und TC97 mit Computern und Informationsverarbeitung. Jedes Komitee besteht aus Unterkomitees (SC = SubCommittees) und diese wiederum aus Arbeitsgruppen (WG = WorkingGroups). Die eigentlichen Resultate werden in den WGs durch über 100.000 freiwillige Mitarbeiter weltweit erarbeitet. Viele der Mitarbeiter der ISO werden durch ihre Brötchengeber ermutigt, für die ISO zu arbeiten, natürlich mit dem Hintergedanken, daß ihre Produkte durch die ISO zum Standard geädelt werden. Aus ähnlichen Motivationsgründen delegieren Regierungen 'freiwillige' Mitarbeiter mit dem Ziel, ihre nationalen Entwicklungen zum internationalen Standard via ISO zu erkoren. Auch akademische Experten arbeiten in mannigfaltiger Weise in den WGs der ISO mit.

Bezüglich der Standardisierungen im Telekommunikationsbereich kooperieren die ISO und die ITU-T (die ISO ist Mitglied der ITU-T) um zu verhindern, daß in diesem Bereich zwei inkompatible Standards in die Welt gesetzt werden.

Beispiele für ISO Standards, die in der IT-Welt eine Rolle spielen:

- ISO 9660 Dateisystem für CD-ROM
- ISO 8859 Codepages für verschiedene länderspezifische Sprachen
- ISO 9000 – 9004 ist eine Normenserie der ISO, um Qualitätsmanagementsysteme international zu vereinheitlichen, und damit vergleichbar zu machen (siehe dazu [7], Kap. 4).
- Das **ISO-OSI Modell** ist *das* Referenzmodell der Netzwerkarchitektur. Unter dem ISO-OSI Modell versteht man das Referenzmodell für die Kommunikation offener Systeme. Dieses Modell der ISO bildet die konzeptionelle Grundlage sämtlicher Netzwerkarchitekturen. Die grundlegende Idee dieses Modells ist eine schichtenförmige Aufteilung der verschiedenen Ebenen eines Netzwerksystems (Hardware, Flußsteuerung, Fehlerprüfung, Benutzerschnittstelle). Jede der insgesamt sieben Schichten hat in diesem Modell eine klar definierte Aufgabe innerhalb der Kommunikation und wechselwirkt ausschließlich mit benachbarten Schichten.

In Deutschland hat das *Deutsche Institut für Normung e.V.* (DIN) ähnliche Aufgaben national wie die ISO auf weltweiter Bühne. Auf der DIN Homepage²⁵ liest man z.B.:

²⁵Siehe die URL <http://www.din.de>

Das DIN Deutsches Institut für Normung e.V. ist ein eingetragener Verein mit Sitz in Berlin, keine staatliche Instanz.

Die Normungsarbeit des DIN ist eine technisch-wissenschaftliche Dienstleistung für alle Bürger unseres Landes. Normung nutzt der Volkswirtschaft insgesamt. Das DIN ist der runde Tisch, an dem sich Hersteller, Handel, Verbraucher, Handwerk, Dienstleistungsunternehmen, Wissenschaft, technische Überwachung, Staat, jedermann, der ein Interesse an der Normung hat, zusammensetzen, um den Stand der Technik zu ermitteln und in Deutschen Normen niederzuschreiben. Diese Regeln der Technik dienen der Rationalisierung, der Qualitätssicherung, der Sicherheit, dem Umweltschutz und der Verständigung in Wirtschaft, Technik, Wissenschaft, Verwaltung und Öffentlichkeit.

Die Normungsarbeit wird in 4300 Arbeitsausschüssen mit 33.800 ehrenamtlichen Mitarbeitern geleistet. Fertige Normen werden mindestens alle 5 Jahre auf ihre Aktualität hin überprüft.

Definitionen:

”Normung ist die einmalige, bestimmte Lösung einer sich wiederholenden Aufgabe unter den jeweils gegebenen wissenschaftlichen, technischen und wirtschaftlichen Möglichkeiten.”

Nach Otto Kienzle, Mitbegründer des DIN

”Normung ist die planmässige, durch die interessierten Kreise gemeinschaftlich durchgeführte Vereinheitlichung von materiellen und immateriellen Gegenständen zum Nutzen der Allgemeinheit.”

DIN 820 Teil 1

1.8.3 Das IEEE



Ein weiteres Schwergewicht in der Welt der Standards bildet das **Institute of Electrical and Electronics Engineers (IEEE)**²⁶. Das IEEE ist die größte

²⁶Siehe zum Beispiel auch die URL <http://www.ieee.org>.

professionelle technische Gesellschaft und wurde im Jahre 1884 von einer Handvoll Praktiker der damals gerade aufkommenden neuen Disziplin des elektrischen Ingenieurs gegründet. Heutzutage arbeiten für dieses Institut über 320.000 Mitarbeiter in etwa 150 Ländern. Neben der Publikation von Zeitschriften oder der alljährlichen Organisation zahlreicher Meetings und Symposien, unterhält das IEEE eine Reihe unterschiedlicher Standardisierungsgruppen, deren Aufgabe u.a. das Erstellen von Normen für die Computerwelt ist. Beispielsweise ist der IEEE Standard IEEE 802 *die* Meßlatte für lokale Netzwerke. Dieses Maß aller Dinge im Bereich lokale Netzwerke wurde von der ISO als Norm ISO 8802 übernommen.

Wenn in Computerkreisen vom IEEE die Rede ist, ist eigentlich stets die Projektgruppe 802 gemeint. Diese ist — wie erwähnt — zuständig für Standardisierung im lokalen Netzwerkbereich, aber auch in geographisch übergreifenden Netzen (*i.e.* LAN und WAN). In der Tabelle [1.2] sind wichtige IEEE–Arbeitsgruppen und ihre Tätigkeitfelder aufgelistet.

Arbeitsgruppen	Aktuelle Tätigkeit
802.1 Higher Layer LAN Protocols	–
802.2 Logical Link Control	inaktiv
802.3 CSMA/CD und 100BaseT	Standardisierung des Gigabit Ethernet für IEEE 802.3z abgeschlossen Standard 802.3ad - Link aggregation
802.4 Token Bus	z. Zt. inaktiv
802.5 Token Ring	Standardisierung des 100 Mbit/s High Speed Token Ring (HSTR) als IEEE 802.5t abgeschlossen
802.6 MAN	z. Zt. inaktiv
802.7 Broadband Tag (BBTAG)	z. Zt. inaktiv
802.8 Fiber Optic Tag (FOTAG)	erarbeitet praktische Empfehlungen zur Glasfasertechnik
802.9 Integrated Services LAN (ISLAN)	–
802.10 Standard for Inoperative LAN Security	Abschlussaktivitäten dieser Arbeitsgruppe
802.11 Wireless LAN (WLAN)	erarbeitet Vorschläge zur Verbesserung der Übertragungsgeschwindigkeit
802.12 Demand Priority Access Method	–
802.13 nicht verwendet	
802.14 Kabelmodems, Datenkommunikation über TV Kabelnetze	Basisstandard festgelegt
802.15 Wireless Personal Area Network (WPAN)	Im Frühjahr 1999 ins Leben gerufen – Kommunikation verschiedener Funk- Endgeräte (PC, Handy, Pager, usw.) untereinander
802.16 Broadband Wireless Access	Im Frühjahr 1999 ins Leben gerufen – Spezifikation des physikalischen und des MAC-Layers zur Definition von Standards für die kabellose Breitbandkommunikation.

Tabelle 1.2: Die Projektgruppe 802 des IEEE.

Weitere IEEE Standards, die in der IT-Welt benutzt werden sind:

- IEEE 754 Standardformat für Gleitkommazahlen,
- IEEE 1394 Firewire Spezifikation,
- Verschiedene Standardisierungen von Programmiersprachen.

1.8.4 Weitere Organisationen

■ Das **American National Standards Institute (ANSI)**



Das *American National Standards Institute* ist ein in den USA im Jahre 1918 gegründetes nicht-staatliches Institut für Normung, was der DIN in Deutschland entspricht. Dennoch werden die von dem ANSI erstellten Empfehlungen allgemein als bindend anerkannt. Oftmals werden diese Empfehlungen dann von der ISO in den Rang einer internationale Norm erhoben. Einige Standards, die ANSI in die Welt ge-

setzt hat, sind:

- ANSI – C Dies ist ein standardisierter Sprachumfang der Programmiersprache C. Ein in ANSI - C geschriebenes C-Programm ist von allen Compilern auf jeder Plattform übersetzbar, wenn dieser Compiler ANSI-C-kompatibel ist.
- FORTRAN 77 und FORTRAN 90; Standards für die Programmiersprache FORTRAN
- ANSI Code und ASCII Code
- ANSI X9 Entwicklung kryptographischer Standards für Bank- und Finanztransaktionen, Authentifizierungsmechanismen,



Abbildung 1.4: Richard Stallman

■ Das **National Institute of Standards and Technology (NIST)**²⁷

²⁷Siehe die URL: <http://www.nist.gov>

Das NIST – früher *National Bureau of Standards* (NBS) – ist eine Abteilung des amerikanischen Wirtschaftsministeriums (U.S. Department of Commerce), dessen Aufgabe die Entwicklung von Standards und Empfehlungen im Umfeld der betriebswirtschaftlichen Anwendung und Einsatzes der Informationstechnologie ist. NIST publiziert die offiziellen Standards als *Federal Information Processing Standards* (FIPS). Beispielsweise ist der *Data Encryption Standard* unter FIPS PUB 46 als offizielles Verschlüsselungsverfahren empfohlen. Unter FIPS PUB 147 sind die Spezifikation des *Advanced Encryption Standards* veröffentlicht, der Nachfolger des DES.

■ Die **Free Software Foundation (FSF)**

Die Free Software Foundation (FSF)²⁸ wurde 1985 von **Richard Stallman** (siehe Abbildung [1.4]) gegründet [23] mit dem Ziel der Erstellung und Verteilung freier — das heißt nicht-proprietärer — Software, die jedem Benutzer zur Verfügung steht. Die FSF organisiert und finanziert das GNU-Projekt²⁹ und koordiniert die *Open Source* Entwicklung. Flaggschiff der Open Source ist das Betriebssystem LINUX.

■ Die **Association for Computing Machinery (ACM)**



Die **Association for Computing Machinery**, kurz ACM³⁰ ist die 1947 gegründete amerikanische Vereinigung von Informatikern mit Sitz in New York City. Seit 1966 verleiht die ACM jährlich den **A. M. Turing Award**, benannt nach ALAN TURING. In technischen Kreisen wird diese Auszeichnung gleichwertig zu dem Nobel-Preis angesehen. Seit 1966 sind die Preisträger:

²⁸Siehe die URL: <http://www.fsf.org>.

²⁹Siehe die URL: <http://www.gnu.org>

³⁰Siehe die URL: <http://www.acm.org>.

1966	A. J. Perlis	1983	Dennis M. Richie, Ken Thompson
1967	Maurice V. Wilkes	1984	Niklaus Wirth
1968	Richard Hamming	1985	Richard M. Karp
1969	Marvin Minsky	1986	John Hopcroft, Robert Tarjan
1970	J. H. Wilkinson	1987	John Cocke
1971	John McCarthy	1988	Ivan Sutherland
1972	E. W. Dijkstra	1989	William Kahan
1973	Charles Bachman	1990	Fernando J. Corbato
1974	Donald E. Knuth	1991	Robin Milnor
1975	Allen Newell, Herbert A. Simon	1992	Butler W. Lampson
1976	Michael O. Rabin, Dana S. Scott	1993	Juris Hartmanis
1977	John Backus	1994	Richard E. Stearns
1978	Robert W. Floyd	1995	Edward Feigenbaum
1979	Kenneth E. Iverson	1996	Amir Pnueli
1980	C. Anthony R. Hoare	1997	Douglas Engelbart
1981	Edgar F. Codd	1998	James Gray
1982	Stephen A. Cook	1999	Frederick P. Brooks, Jr.
		2000	Andrew Chi-Chih - Yao
		2001	Ole-Johan Dahl, Kristen Nygaard
		2002	Ronald L. Rivest, Adi Shamir, Leo Adleman
		2003	Alan Kay
		2004	Vinton Cerf, Robert Kahn
		2005	Peter Naur
		2006	Frances E. Allen
		2007	Edmund M. Clarke, Joseph Sifakis, E. Allen Emerson
		2008	Barbara Liskov
		2009	Charles P. Thacker
		2010	Leslie Valiant

Die ACM publiziert eine Reihe von Fachzeitschriften zu den unterschiedlichsten Teildisziplinen der Informatik. Die ACM untergliedert die Informatikthemen in 34 *Special Interest Groups*, kurz SIGs. Daher gibt es

- SIGACT – das ist die ACM Special Interest Group on Algorithms and Computation Theory,
- SIGCOMM – das ist die ACM Special Interest Group on Data Communications,
- SIGSOFT – das ist die ACM Special Interest Group on Software Engineering,
- SIGPLAN – das ist die ACM Special Interest Group on Programming Languages.

■ Die **Electronic Frontier Foundation (EFF)**

Die Electronic Frontier Foundation³¹ wurde im Juli 1990 von JOHN BARLOW und MITCH KARPOR³² als non-profit Organisation gegründet. Thema der EFF sind die Bürgerrechte im Cyberspace wie Zensur im Internet, Urheberrechte oder Schutz der Privatsphäre.

■ Das **Software Engineering Institute (SEI)**

Das Software Engineering Institute SEI³³ ist ein Forschungs- und Entwicklungszentrum für Software-Technologie an der Carnegie Mellon University. Das

³¹Siehe die URL: <http://www.eff.org>.

³²KARPOR ist auch Gründer der Firma Lotus.

³³Siehe die URL: <http://www.sei.cmu.edu>.

SEI wird vom U.S. Department of Defense (DoD) finanziell unterstützt. Aufgabe des SEI ist die Entwicklung von Methoden, Verfahren und Werkzeuge zur Erstellung zuverlässiger Software (Software Engineering).

Das SEI hat das *Capability Maturing Model* (CMM) entwickelt, eine Zertifizierungsprozedur für Software-Entwicklungsteams um deren Qualitäten zu verbessern.

■ Das Bundesamt für Sicherheit in der Informationstechnik, BSI

Das BSI wurde am 1. Januar 1991 gegründet und ist eine dem Bundesinnenministerium untergeordnete Bundesbehörde. Die Aufgaben des BSI umfassen unter anderem

- Einsatzzentrale des Bundes bei IT Gefährdungslagen
- Mitwirkung in der Task Force *Sicheres Internet*
- Entwicklung von Kriterien, Verfahren und Werkzeugen für die Implementierung, Prüfung und Bewertung der Sicherheit von IT Systemen oder - Komponenten
- Aufbau einer Public Key Infrastruktur durch Einrichtung von Trustcentern

■ Gesellschaft für Informatik (GI)

1.8.5 Internet Organisationen

Das anarchistische Internet verfügt über seine eigenen Standardisierungsmechanismen, klar abgekoppelt von der ITU oder der ISO. Grob erkennt man das daran, dass Leute bei ISO oder ITU Meetings stets Anzüge und Krawatte tragen, während die Treffen anlässlich der Verabschiedung von Internetstandards durch Levis oder Uniformen geprägt sind.³⁴

Mit der Entstehung des ARPANET rief das DoD (Department of Defense) ein informelles Gremium ins Leben, dessen ursprüngliche Aufgabe es war, die Entwicklung des ARPANET zu überwachen (siehe dazu auch [45], Chapt. 1 oder V. Cerf in RFC 1160). Im Jahre 1983 wurde dieses Komitee IAB (Internet Activity Board) getauft und später in *Internet Architecture Board* umfirmiert. Das IAB besteht aus vielen Arbeitsgruppen, jede dieser Gruppen unterhält eine sogenannte "Task Force", die sich um ein aktuelles, dringendes Problem kümmert. Das IAB trifft sich mehrmals im Jahr um Resultate zu erörtern und den Geldgebern, dem DoD und der NSF (National Science Foundation) Bericht zu erstatten. Soll ein neuer Standard in die Welt gesetzt werden (z.B. einen neuen Routing Algorithmus oder neues Format für IP Adressen), dann erstellt das IAB das Grobkonzept und kündigt die anstehenden Änderungen an, damit die Studenten

³⁴Einen Überblick über die Vielfalt an Internetorganisationen und deren Beziehungen untereinander findet man im RFC 4677, P. Hoffman, S. Harris, The Tao of IETF: A novice's guide to the working of the Internet Engineering Task Force.

und Hacker, die die eigentliche Softwareerstellung leisten, wissen, woran sie sind. Diese gesamte Kommunikation und Diskussion geschieht über die sogenannten **RFCs** (*Request For Comments*), die alle online und jedermann zur Verfügung stehen³⁵; die RFCs sind chronologisch nummeriert, gegenwärtig (Ende 2011) ist man bei beinahe 6.500 Dokumenten angelangt.

Im Sommer 1989 wurde das IAB aufgrund des enormen Wachstums des Internets reorganisiert. Die Entwickler wurden in dem **IRTF** (*Internet Research Task Force*) organisiert, das zusammen mit der **IETF** (*Internet Engineering Task Force*)³⁶ dem IAB untergeordnet wurde. Die IETF besteht aus vielen Areas, die bestimmte Themenbereiche der Netzwerktechnologie im Internet bearbeiten. Die Areas wiederum bestehen aus mehreren Working Groups. Eine Working Group des IETF beschäftigte sich beispielsweise mit der Standardisierung des neuen Internetprotokolls IPv6.

Zusammenfassend die wichtigen Gremien innerhalb des INTERNET:

- **Internet Architecture Board** oder **IAB**

Das IAB übernahm zunächst 1983 als Internet Activities Board Aufgaben zur Realisierung einer übergreifenden Internet-Architektur, bevor diese Institution zu Beginn der 90er Jahre in Internet Architecture Board³⁷ umbenannt wurde. Dieses Gremium verwaltet und veröffentlicht die RFCs, wacht über die Vergabe der Internet-Adressen und Domain-Namen und stellt die oberste Instanz bei Entscheidungsfragen dar.

- **Die Internet Engineering Task Force** oder **IETF**

In diesem Gremium³⁸ findet die eigentliche Arbeit der Entwicklung von Standards statt. Hier entwickeln Netzwerkdesigner, Hersteller von technischen Geräten (e.g. Cisco) und Computer Wissenschaftler neue Konzepte für Netzwerkstandards. Das geschieht in einer Reihe von Arbeitsgruppen – die **Areas** heißen – wie z. B.

- applications
- internet

³⁵Siehe zum Beispiel die URL

<http://www.cis.ohio-state.edu/hypertext/information/rfc.html>.

³⁶Eine detaillierte Beschreibung der Arbeitsweise der IETF findet man im RFC 3160 und RFC 4677.

³⁷Die WEB-Site des IAB ist unter der URL

<http://www.isi.edu/iab>

zu bewundern.

³⁸Die WEB-Site der IETF ist unter der URL

<http://www.ietf.org>

zu finden.

- operations and management
- routing
- security
- transport
- user service

und werden von den sogenannten *Area Directors* geleitet. Kommunikationsmedium sind Newsgroups sowie Meetings. Zur Arbeitsweise der Areas der IETF siehe insbesondere den RFC 3160.

- **Internet Engineering Steering Group** oder **IESG**

Die Mitglieder der IESG bestehen aus den Area Directors der Arbeitsgruppen der IETF und sind direkt für die Verabschiedung der RFCs – also der Internet-Standards – verantwortlich.

- **Internet Assigned Number Authority** oder **IANA**

Die IANA³⁹ übernimmt — in Koordination mit der US-Regierung — die Verwaltung und Vergabe über weltweit sämtliche Internet-Adressen und Internet-Domänen. Die eigentliche Arbeit macht aber eine andere Organisation, die ICANN⁴⁰, was für *Internet Corporation for Assigned Names and Numbers* steht.

1.8.6 Request for Comments

Wie bereits erwähnt, sind für die Entwicklung und Bewertung von Protokollen im Internet die *Request for Comments* das Maß aller Dinge. Bei den RFCs handelt es sich also um eine fortlaufend durchnummerierte Dokumentensammlung. Dem IAB obliegt die Verwaltung und Kategorisierung dieser Dokumente. Bei der Kategorisierung der RFCs unterscheidet man zwischen

- *Protocol state*
bezeichnet den Entwicklungsstand eines Protokolls innerhalb des Standardisierungsprozesses
- *Protocol status*
bewertet den Einsatz bzw. die Verwendung des Protokolls

Für den Protocol State gibt es folgende Stufen:

EXPERIMENTAL Die Entwicklung des Protokolls befindet sich in einem Anfangsstadium. Das Protokoll sollte von Produktivsystemen nicht verwendet werden, es sei denn diese sind am Entwicklungsprozeß beteiligt.

³⁹Siehe auch <http://www.iana.org>.

⁴⁰Siehe auch: <http://www.icann.org>.

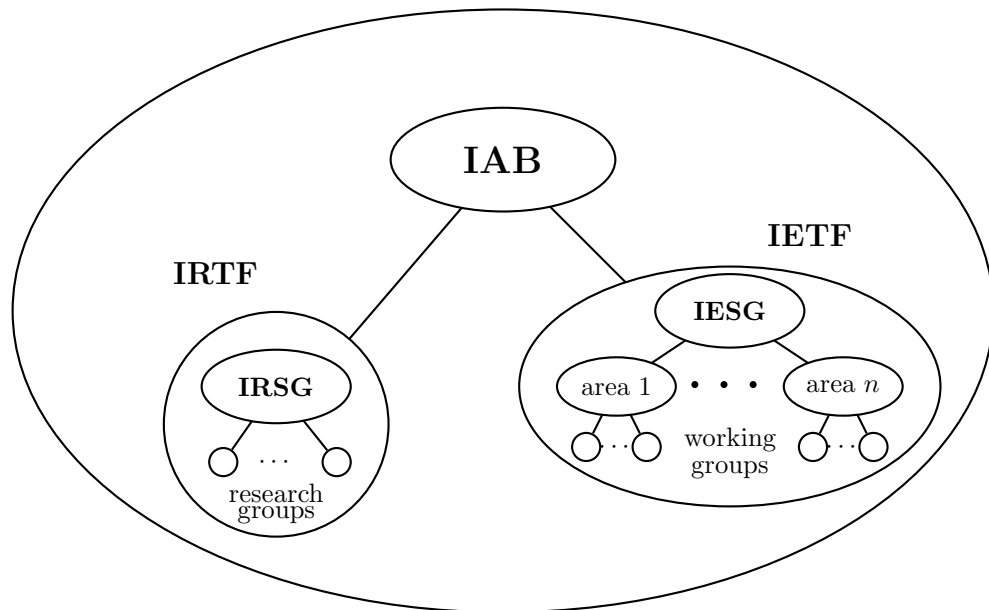


Abbildung 1.5: Die Struktur des IAB nach der Reorganisation im Jahre 1989.

PROPOSED STANDARD Das Protokoll liegt dem IAB zur Standardisierung vor. Es müssen noch Tests und Implementationen vorgenommen werden.

DRAFT STANDARD Das Protokoll steht kurz vor der Erhebung in den FULL-STANDARD-STATE. Das Protokoll wurde bereits ausgiebig getestet und es wurden eine Reihe von Implementierungen durchgeführt. In diesem Stadium sind noch Änderungen möglich.

STANDARD Das Protokoll ist vom IAB zu einem offiziellen Internet - Standard deklariert.

INFORMATIONAL Unter diesem Status werden Protokolle verschiedener Entwickler zusammengefaßt, von denen nicht beabsichtigt wird, sie den Status eines Standard-Protokolls zu erheben. Dennoch können solche Informationen interessant und wichtig für die Internet-Gemeinde sein.

HISTORIC Diese Kategorisierung umfaßt diejenigen RFCs, die technologisch überholt wurden.

Für den Protocol Status gibt es folgende Einteilungen:

REQUIRED Ein System ist zur Verwendung des Protokolls verpflichtet. Das heißt der Einsatz des Protokolls ist unbedingt erforderlich, damit das System funktioniert.

RECOMMENDED Der Einsatz des entsprechenden Protokolls ist zu empfehlen.

ELECTIVE Es besteht die grundsätzliche Möglichkeit, das Protokoll einzusetzen.

LIMITED USE Wegen seiner besonderen Eigenschaften oder aufgrund des aktuellen Entwicklungsstands des Protokolls ist der aktuelle Einsatz des Protokolls nur unter bestimmten Bedingungen zu empfehlen bzw. möglich.

NOT RECOMMENDED Wegen seiner besonderen Eigenschaften oder des aktuellen Entwicklungsstands des Protokolls ist der aktuelle Einsatz des Protokolls nicht zu empfehlen bzw. möglich.

Protokoll	Nummer	State	Status
IP – Internet Protocol	RFC 791	STANDARD	REQUIRED
ICMP – Internet Control Message Protocol	RFC 792	STANDARD	REQUIRED
TELNET – Terminal Emulation	RFC 854	STANDARD	RECOMMENDED
GGP – Gateway to Gateway Protocol		HISTORIC	NOT RECOMMENDED
ARP – Address Resolution Protocol	RFC 826	STANDARD	ELECTIVE

Tabelle 1.3: Stati verschiedener RFCs.

Innerhalb der RFC Dokumente gibt es weitere Unterteilungen in sogenannte **FYI** (*For Your Information*) und **BCP** (*Best Current Practice*) Dokumente. Die FYI Reihe von RFCs dient dazu, den Internet Nutzern eine zentrale Informationsquelle über unterschiedliche Schwerpunkte der Internettechnologie zur Verfügung zu stellen. Die Zielgruppe der FYI ist ein breites Publikum, das Spektrum reicht von Einsteigerthemen bis zu fortgeschrittenen Topics.

Die *Best Current Practice* (BCP) Dokumente – ebenfalls Teil der RFCs – haben wie die Standard RFCs technische Dokumentationen als Thematik. Die BCP dokumentieren jedoch Entwicklungen, die nicht von der IESG als Standard abgesegnet wurden. Die BCP Dokumente sind daher abgenommen, die darin enthaltenen Empfehlungen und Entwicklungen werden jedoch nicht in einen offiziellen Internetstandard erhoben.

1.8.7 Die Internet Society

Die Internet SOCIety (ISOC) ist eine professionelle Gemeinschaft von Mitgliedern, bestehend aus mehr als 150 Organisationen und über 16.000 Einzelpersonen in mehr als 180 Ländern der Erde. Die ISOC ist die führende Institution zu Diskussion von Fragen, welche die Zukunftsentwicklung des Internets betreffen. Sie stellt gleichzeitig den organisatorischen Dachverband für die sämtliche Gruppen dar, die die internet Standards entwickeln, also das IAB, die IETFs.



Seit 1992 bildet die ISOC die internationale Organisation zur globalen Koordination des Internets und der Kooperation der Länder. Die ISOC unterhält dazu ein breites Spektrum an unterschiedlichen Aktivitäten zur Entwicklung des Internets, dessen Verfügbarkeit und damit zusammenhängenden Technologien.

Die ISOC operiert nicht nur als globales Informationszentrum für Internetfragen oder Bildungsfragen, sondern auch als globaler Initiator für sämtliche Internetrelevanten Initiativen. Durch

- jährliche International Networking (INET) Konferenzen
- Training Workshops in Entwicklungsländern
- Tutorials
- Statistische und Markt Analysen
- Publikationen
- Öffentlichkeitsarbeit
- und, und, und,

versucht die ISOC die wachsenden Bedürfnisse der Internet Gemeinschaft zu erfüllen. Das Ziel ist die Verfügbarkeit und den Nutzen des Internets für Wirtschaft, Bildung und alle sozialen Bereiche zu erweitern.

1.8.8 Das WWW Consortium



Das **World Wide Web Consortium**⁴¹ (**W3C**) wurde im Oktober 1994 von **Tim Berners-Lee** am Massachusetts Institute of Technology in Zusammenarbeit mit dem CERN gegründet. Das W3C hat die Aufgabe, die technische Entwicklung der Web-Technologie zu bündeln, zu koordinieren und voranzutreiben.

Langfristige Ziele des W3C:

⁴¹Siehe die URL: <http://www.w3c.org>

- 1 Universeller Zugriff
- 2 Semantisches Web
- 3 Web of Trust

1.8.9 Das Computer Emergency Response Team

Das *Computer Emergency Response Team* (CERT)⁴² ist an der Carnegie Mellon University angesiedelt und wurde im Jahre 1988 von der DARPA/ARPA (Advanced Research Projects Agency) etabliert mit der Aufgabe, in Zusammenarbeit mit der Internetgemeinde Gefahren für die Sicherheit von Computersystemen zu erkennen, zu identifizieren, zu analysieren und adäquate Gegenmaßnahmen zu entwickeln. Speziell obliegen dem CERT folgende Aufgaben:

- Die Verfügungstellung einer umfassenden Beschreibung von Angriffsmethoden, Verwundbarkeiten und die Auswirkungen von Angriffen auf Informationssysteme und Netzwerke. Weiterhin sollen Informationen über das Auftreten von Attacken und Verwundbarkeiten von Systemen bereitgestellt werden.
- Der Aufbau einer Infrastruktur von Sicherheitsexperten mit wachsenden Kompetenzen, die effektiv auf Attacken gegen Internet-basierte Systeme reagieren sowie Maßnahmen entwickeln können, ihre Systeme gegen Angriffe aller Art zu schützen.
- Entwicklung von Methoden zur Evaluierung, Verbesserung und Erhaltung der Sicherheit und Überlebensfähigkeit von Netzwerksystemen.
- Zusammenarbeit mit Herstellern von Hard- und Software um die Sicherheit von Produkten zu erhöhen.

1.8.10 Das Software Engineering Institute (SEI)

Seit 1984 existiert das *Software-Engineering Institute* (SEI)⁴³ an der Carnegie-Mellon University, finanziert durch das DoD. (Department of Defense). Ziel dieser Organisation ist die Verbesserung des Software-Engineerings.

Im Jahre 1987 führte das SEI das sogenannte **Capability Maturity Model** (CMM) ein. Mit Hilfe von Fragebögen, Interviews und dem CMM wird ein Beurteilungskatalog eines Programmiererteams erstellt, der objektive Angaben darüber macht, wie weit das Team in der Lage ist, Software zu erstellen, die zuverlässig die Kundenanforderungen erfüllt.

Die Notenskala reicht von

1 → chaotisch

⁴²Siehe die URL: <http://www.cert.org>.

⁴³Siehe URL: <http://www.sei.cmu.edu>

bis

5 → vorbildlich

Bis 1994: etwa 250 Software Firmen untersucht und klassifiziert ([38])

Resultat:

- 75% der untersuchten Organisationen → Stufe 1
- 24% erreichten Stufe 2 oder 3
- genau 2 erreichten Stufe 5:
 - indisches Programmiererteam von Motorola in Bangalore
 - Loral⁴⁴ (früher IBM); Erstellung der Bordsoftware des Space Shuttles und Kommunikationssatelliten

Bis 1998: etwa 3500 Software Projekte untersucht und klassifiziert ([38])

Resultat:

- 24% erreichten Stufe 2
- 15 % erreichten Level 3
- 2% erreichten Level 4
- 0.6% erreichten Stufe 5

Beispiel: (aus [38])

Die Geräte-Abteilung der US-Firma Raytheon (Radar- und Flugleitsysteme); 1988 Initiative zur Verbesserung des Software-Engineering Prozesses, nachdem die Abteilung im CMM Test durchgefallen war

Innerhalb von 3 Jahren: Aufstieg zur Kategorie CMM 3 durch Verbesserung und Anwendung rigoroser Kontroll- und Testrichtlinien sowie Schulungen der 400 Programmierer.

Resultat dieser Bemühungen: Projekte waren *vor* den vereinbarten Terminen fertig und blieben unter den kalkulierten Kosten. Die Produktivität der Programmierer hat sich innerhalb von drei Jahren verdoppelt.

Im WEB sind einige Quellen, die fehlerhafte Software dokumentieren

- The RISK digest <http://catless.ncl.ac.uk/Risks>
- Computer-Related Incidents with Commercial Aircraft by Peter B. Ladkin
<http://www.rvs.uni-bielefeld.de/publications/Incidents/>

⁴⁴Siehe <http://www.loral.com>

- The Standish Group at <http://www.pm2go.com>
- The Data and Analysis Center for Software (Risk Management)
<http://www.dacs.dtic.mil>

Das CMM wurde zum CMMI weiterentwickelt, CMMI steht für **Capability Maturity Model Integration**.⁴⁵

⁴⁵Siehe die URL <http://www.sei.cmu.edu/cmmi>.

1.9 Computer und Computerklassen

Ähnlich wie bei Kraftfahrzeugen - hier unterscheidet man zwischen LKW, PKW, Bussen und anderen Fahrzeugen - haben sich in der EDV im Laufe der Zeit eine Reihe von Computerklassen entwickelt, die sich bezüglich ihrer Leistungsfähigkeit und ihre Kosten um Größenordnungen unterscheiden. Dabei sind die Übergänge mittlerweile fließend. Die vier Gruppen

- Mikro-Computer
- Mini-Computer
- Großrechner
- Super-Computer

stellen die klassische Einteilung dar. Im alltäglichen Sprachgebrauch werden die Mikrocomputer als 'PC' bezeichnet, die Minicomputer als Workstations. Jede Rechnerklasse weist spezifische Eigenschaften auf, die ihre Einsatzmöglichkeiten weitgehend festlegen.

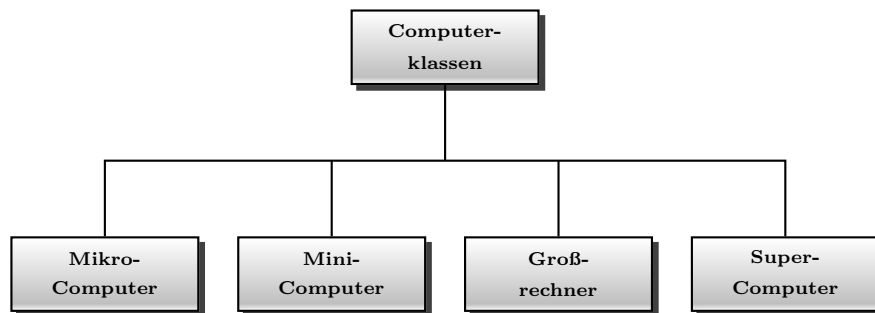


Abbildung 1.6: Computerklassen

1.9.1 Mikro-Computer

Die Klasse der Mikro-Computer entstand durch die mittlerweile legendäre Computer-Revolution Mitte der 70er Jahre und hatte ihren Ursprung im berühmten Silicon Valley. Siehe [35] für eine detaillierte Darstellung dieser Computer Revolution. Die Klasse der Mikro-Computer kann in mehrere Unterklassen unterteilt werden, wobei dies eigentlich nur noch historische Bedeutung hat, denn heutzutage werden sämtliche Aufgaben von den PC abgewickelt. Andererseits hat sich gerade in diesem Sektor in den letzten Jahren eine neue Klasse herauskristallisiert – das sind die Handhelds.

★ *Heim-Computer oder Home - Computer*

Home Computer sind besonders preiswerte, ausschließlich für den privaten Gebrauch konzipierte Minicomputer, die in erster Linie den Spieltrieb des Besitzers befriedigen. Der bekannteste Vertreter dieser Klasse ist der C64 von Commodore, der in den 80er Jahren ein absoluter Verkaufshit war. Mittlerweile sind diese Homecomputer so gut wie vom Markt verschwunden⁴⁶, da die heutigen Standard PC deren Aufgaben übernommen haben.

★ *Handhelds*

Den unteren Bereich im Leistungssektor der Microcomputer bildet heutzutage die Gruppe der Handhelds. Darunter versteht man Computer von der Größe einer Handfläche. Dazu zählen:

- Mini-Notebooks
- Handhelds
- Palmtops
- Organizer
- Personal Digital Assistant (PDA)
- Smartphones

Akkus oder Batterien dienen dieser Spezies als Energiequelle. In der Regel haben Mini-Notebooks ein kleines monochromes oder farbiges Display und eine Nottastatur, über der man geringe Datenmengen eingeben kann.

Verschiedene Modelle dieser Klasse verzichten auf Tastaturen und erlauben die Eingabe der Daten sowie die Steuerung des Computers über einen Stift (Pen). Die Schreib- und Zeichenfläche dieses 'Schreibblocks' ist der Bildschirm

★ *IBM-kompatible Personal-Computer, die PC*

Die heutzutage stärkste Umsatzgruppe unter den Mikrocomputern bilden die IBM kompatiblen Personalcomputer oder auch die eigentlichen PC. Der Ur - PC wurde 1981 von der IBM auf den Markt gebracht, mit der klaren Zielvorgabe, neben dem Mainframemarkt auch den Kleincomputermarkt für IBM zu erobern und dort eine marktführende Position zu erlangen (siehe z.B. [58]). Das IBM Konzept setzte sich mit einem nicht vorhersehbaren Erfolg durch. Da wir später detaillierter auf die Entwicklung dieses Microcomputertyps eingehen, wollen wir an dieser Stelle die IBM - kompatiblen PC nicht weiter diskutieren.

Wichtig ist hier nochmals, wenn von PC die Rede ist, ist im strengen Sinn die Klasse der IBM-kompatiblen Microrechner gemeint.

⁴⁶Auf Flohmärkten findet man noch Exemplare dieser Gattung.

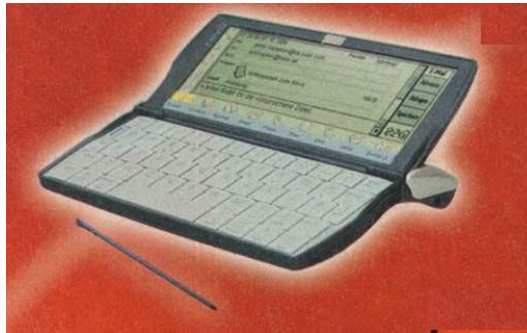


Abbildung 1.7: Ein elektronischer Organizer.

★ *Macintosh von Apple*

Parallel zu IBM - PC - Welt entwickelte sich die Apple - Welt. Die Computer der Firma Apple basieren auf dem Prozessortyp 680x0 (68000, 68020, 68030, usw.) des Chip - Herstellers Motorola, die in ihren Leistungsmerkmalen den entsprechenden INTEL - Prozessoren in den IBM - kompatiblen PC ähneln. Während beim IBM PC im Laufe der Zeit jede Menge Firmen versuchten, mit ihren ureigensten Entwicklungen und Konzeptionen als Standard in die PC - Welt einzugehen, verfolgte Apple von Anfang an das klare Konzept, alle Entwicklungen, die die Apple Rechner betrafen, unter eigener Regie zu leiten. In erster Linie aufgrund der herausragenden Benutzerfreundlichkeit, der Ergonomie der Hardware und der perfekten Feinabstimmung zwischen Hardware und Software waren diese Computer in den 80er Jahren sehr erfolgreich.

Durch dieses Konzept hatte sich Apple schnell einen Namen gemacht, der für Qualität und innovative Konzepte bürgte⁴⁷. Apples Strategie lag in klaren, engen Vorgaben für Programmentwickler, was die Bedienung und Benutzerführung betraf, wenn die Programme auf Apple - Rechner Verbreitung finden sollten. Dies hatte für den Anwender die erfreuliche Auswirkung, daß alle Anwendungsprogramme und das Betriebssystem selbst, nach einem einheitlichen Schema aufgebaut waren (und immer noch sind). Als Folge dessen fand sich ein Anwender in einem neuen Programm sehr schnell zurecht, d.h. das von Apple propagierte Konzept der Einheitlichkeit verkürzt enorm die Einarbeitungszeit in neue Programme. Ähnliches setzte sich in der IBM - PC - Welt erst sehr viel später durch.

⁴⁷Graphische - und bunte - Benutzeroberflächen, wie sie IBM - PC - Usern erst seit WINDOWS Ende der achtziger, bzw. Anfang der neunziger Jahre bekannt sind, gab es bei Apple schon lange zuvor. Mausunterstützung war ebenfalls schon lange vor dem WINDOWS Alltag bei den MACs vorhanden.

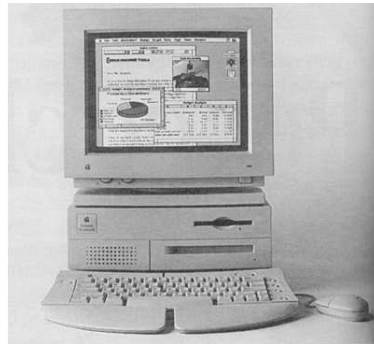


Abbildung 1.8: Der Apple Macintosh Quadra 650.

Apple versäumte es jedoch, aus diesem Vorteil gegenüber den IBM PC auch Kapital zu schlagen⁴⁸, heutzutage fristen Apple Rechner eine Art Nischendasein. Diese sind hauptsächlich im Desktop - Publishing Bereich (z.B. Werbebranche und Grafikbearbeitung) sehr verbreitet. Mittlerweile bahnt sich auch eine Wende dahingehend an, daß sich die abgeschottete Apple - Welt der IBM PC - Welt in Form des PowerPC nähert.

★ *Amiga von Commodore, Atari*

Neben Apple haben auch andere Computerhersteller basierend auf den 680x0 Motorola Prozessoren versucht, ihre eigenen Entwicklungen auf dem Markt zu etablieren und damit hauptsächlich den semiprofessionellen Markt bedient. Ende der achtziger Jahre hatten beispielsweise die Amiga - Rechner von Commodore und die Atari - Rechner (siehe z.B. Abbildungen 1.9,1.10) einigen Markterfolg - dies hauptsächlich bei nicht so zahlungskräftigen Mitmenschen wie Studenten. In der Wirtschaft konnten diese Hersteller aber nie richtig Fuß fassen.

1.9.2 Mini-Computer bzw. Midrange Rechner

Obwohl sich mittlerweile die Grenzen zwischen den leistungsstarken PC (z.B. Pentium Rechner) und den einfachen Workstations überlappen, liegen die meisten Minicomputer in ihren Leistungsmerkmalen um eine bis mehrere Größenordnung(en) höher als die der PC. Sehr häufig basieren Workstations auf der RISC-Prozessortechnologie (siehe z.B. [9,25,90]). Üblicherweise werden auf den Mini-Computern UNIX-Betriebssysteme eingesetzt oder — wie beispielsweise bei der AS/400, diese ist (unter anderem) mit dem Betriebssystem OS/400

⁴⁸Bill Gates war eben auf der Microsoft - IBM - INTEL Schiene.



Abbildung 1.9: Der Atari TT.



Abbildung 1.10: Ein weiteres Atari Modell.

ausgerüstet — proprietäre Systeme.⁴⁹ Diese Computerklasse nennt man gelegentlich auch *Midrange-Rechner*

Workstations (siehe z.B. Abbildung [1.11]) werden in der Regel im professionellen Bereich eingesetzt. Die eingesetzten Programme sind sehr teuer, komplex und für spezielle Aufgaben konzipiert. Die Anwender sind häufig Experten auf dem Gebiet, das mit den eingesetzten Programmen bearbeitet wird – man denke an einen technischen Zeichner, der mit einem CAD (Computer Aided Design) – Programm arbeitet. Typische Einsatzgebiete für Workstations sind:

- CAD (Computer Aided Design)

⁴⁹Proprietäre Systeme bezeichnen in der IT herstellerspezifische Produkte.

- Geographische Informationssysteme,
- Bildverarbeitungssysteme,
- naturwissenschaftliche Programme, Simulationen,
- leistungsfähige Datenbankserver,
-

Die meisten Peripheriegeräte von Workstations unterscheiden sich nur marginal von denen für PC. In der Regel sind diese leistungsfähiger (und teurer). Wichtige Rechnerfamilien der Klasse der Mini-Computer sind:

- ◇ die AS/400 von IBM
- ◇ VAXstations der Firma DEC
- ◇ DEC5000 Serie von DEC
- ◇ Alpha von DEC
- ◇ SPARC Station von SUN
- ◇ RS/6000 von IBM



Abbildung 1.11: Eine SUN SparcStation.

Ein sehr verbreitetes Midrange-System ist die AS/400 von IBM, heute unter der Bezeichnung **IBM System i** auf dem Markt. Die AS/400 hat ein proprietäre Betriebssystem mit dem Namen i5/OS und eine eigene, proprietäre Datenbank mit dem Namen DB2. Typischerweise werden auf solchen Systemen kaufmännische Anwendungen zur Abwicklung von Geschäftsprozessen betrieben, die als Server oder Client/Serveranwendung konfiguriert werden. Die System i Reihe ist *skalierbar*, *i.e.* solche Systeme können als kleine Maschinen (für etwa 10 Benutzer) betrieben werden, aber auch als Systeme, die mit den

Mainframe-Bereich überlappen mit bis zu tausenden Benutzer. Dementsprechend sind die leistungsfähigsten Versionen der System i Rechner mit bis zu 64 Power5+ Prozessoren ausrüstbar, 5 TeraByte Hauptspeicher, und es können bis zu 380 TeraByte Plattenspeicher verwaltet werden.

Die System i Reihe unterstützt nicht nur das proprietäre OS/400 bzw. i5/OS Betriebssystem, sondern auch das IBM-eigene UNIX-Derivat AIX und über zusätzliche Steckkarten kann auch Linux oder Windows betrieben werden. Diese Steckkarten stellen einen eigenen PC dar mit CPU und Arbeitsspeicher, der den Massenspeicher des Systems i verwendet. Je nach Modell können bis zu 60 solcher Steckkarten eingebaut werden. Somit lassen sich Dutzende von Windows- oder Linuxservern auf einem einzigen System i zusammenfassen, was die Administration solcher Systeme enorm erleichtert. Typische, unter OS/400 lauffähige Anwendungen einer AS/400 sind:

- Immobilienwirtschaftliche Software
- Finanzbuchhaltung
- ERP-Systeme verschiedener Hersteller (SAP R/3, Oracle)
- Lagerverwaltungssoftware
- Dokumentenmanagementsysteme
- Betriebsdatenerfassungssysteme (BDE-Systeme)
- J.D. Edwards (ERP-System, heute ORACLE)
- PPS-Systeme
- Apache WEB-Server

IBMs RS/6000 Familie gibt es mit folgender Ausstattung:

Hardware: Vier 64-Bit POWER3-II Prozessoren 375 MHz, 64/32 KByte L1 Cache, 8 MB Level 2Cache, bis 16 GByte RAM, maximal 109 GByte Festplatten, fünf PCI Schnittstellen, 10/100 MBit Ethernet, Gigbit Ethernet, FDDI, PCI Cryptographic Coprozessor

Software: AIX

1.9.3 Großrechner

wie das System/390 von IBM oder BS2000 von Siemens

Großrechner werden auch Hosts oder Mainframes genannt und finden ihren Einsatz in Rechenzentren oder Großbetrieben, kurz, überall dort, wo es auf hohe

Rechenleistung ankommt und sehr große Mengen von Datenbeständen verwaltet werden müssen. Großrechner gehören zu den komplexesten (und kompliziertesten) Maschinen, die es gibt. Die Wurzeln der Großrechner reichen zurück bis in die Anfänge der EDV - also die 50er und 60er Jahre.



Abbildung 1.12: IBMs Mainframe System/390.

Diese Rechnerart ist derart konzipiert, daß bis zu 1.000 Benutzer und mehr gleichzeitig bedient werden können. Dabei ist es nicht unüblich, daß Benutzer am Datenendgerät (das sogenannte Terminal) und Mainframe weit voneinander entfernt sind und mittels Datenfernübertragung (Telefonnetz) miteinander kommunizieren.

Damit sich die teuren Großrechner nicht mit den Kleinarbeiten wie Tastaturabfragen und ähnlichen lästigen Dingen beschäftigen müssen, sind die Terminals (damit ist das Datenendgerät für den User gemeint) mit ausreichender Eigenintelligenz ausgestattet, um diese Dinge selbst zu tun.

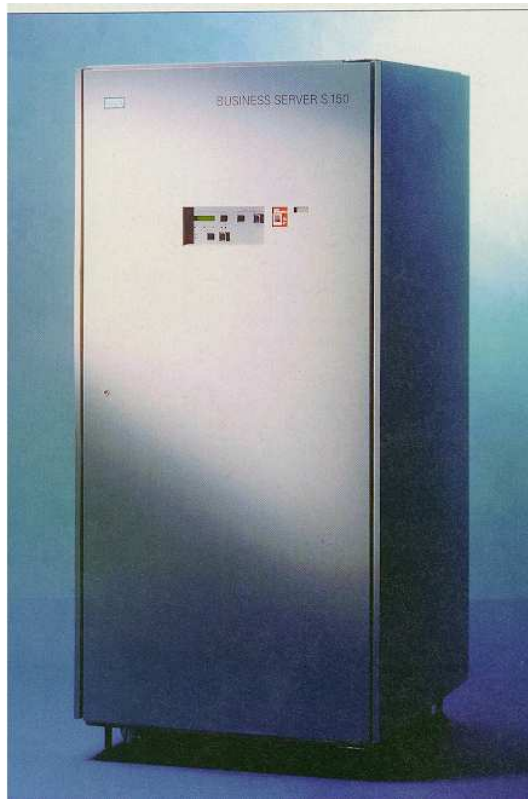


Abbildung 1.13: Der Business Server S150 von Siemens.

Großrechner werden vorwiegend in Banken, Versicherungen, Industriebetrieben und Einrichtungen der öffentlichen Hand betrieben.

Obwohl den Großrechnern vor einigen Jahren das Aussterben nachgesagt wurde, erleben die Mainframes am Ende der 90er Jahre ein unerwartetes Comeback [36, 48]. Gründe für eine Renaissance der Mainframes gibt es reichlich, einer ist zum Beispiel darin zu sehen, daß die früher proprietäre Betriebssystemkonzeption aufgegeben wurde und mittlerweile das offene System propagiert wird. Neue Mainframes können in einer heterogenen UNIX-WINDOWS NT Systemlandschaft integriert werden. Weiterhin stellen Mainframes aufgrund der (in EDV - Maßstäben gemessenen) langen Historie ein sehr ausgereiftes System dar, das sämtliche Neuentwicklungen der IT - Branche unterstützt. So können heute Mainframes als WEB Server genutzt werden, oder als SAP R/3 Datenbankserver.

Hersteller von Großrechneranalgen sind zum Beispiel IBM, Amdahl, Honeywell und Bull, NCR, UNIVAC und Siemens.

Anmerkung:

Vor einigen Jahren hat IBM die Angebotspalette in den verschiedenen Rechnerklassen umgetauft. Heute findet man:

- 1 zSeries \longrightarrow ehemals S/390 Mainframe,
- 2 iSeries \longrightarrow ehemals AS/400 Midrange,
- 3 pSeries \longrightarrow ehemals RS/6000 Workstation,
- 4 xSeries \longrightarrow Intelbasierte Server.

Im Jahre 2006 fand ein weiteres Rebranding bei IBM statt, die heute (2011) gültige Bezeichnungen sind

- S/390 \longrightarrow zSeries \longrightarrow System z,
- AS/400 \longrightarrow iSeries \longrightarrow System i,
- RS/6000 \longrightarrow pSeries \longrightarrow System p,
- System x, Intelbasierte Server.

1.9.4 Supercomputer

Diese Computerart (siehe zum Beispiel Abbildung [1.14]) findet seinen Einsatz hauptsächlich in Forschungszentren. Bezeichnenderweise heißen diese Rechner im EDV-Jargon auch *Numbercruncher*, soviel wie Zahlenfresser. Sie werden zum Beispiel für die Simulation des Weltklimas eingesetzt (siehe [95]), für Wettervorhersagen oder für Berechnungen in der Astrophysik, Kern- und der theoretischen Hochenergiephysik (siehe auch [98, 101] oder [6]). In der chemischen Industrie findet dieser Rechnertyp Einsatz bei der Simulation von Molekülen. Nicht zuletzt versucht man nach Unterzeichnung des Verbots für über- und unterirdische Atomwaffentests durch Bill Clinton im Jahre 1996 solche Explosionen in Computern zu simulieren [62] (siehe auch [49]).

Eine Reihe wichtiger gesellschaftspolitischer, wissenschaftlicher und wirtschaftlicher Fragestellungen lassen sich nur mit Hilfe dieser Rechner lösen, zum Beispiel:

- Weltklima- und lokale Wettersimulationen in der Meteorologie,
- Berechnungen in der kontrollierten Kernfusion,
- Auswertung sehr umfangreicher Datenmengen und Statistiken zur Simulation volkswirtschaftlicher Modelle,
- Modellierung von Proteinmolekülen in der Pharmaindustrie,
- Echtzeitsimulationen (Crash-Simulation) und Aerodynamik in der Automobilindustrie,

- Materialforschung in der Luftfahrtindustrie,
- Entwicklungen in der Nanotechnologie,
- Berechnung der Dynamik von Galaxien in der Astrophysik.

Das Maß aller Dinge im Bereich der Super-Computer ist die Anzahl der Gleitkommaoperationen — z.B. Additionen oder Multiplikationen — die der Rechner pro Sekunde bewältigen kann. Diese Einheit nennt man **Flops** für *floating point operations per second*. Gegenwärtig haben Superrechner eine Leistungsfähigkeit im Bereich der Tera- bis Petaflops (das sind 10^{12} bis 10^{15} Gleitkommaoperationen pro Sekunde).

Bei den Hochleistungsrechnern unterscheidet man zwischen *Vektorrechnern*, die nach dem sogenannten Pipeline-Prinzip arbeiten (CRAY 1, CRAY C 90, Cyber 205, Siemens/Fujitsu) und den eigentlichen Parallelrechnern wie *Mehrprozessorsystemen* (CRAY - X/Y-MP, CRAY 2) mit einigen wenigen Einzelprozessoren sowie massiv parallelen *Multiprozessorsystemen* mit vielen hundert oder tausend Prozessorelementen (N-Cube, Connection Machine CM-5, CRAY T3D). In letzterem Fall spricht man auch von *massiv parallelen Systemen*.



Abbildung 1.14: Der (einstmalige) Supercomputer CRAY - 1, heute ein Relikt im Deutschen Museum in München, das zum Sitzen einlädt. Abgebildet ist ein Modell S / 1000.

Das Rechenzentrum der Universität Mannheim stellt halbjährlich einen High-Score der 500 schnellsten Supercomputer zusammen, die man sich unter der URL <http://www.top500.org> ansehen kann.

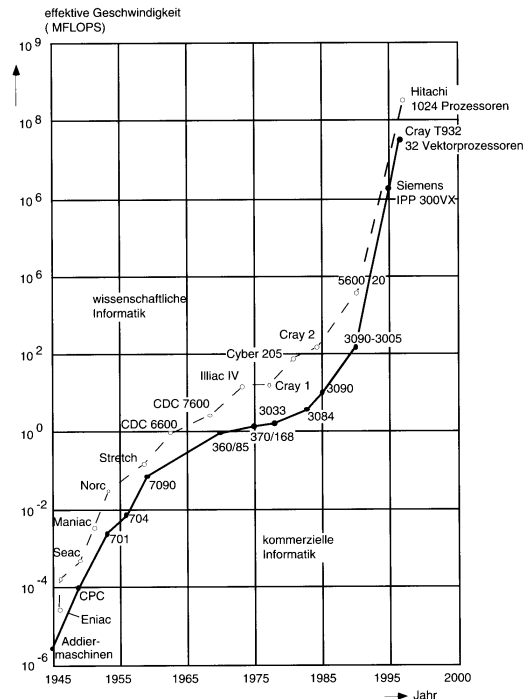


Abbildung 1.15: Entwicklung der Rechenleistung bei Hochleistungsrechnern.

Hersteller von Supercomputern sind die Firmen IBM, Intel, Thinking Machines, Cray, Hitachi, Fujitsu, Sun. Die Abbildung [1.16] zeigt den Jugene Rechner des Jülicher Supercomputer Centers im Jahre 2011. Dieser Rechner vom Type Blue Gene von IBM schaffte es im Ranking Juni 2011 auf Platz 12 der TOP 500 Liste.

In den letzten Jahren werden verstärkt **Rechner-Cluster** eingesetzt ([56] bis [47]), die in ihrer Leistungsfähigkeit an die Superrechner heranreichen. Computer Cluster sind aus einfachen Standard Personal Computern und/oder Workstations aufgebaut und über ein spezielles Hochgeschwindigkeitsnetz verbunden. Eine spezielle Software sorgt für die Verteilung der Rechenarbeit auf die verschiedenen Knoten des Clusters. Solche Systeme sind sehr leistungsfähig, sehr preiswert gegenüber Superrechnern, skalierbar, d.h. bei Bedarf können weitere PC einfach in den Cluster eingebunden werden, und im Vergleich zu Superrechnern einfach zu konfigurieren. Es wurden schon Cluster mit mehreren hundert Rechnern konfiguriert ([110]).

Platz	Hersteller	Computer	Rmax	Standort	Land	Jahr	Anzahl CPUs
1	IBM	ASCI White SP Power 3 375 MHz	4.938	Lawrence Livermore National Laboratory	USA	2000	8.192
2	Intel	ASCI Red	2.379	Sandia National Labs, Albuquerque	USA	1999	9.632
3	IBM	ASCI Blue-Pacific SST, IBM SP 604e	2.144	Lawrence Livermore National Laboratory	USA	1999	5.808
4	SGI	ASCI Blue Mountain	1.608	Los Alamos National Laboratory, Los Alamos	USA	1998	6.144
5	IBM	SP Power 3 375 MHz	1.417	Naval Oceanographic Office Bay Saint Louis	USA	2000	1.336
6	IBM	SP Power 3 375 MHz	1.179	National Centers for Environmental Prediction Camp Spring	USA	2000	1.104
7	Hitachi	SR8000-F1/112	1.035	Leibniz Rechenzentrum München	Ger.	2000	112
8	IBM	SP Power 3 375 MHz 8 way	929	UCSD/San Diego Supercomputer Center, San Diego	USA	2000	1.152
9	Hitachi	SR8000-F1/100	917	High Energy Accelerator Research Organization KEK Tsukuba	Japan	2000	100
10	Cray Inc.	T3E1200	892	Government US Army HPC North Carolina	USA	1998	1084

Tabelle 1.4: Die ersten TOP500 Supercomputer vom 3. November 2000. (Quelle: <http://www.top500.org>) Rmax bezeichnet die theoretisch höchstmögliche Performance in GigaFlops.



Abbildung 1.16: Jugene, der leistungsfähigste Rechner der Jülich Supercomputing Centre. Dieses System enthält rund 72000 Prozessoren, die in 72 wassergekühlten Schränken untergebracht sind. Die Speicherkapazität der eingesetzten Massendatenspeicher können 7 Petabyte (10^{15} Byte) ausnehmen, diese sind wiederum an einen Magnetbandspeicher mit 30 Petabyte Kapazität angeschlossen [6].

Kapitel 2

Die von Neumann Architektur

Hinter dem Begriff der **Rechnerarchitektur** verbirgt sich die Beschreibung der technischen Realisierung einer Datenverarbeitungsanlage. Dies involviert insbesondere

- die Gesamtheit der Bauprinzipien einer Rechenanlage. Dazu gehören:
 - ✗ die Festlegung der internen Darstellung der Daten
 - ✗ die Festlegung der auf den Daten ablaufenden Operationen, das heißt, die Festlegung des Umfangs der Maschinensprache der CPU.
 - ✗ der Aufbau der Maschinenbefehle, insbesondere das Format der Instruktionen
 - ✗ die Definition von Schnittstellen zwischen den Funktionseinheiten und zu externen Geräten
 - ✗ der 'Bauplan' nach dem die einzelnen Funktionseinheiten zu einem Rechner zusammengefügt werden.
- zu beachten sind dabei die *grundlegenden* Aufgaben einer Rechenanlage: die Sammlung, Speicherung, Verarbeitung und Darstellung von Daten.

2.1 Aufbau des von-Neumann Rechners

Die meisten der heute in Gebrauch befindlichen Computer — diese realisieren das Eingabe Verarbeitung Ausgabe – Prinzip — sind nach einem Konzept aufgebaut, das der Mathematiker JOHN VON NEUMANN in den vierziger Jahren des letzten Jahrhunderts entwickelt hat [5, 17, 30, 72, 107, 108]. Dieses Bauprinzip ist (nach EDV Maßstäben) uralt und heißt

von Neumann Architektur.

Ziel VON NEUMANNs war dabei die Entwicklung *genereller* Konzepte, wie ein Rechner unabhängig von der Art der zu lösenden Aufgabe (universell) aufgebaut sein sollte. Diese generellen Konzepte nehmen daher auch keinerlei Bezug auf irgendeine technische Realisierung. VON NEUMANN konstruierte in den Jahren 1944 bis 1952 zusammen mit ARTHUR BURKS und HERMAN GOLDSTINE den EDVAC Rechner, in dem diese Konzepte erstmals realisiert waren. Die entscheidende Neuerung dieser Maschine war unter anderem die binäre Codierung der Programmbefehle und der Daten in dem *gleichen* Speicher.

Wie historische Betrachtungen gezeigt haben, ist JOHN VON NEUMANN nicht alleiniger Urheber der nach ihm benannten Rechnerarchitektur. Die grundlegenden Erkenntnisse, die zu diesem Bauprinzip geführt haben, resultieren aus Arbeiten von JOHN PRESPEER ECKERT und JOHN MAUCHLY und deren Arbeitsgruppe an der Moore School in den Jahren 1943 bis 1945 bei der Konstruktion des ENIAC Rechners.

Für die VON NEUMANN Architektur gibt es keine präzise Definition¹. Daher ordnet man heute einen Rechner der VON-NEUMANN Architektur zu, wenn dieser bestimmte Charakteristiken erfüllt. Im Laufe der Zeit wurde die ursprüngliche VON NEUMANN – Konzeption erweitert zum sogenannten *klassischen Konzept eines Universalrechners*. Diese Konzept umfaßt im Detail folgende Punkte ([89]):²

- Der Rechenautomat wird logisch und physisch in fünf Funktionseinheiten gegliedert (siehe Abbildung [2.1]):

1 Rechenwerk

Hier werden die arithmetischen Operationen (Addition Subtraktion, Multiplikation und Division) und die logischen Operationen (UND, ODER, NICHT) ausgeführt.

2 Leitwerk oder Steuerwerk

Das Steuerwerk veranlaßt den Transfer der Instruktionen aus dem Hauptspeicher in den Prozessor, interpretiert (decodiert) die darin codierten Anweisungen und initialisiert über Steuersignale die zur Ausführung der Instruktion notwendigen Komponenten des Rechners.³

3 Speicherwerk oder Hauptspeicher

In diesem Bauteil des Rechners werden die Programme und die Daten

¹VON NEUMANNs Untersuchungen sind de facto nie über den Status eines *Drafts* hinausgekommen, wie auch schon der Titel der Arbeit verrät [107].

²Siehe H. HEROLD *et al.*, [52], Kap. 5.2, oder H. MALZ, [74], Kap. 4.1.

³Man nennt den Ablauf der Instruktionsverarbeitung den *Fetch-Decode-Execute Zyklus*, oder kurz, FDE-Zyklus.

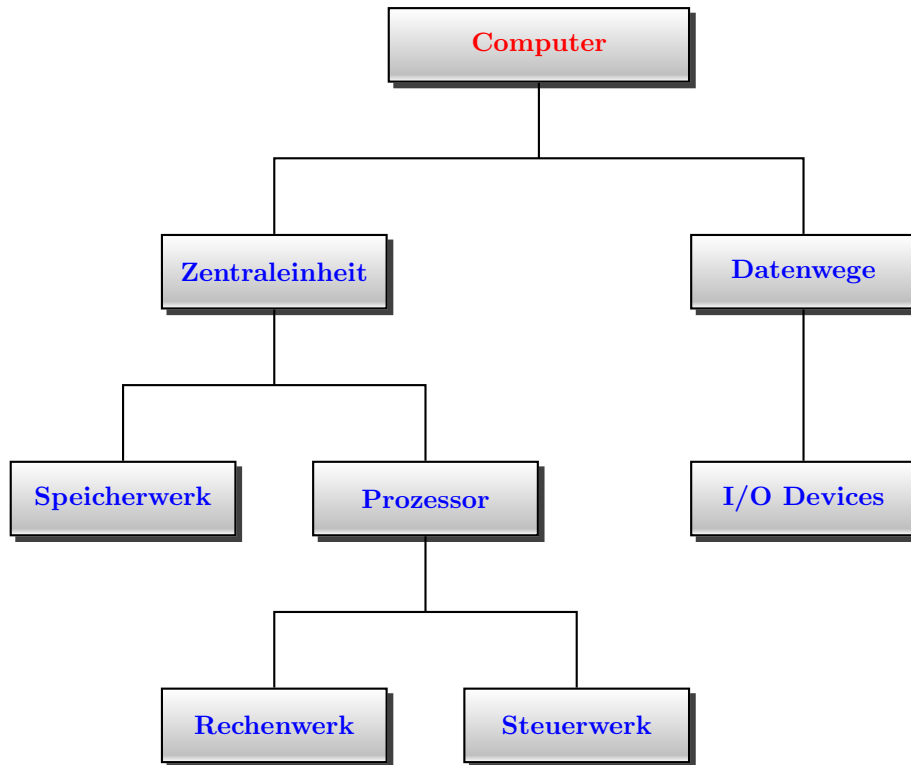


Abbildung 2.1: Die VON NEUMANN Architektur eines Rechners.

abgespeichert. Heutzutage nennt man das Speicherwerk den *Hauptspeicher* oder *Arbeitsspeicher*.

4 Eingabewerk

Daten und Programme werden über die Funktionseinheit *Eingabewerk* in die Rechanlage eingegeben und laufen über **Datenwege** in die Speichereinheit.

5 Ausgabewerk

Diese Einheit dient der Ausgabe von Ergebnissen und/oder Daten an den Rest der Welt. Die von der Zentraleinheit berechneten Ergebnisse werden über **Datenwege** an die Ausgabeeinheiten transferiert.

Das Rechenwerk und das Steuerwerk zusammen nennt man auch den **Prozessor**, der Prozessor und der Hauptspeicher bilden die sogenannte

Zentraleinheit. Ein- und Ausgabewerke heißen heute **Peripheriegeräte** oder **I/O Devices**.⁴ Die Datenwege sind in heutigen Rechnern in Form unterschiedlicher **Bussysteme** realisiert.

- Der Rechenautomat ist in seiner Struktur unabhängig von den zu bearbeitenden Problemen. Daher wird solch ein Rechner auch **Universalrechner** genannt. Für jedes Problem gibt es eine spezielle Bearbeitungsvorschrift, die über Tastatur, Lochstreifen oder ein anderes Eingabegerät eingegeben und im Speicher abgelegt wird. Diese Bearbeitungsvorschrift nennt man **Programm**. Daher nennt man einen VON-NEUMANN Rechner auch *speicherprogrammierbare Rechanlage*
- Zur Ablage von Programmen und Daten dient derselbe Speicher.
- Programme und Daten werden binär gespeichert.
- Der Speicher ist in gleich große Einheiten — den *Speicherzellen* — aufgeteilt. Auf jede Speicherzelle kann über ihre **Adresse** direkt zugegriffen werden.
- Aufeinanderfolgende Instruktionen eines Programmes werden im Speicher in der Regel in aufeinanderfolgenden Speicherzellen abgelegt. Durch die Erhöhung der Befehlsadresse um eine Einheit wird dadurch der nächste Befehl angesprochen. Eine Ausnahme bilden die **Sprungbefehle**.
- Es gibt **Sprungbefehle**. Hierbei wird nach der Ausführung eines Befehls mit Adresse x nicht automatisch der in der nächsten Speicherzelle abgelegte Befehl ausgeführt, sondern ein Befehl, der sich an der Stelle $y, y \neq x + 1$ befindet.
- Eine besondere Form des Sprungbefehls ist der **bedingte Sprungbefehl** (engl.: *conditional jump*). Dabei wird eine Befehlsadresse an eine Bedingung geknüpft. In Pseudocode kann dies folgendermaßen formuliert werden:

```
IF Bedingung erfüllt
  THEN nächste Befehlsadresse =  $y (\neq x + 1)$ 
  ELSE nächste Befehlsadresse =  $x + 1$ 
END
```

Anmerkungen:

- 1 In heutigen Hochsprachen werden bedingte Sprungbefehle in Form sogenannter **Kontrollstrukturen** wie beispielsweise **if-else**, **switch** Anweisungen oder Schleifen verwendet. Unconditional jumps — also Sprünge ohne Auswertung einer Bedingung — entsprechen Funktions- bzw. Methodenaufrufe. Über diese Kontrollstrukturen wird der Ablauf eines Programms gesteuert.

⁴I/O Device steht für Input/Output Device.

2 Man hat Ende der 1960er Jahre gezeigt, dass mit den drei Kontrollstrukturen

- Sequenz,
- Verzweigung,
- und Iteration

alle Algorithmen beschrieben werden können. Daher findet man in allen Programmiersprachen diese Kontrollstrukturen.

3 Das Rechenwerk nennt man heute **Arithmetic Logical Unit**, kurz ALU.

Die bedeutenste Innovation der VON-NEUMANN Architektur zur damaligen Zeit war die Idee, bei der Programmausführung das Programm und die notwendigen Daten in den gleichen Speicher zu laden und dann auszuführen. Zuvor wurden Instruktionen via Lochkarten direkt eingelesen und sofort abgearbeitet, was voraussetzt, dass die Hardware auch entsprechend geschaltet ist. Dies Innovation hat daher folgende Konsequenzen:

- Die Hardware muß bei Änderung einer Aufgabe, die zu lösen ist, nicht neu geschaltet werden.
- Die Speicherprogrammierung erlaubt die Verwendung von Jump-Befehlen, und zwar sowohl auf vorhergehende als auch nachfolgende Instruktionssequenzen.
- Damit kann auch der Programmcode während des Programmablaufs modifiziert werden.⁵

Durch diese Architektur kann der Rechner daher selbständig (natürlich gemäß den Programminstruktionen) logische Entscheidungen treffen. Damit ist der wichtige Schritt von dem starren Programmablauf zur flexiblen Programmsteuerung vollzogen, oder anders ausgedrückt, der Schritt von einer reinen *Rechenanlage* zu einer *Datenverarbeitungsanlage*.

Die VON-NEUMANN Architektur hat aber auch Nachteile:

- ⊕ Im Speicher sind Daten und Instruktionen anhand des gespeicherten Bitmusters nicht unterscheidbar.
- ⊕ Konstante und variable Daten können im Speicher nicht unterschieden werden.

⁵Da dies eine sehr riskante und fehlerträchtige Methode ist, sollte dies tunlichst vermieden werden.

- ⚡ Bei falscher Adressierung können Speicherinhalte verändert werden, die nicht verändert werden dürfen, wie beispielsweise Instruktionen oder Konstanten. Die Änderung eines einzelnen Bits in einer Maschineninstruktion erzeugt u. U. einen völlig anderen Befehl.

2.2 Der Prozessor

Der **Prozessor**⁶ ist *die* zentrale Komponente des Rechnersystems. Im Prozessor findet — koordiniert durch das **Steuerwerk** — die eigentliche Verarbeitung der Daten statt. Es ist die Aufgabe des Steuerwerks, die Maschineninstruktionen aus dem Hauptspeicher in den Prozessor zu laden. Die Abarbeitung der Instruktionen nennt man **Fetch–Decode–Execute Zyklus**. Die Ausführungseinheit für die arithmetischen Rechenoperationen sowie logische Operationen ist das **Rechenwerk** oder **arithmetisch – logische Einheit** (ALU).

Um die aus dem Hauptspeicher gelesenen Daten und Instruktionen im Prozessor zwischenspeichern zu können, verfügt jeder Prozessor über einen Satz sogenannter **Registerspeicher**. Es gibt Allzweckregister sowie Register, die für die Ausführung spezieller Aufgaben reserviert sind, wie das Befehlszählerregister (Instruction Counter), welches die Speicheradresse von auszuführenden Instruktionen enthält. Diese Register ermöglichen sehr schnelle Lese- und Schreibzugriffe, können aber lediglich einige wenige Bits speichern. Unter anderem ist diese Speichergröße — man spricht auch von **Registerbreite** — ein Maß für die Verarbeitungsgeschwindigkeit eines Prozessors. Typische Werte sind 8 Bit, 16 Bit (z.B. INTEL 80286), 32 Bit (z.B. INTEL Pentium), 64 Bit (IBM PowerPC) und 128 Bit (Playstation III mit CELL-Prozessor). Dementsprechend hat man einen 8 Bit, 16 Bit, 32 Bit, 64 Bit oder 128 Bit Prozessor vorliegen.

Anmerkung:

Die Größe des Befehlszähleregisters legt — zusammen mit der Breite des Adressbusses — fest, wie groß der adressierbare Hauptspeicher ist.⁷ Wenn eine Plattform über 16 Bit breite Adressregister verfügt, können $2^{16} = 65.536$ Speicherplätze angesprochen werden, das entspricht einer Speichergröße von 65 KBytes. Eine 32-Bit Plattform kommt da schon auf 2^{32} Speicherplätze, was einer Hauptspeicherkapazität von 4 GigaBytes entspricht.

Heutige Prozessoren unterscheidet man — unter anderem — dahingehend, wie umfassend der Befehlssatz ist, den diese Prozessoren verarbeiten können. Diesen Befehlssatz nennt man **Maschinensprache** und ist für jede Prozessorfamilie unterschiedlich. Demgemäß unterscheidet man:⁸

- ☞ **CISC** Prozessoren; CISC steht für **Complex Instruction Set Computing** und bezeichnet einen Prozessortyp, dem ein umfangreicher Befehls-

⁶ engl.: central processing unit, CPU

⁷ Siehe HEROLD, [52], p. 84.

⁸ Siehe dazu auch HEROLD, [52], Kap. 16.2.

satz zur Verfügung steht.

- ☞ **RISC** Prozessoren; RISC steht für **Reduced Instruction Set Computing** und ist die Bezeichnung für Prozessoren mit reduziertem Umfang der Maschinenbefehle.

In Kapitel [3] werden diese Prozessorarchitekturen näher untersucht.

2.2.1 Das Rechenwerk

Gemäß der VON NEUMANN Architektur besteht die Aufgabe des Rechenwerks in der Ausführung arithmetischer und logischer Operationen. In der Abbildung [2.2] ist das Schaltbild einer arithmetisch-logischen Einheit skizziert.

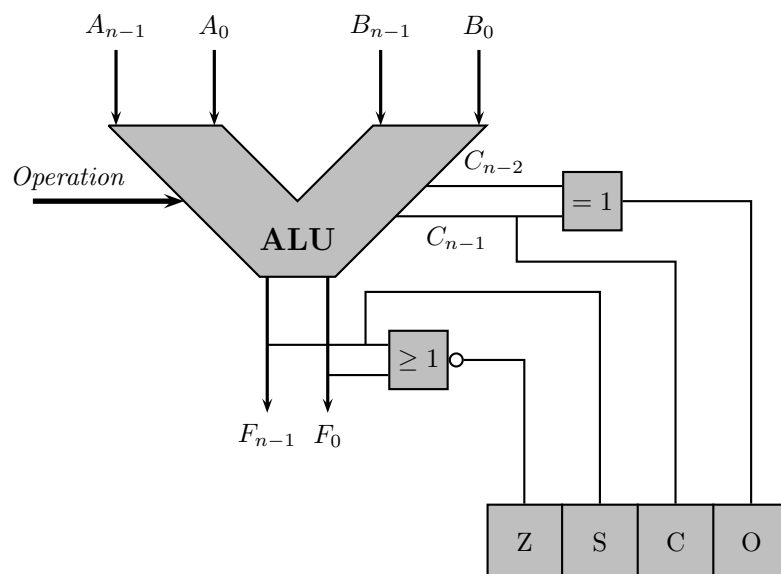


Abbildung 2.2: Aufbau einer ALU mit Flag Register.

Wie das ALU Symbol in dieser Abbildung suggeriert, verknüpft die ALU die beiden Eingänge A und B gemäß der in der zu verarbeitenden Instruktion angegebe-

nen *Operation* und gibt das Resultat am Ausgang **F** aus. Welche der möglichen Operationen auszuführen ist, ist durch den sogenannten **Opcode** festgelegt, der Teil der Instruktion ist. Der **Befehlsdekodierer**, eine Komponente des Steuerwerks, wandelt diesen Opcode in Steuersignale um. Diese Steuersignale wiederum initialisieren die entsprechenden Einheiten der arithmetisch-logischen Einheit.

Neben den arithmetischen Grundoperationen wie Addition, Subtraktion, Multiplikation und Division, verfügt eine ALU über Einheiten, die logische Operationen wie AND, OR, Negation oder Vergleiche durchführen. Weiterhin können arithmetisch-logische Einheiten mit Schaltungen ausgerüstet sein, die die Gleichheit zweier Operanden testet, oder ob ein Operand größer oder kleiner ist als ein zweiter Operand.

Um die Ergebnisse einer arithmetisch-logischen Operation effektiv auswerten zu können, verfügen CPUs über ein spezielles Register, das sogenannte **Flag-Register** oder auch *Status Register*. Unter anderem signalisieren gesetzte bzw. nicht gesetzte Bits dieses Registers die folgenden Angaben:

- ✚ Das **Zero Bit** (Z) zeigt an, ob das Resultat einer ALU Operation Null ist.
- ✚ Das **Sign Bit** (S) zeigt an, ob das Ergebnis – falls es sich um vorzeichen-behaftete Zahlen handelt – negativ ist.
- ✚ Das **Carry Bit** (C) zeigt an, ob ein Übertrag in der höchsten Ergebnisstelle bei der Ausführung einer Addition oder Subtraktion aufgetreten ist.
- ✚ Das **Overflow Bit** (O) zeigt an, ob der Zahlenbereich — falls es sich um eine arithmetische Operation mit Zahlen in der Komplementdarstellung handelt — überschritten wurde.

Bei der Verarbeitung einer Hochsprachen-Anweisung wie zum Beispiel

$$\text{sum} = \text{a} + \text{b};$$

benötigt eine CPU prinzipiell drei Register, zwei für die beiden Operanden **a** und **b**, ein drittes für das Ergebnis **sum**. Die obige Hochsprachen-Anweisung enthält drei Variablen, dies sind die beiden Operanden **a**, **b** und das Resultat **sum**. Beim Übersetzen des Quellcodes durch den Compiler werden diesen Variablen Speicheradressen zugewiesen. An diese Speicherplätze werden die Inhalte dieser Variablen bei der Ausführung des Programms vom **Loader** geladen.

Der Compiler legt in der Phase der Code-Optimierung auch fest, welche CPU-Register bei der Instruktionsverarbeitung nun für welche Funktion genutzt wird.

Werden für eine solche Operation drei Register verwendet, spricht man von einer **Dreiadressmaschine**. Dies ist der Abbildung [2.3] links dargestellt.

Der Schaltungsaufwand kann reduziert werden, indem man die Schaltung rechts in Abbildung [2.3] wählt. Bei einer solchen **Zweiadressmaschine** ist nur die

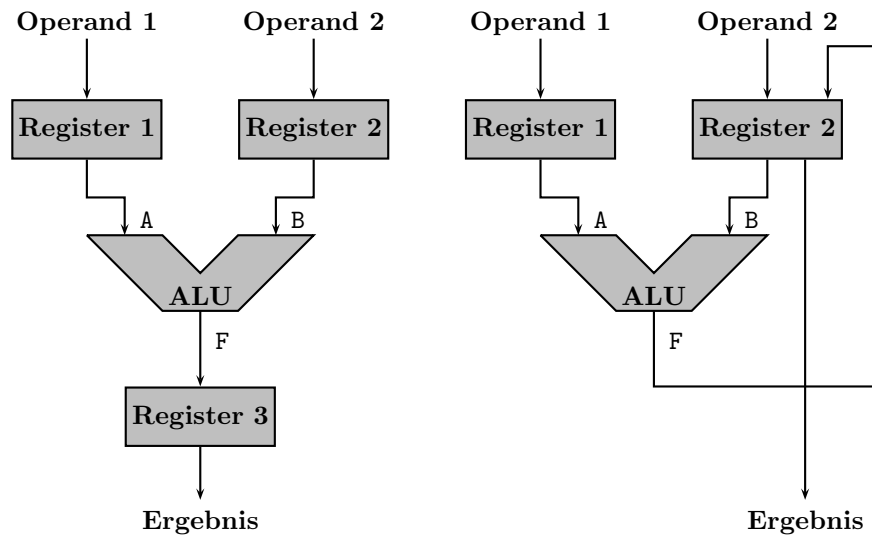


Abbildung 2.3: Verschiedene Architekturen des Rechenwerks.

Angabe der beiden Adressen *Register 1* und *Register 2* erforderlich. Wie in der Abbildung angedeutet, ist der Ausgang **F** der ALU in das Register *Register 2* durchgeschaltet. Das hat zur Folge, dass einer der beiden Operanden durch das Ergebnis überschrieben wird.

Wird nur eine einzige Adresse angegeben, was kurze Befehlsformate für die Maschineninstruktionen zur Folge hat, dann liegt eine sogenannte **Einadressmaschine** vor. Die Schaltung solch einer Architektur ist in der Abbildung [2.4] dargestellt. Der erste Operand wird in das Register 1 geladen, der zweite Operand landet immer im sogenannten **Akkumulator**. Dies ist ein speziell reserviertes Register für genau diesen Zweck. Nach der Ausführung der Anweisung durch die ALU steht das Ergebnis im Akkumulator, da der Ausgang **F** der ALU auf das Akkumulator Register durchgeschaltet ist. Der Vorteil einer Einadressmaschine liegt in der kurzen Befehlslänge. In der Maschineninstruktion muß lediglich eine Adresse für das Register angegeben werden, der zweite Operand wird standardmäßig in den Akkumulator geladen.

Eine Dreiadressmaschine erlaubt es, alle drei Register in einem Befehl zu adressieren. Dies ist flexibler und damit leistungsfähiger als eine Ein- oder Zweiadressmaschine, aber dafür auch länger, *i.e.* solche Instruktionen benötigen mehr Speicherplatz.

Die wichtigsten heutigen Prozessoren (*e.g.* Intel 80x86) sind als Zweiadressmaschinen aufgebaut.

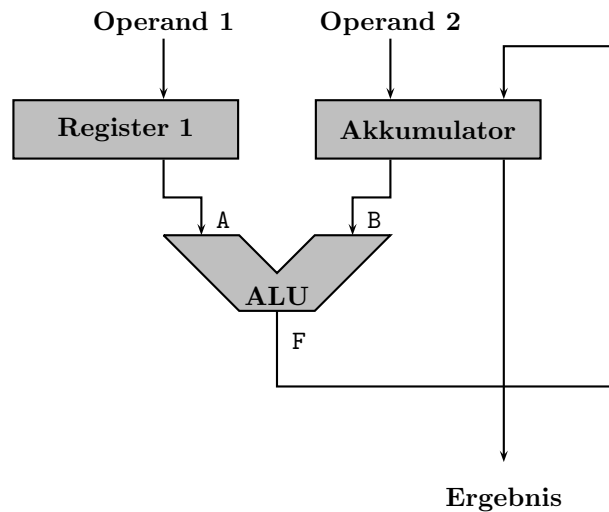


Abbildung 2.4: Einadressmaschine.

2.3 Das Steuerwerk

Zu der wichtigsten Aufgaben des **Steuerwerks** gehört die Koordination der temporären Abläufe im Rechner. Darunter fällt das Holen der auszuführenden Instruktionen aus dem Hauptspeicher, das Decodieren der Instruktion und die Steuerung der Ausführung. Diese Abarbeitung einer Instruktion nennt man den

Fetch-Decode-Execute Zyklus (FDE Zyklus)

2.3.1 Aufbau des Steuerwerks

Zur Durchführung des FDE Zyklus ist das Steuer- oder Leitwerk aus vier Komponenten aufgebaut:

- ☞ das Befehlsregister oder **Instruction Register (IR)**
Im Befehlsregister befindet sich der jeweils aktuelle Befehl, der gerade abgearbeitet wird. Das Instruction Register ist ein spezielles Register der CPU, das anwenderseitig nicht ansprechbar ist.
- ☞ der Befehlszähler oder **Instruction Counter (IC)**
Der Befehlszähler — oder auch Instruction Counter oder Program Counter genannt — enthält die Hauptspeicheradresse des gerade in Abarbeitung befindenden Befehls. Die Erhöhung des Inhalts dieses Registers — das ebenfalls ein reserviertes CPU Register ist — um eine Einheit spricht den nächsten Befehl an.

☞ das **Speicheradressregister** oder **Memory Address Register** (MAR)

Im Speicheradressregister steht entweder

- die Hauptspeicheradresse der nächsten auszuführenden Instruktion

oder

- die Hauptspeicheradresse eines Datenwortes, falls zur Ausführung einer Instruktion ein Datenwort aus dem Hauptspeicher in ein CPU Register geladen werden oder von der CPU in den Hauptspeicher transferiert werden muß.

☞ der Befehlsdecoder

Der Befehlsdecoder decodiert die Maschineninstruktionen und generiert die Steuersignale zu den entsprechenden Hardwarekomponenten.

Die Abbildung [2.5] zeigt schematisch die Struktur einer Zentraleinheit. Neben den Bestandteilen des Steuerwerks sind auch verschiedene Register und die ALU berücksichtigt.

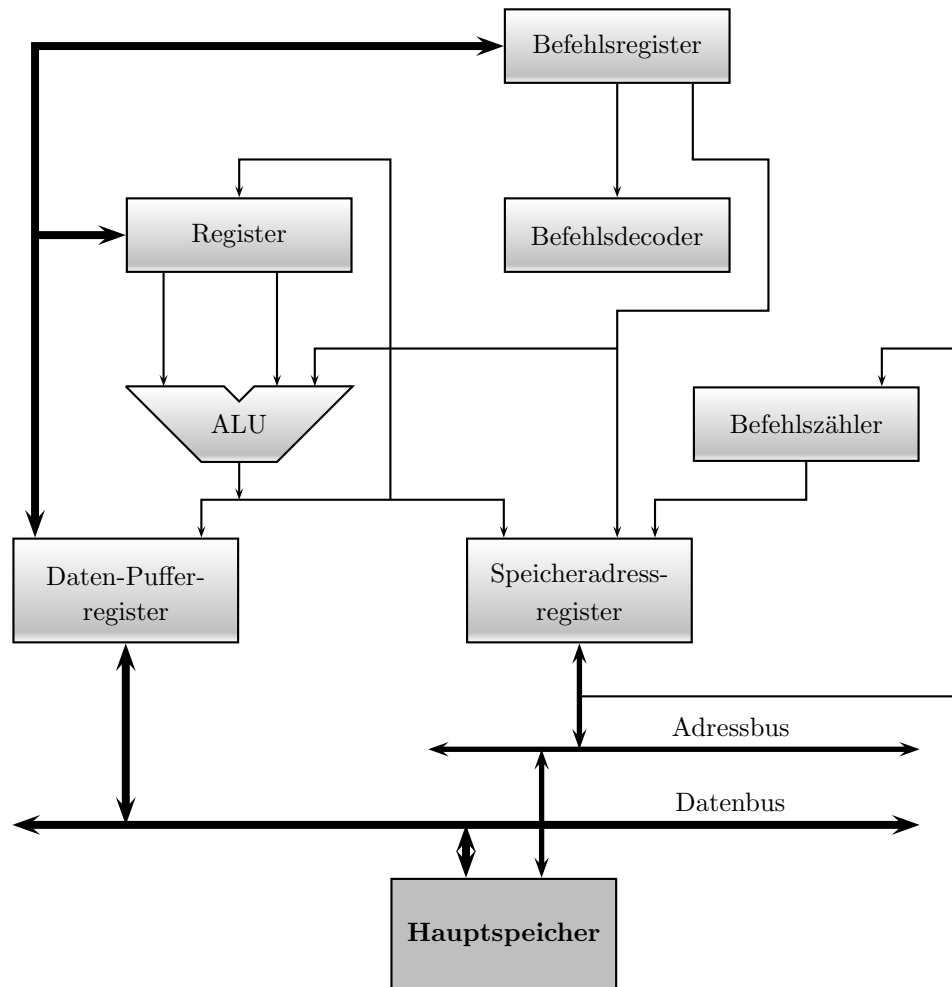


Abbildung 2.5: Struktur und Datenfluß in einer Zentraleinheit.

Neben den oben erklärten Aufgaben hat das Speicheradressregister eine weitere Funktion. Jede Speicheradresse wird solange zwischengespeichert — also gepuffert — bis der Systembus — *i.e.* der Adressbus und der Datenbus — frei ist. Erst dann kann das Speicheradressregister zum richtigen Zeitpunkt die Hauptspeicheradresse auf den Adressbus legen. Die analoge Aufgabe für den Datentransfer erfüllt das Daten-Puffer-Register. Daten, die die zentrale Recheneinheit über den Systembus zum Ziel transferieren muß, müssen so lange gepuffert werden, bis der Datenbus frei ist.

Das Steuerwerk erhält die durchzuführenden Aufgaben durch die Instruktionen des in der Abarbeitung befindenden Programms. Die **Maschinenbefehle** oder Instruktionen geben an, welche Operation mit welchen Daten (Operanden) auszuführen sind und wohin — also an welche Hauptspeicherstelle — das Resultat der Operation abzupeichern ist. Maschineninstruktionen bestehen daher aus zwei Teilen:

- ☞ dem **Opcode** oder **Operation Code**, der die Art des auszuführenden Befehls angibt; also, ob Addition, Subtraktion oder eine andere arithmetische Operation auszuführen ist.
 - ☞ dem **Operandenteil** oder *Operand Specifier*. Je nach Rechnerart muß die Maschineninstruktion die Hauptspeicheradresse(n) und/oder die Registeradressen der Operanden enthalten, auf die Operation angewendet werden soll. Dabei differenziert man in Null-, Ein-, Zwei- und Dreiadressbefehle. Die Nulladressbefehle werden im Abschnitt [2.4.3] näher diskutiert.
- Bei einem **Einadressbefehl** gibt der Operandenteil der Maschineninstruktion die Adresse eines Operanden an. Der andere Operand wird automatisch in den **Akkumulator** geladen. Das Resultat der Operation wird automatisch in den Akkumulator geschrieben.



- Bei dem **Zweiadressbefehl** steht im Operandenteil der Instruktion die Adressen der beiden Operanden. Das Ergebnis der Operation wird je nach Rechnerart unter der (Register)–Adresse des ersten oder zweiten Operanden gespeichert. Einen Akkumulator wird bei diesem Design nicht benötigt.



Eine Instruktion mit solch einem Format ist beispielsweise die Assembleranweisung (z.B. für Intel 80x86)

add AX,BX

Durch diese Anweisung werden die Inhalte der beiden Register **AX** und **BX** additiert, das Ergebnis wird im Register **AX** abgelegt.

- Bei dem **Dreiadressbefehl** wird zusätzlich noch die Adresse (z.B. ein Register) für das Ergebnis in der Instruktion angegeben.

Opcode	Adr. Resultat	Adr. Operand 1	Adr. Operand 2
--------	---------------	----------------	----------------

Anmerkung:

Eine in einer Hochsprache wie zum Beispiel C formulierte Anweisung wie

$$\text{sum} = \text{a} + \text{b};$$

verwendet drei Variable. Prinzipiell sind daher zumindest drei Adressen notwendig, diese Hochsprachenanweisung in Maschinensprache umzusetzen. Der Compiler der Hochsprache muß nun diese Anweisung in das jeweilige Adressformat der zugrundeliegenden Struktur der Maschinenbefehle übersetzen.

Die folgende Tabelle zeigt die unterschiedlichen Implementierungen von Maschineninstruktionen.

Prozessor	Anzahl Instr.	Länge
IBM 370	≈ 140	2,4,6, Byte
DEC VAX	≈ 330	2 - 16 Byte
Motorola 68000	≈ 83	2 - 5 Byte
8086	90	1 - 6 Byte

2.3.2 Der Fetch-Decode-Execute Zyklus

Die vom Steuerwerk koordinierte Ausführung der Maschineninstruktionen wird bei einer Top-Level Betrachtung⁹ in die folgenden drei Phasen aufgeteilt:

FETCH, DECODE und EXECUTE.

In diesen einzelnen Phasen werden folgende Schritte ausgeführt

- **FETCH**: Holen von Befehlen
Der Inhalt der von dem Instruction Counter (IC) bestimmten Speicherzelle wird aus dem Arbeitsspeicher in das Instruktionsregister (IR) geladen.

⁹Wir betrachten zunächst eine vereinfachte Darstellung des FDE Zyklus um die grundlegenden Aspekte herauszuarbeiten.

- **DECODE:** Entschlüsseln von Befehlen
Der Befehl wird in seine Bestandteile zerlegt, nämlich in
 - Operationsteil
 - Adressteil
 - Operandenteil

Die Steuereinheit muß erkennen, um welche Befehlsart es sich handelt (logischer Vergleich, Addition,...).

- **EXECUTE:** Initiierung der Befehlsausführung
Dieser Schritt erfolgt durch die Versorgung aller an der Befehlsausführung beteiligten Funktionseinheiten mit den notwendigen Steuersignalen (beispielsweise durch eine Folge von Mikrooperationen) — beispielsweise für die Adressierung und das Laden von Operanden, die Speicherung von Ergebnissen oder die Veränderung des Befehlszählers. Wenn alle notwendigen Signale gesetzt sind, beginnt automatisch die Ausführung des Befehls.

Beispiel 2.5:

Als Beispiel untersuchen wir hier die Ablaufsteuerung bei der Abarbeitung der Instruktionsfolge

```
int a,b,sum;  
  
a = 5;  
b = 7;  
sum = 0;  
  
sum = a + b;
```

Vereinfachend nehmen wir dabei an, dass direkt auf den Hauptspeicher zugegriffen wird, d.h. wir ignorieren den Zwischenschritt, dass der Compiler die Additionsanweisung der Variablen beispielsweise in die Einzelschritte

```
load R1,a  
load R2,b  
add R3,R2,R1  
store sum,R3
```

auflöst, wobei R1,R2,R3 drei CPU Allzweckregister bezeichnet.

Durch die Deklaration `int a,b,sum;` wird im Hauptspeicher Speicherplatz reserviert für drei Integer Variablen. Der Compiler weist diesen Variablen beispielsweise die Speicheradressen 5000 bis 5002 zu (siehe Abbildung [2.6]). Die Daten eines Programms werden also in einem zusammenhängenden Speicherbereich abgelegt, den *Datenbereich*. Die Instruktionen selbst werden in einem

weiteren zusammenhängenden Bereich gemäß der VON NEUMANN Architektur sequentiell abgespeichert — dies ist der *Programmteil* — in der Speicherzelle 4000 liegt die Instruktion $a = 5$, in 4001 die Initialisierung $b = 7$ usw. und in der Speicherzelle 4003 die Addition $sum = a + b$.

Wir wollen nun die einzelnen Schritte betrachten, die dieses System auszuführen hat, wenn die Instruktion $sum = a + b$ abgearbeitet wird.

- ① Für die Ausgangssituation nehmen wir an, daß im Befehlszähler IC – siehe auch die Abbildung [2.5] – die Hauptspeicheradresse 4002 liegt, im Befehlsregister selbst die Instruktion $sum = 0$. Die CPU hat gerade die Bearbeitung dieser Instruktion beendet. Diese Situation ist in der Abbildung [2.6] dargestellt.

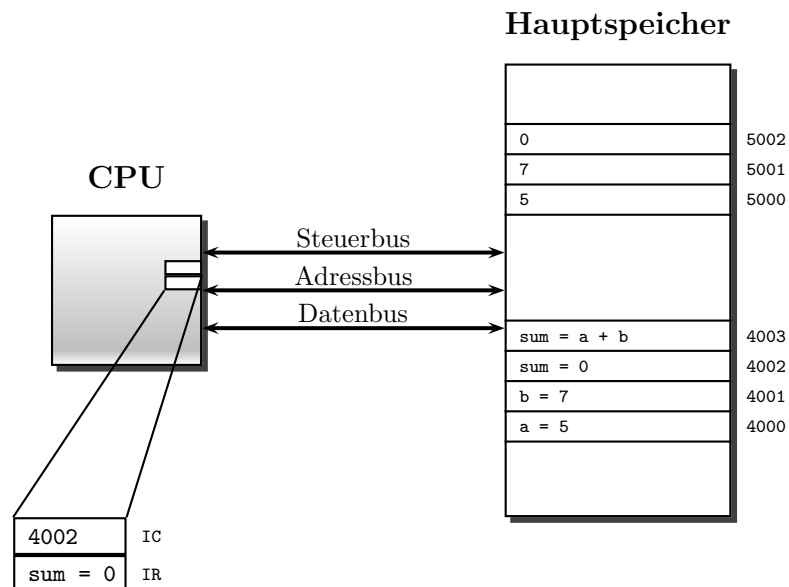


Abbildung 2.6: Zum FDE-Zyklus: Ausgangssituation.

- ② **Fetch:** Der Befehlszähler IC wird vom Steuerwerk um 1 erhöht. Im Instruction Counter steht nun der Wert 4003.
 - ❶ Der Inhalt des Instruction Counters (IC) wird in das Speicheradress-Register (MAR) geladen.
 - ❷ Das Steuerwerk sendet über den Steuerbus ein *Lesen*-Signal an den Hauptspeichercontroller.
 - ❸ Die im Speicheradress-Register (MAR) liegende Instruktionsadresse 4003 wird über den Adressbus an den Hauptspeichercontroller

übertragen. Mit Hilfe des Decoders wird die so adressierte Speicherzelle angesprochen.

- ④ Der Hauptspeichercontroller gibt den Inhalt der Speicherzelle 4003 auf den Datenbus. Die Instruktion¹⁰ $\text{sum} = \text{a} + \text{b}$ wird zur CPU übertragen, in das Daten-Puffer Register geladen und anschließend in das Befehlsregister.
- ③ **Decode:** Sobald die Instruktion $\text{sum} = \text{a} + \text{b}$ in das Befehlsregister geladen ist, ist die Fetch-Phase beendet und die Decode Phase beginnt. Der Befehlsdekodierer muß nun die Instruktion

$$\text{sum} = \text{a} + \text{b}$$

in die Bestandteile zerlegen und führt dabei folgende Schritte aus:

- ① Der Instruction Counter wird um 1 erhöht.
 - ② Lade den Operanden **a**, i.e. den Inhalt der Speicherzelle 5000, in ein Allzweckregister (z.B. R1)
 - ③ Lade den Operanden **b**, i.e. den Inhalt der Speicherzelle 5001, in ein Allzweckregister (z.B. R2)
 - ④ Verbinde die Register R1 und R2 mit den Eingängen der ALU
 - ⑤ Steuersignal an die ALU, so daß gemäß Opcode das Addierwerk initialisiert wird
 - ⑥ Der Ausgang der ALU wird auf ein freies Allzweckregister z.B. R3 geschaltet.
- ④ **Execute:** In der Ausführphase können folgende Schritte durchgeführt werden
- ① Der Inhalt der Register R1, R2 wird in die ALU geladen und die Addition – z.B. durch ein Ripple Carry Adder ausgeführt.
 - ② Die ALU sendet gegebenenfalls Steuersignale, um bestimmte Flags im Flag Register zu setzen.
 - ③ Das Resultat der Addition in der ALU wird in das Allzweckregister R3 geschrieben.
 - ④ Der Inhalt der Registers R3 wird in die Speicherstelle 5002 des Hauptspeichers zurück geschrieben.

Anmerkungen:

- 1 Jeder der im obigen Ablauf auftretenden Zugriffe auf den Hauptspeicher — dies betrifft insbesondere den Transfer der Operanden vom Hauptspeicher in die CPU — muß folgende Schritte durchlaufen:

¹⁰Dies geschieht natürlich in binärer Codierung der Instruktion.

- ① Das Steuerwerk sendet über den Steuerbus ein 'Lesen' Signal an den Hauptspeichercontroller.
- ② Über den Adressbus wird die Adresse der zu lesende Speicherzelle vom Speicheradressregister an den Hauptspeichercontroller transferiert.
- ③ Der Hauptspeichercontroller liest den Inhalt der angeforderten Speicherzelle aus.
- ④ Der Hauptspeichercontroller transferiert den Inhalt der angeforderten Speicherzelle über den Datenbus zur CPU in das Daten Puffer Register.

2 Das Flag Register dient dazu, bestimmte Resultate der arithmetisch logischen Einheit anzuzeigen. Beispielsweise ergibt die Addition zweier `int` Variablen unter Umständen eine Bereichsüberschreitung.

Beispiel: Angenommen man codiert die ganzen Zahlen mit $N = 5$ Bits in Zweierkomplementdarstellung. Damit sind die Zahlen im Intervall

$$I = [-16, +15]$$

darstellbar. Addiert man die beiden Zahlen $z_1 = 12, z_2 = 5$ in dieser Darstellung, dann sind zwar die beiden Zahlen selbst im erlaubten Zahlenbereich, die Summe kann aber nicht mehr dargestellt werden, es liegt eine *Bereichsüberschreitung* vor. Generell wird die Bereichsüberschreitung dadurch erkannt, daß bei der Addition ein Vorzeichenwechsel auftritt, da die obige Addition das Resultat -15 ergibt. Dies wird in der arithmetisch logischen Einheit geprüft, wenn ein solcher Vorzeichenwechsel eintritt, wird ein bestimmtes Bit des Flag Registers gesetzt.

Die obige detailliertere Betrachtung des Fetch Decode Execute Zyklus legt es nahe, die Ausführung einer Instruktion in fünf Phasen einzuteilen:

- ☞ die Fetch-Phase, *i.e.* Holen der Instruktion,
- ☞ die Decode-Phase, *i.e.* dekodieren der Instruktion,
- ☞ die Operanden-Holphase, *i.e.* Laden der Operanden in Allzweckregister,¹¹
- ☞ die Execute-Phase, *i.e.* Ausführung der Operation
- ☞ die Write-Back-Phase, *i.e.* das Zurückschreiben des Resultats in den Hauptspeicher.

¹¹Dies kann – bei entsprechender Implementierung eines der anderen Ausführungsmodelle – auch der Akkumulator oder der Stack sein.

Anmerkung:

Die Anzahl der Speicherzellen des Hauptspeichers — mit anderen Worten, die Größe des adressierbaren Hauptspeichers — die von dem Prozessor angesprochen werden kann, hängt entscheidend von der Anzahl der Datenleitungen des Adressbusses ab. Man nennt dies die **Adressbusbreite**. Stehen zur Übertragung der Hauptspeicheradressen n Leitungen zur Verfügung, können damit 2^n verschiedene Adressen zwischen CPU und Hauptspeicher transferiert werden. Die Terminologie ist eine Busbreite in Bit, *i.e.* ist der Adressbus beispielsweise 32 Bit breit, können damit 2^{32} Speicherzellen adressiert werden.

Der heutige Busstandard PCI (*peripheral component interconnect*) unterstützt eine Busbreite von 32 oder 64 Bit. Um Leitungen zu sparen, werden die Busleitungen abwechselnd als Adress- und Datenleitungen genutzt. Dies nennt man daher **Multiplexbetrieb**.

2.4 Ausführungsmodelle

Durch die unterschiedlichen Möglichkeiten der Adressierungen innerhalb der Maschineninstruktionen haben sich eine Reihe verschiedener **Ausführungsmodelle** gebildet, nach denen Prozessoren arbeiten.

2.4.1 Allzweckregister Architekturen

✕ Register–Register Modell

Alle Operanden und das Ergebnis stehen in Allzweckregistern. Eine Additionsoperation wird auf der Ebene der Maschinensprache daher durch eine Anweisung der Form

`add R1,R2,R3`

umgesetzt, wobei R1,R2 und R3 Register der CPU bezeichnen.

In dieser Architektur wird mit genau zwei Instruktionen auf den Hauptspeicher zugegriffen. Die Instruktion **LOAD** holt die Operanden aus dem Hauptspeicher in Register, die Instruktion **STORE** schreibt Registerinhalte in Hauptspeicherplätze. Diese Architektur nennt man **Load/Store Architektur**

```
load R2,A
load R3,B
add R1,R2,R3
store C,R1
```

Diese Architektur erfordert das Dreiadressformat der Maschineninstruktionen.

Anwendung findet diese Architektur in den Prozessoren Alpha, MIPS, PowerPC und SPARC.

Vorteil:

- ✓ Einfaches und festes Instruktionsformat,
- ✓ einfaches Code-Generierungsmodell,
- ✓ etwa gleiche Ausführungszeit der Instruktionen.

Nachteil:

- ✓ Höhere Anzahl von Instruktionen im Vergleich zu Architekturen mit Hauptspeicherreferenzen,
- ✓ mehr Instruktionen und geringere Befehlsdichte führen zu umfangreicheren Programmen.

✕ Register–Speicher Modell

Charakteristische Eigenschaft dieser Architektur ist, ein Operand ist im Hauptspeicher lokalisiert, der zweite Operand in einem Allzweckregister. Das Ergebnis der Operation wird entweder in ein Register oder in den Hauptspeicher geschrieben.

```
add R1,A    mem[A] ← mem[A] + R1
add A,R1    R1 ← R1 + mem[A]
```

Diese Architektur erfordert explizite und überdeckte Adressierung der Operanden bzw. des Resultats. Explizite Adressierung bedeutet, dass der Opcode die Angabe der Adresse des Operanden enthält. Von einer überdeckten Adressierung spricht man, wenn zwei Adressen (Quelle und Ziel) zusammenfallen.

Das Register–Speicher Modell erfordert das Zweiadressformat für die Instruktionen. Das Befehlsformat enthält zwei explizite Adressangaben für

- 1 ein prozessorinternes Register
- 2 eine prozessorexterne Hauptspeicheradresse

Diese Architektur impliziert die Überdeckung einer Quelladresse mit einer Zieladresse.

Anwendung findet dieses Modell in den Prozessoren IBM 360/370, Intel 80x86 und Motorola 680x0.

Vorteil:

- ✓ Auf die Operanden kann ohne vorherige Lade-Operation direkt zugegriffen werden,
- ✓ Codierung im Befehlsformat führt zu höherer Code-Dichte, *i.e.* kompakteren Programmcode.

Nachteil:

- ✓ Aufgrund der Überdeckung können die Operanden nicht gleich behandelt werden,
- ✓ die Anzahl der Taktzyklen pro Instruktion variiert in Abhängigkeit von der Adressierung.

2.4.2 Akkumulator–Register Architektur

Die CPU verfügt über ein ausgezeichnetes Register, den **Akkumulator**. Der Akkumulator wird bei einer zweistelligen Operation als Quelle einer der beiden Operanden angesprochen. Gleichzeitig dient der Akkumulator als Ziel für das Resultat.


```
load a  
add b  
store c
```

Dieses Modell erfordert das Einadressformat für die Maschineninstruktionen.

2.4.3 Keller Architektur

Ein **Stack** oder auch **Kellerspeicher** genannt, ist eine geordnete Menge von Elementen. Pro Zeiteinheit kann nur auf eines dieser Elemente zugegriffen werden. Dieses Element heißt Top-Element des Stacks. Die Anzahl der Elemente des Stacks — auch Stacklänge genannt — ist variabel. Einträge können lediglich auf den Stack gesetzt oder von oben weggenommen werden. Aus diesem Grund nennt man einen Stack auch **Pushdown Liste** oder **Last-In-First-Out** Liste (LIFO).

Die Abbildung [2.7] zeigt die grundlegenden Stack-Operationen. Diese Operationen sind:

- ☞ **push** → ein Element auf dem Stack ablegen,
- ☞ **pop** → ein Element vom Stack nehmen.

Bei der Verarbeitung beginnt man zu einem bestimmten Zeitpunkt, bei dem der Stack bereits eine Reihe von Einträgen enthält. Die **push** Operation setzt einen neuen Eintrag auf dem Stack ab, die **pop** Operation nimmt das oberste Element vom Stack weg. In beiden Fällen verschiebt sich die Adresse des obersten Stackelementes entsprechend.

Binäre Operationen, die zwei Operanden benötigen wie Addition, Subtraktion, Multiplikation oder Division benutzen die beiden obersten Stackeinträge als Operanden. Diese beiden Einträge werden vom Stack weggenommen und in CPU-Register geladen, das Resultat der Operation wird auf dem Stack oben abgelegt.

Das Stack-Modell ist ein sehr zweckmäßiges Werkzeug für Programmierer, das von einer CPU-Implementierung zur Verfügung gestellt werden kann. Ein typisches Beispiel hierfür sind Rekursionen in Programmen. Die Implementierung eines Stacks hängt zum Teil von dessen potentieller Anwendung ab.

- ☞ Ist es eine gewünschte Eigenschaft, dem Programmierer bestimmte Stack-Operationen zur Verfügung zu stellen, muß der Befehlssatz der Maschine stack-orientierte Instruktionen wie **push** oder **pop** enthalten sowie Operationen, die die beiden obersten Stackeinträge als Operanden nutzen. Da all diese Operationen sich auf eine eindeutige Hauptspeicherlokation beziehen – nämlich den obersten Eintrag im Stack – sind die Adressen der

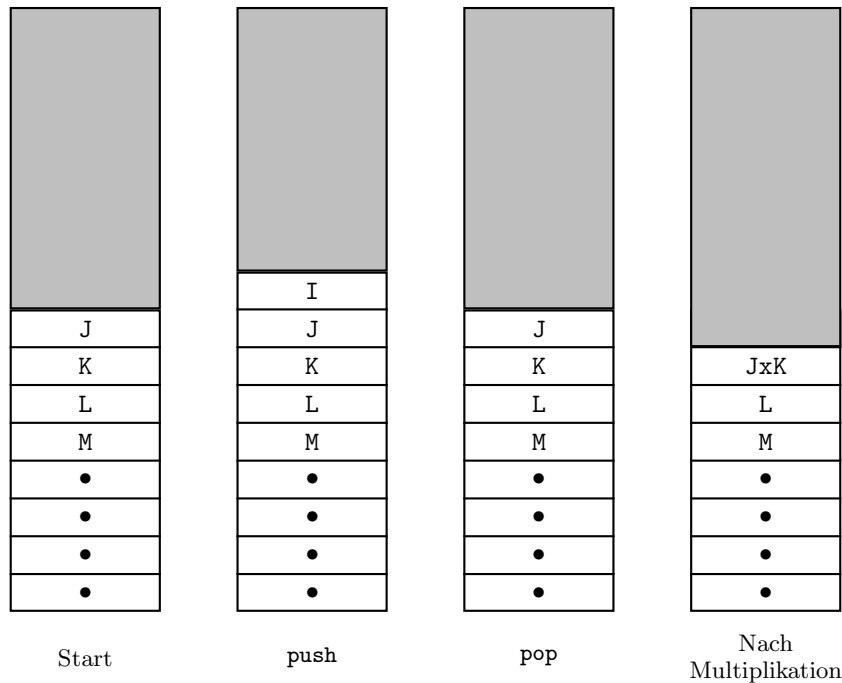


Abbildung 2.7: Grundlegende Operationen eines Kellerspeichers.

Operanden implizit und müssen in den Stack-Instruktionen nicht explizit angegeben werden. Dies sind daher **Nulladdress-Instruktionen**.

- ☞ Soll der Stack-Mechanismus nur durch die CPU genutzt werden — *e.g.* für das Handling von Prozeduren — enthält der Instruktionssatz keine explizit stackorientierte Befehle.

In jedem dieser Fälle erfordert die Implementierung eines Stacks, dass eine Reihe von Informationen über bestimmte Lokationen des Stacks bereitgestellt werden müssen, um die Stackelemente im Hauptspeicher ablegen zu können.

Eine typische Implementierung des Keller Modells ist in der Abbildung [2.8] links dargestellt. Ein zusammenhängender Block von Speicherplätzen ist für den Stack im Hauptspeicher reserviert. Die meiste Zeit über ist dieser Block teilweise mit den Stackelementen gefüllt, im freien Bereich können weitere Stackelemente abgelegt werden.

Für das Management des Stacks sind drei Adressen notwendig. Diese Adressen werden oftmals in (dafür reservierte) CPU Register gespeichert:

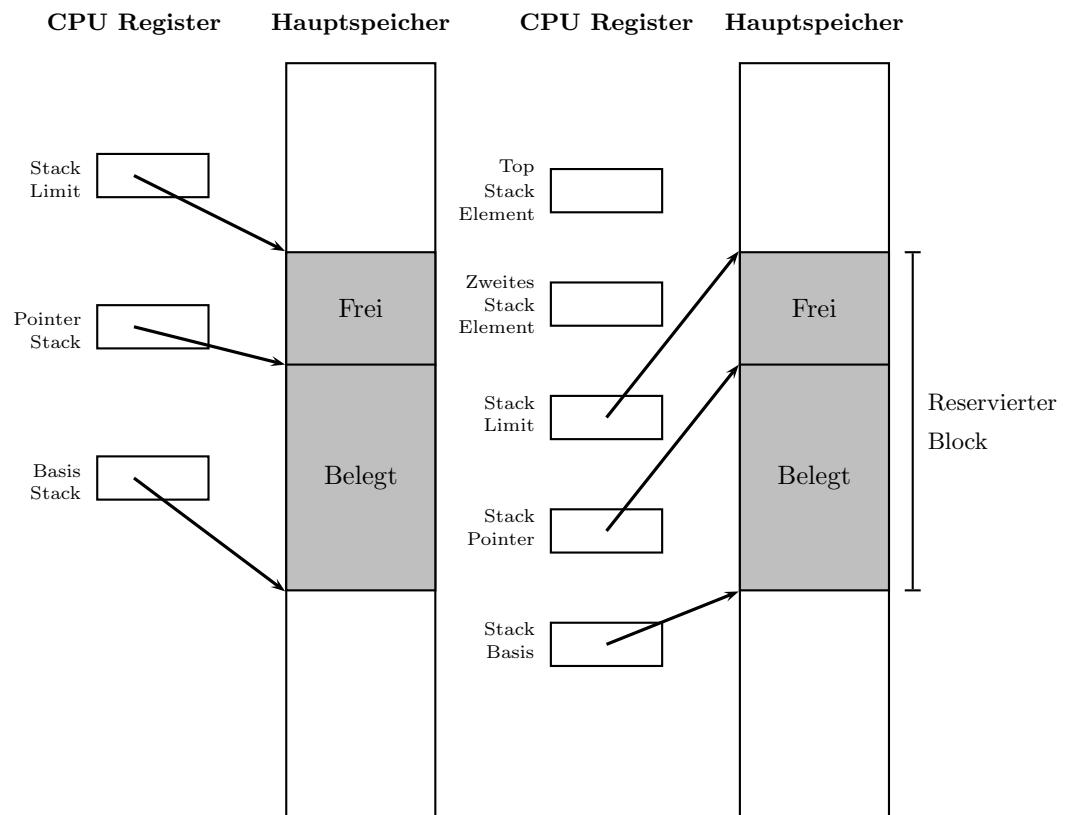


Abbildung 2.8: Organisation des Stacks.

✗ Stack Pointer

Der Stack Pointer enthält die Hauptspeicheradresse des obersten Eintrags auf dem Stack. Wird ein Eintrag auf dem Stack abgelegt oder ein Eintrag vom Stack weggenommen, wird dieser Zeiger entsprechend erhöht oder vermindert. Dadurch zeigt dieser Pointer immer auf den obersten Stack-Eintrag.

✗ Stack Basis

Im Stack Base wird die Hauptspeicheradresse des untersten Stackeintrags abgelegt. Das ist gleichzeitig die Startadresse des für den Stack reservierten Speicherbereichs im Hauptspeicher. Ist der Stack-Speicher leer und wird versucht eine **pop** Operation auszuführen, wird eine Fehlermeldung erzeugt.

✗ Stack Limit

Der Stack Limit enthält die Hauptspeicheradresse des 'oberen' Endes des für den Stack reservierten Bereichs. Ist dieser reservierte Bereich komplett

gefüllt und wird versucht, eine `push` Operation auszuführen, dann wird eine Fehlermeldung erzeugt.

Traditionell — und auf den meisten Plattformen implementiert — ist die Basisadresse des Stacks die hohe Adresse, der Stack Limit ist die niedere Adresse. Daher wächst der Stack von den hohen Hauptspeicheradressen zu den niederen Adressen.

Verarbeitung der Ausdrücke

Mathematische Ausdrücke werden üblicherweise in einer Weise ausgedrückt, die man **Infix Notation** nennt. In dieser wohlvertrauten Schreibweise tritt ein binärer Operator zwischen den beiden Operanden auf, wie z.B. bei $a + b$. Für komplexere Ausdrücke werden Klammern verwendet, um die Reihenfolge der Auswertung des Ausdrucks festzulegen. Beispielsweise liefert

$$a + (b \times c)$$

ein anderes Resultat als

$$(a + b) \times c$$

Um den Gebrauch von Klammern zu minimieren, verwendet man üblicherweise eine implizite Operatorpräzedenz, *i.e.*, die 'Punkt vor Strich' Regel. Multiplikation hat stets Vorrang vor Addition. So ist $a + b \times c$ äquivalent zu $a + (b \times c)$.

Eine alternative Technik ist unter dem Schlagwort **Umgekehrte Polnische Notation** oder **Postfix Notation** bekannt (siehe z.B. [1]). In dieser Schreibweise folgt der Operator auf die beiden Operanden.

Beispiel:

$$\begin{aligned} a + b &\text{ wird zu } a\ b\ + \\ a + (b \times c) &\text{ wird zu } a\ b\ c\ \times\ + \\ (a + b) \times c &\text{ wird zu } a\ b\ +\ c\ \times \end{aligned}$$

Ungeachtet der Komplexität eines Ausdrucks werden bei der Verwendung der umgekehrten polnischen Notation keine Klammern benötigt.

Der Vorteil der Postfix Notation liegt in der einfachen Ausführung, wenn man einen Stack benutzt. Ein Ausdruck in Postfix Notation wird von links nach rechts gescannt. Für jedes Element dieses Ausdrucks gelten nun folgende Regeln:

- ① Ist das Element eine Variable oder eine Konstante, setze diese auf den Stack.
- ② Ist das Element ein Operator, nimm die beiden oberen Elemente des Stacks, führe die Operation aus und lege das Ergebnis auf dem Stack ab.

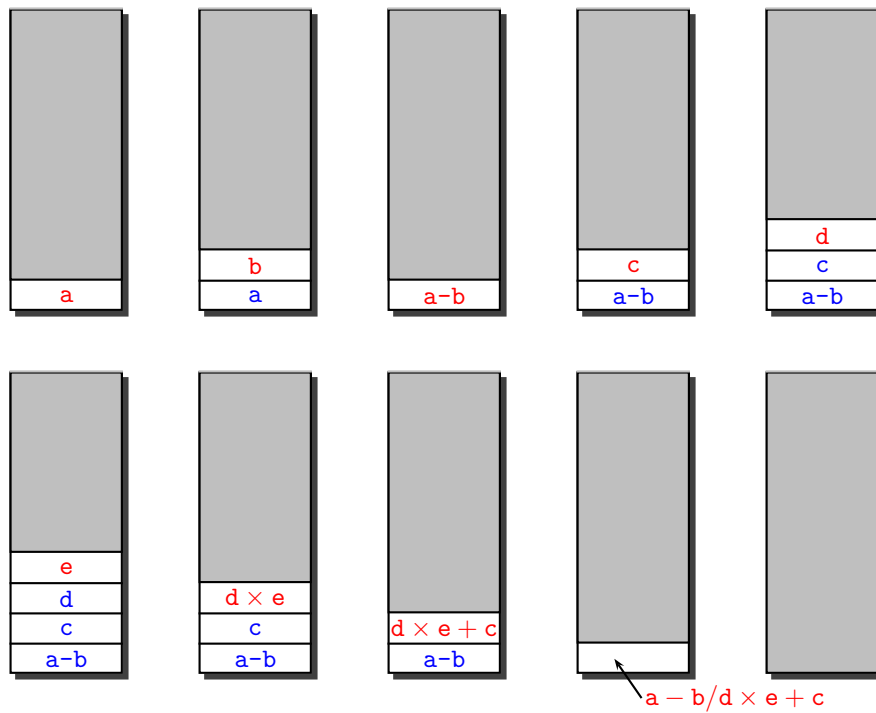
Nachdem der komplette Ausdruck gescannt ist, liegt das Ergebnis oben auf dem Stack. Die Anweisung

$$f = (a - b) / (c + d \times e)$$

wird mit Hilfe der stack-orientierten Maschineninstruktionen folgendermaßen verarbeitet:

```
push a
push b
subtract
push c
push d
push e
multiply
add
divide
pop f
```

Die Abfolge der Stackbelegungen bei der Abarbeitung dieses Programms ist in der Abbildung [2.9] dargestellt.

Abbildung 2.9: Stackverarbeitung des Ausdrucks $f = (a - b) / (d \times e + c)$.

Die Umwandlung eines Ausdrucks in Infix Notation in einen Ausdruck in Postfix Notation kann am einfachsten ebenfalls mit Hilfe eines Stacks realisiert werden. Der folgende Algorithmus geht auf EDGAR DIJKSTRA ([86]) zurück.

Umwandlung Infix \rightarrow Postfix Notation¹²

- ① Input: Ein algebraischer Ausdruck in Infix - Notation
- ② Betrachte das nächste Element des Inputs
- ③ Wenn das Element eine Variable oder eine Konstante ist \rightarrow Ausgabe
- ④ Wenn das Element eine öffnende Klammer ist \rightarrow auf den Stack
- ⑤ Wenn das Element ein Operator ist
 - (a) Ist der Stack leer, push den Operator auf den Stack.
 - (b) Ist das Element auf dem Stack eine öffnende Klammer, push den Operator auf den Stack.
 - (c) Falls der Operator eine höhere Priorität als das Top-Element des Stacks (Multiplikation und Division haben höhere Priorität als Addition und Subtraktion), push den Operator auf den Stack.
 - (d) Sonst, pop die Operatoren so lange vom Stack in den Output bis entweder der Stack leer ist oder ein Operator geringerer Priorität auf dem Stack liegt. Setze den Operator auf den Stack.
- ⑥ Wenn das Element eine schließende Klammer ist, pop die Operatoren vom Stack in den Output bis eine öffnende Klammer Top-Element des Stacks ist. Pop und verwirf die öffnende Klammer.
- ⑦ Gibt es weitere Input Elemente, gehe zu Step 2.
- ⑧ Sonst, pop die verbleibenden Operatoren in den Output STOP

Beispiel 2.6:

Die Arbeitsweise des DIJKSTRA Algorithmus erkennt man am einfachsten, wenn man dieses Verfahren an einem Ausdruck durchexerziert. Dazu betrachten wir den algebraischen Ausdruck

$$a + b \times c + (d + e) \times f$$

Inputi	Step	Output	Stack
$a + b \times c + (d + e) \times f$	–	leer	leer
$+b \times c + (d + e) \times f$	3	a	leer
$b \times c + (d + e) \times f$	5-a	a	+
$\times c + (d + e) \times f$	3	$a\ b$	+
$c + (d + e) \times f$	5-c	$a\ b$	$+\times$
$+(d + e) \times f$	3	$a\ b\ c$	$+\times$
$(d + e) \times f$	5-d	$a\ b\ c \times +$	+
$d + e) \times f$	4	$a\ b\ c \times +$	$+($
$+e) \times f$	3	$a\ b\ c \times +d$	$+($
$e) \times f$	5-b	$a\ b\ c \times +d$	$+(+$
$) \times f$	5-b	$a\ b\ c \times +d\ e$	$+(+$
$\times f$	6	$a\ b\ c \times +d\ e+$	+
f	3	$a\ b\ c \times +d\ e+$	$+\times$
leer	8	$a\ b\ c \times +d\ e + f$	$+\times$
leer	8	$a\ b\ c \times +d\ e + f \times +$	leer

Dieses Modell wird bei dem 80x86 Gleitkomma-Prozessor verwendet, und bei der Java Virtual Machine (JVM) eingesetzt. Die POSTSCRIPT Druckersprache arbeitet ebenfalls nach diesem Prinzip.

2.4.4 Speicher–Speicher Architektur

Die beiden Operanden einer zweistelligen Operation sowie das Ergebnis stehen im Hauptspeicher

`add A,B,C mem[A] \leftarrow mem[B] + mem[C]`

Explizite Adressierung, i.e. die Maschineninstruktionen enthalten explizit die drei Hauptspeicheradressen der Operanden und des Resultats. Dies erfordert das Dreiadressformat. Der Nachteil dieses Modells sind die Hauptspeicherzugriffe, was zu Speicherengpass führt.

Beispiel: DEC VAX

2.5 Klassifikation von Rechnerarchitekturen

Seit der VON NEUMANN Architektur sind eine Reihe von Entwicklungen entstanden mit dem Ziel, die Nachteile dieser Architektur zu umgehen. Diese Entwicklung gehen insbesondere in die Richtung der **Parallelverarbeitung**.

Mitte der 60er Jahre des letzten Jahrhunderts wurde von MICHAEL FLYNN eine (grobe) Einteilung der verschiedenen Rechnerarchitekturen dahingehend vorgenommen [33,34], wieviele Daten pro Instruktion verarbeitet werden können.¹³ Man differenziert gemäß dieser **Flynn Taxonomie** vier Architekturen (siehe Abbildung [2.10]).

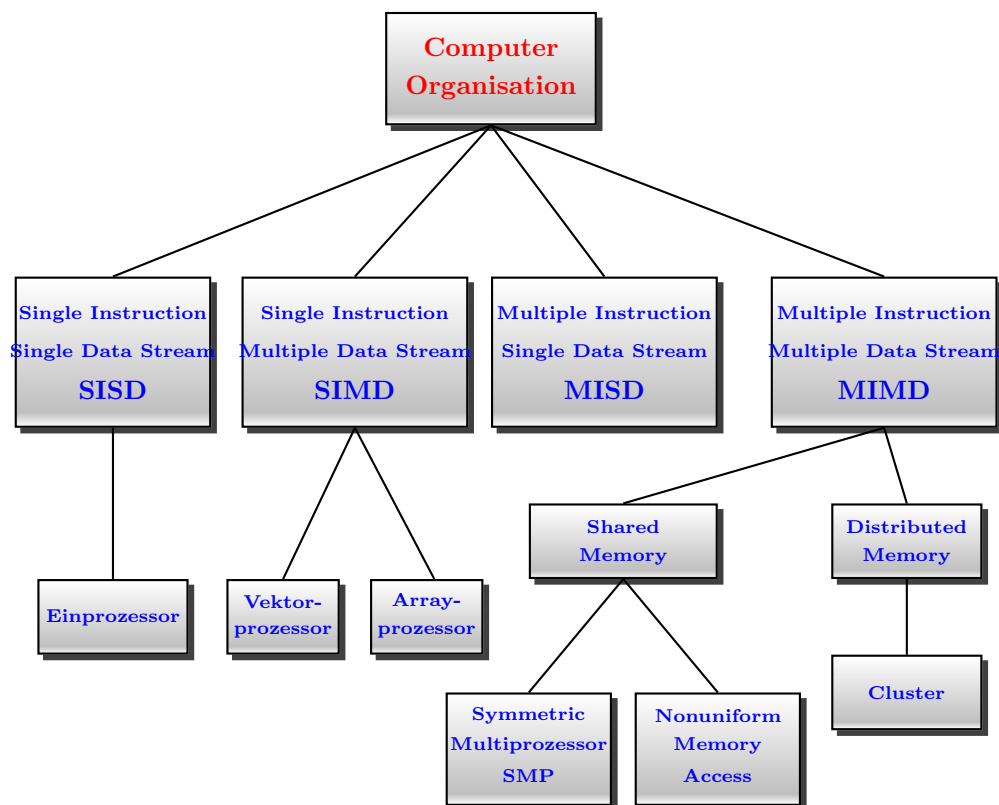


Abbildung 2.10: Taxonomie der Rechner-/Prozessorarchitekturen.

✓ SISD Architektur

SISD steht für *Single Instruction Single Data Stream* und bezeichnet eine Arbeitsweise eines einzelnen Prozessors, der auf Daten operiert, die in

¹³Siehe [73, pp. 48–56] oder [99, pp. 623–635].

einem einzigen Speicher liegen. Hierbei wird pro Instruktion ein Datum verarbeitet. Der klassische VON NEUMANN Rechner ist der SISD Kategorie zuzuordnen. Durch die Ausführung eines Programms erzeugt *ein* Steuerwerk einen Befehlsstrom, dessen Instruktionen sequentiell geholt, decodiert und anschließend von *einem* Rechenwerk ausgeführt werden. Die erforderlichen Operanden werden über einen bidirektionalen Datenstrom vom Hauptspeicher in das Rechenwerk geholt bzw. vom Rechenwerk in den Hauptspeicher geschrieben.

✓ SIMD Architektur

Das Kürzel SIMD steht für *Single Instruction Multiple Data Stream*. SIMD Rechner verfügen — wie ein SISD Rechner — über *ein* Steuerwerk. Der vom Steuerwerk decodierte Befehl kann aber gleichzeitig auf mehrere Operanden in *mehreren* Rechenwerken¹⁴ angewendet werden. Das Steuerwerk kann durch *Instruction Broadcasting* die in einem Maschinenbefehl codierten Informationen über die Art der auszuführenden Operation(en) an alle vorhandenen Verarbeitungseinheiten übertragen. Entscheidend für die SIMD Eigenschaft dabei ist, dass die beteiligten Processing Elements nur von einer Instruktion aktiviert werden und anschließend *gleichzeitig* den Befehl auf unterschiedlichen Datenoperanden ausführen.

Jedes Rechenwerk muß daher über einen separaten bidirektionalen Zugriffspfad auf den Hauptspeicher verfügen, über den die Operanden aus dem Hauptspeicher geholt werden und die Ergebnisse in den Hauptspeicher zurückgeschrieben werden. SIMD Rechner erweitern den klassischen Universalrechner durch die Vervielfachung der Rechenwerke.

Beispiele für die SIMD Architektur sind Vektorrechner und Feldrechner oder auch Array Rechner genannt.

Feldartige Datenstrukturen liegen auch im Bereich von **Multimedia Anwendungen** vor. Um Bilddaten, Videofilme und akustische Sequenzen in Echtzeit verarbeiten zu können, sind sehr hohe Rechenleistungen bei Arbeitsplatzrechnern erforderlich. Moderne Prozessoren — wie beispielsweise Intels Pentium oder der Athlon Prozessor von AMD — verfügen daher über Funktionseinheiten, die solche Aufgaben im SIMD Modus abwickeln. INTEL nennt diese Technik MMX – für **Multimedia Extension**.

Beispiel

Um das grundlegende Prinzip der SIMD Architektur zu verstehen, betrachten die Verarbeitung einer Instruktionssequenz der Form:

```
int x[3], a[3];  
...
```

¹⁴Diese Rechenwerke nennt man in dieser Architektur auch *Verarbeitungseinheiten* oder *Processing Elements*.

```

for(i=0 ; i < 3 ; i++)
    x[i] = x[i] + a[3];
...

```

i.e. die Addition zweier Vektoren $x[]$ und $a[]$. Bei der Abarbeitung der Schleife führt ein SISD Rechner drei separate Additionen hintereinander aus. Bei der Verarbeitung dieses Programmfragmentes mit SIMD Architektur stehen (beispielsweise) drei Additionswerke (für jede Vektor-komponente ein Additionswerk) zur Verfügung.

✓ MISD Architektur

Diese Kategorie der FLYNN Taxonomie kann nur schwer – wenn überhaupt – realen Systemen zugeordnet werden. Eine zweckmäßige Interpretation der Funktionsweise eines MISD Rechners lautet folgendermaßen: In einem MISD Rechner steuern mehrere Leitwerke *gleichzeitig* die Ausführung von Instruktionen aus unterschiedlichen Befehlsströmen (Programme bzw. Tasks).

✓ MIMD Architektur

Das Akronym MIMD steht für *Multiple Instruction Multiple Data Stream*. MIMD Systeme sind mit mehreren vollständigen Prozessoren ausgestattet. Es gibt eine Reihe unterschiedlicher Realisierungen der MIMD Architektur, die man dahingehend differenziert, wie die einzelnen Prozessoren miteinander kommunizieren.

- ☞ Greifen alle Prozessoren auf ein gemeinsames Speicherwerk zu (*shared memory*), – die Prozessoren kommunizieren über den Speicher – so spricht man von **Multiprozessorsystemen** mit enger Kopplung. Hier wiederum differenziert man zwei grundlegende Architekturvarianten. Bei den **symmetric multiprocessor** Systemen teilen sich mehrere Prozessoren einen gemeinsamen Speicher und kommunizieren über ein spezielles Bussystem oder einen anderen Mechanismus. Die charakteristische Eigenschaft solcher Systeme ist, daß die Speicherzugriffszeit auf eine beliebige Speicherzelle für jeden Prozessor etwa gleich groß ist.

Eine neuere Entwicklung ist *nonuniform memory access* Organisation, kurz NUMA . Die Speicherzugriffszeiten der Prozessoren auf verschiedene Speicherbereiche unterscheiden sich in diesen Systemen.

- ☞ Das Speichersystem, auf das die Prozessoren zugreifen, ist nur lose gekoppelt (*distributed memory*), *i.e.* die Prozessoren greifen über feste Datenwege oder ein Netzwerk auf den Speicher zu. Typische Vertreter dieser Klasse sind **Cluster**.

Kapitel 3

RISC und CISC

Mitte der 1970er Jahre hat man untersucht [8, 16], wie viele aller möglichen Grundbefehle, die ein Prozessor versteht, von Programmen auch wirklich genutzt werden. Es waren schlichte 20 Prozent. Diese Untersuchung läutete eine Entwicklung neuartiger Prozessoren ein, die sogenannten RISC-Prozessoren. RISC bedeutet

RISC = Reduced Instruction Set Computing ,

und stellt einen Prozessortyp dar, der — zumindest in der Anfangszeit der RISC-Entwicklung — mit viel weniger Grundbefehlen und reduzierten Befehlsformaten arbeitet als die bis dato entwickelten Prozessoren, die man auch als **CISC-Prozessoren** bezeichnet. CISC steht dabei für

CISC = Complex Instruction Set Computing.

CISC und RISC sind Begriffe, die sich für zwei sehr unterschiedliche Konzepte — und damit auch Philosophien — zum Design der Prozessortechnologie etabliert haben (siehe auch [28]). Diese Begriffe geben die auffälligsten Attribute wieder, die bei der Publizierung der ersten RISC-Konzepte erkennbar waren. Hierzu gehörte die deutliche Reduzierung der Anzahl der Befehle. Die Attribute *komplex* und *reduziert* beschreiben jedoch nur sehr oberflächlich den Unterschied zwischen diesen beiden Architekturmodellen für Mikroprozessoren. So läßt sich nicht eindeutig eine Anzahl von Prozessorbefehlen angeben, unterhalb der man es mit RISC zu tun hat. Vielmehr gehört eine Vielzahl anderer Wesensmerkmale dazu, um typische RISC und CISC – Prozessorarchitektur zu kennzeichnen. Auch gibt es diverse Prozessoren, die sich nicht eindeutig einer Kategorie zuordnen lassen. Recht gut geben die Begriffe komplex und reduziert jedoch ein wichtiges Unterscheidungsmerkmal wieder, wenn man sie nicht nur auf die

Befehle, sondern darüber hinaus auf die Komplexität der Prozessor Hardware bezieht.

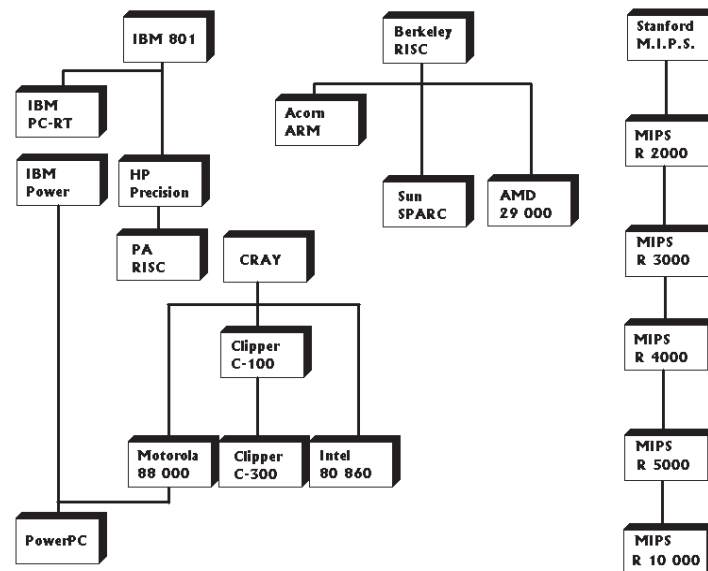


Abbildung 3.1: Wichtige RISC Prozessorfamilien.

Bei Verwendung einer vergleichbaren Halbleitertechnologie und gleicher Taktrate (zum Beispiel CMOS¹ und 25 MHz) erreicht ein RISC-Prozessor typischerweise die zwei- bis vierfache Leistung eines CISC-Prozessors. Die Prozessor-Hardware eines derartigen RISC-Prozessors ist aber so einfach strukturiert (reduziert), dass sie auf einem Bruchteil der Chipfläche des CISC-Prozessors realisiert werden kann.

3.1 CISC-Architekturen bei Mikroprozessoren

Wichtiges Wesensmerkmal fast aller CISC-Architekturen, z. B. der Prozessoren der Intel- (8086, 80186, 80286, 80386, 80486) und Motorola-Familien (68000, 68010, 68020, 68030, 68040), ist die sogenannte **Mikroprogrammierung**.

Hierbei handelt es sich um eine strukturierte Lösung des Aufbaus von Prozes-

¹CMOS steht für *Complementary Metal Oxide Semiconductor* (=komplementärer Metalloxid – Halbleiter) und bezeichnet eine integrierte Schaltung, die besonders stromsparend arbeitet und wenig Verlustwärme erzeugt. CMOS Bausteine sind eine Variante der MOS Technologie.

soren. Jeder Maschinenbefehl wird durch ein sogenanntes **Mikroprogramm** interpretiert, das sich in einem Mikroprogrammspeicher (ROM) direkt auf dem Prozessorchip befindet.

Um die Entwicklung von CISC und RISC Prozessoren zu beleuchten, ist es zweckmäßig, etwas auszuholen. Die ersten digitalen Computer in den vierziger und fünfziger Jahren hatten zwei architektonische Ebenen:

- die *Instruction Set Architecture* – Ebene (ISA), in der die komplette Programmierung erfolgte,
- die *digitale logische Ebene* — das ist die Hardware mit den logischen Schaltkreisen — die die Programmanweisungen interpretierte und ausführte.

Folge: Hochkomplexe Schaltungen, die die digitale logische Ebene realisierte (sehr komplizierte Hardware).

1951: MAURICE WILKES (University of Cambridge)

Entwicklung eines Computers mit drei Ebenen mit dem Ziel, die Hardware zu vereinfachen:

- Die ISA Ebene,
- eine Interpreter-Schicht, die aus einem nicht-austauschbaren Interpreter besteht. Diesen nennt man heute *Mikroprogramm*. Die Aufgabe dieser Schicht ist, die Programme aus der ISA–Ebene zu interpretieren und auszuführen,
- die digitale logische Ebene.

Resultate:

- Die Hardware muß nur Mikroprogramme mit begrenzter Instruktionsmenge ausführen, nicht die Programme der ISA-Ebene, die einen viel größeren Anweisungsvorrat haben. → Weniger elektronische Schaltkreise.
- Zu dieser Zeit waren Arbeitsspeicherbausteine sehr teuer und die Zugriffe auf den Speicher sehr langsam. Die Architektur mit Mikroprogrammen kam diesen Zwangsbedingungen dahingehend zugute, dass ISA-Instruktionen sehr komplex gestaltet werden konnten. Mit einem Ladevorgang wurde eine solche Instruktion in die CPU geladen und durch ein entsprechendes Mikroprogramm interpretiert.

Nachdem sich die Mikroprogrammierung (~ 1970) durchgesetzt hatte, erkannten die Entwickler, dass sich allein durch Erweiterung des Mikroprogramms neue Instruktionen hinzufügen ließ (siehe dazu auch [61]). Anders ausgedrückt: *Hardware (neue Maschineninstruktionen) ließ sich durch Programmierung und Implementierung von Mikroprogrammen hinzufügen.*

Konkret bedeutet Mikroprogrammierung also die Verlagerung von komplexen Befehlsabläufen aus dem Anwenderprogramm auf kleine Programmroutinen (Mikroprogramme), die auf dem Prozessorchip in ROM Bausteinen implementiert sind bei gleichzeitiger Reduzierung der Hardwarekomplexität.

Die Folge dieser Erkenntnis war, dass den Anwenderprogrammen mächtige, sehr komplexe Befehlssätze zur Verfügung gestellt wurden, zum Beispiel:

- Instruktionen für die Multiplikation und Division von Ganzzahlen
- Instruktionen für die Gleitkommaarithmetik
- Instruktionen zur Manipulation von Zeichenketten
-

Die Mikroprogrammierung war also Ende der 60er Jahre durchaus die folgerichtige Lösung, um unter Anwendung aller verfügbaren Technologien ein entwicklungsfähiges Prozessorkonzept mit optimalen Leistungsdaten zu verwirklichen.

Auf diese Weise entstanden Prozessorfamilien mit einer enormen Anzahl von Maschinenbefehlen (bis über 300), deren Befehlsstruktur stark codiert und deren Formate sehr uneinheitlich (16 bis 456 Bit) waren. Man erzielte damit einen sehr kompakten Code, der wenig Platz im Hauptspeicher benötigte. Damit wurde dem Umstand Rechnung getragen, daß Kernspeicher damals die langsamsten und teuersten Elemente der Rechner waren. Auf Basis dieses Grundkonzepts entstanden Mikroprozessorfamilien, die zu Standards in der Computertechnik wurden. In der Folgezeit wurde eine große Anzahl von Softwarepaketen entwickelt, die auf diesen Prozessoren lauffähig waren.

Heutige Situation:

Im Laufe der Entwicklung des Computers wurde immer deutlicher, daß die Komplexität der Hardware abgeschirmt werden mußte. Der Step-by-Step Prozeß führte letztendlich dahin, auf die reine Hardware eine Software-Schicht zu legen. Deren Aufgabe ist die Verwaltung aller Teile des Systems und außerdem wird dem Benutzer eine Schnittstelle – oder *virtuelle Maschine* – angeboten, die um vieles einfacher zu handhaben und zu programmieren ist.

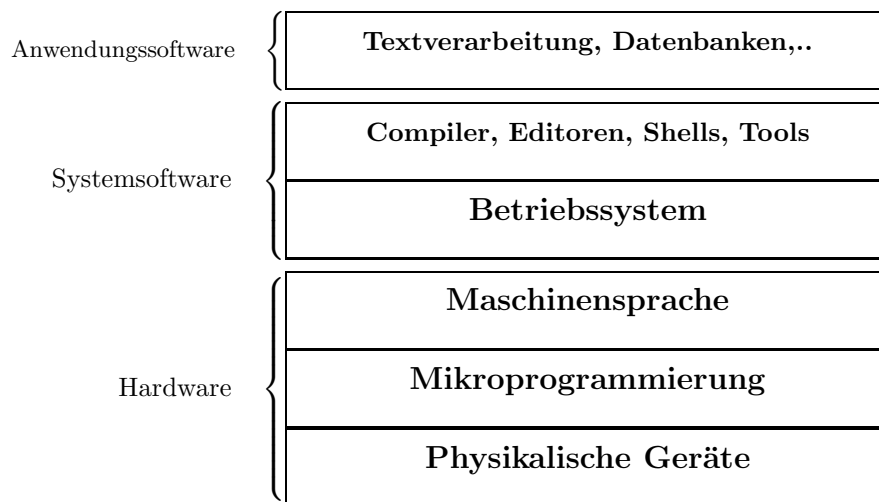


Abbildung 3.2: Struktureller Aufbau eines Rechensystems, bestehend aus Hardware, Systemsoftware und Anwendungsprogrammen.

In der Abbildung [3.2] ist der Aufbau einer Rechenanlage schematisch dargestellt. Die unterste Schicht wird von den physikalischen Geräten gebildet, wie z.B. integrierte Schaltkreise, Drähte, Stromversorgung etc. Die nächste Schicht besteht aus einfacher Software, deren Aufgabe in der direkten Steuerung und Kontrolle der Geräte besteht. Außerdem stellt diese eine Schnittstelle für die darüberliegende Schichten zur Verfügung. Diese Software heißt

Mikroprogramm

und ist üblicherweise in ROM – Bausteinen auf der CPU implementiert.

Die Menge der Instruktionen, die das Mikroprogramm interpretieren kann, definiert die

Maschinensprache

der entsprechenden Rechnerplattform.

3.2 Das RISC-Konzept

Mitte der 70er Jahre wurden bei statistischen Untersuchungen im IBM Thomas J. Watson Forschungszentrum in Yorktown Heights folgende Beobachtungen gemacht:

- Was die Häufigkeit der Verwendung und den Zeitaufwand für einen Befehl betrifft, so belegen bei einem CISC-Prozessor etwa 20 Prozent der Befehle 80 Prozent der gesamten Ausführungszeit eines Programms.
- Es gibt Beispiele, in denen eine Sequenz einfacher Befehle das gleiche bewirkt wie eine vorgegebene komplexe Sequenz, aber eine geringere Ausführungszeit benötigt.

Außerdem war zu diesem Zeitpunkt schon absehbar, dass die Fortschritte in der Halbleitertechnologie den Geschwindigkeitsunterschied zwischen CPU und Hauptspeicher reduzieren. Auch war bekannt, dass sich dieser Unterschied durch Verwendung schneller Cache-Speicher mit Hilfe sogenannter SRAM Bausteinen weiter verringern lässt. Diese Erkenntnisse führten zum IBM-801-Projekt mit der Entwicklung des ersten RISC-Prozessors, der jedoch als Forschungsprojekt nicht in kommerzielle Serienfertigung ging.

Die zunehmend positive Veränderung der realisierungstechnischen Rahmenbedingungen führte 1980/81 zu RISC-Projekten an den Universitäten Berkeley und Stanford, wo **David Patterson** ([77]) und **John L. Hennessy** begannen, mit ihren Teams RISC-Prozessoren als Ein-Chip-Rechner zu entwickeln. Siehe auch [51, 78].

Seit Mitte der 80er Jahre gibt es kommerzielle Implementierungen des RISC-Konzepts. Eine dieser Architekturen ist die von MIPS² auf Basis des Stanford-Projekts entwickelte Prozessorfamilie R2000/3000. Die MIPS-Prozessoren wurden von Silicon Graphics in Unix-Workstations (z. B. SGI Indigo2) und Unix-Servern (z. B. SGI Origin2000) eingesetzt. Früher boten auch andere Workstation-Hersteller wie z. B. die Digital Equipment Corporation (DEC) Maschinen mit MIPS-Prozessoren an, so z. B. die DECstation-Familie (2100, 3100, 5000) und die DECsystem unter dem Betriebssystem Ultrix. Beispielsweise bestückten Siemens bzw. SNI ihre Server der RM-Serie mit MIPS-Prozessoren der R4000-, R5000- und R10000-Familie.

MIPS-Prozessoren werden auch häufig in eingebetteten Systemen eingesetzt. Dazu zählen z. B. Cisco-Router, SUNs Cobalt-Server bis RaQ/Qube2, BMW-Navigationssysteme, die Fritz!Box, Satellitenreceiver, Dreambox³, Konica Minolta DSLRs⁴ und Sony- und Nintendo-Spielkonsolen.

Verschiedene RISC-Entwicklungen sind in Abbildung [3.1] dargestellt.

²Das Akronym MIPS steht für *Microprocessor without interlocked pipeline stages*.

³DVB Receiver mit Linux Betriebssystem

⁴Digitale Spiegelreflexkamera.

Generell lassen sich die Wesensmerkmale einer RISC-Architektur wie folgt zusammenfassen:

- **Pipelining** (überlappende Ausführung mehrerer Befehle)
Die logischen Stufen der Ausführungseinheit(en) – das sind die *Pipelines* – müssen klar definiert sein und optimal gegeneinander ausbalanciert sein (siehe [51], Chapter 6.). Die Einzelschritte der Pipeline sollten sämtlich die gleiche Zeit beanspruchen und die gesamte Verarbeitung in einer Pipeline sollte gleichmäßig auf alle Stufen verteilt sein. Die Abarbeitung in jeder Pipeline-Stufe benötigt einen kompletten Taktzyklus. Typische RISC-Prozessoren benutzen Pipelines mit vier oder fünf Stufen, mehr Stufen bedeutet höhere Parallelverarbeitung aber auch ein höheres Koordinationsproblem.
- **Reduzierter Befehlssatz**
Reduzierung des Befehlssatzes auf einfache Basisbefehle, aus denen sich alle komplexen Operationen zusammensetzen lassen;
- **Register-Register-Modell**
Die Ausführphase einer Instruktion sollte wenn immer möglich nicht länger als einen Taktzyklus beanspruchen. Arithmetische Instruktionen, die Operanden aus dem Arbeitsspeicher benötigen, erfüllen diese Bedingung aufgrund der hohen Latenzzeit von Arbeitsspeicherzugriffen nicht. Sogenannte *Register - Register - Operationen* – i.e. Operationen, die auf Operanden wirken, die in Registern gespeichert sind – vermeiden diese Wartezeiten, denn auf Register kann innerhalb einer Taktzykluszeit zugegriffen werden.
- **Load-Store-Architektur**
Wenn alle Operanden für logische und arithmetische Operationen in den Registern lokalisiert sind, müssen die Register zuvor mit den notwendigen Daten geladen werden. In RISC-Prozessoren wird dies durch eine *load*-Instruktion ausgeführt, die auf Bytes, Half-WORDS (16 Bits) oder WORDs (32 Bit) zugreifen kann. Eine *store*-Anweisung transferiert den Registerinhalt in den Arbeitsspeicher.
- **Multi-Purpose Register**
RISC-Prozessoren verfügen über eine hohe Anzahl von Allzweckregistern
- **einheitliches festes Instruktionsformat**
Bei den CISC-Prozessoren, wie die VAX, haben die Instruktionen eine variable Länge. Aufgrund dessen müssen mehrere Worte aus dem Arbeitsspeicher geladen werden, bis die komplette Instruktion decodiert werden kann. Dies führt ein variables Element in der Dauer der Instruktions-Fetch-Phase ein, die die Pipeline zum Stillstand zwingt, falls die Dekodier-Stufe auf eine Instruktion warten muß. Verschiedene Prozessoren umgehen dieses Problem mit Hilfe eine *Prefetch-Buffers*, der eine Reihe von Instruktionen vor Bedarf zwischenlagern kann.

Die einfachste Technik, eine variable Fetch-Zeit zu vermeiden, besteht darin, die Instruktionen in einem festen WORD-Format zu kodieren. Dann hat die Fetch Stufe eine definierte Zeit, die sie benötigt, um ausgeführt zu werden und kann so zum Beispiel nach einem Taktzyklus eine Instruktion an die Dekodierstufe der Pipeline weitergeben.

- fast alle Befehle können in einem Maschinenzklus abgeschlossen werden – dieses Feature bildet eine wichtige Grundlage für die Reorganisation der Befehlsabläufe durch einen optimierenden Compiler;

Die Tatsache, dass die einfachere Chipstruktur bei einem RISC-Prozessor zu erheblich kleineren Chipflächen führt, wird oft genutzt, um zusätzliche Funktionen auf dem Chip unterzubringen (z. B. eine Einheit zur Fließkommaarithmetik, Memory-Management-Einheit, Cache-Kontrollfunktionen u. a.). Außerdem führt die relative Einfachheit der Architektur von RISC-Prozessoren zu kürzeren Designzyklen bei der Entwicklung neuer Versionen. Dies wiederum macht es möglich, dass jeweils der neueste Stand der Halbleitertechnologie zum Einsatz kommen kann. RISC-Prozessoren setzen damit nicht nur auf einer um Faktor 2 bis 4 höheren (System-) Leistung an, sondern machen auch von Generation zu Generation erheblich größere Leistungssprünge als CISC-Prozessoren. Eine Faustregel besagt, dass aktuell bei modernen RISC-Prozessoren etwa alle zwei Jahre eine Leistungsverdopplung mit neuen Prozessor-Versionen möglich ist.

Es ist für die Leistung eines Computersystems entscheidend, dass die einzelnen Komponenten des Gesamtsystems optimal ausbalanciert werden. Dies gilt auch für RISC - Architekturen. Während bei CISC - Systemen aus ökonomischen Gründen die Komplexität auf den Prozessorchip verlagert wurde, wird bei RISC - Konzepten der Prozessor von den komplexen Befehlen befreit. Durch die Verfügbarkeit großer, preiswerter Speicherbausteine in CMOS-Technologie (1 MBit, 4 MBit) mit Zugriffszeiten von unter 60 ns, SRAM-Bausteinen für Cache-Speicher mit Zugriffszeiten von unter 15 ns und Gehäusetechnologien, die standardmäßig Pinzahlen über 120 ermöglichen, haben sich in der zweiten Hälfte der 80er Jahre die realisierungstechnischen Bedingungen für RISC-Architekturen wesentlich verbessert.

Eine Schlüsselrolle für ein ausbalanciertes RISC- Konzept kommt dem Compiler (Übersetzer) zu. *Zugunsten der optimalen Beschleunigung der wichtigsten Basisfehle durch feste Verdrahtung auf dem Prozessorchip werden alle komplexen Operationen in den Compiler verlagert.* Der Compiler übernimmt die Rolle des intelligenten Mittlers zwischen Anwenderprogramm und Mikroprozessor. Hierbei werden unterschiedlichste Optimierungstechniken auf verschiedenen Software-Hierarchieebenen eingesetzt, um den Prozessor so mit Daten zu füttern, daß dieser optimal beschäftigt ist.

Die RISC-Entwickler bei MIPS und Hewlett-Packard arbeiteten z. B. nach der Regel, dass ein Befehl nur dann aufgenommen, d.h. auf dem Prozessorchip fest verdrahtet wird, wenn dies zu einer Steigerung der gesamten Systemleistung um

mindestens 1 Prozent führt.

Bei MIPS wurde die Systemeinheit Compiler und Prozessorchip in einem Gesamtprogramm entwickelt. Ergebnis war ein Satz von nur 74 Befehlen, wobei auf möglichst einheitliches Befehlsformat geachtet wurde. Hier ist der Prozessor (Hardware) ein Produkt der Compiler Entwicklung (Software). Im Vergleich dazu wurden und werden CISC- und auch manche RISC-Prozessoren typischerweise von Chipentwicklungs-Teams der Halbleiterfirma entwickelt, während die Compiler-Entwicklung oftmals abgetrennt nachfolgt. Die Compiler-Entwickler finden also eine festgelegte Schnittstelle zum Prozessor vor und können die Aufgabenteilung zwischen Prozessor und Compiler nicht mehr optimierend beeinflussen.

Darüber hinaus neigen Softwarehäuser, die Compiler und Anwendersoftware entwickeln, aus Wirtschaftlichkeitsgründen dazu, ihre Software auf verschiedene Prozessorchips zu portieren, statt eine volle Optimierung unter Nutzung der jeweiligen Prozesseigenschaften durchzuführen. Dies erzeugt weitere negative Faktoren der Ineffizienz. Diese (oft auch durch Kompatibilitätszwang vorgegebene) Einschränkung der Möglichkeiten zur Systemoptimierung wurde bei modernen RISC-Entwicklungen überwunden.

3.2.1 Anwendung für RISC-Prozessoren

Die seit 15 Jahren verwendeten CISC-Architekturen haben die Entwicklungen einer großen Anzahl von Softwareprodukten ermöglicht. Dies stellt eine gewaltige Investitionssumme dar und sichert diesen Prozessorfamilien einen wachsenden Markt. Daneben nehmen jedoch Anwendungen zu, bei denen die erreichbare Systemleistung aus Anwendersicht einen größeren Stellenwert einnimmt als die Kompatibilität zu älterer Hard- und Software. Neben Hochleistungs-Subsystemen im Embedded-Bereich, wo schon immer spezielle Lösungen mit höherer Leistung dominieren, gilt dies auch für Workstations.

Diese Geräte waren bislang überwiegend im technischen Bereich von Forschung und Entwicklung zu finden. Immer größer und komplexer werdende Softwarepakete mit hohem Bedarf an visueller Umsetzung verhalfen dieser Geräteklasse nun auch zum Durchbruch in anderen Bereichen, z. B. in Büro-, Medizin- und Banken Anwendungen. Die Anwendersoftware läuft dabei unter dem Betriebssystem Unix. Unix ist in C geschrieben und vorhandene RISC-Architekturen sind für diese Hochsprachenumgebung angepaßt und optimiert. Dies führte binnen weniger Jahre dazu, daß alle namhaften Workstation-Hersteller von CISC- zu RISC- Prozessoren übergingen. Die resultierenden Systeme bieten in der Bauform eines großen PCs oft die Leistung eines Großrechners. Während schon der gesamte Markt der 32-Bit-Mikroprozessoren ein jährliches Wachstum von etwa 100 Prozent aufweist, wächst der RISC-Anteil dieser Sparte sogar mit über 150 Prozent. Innerhalb weniger Jahre wird RISC im 32-Bit-Markt 25 bis 30 Prozent Marktanteil erobern haben – trotz der scheinbar erdrückenden Menge an Soft-

ware, die auf Basis von Standard-CISC-Prozessoren weltweit verkauft wurde.

Innerhalb der RISC-Welt ringen unterschiedliche Architekturen um die Vorherrschaft. Während bei Embedded Systems auch in den nächsten Jahren Raum für unterschiedlichste, z. T. exotische Konzepte bleiben wird, da dort die normierende Wirkung des Unix-Betriebssystems kaum greift, zeichnet sich im Bereich der Reprogrammable Systems, und hier besonders im Workstation-Bereich, eine Dominanz weniger, allen Anwendern zugänglicher Prozessorarchitektur ab. Die MIPS-RISC- Architektur hat hier die stärkste Akzeptanz gefunden.

3.2.2 RISC oder CISC aus anwendungstechnischer Sicht

Die Entscheidung für oder gegen eine bestimmte Prozessorarchitektur muß jeder Anwender unter Berücksichtigung seiner konkreten Applikation treffen. Dies gilt bei der Entscheidung für eine bestimmte CISC- oder RISC-Architektur genauso wie für die Entscheidung, ob sondern immer die erreichbare reale Systemleistung den Ausschlag geben. Auch die Kostenfrage wird wahrscheinlich sehr unterschiedlich zu beurteilen sein. Für den Anwender, der eine Workstation entwickelt, die anschließend für 100.000 Mark verkauft werden soll, sind die Kosten für den Prozessor sekundär, wenn sie unter 500 Mark liegen; er wird sich ausschließlich an der Leistungsfähigkeit orientieren. Ein Anwender dagegen, der einen Laserdrucker entwickelt und ihn für 500 Mark verkaufen möchte, wird an die zu wählende Prozessorarchitektur andere Maßstäbe anlegen. RISC wird überall dort eine attraktive Lösung bieten, wo hohe Leistungsanforderungen und Hochsprachenorientierung auftreten. Im industriellen Bereich gibt es jedoch noch eine große Zahl von Anwendungen, die nicht einmal die Leistungsfähigkeit heutiger 8-Bit-CISC-Controller ausschöpfen. Obwohl der Anwendungsbereich für Hochleistungs-RISC- Architekturen stark wächst, ist dies nicht gleichbedeutend mit dem Ende für andere bewährte Prozessor- und Controllerarchitekturen.

Die existierenden CISC - Standard - Prozessorfamilien werden sicher weiterentwickelt und weiterhin weltweit eingesetzt. Fraglich ist jedoch, ob noch völlig neue CISC- Familien entstehen werden. Inzwischen haben auch Intel und Motorola, die beiden großen Hersteller von CISC- Prozessoren, erkannt, daß RISC-Prozessoren weit mehr sind als nur eine Spielart von Spezialprozessoren. Durch Adaptieren von RISC-typischen Prozesstechniken bei neuen Versionen von CISC-Prozessoren wird versucht, neue Wege der Leistungssteigerung auch bei eingeführten CISC-Prozessorfamilien zu finden. Inzwischen haben RISC - Prozessoren die Domäne der Motorola - 68000 Prozessoren im Workstation-Bereich erobert. Es ist zu erwarten, daß bald im oberen PC-Bereich RISC- Prozessoren die Intel-Architektur bedrängen. Es zeigt sich jedoch, daß nur dort, wo sich neue Standards im Gerätebereich herausbilden, wie mit Unix im Workstation - Bereich, genügend Raum für den hochvolumigen Einsatz von neuen, nicht mit älteren Geräten kompatiblen Systemen entsteht. Die Marktentscheidungen fal-

len auf der Anwenderseite, und hier spielt die Software eine weit größere Rolle als für den Endanwender nicht sichtbaren Architekturunterschiede.

3.2.3 Kommerzielle RISC - Plattformen

Typische RISC-Rechner sind zum Beispiel die Sparcstation der Firma SUN oder die Workstation RS/6000 von IBM. Die neueren Entwicklungen der RISC Prozessoren sind der R4000-Chip von MIPS, die RS6000 Familie von IBM und der Alpha-Chip von Digital Equipment Corporation (DEC). Dieser RISC-Prozessor wird in dem Alpha-PC von DEC eingesetzt. Der Alpha-PC ist eine 1993 von DEC vorgestellte Workstation, die das Betriebssystem WINDOWS NT einsetzt. Ein Alpha-PC ist etwa 1,5 bis 2 mal schneller als ein Pentium-PC. Der Alpha-PC arbeitet mit einem 64-Bit breiten Datenbus und ist mit 150 MHz getaktet.

Hersteller	Workstation	Prozessor	Architektur
Control Data	Cyber 910-600	R 3000	MIPS
Data General	AVION 300	88 000	Motorola
DEC	DS 2100, DS 3100	R 2000	MIPS
Everex Systems	Everex	88 000	Motorola
HP/Apollo	HP 9000/S800	Precision	HP/Apollo
IBM	Serie 6000	Power	IBM
Intergraph	InterPro	Clipper	Intergraph
PCS	Cadmus 9000/RC	R 2000	MIPS
Siemens	WS-30-1000	Prism	Apollo
Silicon Graphics	Serie 4D	R 3000	MIPS
Sony	Net-Workstation	R 3000	MIPS
Sun	SPARCStation	SPARC	SUN
Tatung	VARstation 1	SPARC	SUN
Tektronix	XD 88/10	88 000	Motorola

Tabelle 3.1: Einige Workstations mit RISC Prozessoren.

Die mit RISC-Prozessoren ausgestatteten Workstations arbeiten sehr häufig mit dem Betriebssystem UNIX.

Eine der neueren Entwicklungen auf dem Prozessormarkt (Mitte '94) ist eine Gemeinschaftsproduktion der Firmen IBM, Apple und MOTOROLA, und heißt PowerPC. Das Ziel dieser Kooperation war die Entwicklung eines Prozessors, der nicht auf ein bestimmtes Betriebssystem festgelegt ist, wie zum Beispiel die INTEL 80x86-Serie auf das Betriebssystem DOS. PowerPC ist mal wieder eine Abkürzung und steht für *Performance Optimization with Enhanced RISC*, was soviel wie Leistungsoptimierung durch verbesserte RISC-Architektur bedeu-

tet. Auf diesem PowerPC lassen sich andere Plattformen wie eben den INTEL- und MOTOROLA-Prozessor nachahmen. Man nennt dies *Emulation*. Mit anderen Worten, dieser PowerPC ist also in der Lage, Programme zu verarbeiten, die auf einem IBM-PC lauffähig sind, als auch Programme, die für einen Apple-Computer konzipiert sind. Die ersten PowerPCs heißen Power Macintosh 6100/60 und sind mit 60 MHz getaktet.



Abbildung 3.3: SGI Octane Workstation

- ARM (Advanced Risc Machines)

Die ARM-Architektur ist in Stückzahlen gemessen wohl die erfolgreichste RISC-Familie, sie findet sich in vielen Systemen,⁵ bei denen es um relativ hohe Leistung, geringen Stromverbrauch und niedrige Kosten geht (typisch: 100 – 500 MHz). Die ARM Ltd., die diese Systeme konstruiert, baut allerdings selbst keine Prozessoren, sondern verkauft lediglich Lizenzen für das Design an ihre Kunden. Mittlerweile sollen 10 Milliarden ARM-CPU's im Umlauf sein, die z. B. zum Einsatz kommen in Tablets, Digitalkameras, grafikfähigen Taschenrechnern, NAS,⁶ Routern, Spielkonsolen,

⁵

- Apple: iPods (ARM7TDMI SoC), iPhone (Samsung ARM1176JZF, Apple A4), iPod touch (ARM11), iPad (Apple A4 bzw. Apple A5)
- Canon: IXY Digital 700 Kamera (Eigenentwicklung, ARM-basiert)
- Hewlett-Packard: HP-49/50 Grafikfähiger Taschenrechner (ARM9TDMI)
- Linksys: NSLU2 Netzwerkspeicher/NAS [Intel XScale IXP420]
- Nintendo: Game Boy Advance (ARM7), Nintendo DS (ARM7, ARM9)
- Palm: PocketPC PDAs und Smartphones (Intel XScale und Samsung SC32442 ARM9)
- Sony: verschiedene Mobiltelefone, Network Walkman (Eigenentwicklung, ARM-basiert)

⁶NAS steht für Network Attached Storage.

PDAs, Smartphones und verschiedenen Mobiltelefonen. Einen Einsatz für energiesparende Server wird für die nahe bis mittlere Zukunft angestrebt. Deshalb hat ARM erste Pläne für eine 64-Bit-Architektur vorgestellt.⁷

- **Power Architecture**
Eine Entwicklung von IBM und Freescale (früher Motorola), ist heute der am weitesten verbreitete RISC Prozessor im HighEnd-Bereich. Sie ist eine Architektur mit zahlreichen Einsatzgebieten, angefangen bei leistungsstarken eingebetteten Systemen wie Druckern oder Routern, über Workstations, bis hin zu Supercomputern.
- **MIPS**
Anfangs wurden diese CPUs vor allem in klassischen Workstations und Servern eingesetzt, heute liegt der Haupteinsatzbereich, ähnlich wie bei ARM, im Bereich Eingebettete Systeme. Praktisch alle MIPS-basierten Workstation- und Server-Familien wurden mittlerweile auf Intel Itanium migriert.
- **SPARC**
Oracles (ehemals Sun Microsystems) SPARC-Produktlinie wurde vor allem in klassischen Workstations und Servern von Sun eingesetzt. Die Spitze der TOP500-Liste der schnellsten wissenschaftlichen Rechner wird heute (TOP500 11/2011) von dieser Plattform angeführt.
- **Hewlett-Packards PA-RISC**
Bis zur Einführung des Intel Itanium wurden PA-RISC-CPU's vor allem in klassischen Workstations und Servern von HP eingesetzt. Die CPU-Familie wird nicht mehr weiterentwickelt. Praktisch alle PA-RISC-basierten Workstation- und Server-Familien wurden mittlerweile auf Intel Itanium migriert.
- **DEC Alpha**
Bis zur Einführung der Intel Itanium-CPU's wurden Alpha-CPU's vor allem in klassischen Workstations und Servern von Digital, Compaq und HP eingesetzt. Die Alpha-Plattform war frei verfügbar und wurde von zahlreichen OEM-Partnern genutzt. Die CPU-Familie wird nicht mehr weiterentwickelt. Praktisch alle Alpha-basierten Workstation- und Server-Familien wurden mittlerweile auf Intel Itanium migriert.
- **SuperH**
Hitachis SuperH, war weit verbreitet z. B. in Segas Super 32X-, Saturn- und Dreamcast-Konsolen. SuperH wird heute ähnlich wie die ARM-Plattform hauptsächlich in eingebetteten Systemen eingesetzt (z.B. Daimler-Benz).

⁷Siehe Heiseticker Nov. 2011 unter der URL:

<http://www.heise.de/ix/meldung/ARM-blaest-zum-Angriff-auf-64-Bit-Server-1368660.html>

- Atmel AVR wird in eingebetteten Systemen eingesetzt wie z. B. Xbox-Steuerkontrollern aber auch in BMW-Automobilen.
- OpenRISC
Das OpenRISC-Projekt greift die Philosophie freier Hardware auf. Ziel des Projektes ist es, ein 32-Bit-Ein-Chip-System zu erstellen, auf dem das Betriebssystem Linux läuft und das – im Sinne freier Software – frei verfügbar ist.

Kapitel 4

Repräsentation der Informationen

Wir wissen bereits, dass Informationen durch Zeichen dargestellt werden, dies wollen wir nun etwas näher konkretisieren, insbesondere interessieren wir uns dafür, wie in einem Computer Informationen dargestellt werden. Ziel dieses Kapitels ist daher zu untersuchen, wie das Prinzip der VON NEUMANN Architektur

Daten werden binär codiert

in Computern umgesetzt wird.

4.1 Zeichen

Zeichen sind Elemente einer Menge, die zur Darstellung von Informationen vereinbart wurden. Nun versteht ein Computer in der Regel keine Zeichen wie rote oder grüne Ampeln. In der Informationsverarbeitung werden also andere Zeichen verwendet.

Generell unterscheidet man prinzipiell vier Arten von Zeichen:

1. Buchstaben: A, B, C, ..., Z, a, b, c, ... , z
2. Ziffern: 0, 1, ... , B-1 (B = Basis)
3. Sonderzeichen
 - für arithmetische Operationen: +, -, *, /
 - Interpunktionszeichen: ; , : . ! ? usw.
 - sonstige: #, %, &, | usw.
 - landesspezifische Zeichen, wie ä, ö, ü, ß, ø, ç, œ, å,

4. Nicht darstellbare Zeichen — diese nennt man mitunter auch nicht-druckbare Zeichen oder Steuerzeichen — wie zum Beispiel CR = Carriage Return, was soviel wie Wagenrücklauf bedeutet (eine Anleihe von der guten alten Schreibmaschine) oder LF = Line Feed, was für Zeilenvorschub steht.

Koppelt man einzelne Zeichen zur Darstellung irgendwelcher Informationen, dann entsteht eine Zeichenfolge, ein sogenanntes **Wort** (engl. *word*).

4.2 Alphabete und Daten

In diesem Abschnitt sehen wir uns die Definitionen von Zeichenvorrat und Alphabet etwas formaler an.

Wie bereits bekannt, nennt man eine vereinbarte Zeichenmenge einen *Zeichenvorrat*. Zwei Beispiele sind die Folgen der Buchstaben:

a, b, ... , ä, ö, ü, A, B, C, für deutsche Texte

oder

a, b, ... für englische Texte.

Die folgende Definition formalisiert den Begriff des Zeichenvorrats und des Alphabets.

Definition 4.4:

A sei eine Menge

- a) Eine endliche Menge A ; *i.e.* eine Menge der Mächtigkeit $|A| = n \in \mathbb{N}$ heißt **Zeichenvorrat**.
- b) Das Paar (A, \Leftarrow) heißt **Alphabet**, wenn A ein Zeichenvorrat ist und wenn \Leftarrow eine auf A erklärte totale Ordnung ist.

Beispiel 4.7:

Zeichenvorräte sind

- ① die Menge der Dezimalziffern $\mathbb{D} := \{0, 1, 2, \dots, 9\}$,
- ② die Menge der Zeichen auf einer PC – Tastatur,

- ③ die Buchstaben des deutschen Alphabets samt Umlauten,
 - ④ die Menge der griechischen Buchstaben $\alpha, \beta, \gamma, \delta, \dots$,
 - ⑤ die binären Zeichenvorräte wie $\mathbb{B} = \{0, 1\}$ oder $\{\text{TRUE}, \text{FALSE}\}$,
 - ⑥ die Menge der Ampelfarben $\{\text{Rot}, \text{Gelb}, \text{Grün}\}$,
 - ⑦ die Menge $\{\bullet, -\}$, wobei das Zeichen \bullet für 'kurz' und das Zeichen $-$ für 'lang' steht. Dieser Zeichenvorrat liegt der MORSE Codierung zugrunde.
-

Beispiel 4.8: Ein **Alphabet** ist nach obiger Definition ein Zeichenvorrat mit einer darauf definierten totalen Ordnung. Üblicherweise wird in der Praxis nicht streng zwischen Zeichenvorrat und Alphabet unterschieden.

- ① Die Dezimalziffern \mathbb{D} mit der Ordnung *kleiner gleich, i.e.* das Paar $\{\mathbb{D}, \leq\}$ ist ein Alphabet, denn $0 \leq 1 \leq 2 \dots \leq 9$ ist eine auf \mathbb{D} erklärte totale Ordnung.
- ② Ein weiteres Alphabet bildet die Menge der kleinen lateinischen Buchstaben a, b, c, \dots, z zusammen mit der konventionellen Ordnung $a \Leftarrow b \Leftarrow c \dots \Leftarrow z$, wobei das Zeichen \Leftarrow für die Ordnungsrelation 'steht vor' steht.
- ③ Für den binären Zeichenvorrat \mathbb{B} , bestehend aus den beiden Ziffern $\{0, 1\}$ wählt man in der Regel die Ordnung (dies per Konvention):

$$0 < 1 \text{ bzw. } \text{FALSE} < \text{TRUE}.$$

- ④ Wie jedem Skat- oder Doppelkopfspieler bestens vertraut ist, sticht der Kreuz Bube den Pik, Herz oder Karo Bube. Mit anderen Worten, die Farben von Karten Kreuz, Pik, Herz und Karo bilden einen Zeichenvorrat, auf dem eine von Skatbrüdern und -schwestern allgemein akzeptierte Ordnung erklärt ist:

$$\clubsuit > \spadesuit > \heartsuit > \diamondsuit$$

Mit Hilfe der folgenden Definition können aus einzelnen Zeichen Worte gebildet werden.

Definition 4.5:

A sei ein Zeichenvorrat bzw. ein Alphabet mit einer Ordnung \Leftarrow .

- a) Ein **Wort** über dem Alphabet A ist eine endliche Folge $w = a_1 a_2 a_3 \dots a_k$ mit $a_i \in A; k \in \mathbb{N}_0$.

Die Länge des Wortes w ist k ; Notation: $|w| = k$.

λ bezeichnet das leere Wort; das ist das eindeutige Wort mit der Eigenschaft $|\lambda| = 0$.

- b) A^k sei die Menge aller Worte über A mit Länge k , *i.e.*

$$A^k := \{w \mid w \text{ Wort über } A \text{ mit } |w| = k\} \text{ für } k \in \mathbb{N}_0$$

- c) A^* ist die Menge aller Worte über A ; *i.e.*

$$A^* := A^0 \cup A^1 \cup A^2 \cup A^3 \cup \dots$$

Beispiel 4.9:

Diese ausführliche Definition ruft natürlich nach faßbaren Beispielen.

- ① Zunächst betrachten wir die oben definierte Menge der Dezimalziffern. $\mathbb{D} = \{0, 1, 2, 3, \dots, 9\}$ Gemäß unserer Definition ist nun die Menge der Worte über diesem Alphabet durch das leere Wort, alle einziffrigen Worte, zweiziffrigen Worte, usw.

$$\mathbb{D}^* := \{\lambda\} \cup \{0, 1, 2, \dots, 9\} \cup \{00, 01, 02, \dots, 99\} \cup \{000, 001, 002, \dots\} \dots$$

- ② Das binäre Alphabet $\mathbb{B} = \{0, 1\}$; bildet man über diesem Alphabet die Menge der Worte, dann resultieren alle Bitstrings beliebiger Länge.

$$\begin{aligned} \mathbb{B}^* &= \mathbb{B}^0 \cup \mathbb{B}^1 \cup \mathbb{B}^2 \cup \dots \\ &= \{\lambda\} \cup \{0, 1\} \cup \{00, 01, 10, 11\} \cup \{000, 001, 010, 011, \dots\} \cup \dots \end{aligned}$$

4.3 Codierungen

Informationen, die für Menschen verständlich sind, werden durch Texte oder Zahlen dargestellt. Also, Menschen verstehen Worte, die aus Zeichen des lateinischen Alphabets gebildet sind oder Zahlen, die aus dem dezimalen Alphabet \mathbb{D} gebildet werden.

Die zu verarbeitenden Informationen sind in klassischen Rechnern in Form von Strings von Bits codiert. Jedes Bit solch eines Bitstrings nimmt dabei entweder den Wert 0 oder den Wert 1 an. Wichtig an dieser Stelle ist nun zu sehen, dass in klassischen Rechnern die Bits in Form *klassischer physikalischer Zustände* realisiert sind. Beispiele dafür sind:

☞ Übertragung auf einer Busleitung:

Niedrige Spannung (0 Volt) \longrightarrow logische 0;
hohe Spannung (5 Volt) \longrightarrow logische 1.

☞ Übertragung in Glasfasern:

Licht aus \longrightarrow logische 0;
Licht an \longrightarrow logische 1.

☞ Speichern in RAM oder ROM Chip:

'Kondensator' nicht geladen \longrightarrow logische 0;
'Kondensator' geladen \longrightarrow logische 1.

☞ Speichern auf CD-ROM:

'Land \longrightarrow logische 0';
'Pit \longrightarrow logische 1'.

☞ Speichern auf Festplatte:

'Bereich nicht magnetisiert \longrightarrow logische 0';
'Bereich magnetisiert \longrightarrow logische 1'.

Die Zuordnung klassischer physikalischer Zustand \Longleftrightarrow Bit — mit Wert 0 oder 1 — stellt also eine *Abstraktion* dar um die Verschiedenartigkeit und Komplexität der zugrundeliegenden Hardware transparent zu machen. Durch diese Zuordnung

$$\text{physikalischer Zustand} \Longleftrightarrow \text{Wert des Bits}$$

werden Informationen durch das Muster von unterschiedlichen physikalischen Zuständen übertragen, gespeichert und/oder verarbeitet. In einem klassischen Rechner sind zu jedem Zeitpunkt sämtliche In- und Outputs in einem wohldefinierten Zustand (logische Werte 0 oder 1), ein Zwischenwert ist nicht erlaubt.

Bit ist ein Kunstwort und steht für *binary digit*. Ein Bit kann ausschließlich die diskreten Werte 0 oder 1 annehmen. Ein Bit ist also ein Platzhalter, der entweder den Wert 0 oder den Wert 1 annehmen kann, entsprechend dem Grundwortschatz eines Computers, Strom aus oder Strom an, geladen oder ungeladen, etc. Per Konvention ist festgelegt, welcher physikalische Zustand (Strom aus, Strom an, Licht an/Licht aus bei Glasfaser) welchem Wert zugeordnet ist.

Also: In einem Computer werden keine 0en oder 1en gespeichert oder verarbeitet, diese Werte sind lediglich abstrahierte Bezeichner für verschiedene Zustände, in denen sich Teile des Rechners befinden.

In einem anderen Kontext formuliert: *Ein Bit ist die minimale Informations-einheit, die ein Computer verarbeiten kann.*

Das zu lösende Problem besteht nun darin, wie die für Menschen verständlichen Zeichen wie Buchstaben, Zahlen, usw. in Zeichen abgebildet werden, die ein Computer speichern und verarbeiten kann. Etwas formaler ausgedrückt, die Zeichen des lateinischen Alphabets, Ziffern usw. müssen durch das binäre Alphabet $\mathbb{B} = \{0, 1\}$ geeignet dargestellt oder **codiert** werden.

Eine solche zweiwertige Codierung nennt man

Binär–Codierung.

Bevor wir konkrete Codierungen diskutieren, sehen wir uns an, wie eine Codierung formal definiert ist.

Codierung nach DIN 44300

1. Eine Vorschrift für die eindeutige Zuordnung (Codierung) der Zeichen eines Zeichenvorrates zu denjenigen eines anderen Zeichenvorrates (Bildmenge).
2. Unter Code bezeichnet man auch den bei der Codierung als Bildmenge auftretenden Zeichenvorrat.

Die präzise mathematische Definition einer Codierung lautet folgendermaßen:

Definition 4.6:

A und B seien zwei Zeichenvorräte bzw. Alphabete.

Eine Abbildung

$$c : A \longrightarrow B^*$$

heißt **Codierung**, der Bildbereich $c(A) := \{c(a), a \in A\} \subset B^*$ heißt **Code** und jedes Element $b \in c(A)$ nennt man **Codewort**.

Eine Codierung ist demnach eine Abbildung (=Funktion) von einem Zeichenvorrat auf Wörter eines anderen Zeichenvorrats. Wichtig ist hier, dass der Zielbereich die **Wörter** eines Zeichenvorrats — also nicht nur den Zeichenvorrat — umfaßt.

Codierungen können nun auf verschiedene Art und Weisen angegeben werden:

- ☞ durch die explizite Angabe einer Wertetabelle,
- ☞ durch graphische Methoden,
- ☞ durch logische Gleichungen,
- ☞ durch algebraische Methoden.

Beispiel 4.10:

Eine früher sehr verbreitete Codierung für Buchstaben und Zahlen ist die sogenannte **Morse-Codierung**, die in der Telegraphie benutzt wird. Diese Codierung geht zurück auf SAMUEL MORSE (1791 - 1872), der die Umwandlungsvorschrift nach der Häufigkeit des Auftretens von Buchstaben in der englischen Sprache entwickelt hat. Die Philosophie dabei ist, dass häufig benutzte Buchstaben wie 'e' sehr kurzen Codewörter zugewiesen werden, selten auftretende Buchstaben wie 'q' dagegen langen Codewörter.

Das Zielalphabet der Morsecodierung ist die Menge $\{\bullet, -\}$, wobei in der Telegraphie das Zeichen ' \bullet ' durch eine kurze Pause zwischen zwei Signalen (e.g. Stromimpulse oder akustische Signale) und ' $-$ ' durch eine lange Pause dargestellt wird. Das Längenverhältnis von kurzer Pause zu langer Pause beträgt 1 : 3.

Im obigen Formalismus ausgedrückt, ist die MORSE-Codierung eine Abbildung

$$c_M : \{A, B, C, \dots, Z, 0, 1, 2, \dots, 9\} \cup \{\ddot{A}, \ddot{U}, \ddot{O}, CH\} \longrightarrow \{\bullet, -\}^*,$$

mit der in der Tabelle [4.1] angegebenen expliziten Zuordnung.

A	• –	I	• •	R	• – •	1	• – – –
Ä	• – • –	J	• – – –	S	• • •	2	• • – –
B	– • • •	K	– • –	T	–	3	• • • –
C	– • – •	L	• – • •	U	• • –	4	• • • •
CH	– – – –	M	– –	Ü	• • – –	5	• • • • •
D	– • •	N	– •	V	• • • –	6	– • • • •
E	•	O	– – –	W	• – –	7	– – • • •
F	• • – •	Ö	– – – •	X	– • • –	8	– – – • •
G	– – •	P	• – – •	Y	– • – –	9	– – – – •
H	• • • •	Q	– – • –	Z	– – • •	0	– – – – –

Tabelle 4.1: Der MORSE-Code.

Beispiel 4.11: Eine andere, ebenfalls häufig eingesetzte Codierung, ist die sogenannte

Zählcodierung.

Diese Codierung liegt dem Telefonwahlsystem zugrunde. Bei dieser Codierung wird jede Ziffer durch einen Binärwert mit eindeutiger Wortlänge dargestellt.

Formal ist die Zählcodierung eine Abbildung

$$c_T : \{0, 1, 2, \dots, 9\} \longrightarrow \mathbb{B}^*$$

mit der in Tabelle [4.2] angegebenen Zuordnung.

1 → 10	6 → 1111110
2 → 110	7 → 11111110
3 → 1110	8 → 111111110
4 → 11110	9 → 1111111110
5 → 111110	0 → 11111111110

Tabelle 4.2: Die Zählcodierung.

Die beiden Beispiele, der Morsecode und die Zählcodierung wie sie in der Tabelle [4.1] bzw. [4.2] angegeben sind, stellen Codierungen in Form von Wertetabellen dar.

Im Laufe der Zeit haben sich für eine Reihe spezieller Codierungen feste Namen eingebürgert. Sind A und B zwei Zeichenvorräte bzw. Alphabete, dann nennt man eine Codierung der Form

$$c : A \longrightarrow B^n \quad (n \in \mathbb{N} \text{ fest})$$

eine **Blockcodierung**, da der Zeichenvorrat A durch Worte gleicher Länge des Zeichenvorrats B codiert wird.

In der Informationstechnik betrachtet man in der Regel Codierungen der Form

$$c : A \longrightarrow \mathbb{B}^* \quad \mathbb{B} = \{0, 1\},$$

welche man **Binärcodierung** nennt. Wird für die Binärcodierung eine feste Wortlänge n vereinbart, nennt man dies eine

n -Bit Codierung.

Beispiele für n -Bit Codierungen werden wir gleich kennenlernen, beispielsweise sind die in Rechnern verwendeten Zeichencodes typische n -Bit Codierungen.

4.4 8-Bit Zeichencodes

Codierungen können nun auf verschiedene Weisen beschrieben werden, beispielsweise in Form von Tabellen. Genau diesen Weg wollen wir nun einmal durchexerzieren, um die ASCII Codierung zu erläutern.

Mit einem einzigen Bit als Zielmenge der Codierungsabbildung kommt man nicht allzuweit, denn damit kann man genau zwei Zeichen darstellen, sagen wir das Zeichen 0 steht für den Buchstaben "a" und das Zeichen 1 ist dem Buchstaben "b" zugeordnet:

1. Bit		Zeichen
0	\iff	a
1	\iff	b

Die Kombination zweier Bits als Zielbereich der Codierungsabbildung ermöglichen die Darstellung von 4 Zeichen, denn zwei Bits erlauben die folgenden 0 und 1 Kombinationen:

2. Bit	1. Bit		Zeichen
0	0	\iff	a
0	1	\iff	b
1	0	\iff	c
1	1	\iff	d

Anhand obiger Anordnung erkennt man bereits, dass wir die Konvention benutzen, Bitfolgen immer von rechts nach links zu lesen, das bedeutet, das niedrigstwertigste Bit steht immer rechts, das Bit mit dem höchsten Stellenwert steht immer ganz links. Die Kombination dreier Bits führt auf acht verschiedene 0, 1 Folgen:

3. Bit	2. Bit	1. Bit		Zeichen
0	0	0	\iff	a
0	0	1	\iff	b
0	1	0	\iff	c
0	1	1	\iff	d
1	0	0	\iff	e
1	0	1	\iff	f
1	1	0	\iff	g
1	1	1	\iff	h

was längst noch nicht ausreicht, allein die 26 Buchstaben des Alphabets darzustellen.

Um es kurz zu machen, aus obigen Überlegungen ist es nicht schwer zu entnehmen, dass mit n Bits genau 2^n verschiedene 0, 1-Kombinationen möglich sind,

also 2 Bits liefern $2^2 = 4$ Möglichkeiten, 3 Bits ermöglichen $2^3 = 8$ Kombinationen usw. Zusammengefaßt erhält man damit eine Zuordnung wie in der Tabelle [4.3].

1 Bit	$2^1 =$	2 Kombinationen \Rightarrow	2 Zeichen
2 Bit	$2^2 =$	4 Kombinationen \Rightarrow	4 Zeichen
3 Bit	$2^3 =$	8 Kombinationen \Rightarrow	8 Zeichen
4 Bit	$2^4 =$	16 Kombinationen \Rightarrow	16 Zeichen
5 Bit	$2^5 =$	32 Kombinationen \Rightarrow	32 Zeichen
6 Bit	$2^6 =$	64 Kombinationen \Rightarrow	64 Zeichen
7 Bit	$2^7 =$	128 Kombinationen \Rightarrow	128 Zeichen
8 Bit	$2^8 =$	256 Kombinationen \Rightarrow	256 Zeichen

Tabelle 4.3: Anzahl der Zeichen, die sich durch die Kombination mehrerer Bits darstellen lassen.

Offensichtlich ergeben sich erst ab der Aneinanderreihung von 6 Bits ausreichende Kombinationsmöglichkeiten, die unterschiedlichen Zeichen darzustellen. Man unterscheidet demgemäß die folgenden Binärcodierungen:

- 6-Bit-Code: der BCD-Code (für binary coded decimal), mit 64 darstellbaren Zeichen
- 7-Bit-Code: der ursprüngliche 7-Bit ASCII-Code mit 128 darstellbaren Zeichen
- 8-Bit-Codes, mit 256 darstellbaren Zeichen, darunter:
 - * der EBCDIC-Code (für *extended binary coded decimal interchange code*)
 - * der erweiterte ASCII-Code, der beim PC verwendet wird

Der erweiterte ASCII-Code war lange Zeit die relevante Zeichendarstellung in der PC-Welt. Der 8-Bit EBCDIC-Code findet in den IBM Großrechenanlagen Verwendung.

4.4.1 Der erweiterte ASCII-Code

Das Kürzel ASCII (sprich aski) steht für *American Standard Code for Information Interchange* und bezeichnet einen Standard in der Computerwelt. Dieser Standard wurde ursprünglich in dem 7-Bit Format im Jahre 1963 von dem amerikanischen Standardisierungsinstitut ANSI¹ vorgeschlagen und 1968 freigegeben.

¹Das Akronym ANSI steht für *American National Standards Institute*. Siehe auch die Web-Site <http://www.ansi.org>.

Der ASCII-Code ist eine tabellarische Zuordnungsregel (siehe die Tabelle [4.1]), welche die Darstellung von Zeichen in Form binärer Zahlen vorschreibt. Da der erweiterte ASCII-Code ein 8-Bit-Code ist, können damit 256 verschiedene Zeichen dargestellt werden. Man bezeichnet die Einheit von 8 Bit auch als ein Byte.

$$8 \text{ Bit} = 1 \text{ Byte}$$

Betrachten wir nun zwei Beispiele: In der Tabelle [4.4] ist die ASCII-Darstellung des Buchstabens "a" und des Sonderzeichens '§' dargestellt. Hierbei ist wichtig sich zu merken, dass die Bits von rechts nach links gelesen werden. Also im Bit-Code des Buchstabens "a" ist das erste Bit ganz rechts auf den Wert 1 gesetzt, dann folgen vier Bits mit 0, das sechste und siebente Bit hat den Wert 1 und das höchste Bit ganz links den Wert 0.

Zeichen	a							
Bit-Code	0	1	1	0	0	0	0	1

Zeichen	§							
Bit-Code	0	0	0	1	0	1	0	1

Tabelle 4.4: Die ASCII-Darstellungen der Zeichen "a" und "§".

Ein Blick auf die ASCII-Tabelle [4.1] offenbart, dass in den ASCII-Tabellen keine Bitfolgen aufgelistet sind, sondern Dezimalzahlen. In manchen Tabellen findet man auch eine Zuordnung der Form: Zeichen \longleftrightarrow Hexadezimalzahl oder sogar beides.

Um diese Zuordnung

$$\text{Dezimalzahl} \longleftrightarrow \text{8 Bit String}$$

zu erhalten, wird der 8 Bit String als **Binärzahl** interpretiert. Jede Binärzahl, das heißt also, jede Zahl, die als Folge von 0en und 1en gegeben ist, hat nun einen bestimmten *Wert*. Dabei ist wesentlich, welchen Wert jedes einzelne Bit in der Binärzahl hat. Hier benutzen wir — wie bereits erwähnt — die Konvention, dass das Bit mit dem niedrigsten Wert ganz rechts steht, das Bit mit dem höchsten Wert steht ganz links. Die Tabelle [4.5] zeigt die Wertigkeiten der verschiedenen Bits innerhalb eines 8 Bit Strings.

Mit Hilfe der Tabelle [4.5] läßt sich nun der sogenannte **ASCII-Wert** eines Zeichens bestimmen. Dazu multipliziert man einfach den Bitcode mit den entsprechenden Potenzen der Zahl 2 und addiert die daraus resultierenden Werte

	Byte							
Potenz	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Wert	128	64	32	16	8	4	2	1

Tabelle 4.5: Die Wertigkeit der Bits innerhalb eines Bytes.

auf. Wir wollen dies an den oben dargestellten Beispielen des Buchstabens "a" und des Sonderzeichens "§" demonstrieren.

Multipliziert man den Bitcode des Buchstabens "a" mit den entsprechenden Potenzen von 2, die dem jeweiligen Stellenwert entsprechen, so verbleiben die Dezimalwerte 64, 32 und 1 (siehe Tabelle [4.6]).

Zeichen	a							
Dualzahl	0	1	1	0	0	0	0	1
Potenz	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Wert	0	64	32	0	0	0	0	1

Tabelle 4.6: Die Wertigkeit des ASCII Zeichens "a".

Addiert man nun die von Null verschiedenen Wertigkeiten in Tabelle [4.6] auf, so ergibt sich der ASCII-Wert des Zeichens "a" zu:

$$\text{ASCII-Wert(a)} = 64 + 32 + 1 = 97$$

In der ASCII Tabelle [4.1] findet man gerade diese Zuordnung, das Zeichen "a" hat den ASCII-Wert 97.

Dies ist auch der Grund für unsere Betrachtungen. Es besteht hin und wieder durchaus die Notwendigkeit, auf den ASCII-Code zurückzugreifen, um das eine oder andere Zeichen darstellen zu können. Die ASCII-Code-Tabellen sind nun gerade so angelegt, dass einem Zeichen ein bestimmter Wert zugeordnet ist, wie in unserem Beispiel der Wert 97 dem Zeichen "a". Daher ist es durchaus sinnvoll, diese ASCII-Codierung in etwas mehr Detail zu verstehen.

Dies hat nicht nur akademische Bedeutung sondern mitunter ganz handfeste praktische Konsequenzen. Unter Umständen ist man gezwungen, ein bestimmtes

ASCII-Zeichen über die Tastatur direkt einzugeben, beispielsweise den Backslash, *i.e.* das Zeichen \. Ein Blick in eine ASCII Tabelle offenbart, dieses Zeichen hat den ASCII Wert 92. Gibt man nun über die Tastatur die Tastenkombination

`<Alt> + <0>+<9>+<2>`

ein, wobei die Ziffern auf dem Num-Block der Tastatur gedrückt werden müssen (<Alt>-Taste gedrückt halten), dann wird der Backslash ausgegeben. Auf diese Art und Weise kann jedes ASCII-Zeichen am Bildschirm ausgegeben werden, wichtig ist dabei, dass die Zifferntasten des Num-Blocks benutzt werden.

Zeichen	§							
Dualzahl	0	0	0	1	0	1	0	1
Potenz	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Wert	0	0	0	16	0	4	0	1

Tabelle 4.7: Die Wertigkeit des ASCII Zeichens "§".

Zwecks Übung kann man die gleiche Betrachtung auch für das Sonderzeichen "§" durchführen, siehe Tabelle [4.7]. Die Addition der verschiedenen Wertigkeiten der acht Bits liefert den folgenden ASCII-Wert für das Zeichen "§":

$$\text{ASCII-Wert}(\text{§}) = 16 + 4 + 1 = 21.$$

Zur Codierung eines fortlaufenden Textes werden die Codeworte der einzelnen Zeichen einfach aneinandergefügt. Das ist mit anderen Worten genau die natürliche Fortsetzung der ASCII-Codierung. Die Zeichenkette "BA Mannheim" wird zum Beispiel durch die Folge (inklusive des Leerzeichens, was den ASCII Wert 032 hat)

066 065 032 077 097 110 110 104 101 105 109

dargestellt. Diese ASCII-Nummern werden natürlich seinerseits nochmals in Bitfolgen umgesetzt, so dass die reale interne Repräsentation der Zeichenkette "BA Mannheim" im Rechner folgendermaßen aussieht:

01000010 01000001 00100000 01001101 01100001
 01101110 01101110 01101000 01100101 01101001 01101101

In der Abbildung [4.1] sind die 256 ASCII Zeichen dargestellt. Die ersten 32 Zei-

000 NUL	032	071 G	110 n	149 ò	188 ƒ	227 π
001 SOH	033 !	072 H	111 o	150 û	189 ƒ	228 Σ
002 STX	034 "	073 I	112 p	151 ù	190 ƒ	229 σ
003 ETX	035 #	074 J	113 q	152 ý	191 ƒ	230 μ
004 EOT	036 \$	075 K	114 r	153 Ò	192 ƒ	231 τ
005 ENQ	037 %	076 L	115 s	154 Û	193 ƒ	232 Φ
006 ACK	038 &	077 M	116 t	155 ø	194 ƒ	233 Θ
007 BEL	039 '	078 N	117 u	156 ƒ	195 ƒ	234 Ω
008 BS	040 (079 O	118 v	157 ƒ	196 ƒ	235 δ
009 HT	041)	080 P	119 w	158 Pt	197 ƒ	236
010 LF	042 *	081 Q	120 x	159 f	198 ƒ	237 Φ
011 VT	043 +	082 R	121 y	160 á	199 ƒ	238 ε
012 FF	044 ,	083 S	122 z	161 í	200 ƒ	239
013 CR	045 -	084 T	123 {	162 ó	201 ƒ	240
014 SO	046 .	085 U	124	163 ú	202 ƒ	241 ±
015 SI	047 /	086 V	125 }	164 ñ	203 ƒ	242
016 DLE	048 0	087 W	126 ~	165 Ñ	204 ƒ	243
017 DC1	049 1	088 X	127 ∅	166 ª	205 ƒ	244 f
018 DC2	050 2	089 Y	128 Ç	167 °	206 ƒ	245
019 DC3	051 3	090 Z	129 ü	168 ¿	207 ƒ	246
020 DC4	052 4	091 [130 é	169 ƒ	208 ƒ	247
021 NAK	053 5	092 \	131 â	170 ƒ	209 ƒ	248 °
022 SYN	054 6	093]	132 ä	171 ½	210 ƒ	249
023 ETB	055 7	094 ^	133 à	172 ¼	211 ƒ	250
024 CAN	056 8	095 _	134 â	173 i	212 ƒ	251 √
025 EM	057 9	096 `	135 ç	174 «	213 ƒ	252
026 SUB	058 :	097 a	136 è	175 »	214 ƒ	253
027 ESC	059 ;	098 b	137 ë	176 ¶	215 ƒ	254
028 FS	060 <	099 c	138 è	177 ¶	216 ƒ	255
029 GS	061 =	100 d	139 ì	178 ¶	217 ƒ	
030 RS	062 >	101 e	140 î	179	218 ƒ	
031 US	063 ?	102 f	141 ï	180 †	219 ƒ	
	064 @	103 g	142 Ä	181 ‡	220 ƒ	
	065 A	104 h	143 Å	182 ‡	221 ƒ	
	066 B	105 i	144 É	183 ¶	222 ƒ	
	067 C	106 j	145 æ	184 ¶	223 ƒ	
	068 D	107 k	146 Æ	185 ¶	224 α	
	069 E	108 l	147 ô	186 ¶	225 β	
	070 F	109 m	148 ö	187 ¶	226 Γ	

Abbildung 4.1: Der ASCII-Zeichensatz.

chen stellen die sogenannten **nicht-druckbaren Zeichen** dar — diese nennt man mitunter auch **Steuerzeichen**, siehe Tabelle [4.8] —, die früher zur Datenübertragung dienten. Beispielsweise kann eine Nachricht aus einem SOH-Zeichen (Start of Header), dem eigentlichen Header, einem STX-Zeichen (Start of Text), dem eigentlichen Text, einem ETX (End of Text) und einem EOT (End of Transmission) bestehen. In der Praxis werden die über Telefonleitung und Netze gesendeten Nachrichten jedoch anders formatiert, die ASCII Steuerzeichen werden heute kaum noch verwendet.

Anmerkungen:

1. Wir haben hier — stillschweigend — die Konvention benutzt, dass die Wertigkeit der Bits in einem Byte von rechts nach links ansteigen. Also ganz rechts in einer Bitfolge steht das niedrigstwertige Bit, ganz links in der Bitfolge das Bit mit dem höchsten Stellenwert. Diese Ordnung ist man

Dez	Name	Bedeutung	Dez	Name	Bedeutung
0	NUL	Null	16	DLE	Data Link Escape
1	SOH	Start of Heading	17	DC1	Device Control 1
2	STX	Start of Text	18	DC2	Device Control 2
3	ETX	End of Text	19	DC3	Device Control 3
4	EOT	End of Transmission	20	DC4	Device Control 4
5	ENQ	Enquiry	21	NAK	Neg. Acknowledgement
6	ACK	Acknowledgement	22	SYN	Synchronous idle
7	BEL	Bell	23	ETB	End of Transmission Block
8	BS	Backspace	24	CAN	Cancel
9	HT	Horizontal Tab	25	EM	End of Medium
10	LF	Line Feed	26	SUB	Substitute
11	VT	Vertical Tab	27	ESC	Escape
12	FF	Form Feed	28	FS	File Separator
13	CR	Carriage Return	29	GS	Group Separator
14	SO	Shift Out	30	RS	Record Separator
15	SI	Shift In	31	US	Unit Separator

Tabelle 4.8: Die Steuerzeichen des ASCII Zeichensatzes.

auch von den Dezimalzahlen gewohnt, hier stehe ja auch die Ziffern mit den kleinen 10er Potenzen ganz rechts, die Ziffer mit dem höchsten Stellenwert steht ganz links. Diese Konvention nennt man im Computerjargon

little endian Codierung

Diese Konvention wird auch von vielen Plattformen, darunter Intel PC, RS/6000, DEC-Stations von VAX verwendet, d.h. die Bits, die Zahlen repräsentieren, werden in dieser Konvention im Arbeitsspeicher und in Registern auf der CPU abgelegt.

Wenn's ein kleines endian gibt, wird es wohl auch ein großes endian geben. Es gibt in der Computerwelt selbstverständlich alles, was möglich ist. So verwenden IBM Großrechner und die SPARC genau die umgekehrte Konvention, die Bits von rechts nach links zu zählen. Diese Konvention nennt man

big endian Codierung

Diese Begriffe gehen auf JONATHAN SWIFT zurück, in dessen *Gullivers Reisen* Politiker karikiert werden, die Kriege über die Kontroverse führen, ob Eier am spitzen oder am runden Ende aufgeschlagen werden sollten.

- Um die Sache noch etwas komplizierter zu machen, sei hier angemerkt, dass das Betriebssystem Windows 95/98 und sämtliche Windows-Programme einen von ASCII-Zeichensatz abweichenden Zeichensatz benutzen. Dies ist der sogenannte ANSI-Zeichensatz. Dies ist ebenfalls ein genormter 8-Bit Zeichensatz. Dabei ist die Belegung des ANSI- und ASCII-Zeichensatzes für die Nummern 32 bis 127 identisch, unterschiedliche Belegungen haben die Nummern 128 bis 256.

Dies hat dann Konsequenzen, wenn Daten zwischen unterschiedlichen Umgebungen ausgetauscht werden müssen.

4.4.2 Der EBCDIC-Zeichensatz

Neben dem ASCII-Code ist noch der EBCDIC (Abkürzung für: *extended binary coded decimal interchange code*) weit verbreitet. Dieser Code verwendet acht Bits zur Zeichendarstellung, es werden jedoch im Gegensatz zum ASCII Code nicht alle möglichen Codewörter tatsächlich genutzt sondern nur 108 der möglichen Codewörter.

		Zonenteil															
Spalte	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
Zeile																	
Ziffernteil	0	NUL				SPA	&	-								0	
	1						/		a	j			A	J		1	
	2								b	k	s		B	K	S	2	
	3								c	l	t		C	L	T	3	
	4	PF	RES	BYP	PN				d	m	u		D	M	U	4	
	5	HT	NL	LF	RS				e	n	v		E	N	V	5	
	6	LC	BS	EOB	UC				f	o	w		F	O	W	6	
	7	DEL	IL	PRE	EOT				g	p	x		G	P	X	7	
	8								h	q	y		H	Q	Y	8	
	9								i	r	z		I	R	Z	9	
	A			SM		€	!	^	:								
	B					.	\$,	#								
	C					<	*	%	@								
	D					()	-	'								
	E					+	;	>	=								
	F						_	?	"								

Codierung mit dem EBCDIC

Codierung mit dem EBCDIC

Abbildung 4.2: Der EBCDIC-Zeichensatz.

Ein Byte setzt sich aus zwei Nibbels zusammen, wobei man den ersten Nibble als *Zonenteil* und den zweiten als *Ziffernteil* bezeichnet. In der Abbildung [4.2] sind die darstellbaren Zeichen aufgeführt, wobei die Zonen- und Ziffernteile jeweils hexadezimal angegeben sind. Um ein Zeichen — beispielsweise das 'A' — zu codieren, schreibt man zunächst den Zonenteil — im Beispiel **Ch** — und danach den Ziffernteil — hier **1h**. Demnach ist das Codewort für das Zeichen 'A' durch **C1h** gegeben.

In unserer Notation ist die EBCDIC-Codierung also durch folgende Vorschrift gegeben:

$$c_{EBCDIC} : A \longrightarrow \mathbb{B}^4 \times \mathbb{B}^4$$

4.5 Unicode

Die Computer-Industrie entwickelte sich überwiegend in den USA, was zum ASCII-Zeichensatz führte. ASCII eignet sich recht gut für die Codierung der Zeichen der englischen Sprache, jedoch weniger gut für andere Sprachen. Französisch braucht Akzente und die deutsche Sprache benutzt Umlaute und so weiter. Einige europäische Sprachen haben Zeichen, die es im ASCII-Code überhaupt nicht gibt, wie das deutsche ß oder das dänische ø. Einige Sprachen haben ganz andere Alphabete (z.B. Griechisch, Russisch oder Arabisch), während einige wenige Sprachen überhaupt kein Alphabet haben (z.B. Chinesisch). Nachdem der Computer die ganze Welt eroberte und die Software-Anbieter ihre Produkte natürlich auch in Ländern verkaufen wollen, in denen die meisten Benutzer kein Englisch sprechen, wurde ein andersartiger Zeichensatz benötigt. Schließlich erfordert die Internationalisierung — insbesondere auch die Kommunikation via Internet — dass ein Dokument oder eine Web-Seite Zeichen aus unterschiedlichen Sprachen enthalten muss. Benutzt man nur die ASCII-Codierung, ist es nicht möglich, in einem Dokument Zeichen aus unterschiedlichen Alphabeten (e.g. lateinische und griechische Buchstaben) zu verwenden.

Der erste Versuch, den ursprünglichen 7-Bit ASCII-Zeichensatz zu erweitern, war der Standard ISO 646, wodurch ASCII um weitere 128 Zeichen bereichert wurde. Dadurch entstand ein 8-Bit-Code mit der Bezeichnung **Latin-1**. Die zusätzlichen Zeichen waren überwiegend lateinische Buchstaben mit Akzenten und diakritischen Zeichen. Der nächste Versuch war ISO 8859, mit dem das Konzept einer **Codeseite** (Code Page) eingeführt wurde, und das 256 Zeichen für eine bestimmte Sprache oder Sprachgruppe bot. ISO 8859-1 ist Latin-1, ISO 8859-2 versorgt die osteuropäischen Sprachen mit lateinischem Alphabet (z.B. Tschechisch, Polnisch und Ungarisch). ISO 8859-3 enthält die Zeichen für Türkisch, Maltesisch, Esperanto, Galizisch usw. Problematisch am Codeseitenkonzept ist nur, dass die Software auf dem Laufenden darüber gehalten werden muß, mit welcher Codeseite sie arbeitet. Außerdem kann man keine Sprachen auf verschiedenen Codeseiten miteinander kombinieren, und das Schema ignoriert Japanisch und Chinesisch ganz.

Einige Computer-Unternehmen beschlossen daher, das Problem durch die Bildung eines Konsortiums² zu beheben, das ein neues System mit der Bezeichnung **Unicode** entwickelt hat. Die International Organisation for Standards (ISO) hat den Unicode als internationalen Standard (ISO 10646) übernommen, hier heißt diese Codierung UCS, was für *Universal Character Set* steht. Beide Gremien haben die Übereinkunft getroffen, die Entwicklung zu koordinieren, so dass beide Codierungen identisch sind (und bleiben). Unicode wird ständig um neue Schriftzeichen ergänzt. Unicode wird heute von den meisten Programmiersprachen (z.B. Java), allen Betriebssystemen (z.B. Windows 7, Linux) und vielen Anwendungsprogrammen unterstützt.

²Siehe die URL: <http://unicode.org>.

In Unicode wird jedem Zeichen und Symbol ein eindeutiger **Codepunkt** (Code Point) zugeordnet. Der Codepunkt ist zunächst *nicht* die Art und Weise, wie das Zeichen im Rechner dargestellt wird, sondern stellt ein Verfahren dar, alle möglichen Zeichen, die es in den verschiedenen Sprachen weltweit gibt, systematisch aufzulisten. Das Unicode-Consortium hat daher jedem Zeichen eines beliebigen Alphabets (lateinisch, hebräisch, arabisch, kyrillisch usw.) eine Zahl zugeordnet, die in der folgenden Weise geschrieben wird:

U + 0639 oder U + 0041.

Diese Zahl ist genau der Codepunkt. Hierbei steht U+ für Unicode und die nachfolgenden Zahlen sind Hexadezimalzahlen. Der Codepunkt U+0639 steht für den arabischen Buchstaben *ain*, der Codepunkt U+0041 für den Buchstaben A.³ Momentan ist die aktuelle Version Unicode 6.0, diese Version stellt 1.114.112 Codepunkte bereit.

Eine Zeichenkette wie

Hello

entspricht in Unicode den fünf Codepunkten

U + 0048 U + 0065 U + 006C U + 006C U + 006F.

Dies ist zunächst nur eine Kette von Zahlen, also eine Reihe von Codepunkten. Es ist bisher noch keine Aussage darüber gemacht, wie dies gespeichert wird oder in einer E-Mail dargestellt wird.

Das ist genau der Punkt, bei dem die Codierung ins Spiel kommt. Es gibt eine Reihe von Verfahren, wie der Unicode dargestellt werden kann. ISO 10646 bzw. Unioide definieren verschiedene Codierungsformate wie UTF-8, UCS-2, UTF-16, UCS-4 und UTS-32. Alle Verfahren — bis auf UTF-8 — verwenden mehrere Bytes, um die Unicodezeichen darzustellen, was die Verwendung solcher Codierungsverfahren in aktuellen Anwendungen und Netzwerkprotokollen beträchtlich erschwert, da diese 8 oder sogar nur 7 Bit Zeichen verwenden. Eine dieser Option ist der sogenannte UTF--8 Standard, wobei das Akronym UTF-8 für *UCS Transformation Format 8 Bit* steht. Diese Version wird beispielsweise für die Darstellung von Unicodezeichen in E-Mails genutzt.⁴

Im UTF-8 Format wird jeder Codepunkt zwischen 0 und 127 in einem einzigen Byte abgespeichert. Codepunkte mit Werten größer als 128 werden durch 2, 3 oder bis zu 6 Bytes gespeichert.

³Eine komplette Liste der dieser Codepunkte findet man mit dem **charmap** Programm unter Windows. Dazu ruft man über das Startmenü von Windows die *Ausführen* ... Option auf und gibt in dem sich öffnenden Fenster das Kommando **charmap** ein. Es öffnet sich ein weiteres Fenster mit den Zeichentabellen.

⁴Siehe z.B. RFC 3629, F. Yergeau, UTF-8, a transformation format of ISO 10646.

Codepunkt	UTF-8 Codewort
0000 0000 – 0000 007F	0xxxxxxx
0000 0080 – 0000 07FF	110xxxxx 10xxxxxx
0000 0800 – 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000 – 0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

4.6 Weitere Codes

4.6.1 Barcodes

Strich- oder Barcodes sind heutzutage in der gesamten Warenwirtschaft zu finden. Man sieht sie auf Verpackungen, Lieferscheinen, Flugtickets, Ausweisen, Etiketten, Leihbüchern und in vielen anderen Bereichen. Auch in Lagerverwaltungen, Kliniken und wissenschaftlichen Einrichtungen werden Barcodes dazu verwendet, Daten zu erfassen. Diese Codierung wird zwar nicht intern in Rechnern verwendet, jedoch von Rechnern dekodiert und dann intern z.B. als ASCII-Zeichenstring gespeichert.

Ein Strichcodeleser ist ein Gerät, das in der Lage ist, den sogenannten Strich- oder auch Barcode zu lesen. Damit lassen sich sehr schnell und effektiv geringe Datenmengen für die Weiterverarbeitung in einem Computer erfassen.

Es gibt eine Vielzahl von Codes für unterschiedliche Einsatzgebiete, häufig eingesetzte Codes sind EAN,⁵ 2/5 Interleaved, 2/5 Matrix, Codabar, Code 39 und Code 128. Auf den meisten Gebrauchsgütern findet man die EAN als Strichcode.

Dargestellt werden die Codes durch eine Sequenz senkrechter schmaler und breiter Striche bzw. Lücken.



Abbildung 4.3: EAN Code für eine hervorragende Pfeifentabaksorte.

Ein Barcodeleser tastet mit Hilfe von Photozellen den Strichcode ab. Je nach Helligkeit verändert sich der elektrische Widerstand der Photozellen. Diese Wi-

⁵EAN steht für Europäische Artikel Nummer.

derstandsänderungen lassen sich in digitale (d.h. binäre) Informationen übertragen.

Mit einem Barcodeleser lassen sich auf diese Weise etwa 1.000 Zeichen pro Sekunde einlesen.

Die EAN besteht aus einer Reihe vertikaler, unterschiedlich dicker Balken, unter denen in Klartext eine 13stellige Zahl ausgedruckt ist. Jeweils zwei dünne, unten etwas längere Striche fungieren als Trennbalken in der Mitte sowie als linke und rechte Begrenzung des Strichcodes. Die erste Ziffer steht immer etwas links außerhalb des Strichcodes. Der linke Block enthält die Ziffern zwei bis sieben, der rechte die Stellen acht bis 13. Die Bedeutung der Nummern — die Betriebsnummer des Herstellers wird übrigens in Deutschland von der Centrale für Coorganisation in Köln vergeben — ist für deutsche Artikel:

Stelle der Ziffer	Funktion
1 und 2	Länderkennzeichen
3 bis 7	Betriebsnummer des Herstellers
8 bis 12	vom Hersteller vergebene Artikelnummer
13	Prüfziffer

Die EAN ist eine 13stellige Nummer der Form

$$a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12} a_{13}$$

mit

$$a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

Die Stelle a_{13} ist die Prüfziffer, die folgende Bedingung erfüllen muß:

$$\sum_{k=0}^5 a_{2k+1} + 3 \times \left(\sum_{k=1}^6 a_{2k} \right) + a_{13} \equiv \quad \text{mod } 10 \quad (4.1)$$

Das bedeutet im Klartext: a_{13} muß die Summe $a_1 + a_3 + a_5 + a_7 + a_9 + a_{11} + 3 \times (a_2 + a_4 + a_6 + a_8 + a_{10} + a_{12})$ so ergänzen, dass die Kombination durch 10 teilbar wird.

Diese Prüfsumme ist so konstruiert, dass alle Verwechslungen einer Ziffer in der EAN durch das Verfahren entdeckt werden; die Vertauschung von Ziffern an ungeraden bzw. geraden Positionen kann aber nicht entdeckt werden.

Im EAN Code des Pfeifentabaks aus der Abbildung [4.3] haben wir folgende Struktur:

57	Ländercode (Dänemark)
12341	Hersteller (W Ø Larsen)
25630	Artikel (Sweet Aromatic)

Aus diesen Ziffern ermittelt sich die Prüfziffer zu

$$5 + 1 + 3 + 1 + 5 + 3 + 3 \times (7 + 2 + 4 + 2 + 6 + 0) = 18 + 3 \times 21 = 81.$$

Damit dies durch zehn teilbar wird (ohne Rest) muß 9 addiert werden, also ist $a_{13} = 9$, was zur korrekten EAN Ziffer aus Abbildung [4.3] führt.

4.6.2 ISBN

Jedes neuere Buch sollte eines sogenannte **International Standard Book Number** – kurz ISBN – haben. Dies ist ein 10-ziffriges Codewort, welches dem Buch durch den Verlag zugewiesen wird.⁶ Ein Buch hat beispielsweise die ISBN

0-19-859617-0.

Obwohl die Bindestriche an unterschiedlichen Positionen auftreten können, sind sie unwichtig. Die Zahl ist folgendermaßen aufgebaut. Die zehn-ziffrige Zahl ist in vier Zahlengruppen unterschiedlicher, variabler Länge aufgeteilt, die durch Bindestriche oder Leerzeichen getrennt sind.

- *Group identifier*
Die führende Zahlengruppe der ISBN, in unserem Beispiel die '0', identifiziert ein Land oder einen Sprachraum. Diesen Bestandteil der ISBN nennt man **Group Identifier**. In unserem Beispiel steht die 0 für 'Englisch', eine '3' steht für den deutschsprachigen Raum, die Gruppennummer 982 entspricht dem südpazifischen Raum. Ein Group Identifier kann bis zu 5 Ziffern lang sein.
- *Publisher prefix*
Der zweite Teil einer ISBN – der sogenannte Publisher Prefix – identifiziert einen bestimmten Verlag innerhalb einer Gruppe. In unserem Beispiel entspricht der 19 dem Verlag Oxford University Press.
- *Title identifier*
Der dritte Teil der ISBN identifiziert eine spezielle Auflage einer Veröffentlichung eines Verlags. Ein Title Identifier kann bis zu sechs Ziffern lang sein. Die Ziffernfolge 859617 identifiziert ein spezielles Buch.
- *Prüfziffer*
Die Prüfziffer ist immer die letzte Ziffer einer ISBN. Sie ist so festgelegt, dass für die gesamte 10-ziffrige Zahl

$$x_1x_2 \dots x_9x_{10}$$

⁶Umfassende Informationen über ISBN findet man auf der folgenden Web-Seite:

<http://www.isbn.org>.

Siehe auch die Darstellung in [10], Kap. 6.2 und [53], Chap.3.

gilt:

$$\sum_{k=1}^{10} k \cdot x_k \equiv 0 \pmod{11}.$$

Die linke Seite dieser Kongruenz heißt *gewichtete Prüfsumme* der Zahlen $x_1 \dots x_{10}$. Für die gewählte 9-stellige Zahl x_1, x_2, \dots, x_9 ist die Ziffer x_{10} definiert durch:

$$x_{10} = \sum_{k=1}^9 k \cdot x_k \pmod{11}.$$

Falls sich herausstellt, dass die Prüfziffer den Wert '10' hat, wird das Symbol X verwendet.

Der ISBN Code ist so designed, dass folgende Probleme erkannt werden:

1. jeder Einzelfehler,
2. jeder Doppelfehler, der durch die Vertauschung zweier Ziffern entsteht.

Das Schema zur Fehlererkennung funktioniert folgendermaßen. Für einen Vektor $y_1 y_2 \dots y_{10}$ berechnet man die gewichtete Prüfsumme

$$Y = \sum_{k=1}^{10} k \cdot y_k.$$

Falls $Y \not\equiv 0 \pmod{11}$, dann muss ein Fehler aufgetreten sein. Wir verifizieren dies für die obigen beiden Fälle. Wir nehmen dazu an, dass

$$x = x_1 x_2 \dots x_{10}$$

das gesendete Codewort ist.

1. Angenommen, das empfangene Codewort $y = y_1 y_2 \dots y_{10}$ ist identisch mit x mit der Ausnahme, dass die Ziffer x_j ersetzt ist durch $x_j + a$; $a \neq 0$. Dann

$$\begin{aligned} Y &= \sum_{k=1}^{10} k \cdot y_k \\ &= \sum_{k=1}^{10} k \cdot x_k + j \cdot a \\ &\equiv j \cdot a \pmod{11} \\ &\not\equiv 0 \pmod{11}. \end{aligned}$$

2. Angenommen, in dem empfangenen Vektor Y wurden die beiden Komponenten x_j und x_k von x vertauscht. Dann ist

$$\begin{aligned}
 Y &= \sum_{k=1}^{10} k \cdot y_k \\
 &= \sum_{k=1}^{10} k \cdot x_k + (k-j) \cdot x_j + (j-k) \cdot x_k \\
 &= \sum_{k=1}^{10} k \cdot x_k + (k-j) \cdot (x_j - x_k) \\
 &\equiv (k-j) \cdot (x_j - x_k) \pmod{11} \\
 &\not\equiv 0 \pmod{11}.
 \end{aligned}$$

falls $k \neq j$ und $x_j \neq x_k$.

Anmerkung:

Der mathematisch wesentliche Punkt, warum die Fehlererkennung in der obigen Form funktioniert, liegt darin, dass die Eigenschaft eines Zahlkörpers genutzt wird,⁷ dass das Produkt zweier nicht-verschwindender Elemente ebenfalls nicht 0 ist. Diese Eigenschaft gilt nicht in $\mathbb{Z}_{10} = \{0, 1, 2, \dots, 9\}$ mit der Addition und Multiplikation modulo 10, da hier beispielsweise gilt: $2 \cdot 5 \equiv 0 \pmod{10}$. Das ist genau der Grund, warum ISBN mit Modulus 11 statt 10 arbeitet.

Der ISBN Code kann nicht genutzt werden, einen aufgetretenen Fehler zu korrigieren, es sei denn, wir wissen, dass eine gegebene Ziffer ein Fehler ist. Aus diesem Grund funktioniert das folgende Szenario.

Man bittet jemanden, irgendein Buch zu nehmen, welches man selbst nicht kennt, und die ISBN dieses Buchs laut vorzulesen. Für eine der Ziffern soll jedoch statt der Ziffer ein 'x' gesagt werden. Nach einigen Sekunden Arbeit, wissen wir den Wert von 'x'. Ist die ISBN zum Beispiel

$$3 - 86x41 -053-7,$$

dann geht die Rechnung folgendermaßen:⁸

$$(3 \cdot 1 + 8 \cdot 2 + 6 \cdot 3 + x \cdot 4 + 4 \cdot 5 + 1 \cdot 6 + 0 \cdot 7 + 5 \cdot 8 + 3 \cdot 9 + 7 \cdot 10) \pmod{11} \equiv 4x + 2 \pmod{11}.$$

Es ist also die modulare Gleichung

$$(4 \cdot x + 2) \pmod{11} \equiv 0 \pmod{11}.$$

zu lösen. Dies führt auf

$$x \equiv 9 \cdot 4^{-1} \pmod{11} \equiv 9 \cdot 3 \pmod{11} \equiv 27 \pmod{11} \equiv 5,$$

was die gesuchte Ziffer ist.

⁷Das ist in diesem Fall die Menge $\mathbb{Z}_{11} = \{0, 1, \dots, 10\}$ mit der Addition und Multiplikation modulo 11.

⁸Man beachte, dass man im Zahlkörper $GF(11) = \{\mathbb{Z}_{11}, +, \times\}$ arbeitet, daher ist jede arithmetische Operation modulo 11.

4.6.3 QR-Codes



Abbildung 4.4: Ein QR-Code

Der QR-Code⁹ ist ein zweidimensionaler Code, der von der japanischen Firma Denso Wave im Jahr 1994 entwickelt wurde.

Ursprünglich wurde der QR-Code in der Materialwirtschaft entwickelt, er diente zur Markierung von Baugruppen und Komponenten für die Logistik in der Automobilproduktion des Toyota-Konzerns. Die konkrete Entwicklung des 2D-Codes übernahm die Tochterfirma Denso Wave, die auch Identifikationssysteme und Geräte zur mobilen Datenerfassung entwickelt.

Der QR-Code besteht aus einer quadratischen Matrix aus schwarzen und weißen Punkten, die die kodierte Daten binär darstellen. Eine spezielle Markierung in drei der vier Ecken des Quadrats gibt die Orientierung vor. Die Daten im QR-Code sind durch einen Fehler korrigierenden Code geschützt. Dadurch wird der Verlust von bis zu 30 % des Codes toleriert.

Es gibt mehrere Standards, welche die Kodierung von QR-Codes beschreiben.

- Oktober 1997 - AIM (Association for Automatic Identification and Mobility) International
- Januar 1999 JIS X 0510
- Juni 2000 ISO/IEC 18004:2000 (zurückgezogen)
- 1. September 2006 ISO/IEC 18004:2006
Dieser ISO-Standard definiert den QR-Code 2005, eine Erweiterung des QR-Code-Modells 2. Spezifiziert nicht, wie QR-Code-Modell 1 gelesen werden kann bzw. baut auf QR-Code-Modell-1-Spezifikationen auf.

⁹Englisch: *Quick Response*.

Der strukturelle Aufbau des Codes beinhaltet die Versionsinformation (1) (siehe Abbildung [4.5]) und das benutzte Datenformat (2). Der Datenteil (3) enthält die kodierten Daten in redundanter Form. Zur Feldbegrenzung enthält der QR-Code in nur drei seiner Ecken ein bestimmtes Muster (4.1). Über das fehlende Muster in der vierten Ecke erkennt das Lesegerät die Orientierung. Mit zunehmender Größe des Codes werden weitere Muster (4.2) hinzugefügt, um die Ausrichtung des Codes besser erkennbar zu machen. Zwischen den drei Hauptpositionsmarkierungen befindet sich eine Linie (4.3) aus einer Folge streng abwechselnder Bits, worüber sich die Matrix definiert.

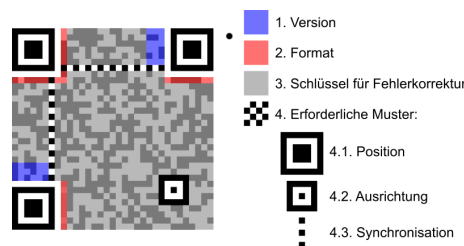


Abbildung 4.5: Der strukturelle Aufbau des QR-Codes.

Die Symbolelemente sind Quadrate, von denen sich mindestens 21×21 und maximal 177×177 Elemente im Symbol befinden. Die Randzone (quiet zone) sollte mindestens 4 Elemente breit sein. Größere Inhalte lassen sich auf bis zu 16 einzelne Codes aufteilen.

Es existieren vier Fehlerkorrektur-Levels, die eine Rekonstruktion von 7 % (Level L) bis zu 30 % (Level H) beschädigter Daten zulassen. Dabei kommt die Fehlerkorrektur der Reed-Solomon-Codierung zum Einsatz. Diese Eigenschaft wird bei der Erstellung sogenannter Design Codes ausgenutzt.

Kapazität der verschiedenen Fehlerkorrektur-Levels:

- Level L 7 % der Codewörter/Daten können wiederhergestellt werden.
- Level M 15 % der Codewörter/Daten können wiederhergestellt werden.
- Level Q 25 % der Codewörter/Daten können wiederhergestellt werden.
- Level H 30 % der Codewörter/Daten können wiederhergestellt werden.

Der maximale Informationsgehalt eines QR-Codes (177×177 Elemente, Fehlerkorrektur-Level L) beträgt 23.624 Bit (2.953 Byte). Damit lassen sich laut Hersteller 7.089 Dezimalziffern, 4.296 alphanumerische Zeichen oder 1.817 Kanji-/Kana-Zeichen kodieren.

Der Micro-QR-Code mit einer Größe zwischen 11×11 und 17×17 Elementen nimmt bis zu 35 Ziffern auf bei einer Randbreite von mindestens 2 Elementen.

Kapitel 5

Zahlen und Stellenwertsysteme

5.1 Einführung

Im Lauf der Kulturgeschichte der Menschheit sind sehr unterschiedliche Systeme und Verfahren entwickelt worden, wie man Zahlen darstellen kann.¹ Heute unterscheidet man zwischen drei Systemen:

- Additionssystemen,
- hybride Zahlensysteme,
- und Stellenwertsysteme.

In der Abbildung [5.1] ist diese Unterteilung nochmals skizziert.

Zunächst erst mal eine Definition, was wir unter Zahlensysteme verstehen.

Definition 5.7:

Ein **Zahlensystem** wird dazu verwendet, Zahlen darzustellen. Eine bestimmte Zahl wird gemäß den in dem verwendeten System bestehenden Regeln gebildet.

¹Siehe dazu beispielsweise das Buch von IFRAH [59] oder die Monographie von KATZ, [63]. Eine umfassende und tiefgehende Darstellung dieser Thematik — insbesondere die Arithmetik — findet man in dem zweiten Band von DONALD KNUTH [66], Chapter 4.

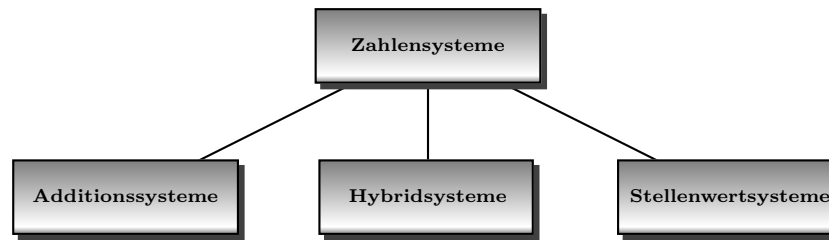


Abbildung 5.1: Unterteilung der Zahlensysteme.

5.1.1 Additionssysteme

Der einfachste Typ von Zahlensystemen sind die additiven Systeme. Bei Additionssystemen ergibt sich der Wert einer Zahl durch Addition der Werte ihrer Ziffern. Die Position einer Ziffer in der Zahl spielt hierbei keine Rolle.

Das einfachste Additionssystem ist das **unäre Zahlensystem**, mit einer Ziffer $|$. Eine Zahl in diesem Zahlensystem ist eine Folge von mehreren Strichen, beispielsweise ist

$$\begin{aligned}
 1 &\longleftrightarrow | \\
 2 &\longleftrightarrow || \\
 3 &\longleftrightarrow ||| \\
 &\vdots
 \end{aligned}$$

Der *Wert* einer Zahl im unären Zahlensystem ist die Anzahl der Striche.

Das unäre Zahlensystem wird — neben Bierdeckeln — in der Theoretischen Informatik verwendet, um Zahlen möglichst einfach kodieren zu können, die von sog. TURING-Maschinen bearbeitet werden.

Das unäre Zahlensystem hat den Nachteil,

- dass die Darstellung großer Zahlen sehr unübersichtlich wird. Aus diesem Grund entsteht die Notwendigkeit, neben dem Symbol $|$ weitere Zeichen einzuführen, mit denen die Zahlen dargestellt werden,
- es gibt keine 0.

Um das erste Problem zu umgehen, enthält beispielsweise das römische Zahlen-

system sieben Ziffern, mit

$$\begin{aligned} | &\longleftrightarrow 1, \\ V &\longleftrightarrow 5, \\ X &\longleftrightarrow 10, \\ L &\longleftrightarrow 50, \\ C &\longleftrightarrow 100, \\ D &\longleftrightarrow 500, \\ M &\longleftrightarrow 1000. \end{aligned}$$

Jedoch hat das römische Zahlensystem ebenfalls keine Ziffer für die 0.

Mit solchen additiven Systemen lassen sich Zahlen durchaus darstellen, die Durchführung der Arithmetik — also Addition, Subtraktion, Multiplikation und Division ist in der Regel jedoch kompliziert.

5.1.2 Hybridsysteme

Abweichend und zusätzlich zu einem additiven Zahlensystem enthält ein hybrides Zahlensystem eine sogenannte **Basis** — zum Beispiel die Basis 12. Eine Zahl in solch einem Zahlensystem wird dadurch konstruiert, dass eine Ziffer den Potenzen dieser Basis vorangestellt wird. Die Werte beider Bestandteile werden multipliziert.

5.1.3 Stellenwertsysteme

Im alltäglichen Leben ist man es gewohnt, in dem sogenannten **Dezimalsystem** zu arbeiten. Im Dezimalsystem werden mit Hilfe der zehn Ziffern $0, 1, 2, \dots, 9$ Zahlen gebildet. Um zu sehen, wie die Regeln der Bildung einer Zahl im Dezimalsystem aussehen, betrachten wir einmal genauer, was die *Zahl* 96 bedeutet. Diese Schreibweise bedeutet nichts anderes als neun mal 10 plus 6:

$$96 = (9 \times 10) + 6.$$

Betrachten wir eine etwas größere Zahl, die Zahl 6745 bedeutet: 6 mal 1000 plus 7 mal 100 plus 4 mal 10 plus 5:

$$\begin{aligned} 6745 &= (6 \times 1000) + (7 \times 100) + (4 \times 10) + 5 \\ &= 6 \times 10^3 + 7 \times 10^2 + 4 \times 10^1 + 5 \times 10^0. \end{aligned}$$

Wir beobachten folgendes: Die einzelnen Ziffern der Folge von Ziffern (wie 6745) werden mit den Potenzen von 10 multipliziert. Dabei legt die Stelle der Ziffer fest, um welche Potenz es sich handelt. Aus diesem Grund nennt man das Dezimalsystem ein **Stellenwertsystem** zur **Basis** 10. Halten wir dies in einer Definition fest und verallgemeinern wir diese Art und Weise, Zahlen darzustellen.²

²Eine mathematisch tieferegehende Diskussion dieser Thematik findet man in dem Buch von REISS und SCHMIEDER, [87], Kapitel 3.

Definition 5.8:

Bei einem Stellenwertsystem mit der Basis B und den B Ziffern im Intervall $0, 1, \dots, B-1$ hängt der Wert einer Ziffer nicht nur von der Form des Zeichens sondern auch von der Stelle in der Zahl ab. Die Ziffern einer Zahl eines Stellenwertsystems stellen die Koeffizienten der Potenzen der Basis dar. Dabei steht die niedrigwertigste Ziffer (in der Regel) ganz rechts.

Eine Zahl im Stellenwertsystem des Dezimalsystems bedeutet also, dass jede Ziffer der Zahl multipliziert wird mit der Potenz von 10; diese ist also durch die Stelle der Ziffer festgelegt.

$$96 = (9 \times 10^1) + (6 \times 10^0),$$

$$6745 = (6 \times 10^3) + (7 \times 10^2) + (4 \times 10^1) + (5 \times 10^0).$$

Diese Entwicklung gilt auch für Dezimalbrüche:

$$564,887 = (5 \times 10^2) + (6 \times 10^1) + (4 \times 10^0) + (8 \times 10^{-1}) + (8 \times 10^{-2}) + (7 \times 10^{-3}).$$

Dieses Beispiel zeigt, dass das Komma ein Indikator dafür ist, dass die Potenzen der Basis 10 das Vorzeichen ändert.

Generell hat man folgendes Szenario: Die *Dezimaldarstellung* von einer Zahl z ist

$$z = x_{n-1} \dots x_2 x_1 x_0 x_{-1} x_{-2} \dots x_{-m}, \quad x_i \in \{0, 1, \dots, 9\}, i = -m, \dots, n-1.$$

Diese Zahl hat den *Zahlenwert*

$$z = \sum_{i=-m}^{n-1} x_i \cdot 10^i = \sum_{i=0}^{n-1} x_i \cdot 10^i + \sum_{i=-m}^{-1} x_i \cdot 10^i. \quad (5.1)$$

Was wir hier für das bekannte Dezimalzahlensystem untersucht haben, kann man nun allgemeiner fassen. Ein Stellenwertsystem — auch **Positionssystem** oder **polyadisches Zahlensystem** genannt — kann mit einer begrenzten Anzahl von Zeichen beliebig große Zahlen darstellen. Die Ziffern einer Zahl sind die Koeffizienten der Potenzen einer Basis B . Jede ganze Zahl $B \geq 2$ ist als Basis eines Stellenwertsystems geeignet. Für ein Stellenwertsystem zu einer beliebigen Basis B haben wir in Analogie zur Darstellung (5.1) die Entwicklung:

$$z = \sum_{i=-m}^{n-1} x_i B^i = \underbrace{\sum_{i=0}^{n-1} x_i B^i}_{\text{ganzzahliger Anteil}} + \underbrace{\sum_{i=-m}^{-1} x_i B^i}_{\text{Nachkomma-Anteil}}. \quad (5.2)$$

Hierbei ist

z darzustellende Zahl,

x_i Wert der Ziffer an der Stelle i , $x_i \in [0, 1, \dots, B - 1]$,

B Basis der Zahlensystems,

n Länge des ganzzahligen Anteils,

m Länge des Nachkommaanteils.

Es kann also eine beliebige natürliche Zahl größer oder gleich 2, *i.e.* $2 \leq B \in \mathbb{N}$ als Basis verwendet werden. In der Praxis werden folgende Zahlensysteme verwendet:

☞ das **Binärsystem** oder auch **Dualsystem** — manchmal verwendet man auch den Ausdruck **dyadisches Zahlensystem** — mit der Basis $B = 2$ und Ziffern 0, 1,

☞ das **Oktalsystem** mit der Basis $B = 8$ und den Ziffern 0, 1, 2, ..., 7,

☞ das **Dezimalsystem** mit der Basis $B = 10$ und den Ziffern 0 – 9,

☞ das **Hexadezimalsystem** mit der Basis $B = 16$ mit den Ziffern

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F,

☞ und das **Sexagesimalsystem** mit der Basis $B = 60$.

Beispiel 5.12:

Die folgende Tabelle listet die ersten 18 Zahlen in den verschiedenen Zahlensystemen auf.

Dezimal	Oktal	Dual	Hexadezimal
0	0	0	0
1	1	1	1
2	2	10	2
3	3	11	3
4	4	100	4
5	5	101	5
6	6	110	6
7	7	111	7
8	10	1000	8
9	11	1001	9
10	12	1010	A
11	13	1011	B
12	14	1100	C
13	15	1101	D
14	16	1110	E
15	17	1111	F
16	20	10000	10
17	21	10001	11

Beispiel 5.13:

Oftmals ist es von Interesse, zu wissen, wie viele Stellen eine gegebene Zahl in einem bestimmten Zahlensystem hat. In der Kryptographie ist beispielsweise die Größe eines Schlüssels — das ist in der Regel eine Zahl — entscheidend für die Sicherheit eines Systems. So muss man wissen, wieviele Bits notwendig sind, um eine Zahl wie $z = 1.234.567.890$ abzuspeichern.

Für eine Zahl z mit n Stellen gilt:

$$z < B^n.$$

Dann ist auch

$$\ln z < \ln(B^n) = n \cdot \ln(B),$$

woraus wir erhalten

$$n > \frac{\ln z}{\ln B}.$$

Dies sagt, aus, um die Zahl z im Stellenwertsystem zur Basis B darzustellen, benötigt man mindestens $\ln z / \ln B$ Stellen.

5.2 Das Binärsystem

Im Dezimalsystem werden also zehn verschiedene Ziffern benutzt, um Zahlen zur Basis 10 darzustellen. Im **Binärsystem** hat man lediglich zwei Ziffern 0 und 1 zur Verfügung. Die Zahlen im Binärsystem werden zur Basis 2 dargestellt.

Um Mißverständnisse über die Bedeutung einer Zahl zu vermeiden — ist 100 eine Zahl im Dezimal- oder im Binärsystem??? — fügt man oftmals die Basis als Index an die Zahl an.

Notation:

Binärzahlen werden durch ein angehängtes oder tiefgestelltes **b** oder durch eine angehängte und tiefgestellte '2' gekennzeichnet.

Beispiel: $1\ 101b$ oder 1101_2

Analog kennzeichnet man Dezimalzahlen entweder mit einem angehängten d oder angehängten und tiefgestellten '10'. Hexadezimalzahlen werden mit einem angehängten h oder einem vorgestellten '0x' gekennzeichnet.

Die Binärzahlen 0 und 1 haben die gleiche Bedeutung wie im Dezimalsystem:

$$\begin{aligned}0_2 &= 0_{10} \\1_2 &= 1_{10}\end{aligned}$$

Um größere Zahlen darzustellen, hat jede Ziffer einer Binärzahl einen Wert, der von der Stelle der Ziffer in der Zahl abhängt. So ist

$$\begin{aligned}10_2 &= (1 \times 2^1) + (0 \times 2^0) = 2_{10} \\11_2 &= (1 \times 2^1) + (1 \times 2^0) = 3_{10} \\100_2 &= (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 4_{10}\end{aligned}$$

Eine Binärzahl, die aus einer Folge von 0en und 1en besteht, ist in analoger Weise zu Dezimalzahlen also:

$$\begin{aligned}z &= 10100111_2 \\&= 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0,\end{aligned}$$

das heißt die 0en und 1en in der Ziffernfolge $z = 10100111_2$ stellen die Koeffizienten der Potenzen von 2 dar. Im Binärsystem werden Kommazahlen — wie im Dezimalsystem — mit negativen Potenzen der Basis dargestellt:

$$10011.1101_2 = 2^4 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4}.$$

5.2.1 Umrechnungen Dezimalsystem \iff Binärsystem

Es ist natürlich interessant, wie man die Darstellung einer Zahl von einem System in ein anderes Zahlensystem umrechnen kann — man beachte, es handelt

sich um *ein und die selbe Zahl* die in verschiedenen Zahlensystemen dargestellt wird. Sehen wir uns daher an, wie man vom Dezimalsystem in das Dualsystem umrechnet und umgekehrt.

Wir betrachten zunächst die Umwandlung des ganzzahligen Anteils.

1. Umrechnung Binärsystem \rightarrow Dezimalsystem

Gegeben ist also die Darstellung einer Zahl in binärer Notation, gesucht ist die Dezimalnotation. Die ist recht einfach, alles was dazu erforderlich ist, ist jede binäre Ziffer mit der geeigneten Potenz von 2 zu multiplizieren und die Resultate zu addieren.

$$\begin{aligned} z &= 10100111_2 \\ &= 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 2^7 + 2^5 + 2^2 + 2^1 + 2^0 \\ &= 128 + 32 + 4 + 2 + 1 \\ &= 167_{10}. \end{aligned}$$

Anmerkung:

Da die niedrigwertigste Stelle einer Zahl im Binärsystem der Koeffizient der Potenz 2^0 darstellt, ist die niedrigwertigste Ziffer einer geraden Zahl eine 0, eine ungerade Zahl endet mit 1.

Ist eine Binärzahl mit Nachkommastellen umzuwandeln, verfährt man in genau der gleichen Weise.

Beispiel 5.14:

Betrachte die Binärzahl

$$z = 0.1001101_2.$$

Gesucht ist die Darstellung dieser Zahl als Dezimalbruch. Dazu wendet man an, dass die 0en und 1en die Koeffizienten der Potenzen von 2 sind, also

$$\begin{aligned} z &= 0.1001101_2 \\ &= 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \\ &\quad + 1 \times 2^{-5} + 0 \times 2^{-6} + 1 \times 2^{-7} \\ &= 2^{-1} + 2^{-4} + 2^{-5} + 2^{-7} \\ &= \frac{1}{2} + \frac{1}{16} + \frac{1}{32} + \frac{1}{128} \\ &= 0.6015625_{10}. \end{aligned}$$

2. Umrechnung Dezimalsystem \rightarrow Binärsystem

Die umgekehrte Richtung bedeutet also: Gegeben ist die Darstellung einer Zahl im Dezimalsystem, gesucht ist die Binärdarstellung dieser Zahl.

Sehen wir uns zunächst das allgemeine Verfahren dieser Umrechnung für den ganzzahligen Anteil an.

Wir nehmen also an, die ganze Zahl z soll in Binärdarstellung umgerechnet werden. Dividieren wir z durch 2 — im Dezimalsystem — dann erhalten einen Quotienten z_1 und einen Rest R_0 . Daher können wir setzen:

$$z = 2 \times z_1 + R_0 \quad R_0 = 0 \text{ oder } 1.$$

Im nächsten Schritt dividieren wir den Quotienten z_1 durch 2. Angenommen der neue Quotient ist z_2 und der neue Rest R_1 . Dann folgt:

$$z_1 = 2 \times z_2 + R_1 \quad R_1 = 0 \text{ oder } 1.$$

Setzt man dieses Resultat in die obere Division ein, folgt:

$$\begin{aligned} z &= 2 \times (2 \times z_2 + R_1) + R_0 \\ &= z_2 \times 2^2 + R_1 \times 2^1 + R_0 \times 2^0. \end{aligned}$$

Setzen wir für den nächsten Schritt

$$z_2 = 2 \times z_3 + R_2 \quad R_2 = 0 \text{ oder } 1,$$

dann erhalten wir:

$$z = z_3 \times 2^3 + R_2 \times 2^2 + R_1 \times 2^1 + R_0 \times 2^0.$$

Da $z > z_1 > z_2 \dots$ führt eine Fortsetzung dieser Division durch 2 schließlich auf einen Quotienten $z_k = 1$ und einen Rest R_{k-} , der 0 oder 1 ist. Damit erhält man die Darstellung:

$$z = 1 \times 2^k + R_{k-1} \times 2^{k-1} + \dots + R_2 \times 2^2 + R_1 \times 2^1 + R_0 \times 2^0.$$

Das bedeutet, dieses Verfahren³ liefert systematisch die Umrechnung einer im Dezimalsystem gegebenen Zahl in eine Darstellung im Binärsystem.

Beispiel 5.15:

- ① Wir berechnen mit obigem Verfahren die Binärdarstellung der Dezimalzahl $z = 47_{10}$:

³In der Informatik nennt man ein solches systematisches Verfahren einen **Algorithmus**.

Die formale Definition eines Algorithmus lautet folgendermaßen:

$$\begin{array}{rclcl}
47 \text{ div } 2 & = & 23 \text{ Rest } 1 & \implies & R_0 = 1 \\
23 \text{ div } 2 & = & 11 \text{ Rest } 1 & \implies & R_1 = 1 \\
11 \text{ div } 2 & = & 5 \text{ Rest } 1 & \implies & R_2 = 1 \\
5 \text{ div } 2 & = & 2 \text{ Rest } 1 & \implies & R_3 = 1 \\
2 \text{ div } 2 & = & 1 \text{ Rest } 0 & \implies & R_4 = 0 \\
1 \text{ div } 2 & = & 0 \text{ Rest } 1 & \implies & R_5 = 1
\end{array}$$

Damit ist die Binärdarstellung der Dezimalzahl 47_{10} :

$$47_{10} = 10 \ 1111_2$$

Das Resultat kann man dadurch überprüfen, dass man die erhaltene Binärzahl wieder in das Dezimalsystem zurückrechnet.

$$\begin{aligned}
10 \ 1111_2 &= 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\
&= 32 + 8 + 4 + 2 + 1 \\
&= 47_{10}
\end{aligned}$$

- ② In einem zweiten Beispiel demonstrieren wir diesen Algorithmus für die Zahl $z = 1997_{10}$:

Definition: **Deterministischer Algorithmus**

Ein **deterministischer Algorithmus** ist eine Funktion

$$f : D_{in} \longrightarrow D_{out}$$

die *Eingabedaten* oder *Eingabenachrichten* D_{in} in *Ausgabedaten* oder *Ausgabenachrichten* D_{out} abbildet und dabei die folgenden sechs Bedingungen erfüllt:

- ① **Exaktheit**
Die Funktion f kann präzise auf formale Weise beschrieben werden.
- ② **Fintheit**
Die Beschreibung von f ist endlich lang.
- ③ **Vollständigkeit**
Die Beschreibung von f ist vollständig, i.e. sie umfaßt sämtliche Einzelfälle.
- ④ **Effektivität**
Die Einzelschritte von f sind elementar und real ausführbar.
- ⑤ **Terminierung**
Die Funktion f hält nach *endlich* vielen Schritten an und liefert ein Resultat. Damit wird die *Haltebedingung* erfüllt.
- ⑥ **Determinismus**
Die Funktion f liefert für die gleichen Eingabewerte stets das gleiche Ergebnis, wobei die Folge der Einzelschritte für jeden Eingabewert genau festgelegt sind.

$$\begin{array}{rclcl}
1997 \text{ div } 2 & = & 998 \text{ Rest } 1 & \implies & R_0 = 1 \\
998 \text{ div } 2 & = & 499 \text{ Rest } 0 & \implies & R_1 = 0 \\
499 \text{ div } 2 & = & 249 \text{ Rest } 1 & \implies & R_2 = 1 \\
249 \text{ div } 2 & = & 124 \text{ Rest } 1 & \implies & R_3 = 1 \\
124 \text{ div } 2 & = & 62 \text{ Rest } 0 & \implies & R_4 = 0 \\
62 \text{ div } 2 & = & 31 \text{ Rest } 0 & \implies & R_5 = 0 \\
31 \text{ div } 2 & = & 15 \text{ Rest } 1 & \implies & R_6 = 1 \\
15 \text{ div } 2 & = & 7 \text{ Rest } 1 & \implies & R_7 = 1 \\
7 \text{ div } 2 & = & 3 \text{ Rest } 1 & \implies & R_8 = 1 \\
3 \text{ div } 2 & = & 1 \text{ Rest } 1 & \implies & R_9 = 1 \\
1 \text{ div } 2 & = & 0 \text{ Rest } 1 & \implies & R_{10} = 1
\end{array}$$

Daher lautet die Binärdarstellung der Zahl $z = 1997_{10}$:

$$1997_{10} = 111 \ 1100 \ 1101_2$$

Die Umrechnung eines **Dezimalbruchs** in die Binärdarstellung erfordert die wiederholte *Multiplikation* mit 2. Bei jedem Schritt werden die Nachkommastellen des Dezimalbruchs mit 2 multipliziert. Die Ziffer links vom Dezimalpunkt des Produktes ist entweder 0 oder 1 und trägt zur Binärdarstellung der Zahl bei. Hier startet man mit dem höchstwertigen Bit. Der Bruchteil — also die Nachkommastellen des Produkts — wird als Input (Multiplikant) des nächsten Schrittes benutzt.

Um zu sehen, dass dieses Verfahren funktioniert, betrachten wir einen positiven Dezimalbruch $F < 1$. Diesen Bruch können wir in der Form schreiben:

$$F = \left(b_{-1} \times \frac{1}{2}\right) + \left(b_{-2} \times \frac{1}{2^2}\right) + \left(b_{-3} \times \frac{1}{2^3}\right) + \dots$$

wobei jeder Koeffizient b_{-i} entweder den Wert 0 oder 1 hat. Multipliziert man diese Entwicklung mit dem Faktor 2 folgt:

$$2 \cdot F = b_{-1} + \left(b_{-2} \times \frac{1}{2}\right) + \left(b_{-3} \times \frac{1}{2^2}\right) + \left(b_{-4} \times \frac{1}{2^3}\right) \dots$$

Der ganzzahligen Anteil der beiden Seiten muss gleich sein. Daher ist der ganzzahligen Anteil von $2 \cdot F$ — der entweder 0 oder 1 ist, da $0 < F < 1$ — ist einfach b_{-1} . Damit ist

$$2 \cdot F = b_{-1} + F_1,$$

mit $0 < F_1 < 1$ und

$$F_1 = \left(b_{-2} \times \frac{1}{2}\right) + \left(b_{-3} \times \frac{1}{2^2}\right) + \left(b_{-4} \times \frac{1}{2^3}\right) + \dots$$

Um nun der Koeffizienten b_{-2} zu erhalten, wiederholen wir den Prozess. Dieses Verfahren ist *nicht* notwendigerweise exakt. Das heißt, die Darstellung eines Dezimalbruchs mit endlich vielen Dezimalstellen kann einen Binärbruch mit einer unendlichen Anzahl von Ziffern erfordern. In solchen Fällen terminiert der Umwandlungsalgorithmus nach einer vorgegebenen Anzahl von Stellen. Diese Anzahl hängt von der *Genauigkeit* der Zahldarstellung ab.

Beispiel 5.16:

- ① In einem ersten Beispiel berechnen wir mit Hilfe dieses Verfahrens die Binärdarstellung des Dezimalbruchs

$$z = 0.828125_{10}.$$

Es folgt:

$$\begin{array}{llll} 0.828125 \times 2 & = 1.65625 & \implies & \text{Output: 1} \\ 0.65625 \times 2 & = 1.3125 & \implies & \text{Output: 1} \\ 0.3125 \times 2 & = 0.625 & \implies & \text{Output: 0} \\ 0.625 \times 2 & = 1.25 & \implies & \text{Output: 1} \\ 0.25 \times 2 & = 0.5 & \implies & \text{Output: 0} \\ 0.5 \times 2 & = 1.0 & \implies & \text{Output: 1} \end{array}$$

Damit erhalten wir:

$$z = 0.828125_{10} = 0.110101_2$$

Check: Um das Resultat zu überprüfen, rechnen wir die Binärdarstellung in das Dezimalsystem um:

$$\begin{aligned} 0.110101_2 &= 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-4} + 1 \cdot 2^{-6} \\ &= \frac{1}{2} + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} \\ &= 0.5_{10} + 0.25_{10} + 0.0625_{10} + 0.015625_{10} \\ &= 0.828125_{10} \end{aligned}$$

- ② Das zweite Beispiel zeigt die Umrechnung des Dezimalbruchs

$$z = 0.81_{10}$$

in die Binärdarstellung. Wendet man das oben beschriebene Verfahren an, erhält man:

0.81×2	$= 1.62$	\implies	Output: 1
0.62×2	$= 1.24$	\implies	Output: 1
0.24×2	$= 0.48$	\implies	Output: 0
0.48×2	$= 0.96$	\implies	Output: 0
0.96×2	$= 1.92$	\implies	Output: 1
0.92×2	$= 1.84$	\implies	Output: 1
0.84×2	$= 1.68$	\implies	Output: 1
0.68×2	$= 1.36$	\implies	Output: 1
0.36×2	$= 0.72$	\implies	Output: 0
0.72×2	$= 1.44$	\implies	Output: 1

usw. Dieses Verfahren endet nicht. Arbeitet man daher mit der Genauigkeit von 10 Binärstellen, ist die Darstellung des Dezimalbruchs 0.81_{10} :

$$0.81_{10} = 0.1100\ 1111\ 01_2$$

Um den Fehler zu ermitteln, der durch diese Umwandlung auftritt, rechnet man den Binärbruch in das Dezimalsystem zurück:

$$\begin{aligned} 0.1100\ 1111\ 01_2 &= 2^{-1} + 2^{-2} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-8} + 2^{-10} \\ &= \frac{1}{2} + \frac{1}{4} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256} + \frac{1}{1024} \\ &= 0.8095703125_{10} \end{aligned}$$

Der Fehler, der auftritt, ist:

$$\Delta = 0.81_{10} - 0.8095703125_{10} = 0.0004296875_{10}$$

Übung 5.1:

Berechnen Sie die Binärdarstellung des Dezimalbruchs $\frac{1}{10}$.

5.2.2 Arithmetik

In sämtlichen Stellenwertsystemen gelten die gleichen Rechenregeln wie man sie aus dem Dezimalsystem kennt. Wir sehen uns hier im Detail die Arithmetik im Binärsystem an.

(a) Addition

Wie andere Zahlen auch, können Binärzahlen natürlich auch addiert werden. Dies funktioniert genau wie die Addition von Dezimalzahlen, ausgenommen der Tatsache, dass bereits ein Übertrag entsteht, sobald das

Ergebnis größer als 1 ist. Die Tabelle [5.1] legt die grundlegende Logik der Addition im Binärsystem fest. Die Größen **a** und **b** sind dabei Platzhalter für 0 oder 1.⁴ Durch die Addition zweier Bits entsteht ein Summenbit **a + b**, sowie einen Übertrag in die nächste Stelle. Diesen Übertrag nennt man auch *carry out Bit*, oder *c_o-Bit*.

Die Schaltungslogik, die in der Tabelle [5.1] aufgeführt ist, ist die Schaltlo-

a	b	a+b	Übertrag
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabelle 5.1: Addition von 0 und 1.

gik eines **Halbaddierers**. Halbaddierer berücksichtigen keinen Übertrag aus einer vorangehenden Stelle. Zwei aus mehreren Ziffern bestehende Binärzahlen werden addiert, wie man es von der Addition zweier Dezimalzahlen kennt. Ein an einer Ziffernposition entstehender Übertrag wird zur nächsthöheren Position addiert. Dies sind im wesentlichen die in der Tabelle [5.1] gegebenen Regeln.

Beispiel 5.17:

$$\begin{array}{r}
 101 \\
 + \quad 11 \\
 \text{Übertrag} \quad \color{red}{111} \\
 = \quad 1000
 \end{array}$$

was dem allseits bekannten Resultat $5 + 3 = 8$ entspricht.

$$\begin{array}{r}
 1011 \ 0001 \\
 + \quad 101 \ 0111 \\
 \text{Übertrag} \quad \color{red}{111} \ \color{red}{111} \\
 = \quad 1 \ 0000 \ 1000
 \end{array}$$

was in Dezimalzahlen der Addition $177 + 87 = 264$ entspricht.

⁴In der Informatik nennt man solche Platzhalter **Bits**, in der Mathematik BOOLEsche Variable.

Wie diese Beispiele zeigen ist die Addition zweier Bits mit Berücksichtigung eines Übertrags aus einer vorhergehenden Stelle — diesen Übertrag nennt man *carry in Bit* oder kurz c_{in} -Bit — durch die Logik eines sogenannten **Volladdierers** beschreibbar. Die Tabelle [5.2] zeigt diese Logik.

a	b	c_{in}	$\text{sum}(a, b, c_{in})$	$c_{out}(a, b, c_{in})$
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	0

Tabelle 5.2: Schaltlogik eines Volladdierers.

(b) **Subtraktion**

Die Subtraktion zweier Binärzahlen geschieht nach dem gleichen Verfahren, wie man es von Dezimalzahlen kennt.

Beispiel 5.18:

In diesem Beispiel subtrahieren wir die Zahl 13 von der Zahl 22, beide Zahlen dargestellt im Binärsystem.

$$\begin{array}{r}
 10110 \\
 - 1101 \\
 \hline
 \text{Borgen } 1 \quad 1 \\
 1001
 \end{array}$$

Diese Art der Subtraktion wird in Computersystemen nicht verwendet. Der Grund liegt darin, dass zur Durchführung dieser Operation ein eigenes Subtraktionswerk in der Recheneinheit des Prozessors benötigt wird. Optimaler ist es, eine geeignete Zahlendarstellung für negative Zahlen zu finden, so dass die Subtraktion auf die Addition zurückgeführt werden kann. Dadurch ist die Hardware einfach gehalten, denn eine CPU benötigt dann lediglich ein Addierwerk, um Addition und Subtraktion auszuführen. Diesen Aspekt behandeln wir ausführlich in Abschnitt [5.3].

(c) **Multiplikation**

Die Multiplikation zweier vorzeichenloser Binärzahlen führt man nach dem gleichen Schema aus, wie man es vom Dezimalsystem kennt, im Binärsystem reduziert sich dies — salopp gesagt — einfach auf 'hinschreiben' oder 'nicht hinschreiben'.

Beispiel 5.19:

In diesem Beispiel multiplizieren wir die beiden Binärzahlen 10110_2 ($= 22_{10}$) und 101_2 ($= 5_{10}$).

$$\begin{array}{r}
 10110 \times 101 \\
 \hline
 10110 \\
 00000 \\
 10110 \\
 \hline
 1101110
 \end{array}$$

Das Beispiel [5.19] illustriert die Multiplikation zweier vorzeichenloser Binärzahlen mit der Bleistift und Papier Methode. Man beachte folgende Aspekte:

- (a) Die Multiplikation führt auf die Erzeugung von Partialprodukten, ein Partialprodukt für jede Ziffer des Multiplikators. Diese Teilprodukte werden letztendlich summiert um das Produkt der beiden Ausgangszahlen zu erhalten
- (b) Die Teilprodukte ergeben sich einfach: Ist das Multiplikatorbit 0, dann ist das Teilprodukt einfach 0. Ist das Multiplikatorbit 1, ist das Teilprodukt der Multiplikand.
- (c) Das Gesamtprodukt ergibt sich aus der Summe der Teilprodukte. Für diese Operation werden die aufeinanderfolgenden Partialprodukte eine Stelle nach links verschoben relativ zum vorherigen Teilprodukt.
- (d) Die Multiplikation zweier n -Bit Zahlen führt auf einen Bitstring der Länge $2n$ Bit.

Im Vergleich zur Bleistift und Papier Methode gibt es einige Dinge, die man tun kann um die Multiplikation effektiver zu machen.

- Zunächst kann man eine laufende Aufsummierung der Partialprodukte durchführen anstatt zu warten, bis alle Teilprodukte berechnet wurden und dann die Summe zu bilden. Dies eliminiert das Speicherproblem für alle Teilprodukte, man benötigt eine geringere Anzahl von Registern.

- Zweitens kann man bei der Erzeugung der Teilprodukte Zeit sparen. Falls das Multiplikatorbit eine 1 ist, sind eine Addition und eine Shiftoperation erforderlich. Ist das Multiplikatorbit 0, benötigt man nur eine Shiftoperation.

Die Abbildung [5.2] zeigt eine mögliche Implementierung dieser Ideen. Der Multiplikator und der Multiplikand werden in zwei Register geladen, die wir Q (Multiplikator) und M (Multiplikand) nennen. Wir benötigen ein weiteres n -Bit Register, das wir A nennen und mit 0 initialisieren. Dieses zusätzliche Register ist aus dem Grund erforderlich, weil die Multiplikation einer n Bit Zahl mit einer m Bit Zahl auf ein Ergebnis mit $n + m$ Stellen führt. Weiterhin wird ein 1-Bit Register C benötigt, das den potentiell bei der Addition der Partialsummen entstehenden Übertrag speichert. Wir initialisieren diese Register ebenfalls mit 0.

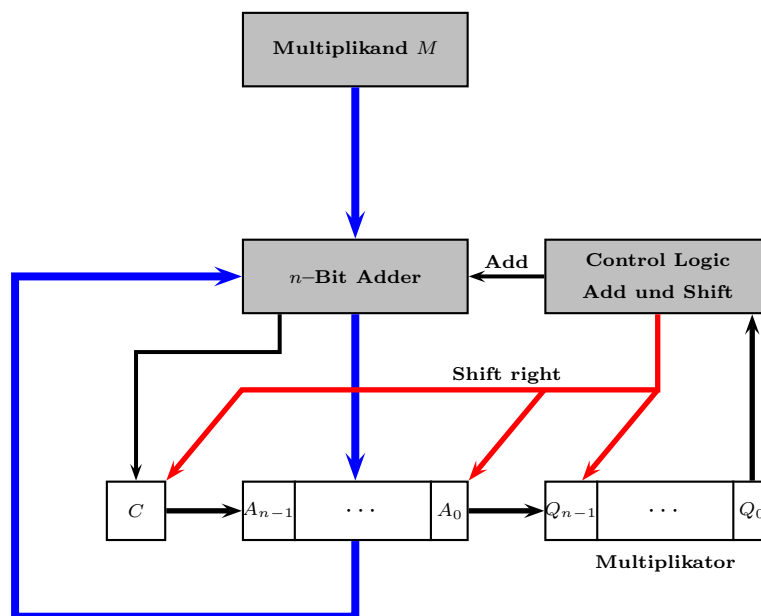


Abbildung 5.2: Blockdiagramm für die Hardware-Implementierung der Multiplikation für vorzeichenlose Ganzzahlen.

Die Multiplikationsoperation läuft folgendermaßen ab. Die Control Logik

liest bei jedem Schritt die Bits des Multiplikators im Register Q .

- Wenn Q_0 den Wert 1 hat, wird der Multiplikand zu dem A Register addiert. Das Ergebnis wird im A Register gespeichert, ein eventuell entstehender Überlauf in die n -te Stelle wird im C Register gespeichert. Dann werden alle Bits der Register C , A und Q ein Bit nach rechts verschoben, *i.e.* das C Bit wird in das A_{n-1} verschoben, das A_0 Bit in Q_{n-1} , Q_0 fällt weg.
- Wenn Q_0 den Wert 0 hat, wird keine Addition durchgeführt, lediglich der Right Shift.

Dieser Prozess wird für jedes Bit des ursprünglichen Multiplikators wiederholt. Das resultierende $2n$ -Bit Resultat liegt in den Registern A und Q .

Die Abbildung [5.3] zeigt das Flußdiagramm dieser Operation.

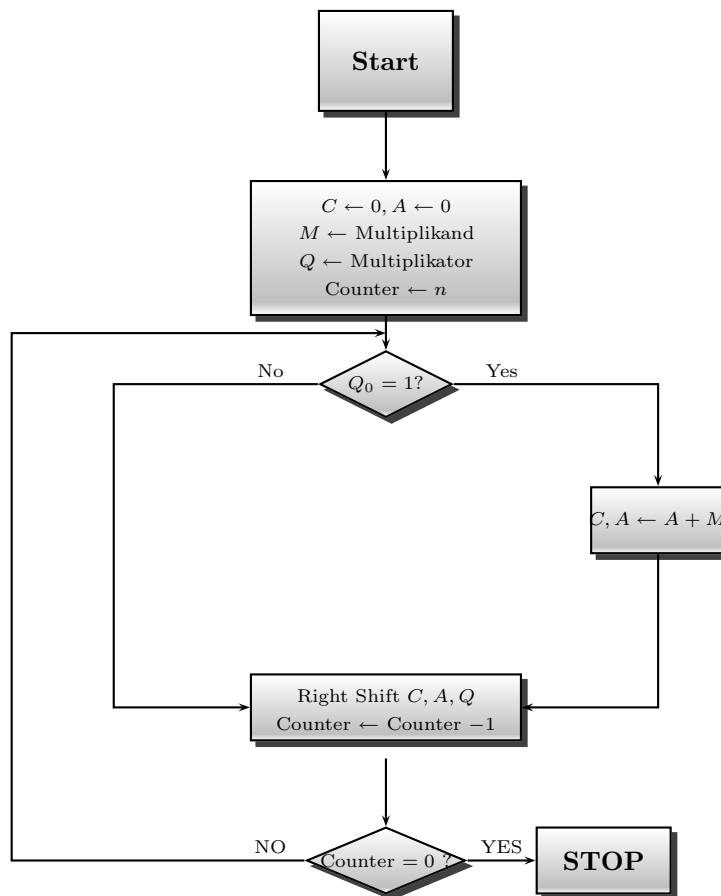


Abbildung 5.3: Flußdiagramm für die Multiplikation zweier vorzeichenloser Binärzahlen.

Beispiel 5.20:

In diesem Beispiel führen wir die Multiplikation 11×13 mit Hilfe dieses Verfahrens durch.

Register C	Register A	Register Q	Register M	
0	0000	1101	1011	Initialisierung

1. Step $Q_0 = 1$:

Register C	Register A	Register Q	Register M	
0	1011	1101	1011	Add
0	0101	1110	1011	Shift

2. Step $Q_0 = 0$:

Register C	Register A	Register Q	Register M	
0	0010	1111	1011	Shift

3. Step $Q_0 = 1$:

Register C	Register A	Register Q	Register M	
0	1101	1111	1011	Add
0	0110	1111	1011	Shift

4. Step $Q_0 = 1$:

Register C	Register A	Register Q	Register M	
1	0001	1111	1011	Add
0	1000	1111	1011	Shift

Nach vier Schritten terminiert das Verfahren, der Inhalt der beiden Register A und Q ist:

$$1000\ 1111 = 143.$$

Die Multiplikation einer Zahl mit einer Zweierpotenz ist im Binärsystem sehr einfach.⁵ Dazu werden in der Binärzahl sämtliche Ziffern um die Potenz nach links verschoben und ebenso viele 0en angefügt. Man nennt dies einen *Left-Shift* um n Stellen.

Beispiel 5.21:

Wir betrachten die Binärzahl

$$z = 11010_2.$$

⁵Dies ist analog zur Multiplikation einer Zahl mit einer Potenz von 10 im Dezimalsystem. Wird hier eine Zahl mit 1000 multipliziert, verschiebt man die Stellen einfach um drei Positionen nach links, bzw. fügt drei zusätzliche Stellen hinzu.

Im Dezimalsystem hat diese den Wert $z = 26_{10}$. Verschieben wir sämtliche Ziffern eine Stelle nach links und fügen eine 0 an, erhält man die Binärzahl

$$z' = 110100_2.$$

Diese hat den Wert $z' = 52_{10} = 2 \cdot z$. Dies illustriert, dass das Verschieben der Bits um eine Stelle nach links und das Anfügen einer 0 einer Multiplikation mit 2 entspricht. Auf ähnliche Weise kann die Multiplikation mit einer beliebigen Potenz von 2 durchgeführt werden. Ist

$$z = \sum_{i=0}^{n-1} b_i 2^i$$

die Binärdarstellung einer Zahl z , dann ist das Produkt

$$\begin{aligned} 2^m \cdot z &= 2^m \cdot \left(\sum_{i=0}^{n-1} b_i 2^i \right) \\ &= \sum_{i=0}^{n-1} b_i 2^{i+m}. \end{aligned}$$

Dies sagt aus, dass die Ziffern b_i die Koeffizienten der Potenzen von 2^{i+m} sind, dies ist exakt die Verschiebung der Bits um m Stellen nach links.

(d) **Division**

In den Rechnersystemen wird die Division durch wiederholte Subtraktion durchgeführt. Der Vollständigkeit halber wollen wir uns dennoch hier ansehen, wie man im Binärsystem die Division entsprechend den Regeln durchführt, die man aus dem Dezimalsystem kennt.

- Die Division ist der umgekehrte Prozess der Multiplikation und aus diesem Grund eng mit ihr verwandt. Bei der Division wird gefragt, wie viele Mal der Divisor im Dividenten enthalten ist. Das Ergebnis ist der Quotient, wobei in der Regel ein Rest bleibt.
- Die einfachste Art der Division beruht auf der fortgesetzten Subtraktion des Divisors vom Dividenten.

Es ergibt sich das folgende Verfahren. Der Divisor wird stellenrichtig vom Dividenten subtrahiert. Ist die Differenz positiv, so ergibt sich an der Quotientenstelle eine 1. Ist die Differenz negativ, dann wird in die Quotientenstelle eine 0 gesetzt. In diesem Fall wird die gescheiterte Subtraktion rückgängig gemacht (*Rückstellung des Rests*).

(e) **Wurzelziehen**

Im Binärsystem ist es auch möglich, komplexere arithmetische Operationen durchzuführen, allerdings sind die Verfahren i.a. anders als im Dezimalsystem. Dies wollen wir anhand des Wurzelziehens demonstrieren.

Das Wurzelziehen ist im Binärsystem recht einfach durchführbar.⁶ Wir demonstrieren das Verfahren anhand eines Beispiels.

Beispiel 5.22:

Der Algorithmus bestimmt ein Bit der Nachkommastellen pro Schritt. Angenommen, wir möchten die Zahl $\sqrt{2}$ bestimmen. Dabei sind bereits die ersten vier Bits berechnet, diese sind

$$\sqrt{2}_{10} = 1.011_2,$$

denn

$$1.011_2 = 2^0 + 2^{-2} + 2^{-3} = 1\frac{3}{8}_{10} = 1.375_{10}.$$

Was ist das nächste Bit? Angenommen, dieses ist 1. Um zu prüfen, ob dies korrekt ist, multiplizieren wir die Binärzahl $z = 1.0111$ mit sich selbst, wir bilden also das Quadrat dieser Zahl.

$$\begin{array}{r} 1.0111 \times 1.0111 \\ \hline 10111 \\ 10111 \\ 10111 \\ 00000 \\ 10111 \\ \hline 10.00010001 \end{array}$$

Das Produkt ist größer als 2, daher war die ursprüngliche Annahme — das nächste Bit ist eine 1 — nicht korrekt. Daher ist das nächste Bit eine 0 und die ersten fünf Ziffern von $\sqrt{2}$ im Binärsystem lauten 1.0110.

Bestimmen wir die nächste Ziffer in analoger Weise. Nehmen wir wieder an, die nächste Ziffer ist eine 1. Wir berechnen wieder das Produkt der Binärzahl 1.01101 mit sich selbst:

$$\begin{array}{r} 1.01101 \times 1.01101 \\ \hline 101101 \\ 000000 \\ 101101 \\ 101101 \\ 000000 \\ 101101 \\ \hline 1.111101001 \end{array}$$

Das Resultat ist kleiner als 2_{10} , daher war unsere Annahme in Ordnung. Wir haben nun sechs Binärziffern erhalten, bis zu diesem Schritt ist

$$\sqrt{2}_{10} \approx 1.01101_2 = 1\frac{3}{32} = 1.40625_{10}.$$

⁶Dieses Verfahren ist in dem Buch von PETZOLD [81], Chap. 6 diskutiert. Die Tatsache, dass das Wurzelziehen im Binärsystem einfach ist, beruht letztendlich darauf, dass im Binärsystem gilt: $(a + b)^2 = a^2 + b^2$.

5.3 Komplementärarithmetik

5.3.1 Motivation:

Wir betrachten zur Motivation des Konzepts des Komplements einer Zahl die Dezimalzahl 7, von der wir die Dezimalzahl 4 subtrahieren wollen, also

$$7 - 4 = 3.$$

Zu dem Subtrahenden 4 suchen wir jetzt die Zahl $\overline{4}$, so dass

$$4 + \overline{4} = 10.$$

Offensichtlich ist $\overline{4} = 6$. Man nennt $\overline{4} = 6$ die zu 4 komplementäre Zahl oder das **Komplement** zu 4. Die charakteristische Eigenschaft des Komplements \overline{z} einer Zahl z im Dezimalsystem ist, dass die Addition der Zahl z und ihr Komplement \overline{z} die nächsthöhere Potenz der Basis 10 ergibt.

Zurück zur obigen Subtraktion. Anstelle der Subtraktion $7 - 4 = 3$ betrachten wir nun die Addition

$$7 + \overline{4} = 7 + 6 = 13.$$

Ignorieren wir nun den 'Überlauf' in die Zehnerstelle, *i.e.* schreiben wir

$$7 + \overline{4} = 7 + 6 = (1)3,$$

und betrachten nur die Einerstellen, erhalten wir das Ergebnis 3, was dem Resultat der obigen Subtraktion entspricht. Mit anderen Worten, die Subtraktion kann durch die Addition des Komplements einer Zahl durchgeführt werden. Dies ist für Rechnersystemen äußerst relevant. Wenn die Subtraktion durch die Addition des Komplements ersetzt werden kann, benötigt das Rechenwerk der CPU lediglich ein Addierwerk, um alle elementaren arithmetischen Operationen auszuführen. Die Voraussetzung ist aber, dass die Komplementbildung im Binärsystem einfach umzusetzen ist.

Beispiel 5.23:

Betrachte die Zahlen $z_1 = 253_{10}$ und $z_2 = 109_{10}$, wir wollen die Differenz $z_1 - z_2$ berechnen. Dies führen wir durch die Addition des Komplements des Subtrahenden aus. Das Komplement von 109_{10} ist $\overline{z_2} = 891_{10}$, und

$$253_{10} + 891_{10} = (1)144_{10}.$$

Ignoriert man wieder den Überlauf in die 1000er Stelle, ergibt sich die Differenz $z_1 - z_2$.

Wir betrachten hier nur zwei Zahlen, die subtrahiert werden. Der Fachterminus dieser beiden Zahlen ist der *Minuend* und der *Subtrahend*, wobei der Subtrahend vom Minuend abgezogen wird. Das Resultat dieser Operation nennt man *Differenz*

$$\begin{array}{r} \text{Minuend} \\ - \text{Subtrahend} \\ \hline = \text{Differenz} \end{array}$$

Das Verfahren, das man in der Schule lernt, ist algorithmisch sehr aufwendig, daher verwendet man in Rechnersystemen eine einfachere Methode. Um dieses Verfahren kennenzulernen, wollen wir die Subtraktion

$$\begin{array}{r} 453 \\ - 109 \end{array}$$

durchführen. Um zu subtrahieren ohne einen Betrag aus einer weiteren Stelle zu borgen, subtrahieren wir den Subtrahenden — *i.e.* 109 — von der Zahl 999, *i.e.* wir betrachten die Operation

$$\begin{array}{r} 999 \\ - 109 \\ \hline = 890 \end{array}$$

Wir verwenden die Zahl 999 aus zwei Gründen:

- Die 999 hat die gleiche Anzahl von Stellen wie der Subtrahend.⁷
- Die obige Subtraktion der 109m von der größten Zahl mit drei Stellen erfordert *kein* Borgen eines Werts aus der nächsten Stelle.

Das Ergebnis der Subtraktion einer Zahl von einem String, der aus 9ern besteht, nennt man **Neunerkomplement**. Das Neunerkomplement von 109 ist 890, und umgekehrt funktioniert dies ebenfalls, das Neunerkomplement von 890 ist 109. Zunächst liegt der Vorteil darin, *egal welchen Zahlenwert der Subtrahend hat, die Berechnung des Neunerkomplements erfordert kein Borgen aus einer weiteren Stelle.*

Nachdem man das Neunerkomplement des Subtrahenden berechnet hat, addiert man das Neunerkomplement zu dem ursprünglichen Minuenden, in unserem Beispiel

$$\begin{array}{r} 453 \\ + 890 \\ \hline = 1343 \end{array}$$

⁷Hat der Subtrahend n Stellen, subtrahiert man von der Zahl mit n 9 ern.

Anschließend addiert man 1 und zieht 1000 ab:

$$1343 + 1 = 1344, \quad 1344 - 1000 = 344.$$

Damit ist das Ziel erreicht, um die Differenz zu erhalten, haben wir an keiner Stelle des Verfahrens einen Wert borgen müssen.

Sehen wir uns an, warum dies funktioniert. Das ursprüngliche Subtraktionsproblem ist

$$453 - 109.$$

Wenn man eine beliebige Zahl addiert und wieder subtrahiert, ändert sich an der Differenz nichts, *i.e.* wir können die Zahl 1000 addieren und wieder subtrahieren:

$$453 - 109 + 1000 - 1000.$$

Das ist äquivalent zu

$$453 - 109 + 999 + 1 - 1000.$$

Wir können dies in einer anderen Form schreiben:

$$453 + (999 - 109) + 1 - 1000.$$

Damit erhalten wir das obige Resultat.

5.3.2 Zweierkomplementdarstellung

Das Komplement einer Zahl kann auf beliebige Stellenwertsysteme verallgemeinert werden. Dazu die folgenden Definitionen.

Definition 5.9:

- (a) Das **Komplement** einer Zahl ist die Ergänzung einer vorgegebenen Zahl.
- (b) Das **B-1-Komplement** einer n -stelligen Zahl im Stellenwertsystem zur Basis B ist die Ergänzung zur größten Zahl mit n Stellen.
- (c) Das **B-Komplement** einer n -stelligen Zahl im Stellenwertsystem zur Basis B ist die Ergänzung zur kleinsten Zahl mit $n+1$ Stellen.

Im folgenden betrachten wir die Komplementbildung im Binärsystem, *i.e.* $B = 2$. Die obige Definition besagt also, addiert man zur einer gegebenen Zahl das $B-1$ -Komplement, ergibt sich die größte Zahl mit der gleichen Stellenanzahl.

Beispiel 5.24:

Sei $B = 2$, *i.e.* wir betrachten das Binärsystem. Sei die Zahl

$$z = 1010\ 0010_2$$

mit $n = 8$ Stellen vorgegeben. Dann erhält man das Einerkomplement durch die Subtraktion der größten darstellbaren Zahl mit $n = 8$ Stellen und z , *i.e.*

$$\begin{array}{r} 1111\ 1111 \\ - 1010\ 0010 \\ \hline = 0101\ 1101. \end{array}$$

Man beachte, dass auch bei dieser Subtraktion kein Borgen-Bit notwendig ist. Damit ist das **Einerkomplement** von z die Zahl

$$\bar{z} = 0101\ 1101_2.$$

Im Binärsystem — und nur in diesem Zahlensystem — kann man das Einerkomplement einer Binärzahl auch auf andere Weise erhalten. Ist beispielsweise wieder die Zahl

$$z = 1010\ 0010_2$$

vorgegeben, dann kippt man jedes Bit dieser Zahl, *i.e.* aus eine 0 wird eine 1 und aus eine 1 eine 0. Damit

$$1010\ 0010_2 \longrightarrow 0101\ 1101_2,$$

und Addition dieser beiden Binärzahlen ergibt die größte darstellbare Binärzahl mit acht Stellen:

$$\begin{array}{r} 1010\ 0010 \\ + 0101\ 1101 \\ \hline = 1111\ 1111. \end{array}$$

Anmerkung:

Das Kippen der Bits kann durch die XOR-Operation ausgeführt werden. Die Logik der XOR-Operation $a \oplus b$ ist:

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

Ist nun a ein Bit einer Binärzahl, dann erhalten wir damit

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

Das bedeutet, die XOR-Verknüpfung eines Bits a mit 1 kehrt den Wert des Bits a um.

Wie das Beispiel [5.24] zeigt, erhält man das Einerkomplement einer Binärzahl einfach dadurch, dass man sämtliche Stellen der vorgegebenen Binärzahl z invertiert, *i.e.* $0 \rightarrow 1$ und $1 \rightarrow 0$.

Addiert man zu einer Binärzahl z ihr Einerkomplement, ergibt sich also die größte darstellbare Zahl gleicher Stellenzahl:

$$\begin{array}{r} z \quad 1010\ 0010 \\ + \bar{z} \quad 0101\ 1101 \\ \hline = \quad 1111\ 1111. \end{array}$$

Addiert man zu diesem Ergebnis noch 1, erhält man:

$$\begin{array}{r} \quad 1111\ 1111 \\ + \quad 0000\ 0001 \\ \hline = \quad 1\ 0000\ 0000 \end{array}$$

Abgesehen von dem Überlauf in die $n+1$ te Stelle ist das Ergebnis also 0. Wenn also der Überlauf ignoriert wird, können wir schreiben:

$$z + \bar{z} + 1 = 0,$$

oder

$$\bar{z} + 1 = -z.$$

Damit erhalten wir folgende Resultate:

- $\bar{z} + 1$ ist eine Darstellung für $-z$.
- Die Binärzahl $\bar{z} + 1$ heißt **Zweierkomplement** der Zahl z .
- Die Subtraktion einer Binärzahl kann auf die Addition des Zweierkomplements zurückgeführt werden, wobei der Übertrag ignoriert wird.
- Die Bildung des Zweierkomplements ist im Binärsystem besonders effektiv umsetzbar, da dies im wesentlichen eine Negation der Bits ist.

Damit erhalten wir ein Verfahren, wie man zu einer gegebenen ganzen Zahl in Dezimaldarstellung z_{10} die zugehörige Zweierkomplementdarstellung findet.

Algorithmus Umrechnung einer Dezimalzahl $z_{10} \in \mathbb{Z}$ in die Zweierkomplementdarstellung:

1. Bestimme die Anzahl der Binärstellen n .⁸

⁸Dies ergibt sich durch

$$n = \lfloor \log_2 |z_{10}| \rfloor + 2.$$

2. Wenn z_{10} nicht negativ ist, Umrechnung wie bei Binärcodierung
3. Wenn z_{10} negativ ist
 - (a) Binärcode von $|z_{10}|$
 - (b) Einserkomplement bilden (alle Bits kippen)
 - (c) 1 addieren

Beispiel 5.25:**Zweierkomplementdarstellung $n = 4$**

Wir konstruieren in diesem Beispiel die Zweierkomplementdarstellung für vier Bit, *i.e.* die Anzahl n von Bit ist $n = 4$.

Da die Anzahl der Bits $n = 4$ ist, kann man $2^4 = 16$ ganze Zahlen codieren. Das darstellbare Zahlenintervall ganzer Zahlen wählt man zu

$$[-8, +7] = [-2^3, +2^3 - 1] = [-2^{4-1}, +2^{4-1} - 1] = [-2^{n-1}, +2^{n-1} - 1]_{n=4}. \quad (5.3)$$

Nach obigen Algorithmus ist für positive ganze Zahlen die Zweierkomplementdarstellung identisch mit der Binärdarstellung, *i.e.*

$$\left. \begin{array}{ll} 0 \longrightarrow & 0000_{2k} \\ 1 \longrightarrow & 0001_{2k} \\ 2 \longrightarrow & 0010_{2k} \\ 3 \longrightarrow & 0011_{2k} \\ 4 \longrightarrow & 0100_{2k} \\ 5 \longrightarrow & 0101_{2k} \\ 6 \longrightarrow & 0110_{2k} \\ 7 \longrightarrow & 0111_{2k} \end{array} \right\} \quad (5.4)$$

Wir bezeichnen Bitfolgen, die die Zweierkomplementdarstellung von Zahlen beschreiben, durch den Index $2k$.

Die Zweierkomplementdarstellung von -1 erhalten wir gemäß:

$$1 = 0001_{2k} \xrightarrow{\text{Bit kippen}} 1110 \xrightarrow{+1} 1111_{2k} = -1.$$

Für -2 ergibt sich:

$$2 = 0010_{2k} \xrightarrow{\text{Bit kippen}} 1101 \xrightarrow{+1} 1110_{2k} = -2.$$

Auf diese Weise erhalten wir:

$$\left. \begin{array}{ll} -1 & \longrightarrow 1111_{2k} \\ -2 & \longrightarrow 1110_{2k} \\ -3 & \longrightarrow 1101_{2k} \\ -4 & \longrightarrow 1100_{2k} \\ -5 & \longrightarrow 1011_{2k} \\ -6 & \longrightarrow 1010_{2k} \\ -7 & \longrightarrow 1001_{2k} \\ -8 & \longrightarrow 1000_{2k} \end{array} \right\} \quad (5.5)$$

Anhand dieses Beispiels erkennt man

- Ist das höchstwertige Bit 0, dann ist die Zahl positiv,
- ist das höchstwertige Bit 1, dann ist die Zahl negativ.

Daher stellt das höchstwertige Bit der Zweierkomplementdarstellung das Vorzeichen dar.

Mit Zweierkomplementzahlen der Länge n Bit können 2^n ganze Zahlen codiert werden. Die Gleichung (5.3) zeigt den Range von ganzen Zahlen, der dadurch dargestellt werden kann. Mit n -Bit Zweierkomplementzahlen wird der Zahlenbereich

$$[-2^{n-1}, +2^{n-1} - 1] \quad (5.6)$$

dargestellt.

n	Range
4	$[-8, 7]$
5	$[-16, 15]$
6	$[-32, 31]$
7	$[-64, 63]$
8	$[-128, 127]$
9	$[-256, 255]$
\vdots	\vdots
32	$[-2^{31}, 2^{31} - 1] = [-2\,147\,483\,648, +2\,147\,483\,647]$

In Computern wird bevorzugt die Zweierkomplementdarstellung mit $n = 32$ oder $n = 64$ Bit benutzt, um ganze Zahlen zu codieren. Der dazu passende Datentyp ist der **Integer Datentyp**.

Anmerkung:

Die Zweierkomplementdarstellung ist eine Codierung der Zahlenmenge \mathbb{Z}_n , dies impliziert den Sachverhalt, dass darstellbare Zahlenbereich das Intervall

$$[-2^{n-1}, +2^{n-1} - 1]$$

abdeckt.

Beispiel 5.26:

Wir berechnen in diesem Beispiel die Zweierkomplementdarstellung der Dezimalzahl $z_{10} = -109$.

Die Anzahl der Bits für die Darstellung von -109 erhalten wir aus der Überlegung, dass das kleinste Intervall, das die Zahl -109 enthält, $[-128, +127]$ ist. Daher benötigt man $n = 8$ Bits.

Die Binärdarstellung von $|z_{10}| = 109$ ist

$$109_{10} = 0110\,1101_2.$$

Kippt man sämtliche Bits, ergibt sich $1001\,0010$. Die Addition von 1 liefert:

$$-109_{10} = 1001\,1011_{2k}.$$

Sieht man sich die Bitfolgen der Zweierkomplementdarstellung für $n = 4$ im Beispiel [5.25] genauer an, dann erkennt man, dass die negativen Zahlen auch in der Form

$$\begin{array}{ll} -1 = -8 + 7 & 1111 = 1000 + 0111 \\ -2 = -8 + 6 & 1110 = 1000 + 0110 \\ -3 = -8 + 5 & 1101 = 1000 + 0101 \\ -4 = -8 + 4 & 1100 = 1000 + 0100 \\ -5 = -8 + 3 & 1011 = 1000 + 0011 \\ -6 = -8 + 2 & 1010 = 1000 + 0010 \\ -7 = -8 + 1 & 1001 = 1000 + 0001 \end{array}$$

geschrieben werden können. Dies führt auf die folgende Umrechnung eines Bitstrings, der ganze Zahlen in Zweierkomplement codiert, in eine ganze Zahl im Dezimalsystem. Es gilt:

Gegeben ist eine n -Bit Binärzahl in Zweierkomplementdarstellung

$$z = (b_{n-1}b_{n-2} \dots b_1b_0)_{2k}.$$

Dann ist der Wert dieser Zahl:

$$z = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i. \quad (5.7)$$

Beispiel 5.27:

Gegeben ist die folgende Zweierkomplementzahl

$$z = 1000\,1101_{2k}.$$

Dann ist der Wert dieser Zweierkomponent Zahl:

$$\begin{aligned} z &= 1000\,1101_{2k} \\ &= -1 \cdot 2^7 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 \\ &= -128 + 8 + 4 + 1 \\ &= -115_{10}. \end{aligned}$$

Man beachte, dass für negative Zahlen in der Zweierkomplementdarstellung führende 1en den Wert einer Zahl nicht ändern. Es ist:

$$\begin{aligned} n = 4 \text{ Bit : } \quad 1011 &= -2^3 + 2^1 + 2^0 = -5, \\ n = 6 \text{ Bit : } \quad 111011 &= -2^5 + 2^4 + 2^3 + 2^1 + 2^0 = -32 + 16 + 8 + 2 + 1 = -5, \\ n = 8 \text{ Bit : } \quad 11111011 &= -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 = -5. \end{aligned}$$

Beispiel 5.28:

In diesem Beispiel demonstrieren wir, wie die Subtraktion einer Binärzahl durch die Addition des Zweierkomplements umgesetzt werden kann. Seien also die beiden Dezimalzahlen

$$z_1 = 109_{10} \quad \text{und} \quad z_2 = 44_{10}$$

gegeben. Wir wollen die Differenz $z_1 - z_2$ berechnen. Die Binärdarstellung von z_1 ist:

$$z_1 = 109_{10} = 110\,1101_2.$$

Da wir die Subtraktion $z_1 - z_2$ ausführen, benötigen wir die Zweierkomplementdarstellung von $z_2 = -44_{10}$. Gemäß obigem Algorithmus ist zunächst die Anzahl der Stellen zu bestimmen. Da

$$-44 \in [-64, +63] = [-2^6, 2^6 - 1],$$

benötigt man sieben Stellen, also $n = 7$. Da z_2 negativ ist, bestimmen wir zunächst die Binärdarstellung von $|z_2|$ mit der Divisionsmethode, wir erhalten:

$$|z_2| = 010\,1100.$$

Die führende 0 ist hinzugefügt, da man 7 Stellen benötigt. Im nächsten Schritt werden alle Bits gekippt:

$$|z_2| = 010\,1100 \longrightarrow 101\,0011.$$

Schließlich addiert man 1, es ergibt sich:

$$-z_2 = 101\ 0100.$$

Damit ist

$$\begin{array}{r} z_1 \quad 110\ 1101 \\ +(-z_2) \quad 101\ 0100 \\ \hline = \quad (1)100\ 0001 \end{array}$$

Wird wieder der Übertrag ignoriert, erhält man als Resultat 65.

5.3.3 Addition und Subtraktion von Zweierkomplementzahlen

Die Addition⁹ zweier Zweierkomplementzahlen ist in den folgenden Beispielen illustriert. Wir betrachten der Einfachheit halber Zweierkomplementzahlen im Bereich $[-8, +7]$, *i.e.* $n = 4$ Bits (siehe Beispiel [5.25]).

1. Addition einer negativen und einer positiven Zahl $-7 + 5 = -2$. Die Zweierkomplementdarstellung der beiden Summanden ist:

$$\begin{aligned} -7 &\longrightarrow 1001_{2k}, \\ 5 &\longrightarrow 0101_{2k}. \end{aligned}$$

$$\begin{array}{r} \text{Dann ist:} \quad + \quad 1001_{2k} \\ \quad \quad \quad + \quad 0101_{2k} \\ \hline \quad \quad \quad = \quad 1110_{2k} \end{array}$$

Insbesondere erhalten wir $(-4) + 4 = 0$, denn es ist

$$\begin{aligned} 4 &\longrightarrow 0100_{2k}, \\ -4 &\longrightarrow 1100_{2k}. \end{aligned}$$

Dann ist:

$$\begin{array}{r} (-4) \quad 1100_{2k} \\ +4 \quad 0100_{2k} \\ \hline = 0 \quad (1)0000_{2k} \end{array}$$

Wird der Überlauf ignoriert, erhalten wir das Ergebnis 0.

⁹Wir folgen hier der Darstellung von STALLINGS [99], Chapter 8.3.

2. Addition zweier positiven Zahlen $3 + 4 = 7$ mit

$$3 \longrightarrow 0011_{2k},$$

$$4 \longrightarrow 0100_{2k}.$$

ergibt sich

$$\begin{array}{r} 0011_{2k} \\ + 0100_{2k} \\ \hline = 0111_{2k} \end{array}$$

3. Addition zweier negativen Zahlen $(-3) + (-4) = -7$:

$$\begin{array}{r} 1101 \\ + 1100 \\ \hline = (1)1001 \end{array}$$

4. Addition zweier positiven Zahlen $5 + 4$ Überlauf:

$$\begin{array}{r} 0101 \\ + 0100 \\ \hline = 1001 \end{array}$$

5. Addition zweier negativen Zahlen $(-5) + (-6)$ Überlauf:

$$\begin{array}{r} 1011 \\ + 1010 \\ \hline = (1)0101 \end{array}$$

Die ersten vier Beispiele zeigen erfolgreiche Operationen. Falls das Ergebnis der Operation positiv ist, erhalten wir eine positive Zahl in der üblichen Binärdarstellung. Ist das Ergebnis der Operation negativ, dann erhalten wir eine negative Zahl in der Zweierkomplementdarstellung. Bei einigen Operationen entsteht ein Carry Out Bit, das ignoriert wird.

Bei jeder Addition kann der Fall eintreten, dass das Ergebnis den durch die 4 Bit definierten darstellbaren Bereich überschreitet. Dies nennt man einen **Überlauf**. Falls ein Überlauf auftritt, muss die ALU diesen Sachverhalt signalisieren, damit dieses Resultat nicht weiter verwendet wird. Um den Überlauf zu erkennen, wendet man die folgende Regel an:

Falls zwei Zahlen addiert werden, entweder sind beide Zahlen positiv oder beide Zahlen negativ, dann entsteht ein Überlauf genau dann, wenn das Ergebnis das entgegengesetzte Vorzeichen hat.

Die beiden letzten Beispiele zeigen diesen Sachverhalt.

Die Subtraktion zweier Zweierkomplementzahlen wird nach folgendem Verfahren durchgeführt. Um eine Zahl (den Subtrahenden) von einer anderen Zahl (den Minuenden) abzuziehen, bildet man das Zweierkomplement des Subtrahenden und addiert diesen Wert zu dem Minuenden. Beispiele dazu sind im folgenden aufgeführt.

1. $2 - 7 = -5$. Die Zweierkomplementdarstellungen des Minuenden und des Subtrahenden sind:

$$2 \longrightarrow 0010_{2k},$$

$$7 \longrightarrow 0111_{2k}.$$

Wir erhalten die -7 durch das Invertieren der Bits und die Addition von 1, *i.e.*

$$-7 \longrightarrow 1001_{2k}.$$

Damit erhalten wir:

$$\begin{array}{r} 2 \quad 0010_{2k} \\ +(-7) \quad 1001_{2k} \\ \hline = -5 \quad 1011_{2k} \end{array}$$

2. $5 - 2 = 3$. Die Zweierkomplementdarstellungen des Minuenden und des Subtrahenden sind:

$$5 \longrightarrow 0101_{2k},$$

$$2 \longrightarrow 0010_{2k}.$$

Wir erhalten die -2 durch das Invertieren der Bits und die Addition von 1, *i.e.*

$$-2 \longrightarrow 1110_{2k}.$$

Damit erhalten wir:

$$\begin{array}{r} 5 \quad 0101_{2k} \\ +(-2) \quad 1110_{2k} \\ \hline = 3 \quad (1)0011_{2k} \end{array}$$

Hier ignoriert man wieder den Überlauf.

3. $-5 - 2 = -7$. Die Zweierkomplementdarstellungen des Minuenden und des Subtrahenden sind:

$$-5 \longrightarrow 1011_{2k},$$

$$2 \longrightarrow 0010_{2k}.$$

Wir erhalten die -2 durch das Invertieren der Bits und die Addition von 1, *i.e.*

$$-2 \longrightarrow 1110_{2k}.$$

Damit erhalten wir:

$$\begin{array}{r}
 -5 \quad 1011_{2k} \\
 +(-2) \quad 1110_{2k} \\
 \hline
 = -7 \quad (1)1001_{2k}
 \end{array}$$

4. $5 - (-2) = 7$. Die Zweierkomplementdarstellungen des Minuenden und des Subtrahenden sind:

$$\begin{aligned}
 5 &\longrightarrow 0101_{2k}, \\
 -2 &\longrightarrow 1110_{2k}.
 \end{aligned}$$

Wir erhalten die $+2$ durch das Invertieren der Bits und die Addition von 1, *i.e.*

$$2 \longrightarrow 0010_{2k}.$$

Damit erhalten wir:

$$\begin{array}{r}
 5 \quad 0101_{2k} \\
 +(2) \quad 0010_{2k} \\
 \hline
 = 7 \quad 0111_{2k}
 \end{array}$$

Die folgenden beiden Beispiele zeigen, dass die Überlaufregel bei der Subtraktion ebenfalls anwendbar ist.

1. $7 - (-7) = \text{Überlauf}$. Die Zweierkomplementdarstellungen des Minuenden 7 und des Subtrahenden -7 sind:

$$\begin{aligned}
 7 &\longrightarrow 0111_{2k}, \\
 -7 &\longrightarrow 1001_{2k}.
 \end{aligned}$$

Wir erhalten die $+7$ durch das Invertieren der Bits und die Addition von 1, *i.e.*

$$7 \longrightarrow 0111_{2k}.$$

Damit erhalten wir (beide Zahlen sind jetzt positiv):

$$\begin{array}{r}
 7 \quad 0111_{2k} \\
 +7 \quad 0111_{2k} \\
 \hline
 = 1110_{2k} \quad \text{Überlauf, Ergebnis negativ}
 \end{array}$$

2. $-6 - 4 = \text{Überlauf}$. Die Zweierkomplementdarstellungen des Minuenden -6 und des Subtrahenden 4 sind:

$$\begin{aligned}
 -6 &\longrightarrow 1010_{2k}, \\
 4 &\longrightarrow 0100_{2k}.
 \end{aligned}$$

Wir erhalten die -4 durch das Invertieren der Bits und die Addition von 1, *i.e.*

$$-4 \longrightarrow 1100_{2k}.$$

Damit erhalten wir (beide Zahlen sind jetzt negativ):

$$\begin{array}{rcl}
 -6 & 1010_{2k} & \\
 +(-4) & \underline{1100_{2k}} & \\
 = & (1)0110_{2k} & \text{Überlauf, Ergebnis positiv}
 \end{array}$$

5.3.4 Multiplikation zweier Zweierkomplementzahlen

Es gibt mehrere Verfahren, die es ermöglichen, zwei Zweierkomplementzahlen zu multiplizieren. Hierbei ist insbesondere das Vorzeichen zu beachten. Das gebräuchlichste Verfahren, zwei Zweierkomplementzahlen zu multiplizieren, ist der **Algorithmus von Booth**. Dieses Verfahren wurde 1951 von DONALD ANDREW BOOTH bei kristallographischen Verfahren entwickelt.

Zur Notation, bei der Multiplikation zweier Zahlen verwendet man die folgende Bezeichnung für die beiden Faktoren:

$$\text{Multiplikand} \times \text{Multiplikator} = \text{Produkt}.$$

Das Verfahren von BOOTH kann man folgendermaßen beschreiben. Der Multiplikator und der Multiplikand — zwei Zweierkomplementzahlen mit je n Bit — werden in zwei n -Bit Register Q respektive M plaziert. Es gibt weiterhin ein 1-Bit Register — dieses bezeichnen wir mit Q_{-1} , welches sich rechts vom kleinstwertigen Bit des Q -Registers befindet. Wir benötigen ein weiteres n -Bit Register A , denn die Multiplikation zweier n -Bit Zahlen ergibt eine $2n$ -Bit Zahl. Das Ergebnis der Multiplikation liegt in den beiden Registern A und Q . Die beiden Register A und Q_{-1} werden mit 0 initialisiert. Die eigentliche Multiplikation geschieht in einer Iteration von n Schritten. In jedem Schritt wird folgendes ausgeführt.

Die Steuerlogik scant die Bits des Multiplikators im Register Q bitweise in jedem Step. Gleichzeitig wird das rechts daneben liegende Bit gelesen, in der obigen Implementierung sind dies die beiden Bits Q_0, Q_{-1} , wobei Q_0 das niedrigwertigste Bit des Q Registers ist. Dann werden folgende Fälle unterschieden.

1. **Fall** Die beiden Bits sind gleich, *i.e.* 00 oder 11. Dann werden alle Bits der drei Register AQQ_{-1} um ein Bit nach rechts verschoben (right shift).
2. **Fall** Die beiden Bits sind 10. Der Inhalt des Registers M wird vom Register A subtrahiert, *i.e.* wir ersetzen

$$A \leftarrow A - M.$$

Anschließend werden wieder alle Bits der drei Register AQQ_{-1} um ein Bit nach rechts verschoben (right shift).

3. **Fall** Die beiden Bits sind 01. Der Inhalt des Registers M wird zum Register A addiert, *i.e.* wir ersetzen

$$A \leftarrow A + M.$$

Anschließend werden wieder alle Bits der drei Register AQQ_{-1} um ein Bit nach rechts verschoben (right shift).

In jedem der obigen Fälle handelt es sich bei der Rechtsverschiebung um einen sogenannten *arithmetic right shift*, i.e. das ganz links stehende Bit des Registers A , also A_{n-1} wird in A_{n-2} verschoben und bleibt ebenso in A_{n-1} . Dies ist erforderlich um das Vorzeichen der Zahl in den beiden Registern AQ korrekt zu erhalten.

Die Abbildung [5.4] zeigt das Flußdiagramm dieses Algorithmus.

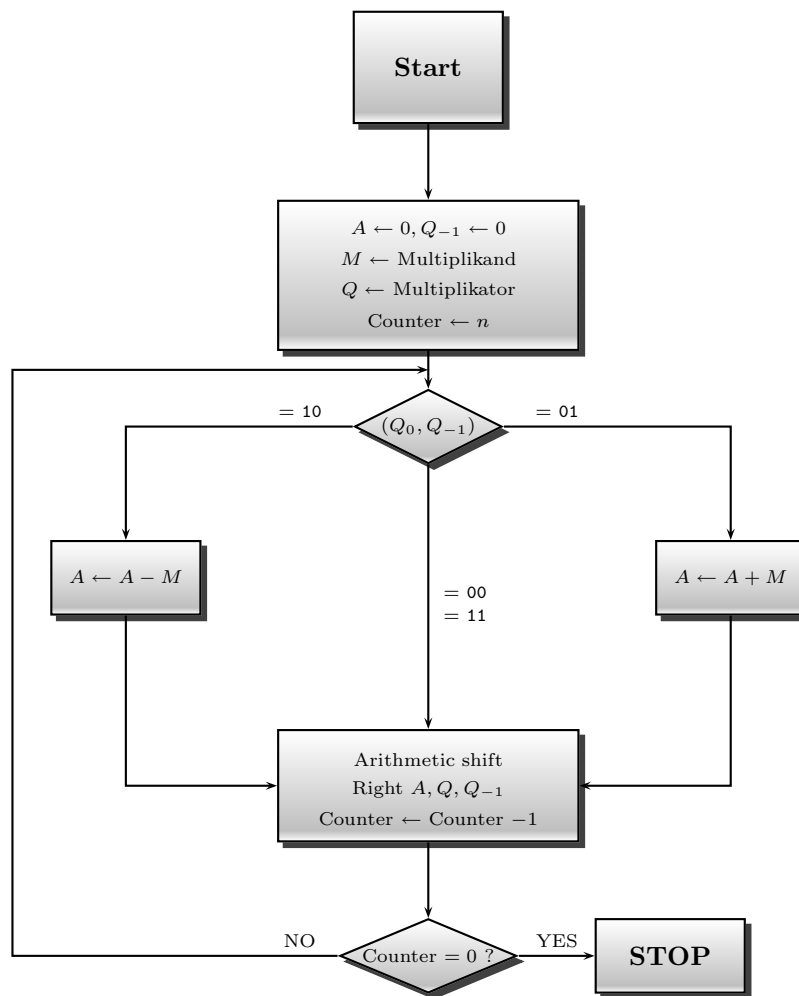


Abbildung 5.4: Flußdiagramm des Algorithmus von BOOTH für die Multiplikation zweier Zweierkomplementzahlen.

Beispiel 5.29:

Multiplikation von 3 und 7 mit $n = 4$ Bit Zweierkomplementzahlen. Sämtliche

Bitfolgen repräsentieren Zweierkomplementzahlen.

Register A	Register Q	Q_{-1}	Register M	
0000	0011	0	0111	Initialisierung

Erste Runde $Q_0Q_{-1} = 10$:

Register A	Register Q	Q_{-1}	Register M	
1001	0011	0	0111	$A \leftarrow A - M$
1100	1001	1	0111	Arithmetic right shift

Zweite Runde $Q_0Q_{-1} = 11$:

Register A	Register Q	Q_{-1}	Register M	
1110	0100	1	0111	Arithmetic right shift

Dritte Runde $Q_0Q_{-1} = 01$:

Register A	Register Q	Q_{-1}	Register M	
0101	0100	1	0111	$A \leftarrow A + M$
0010	1010	0	0111	Arithmetic right shift

Vierte Runde $Q_0Q_{-1} = 00$:

Register A	Register Q	Q_{-1}	Register M	
0001	0101	0	0111	Arithmetic right shift

Stop.

In den beiden Registern A und Q steht das Resultat, *i.e.*

$$0001\,0101_{2k} = 21_{10}.$$

Beispiel 5.30:

Multiplikation von 7 und -3 mit $n = 4$ Bit Zweierkomplementzahlen.

Die $-3 \rightarrow Q$ stellen wir durch 1101_{2k} dar.

Register A	Register Q	Q_{-1}	Register M	
0000	1101	0	0111	Initialisierung

Erste Runde $Q_0Q_{-1} = 10$:

Register A	Register Q	Q_{-1}	Register M	
1001	1101	0	0111	$A \leftarrow A - M$
1100	1110	1	0111	Arithmetic right shift

Zweite Runde $Q_0Q_{-1} = 01$:

Register A	Register Q	Q_{-1}	Register M	
0011	1110	1	0111	$A \leftarrow A + M$
0001	1111	0	0111	Arithmetic right shift

Dritte Runde $Q_0Q_{-1} = 10$:

Register A	Register Q	Q_{-1}	Register M	
1010	1111	0	0111	$A \leftarrow A - M$
1101	0111	1	0111	Arithmetic right shift

Vierte Runde $Q_0Q_{-1} = 11$:

Register A	Register Q	Q_{-1}	Register M	
1110	1011	1	0111	Arithmetic right shift

Stop.

In den beiden Registern A und Q steht das Resultat, *i.e.*

$$1110\ 1011_{2k} = -21_{10}.$$

Beispiel 5.31:

Multiplikation von -7 und 3 mit $n = 4$ Bit Zweierkomplementzahlen.

Die $-7 \rightarrow M$ stellen wir durch 1001_{2k} dar.

Register A	Register Q	Q_{-1}	Register M	
0000	0011	0	1001	Initialisierung

Erste Runde $Q_0Q_{-1} = 10$:

Register A	Register Q	Q_{-1}	Register M	
0111	0011	0	1001	$A \leftarrow A - M$
0011	1001	1	1001	Arithmetic right shift

Zweite Runde $Q_0Q_{-1} = 11$:

Register A	Register Q	Q_{-1}	Register M	
0001	1100	1	1001	Arithmetic right shift

Dritte Runde $Q_0Q_{-1} = 01$:

Register A	Register Q	Q_{-1}	Register M	
1010	1100	1	1001	$A \leftarrow A + M$
1101	0110	0	1001	Arithmetic right shift

Vierte Runde $Q_0Q_{-1} = 00$:

Register A	Register Q	Q_{-1}	Register M	
1110	1011	1	0111	Arithmetic right shift

Stop.

In den beiden Registern A und Q steht das Resultat, *i.e.*

$$1110\,1011_{2k} = -21_{10}.$$

Beispiel 5.32:

Multiplikation von -7 und -3 mit $n = 4$ Bit Zweierkomplementzahlen.
Die $-7 \rightarrow M$ stellen wir durch 1001_{2k} dar, die $-3 \rightarrow Q$ durch die Zweierkomplementzahl 1101_{2k} .

Register A	Register Q	Q_{-1}	Register M	
0000	1101	0	1001	Initialisierung

Erste Runde $Q_0Q_{-1} = 10$:

Register A	Register Q	Q_{-1}	Register M	
0111	1101	0	1001	$A \leftarrow A - M$
0011	1110	1	1001	Arithmetic right shift

Zweite Runde $Q_0Q_{-1} = 01$:

Register A	Register Q	Q_{-1}	Register M	
1100	1110	1	1001	$A \leftarrow A - M$
1110	0111	0	1001	Arithmetic right shift

Dritte Runde $Q_0Q_{-1} = 10$:

Register A	Register Q	Q_{-1}	Register M	
0101	0111	0	1001	$A \leftarrow A - M$
0010	1011	1	1001	Arithmetic right shift

Vierte Runde $Q_0Q_{-1} = 11$:

Register A	Register Q	Q_{-1}	Register M	
0001	0101	1	0111	Arithmetic right shift

Stop.

In den beiden Registern A und Q steht das Resultat, *i.e.*

$$0001\,0101_{2k} = 21_{10}.$$

Sehen wir uns an, warum (und wie) dieser Algorithmus funktioniert.

1. Positiver Multiplikator

Wir betrachten zunächst einen positiven Multiplikator, der aus einem Block 1en besteht mit 0en vor und hinter dem Block, beispielsweise 000011110. Die Multiplikation mit einer Potenz von 2 können wir durch eine geeignete Verschiebung von Kopien des Multiplikanden erreichen. Dann ist

$$\begin{aligned} M \times 000011110 &= M \times (2^4 + 2^3 + 2^2 + 2^1) \\ &= M \times (16 + 8 + 4 + 2) \\ &= M \times 30. \end{aligned}$$

Die Anzahl solcher Operationen kann man auf zwei reduzieren, denn es gilt:

$$2^n + 2^{n-1} + \dots + 2^{n-K} = 2^{n+1} - 2^{n-K}. \quad (5.8)$$

Anmerkungen:

Die Gl. (5.8) ergibt sich aus

$$\sum_{k=0}^n 2^k = 2^{n+1} - 1. \quad (5.9)$$

Beweis:

Wir führen den Beweis dieser Aussage durch Induktion über n .

INDUKTIONSBASIS: Die Aussage gilt für $n = 1$:

Die linke Seite der Aussage ergibt für $n = 1$:

$$\sum_{k=0}^1 2^k = 2^0 + 2^1 = 3 = 2^2 - 1.$$

Die rechte Seite ergibt mit $n = 1$:

$$2^2 - 1 = 3,$$

das heisst, die Aussage ist wahr für $n = 1$.

INDUKTIONSSANNAHME oder INDUKTIONSVORAUSSETZUNG (IV): Die Aussage gilt für $n \geq 0$:

$$\sum_{k=0}^n 2^k = 2^{n+1} - 1.$$

INDUKTIONSSCHRITT: Betrachte

$$\begin{aligned} \sum_{k=0}^{n+1} 2^k &= \sum_{k=0}^n 2^k + 2^{n+1} \\ &\stackrel{IV}{=} 2^{n+1} - 1 + 2^{n+1} \\ &= 2 \cdot 2^{n+1} - 1 \\ &= 2^{(n+1)+1} - 1 \end{aligned}$$

was zu zeigen war.

Damit erhalten wir:

$$\begin{aligned} 2^n + 2^{n-1} + \dots + 2^{n-K} &= 2^n + 2^{n-1} + \dots + 2^{n-K} \\ &\quad + 2^{n-K-1} + 2^{n-K-2} + \dots + 2^1 + 2^0 \\ &\quad - 2^{n-K-1} - 2^{n-K-2} - \dots - 2^1 - 2^0 \\ &= \sum_{l=0}^n 2^l - (2^{n-K-1} + 2^{n-K-2} + \dots + 2^1 + 2^0) \\ &= \left(\sum_{l=0}^n 2^l \right) - \left(\sum_{i=0}^{n-K-1} 2^i \right) \\ &= 2^{n+1} - 1 - 2^{n-K} + 1 \\ &= 2^{n+1} - 2^{n-K}. \end{aligned}$$

Im vorletzten Schritt wird die Gl. (5.9) benutzt.

Damit ist Gl. (5.8) gezeigt.

Damit

$$\begin{aligned} M \times 000011110 &= M \times (2^5 - 2^1) \\ &= M \times (32 - 2) \\ &= M \times 30. \end{aligned}$$

Daher kann man das Produkt durch eine Addition und eine Subtraktion des Multiplikanden erzeugen. Dieses Muster ist auf alle 1er Blöcke in einem Multiplikanden anwendbar, dies trifft auch auf eine einzelne 1 zu.

$$\begin{aligned} M \times 011110110010 &= M \times (2^{10} + 2^9 + 2^8 + 2^7 + 2^5 + 2^4 + 2^1) \\ &= M \times (2^{11} - 2^7 + 2^6 - 2^4 + 2^2 - 2^1) \end{aligned}$$

Der Algorithmus von BOOTH führt genau diese Schritte aus indem eine Subtraktion durchgeführt wird bei einer $Q_0Q_{-1} = 10$ Konstellation und eine Addition bei der $Q_0Q_{-1} = 01$ Konstellation.

2. Negativer Multiplikator

Um zu sehen dass das gleiche Schema für einen negativen Multiplikator funktioniert, betrachten wir folgendes. Sei z eine negative Zahl in Zweierkomplementdarstellung, *i.e.*

$$z = (1b_{n-2}b_{n-3} \dots b_1b_0)_{2k}.$$

Dann können wir den Zahlenwert von z schreiben als

$$z = -2^{n-1} + \sum_{k=0}^{n-2} b_k 2^k. \quad (5.10)$$

Das höchstwertige Bit von z ist 1, da die Zahl negativ ist. Wir nehmen nun an, dass an der Stelle l die höchstwertige 0 steht, *i.e.* dann hat z die Form

$$z = (1111 \dots 10b_{l-1}b_{l-2} \dots b_1b_0)_{2k}. \quad (5.11)$$

Dann ist der Wert von z :

$$z = -2^{n-1} + 2^{n-2} + \dots + 2^{l+1} + b_{l-1}2^{l-1} + \dots + b_02^0. \quad (5.12)$$

Nun ist

$$2^{n-2} + 2^{n-3} + \dots + 2^{l+1} = 2^{n-1} - 2^{l+1},$$

damit ergibt sich:

$$-2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 2^{l+1} = -2^{l+1}. \quad (5.13)$$

Mit Hilfe von Gl. (5.13) erhalten wir für Gl. (5.12):

$$z = -2^{l+1} + \sum_{k=0}^{l-1} b_k 2^k. \quad (5.14)$$

Zurück zu dem Algorithmus von BOOTH. Aus der Darstellung von z in Gl. (5.11) folgt, dass alle Bits von b_0 bis zur höchstwertigen 0 durch den Algorithmus korrekt gehandelt werden, denn sie erzeugen — bis auf den -2^{l+1} Term — alle Terme in Gl. (5.14). Liest der Algorithmus die höchstwertige

0 und trifft auf die nächste 1, liegt ein 10-Übergang vor, daher wird eine Subtraktion durchgeführt, das ist der -2^{l+1} -Term. Dies ist der verbleibende Summand in der Entwicklung (5.14).

Betrachte beispielsweise die Multiplikation eines Multiplikanden M mit dem Multiplikator (-6) . Als 8-Bit Zweierkomplementzahl ist:

$$-6 = 1111\ 1010_{2k}.$$

Mit Gl. (5.10) erhalten wir:

$$-6 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1.$$

Daher:

$$\begin{aligned} M \times (-6) &= M \times (1111\ 1010)_{2k} \\ &= M \times (-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1). \end{aligned}$$

Mit Gl. (5.13) erhalten wir:

$$M \times (1111\ 1010)_{2k} = M \times (-2^3 + 2^1),$$

und mit obiger Argumentation:

$$M \times (1111\ 1010)_{2k} = M \times (-2^3 + 2^2 - 2^1).$$

Dieses Beispiel illustriert, dass auch für negative Multiplkatoren das Schema in dem Algorithmus von BOOTH funktioniert. Der Algorithmus führt eine Subtraktion aus, sobald die erste 1 gelesen wird¹⁰ (10 Übergang), es wird eine Addition durchgeführt, wenn ein 01 Übergang vorliegt und anschließend wird erneut subtrahiert.

Daher führt der Algorithmus von BOOTH weniger Additionen und Subtraktionen durch als die Papiet-und-Bleistift Methode.

¹⁰Der Bitstring des Multiplikators wird von rechts nach links gescant.

5.4 Oktalsystem

Das Oktalsystem spielt gelegentlich in der Informatik eine Rolle. Dieses Zahlensystem hat als Basis $B = 8$ und die Zahlen werden durch Ziffern 0, 1, 2, 3, 4, 5, 6, 7 dargestellt. Eine Zahl $z = 543_8$ im Oktalsystem bedeutet also

$$z = 543_8 = 5 \times 8^2 + 4 \times 8^1 + 3 \times 8^0.$$

Die Umrechnung vom Oktalsystem in das Dezimalsystem und umgekehrt geschieht exakt nach dem gleichen Verfahren wie im Binärsystem.

Beispiel 5.33:

Der Wert der Oktalzahl $z = 543_8$ im Dezimalsystem ergibt sich aus der Entwicklung nach Potenzen von 8 und anschließender Addition. Damit

$$\begin{aligned} z &= 543_8 \\ &= 5 \times 8^2 + 4 \times 8^1 + 3 \times 8^0 \\ &= 5 \times 64 + 4 \times 8 + 3 \\ &= 355_{10}. \end{aligned}$$

Die umgekehrte Richtung, *i.e.* gegeben eine Dezimalzahl, gesucht ist die Darstellung dieser Zahl im Oktalsystem, ergibt sich wieder mit Hilfe des Divisionsverfahrens mit Rest.

$$\begin{aligned} 2734_{10} &= 341 \times 8 + 6 \\ 341 &= 42 \times 8 + 5 \\ 42 &= 5 \times 8 + 2 \\ 5 &= 0 \times 8 + 5 \end{aligned}$$

Damit

$$2734_{10} = 5256_8.$$

Check:

$$\begin{aligned} 5256_8 &= 5 \times 8^3 + 2 \times 8^2 + 5 \times 8^1 + 6 \times 8^0 \\ &= 5 \times 512 + 2 \times 64 + 5 \times 8 + 6 \\ &= 2734_{10}. \end{aligned}$$

5.5 Hexadezimalsystem

Aufgrund des binären Charakters von digitalen Computerkomponenten, werden alle Arten von Daten durch verschiedene Binärcodes dargestellt. Ungeachtet

dessen, wie vorteilhaft das Binärsystem ist, um Daten in Computern zu codieren, ist dieses System ausgesprochen schwerfällig für Menschen. Daher preferiert man eine kompakte Notation, wenn mit Rohdaten in Computern gearbeitet werden muss.

Diese Notation ist das **Hexadezimalsystem**. Das Hexadezimalsystem ist ein Stellenwertsystem zur Basis $B = 16$, die Zahlen dieses Systems werden also durch 16 Ziffern dargestellt. Da das Dezimalsystem die zehn Ziffern $0, 1, \dots, 9$ bereitstellt, benötigt man weitere sechs Zeichen, um sämtliche Ziffern des Hexadezimalsystems darstellen zu können. Gemäß Konvention wählt man dazu die ersten sechs Großbuchstaben **A, B, C, D, E** und **F**.

Da 16 Symbole notwendig sind, nennt man dieses System **Hexadezimal** und die Symbole nennt hexadezimale Ziffern.

Eine Folge hexadezimaler Ziffern ist eine Zahl im Stellenwertsystem zur Basis 16. Die Umrechnung vom Hexadezimalsystem in das Dezimalsystem und umgekehrt — oder ein anderes Stellenwertsystem — erfolgt nach den bekannten Verfahren.

Beispiel 5.34:

- ❶ Wir rechnen in diesem Beispiel die Hexadezimalzahl $2F4_{16}$ in das Dezimalsystem um:

$$\begin{aligned} 2F4_{16} &= 2_{16} \cdot 16^2 + F_{16} \cdot 16^1 + 4_{16} \cdot 16^0 \\ &= 2_{10} \cdot 16^2 + 15_{10} \cdot 16^1 + 4_{10} \cdot 16^0 \\ &= 756_{10} \end{aligned}$$

Die Umrechnung der Hexadezimalzahl $2F4_{16}$ in das Binärsystem ist einfach. Die Ziffern der Hexadezimalzahl werden einfach durch die jeweiligen Vierergruppen von Bits ersetzt:¹¹

$$2F4_{16} = 0010 \ 1111 \ 0100_2$$

- ❷ Um eine Dezimalzahl in das Hexadezimalsystem umzurechnen, wendet man das gleiche Verfahren an, wie bei der Umrechnung einer Dezimalzahl in das Binärsystem. Die Division (mit ganzzahligem Rest) ist dabei jedoch mit 16 durchzuführen. Wir demonstrieren dies anhand der Umrechnung der Zahl 3521_{10} :

$$\begin{array}{rclcl} 3521 \text{ div } 16 & = & 220 \text{ Rest } 1_{10} & \implies & R_0 = 1_{16} \\ 220 \text{ div } 16 & = & 13 \text{ Rest } 12_{10} & \implies & R_1 = C_{16} \\ 13 \text{ div } 16 & = & 0 \text{ Rest } 13_{10} & \implies & R_2 = D_{16} \end{array}$$

Dies liefert die Darstellung der Dezimalzahl 3521_{10} zu:

$$3521_{10} = DC1_{16}$$

¹¹Siehe die Auflistung in dem Beispiel [5.12].

Hexadezimalzahlen werden aus folgenden Gründen verwendet:

- ☞ Die Notation ist wesentlich kompakter als die Binärschreibweise.
- ☞ In den meisten Computern werden die Daten in Vielfachen von 4 Bit Gruppen organisiert, und daher im hexadezimalen Raster.
- ☞ Es ist sehr einfach, zwischen dem Hexadezimalsystem und dem Binärsystem umzurechnen.

Binäre Ziffern werden dazu in Viererblöcke gruppiert. Jeder möglichen Kombination von vier binären Ziffern wird dann ein Symbol vergeben:

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = A
0011 = 3	1011 = B
0100 = 4	1100 = C
0101 = 5	1101 = D
0110 = 6	1110 = E
0111 = 7	1111 = F

Beispiel 5.35:

Wir betrachten die Binärzahl

$$z = 1011001100001010101000_2$$

Gruppiert man nun diese Bitfolge in Viererblöcke von rechts nach links — man füllt eventuell den Block ganz links mit 0en auf — dann können diese Blöcke einfach durch Hexadezimalziffern ersetzt werden:

$$\begin{aligned} z &= 10 \ 1100 \ 1100 \ 0010 \ 1010 \ 1000 \\ &= 0010 \ 1100 \ 1100 \ 0010 \ 1010 \ 1000 \\ &= 2CC2A8_{16}. \end{aligned}$$

5.6 Übungen

Übung 5.2:

Rechnen Sie folgende Zahlen in die jeweiligen Zahlensysteme um.

$$109_{10} = \dots_2,$$

$$109_{16} = \dots_{10},$$

$$109_{10} = \dots_8,$$

$$5DF_{16} = \dots_{10},$$

$$2011_{10} = \dots_{16},$$

$$1001010101010101_2 = \dots_{16}.$$

Übung 5.3:

Rechnen Sie die folgenden Zahlen in die jeweiligen Zahlensysteme um:

$$0.78515625_{10} = \dots_2,$$

$$0.78515625_{10} = \dots_{16},$$

$$0.1_{10} = \dots_2,$$

$$101.1101_2 = \dots_{10}$$

$$3FD.A2 = \dots_{10}$$

Übung 5.4:

Rechnen Sie die folgenden Zahlen in die jeweils angegebenen Zahlensysteme um:

$$109,65625_{10} = \dots\dots_2,$$

$$7863_{10} = \dots\dots_{16},$$

$$101101.1011_2 = \dots\dots_{10}.$$

Übung 5.5:

(a) Addieren Sie folgende Binärzahlen:

$$11001001$$

$$10011$$

$$11111$$

$$10101111$$

$$101$$

$$1$$

(b) Subtrahieren Sie folgende Binärzahlen:

11001101
10101111

10011
101

111011
111

(c) Multiplizieren Sie die folgenden Binärzahlen:

$$1001 \times 11 = \dots$$

$$1111 \times 101 = \dots$$

$$11.01 \times 10.1 = \dots$$

Kapitel 6

Codierung von Zahlen

6.1 Codierung natürlicher Zahlen

Wie alle anderen Informationen auch, werden positive ganze Zahlen (die nennt man auch natürliche Zahlen und meint damit Zahlen der Form 0, 1, 2, 3, 4, 5,) durch Bitfolgen im Computer dargestellt. Auch hier muß festgelegt werden, welche Bitfolge welcher Zahl zugeordnet ist. Da man immer nur Bitfolgen einer fester Länge N betrachten möchte, resultiert das Problem, dass dadurch nur 2^N verschiedene Zahlen dargestellt werden können.

Sollen nur positive ganze Zahlen dargestellt werden, wird der Wertebereich

$$0 \text{ bis } 2^N - 1$$

gewählt, das entspricht 2^N Zahlen, die mit N Bits dargestellt werden. Die Zuordnung der Bitfolgen zu den entsprechenden natürlichen Zahlen geschieht so, dass die Bitfolge die Binärdarstellung der Zahl darstellt. Nachfolgend sind die Fälle $N=1$ bis $N=4$ einmal explizit aufgelistet:

N = 1	N = 2	N = 3	N = 4
0 0	00 0	000 0	0 000 0
1 1	01 1	001 1	0 001 1
	10 2	010 2	0 010 2
	11 3	011 3	0 011 3
		100 4	0 100 4
		101 5	0 101 5
		110 6	0 110 6
		111 7	0 111 7
			1 000 8
			1 001 9
			1 010 10
			1 011 11
			1 100 12
			1 101 13
			1 110 14
			1 111 15

In allen Fällen stellen die Bits von rechts nach links gelesen die Koeffizienten der (aufsteigenden) Zweierpotenzen 2^N dar. Beispielsweise liest sich die Bitfolge 1101 wie folgt:

$$\begin{aligned}
 1101_2 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
 &= 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \\
 &= 8 + 4 + 1 \\
 &= 13_{10}
 \end{aligned}$$

Mit vier Bits lassen sich also die natürlichen Zahlen von 0 bis 15 abbilden, mit 8 Bits = 1 Byte die 256 Zahlen 0 bis 255 usw. In der Tabelle [6.1] sind die in der IT am häufigsten benutzen Zahlenformate nebst üblicher Bezeichnung für die Darstellung natürlicher Zahlen aufgelistet.

Anzahl Bits	Speichergröße	Wertebereich	Standardname
N = 2		0,...,3	
N = 3		0,...,7	
N = 4		0,...,15	
N = 8	1 Byte	0, ..., 255	unsigned shortint
N = 16	2 Byte	0, ..., 65.535	unsigned integer
N = 32	4 Byte	0,...,4.294.967.295	unsigned longint

Tabelle 6.1: Zahlenformate für die Darstellung natürlicher Zahlen.

Definition 6.10:

Bei der Darstellung natürlicher Zahlen im Rechner beschränkt man sich auf n -Bit Worte. Natürliche Zahlen $0, 1, 2, 3, \dots$ werden durch die **Binärdarstellung** codiert. In der Binärdarstellung beschreibt die Folge von 0en und 1en die Koeffizienten der Potenzen im Stellenwertsystem zur Basis 2.

Die Binärdarstellung ist also eine Abbildung

$$\begin{aligned} c_{2,N} : \{0, 1, 2, \dots, 2^N - 1\} \subset \mathbb{N}_0 &\longmapsto \mathbb{B}^N \\ x &\longmapsto c_{2,N}(x), x \in \{0, 1, \dots, 2^N - 1\} \end{aligned}$$

wobei mit $c_{2,N}(x)$ die Darstellung von x im Binärsystem in N Stellen bezeichnet.

Wie andere Zahlen auch, können Binärzahlen natürlich auch addiert werden. Dies funktioniert genau wie die Addition von Dezimalzahlen, ausgenommen der Tatsache, dass bereits ein Übertrag entsteht, sobald das Ergebnis größer als 1 ist.

+	0	1
0	0	1
1	1	0 Übertrag 1

Tabelle 6.2: + Verknüpfung von 0 und 1.

Zwei aus mehreren Ziffern bestehende Binärzahlen werden addiert, wie man es von der Addition zweier Dezimalzahlen kennt. Ein an einer Ziffernposition entstehender Übertrag wird zur nächsthöheren Position addiert.

Beispiel 6.36:

$$\begin{array}{r} 101 \\ + 11 \\ \text{Übertrag} \quad 111 \\ = 1000 \end{array}$$

was dem allseits bekannten Resultat $5 + 3 = 8$ entspricht.

$$\begin{array}{r} 1\ 001 \\ + \quad 11 \\ + \quad 10\ 111 \\ \text{Übertrag} \quad 11\ 111 \\ = \quad 100\ 011 \end{array}$$

was in Dezimalzahlen der Addition $9 + 3 + 23 = 35$ entspricht. Etwas Aufmerksamkeit erfordert der Fall, wenn der Übertrag aus 'mehreren' Einsen besteht wie im folgenden Fall:

$$\begin{array}{r} 10 \\ + \quad 10 \\ + \quad 10 \\ + \quad 10 \\ \text{Übertrag} \quad 1\ 000 \\ = \quad 1\ 000 \end{array}$$

6.2 Gleitkommazahlen

Bisher haben wir natürliche und ganze Zahlen als Bitfolgen dargestellt. Erstere werden also durch Binärzahlen repräsentiert, die zweite Art von Zahlen durch die Zweierkomplementdarstellung.

Häufig haben es Computer (und die Menschen, die damit arbeiten) aber mit Zahlen zu tun wie $\pi = 3,141592\dots$, 12,50 Euro oder mit sehr großen Zahlen wie¹ 10^{80} oder sehr kleinen Zahlen wie² z.B. 10^{-33} . Die Frage, deren wir uns in diesem Abschnitt zuwenden, ist, wie solche Zahlen in einem Computer dargestellt werden. Siehe auch die Darstellungen in [104, Anhang B], [51, App. A.3], [89], Kapitel 1.3, [22] und [99, Chap. 8.4] und HEROLD *et al.*, [52], Kap. 3.6.

6.2.1 Das IEEE 754 Gleitkommaformat

Generell spricht man von sogenannten Real-Zahlen, die im Rechner als sogenannte Gleitpunktzahlen³ (oder Gleitkommazahlen) gespeichert werden. Ein Computer, der Gleitpunktzahlen durch eine Folge von 32 Bit (= 4 Bytes) darstellt, kann höchstens 2^{32} viele Werte exakt darstellen. Alle anderen Werte sind Näherungen.

Gleitkommazahlen werden zum Beispiel in der Form

$$1,2345 \times 10^5$$

geschrieben. Jede Gleitkommazahl kann zerlegt werden in eine **Mantisse** und einen **Exponenten**. Die Mantisse bezeichnet die signifikanten Dezimalstellen, der Exponent ist die entsprechende Potenz von 10. In obiger Gleitkommazahl ist also 1,2345 die Mantisse und 5 der Exponent.

Die Form mit Mantisse und Exponent ist die Art und Weise, wie Computer Gleitkommazahlen verarbeiten. Der Unterschied zur oben dargestellten Schreibweise besteht jedoch darin, dass der Exponent nicht zur Basis 10 sondern zur Basis 2 genommen wird, da 2 die 'natürliche' Recheneinheit eines Computers ist.

Bis etwa 1980 hatte jeder Computer-Hersteller sein eigenes Gleitkomma-Format, die sich selbstverständlich alle unterschieden. Einige dieser Formate rechneten auch falsch, denn die Gleitkommarithmetik birgt eine Reihe von Feinheiten in sich.

¹Das ist annähernd die Anzahl der Nukleonen im Universum.

²Das ist die sogenannte PLANCK-Länge, eine Längeneinheit, die Physiker aus einer geeigneten Kombination von Naturkonstanten (z.B. Lichtgeschwindigkeit, Gravitationskonstante und PLANCKsches Wirkungsquantum) basteln. Auch Physiker benutzen hin und wieder mal einen Computer.

³Im Englischen nennt man dies floating point numbers.

So rief die IEEE⁴ Ende der siebziger Jahre einen Ausschuß zur Standardisierung der Gleitkommaarithmetik ins Leben. Ziel dieser Standardisierung war nicht nur, den Austausch von Gleitkommazahlen zwischen verschiedenen Plattformen zu ermöglichen, sondern auch Hardware-Designern ein korrektes Modell zur Verfügung zu stellen. Die resultierende Arbeit führte 1985 zum **IEEE-Standard 754**⁵. Die aktuelle Version dieses Standards ist unter der Bezeichnung **ANSI/IEEE 754-2008** veröffentlicht. Dieser Standard wird heute von allen Plattformen (Intel, SPARC, MIPS, JVM,...) genutzt, *i.e.* diese CPUs verfügen über Instruktionen, die diesem Standard entsprechen. Größtenteils ist dieser Standard die Arbeit des Mathematikers WILLIAM KAHAN, University of Berkeley.

Die Grundlage des IEEE 754 Formats ist die Darstellung einer Gleitkommazahl Z in *halblogarithmischer* Schreibweise. Jede solche Zahl kann in der Form

$$Z = \pm m \cdot B^e \quad (6.1)$$

geschrieben werden. Man nennt m die **Mantisse**, B die **Basis** und e den **Exponenten** ist.

Beispiel 6.37:

Betrachten wir die Zahl 0.5 im Dezimalsystem, so können wir diese Zahl im Binärsystem darstellen als

$$\frac{1}{2} = 0.1_2 \cdot 2^0 = 1.0_2 \cdot 2^{-1} = 0.01_2 \cdot 2^1 = 0.0001_2 \cdot 2^3.$$

Damit ist 0.1 oder 1.0 oder 0.01 oder 0.0001 die Mantisse, 2 ist die Basis und 0, oder -1 oder 1 oder 3 der Exponent.

Wie dieses Beispiel zeigt, ist diese Darstellung in der Form (6.1) nicht eindeutig. Um die Eindeutigkeit der Darstellung zu gewährleisten, werden Gleitkommazahlen in *binärer, normalisierter* Form ausgedrückt. Dazu setzt man

1. $\frac{1}{2} \leq m < 1$, *i.e.* die Mantisse in Binärdarstellung hat die Form $0.1b_1b_2\dots$; das binäre Komma wird also so weit nach links verschoben — bei gleichzeitiger Anpassung des Exponenten — dass das Komma *vor* der führenden 1 steht,

oder

2. $1 \leq m < 2$, *i.e.* die Mantisse in Binärdarstellung hat die Form $1.b_1b_2\dots$; das binäre Komma wird also so weit nach links verschoben — bei gleichzeitiger Anpassung des Exponenten — dass das Komma *hinter* der führenden 1 steht.

⁴IEEE = Institute of Electrical and Electronics Engineers.

⁵Siehe auch die Web-Site:

<http://grouper.ieee.org/groups/754>

Werden normalisierte Mantissen vorausgesetzt, dann ist das erste Bit immer eine 1. Da es unnötig ist, dieses Bit zu speichern, ist dieses Bit implizit — man nennt dieses Bit **hidden bit**.

Dies wollen wir an einem Beispiel illustrieren. Wir starten mit der Dezimalzahl 7.625_{10} . Diese Dezimalzahl läßt sich in der Form

$$\begin{aligned} 7.625_{10} &= 4 + 2 + 1 + \frac{1}{2} + \frac{1}{8} \\ &= 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \end{aligned}$$

schreiben. In binärer Form schreibt sich dies daher

$$111.101$$

Die normierte Form erhält man nun dadurch, dass der Exponent so angepaßt wird, dass das binäre Komma rechts von der 1 in der höchsten Stelle steht.⁶ In unserem Fall bedeutet dies, dass das binäre Komma um zwei Stellen nach links verschoben werden muß:

$$1.11101 \times 2^2$$

Im Dezimalsystem ist dies vertraut, dort kann die Zahl 7.625 auch in unterschiedlicher Form geschrieben werden:

$$\begin{aligned} 7.625 &= 0.7625 \times 10^1 \\ &= 76.25 \times 10^{-1} \\ &= 7625 \times 10^{-3} \end{aligned}$$

Wie oben erwähnt ist einer der Vorteile der Normalisierung, dass die 1 vor dem binären Komma (oder Punkt) gar nicht mehr gespeichert werden muß, da sie per Konvention dann sowieso immer vorhanden ist. Normierte Gleitpunktzahlen haben den Vorteil, dass die Mantissenbits optimal ausgenutzt werden, da keine überflüssigen 0-Bits abgespeichert werden müssen.

Im IEEE-Standard wird eine Gleitkommazahl also abgespeichert in der Form

- ⇒ Vorzeichen-Bit,
- ⇒ Mantissebits,
- ⇒ und Exponent Potenz von 2.

Eine normalisierte, nichtverschwindende Gleitkommazahl hat also die Form:

$$\pm 1. bbb \dots b \times 2^{\pm e}$$

⁶Man beachte, dass in der Literatur an dieser Stelle unterschiedlich verfahren wird gemäß der beiden obigen Optionen, wie man normalisieren kann. STALLINGS [99], p. 299 verwendet z.B. die Option 1, während der IEEE 754 Standard die Option 2. vorsieht. Wir verwenden hier Option 2. Beide Optionen sind erlaubt, Option 2 hat den Vorteil eines größeren Bereichs an darzustellenden Zahlen.

wobei b entweder 0 oder 1 ist.

Personalcomputer, die auf Mikroprozessoren der INTEL-Baureihe 80x86 aufbauen, verwenden 32-Bit Gleitpunktzahlen, bei denen das Vorzeichen durch ein Bit dargestellt wird. Dabei gilt:

Vorzeichen-Bit = 0 \longrightarrow positiver Wert
 Vorzeichen-Bit = 1 \longrightarrow negativer Wert

Die **Mantisse** wird binär dargestellt. Der Vorteil der IEEE-Norm besteht nun gerade darin, dass die erste 1 links vom binären Komma nicht abgespeichert werden muß, da in dieser Norm an dieser Stelle immer eine 1 steht.

Da **Exponenten** von Gleitkommazahlen positiv oder negativ sein können, wird der Exponent als ganze Zahl in der sogenannten **Biasdarstellung** gespeichert. Die Biasdarstellung ist also eine Möglichkeit, vorzeichenbehaftete ganze Zahlen in Rechnern zu codieren.

Intermezzo: Biasdarstellung

Das Vorzeichen des Exponenten wird in der Biasdarstellung dadurch codiert, dass der Zahlenbereich um einen Basiswert (bias) verschoben wird. Der verschobene Exponent wird als *Charakteristik* oder *biased exponent* bezeichnet. Der Grund für die Verwendung dieser Codierung — und nicht die Zweierkomplementdarstellung — liegt darin, dass Exponenten für Gleitkommaarithmetik schnell verglichen werden können. Für die Umrechnung gilt also:

$$\text{Charakteristik} = \text{Biased Exponent} = \text{Exponent} + \text{Bias}$$

Bei der Charakteristik ist der kleinste und der größte Wert für Ersatzdarstellungen reserviert.

Beispiel 6.38:

Wir betrachten eine Darstellung des Exponenten mit $n = 8$ Bit. Der reale Exponent kann damit die 256 ganzzahligen Werte von -127 bis +128 annehmen.⁷ Dieser Zahlenbereich wird nun um den Betrag 127 in den positiven Zahlenbereich verschoben. Die daraus resultierenden Randwerte 0 und 255, binär 0000 0000 und 1111 1111 haben eine besondere Bedeutung, sie codieren die 0 bzw. $+\infty$. Die Charakteristik ist damit eine Zahl im ganzzahligen Bereich [1,254], siehe Abbildung [6.1].

Diese Charakteristik wird binär abgespeichert.

Ist der Exponent beispielsweise 45, dann ist:

$$\text{Charakteristik} = 45 + 127 = 172 \longrightarrow 10101100$$

⁷Man beachte, dass in der Biasdarstellung der Wertebereich der darstellbaren ganzen Zahlen um 1 gegenüber dem Wertebereich bei der Zweierkomplementdarstellung verschoben ist.

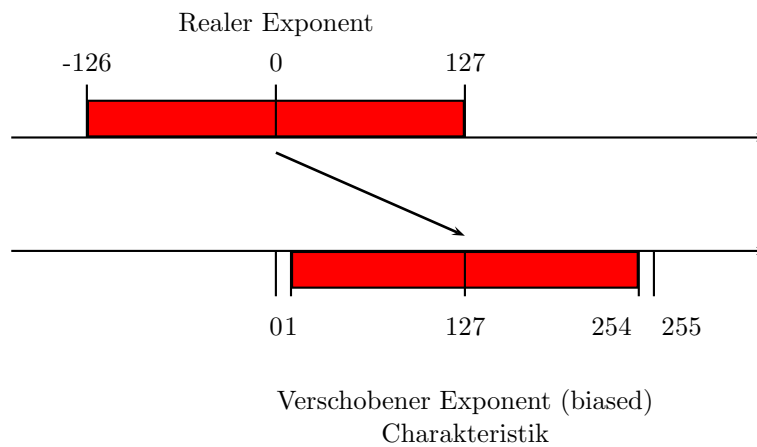


Abbildung 6.1: Verschiebung des Exponenten zur Bildung der Charakteristik.

Ist der Exponent -33, dann ist:

$$\text{Charakteristik} = -33 + 127 = 94 \longrightarrow 01011110$$



Abbildung 6.2: Speicherung einer Gleitkommazahl in einem 32 Bit Format.

Das `float` Format der IEEE 754 Gleitkommadarstellung sieht vor, dass eine Gleitkommazahl in einem 32-Bit Format abgespeichert (siehe Abbildung [6.2]) wird. Dieses Format wird folgendermaßen aufgeteilt:

- ☞ 1 Bit für das Vorzeichen,
- ☞ 8 Bits für den Exponenten
- ☞ die restlichen 23 Bits sind für die Mantisse reserviert.

Für die Speicherung des Exponenten stehen also 8 Bits als Speicherplatz zur Verfügung. Da der Exponent in der Biasdarstellung deklariert ist, kann dieser Teil der Gleitkommazahl die Werte -126 bis +127 annehmen:

$$\text{Exponent} \in [-126, +127]$$

Beispiel 6.39:

Wir betrachten in diesem Beispiel die IEEE Gleitkommadarstellung der Dezimalzahl

$$x = 472,56640625$$

Die Umrechnung in das Binärsystem mit anschließender Normalisierung ergibt:

$$\begin{aligned} x &= 472,56640625 \\ &= 256 + 128 + 64 + 16 + 8 + \frac{1}{2} + \frac{1}{16} + \frac{1}{256} \\ &= 1 \cdot 2^8 + 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + \\ &\quad + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + 0 \cdot 2^{-7} + 1 \cdot 2^{-8} \\ &= 111\ 011\ 000.100\ 100\ 01 \\ &= 1.110\ 110\ 001\ 001\ 000\ 1 \times 2^8 \end{aligned}$$

Damit wird diese Zahl im **float** Format folgendermaßen dargestellt:

☞ Das Vorzeichen 1 Bit $\rightarrow 0$

☞ Der Exponent ist dezimal 8 \rightarrow die Biasdarstellung ist $8 + 127 = 135$

$$135 \rightarrow 1000\ 0111$$

☞ Die signifikanten Nachkommastellen

$$110110001001000100000000$$

Damit:

$$472,56640625 = 0\ 1000111\ 11011000010010000100000000$$

Dieses Speicherschema impliziert nun, dass der kleinste Wert, der durch das 32 Bit **float** Format abgespeichert werden kann, durch 2^{-126} gegeben ist. Dies entspricht in etwa 10^{-38} . Der größte Wert ist 2^{128} , was ungefähr $3,4 \times 10^{+38}$ ist.

Die Rechnung geht folgendermaßen: Die vollständige Darstellung der kleinsten Zahl in binärer Schreibweise ist $1,0 \times 2^{-126}$, was dezimal durch 2^{-126} gegeben ist. Die größte Zahl ist $1, \times 2^{127}$, also in Dezimalschreibweise gerade 2^{128} .

$$1,11111111111111111111111111111111 \times 2^{127}$$

Im `double` Format des IEEE 754 Standards werden Gleitkommazahlen im 64 Bit Format abgespeichert. Ein Bit ist für das Vorzeichen reserviert, 11 Bits für den Exponenten in der Biasdarstellung und die restlichen 52 Bits speichern die 53stellige Mantisse. Damit wird der Wertebereich von 2^{-1022} (etwa $2,2 \times 10^{-308}$) bis 2^{1024} (etwa $1,8 \times 10^{308}$) abgedeckt.

Die unterschiedlichen Speicherkapazitäten der Mantisse im 32 Bit bzw. 64 Bit Format hat zur Folge, dass die Rechengenauigkeit für Gleitkommazahlen im 64 Bit Speicherformat wesentlich höher ist als die von Gleitkommazahlen im 32 Bit Speicherformat.

Die IEEE 754 Darstellung ermöglicht keine Darstellung der Zahl `0.0` oder ∞ . Die Gleitkommadarstellung reserviert jedoch spezielle Bitmuster für diese Zahlen.

Es gilt die Konvention:

- ☞ Vorzeichen: 0 oder 1
- ☞ Biased Exponent: 0
- ☞ Mantissenbits: Alle 0
- ☞ Gleitkommazahl: `0.0`

und:

- ☞ Vorzeichen: 0 oder 1
- ☞ Biased Exponent: `11111111`
- ☞ Mantissenbits: Alle 0
- ☞ Gleitkommazahl: $\pm\infty$

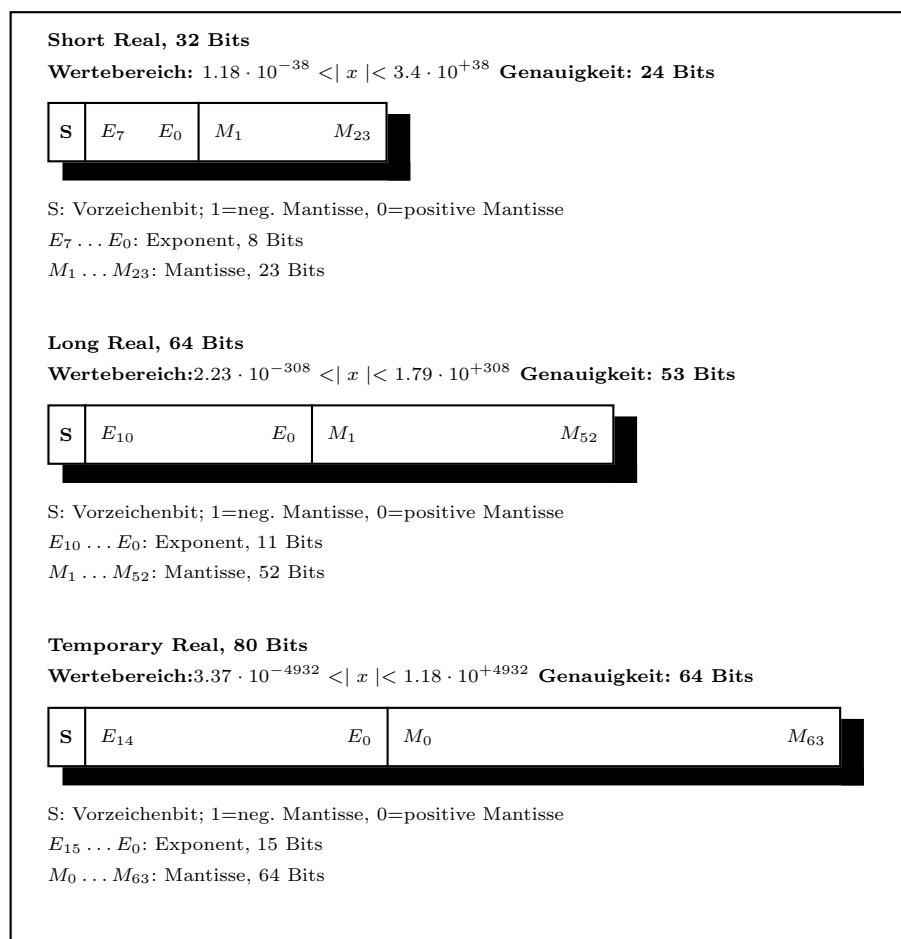


Abbildung 6.3: Die drei IEEE Formate für short real, long real und temporary real.



Abbildung 6.4: Verteilung der Gleitkommazahlen.

6.2.2 Genauigkeit

Ein wesentlicher Punkt, den wir bisher noch nicht diskutiert haben, ist die Genauigkeit, mit der reelle Zahlen dargestellt werden können. Die Abbildung [6.4] zeigt die Verteilung der Fließkommazahlen, die in einem typischen Computersystem gespeichert werden können. Die Abbildung [6.4] zeigt drei disjunkte Bereiche:

- positive Zahlen $\epsilon \leq n \leq \omega$
- 0.0
- und negative Zahlen $-\omega \leq n \leq -\epsilon$

Die Zahl ω ist die größte positive Zahl, die im Short-Real-Format des IEEE Standards abgespeichert werden kann, $\omega \sim 10^{38}$. Dagegen ist ϵ die kleinste positive Zahl, von der Größenordnung 10^{-47} .

Es gibt hier drei Aspekte, die zu beachten sind:

1. Der Zahlenbereich zwischen $-\omega$ und $+\omega$ umfaßt grob 10^{38} ganze Zahlen. Das IEEE-Format Short Real verfügt aber nur über 32 Bits; damit lassen sich also 2^{32} Zahlen darstellen, was etwa 10^9 ganze Zahlen sind. Daraus resultiert die Tatsache, dass es in diesem Zahlenbereich Zahlen gibt, die sich in der IEEE-Short-Real Darstellung nicht codieren lassen. Wann immer eine Berechnung auf solch eine Zahl führt, entsteht ein Rundungsfehler dadurch, dass das System das Ergebnis durch die (hoffentlich) nächstgelegene darstellbare Zahl approximiert.
2. Es klafft eine kleine Lücke zwischen der kleinsten darstellbaren positiven Zahl ϵ und 0.0. Ein Rundungsfehler in einer Berechnung, die anstelle eines kleinen Wertes als Resultat die 0.0 liefert, nennt man *Underflow*.

3. Die Zahlen, die dargestellt werden können, sind nicht gleichmäßig über den Bereich $[\epsilon, \omega]$ verteilt. Die darstellbaren Zahlen liegen sehr dicht in der Umgebung von 0.0, die Abstände wachsen jedoch stetig, wenn die Größenordnung der Zahlen steigt. Es ist einfach zu verstehen, dass die Abstände der darstellbaren Zahlen nicht gleichmäßig ist, wenn man untersucht, welche beiden Zahlen dargestellt werden können durch zwei aufeinanderfolgende Mantissenwerte bei einem gegebenen festen Exponenten. Um die Diskussion etwas überschaubarer zu gestalten, betrachten wir ein 16-Bit System mit einer 7-Bit Mantisse und einem 8-Bit Exponenten. Ungeachtet dessen, wie groß der Exponent ist, der Unterschied zweier aufeinanderfolgender Werte der Mantisse – beispielsweise zwischen $0.111\ 000\ 0_2$ und $0.111\ 000\ 1_2$ – ist $0.000\ 000\ 1_2$, was dezimal $2^{-7} = 0.0078125_{10}$ ist. Für kleine Zahlen in der Nähe von 0.0 ist der Exponent eine negative Zahl, beispielsweise -100. Dann ist der Abstand zweier aufeinanderfolgenden Gleitkommazahlen von der Größenordnung

$$0.000\ 000\ 1_2 \times 2^{-100} \sim 0.0078 \times 10^{-30} = 7.8 \times 10^{-33}$$

Am anderen Ende der Skala – wo der Exponent groß ist, ist der Unterschied zwischen zwei aufeinanderfolgenden Zahlen von der Größenordnung 2^{100} , denn

$$0.000\ 000\ 1_2 \times 2^{100} \sim 0.0078 \times 10^{30} = 7.8 \times 10^{27}$$

Die Fehler, die durch die beschränkte Genauigkeit der Gleitkommazahlen entstehen, sind zunächst sehr klein, können sich jedoch bei langen Rechnungen sehr schnell vergrößern.

Während des Desert Storms traf im Februar 1991 eine iranische Scud-Rakete eine Kaserne der US-Armee in Dhahran (Saudi Arabien), trotz des vorhandenen Patriot-Raketenabwehrsystems. 28 Soldaten kamen ums Leben, weil eine auf Gleitkommazahlen basierende Berechnung immer ungenauer wurde je länger das zugehörige Radarsystem ununterbrochen in Betrieb war. Nach 100 Stunden konnte das Radarsystem die anfliegende Rakete nicht mehr korrekt erfassen und ignorierte sie als Messfehler. Details sind im offiziellen Untersuchungsbericht verfügbar.⁸

6.2.3 Gleitkomma Arithmetik

Die arithmetischen Operationen für Gleitkommazahlen sind wesentlich komplexer als die entsprechenden Operationen auf Integerzahlen:

- ☞ Bei Addition und Subtraktion müssen die Exponenten der beiden Operanden auf den gleichen Wert gebracht werden.
- ☞ Mantisse und Exponent müssen getrennt behandelt werden.

⁸Siehe <https://www.gao.gov/products/IMTEC-92-26>.

Die Gleitkomma Arithmetik für die Grundrechenarten wird für zwei nicht normalisierte Zahlen

$$x_1 = m_1 B^{e_1}$$

und

$$x_2 = m_2 B^{e_2}$$

durch die folgenden vier Regeln festgelegt:

$$x_1 + x_2 = (m_1 + m_2 B^{e_2 - e_1}) B^{e_1}$$

$$x_1 - x_2 = (m_1 - m_2 B^{e_2 - e_1}) B^{e_1}$$

$$x_1 \times x_2 = (m_1 \times m_2) B^{e_1 + e_2}$$

$$x_1 / x_2 = (m_1 / m_2) B^{e_1 - e_2}$$

Beispiel 6.40:

Um das Prinzip der Addition von Gleitkommazahlen zu verstehen, sehen wir uns an, wie man im Dezimalsystem Gleitkommazahlen addiert. Dazu betrachten wir die beiden Zahlen:

$$x = 1.75 \cdot 10^3$$

und

$$y = 0.25 = 0.25 \cdot 10^0.$$

Um diese beiden Zahlen korrekt addieren zu können, müssen in einem ersten Schritt die Exponenten auf den gleichen Wert gebracht werden. Dazu wird der größere Exponent auf den Wert des kleineren gebracht, indem das Komma um entsprechend viele Stellen nach rechts verschoben wird (oder umgekehrt).

$$x = 1.75 \cdot 10^3 = 1750.0 \cdot 10^0; \quad y = 0.25 \cdot 10^0$$

Dann werden in einem zweiten Schritt die Mantissen addiert:

$$\begin{aligned} x + y &= 1750.0 \cdot 10^0 + 0.25 \cdot 10^0 \\ &= 1750.25 \cdot 10^0 \\ &= 1.75025 \cdot 10^3 \end{aligned}$$

Beispiel 6.41:

In diesem ausführlichen Beispiel betrachten wir die beiden Dezimalzahlen

$$13,75 \quad 1,125$$

1. Die beiden Zahlen werden im normalisierten binären 10 Bit Gleitkommaformat dargestellt. Dazu stehen zur Verfügung:
 - 1 Bit Vorzeichen (0 = positiv, 1 = negativ)
 - 4 Bit Exponent in Biasdarstellung
 - 5 Bit Mantisse
2. Anschließend addieren wir die beiden Zahlen in obiger binären Darstellung.
3. Schließlich bestimmen wir beim Resultat den Rundungsfehler, der durch die Darstellung auftritt.

1. Es gilt:

$$\begin{aligned} 13.75 &= 8 + 4 + 1 + 0.5 + 0.25 \\ &= 2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-2} \end{aligned}$$

Damit ist

$$\begin{aligned} 13.75 &\longrightarrow 1101.11 \\ &\xrightarrow{\text{Normalisierung}} 1.10111 \cdot 2^3 \end{aligned}$$

Für die Speicherung des Exponenten ist eine 4-Bit Biasdarstellung vorgesehen. Daher ist der Wertebereich des (realen) Exponenten das Intervall $[-7, +8]$. Um die Charakteristik zu erhalten, wird dieser reale Exponent um den Wert 7 (bias) verschoben; damit:

$$\text{Charakteristik} = 3 + 7 = 10_{10} = 1010_2$$

In der 10 Bit Gleitkommadarstellung wird $13,75_{10}$ damit folgendermaßen dargestellt:

$$13,75 \longrightarrow 0\ 1010\ 10111.$$

Analog erhält man:

$$\begin{aligned} 1,125 &= 1 + 0.125 \\ &= 2^0 + 2^{-3} \end{aligned}$$

Damit ist

$$\begin{aligned} 1,125 &\longrightarrow 1.001 \\ &\xrightarrow{\text{Normalisierung}} 1.001 \cdot 2^0. \end{aligned}$$

Damit wird ist die Charakteristik:

$$\text{Charakteristik} = 0 + 7 = 7_{10} = 0111_2$$

In der 10 Bit Gleitkommadarstellung wird die Zahl 1.125_{10} daher folgendermaßen dargestellt:

$$1.125_{10} \longrightarrow 0\ 0111\ 00100$$

2. Um die beiden Zahlen im binären Gleitkommaformat addieren zu können, müssen in einem ersten Schritt die Exponenten der beiden Zahlen angepaßt werden. Es muß daher die Zahl 1.125 auf die Form

$$1.125 \longrightarrow 1.001 \cdot 2^0 \longrightarrow 0.001001 \cdot 2^3$$

gebracht werden, damit die beiden Mantissen der Zahlen addiert werden können. Man nennt diesen Zwischenschritt **Denormalisierung**. Im 10 Bit Gleitkommaformat führt dies zu einer Verschiebung wie folgt:

$$\begin{aligned} 1.125 &\longrightarrow 0\ 0111\ 00100 \\ &\longrightarrow 0\ 1010\ 00100(1) \end{aligned}$$

Die verschobene (1) kann in diesem Format nicht mehr dargestellt werden (Rundungsfehler), da lediglich fünf Mantissenbits gespeichert werden können. Damit ist:

$$\begin{array}{r} 0\ 1010\ 10111 \\ +\ 0\ 1010\ 00100 \\ \hline 0\ 1010\ 11011 \end{array}$$

3. Wird die Summe der beiden Zahlen als Dezimalzahl im Zehnersystem umgeschrieben, folgt:

$$\begin{aligned} 0\ 1010\ 11011 &\longrightarrow 1.11011 \cdot 2^3 \\ &\longrightarrow 1110.11 \\ &\longrightarrow 2^3 + 2^2 + 2^1 + 2^{-1} + 2^{-2} \\ &\longrightarrow 8 + 4 + 2 + 0.5 + 0.25 \\ &\longrightarrow 14.75. \end{aligned}$$

Dies führt auf einen Rundungsfehler von

$$\Delta = 0.125$$

was gerade der 1 entspricht, die durch die Anpassung des Exponenten in der 10 Bit Gleitkommadarstellung nicht mehr dargestellt werden kann.

Beispiel 6.42:

In diesem Beispiel berechnen wir die größte und die kleinste Zahl, die in dem obigen 10-Bit Gleitkommaformat exakt abgespeichert werden kann. Weiterhin kann damit auch die *Genauigkeit* der gewählten Gleitkommadarstellung festgestellt werden.

1. Die in dem 10 Bit Gleitkommaformat größte darstellbare Zahl hat die Darstellung

$$z_1 \longrightarrow 0\ 1110\ 1111$$

denn die Biasdarstellung reserviert den Exponenten 1111 für das Symbol ∞ . Daher ist die Charakteristik 14. Mit der Verschiebung von 7 erhält man den realen Exponenten zu 7. Daher:

$$\begin{aligned} z_1 &= 1.11111 \times 2^7 \\ &= 111111 \times 2^2 \\ &= 63 \times 2^2 \\ &= (64 - 1) \times 2^2 \\ &= (2^6 - 1) \times 2^2 \\ &= 2^8 - 2^2 \\ &= 256 - 4 \\ &= 252_{10}. \end{aligned}$$

Die zweitgrößte darstellbare Zahl wird durch das Bitmuster

$$z_2 = 0\ 1110\ 1110$$

gespeichert. Die Recodierung dieser Bitfolge ergibt die Zahl:

$$\begin{aligned} z_2 &= 1.11110 \times 2^7 \\ &= 111110 \times 2^2 \\ &= 62 \times 2^2 \\ &= (64 - 2) \times 2^2 \\ &= (2^6 - 2) \times 2^2 \\ &= 2^8 - 2 \cdot 2^2 \\ &= 256 - 8 \\ &= 248_{10}. \end{aligned}$$

Damit hat man eine Genauigkeit von 4 bei großen Zahlen.

2. Die kleinste darstellbare Zahl im 10 Bit Gleitkommaformat hat die Darstellung

$$z_3 \longrightarrow 0\ 0001\ 0000$$

Denn für die 4-Bit Biasdarstellung ist der kleinstmögliche Wert der Charakteristik +1, die 0000 ist reserviert für die Zahl 0.0. Da die Charakteristik 1 ist und eine Verschiebung um +7 stattfindet, ist der reale Exponent -6, daher:

$$\begin{aligned} z_3 &= 1.00000 \cdot 2^{-6} \\ &= 1.0 \cdot 2^{-6} \\ &= \frac{1}{64} = 0.015625. \end{aligned}$$

Die nächst größere Zahl hat die Darstellung:

$$z_4 \longrightarrow 0\ 0001\ 00001$$

was die Entwicklung

$$\begin{aligned} z_4 &= 1.00001 \cdot 2^{-6} \\ &= 1.00001 \cdot 2^{-6} \\ &= \frac{1}{64} + \frac{1}{2048} = 0.01611328125. \end{aligned}$$

Der Fehler liegt bei kleinen Gleitkommazahlen also lediglich bei

$$\delta z = \frac{1}{2048} = 0.00048828125.$$

Kapitel 7

Digitale Schaltungen

7.1 Gatter

Bei digitalen Schaltungen gibt es genau zwei logische Werte. Üblicherweise stellt ein Signal zwischen 0 und 1 Volt ein Wert dar (üblicherweise per Konvention die binäre 0) und ein Signal zwischen 2 und 5 Volt einen anderen Wert (die binäre 1). Es gibt nun elementare Schaltungen (Bausteine) — aus denen digitale Schaltungen aufgebaut werden — die mit solchen zweiwertigen Signalen eine Reihe von Funktionen berechnen können. Diese Bausteine heißen:

Gatter

oder auch *Gates*¹. Mit Hilfe von Gattern werden also in Rechnern die grundlegenden logischen Funktionen implementiert. So implementiert das **AND**-Gatter die BOOLEsche **AND**-Funktion und ein **OR**-Gatter implementiert die BOOLEsche **OR**-Funktion. Da die BOOLEschen **AND** und **OR**-Operationen kommutativ und assoziativ sind, können **AND** und **OR** Gatter mehrere Inputs haben, der Output ist immer die **AND** bzw. **OR** Verknüpfung des gesamten Inputs.

Definition 7.11:

Eine Schaltung zur Realisierung spezieller schaltalgebraischer Verknüpfungen nennt man **Schaltgatter** oder kurz nur **Gatter**.

Um zu verstehen, wie die Gatter operieren, ein kurzer Blick zurück auf die Physik. Jede moderne digitale Logik basiert auf der Transistortechnologie.

In der Abbildung [7.1] ist das Grundprinzip der Transistorschaltung dargestellt. Ein Transistor verfügt über drei Ausgänge:

¹Ähnlichkeiten mit Personen sind hier nicht beabsichtigt oder auch nur relevant.

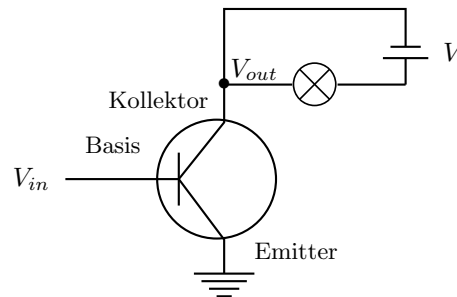


Abbildung 7.1: Ein Transistor, der als Schaltung ein NOT Gatter darstellt.

- den Kollektor,
- die Basis,
- und den Emitter.

Liegt die Eingangsspannung V_{in} auf der Basis unter einem kritischen Wert, schaltet sich der Transistor aus und stellt de facto einen unendlichen Widerstand dar. In diesem Zustand nimmt die Ausgangsspannung V_{out} den Wert V an, eine externe Spannung, die bei den uns interessierenden Transistortypen bei etwa 5 Volt liegt. Mit anderen Worten, der Stromkreis in der Abbildung [7.1] oben rechts ist geschlossen und die skizzierte Lampe brennt. Überschreitet V_{in} den kritischen Wert, schaltet sich der Transistor ein, wodurch der skizzierte Stromkreis geerdet wird, *i.e.* V_{out} wird auf Masse (Erde), d.h. ~ 0 Volt, geschaltet. Dann ist die in Abb. [7.1] skizzierte Lampe aus.

Diese Tatsache impliziert folgende Schaltungslogik:

$$\begin{aligned} \text{Wenn } V_{in} \text{ niedrig} &\implies V_{out} \text{ hoch} \\ \text{Wenn } V_{in} \text{ hoch} &\implies V_{out} \text{ niedrig} \end{aligned}$$

Daraus folgt, dass ein Transistor nichts anderes als einen **Inverter** darstellt, der eine logische 0 in eine 1 transformiert und eine logische 1 in eine 0. Bezeichnen wir daher das Signal der Eingangsspannung V_{in} eines Transistors als BOOLEsche Variable a und die Ausgangsspannung V_{out} als BOOLEsche Variable b , und vereinbaren wir die folgende Zuordnung:

$$\begin{aligned} \text{Hoch } (\sim V_{cc}) &\iff \text{logische 1,} \\ \text{Niedrig } (\sim 0 \text{ Volt}) &\iff \text{logische 0,} \end{aligned}$$

so können wir einem Transistor die folgende Schaltungslogik zuordnen:

a	b
0	1
1	0

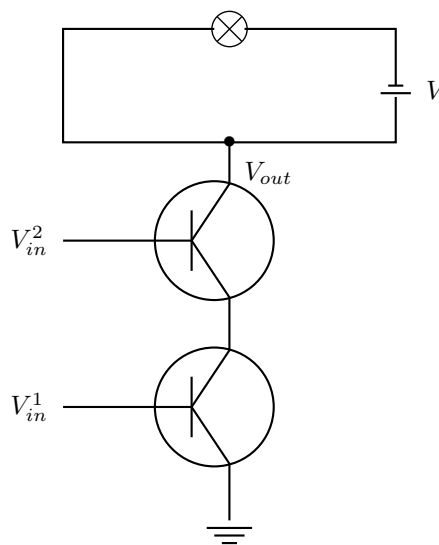


Abbildung 7.2: Zwei seriell geschaltete Transistoren, die ein **NAND**-Gatter darstellen.

Das zweite Schaltbild, Abbildung [7.2], zeigt zwei *in Reihe* (oder auch *seriell*) geschaltete Transistoren. Hier gilt folgende Schaltlogik:

ist V_{in}^1 und V_{in}^2 hoch	\Rightarrow	V_{out} niedrig
ist V_{in}^1 hoch, V_{in}^2 niedrig	\Rightarrow	V_{out} hoch
ist V_{in}^1 niedrig, V_{in}^2 hoch	\Rightarrow	V_{out} hoch
ist V_{in}^1 und V_{in}^2 niedrig	\Rightarrow	V_{out} hoch

Setzen wir wieder $a \leftarrow V_{in}^1$, $b \leftarrow V_{in}^2$ und $c \leftarrow V_{out}$, dann läßt sich die Reihenschaltung zweier Transistoren durch folgende Schaltungslogik beschreiben:

a	b	c
0	0	1
0	1	1
1	0	1
1	1	0

Dies ist die Schalttable der BOOLEschen **NAND**-Funktion. Zwei in Reihe geschaltete Transistoren realisieren daher ein **NAND**-Gatter.

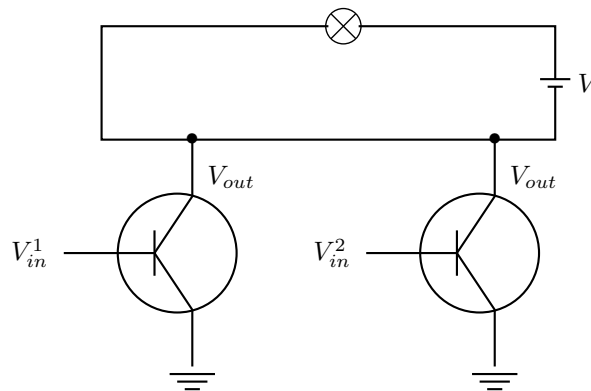


Abbildung 7.3: Zwei parallel geschaltete Transistoren. Diese Schaltung bildet ein sogenanntes **NOR**-Gatter.

Die dritte Schaltung in Abbildung [7.3] zeigt zwei *parallel* geschaltete Transistoren. Hier kann man folgende Schaltungslogik ablesen:

ist V_{in}^1 und V_{in}^2 hoch	\Rightarrow	V_{out} niedrig
ist V_{in}^1 hoch, V_{in}^2 niedrig	\Rightarrow	V_{out} niedrig
ist V_{in}^1 niedrig, V_{in}^2 hoch	\Rightarrow	V_{out} niedrig
ist V_{in}^1 und V_{in}^2 niedrig	\Rightarrow	V_{out} hoch

Setzen wir wieder $a \leftarrow V_{in}^1$, $b \leftarrow V_{in}^2$ und $c \leftarrow V_{out}$, dann lässt sich die Parallelschaltung zweier Transistoren durch folgende Schaltungslogik beschreiben:

a	b	c
0	0	1
0	1	0
1	0	0
1	1	0

Diese drei Schaltungen bilden die einfachsten Gatter. Sie heißen

NOT, NAND bzw. NOR – Gatter

Für die Schaltgatter gibt es eine feste Symbolik, seit 1976 ist sie in Deutschland festgelegt durch die Norm DIN 40700. International werden andere Symbole benutzt. Diese Symbole sind in der Abbildung [7.5] dargestellt.

Beispiel 7.43:

In diesem Beispiel wollen wir die Schaltung $a \longrightarrow b$ durch die elementaren Gatter darstellen. Wir können die Operation IMPLIKATION darstellen in der Form

$$f(a, b) = a \longrightarrow b \equiv \bar{a} \vee b.$$

Dies ergibt die in Abbildung [7.4] dargestellte Schaltfunktion.

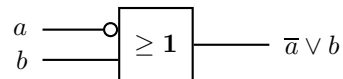


Abbildung 7.4: Schaltfunktion für $a \longrightarrow b$.

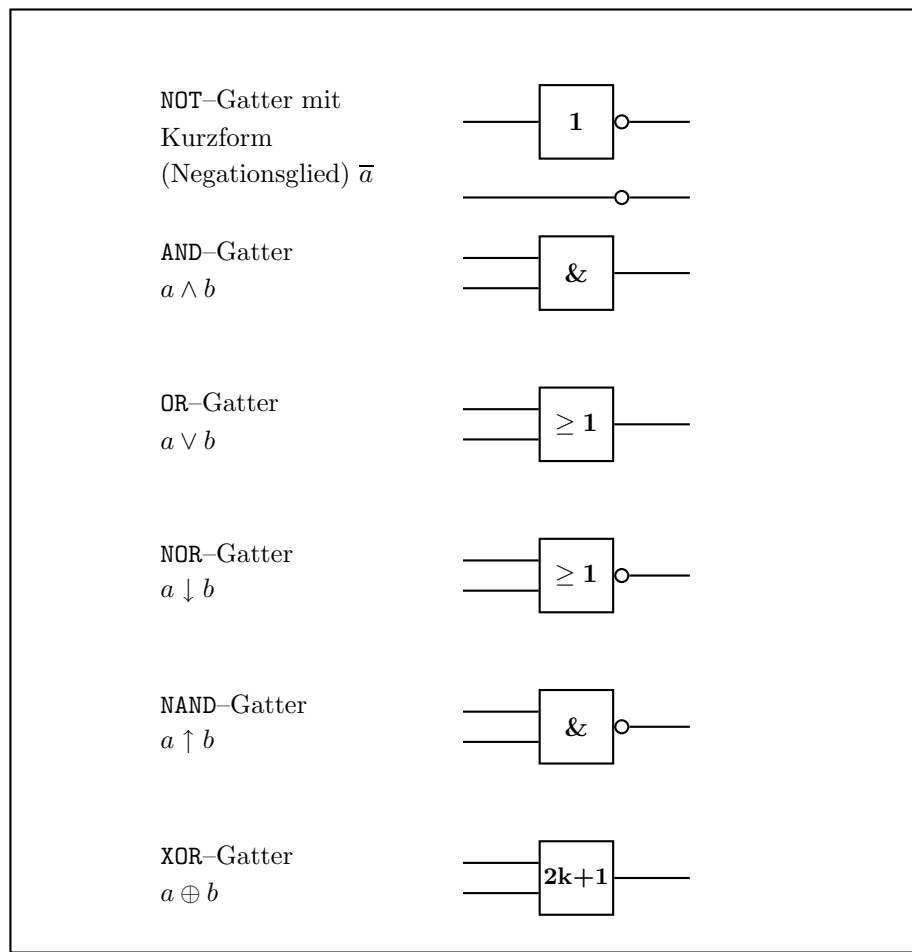


Abbildung 7.5: Darstellung der Gatter nach DIN 40700.

7.2 Schaltnetze

Mit einem Schaltgatter allein können viele Operationen – wie zum Beispiel die Addition – nicht realisiert werden. Dies ist nur dann möglich, wenn mehrere Gatter zu einer größeren Einheit zusammengefaßt werden. Zu diesem Zweck zunächst eine Definition.

Definition:

Ein **Schaltnetz** \mathbf{F} ist die technische Realisierung einer Abbildung

$$f : \mathbb{B}^n \longrightarrow \mathbb{B}^m$$

$$a = (a_1, \dots, a_n) \longmapsto f(a) = (f_1(a), f_2(a), \dots, f_m(a))$$

Dabei bezeichnet:

a_i Schaltzustände an den Eingängen von \mathbf{F} (n Eingänge)

f_j sind die Schaltfunktionen

$f_j(a)$ sind Schaltzustände an den Ausgängen von \mathbf{F} (m Ausgänge)

In Form eines Symbols kann man sich ein Schaltnetz wie in Abbildung [7.6] vorstellen. Die Abbildung f ist im allgemeinen eine Zusammenfassung mehrerer Schaltfunktionen, im Fall $m = 1$ – also ein Ausgang – ist f selbst eine Schaltfunktion und \mathbf{F} eine Schaltung.

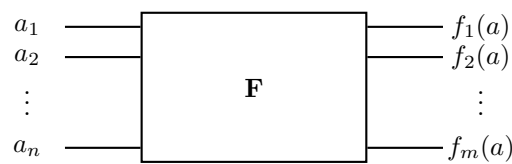


Abbildung 7.6: Schema eines Schaltnetzes.

Im folgenden sehen wir uns einige wichtige Schaltnetze an, die in Computern eine große Rolle spielen.

7.2.1 Halbaddierer

Ein Computer, der nicht addieren kann, ist kein Computer. Folglich sind Schaltungen zur Durchführung der Additionsoperation ein essentieller Bestandteil einer jeden CPU². In der Tabelle [7.1] ist die Wahrheitstabelle einer 1-Bit Addition gezeigt. Hier gibt es zwei Ausgänge,

a	b	Summe	Übertrag
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabelle 7.1: Wahrheitstabelle für eine 1-Bit Addition. Die Spalte **Summe** ist nichts anderes als $a \text{ XOR } b$ und der Übertrag ist $a \text{ AND } b$.

².. egal ob Intel inside oder nicht.

- die Summe der Eingänge a und b
- der Übertrag zur nächsten Stelle (nach links)

Mit anderen Worten, wir haben ein Schaltnetz mit zwei Eingängen und zwei Ausgängen vorliegen. Der Halbaddierer realisiert dabei folgende Abbildung:

$$\begin{aligned} \mathbb{F} : \mathbb{B}^2 &\longrightarrow \mathbb{B}^2 \\ (a, b) &\mapsto \mathbb{F}(a, b) = (a \text{ XOR } b, a \text{ AND } b). \\ &= (a \oplus b, a \wedge b). \end{aligned}$$

Der Halbaddierer besteht also einerseits aus einem **XOR**-Gatter und einem **AND**-Gatter. Die Abbildung [7.7] zeigt eine Schaltung zur Berechnung von Summe und Übertrag. Dies ist also das Schaltnetz für den **Halb-Addierer**. Nun ist ein solcher Halbaddierer gut und schön für die Addition der niederwertigen Bits zweier aus mehreren Bits bestehenden Eingabewörtern, reicht aber nicht aus für eine Bitstelle in der Wortmitte, denn der Übertrag wird nicht durchgeführt. Zu diesem Zweck benötigt man den sogenannten **Volladdierer**.

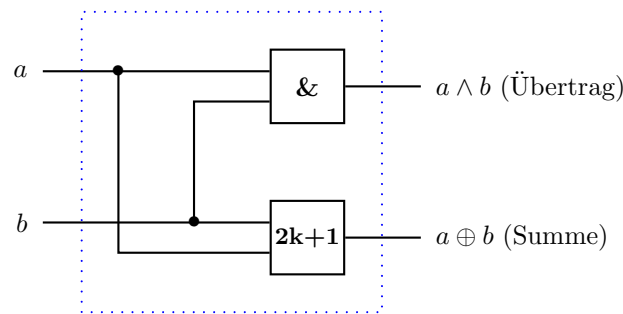


Abbildung 7.7: Schaltnetz für den Halbaddierer.

Für folgende Zwecke ist es sinnvoll, den Halbaddierer als Blockschaltung anzugeben wie in der Abbildung [7.8] gezeigt.

Wir bezeichnen die Ausgänge des Halbaddierers mit $\mathbf{HA}_1(a, b)$ und $\mathbf{HA}_2(a, b)$, wobei nach der vorangehenden Diskussion also gilt:

$$\mathbf{HA}_1(a, b) = a \oplus b \quad (\text{ Summenbit}), \quad (7.1a)$$

$$\mathbf{HA}_2(a, b) = a \wedge b \quad (\text{ Übertragbit}). \quad (7.1b)$$

Bevor wir uns den Volladdierer ansehen, eine

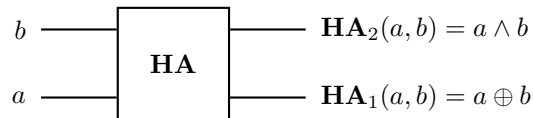


Abbildung 7.8: Der Halbaddierer als Blockschaltbild.

Anmerkung:

Die XOR-Funktion kann dargestellt werden in der Form:

$$\begin{aligned}
 a \text{ XOR } b &= a \oplus b \\
 &= ((\text{NOT } a) \text{ AND } b) \text{ OR } (a \text{ AND } (\text{NOT } b)) \\
 &= (\bar{a} \wedge b) \vee (a \wedge \bar{b}).
 \end{aligned}$$

Dementsprechend kann das XOR-Schaltungselement durch AND, OR und NOT Gatter dargestellt werden. Dies ist in Abbildung [7.9] zu sehen.

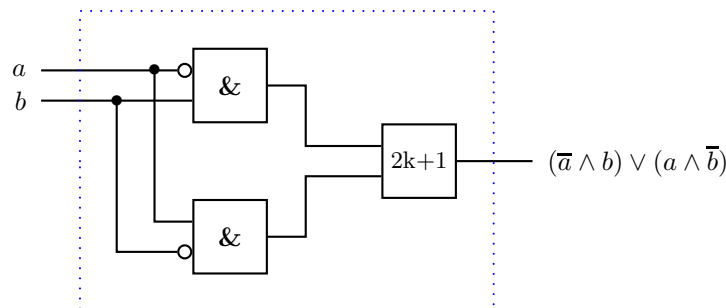


Abbildung 7.9: Darstellung eines XOR-Gatters durch NOT, AND und OR-Gatter.

Dieser Aspekt stellt eigentlich ein sehr wichtiger Punkt dar, es handelt sich um die sogenannte **Schaltungsäquivalenz**. Entwickler von Schaltungen versuchen in der Regel, die Anzahl von Gates in den Produkten zu minimieren. Ziel dabei ist natürlich, Kosten zu senken, Platz auf dem Chip zu sparen und den Stromverbrauch zu reduzieren. Zur Verringerung der Komplexität einer Schaltung muß der Entwickler eine andere Schaltung finden, die zwar exakt die gleiche

Funktionalität wie das Original aufweist, dies aber mit weniger Gattern oder auch einfacheren Gattern erreicht.

7.2.2 Volladdierer

Der Volladdierer ist ebenfalls ein Beispiel eines Schaltnetzes. Er erweitert den Halbaddierer insofern, als er die Summe zweier BOOLEschen Variablen a und b unter der Berücksichtigung eines Übertrags aus der vorgehenden Stelle (**Carry In** oder CI) berechnet. Das Resultat wird in zwei Ausgangsvariablen **Summe** und **Carry Out** dargestellt. Die Wertetabelle ist in der Tabelle [7.2] dargestellt.

a	b	Carry In	Summe	Carry Out
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

Tabelle 7.2: Wahrheitstabelle für einen 1-Bit Volladdierer.

Der Output-Kanal **Summe** ist 1, wenn eine ungerade Anzahl der Eingangskanäle a , b oder **Carry In** auf 1 gesetzt sind. Der Ausgang **Carry Out** ist 1, wenn

- entweder sind a und b beide 1,
- a oder b ist 1 und **Carry In** ist 1.

Die Schaltfunktion des Volladdierers läßt sich also formal folgendermaßen formulieren:

$$\mathbb{F} : \mathbb{B}^3 \longrightarrow \mathbb{B}^2,$$

$$(a, b, CI) \longmapsto \mathbb{F}(a, b, CI) = (\text{Summe}(a, b, CI), \text{Carry Out}(a, b, CI)).$$

Wir zeigen nun explizit, dass gilt:

$$\text{Summe}(a, b, CI) = (a \oplus b) \oplus CI, \quad (7.2a)$$

$$\text{Carry Out}(a, b, CI) = (a \wedge b) \vee (a \wedge CI) \vee (b \wedge CI) \quad (7.2b)$$

$$= [a \wedge (b \oplus CI)] \oplus [b \wedge CI]. \quad (7.2c)$$

Beweis:

1. Beweis von Gl. (7.2a).

Aus der Tabelle [7.2] bilden wir für das Summenbit die disjunktive Normalform DNF und erhalten:

$$f_1(a, b, CI) = (\bar{a} \wedge \bar{b} \wedge CI) \vee (\bar{a} \wedge b \wedge \overline{CI}) \vee (a \wedge \bar{b} \wedge \overline{CI}) \vee (a \wedge b \wedge CI).$$

Die DNF für das Summenbit können wir umformen in

$$f_1(a, b, CI) = (a \oplus b) \oplus CI.$$

Wir zeigen dies auf zwei Arten:

(a) Wertetabelle

a	b	CI	$a \oplus b$	$(a \oplus b) \oplus CI$
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Vergleicht man die letzte Spalte mit der vorletzten Spalte der Tabelle [7.2], dann erkennt man die Übereinstimmung dieser Spalten wodurch die Beziehung (7.2a) in Form der Wertetabelle gezeigt ist.

(b) Algebraisch

Wir beginnen mit

$$f = (a \oplus b) \oplus CI.$$

Da per Definition die XOR Funktion durch

$$a \oplus b = (\bar{a} \wedge b) \vee (a \wedge \bar{b})$$

gegeben ist, folgt:

$$\begin{aligned} f &= (a \oplus b) \oplus CI \\ &= [(\bar{a} \wedge b) \vee (a \wedge \bar{b})] \oplus CI \\ &= \left([(\bar{a} \wedge b) \vee (a \wedge \bar{b})] \wedge CI \right) \vee \left([(\bar{a} \wedge b) \vee (a \wedge \bar{b})] \wedge \overline{CI} \right) \end{aligned}$$

Im nächsten Schritt wendet man zweimal die Regeln von DEMORGAN auf den ersten Teil sowie das Distributivgesetz auf den zweiten Term an und erhält:

$$f = [(a \vee \bar{b}) \wedge (\bar{a} \vee b)] \wedge CI \vee [(\bar{a} \wedge b \wedge \overline{CI}) \vee (a \wedge \bar{b} \wedge \overline{CI})].$$

Wir wenden nun erneut das Assoziativ- und das Distributivgesetz im ersten Term an und erhalten

$$\begin{aligned} f &= ((a \vee \bar{b}) \wedge [(\bar{a} \wedge CI) \vee (b \wedge CI)]) \\ &\quad \vee [(\bar{a} \wedge b \wedge \overline{CI}) \vee (a \wedge \bar{b} \wedge \overline{CI})]. \end{aligned}$$

Das Distributivgesetz wird im ersten Term nochmals angewendet

$$f = [(a \vee \bar{b}) \wedge (\bar{a} \wedge CI)] \vee [(a \vee \bar{b}) \wedge (b \wedge CI)] \\ \vee [(\bar{a} \wedge b \wedge \overline{CI}) \vee (a \wedge \bar{b} \wedge \overline{CI})].$$

Schließlich wendet man das Distributivgesetz ein drittes Mal an:

$$f = (a \wedge \bar{a} \wedge CI) \vee (\bar{b} \wedge \bar{a} \wedge CI) \vee (a \wedge b \wedge CI) \vee (\bar{b} \wedge b \wedge CI) \\ \vee (\bar{a} \wedge b \wedge \overline{CI}) \vee (a \wedge \bar{b} \wedge \overline{CI}).$$

Da

$$a \wedge \bar{a} = 0 \quad \text{und} \quad 0 \vee c = c,$$

erhalten wir

$$f = (\bar{a} \wedge \bar{b} \wedge CI) \vee (a \wedge b \wedge CI) \vee (\bar{a} \wedge b \wedge \overline{CI}) \vee (a \wedge \bar{b} \wedge \overline{CI}).$$

Damit haben wir unter Anwendung algebraischer Umformungsregeln der BOOLEschen Algebra die Gleichung (7.2a) gezeigt.

2. Die DNF für das Übertragbit Carry Out ist:

$$f_2(a, b, CI) = (\bar{a} \wedge b \wedge CI) \vee (a \wedge \bar{b} \wedge CI) \vee (a \wedge b \wedge \overline{CI}) \vee (a \wedge b \wedge CI). \quad (7.3)$$

Wir zeigen zunächst Gl. (7.2b) wieder auf zwei Arten:

(a) Wertetabelle

Bildet man die Wertetabelle des BOOLEschen Ausdrucks $f_2(a, b, CI)$, dann erhält man:

a	b	CI	$a \wedge b$	$a \wedge CI$	$b \wedge CI$	$(a \wedge b) \vee (a \wedge CI) \vee (b \wedge CI)$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	0	0
1	0	1	0	1	0	1
1	1	0	1	0	0	1
1	1	1	1	1	1	1

Die letzte Spalte stimmt mit der letzten Spalte der Tabelle [7.2] überein. Damit ist die Gl. (7.2b) mit Hilfe von Wertetabellen verifiziert.

(b) Algebraisch.

Die DNF (7.3) können wir folgendermaßen als KV-Diagramm darstellen:

	$a \wedge b$	$a \wedge \bar{b}$	$\bar{a} \wedge \bar{b}$	$\bar{a} \wedge b$
CI				
\overline{CI}				

Um die Minimalform zu erhalten, können wir also gemäß diesem KV-Diagramm drei Kontraktionen durchführen. Insbesondere zeigt dieses KV-Diagramm, dass der Konjunktionsterm $a \wedge b \wedge CI$ in drei Kontraktionen auftritt. Daher können wir das Idempotenzgesetz anwenden. Es folgt:

$$\begin{aligned}
 f_{2,DNF}(a,b,CI) &= (\bar{a} \wedge b \wedge CI) \vee (a \wedge \bar{b} \wedge CI) \\
 &\quad \vee (a \wedge b \wedge \overline{CI}) \vee (a \wedge b \wedge CI) \\
 &\stackrel{1}{=} (\bar{a} \wedge b \wedge CI) \vee (a \wedge \bar{b} \wedge CI) \vee (a \wedge b \wedge \overline{CI}) \\
 &\quad \vee (a \wedge b \wedge CI) \vee (a \wedge b \wedge CI) \vee (a \wedge b \wedge CI) \\
 &\stackrel{2}{=} [(\bar{a} \wedge b \wedge CI) \vee (a \wedge b \wedge CI)] \\
 &\quad \vee [(a \wedge \bar{b} \wedge CI) \vee (a \wedge b \wedge CI)] \\
 &\quad \vee [(a \wedge b \wedge \overline{CI}) \vee (a \wedge b \wedge CI)] \\
 &\stackrel{3}{=} [(\bar{a} \vee a) \wedge (b \wedge CI)] \\
 &\quad \vee [(\bar{b} \vee b) \wedge (a \wedge CI)] \\
 &\quad \vee [(\overline{CI} \vee CI) \wedge (a \wedge b)] \\
 &\stackrel{4}{=} (a \wedge b) \vee (a \wedge CI) \vee (b \wedge CI).
 \end{aligned}$$

Hierbei verwendet man:

Schritt 1: zwei Mal Idempotenzgesetz,

Schritt 2: Assoziativgesetz und Kommutativgesetz,

Schritt 3: Distributivgesetz,

Schritt 4: Komplementgesetze und neutrales Element.

Damit ist die Beziehung (7.2b) gezeigt.

3. Schließlich ist noch Gl. (7.2c) zu zeigen. Eine Wertetabelle ergibt:

a	b	CI	$b \oplus CI$	$a \wedge (b \oplus CI)$	$b \wedge CI$	$[a \wedge (b \oplus CI)] \oplus (b \wedge CI)$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	1	1	1	0	1
1	1	0	1	1	0	1
1	1	1	0	0	1	1

Ein Vergleich der letzten Spalte dieser Wertetabelle mit der letzten Spalte der Tabelle [7.2] zeigt eine Übereinstimmung dieser Spalten, daher ist

$$\text{Carry Out}(a, b, CI) = [a \wedge (b \oplus CI)] \oplus (b \wedge CI).$$

und somit ist Gl. (7.2c) verifiziert.

In der Abbildung 7.10 ist der Schaltungsaufbau eines Volladdierers dargestellt, hierbei sind AND, OR und XOR Gatter verwendet. Dieser Schaltungsaufbau realisiert die BOOLEsche Funktionen (7.2a) und (7.2b).

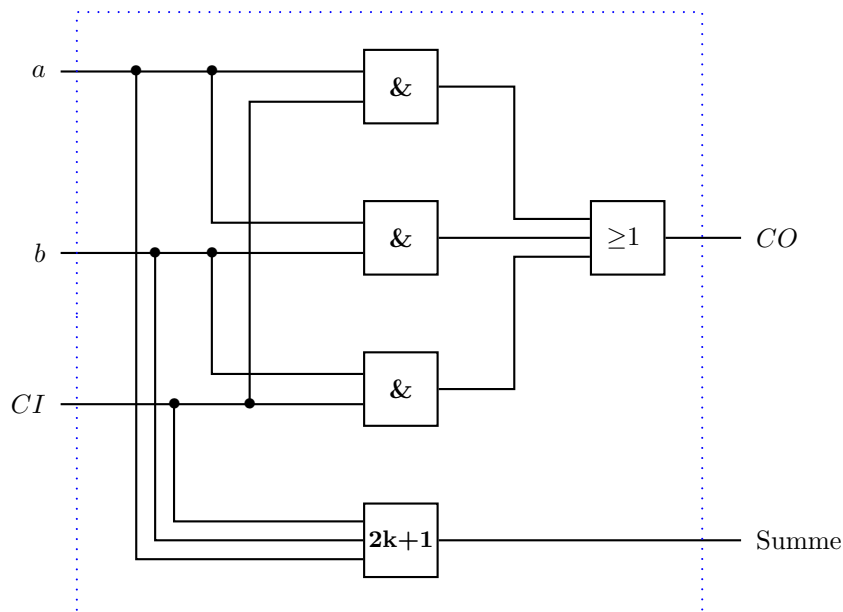


Abbildung 7.10: Schaltung für einen Volladdierer.

Alternativ kann der Volladdierer auch als Hintereinanderschaltung zweier Halbaddierer und eines XOR-Gatters konstruiert werden. Diese Schaltung realisiert

die BOOLEschen Funktionen (7.2a) und (7.2c). Diese Schaltung ist in der Abbildung [7.11] skizziert.

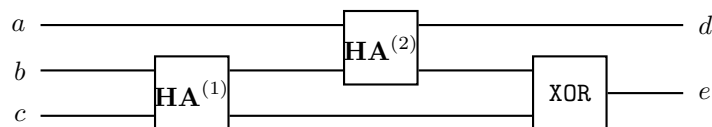


Abbildung 7.11: Schaltbild des Volladdierers aus zwei Halbaddierern und einem XOR-Gatter.

Aus den Gl. (7.1a) und (7.1b) erhalten wir die folgende Schalttable:

a	b	c	$\mathbf{HA}_1^{(1)}(b, c)$	$\mathbf{HA}_2^{(1)}(b, c)$	$\mathbf{HA}_1^{(2)}(\mathbf{HA}_1^{(1)}, a)$	$\mathbf{HA}_2^{(2)}(\mathbf{HA}_1^{(1)}, a)$	$\mathbf{HA}_1^{(1)} \oplus \mathbf{HA}_1^{(2)}$
0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	0	1
1	0	0	0	0	1	0	0
1	0	1	1	0	0	1	1
1	1	0	1	0	0	1	1
1	1	1	0	1	1	0	1

Die Spalten 6 und 8 stimmen mit den Spalten 4 und 5 der Tabelle [7.2] überein.

Es hat sich auch hier eine Kurzschreibweise eingebürgert, die es erlaubt, Schaltnetze in Form von Symbolen darzustellen. Ein Volladdierer ist demnach ein Art 'Black Box' mit drei Eingängen und zwei Ausgängen. Dies wird durch das Symbol in der Abbildung [7.12] repräsentiert

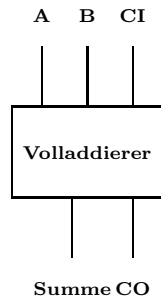


Abbildung 7.12: Symbol für das Schaltungsnetz eines Volladdierers.

7.2.3 Ripple Carry Adder

Es mag beruhigend sein, dass man zwei Bits addieren kann, die Frage, die aufkeimt, ist, wie addiert ein Computer beispielsweise zwei 16 Bit-Zahlen? Genau diesem Zweck dient der sogenannte **Ripple Carry Adder** oder zu deutsch in etwa *Addierer mit durchlaufendem Übertrag*. Dieser Addierer berechnet die Summe

$$(CO_{n-1} \ S_{n-1} \ S_{n-2} \ \dots \ S_1 \ S_0)$$

zweier n -stelliger Dualzahlen

$$\begin{aligned} a &= (a_{n-1} \ a_{n-2} \ \dots \ a_1 \ A_0) \\ b &= (b_{n-1} \ b_{n-2} \ \dots \ b_1 \ B_0) \end{aligned}$$

Dieser Addierer ist im Prinzip nichts anderes als mehrere miteinander verschaltete Volladdierer. Das Carry-Out Signal eines niedrigeren Bits wird als Carry-In des nächst höheren eingespeist. Das sieht dann etwa aus wie in der Abbildung [7.13].

Die Summe von a_0 und b_0 wird von einem Halbaddierer berechnet, da an der niedrigsten Bitstelle ja noch kein Übertrag auftaucht. Das Ergebnis S_0 gibt die niederwertigste Stelle der Gesamtsumme von a und b an. Der Übertrag CI/O_0 wird zusammen mit a_1 und b_1 als Eingangswert des ersten Volladdierers verwendet. Das geht so weiter bis zum höchsten Bit.

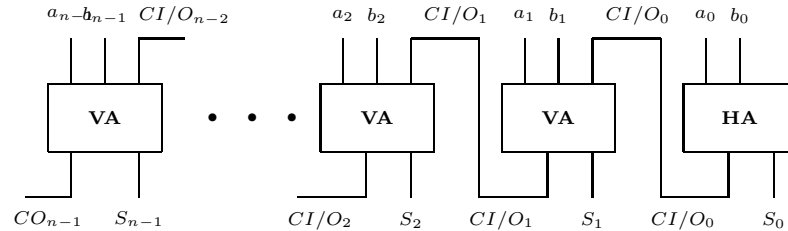


Abbildung 7.13: Schaltung für einen Ripple Carry Adder.

7.2.4 Lichtschalter

Die Beleuchtungen in Hausfluren oder Treppenhäusern werden üblicherweise durch zwei oder mehr Schalter gesteuert. Der Übersichtlichkeit halber nehmen wir an, dass es drei Schalter gibt, die die Beleuchtung steuern und das Licht ist an. Wenn der Umschalter eines Schalters gekippt wird, geht das Licht aus. Andererseits, wenn das Licht aus ist und irgendeiner der Schalter wird betätigt, dann geht das Licht an. Wir konstruieren einen Schaltkreis für dieses Szenario.

Wir bezeichnen mit a, b und c die drei Schalter und $f(a, b, c)$ ist der Ausgang der gesuchten digitalen Schaltung. Wir bezeichnen mit $f(a, b, c) = 1$ den Zustand, dass das Licht angeschaltet ist. Wir nehmen weiterhin an, dass das Licht ausgeschaltet sind, wenn sämtliche Schalter auf 'aus' stehen, *i.e.* wir setzen $f(0, 0, 0) = 0$. Dies ist reine Konvention, jede andere Startbelegung der Schaltvariablen führt auf das gleiche Resultat.

Wenn ein Schalter gekippt wird, geht das Licht an, *i.e.*

$$f(0, 0, 1) = f(0, 1, 0) = f(1, 0, 0) = 1.$$

Nun wird ein weiterer Schalter gekippt, dann wird das Licht ausgeschaltet, damit

$$f(0, 1, 1) = f(1, 1, 0) = f(1, 0, 1) = 0.$$

Wird der dritte Schalter betätigt, wird das Licht wieder angeschaltet, *i.e.*

$$f(1, 1, 1) = 1.$$

Dieses Szenario können wir in Form einer Schalttablelle arrangieren und erhalten

a	b	c	$f(a, b, c)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Hieraus lesen wir sofort die DNF ab und erhalten

$$f_{DNF}(a, b, c) = (\bar{a} \wedge \bar{b} \wedge c) \vee (\bar{a} \wedge b \wedge \bar{c}) \vee (a \wedge \bar{b} \wedge \bar{c}) \vee (a \wedge b \wedge c). \quad (7.4)$$

Die Abbildung [7.14] zeigt diese disjunktive Normalform als Schaltung.

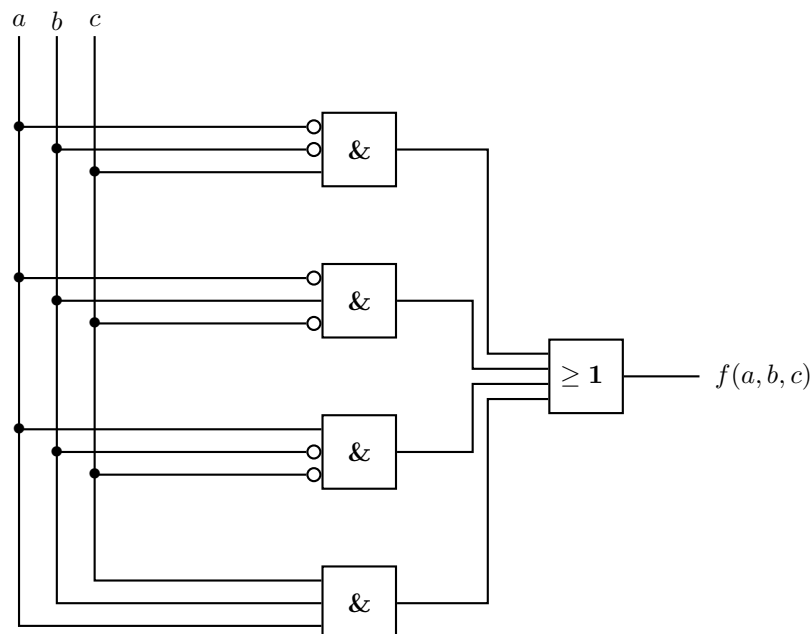


Abbildung 7.14: Schaltkreis zur Steuerung der Beleuchtung mit drei Schaltern.

Erstellt man das KARNAUGH–VEITCH Diagramm zu der disjunktiven Normalform (7.4), dann ergibt sich das in der Abbildung [7.15] gezeigte Diagramm. Da keine der belegten Zellen benachbart ist, hat dies zur Folge, dass die Funktion (7.4) nicht weiter vereinfacht werden kann. Dies zeigt die Mächtigkeit der

KV-Diagramme, denn aus dem Diagramm [7.15] kann sofort abgelesen werden, dass eine Minimierung der disjunktiven Normalform (7.4) keine weitere Vereinfachung bringt. Will man dies algebraisch zeigen, muss ausgehend von der DNF (7.4) durch Anwendung der Regeln der BOOLEschen Algebra gezeigt werden, dass keine Regel auf eine Vereinfachung führt. Dies ist schlichtweg so gut wie nicht möglich.

	$a \wedge b$	$a \wedge \bar{b}$	$\bar{a} \wedge \bar{b}$	$\bar{a} \wedge b$
c	•		•	
\bar{c}		•		•

Abbildung 7.15: KV-Diagramm der BOOLEschen Funktion (7.4).

7.2.5 Das Sieben-Segment-Display

Eine sehr lehrreiche Anwendung der Schaltalgebra ist das sogenannte **Sieben-Segmente-Display**,³ das in sehr vielen Geräten — beispielsweise digitale Uhren — eingesetzt wird. Das typische Sieben-Segment Display ist in Abbildung [7.16] dargestellt. Jede Ziffer von 0 bis 9 wird dadurch gebildet, dass verschiedene Abschnitte des Displays schwarz (sie werden angeschaltet) oder transparent (abgeschaltet) dargestellt werden. Wird beispielsweise die Ziffer 8 dargestellt, sind alle Balken angeschaltet.

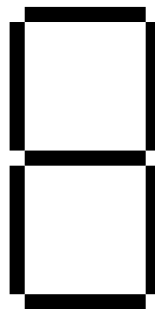


Abbildung 7.16: Das Sieben-Segment-Display.

Wenn die digitale Uhr 21:16 anzeigt, befinden sich irgendwo in einem internen Speicher der Uhr die vier binären Zahlen 0010, 0001, 0001, 0110 — die binären

³Siehe zum Beispiel GREGG, [41], oder KOSHY, [68, Chap. 12].

Darstellungen von 2, 1, 1 und 6. Im Inneren der Digitaluhr gibt es auch Schaltkreise, die die binären Zahlen als Input verwenden um daraus den richtigen An-/Aus-Output zu produzieren, der die Balken des Sieben-Segment Displays an- oder ausschaltet. Durch das Senden einer '1' wird ein Balken angeschaltet, das Senden einer '0' schaltet den Balken aus.

Wir betrachten die rechte Ziffer unseres Beispiels, die 'sechs' der Anzeige 21:16. Mit dieser Ziffer auf der Uhr ist ein Schaltkreis verbunden, der die Bitfolge 0110 ($=6_{10}$) von dem internen Timer erhält. Dieser Display-Schaltkreis hat einen Ausgang zu jedem der sieben Balken des Display an der rechten Anzeige. Jeder dieser sieben Ausgänge realisiert eine andere BOOLEsche Funktion bei dem gleichen vier Bit Input.

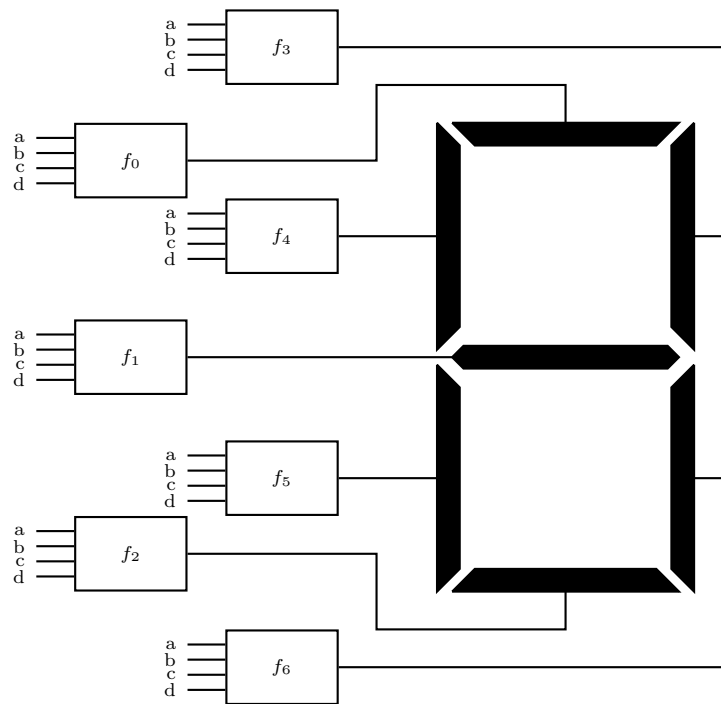


Abbildung 7.17: Das Sieben-Segment Display mit Kontrollfunktionen.

Da wir Minterme benutzen können, um jede BOOLEsche Funktion zu realisieren, für die Schalttabellen aufgestellt werden können, müssen wir einfach für jede der sieben Funktionen eine Wahrheitstabelle aufstellen, die deren Funktion beschreibt. Der erste Schritt dazu besteht in der Bestimmung der Anzahl der

benötigten Inputbits. Da die Uhr Ziffern zur Basis 10 darstellen soll, muss jedes Display auf der Uhr in der Lage sein, die Ziffern 0 bis 9 darstellen zu können. Damit sind

$$m = \lceil \ln_2 10 \rceil = 4$$

Bits notwendig, um die Ziffern 0 bis 9 darzustellen.

Nun haben wir also vier Bits vom Zeitgeber und wir wollen sieben Funktionen konstruieren, die den gleichen Input haben. Jede dieser Funktionen schaltet einen der sieben einzelnen Balken des Displays entweder an oder aus. Wir bezeichnen die vier Inputbits mit a, b, c und d sowie die sieben Funktionen mit f_0, f_1, \dots, f_6 . Dies ist in Abbildung [7.17] dargestellt.

Wir müssen nun für die sieben Funktionen f_0, f_1, \dots, f_6 die Wahrheitstabellen aufbauen und hieraus die Minterme erstellen, die diese Funktionen durch AND, OR und NOT Operatoren darstellen. Wir betrachten dazu explizit die Konstruktion der Funktion f_5 , die den Balken unten links kontrolliert. Diese Funktion hat vier Bits als Input, die die Dezimalziffern 0 (0000) bis 9 (1001) darstellen. Alle anderen Bitkombinationen, die die Zahlen 10 bis 15 binär darstellen sind irrelevant, da vom Timer keine Zahl größer als 1001 ausgegeben wird.

Um herauszufinden, wann der Output der Funktion 1 sein soll, schreibt man sich die Wahrheitstafel mit den Dezimalzahlen von 0 bis 9 in der linken Spalte. Bei der Anzeige der Ziffer **3** beispielsweise, darf kein Balken links unten aufleuchten. Daher muss die Funktion f_5 , die diesen Balken kontrolliert, eine 0 produzieren, wenn der Input die Bitfolge 0011 ist. Auf die gleiche Weise geht man alle Ziffern durch, was letztendlich auf die Tabelle [7.3] führt.

Der Output für die Folgen 1010 bis 1111 wird mit + bezeichnet; dies sind *don't care* Bits.

Die Funktion f_5 ist nun komplett spezifiziert und es ist einfach die Minterme zu konstruieren. Aus der Tabelle [7.3] liest man ab, dass es vier 1en in den Zeilen 0, 2, 6 und 8 gibt. Damit ergibt sich die disjunktive Normalform zu:

$$f_5(a, b, c, d) = M_0 \vee M_2 \vee M_6 \vee M_8, \quad (7.5a)$$

mit

$$M_0 = \bar{a} \wedge \bar{b} \wedge \bar{c} \wedge \bar{d}, \quad (7.5b)$$

$$M_2 = \bar{a} \wedge \bar{b} \wedge c \wedge \bar{d}, \quad (7.5c)$$

$$M_6 = \bar{a} \wedge b \wedge c \wedge \bar{d}, \quad (7.5d)$$

$$M_8 = a \wedge \bar{b} \wedge \bar{c} \wedge \bar{d}. \quad (7.5e)$$

Die Abbildung [7.19] zeigt den Schaltkreis, der Funktion $f_5(a, b, c, d)$ in disjunktiver Normalform realisiert.

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	$f_5(a, b, c, d)$
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
–	1	0	1	0	+
–	1	1	0	0	+
–	1	0	1	1	+
–	1	1	1	0	+
–	1	1	1	1	+

Tabelle 7.3: Schalttabelle der Funktion $f_5(a, b, c, d)$ zur Steuerung des links-unten Balkens.

Wir minimieren nun die Schaltfunktion f_5 . Das KV-Diagramm der Funktion f_5 ist in der Abbildung [7.18] dargestellt. Hierbei sind die *don't care* Bedingungen berücksichtigt, die bei der Minimierung relevant sind.

Die Minimalform der Funktion f_5 ist gemäß dem KV-Diagramm [7.18]:

$$f_{5,min}(a, b, c, d) = (\overline{b} \wedge \overline{d}) \vee (\overline{a} \wedge c \wedge \overline{d}).$$






	$a \wedge b$	$a \wedge \bar{b}$	$\bar{a} \wedge \bar{b}$	$\bar{a} \wedge b$
$c \wedge d$				
$c \wedge \bar{d}$				
$\bar{c} \wedge \bar{d}$				
$\bar{c} \wedge d$				

Abbildung 7.18: KV-Diagramm der Funktion $f_5(a, b, c, d)$.

Algebraisch leitet man diese Minimalform folgendermaßen ab:

$$\begin{aligned}
 f_5(a, b, c, d) &= (\bar{a} \wedge \bar{b} \wedge \bar{c} \wedge \bar{d}) \vee (\bar{a} \wedge \bar{b} \wedge c \wedge \bar{d}) \\
 &\quad \vee (\bar{a} \wedge b \wedge c \wedge \bar{d}) \vee (a \wedge \bar{b} \wedge \bar{c} \wedge \bar{d}) \\
 &\stackrel{(1)}{=} [(a \wedge \bar{b} \wedge c \wedge \bar{d}) \vee (\bar{a} \wedge \bar{b} \wedge c \wedge \bar{d}) \\
 &\quad \vee (a \wedge \bar{b} \wedge \bar{c} \wedge \bar{d}) \vee (\bar{a} \wedge \bar{b} \wedge \bar{c} \wedge \bar{d})] \\
 &\quad \vee [(\bar{a} \wedge \bar{b} \wedge c \wedge \bar{d}) \vee (\bar{a} \wedge b \wedge c \wedge \bar{d})] \\
 &\stackrel{(2)}{=} [((a \vee \bar{a}) \wedge \bar{b} \wedge c \wedge \bar{d}) \vee ((a \vee \bar{a}) \wedge \bar{b} \wedge \bar{c} \wedge \bar{d})] \\
 &\quad \vee [(b \vee \bar{b}) \wedge \bar{a} \wedge c \wedge \bar{d}] \\
 &\stackrel{(3)}{=} (\bar{b} \wedge c \wedge \bar{d}) \vee (\bar{b} \wedge \bar{c} \wedge \bar{d}) \vee (\bar{a} \wedge c \wedge \bar{d}) \\
 &\stackrel{(4)}{=} ((c \vee \bar{c}) \wedge \bar{b} \wedge \bar{d}) \vee (\bar{a} \wedge c \wedge \bar{d}) \\
 &\stackrel{(5)}{=} (\bar{b} \wedge \bar{d}) \vee (\bar{a} \wedge c \wedge \bar{d}) \\
 &= f_{5,min}(a, b, c, d).
 \end{aligned}$$

Bei den Zwischenschritten werden folgende Gesetze verwendet:

- (1) Idempotenzgesetz,
- (2) Distributivgesetz,
- (3) Komplementgesetz,
- (4) Distributivgesetz,
- (5) Komplementgesetz.

Über die Schalttafel [7.4] läßt sich verifizieren, dass sowohl $f_5(a, b, c, d)$ als auch $f_{5,min}(a, b, c, d)$ die gleichen Funktionswerte haben für die relevanten Be-

gungen. Vergleicht man die letzte Spalte der Tabelle [7.4] mit der zehn ersten Zeilen der letzten Spalte der Tabelle [7.3], dann erkennt man eine Übereinstimmung der Funktionswerte..

a	b	c	d	\bar{a}	\bar{b}	\bar{d}	$\bar{b} \wedge \bar{d}$	$\bar{a} \wedge c \wedge \bar{d}$	$f_{5,min}(a, b, c, d)$
0	0	0	0	1	1	1	1	0	1
0	0	0	1	1	1	0	0	0	0
0	0	1	0	1	1	1	1	1	1
0	0	1	1	1	1	0	0	0	0
0	1	0	0	1	0	1	0	0	0
0	1	0	1	1	0	0	0	0	0
0	1	1	0	1	0	1	0	1	1
0	1	1	1	1	0	0	0	0	0
1	0	0	0	0	1	1	1	0	1
1	0	0	1	0	1	0	0	0	0

Tabelle 7.4: Schalttable der minimierten Funktion f_5 .

Die Realisierung der Minimalform $f_{5,min}(a, b, c, d)$ in Form eines Schaltkreises ist in der Abbildung [7.20] gezeigt.

Auf diese Art und Weise lassen sich alle Funktionen f_0 bis f_6 bestimmen. Die Wahrheitstafel dieser BOOLEschen Funktionen ist in der Tabelle [7.5] aufgeführt.

	a	b	c	d	f_0	f_1	f_2	f_3	f_4	f_5	f_6
0	0	0	0	0	1	0	1	1	1	1	1
1	0	0	0	1	0	0	0	1	0	0	1
2	0	0	1	0	1	1	1	1	0	1	0
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	0	1	1	0	1
5	0	1	0	1	1	1	1	0	1	0	0
6	0	1	1	0	0	1	1	0	1	1	1
7	0	1	1	1	1	0	0	1	0	0	1
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	0	1	1	0	1

Tabelle 7.5: Wahrheitstafel der Funktionen f_0 bis f_6 zur Steuerung der Balken eine Sieben-Segment Displays.

Die vollständigen disjunktiven Normalformen zur Steuerung eines Sieben-Segmente

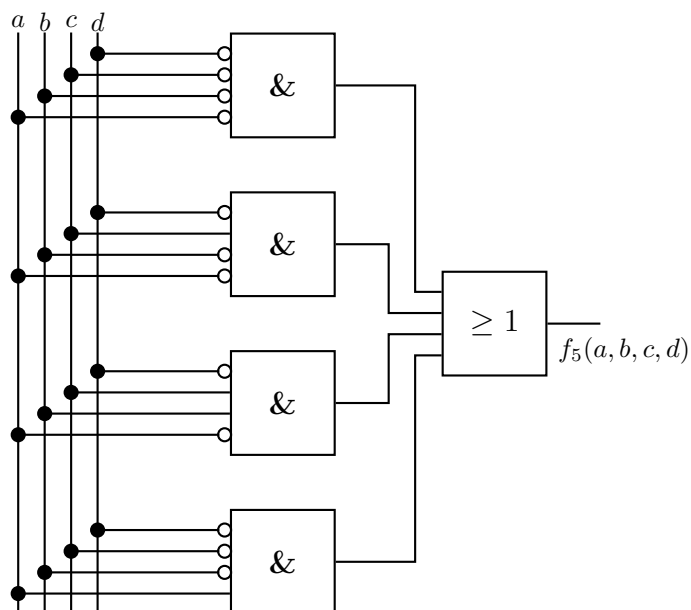


Abbildung 7.19: Schaltkreis zur Realisierung der disjunktiven Normalform Gln. (7.5a) – (7.5e).

Displays sind dann:

$$\begin{aligned}
 f_0(a, b, c, d) &= M_0 \vee M_2 \vee M_3 \vee M_5 \vee M_7 \vee M_8 \vee M_9 \\
 f_1(a, b, c, d) &= M_2 \vee M_3 \vee M_4 \vee M_5 \vee M_6 \vee M_8 \vee M_9 \\
 f_2(a, b, c, d) &= M_0 \vee M_2 \vee M_3 \vee M_5 \vee M_6 \vee M_8 \\
 f_3(a, b, c, d) &= M_0 \vee M_1 \vee M_2 \vee M_3 \vee M_4 \vee M_7 \vee M_8 \vee M_9 \\
 f_4(a, b, c, d) &= M_0 \vee M_4 \vee M_5 \vee M_6 \vee M_8 \vee M_9 \\
 f_5(a, b, c, d) &= M_0 \vee M_2 \vee M_6 \vee M_8 \\
 f_6(a, b, c, d) &= M_0 \vee M_1 \vee M_3 \vee M_4 \vee M_6 \vee M_7 \vee M_8 \vee M_9
 \end{aligned}$$

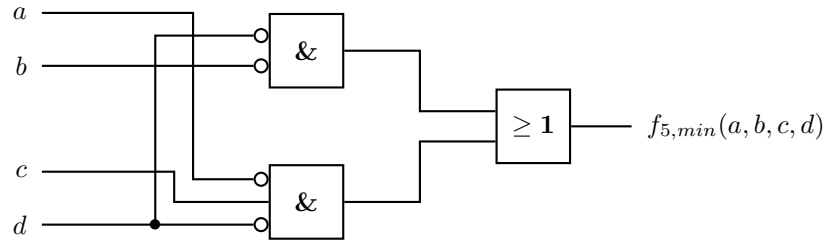


Abbildung 7.20: Realisierung der Minimalfunktion $f_{5,min}(a, b, c, d)$ als Schaltkreis.

mit den Disjunktionstermen

$$M_0 = \bar{a} \wedge \bar{b} \wedge \bar{c} \wedge \bar{d}$$

$$M_1 = \bar{a} \wedge \bar{b} \wedge \bar{c} \wedge d$$

$$M_2 = \bar{a} \wedge \bar{b} \wedge c \wedge \bar{d}$$

$$M_3 = \bar{a} \wedge \bar{b} \wedge c \wedge d$$

$$M_4 = \bar{a} \wedge b \wedge \bar{c} \wedge \bar{d}$$

$$M_5 = \bar{a} \wedge b \wedge \bar{c} \wedge d$$

$$M_6 = \bar{a} \wedge b \wedge c \wedge \bar{d}$$

$$M_7 = \bar{a} \wedge b \wedge c \wedge d$$

$$M_8 = a \wedge \bar{b} \wedge \bar{c} \wedge \bar{d}$$

$$M_9 = a \wedge \bar{b} \wedge \bar{c} \wedge d$$

7.2.6 Codierer

Die Schaltung eines **Codierers** hat n Eingänge, die mit $e_i, i = 1, \dots, n$ bezeichnet werden. Außerdem hat der Codierer m Ausgänge, wobei gilt

$$m = \lceil \ln_2 n \rceil$$

wobei

$$\lceil x \rceil$$

für die kleinste ganze Zahl größer oder gleich x steht. Die Ausgänge haben die Bezeichnung $a_j, j = 1, \dots, m$. Unter der Voraussetzung, dass immer nur einer der n Eingänge aktiv — *i.e.* logisch 1 — sein kann, hat ein Codierer die Aufgabe (d.h. Funktion), die Bitfolge, die an den Eingängen e_i anliegt, in eine Binärzahl an den Ausgängen a_j umzuwandeln.

Die folgende Wahrheitstabelle zeigt die Werte für einen Codierer mit $n = 8$ Eingängen und $m = \ln_2 8 = 3$ Ausgängen

e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	a_2	a_2	a_2
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

7.2.7 Decoder

Ein Decoder ist ein Schaltnetz⁴, das aus AND, OR und NOT Gattern konstruiert werden kann. Für eine bestimmte Zahl n — das sind die Eingänge — hat ein Decoder 2^n Ausgänge. Hat also eine bestimmte Decoderschaltung drei Eingänge, dann hat diese Schaltung $2^3 = 8$ Ausgänge. Jeder Eingang spezifiziert genau ein Bit einer Zahl in der Basis 2 und der Decoder sendet einfach eine 1 an den Ausgang, der der n -Bit Input Zahl als Ganzes entspricht. Alle anderen Ausgänge sind 0.

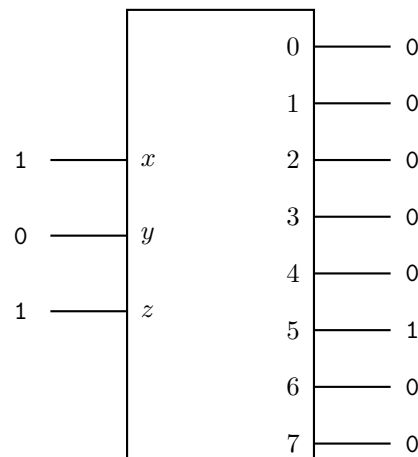


Abbildung 7.21: Ein 3 - 8 Decodierer mit dem Input 1 0 1.

Wir bezeichnen die Inputs eines 3–8 Decoders mit x, y und z . Wir nehmen

⁴Siehe die Monographi von JOHN R. GREGG, [41], Chapter 5.1.1.

weiterhin an, dass x das höchstwertigste Bit der binären Input-Zahl ist und z stellt das niedrigwertigste Bit dieser Zahl dar. Erhält dieser Decoder den Input $(x, y, z) = 1\ 0\ 1$, sendet er eine 1 an den Ausgangskanal 5, da $101_2 = 5_{10}$ und alle anderen Ausgänge werden mit 0 belegt. Zu einem Zeitpunkt ist also immer nur eine einzige Ausgangsleitung auf 1 gesetzt.

Die Schalttable eines 3–8 Decoders ergibt sich damit wie folgt:

x	y	z	0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

7.2.8 Multiplexer

Ein Rechner ist üblicherweise mit einer Vielzahl an Peripheriegeräten ausgestattet, beispielsweise Festplatte, DVD-Laufwerk, Lautsprecher, usw. Um zwischen diesen Geräten Daten austauschen zu können, verfügt ein Rechner über Datenleitungen, die man **Bussystem** nennt,⁵ genauer nennt man die Leitungen, über die Daten transferiert werden, den **Datenbus**.⁶ Sämtlicher Geräte eines Rechners verwenden ein und denselben Bus.

Multiplexer — kurz MUX — sind digitale Schaltungen, die eine ausgewählte Datenquelle 'durchschalten' und die restlichen Datenquellen für eine Übertragung sperren können.

Ein Multiplexer hat einen Ausgangskanal und zwei verschiedene Arten von Inputkanälen, **Datenkanäle** oder Datenbits und **Steuerkanäle**, diese nennt man auch Steuerbits oder Kontrollbits. Für eine gegebene Zahl n — diese hängt von der gewünschten Größe des Multiplexers ab — gibt es n Steuerbits und 2^n Dateninputbits. Die Steuerbits legen fest, welcher der 2^n Datenkanäle auf den Ausgang durchgeschaltet wird.

Beispiel 7.44:

Um das Prinzip der Multiplexerschaltung zu verstehen, betrachten wir den einfachsten Fall, *i.e.* eine Steuerleitung, die zwei Datenleitungen kontrolliert. Wir nennen die Steuerleitung c_0 und die beiden Datenleitungen d_0 bzw. d_1 . Die Schaltung definieren wir durch die Schalttable [7.6].

⁵In heutigen Personalcomputern ist das Standardbussystem der PCI-Bus. Ein weiteres Bussystem, mit dem externe Geräte wie Drucker, Tastatur oder Maus die Daten an die Zentraleinheit übertragen, ist der Universal Serial Bus, kurz USB.

⁶Ein komplettes Bussystem verfügt weiter noch über einen Steuerbus und einen Adressbus.

c_0	d_0	d_1	$f(c_0, d_0, d_1)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Tabelle 7.6: Schalttable eines Multiplexers mit einer Steuerleitung und zwei Datenleitungen.

Das grundlegende Prinzip eines Multiplexers kann aus dieser Tabelle abgelesen werden. Wenn das Steuerbit den Wert 0 hat, dann erkennt man, dass die Werte der Funktion $f(c_0, d_0, d_1)$ identisch mit den Werten der Datenquelle d_0 ist. Ist die Steuerleitung auf 1 gesetzt, dann nimmt f die Werte der zweiten Datenquelle d_1 an.

Die disjunktive Normalform der Funktion $f(c_0, d_0, d_1)$ ist direkt aus der Tabelle [7.6] ablesbar. Wir erhalten:

$$f_{DNF}(c_0, d_0, d_1) = (\overline{c_0} \wedge d_0 \wedge \overline{d_1}) \vee (\overline{c_0} \wedge d_0 \wedge d_1) \vee (c_0 \wedge \overline{d_0} \wedge d_1) \vee (c_0 \wedge d_0 \wedge d_1). \quad (7.6)$$

Aus Gl. (7.6) ergibt sich das KARNAUGH-VEITCH Diagramm der Abbildung [7.22].

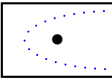
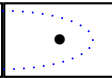
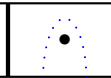
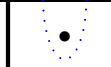
	$c_0 \wedge d_0$	$c_0 \wedge \overline{d_0}$	$\overline{c_0} \wedge \overline{d_0}$	$\overline{c_0} \wedge d_0$
d_1				
$\overline{d_1}$				

Abbildung 7.22: KARNAUGH-VEITCH Diagramm der Funktion (7.6).

Aus diesem KV-Diagramm erhält man die Minimalform der BOOLEschen Funktion (7.6) zu:

$$f_{min} = (c_0 \wedge d_1) \vee (\overline{c_0} \wedge d_0). \quad (7.7)$$

Die Schalttable der Funktion (7.7) ist in der Tabelle [7.7] dargestellt. Die Funktionswerte von f_{min} stimmen mit denen der Tabelle [7.6] überein.

c_0	d_0	d_1	$\overline{c_0}$	$c_0 \wedge d_1$	$\overline{c_0} \wedge d_0$	f_{min}
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	1	0	1	0	1
1	1	0	0	0	0	0
1	1	1	0	1	0	1

Tabelle 7.7: Schalttabelle der Funktion f_{min} .

Hieraus wiederum ergibt sich der Schaltkreis, der den Multiplexer realisiert. Diese ist in der Abbildung [7.23] dargestellt.

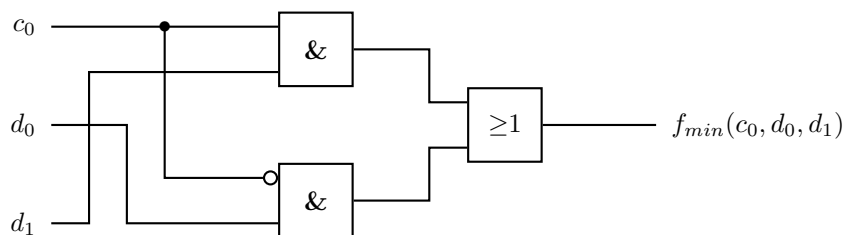


Abbildung 7.23: Schaltung des Multiplexers.

Beispiel 7.45:

Wir betrachten als weiteres Beispiel einen Multiplexer⁷ mit zwei Steuerbits — diese nennen wir c_0 bzw. c_1 — und $2^2 = 4$ Datenkanäle. Diese nennen wir d_0, d_1, d_2 und d_3 (siehe Abb. [7.24]). Es gibt genau ein Dateninputbit für jede der vier möglichen Kombinationen der beiden Kontrollbits. Die spezielle Kombination von Steuerbits legt fest, welcher Datenkanal auf den Ausgang gelegt wird.

⁷Siehe GREGG, [41], Chapter 6.

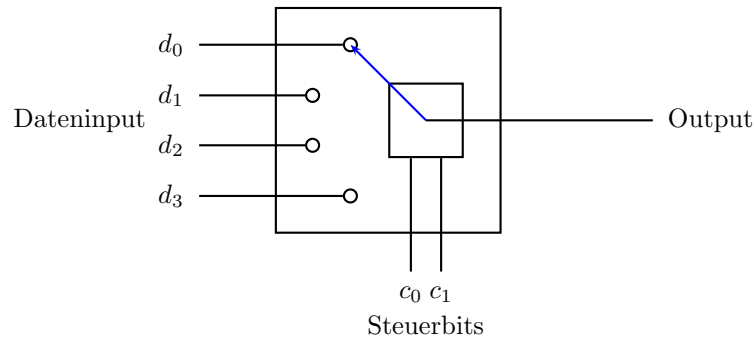


Abbildung 7.24: Prinzipiell kann man sich einen Multiplexer so vorstellen, dass dieser aus einem Umschalter besteht, der von der Steuerbits kontrolliert wird. Diese Umschalter verbindet einen der Dateneingänge mit dem Ausgang.

Der Output des Multiplexers ist exakt gleich dem Dateninput d_0 , falls $(c_0, c_1) = (0, 0)$, d_1 , falls $(c_0, c_1) = (0, 1)$, d_2 , falls $(c_0, c_1) = (1, 0)$ und d_3 , falls die beiden Steuerbits den Wert $(c_0, c_1) = (1, 1)$ haben. Konzeptuell kann man sich einen Multiplexer vorstellen wie einen Schalter, die ist in der Abbildung [7.24] skizziert.

Wie lässt sich nun der Multiplexer aus den drei Gattern AND, OR und NOT zusammenbauen? Wir können natürlich den Multiplexer mit zwei Steuerbits und vier Datenbits als BOOLEsche Funktion auffassen mit sechs Inputbits und einem Outputbit. Wir können die Schalttable erstellen wie im Beispiel [7.44], die Minterme ablesen und die disjunktive Normalform ableiten. Dies ist jedoch für eine Schalttable mit $2^6 = 64$ Zeilen nicht praktikabel. Es gibt jedoch ein Verfahren, das es ermöglicht, dieses Problem in handhabbare Teile zu zerlegen.

Obwohl die Daten- und Steuersignaleingänge lediglich Bits darstellen, besteht ein konzeptueller Unterschied zwischen ihnen. Die Steuerbits treten bei einem Multiplexer niemals als Outputsignale auf. Sie dienen dazu, intern den kompletten Schaltkreis zu steuern. Die Datenbits andererseits laufen unverändert durch die Schaltung. Diese bilden das Rohmaterial, das durch den Schaltkreis unter der Leitung der Steuerbits gefiltert wird.

Ein möglicher Ansatzpunkt für das Design eines Multiplexers besteht darin, die Steuerbits in einen Decoder zu leiten. Jede Outputleitung des Decoders wird AND-Verknüpft mit einer unterschiedlichen Datenleitung, die entstehenden Outputbits werden geORed. Der entsprechende Schaltkreis eines Decoder-basierten Multiplexers mit zwei Steuerbits und vier Datenbits ist in der Abbildung [7.25] dargestellt.

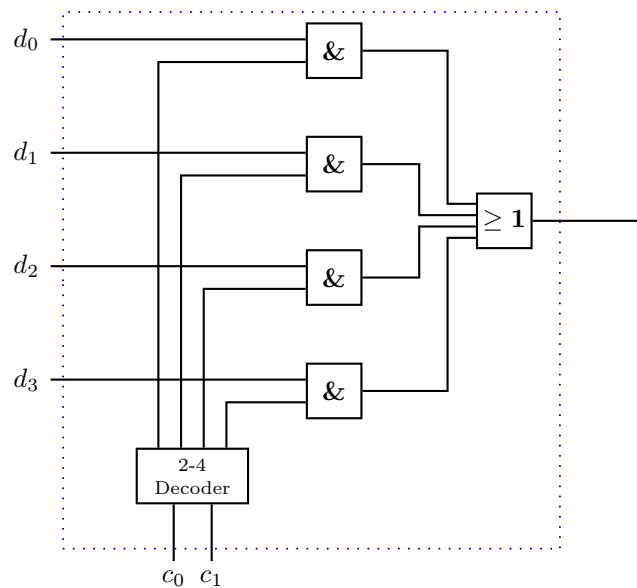


Abbildung 7.25: Decoder-basierte Realisierung eines 4-Daten, 2-Steuerbit Multiplexers.

Wir können den Schaltkreis eines Multiplexers noch etwas vereinfachen, indem wir uns verdeutlichen, was der Multiplexer eigentlich macht. Dadurch läßt sich die Logik des Decoders mit dem restlichen Schaltkreis des Multiplexers kombinieren.

Ziel ist es, einen Schaltkreis zu erhalten mit den Eigenschaften:

Output = d_0 genau dann, wenn $\overline{c_0} \wedge \overline{c_1} = 1$,

Output = d_1 genau dann, wenn $\overline{c_0} \wedge c_1 = 1$,

Output = d_2 genau dann, wenn $c_0 \wedge \overline{c_1} = 1$,

Output = d_3 genau dann, wenn $c_0 \wedge c_1 = 1$.

Schreibt man die Schaltfunktion auf diese Weise einfach hin, kommt man dem gesuchten Schaltkreis sehr nahe:

$$f(c_0, c_1, d_0, d_1, d_2, d_3) = (d_0 \wedge \overline{c_0} \wedge \overline{c_1}) \vee (d_1 \wedge \overline{c_0} \wedge c_1) \vee (d_2 \wedge c_0 \wedge \overline{c_1}) \vee (d_3 \wedge c_0 \wedge c_1). \quad (7.8)$$

Wir ANDen einfach jeden Dateneingang mit der speziellen Kombination von Kontrollbits, die den Datenkanal ausfiltern soll und verknüpfen die Teilresultate mit

einem OR Gatter. Man beachte, dass die Schaltfunktion (7.8) die gleiche Struktur hat wie die Funktion (7.7) aus dem Beispiel [7.44]. Der Schaltkreis zu der Funktion (7.8) ist in der Abbildung [7.26] dargestellt.

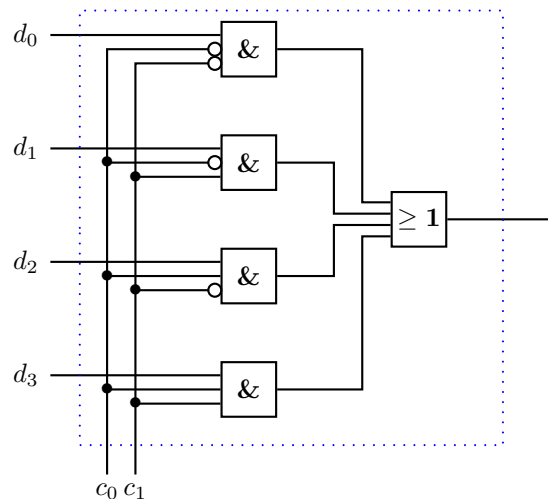


Abbildung 7.26: Einfachere Realisierung eines 4-Daten, 2-Steuerbit Multiplexers.

7.2.9 Komparator

Ein weiterer, wichtiger Schaltkreis, der in Rechnersystemen eingesetzt wird, ist der **Komparator**. Dieser Schaltkreis vergleicht zwei binäre Zahlen und bestimmt, welcher dieser Zahlen größer ist — oder, falls diese Eigenschaft im Komparator implementiert wird, ob die beiden Zahlen gleich sind oder nicht. Wie beim Addierer (siehe Kap. [7.2.3]) bauen wir einen Komparator durch Kaskadierung einer beliebigen Anzahl identischer Grundschaltschaltkreise zusammen. Die Anzahl der Schaltkreise, die wir auf diese Art kaskadieren, legt letztendlich die Größe der zu vergleichenden Binärzahlen fest. Wir setzen dabei voraus, dass beide Zahlen die gleiche Länge haben, andernfalls werden Füllbits verwendet, um beide Zahlen anzupassen. Wir nennen die Zahlen A und B .

Wir überlegen uns zunächst, was der Komparator machen soll. Der Komparator soll ermitteln, welcher der Zahlen A oder B größer ist, falls die beiden Zahlen

nicht gleich sind, und ausgeben, ob die Zahlen gleich sind oder nicht. Diese Forderung führt dazu, dass der Komparator zwischen drei möglichen Zuständen unterscheiden muss:

$$A = B,$$

$$A > B,$$

$$A < B.$$

Wir wissen, dass wir mindestens zwei Bits benötigen, um drei unterschiedliche Zustände unterscheiden zu können. Aus diesem Grund wird der Komparator zwei Outputbits haben. Wir bezeichnen diese Outputbits mit:

e für *equal*,

a_g für *A ist größer*.

Damit ist die Bedeutung der beiden Outputbits des Komparators wie folgt:

$e = 1$ impliziert $A = B$, dann muss $a_g = 0$ sein, *i.e.* $(e, a_g) = (1, 0)$ ist der Zustand $A = B$.

Der Output $(e, a_g) = (0, 1)$ bedeutet $A > B$.

Der Output $(e, a_g) = (0, 0)$ bedeutet $A < B$.

Der Output $(e, a_g) = (1, 1)$ wird von dem Komparator nicht erzeugt.

Wir überlegen uns zunächst, wie man zwei Zahlen A und B , die aus mehreren Ziffern bestehen, in einem beliebigen Stellenwertsystem vergleicht. Falls man an der höchstwertigen Stelle, vergleicht man die Ziffer in dieser Position der Zahl A mit der Ziffer in der gleichen Position der Zahl B . Falls diese Ziffern nicht gleich sind, sind wir bereits fertig, *i.e.* unabhängig davon, welche Zahl A oder B an der höchstwertigen Stelle die größere Ziffer hat, ist die größere Zahl, die folgenden Ziffern spielen dann keine Rolle mehr. Sobald man die Ziffer 7 in der Zahl $A = 756$ mit der Ziffer 1 der Zahl $B = 109$ vergleicht, ist klar, dass $756 > 109$ ist.

Wenn nun die beiden höchstwertigen Ziffern in A und B gleich sind, muss die folgende Stelle untersucht werden. Falls die Ziffern an dieser Stelle ungleich sind, haben wir wieder sofort das Ergebnis. Falls die Ziffern gleich sind, muss wieder die nächste Position untersucht werden. Falls auf diese Art und Weise sämtliche Stellen der Zahlen A und B bis zur niedrigwertigsten Stelle verglichen werden und an jeder Stelle ergibt sich eine Gleichheit der Ziffern, dann müssen die beiden Zahlen A und B gleich sein, *i.e.* $A = B$.

Um nun einen Komparator als kaskadierten Schaltkreis zu entwerfen, benötigen wir einen Schaltkreis, der eine einzelne Stelle vergleicht und verketteten anschließend eine beliebige Anzahl solcher Schaltkreise.

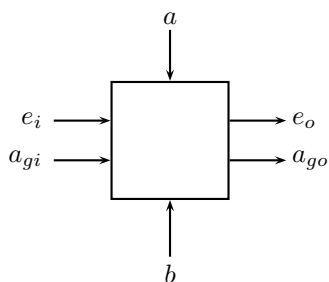


Abbildung 7.27: Ein Komparator für eine Ziffer zweier Zahlen als Black-Box.

Der elementare Schaltkreis hat vier Inputbits a, b, e_i, a_{gi} und zwei Outputbits e_o, a_{go} , dies ist als Black-Box in der Abbildung [7.27] dargestellt. Die beiden Inputbits a und b sind die Ziffern der Zahlen A bzw. B , die der elementare Komparator vergleicht. Die Bits e_i und a_{gi} sind Ergebnisbits aus der unmittelbar vorhergehenden Stelle, *i.e.* dies sind die Outputbits eines elementaren Komparators, der eine höhere Stelle der beiden Zahlen A und B verglichen hat als der elementare Komparator, den wir gerade untersuchen.

Erhält der momentan untersuchte elementare Komparator den Input $(e_i, a_{gi}) = (0, 1)$, dann

7.3 Übungen und Wiederholungsfragen

Übung 7.6:

Man beweise die Idempotenzgesetze der Schaltalgebra unter ausschließlichem Rückgriff auf die Axiomatik BOOLEscher Algebren.

Übung 7.7:

Seien a, b, c Schaltvariable.

1. Zeigen Sie mittels algebraischer Methoden, dass gilt:

$$(a \wedge c) \vee (\bar{a} \vee b) = \bar{a} \vee b \vee c$$

2. Zeigen Sie mit Hilfe von Wertetafeln, dass obige Beziehung gilt.
3. Skizzieren Sie die zugehörigen Schaltkreise

Übung 7.8:

Seien a, b zwei Schaltvariablen. Zeigen Sie folgende Beziehung:

$$(a \wedge \bar{b}) \vee (\bar{a} \wedge b) \vee (a \wedge b) = a \vee b$$

Übung 7.9:

Skizzieren Sie die Schaltungen, die das Assoziativgesetz der Disjunktion dreier Schaltvariablen modellieren:

$$a \vee b \vee c = a \vee (b \vee c) = (a \vee b) \vee c = b \vee (a \vee c)$$

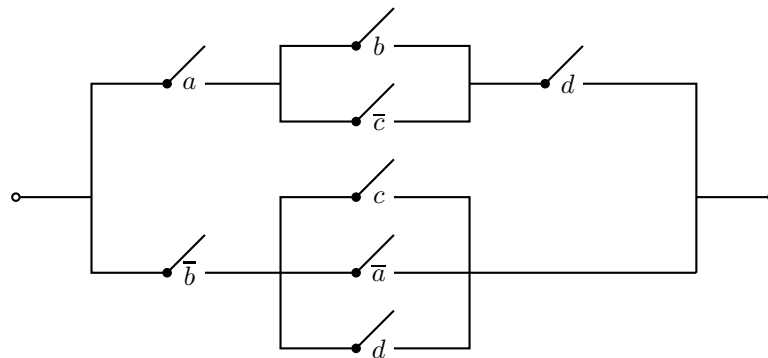
Übung 7.10:

Was ist das Dualitätsprinzip?

Übung 7.11:

Was ist das Kommutativgesetz, das Distributivgesetz und die DEMORGANSchen Regeln für Boolesche Variable?

Übung 7.12:



Formulieren Sie für die nachfolgende Schaltung die Bedingung dafür, daß Strom fließt.

Übung 7.13:

Das SAT Problem. Dieses Entscheidungsproblem spielt in der theoretischen Informatik eine herausragende Rolle und bedeutet folgendes.

Gegeben sind BOOLEsche Variablen a, b, c, \dots . Eine BOOLEsche Funktion ist ein Ausdruck, der BOOLEsche Variable enthält, die durch die Operatoren \wedge, \vee und $\overline{}$ verknüpft sind. Die BOOLEsche Funktion ist *erfüllbar* (engl.: *satisfiable*, daher SAT-Problem), wenn es eine Belegung der BOOLEschen Variablen mit 0 und 1 gibt, so dass die Funktion den Wert 1 annimmt.

Seien

$$f_1(a, b) = a \wedge [(\overline{a} \vee b) \wedge \overline{b}]$$

$$f_2(a, b, c) = a \wedge (b \vee \overline{c}) \wedge (a \vee \overline{b})$$

gegeben. Sind $f_1(a, b), f_2(a, b, c)$ erfüllbar?

Übung 7.14:

Die Beleuchtung eines Zimmers soll an drei verschiedenen Eingangstüren unabhängig ein- und ausgeschaltet werden können, d.h. man soll in einer beliebigen Reihenfolge an einer Stelle das Licht ein- und an einer anderen Stelle das Licht ausschalten können. Geben Sie die Schaltfunktion an.

Übung 7.15:

Eine nicht-negative ganze Zahl kleiner als 10 wird durch eine Dualzahl $b_3b_2b_1b_0$ dargestellt. Man konstruiere eine Schaltfunktion

$$f : \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \longrightarrow \mathbb{B},$$

mit

$$b_3, b_2, b_1, b_0 \longmapsto f(b_3, b_2, b_1, b_0)$$

und

$$f(b_3, b_2, b_1, b_0) = \begin{cases} 1 & \text{wenn } b_3b_2b_1b_0 \text{ Primzahl codiert} \\ 0 & \text{sonst} \end{cases}$$

Übung 7.16:

Bei der Entwicklung eines Kopierers ist eine Schaltung zu entwerfen, welche die Kopierfunktion blockiert und die Störlampe aufleuchten läßt ($S = 1$), wenn der Kopierer angeschaltet ist ($A = 1$) und einer der folgenden Fälle eintritt:

1. Der Toner ist leer ($T = 0$)
2. Es tritt ein Papierstau an den Stellen S_1, S_2, S_3, S_4 auf ($S_i = 1$)
3. Entweder ist das Papierfach leer ($P = 0$) und die Klappe der manuellen Papierzufuhr ist zu ($M = 1$) oder es liegt Papier im Fach und die Klappe für manuelle Papierzufuhr ist gleichzeitig offen.

Anhang A

Glossar

ANSI

Abkürzung für *American National Standards Institute*. ANSI ist ein seit 1918 bestehendes Standardisierungsinstitut in den USA.

ASCII

Abkürzung für *American Standard Code for Information Interchange*. Damit wird eine 7-Bit Zeichencodierung bezeichnet. Der erweiterte ASCII Zeichensatz ist ein 8-Bit Zeichensatz, mit dem 256 Zeichen darstellbar sind.

BIOS

Das Kürzel BIOS steht für *Basic Input/Output System*. Das BIOS enthält alle Grundinformationen für den PC, ohne die er nicht starten kann. Das BIOS ist ein ROM-Speicherchip, auf dem die grundlegenden Programme abgespeichert sind. Diese Programme sorgen unter anderem dafür, daß während der Startphase das Betriebssystem in den Hauptspeicher geladen wird.

Bit

Kunstwort aus *binary digit*. Darunter versteht man ein Platzhalter für genau zwei Zustände: 0 = aus, 1 = ein. Alle in der Datenverarbeitung verwendeten Zeichen werden durch Kombination einer Reihe von Bits dargestellt.

Bus

Datenleitungen in einem Rechnersystem, die verschiedene funktionale Einheiten eines Rechners verbinden. Man unterscheidet Datenbus, Adreßbus und Steuerbus.

Byte

Ein Byte ist eine Informationseinheit bestehend aus 8 Bit.

Cache

Unter Cache versteht man in der Computerarchitektur Zwischenspeicher, die dazu dienen, zwei Komponenten, die Daten unterschiedlich schnell verarbeiten können, abzupuffern. Cache Speicher werden zwischen CPU und Hauptspeicher implementiert, zwischen Hauptspeicher und Festplatte oder als Software Cache auch in Browsern.

CAD

Abkürzung für *Computer-Aided-Design*. Darunter versteht man die Computerunterstützte Konstruktion. CAD kann neben dem Erstellen von Berechnungen auch kaufmännische Aspekte berücksichtigen. Durch den Einsatz von CAD-Systemen können mit wenigen Änderungen schnell und sicher Varianten oder Korrekturen an einer Konstruktion vorgenommen werden.

CD-R

Abkürzung für *Compact Disk Recordable*. CD-Rs sind nur lesbare, digitale Datenträger mit 12 cm Durchmesser. CD-Rs können mit einem CD-Writer (CD-Brenner) einmal beschrieben werden und können bis zu 650 MByte Daten speichern.

CD-ROM

Abkürzung für *Compact Disk Read Only Memory*. CD-ROMs sind nur lesbare, digitale Datenträger mit 12 cm Durchmesser. CD-ROMs können bis zu 650 MByte Text, Grafik, Bild, Ton und bewegte Bilder speichern. Die CD-ROM wird bei ihrer Herstellung beschrieben.

CISC

Complex Instruction Set Computing; Prozessor mit einem umfangreichen Befehlssatz, unterschiedlicher Ausführungsdauer der Befehle und Mikroprogrammsteuerung.

ISO 8859

Codepages

ISO 9660

Von der International Organization for Standardization (ISO) spezifiziertes Dateisystem für CD-ROMs.

RISC

Reduced Instruction Set Computing; Prozessortyp mit einem reduzierten Befehlssatz, vielen Allzweckregistern, Pipelineverarbeitung, superscalare Architektur.

Unicode

16 Bit Code, mit dem sich $2^{16} = 65.536$ Zeichen codieren lassen.

Anhang B

Einheiten

Im Umfeld der IT finden eine Reihe von Einheiten Verwendung, die wir der Vollständigkeit halber an dieser Stelle auflisten wollen.

Weitere Einheiten, die gelegentlich benutzt werden sind:

$$\begin{array}{llll} 1 \text{ WORD} & = & 2 \text{ Bytes} & = 16 \text{ Bit} \\ 1 \text{ DOUBLEWORD} & = & 4 \text{ Bytes} & = 32 \text{ Bit} \\ 1 \text{ NIBBLE} & = & \frac{1}{2} \text{ Byte} & = 4 \text{ Bit} \end{array}$$

$$\begin{array}{ll} 1 \text{ dpi} & = 1 \text{ dot per inch} \\ & = 1 \text{ Bildpunkt pro } 2,54 \text{ cm} \end{array}$$

$$1 \text{ Zoll} = 2,54 \text{ cm}$$

Bei Zugriffszeiten – beispielsweise auf Speichermedien – findet man häufig die folgenden Einheiten:

$$\begin{array}{llll} 1 \text{ ms} & = & 1 \text{ Millisekunde} & = 10^{-3} \text{ sec} \\ 1 \mu\text{s} & = & 1 \text{ Mikrosekunde} & = 10^{-6} \text{ sec} \\ 1 \text{ ns} & = & 1 \text{ Nanosekunde} & = 10^{-9} \text{ sec} \end{array}$$

1 BYTE	=	8 Bit
1 KILOBYTE	=	1 KB
	=	2^{10} BYTE
	=	1.024 BYTE
1 MEGABYTE	=	1 MB
	=	2^{10} KILOBYTE
	=	1.024 KILOBYTE
	=	1.024 x 1.024 BYTE
	=	1.048.576 BYTE
	=	2^{20} BYTE
1 GIGABYTE	=	1 GB
	=	2^{10} MEGABYTE
	=	1.024 MEGABYTE
	=	1.024 x 1.024 KILOBYTE
	=	1.024 x 1.024 x 1.024 BYTE
	=	1.073.741.824 BYTE
	=	2^{30} BYTE
1 TERABYTE	=	1 TB
	=	2^{10} GIGABYTE
	=	1.024 GIGABYTE
	=	1.024 x 1.024 MEGABYTE
	=	1.024 x 1.024 x 1.024 KILOBYTE
	=	1.024 x 1.024 x 1.024 x 1.024 BYTE
	=	1.099.511.627.776 BYTE
	=	2^{40} BYTE
1 PETABYTE	=	1.024 TB
	=	2^{50} BYTE
1 EXABYTE	=	1.024 PB
	=	2^{60} BYTE

Anhang C

Akronyme

Da es im IT Bereich sehr verbreitet ist, mit Abkürzungen zu arbeiten, ist es sehr zweckmäßig, hier ein – sicher nicht vollständiges – Abkürzungsverzeichnis anzufügen. Diese Liste enthält auch Akronyme, die im vorliegenden Skript nicht auftreten.

Akronym	Klartext
ACM	Association for Computing Machinery
AES	Advanced Encryption Standard
AGB	Accelerated Graphics Port
AI	Artificial Intelligence
ALGOL	Algorithmic Language
ALU	Arithmetic Logical Unit
AMD	Advanced Micro Device
ANSI	American National Standards Institute
API	Application Programming Interface
ARPA	Advanced Research Projects Agency
ARM	Advanced Risc Machines
ASCII	American Standard Code of Information Interchange
ASP	Active Server Page
ASP	Application Service Providing
AT	Advanced Technology
AT&T	American Telephone and Telegraph Company
ATM	Automatic Teller Machine
B2A	Business to Administration
B2B	Business to Business
B2C	Business to Consumer
BCD	Binary Coded Decimal

Akronym	Klartext
BI	Business Intelligence
BIOS	Basic Input Output System
BSD	Berkeley System Distribution
BSI	Bundesamt für Sicherheit in der Informationstechnologie
BIT	Binary Digit
C2C	Consumer to Consumer
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CAS	Computer Aided Selling
CCD	Charge Coupled Device
CCITT	Comite Consultatif International Telegraphique et Telephonique
CD	Compact Disk
CD-R	CD Recordable
CD-RW	Rewritable CD
CEO	Chief Executive Officer
CERT	Computer Emergency Resonse Team
CGI	Common Gateway Interface
CHS	Cylinder Head Sector
CIM	Computer Integrated Manufacturing
CISC	Complex Instruction Set Computing
CMM	Capabilty Maturing Model
CMOS	Complementary Metal Oxide Semiconductor
COBOL	Common Business Oriented Language
COW	Cluster of Workstations
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CR	Carriage Return
CRM	Customer Relationship Management
CRT	Cathode Ray Tube
DB	Datenbank
DBMS	Datenbank Management System
DEC	Digital Equipment Corporation
DES	Data Encryption Standard
DIN	Deutsches Institut für Normung
DLL	Dynamic Linked Library
DMA	Direct Memory Access
DMS	Dokumentmanagement System
DOS	Disk Operating System
DPI	Dot Per Inch

Akronym	Klartext
DRAM	Dynamic RAM
DTP	Desktop Publishing
DTD	Document Type Definition
DVD	Digital Versatile Disk
DVD-R	Digital Versatile Disk Recorder
EAN	Europäische Artikel Nummer
EBCDIC	Extended Binary Coded Decimal Interchange Code
EC	Electronic Commerce
ECC	Error Correcting Code
ECMA	European Computer Manufacturers Association
EDI	Electronic Data Interchange
EDIFACT	EDI for Administration, Commerce and Transport
EDS	Electronic Data Systems
EEPROM	Electrically Erasible Programmable ROM
EFF	Electronic Frontier Foundation
EFS	Encryption File System
EGA	Enhanced Graphics Adapter
EISA	Extended Industry Standard Architecture
EPROM	Erasible Programmable ROM
ERP	Enterprise Resource Planning
ETL	Extract, Transform, Load
FAQ	Frequently Asked Questions
FAT	File Allocation Table
FDD	Floppy Disk Drive
FF	Form Feed
FIPS-PUB	Federal Information Standards Publication
FLOPS	Floating Point Operation Per Second
FORTRAN	Formular Translator
FPU	Floating Point Unit
FSB	Front Side Bus
FSF	Free Software Foundation
GB	Giga Byte
GHz	Giga Hertz
GIF	Graphic Interchange Format
GNU	Gnu is Not Unix
GPL	GNU Public License
GSM	Global System for Mobile Communication
GUI	Graphical User Interface

Akronym	Klartext
HDD	Hard Disk Drive
HP	Hewlett Packard
HPFS	High Performance File System
HP-PA	Hewlett Packard Precision Architecture
HPPCL	Hewlett Packard Printer Communication Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IAB	Internet Architecture Board
IANA	Internet Assigned Numbers Authority
ICANN	Internet Corporation for Assigned Names and Numbers
IDE	Integrated Development Environment
IDEA	International Data Encryption Algorithm
IEEE	Institute of Electrical and Electronics Engineers
IESG	Internet Engineering Steering Group
IETF	Internet Engineering Task Force
IBM	International Business Machines
INTEL	Intergated Electronics
IP	Internet Protocol
IR	Information Retrieval
IrDA	Infrared Data Association
ISA	Instruction Set Architecture
ISA	Industry Standard Architecture
ISO	International Organisation for Standardization
ISOC	Internet Society
IT	Information Technology
ITU	International Telecommunication Union
J2EE	Java 2 Enterprise Edition
JDBC	Java Database Connectivity
JPEG	Joint Photographic Expert Group
JRE	Java Runtime Environment
JSP	Java Server Pages
JVM	Java Virtual Machine
KB	Kilo Byte
KHz	Kilo Hertz
KI	Künstliche Intelligenz
L1	Level 1
L2	Level 2
LCD	Liquid Crystal Display

Akronym	Klartext
LF	Line Feed
LTE	Long Term Evolution
LZW	Lempel–Ziv–Welch
MB	Mega Byte
MBR	Master Boot Record
MCA	Micro Channel Architecture
MHz	Mega Hertz
MIMD	Multiple Instruction Multiple Data
MIPS	
MIT	Massachusetts Institute of Technology
MMX	Multi Media Extension
MPEG	Moving Picture Experts Group
MPR II	
MS	Microsoft
NAS	Network Attached Storage
NBS	National Bureau of Standards
NCR	
NIST	National Institute of Standards and Technology
NSA	National Security Agency
NT	New Technology
NTFS	New Technology File System
OCR	Optical Character Recognition
ODBC	Open Database Connectivity
OEM	Original Equipment Manufacturer
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
OMG	Object Management Group
OODB	Object Oriented Database
OSF	Open Software Foundation
OS	Operating System
OS/2	Operating System/2
OSI	Open Systems Interconnection
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCMCIA	Personal Computer Memory Card International Association
PDA	Personal Digital Assistant
PDF	Portable Document Format
PERL	Practical Extraction and Reporting Language
PGP	Pretty Good Privacy

Akronym	Klartext
PIN	Personal Identification Number
PNG	Portable Network Graphics
POSIX	Portable Operating System Interface for Unix
POWERPC	Performance Optimization With Enhanced RISC
PROM	Programmable ROM
PS/2	Personal System/2
QoS	Quality of Service
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RFC	Request for Comments
RFID	Radio Frequency Identification
RGB	Red Green Blue
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
RSA	Rivest Shamir Adleman
RTF	Rich Text Format
SAA	Systems Application Architecture
SAN	Storage Area Network
SAP	Systeme, Anwendungen, Produkte in der Datenverarbeitung
SCSI	Small Computer System Interface
SCM	Supply Chain Management
SCO	Santa Cruz Corporation
SEI	Software Engineering Institute
SGI	Silicon Graphics International
SGML	Standard Generalized Markup Language
SHA	Secure Hash Algorithm
SIG	Special Interest Group
SIGACT	Special Interest Group on Algorithms and Computation Theory
SIGCOMM	Special Interest Group on Data Communications
SIGPLAN	Special Interest Group on Programming Languages
SIGSOFT	Special Interest Group on Software Engineering
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SLA	Service Level Agreement
SMP	Symmetric Multi Processing
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SNA	Systems Network Architecture
SOAP	Simple Object Access Protocol

Akronym	Klartext
SPARC	Scalable Processor Architecture
SPEC	Systems Performance Evaluation Cooperative
SQL	Structured Query Language
SRAM	Static RAM
SSD	Solid State Drive
SSL	Secure Socket Layer
SVGA	Super Video Graphics Array
TAN	Transaction Number
TB	Tera Byte
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TFT	Thin Film Transistor
TIFF	Tagged Image File Format
TLS	Transport Layer Security
UCS	Universal Character Set
UDP	User Datagram Protocol
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
UPM	Umdrehungen Pro Minute
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTF-8	8-Bit UCS Transformation Format
UV	Ultra Violett
VBA	Visual Basic for Applications
VESA	Video Electronics Standard Association
VGA	Video Graphics Array
VoIP	Voice over IP
VPN	Virtual Private Network
VRAM	Video RAM
VRML	Virtual Reality Markup Language
W3C	World Wide Web Consortium
WWW	World Wide Web
WYSIWYG	What you see is what you get
XeroxPARC	Xerox Palo Alto Research Center
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XT	Extended Technology

Anhang D

Literaturhinweise

In diesem Anhang versuchen wir, einen Überblick über die mittlerweile undurchdringliche Vielfalt der Computer- und Informatikliteratur zu erhalten.

Generell ist die 14-tägig erscheinende Zeitschrift *c't, Magazin für computer technik* als Standard-Zeitschrift zu empfehlen. Etwas spezieller sind die Inhalte des *iX*-Magazins. Die Beiträge sind eher für die professionelle Anwendung im Netzwerkbereich oder auf Unix-Plattformen konzipiert. Die eher betriebswirtschaftliche Seite wird in der *Computerwoche* abgedeckt.

Selbstredend ist die Auswahl hier subjektiv.

D.1 Historische Entwicklung

Eine umfassenden Überblick der historischen Entwicklung über verschiedene Aspekte der EDV findet man in [45, 76, 105, 111]. Speziellere Betrachtungen – zum Teil autobiografischer Natur – findet man in [40] und [113]. Empfehlenswert sind die Biografien über Turing [54], John von Neumann [72] und Charles Babbage [57].

Eine sehr gute Darstellung der Computer-Revolution Mitte der 70er Jahre, die zur Entwicklung der Personal Computer führte, ist [35].

D.2 Hindergründiges

Es gibt eine Reihe empfehlenswerter Bücher, die sich mit dem 'Drumherherum' von Teilgebieten der Informatik befassen. Dies sind Sachbücher, die sich eher an ein allgemeines Publikum richten. Dennoch lohnt es sich, solche Bücher zu lesen, da sie Hintergründe beleuchten.

Ein Insiderblick hinter die Kulissen von INTEL gibt das Buch [61] von Tim Jackson. Die Erfolgsstory von INTEL findet man in [42]. Die Story der Firma Microsoft wird in [58] geschildert, ein etwas anderer Blickwinkel gibt hier [27].

Der Konflikt zwischen IBM und Microsoft ist Thema des Buches [18]. Bill Gates' Zukunftsvisionen hat er höchstpersönlich in [37] der Welt bekanntgegeben. Die Entwicklung und Standpunkte der Open Source Bewegung findet man in [85] und [23].

D.3 Informatik Einführung

D.4 Theoretische Informatik

Die Bibel in Sachen Algorithmen ist Donald Knuths epochales Werk [65] bis [67]

D.5 Betriebssysteme

Das Standardwerk über Betriebssysteme ist Tanenbaum [103].

D.6 Rechnerarchitektur

D.7 Programmiersprachen

D.8 Netzwerke

Literaturverzeichnis

- [1] ALFRED V. AHO, RAVI SETHI und JEFFREY D. ULLMANN
Compilerbau
Teil 1
2. Auflage
Oldenbourg Verlag
München, Wien, 1999.
- [2] JÜRGEN ALEX
Wege und Irrwege des Konrad Zuse
Spektrum der Wissenschaften
Januar 1997, Seite 78.
- [3] W. AMELING
Digitalrechner
Grundlagen und Anwendungen
Friedrich Vieweg Verlag
Braunschweig 1990.
- [4] ISAAC ASIMOV
Asimov's New Guide to Science
Penguin Books
Harmondsworth, Middlesex, England, 1972
Deutsche Ausgabe:
Die exakten Geheimnisse unserer Welt
Bausteine des Lebens
Droemer Knauer, München, 1986, Kapitel 7.
- [5] WILLIAM ASPRAY
John von Neumann and The Origins of Modern Computing
The MIT Press
Cambridge, MA, London, 1990.
- [6] ACHIM BACHEM, THOMAS LIPPERT
Wie Supercomputer die Forschung prägen
Spektrum der Wissenschaft, 11/2011, pp. 84.

- [7] HELMUT BALZERT
Lehrbuch der Software-Technik
Spektrum Akademischer Verlag
Heidelberg, 1996.
- [8] GUIDO BARTSCH
Hitzköpfe
Konzepte der Mikroprozessortechnik
iX Magazin für professionelle Informationstechnik
Oktober 1997, Seite 150.
- [9] GUIDO BARTSCH, RALF HÜLSENBUSCH
Serengeti lebt
SunFire 6800 als Server und Number Cruncher
iX Magazin für professionelle Informationstechnik
August 2001, Seite 60 - 64.
- [10] ALBRECHT BEUTELSPACHER und MARC-ALEXANDER ZSCHIEGNER
Diskrete Mathematik für Einsteiger
Mit Anwendungen in Technik und Informatik
2. Auflage
Vieweg Verlag
Braunschweig, 2002.
- [11] NORBERT BLUM
Theoretische Informatik
Eine anwendungsorientierte Einführung
R. Oldenbourg Verlag
München, 1998.
- [12] ACHIM BORN
Kettenspiele
Supply Chain Management: Optimierte Lieferketten
iX - Multiuser Multitasking Magazin
September 1998, S. 68 - 71.
- [13] WERNER BRECHT
Theoretische Informatik
Grundlagen und praktische Anwendungen
Friedrich Vieweg Verlag
Braunschweig 1995.
- [14] ULLRICH BREYMAN
C++ Eine Einführung
Carl Hanser Verlag
München, 1997.
- [15] FREDERICK P. BROOKS, JR.
The Mythical Man-Month

- Addison-Wesley Publishing Company
Reading, Massachusetts, 1995.
- [16] ROLF JÜRGEN BRÜSS
Von CISC zu RISC?
Design & Elektronik
Nr. 24, November 1991.
- [17] ARTHUR W. BURKS, HERMAN H GOLDSTINE and JOHN VON NEUMANN
Preliminary Discussion of the Logical Design of an Electronic Computing Instrument
Reprinted in:
William Aspray and Arthur W. Burks
Papers of John von Neumann on Computing and Computer Theory
Charles Babbage Institute Reprint Series for the History of Computing,
Vol. 12
The MIT Press
Cambridge, London, 1987.
- [18] PAUL CARROLL
Der Computerkrieg
IBM gegen Bill Gates Microsoft, ein Kampf ums Überleben
Paul Zsolnay Verlag
Wien, 1994.
- [19] JOHN L. CASTI
Das Cambridge Quintett
Berlin Verlag
Berlin, 1998.
- [20] E.F. CODD
A Relational Model for Large Shared Data Banks
Communications of the ACM
Vol. 13, Nr. 6, pp. 377, 1970.
- [21] S. COFFIN
UNIX System V Release 4
The Complete Reference
Osborne, McGraw-Hill, Berkeley 1991.
- [22] Computational Science Education Project
Computer Architecture
1995.
Electronic Document, verfügbar unter
<http://csep1.phy.ornl.gov/csep.html>
- [23] CHRIS DiBONA, SAM OCKMAN and MARK STONE (Ed.)
Open Sources

- Voices from the Open Source Revolution*
O'Reilly & Associates, Inc.
Beijing, Cambridge, Köln, Paris, Sebastopol, 1999.
- [24] JÜRGEN DIERCKS
Websourcing
ASP: Zukunftsmarkt mit Startproblemen
iX Magazin für professionelle Informationstechnik
Januar 2001, Seite 106 - 111.
- [25] RALF DRAEGER
Messers Schneide
Sun Blade 1000 mit UltraSPARC-III-CPU
iX Magazin für professionelle Informationstechnik
Januar 2001, Seite 68 - 72.
- [26] HUBERT L. DREYFUS, STUART E. DREYFUS
Künstliche Intelligenz
Von den Grenzen des Maschinendenkens und dem Wert der Intuition
Rowohlt Taschenbuch Verlag
Reinbek bei Hamburg, 1990.
- [27] JENNIFER EDSTROM, MARLIN ELLER
Barbarians Led by Bill Gates
Microsoft von innen betrachtet
MITP Verlag
Bonn, 1999.
- [28] MARGARITA ESPONDA, RAÚL ROJAS
The RISC Concept – A Survey of Implementations
<http://www.inf.fu-berlin.de/lehre/WS94/RA/RISC-9.html>.
- [29] USAMA FAYYAD, GREGORY PIATETSKY-SHAPIRO and PADHRAIC SMYTH
From Data Mining to Knowledge Discovery in Databases
AI Magazine **17**, (1996), 37 – 54.
- [30] RICHARD P. FEYNMAN
Feynman Lectures on Computation
Westview Press
Boulder, Colorado, 2000
- [31] LUCIANO FLORIDI
Information
A very short Introduction
Oxford University Press
Oxford, 2010.
- [32] LUCIANO FLORIDI
Semantic Conceptions of Information

- The Stanford Encyclopedia of Philosophy (Spring 2011 Edition),
Edward N. Zalta (ed.),
URL = <<http://plato.stanford.edu/archives/spr2011/entries/information-semantic/>>.
- [33] MICHAEL FLYNN
Very High Speed Computing Systems
Proceedings of the IEEE, Vol. **54**, 1901-1909, 1966.
- [34] MICHAEL FLYNN
Some Computer Organizations and Their Effectiveness
IEEE Trans. Comp., Vol. C-21, pp. 94, 1972.
- [35] PAUL FREIBERGER, MICHAEL SWAINE
Fire in the Valley
The Making of the Personal Computer
Second Edition
McGraw-Hill
New York, San Francisco, 2000.
- [36] MARCUS FRIEDRICH
Eingerahmt
IBMs neue System/390 - Generation
iX Magazin für professionelle Informationstechnik
Oktober 1998, Seite 38-43.
- [37] BILL GATES
The Road Ahead
Viking Penguin
London, New York, 1995.
- [38] W. WAYT GIBBS
Software: chronisch mangelhaft
Spektrum der Wissenschaft, Seite 56,
Dezember 1994.
- [39] JAMES GLEICK
The Information
A History, a Theory, a Flood
Fourth Estate
London, 2011.
- [40] HERMAN H. GOLDSTINE
The Computer from Pascal to von Neumann
Princeton University Press
Princeton, New Jersey, 1972.
- [41] JOHN R. GREGG
Ones and Zeros
Understanding Boolean Algebra, Digital Circuits and the Logic of Sets

- IEEE Press
New York, 1998.
- [42] ANDREW S. GROVE
Nur die Paranoiden überleben
Strategische Wendepunkte vorzeitig erkennen
Wilhelm Heyne Verlag
München, 1999.
- [43] J. GULBINS und K. OBERMAYR
UNIX System V.4.
Begriffe, Konzepte, Kommandos, Schnittstellen
4. Auflage
Springer Verlag
Berlin, Heidelberg, New York, 1995.
- [44] HEINZ-PETER GUMM und MANFRED SOMMER
Einführung in die Informatik
Addison Wesley
Bonn, Paris, Reading Massachusetts, 1994.
- [45] KATIE HAFNER und MATTHEW LYON
Arpa Kadabra
Die Geschichte des Internet
dpunkt Verlag
Heidelberg 1997.
- [46] RICHARD W. HAMMING
Coding and Information Theory
Prentice Hall, Englewood Cliffs
New Jersey, 1986.
- [47] WILLIAM W. HARGROVE, FORREST M. HOFFMAN, THOMAS STERLING
Der selbstgebastelte Supercomputer
Hypercomputer (Teil II)
Spektrum der Wissenschaften
Heft März 2002, pp. 88 - 96.
- [48] UWE HARMS
Klassiker in Neuauflage
BS2000/OSD – Großrechner von Siemens
iX Magazin für professionelle Informationstechnik
Oktober 1998, Seite 44-49.
- [49] UWE HARMS
Auf dem Absprung
Supercomputer haben stark nachgelegt
c't Magazin für Computertechnik
Heft 14, 2002, Seite 74 - 75.

- [50] H.-J. HECK
Standardbetriebssystem UNIX
Eine strukturierte Einführung
Rowohlt Taschenbuch Verlag
Reinbek bei Hamburg, 1991.
- [51] JOHN L. HENNESSY and DAVID A. PATTERSON
Computer Architecture A Quantitative Approach
Second Edition
Morgan Kaufman Publishers, Inc.
San Francisco, California, 1996.
- [52] HELMUT HEROLD, BRUNO LURZ und JÜRGEN WOHLRAB
Grundlagen der Informatik
Praktisch — Technisch — Theoretisch
Pearson Studium
München, 2007.
- [53] RAYMOND HILL
A First Course in Coding Theory
Oxford University Press
Oxford, 2000.
- [54] ANDREW HODGES
Alan Turing, Enigma
Zweite Auflage
Springer-Verlag
Wien, New York, 1994.
- [55] DOUGLAS R. HOFSTADTER
Gödel, Escher, Bach
Ein Endloses Geflochtenes Band
Klett Cotta
Stuttgart, 1985.
- [56] RALPH HÜLSENBUSCH
SALTo Mortale
SuSEs Advanced Linux Technology
iX Magazin für professionelle Informationstechnik
Juli 2000, Seite 72 - 76.
- [57] ANTHONY HYMAN
Charles Babbage, 1791 – 1871
Philosoph, Mathematiker, Computerpionier.
Ernst Klett Verlage GmbH u. Co. KG
Stuttgart, 1987.
- [58] DANIEL ICHBIAH
Die Microsoft Story

Wilhelm Heyne Verlag
München, 1996.

- [59] GEORGES IFRAH
Universalgeschichte der Zahlen
Campus Verlag
Frankfurt/New York, 1998.
- [60] BERNHARD IRRGANG und JÖRG KLAWITTER
Künstliche Intelligenz
S. Hirzel, Wissenschaftliche Verlagsgesellschaft
Stuttgart, 1990.
- [61] TIM JACKSON
Inside Intel
Die Geschichte des erfolgreichsten Chip-Produzenten der Welt
Hoffmann und Campe Verlag
Hamburg, 1998.
- [62] MARTIN B. KALINOWSKI
Bombengeschäft
Atomtests im Rechner: Ausweg oder Gefahr?
c't Magazin für Computertechnik
Heft 2, 1996, Seite 70.
- [63] VICTOR J. KATZ
A History of Mathematics
An Introduction
3rd Edition
Addison Wesley
Reading, Massachusetts, 2009.
- [64] BRIAN W. KERNINGHAN und DENNIS M. RITCHIE
Programmieren in C
Carl Hanser Verlag
München, 1990.
- [65] DONALD E. KNUTH
The Art of Computer Programming
VOLUME 1: Fundamental Algorithms
Second Edition
Addison Wesley
Reading, Massachusetts, 1998.
- [66] DONALD E. KNUTH
The Art of Computer Programming
VOLUME 2: Seminumerical Algorithms
Second Edition

- Addison Wesley
Reading, Massachusetts, 1998.
- [67] DONALD E. KNUTH
The Art of Computer Programming
VOLUME 3 Sorting and Searching
Second Edition
Addison Wesley
Reading, Massachusetts, 1998.
- [68] THOMAS KOSHY
Discrete Mathematics with Applications
Elsevier, Academic Press
Amsterdam, 2004.
- [69] GUIDO KRÜGER
Programmieren in C
Grundlagen, Konzepte, Übungen, 2. Auflage
Addison-Wesley Publishing Company
Bonn, 1995.
- [70] KENNETH C. LAUDON, JANE P. LAUDON und DETLEF SCHODER
Wirtschaftsinformatik
Eine Einführung
Pearson Studium
München, 2006.
- [71] DOUGLAS B. LENAT
Software für Künstliche Intelligenz
Spektrum der Wissenschaften
November 1984, Seite 178.
- [72] NORMAN MACRAE
John von Neumann
Mathematik und Computerforschung - Facetten eines Genies
Birkhäuser Verlag
Basel, Boston, Berlin, 1994.
- [73] CHRISTIAN MÄRTIN
Rechnerarchitekturen
CPUs, Systeme, Software-Schnittstellen
Fachbuchverlag Leipzig
Carl Hanser Verlag
München, Wien, 2001.
- [74] HELMUT MALZ
Rechnerarchitektur
Eine Einführung für Ingenieure und Informatiker

- Friedrich Vieweg Verlag
Braunschweig, 2001.
- [75] MICHAEL MERZ
E-Commerce und E-Business
Marktmodelle, Anwendungen und Technologien
dpunkt Verlag
Heidelberg, 2002.
- [76] JOHN PALFREMAN and DORON SWADE
The Dream Machine
Exploring the Computer Age
BBC Books, London, 1992.
- [77] DAVID PATTERSON
Reduced Instruction Set Computers
Communications of the ACM
Vol. **28**, 1, Jan. 1985, pp. 9 - 21.
- [78] DAVID A. PATTERSON and JOHN L. HENNESSY
Computer Organization & Design
The Hardware/Software Interface
Second Edition
Morgan Kaufmann Publishers, Inc.
San Francisco, 1998.
- [79] ROGER PENROSE
The Emperor's New Mind
Concerning Computers, Minds, and the Laws of Physics
Oxford University Press
New York, 1989.
Deutsche Übersetzung:
Computerdenken
Die Debatte um Künstliche Intelligenz, Bewußtsein und die Gesetze der Physik
Spektrum der Wissenschaft Verlagsgesellschaft
Heidelberg.
- [80] ROGER PENROSE
Das Große, das Kleine und der menschliche Geist
Spektrum Akademischer Verlag GmbH
Heidelberg, Berlin, 1998.
- [81] CHARLES PETZOLD
The Annotated Turing
Wiley, 2008.
- [82] JOHN R. PIERCE
An Introduction to Information Theory

- Symbols, Signals and Noise
Dover Publications, Inc.
New York, 1980.
- [83] CHRISTOPH PÖPPE
Der erste ernstzunehmende Computerbeweis
Spektrum der Wissenschaften
August 1997, Seite 32.
- [84] CHRISTOPH PÖPPE (Hrsg.)
Rechnerarchitekturen
Dossier Spektrum der Wissenschaften
Dossier 4/2000.
- [85] ERIC S. RAYMOND
The Cathedral & the Bazaar
Musings on Linux and Open Source by an Accidental Revolutionary
O'Reilly & Associates, Inc.
Beijing, Cambridge, Köln, Paris, Sebastopol, 1999.
- [86] PETER RECHENBERG
Was ist Informatik?
Eine allgemeinverständliche Einführung
3. Auflage
Carl Hanser Verlag
München, Wien, 2000.
- [87] K. REISS, G. SCHMIEDER
Basiswissen Zahlentheorie
Eine Einführung in Zahlen und Zahlbereiche
Springer Verlag
Berlin, Heidelberg, 2005.
- [88] ULRICH REMOLD, PAUL LEVI
Einführung in die Informatik
3., vollständig überarbeitete und erweiterte Auflage
Carl Hanser Verlag
München, Wien, 1999.
- [89] R. RICHTER, P. SANDER und W. STUCKY
Der Rechner als System
Organisation, Daten, Programme
B.G. Teubner, Stuttgart 1997.
- [90] MICHAEL RIEPE
Ein U vorgemacht
Einbausysteme mit zwei UltraSPARC-III von Sun und Tatung
iX Magazin für professionelle Informationstechnik
Dezember 2001, Seite 78 - 83.

- [91] MICHAEL RIORDAN und LILIAN HODDESON
Der Beginn der Mikroelektronik
Spektrum der Wissenschaft
März 3/1998, Seite 80.
- [92] RAÚL ROJAS
Konrad Zuses Rechenmaschinen: sechzig Jahre Computergeschichte
Spektrum der Wissenschaften
Mai 1997, Seite 54.
- [93] STEVEN ROMAN
Introduction to Coding and Information Theory
Springer Verlag
New York, Berlin, Heidelberg, 1997.
- [94] CLAUDE E. SHANNON
A Symbolic Analysis of Relay and Switching Circuits
Transactions American Institute of Electrical Engineers, Vol 57 (1938) pp.
713 – 723.
- [95] GABY SCHULEMANN
Richtige Rechner
Supercomputer auf dem Weg zum Weltsimulator
c't Magazin für Computertechnik
Heft 11, 1998, Seite 82 - 92.
- [96] HUBERT SIEVERDING
In weiter Ferne
Unix ist in die Jahre gekommen
iX Multiuser Multitasking Magazin
Oktober 1995, Seite 132.
- [97] MICHAEL SIPSER
Introduction to the Theory of Computation
PWS Publishing Company
Boston, Albany, Bonn, 1996.
- [98] DAVID H.M. SPECTOR
Building Linux Clusters
O'Reilly & Associates, Inc.
Beijing, Cambridge, 2000.
- [99] WILLIAM STALLINGS
Computer Organization and Architecture
Fifth Edition
Prentice Hall
New Jersey, 2000.

- [100] WILLIAM STALLINGS
Operating Systems
Internals and Design Principles
Third Edition
Prentice Hall
New Jersey, 1998.
- [101] THOMAS STERLING
Auf dem Weg zum Billiarden-Rechner
Hypercomputer (Teil I)
Spektrum der Wissenschaft
Heft Januar 2002, pp. 78 - 90.
- [102] DORON D. SWADE
Der mechanische Computer des Charles Babbage
Spektrum der Wissenschaft
April 1993, Seite 78.
- [103] ANDREW S. TANENBAUM
Moderne Betriebssysteme
2. Auflage
Carl Hanser Verlag
München, Wien, 1995
- [104] ANDREW S. TANENBAUM, JAMES GOODMAN
Computer Architektur
Strukturen, Konzepte, Grundlagen
4. Auflage
Prentice Hall
München, 1999.
- [105] HANNSPETER VOLTZ
Menschen und Computer
Markt & Technik Buch - und Software - Verlag GmbH & Co.
Haar bei München 1993.
- [106] HANS CHRISTIAN VON BAEYER
Information
The New Language of Science
Harvard University Press,
Cambridge, Massachusetts, 2004.
- [107] JOHN VON NEUMANN
First Draft of a Report on the EDVAC
Reprinted in:
William Aspray and Arthur W. Burks
Papers of John von Neumann on Computing and Computer Theory

- Charles Babbage Institute Reprint Series for the History of Computing,
Vol. 12
The MIT Press
Cambridge, London, 1987.
- [108] JOHN VON NEUMANN
The Computer and the Brain
Second Edition
Yale University Press
New Haven, London, 2000.
- [109] GOTTFRIED VOSSEN, KURT-ULRICH WITT
Grundlagen der Theoretischen Informatik mit Anwendungen
Friedrich Vieweg Verlag
Braunschweig, Wiesbaden, 2000.
- [110] HANNO WAGNER
Vom Traum zur Wirklichkeit
CLOWN '98 aus der Sicht der Netzwerker
Linux-Magazin, Heft 2/99, Seite 64 – 68.
- [111] ROBERT WEISS
Mit dem Computer auf Du
Midas Verlag
Männedorf, Schweiz, 1993.
- [112] NORBERT WIENER
Cybernetics
or Control and Communication in the Animal and the Machine
Second edition
The MIT Press
Cambridge, Massachusetts, 1998.
- [113] MAURICE V. WILKES
Computer Perspectives
Morgan Kaufmann Publishers Inc.
San Francisco, California, 1995.
- [114] C. WOLFINGER
Keine Angst vor UNIX
Ein Lehrbuch für Einsteiger
VDI Verlag, München, 1993.

Index

- Ueberlauf, 178
- ACM, 45
- Adressbusbreite, 89
- Adressierung
 - überdeckte, 91
- AIX, 62
- Akkumulator, 79, 83
- Algorithmus, 153
 - Determinismus, 154
 - Effektivität, 154
 - Exaktheit, 154
 - Fintheit, 154
 - Terminierung, 154
 - Vollständigkeit, 154
- Algorithmus von BOOTH, 180
- Algorithmus, deterministischer, 154
- Alphabet, 118
- ALU, 75
- ANSI, 44, 127, 255
- ANSI/IEEE 754–2008, 200
- ANSI C, 44
- ANSI Zeichensatz, 132
- Apple, 59
- Application Service Providing, 25
- Arbeitsspeicher, 73
- Area, 48
- Area Director, 49
- Arithmetic right shift, 181
- Arithmetical Logical Unit, 75
- ARPANET, 47
- AS/400, 61
- ASCII, 127, 255
- ASCII Code, 126
- ASCII–Wert, 128
- Ausgabewerk, 73
- Automatentheorie, 7, 8
- B2A, 20
- B2B, 19
- B2C, 20
- Barcode, 137–139
- Barlow, John, 46
- BCD-Code, 127
- BCP, 51
- BDE–Systeme, 62
- Befehlsdecoder, 81
- Befehlsdekodierer, 78
- Befehlsregister, 80
- Befehlszähler, 80
- Bereichsüberschreitung, 88
- Berners-Lee, Tim, 52
- Betriebsdatenerfassung, 62
- Biasdarstellung, 202
 - biased exponent, 202
 - Charakteristik, 202
- Bilderkennung, 26
- Binärcodierung, 122, 125
- Binärdarstellung, 190
 - Definition, 197
- Binärsystem, 149, 151–189
 - Addition, 159
 - Division, 165
 - Multiplikation, 160
 - Subtraktion, 159
 - Wurzelziehen, 165
- Binärzahl, 128
- Binary Digit, 122
- Bio–Informatik, 11
- BIOS, 255
- Bit, 122, 158, 255
- Blockcodierung, 125
- BOOLEsche Variable, 158
- BSI, 47
- Bundesnetzagentur, 39

- Burks, Arthur, 72
- Bus, 255
- Business Intelligence, 25
- Bussystem, 74
- Byte, 128, 256

- C2B, 20
- C2C, 20
- C64, 57
- Cache, 256
- Cache-Speicher, 108
- CAD, 256
- Capability Maturing Model, 47
- Capability Maturity Model, 53
- Carriage Return, 118
- Carry Bit, 78
- CAS Systeme, 24
- CCITT, 36
- CD-R, 256
- CD-ROM, 256
- CISC, 256
- CISC Prozessor, 76, 103
- Client/Server Architektur, 14
- Cluster, 67, 102
- Clusteranalyse, 22
- CMMI, 55
- Code, 123
- Codeseite, 135
- Codewort, 123
- Codierer, 241
- Codierung
 - big endian, 132
 - Definition, mathematische, 123
 - DIN 44300, 122
 - little endian, 132
- Commodore, 59
- Compiler, 8, 78
- Compilerbau, 8
- Computer Emergency Response Team, 53
- Computer Science, 7
- Computerklassen, 56
- CRM Systeme, 24

- Data Mining, 5
- Data Warehouse, 20

- Data-Mart, 22
- Data-Mining, 21–24
 - Assoziationsanalyse, 23
 - Ausreißererkennung, 22
 - Clusteranalyse, 23
 - Klassifikation, 23
 - Regressionsanalyse, 23
 - Zusammenfassung, 23
- Datenbus, 243
- Datenverarbeitung, 28
- Datenwege, 73
- DB2, 61
- DEC, 113
- Decoder, 87, 242
- Deduktionssysteme, 27
- Department of Defense, 47
- Dezimalbruch, 155
- Dezimalsystem, 147, 149
- Differenz, 168
- DIJKSTRA, EDGAR, 97
- DIN, 40
- Dokumentenmanagementsystem, 24
- DOUBLEWORD, 260
- Dreiadressbefehl, 84
- Dreiadressmaschine, 78
- Dualsystem, 149

- E-Governmant, 11
- E-Health, 11
- eBay, 20
- EBCDIC - Code, 127
- eBusiness, 19
- ECBDIC-Zeichensatz, 133–134
- Eckert, John, 72
- eCommerce, 19
- EDIFACT, 20
- EDVAC, 72
- EFF, 46
- eGovernment, 20
- Einadressbefehl, 83
- Einadressmaschine, 79
- Einerkomplement, 170
- Eingabewerk, 73
- ENIAC, 72
- Enterprise Resource Planning System, 16

- ERP System, 16
 - Customizing, 19
 - Erweiterbarkeit, 19
 - Integration, 18
 - Kommerzielle Vollständigkeit, 18
 - Modularität, 19
 - Prozeßorientierung, 19
- Expertensysteme, 27
- FDE-Zyklus, 72
- Feldrechner, 101
- Fetch Decode Execute Zyklus, 80, 84–89
- FIPS, 45
- Firewire, 43
- Flag Register, 87, 88
- Flag-Register, 78
- Flops, 66
- FLYNN, MICHAEL, 100
- FLYNN Taxonomie, 100
- Formale Semantik, 9
- Formale Sprachen, 9
- FORTRAN, 44
- FSF, 45
- FYI, 51
- Gatter, 215
- Gleitkommazahl
 - ∞ , 205
 - 0.0, 205
 - Basis, 200
 - Denormalisierung, 211
 - double, 205
 - Exponent, 199, 200
 - float, 203
 - Genauigkeit, 212
 - Hidden bit, 201
 - Mantisse, 199, 200
 - Normalisierung, 200
- Gleitkommazahldarstellung, 199–213
- GNU Projekt, 45
- Goldstine, Herman, 72
- Grossrechner, 56
- Großrechner, 62
- Halb-Addierer, 222
- Halbaddierer, 158
- Handheld, 57
- Hauptspeicher, 72, 73
- HENNESSY, JOHN L., 108
- Hexadezimalsystem, 149, 190–192
- Hexadezimalzah, 151
- Hexadezimalzahl, 136
- Home Computer, 57
- Host, 62
- I/O Device, 74
- IAB, 47
- IANA, 49
- IBM, 61, 64
- IBM System i, 61
- IBM-PC, 57
- IBM-System/390, 62
- ICANN, 49
- IEEE, 41, 200
- IEEE 1394, 43
- IEEE 754, 200
- IESG, 49
- IETF, 48
- Informatik, 7
 - Angewandte, 9
 - Praktische, 8
 - Technische, 8
 - Theoretische, 8
- Information, 2
- Information Retrieval, 22
- Informationssystem, 13
- Ingenieur-Informatik, 10
- Instruction Broadcasting, 101
- Instruction Counter, 80
- Instruction Register, 80
- Instruction Set Architecture, 105
- Integer Datentyp, 173
- International Telecommunication Union, 36
- Internet Architecture Board, 48
- Internet Engineering Task Force, 48
- Internet Society, 51
- Inverter, 216
- IPSec, 38
- IPv6, 48
- IRTF, 48

- ISBN, 139–141
 - Group Identifier, 139
 - Prüfziffer, 139
 - Publisher Prefix, 139
 - Title Identifier, 139
- ISDN, 38
- ISO, 39, 135
- ISO 10646, 135
- ISO 646, 135
- ISO 8859, 40, 135, 256
- ISO 9000, 40
- ISO 9660, 40, 256
- ISO-OSI Modell, 40
- ISOC, *siehe* Internet Society
- IT-Berater, 34
- ITU, 36
- ITU-D, 36
- ITU-R, 36
- ITU-T, 36
- Joint Photographics Experts Group, 39
- jpg-Dateiformat, 39
- Künstliche Intelligenz, 26
- KAHAN, WILLIAM, 200
- Karpor, Mitch, 46
- Kellerspeicher, 92
- Komparator, 248–250
- Komplement, 167
- Komplexitätstheorie, 9
- Kontrollstruktur, 74
- Latin-1, 135
- Leitwerk, 72
- Line Feed, 118
- Linux, 45
- Load/Store Architektur, 90
- Long Term Evolution, 39
- LTE, 39
- Mainframe, 13, 62
- Maschinenbefehl, 83
- Mauchly, John, 72
- Medien-Informatik, 12
- Medizinische Informatik, 11
- Memory Address Register, 81
- Metadaten, 5
- Metainformation, 5
- MicroSoft, 17
- Midrange Rechner, 59
- Midrange-Rechner, 60
- Mikro-Computer, 56
- Mikroprogrammierung, 104
- MIMD Architektur, 102
- Mini-Computer, 59
- Mini-Computer, 56
- Minuend, 168
- MIPS, 108
- MISD Architektur, 102
- MMX, *siehe* Multimedia Extension
- Modem, 38
- MORSE, SAMUEL, 123
- MORSE-Codierung, 119, 123
- Multimedia, 12
- Multimedia Extension, 101
- Multiplexbetrieb, 89
- Multiplexer, 243–248
- Multiprozessorsystem, 102
- n -Bit Codierung, 125
- Nachrichten, 6
- National Science Foundation, 47
- Neunerkomplement, 168
- NIBBLE, 260
- NIST, 44
- Non Uniform Memory Access, 102
- Notation
 - Infix, 95
 - Postfix, 95
 - umgekehrte polnische, 95
- Notebook, 14
- Nulladressbefehl, 93
- NUMA, *siehe* Non Uniform Memory Access
- Numbercruncher, 65
- ODBC, 19
- Oktalsystem, 149, 190
- OLAP, 21
- Online Analytical Processing, 21
- Opcode, 78, 83
- Open Source, 45

- Operandenteil, 83
- Operation
 - pop, 92
 - push, 92
- Operation Code, 83
- Oracle, 17
- Overflow Bit, 78
- Palmtop, 57
- Parallelverarbeitung, 100
- PATTERSON, DAVID, 108
- PCI Bus, 89
- Peripheriegeräte, 73
- Positionssystem, 148
- PowerPC, 113
- Programm, 74
- Projekt-Manager, 34
- Projektleiter, 34
- Proprietäre Systeme, 60
- Prozessor, 73
- QR-Codes, 142–143
- Realwissenschaft, 13
- Rechenanlage, speicherprogrammierbar, 30
- Rechenwerk, 72
- Rechnerarchitektur, 8, 71
- Register, 76
- Registerbreite, 76
- Rekursion, 92
- Request For Comments, 48
- RFC, 48
- Ripple Carry Adder, 87
- RISC, 59, 257
- RISC Prozessor, 77, 103
- Robotik, 27
- RS-232, 38
- RS/6000, 61, 113
- S/MIME, 38
- SAGE, 17
- SAP, 16
- Schaltnetz, 220
- Security Engineer, 34
- SEI, 46
- Semantik, 2
- Service Level Agreement, 25
- Sexagesimalsystem, 149
- Sieben-Segment Display, 233–241
- Siemens, 64
- Siemens BS2000, 62
- Sign Bit, 78
- SIMD Architektur, 101
- SISD Architektur, 100
- Software as a Service, 25
- Software Engineering Institute, 53
- Software-Ingenieur, 33
- Softwaretechnik, 8
- Sonderzeichen, 117
- SPARC Station, 61
- Speicheradressregister, 81
- Speicherprogrammierbar, 74
- Speicherwerk, 72
- Speicherzelle, 74
- Sprachverarbeitung, 26
- Sprungbefehl, 74
 - bedingter, 74
- SSL/TLS, 38
- Stack, 92
- Stack Basis, 94
- Stack Limit, 94
- Stack Pointer, 94
- Stacklänge, 92
- Stallman, Richard, 45
- Status Register, 78
- Stellenwertsystem, 147, 197
 - Basis, 147
- Steuerwerk, 72, 80
- Steuerzeichen, 118, 131
- Subtrahend, 168
- SUN, 61, 113
- Super-Computer, 56
- Supply Chain Management, 24
- Symmetric Multiprocessor, 102
- Syntax, 2
- System-Entwickler, 33
- Systems Engineer, 33
- Systemverwalter, 33
- Telemedizin, 11
- Textmining, 22

- Transistor, 215
- Turing Award, 45
- Turing, Alan, 45

- UCS, 135
- UMTS, 39
- Unicode, 135–137, 257
 - Codepunkt, 136
- Universalrechner, 72, 74
- UTF-16, 136
- UTF-8, 136

- V.24, 38
- V.90, 38
- VAX, 61
- Vektorrechner, 66, 101
- Verwaltungs-Informatik, 10
- Virtueller Marktplatz, 20
- Volladdierer, 225–231
- VON NEUMANN, JOHN, 71
- von Neumann Architektur, 71–76, 86, 117

- W3C, 52
- Webmining, 22
- WILKES, MAURICE, 105
- Windows NT, 113
- Wirtschaftsinformatik, 13–25
 - Aufgabenfelder, 15
 - Gegenstand, 13
 - Ziele, 15
- WORD, 260
- World Wide Web Consortium, 52
- Wort, 120

- X.25, 38
- X.400, 38
- X.500, 38
- X.509, 38
- XOR Verknüpfung, 170

- Zählcodierung, 124
- Zahldarstellung
 - Genauigkeit, 156
- Zahlensystem, 145
 - Additionssystem, 146–147
 - Basis, 117
 - dyadisches, 149
 - Hybridsystem, 147
 - polyadisches, 148
 - römisches, 147
 - Stellenwertsystem, 147–150
 - unäres, 146
- Zeichen, 3, 117
 - nicht druckbare, 118
- Zeichenvorrat, 4, 118
- Zentraleinheit, 73
- Zero Bit, 78
- Ziffer
 - hexadezimale, 191
- Zoll, 260
- Zweiadressbefehl, 83
- Zweiadressmaschine, 78
- Zweierkomplement, 171
- Zweierkomplementzahlen
 - Multiplikation, 180