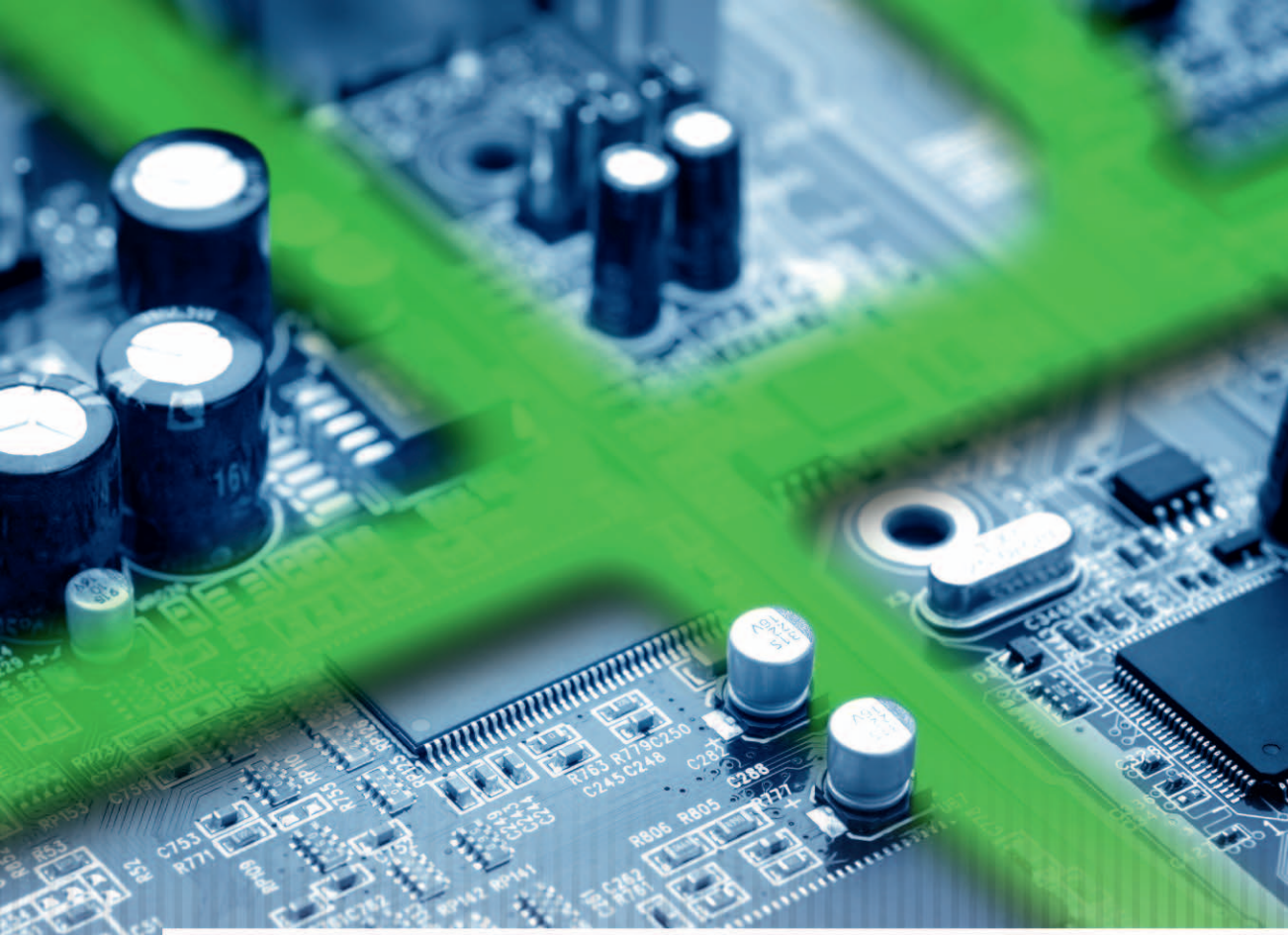

Rechnerarchitektur

Von der digitalen Logik zum Parallelrechner

6., aktualisierte Auflage

Andrew S. Tanenbaum
Todd Austin

Pearson

EXTRAS ONLINE

Rechnerarchitektur

Von der digitalen Logik zum Parallelrechner

6., aktualisierte Auflage

Andrew S. Tanenbaum
Todd Austin

ALWAYS LEARNING

PEARSON

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.dnb.de* abrufbar.

Die Informationen in diesem Buch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht ausgeschlossen werden.
Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Autor dankbar.

Authorized translation from the English language edition, entitled STRUCTURED COMPUTER ORGANIZATION, 6th Edition by ANDREW TANENBAUM; TODD AUSTIN, published by Pearson Education, Inc, publishing as Prentice Hall, Copyright © 2013
All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc. GERMAN language edition published by PEARSON EDUCATION DEUTSCHLAND GMBH, Copyright © 2014.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Es konnten nicht alle Rechteinhaber von Abbildungen ermittelt werden. Sollte dem Verlag gegenüber der Nachweis der Rechtsinhaberschaft geführt werden, wird das branchenübliche Honorar nachträglich gezahlt.

Fast alle Produktbezeichnungen und weitere Stichworte und sonstige Angaben, die in diesem Buch verwendet werden, sind als eingetragene Marken geschützt.
Da es nicht möglich ist, in allen Fällen zeitnah zu ermitteln, ob ein Markenschutz besteht, wird das ®-Symbol in diesem Buch nicht verwendet.

10 9 8 7 6 5 4 3 2 1

16 15 14

ISBN 978-3-86894-238-5 (Buch)
ISBN 978-3-86326-687-5 (E-Book)

© 2014 by Pearson Deutschland GmbH
Lilienthalstr. 2, D-85399 Hallbergmoos/Germany
Alle Rechte vorbehalten
www.pearson.de
A part of Pearson plc worldwide

Programmleitung: Birger Peil, bpeil@pearson.de
Übersetzung: Frank Langenau, Dresden
Fachlektorat: Prof. Dr. Michael Gerndt, TU München
Korrektorat: Katharine Pieper, Berlin
Herstellung: Claudia Bäurle, cbaeurle@pearson.de
Satz: Nadine Krumm, mediaService, Siegen (www.mediaservice.tv)
Coverdesign: Martin Horngacher, München
Coverabbildung: Fotolia.com
Druck und Verarbeitung: Drukarnia Dimograf, Bielsko-Biala

Printed in Poland

Inhaltsverzeichnis

Vorwort Originalausgabe 13

Vorwort zur deutschen Ausgabe 17

Kapitel 1 Einführung 19

1.1 Strukturierte Computerorganisation . . . 21
1.1.1 Sprachen, Ebenen und virtuelle Maschinen . . . 21
1.1.2 Moderne mehrschichtige Maschinen . . . 23
1.1.3 Evolution mehrschichtiger Maschinen . . . 26
Die Erfindung der Mikroprogrammierung . . . 27
Die Erfindung des Betriebssystems . . . 27
Verlagerung von Funktionen in den Mikrocode . . . 30
Beseitigung der Mikroprogrammierung . . . 31
1.2 Meilensteine der Computerarchitektur . . . 31
1.2.1 Die nullte Generation – Mechanische Computer (1642–1945) . . . 33
1.2.2 Die erste Generation – Vakuumröhren (1945–1955) . . . 35
1.2.3 Die zweite Generation – Transistoren (1955–1965) . . . 37
1.2.4 Die dritte Generation – integrierte Schaltungen (1965–1980) . . . 40
1.2.5 Die vierte Generation – VLSI (1980 – ?) . . . 41
1.2.6 Die fünfte Generation – leistungsarme und unsichtbare Computer . . . 44
1.3 Vielfalt der Computer . . . 46
1.3.1 Technologische und wirtschaftliche Kräfte . . . 46
1.3.2 Das Computerspektrum . . . 48
1.3.3 Wegwerfcomputer . . . 49
1.3.4 Mikrocontroller . . . 51
1.3.5 Mobile Computer und Spielkonsolen . . . 53
1.3.6 Personalcomputer . . . 54
1.3.7 Server . . . 55
Cluster . . . 55
1.3.8 Mainframes . . . 56
1.4 Beispiele von Computerfamilien . . . 57
1.4.1 Einführung in die x86-Architektur . . . 57
1.4.2 Einführung in ARM-Architektur . . . 62
1.4.3 Einführung in die AVR-Architektur . . . 64
1.5 Metrische Einheiten . . . 66
1.6 Gliederung dieses Buchs . . . 67

Kapitel 2 Aufbau von Computersystemen 73

2.1 Prozessoren . . . 75
2.1.1 Aufbau der CPU . . . 76
2.1.2 Befehlsausführung . . . 77
2.1.3 RISC kontra CISC . . . 81

2.1.4 Designprinzipien moderner Computer 82
Die Hardware führt alle Befehle direkt aus 83
Die Befehle werden mit maximaler Rate initiiert 83
Die Befehle müssen leicht zu decodieren sein 83
Nur Lade- und Speichervorgänge sollen auf den Speicher verweisen 84
Ausreichend Register bereitstellen 84
2.1.5 Parallelität auf Befehlsebene 84
Pipelining (Fließbandverarbeitung) 84
Superskalare Architekturen 86
2.1.6 Parallelität auf Prozessorebene 88
Datenparallele Rechner 89
Mehrprozessorsysteme 91
Multicomputersysteme 92
2.2 Hauptspeicher (Primärspeicher) 92
2.2.1 Bits. 93
2.2.2 Speicheradressen 93
2.2.3 Bytereihenfolge. 95
2.2.4 Fehlerkorrekturcodes 97
2.2.5 Cache-Speicher 100
2.2.6 Speichermodule und -typen. 103
2.3 Sekundärspeicher. 104
2.3.1 Speicherhierarchien 104
2.3.2 Festplatten. 105
2.3.3 IDE-Festplatten 109
2.3.4 SCSI-Festplatten 110
2.3.5 RAID 112
2.3.6 Solid-State-Disks. 115
2.3.7 CD-ROMs. 117
2.3.8 Einmal beschreibbare CDs 121
2.3.9 Wiederbeschreibbare CDs. 123
2.3.10 DVD 123
2.3.11 Blu-Ray 125
2.4 Eingabe/Ausgabe 126
2.4.1 Busse 126
Die PCI- und PCIe-Busse 128
2.4.2 Terminals 130
Tastaturen 130
Touchscreens 131
Flachbildschirme 132
Videospeicher 135
2.4.3 Mäuse 135
2.4.4 Gamecontroller 137
Der Wiimote-Controller 137
Der Kinect-Controller 139
2.4.5 Drucker 139
Laserdrucker 139
Farbdrucker 141

Tintenstrahldrucker 142
Spezialdrucker 143
2.4.6 Telekommunikationsgeräte 144
Modems 144
DSL 146
Internet über Kabel 149
2.4.7 Digitalkameras 152
2.4.8 Zeichencodes 155
ASCII 155
Unicode 157
2.4.9 UTF-8 159

Kapitel 3 Die Ebene der digitalen Logik 165

3.1 Gatter und boolesche Algebra 167
3.1.1 Gatter 167
3.1.2 Boolesche Algebra 169
3.1.3 Implementierung von booleschen Funktionen 171
3.1.4 Schaltungsäquivalenz 173
3.2 Grundschaltungen der digitalen Logik 176
3.2.1 Integrierte Schaltungen 176
3.2.2 Schaltnetze 178
Multiplexer 178
Decodierer 180
Komparatoren 181
3.2.3 Arithmetische Schaltungen 181
Schieberegister 181
Addierer 182
Rechenwerk 184
3.2.4 Taktgeber 185
3.3 Speicher 187
3.3.1 Latches 187
Getaktete SR-Latches 188
Getaktete D-Latches 189
3.3.2 Flipflops 189
3.3.3 Register 191
3.3.4 Speicherorganisation 192
3.3.5 Speicherchips 195
3.3.6 RAM und ROM 197
Nicht flüchtige Speicherchips 199
FPGA 200
3.4 CPU-Chips und Busse 202
3.4.1 Prozessorchips 202
3.4.2 Computer-Busse 204
3.4.3 Busbreite 206
3.4.4 Bustaktung 208
Synchrone Busse 208
Asynchrone Busse 211

3.4.5 Bus-Arbitration . . . 213
3.4.6 Busoperationen . . . 215
3.5 Beispiele für CPU-Chips . . . 218
3.5.1 Der Intel Core i7 . . . 218
Die logische Anschlussbelegung des Core i7 . . . 220
Fließbandverarbeitung auf dem DDR3-Speicherbus des Core i7 . . . 222
3.5.2 Das Ein-Chip-System Texas Instruments OMAP4430 . . . 224
3.5.3 Der Mikrocontroller Atmel ATmega168 . . . 228
3.6 Beispielbusse . . . 230
3.6.1 Der PCI-Bus . . . 231
Arbeitsweise des PCI-Busses . . . 234
PCI-Bus-Arbitration . . . 234
PCI-Bussignale . . . 235
PCI-Bustransaktionen . . . 238
3.6.2 PCI Express . . . 239
Die PCI-Express-Architektur . . . 239
Der PCI-Express-Protokollstapel . . . 241
3.6.3 USB (Universal Serial Bus) . . . 243
3.7 Schnittstellen . . . 247
3.7.1 E/A-Schnittstellen . . . 247
PIO-Schnittstellen . . . 248
3.7.2 Decodierung von Adressen . . . 249

Kapitel 4 Die Mikroarchitekturebene 259

4.1 Beispiel einer Mikroarchitektur . . . 261
4.1.1 Der Datenpfad . . . 262
Datenpfadzeiten . . . 264
Arbeitsweise des Speichers . . . 266
4.1.2 Mikrobefehle . . . 267
4.1.3 Mikrobefehlssteuerung: Mic-1 . . . 269
4.2 Eine Beispiel-ISA: IJVM . . . 274
4.2.1 Keller . . . 274
4.2.2 Das IJVM-Speichermodell . . . 276
4.2.3 Der IJVM-Befehlssatz . . . 278
4.2.4 Java zu IJVM kompilieren . . . 282
4.3 Beispielimplementierung . . . 283
4.3.1 Mikrobefehle und Notation . . . 283
4.3.2 IJVM mit der Mic-1 implementieren . . . 288
4.4 Design der Mikroarchitekturebene . . . 300
4.4.1 Geschwindigkeit und Kosten . . . 301
4.4.2 Die Länge des Ausführungspfads reduzieren . . . 303
Die Interpreter-Schleife mit dem Mikrocode verschmelzen . . . 303
Architektur mit drei Bussen . . . 305
Befehlsabrufeinheit (IFU) . . . 306
4.4.3 Design mit Prefetching: Mic-2 . . . 309

4.4.4 Fließbandverarbeitung: Mic-3 . . . 314
4.4.5 Siebenstufige Pipeline: Mic-4 . . . 319
4.5 Leistungsverbesserung . . . 322
4.5.1 Cache-Speicher . . . 323
Direkt abbildende Caches . . . 325
Mengenassoziative Caches . . . 327
4.5.2 Sprungvorhersage . . . 329
Dynamische Sprungvorhersage . . . 331
Statische Sprungvorhersage . . . 333
4.5.3 Out-of-Order-Ausführung und Registerumbenennung . . . 334
4.5.4 Spekulative Ausführung . . . 339
4.6 Beispiele der Mikroarchitekturebene . . . 342
4.6.1 Die Mikroarchitektur der Core-i7-CPU . . . 342
Die Sandy-Bridge-Mikroarchitektur des Core i7 . . . 342
Die Sandy-Bridge-Pipeline des Core i7 . . . 344
4.6.2 Die Mikroarchitektur der OMAP4430-CPU . . . 348
Übersicht über die Cortex-A9-Mikroarchitektur des OMAP4430 . . . 348
Die Cortex-A9-Pipeline des OMAP4430 . . . 350
4.6.3 Die Mikroarchitektur des Mikrocontrollers ATmega168 . . . 352
4.7 Core i7, OMAP4430 und ATmega168 im Vergleich . . . 354

Kapitel 5 Die ISA-Ebene 361

5.1 Übersicht über die ISA-Ebene . . . 364
5.1.1 Eigenschaften der ISA-Ebene . . . 364
5.1.2 Speichermodelle . . . 365
5.1.3 Register . . . 367
5.1.4 Befehle . . . 369
5.1.5 Übersicht über die ISA-Ebene des Core i7 . . . 369
5.1.6 Übersicht über die ISA-Ebene des ARM OMAP4430 . . . 372
5.1.7 Übersicht über die ISA-Ebene des AVR ATmega168 . . . 374
5.2 Datentypen . . . 376
5.2.1 Numerische Datentypen . . . 376
5.2.2 Nicht numerische Datentypen . . . 377
5.2.3 Datentypen des Core i7 . . . 378
5.2.4 Datentypen des OMAP4430 . . . 378
5.2.5 Datentypen der ATmega168 AVR-CPU . . . 379
5.3 Befehlsformate . . . 379
5.3.1 Designkriterien für Befehlsformate . . . 380
5.3.2 Opcodes erweitern . . . 382
5.3.3 Befehlsformate des Core i7 . . . 384
5.3.4 Befehlsformate der OMAP4430-ARM-CPU . . . 385
5.3.5 Die Befehlsformate des ATmega168 AVR . . . 387
5.4 Adressierung . . . 388
5.4.1 Adressierungsmodi . . . 388
5.4.2 Unmittelbare Adressierung . . . 388

5.4.3 Direkte Adressierung 389
5.4.4 Registeradressierung 389
5.4.5 Indirekte Registeradressierung 389
5.4.6 Indizierte Adressierung 391
5.4.7 Basisindizierte Adressierung 392
5.4.8 Kelleradressierung 392
Umgekehrte polnische Notation (Postfix-Notation) 392
Auswertung von Formeln in umgekehrter polnischer Notation 395
5.4.9 Adressierungsmodi für Verzweigungen 396
5.4.10 Orthogonalität von Opcodes und Adressierungsmodi 396
5.4.11 Adressierungsmodi des Core i7 398
5.4.12 Adressierungsmodi der OMAP4430-ARM-CPU 400
5.4.13 Adressierungsmodi des ATmega168 AVR 400
5.4.14 Adressierungsmodi im Vergleich 401
5.5 Befehlstypen 402
5.5.1 Befehle zur Datenbewegung 402
5.5.2 Dyadische Operationen 403
5.5.3 Monadische Operationen 404
5.5.4 Vergleiche und bedingte Verzweigungen 406
5.5.5 Befehle für Prozeduraufrufe 407
5.5.6 Schleifensteuerung 408
5.5.7 Ein-/Ausgabe 410
5.5.8 Befehle des Core i7 413
5.5.9 Befehle des OMAP4430 417
5.5.10 Befehle des ATmega168 AVR 421
5.5.11 Befehlssätze im Vergleich 423
5.6 Ablaufsteuerung 424
5.6.1 Sequenzielle Ablaufsteuerung und Sprünge 424
5.6.2 Prozeduren 425
5.6.3 Coroutinen 429
5.6.4 Traps 432
5.6.5 Interrupts 432
5.7 Ausführliches Beispiel: Die Türme von Hanoi 436
5.7.1 Türme von Hanoi in der Core-i7-Assemblersprache 436
5.7.2 Türme von Hanoi in der OMAP4430-ARM-Assemblersprache . . 438
5.8 Die IA-64-Architektur und der Itanium 2 440
5.8.1 Das Problem mit der IA-32-ISA 440
5.8.2 Das IA-64-Modell: EPIC 442
5.8.3 Verringerung von Speicherreferenzen 442
5.8.4 Befehlsplanung 444
5.8.5 Bedingte Verzweigungen verringern: bedingte Befehlsausführung 445
5.8.6 Spekulative Ladeoperationen 448

Kapitel 6 Die Ebene der Betriebssystemmaschine 455

6.1 Virtueller Speicher . . . 458
6.1.1 Seitenauslagerung . . . 458
6.1.2 Implementierung der Seitenauslagerung . . . 460
6.1.3 Seitenanforderung und Arbeitsbereich . . . 464
6.1.4 Seitenersetzung . . . 465
6.1.5 Seitengröße und Fragmentierung . . . 467
6.1.6 Segmentierung . . . 468
6.1.7 Implementierung der Segmentierung . . . 471
6.1.8 Virtueller Speicher im Core i7 . . . 473
6.1.9 Virtueller Speicher in der OMAP4430 ARM-CPU . . . 478
6.1.10 Virtueller Speicher und Caching . . . 480
6.2 Hardwarevirtualisierung . . . 480
6.2.1 Hardwarevirtualisierung beim Core i7 . . . 482
6.3 E/A-Befehle auf OSM-Ebene . . . 482
6.3.1 Dateien . . . 483
6.3.2 Implementierung von E/A-Befehlen auf OSM-Ebene . . . 485
6.3.3 Befehle für die Verzeichnisverwaltung . . . 488
6.4 Befehle der OSM-Ebene für parallele Verarbeitung . . . 489
6.4.1 Erstellen von Prozessen . . . 490
6.4.2 Race Conditions . . . 491
6.4.3 Semaphore zur Synchronisierung von Prozessen . . . 495
6.5 Beispiele von Betriebssystemen . . . 498
6.5.1 Einführung . . . 498
Unix . . . 499
Windows 7 . . . 502
6.5.2 Beispiele für virtuellen Speicher . . . 505
Virtueller Speicher unter Unix . . . 505
Virtueller Speicher unter Windows 7 . . . 506
6.5.3 Beispiele für Ein-/Ausgabe auf Betriebssystemebene . . . 508
Ein-/Ausgabe unter Unix . . . 509
Ein-/Ausgabe unter Windows 7 . . . 515
6.5.4 Beispiele der Prozessverwaltung . . . 520
Prozessverwaltung unter Unix . . . 520
Prozessverwaltung unter Windows 7 . . . 523

Kapitel 7 Die Ebene der Assemblersprache 535

7.1 Einführung in die Assemblersprache . . . 537
7.1.1 Was ist eine Assemblersprache? . . . 537
7.1.2 Wofür braucht man eine Assemblersprache? . . . 538
7.1.3 Format einer Anweisung in Assemblersprache . . . 538
7.1.4 Pseudobefehle . . . 540
7.2 Makros . . . 543
7.2.1 Definition, Aufruf und Erweiterung eines Makros . . . 543
7.2.2 Makros mit Parametern . . . 545
7.2.3 Weitere Eigenschaften . . . 546
7.2.4 Makrotechnik in einem Assemblierer implementieren . . . 546

7.3 Der Assemblierungsprozess . . . 547
7.3.1 Zwei-Pass-Assemblierer . . . 547
7.3.2 Erster Lauf . . . 548
7.3.3 Zweiter Lauf . . . 552
7.3.4 Die Symboltabelle . . . 553
7.4 Binden und Laden . . . 555
7.4.1 Aufgaben des Linkers . . . 556
7.4.2 Struktur eines Objektmoduls . . . 559
7.4.3 Bindezeit und dynamische Relokation . . . 560
7.4.4 Dynamisches Binden . . . 563
Dynamisches Binden in MULTICS . . . 563
Dynamisches Binden unter Windows . . . 564
7.4.5 Dynamisches Binden unter Unix . . . 566

Kapitel 8 Parallele Rechnerarchitekturen 571

8.1 Parallelität auf demselben Chip . . . 574
8.1.1 Parallelität auf Befehlsebene . . . 574
Die VLIW-CPU TriMedia . . . 575
8.1.2 On-Chip-Multithreading . . . 581
Hyperthreading im Core i7 . . . 583
8.1.3 Ein-Chip-Multiprozessoren . . . 587
Homogene Multiprozessoren auf einem Chip . . . 587
Der Core-i7-Ein-Chip-Multiprozessor . . . 588
Heterogene Multiprozessoren auf einem Chip . . . 588
8.2 Coprozessoren . . . 592
8.2.1 Netzwerkprozessoren . . . 593
Einführung in Netzwerke . . . 593
Einführung in Netzwerkprozessoren . . . 595
Paketverarbeitung . . . 598
Leistungsverbesserung . . . 599
8.2.2 Grafikprozessoren . . . 600
Die Fermi-GPU von NVIDIA . . . 600
8.2.3 Kryptoprozessoren . . . 603
8.3 Multiprozessoren mit gemeinsamem Speicher . . . 603
8.3.1 Multiprozessoren und Multicomputer . . . 603
Multiprozessoren . . . 604
Multicomputer . . . 605
Klassifizierung von Parallelrechnern . . . 608
8.3.2 Speichersemantik . . . 611
Strikte Konsistenz . . . 611
Sequenzielle Konsistenz . . . 611
Prozessorkonsistenz . . . 612
Schwache Konsistenz . . . 613
Freigabekonsistenz . . . 614
8.3.3 Symmetrische UMA-Multiprozessorarchitekturen . . . 615
Snooping-Caches . . . 616
Das Cache-Kohärenzprotokoll MESI . . . 618

UMA-Multiprozessoren mit Kreuzschienenverteilern 620
UMA-Multiprozessoren mit mehrstufigen Netzwerken 621
8.3.4 NUMA-Multiprozessoren 623
Cachekohärente NUMA-Multiprozessoren 624
Der NUMA-Multiprozessor Sun Fire E25K 627
8.3.5 COMA-Multiprozessorsysteme 631
8.4 Multicomputer mit Nachrichtenaustausch 633
8.4.1 Verbindungsnetze 634
Topologie 634
8.4.2 MPPs – Massiv parallele Prozessorsysteme 637
BlueGene 638
Red Storm 642
BlueGene/P und Red Storm im Vergleich 645
8.4.3 Cluster-Rechner 647
Google 648
8.4.4 Kommunikationssoftware für Multicomputer 652
MPI – Message-Passing Interface 653
8.4.5 Scheduling 655
8.4.6 Gemeinsamer Speicher auf Anwendungsebene 656
DSM – Distributed Shared Memory 656
Linda 658
Orca 660
8.4.7 Leistung 662
Hardware-Kenngrößen 662
Software-Kenngrößen 664
Hohe Leistung erreichen 666
8.5 Grid-Computing 668

Literaturverzeichnis 675

Anhang A Binärzahlen 683

A.1 Zahlen mit endlicher Genauigkeit 685
A.2 Zahlensysteme mit anderen Basen 686
A.3 Umwandlung von einer Basis in eine andere 689
A.4 Negative Binärzahlen 691
A.5 Binärarithmetik 693

Anhang B Gleitkommazahlen 697

B.1 Grundlagen der Gleitkommaarithmetik 699
B.2 IEEE-Standard 754 für Gleitkommazahlen 702

Anhang C Assemblerprogrammierung 709

C.1 Überblick 711
C.1.1 Assemblersprache 711
C.1.2 Ein kleines Assemblerprogramm 711

C.2 Der 8088-Prozessor . . . 712
C.2.1 Der Prozessorzyklus . . . 713
C.2.2 Die allgemeinen Register . . . 714
C.2.3 Zeigerregister . . . 715
C.3 Speicher und Adressierung . . . 717
C.3.1 Speicherorganisation und Segmente . . . 717
C.3.2 Adressierung . . . 719
C.4 Der Befehlssatz des 8088 . . . 722
C.4.1 Lade-, Kopier und Arithmetikbefehle . . . 725
C.4.2 Logische Operationen, Bit- und Verschiebeoperationen . . . 726
C.4.3 Schleifen und wiederholte Zeichenfolgenoperationen . . . 727
C.4.4 Sprung- und Aufrufbefehle . . . 728
Bedingte Sprünge . . . 728
C.4.5 Unterprogrammaufrufe . . . 730
C.4.6 Systemaufrufe und Systemunterprogramme . . . 731
C.4.7 Abschließende Bemerkungen zum Befehlssatz . . . 734
C.5 Der Assemblierer . . . 734
C.5.1 Einführung . . . 734
C.5.2 Der ACK-basierte Assemblierer as88 . . . 735
C.5.3 Einige Unterschiede zu anderen 8088-Assemblierern . . . 739
C.6 Der Tracer . . . 740
C.6.1 Tracer-Kommandos . . . 742
C.7 Erste Schritte . . . 743
C.8 Beispiele . . . 744
C.8.1 Das Beispiel Hello World . . . 744
C.8.2 Beispiel für allgemeine Register . . . 747
C.8.3 Der `CALL`-Befehl und Zeigerregister . . . 748
C.8.4 Fehler in einem Programm zur Ausgabe von Feldern suchen . . . 752
C.8.5 Zeichenfolgenmanipulation und Zeichenfolgenbefehle . . . 754
C.8.6 Verzweigungstabellen . . . 758
C.8.7 Gepufferter und wahlfreier Dateizugriff . . . 760
Danksagungen . . . 765

Namensregister 767

Register 771

Vorwort Originalausgabe

Die ersten fünf Auflagen dieses Buchs basierten auf der Vorstellung, dass sich ein Computer als Hierarchie betrachten lässt, deren Ebenen jeweils eine genau definierte Funktion erfüllen. Dieses grundlegende Konzept ist heute genauso gültig wie bei der Erstauflage dieses Buchs. Deshalb wurde es auch in dieser sechsten Auflage beibehalten. Wie in den ersten fünf Auflagen werden die Ebene der digitalen Logik, die Ebene der Mikroarchitektur, die ISA-Ebene (d.h. die Ebene der Befehlssatzarchitektur), die Ebene der Betriebssystemmaschine und die Ebene der Assemblersprache ausführlich behandelt.

Obwohl die Grundstruktur gleich geblieben ist, weist die sechste Auflage viele kleine und große Änderungen auf, die sich zwangsläufig durch den schnellen Wandel in der Computerindustrie ergeben haben. So wurden alle Beispielmaschinen bzw. Prozessoren in diesem Buch dem aktuellen Stand entsprechend gewählt – Intel Core i7, Texas Instruments OMAP4430 und Atmel ATmega168. Der Core i7 ist ein Beispiel für eine bei Laptop- und Desktopcomputern sowie Servern weit verbreitete CPU. Der OMAP4430 liefert ein Beispiel für eine verbreitete ARM-basierte CPU, die man vor allem in Smartphones und Tablet-PCs findet.

Auch wenn Ihnen der Mikrocontroller ATmega168 unbekannt sein dürfte, sicherlich haben Sie schon oft damit interagiert. Der AVR-basierte Mikrocontroller ATmega168 ist in vielen eingebetteten Systemen zu finden – von Uhrenradios bis hin zu Mikrowellenherden. Eingebettete Systeme sind stark im Kommen. Die Wahl fällt dabei häufig auf den weit verbreiteten ATmega168, was mit seinem äußerst geringen Preis (der im Cent-Bereich liegt), dem reichhaltigen Angebot an Software und Peripheriebauelementen sowie der riesigen Schar von Programmierern zusammenhängt. Die Anzahl der weltweit verbauten ATmega168-Controller übersteigt die Anzahl der Core i3-, i5- und i7-CPUs um mehrere Größenordnungen. Zudem ist der ATmega168 im Einkartenrechner Arduino enthalten, einem bekannten Amateur-System, das von einer italienischen Universität konzipiert wurde und weniger als ein Mittagessen in einem Pizza-Restaurant kosten sollte.

In den letzten Jahren haben viele Dozenten wiederholt nach Material für die Programmierung in Assemblersprache gefragt. Mit der sechsten Auflage steht dieses Material nun auf der buchbegleitenden Website zur Verfügung, wo es sich problemlos erweitern und auf dem neuesten Stand halten lässt. Als Basis wurde die Assemblersprache für den Prozessor 8088 gewählt, da sie letztlich eine abgespeckte Version des stark verbreiteten IA32-Befehlssatzes darstellt, den auch der Prozessor Core i7 verwendet. Wir hätten uns auch für den Befehlssatz der ARM- oder AVR-Prozessoren oder einer anderen ISA-Architektur entscheiden können, die fast niemandem bekannt ist, doch stellt der 8088 im Hinblick auf die Motivation eine bessere Wahl dar, da viele Studenten einen Rechner mit einer 8088-kompatiblen CPU zu Hause stehen haben. Der vollständige Befehlssatz des Core i7 ist für Studenten viel zu komplex, um ihn im Detail verstehen zu können. Der 8088 ist ähnlich, aber viel einfacher.

EXTRAS ONLINE
Lösungen

Darüber hinaus ist der Core i7, mit dem sich diese Ausgabe des Buchs ausführlich beschäftigt, in der Lage, 8088-Programme auszuführen. Da allerdings das Debuggen von Assemblercode recht schwierig ist, haben wir einen Satz von Werkzeugen zusammengestellt, der die Beschäftigung mit der Assemblersprache erleichtern soll. Hierzu gehören ein Assemblierer, ein Simulator und ein Tracer für den 8088. Diese Werkzeuge sind für Windows, UNIX und Linux auf der Website verfügbar.

Der Umfang dieses Buchs hat mit den Jahren zugenommen – gegenüber der ersten Ausgabe des Buchs ist die Seitenanzahl fast doppelt so groß. Solche Erweiterungen sind natürlich unvermeidbar, wenn sich ein Gebiet weiterentwickelt. Das kann allerdings auch bedeuten, dass das Buch in einem einzigen Kurs (sprich Semester) nicht ganz durchgearbeitet werden kann. Ein möglicher Ansatz ist es, die Kapitel 1, 2 und 3 sowie den ersten Teil von Kapitel 4 (bis einschließlich Abschnitt 4.4) und Kapitel 5 als absolutes Minimum in einen Kurs zu packen. Soweit noch Zeit verbleibt, kann der Rest von Kapitel 4 mit Teilen der Kapitel 6, 7 und 8 in Angriff genommen werden. Dies hängt natürlich von den Interessen des Dozenten und der Studenten ab.

Die folgenden Absätze beschreiben kurz die wesentlichen Änderungen seit der fünften Ausgabe. **Kapitel 1** enthält immer noch einen Überblick über die Geschichte von Rechnerarchitekturen und betont dabei, wie wir den heutigen Stand der Technik erreicht und welche Höhepunkte wir auf dem Weg dorthin erlebt haben. Für viele Studenten dürfte es überraschend sein zu erfahren, dass die weltweit leistungsfähigsten Computer der 1960er Jahre, die Millionen von US-Dollar kosteten, weit weniger als 1 Prozent der Rechenleistung moderner Smartphones erreichten. Wir gehen auch auf das vergrößerte Spektrum heutiger Computer ein – unter anderem FPGAs, Smartphones, Tablet-PCs und Spielkonsolen. Außerdem führt das Kapitel in die drei neuen Beispielarchitekturen ein (Core i7, OMAP4430 und ATmega168).

In **Kapitel 2** wurde der Stoff zu den Verarbeitungsarten erweitert und berücksichtigt jetzt auch Parallelprozessoren einschließlich Grafikprozessoren (Graphic Processing Units, GPUs). Die Palette der Speichertechnologien wurde ebenfalls erweitert, um der zunehmenden Popularität Flash-basierter Speicherbausteine Rechnung zu tragen. Neuer Stoff ist auch in den Abschnitt zur Ein-/Ausgabe eingeflossen. Wir gehen ausführlich auf moderne Spielcontroller ein, einschließlich der Wii-Fernbedienung (kurz Wiimote) und der Kinect sowie der Touchscreens, die in Smartphones und Tablet-PCs eingesetzt werden.

Kapitel 3 hat an mehreren Stellen eine Überarbeitung erfahren. Gleich geblieben ist der Anfang des Kapitels, der die Funktionsweise von Transistoren erläutert. Anhand der hier vermittelten Grundlagen sind selbst Studenten ohne Hintergrundwissen zur Hardware in der Lage, die Arbeitsprinzipien eines modernen Computers zu verstehen. Neu sind die Abschnitte zu FPGAs (Field Programmable Gate Arrays). Diese programmierbaren Gatteranordnungen bringen echte Einsparungen bei den Entwurfskosten auf Gatterebene, sodass sie heute auch im Unterricht verbreitet sind. Die drei neuen Beispielarchitekturen werden hier im Überblick beschrieben.

Kapitel 4 hat schon von Anfang an beschrieben, wie ein Computer tatsächlich funktioniert, sodass der größte Teil seit der fünften Ausgabe unverändert geblieben ist. Allerdings gibt es einige neue Abschnitte, die sich mit der Mikroarchitekturebene der Prozessoren Core i7, OMAP4430 und ATmega168 beschäftigen.

Die **Kapitel 5** und **6** wurden im Hinblick auf die neuen Beispielarchitekturen überarbeitet. Insbesondere enthalten sie neue Abschnitte, die die ARM- und AVR-Befehlssätze beschreiben. **Kapitel 6** verwendet nun Windows 7 statt Windows XP als Beispiel.

Kapitel 7 zur Programmierung in Assemblersprache ist gegenüber der fünften Ausgabe weitgehend unverändert geblieben.

Kapitel 8 hat viele Überarbeitungen erfahren, um neue Entwicklungen auf dem Gebiet der Parallelcomputer widerzuspiegeln. Dazu gehören ausführliche Erläuterungen zur Multiprozessor-Architektur des Core i7 und zur Architektur des Allzweck-Grafikprozessors *Fermi* von NVIDIA. Schließlich wurden die Abschnitte zu den Supercomputern BlueGene und

Red Storm aktualisiert, um die jüngsten Weiterentwicklungen dieser gewaltigen Maschinen darzustellen.

Kapitel 9 hat sich geändert. Die empfohlene Literatur ist auf die Website umgezogen, sodass das neue Kapitel 9 nur die im Buch angeführten Verweise enthält, von denen viele neu sind. Die Computerarchitektur ist ein dynamisches Gebiet.

Die **Anhänge A** und **B** sind seit der letzten Auflage unverändert geblieben. Binärzahlen und Gleitkommazahlen haben in den letzten Jahren keinen Wandel durchgemacht. Anhang C zur Programmierung in Assemblersprache wurde von Dr. Evert Wattel von der Vrije Universiteit Amsterdam geschrieben. Dr. Wattel besitzt langjährige Erfahrungen bei der Unterrichtung von Studenten mit diesen Werkzeugen. Für das Verfassen dieses Anhangs gilt ihm unser besonderer Dank. Der Inhalt ist gegenüber der fünften Ausgabe weitgehend gleich geblieben, die Werkzeuge finden Sie nun aber auf der Website statt auf einer zum Buch gehörenden CD.

Neben den Werkzeugen für die Assemblerprogrammierung finden Sie auf der Website auch einen grafischen Simulator, den Sie in Verbindung mit Kapitel 4 einsetzen können. Diesen Simulator hat Prof. Richard Salter vom Oberlin College geschrieben. Studenten können sich mit diesem Simulator die in diesem Kapitel erläuterten Prinzipien besser klar machen. Prof. Salter gebührt großer Dank, dass er diese Software bereitgestellt hat.

Die Website mit den genannten Tools und anderem Material ist unter

http://www.pearsonhighered.com/tanenbaum

zu finden. Klicken Sie dort auf die Website für dieses Buch und wählen Sie die gesuchte Seite aus. Für Studenten sind die folgenden Ressourcen verfügbar:

- die Assembler-/Tracer-Software
- der grafische Simulator
- die empfohlene Literatur

Dozenten finden auf der Website unter anderem folgende Ressourcen:

- PowerPoint-Folien für den Kurs
- Lösungen für die am Ende jedes Kapitels enthaltenen Übungen

Für den Zugriff auf die Ressourcen für Dozenten ist ein Kennwort erforderlich. Dozenten sollten diesbezüglich ihren Vertreter von Pearson Education konsultieren.

Viele haben dieses Manuskript (zumindest teilweise) gelesen und nützliche Vorschläge unterbreitet oder auf andere Weise bei diesem Buch geholfen. Unser besonderer Dank gilt Anna Austin, Mark Austin, Livio Bertacco, Valeria Bertacco, Debapriya Chatterjee, Jason Clemons, Andrew DeOrio, Joseph Greathouse und Andrea Pellegrini.

Für die kritische Durchsicht des Manuskripts und die vorgeschlagenen Änderungen danken wir Jason D. Bakos (University of South Carolina), Bob Brown (Southern Polytechnic State University), Andrew Chen (Minnesota State University, Moorhead), J. Archer Harris (James Madison University), Susan Krucke (James Madison University), A. Yavuz Oruc (University of Maryland), Frances Marsh (Jamestown Community College) und Kris Schindler (University at Buffalo).

Byron A. Jeff (Clayton University), Laura W. McFall (DePaul University), Taghi M. Mostafavi (University of North Carolina at Charlotte) und James Nystrom (Ferris State University) haben uns dankenswerterweise geholfen, neue Übungen zu erstellen.

Unser Herausgeber Tracy Johnson ist in vielerlei Hinsicht immer hilfreich gewesen, sowohl im Großen wie auch im Kleinen, und er hat zudem viel Geduld mit uns bewiesen. Carole Snyder ist es hervorragend gelungen, die verschiedenen Projektbeteiligten zu koordinieren. Bob Englehardt hat die Produktion des Buchs großartig gemeistert.

Ich (AST) danke Suzanne wieder einmal für ihre Liebe und Geduld, die auch nach 21 Büchern nicht endet. Barbara und Marvin sind immer eine Freude für mich und sie wissen jetzt, womit Professoren ihren Lebensunterhalt verdienen. Aron gehört zur nächsten Generation: Kinder, die Computer schon ausgiebig nutzen, noch bevor sie in den Kindergarten gehen. Nathan ist noch nicht so weit, doch wenn er erst einmal Laufen gelernt hat, dürfte er sich als Nächstes mit dem iPad anfreunden.

Schließlich möchte ich (TA) diese Gelegenheit wahrnehmen, um meiner Schwiegermutter Roberta zu danken, die mir den Rücken für die Arbeit an diesem Buch freigehalten hat. Ihr Esszimmer in Bassano Del Grappa, Italien, bot gerade das richtige Maß an Abgeschiedenheit, Obdach und Wein, um diese wichtige Aufgabe zu bewältigen.

Andrew S. Tanenbaum

Todd Austin

AST: Suzanne, Barbara, Marvin, Aron und Nathan

TA: Für Roberta, die mir für dieses Projekt Raum (und Zeit) verschafft hat.

Vorwort zur deutschen Ausgabe

Die vorliegende erweiterte und aktualisierte 6. Ausgabe des Standardwerks von Andrew S. Tanenbaum gibt einen exzellenten Einblick in die Architektur und Organisation von Rechensystemen. Sie umfasst sowohl sequentielle als auch parallele Architekturen und deckt somit das gesamte Spektrum heutiger Rechner ab.

Der Bereich der sequentiellen Architekturen mit den grundlegenden Konzepten und Techniken wurde um aktuelle Prozessoren ergänzt. Es werden die Prozessoren Intel Core i7, Texas Instruments OMAP4430 und Atmel ATmega168 vorgestellt. Für alle Bereiche von der Mikroarchitektur bis zur Assemblerprogrammierung werden diese Prozessoren durchgehend als Beispiele eingesetzt.

Das Kapitel über Parallelrechner wurde aktualisiert und erweitert. Die vorgestellten großen parallelen Systeme wie BlueGene/P werden auch in europäischen Rechenzentren im Supercomputing eingesetzt. Als Beispiel für den Einsatz von Clustern wird die Verarbeitung von Google-Anfragen in Googles großen Datenzentren vorgestellt.

Hervorzuheben ist auch die umfassende Darstellung der Peripheriegeräte. Sie stellt die aktuellen Technologien von Festplatten, CD- und DVD-Speichern als auch von Bildschirmen und Druckern vor. Das Kapitel enthält zusätzlich eine kurze aber sehr aufschlussreiche Einführung in DSL und Internet über Kabelnetze.

Besonders empfehlenswert ist das Buch, da es über den typischen Inhalt einer einsemestrigen Rechnerarchitekturvorlesung hinausgeht. Es zeigt nicht nur die eigentliche Architektur der Rechner auf, sondern enthält auch weitergehendes Material, das einen guten Gesamtüberblick über die Aspekte der Technischen Informatik vermittelt. So werden zusätzlich die Ebenen der digitalen Bausteine, des Betriebssystems und die Assemblerprogrammierung besprochen.

Die deutsche Übersetzung wurde im Vergleich zur Übersetzung der 5. Auflage vollständig überarbeitet. Hierbei wurde auf die Verwendung etablierter deutscher Fachbegriffe geachtet, dort wo diese nicht vorhanden sind, wurden aber die englischen Fachbegriffe verwendet. Der ausführliche Index sollte jedem ermöglichen, sowohl über den englischen als auch den deutschen Fachbegriff das Thema schnell aufzufinden. Die neu eingeführten Randbemerkungen sollen einen Überblick über den Inhalt geben und so dem Leser einen leichteren Einstieg und eine gezielte Auffrischung seines Wissens ermöglichen.

Das Buch ist sehr gut als begleitendes Buch für Vorlesungen im Bereich Rechnerarchitektur und Rechnerorganisation geeignet. Durch Tanenbaums gut lesbaren zuweilen auch unterhaltsamen Schreibstil, der in der deutsche Übersetzung exzellent erhalten ist, ist das Buch aber auch allen Studenten und anderen Interessierten zur selbständigen Lektüre zu empfehlen.

Michael Gerndt

Einführung

1

ÜBERBLICK

1.1 Strukturierte Computerorganisation 21
1.2 Meilensteine der Computerarchitektur 31
1.3 Vielfalt der Computer 46
1.4 Beispiele von Computerfamilien 57
1.5 Metrische Einheiten 66
1.6 Gliederung dieses Buchs 67

Einleitung

» Ein Computer oder Digitalrechner ist eine Maschine, die Probleme für den Menschen lösen kann, indem sie die ihr gegebenen Befehle ausführt. Eine Befehlssequenz, die beschreibt, wie eine bestimmte Aufgabe auszuführen ist, nennt man **Programm**. Die elektronischen Schaltungen eines Computers verstehen eine begrenzte Menge einfacher Befehle, in die alle Programme konvertiert werden müssen, bevor sie sich ausführen lassen. Diese grundlegenden Befehle sind selten komplizierter als beispielsweise:

- Addiere zwei Zahlen.
- Prüfe eine Zahl, um festzustellen, ob sie null ist.
- Kopiere ein Datenelement von einem Teil des Computerspeichers in einen anderen.

Die Gesamtheit der elementaren Befehle eines Computers bildet die sogenannte **Maschinensprache** (Machine Language), in der der Mensch mit dem Computer kommunizieren kann. Für einen neu zu konzipierenden Computer müssen die Entwickler entscheiden, welche Befehle in der Maschinensprache realisiert werden sollen. Normalerweise versucht man, die elementaren Befehle so einfach wie möglich zu halten. Dabei geht man vom vorgesehenen Einsatzgebiet des Computers und von konkreten Leistungsanforderungen aus, damit sich die Komplexität und die Kosten der erforderlichen Elektronik in Grenzen halten. Da die meisten Maschinensprachen sehr einfach sind, ist es für den Menschen schwierig und umständlich, mit ihnen zu arbeiten.

Dies hat im Laufe der Zeit dazu geführt, Computer als Folge von Abstraktionen zu strukturieren, wobei jede Abstraktion auf der jeweils darunterliegenden aufbaut. Somit bekommt man die Komplexität in den Griff und kann Computersysteme systematisch und strukturiert entwickeln. Dieser Ansatz wird **strukturierte Computerorganisation** genannt. Der nächste Abschnitt erläutert, was dieser Begriff bedeutet. Danach sehen wir uns einige historische Entwicklungen, den Stand der Technik und wichtige Beispiele an. «

1.1 Strukturierte Computerorganisation

Wie oben festgestellt, besteht zwischen dem, was für Menschen, und dem, was für Computer bequem ist, ein großer Unterschied. Der Mensch möchte *X* tun, der Computer beherrscht aber nur *Y*. Das führt zu einem Problem, dessen Lösung in diesem Buch aufgezeigt werden soll.

1.1.1 Sprachen, Ebenen und virtuelle Maschinen

Das Problem kann auf zweierlei Weise angegangen werden. Beide Methoden basieren auf dem Design eines neuen Befehlssatzes, den Menschen leichter nutzen können als die von der Hardware realisierten Maschinenbefehle. Insgesamt bilden diese neuen Befehle eine Sprache, die wir L1 nennen, so wie die integrierten Maschinenbefehle eine Sprache darstellen, die L0 heißen soll. Die beiden Ansätze unterscheiden sich darin, wie der Computer die in L1 geschriebenen Programme bearbeitet. Der Computer kann ja nur Programme ausführen, die in seiner Maschinensprache – L0 – geschrieben sind.

Bei der ersten Methode, ein in L1 geschriebenes Programm auszuführen, wird jeder Befehl dieses Programms durch eine äquivalente Sequenz von Befehlen in L0 ersetzt. Das daraus resultierende Programm besteht gänzlich aus L0-Befehlen. Der Computer führt dann das neue L0-Programm und nicht das alte L1-Programm aus. Diese Technik heißt **Übersetzung**.

Bei der zweiten Methode wird ein Programm in L0 geschrieben, das in L1 geschriebene Programme als Eingabedaten übernimmt und sie ausführt, indem es alle Befehle nacheinander untersucht und direkt als äquivalente Folge von L0-Befehlen ausführt. Hierbei ist es nicht erforderlich, zuerst ein neues Programm in L0 zu erzeugen. Diese Technik heißt **Interpretation** und das Programm, das L1 ausführt, wird **Interpreter** genannt.

Übersetzung und Interpretation ähneln sich. Bei beiden Methoden verarbeitet der Computer die Befehle in L1, indem er äquivalente Sequenzen in L0 ausführt. Der Unterschied besteht darin, dass bei der Übersetzung zuerst das gesamte Programm in ein L0-Programm konvertiert, das L1-Programm verworfen und dann das neue L0-Programm in den Computerspeicher geladen und ausgeführt wird. Zur Ausführungszeit läuft das neu generierte L0-Programm im Computer und steuert diesen.

Programme in einer höheren Programmiersprache können in Maschinenprogramme **übersetzt** werden oder durch ein Maschinenprogramm **interpretiert** und so ausgeführt werden.

Bei der Interpretation wird jede L1-Anweisung untersucht, decodiert und unmittelbar ausgeführt. Es wird kein übersetztes Programm generiert. Hier übernimmt der Interpreter die Steuerung des Computers. Das L1-Programm ist lediglich eine Menge von Daten. Beide Methoden sind weit verbreitet und werden in zunehmendem Maße kombiniert angewendet.

Anstatt in den Kategorien von Übersetzung und Interpretation zu denken, ist es oftmals einfacher, sich einen hypothetischen Computer oder eine **virtuelle Maschine** vorzustellen, deren Maschinensprache L1 ist. Wir nennen diese virtuelle Maschine M1 (und die virtuelle Maschine, die L0 entspricht, M0). Ließe sich eine solche Maschine kostengünstig produzieren, dann wäre eine Sprache L0 oder eine Maschine, die Programme in L0 ausführt, überhaupt nicht notwendig. Man schreibt die Programme einfach in L1 und lässt sie vom Computer direkt ausführen. Selbst wenn es zu teuer oder zu kompliziert ist, die virtuelle L1-Maschine in Form konkreter Elektronik zu realisieren, kann man dennoch Programme dafür schreiben. Diese Programme lassen sich von einem in L0

geschriebenen Programm, das ein existierender Computer direkt ausführen kann, entweder interpretieren oder übersetzen. Mit anderen Worten kann man Programme für virtuelle Maschinen schreiben, als würden solche Maschinen wirklich existieren.

Damit die Übersetzung oder Interpretation realisierbar ist, dürfen sich die Sprachen L0 und L1 nicht „zu sehr" unterscheiden. Diese Einschränkung bedeutet oftmals, dass L1 zwar besser ist als L0, für die meisten Anwendungen aber noch lange nicht dem Ideal entspricht. Dieses Ergebnis ist vielleicht entmutigend, wenn man bedenkt, was das Ziel von L1 war – den Programmierer davon zu befreien, dass er Algorithmen in einer Sprache ausdrücken muss, die eher für Maschinen als für Menschen geeignet ist. Die Situation ist aber nicht hoffnungslos.

Es liegt nahe, eine weitere Befehlsmenge einzuführen, die benutzerfreundlicher und weniger maschinenorientiert ist als L1. Diese dritte Menge bildet ebenfalls eine Sprache, die wir L2 nennen (mit der virtuellen Maschine M2). Man kann nun Programme in L2 schreiben, als würde eine virtuelle Maschine mit L2 als Maschinensprache wirklich existieren. Solche Programme können entweder in L1 übersetzt oder von einem in L1 geschriebenen Interpreter ausgeführt werden.

Ein Computer kann als eine Hierarchie von virtuellen Maschinen betrachtet werden, die durch die jeweiligen unteren **Schichten** realisiert werden.

Man könnte nun diese Kette endlos fortsetzen, d.h. neue Sprachen einführen, die jeweils etwas komfortabler als ihre Vorgänger sind, bis man schließlich bei einer geeigneten Sprache angekommen ist. Da jede Sprache auf der Vorgängersprache aufbaut, können wir einen Computer, der diese Technik nutzt, als Folge von übereinanderliegenden **Schichten** oder **Ebenen** (Levels) betrachten (▶Abbildung 1.1). Die unterste Sprache bzw. Ebene ist die einfachste und die höchste die komfortabelste.

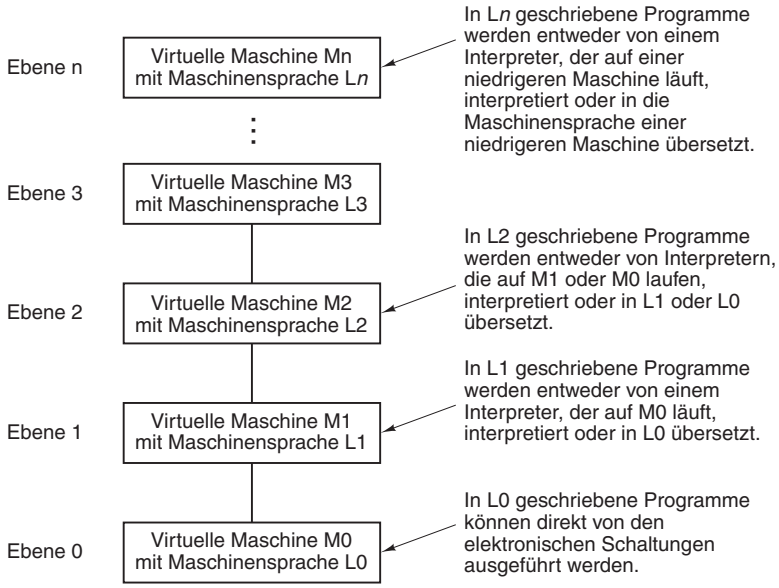

Abbildung 1.1: Struktur einer aus mehreren Ebenen bestehenden Maschine

Zwischen einer Sprache und einer virtuellen Maschine besteht ein wichtiger Zusammenhang. Jede Maschine besitzt eine Maschinensprache mit allen Befehlen, die die Maschine ausführen kann. Praktisch definiert eine Maschine eine Sprache. In gleicher Weise defi-

niert eine Sprache eine Maschine – nämlich die Maschine, die alle in dieser Sprache geschriebenen Programme ausführen kann. Natürlich kann die von einer bestimmten Sprache definierte Maschine außerordentlich kompliziert sein und immense Kosten verschlingen, wenn man sie maßgeschneidert aus elektronischen Schaltungen aufbaut. Dennoch ist eine derartige Maschine vorstellbar. Eine Maschine mit C/C++ oder Java als Maschinensprache wäre tatsächlich komplex, ließe sich aber mit der heutigen Technik leicht realisieren. Andererseits gibt es gute Gründe, derartige Computer nicht zu bauen: Sie wären im Vergleich zu anderen Verfahren viel zu teuer. Es genügt nicht, etwas realisieren zu können: Ein praktisches Konzept muss auch kostengünstig sein.

Einen Computer mit *n* Ebenen kann man sich als *n* unterschiedliche virtuelle Maschinen jeweils mit einer anderen Maschinensprache vorstellen. Wir benutzen die Begriffe „Ebene“ und „virtuelle Maschine“ gleichberechtigt nebeneinander. Nur die in Sprache L0 geschriebenen Programme können direkt von den elektronischen Schaltkreisen ohne Übersetzung oder Interpretation ausgeführt werden. Die in L1, L2, ..., L*n* geschriebenen Programme müssen entweder von einem Interpreter, der auf einer niedrigeren Ebene läuft, interpretiert oder aber in eine andere Sprache, die der niedrigeren Ebene entspricht, übersetzt werden.

Ein Programmierer, der Programme für die virtuelle Maschinenebene *n* schreibt, braucht die zugrunde liegenden Interpreter und Übersetzer nicht zu kennen. Die Maschinenstruktur stellt sicher, dass diese Programme auf irgendeine Weise ausgeführt werden. Praktisch ist es belanglos, ob ein Interpreter, der seinerseits von einem anderen Interpreter ausgeführt wird, diese Programme schrittweise ausführt, oder ob die Elektronik die Programmbefehle direkt verarbeitet. Das Ergebnis ist in beiden Fällen gleich: Die Programme werden ausgeführt.

Die meisten Programmierer, die mit einer Maschine aus *n* Ebenen arbeiten, interessieren sich nur für die oberste Ebene. Das ist diejenige, die der Maschinensprache ganz unten am wenigsten ähnelt. Möchte man aber wissen, wie ein Computer wirklich funktioniert, muss man sich mit allen Ebenen befassen. Entwickler, die neue Computer oder neue Ebenen (d.h. neue virtuelle Maschinen) entwerfen, müssen mit allen Ebenen – und nicht nur mit der obersten – vertraut sein. Die Konzepte und Techniken für die Entwicklung von Maschinen in Form mehrerer Ebenen und die Details dieser Ebenen selbst bilden das Hauptthema dieses Buchs.

1.1.2 Moderne mehrschichtige Maschinen

Die meisten modernen Computer bestehen aus zwei oder mehr Ebenen. Wie ▶Abbildung 1.2 zeigt, gibt es Maschinen mit bis zu sechs Ebenen. Die unterste Ebene 0 ist die reale Hardware der Maschine. Ihre elektronischen Schaltungen führen die Programme in der Maschinensprache von Ebene 1 aus. Der Vollständigkeit halber sei erwähnt, dass es unter unserer Ebene 0 noch eine Ebene gibt. Diese sogenannte (in Abbildung 1.2 nicht dargestellte) **Geräteebene** (**Device Level**) fällt in das Hoheitsgebiet der Elektrotechnik und liegt damit außerhalb der Themen dieses Buchs. Auf dieser Ebene hat der Hardwareentwickler mit einzelnen Transistoren zu tun, d.h. mit konkreten elektronischen Bauelementen, aus denen letztlich ein Computer aufbaut ist. Für die Elektroniker ist dies die unterste Ebene. Die Frage nach der internen Arbeitsweise eines Transistors führt schon in das Gebiet der Festkörperphysik.

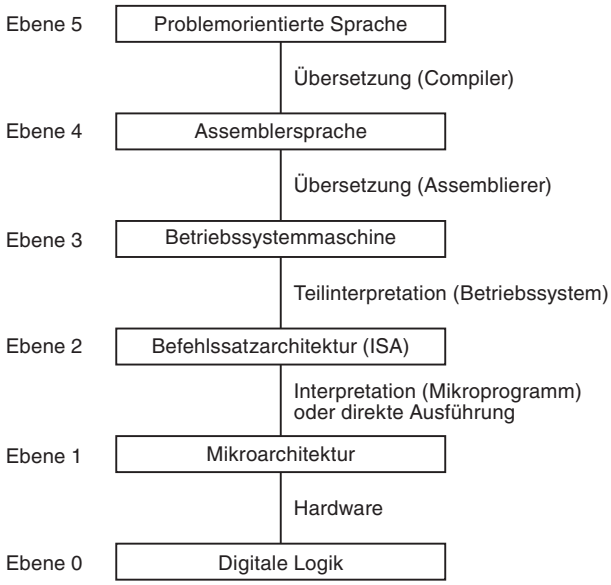

Abbildung 1.2: Struktur eines Computers mit sechs Ebenen; wie die einzelnen Ebenen unterstützt werden, steht rechts unter der jeweiligen Ebene (mit dem Namen des unterstützenden Programms)

Auf der untersten der von uns betrachteten Ebenen – der **Ebene der digitalen Logik** (Digital Logic Level) – interessieren vor allem die sogenannten **Gatter** (**Gates**). Diese bestehen zwar selbst aus analogen Bauelementen, unter anderem aus Transistoren, ergeben aber durch ihre Schaltungstechnik genaue Modelle digitaler Funktionseinheiten. Ein Gatter besitzt einen oder mehrere digitale Eingänge (wobei die Signale 0 oder 1 darstellen) und verknüpft diese nach einer einfachen Funktion – wie zum Beispiel AND oder OR – zu einem Ausgangssignal. Elementare Gatterschaltungen setzen sich aus höchstens einer Handvoll Transistoren zusammen. Mit wenigen Gattern lässt sich bereits ein 1-Bit-Speicher aufbauen, um eine 0 oder eine 1 zu speichern. Mehrere 1-Bit-Speicher (z.B. 16, 32 oder 64) können zu Registern gruppiert werden. Ein **Register** kann eine einzelne Binärzahl aufnehmen, deren maximale Größe von der Anzahl der 1-Bit-Speicher bestimmt wird. Auch die zentrale Verarbeitungseinheit eines Computers (der „Prozessor“) besteht letztlich aus einzelnen Gattern. *Kapitel 3* beschäftigt sich ausführlich mit Gattern und der Ebene der digitalen Logik.

Die nächsthöhere Ebene ist die **Mikroarchitekturebene** (**Microarchitecture Level**). Hier findet man eine Gruppe von (normalerweise) 8 bis 32 Registern, die einen lokalen Speicher bilden, sowie eine als **ALU** (**Arithmetic Logic Unit** – **Rechenwerk**) bezeichnete Schaltung, die einfache arithmetische Operationen und logische Verknüpfungen ausführen kann. Die Register sind mit der ALU verbunden, um einen **Datenpfad** (**Data Path**) zu bilden, über den die Daten fließen. Die grundlegende Aufgabe des Datenpfads besteht darin, ein oder zwei Register auszuwählen, mit denen die ALU arbeitet (beispielsweise die Inhalte der Register addiert), und das Ergebnis der Operation in einem Register abzulegen.

Der Datenpfad wird bei manchen Maschinen durch ein sogenanntes **Mikroprogramm** (Microprogram) gesteuert, bei anderen Maschinen direkt von der Hardware. In früheren Ausgaben dieses Buchs wurde diese Ebene als „Mikroprogrammierebene“ bezeichnet, weil es sich in der Vergangenheit fast immer um einen Interpreter gehandelt hat. Da der

Datenpfad heute vielfach (zumindest teilweise) durch die Hardware gesteuert wird, haben wir die Bezeichnung entsprechend geändert, um dieser Entwicklung Rechnung zu tragen.

Bei Maschinen mit Softwaresteuerung des Datenpfads ist das Mikroprogramm ein Interpreter für die Befehle auf Ebene 2. Über den Datenpfad ruft das Mikroprogramm die Befehle nacheinander ab, analysiert sie und führt sie aus. Beispielsweise stellt sich das für einen `ADD`-Befehl wie folgt dar: Befehl abrufen, dessen Operanden lokalisieren und in Register laden, die Summe von der ALU berechnen lassen und schließlich das Ergebnis zu der Stelle übertragen, wo es erwartet wird. Bei Maschinen mit fest verdrahteter Steuerung des Datenpfads laufen die gleichen Schritte ab, außer dass kein explizit gespeichertes Programm die Interpretation der Befehle von Ebene 2 steuert.

Die Ebene 2 wird als **Befehlssatzarchitektur-Ebene** (**ISA**, **Instruction Set Architecture**) bezeichnet. Jeder Computerhersteller liefert seine Computer mit einem Handbuch aus, das einen Titel wie „Leitfaden der Maschinensprache“ oder „Grundlegende Arbeitsweise des Computermodells Western Wombat 100X“ trägt. Diese Handbücher behandeln die ISA-Ebene und nicht die zugrunde liegenden Ebenen. Wenn vom Befehlssatz der Maschine die Rede ist, geht es tatsächlich um die Befehle, die interpretierend durch das Mikroprogramm oder direkt durch die Hardware ausgeführt werden. Sieht ein Hersteller für einen seiner Computer zwei Interpreter vor, die zwei unterschiedliche ISA-Ebenen realisieren, muss er für beide ein Handbuch der „Maschinensprache“ bereitstellen.

Die nächste Ebene ist normalerweise hybrid. Die meisten Befehle in ihrer Sprache sind auch in der ISA-Ebene vorhanden. (Es spricht nichts dagegen, dass ein Befehl in mehreren Ebenen präsent ist.) Zusätzlich gibt es einen Satz neuer Befehle, eine andere Speicherorganisation, die Möglichkeit, zwei oder mehr Programme gleichzeitig auszuführen, und verschiedene andere Merkmale. Für Ebene 3 gibt es sehr viel mehr unterschiedliche Designs als für die Ebenen 1 und 2.

Für die in Ebene 3 neu hinzukommende Funktionalität ist ein Interpreter zuständig, der auf Ebene 2 läuft und den man aus historischer Sicht als Betriebssystem bezeichnet. Befehle der Ebene 3, die mit denen der Ebene 2 identisch sind, führt das Mikroprogramm (bzw. die fest verdrahtete Steuerung) und nicht das Betriebssystem aus. Mit „hybrid“ ist gemeint, dass einige Befehle der Ebene 3 vom Betriebssystem und andere direkt vom Mikroprogramm interpretiert werden. Wir nennen dies die **Ebene der Betriebssystemmaschine** (**Operating System Machine Level**).

Zwischen den Ebenen 3 und 4 besteht eine grundlegende Trennung. Die untersten drei Ebenen sind kein Tummelplatz für den Durchschnittsprogrammierer, sondern dienen hauptsächlich dazu, die Interpreter und Übersetzer auszuführen, die zur Unterstützung der höheren Ebenen benötigt werden. Diese Interpreter und Übersetzer fallen in den Aufgabenbereich der **Systemprogrammierer**, die sich auf die Entwicklung und Implementierung neuer virtueller Maschinen spezialisiert haben. Die Ebenen von 4 an aufwärts sind für den Anwendungsprogrammierer vorgesehen, der ein Problem zu lösen hat.

Anwendungen werden in Assembler oder einer problemorientierten Programmiersprache geschrieben und durch das Betriebssystem und die darunterliegenden Schichten ausgeführt.

Auf Ebene 4 wechselt auch die Methode, wie die höheren Ebenen unterstützt werden. Auf den Ebenen 2 und 3 läuft immer ein Interpreter. Die Ebenen 4, 5 und höher werden normalerweise – wenn auch nicht immer – durch Übersetzung unterstützt.

Zwischen den ersten drei Ebenen und den Ebenen ab 4 aufwärts besteht ein weiterer Unterschied, der das Wesen der bereitgestellten Sprache betrifft. Die Maschinensprachen der Ebenen 1, 2 und 3 sind numerisch. Die in diesen Sprachen geschriebenen Pro-

gramme bestehen aus langen Zahlenreihen, die für die Maschine gut und für den Menschen schlecht verständlich sind. Die Sprachen ab Ebene 4 enthalten Wörter und Abkürzungen, die auch für den Menschen ohne Weiteres verständlich sind.

Ebene 4 – die Ebene der Assemblersprache – ist eigentlich eine symbolische Form einer der zugrunde liegenden Sprachen. Diese Ebene bietet die Möglichkeit, Programme für die Ebenen 1, 2 und 3 in einer Form zu schreiben, die etwas komfortabler ist als die Sprachen der virtuellen Maschinen an sich. Programme in Assemblersprache werden zuerst in die Sprache der Ebene 1, 2 oder 3 übersetzt und dann von der geeigneten virtuellen oder physischen Maschine interpretiert. Das Programm, das die Übersetzung ausführt, ist ein **Assembler**.

Ebene 5 besteht normalerweise aus Sprachen, die speziell auf den Anwendungsprogrammierer zugeschnitten sind. Von diesen sogenannten **Hochsprachen** (**High-Level Languages**) gibt es im wahrsten Sinne des Wortes Hunderte. Bekanntere Vertreter sind C, C++, Java, Perl, Python und PHP. Die in diesen Sprachen geschriebenen Programme werden im Allgemeinen von Übersetzern (**Compilern**) in die Ebene 3 oder 4 übersetzt, gelegentlich aber auch interpretierend abgearbeitet. Zum Beispiel werden Java-Programme normalerweise zuerst in eine ISA-ähnliche Sprache – den Java-Bytecode – übersetzt, die dann interpretiert wird.

In manchen Fällen enthält Ebene 5 einen Interpreter, der Daten und Operationen für einen bestimmten Anwendungsbereich – wie zum Beispiel symbolische Mathematik – bereitstellt. Die Daten und Operationen sind so gewählt, dass sie für einen Experten auf diesem Gebiet ohne Weiteres verständlich sind.

Unterm Strich ist festzuhalten, dass Computer als Folge von Ebenen konzipiert sind, die jeweils auf ihren Vorgängern aufbauen. Jede Ebene verkörpert eine bestimmte Abstraktion mit unterschiedlichen Objekten und Operationen. Indem wir Computer auf diese Art und Weise entwerfen und analysieren, können wir vorläufig irrelevante Details ignorieren und somit ein komplexes Thema auf ein verständlicheres Niveau bringen.

Den Satz von Datentypen, Operationen und Merkmalen jeder Ebene bezeichnet man als ihre **Architektur** (**Architecture**). Die Architektur betrifft die Aspekte, die für den Benutzer der jeweiligen Ebene sichtbar sind, wie zum Beispiel der verfügbare Hauptspeicher. Dagegen haben Implementierungsdetails, wie z.B. die für die Speicherchips verwendete Technologie, nichts mit der Architektur zu tun. Die **Computerarchitektur** (Computer Architecture) untersucht, wie die für den Programmierer sichtbaren Teile eines Computersystems zu entwerfen sind. Computerarchitektur und Computerorganisation bedeuten in der Praxis das Gleiche.

1.1.3 Evolution mehrschichtiger Maschinen

Um einen Überblick über einige Aspekte mehrschichtiger Maschinen zu geben, beschreiben wir kurz ihre historische Entwicklung, d.h., wie sich Anzahl und Art der Ebenen im Lauf der Jahre entwickelt haben. Programme, die in der echten Maschinensprache eines Computers (Ebene 1) geschrieben sind, lassen sich ohne Interpreter oder Übersetzer direkt von den elektronischen Schaltungen des Computers (Ebene 0) ausführen. Diese elektronischen Schaltungen bilden zusammen mit dem Speicher und den Ein-/Ausgabegeräten die **Hardware** des Computers. Hardware besteht aus greifbaren Objekten – integrierten Schaltungen, gedruckten Leiterplatten (PCBs – Printed Circuit Boards), Kabeln, Netzteilen, Speichern und Druckern – und nicht aus abstrakten Ideen, Algorithmen oder Befehlen.

Demgegenüber besteht **Software** aus **Algorithmen** (detaillierte Vorschriften, wie etwas auszuführen ist) und deren Computerdarstellungen – den Programmen. Programme können auf Festplatte, Diskette, CD-ROM oder anderen Medien gespeichert werden. Das Wesentliche der Software ist aber die Menge der Befehle, die das Programm ausmachen, und nicht das physische Medium, auf dem sie aufgezeichnet sind.

Bei den allerersten Computern war die Grenze zwischen Hardware und Software glasklar zu erkennen. Allerdings hat sie sich im Laufe der Zeit beträchtlich verwischt, was vor allem damit zusammenhängt, dass Ebenen im Zuge der Computerentwicklung hinzugefügt, entfernt oder miteinander verschmolzen wurden. Heutzutage ist es oftmals schwer, sie genau auseinanderhalten zu können [Vahid, 2003]. Das führt uns zu einem der Hauptthemen dieses Buchs:

Die Realisierung von Rechnern besteht aus **Hardware** und **Software**. Welche Funktionen durch Hardware bzw. Software implementiert werden, wird abhängig von Faktoren wie Kosten, Geschwindigkeit und Anpassbarkeit bestimmt.

Hardware und Software sind logisch äquivalent.

Eine durch Software ausgeführte Operation kann auch direkt in die Hardware integriert werden, vorzugsweise nachdem sie ausreichend bekannt ist. Karen Panetta sagte einmal: „Hardware ist nur versteinerte Software.“ Selbstverständlich trifft auch das Gegenteil zu: Jeder von der Hardware ausgeführte Befehl lässt sich auch durch Software simulieren. Welche Funktionen in Hardware und welche in Software zu implementieren sind, wird anhand von Faktoren wie Kosten, Geschwindigkeit, Zuverlässigkeit und erwarteter Häufigkeit von Änderungen entschieden. Es gibt wenige verbindliche Regeln, die besagen, dass X in die Hardware kommt und Y explizit programmiert werden muss. Diese Entscheidungen ändern sich je nach Technologie, Wirtschaftlichkeit, Bedarf und Computernutzung.

Die Erfindung der Mikroprogrammierung

Die ersten digitalen Computer in den 1940er Jahren hatten nur zwei Ebenen: die ISA-Ebene, auf der die gesamte Programmierung erfolgte, und die Ebene der digitalen Logik, die diese Programme ausführte. Die Schaltungen auf der Ebene der digitalen Logik waren kompliziert, schwer zu verstehen, schwierig herzustellen und unzuverlässig.

1951 kam ein Forscher an der Universität von Cambridge, Maurice Wilkes, auf die Idee, einen Computer mit drei Ebenen zu entwickeln, um die Hardware stark zu vereinfachen [Wilkes, 1951]. Diese Maschine sollte einen integrierten, unveränderlichen Interpreter (das Mikroprogramm) erhalten, um Programme auf ISA-Ebene interpretierend auszuführen. Da die Hardware dann nur Mikroprogramme mit einer beschränkten Befehlsmenge, nicht aber Programme auf der ISA-Ebene mit einem wesentlich umfangreicheren Befehlssatz ausführen musste, konnte man mit weniger elektronischen Schaltungen auskommen. Da elektronische Schaltungen damals mit Vakuumröhren arbeiteten, versprach diese Vereinfachung eine geringere Röhrenanzahl und höhere Zuverlässigkeit (d.h. weniger Abstürze pro Tag).

Einige dieser dreischichtigen Maschinen wurden in den 1950er und 1960er Jahren gebaut. 1970 hatte sich das Konzept, die ISA-Ebene durch ein Mikroprogramm statt direkt durch die Elektronik interpretieren zu lassen, durchgesetzt. Alle bedeutenden Computer dieser Zeit arbeiteten damit.

Die Erfindung des Betriebssystems

Zu jener Zeit waren die meisten Computer als „Open Shop“ eingerichtet, d.h., der Programmierer musste die Maschine eigenverantwortlich bedienen. Neben jeder Maschine lag

ein Anmeldeformular. Wollte ein Programmierer ein Programm ausführen, trug er[1] sich für einen Zeitraum, sagen wir Mittwoch früh von 3 Uhr bis 5 Uhr, ein (viele Programmierer zogen Arbeitszeiten vor, in denen im Maschinenraum Ruhe herrschte). Zum entsprechenden Zeitpunkt begab sich der Programmierer in den Maschinenraum – in der einen Hand einen Stapel 80spaltiger Lochkarten (dem damaligen Eingabemedium), in der anderen einen gespitzten Bleistift. Im Computerraum angekommen, bugsierte er den noch arbeitenden Programmierer sanft in Richtung Tür und übernahm den Computer.

Wollte der Programmierer ein FORTRAN-Programm ausführen, waren die folgenden Schritte erforderlich:

1. Er ging zum Schrank, in dem alle Programme aufbewahrt wurden, nahm einen großen grünen Stapel mit der Beschriftung FORTRAN-Compiler heraus, legte ihn in den Kartenleser und drückte die START-Taste.
2. Er legte sein FORTRAN-Programm in den Kartenleser ein und drückte die FORTSETZEN-Taste. Das Programm wurde eingelesen.
3. Wenn der Computer anhielt, las der Programmierer sein FORTRAN-Programm ein zweites Mal ein. Obwohl manche Compiler nur einen Durchlauf über die Eingabe verlangten, waren bei vielen zwei oder mehr Durchläufe erforderlich. Bei jedem Durchlauf war ein riesiger Kartenstapel zu bewältigen.
4. Langsam näherte sich die Übersetzung ihrem Ende – ein Zeitpunkt, an dem die meisten Programmierer oft nervös wurden. Falls nämlich der Compiler einen Fehler im Programm gefunden hatte, musste dieser korrigiert und der gesamte Prozess von vorn ausgeführt werden. Hatte das Programm keine Fehler, stanzte der Compiler das in Maschinensprache übersetzte Programm auf Lochkarten.
5. Der Programmierer legte dann das Maschinenspracheprogramm und den Stapel mit der Unterprogrammbibliothek in den Kartenleser und las beide ein.
6. Die Ausführung des Programms begann. Meist funktionierte es nicht richtig und hielt unerwartet irgendwo mittendrin an. Dann versuchte der Programmierer mehr oder weniger zielstrebig, das Ganze mithilfe der Konsolenschalter hinzubiegen und starrte eine Weile auf die Konsolenlampen. Hatte er Glück, dann erkannte er das Problem, korrigierte den Fehler und lief wieder zum Schrank mit dem großen grünen FORTRAN-Compiler, um von vorn zu beginnen. Der weniger Glückliche druckte den Speicherinhalt aus – einen sogenannten **Core-Dump** – und nahm den Ausdruck zur Fehlersuche mit nach Hause, um sich eingehend mit dem Problem zu beschäftigen.

Mit geringen Variationen war diese Prozedur in vielen Rechenzentren jahrelang die Normalität. Sie zwang die Programmierer, die Betriebsweise der Maschine zu kennen und zu verstehen, was es zu tun galt, wenn sich einer der häufigen Abstürze ereignete. Die Maschine stand häufig untätig da, während die Leute Kartenstapel herumtrugen oder sich am Kopf kratzten und überlegten, warum ihre Programme nicht richtig liefen.

Etwa 1960 versuchten Programmierer, die Zeitvergeudung zu reduzieren, indem sie die Arbeit des Bedieners automatisierten. Daraus entstand ein Programm namens **Betriebssystem** (**Operating System**), das die ganze Zeit im Computer verblieb. Der Programmierer lieferte bestimmte Steuerkarten zusammen mit dem Programm; beides wurde vom Betriebssystem gelesen und ausgeführt. ▶Abbildung 1.3 zeigt einen Beispiel-Job für eines der ersten weit verbreiteten Betriebssysteme – FMS (FORTRAN Monitor System) auf dem IBM 709.

1 „Er“ möchte der Autor im gesamten Buch als „er oder sie“ verstanden wissen.

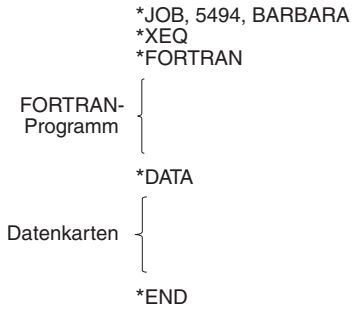

Abbildung 1.3: Ein Beispiel-Job für das Betriebssystem FMS

Das Betriebssystem las die Karte *JOB ein und benutzte die darauf befindlichen Informationen für Verwaltungszwecke. (Mit dem Sternchen wurden Steuerkarten gekennzeichnet, damit man sie nicht mit Programm- und Datenkarten verwechselte.) Später las es die Karte *FORTRAN ein – eine Anweisung zum Laden des FORTRAN-Compilers von einem Magnetband. Anschließend las der Compiler das FORTRAN-Programm ein und kompilierte es. War der Compiler damit fertig, gab er die Steuerung wieder an das Betriebssystem ab, das dann die Karte *DATA einlas. Sie enthielt eine Anweisung zur Ausführung des übersetzten Programms, das die auf *DATA folgenden Karten als Daten verwendete.

Obwohl das Betriebssystem dafür gedacht war, die Arbeit (den Job) des Bedieners (Operator heißt daher die englische Bezeichnung) zu automatisieren, war es auch der erste Schritt in der Entwicklung einer neuen virtuellen Maschine. Man könnte die *FORTRAN-Karte als virtuellen Befehl „Kompiliere das Programm“ betrachten. Die *DATA-Karte wäre dann der virtuelle Befehl „Führe das Programm aus“. Mit nur zwei Befehlen konnte man kaum von einer richtigen Ebene sprechen, aber immerhin war es der erste Schritt in diese Richtung.

In den Folgejahren wurden Betriebssysteme ständig verbessert. Neue Befehle, Funktionen und Merkmale wurden der ISA-Ebene hinzugefügt, bis sie begann, das Aussehen einer neuen Ebene anzunehmen. Einige Befehle dieser neuen Ebene waren mit denen der ISA-Ebene identisch, andere wiederum unterschieden sich völlig, insbesondere die Ein-/Ausgabebefehle. Die neuen Befehle wurden oftmals als **Betriebssystemmakros** (**Operating System Macros**) oder **Supervisor-Aufrufe** bezeichnet. Der heute gängige Begriff lautet **Systemaufruf** (**System Call**).

Betriebssysteme wurden auch in anderer Hinsicht weiterentwickelt. Die frühen Varianten lasen Kartenstapel und gaben Ausdrucke auf dem Zeilendrucker aus. Diese Organisation wurde **Batch-System** (**Stapelsystem**) genannt. Von der Programmübergabe an das System bis zu dem Zeitpunkt, zu dem die Ergebnisse vorlagen, waren Wartezeiten von mehreren Stunden üblich. Unter diesen Bedingungen war es recht schwierig, Software zu entwickeln.

Anfang der 1960er Jahre entwickelten Forscher am Dartmouth College des M.I.T und an anderen Orten Betriebssysteme, die es (mehreren) Programmierern gestatteten, direkt mit dem Computer zu kommunizieren. Bei diesen Systemen waren entfernte Terminals über Telefonleitungen mit dem Zentralrechner verbunden. Die CPU wurde von vielen Benutzern gemeinsam genutzt. Ein Programmierer konnte ein Programm eintippen und erhielt die Ergebnisse fast sofort zurück – im Büro, in der Garage, zu Hause oder wo immer das Terminal stand. Diese Systeme wurden **Timesharing-Systeme** genannt.

Unser Interesse an Betriebssystemen gilt denjenigen Teilen, die Befehle und Merkmale interpretieren, die sich auf Ebene 3 und nicht auf der ISA-Ebene befinden, und weniger den Timesharing-Aspekten. Obwohl es hier nicht besonders betont wird, sollten Sie daran denken, dass Betriebssysteme mehr tun als lediglich Merkmale zu interpretieren, die der ISA-Ebene hinzugefügt wurden.

Verlagerung von Funktionen in den Mikrocode

Komplexe Befehle lassen sich als **Mikroprogramme** in der Hardware realisieren. Aktuelle Prozessoren beschränken sich aber meist auf elementare Befehle, die direkt durch Hardware realisiert sind und schnell ausgeführt werden können.

Nachdem sich die Mikroprogrammierung (um 1970 herum) durchgesetzt hatte, erkannten die Entwickler, dass sie allein durch Erweiterung des Mikroprogramms neue Befehle hinzufügen konnten. Mit anderen Worten: Sie konnten „Hardware“ (neue Maschinenbefehle) durch Programmierung ergänzen. Diese Erkenntnis führte zu einer wahren Explosion der Maschinenbefehlssätze – die Entwickler wollten sich gegenseitig mit größeren und besseren Befehlssätzen übertreffen. Viele dieser Befehle waren eigentlich bedeutungslos in dem Sinne, dass sich ihre Wirkung leicht mit bestehenden Befehlen erreichen ließ. Oftmals aber waren sie etwas schneller als eine Sequenz vorhandener Befehle. Zum Beispiel hatten viele Maschinen einen Befehl `INC` (INCrement, d.h. inkrementieren), der 1 zu einer Zahl addiert. Da diese Maschinen auch einen allgemeinen `ADD`-Befehl besaßen, war ein spezieller Befehl zum Addieren von 1 (oder eben auch zum Addieren von 720) nicht nötig. Allerdings war `INC` meist ein bisschen schneller als `ADD`, also wurde er einfach hinzugefügt.

Aus dem gleichen Grund wurde das Mikroprogramm um viele andere Befehle erweitert, z.B.:

1. Befehle für die Multiplikation und Division von Ganzzahlen
2. Befehle für die Gleitkommaarithmetik
3. Befehle zum Aufrufen von Prozeduren und zum Zurückkehren aus Prozeduren
4. Befehle zur Beschleunigung von Schleifen
5. Befehle für den Umgang mit Zeichenfolgen

Und nachdem die Maschinenentwickler gesehen hatten, wie leicht sich neue Befehle hinzufügen ließen, suchten sie nach anderen Funktionsmerkmalen, um ihre Mikroprogramme zu erweitern. Dazu gehören beispielsweise:

1. Funktionen zur Beschleunigung von Berechnungen mit Arrays (Indizierung und indirekte Adressierung)
2. Funktionen, die das Verschieben von Programmen im Speicher nach Beginn der Ausführung erlauben (sogenannte Relokations-Funktionen)
3. Unterbrechungssysteme (Interrupts), die dem Computer signalisieren, dass eine Ein- oder Ausgabeoperation abgeschlossen ist
4. Die Möglichkeit, mit wenigen Befehlen ein Programm zu suspendieren und ein anderes zu starten (Prozessumschaltung)
5. Spezielle Befehle zur Verarbeitung von Audio-, Bild- und Multimediadateien

Daneben sind im Lauf der Jahre viele andere Konzepte und Eigenschaften hinzugekommen, die insbesondere darauf abzielen, die Geschwindigkeit einer bestimmten Aktivität zu steigern.

Beseitigung der Mikroprogrammierung

In den goldenen Jahren der Mikroprogrammierung (in den 1960ern und 1970ern) nahmen Mikroprogramme drastisch an Umfang zu und wurden damit leider auch immer langsamer. Schließlich haben einige Informatiker erkannt, dass sich die Maschinen beschleunigen lassen, wenn man das Mikroprogramm abschafft, den Befehlssatz abspeckt und die verbleibenden Befehle direkt ausführen lässt (d.h. Hardwaresteuerung des Datenpfads). In gewissem Sinn hat sich damit der Kreis der Computerentwicklung geschlossen. Man war wieder dort angekommen, wo Wilkes die Mikroprogrammierung auf den Weg gebracht hatte.

Doch das Rad der Geschichte dreht sich weiter: Moderne Prozessoren stützen sich weiterhin auf Mikroprogrammierung, um komplexe Befehle in internen Mikrocode zu übersetzen, der sich dann direkt auf der abgespeckten Hardware ausführen lässt.

Dieser Überblick soll deutlich machen, dass die Grenze zwischen Hardware und Software beliebig verläuft und sich ständig ändert. Die Software von heute ist möglicherweise die Hardware von morgen und umgekehrt. Auch die Grenzen zwischen den verschiedenen Ebenen sind fließend. Aus Sicht des Programmierers ist es unwichtig, wie ein Befehl tatsächlich implementiert ist (außer vielleicht hinsichtlich seiner Geschwindigkeit). Wer auf der ISA-Ebene programmiert, kann den hier implementierten Multiplikationsbefehl genau wie einen Hardwarebefehl aufrufen – der Programmierer muss sich nicht darum kümmern und noch nicht einmal wissen, ob es sich in Wirklichkeit um einen Hardwarebefehl handelt. Des einen Hardware ist des anderen Software. Auf all diese Themen kommen wir später in diesem Buch noch zurück.

1.2 Meilensteine der Computerarchitektur

Die Evolution des modernen Digitalrechners hat Hunderte verschiedener Varianten hervorgebracht. Auch wenn die meisten längst vergessen sind, haben einige einen bedeutenden Einfluss auf moderne Konzepte ausgeübt. Dieser Abschnitt skizziert die wichtigsten Stationen der Computergeschichte, um besser verstehen zu können, wie wir dorthin gelangt sind, wo wir heute stehen. Es liegt auf der Hand, dass dieser Abschnitt nur einige Highlights streifen und nicht auf alle Wegbereiter eingehen kann. ▶Tabelle 1.1 listet herausragende Maschinen auf, die in diesem Abschnitt näher behandelt werden. Slater [Slater, 1987] ist eine Fundgrube für zusätzliche Informationen zu den Pionieren des Computerzeitalters. Einschlägige Biographien und hervorragende Farbfotos von Louis Fabian Bachrach finden Sie im Bildband von Morgan [Morgan, 1997].

Jahr	Bezeichnung	Hersteller	Anmerkungen
1834	Analytical Engine	Babbage	Erster Versuch, einen digitalen Computer zu bauen
1936	Z1	Zuse	Erste funktionierende Relais-Rechenmaschine
1943	COLOSSUS	Britische Regierung	Erster elektronischer Computer
1944	Mark I	Aiken	Erster amerikanischer Universalcomputer
1946	ENIAC I	Eckert/Mauchley	Beginn der modernen Computergeschichte
1949	EDSAC	Wilkes	Erster speicherprogrammierter Computer
1951	Whirlwind I	M.I.T.	Erster Echtzeitcomputer
1952	IAS	Von Neumann	Die meisten heutigen Maschinen weisen dieses Design auf
1960	PDP-1	DEC	Erster Minicomputer (50 Stück verkauft)
1961	1401	IBM	Sehr beliebter Rechner bei kleinen Firmen
1962	7094	IBM	Beherrschte Anfang der 60er Jahre die wissenschaftliche Rechenwelt
1963	B5000	Burroughs	Erste für eine Hochsprache entwickelte Maschine
1964	360	IBM	Erste als Familie ausgelegte Produktlinie
1964	6600	CDC	Erster wissenschaftlicher Supercomputer
1965	PDP-8	DEC	Erster Minicomputer für den Massenmarkt (50.000 Stück verkauft)
1970	PDP-11	DEC	Vorherrschender Minicomputer der 70er Jahre
1974	8080	Intel	Erster 8-Bit-Universalcomputer auf einem Chip
1974	CRAY-1	Cray	Erster Vektor-Supercomputer
1978	VAX	DEC	Erster 32-Bit-Superminicomputer
1981	IBM PC	IBM	Beginn der modernen Personalcomputerära
1981	Osborne-1	Osborne	Erster portabler Computer
1983	Lisa	Apple	Erster Personalcomputer (PC) mit grafischer Benutzeroberfläche (GUI)
1985	386	Intel	Erster 32-Bit-Vorgänger der Pentium-Linie
1985	MIPS	MIPS	Erster kommerzieller RISC-Rechner
1985	XC2064	Xilinx	Erstes FPGA (Field Programmable Gate Array)
1987	SPARC	Sun	Erste SPARC-basierte RISC-Workstation
1989	GridPad	Grid Systems	Erster kommerzieller Tablet-PC
1990	RS6000	IBM	Erste superskalare Maschine
1992	Alpha	DEC	Erster 64-Bit-Personalcomputer
1992	Simon	IBM	Erstes Smartphone

Tabelle 1.1: Einige Meilensteine in der Entwicklung der modernen Digitalrechner

Jahr	Bezeichnung	Hersteller	Anmerkungen
1993	Newton	Apple	Erster Palmtop
2001	POWER4	IBM	Erster Prozessor mit zwei Kernen auf einem Chip (Dual-Core)

Tabelle 1.1: Einige Meilensteine in der Entwicklung der modernen Digitalrechner *(Forts.)*

1.2.1 Die nullte Generation – Mechanische Computer (1642–1945)

Der französische Wissenschaftler Blaise Pascal (1623–1662) hat die erste funktionierende Rechenmaschine gebaut. Ihm zu Ehren wurde eine Programmiersprache Pascal genannt. Pascal baute dieses Gerät 1642 im Alter von 19 Jahren, um seinem Vater, einem Steuereintreiber für die französische Regierung, zu helfen. Die Maschine arbeitete rein mechanisch, hatte ein Räderwerk und wurde mit einer Handkurbel betrieben.

Pascals Maschine konnte nur addieren und subtrahieren. Dreißig Jahre später baute der große deutsche Mathematiker Gottfried Wilhelm Leibniz (1646–1716) eine andere mechanische Maschine, die auch multiplizieren und dividieren konnte. Das Gerät von Leibniz entsprach dem klassischen Taschenrechner mit vier Funktionen, der drei Jahrhunderte später entwickelt wurde.

In den nächsten 150 Jahren passierte nicht viel, bis der Mathematikprofessor der Universität von Cambridge und Erfinder des Geschwindigkeitsmessers Charles Babbage (1792–1871) seine **Difference Engine** (Differenzmaschine) entwickelte und baute. Dieses mechanische Gerät, das wie das von Pascal nur addieren und subtrahieren konnte, diente der Berechnung von Zahlentabellen für die Schiffsnavigation. Die Maschine führte einen einzigen Algorithmus aus – die Methode finiter Differenzen mithilfe von Polynomen. Das interessanteste Merkmal der Differenzmaschine war ihre Ausgabemethode: Sie stanzte die Ergebnisse mit stählernen Prägestempeln auf eine Kupferplatte und ließ somit schon eine Entwicklung erahnen, die erst viel später mit einmal beschreibbaren Medien wie zum Beispiel Lochkarten und CD-ROM einsetzte.

Die Differenzmaschine funktionierte zwar gut, langweilte Babbage aber bald, weil sie nur einen Algorithmus ausführen konnte. Er steckte viel Zeit und einen Großteil des Familienvermögens (ganz zu schweigen von 17.000 Pfund an Regierungsgeldern) in das Design und die Konstruktion eines Folgegeräts namens **Analytical Engine**. Die Analytical Engine hatte vier Komponenten: Speicher, Mühle (Recheneinheit), Eingabeeinheit (Lochkartenleser) und Ausgabeeinheit (gestanzte und gedruckte Ausgabe). Der Speicher fasste 1000 Wörter zu je 50 Dezimalziffern, die Variablen und Ergebnisse aufnahmen. Die Mühle konnte Operanden aus dem Speicher entgegennehmen, dann addieren, subtrahieren, multiplizieren oder dividieren und schließlich das Ergebnis an den Speicher zurückgeben. Wie die Differenzmaschine war sie gänzlich mechanisch.

Als großer Fortschritt ist der universelle Charakter der Analytical Engine zu sehen. Sie las Befehle von Lochkarten und führte sie aus. Bestimmte Befehle wiesen die Maschine an, zwei Zahlen aus dem Speicher zu holen, zur Mühle zu befördern, dort zu verarbeiten (z.B. zu addieren) und schließlich die Ergebnisse wieder zurück an den Speicher zu schicken. Andere Befehle konnten eine Zahl prüfen und bedingt verzweigen, je nachdem, ob die Zahl positiv oder negativ war. Durch Einstanzen eines anderen Programms auf die Eingabekarten konnte die Analytical Engine unterschiedliche Berechnungen ausführen, wofür die Differenzmaschine gar nicht ausgelegt war.

Da sich die Analytical Engine in einer einfachen Assemblersprache programmieren ließ, brauchte sie Software. Um diese Software zu produzieren, stellte Babbage eine junge Frau namens Ada Augusta Lovelace – die Tochter des berühmten englischen Dichters Lord Byron – ein. Ada Lovelace war damit nicht nur die erste Programmierer**in**, sondern überhaupt der erste Computerprogrammierer der Welt. Die Programmiersprache Ada wurde nach ihr benannt.

Wie vielen modernen Entwicklern gelang es Babbage leider nie, die Hardware ganz von Fehlern zu befreien. Das Problem war, dass Abertausende von Zahnrädern, Wellen und Getrieben mit einer Genauigkeit hergestellt werden mussten, die die Technik des 19. Jahrhunderts noch nicht liefern konnte. Dennoch waren Babbages Ideen seiner Zeit weit voraus. Die meisten modernen Computer weisen heute noch eine Struktur auf, die derjenigen der Analytical Engine ähnelt. Man kann Babbage deshalb ruhigen Gewissens den (Groß-)Vater des modernen Digitalrechners nennen.

Die nächste größere Entwicklung folgte Ende der 1930er Jahre, als der deutsche Maschinenbaustudent Konrad Zuse eine Reihe von automatischen Rechenmaschinen mittels elektromagnetischer Relais baute. Nach Ausbruch des Zweiten Weltkriegs konnte er keine Regierungsmittel erhalten, weil die Regierungsbürokraten damit rechneten, den Krieg so schnell zu gewinnen, dass die neue Maschine erst fertig sein würde, wenn alles vorbei wäre. Zuse wusste nichts von Babbages Arbeit, und seine Maschinen wurden durch die Bombenangriffe auf Berlin durch die Alliierten 1944 zerstört. Aus diesem Grund hatte seine Arbeit keinen Einfluss auf spätere Maschinen. Er war aber einer der Pioniere auf diesem Gebiet.

Einige Zeit später arbeiteten John Atanasoff am Iowa State College und George Stibbitz in den Bell Laboratories an der Entwicklung von Rechenmaschinen. Atanasoffs Maschine war für die damalige Zeit erstaunlich fortschrittlich. Sie benutzte Binärarithmetik und hatte Kondensatoren als Speicher, die periodisch aufgefrischt wurden, um die Ladung aufrechtzuerhalten, was man „Jogging the Memory“ nannte, d.h. „dem Gedächtnis auf die Sprünge helfen“. Moderne dynamische Speicherchips (DRAM) funktionieren auf genau die gleiche Weise. Leider wurde die Maschine nie in Betrieb genommen. Irgendwie war Atanasoff wie Babbage: ein Visionär, der sich letztlich der unzureichenden Hardwaretechnik seiner Zeit geschlagen geben musste.

Der Computer von Stibbitz war zwar primitiver als der von Atanasoff, funktionierte aber. Stibbitz stellte ihn anlässlich einer Tagung am Dartmouth College 1940 der Öffentlichkeit vor. Unter den Teilnehmern befand sich John Mauchley, ein unbekannter Physikprofessor der Universität von Pennsylvania. Die Computerwelt sollte von Mauchley noch hören.

Während Zuse, Stibbitz und Atanasoff an der Entwicklung automatischer Rechenmaschinen arbeiteten, quälte sich ein junger Mann namens Howard Aiken mit langweiligen numerischen Berechnungen im Rahmen seiner Doktorarbeit an der Harvard-Universität. Nach seiner Promotion wurde ihm klar, wie hilfreich es wäre, solche Berechnungen mit einer Maschine durchführen zu können. Er stöberte in der Bibliothek, fand Babbages Arbeiten und entschloss sich, aus Relais den Universalcomputer zu bauen, den Babbage aus Zahnrädern nicht hatte erschaffen können.

Aikens erste Maschine, die Mark I, wurde 1944 in Harvard fertiggestellt. Sie hatte 72 Wörter mit je 23 Dezimalziffern und eine Befehlszeit von 6 Sekunden. Zur Ein- und Ausgabe wurden Lochbänder verwendet. Bis zur Fertigstellung von Mark II, Aikens Nachfolgermodell, waren Relais-Computer überholt. Das elektronische Zeitalter hatte begonnen.

1.2.2 Die erste Generation – Vakuumröhren (1945–1955)

Angeregt wurde der Bau elektronischer Computer durch den Zweiten Weltkrieg. In den ersten Kriegsjahren wurde ein britisches Schiff nach dem anderen von deutschen U-Booten versenkt. Über Funk erteilte die deutsche Admiralität in Berlin Befehle an die U-Boote, die von den Briten abgehört wurden. Ärgerlich daran war, dass die Nachrichten mit einem Gerät namens **ENIGMA**, dessen Vorläufer vom Amateurerfinder und früheren US-Präsidenten Thomas Jefferson entwickelt wurde, verschlüsselt waren.

Schon zu Beginn des Kriegs gelang es dem britischen Geheimdienst, vom polnischen Geheimdienst eine ENIGMA-Maschine zu bekommen, die dieser den Deutschen gestohlen hatte. Um eine verschlüsselte Nachricht zu knacken, waren aber immense Berechnungen vonnöten, und das in möglichst kurzer Zeit. Die britische Regierung richtete ein streng geheimes Labor ein, das sie mit dem Bau eines elektronischen Computers beauftragte. Der berühmte britische Mathematiker Alan Turing half bei der Entwicklung dieser Maschine namens **COLOSSUS**. Die COLOSSUS wurde 1943 in Betrieb genommen. Da die britische Regierung aber praktisch jeden Aspekt des Projekts 30 Jahre als Militärgeheimnis unter Verschluss hielt, führte die Entwicklung dieser Maschine in eine Sackgasse. Die Maschine wird hier nur erwähnt, weil sie der erste elektronische Digitalrechner der Welt war.

Abgesehen davon, dass Zuses Maschinen zerstört und die COLOSSUS gebaut wurde, wirkte sich der Krieg auch auf die Rechentechnik in den USA aus. Die Armee brauchte Reichweitentabellen zur Ausrichtung von schwerem Geschütz. Zur Produktion dieser Tabellen mit handbetriebenen Rechnern wurden Hunderte von Frauen eingestellt (weil man davon ausging, dass Frauen genauer arbeiten als Männer). Trotzdem war der Prozess sehr zeitaufwendig und fehleranfällig.

John Mauchley, dem die Arbeiten von Atanasoff und Stibbitz bekannt waren, erkannte das Interesse der Armee an mechanischen Rechenmaschinen. Wie viele Computerwissenschaftler nach ihm beantragte er bei der Armee Mittel zur Konstruktion eines elektronischen Computers. Sein Angebot wurde 1943 angenommen und Mauchley und einer seiner Studenten, J. Presper Eckert, nahmen die Entwicklungsarbeiten an einem elektronischen Computer auf, den sie **ENIAC** (**Electronic Numerical Integrator And Computer**) nannten. Er bestand aus 18.000 Vakuumröhren und 1500 Relais, wog 30 Tonnen und hatte eine Leistungsaufnahme von mehr als 150 kW. Architektonisch bestand die Maschine aus 20 Registern, die je eine 10-stellige Dezimalzahl aufnehmen konnten. (Ein Dezimalregister ist ein sehr kleiner Speicher, der eine Zahl mit einer bestimmten Anzahl von Dezimalziffern aufnehmen kann, ähnlich dem Kilometerzähler im Auto.) Der ENIAC wurde über 6000 Drehschalter und Schalttafeln mit einem wahrhaften Dickicht aus Kabeln programmiert.

Die Maschine wurde erst 1946 fertig – zu spät für ihren ursprünglichen Zweck. Da der Krieg vorbei war, erlaubte man Mauchley und Eckert, einen Ferienkurs zu organisieren, um ihren Wissenschaftskollegen von ihrer Arbeit zu berichten. Dieser Ferienkurs war der Anfang eines explosionsartigen Interesses am Bau großer Digitalrechner.

Nach diesem historischen Ereignis schickten sich viele Wissenschaftler an, elektronische Computer zu bauen. Als Erster war der EDSAC (1949) betriebsfähig, den Maurice Wilkes an der Universität von Cambridge baute. Ihm folgten JOHNNIAC von der Rand Corporation, ILLIAC an der Universität von Illinois, MANIAC am Los Alamos Laboratory und WEIZAC am Weizmann-Institut in Israel.

Eckert und Mauchley begannen bald schon mit der Entwicklung eines Nachfolgers, dem **EDVAC** (**Electronic Discrete Variable Automatic Computer**). Dieses Projekt war aber zum Sterben verurteilt, als die beiden die Universität von Pennsylvania verließen und in Philadelphia (das Silicon Valley gab es damals noch nicht) eine Firma – die Eckert-Mauchley Computer Corporation – gründeten. Nach einer Reihe von Fusionen entstand daraus die moderne Unisys Corporation.

Nebenbei sei erwähnt, dass Eckert und Mauchley ein Patent mit dem Anspruch anmeldeten, den Digitalrechner erfunden zu haben. Rückblickend hätte es sich gewiss gelohnt, über ein solches Patent zu verfügen. Nach jahrelangem Rechtsstreit urteilte das Gericht, dass das Eckert-Mauchley-Patent ungültig sei und John Atanasoff den Digitalrechner erfunden habe, auch wenn er ihn nie patentieren ließ. Dadurch wurde die Erfindung praktisch zum Gemeingut.

Während Eckert und Mauchley am EDVAC arbeiteten, ging ein Mitglied des ENIAC-Projekts, John von Neumann, an das Princeton Institute of Advanced Studies und entwickelte seine eigene Version des EDVAC, die **IAS-Maschine**. Von Neumann war ein Genie vom Rang eines Leonardo da Vinci. Er beherrschte viele Sprachen, war Experte in Physik und Mathematik und merkte sich alles, was er je hörte, sah oder las. Er konnte aus Büchern zitieren, die er Jahre zuvor gelesen hatte. Als er anfing, sich für Computer zu interessieren, war er schon einer der renommiertesten Mathematiker der Welt.

Ihm war recht schnell klar, dass die Programmierung von Computern mit Unmengen von Schaltern und Kabeln langsam, mühselig und unflexibel war. Er erkannte, dass sich das Programm in digitaler Form zusammen mit den Daten im Speicher des Computers darstellen ließ. Außerdem sah er, dass man die schwerfällige serielle Dezimalarithmetik des ENIAC mit 10 Vakuumröhren für jede Ziffer (1 Röhre an- und 9 ausgeschaltet) durch parallele Binärarithmetik ersetzen konnte, etwas, was Atanasoff schon Jahre früher erkannt hatte.

Sein erster Entwurf ist heute als die **Von-Neumann-Maschine** bekannt. Diese Maschine wurde im EDSAC, dem ersten speicherprogrammierten Computer, benutzt und ist heute, über ein halbes Jahrhundert später, immer noch die Grundlage fast aller digitalen Computer. Dieses Design (und die IAS-Maschine, die unter der Mitwirkung von Herman Goldstine entwickelt wurde) hatte einen derart starken Einfluss, dass es eine kurzen Beschreibung wert ist. Obwohl der Name *von Neumann* immer mit diesem Entwurf verbunden wird, haben Goldstine und andere ebenfalls große Anteile daran. ▶Abbildung 1.4 zeigt eine Skizze der Architektur.

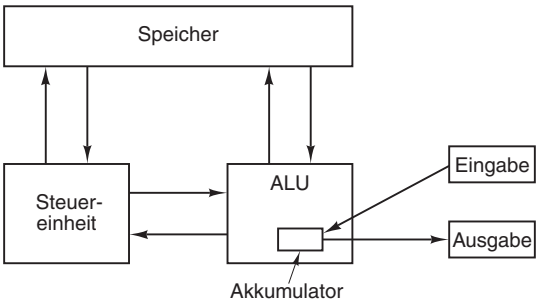

Abbildung 1.4: Die ursprüngliche Von-Neumann-Maschine

Die Von-Neumann-Maschine besteht aus fünf grundlegenden Teilen: dem Speicherwerk, dem Rechenwerk (Arithmetic Logic Unit, ALU), dem Steuerwerk, dem Eingabewerk und dem Ausgabewerk. Der Speicher bestand aus 4096 Wörtern. Ein Wort umfasste 40 Bit, die jeweils 0 oder 1 sein konnten. Jedes Wort nahm entweder zwei 20-Bit-Befehle oder eine 40-Bit-Ganzzahl mit Vorzeichen auf. Ein Befehl kennzeichnete mit 8 Bit den Befehlstyp und mit 12 Bit eines der 4096 Speicherwörter. Das Rechenwerk (ALU) und das Steuerwerk bildeten zusammen das „Gehirn" des Computers. In modernen Computern sind sie auf einem einzelnen Chip vereint, der **zentralen Verarbeitungseinheit** oder **CPU** (**Central Processing Unit**).

Von-Neumann-Maschine: John von Neumann entwarf einen Rechner mit Binärarithmetik, bei dem das Programm im gleichen Speicher wie die Daten abgelegt wurde. Dieser Entwurf hat die Entwicklung der Rechner bis in die heutige Zeit geprägt.

Im Rechenwerk befand sich ein spezielles internes 40-Bit-Register, der **Akkumulator**. Ein typischer Befehl addierte ein Speicherwort zum Akkumulator oder legte den Inhalt des Akkumulators im Speicher ab. Die Maschine besaß keine Gleitkommaarithmetik, weil laut von Neumann jeder Mathematiker in der Lage sein sollte, den Dezimalpunkt (eigentlich den Binärpunkt) im Kopf „mitzuführen".

Etwa zu der Zeit, als von Neumann die IAS-Maschine entwickelte, arbeiteten Wissenschaftler am M.I.T. ebenfalls an einem Computer. Im Gegensatz zu IAS, ENIAC und anderen derartigen Maschinen, die lange Wortlängen hatten und für das Verarbeiten großer Zahlenmengen (Number Crunching) gedacht waren, hatte Whirlwind I, die Erfindung des M.I.T.-Teams, ein 16-Bit-Wort und war auf Echtzeitsteuerung ausgelegt. Dieses Projekt führte zur Erfindung des magnetischen Kernspeichers durch Jay Forrester und letztendlich zum ersten kommerziellen Minicomputer.

Zu dieser Zeit war IBM eine kleine Firma, die mit der Fertigung von Kartenlochern und mechanischen Kartensortiermaschinen befasst war. Ein Teil von Aikens Forschungsmitteln stammte zwar von IBM, die Firma war aber nicht sonderlich an Computern interessiert, bis sie 1953 den 701 produzierte, lange nachdem sich das Unternehmen von Eckert und Mauchley mit dem UNIVAC-Computer zur Nummer Eins auf dem kommerziellen Markt etabliert hatte. Der 701 hatte 2048 Wörter zu je 36 Bit mit zwei Befehlen pro Wort. Er war der erste einer Reihe von wissenschaftlichen Computern, die ein Jahrzehnt danach die Branche beherrschen sollten. Drei Jahre später folgte das Modell 704 mit einem Kernspeicher von 4096 Wörtern, 36-Bit-Befehlen und als Neuerung mit einer Gleitkomma-Hardware. 1958 begann IBM mit der Produktion ihrer letzten Vakuumröhrenmaschine (Modell 709), bei der es sich im Grunde um einen aufgerüsteten 704-Rechner handelte.

1.2.3 Die zweite Generation – Transistoren (1955–1965)

Der Transistor wurde 1948 von John Bardeen, Walter Brattain und William Shockley an den Bell Laboratories erfunden. 1956 erhielten die Erfinder dafür den Physik-Nobelpreis. Innerhalb von zehn Jahren revolutionierte der Transistor den Computer und in den späten 1950ern waren Computer mit Vakuumröhren überholt. Der erste mit Transistoren bestückte Computer wurde am Lincoln Laboratory des M.I.T. entwickelt – eine 16-Bit-Maschine nach Art von Whirlwind I. Der Computer trug den Namen **TX-0** (**Transistorized eXperimental computer 0**) und diente lediglich als Testgerät für einen besseren TX-2.

Der TX-2 erlangte nie größere Bedeutung, aber einer der Konstrukteure, Kenneth Olsen, gründete 1957 eine Firma, die er Digital Equipment Corporation (DEC) nannte. Er begann mit der Entwicklung einer kommerziellen Maschine, die dem TX-0 sehr ähnelte: die PDP-1. Die Fertigstellung dauerte vier Jahre, weil die Investoren, mit deren Geld DEC vorwiegend

arbeitete, der Meinung waren, es gäbe keinen Markt für Computer. Schließlich hat der frühere Direktor von IBM T.J. Watson einmal gesagt, dass der Weltmarkt für Computer bei vier oder fünf Exemplaren erschöpft wäre. Stattdessen verkaufte DEC vor allem kleine Leiterplatten, die andere Firmen in ihre Produkte einbauten.

Als die PDP-1 schließlich 1961 auf den Markt kam, verfügte sie über 4096 Wörter zu je 18 Bit und konnte 200.000 Befehle in der Sekunde ausführen. Das entsprach der halben Leistung des IBM-Computers 7090, dem transistorisierten Nachfolger des 709, dem zur damaligen Zeit schnellsten Computer der Welt. Die PDP-1 kostete 120.000 US-Dollar, ein 7090-Rechner kostete Millionen. DEC verkaufte Dutzende des PDP-1. Die Minicomputer-Industrie war geboren.

Eine der ersten PDP-1 wurde dem M.I.T. zur Verfügung gestellt, wo die Maschine schnell die Aufmerksamkeit einiger der jungen Genies erregte, die das M.I.T. so reichlich hervorbrachte. Eine der zahlreichen Innovationen der PDP-1 war eine visuelle Anzeige und die Möglichkeit, Punkte an beliebiger Stelle auf ihrem Bildschirm von 512 mal 512 Bildpunkten darzustellen. Innerhalb kürzester Zeit hatten einige Studenten die PDP-1 für das Spiel Spacewar umprogrammiert. Die Welt hatte ihr erstes Videospiel.

Ein paar Jahre später führte DEC die PDP-8 ein, eine gegenüber der PDP-1 wesentlich billigere (16.000 US-Dollar) 12-Bit-Maschine. Die PDP-8 wies eine wichtige Neuerung auf: einen einzelnen Bus, den Omnibus (▶Abbildung 1.5). Ein **Bus** ist ein Leitungssystem, dessen (parallel geführte) Leitungen die Komponenten eines Computers verbinden. Diese Architektur stellte einen entscheidenden Wendepunkt gegenüber der speicherzentrierten IAS-Maschine dar und wurde seitdem von fast allen Kleincomputern übernommen. Mit 50.000 verkauften PDP-8 etablierte sich DEC als Marktführer im Minicomputer-Geschäft.

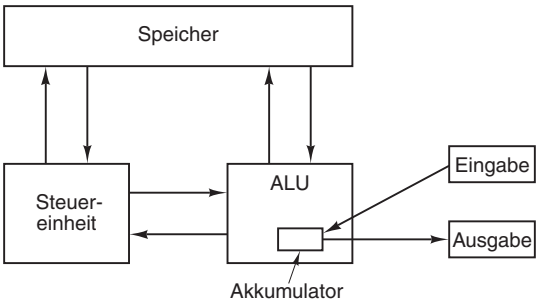

Abbildung 1.5: Der Omnibus der PDP-8

Zwischenzeitlich reagierte IBM auf den Transistor mit der Produktion einer transistorisierten Version des 709, dem oben erwähnten Modell 7090 und später dem 7094. Das Modell 7094 wies eine Zykluszeit von 2 µs und einen Kernspeicher mit 32.768 Wörtern zu je 36 Bit auf. Die Modelle 7090 und 7094 markierten das Ende der Maschinen vom Typ ENIAC, beherrschten aber die wissenschaftliche Computerszene in den sechziger Jahren.

In dieser Zeit profilierte sich IBM mit dem 7094-Computer im wissenschaftlichen Anwendungsbereich. Mit einem kleinen Computer (dem 1401) für den kommerziellen Bereich machte die Firma ein Vermögen. Dieser Computer konnte fast so schnell Magnetbänder lesen/beschreiben, Karten lesen/stanzen und drucken wie der 7094, aber zu einem Bruchteil des Preises. Für den wissenschaftlichen Bereich war der 1401 gänzlich ungeeignet, für Büroanwendungen aber perfekt.

Der 1401 war insofern ungewöhnlich, als er keine Register und auch keine feste Wortlänge hatte. Der Speicher umfasste 4000 Bytes zu je 8 Bit, spätere Modelle unterstützten sogar bis zu 16.000 Bytes – eine für die damalige Zeit verblüffende Kapazität. Jedes Byte enthielt ein 6-Bit-Zeichen, ein Verwaltungsbit und ein Bit zur Kennzeichnung des Wortendes. Zum Beispiel hatte ein `MOVE`-Befehl eine Quell- und eine Zieladresse und transportierte so lange Bytes von der Quelle zum Ziel, bis er auf ein Byte mit einem auf 1 gesetztes Wortende-Bit stieß.

1964 führte eine winzige unbekannte Firma, die Control Data Corporation (CDC), den Computer 6600 ein, der um fast eine Größenordnung schneller war als das mächtige Modell 7094 und jede andere zur damaligen Zeit existierende Maschine. Für die Zahlenakrobaten war der Computer Liebe auf den ersten Blick und die erfolgreiche Karriere von CDC nahm ihren Anfang. Das Geheimnis seiner Geschwindigkeit – und der Grund für den wesentlich niedrigeren Preis gegenüber dem 7094 – lag darin, dass in der CPU eine parallele Maschine arbeitete. Sie hatte mehrere Funktionseinheiten für die Addition, andere für die Multiplikation und wieder andere für die Division, und alle konnten parallel laufen. Um diesen Computer optimal zu nutzen, war zwar sorgfältige Programmierung vonnöten, doch mit einigem Aufwand ließen sich zehn Befehle gleichzeitig ausführen.

Erstmals wurde mit der CDC 6600, die von Seymour Cray entwickelt wurde, die parallele Verarbeitung von Befehlen zur **Steigerung der Rechengeschwindigkeit** eingesetzt. Dieses Prinzip verwenden alle heutigen Prozessoren bis hin zu heutigen Supercomputern.

Als ob das noch nicht genug war, werkelten in der 6600-Maschine mehrere kleine Computer, die ihr nach dem Motto „Schneewittchen und die sieben Kleinwüchsigen[2]“ hilfreich zur Seite standen. Das bedeutete, dass die CPU ihre ganze Zeit mit der Verarbeitung riesiger Zahlenmengen zubringen konnte, während sie alle Details der Job-Verwaltung und Ein-/Ausgabe an die kleineren Computer delegierte. Rückblickend lässt sich feststellen, dass der 6600-Computer seiner Zeit um Jahrzehnte voraus war. Viele Grundkonzepte moderner Computer lassen sich geradewegs auf den 6600 zurückführen.

Seymour Cray, der Entwickler des 6600-Computers, war eine genauso legendäre Persönlichkeit wie von Neumann. Er widmete sein ganzes Leben der Entwicklung immer schnellerer Maschinen, die man heute **Supercomputer** nennt, darunter 6600, 7600 und Cray-1. Außerdem erfand er den heute berühmten Algorithmus für den Kauf von Autos: Du gehst zum nächstliegenden Händler, zeigst auf das Auto gleich neben der Tür und sagst: „Ich nehme dieses hier.“ Dieser Algorithmus vergeudet die wenigste Zeit für unwichtige Dinge (wie Autos kaufen), sodass mehr Zeit für Wichtiges (wie Supercomputer entwickeln) bleibt.

In dieser Ära der zweiten Generation kamen viele andere Computer auf den Markt, von denen sich aber nur einer aus einem völlig anderen Grund abhebt: Burroughs B5000. Die Entwickler von Maschinen wie PDP-1, 7094 und 6600 konzentrierten sich voll auf die Hardware, um sie billiger (DEC) oder schneller (IBM und CDC) zu machen. Software war fast völlig belanglos. Die B5000-Entwickler schlugen eine andere Richtung ein. Sie entwickelten eine Maschine speziell mit der Absicht, sie in ALGOL 60, einem Vorläufer von C und Java, programmieren zu lassen, und bauten viele Funktionsmerkmale in die Hardware ein, um die Arbeit des Compilers zu vereinfachen. Das Konzept, dass auch Software eine wichtige Rolle spielt, war geboren. Leider hatte man es genauso schnell wieder vergessen.

2 Ein Wortspiel des Autors („Snow White and the Seven Vertically Challenged People“), das übertriebene „Political Correctness“ andeutet, d.h. diskriminierende Wörter – hier das Wort „Zwerg“ – vermeiden soll.

1.2.4 Die dritte Generation – integrierte Schaltungen (1965–1980)

Die Erfindung des integrierten Schaltkreises auf einem Siliziumchip durch (die unabhängig voneinander arbeitenden) Jack Kilby und Robert Noyce im Jahr 1958 ermöglichte es, Dutzende von Transistoren auf einen einzigen Chip zu packen. Dadurch konnten kleinere, schnellere und billigere Computer gebaut werden. Einige wichtige Computer dieser Generation werden in den folgenden Abschnitten beschrieben.

1964 war IBM die führende Computerfirma mit einem großen Problem in Bezug auf die beiden besonders erfolgreichen Rechner 7094 und 1401: Sie konnten inkompatibler nicht sein. Bei dem einen handelte es sich um einen Hochgeschwindigkeitsrechner mit paralleler Binärarithmetik auf 36-Bit-Registern. Der andere war ein glorifizierter Ein-/Ausgabeprozessor mit serieller Dezimalarithmetik und Speicherung von Wörtern mit variabler Länge. Viele Großkunden von IBM besaßen beide und es gefiel ihnen gar nicht, zwei getrennte Programmierabteilungen unterhalten zu müssen, die nichts hatten, was sie verbindet.

Als die Zeit kam, diese beiden Serien abzulösen, tat IBM einen radikalen Schritt. Man führte mit dem System/360 eine einzige Produktlinie auf der Grundlage integrierter Schaltungen für wissenschaftliche *und* kommerzielle Rechenumgebungen ein. Das System/360 wies viele Innovationen auf, von denen eine besonders hervorstach: Es handelte sich um eine Familie mit etwa einem halben Dutzend Rechnern mit der gleichen Assemblersprache, die sich nur hinsichtlich Größe und Leistung unterschieden. Ein Unternehmen konnte also seinen 1401 durch ein System/360, Modell 30, und einen 7094 durch ein System/360, Modell 75, ablösen. Modell 75 war größer und schneller (und teurer); die dafür geschriebene Software konnte aber prinzipiell auch auf einem anderen Modell eingesetzt werden. In der Praxis lief dies so ab, dass die für ein kleines Modell geschriebene Software problemlos auf einem größeren Modell lief, das jeweilige Programm auf einem kleineren Modell aber meist nicht in den Speicher passte. Dennoch war das im Vergleich zur 7094- und 1401-Situation eine enorme Verbesserung. Das Konzept von Rechnerfamilien fand sofort Anklang und innerhalb weniger Jahre boten die meisten Computerhersteller Rechnerfamilien über eine große Preis- und Leistungsspanne an. ▶Tabelle 1.2 nennt einige Merkmale der ersten 360-Familie. Andere Modelle wurden später eingeführt.

Eigenschaft	Modell 30	Modell 40	Modell 50	Modell 65
Relative Leistung bezogen auf Modell 30	1	3,5	10	21
Zykluszeit (ns)	1000	625	500	250
Maximaler Speicher (KB)	64	256	256	512
Pro Zyklus eingelesene Bytes	1	2	4	16
Maximale Anzahl von Datenkanälen	3	3	4	6

Tabelle 1.2: Die erste 360-Produktlinie von IBM

Eine weitere Neuerung des 360-Rechners war der **Mehrprogrammbetrieb** (**Multiprogramming**). Dabei befanden sich mehrere Programme gleichzeitig im Speicher. Während das eine auf Ein- und Ausgaben wartete, konnte das andere rechnen. Das resultierte in einer besseren Ausnutzung der CPU.

Das System/360 war auch der erste Rechner, der andere Computer emulieren (simulieren) konnte. Die kleineren Modelle emulierten den 1401, die größeren den 7094, sodass Kunden weiterhin ihre alten Binärprogramme unverändert ausführen konnten, während sie auf den 360 umstellten. Einige Modelle führten 1401-Programme viel schneller als der 1401 aus, was dazu führte, dass viele Kunden ihre Programme nie konvertierten.

Eine Emulation war auf dem 360 einfach auszuführen, weil alle anfänglichen und die meisten späteren Modelle auf Mikroprogrammierung basierten. IBM musste lediglich drei Mikroprogramme – für den jeweiligen Befehlssatz von 360, 1401 und 7094 – schreiben. Diese Flexibilität war einer der Hauptgründe, die Mikroprogrammierung im 360 einzuführen. Die Motivation von Wilkes, die Anzahl der Röhren zu verringern, spielte keine Rolle mehr, weil ja der 360 überhaupt keine Röhren mehr enthielt.

Der 360 löste das Dilemma „parallel binär/seriell dezimal“ durch einen Kompromiss: Die Maschine hatte 16 32-Bit-Register für binäre Arithmetik, aber – wie der 1401 – einen byteorientierten Speicher und serielle Befehle zum Transportieren von Datensätzen mit variabler Größe im Speicher.

Ein weiteres hervorstechendes Merkmal des 360 war der (für damalige Begriffe) riesige Adressraum von 2^{24} (16.777.216) Bytes. Bei den damaligen Speicherpreisen von mehreren US-Dollar pro Byte erschienen 16 MB schier grenzenlos. Die 360-Serie wurde später von den Serien 370, 4300, 3080 und 3090 abgelöst – leider alle mit der gleichen Architektur. Mitte der achtziger Jahre erwies sich die 16-MB-Grenze nämlich als echtes Problem. IBM musste die Kompatibilität teilweise aufgeben, als die Umstellung auf 32-Bit-Adressen erfolgte, die für die Adressierung des neuen 2^{32}-Byte-Speichers nötig war.

Rückblickend mag man sich fragen, warum keine 32-Bit-Adressen verwendet wurden, da man ja ohnehin mit 32-Bit-Wörtern und -Registern arbeitete, damals aber konnte sich niemand einen Rechner mit 16 MB vorstellen. Auch wenn der Übergang zu 32-Bit-Adressen für IBM erfolgreich vonstatten ging, war es nur eine vorübergehende Lösung für das Problem der Speicheradressierung, da Rechensysteme bald nach einer Möglichkeit verlangten, mehr als 2^{32} (4.294.967.296) Bytes im Speicher zu adressieren. Wenige Jahre später würden Computer mit 64-Bit-Adressen auftauchen.

Mit der Einführung der Serie PDP-11, einem 16-Bit-Nachfolger der PDP-8, durch DEC vollzog auch die Minicomputer-Welt große Schritte hin zur dritten Generation. Auf vielerlei Art und Weise waren die PDP-11- und PDP-8-Serien wie kleinere Brüder der 360- und 7094-Serie, so wie die PDP-1 ein kleinerer Bruder des 7094 war. Sowohl der 360 als auch die PDP-11 hatten wortorientierte Register und einen byteorientierten Speicher und beide wiesen ein breitgefächertes Preis-Leistungs-Verhältnis auf. Die PDP-11 war besonders an Universitäten sehr erfolgreich und bescherte DEC einen Wettbewerbsvorsprung gegenüber anderen Minicomputer-Herstellern.

1.2.5 Die vierte Generation – VLSI (1980 – ?)

In den 1980er Jahren bot sich mit der Höchstintegration oder **VLSI** (**Very Large Scale Integration**) die Möglichkeit, zuerst etliche Tausende, dann Hunderttausende und schließlich Millionen von Transistoren auf einen einzigen Chip zu packen. Diese Entwicklung führte zu kleineren und schnelleren Computern. Vor der PDP-1 waren Computer so groß und teuer, dass sich Unternehmen und Universitäten für deren Betrieb sogenannte **Rechenzentren** einrichten mussten. Mit dem Aufkommen des Minicomputers war es

erstmals möglich, einzelne Abteilungen mit eigenen Computern auszustatten. Bis 1980 waren die Preise derart gesunken, dass sich auch Privatpersonen einen Computer leisten konnten. Die Ära des Personalcomputers (PC) hatte begonnen.

Personalcomputer wurden völlig anders als Großrechner eingesetzt. Man nutzte sie für Textverarbeitung, Tabellenkalkulation und zahlreiche interaktive Anwendungen (wie zum Beispiel Spiele), mit denen Großrechner nicht gut umgehen konnten.

Die ersten Personalcomputer wurden als Bausatz verkauft. Ein Bausatz umfasste eine gedruckte Leiterplatte (PCB – Printed Circuit Board), eine Reihe von Chips – darunter meist ein Intel 8080 –, einige Kabel, ein Netzteil und vielleicht ein 8-Zoll-Diskettenlaufwerk. Dem Käufer wurde es überlassen, diese Teile zu einem Computer zusammenzubauen. Software war nicht dabei. Die musste man sich nach Bedarf selbst schreiben. Später wurde das Betriebssystem CP/M von Gary Kildall auf 8080-Computern populär. Dabei handelte es sich um ein Betriebssystem, das von Diskette aus ausgeführt wurde, ein Dateisystem hatte und Befehle entgegennahm, die der Benutzer von der Tastatur in einen Kommandoprozessor (Shell) eintippte.

Weitere Personalcomputer aus der Frühzeit waren der Apple und später der Apple II, die von Steve Jobs und Steve Wozniak in der sprichwörtlichen Garage gebaut wurden. Dieser Rechner war besonders bei Heimanwendern und an Schulen beliebt und machte Apple über Nacht zu einem ernsthaften Konkurrenten.

Nach reiflichen Überlegungen und langer Beobachtung der Konkurrenz entschloss sich IBM als die herrschende Kraft auf dem damaligen Computermarkt, in das PC-Geschäft einzusteigen. Anstatt den gesamten Rechner von Grund auf ausschließlich mit IBM-Teilen neu zu entwickeln, was viel zu lange gedauert hätte, unternahm IBM etwas recht Untypisches: Das Unternehmen gab einem seiner leitenden Mitarbeiter, Philip Estridge, einen Sack voll Geld und die Anweisung, sich von den Bürokraten im Stammhaus in Armonk, New York, weit zu entfernen und erst wieder zurückzukommen, wenn er einen funktionierenden Personalcomputer vorzeigen konnte. Estridge schlug in Boca Raton in Florida sein Quartier auf, wählte den Intel 8088 als seine CPU und baute den IBM-Personalcomputer aus bereits am Markt erhältlichen Teilen. Dieser wurde 1981 eingeführt und avancierte sofort zum bestverkauften Computer aller Zeiten. Als der PC seinen 30. Geburtstag feierte, erschienen zahlreiche Artikel über seine Geschichte, wovon insbesondere die von Bradley (2011), Goth (2011), Bride (2011) und Singh (2011) zu nennen sind.

IBM unternahm noch einen weiteren untypischen Schritt, den man später aber bereuen sollte. Anstatt das Design der Maschine – wie für das Unternehmen üblich – absolut geheim zu halten (oder zumindest durch eine Mauer von Patenten zu schützen), veröffentlichte es sämtliche Unterlagen, einschließlich der Schaltpläne, in einem Buch, das jeder für 49 US-Dollar im Handel erwerben konnte. Damit sollte anderen Unternehmen die Möglichkeit geboten werden, Steckkarten für den IBM-PC herzustellen, um die Flexibilität und Verbreitung des Geräts zu fördern. Da das Design somit vollständig öffentlich und alle Teile leicht von kommerziellen Anbietern erhältlich waren, stellten zahlreiche andere Unternehmen zum Leidwesen von IBM PC-**Klone** des Personalcomputers her – meist preisgünstiger als das Original von IBM. So entstand ein ganz neuer Industriezweig.

Obwohl andere Firmen – darunter Commodore, Apple und Atari – Personalcomputer herstellten, deren Prozessoren nicht von Intel stammten, war die Triebkraft der IBM-PC-Industrie so groß, dass alle anderen wie von einer Dampfwalze überrollt wurden. Die wenigen Überlebenden mussten auf Marktnischen ausweichen.

Personalcomputer: Durch die Höchstintegration von Transistoren wurden Computer auch für private Anwender erschwinglich. Die Einführung einer grafischen Benutzeroberfläche im Apple Macintosh ermöglichte auch eine leichtere Bedienung und hat sich in heutigen Computern bis hin zu Smartphones und Tablets durchgesetzt.

Zu denjenigen, die knapp überlebt hatten, gehörte der Apple Macintosh. Er wurde 1984 als Nachfolger des erfolglosen Apple Lisa eingeführt und ist der erste Computer mit einer **grafischen Benutzeroberfläche** (**GUI – Graphical User Interface**), ähnlich der heute bekannten Windows-Oberfläche. Der Lisa ist gescheitert, weil er zu teuer war, doch der ein Jahr später eingeführte preiswertere Macintosh war ein riesiger Erfolg und entfachte Feuer und Leidenschaft unter seinen vielen Bewunderern.

Der frühe Personalcomputermarkt führte ebenfalls zu dem damals fast unerhörten Verlangen nach tragbaren Computern. Zu dieser Zeit war ein portabler Computer genauso sinnvoll wie es heutzutage ein portabler Kühlschrank wäre. Der erste wirklich tragbare Personalcomputer war der Osborne-1, der allerdings mit seiner Masse von 11 kg mehr an einen Koffercomputer erinnerte. Dennoch trat er den Beweis an, dass portable Geräte möglich sind. Der wirtschaftliche Erfolg des Osborne-1 hielt sich in Grenzen. Doch ein Jahr später brachte die Firma Compaq ihren ersten tragbaren IBM-PC-Klon heraus und etablierte sich schnell als Marktführer für portable Computer.

Die ursprüngliche Version des IBM-PC war mit dem von der damals winzigen Firma Microsoft Corporation gelieferten Betriebssystem MS-DOS ausgestattet. Da Intel zunehmend leistungsstärkere CPUs produzierte, war es IBM und Microsoft möglich, einen Nachfolger für MS-DOS namens OS/2 zu entwickeln. Dieses Betriebssystem hatte eine mit dem Apple Macintosh vergleichbare grafische Benutzeroberfläche. Für den Fall, dass OS/2 nicht erfolgreich sein sollte, entwickelte Microsoft parallel dazu sein eigenes Betriebssystem weiter – das auf MS-DOS aufsetzende Windows. Kurz: OS/2 war nicht erfolgreich, IBM und Microsoft hatten einen äußerst heftigen Streit und Microsoft machte aus Windows einen Riesenerfolg. Wie die winzige Firma Intel und die noch winzigere Firma Microsoft es schafften, IBM – eines der größten, reichsten und mächtigsten Unternehmen in der Geschichte – zu entthronen, ist eine Geschichte, die zweifellos bis ins kleinste Detail in den Wirtschaftsfakultäten aller Welt eingehend diskutiert wird.

Aufbauend auf dem Erfolg des 8088 machte sich Intel daran, größere und bessere Versionen dieses Mikroprozessors herzustellen. Besonders zu erwähnen ist der im Jahre 1985 erschienene 80386, eine 32-Bit-CPU. Daran schloss sich eine aufgerüstete Version an, die natürlicherweise 80486 genannt wurde. Darauffolgende Versionen dieser Prozessorfamilie erhielten die Namen Pentium und Core. Diese Chips sind in nahezu allen modernen PCs zu finden. Für die Architektur dieser Prozessoren hat sich die Sammelbezeichnung **x86** eingebürgert. Die von AMD produzierten kompatiblen Chips laufen ebenfalls unter x86.

Mitte der achtziger Jahre fasste eine neue Entwicklung namens RISC Fuß, die die komplizierten (CISC-)Architekturen durch viel einfachere (und schnellere) ablöste. In den 1990er Jahren tauchten erstmals superskalare CPUs auf. Diese Maschinen konnten mehrere Befehle gleichzeitig ausführen, und dies oft in einer anderen Reihenfolge, als sie im Programm auftauchten. Die CISC-, RISC- und superskalaren Konzepte werden in *Kapitel 2* vorgestellt und das ganze Buch hindurch ausführlich behandelt.

Ebenfalls in der Mitte der 1980er Jahre entwickelte Ross Freeman mit seinen Kollegen bei Xilinx einen geschickten Ansatz, um integrierte Schaltungen zu erstellen, ohne dass dafür Schubkarren voller Geld oder der Zugang zu einer Halbleiterfabrik erforderlich waren. Diese als **FPGA** (**Field Programmable Gate Array**) bezeichnete neue Art von Computerchip enthält ein großes Reservoir an generischen Logikgattern, die sich zu einer beliebigen Schaltung, die in das Bauelement passt, „programmieren" lassen. Dieser bemerkenswerte neue Ansatz für den Hardwareentwurf machte FPGA-Hardware genauso geschmeidig wie Software. Mit FPGAs im Preissegment von einigen zehn bis hundert US-Dollar wurde es möglich, Rechensysteme für spezielle Anwendungen aufzubauen, die für nur wenige Benutzer gedacht sind. Erfreulicherweise kann die Halbleiterindustrie trotzdem noch schnellere, energiesparendere und billigere Chips herstellen für Anwendungen, die Millionen von Chips benötigen. Doch bei Anwendungen mit nur wenigen Benutzern – wie beispielsweise im Prototyping, in Design-Anwendungen mit geringem Umfang und in der Ausbildung – bleiben FPGAs ein beliebtes Werkzeug für den Aufbau von Hardware.

Bis 1992 waren Personalcomputer für eine Verarbeitungsbreite von 8, 16 oder 32 Bit ausgelegt. Dann brachte DEC den revolutionären 64-Bit-Alpha heraus, einen echten 64-Bit-RISC-Prozessor, der alle anderen Personalcomputer leistungsmäßig weit hinter sich ließ. Der Erfolg des Alpha war recht bescheiden. Es dauerte fast ein Jahrzehnt, bis 64-Bit-Prozessoren in größerem Umfang eingesetzt wurden – und dann vor allem als High-End-Server.

In den 1990er Jahren sind die Rechensysteme aufgrund einer Vielzahl von Mikroarchitekturoptimierungen immer schneller geworden. Viele dieser Optimierungen untersuchen wir in diesem Buch. Benutzer derartiger Systeme werden von Computeranbietern verwöhnt, da jedes neu gekaufte System ihre Programme schneller ausführen würde als ihr altes System. Allerdings flaute diese Tendenz gegen Ende der 1990er Jahre ab, was zwei wichtigen Hindernissen im Computerentwurf zuzuschreiben ist: Architekten gingen die Tricks aus, um ihre Programme schneller zu machen, und es wurde zu teuer, die Prozessoren zu kühlen. Obwohl weiterhin immer schnellere Prozessoren gebaut wurden, gingen die meisten Computerfirmen zu parallelen Architekturen über, um mehr Leistung aus ihrem Silizium herauszupressen. IBM führte 2001 die Dual-Core-Architektur POWER4 ein. Damit vereinte eine Mainstream-CPU erstmals zwei Prozessoren auf demselben Chip. Heute sind bei den meisten Prozessoren der Desktop- und Serverklasse, ja selbst bei einigen eingebetteten Prozessoren mehrere Prozessoren auf einem Chip zu finden. Die wahre Leistungsfähigkeit dieser Multiprozessoren ist für den typischen Benutzer leider kaum zu spüren, da (wie wir in späteren Kapiteln noch sehen werden) parallele Maschinen Programmierer voraussetzen, die ihre Programme explizit parallelisieren – eine schwierige und fehlerträchtige Aufgabe.

1.2.6 Die fünfte Generation – leistungsarme und unsichtbare Computer

Im Jahre 1981 hat die japanische Regierung angekündigt, die Industrie des Landes mit 500 Millionen US-Dollar bei der Entwicklung von Computern der fünften Generation zu unterstützen. Diese sollten auf künstlicher Intelligenz basieren und einen Quantensprung gegenüber den „stummen" Computern der vierten Generation darstellen. Angesichts der Tatsache, dass japanische Firmen ohnehin schon in vielen Industriezweigen den Markt beherrschten, ob es sich nun um Kameras, Stereoanlagen oder Fernsehgeräte handelte, läuteten bei amerikanischen und europäischen Computerherstellern umgehend die Alarmglocken. Von Panik erfasst forderte man Regierungssubventionen und

mehr. Doch trotz vollmundiger Ankündigungen ist das japanische Projekt der fünften Generation grundsätzlich gescheitert und lautlos in der Versenkung verschwunden. In gewissem Sinne verhält es sich wie mit der Analytical Engine von Babbage – eine visionäre Idee, doch so weit ihrer Zeit voraus, dass die Technik zu ihrer Umsetzung nirgendwo in Sicht ist.

Trotzdem hat sich das ereignet, was man als fünfte Generation bezeichnen könnte, allerdings nicht in der erwarteten Weise: Computer sind geschrumpft. 1989 brachte GRiD Systems den ersten Tablet-PC heraus, GRiDPad genannt. Er bestand aus einem kleinen Bildschirm, auf dem die Benutzer mit einem speziellen Stift schreiben konnten, um das System zu steuern. Systeme wie das GRiDPad bewiesen, dass Computer nicht auf einem Schreibtisch oder in einem Serverraum stehen müssen. Stattdessen lassen sie sich in leicht transportierbare Gehäuse mit Touchscreens und Handschrifterkennung einbauen, um ihren Wert zu erhöhen.

Der 1993 veröffentlichte Apple Newton hat gezeigt, dass sich ein Computer in einem Gehäuse unterbringen lässt, das nicht größer als ein tragbares Audiokassettengerät ist. Wie das GRiDPad arbeitet der Newton mit Handschrifterkennung, was sich zwar als Stolperstein herausgestellt hat, doch spätere Geräte dieser Klasse, die man jetzt als **PDAs** (**Personal Digital Assistants**) bezeichnet, brachten wesentlich verbesserte Benutzeroberflächen mit und erfreuen sich großer Beliebtheit. Mittlerweile haben sich derartige Geräte zu **Smartphones** gemausert.

Schließlich wurde die Handschrifteingabe des PDA durch Jeff Hawkins perfektioniert. Er gründete die Firma Palm, um einen kostengünstigen PDA für den privaten Massenmarkt zu entwickeln. Hawkins war gelernter Elektrotechniker, hatte aber ein leidenschaftliches Interesse an der Neurowissenschaft, d.h. der Untersuchung der menschlichen Denkprozesse. Er begriff, dass die Handschrifterkennung zuverlässiger funktioniert, wenn man dem Benutzer beibringt, in einer Weise zu schreiben, die für Computer leichter zu lesen ist. Diese Eingabetechnik nannte er „Graffiti“. Sie verlangte seitens des Benutzers nur wenig Übung, führte aber zu schnelleren und zuverlässigeren Schreibeingaben. So war auch der erste PDA von Palm, der Palm Pilot, ein riesiger Erfolg. Graffiti gehört zu den herausragenden Fortschritten in der Rechentechnik, denn sie demonstriert, wie sich die Leistung des menschlichen Gehirns umsetzen lässt.

Benutzer der PDAs waren auf ihre Geräte eingeschworen und verwendeten sie ehrfürchtig, um ihre Zeitpläne und Kontakte zu verwalten. Als sich Anfang der 1990er Jahre Mobiltelefone immer weiter verbreiteten, ergriff IBM die Gelegenheit, das Mobiltelefon mit dem PDA zusammenzufassen, wobei das „Smartphone“ herauskam. Das erste Smartphone, **Simon** genannt, realisierte die Eingabe über einen Touchscreen und bot dem Benutzer alle Funktionen eines PDA sowie zusätzlich ein Telefon, Spiele und E-Mail. Die abnehmenden Größen und Kosten der Bauelemente mündeten schließlich in der breiten Nutzung von Smartphones, die im bekannten Apple iPhone und den Android-Plattformen von Google verkörpert sind.

Doch selbst die PDAs und Smartphones sind nicht wirklich revolutionär. Noch bedeutender sind die „unsichtbaren“ Computer, die in Apparaten, Uhren, Kreditkarten und zahlreichen anderen Geräten eingebettet sind [Bechini et al., 2004]. Diese Prozessoren erlauben zunehmende Funktionalität bei geringeren Kosten in einer breiten Palette von Anwendungen. Es lässt sich darüber streiten, ob diese Chips eine wirkliche Generation bilden (eigentlich gibt es sie schon seit den 1970er Jahren), aber sie revolutionieren die Arbeitsweise Tausender Apparate und anderer Geräte. Bereits heute wirken sie sich auf die gesamte Welt aus und ihr Einfluss wird in den kommenden Jahren schnell zuneh-

men. Ein ungewöhnlicher Aspekt dieser eingebetteten Computer ist, dass Hardware und Software oftmals durch **Co-Design** entstehen [Henkel et al., 2003]. Wir kommen später in diesem Buch noch darauf zurück.

Ubiquitous Computing: Computer sind in Form von Smartphones und Tablets, Smartcards und Steuerungselektronik heute allgegenwärtig.

Betrachtet man die erste Generation als Maschinen mit Vakuumröhren (z.B. ENIAC), die zweite Generation als transistorisierte Computer (z.B. IBM 7094), die dritte Generation als erste Maschinen mit integrierten Schaltungen (z.B. IBM 360) und die vierte Generation als Personalcomputer (z.B. die Intel-Prozessoren), dann ist die eigentliche fünfte Generation mehr eine Paradigmenverschiebung als eine spezifische neue Architektur. In Zukunft werden Computer überall und in fast allen Gegenständen eingebettet sein – selbstverständlich unsichtbar. Aus dem Alltag sind sie nicht mehr wegzudenken – sie öffnen Türen, schalten das Licht ein, geben Geld aus und tun Tausende andere Dinge. Dieses von Mark Weiser hinterlassene Modell wurde ursprünglich **Ubiquitous Computing** (Allgegenwart von Computern) genannt, doch der Begriff **Pervasive Computing** (Durchdringung mit Computern) ist heute ebenso häufig anzutreffen [Weiser, 2002]. Dieses Modell wird die Welt genauso tief greifend verändern wie die industrielle Revolution. In diesem Buch gehen wir nicht weiter darauf ein, dem interessierten Leser seien [Lyytinen und Yoo, 2002], [Saha und Mukherjee, 2003] und [Sakamura, 2002] empfohlen.

1.3 Vielfalt der Computer

Der letzte Abschnitt hat die Geschichte der Computersysteme im Eiltempo durchstreift. Jetzt wenden wir uns der Gegenwart zu und werfen einen Blick in die Zukunft. Personalcomputer sind zwar die bekanntesten Computer, aber nicht die einzigen. Es lohnt sich also, die anderen Mitspieler wenigstens kurz kennenzulernen.

1.3.1 Technologische und wirtschaftliche Kräfte

Die Computerindustrie entwickelt sich wie keine andere. Die treibende Kraft kommt vor allem von der Fähigkeit der Chiphersteller, jedes Jahr mehr Transistoren auf einen Chip zu packen. Mehr Transistoren (winzige elektronische Schalter) bedeuten größere Speicher und leistungsfähigere Prozessoren. Gordon Moore, der Mitbegründer und ehemalige Direktor von Intel, hat das einmal so ausgedrückt: Wenn sich die Luftfahrtindustrie genauso schnell wie die Computertechnologie entwickelt hätte, würde ein Flugzeug 500 US-Dollar kosten und könnte die Erde in 20 Minuten mit 5 Gallonen Treibstoff umrunden. Allerdings hätte es die Größe eines Schuhkartons.

Insbesondere bemerkte Moore, als er sich auf eine Rede vor einer Industrievereinigung vorbereitete, dass eine neue Generation von Speicherchips drei Jahre nach der vorherigen eingeführt wird. Da jede neue Generation das Vierfache der alten Version an Speicherkapazität aufweist, konstatierte er, dass die Anzahl der Transistoren auf einem Chip in einem konstanten Verhältnis stieg. Folglich sagte er voraus, dass sich dieses Wachstum über die nächsten Jahrzehnte hinweg fortsetzen würde. Diese Beobachtung ist als das **Moore'sche Gesetz** bekannt geworden. Heute wird das Moore'sche Gesetz oft so formuliert, dass sich die Anzahl der Transistoren alle 18 Monate verdoppelt. Dies entspricht einer Steigerung der Transistorenzahl von etwa 60% pro Jahr. Die Größen der Speicherchips und ihre Einführungszeiten (▶Abbildung 1.6) bestätigen, dass das Moore'sche Gesetz bereits für mehr als vier Jahrzehnte gültig ist.

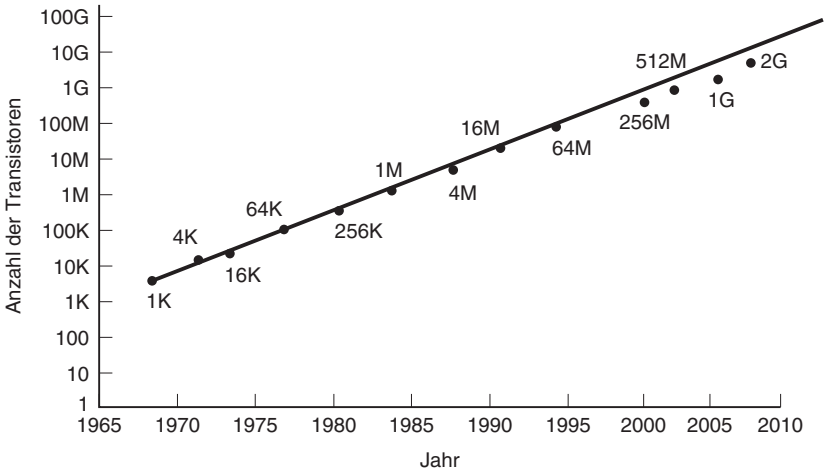

Abbildung 1.6: Gemäß dem Moore'schen Gesetz steigt die Anzahl der Transistoren, die auf einen Chip gepackt werden können, um 60% jährlich. Die Datenpunkte in diesem Bild verkörpern Speichergrößen in Bits.

Moore'sches Gesetz: Das von Gordon Moore beschriebene Wachstum der Integrationsdichte um 60% jedes Jahr gilt seit mehr als vier Jahrzehnten.

Selbstverständlich ist das Moore'sche Gesetz kein Gesetz im naturwissenschaftlichen Sinne, sondern eine empirische Beobachtung, wie schnell die Festkörperphysiker und die Verfahrenstechniker den Stand der Technik voranbringen, sowie eine Voraussage, dass diese Entwicklung künftig mit dem gleichen Tempo verläuft. Viele Branchenkenner gehen davon aus, dass das Moore'sche Gesetz mindestens noch ein Jahrzehnt, wenn nicht sogar mehr seine Gültigkeit behält. Andere Beobachter gehen jedoch davon aus, dass schon vorher ernsthafte Probleme durch Verlustleistung, Leckströme und andere Effekte auftreten und gelöst werden müssen [Bose, 2004; Kim et al., 2003]. Die Konsequenz aus der Verkleinerung der Transistoren ist aber, dass die Dicke dieser Bauelemente bald nur noch wenige Atomlagen ausmacht. An diesem Punkt werden Transistoren dann aus zu wenigen Atomen bestehen, um noch zuverlässig zu sein, oder werden einfach einen Punkt erreichen, wo eine weitere Größenverringerung Bausteine im subatomaren Bereich erfordert. (Den Mitarbeitern in einem Halbleiterbetrieb sei als guter Rat auf den Weg gegeben, sich den Tag freizunehmen, wenn entschieden wird, den einatomigen Transistor aufzuteilen!) Trotz der vielen Herausforderungen in Bezug auf die weiteren Trends des Moore'schen Gesetzes sind hoffnungsvolle Technologien in Sicht. Dazu gehören Fortschritte bei Quantencomputern [Oskin et al., 2002] sowie Kohlenstoffnanoröhren [Heinze et al., 2002], die die Möglichkeiten schaffen, die Elektronik auf Abmessungen unter die durch das Silizium vorgegebenen Grenzen zu reduzieren.

Das Moore'sche Gesetz hat etwas geschaffen, das manche Ökonomen eine **Wachstumsspirale** nennen: Technologische Fortschritte (Transistoren/Chip) führen zu besseren Produkten und niedrigeren Preisen. Niedrigere Preise führen zu neuen Anwendungen (keiner hat Videospiele für Computer hergestellt, als Computer 10 Millionen US-Dollar pro Stück kosteten, auch wenn sich Studenten am M.I.T. der Herausforderung stellten, als der Preis auf 120.000 US-Dollar gefallen war). Neue Anwendungen öffnen neue Märkte, und neue Firmen schießen aus dem Boden, um von diesen Märkten zu profitieren. Die Fülle dieser Firmen führt zu Wettbewerb, in dem technologische Weiterentwicklung für den Erfolg der Firmen von entscheidender Bedeutung ist. Der Kreis schließt sich somit.

Ein weiterer Faktor, der technologische Verbesserungen vorantreibt, ist in Nathans erstem Software-Gesetz (nach Nathan Myhrvold, einem ehemaligen Topmanager von Microsoft) beschrieben, das besagt: „Software ist ein Gas. Es dehnt sich aus und füllt den Behälter, in dem es sich befindet.“ Damals in den 1980er Jahren wurde Textverarbeitung mit Programmen wie zum Beispiel *Troff* bewältigt. Das Textsatzsystem *Troff* beansprucht nur wenige KB an Speicherplatz. Moderne Textverarbeitungsprogramme schlucken leicht mehrere MB. Künftige werden sich zweifellos im GB-Bereich bewegen. (Die Vorsätze Kilo, Mega, Giga und Tera bedeuten in erster Näherung Tausend, Million, Milliarde und Billion. Mehr dazu finden Sie in Abschnitt 1.5, Metrische Einheiten.) Software, deren Funktionsumfang laufend zunimmt, schafft eine ständige Nachfrage nach schnelleren Prozessoren, größeren Speichern und mehr Ein-/Ausgabekapazität.

Während die Anzahl der Transistoren pro Chip im Lauf der Jahre drastisch zugenommen hat, sind die Steigerungen in anderen Bereichen der Computertechnologie kaum weniger spektakulär. Zum Beispiel wurde der IBM PC/XT 1982 mit einer 10-MB-Festplatte eingeführt. Dreißig Jahre später sind bei den PC/XT-Nachfolgern 1-TB-Festplatten üblicher Standard. Diese Verbesserung von fünf Größenordnungen in dreißig Jahren repräsentiert einen jährlichen Kapazitätszuwachs von fast 50 Prozent. Allerdings ist es bei Festplatten komplizierter, vergleichbare Maße anzugeben, da neben der Kapazität auch andere Faktoren wie Datenübertragungsgeschwindigkeit, Suchzeit und Preis eine wichtige Rolle spielen. Dennoch zeigt fast jede Metrik, dass sich das Preis-Leistungs-Verhältnis seit 1982 um rund 50 Prozent jährlich verbessert hat. Diese enormen Steigerungen der Festplattenperformance und die Tatsache, dass das Dollarvolumen der im Silicon Valley produzierten Festplatten das der CPU-Chips übersteigt, haben Al Hoagland zum Vorschlag verleitet, den Standort umzubenennen: Er sollte Iron Oxide Valley heißen (da Eisenoxid das für Festplatten verwendete Aufzeichnungsmedium ist). Langsam aber verlagert sich dieser Trend wieder zugunsten des Siliziums, da Siliziumbasierte Flashspeicher in vielen Systemen die herkömmlichen Festplatten (mit rotierenden Scheiben) ersetzen.

Ein weiterer Bereich mit spektakulären Fortschritten sind Telekommunikation und Netzwerke. In weniger als zwei Jahrzehnten sind wir von 300-Bit/s-Modems über analoge Modems mit 56 Kbit/s zu Glasfasernetzen mit 10^{12} Bit/s gekommen. Transatlantische Glasfasertelefonkabel, z.B. das TAT-12/13, kosten etwa 700 Millionen US-Dollar, überdauern zehn Jahre und können 300.000 Gespräche gleichzeitig übertragen. Das entspricht weniger als einem US-Cent für ein zehnminütiges interkontinentales Gespräch. In Testanlagen haben sich optische Kommunikationssysteme mit einer Geschwindigkeit von 1 Tbit/s (10^{12} Bit/s) über Entfernungen von über 100 km ohne Verstärker als machbar erwiesen. Das exponentielle Wachstum des Internets bedarf hier keines Kommentars.

1.3.2 Das Computerspektrum

Richard Hamming, früher Wissenschaftler an den Bell Laboratories, hat einmal beobachtet, dass eine Mengenänderung von einer Größenordnung eine Qualitätsänderung nach sich zieht. Das würde bedeuten, dass ein Rennwagen, der 1000 km/h in der Wüste von Nevada schafft, eine völlig andere Maschine ist als ein normales Auto, das 100 km/h auf einer Autobahn fährt. Ein Wolkenkratzer mit 100 Stockwerken wäre dann etwas ganz anderes als ein entsprechend vergrößertes 10-stöckiges Wohnhaus. Bei Computern würden wir nicht über einen Faktor von 10, sondern von einer Million im Verlauf von drei Jahrzehnten sprechen.

Die aus dem Moore'schen Gesetz abgeleiteten Resultate lassen sich unterschiedlich umsetzen. Eine Möglichkeit wäre es, immer leistungsstärkere Computer zum gleichen Preis herzustellen. Man könnte aber auch den gleichen Computer zu einem ständig niedrigeren Preis produzieren. Die Computerindustrie hat diese und andere Möglichkeiten in vielen heute erhältlichen Computern umgesetzt. Eine grobe Klassifizierung der heutigen Computer ist in ▶Tabelle 1.3 aufgeführt.

Typ	Preis (US-$)	Anwendungsbeispiel
Wegwerfcomputer	0,5	Glückwunschkarten
Mikrocontroller	5	Uhren, Autos, Geräte
Mobile Computer und Spielkonsolen	50	Heimvideospiele und Smartphones
Personalcomputer	500	Desktop- oder Notebook-Computer
Server	5.000	Netzwerkserver
Großrechner (Mainframe)	5 Millionen	Batch-Verarbeitung in einer Bank

Tabelle 1.3: Das heute verfügbare Computerspektrum; die aufgeführten Preise dienen lediglich der groben Orientierung

Die folgenden Abschnitte untersuchen die einzelnen Kategorien und gehen kurz auf ihre Eigenschaften ein.

1.3.3 Wegwerfcomputer

Am unteren Ende finden wir einzelne Chips, die in Glückwunschkarten eingeklebt werden, um Geburtstagswünsche, Hochzeitsmärsche oder ähnlich nervige Liedchen zu plärren. Dem Autor ist bis jetzt noch keine Beileidskarte in die Hand gefallen, die einen Trauermarsch spielt. Diese Idee wird sich nun aber mit diesem Buch verbreiten, sodass man bald darauf hoffen darf. Für jeden, der mit Großrechnern zum Preis von mehreren Millionen Dollar aufgewachsen ist, sind Wegwerfcomputer so sinnvoll wie Wegwerfflugzeuge.

Inzwischen sind Wegwerfcomputer nicht mehr aus dem Alltag wegzudenken. Wahrscheinlich die wichtigste Entwicklung auf dem Gebiet der Wegwerfcomputer ist der **RFID**-Chip (**Radio Frequency IDentification**). Heutzutage kann man batterielose RFID-Chips mit einer Kantenlänge von weniger als 0,5 mm herstellen. Sie enthalten einen winzigen Funktransponder und eine integrierte 128-Bit-Nummer. Bei Anregung von einer externen Antenne erhalten sie genügend Energie, um ihre Nummer an die Antenne zurückzusenden. Die Chips sind zwar winzig – die Konsequenzen dieser Technik zweifellos nicht.

RFID-Chips: Winzige Chips senden auf Anregung durch eine externe Antenne eine 128-Bit-Identifikation und ermöglichen so ungezählte neue Anwendungen.

Beginnen wir mit einer banalen Anwendung: dem Wegfall der Barcodes zur Produktkennzeichnung. Man hat bereits Feldversuche durchgeführt, in denen die Hersteller ihre Produkte mit RFID-Chips (anstelle von Barcodes) versehen haben. Der Kunde wählt Waren aus, legt sie in seinen Einkaufswagen und verlässt damit einfach das Geschäft, ohne an der Kasse vorbeizugehen. Am Ausgang des Ladens sendet ein Leser mit einer Antenne Signale aus, die jedes Produkt auffordern, sich selbst zu identifizieren, was durch eine kurze drahtlose Übertragung geschieht. Der Kunde wird ebenfalls durch einen Chip auf seiner Bankkarte oder Kreditkarte identifiziert. Am Monatsende schickt

das Geschäft dem Kunden eine Rechnung mit der Einzelaufstellung seiner Käufe im laufenden Monat. Falls der Kunde keine gültige RFID-Bank-/Kreditkarte besitzt, ertönt ein Alarm. Dieses System macht nicht nur den Kassierer und die an der Kasse üblichen Schlangen überflüssig, sondern es dient auch als Antidiebstahlsystem, weil es vollkommen sinnlos wird, einen Artikel in der Jackentasche oder einem Einkaufsbeutel verschwinden zu lassen.

Barcodes können zwar die Warenart identifizieren, aber nicht den einzelnen Artikel. RFID-Chips mit ihren 128 Bit sind dazu aber in der Lage. Als Folge hat jede Packung von, sagen wir, Aspirin in einem Supermarktregal einen eigenen RFID-Code. Wenn jetzt ein Arzneimittelhersteller einen Produktionsfehler in einer bereits ausgelieferten Aspirin-Charge entdeckt, kann er die Supermärkte auf der ganzen Welt anweisen, einen Alarm auszulösen, wenn ein Kunde eine Packung kauft, deren RFID-Nummer im betroffenen Bereich liegt, selbst wenn der Kauf erst Monate später und in einem anderen Land stattfindet. Aspirin, das nicht zur fehlerhaften Charge gehört, löst den Alarm nicht aus.

Doch die Kennzeichnung von Aspirinpäckchen, Keksen und Hundefutter ist erst der Anfang. Warum sollte man nur das Hundefutter kennzeichnen, wenn man den ganzen Hund markieren kann? Tierbesitzer fragen schon bei Tierärzten nach, RFID-Chips in ihre Lieblinge implantieren zu lassen, damit sie sich auch bei Verlust oder Diebstahl wieder aufspüren lassen. (Zum Teil ist dies sogar schon gesetzlich vorgeschrieben.) Landwirte möchten ihren Viehbestand kennzeichnen. Nervöse Eltern könnten den Kinderarzt bitten, ihren Kindern RFID-Chips einzupflanzen für den Fall, dass sie verloren gehen oder entführt werden. Und wenn wir schon dabei sind, warum lassen wir die Chips nicht gleich jedem Neugeborenen einsetzen, um Verwechslungen auf der Kinderstation zu vermeiden? Regierungen und Polizei finden zweifellos gute Gründe, um alle Bürger jederzeit verfolgen zu können. Mithin dürfte etwas klarer sein, was mit den eingangs erwähnten Konsequenzen gemeint ist.

Eine andere (nicht ganz so kontrovers diskutierte) Anwendung von RFID-Chips ist die Fahrzeugkennzeichnung. Passiert eine Kette von Fahrzeugen mit eingebetteten RFID-Chips auf der Autobahn einen Leser, bekommt der mit dem Leser verbundene Computer eine Liste, welche Fahrzeuge vorbeigefahren sind. Mit diesem System lassen sich leicht die Standorte aller Autobahnbenutzer verfolgen, was Lieferanten, ihren Kunden und den Autobahnbetreibern hilft. Ein ähnliches Schema lässt sich auf Lastkraftwagen anwenden. Die Mautgebühren können dann elektronisch erfasst und abgerechnet werden (wie zum Beispiel beim System E-ZPass).[3]

Gepäcksysteme von Luftfahrtunternehmen und viele andere Transportsysteme können ebenfalls auf RFID-Chips zurückgreifen. Ein experimentelles System wurde auf dem Flughafen Heathrow in London getestet. Die Passiere brauchten keine Gepäckkennzeichnung mehr an ihren Gepäckstücken anzubringen. Die Koffer der Passagiere, die diesen Service in Anspruch nahmen, wurden mit RFID-Chips gekennzeichnet, getrennt über den Flughafen befördert und direkt in den Hotels der Passagiere abgeliefert.

RFID-Chips verwendet man auch in der Automobilindustrie (beispielsweise, um die gewünschte Farbe der vom Montageband in der Lackiererei ankommenden Fahrzeuge zu bestimmen), bei der Untersuchung von Tierwanderungen, zur Steuerung der Waschmaschinentemperatur durch die Kleidungsstücke selbst und in vielen anderen Bereichen. In bestimmte Chips lassen sich Sensoren integrieren, sodass die niederwertigen Bits aktuelle Werte von Temperatur, Druck, Luftfeuchtigkeit oder anderer Umgebungsdaten enthalten können.

3 Dieses System ist vergleichbar mit Toll Collect, arbeitet aber auf der Basis von RFID.

Erweiterte RFID-Chips können einen Permanentspeicher enthalten. Diese Möglichkeit hat die Europäische Zentralbank dazu veranlasst, in den kommenden Jahren RFID-Chips in Euro-Banknoten einzusetzen. Die Chips können aufzeichnen, wo die Scheine gewesen sind. Das macht nicht nur das Fälschen von Euro-Banknoten nahezu unmöglich, sondern erlaubt auch, das Lösegeld bei Entführungen, die Beute von Überfällen und gewaschenes Geld wesentlich leichter zu verfolgen und möglicherweise aus der Ferne ungültig zu machen. Wenn das Geld nicht mehr anonym ist, könnte es in Zukunft zu den Routineaufgaben der Polizei gehören, den Weg von verdächtigen Geldscheinen zurückzuverfolgen. Wer muss schon Chips in die Bürger implantieren, wenn deren Brieftaschen voll davon sind? Auch diese Themen wird man noch heiß diskutieren, wenn der Öffentlichkeit erst einmal richtig bewusst wird, wozu RFID-Chips in der Lage sind.

Die Technik der RFID-Chips entwickelt sich schnell weiter. Die kleinsten Chips arbeiten passiv (ohne Stromversorgung) und können lediglich ihre eindeutigen Nummern auf Abfrage senden. Dagegen sind die größeren Chips aktiv, können eine kleine Batterie und einen einfachen Computer enthalten und sind in der Lage, Berechnungen durchzuführen. In diese Kategorie fallen auch die bei Finanztransaktionen verwendeten Smartcards.

RFID-Chips unterscheiden sich nicht nur darin, ob sie aktiv oder passiv arbeiten, sondern auch durch den Frequenzbereich, auf den sie reagieren. Die bei niedrigeren Frequenzen arbeitenden Chips bieten zwar eine geringe Datenrate, lassen sich aber von der Antenne über große Entfernungen erfassen. Bei Chips mit hohen Frequenzen liegt die Datenrate höher, der Erfassungsbereich ist aber geringer. Das Internet ist voll von Informationen über RFID-Chips. Als Ausgangspunkt sei die Seite *www.rfid.org* empfohlen.

1.3.4 Mikrocontroller

Die Computer auf der nächsten Stufe der Leiter sind in Geräte eingebettet, die nicht als Computer verkauft werden. Die auch als **Mikrocontroller** bezeichneten eingebetteten Computer verwalten die Geräte und realisieren die Benutzeroberfläche. Mikrocontroller findet man in den verschiedensten Geräten. Die folgende Liste gibt eine Auswahl an und nennt für jede Kategorie in Klammern einige Beispiele.

1. Haushaltsgeräte (Uhrenradio, Waschmaschine, Trockner, Mikrowellenherd, Alarmanlagen)
2. Kommunikationsgeräte (Schnurlostelefon, Mobiltelefon, Fax, Pager)
3. Computerperipheriegeräte (Drucker, Scanner, Modem, CD-ROM-Laufwerk)
4. Unterhaltungselektronik (VCR, DVD, Stereoanlage, MP3-Player, Set-Top-Box)
5. Bildverarbeitungsgeräte (TV, Digitalkamera, Camcorder, Teleskop, Fotokopierer)
6. Medizinische Geräte (Röntgenapparate, MRI, Herzmonitor, Digitalthermometer)
7. Waffensysteme (Marschflugkörper, Interkontinentalrakete, Torpedo)
8. Verkaufseinrichtungen (Warenautomat, Geldautomat, Registrierkasse)
9. Spielzeug (Sprechpuppe, Spielkonsole, ferngesteuertes Auto oder Boot)

Ein Auto kann leicht 50 Mikrocontroller enthalten, die Subsysteme ausführen, wie zum Beispiel Antiblockiersysteme, Benzineinspritzung, Radio, Beleuchtung und GPS. In einem Strahlflugzeug sind ohne Weiteres mehr als 200 Mikrocontroller zu finden. Eine Familie

besitzt vielleicht mehrere Hundert Computer, ohne es zu wissen. Innerhalb weniger Jahre wird praktisch alles, was mit Netzspannung oder Batterien läuft, einen Mikrocontroller enthalten. Die Anzahl der jedes Jahr verkauften Mikrocontroller übersteigt alle anderen Arten von Computern um Größenordnungen, mit Ausnahme der Wegwerfcomputer.

Mikrocontroller sind kleine vollständige Computer mit Prozessor, Speicher und Ein-/Ausgabe, die für Steuerungsaufgaben eingesetzt werden.

Während RFID-Chips Minimalsysteme darstellen, sind Mikrocontroller kleine, doch vollständige Computer. Jeder Mikrocontroller besteht aus Prozessor, Speicher und Ein-/Ausgabeeinrichtungen. Die Ein-/Ausgabeeinrichtungen fragen Tasten und Schalter ab, steuern Kontrolllämpchen und Displays an, erzeugen akustische Signale und schalten Motoren. In den meisten Fällen ist die Software fest im Chip verdrahtet und zwar als Nur-Lese-Speicher, dessen Inhalt bei der Herstellung des Mikrocontrollers festgelegt wird. Allgemein kann man Mikrocontroller in zwei Gruppen einteilen: Universalcontroller und Spezialcontroller. Die erste Gruppe verkörpert lediglich kleine, aber normale Computer, während bei der zweiten Gruppe Architektur und Befehlssatz auf eine bestimmte Anwendung abgestimmt sind, beispielsweise für ein Multimediagerät. Mikrocontroller sind in Versionen für 4, 8, 16 und 32 Bit Verarbeitungsbreite erhältlich.

Allerdings unterscheiden sich selbst die Universalmikrocontroller in wesentlichen Punkten vom Standard-PC. Als Erstes ist der Kostenfaktor zu nennen. Wenn eine Firma Millionen Exemplare kauft, macht sie die Entscheidung für einen bestimmten Mikrocontrollertyp von Preisunterschieden im Cent-Bereich abhängig. Aufgrund dieser Einschränkung wählen Mikrocontrollerhersteller eine Architektur, die wesentlich stärker von den Herstellungskosten bestimmt wird als bei Chips, die sowieso mehrere Hundert Dollar kosten. Die Preise von Mikrocontrollern variieren sehr stark und hängen von der Verarbeitungsbreite, der Größe und Art des Speichers sowie anderen Faktoren ab. Wenn zum Beispiel ein 8-Bit-Mikrocontroller in sehr großen Stückzahlen geordert wird, ist er vielleicht schon für rund 10 US-Cent je Exemplar zu haben. Derartige Preise machen es möglich, in einem Uhrenradio für 9,95 US-Dollar einen Mikrocontroller einzubauen.

Zweitens arbeiten praktisch alle Mikrocontroller in Echtzeit. Auf ein Eingangssignal hin sollen sie die Antwort möglichst unmittelbar liefern. Drückt beispielsweise der Benutzer eine Taste, leuchtet als Rückmeldung eine Lampe auf. Zwischen dem Tastendruck und dem Aufleuchten der Lampe darf keine spürbare Verzögerung entstehen. Der erforderliche Echtzeitbetrieb wirkt sich oftmals auf die Architektur aus.

Drittens sind bei eingebetteten Systemen meistens auch physische Einschränkungen hinsichtlich Größe, Gewicht und Stromverbrauch sowie andere elektrische und mechanische Beschränkungen zu beachten. Der Entwurf eines Mikrocontrollers für derartige Geräte muss von vornherein diese Beschränkungen berücksichtigen.

Eine besonders interessante Anwendung von Mikrocontrollern findet sich in der eingebetteten Steuerungsplattform Arduino. Der Entwurf des Arduino stammt von Massimo Banzi und David Cuartielles aus Invrea in Italien. Mit diesem Projekt wollten sie eine vollständig eingebettete Computerplattform schaffen, die weniger als eine große und reichlich belegte Pizza kostet, um sie auch für Studenten und Amateure erschwinglich zu machen. (Eine schwierige Aufgabe, weil es in Italien ein Überangebot von Pizzas gibt und sie deshalb wirklich billig sind.) Ihr Ziel haben sie erreicht: Ein vollständiges Arduino-System kostet weniger als 20 US-Dollar!

Da das Arduino-System ein Open-Source-Hardwarekonzept ist, sind sämtliche Details öffentlich und kostenlos zugänglich, sodass jeder ein Arduino-System aufbauen (und sogar verkaufen) kann. Es beruht auf dem 8-Bit-RISC-Mikrocontroller aus der AVR-

Familie von Atmel und die meisten Boardkonzepte sehen auch grundlegende E/A-Unterstützung vor. Programmiert wird das Board mit der eingebetteten Programmiersprache Wiring, die von Haus aus über allen Schnickschnack verfügt, der für die Steuerung von Geräten in Echtzeit gebraucht wird. Was den Einsatz der Arduino-Plattform so interessant macht, ist seine große und aktive Entwicklergemeinde. Es gibt Tausende veröffentlichter Arduino-Projekte – u.a. einen elektronischen Schadstoffdetektor, ein Biker-Jacket mit Fahrtrichtungsanzeigern, einen Feuchtigkeitssensor, der E-Mails verschickt, wenn eine Pflanze mit Wasser zu versorgen ist, und ein unbemanntes autonomes Flugzeug. Um mehr über den Arduino zu erfahren und sich an eigenen Arduino-Projekten zu versuchen, empfiehlt sich die Seite *www.arduino.cc*.

1.3.5 Mobile Computer und Spielkonsolen

Auf der nächsthöheren Stufe stehen mobile Plattformen und Videospielgeräte. Das sind normale Computer mit speziellen Grafik- und Soundfähigkeiten, aber begrenzter Software und fast keiner Erweiterungsmöglichkeit. Angefangen hat alles bei einfachsten CPUs für primitive Actionspiele wie Ping-Pong mit Darstellung auf einem Fernsehgerät. Im Laufe der Jahre haben sie sich zu weit leistungsfähigeren Systemen gemausert, die in bestimmten Parametern den Personalcomputern ebenbürtig oder sogar überlegen sind.

Um eine Vorstellung davon zu vermitteln, was in einer Spielkonsole steckt, sehen wir uns drei bekannte Produkte an. Erstens die PlayStation 3 von Sony: Sie enthält eine proprietäre 3,2-GHz-Mehrkern-CPU (den sogenannten Cell-Mikroprozessor), die auf der PowerPC-RISC-CPU von IBM basiert, und sieben 128-Bit-SPEs (Synergistic Processing Elements). Außerdem verfügt die PlayStation 3 über 512 MB RAM, einen speziellen 550-MHz-Grafikchip von Nvidia und einen Blu-ray-Player. Zweitens die XBox 360 von Microsoft: Sie ist mit einer 3,2-GHz-Dreikern-PowerPC-CPI von IBM, 512 MB RAM, einem speziellen 500-MHz-Grafikchip von ATI, einem DVD-Player und einer Festplatte ausgestattet. Drittens das Samsung Galaxy-Tablet (auf dem dieses Buch korrekturgelesen wurde). Es enthält zwei 1-GHz-ARM-Kerne und einen Grafikprozessor (die im Nvidia-Tegra-2-System auf einem Chip integriert sind), 1 GB RAM, zwei Kameras, ein 3-Achsen-Gyroskop und Flashspeicher.

Auch wenn diese Maschinen nicht so leistungsfähig sind wie die im gleichen Zeitraum hergestellten High-End-Personalcomputer, liegen sie doch nicht allzu weit zurück und haben in manchen Punkten sogar die Nase vorn (z.B. wartet die 128-Bit-SPE in der PlayStation 3 mit einer Verarbeitungsbreite auf, die noch kein Personalcomputer erreicht hat). Die Unterschiede zwischen Spielkonsole und PC betreffen weniger die CPU als die Tatsache, dass die Spielkonsolen geschlossene Systeme sind. Der Benutzer kann sie nicht mit Einsteckkarten erweitern, abgesehen von eventuell vorhandenen USB- oder FireWire-Schnittstellen. Der wohl wichtigste Punkt ist, dass Spielkonsolen sorgfältig für einen einzigen Anwendungsbereich optimiert sind: stark interaktive 3-D-Spiele mit Multimedia-Ausgabe. Alles andere ist zweitrangig. Diese Hardware- und Softwarebeschränkungen, der Verzicht auf Erweiterbarkeit, kleine Speicher, das Fehlen eines hochauflösenden Monitors und eine kleine (und manchmal gar nicht vorhandene) Festplatte machen es möglich, diese Geräte wesentlich billiger als Personalcomputer auf den Markt zu bringen. Trotz dieser Einschränkungen wurden Millionen dieser Geräte verkauft und ihre Anzahl wächst ständig.

Mobile Computer sollen darüber hinaus ihre Aufgaben mit möglichst wenig Energie ausführen. Je weniger Energie sie verbrauchen, desto länger halten die Batterien. Dies bedeutet hohe Anforderungen an den Entwurf, da mobile Plattformen wie Tablets und

Smartphones genügsam in ihrem Energieverbrauch sein müssen, gleichzeitig aber die Benutzer dieser Geräte eine hohe Leistungsfähigkeit erwarten, wie zum Beispiel 3-D-Grafik, hochauflösende Multimediaverarbeitung und flüssige Spielabläufe.

1.3.6 Personalcomputer

Als Nächstes kommen wir zur Kategorie der Personalcomputer (PC), an die man meistens zuerst denkt, wenn von „Computer“ die Rede ist. Dazu zählen Desktop- und Notebook-Modelle. Ihre Ausstattung umfasst normalerweise einige Gigabyte (GB) Hauptspeicher, eine Festplatte mit Kapazitäten bis in den Terabyte-Bereich (TB), ein CD-ROM-/DVD-/Blu-ray-Laufwerk, eine Soundkarte, eine Netzwerkkarte, einen hochauflösenden Monitor und andere Peripheriegeräte. Zudem verfügen sie über aufwendige Betriebssysteme, viele Erweiterungsoptionen und eine Riesenauswahl an Software.

Das Herz jedes Personalcomputers ist eine gedruckte Leiterplatte (Hauptplatine, Mainboard), die am Gehäuseboden oder an der Seite montiert ist. Sie enthält normalerweise die CPU, den Speicher, verschiedene Ein-/Ausgabebaugruppen (beispielsweise Soundchip und vielleicht ein Modem) und die Schnittstellen zu Tastatur, Maus, Festplattenlaufwerken, Netzwerk usw. sowie einige Erweiterungssteckplätze („Slots“). ▶Abbildung 1.7 zeigt eine derartige Hauptplatine der Firma Intel.

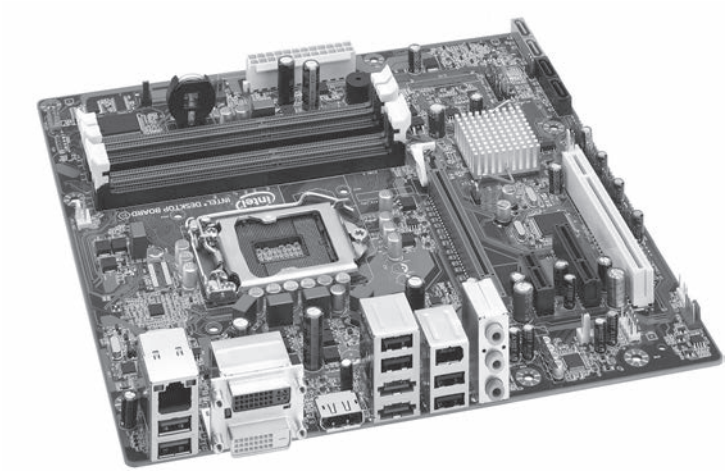

Abbildung 1.7: Eine gedruckte Leiterplatte bildet das Herz jedes Personalcomputers. Das Foto zeigt die Hauptplatine DQ67SW von Intel. [Copyright by Intel 2011, mit freundlicher Genehmigung]

Notebooks sind grundsätzlich PCs in einem kleineren Gehäuse. Sie verwenden die gleichen Hardwarekomponenten, allerdings in kleineren Ausführungen. Außerdem läuft auf ihnen die gleiche Software wie auf Desktop-PCs. Da die meisten Leser höchstwahrscheinlich mit Notebooks und Personalcomputern hinreichend vertraut sind, dürfte zusätzlicher Stoff zur Einführung kaum erforderlich sein.

Ein eng verwandter Computertyp ist der Tablet-PC wie zum Beispiel das bekannte iPad. Diese Geräte sind noch kleiner als Notebooks, besitzen ein Solid-State-Drive (Halbleiterlaufwerk) statt einer rotierenden Festplatte, einen Touchscreen und eine andere CPU als x86. Unter dem Aspekt der Architektur sind Tablets aber nichts weiter als Notebooks mit einem anderen Formfaktor.

1.3.7 Server

Aufgerüstete Personalcomputer oder Workstations setzt man häufig als Netzwerkserver sowohl für lokale Netzwerke (typischerweise innerhalb einer Firma) als auch für das Internet ein. Übliche Konfigurationen umfassen einen oder mehrere Prozessoren, einen Speicher von einigen GB, Festplatten mit Kapazitäten von mehreren TB und Anschlüsse für Hochgeschwindigkeitsnetze. Manche sind in der Lage, Tausende von Transaktionen pro Sekunde zu bewältigen.

Von der Architektur her betrachtet unterscheidet sich ein Einprozessor-Server nicht allzu sehr von einem Einprozessor-Personalcomputer. Er ist nur schneller, größer, besitzt mehr Festplattenkapazität und möglicherweise eine schnellere Netzwerkanbindung. Auf Servern laufen die gleichen Betriebssysteme wie auf Personalcomputern, in der Regel eine Variante von UNIX oder Windows.

Cluster

Zur parallelen Bearbeitung von Anwendungen werden Standard-PCs über schnelle Netzwerke zu **Clustern** gekoppelt.

Infolge der fast stetigen Verbesserungen im Preis-Leistungs-Verhältnis von Workstations und Personalcomputern hat man in den letzten Jahren damit begonnen, eine große Anzahl dieser Computer zu sogenannten **Clustern** zusammenzuschalten. Die Cluster bestehen aus Standard-PCs oder Workstations, sind über schnelle Gigabit-Netzwerke verbunden und führen spezielle Software aus, die es erlaubt, dass alle Rechner gemeinsam an der Lösung eines Problems arbeiten. Derartige Systeme setzt man insbesondere in der Wissenschaft und in der Produktentwicklung ein. Bei den einzelnen Computern spricht man von **COTS**-Computern (**Commodity Off The Shelf**), d.h. Computern, die aus Standardbauteilen bestehen. Zu den Erweiterungen gehören in erster Linie Hochgeschwindigkeitsanschlüsse, obwohl manchmal auch handelsübliche Netzwerkkarten genügen.

Große Cluster sind in der Regel in speziellen Räumen oder Gebäuden – den sogenannten Datenzentren – untergebracht, die von einer Handvoll bis hin zu 100.000 oder mehr Computern ausgestattet sein können. Grenzen werden praktisch nur durch das Budget gesetzt. Aufgrund des niedrigen Komponentenpreises können sich mittlerweile auch einzelne Firmen solche Systeme leisten. Die Begriffe „Cluster“ und „Datenzentrum“ werden häufig gleichbedeutend verwendet, auch wenn es sich aus technischer Sicht bei Clustern um eine Sammlung von Servern und bei Datenzentren um Räume oder Gebäude handelt.

Auch für Internet-Webserver kommen Cluster infrage. Wenn für die Seiten einer Website Tausende von Anforderungen je Sekunde zu erwarten sind, besteht die wirtschaftlichste Lösung oftmals in einem Cluster mit Hunderten oder sogar Tausenden von Servern. Die eintreffenden Anforderungen werden dann auf die Server aufgeteilt, um sie parallel verarbeiten zu lassen. Zum Beispiel sind die Datenzentren von Google an vielen Standorten weltweit zu finden, um Suchanfragen zu bedienen. Das größte Datenzentrum in The Dalles, Oregon, hat die Größe von zwei Football-Feldern. Die Wahl fiel auf diesen Standort, da Datenzentren riesige Mengen an elektrischer Energie verbrauchen und es in The Dalles ein 2-GW-Wasserkraftwerk am Columbia River gibt, das diesen Bedarf decken kann. Man geht davon aus, dass Google in seinen Datenzentren mehr als eine Million Server betreibt.

Das Computergeschäft ist von Dynamik gekennzeichnet, die Situation ändert sich ständig. In den 1960er Jahren waren riesige Mainframes (siehe unten) vorherrschend, die zig Millionen Dollar kosteten und an die Benutzer über kleine Remoteterminals angeschlossen waren. Dies war ein sehr zentralisiertes Modell. In den 1980er Jahren erschienen die

Personalcomputer auf dem Markt und Millionen Käufern ist es zu verdanken, dass die Rechentechnik dezentralisiert wurde.

IT-Dienste werden im **Cloud-Computing** zentralisiert über das Internet angeboten.

Mit dem Auftauchen der Datenzentren lassen wir die Vergangenheit in Form des **Cloud-Computing** wiederaufleben – gewissermaßen als Mainframe-Computing V2.0. Vom Konzept her fungieren dabei einfache Geräte der Benutzer wie PCs, Notebooks, Tablets und Smartphones als Benutzeroberflächen der Cloud (d.h. des Datenzentrums), in der sämtliche Fotos, Videos, Musik und andere Daten der Benutzer gespeichert sind. Dieses Modell macht die Daten von unterschiedlichen Geräten aus überall und jederzeit zugänglich, ohne dass der Benutzer darauf achten muss, wo er sich befindet. Hier hat das Datenzentrum voller Server den einzelnen großen zentralisierten Computer ersetzt, doch ist inzwischen alles wieder wie früher: Die Benutzer arbeiten an einfachen Terminals und mit einfachen Daten, die Rechenleistung ist an anderer Stelle zentralisiert.

Wer kann wissen, wie lange dieses Modell populär sein wird? In den nächsten 10 Jahren kann es leicht passieren, dass so viele Leute so viele Songs, Fotos und Videos in der Cloud gespeichert haben, dass die (drahtlose) Infrastruktur für die Kommunikation mit der Cloud ins Stocken gerät. Dies könnte zu einer neuen Revolution führen: Personalcomputer, auf denen die Benutzer ihre eigenen Daten auf ihren eigenen Computern lokal speichern und damit den Verkehrsstau über die Luftschnittstelle umgehen.

Unterm Strich heißt das, dass das Modell zum Einsatz von Rechnern, das in einer bestimmten Epoche populär ist, zum großen Teil von der Technologie, der Wirtschaftlichkeit und den jeweils verfügbaren Anwendungen abhängt und umschwenken kann, wenn sich diese Faktoren ändern.

1.3.8 Mainframes

In die Kategorie der Mainframes fallen raumfüllende Rechner, die auf die sechziger Jahre zurückgehen. Oftmals handelt es sich um direkte Nachfahren der IBM-Großrechner, Modell 360, die vor Jahrzehnten angeschafft wurden. Meist sind sie nicht viel schneller als leistungsstarke Server, haben aber durchgängig mehr E/A-Kapazität und sind mit großen Plattenfarmen für Tausende von GB Daten ausgestattet. Aufgrund der immensen Investitionen in Software, Daten, Betriebsprozeduren und Personal werden sie aber weiter betrieben. Viele Unternehmen finden es kostengünstiger, hin und wieder ein paar Millionen Dollar für einen neuen Rechner auszugeben, als alle vorhandenen Anwendungen auf kleinere Maschinen umschreiben zu lassen.

Es ist aber auch die Computerklasse, die das mittlerweile berüchtigte Jahr-2000-Problem verursacht hat. In den 1960er und 1970er Jahren schrieben (vorwiegend) COBOL-Programmierer die Jahreszahl in zwei Dezimalziffern (um Speicher zu sparen). Wahrscheinlich konnten sie sich nicht vorstellen, dass ihre Software drei bis vier Jahrzehnte überdauern würde. Obwohl die vorhergesagte Katastrophe ausblieb, weil das Problem mit ungeheuren Anstrengungen aus der Welt geschafft wurde, haben viele Firmen diesen Fehler wiederholt und einfach zwei weitere Ziffern an das Jahr angehängt. Der Autor erlaubt sich an dieser Stelle die Voraussage, dass die Zivilisation in der uns bekannten Form am 31. Dezember 9999 um Mitternacht endet, wenn 8000 Jahre alte COBOL-Programme allesamt gleichzeitig abstürzen.

Mainframes müssen ihr Dasein nicht mehr nur damit fristen, 40 Jahre alte Software auszuführen – das Internet hat ihnen in den letzten Jahren neues Leben eingehaucht. Sie haben eine Nische als leistungsfähige Internetserver gefunden, da sie beispielsweise rie-

sige Mengen von Transaktionen je Sekunde im E-Commerce bewältigen können, insbesondere in Unternehmen, die sehr große Datenbanken verlangen.

Bis vor kurzem gab es noch eine andere Kategorie von Computern, die noch leistungsfähiger als Mainframes sind: **Supercomputer**. Sie hatten außerordentlich schnelle CPUs, viele GB Hauptspeicher, sehr schnelle Festplatten und Netzwerkanschlüsse. Man hat sie zur Lösung rechenintensiver Probleme in Wissenschaft und Technik eingesetzt, um z.B. kollidierende Galaxien zu simulieren, neuartige Medikamente zu synthetisieren oder den Luftstrom rund um eine Flugzeugtragfläche zu modellieren. Seit einigen Jahren bieten jedoch Datenzentren, basierend auf Standardkomponenten, eine vergleichbare Rechenleistung zu wesentlich günstigeren Preisen und die echten Supercomputer gehören nun einer aussterbenden Spezies an.

1.4 Beispiele von Computerfamilien

Dieses Buch konzentriert sich auf drei verbreitete Befehlssatzarchitekturen (Instruction Set Architectures, ISAs): x86, ARM und AVR. Die x86-Architektur findet sich in nahezu allen Personalcomputern (einschließlich Windows- und Linux-PCs sowie Macs) und Serversystemen. Personalcomputer sind von Interesse, weil jeder Leser zweifellos schon mit einem gearbeitet hat. Server interessieren, weil sie alle Dienste im Internet ausführen. Die ARM-Architektur dominiert den Markt der mobilen Geräte. Zum Beispiel basieren die meisten Smartphones und Tablets auf ARM-Prozessoren. Schließlich ist die AVR-Architektur in äußerst preisgünstigen Mikrocontrollern realisiert, die in vielen eingebetteten Rechneranwendungen üblich sind. Eingebettete Computer sind zwar für ihre Benutzer unsichtbar, steuern aber Autos, Fernsehgeräte, Mikrowellenherde, Waschmaschinen und praktisch jedes andere elektrische Gerät, das mehr als 50 US-Dollar kostet.

Dieser Abschnitt bietet eine kurze Einführung zu den drei Befehlssatzarchitekturen, die wir durchgängig im gesamten Buch als Beispiele heranziehen.

1.4.1 Einführung in die x86-Architektur

Im Jahre 1968 gründeten Robert Noyce, Erfinder der integrierten Schaltung, Gordon Moore, von dem das gleichnamige Gesetz stammt, und Arthur Rock, ein Risikokapitalgeber aus San Francisco, die Intel Corporation, um Speicherchips herzustellen. Im ersten Geschäftsjahr verkaufte Intel Chips im Wert von nur 3 000 US-Dollar, aber das Geschäft hat sich bekanntlich später gut entwickelt. (Intel ist heute der weltweit größte Hersteller von CPU-Chips.)

Ende der 1960er Jahre waren Rechner schwerfällige elektromechanische Maschinen in der Größe eines modernen Laserdruckers mit einem Gewicht von 20 kg. Im September 1969 erteilte die japanische Firma Busicom Intel einen Auftrag für die Fertigung von 12 kundenspezifischen Chips für einen geplanten elektronischen Rechner. Intel-Ingenieur Ted Hoff, dem dieses Projekt übertragen wurde, sah sich den Plan an und erkannte, dass er eine universelle 4-Bit-CPU auf einen einzigen Chip packen konnte, der das Gleiche bewirkte und obendrein einfacher und billiger ist. Damit war 1970 die erste Ein-Chip-CPU – der 2300 Transistoren umfassende 4004 – geboren [Faggin et al., 1996].

Bemerkenswert ist, dass weder Intel noch Busicom ahnten, was sie damit schufen. Als der Versuch lohnenswert erschien, den 4004 in andere Projekte einzubinden, bot Intel der Firma Busicom an, alle Rechte an dem neuen Chip zurückzukaufen, indem Busicom

die 60.000 US-Dollar zurückerhalten sollte, mit der die Firma die Intel-Entwicklung finanziert hatte. Intels Angebot wurde ohne Zögern angenommen. Unmittelbar danach begann die Arbeit an einer 8-Bit-Version des Chips, dem 8008, der 1972 eingeführt wurde. ▶Tabelle 1.4 zeigt die Intel-Prozessorfamilie beginnend mit dem 4004 und 8008.

Chip	Monat/ Jahr der Einführung	Takt-frequenz in MHz	Anzahl der Tran-sistoren	Adressier-barer Speicher	Bemer-kungen
4004	04/1971	0,108	2 300	640 Byte	Erster Mikroprozessor auf einem Chip
8008	04/1972	0,108	3 500	16 KB	Erster 8-Bit-Mikroprozessor
8080	04/1974	2	6 000	64 KB	Erste universelle CPU auf einem Chip
8086	06/1978	5–10	29 000	1 MB	Erste 16-Bit-CPU auf einem Chip
8088	06/1979	5–8	29 000	1 MB	Im IBM-PC eingesetzte „abge-speckte" Version des 8086
80286	02/1982	8–12	134 000	16 MB	Speicherschutz vorhanden
80386	10/1985	16–33	275 000	4 GB	Erste 32-Bit-CPU
80486	04/1989	25–100	1,2 Mill.	4 GB	Integrierter 8-KB-Cache
Pentium	03/1993	60–233	3,1 Mill.	4 GB	Zwei Pipelines; spätere Modelle mit MMX
Pentium Pro	03/1995	150–200	5,5 Mill.	4 GB	Zwei integrierte Cache-Ebenen
Pentium II	05/1997	233–400	7,5 Mill.	4 GB	Pentium Pro plus MMX-Befehle
Pentium III	02/1999	650–1400	9,5 Mill.	4 GB	SSE-Befehle für 3-D-Grafiken
Pentium 4	11/2000	1300–3800	42 Mill.	4 GB	Hyperthreading; zusätzliche SSE-Befehle
Core Duo	1/2006	1600–3200	152 Mill.	2 GB	Zwei Kerne auf einem einzelnen Die
Core	7/2006	1200–3200	410 Mill.	64 GB	64-Bit-Architektur mit 4 Kernen (Quad-Core)
Core i7	1/2011	1100–3300	1160 Mill.	24 GB	Integrierter Grafikprozessor

Tabelle 1.4: Die wichtigsten Vertreter der Intel-Prozessorfamilie

Intel erwartete keine große Nachfrage nach dem 8008-Chip und fertigte ihn deshalb nur in kleinen Stückzahlen. Zu jedermanns Erstaunen war das Interesse jedoch sehr groß. Folglich machte sich Intel daran, einen neuen CPU-Chip zu entwerfen, der die 8008-Speichergrenze von 16 KB (aufgrund der geringen Anzahl von Pins auf dem Chip) beseitigte. Aus diesem Design ist der 8080 hervorgegangen, eine kleine Universal-CPU, die 1974 eingeführt wurde. Ähnlich wie die PDP-8 eroberte dieses Produkt den Markt im Sturm und erwies sich als Massenartikel. Im Unterschied zu DEC verkaufte Intel aber nicht Tausende, sondern Millionen.

Als echte 16-Bit-CPU auf einem Chip kam 1978 der 8086 auf den Markt. Das 8086-Design war mit dem 8080-Design vergleichbar, aber nicht voll kompatibel. Auf den 8086 folgte der 8088, der die gleiche Architektur aufwies und die gleichen Programme ausführte, anstelle eines 16-Bit-Busses aber einen 8-Bit-Bus hatte und somit langsamer und billiger als der 8086 war. Als IBM den 8088-Chip als CPU für den ursprünglichen IBM-PC wählte, avancierte dieser Chip schnell zum Industriestandard für Personalcomputer.

Die ISA des 8086 charakterisiert bis heute die Prozessoren der Firma Intel. Sie wurde über die Zeit an neue technologische Entwicklungen angepasst. Aktuelle Prozessoren sind abwärtskompatibel mit dem 8086.

Weder der 8088 noch der 8086 konnten mehr als 1 MB Speicher adressieren. Ende der achtziger Jahre wurde dies immer mehr zu einem ernsthaften Problem und Intel entwickelte den 80286, der zu dem 8086 abwärtskompatibel war. Der 80286 hatte im Wesentlichen den gleichen Befehlssatz wie der 8086/8088, aber eine gänzlich andere Speicherorganisation, was angesichts der Kompatibilitätsforderung zu den älteren Chips recht seltsam erscheint. Eingesetzt wurde der 80286 im IBM-PC/AT und in den PS/2-Modellen des mittleren Leistungsbereichs. Wie beim 8088 war dieser Chip ein großer Erfolg, vor allem, weil man ihn als schnelleren 8088 betrachtete.

Der nächste logische Schritt war eine echte 32-Bit-CPU auf einem Chip – der 1985 eingeführte 80386. Wie der 80286 war der 80386 mehr oder weniger kompatibel mit allen zurückliegenden Chips bis zum 8080. Die Abwärtskompatibilität war ein Segen für alle, denen ihre alte Software wichtig war, aber ein Ärgernis für diejenigen, die sich eine einfache, saubere, moderne Architektur wünschten, die nicht mit den Fehlern und Technologien der Vergangenheit behaftet ist.

Jahre später kam der 80486 heraus. Es handelte sich prinzipiell um eine schnellere Version des 80386, die außerdem eine Gleitkommaeinheit und einen Cache-Speicher von 8 KB auf dem Chip integrierte. Ein **Cache-Speicher** (Cache Memory) nimmt die am häufigsten benutzten Speicherwörter in bzw. nahe der CPU auf, um (langsame) Zugriffe auf den Hauptspeicher zu vermeiden. Darüber hinaus unterstützte der 80486 mehrere Prozessoren, was es Herstellern ermöglichte, Systeme mit mehreren CPUs zu bauen, die einen gemeinsamen Speicher nutzten.

Zu dieser Zeit lernte Intel auf unsanfte Art und Weise (Niederlage bei einem Gerichtsverfahren wegen Verletzung des Warenzeichenrechts), dass man Zahlen (wie 80486) nicht als Warenzeichen schützen kann. Folglich erhielt die nächste Generation einen Namen: **Pentium** (nach dem griechischen Wort für fünf, πεντε). Im Gegensatz zum 80486 mit nur einer internen Pipeline wies der Pentium zwei auf, was ihn doppelt so schnell machte. (Pipelines werden ausführlich in *Kapitel 2* behandelt.)

In einem späteren Produktionsstadium fügte Intel spezielle **MMX**-Befehle (**MultiMedia eXtensions**) hinzu. Diese sollten die bei der Audio- und Videoverarbeitung anfallenden Berechnungen beschleunigen, was einen speziellen Multimedia-Coprozessor überflüssig machte.

Die als nächstes erschienene Generation dürfte wohl diejenigen, die auf den Sexium (*Sex* ist Lateinisch und bedeutet sechs) hofften, enttäuscht haben. Die Bezeichnung „Pentium“ war inzwischen so bekannt, dass die Marketingabteilung daran festhalten wollte, und der neue Chip erhielt den Namen Pentium Pro. Trotz der nur geringen Namensänderung stellte dieser Prozessor gegenüber seinem Vorläufer einen einschneidenden Bruch mit der Vergangenheit dar. Statt zwei oder mehr Pipelines hatte der Pentium Pro einen gänzlich anderen internen Aufbau und konnte bis zu fünf Befehle gleichzeitig ausführen.

Als weitere Neuerung wies der Pentium Pro einen aus zwei Ebenen bestehenden Cache auf. Der Prozessorchip selbst hatte einen schnellen 8-KB-Speicher für häufig benutzte Befehle und einen weiteren schnellen 8-KB-Speicher für häufig benutzte Daten. Im selben Prozessorgehäuse (aber getrennt vom eigentlichen Prozessorchip) befand sich ein zweiter Cache mit einer Kapazität von 256 KB.

Der Pentium Pro hatte zwar einen großen Cache, ihm fehlten aber die MMX-Befehle (weil Intel keinen so großen Chip mit akzeptabler Ausbeute herstellen konnte). Als es die verbesserte Technologie endlich erlaubte, sowohl die MMX-Befehle als auch den Cache auf einem Chip unterzubringen, wurde das kombinierte Produkt als Pentium II herausgebracht. Im nächsten Schritt fügte Intel weitere Multimediabefehle hinzu, die sogenannten **SSE-Befehle** (**Streaming SIMD Extensions**), um die 3-D-Grafikfähigkeiten zu erweitern [Raman et al., 2000]. Der neue Chip wurde Pentium III getauft, war aber intern im Wesentlichen ein Pentium II.

Der nächste Pentium (im November 2000 freigegeben) basierte auf einer anderen internen Architektur. Um dieses Ereignis zu würdigen, wechselte Intel von römischen Zahlen zu arabischen und nannte den Prozessor Pentium 4. Wie gewohnt war der Pentium 4 schneller als alle seine Vorgänger. Die 3,06-GHz-Version führte auch ein faszinierendes neues Funktionsmerkmal ein – Hyperthreading. Dadurch können Programme ihre Arbeit auf zwei Threads aufteilen, die der Pentium 4 parallel ausführt, was die Ausführungsgeschwindigkeit deutlich steigert. Außerdem kam ein weiterer Stapel von SSE-Befehlen hinzu, um die Audio- und Videoverarbeitung nochmals zu beschleunigen.

Im Jahr 2006 hat Intel den Markennamen von Pentium in Core geändert und mit dem **Core 2 duo** einen Doppelkernchip (Dual Core) herausgebracht. Als es Intel darum ging, eine preiswerte Einzelkernversion des Chips auf den Markt zu bringen, wurden einfach Core-2-duo-Prozessoren verkauft, bei denen ein Kern deaktiviert war. Denn letztlich war es billiger, ein wenig Silizium auf jedem hergestellten Chip zu verschwenden, als einen neuen Chip von Grund auf zu entwerfen und zu testen, was mit enormen Kosten verbunden ist. Die Core-Reihe wurde weiterentwickelt, wobei die i3, i5 und i7 zu bekannten Varianten für Computer im unteren, mittleren und oberen Leistungsbereich avancierten. Zweifellos werden weitere Varianten folgen. ▶Abbildung 1.8 zeigt ein Foto des i7. Tatsächlich enthält der Chip 8 Kerne, doch sind außer in der Xeon-Version nur 6 aktiviert. Dieses Konzept erlaubt es, einen Chip mit einem oder zwei defekten Kernen trotzdem verkaufen zu können, indem man die defekten Kerne deaktiviert. Jeder Kern besitzt eigene Level-1- und Level-2-Caches, ein Level-3-Cache (L3) wird von allen Kernen gemeinsam genutzt. Darauf gehen wir später in diesem Buch noch näher ein.

Neben der Hauptproduktlinie mit den oben behandelten Desktopprozessoren stellt Intel Varianten einiger Pentium-Chips für spezielle Marktsegmente her. Anfang 1998 führte Intel eine neue Produktlinie mit der Bezeichnung **Celeron** ein. Dabei handelt es sich grundsätzlich um eine preisgünstige, schwächere Ausführung des Pentium II für PCs im unteren Leistungssegment. Da der Celeron die gleiche Architektur wie der Pentium II hat, gehen wir hier nicht weiter darauf ein. Im Juni 1998 führte Intel eine spezielle Version des Pentium II für den oberen Leistungsbereich ein. Dieser Prozessor – **Xeon** – hat einen größeren Cache, einen schnelleren Bus und eine bessere Mehrprozessor-Unterstützung, ist sonst aber ein normaler Pentium II, sodass wir auch diesen Prozessor nicht separat besprechen. Der Pentium III existierte wie auch die Nachfolger ebenfalls in einer Xeon-Version. Die neueren Xeon-Prozessoren zeichnen sich vor allem durch mehr Kerne aus.

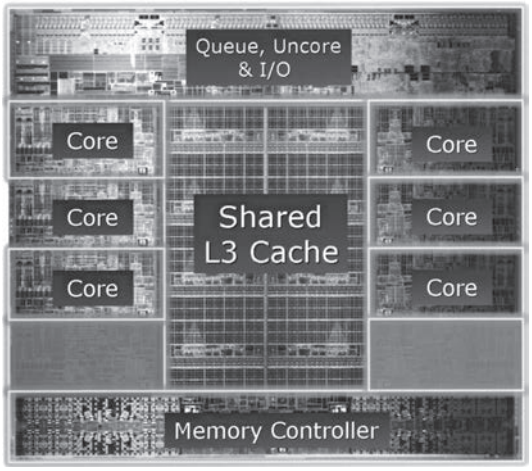

Abbildung 1.8: Der Die des Intel Core i7-3960X hat eine Fläche von 21 × 21 mm^2 und enthält 2,27 Milliarden Transistoren. [Copyright by Intel Corporation, 2011, verwendet mit freundlicher Genehmigung]

Im Jahre 2003 brachte Intel den Pentium M (für Mobile) auf den Markt, der speziell für Notebooks konzipiert ist. Dieser Chip ist Teil der Centrino-Architektur, die auf einen geringeren Leistungsverbrauch (längere Batterielaufzeit) abzielt und für kleinere und leichtere Computer sowie integrierte Unterstützung für Drahtlosnetzwerke nach dem Standard IEEE 802.11 (WiFi) vorgesehen ist. Der Pentium M hatte einen sehr geringen Leistungsverbrauch und war wesentlich kleiner als der Pentium 4, zwei Eigenschaften, die ihm (und seinen Nachfolgern) bald erlaubten, die Pentium-4-Mikroarchitektur in zukünftige Intel-Produkte einzubeziehen.

Alle Intel-Chips sind mit ihren Vorgängern bis zum 8086 abwärtskompatibel. Das heißt, dass ein Pentium 4 alte 8086-Programme ohne Änderungen ausführen kann. Diese Kompatibilität war bei Intel immer eine Designvorgabe, damit die Benutzer ihre vorhandene Software weiter verwenden können. Natürlich ist der Core um vier Größenordnungen komplexer als der 8086, beherrscht also Funktionen, die dem 8086 fremd sind. Diese stückweisen Erweiterungen haben zu einer Architektur geführt, die sicherlich eleganter ausgefallen wäre, wenn man den Pentium 4-Architekten 42 Millionen Transistoren zugestanden und sie angewiesen hätte, ganz von vorn zu beginnen.

Interessant dabei ist, dass man zwar das Moore'sche Gesetz lange Zeit nur auf die Anzahl der Speicherbits angewendet hat, es aber gleichwohl für Prozessorchips gültig ist. Trägt man die in Abbildung 1.6 angegebenen Anzahlen von Transistoren im halblogarithmischen Maßstab auf, ist zu erkennen, dass das Moore'sche Gesetz auch hier zutrifft. ▶Abbildung 1.9 zeigt das entsprechende Diagramm.

Auch wenn das Moore'sche Gesetz vermutlich noch einige Jahre gültig bleibt, wird es zunehmend von einem anderen Problem überschattet: der Wärmeableitung. Kleinere Transistoren ermöglichen höhere Taktfrequenzen, was die Forderung nach höheren Betriebsspannungen nach sich zieht. Der Leistungsverbrauch und die abgegebene Wärme wachsen proportional zum Quadrat der Spannung. Je schneller ein Bauelement, desto mehr Wärme ist abzuführen. Ein 3,6-GHz-Pentium 4 verbraucht 115 W, wird also heißer als eine 100-W-Glühbirne. Mit höherer Taktfrequenz verschlimmert sich die Lage noch.

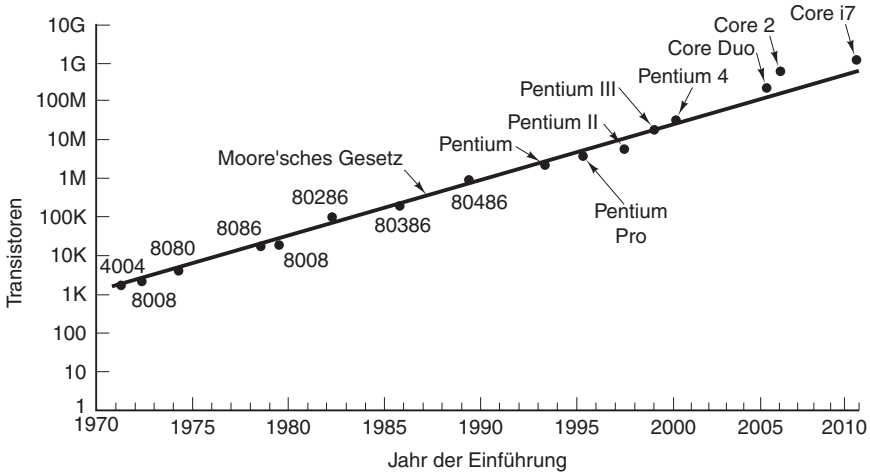

Abbildung 1.9: Das Moore'sche Gesetz für (Intel-)Prozessoren

Im November 2004 hat Intel die Produktion des 4-GHz-Pentium 4 aufgrund von Problemen mit der Wärmeableitung eingestellt. Große Lüfter lindern zwar das Problem, doch der von ihnen erzeugte Geräuschpegel ist bei den Benutzern nicht gerade beliebt. Die in großen Mainframes übliche Wasserkühlung kommt für Desktopcomputer (geschweige denn für Notebooks) nicht infrage. Als Folge scheint der bislang so unerbittliche Vormarsch auf den Taktfrequenzgipfel vorübergehend ins Stocken zu geraten, zumindest bis Intels Techniker eine Lösung gefunden haben, um die produzierte Wärme effizient wieder loszuwerden. Stattdessen sehen Intels Zukunftspläne vor, zwei oder mehr CPUs auf einem einzelnen Chip zusammen mit einem großen, gemeinsam genutzten Cache unterzubringen. Aufgrund der Beziehung zwischen Leistungsaufnahme, Spannung und Taktfrequenz verbrauchen zwei CPUs auf einem Chip weit weniger Leistung als eine CPU bei der doppelten Geschwindigkeit. Als Folge könnte man den aus dem Moore'schen Gesetz abgeleiteten Gewinn zukünftig in größere On-Chip-Caches ummünzen, anstatt nach immer höheren Taktfrequenzen zu streben. Für die Programmierer bedeutet es eine große Herausforderung, diese Multiprozessoren richtig zu nutzen. Denn während die ausgefeilten Einprozessor-Mikroarchitekturen der Vergangenheit aus vorhandenen Programmen mehr Leistung herausholen konnten, verlangen Multiprozessoren vom Programmierer, die parallele Ausführung explizit mithilfe von Threads, Semaphoren, gemeinsamem Speicher und anderen Techniken – die Kopfschmerzen bereiten und fehlerträchtig sind – zu organisieren.

1.4.2 Einführung in ARM-Architektur

In den frühen 1980er Jahren begann die britische Firma Acorn Computer noch im Erfolgsrausch ihres 8-Bit-Personalcomputers BBC Micro an einem zweiten Computer zu arbeiten in der Hoffnung, den erst kürzlich veröffentlichten IBM-PC zu schlagen. Steve Furber und seine Kollegen bei Acorn waren der Ansicht, dass der 8-Bit-Prozessor 6502 des BBC Micro keine Chancen hatte, gegen den 16-Bit-Prozessor 8086 des IBM-PC anzutreten. Eine Marktanalyse bestätigte die geringen Erfolgsaussichten.

Inspiriert vom Berkeley-RISC-Projekt, in dem ein kleines Team einen bemerkenswert schnellen Prozessor entwickelt hatte (der letztlich zur SPARC-Architektur führte), entschieden sich Furber und seine Mitarbeiter dafür, ihre eigene CPU für das Projekt zu bauen. Das

Design nannten sie die Acorn RISC Machine (oder ARM, was später wieder umgetauft wurde zu Advanced RISC Machine, als sich ARM schließlich von Acorn trennte). Im Jahre 1985 war der Entwurf fertiggestellt. Befehle und Daten hatten eine Breite von 32 Bit, für den Adressraum waren 26 Bit vorhanden. Gefertigt wurde der Prozessor von VLSI Technology.

Die erste ARM-Architektur (ARM2 genannt) erschien im Personalcomputer Acorn Archimedes. Der Archimedes war für seine Zeit ein sehr schneller und preisgünstiger Computer, der bis zu 2 MIPS (Million Instructions Per Second) ausführte und zum Verkaufsstart nur 899 Britische Pfund kostete. In Großbritannien, Irland, Australien und Neuseeland war der Computer sehr beliebt, insbesondere in Schulen.

Prozessoren basierend auf der RISC-Architektur der Firma ARM in Cambridge, UK, dominieren den Markt der mobilen Geräte. Ihre Designs werden von vielen Herstellern in System-on-a-Chip-Lösungen integriert.

Auf dem Erfolg des Archimedes aufbauend trat Apple an Acorn heran, um einen ARM-Prozessor für das geplante Apple-Newton-Projekt – den ersten Palmtop – zu entwickeln. Um sich besser auf das Projekt konzentrieren zu können, verließ das ARM-Architekturteam Acorn und gründete die Firma Advanced RISC Machines (ARM). Ihr neuer Prozessor, der ARM 610, lief im Apple Newton, der 1993 auf den Markt kam. Im Gegensatz zum ursprünglichen ARM-Design enthielt dieser neue ARM-Prozessor einen 4-KB-Cache, der die Leistung des Designs deutlich verbesserte. Auch wenn dem Apple Newton kein großer Erfolg beschieden war, konnte der ARM 610 in anderen Anwendungen punkten, unter anderem im RISC PC von Acorn.

Mitte der 1990er Jahre arbeitete ARM mit Digital Equipment Corporation zusammen, um eine Hochgeschwindigkeitsversion des ARM mit geringem Leistungsbedarf für batterieschonende mobile Anwendungen wie zum Beispiel PDAs zu entwickeln. Herausgekommen ist das StrongARM-Design, das vom ersten Augenblick an wegen seiner hohen Geschwindigkeit (233 MHz) und äußerst geringen Leistungsaufnahme (1 W) für Aufsehen in der Industrie sorgte. Seine Effizienz verdankt er einem einfachen klaren Design, das zwei 16-KB-Caches für Befehle und Daten vorsieht. Auf dem Markt waren der StrongARM und seine Nachfolger bei DEC mittelmäßig erfolgreich, sich in einer Reihe von PDAs, Set-Top-Boxen, Mediengeräten und Routern zu etablieren.

Der vielleicht ehrwürdigste Vertreter der ARM-Architekturen ist das ARM7-Design, das ARM erstmals 1994 veröffentlicht hat und heute immer noch gebräuchlich ist. Das Design zeichnet sich durch getrennte Befehls- und Datencaches aus. Außerdem arbeitet es mit dem 16-Bit-Thumb-Befehlssatz, der eine gekürzte Version des vollen 32-Bit-ARM-Befehlssatzes verkörpert und es Programmierern erlaubt, viele der gebräuchlichsten Operationen in den kleineren 16-Bit-Anweisungen zu codieren, wodurch sich die Größe des benötigten Programmspeichers verringert. Der Prozessor hat sich in einer breiten Palette von eingebetteten Anwendungen im unteren und mittleren Leistungsbereich bewährt, beispielsweise in Toastern, als Motorsteuerung und sogar in der Handheld-Spielkonsole Nintendo Gameboy Advance.

Im Unterschied zu vielen Computerfirmen fertigt ARM keine Mikroprozessoren, sondern entwickelt Designs mit den dazugehörenden ARM-basierten Entwicklungstools und -bibliotheken und lizenziert sie für Entwickler und Chiphersteller. Zum Beispiel ist die im Android-basierten Tablet Samsung Galaxy Tab eingesetzte CPU ein ARM-basierter Prozessor. Das Galaxy Tab enthält das Ein-Chip-System (System-on-a-Chip) Tegra 2, das zwei ARM-Cortex-A9-Prozessoren und einen GeForce-Grafikprozessor von Nvidia vereint. Die Tegra-2-Kerne wurden von ARM entworfen, in ein von Nvidia stammendes Ein-Chip-System integriert und von TSMC (Taiwan Semiconductor Manufacturing Company) gefertigt. In dieser eindrucksvollen Zusammenarbeit von Firmen aus verschiedenen Ländern tragen alle Beteiligten zum Erfolg des Endprodukts bei.

▶Abbildung 1.10 zeigt das Ein-Chip-System Tegra 2 von Nvidia. Es enthält drei ARM-Prozessoren: zwei 1,2-GHz-ARM-Cortex-A9-Kerne und einen ARM7-Kern. Die Cortex-A9-Kerne sind Dual-Issue-Out-of-Order-Kerne mit einem L2-Cache von 1 MB und Unterstützung für Shared-Memory-Multiprocessing. (Die vielen Fachbegriffe werden in späteren Kapiteln näher erläutert. Momentan genügt es zu wissen, dass diese Merkmale für das sehr schnelle Design zuständig sind!) Der ARM7-Kern ist ein älterer und kleinerer ARM-Kern, der für Systemkonfiguration und Energieverwaltung zuständig ist. Beim Grafikkern handelt es sich um einen 333-MHz-GeForce-Grafikprozessor (Graphics Processing Unit, GPU), der für geringen Energieverbrauch optimiert wurde. Darüber hinaus sind auf dem Tegra 2 ein Video-En-/Decoder, ein Audio-Prozessor und eine HDMI-Schnittstelle für die Videoausgabe integriert.

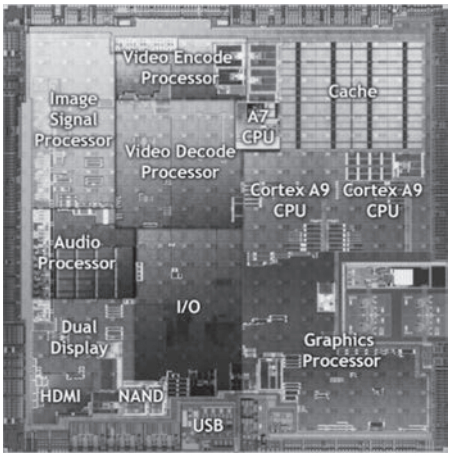

Abbildung 1.10: Das Tegra-2-System auf einem Chip von Nvidia [Copyright 2011 by Nvidia Corporation. Verwendet mit freundlicher Genehmigung]

Die ARM-Architektur ist im Markt für energiesparende, mobile und eingebettete Prozessoren sehr erfolgreich. Im Januar 2011 ließ ARM verlauten, dass die Firma seit ihrer Gründung 15 Milliarden ARM-Prozessoren verkauft hat und die Verkaufszahlen stetig steigen. Auch wenn die ARM-Architektur auf die unteren Marktsegmente zugeschnitten ist, besitzt sie doch die technischen Funktionen, um sich in jedem Markt zu behaupten, und es gibt Hinweise, dass sie ihre Horizonte erweitern könnte. Zum Beispiel wurde im Oktober 2011 ein 64-Bit-ARM angekündigt. Außerdem hatte Nvidia im Januar 2011 das „Project Denver“ angekündigt, ein ARM-basiertes Ein-Chip-System, das für Server und andere Märkte entwickelt werde. Das Design vereint mehrere 64-Bit-ARM-Prozessoren und einen Allzweckgrafikprozessor (General Purpose GPU, GPGPU). Die energiesparenden Aspekte des Designs helfen dabei, die Anforderungen an die Kühlung bei Serverfarmen und Datenzentren zu verringern.

1.4.3 Einführung in die AVR-Architektur

Unser drittes Beispiel unterscheidet sich deutlich vom ersten (der x86-Architektur bei Personalcomputern und Servern) und zweiten (der ARM-Architektur bei PDAs und Smartphones). Es handelt sich um die AVR-Architektur, die im unteren Marktsegment für eingebettete Systeme verwendet wird. Die AVR-Geschichte beginnt 1996 an der Norwegischen Technischen Hochschule, wo die Studenten Alf-Egil Bogen und Vegard Wol-

lan eine 8-Bit-RISC-CPU namens AVR entwarfen. Berichten zufolge soll der Name von „(A)lf and (V)egard's (R)ISC processor" herrühren. Kurz nach Abschluss des Entwurfs kaufte Atmel das Design und gründete Atmel Norway, wo die beiden Architekten weiter an der Verfeinerung des AVR-Prozessordesigns arbeiteten. Atmel veröffentlichte 1997 ihren ersten AVR-Mikrocontroller, den AT90S1200. Um seine Akzeptanz bei Systemdesignern zu fördern, implementierten sie die gleiche Anschlussbelegung wie beim Intel 8051, der seinerzeit einer der bekanntesten Mikrocontroller war. Heute besteht großes Interesse an der AVR-Architektur, da sie den Kern der beliebten eingebetteten Open-Source-Controllerplattform Arduino bildet.

AVR-Architektur: Mikrocontroller-Architektur aus Norwegen, die der Open-Source-Plattform Arduino zugrunde liegt.

Die AVR-Architektur ist in drei Klassen von Mikrocontrollern realisiert, die in ▶Tabelle 1.5 aufgeführt sind. Die unterste Klasse mit dem tinyAVR ist für die meisten flächen-, energie- und kostenbegrenzten Anwendungen vorgesehen. Der Controller enthält eine 8-Bit-CPU, grundlegende digitale E/A-Unterstützung und einen analogen Eingang (um zum Beispiel Temperaturwerte von einem Thermistor einzulesen). Aufgrund der kleinen Bauweise sind die Anschlüsse des tinyAVR doppelt belegt. Sie lassen sich zur Laufzeit für die digitalen oder analogen Funktionen des Mikrocontrollers programmieren. Der megaAVR, der auch im bekannten eingebetteten Open-Source-System Arduino enthalten ist, bringt außerdem serielle E/A-Unterstützung, interne Taktgeber und programmierbare Analogausgänge mit. Der AVR-XMEGA-Mikrocontroller am oberen Ende der Familie verfügt über eine Beschleunigung für kryptografische Operationen und Unterstützung für USB-Schnittstellen.

Chip	Flash	EEPROM	RAM	Pins	Features
tinyAVR	0,5 – 16 KB	0 – 512 B	32 – 512 B	6 – 32	Winzig, digitale Ein-/Ausgänge, analoger Eingang
megaAVR	8 – 256 KB	0,5 – 4 KB	0,25 – 8 KB	28 – 100	Viele Peripherieanschlüsse, Analogausgang
AVR XMEGA	16 – 256 KB	1 – 4 KB	2 – 16 KB	44 – 100	Kryptobeschleunigung, USB-E/A

Tabelle 1.5: Mikrocontroller-Klassen in der AVR-Familie

Zusammen mit verschiedenen zusätzlichen Peripherieeinrichtungen umfasst jede AVR-Prozessorklasse einige zusätzliche Speicherressourcen. Mikrocontroller haben normalerweise drei Arten von Speicher an Bord: Flash, EEPROM und RAM. Flashspeicher ist über eine externe Schnittstelle mit hohen Spannungen programmierbar. Hier werden Programmcode und Daten gespeichert. Flash-RAM ist nicht flüchtig, behält also seinen Inhalt, auch wenn die Betriebsspannung vom System abgeschaltet wird. Wie Flashspeicher ist auch ein EEPROM nicht flüchtig, lässt sich aber im Unterschied zum Flash-RAM durch das Programm zur Laufzeit ändern. In diesem Speichertyp legen eingebettete Systeme die Konfigurationsdaten des Benutzers ab, beispielsweise ob die Alarmzeit im 12- oder 24-Stundenformat angezeigt wird. Schließlich dient das RAM dazu, die Programmvariablen des laufenden Programms zu speichern. Dieser Speichertyp ist flüchtig, sodass alle hier gespeicherten Werte verloren gehen, sobald das System ohne Betriebsspannung ist. Mit den flüchtigen und nicht flüchtigen RAM-Typen beschäftigt sich Kapitel 2 ausführlich.

Das Rezept für den Erfolg im Mikrocontrollergeschäft besteht darin, in den Chip alles das zu packen, was möglicherweise benötigt wird (und auch das Spülbecken, wenn es sich auf einen Quadratmillimeter verkleinern lässt), und es dann in einem preiswerten

und kleinen Gehäuse mit wenigen Anschlüssen unterzubringen. Indem man viele Funktionen in den Mikrocontroller integriert, lässt er sich für viele Anwendungen einsetzen, und wenn er preiswert und klein ist, eignet er sich für viele Formfaktoren. Um eine Vorstellung davon zu bekommen, was alles in einem modernen Mikrocontroller steckt, sehen wir uns die Peripherieeinrichtungen des megaAVR-168 von Atmel an:

1. Drei Timer (zwei 8-Bit-Timer und ein 16-Bit-Timer)
2. Echtzeituhr mit Oszillator
3. Sechs Kanäle mit Pulsbreitenmodulation, die zum Beispiel die Helligkeit oder die Motordrehzahl steuern können
4. Acht Kanäle mit Analog-Digital-Konvertern zum Einlesen von Spannungspegeln
5. Universeller serieller Empfänger/Sender
6. Serielle I2C-Schnittstelle, ein gebräuchlicher Standard für den Anschluss von Sensoren
7. Programmierbarer Watchdog-Timer, der ein blockiertes System erkennt
8. Analogkomparator, der zwei Eingangsspannungen vergleicht
9. Stromausfalldetektor, der das System unterbricht, wenn die Betriebsspannung ausfällt
10. Interner programmierbarer Taktgeber, der den CPU-Takt liefert

1.5 Metrische Einheiten

Um Unklarheiten auszuschließen, sei ausdrücklich darauf hingewiesen, dass in diesem Buch – wie in Wissenschaft und Technik allgemein üblich – das metrische Einheitensystem und nicht das Zoll-System verwendet wird. ▶Tabelle 1.6 gibt die gesetzlichen Einheitenvorsätze des Internationalen Einheitensystems (SI) an.

Faktor	Vorsatz	Zeichen	Faktor	Vorsatz	Zeichen
10^1	Deka	da	10^{-1}	Dezi	d
10^2	Hekto	h	10^{-2}	Zenti	c
10^3	Kilo	k	10^{-3}	Milli	m
10^6	Mega	M	10^{-6}	Mikro	µ
10^9	Giga	G	10^{-9}	Nano	n
10^{12}	Tera	T	10^{-12}	Piko	p
10^{15}	Peta	P	10^{-15}	Femto	f
10^{18}	Exa	E	10^{-18}	Atto	a
10^{21}	Zetta	Z	10^{-21}	Zepto	z
10^{24}	Yotta	Y	10^{-24}	Yokto	y

Tabelle 1.6: Einheitenvorsätze des Internationalen Einheitensystems (SI)

Größenangaben in der Computertechnik – beispielsweise für Speicher, Festplatten, Dateien oder Datenbanken – beruhen oftmals auf 2er-Potenzen. Um keine neuen Einheitenvorsätze erfinden zu müssen, hat es sich eingebürgert, die gesetzlichen Vorsätze mit leicht abweichender Bedeutung und Schreibweise zu übernehmen. Der Vorsatz Kilo bedeutet dann 2^{10} (= 1024) statt 10^3 (= 1000). Außerdem schreibt man das Kurzzeichen auch bei Kilo mit einem großen K. Somit enthält ein 1-KB-Speicher genau 1024 Bytes und nicht 1000 Bytes, ein 1-MB-Speicher 2^{20} = 1024 KB = 1.048.576 Bytes, eine 1-GB-Festplatte 2^{30} = 1024 MB = 1024×1024 KB = 1.073.741.824 Bytes und eine 1-TB-Datenbank 2^{40} = 1024 GB = 1024×1024 MB = $1024 \times 1024 \times 1024$ KB = 1.099.511.627.776 Bytes.

Da eine solche Verwendung der Einheitenvorsätze nicht verbindlich geregelt ist, kann es durchaus zu Fehlinterpretationen und Verwechslungen kommen. So stehen die Einheitenvorsätze bei Geschwindigkeitsangaben meistens für die „wirklichen“ Zehnerpotenzen, d.h., eine Netzwerkverbindung für eine Geschwindigkeit von 10 Mbit/s überträgt 10.000.000 Bit/s, weil es sich hier nicht um Zweierpotenzen handelt.

Um Mehrdeutigkeiten in diesem Buch zu vermeiden, werden durchgehend die Symbole KB, MB, GB und TB für 2^{10}, 2^{20}, 20^{30} und 2^{40} sowie die Symbole Kbit/s, Mbit/s, Gbit/s und Tbit/s für 10^3, 10^6, 10^9 und 10^{12} Bit/s verwendet.

1.6 Gliederung dieses Buchs

Dieses Buch behandelt mehrschichtige Computer (was nahezu alle modernen Computer betrifft) und ihre Organisation. Vier Ebenen beschreibt es ausführlich – die Ebene der digitalen Logik, die Mikroarchitekturebene, die ISA-Ebene und die Ebene der Betriebssystemmaschine. Dabei geht es um grundlegende Fragen, die das Gesamtdesign der jeweiligen Ebene (und warum sie gerade so und nicht anders entworfen wurde), die Art der verfügbaren Befehle und Daten, die Speicherorganisation und -adressierung sowie die Methode, nach der die jeweilige Ebene implementiert wird, betreffen. Das Gebiet, das sich mit diesen und ähnlichen Themen beschäftigt, heißt Computerorganisation oder Computerarchitektur.

Vorrangig geht es in diesem Buch um Konzepte und weniger um Einzelheiten oder formale Mathematik. So sind einige der angegebenen Beispiele stark vereinfacht, um die Kernideen und nicht die Details hervorzuheben.

Damit Sie einen Einblick gewinnen, wie die in diesem Buch dargestellten Prinzipien in der Praxis umgesetzt sind, verwenden wir die x86-, ARM- und AVR-Architekturen als durchgängige Beispiele in allen Kapiteln. Die Wahl ist aus mehreren Gründen auf diese drei Vertreter gefallen. Erstens sind sie weit verbreitet und dem Leser ist wahrscheinlich mindestens einer davon zugänglich. Zweitens besitzt jeder seine eigene einzigartige Architektur, die eine Grundlage für Vergleiche bietet und zur Frage nach Alternativen anregt. Bücher, die sich nur mit einer Architektur befassen, hinterlassen beim Leser oft ein Gefühl, dass ein „bestes Maschinendesign enthüllt“ wird, was angesichts der vielen Kompromisse und willkürlichen Entscheidungen, die Designer gezwungenermaßen treffen müssen, natürlich unsinnig ist. Der Leser sollte sich diese und alle anderen Computer mit einem kritischen Auge ansehen und versuchen zu verstehen, warum die Dinge so sind, wie sie sind, und was man hätte anders machen können, anstatt sie einfach als gegeben hinzunehmen.

Von Anfang an sollte klar sein, dass es in diesem Buch nicht darum geht, wie man die x86-, ARM- oder AVR-Architekturen programmiert. Diese Prozessoren dienen lediglich dazu, einzelne Dinge zu veranschaulichen, wo es passend erscheint, ohne dabei vollständig sein zu wollen. Wer an einer gründlichen Einführung zu einem dieser Prozessoren interessiert ist, sollte sich die Veröffentlichungen des Herstellers ansehen.

Kapitel 2 ist eine Einführung in die grundlegenden Bestandteile eines Computers – Prozessoren, Speicher und Ein-/Ausgabegeräte. Es behandelt die Systemarchitektur im Überblick und bereitet den Weg für die folgenden Kapitel.

Die **Kapitel 3**, **4**, **5** und **6** sind jeweils einer der in Abbildung 1.2 dargestellten Ebenen gewidmet. Wir gehen von unten nach oben vor, weil dies die traditionelle Entwicklungsrichtung von Computern widerspiegelt. Der Entwurf von Ebene k ergibt sich weitgehend aus den Eigenschaften der Ebene $k - 1$. Deshalb kann man eine bestimmte Ebene nur verstehen, wenn man sich bereits mit den darunterliegenden Ebenen befasst hat. Zudem ist es vom pädagogischen Standpunkt her vernünftig, sich von den einfacheren unteren zu den komplizierteren oberen Ebenen vorzuarbeiten und nicht umgekehrt.

Kapitel 3 behandelt die Ebene der digitalen Logik – die echte Hardware eines Computers. Es zeigt, was Gatter sind und wie man sie zu kombinatorischen Schaltungen verbindet. Dabei lernen Sie die boolesche Algebra als Werkzeug für die Analyse digitaler Schaltungen kennen. Außerdem werden Computerbusse, insbesondere der verbreitete PCI-Bus, erläutert. Dieses Kapitel stellt zahlreiche Beispiele aus der Industrie vor, einschließlich der oben erwähnten drei Beispielarchitekturen.

Kapitel 4 führt die Mikroarchitekturebene und ihre Steuerung ein. Da diese Ebene die Aufgabe hat, die Befehle der Ebene 2 in der darüberliegenden Schicht zu interpretieren, steht dieses Thema im Mittelpunkt und wird anhand von Beispielen veranschaulicht. Außerdem beschäftigt sich dieses Kapitel mit der Mikroarchitekturebene einiger tatsächlich gebauter Computer.

Kapitel 5 befasst sich mit der ISA-Ebene, die von den meisten Computeranbietern als Maschinensprache bezeichnet wird. Hier sehen wir uns die Beispielcomputer im Detail an.

Kapitel 6 geht auf einige Befehle, die Speicherorganisation und die Steuermechanismen auf der Ebene der Betriebssystemmaschine ein. Als Beispiele wurden hierfür die Betriebssysteme Windows (verbreitet auf x86-basierten Desktopsystemen) und UNIX (auf vielen x86- und ARM-basierten Systemen anzutreffen) gewählt.

Kapitel 7 ist der Ebene der Assemblersprache gewidmet. Dabei geht es sowohl um die Assemblersprache selbst als auch um die Assemblierung. Zudem kommt hier das Thema Linken – das bedeutet, die Verbindung mehrerer getrennt übersetzter Programmteile zu einem ausführbaren Programm – zur Sprache.

Kapitel 8 behandelt Parallelrechner, die in der heutigen Zeit zunehmend an Bedeutung gewinnen. Einige dieser Parallelrechner verfügen über mehrere CPUs, die einen Speicher gemeinsam nutzen. Andere arbeiten mit mehreren CPUs ohne gemeinsamen Speicher. Einige sind Supercomputer, andere stellen Ein-Chip-Systeme dar und wieder andere bilden Cluster.

Kapitel 9 enthält eine kommentierte Liste von Literaturvorschlägen, die thematisch angeordnet sind, und eine alphabetische Liste der im gesamten Buch angeführten Literaturhinweise. Dieses wichtige Kapitel sollten Sie ausgiebig nutzen.

Aufgaben

EXTRAS ONLINE
Lösungen

Die Lösungen zu den Aufgaben finden Sie auf der Website unter *www.pearson-studium.de*.

1 Erklären Sie die folgenden Begriffe mit eigenen Worten:

a. Übersetzer

b. Interpreter

c. Virtuelle Maschine

2 Ist es vorstellbar, dass ein Compiler Ausgaben für die Mikroarchitekturebene statt für die ISA-Ebene erzeugt? Erläutern Sie die Vor- und Nachteile dieses Vorschlags.

3 Können Sie sich einen mehrschichtigen Computer vorstellen, bei dem die Geräteebene und die Ebene der digitalen Logik nicht die untersten Ebenen sind? Erläutern Sie Ihre Antwort.

4 Stellen Sie sich einen mehrschichtigen Computer vor, bei dem alle Ebenen unterschiedlich sind. Jede Ebene verfügt über Befehle, die m-mal so leistungsfähig wie die der darunterliegenden Ebene sind; d.h., ein Befehl der Ebene r kann die Arbeit von m Befehlen der Ebene $r - 1$ bewältigen. Wie lange laufen gleichwertige Programme auf den Ebenen 2, 3 und 4 unter der Annahme, dass n Befehle der Ebene r erforderlich sind, um einen einzelnen Befehl der Ebene $r + 1$ zu interpretieren, und dass ein Programm der Ebene 1 für die Ausführung k Sekunden benötigt?

5 Einige Befehle auf der Ebene der Betriebssystemmaschine sind identisch mit Befehlen in der ISA-Sprache. Diese Befehle führt das Mikroprogramm direkt und nicht das Betriebssystem aus. Warum ist dies der Fall? Berücksichtigen Sie dabei die Antwort auf die vorhergehende Frage.

6 Stellen Sie sich einen Computer mit identischen Interpretern auf den Ebenen 1, 2 und 3 vor. Ein Interpreter braucht n Befehle, um einen Befehl zu holen, zu analysieren und auszuführen. Die Ausführungszeit eines Befehls der Ebene 1 beträgt k Nanosekunden. Wie lange benötigt ein Befehl auf den Ebenen 2, 3 und 4?

7 In welcher Hinsicht sind Hardware und Software a) gleichwertig, b) nicht gleichwertig?

8 Die Differenzmaschine von Babbage besaß ein feststehendes Programm, das nicht verändert werden konnte. Ist das im Wesentlichen das Gleiche wie eine heutige CD-ROM, die nicht veränderlich ist? Erläutern Sie Ihre Antwort.

9 Aus dem Von-Neumann-Konzept, Programme im Speicher abzulegen, folgt unter anderem, dass sich Programme genauso wie Daten ändern lassen. Können Sie sich ein Beispiel vorstellen, bei dem diese Möglichkeit nützlich wäre? (*Hinweis:* Denken Sie an Arithmetik auf Datenfeldern.)

10 Im IBM System/360 weist das Modell 75 eine 50-mal höhere Leistung als das Modell 30 auf, obwohl die Zykluszeit nur ein Fünftel beträgt. Wie erklären Sie sich diese Diskrepanz?

11 Die Abbildungen 1.4 und 1.5 zeigen zwei grundlegende Systementwürfe. Beschreiben Sie, wie die Ein-/Ausgabe in jedem System ablaufen könnte. Welches System hat das Potenzial für eine bessere Gesamtleistung?

12 Angenommen, jeder der 300 Millionen Einwohner in den Vereinigten Staaten konsumiert jeden Tag zwei Lebensmittelpakete, die mit RFID-Etiketten versehen sind. Wie viele RFID-Etiketten müssen dafür jährlich produziert werden? Wie hoch sind die Gesamtkosten für die Etiketten, wenn ein einzelnes Etikett ein Cent kostet? Würde diese Summe angesichts der Größe des Bruttoinlandsprodukts ein Hindernis darstellen, um die Etiketten auf jedem verkauften Paket zu verwenden?

13 Nennen Sie drei Geräte, die für die Steuerung durch eine eingebettete CPU infrage kommen.

14 Zu einem bestimmten Zeitpunkt betrage der Durchmesser eines Transistors auf einem Mikroprozessor 0,1 µm. Wie groß wäre ein Transistor im Modell des Folgejahres gemäß dem Moore'schen Gesetz?

15 Es wurde gezeigt, dass das Moore'sche Gesetz nicht nur auf die Packungsdichte von Halbleiterbauelementen anwendbar ist, sondern auch den Zuwachs an (sinnvollen) Simulationsgrößen und die Verringerung der Anzahl von Simulationsläufen vorhersagt. Zeigen Sie zuerst für eine Strömungssimulation, die auf einem heutigen Computer 4 Stunden läuft, dass sie auf Computern, die in 3 Jahren gebaut werden, nur noch 1 Stunde braucht, und auf Computern, die in 6 Jahren vorhanden sein werden, nur noch 15 Minuten dauert. Zeigen Sie dann, dass eine große Simulation mit einer geschätzten Ausführungszeit von 5 Jahren eher abgeschlossen sein würde, wenn wir 3 Jahre warten, bevor wir mit der Simulation beginnen.

16 Im Jahr 1959 konnte ein IBM-7090-Computer rund 500.000 Befehle je Sekunde ausführen, besaß einen Speicher von 32.768 36-Bit-Wörtern und kostete 3 Millionen US-Dollar. Vergleichen Sie dies mit einem aktuellen Computer und bestimmen Sie, wie viel besser der aktuelle Computer abschneidet. Bilden Sie dazu das Verhältnis von Speichergrößen und Geschwindigkeiten und dividieren Sie das Ergebnis durch das Verhältnis der Preise. Stellen Sie sich nun vor, was die gleichen Zuwächse im selben Zeitraum für die Luftfahrtindustrie bedeutet hätten. Die Boeing 707 wurde im Jahre 1959 in nennenswerten Stückzahlen an die Airlines ausgeliefert. Ihre Geschwindigkeit betrug 950 km/h und ihre anfängliche Kapazität 180 Passagiere. Sie kostete 4 Millionen US-Dollar. Welche Werte hätten Geschwindigkeit, Kapazität und Kosten eines Flugzeugs heute, wenn die Entwicklung im gleichen Tempo wie beim Computer erfolgt wäre? Formulieren Sie Ihre Annahmen hinsichtlich Geschwindigkeit, Speichergröße und Preis.

17 Entwicklungen in der Computerindustrie verlaufen oftmals zyklisch. Befehlssätze wurden ursprünglich fest verdrahtet, dann als Mikroprogramme realisiert, später durch RISC-Computer abgelöst und nun erneut fest verdrahtet. Die Rechentechnik war ursprünglich auf großen Mainframes zentralisiert. Nennen Sie zwei Entwicklungen, die ebenfalls das zyklische Verhalten demonstrieren.

18 Die gerichtliche Auseinandersetzung um die Frage, wer den Computer erfunden hat, wurde im April 1973 durch Richter Earl Larson beigelegt. Dieser verhandelte einen Patentverletzungsprozess, der von der Firma Sperry Rand – die die ENIAC-Patente erworben hatte – angestrengt wurde. Nach Ansicht von Sperry Rand müsse jeder, der einen Computer herstellt, Lizenzgebühren an Sperry Rand zahlen, weil diese Firma die Schlüsselpatente besitze. Der Fall ging im Juni 1971 vor Gericht und es wurden mehr als 30.000 Beweisstücke eingebracht. Das Gerichtsprotokoll erstreckte sich über 20.000 Seiten. Machen Sie sich mit diesem Fall anhand der umfassenden Informationen im Internet bekannt und erörtern Sie die technischen Aspekte dieses Falls. Was genau haben Eckert und Mauchley patentiert und warum glaubte das Gericht, dass deren System auf der früheren Arbeit von Atanasoff basierte?

19 Wählen Sie die drei Personen aus, die Ihrer Meinung nach den stärksten Einfluss auf die Schaffung moderner Computerhardware hatten. Beschreiben Sie die Beiträge dieser Personen und begründen Sie, warum Sie sie ausgewählt haben.

20 Wiederholen Sie die vorherige Aufgabe für Computersoftware.

21 Wiederholen Sie die vorherige Aufgabe für Websites.

Aufbau von Computersystemen

2

ÜBERBLICK

2.1 Prozessoren 75

2.2 Hauptspeicher (Primärspeicher) 92

2.3 Sekundärspeicher 104

2.4 Eingabe/Ausgabe 126

Einleitung

» Ein Digitalrechner besteht aus Prozessoren, Speichern und Ein-/Ausgabegeräten, die untereinander verbunden sind. Dieses Kapitel gibt einen einführenden Überblick über diese drei Komponenten mit ihren Verbindungen und liefert den Hintergrund für die eingehende Untersuchung bestimmter Ebenen in den fünf folgenden Kapiteln. Prozessoren, Speicher und Ein-/Ausgabegeräte sind grundlegende Konzepte, denen man auf jeder Ebene begegnet. Deshalb schauen wir uns zunächst diese drei Konzepte an. «

2.1 Prozessoren

▶Abbildung 2.1 zeigt den Aufbau eines einfachen busorientierten Computers. Die **CPU** (Central Processing Unit – zentrale Verarbeitungseinheit) oder der **Prozessor**[1] ist das „Gehirn" des Computers. Sie führt die im Hauptspeicher abgelegten Programme aus, indem sie deren Befehle nacheinander abruft, analysiert und dann ausführt. Die Komponenten sind über einen **Bus** miteinander verbunden, d.h. durch ein Leitungssystem, dessen (parallel geführte) Leitungen Adressen, Daten und Steuersignale übertragen. Externe Busse verbinden die CPU mit dem Speicher und den E/A-Geräten, aber auch innerhalb der CPU werden Busse verwendet – mehr dazu in Kürze. Moderne Computer verfügen über mehrere Busse.

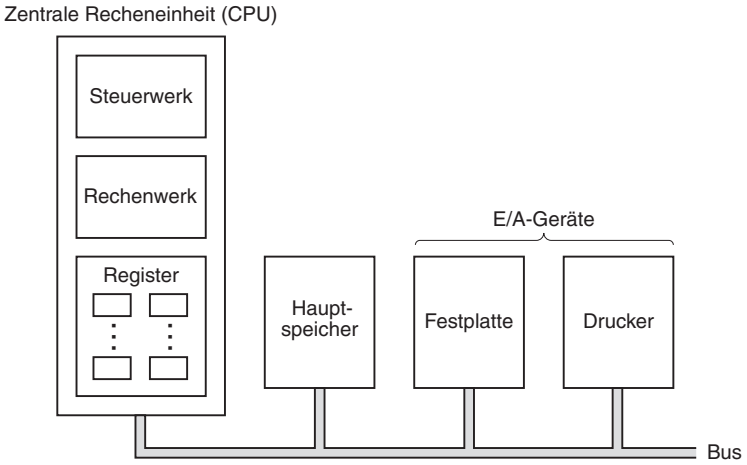

Abbildung 2.1: Der Aufbau eines einfachen Computers mit einer CPU und zwei E/A-Geräten

Die CPU gliedert sich in mehrere Funktionseinheiten. Das Steuerwerk (die Steuereinheit – Control Unit) ruft Befehle aus dem Hauptspeicher ab und bestimmt den Befehlstyp. Das Rechenwerk (Arithmetic Logic Unit, ALU) führt Operationen aus, die für den jeweiligen Befehl erforderlich sind, beispielsweise Addition oder boolesche AND-Verknüpfung.

Außerdem enthält die CPU einen kleinen Hochgeschwindigkeitsspeicher, der Zwischenergebnisse und bestimmte Steuerinformationen temporär aufnimmt. Dieser Speicher besteht aus mehreren Registern, die jeweils eine bestimmte Größe und Funktion haben. Gewöhnlich sind alle Register gleich groß. Jedes Register kann einen Wert aufnehmen, dessen Größe von der Registerbreite – d.h. der Anzahl der Bits in einem Register – abhängt. Da sich die Register innerhalb der CPU befinden, können sie sehr schnell gelesen und geschrieben werden.

Das wichtigste Register ist der **Befehlszähler** (Program Counter – PC). Er zeigt auf den nächsten Befehl, der zur Ausführung abzurufen ist. Die Bezeichnung „Befehlszähler" ist etwas irreführend, da im eigentlichen Sinne nichts *gezählt* wird. Allerdings hat sich dieser Begriff so eingebürgert. Ebenfalls wichtig ist das **Befehlsregister** (Instruction Register – IR), das den momentan ausgeführten Befehl aufnimmt. Die meisten Computer besitzen außerdem eine Reihe weiterer Register, von denen einige universellen Charakter haben,

1 Sofern nicht anders angegeben, werden die Begriffe „CPU" und „Prozessor" in diesem Buch gleichberechtigt nebeneinander verwendet (Anm. d. Übersetzers).

andere nur für spezielle Aufgaben vorgesehen sind. Darüber hinaus gibt es noch Register, über die das Betriebssystem den Computer steuert.

2.1.1 Aufbau der CPU

▶Abbildung 2.2 zeigt die innere Organisation für einen Teil einer typischen Von-Neumann-CPU. Dieser sogenannte **Datenpfad** (Data Path) besteht aus den Registern (normalerweise 1 bis 32), dem **Rechenwerk** (**ALU** – Arithmetic Logic Unit) und mehreren Bussen, die die einzelnen Teile untereinander verbinden. Die Register münden in zwei Eingaberegister der ALU, die in der Abbildung mit *A* und *B* bezeichnet sind. Diese Register speichern die Eingabedaten der ALU, während diese beschäftigt ist. Der Datenpfad spielt in allen Prozessoren eine wichtige Rolle und wird deshalb im gesamten Buch ausführlich behandelt.

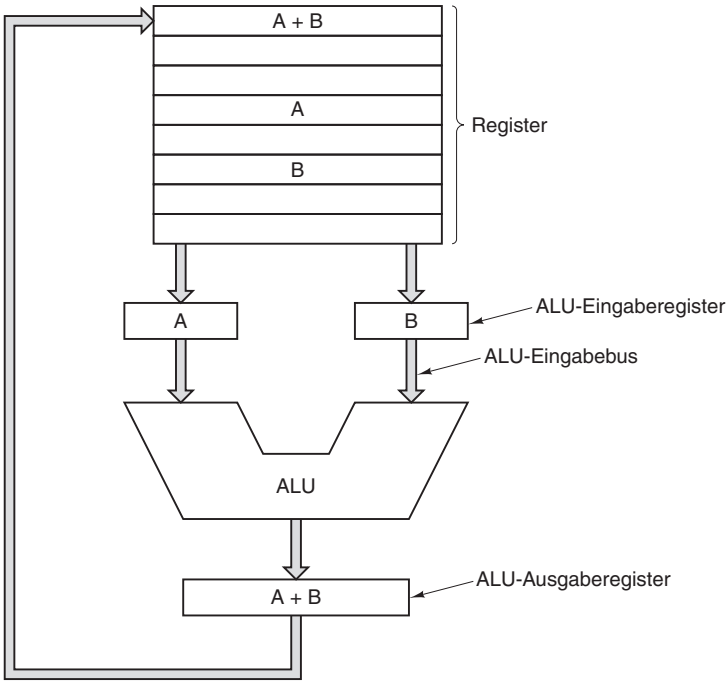

Abbildung 2.2: Der Datenpfad eines typischen Von-Neumann-Rechners

Die CPU besteht aus dem Steuerwerk, dem Rechenwerk (ALU) und den Registern, in denen Daten zwischengespeichert werden, die in der ALU verarbeitet werden.

Die ALU selbst führt Additionen, Subtraktionen und andere einfache Operationen mit ihren Eingabewerten aus und stellt das Ergebnis in das Ausgaberegister. Der Inhalt dieses Ausgaberegisters kann wieder zurück in ein Register gespeichert werden. Später lässt sich das Register bei Bedarf in den Hauptspeicher übernehmen. Nicht alle Entwürfe verfügen über die A-, B- und Ausgaberegister. Das Beispiel zeigt eine Addition, doch können ALUs auch andere Operationen ausführen.

Die meisten Befehle lassen sich in zwei Kategorien aufteilen: Register-Speicher- und Register-Register-Befehle. Mit Register-Speicher-Befehlen können Speicherwörter in Register übernommen werden, wo sie sich beispielsweise in darauffolgenden Befehlen als ALU-Eingaben verwenden lassen. („Wörter“ sind die Dateneinheiten, die zwischen Speicher und Registern bewegt werden. Ein Wort kann beispielsweise eine Ganzzahl sein. Auf den Speicheraufbau geht dieses Kapitel später ein.) Andere Register-Speicher-Befehle schreiben den Inhalt von Registern zurück in den Speicher.

Der zweite Typ sind die Register-Register-Befehle. Ein typischer Register-Register-Befehl ruft zwei Operanden aus den Registern ab, bringt sie in die ALU-Eingaberegister, führt mit ihnen eine Operation durch (z.B. eine Addition oder eine boolesche AND-Verknüpfung) und speichert das Ergebnis wieder in einem Register. Der Vorgang, bei dem zwei Operanden durch die ALU geschleust werden und das Ergebnis gespeichert wird, heißt **Datenpfadzyklus** (Data Path Cycle); er bildet die Basis der meisten Prozessoren. Er definiert maßgeblich, was der Computer tun kann. Moderne Computer besitzen mehrere ALUs, die parallel arbeiten und für unterschiedliche Funktionen spezialisiert sind. Je kürzer der Datenpfadzyklus ist, umso schneller ist der Prozessor.

2.1.2 Befehlsausführung

Die CPU führt jeden Befehl in einer Reihe kleiner Schritte aus. Diese stellen sich ungefähr wie folgt dar:

1. Den nächsten Befehl aus dem Speicher in das Befehlsregister abrufen.
2. Den Programmzähler ändern, damit er auf den nächsten Befehl zeigt.
3. Den Typ des gerade eingelesenen Befehls bestimmen.
4. Falls der Befehl ein Speicherwort benötigt, dessen Position bestimmen.
5. Das Wort bei Bedarf in ein CPU-Register laden.
6. Den Befehl ausführen.
7. Zu Schritt 1 gehen, um den nächsten Befehl auszuführen.

Diese Schrittfolge bezeichnet man auch als **Abrufen-Decodieren-Ausführen-Zyklus** (Fetch-Decode-Execute Cycle). Sie liegt der Arbeitsweise aller Computer zugrunde.

Die Beschreibung der Abläufe in einer CPU ähnelt einem in Umgangssprache formulierten Programm. ▶Listing 2.1 zeigt, wie sich dieses formlose Programm als Java-Methode (d.h. als Prozedur) namens `interpret` neu schreiben lässt. Die damit interpretierte Maschine verfügt über zwei Register, die für die Benutzerprogramme sichtbar sind: den Befehlszähler (`PC`), der die Adresse des nächsten einzulesenden Befehls verfolgt, und den Akkumulator (`AC`), der das Ergebnis einer arithmetischen Operation aufnimmt. Weitere interne Register der Maschine speichern den aktuellen Befehl während seiner Ausführung (`instr`), den Typ des aktuellen Befehls (`instr_type`), die Operandenadresse des Befehls (`data_loc`) und den aktuellen Operanden selbst (`data`). Hier wird davon ausgegangen, dass Befehle jeweils eine einzige Speicheradresse enthalten. Die adressierte Speicherstelle enthält den Operanden, z.B. das zum Akkumulator zu addierende Datenelement.

```
public class Interp {
  static int PC;          // Programmzähler speichert Adresse des nächsten Befehls
  static int AC;          // Akkumulator, ein Register für Arithmetik
  static int instr;       // Register, das den aktuellen Befehl aufnimmt
  static int instr_type;  // Befehlstyp (Opcode)
  static int data_loc;    // Adresse der Daten oder -1, falls keine Daten
  static int data;        // Nimmt den aktuellen Operanden auf
  static boolean run_bit = true;  // Ein Bit, das ausgeschaltet werden kann, um
                          // die Maschine anzuhalten

  public static void interpret(int memory[ ], int starting_address) {
    // Diese Prozedur interpretiert Programme für eine einfache Maschine, wobei
    // jeder Befehl genau einen Speicheroperanden enthält.
    // Das Register AC (Akkumulator) ist für arithmetische Operationen vorgesehen.
    // Z.B. addiert der Befehl ADD eine im Speicher befindliche Ganzzahl zum AC.
    // Der Interpreter läuft so lange, bis das Ausführungsbit (run_bit) durch den
    // Befehl HALT ausgeschaltet wird.
    // Der Zustand eines auf dieser Maschine laufenden Prozesses ist durch den
    // Speicher, den Programmzähler, das Ausführungsbit und den Akkumulator
    // gekennzeichnet.
    // Die Eingabeparameter setzen sich aus dem Speicherbild und der
    // Anfangsadresse zusammen.

    PC = starting_address;
    while (run_bit) {
      instr = memory[PC];      // Den nächsten Befehl nach instr abrufen
      PC = PC + 1;             // Programmzähler inkrementieren
      instr_type = get_instr_type(instr);       // Befehlstyp bestimmen
      data_loc = find_data(instr, instr_type);  // Daten suchen (-1 falls keine)
      if (data_loc >= 0)       // Wenn data_loc gleich -1, kein Operand erforderlich
        data = memory[data_loc];                // Daten abrufen
      execute(instr_type, data);                // Befehl ausführen
    }
  }

  private static int get_instr_type(int addr) { ... }
  private static int find_data(int instr, int type) { ... }
  private static void execute(int type, int data){ ... }
}
```

Listing 2.1: Ein in Java geschriebener Interpreter für einen einfachen Computer

Dass man mit einem Programm die Funktionen einer CPU nachbilden kann, zeigt bereits, dass Programme nicht unbedingt von einer „Hardware"-CPU – also einem Kästchen voller Elektronik – ausgeführt werden müssen. Stattdessen kann ein Programm auch dadurch ausgeführt werden, dass es seine Befehle durch ein anderes Programm abrufen, decodieren und ausführen lässt. Ein Programm (wie das in Listing 2.1), das die Befehle eines anderen Programms abruft, decodiert und ausführt, heißt (wie bereits in Kapitel 1 erwähnt) **Interpreter**.

Aus dieser Gleichwertigkeit von Hardware-Prozessoren und Interpretern ergeben sich wichtige Schlussfolgerungen für den Aufbau eines Computers und den Entwurf von Computersystemen. Nachdem die Maschinensprache *L* für einen neuen Computer spezifiziert ist, kann das Entwicklerteam entscheiden, ob die Programme in *L* direkt durch einen Hardware-Prozessor oder stattdessen durch einen Interpreter ausgeführt bzw.

interpretiert werden sollen. Entscheidet sich das Team für einen Interpreter, muss es auch für eine Hardware-Maschine sorgen, die den Interpreter ausführen kann. Außerdem sind Hybridkonstruktionen möglich, bei denen ein Teil durch die Hardware ausgeführt und ein Teil durch die Software interpretiert wird.

Ein Interpreter zerlegt die Befehle seiner Zielmaschine in kleine Schritte. Daher kann die Maschine, auf der der Interpreter läuft, viel einfacher und billiger als ein Hardware-Prozessor für die Zielmaschine sein. Solche Einsparungen spielen besonders dann eine wichtige Rolle, wenn die Zielmaschine über eine große Zahl von Befehlen verfügt und es sich um relativ komplexe Befehle mit vielen Optionen handelt. Die Einsparungen ergeben sich im Wesentlichen daraus, dass Hardware durch Software (den Interpreter) ersetzt wird.

Die ersten Computer verfügten über kleine einfache Befehlssätze. Doch das Streben nach leistungsfähigeren Computern führte unter anderem auch zu leistungsstärkeren Einzelbefehlen. Schon recht früh erkannte man, dass komplexere Befehle oft eine schnellere Programmausführung zur Folge haben, auch wenn die Ausführung der einzelnen Befehle etwas länger dauert. Gleitkommabefehle und die direkte Unterstützung des Zugriffs auf Feldelemente sind Beispiele für komplexe Befehle. Manchmal ergab sich rein aus der Beobachtung, dass häufig dieselben zwei Befehle aufeinanderfolgten, sodass man diese beiden Befehle zu einem einzigen zusammenfassen konnte.

Die komplexeren Befehle waren besser, weil Operationen manchmal überlappend bzw. parallel auf verschiedener Hardware ausgeführt werden konnten. Bei teuren Hochleistungsrechnern ließen sich die Kosten für diese zusätzliche Hardware ohne Weiteres rechtfertigen. So kam es, dass teure Hochleistungsrechner über viel mehr Befehle als die billigeren verfügten. Doch steigende Kosten für die Softwareentwicklung und Forderungen nach Befehlskompatibilität führten dazu, dass man komplexe Befehle selbst in Computern des unteren Preissegments implementieren musste, wo es mehr auf niedrige Kosten als auf hohe Geschwindigkeit ankam.

Ende der 1950er Jahre hatte IBM (das zu dieser Zeit dominierende Computerunternehmen) erkannt, dass es sowohl für IBM als auch für ihre Kunden viele Vorteile bringt, wenn man nur eine einzige Familie von Rechnern unterstützt, die alle die gleichen Befehle ausführen. IBM führte den Begriff **Architektur** ein, um diese Kompatibilitätsebene zu beschreiben. Eine neue Computerfamilie bekam nun eine einheitliche Architektur, aber viele unterschiedliche Implementierungen, die alle dasselbe Programm ausführen konnten und sich nur hinsichtlich Preis und Geschwindigkeit unterschieden. Doch wie baut man einen billigen Computer, der all die komplexen Befehle der teuren Hochleistungsrechner ausführen kann?

Die Antwort liegt in der Interpretation. Diese erstmals von Maurice Wilkes (1951) vorgeschlagene Technik ermöglichte die Konstruktion einfacher und billiger Computer, die dennoch eine große Anzahl von Befehlen ausführen konnten. Das Ergebnis war die IBM-System/360-Architektur – eine Familie kompatibler Computer, die in Preis und Fähigkeiten nahezu zwei Größenordnungen umspannte. Nur in den teuersten Modellen kam eine direkte Hardware-Implementierung (d.h. ohne Interpretation) zur Anwendung.

Durch **Interpretation** kann eine komplexe Architektur auf einer Familie von Rechnern unterschiedlicher Leistungsfähigkeit kostengünstig realisiert werden.

Einfache Computer mit interpretierten Befehlen hatten noch weitere Vorteile. Zu den wichtigsten gehörten:

1. Die Möglichkeit, falsch implementierte Befehle vor Ort zu korrigieren und sogar Entwurfsmängel in der zugrunde liegenden Hardware auszugleichen.
2. Die Gelegenheit, neue Befehle zu minimalen Kosten selbst nach Auslieferung des Rechners hinzuzufügen.
3. Ein strukturiertes Design, das effizientes Entwickeln, Testen und Dokumentieren komplexer Befehle erlaubt.

Als der Computermarkt in den 1970er Jahren einen gewaltigen Aufschwung erfuhr und das Rechenpotenzial schnell wuchs, kam die Nachfrage nach billigen Rechnern insbesondere den Computerkonzepten mit Interpretern zugute. Die Möglichkeit, Hardware und Interpreter auf einen bestimmten Befehlssatz zuzuschneiden, erwies sich als kostengünstiges Entwurfsprinzip für Prozessoren. Da sich die zugrunde liegende Halbleitertechnik schnell weiterentwickelte, kam den Kostenvorteilen mehr Gewicht zu als den Möglichkeiten für höhere Leistung, und Interpreter-basierte Architekturen etablierten sich als Methode für den Computerentwurf. Fast alle in den 1970er Jahren neu entwickelten Computer – von Minicomputern bis Mainframes – basierten auf der Interpretation.

Ende der 1970er Jahre waren einfache Prozessoren, auf denen Interpreter liefen, mit Ausnahme der teuersten und leistungsstärksten Modelle wie der Cray-1 und der Cyber-Serie von Control Data, weit verbreitet. Durch Interpreter konnte man die höheren Kosten abfangen, die mit komplexen Befehlen verbunden sind. Die Architekten untersuchten immer komplexere Befehle und insbesondere Wege, um die zu verwendenden Operanden zu spezifizieren.

Dieser Trend erreichte seinen Höhepunkt mit dem VAX-Computer der Digital Equipment Corporation (DEC), der mehrere Hundert Befehle beherrschte und mehr als 200 unterschiedliche Varianten kannte, um die in jedem Befehl zu verwendenden Operanden zu spezifizieren. Leider war die VAX-Architektur von Anfang an als Interpreter-Implementierung konzipiert und man hatte sich wenig um die Implementierung eines Hochleistungsmodells gekümmert. Durch diese Einstellung führte man sehr viele Befehle mit geringem Nutzen ein, die sich zudem nur schwer direkt ausführen ließen. Dieser Mangel erwies sich letztlich als tödlich für die VAX und letztlich auch für DEC (Compaq kaufte DEC im Jahr 1998 und Hewlett-Packard übernahm Compaq im Jahr 2001).

Auch wenn die ersten 8-Bit-Mikroprozessoren einfache Maschinen mit sehr einfachen Befehlssätzen waren, hatte man bis Ende der 1970er Jahre sogar bei den Mikroprozessoren auf Interpreter-basierte Konzepte gewechselt. Zu dieser Zeit mussten sich die Entwickler von Mikroprozessoren vor allem damit auseinandersetzen, wie sie die zunehmende, durch integrierte Schaltungen ermöglichte Komplexität in den Griff bekommen. Mit dem Interpreter-basierten Ansatz konnte man einen einfachen Prozessor entwickeln, wobei sich die Komplexität hauptsächlich auf den Speicher beschränkte, der den Interpreter aufnahm. So ließ sich ein komplexes Hardware-Design in ein komplexes Software-Design überführen.

Der Erfolg des Motorola 68000 mit seinem großen interpretierten Befehlssatz und der gleichzeitige Misserfolg des Zilog Z8000 (mit einem ebenso großen Befehlssatz, aber ohne Interpreter) stellten die Vorteile des Interpreter-Ansatzes unter Beweis, wenn es darum ging, einen neuen Mikroprozessor so schnell wie möglich auf den Markt zu bringen. Besonders überraschend war dieser Erfolg angesichts des Vorsprungs, den Zilog hatte (der Z80 als Vorgänger des Z8000 war weitaus populärer als der Vorgänger des