IN DIESEM KAPITEL

Kapitel 1
Blick aus der Vogelperspektive

In diesem Kapitel soll die Vielfalt der Rechner beleuchtet werden. Einerseits in ihrer Struktur und andererseits in der großen Bandbreite ihrer Verwendung. Es folgt der Versuch einer Präzisierung des Begriffs »Rechnerarchitektur«, was nicht ganz einfach ist, da es verschiedene Sichtweisen dazu gibt. Der massenweise Einsatz von Mikroprozessoren in unzähligen Produkten wurde ermöglicht durch die Fortschritte in der Mikroelektronik, solche hochintegrierten Chips herstellen zu können. Diese Entwicklung, basierend auf dem Mooreschen Gesetz, wird retrospektiv betrachtet und es werden die Grenzen aufgezeigt, die sich heute deutlich abzeichnen. Um Strukturen und Prozesse klar darstellen zu können, benötigt man eine leicht verständliche Modellierung. Der letzte Abschnitt in diesem Kapitel widmet sich deshalb diesem Thema.

Das Zeitalter der Computer

Computerbasierte Systeme sind heute in alle Bereiche unseres Lebens eingedrungen. Hardware und Software bilden unabdingbare Grundelemente der meisten technischen Systeme, mit denen wir umgehen oder von denen wir umgeben sind. Neben Materie und Energie ist Information zur neuen Grundgröße geworden. Wir leben heute in einer Informationsgesellschaft. Der Umgang und die Verarbeitung von Informationen ist eine Domäne, in der auch diese seltsamen Maschinen zu Hause sind, die sich noch immer weigern, uns das Denken abzunehmen. Das könnte sich in Zukunft jedoch ändern. Fest steht: Computer haben unsere Welt dramatisch verändert.

Der elektronisch programmierte Wecker, der uns am Morgen weckt, die Smartwatch, die während der Nacht unsere Vitalparameter und Schlaftiefe aufzeichnet und auswertet, die Kaffeemaschine, die den (fast) perfekten Cappuccino produziert oder das Internetradio, das uns beim Frühstück die richtige Musik liefert. Sieht es bei Ihnen ähnlich aus? Vielleicht

lesen Sie beim Frühstück noch die neuesten Schlagzeilen auf dem Tablet-PC oder Sie lassen sich vom Morgenmagazin auf ihrem OLED-TV über den neuesten Stand der Nachrichten informieren. Vor dem Verlassen des Hauses geben Sie dann mittels Ihres Smartphones noch ein paar Steuersignale an Ihr vernetztes Smart Home, um dann in ihr Elektroauto einzusteigen und zur Arbeit zu fahren.

Natürlich befinden sich auch in einem Auto eine riesige Zahl von kleinen Sensoren und Rechnern, die für die verschiedensten Steuerungen zuständig sind: die Motorsteuerung, das Multimedia-Interface, die Klimaanlage, die Erkennung der Verkehrszeichen, die Anti-Schlupf-Regelung und viele andere mehr.

In fast allen Bereichen der Arbeitswelt ist ein Arbeiten ohne Computer heute nicht mehr vorstellbar. Überall sorgen Computer dafür, dass Berechnungen schnell durchgeführt werden können, Prozesse automatisiert werden oder Kommunikation überhaupt stattfinden kann. Dabei werden diese Rechner von Jahr zu Jahr leistungsfähiger. Die Funktionalität eines heute gängigen Smartphones übersteigt die Leistungsfähigkeit des damaligen Steuerungscomputers im Apollo-Mondprogramm der NASA um einiges.

Embedded Systems und Ubiquitous Computing

Dabei werden die Rechner zunehmend kleiner, ja fast unsichtbar, haben aber gleichzeitig an der Zahl dramatisch zugenommen. Wer denkt schon daran, dass beim Aufschlagen einer Glückwunschkarte, bei der dann »Happy Birthday to You« abgespielt wird, ein kleiner Mikroprozessor in der Karte versteckt ist, der diese Funktion ausführt.

Werden die kleinen Rechnerbausteine in ein technisches Umfeld, wie eine Kaffeemaschine, integriert, spricht man von *eingebetteten Systemen* (*Embedded Systems*). Darunter versteht man Rechnersysteme, die auf einen bestimmten Anwendungsbereich spezialisiert sind und in einen technischen Kontext eingebunden werden. Denken Sie etwa an Ihren Küchenherd zu Hause. Dieser erlaubt verschiedenste Einstellungen zum Backen oder Grillen, mit einer Vielzahl einstellbarer Parameter wie Temperatur, Art der Belüftung oder der Garzeit. All dies können Sie über eine Benutzerschnittstelle, meist ein berührungssensitives Display, eingeben. Der Rechner im Inneren des Küchenherds verwendet diese Eingaben dann in den programmierten Funktionsabläufen als variable Steuerungsparameter. Die Unsichtbarkeit der Rechner hinter den eigentlichen Funktionen eines technischen Geräts zeichnet diese eingebetteten Systeme aus. In diesem Sinne ist ein Personal Computer (PC) kein eingebettetes System, da seine Funktion ausschließlich der Datenverarbeitung dient.

Um auf das Beispiel mit dem Auto zurückzukommen: Auch da gibt es, je nach dem Ausstattungsgrad, bis zu hundert verschiedene eingebettete Rechnersysteme, die im Hintergrund ihren Dienst tun. Weiter vorne wurden schon einige aufgezählt. Besonderes Augenmerk liegt dabei auf solchen eingebetteten Systemen, die innerhalb eines eng vorgegebenen Zeitrasters auf ein Ereignis reagieren müssen. Dies sind die *Echtzeitsysteme (Real Time Systems)*. Typisches Beispiel dazu ist das ABS (Anti-Blockier-System). Das ABS soll ein Blockieren der Räder beim Bremsen verhindern. Dazu muss im Millisekundenbereich die Bremskraft und die Rotation der Räder gemessen und die Steuerfunktion berechnet werden, um die Anti-Blockierfunktion für die Räder zu gewährleisten. Echtzeitsysteme sind also spezielle, eingebettete Systeme, die in solchen zeitkritischen Anwendungen eingesetzt werden, bei denen eine Reaktion innerhalb

vorgegebener Zeitschranken erforderlich ist. Ein ABS, das seine Steuersignale erst dann liefert, wenn das Auto die Straße bereits seitwärts verlassen hat, ist unbrauchbar. Die verschiedenen Rechner eines eingebetteten Systems sind meist verteilt realisiert, das heißt an örtlich unterschiedlichen Stellen eines technischen Systems. Sie sind dann über ein Kommunikationssystem miteinander verbunden. So gibt es im Auto einen speziellen genormten Fahrzeugbus (CAN-Bus), über den die verschiedenen Rechnersysteme miteinander kommunizieren.

Die Möglichkeit der Vernetzung stellt heute ein wesentliches Merkmal der Rechnersysteme dar. Alles kann mit allem verbunden werden. Kommunikationstechnologien, wie Bluetooth, wireless LANs, Mobilfunknetzwerke, wie LTE oder 5G-Netzwerke, stehen (fast überall) zur Verfügung und erlauben die Anbindung unserer mobilen Rechnersysteme an ein globales, weltumspannendes Rechnernetz.

Die Allgegenwärtigkeit der Rechner und ihre Möglichkeiten der Kommunikation sind kennzeichnend für sogenannte *ubiquitäre Systeme*, die eine Erweiterung der eingebetteten Systeme darstellen. Ubiquität kennzeichnet das Nichtgebundensein an einen Standort, denn Information ist ein prinzipiell überall erhältliches Gut: Information technology beyond the PC. Heutzutage kann man den Zugang zu beliebigen IT-Diensten mittels sogenannter *Wearables* mit sich herumtragen. Das Smartphone garantiert den Zugang zu informationstechnischen Diensten überall dort, wo es ein Netzwerk gibt. Meist ist es sogar so, dass die ubiquitäre Umgebung selbstständig im Hintergrund wirkt. Das heißt, der Mensch muss gar nicht mehr konfigurierend eingreifen, sondern die Konfiguration erfolgt autonom im Hintergrund. So kann sich das Smartphone automatisch auf die verschiedenen Funknetz-Technologien (3G, LTE) oder den Betreiber (Heimatland, Ausland) einstellen.

Durch die Miniaturisierung der Sensoren können völlig neue Einsatzgebiete erschlossen werden. So hat ein japanischer Unterwäschehersteller ein Gewebe entwickelt, in dem mikroelektronische intelligente Sensoren eingearbeitet sind. Damit kann per Smartphone neben der Herzfrequenz und dem Kalorienverbrauch auch die Haltung des Trägers kontrolliert werden, also wie aufrecht jemand geht.

Intelligente Sensoren werden beispielsweise in der Prozessautomatisierung eingesetzt und können da zu völlig neuen Lösungen führen. So gibt es Sensoren, die nur wenige Millimeter groß sind und die gleichzeitig Beschleunigung, Schwerkraft, Neigung, Bewegung, Luftdruck, Lichtstärke und Feuchte messen und die Messergebnisse per integriertem Internetprotokoll zu einem Smartphone oder PC übertragen können. Damit kann dann beispielsweise der Lagerbestand (Drucksensor auf einer Palette), Defekte an sich drehenden Motoren (Drehsensor), Klimaregelungen (Feuchtesensor) oder einfach offenstehende Fenster oder Türen gemeldet werden.

In Zukunft wird die Zusammenführung der verschiedenen Technologien, wie Sensorik, Aktorik, Ubiqitous Computing über das alles verbindende *Internet of Things and Services* zu einer interoperablen Lösung noch stärker intensiviert werden. Dieser Technologiemix hat auch Auswirkungen auf bestehende Geschäftsprozesse, da durch diese Technologien völlig neue Innovationen möglich sind. Ein Schlagwort in diesem Zusammenhang ist Industrie 4.0. Denken Sie dabei an die flexible Berücksichtigung von Kundenwünschen in einem Produktionsprozess, kontextsensitive Arbeitsplätze oder den Bereich des Predictive Maintenance, in dem sich deutliche Kostenersparnisse erzielen lassen. Denn statt fester Wartungsintervalle, beispielsweise für einen Aufzug, erfolgt die Wartung nun abhängig vom realen Verschleißzustand

eines Systems, das dieses automatisch an einen Überwachungsrechner meldet. Eine Fülle neuer Anwendungsgebiete hat sich bereits etabliert, wie Smart Home, Smart Cars, Smart Cities oder Smart Grids, von den intelligenten Kühlschränken und Zahnbürsten nicht zu reden.

Die Zusammenführung mobiler Informations- und Kommunikationstechnologien gipfelt zurzeit im intelligenten, selbstfahrenden Auto. Unabhängig davon, ob es sinnvoll und verantwortungsbewusst ist, dieses zukünftig in Masse einzusetzen, ist in diesem Projekt ein extrem umfangreicher Datenstrom von verschiedenen Sensordaten gleichzeitig zu bewältigen (mehrere Terabyte pro Fahrzeug) und es ist eine in Echtzeit agierende Rechnerfunktionalität und Aktorik erforderlich.

Wie sieht das intelligente Handwerkerfahrzeug zukünftig aus? Jedes größere Werkzeug, das der Handwerker in seinem Einsatzfahrzeug lagert, besitzt eine eigene ID, beispielsweise in der Halterung des Werkzeugs im Laderaum. Wenn abends die Aufträge für den kommenden Tag festgelegt werden, wird kurz beschrieben, welche Arbeiten bei welchem Kunden ausgeführt werden sollen. Dieser Arbeitsauftrag wird in einen PC eingegeben. Die entsprechende Applikation auf dem PC »berechnet« dann, welches Werkzeug für die auszuführenden Arbeiten mitgeführt werden muss. Diese Liste wird per WLAN oder Mobilfunknetz in den Bordcomputer des Einsatzfahrzeugs übertragen. Bevor der Handwerker morgens zum Kunden fährt, aktiviert er den Bordcomputer, der darauf die IDs aller erforderlichen Werkzeuge im Fahrzeug auf Vorhandensein abfragt. Sollte ein Werkzeug fehlen, das für den Auftrag benötigt wird, bekommt der Handwerker eine Nachricht, entweder auf dem Bordcomputer oder auf seinem Smartphone. An das Pausenbrot muss er dabei (bisher) aber noch alleine denken.

Ziel dieses Buchs ist es, dass Sie mit den Grundlagen und gegenwärtigen Konzepten der Rechnertechnik vertraut werden. Auch wenn die Vielfalt der heutzutage eingesetzten Rechner immens ist, so basieren sie doch alle auf dem fundamentalen Konzept eines Digitalrechners. Dieser besteht im Wesentlichen aus den Komponenten Zentraleinheit oder CPU (Central Processing Unit), einem Speicher für Programm und Daten sowie einer Ein-/Ausgabeeinheit für die Kommunikation des Rechners mit seiner Umgebung. Zwar sind die heutigen Rechner prinzipiell nach diesem groben Prinzip aufgebaut, aber wenn man deren Struktur und die Organisation im Detail betrachtet, haben sich im Laufe der Entwicklung sehr viele unterschiedliche Merkmale herausgebildet, die die Leistungsfähigkeit der heutigen Maschinen begründen. Beispielsweise werden heute mehrere Prozessoren oder Rechenkerne auf einem einzigen Chip integriert als sogenannte Multi-Core-Systeme. Fließbandverarbeitung und Superskalarität, Cachespeicher und virtuelle Speicher, Sprungzielvorhersage und spekulative Befehlsausführung sind strukturelle und funktionelle Eigenschaften, die die heutigen Rechner auszeichnen und zu einer hohen Performance führen. Aber nur keine Panik, alle diese genannten Begriffe werden Sie im Laufe der nächsten Kapitel ausführlich kennenlernen.

Klassen von Rechnern

Kernelement jedes Rechners ist der Prozessor, meist Mikroprozessor genannt. Dieser ist für die Abwicklung von Programmen zuständig. Dazu benötigt der Prozessor einen

Hauptspeicher, der das abzuwickelnde Programm in einer binären Form enthält, ebenso eventuelle Daten, die verarbeitet werden sollen. Kommen noch Einheiten dazu, mit denen Daten für den Prozessor eingegeben werden können, wie etwa Tastatur, Maus oder berührungssensitive Displays sowie Einheiten, die die berechneten Daten wieder an die Umgebung abgeben können, wie Bildschirme oder Drucker, dann hat man es mit einem Rechnersystem zu tun.

Die bekanntesten Rechnersysteme sind die Arbeitsplatzrechner oder *Personal Computer,* kurz PCs. Diese haben sich vor allem in Büroumgebungen als Standard etabliert. Die Verwendung dieser Rechner ist eigentlich für Einzelpersonen konzipiert, in vielen gängigen Konfigurationen, die Sie ja sicherlich von Ihrem eigenen PC kennen. Etwa eine Ausstattung mit einem oder mehreren Flachbildschirmen, Tastatur, Maus und Anschluss an ein *lokales Netzwerk* (Local Area Network LAN), entweder per Kabel oder per wireless LAN. In einem solchen LAN befindet sich im beruflichen Umfeld meist ein *Server*. Dies ist ein Rechner, der zwar prinzipiell wie ein PC aufgebaut ist, aber eine deutlich höhere Leistungsfähigkeit besitzt. Server werden vielfältig eingesetzt. In den Unternehmen laufen dort die unternehmensspezifischen Programmpakete, wie beispielsweise SAP für alle betriebswirtschaftlichen Belange, ORACLE als Datenbankanwendung, die Mailprogramme oder die gesamte Palette der MS OFFICE Software. Server werden häufig auch als sogenannte *Webserver* betrieben, etwa für die Präsentation der Produkte und Leistungen von Unternehmen im Internet. Schließlich gibt es noch die sogenannten *Serverfarmen*, bei denen hunderte oder gar tausende von Servern zusammengeschaltet werden, um eine extrem hohe Leistungsfähigkeit zu erzielen. Denken Sie dabei an die Suchmaschinen von Google oder an Wikipedia.

In den letzten Jahren haben vor allem die mobilen Rechnersysteme einen ungeheuren Boom erlebt. Notebooks und Tablet-Computer werden in riesigen Stückzahlen verkauft. Durch deren WLAN- oder Mobilfunkanschluss können Rechenleistung und Informationen ortsunabhängig bezogen werden, sowohl im geschäftlichen als auch im privaten Bereich. Einzige Voraussetzung ist, dass ein Kommunikationsnetz vorhanden sein muss.

Wenn die Rechner noch kleiner werden und hinter der Funktion eines technischen Gerätes verschwinden, spricht man von den eingebetteten Systemen, wie sie weiter vorne bereits eingeführt wurden. In diesem Marktsegment werden Mikroprozessoren millionen- oder gar milliardenfach verbaut.

In Tabelle 1.1 wurde für die heute gängigen Rechnersysteme im Massenmarkt, wie Personal Computer (PC), Digital Personal Assistent (PDA), auch Wearables genannt, und für die Elemente im Bereich des Ubiquitous Computing (UC) einige Daten zusammengestellt, beispielsweise die ungefähre Markteinführung des jeweiligen Systems und der bevorzugte Anwendungsort. Die weltweit verkauften Einheiten beziehen sich auf das Jahr 2021. Die genannten Preise variieren, je nach Ausbau des Systems. So können einfache PCs für Büroanwendungen relativ preisgünstig erworben werden, während bei getunten Spiele-PCs horrende Preise gezahlt werden. Die Anzahl der Schreibtisch- und Notebook-PCs hatten im Jahr 2011 ihren Höhepunkt erreicht und gehen seitdem langsam, aber kontinuierlich zurück. Die Tablet-Computer hatten 2014 die höchsten Verkaufszahlen erreicht und verharren seitdem etwa auf diesem Niveau. Die Anzahl der verkauften Smartphones hat in den letzten Jahren stetig zugenommen, da viele der Informationsdienste aus dem Internet heute nicht mehr mit den PCs oder Notebooks, sondern mit den Smartphones oder Tablets mobil

	Personal Computer (PC)		Digital Personal Assistant (PDA), Wearables		Ubiquitous Computing (UC)
Einführung in den Markt	Ab 1980		Ab 2000		Ab 2010
Anwendungsort	Schreibtisch		Am Körper getragen (Accessoire)		Internet der Dinge und Embedded Systems
Preis in Euro	300–10.000		150–1500		20–300
Verkaufte Einheiten weltweit im Jahr 2021	PCs	Notebooks	Tablets	Smartphones	Embedded Devices
	340 Mill.	208 Mill.	160 Mill.	1,4 Mrd.	43 Mrd.
Anwendungen	Rechenleistung am Arbeitsplatz, Spielecomputer		Mobiles Telefonieren und Nutzung mobiler Internetdienste		Intelligente Vernetzung von Gegenständen, Überwachung und Steuerung von Prozessen

Tabelle 1.1: Übersicht über die verschiedenen Rechnerkategorien

abgerufen werden. Die Anzahl der im Bereich des Internets der Dinge und der Embedded Systems eingesetzten intelligenten Geräte wächst sprunghaft. So soll nach Meinung von Herstellern die Anzahl dieser vernetzten Geräte im Jahre 2025 die Größe von 75 Milliarden erreichen.

Die breite Verwendungsmöglichkeit der mobilen Rechner und vor allem der Prozessoren für eingebettete Systeme sind nur durch die Fortschritte in der Höchstintegration möglich geworden. Dabei geht es um die Fähigkeit, möglichst viele elektronische Komponenten, meist Transistoren, auf einem Halbleiterchip unterzubringen. Diese Chips sind die Treiber der fulminanten Entwicklung auf dem Markt der mobilen und eingebetteten Rechnersysteme. Dadurch, dass die Anzahl der Transistoren auf einem Prozessorchip von ein paar Tausend auf mittlerweile einige Milliarden gesteigert werden konnte, sind jetzt Rechenkapazitäten möglich, die früher nur ein Großrechner erbringen konnte. Dazu konnte der Leistungsverbrauch der Chips derart reduziert werden, dass auf eine Kühlung des Prozessors verzichtet werden kann. Das ist wesentlich, vor allem, wenn es sich um mobile Systeme handelt. Der mit dieser Höchstintegration einhergehende Preisverfall der Komponenten hat dafür gesorgt, dass heute Rechner mit der Leistungsfähigkeit eines früheren Großrechners für fast jedermann erschwinglich sind.

Da alle diese Rechnersysteme, vom Supercomputer bis zum winzigen eingebetteten Mikroprozessor, im Prinzip die gleichen funktionalen Komponenten besitzen wie Prozessor, Speicher und Ein-Ausgabe-Elemente, natürlich jeweils in extrem unterschiedlichen Ausprägungen, ergibt sich die Frage nach dem prinzipiellen Aufbau und der Organisation eines solchen digitalen Rechners. Man spricht dabei auch von der Architektur eines Rechners.

Aber welche Merkmale sind für eine Rechnerarchitektur eigentlich ausschlaggebend? Wie kann Rechnerarchitektur genauer definiert werden? Damit beschäftigt sich der nächste Abschnitt.

Der Begriff »Rechnerarchitektur«

Bei der Gestaltung von Gebäuden umfasst der Begriff »Architektur« das Entwerfen, Gestalten und die Konstruktion von Bauwerken. In diesem Kontext ist der Begriff »Architektur« nicht eindeutig definiert, sondern die Definition des Begriffs variiert mit den Intentionen des Definierenden. Für den Architekten spielt sicher der visuelle Eindruck, also die Fassade eines Gebäudes als äußere Hülle, eine große Rolle. Für die Menschen, die in dem Gebäude leben oder arbeiten sollen, ist die Funktion essenziell. Dies betrifft die funktionale Gliederung des Inneren sowie das technische Funktionieren des Gebäudes, aber auch ökologische Aspekte und den Energieverbrauch des Gebäudes.

Beim Entwurf eines Rechners spielen viele Aspekte eine Rolle, etwa die Größe und Leistungsfähigkeit, der Energieverbrauch, die Taktfrequenz, mit der der Rechner betrieben werden kann, oder die Menge und Art der Befehle, die der Rechner ausführen kann. Das letztgenannte bezeichnet man als den *Befehlssatz (Instruction Set)* eines Rechners. Für einen Rechnerarchitekten sind der Entwurf und die Organisation der Komponenten, aus denen ein Rechner aufgebaut sind, wie etwa das Rechenwerk oder die Speicher, die Zugriffsmöglichkeiten auf Speicherelemente, innerhalb und außerhalb des Prozessors sowie die Verbindungsstrukturen von großem Interesse.

Die genannten Aspekte betreffen im Wesentlichen den *funktionalen Entwurf*. Für die Realisierung muss die Rechnerhardware aus digitalen integrierten Bausteinen aufgebaut werden, die ihrerseits wieder aus Millionen von Transistoren und deren Verbindungen untereinander bestehen. Dabei werden verschiedene funktionale Einheiten, wie die des Rechenwerks, lokale Speicher oder deren Verbindungsstruktur als hochintegrierte Schaltkreise, auf einem Chip realisiert. Die optimale Platzierung der einzelnen Komponenten auf einem Chip sowie deren Verbindungen untereinander ist ein zentrales Entwurfsproblem in der Mikroelektronik. Immer wichtiger wurde dabei der Energieverbrauch des Chips. Nur ein Entwurf, der eine Minimierung des Energieverbrauchs und der produzierten Wärme eines Chip-Bausteins zum Ziel hat, erlaubt die Verwendung in mobilen Rechnern, die nur mit Akku betrieben und nicht gekühlt werden. Aber ist das Ergebnis des funktionalen Entwurfs gleichzusetzen mit der Architektur eines Rechners? Oder um was geht es dabei?

Wie kann man sich nun einer Definition des Begriffs »Rechnerarchitektur« nähern?

In den 1970er Jahren des vorigen Jahrhunderts wurde eine Definition dieses Begriffs von IBM (Amdahl, Blaauw, Brooks, 1967) aufgestellt, die bis heute in den Lehrbüchern zu finden ist: *Computer architecture is defined as the attributes and behavior of a computer as seen by a machine-language programmer. This definition includes the instruction set, instruction formats, operation codes, addressing modes, and all registers and memory locations that may be directly manipulated by a machine language programmer.*

Die Architektur eines Rechners wurde damit bestimmt durch die Sichtweise, wie ein *Maschinenprogrammierer* den Rechner sieht. Dies ist eine Black-Box Sicht oder Verhaltensbeschreibung, die vom inneren Aufbau des Rechners abstrahiert. Diese Sicht ist bestimmt durch die folgenden Fragen:

- ✔ Welche Befehle gibt es?
- ✔ Wie sind diese Befehle codiert?
- ✔ Welche Datenformate gibt es?
- ✔ Wie können die Daten gespeichert werden?
- ✔ Wie kann auf die Daten zugegriffen werden?

Die Beantwortung dieser Fragen führt zur sogenannten *Befehlssatzarchitektur (Instruction Set Architecture, ISA)*.

Die Instruction Set Architecture (ISA)

Die Befehlssatzarchitektur oder ISA beschreibt das funktionelle Verhalten eines Rechners aus der Sicht eines Maschinenprogrammierers. Sie werden sich jetzt fragen, ob diese Definition denn heute überhaupt noch sinnvoll ist, da fast alle Programme, die man erstellt, gerade keine Maschinenprogramme sind, sondern in einer Hochsprache geschrieben werden. Aber eine andere Perspektive verdeutlicht, dass diese Definition auch heute noch eine Bedeutung besitzt. Die ISA ist nämlich die Schnittstelle zwischen der Software und der Hardware. Die Rechnerhardware ist nämlich so konstruiert, dass sie Maschinenprogramme, basierend auf der ISA, direkt ausführen kann.

Alle Hochsprachenprogramme benötigen einen Übersetzer, der diese Programme in die Sprache der ISA umsetzt, um sie dann auf der Rechnerhardware auszuführen. Diese Übersetzerprogramme heißen *Compiler* oder *Interpreter*. Compiler transformieren Programme, die in einer Hochsprache geschrieben sind, in die jeweilige Maschinensprache, also in ein binäres Format des Programms, die durch eine ISA vorgegeben ist. Somit ist die genaue Kenntnis der ISA eine Prämisse, um überhaupt einen passenden Compiler entwickeln zu können.

Abbildung 1.1 stellt diesen Zusammenhang grafisch dar.

Programme, die nicht in einer Hochsprache, sondern in einer Assemblersprache geschrieben sind, können nach der Übersetzung in die ISA, man sagt in diesem Fall *Assemblierung*, direkt von einem Rechner ausgeführt werden. Oder eine andere Möglichkeit besteht auch darin, Programme durch einen Interpreter ausführen zu lassen. Oberhalb der ISA-Ebene existiert in den meisten Fällen ein Betriebssystem. Dies ist aber keine Voraussetzung für die Transformation eines Programms in die ISA-Ebene.

Lassen Sie mich an dieser Stelle noch kurz den Unterschied zwischen der Kompilierung und der Interpretation von Programmen erläutern.

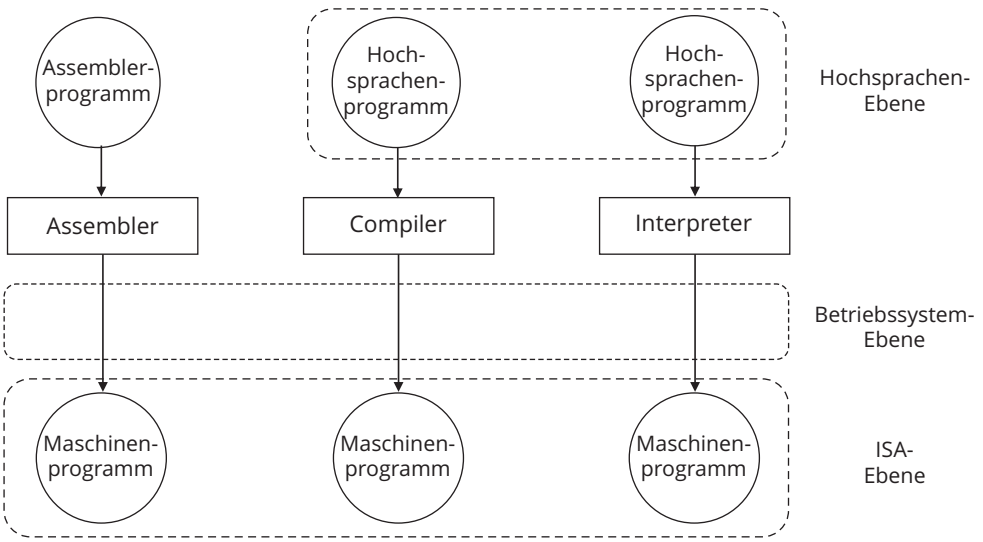

Abbildung 1.1: Die möglichen Transformationen in die ISA-Ebene

Ein *L_i-Compiler* ist eine Software, die ein in der Sprache L_i geschriebenes Programm als Eingabe erhält und ein äquivalentes Programm in der Sprache L_j ausgibt, wobei $j < i$ gilt. Dabei bedeutet j eine gegenüber i tiefer liegende Schicht. Auf der untersten Ebene j handelt es sich dann um ein binäres Maschinenprogramm. Die Ausführung eines L_i-Programms gliedert sich also in zwei Schritte, wenn j die Ebene der Maschinensprache darstellt:

- ✔ Übersetzung eines L_i-Programms in ein L_j-Programm
- ✔ Ausführung des L_j-Programms durch die Hardware

Dem gegenüber gibt es auch die Interpretation eines Programms. Dabei wird ein Hochsprachenprogramm Befehl für Befehl interpretiert und ausgeführt.

Die Ausführung eines Programms durch Interpretation kann durch folgenden Pseudocode beschrieben werden:

cmd_i:= erster Befehl des L_i-Programms

do {

Übersetze cmd_i in eine Befehlssequenz cmd_j der Sprache L_j;

Führe cmd_j aus;

cmd_{i+1}:= nächster auszuführender Befehl des L_i-Programms;

} **while** (L_i-Programm ist fertig abgearbeitet);

Bei der Kompilierung wird das gesamte Quellprogramm, das der Programmierer erstellt hat, als Ganzes vom Compiler in die Zielsprache übersetzt. Bei der Interpretation eines Programms wird die Übersetzung und Ausführung für jeden einzelnen Befehl des Programms ausgeführt.

Die Spezifikation einer Befehlssatzarchitektur (ISA) soll nun präzisiert werden. Es sind folgende Elemente für die Spezifikation einer ISA erforderlich:

- ✔ Der Befehlssatz
- ✔ Das Befehlsformat
- ✔ Die Adressierungsarten
- ✔ Die Datentypen und Datenformate
- ✔ Die Register des Prozessors und das Speichermodell
- ✔ Das Unterbrechungssystem

Der Befehlssatz umfasst die Menge aller Befehle, die der Prozessor eines Rechners ausführen kann. Befehlsformat und Adressierungsarten sind strukturelle Details von Maschinenbefehlen, die im Assemblerformat formuliert werden können und die Sie im Verlauf dieses Buchs noch detailliert kennenlernen werden. Datentypen und Datenformate sind auf der Ebene der Assemblerprogrammierung wenig ausgeprägt. Register sind die im Prozessor vorhandenen Speicherorte und das Speichermodell hängt mit der Philosophie des Rechnerherstellers zusammen. Auch dies wird Ihnen in Kapitel 3 erläutert werden. Unterbrechungen sind bei einem Rechner erforderlich, um gezielt auf interne und externe Ereignisse, die während einer Programmabwicklung auftreten, reagieren zu können.

Die Instruction Set Architecture beschreibt keine Details der Rechnerhardware und der technischen Ausführung von Befehlen, sondern beschreibt eine *abstrakte Schnittstelle* zwischen der Hardware und der Softwareebene des Rechners. Dies soll durch Abbildung 1.2 verdeutlicht werden.

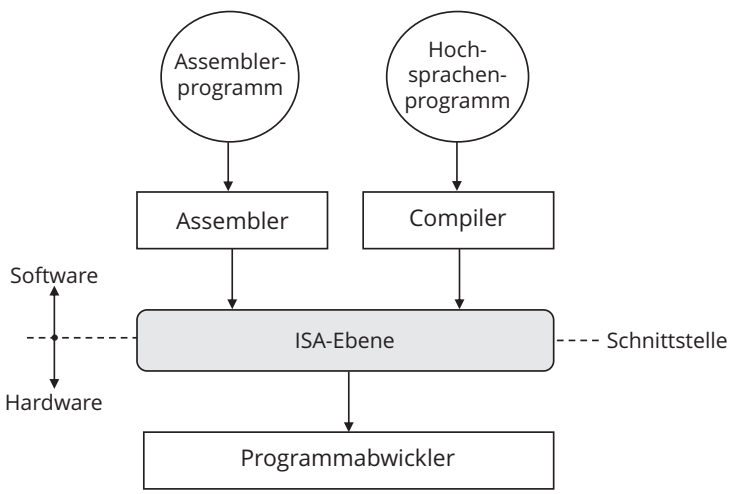

Abbildung 1.2: ISA-Ebene als Schnittstelle zwischen Hard- und Software

Der *Programmabwickler* ist die Instanz, die die Maschinenprogramme ausführt. Man kann sich den Programmabwickler als eine *virtuelle Maschine* vorstellen, die in der Lage ist, die

einzelnen Befehle der ISA-Ebene zu interpretieren und auszuführen. Wobei die Art und Weise, wie der Programmabwickler das intern macht, an dieser Stelle nicht betrachtet wird. Um diese Sichtweise zu verdeutlichen, haben im Jahre 1992 die Rechnerarchitekten des DEC Alpha Prozessors der damaligen Firma Digital Equipment definiert: *The architecture must therefore carefully describe the behavior that a machine-language programmer sees, but must not describe the means by which a particular implementation achieves that behavior*.

Diese Definition behandelt, wie die vorherige Definition von IBM, nur das äußere Erscheinungsbild des Rechners, also letztlich die Befehlssatzarchitektur. Dies ist also im Wesentlichen die Sicht, die ein Anwender von einem Rechner hat. Für die Entwickler eines Programmabwicklers, also eines konkreten Rechners, der in der Lage ist, Programme der ISA-Ebene auszuführen, spielt die Implementierung jedoch eine sehr große Rolle. Die Art der Realisierung eines Prozessors, seine Speicher und die Ein-/Ausgabemöglichkeiten sind essentiell, wenn es um die effiziente Implementierung der abstrakten Architektur geht.

Warum ist es eigentlich überhaupt erforderlich, eine ISA-Ebene zu haben? Kann man nicht einen Rechner konstruieren, der in Java oder in C++ geschriebene Programme direkt ausführt, ohne über den Umweg einer ISA zu gehen? Diese Idee ist bestechend und man fragt sich, warum es sowas nicht gibt. Der Grund dafür liegt in der Historie der Rechnerentwicklung. Zu den Zeiten, als die ersten Rechner konstruiert wurden, gab es Programmiersprachen wie Java, C, C++, Python oder auch FORTRAN noch gar nicht. Als man in der Lage war, frei programmierbare Rechner herzustellen, mussten diese mit einer Maschinensprache, also mit Nullen und Einsen, programmiert werden. Diese Art der Programmierung war extrem fehleranfällig, weil sie für Menschen ganz schlecht handhabbar ist. Also überlegte man sich, wie man die Programmierung vereinfachen konnte. Das Ergebnis waren die Assemblersprachen und später die »höheren Programmiersprachen«.

Die *Assemblerprogrammierung* erlaubte eine gewisse Abstraktion von der Maschinensprache, auch wenn es sich dabei lediglich um eine symbolische Darstellung der binären Maschinenbefehle handelt. Aber nach und nach wurden *höhere Programmiersprachen* wie FORTRAN, ALGOL, Pascal, C, C++ oder JAVA entwickelt, die das Programmieren erleichterten, weil die Programmierung sich jetzt an der Problemlösung orientierte und nicht an den Möglichkeiten eines Maschinenbefehlssatzes.

Aber es gab dabei stets ein ökonomisches Problem. Wenn der Kunde eines Rechnerherstellers in eine Hardwareplattform mit den entsprechenden Programmen investiert hatte, galt es unbedingt zu vermeiden, dass mit der neuen Version des Rechners oder mit der Einführung einer neuen Programmiersprache, die alten Programme nicht mehr funktionsfähig waren. Eine Grundvoraussetzung bei der Weiterentwicklung bereits existierender Rechner war, dass die neuen Modelle *abwärtskompatibel* waren, was bedeutet, dass die alten Programme nach wie vor auf der neuen Hardware lauffähig bleiben. Die neue ISA konnte zwar funktional erweitert werden, indem beispielsweise neue Befehle dazukamen oder schon bestehende Befehle effizienter implementiert wurden, aber die »alten Programme« mussten weiterhin ohne Änderungen auf der neuen Hardware ausgeführt werden können.

Wenn die ISA für einen bestimmten Rechner oder genauer für einen bestimmten Prozessor festgelegt wird, ist damit die Grundlage auch für die weiteren Versionen des Prozessors manifestiert. Ob die Prozessoren von Intel, AMD, IBM oder Motorola kommen und damit alle eine unterschiedliche ISA besitzen, ist mit der Festlegung einer initialen ISA der Grundstein

gelegt. Alle weiteren Versionen des Prozessors müssen sich im Verlauf der Jahre an dieser ISA orientieren. Es hat sich im Verlauf der früheren Jahre gezeigt, dass nachfolgende Versionen eines Prozessors zwar meist funktional deutlich erweitert und optimiert wurden, aber die Abwärtskompatibilität war ein Muss – aus kommerziellen Gesichtspunkten.

Es gab allerdings auch Strömungen, die diese Linie verließen, um leistungsfähigere Rechner zu bauen. Sie werden diese Varianten bei der Diskussion der sogenannten RISC-Architektur kennenlernen.

Die Mikro-Architekturebene

Wenn man die Ebene der Realisierung eines Rechners betrachtet, dann spricht man von der *Mikro-Architektur*. Diese betrifft die konkrete Hardwarestruktur, das heißt, die Art und Weise der Implementierung der verschiedenen Rechnerkomponenten sowie deren Datenpfade. Hier spielen beispielsweise folgende Implementierungstechniken eine Rolle:

- ✔ Art und Anzahl der internen Ausführungseinheiten des Prozessors
- ✔ Die genaue Ausführung des Befehlspipelining (nach Art und Stufenanzahl)
- ✔ Die Verwendung der Superskalartechnik
- ✔ Der Einsatz von Cache-Speichern
- ✔ Der Einsatz der Mikroprogrammierung

Sicherlich werden Sie die gerade genannten Begriffe an dieser Stelle noch nicht einordnen können oder ihre Bedeutung verstehen. Aber diese Begriffe und die damit verbundenen Konzepte werden im weiteren Verlauf des Buches an den entsprechenden Stellen detailliert erläutert.

Rechner werden folglich auf unterschiedlichen hierarchischen Ebenen beschrieben und entworfen, wobei der Detaillierungsgrad von den oberen zu den unteren Ebenen stark zunimmt. Dies werden Sie gleich etwas genauer kennenlernen. Jede Ebene besitzt dabei eine abstrakte Schnittstelle, die den Rechner auf dieser Ebene beschreibt. Eine besonders wichtige Schnittstelle zwischen der Hardware und der Software auf Maschinenebene beschreibt die ISA. Für eine gegebene Befehlssatzarchitektur sind verschiedene Implementierungen hardwaremäßig möglich. Diese werden durch die Mikro-Architekturebene beschrieben.

Die Definition der Rechnerarchitektur aus meiner Sicht

Abschließend möchte ich noch mein Verständnis des Begriffs Rechnerarchitektur vorstellen. Genau wie bei der Architektur von Gebäuden nicht nur die Fassade, also das, was der Betrachter von außen sieht, die Architektur ausmacht, so ist auch im Bereich der Rechner die ISA nur ein Aspekt, der die Rechnerarchitektur prägt. Ebenso wichtig sind die Organisation der Komponenten und Prozesse des Programmabwicklers sowie die verwendete Hardware.

Die Organisation beschreibt dabei die Mikroarchitektur des Prozessors mit seinen Komponenten wie der arithmetisch-logischen Einheit, den internen Registern, der Fließbandverarbeitung

oder dem Bussystem. Auch das hierarchische Speichersystem, die Zugriffsmöglichkeiten darauf und die Kommunikationsmöglichkeiten mit der Umgebung spielen für die Organisation eine große Rolle. Es gibt verschiedene Möglichkeiten, eine gegebene ISA zu implementieren; die Organisation eines Rechners beschreibt genau eine davon, nämlich die, die konkret umgesetzt wurde. Die Hardware beschreibt den detaillierten logischen Entwurf des Prozessors wie die Platzierung der einzelnen Komponenten, ihre Verdrahtung, das sogenannte Routing, die Packaging Technologie oder die Taktfrequenz, mit der der Prozessor betrieben werden kann.

All die genannten Aspekte zusammen definieren aus meiner Sicht den Begriff der Rechnerarchitektur. Dies soll Abbildung 1.3 wiedergeben.

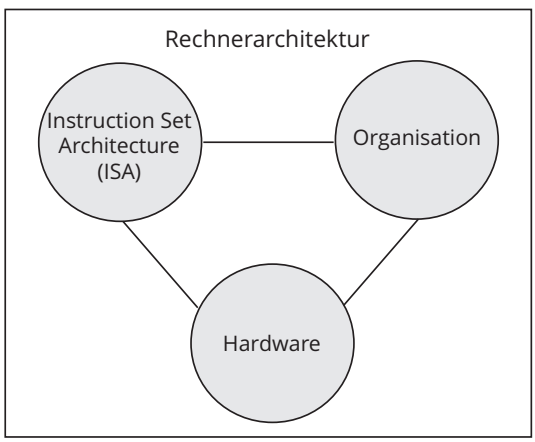

Abbildung 1.3: Wesentliche Aspekte des Begriffs »Rechnerarchitektur« aus Sicht des Autors

Höchstintegration und die Grenzen des Wachstums

In den vergangenen Jahrzehnten wurde die enorme Leistungssteigerung bei den mikroelektronischen Komponenten, aus denen die Rechner aufgebaut sind, durch die folgenden Entwicklungen vorangetrieben:

- ✔ Steigerung der Anzahl der Transistoren auf einem Chip
- ✔ Steigerung der Taktrate von Prozessorchips
- ✔ Fortschritte beim automatisierten Entwurf solcher Chips

Steigerung der Anzahl der Transistoren auf einem Chip

Gegen Ende der 60er-Jahre des vorigen Jahrhunderts wurde eine Prognose aufgestellt, die das *Mooresche Gesetz* genannt wird. Benannt nach dem ersten CEO der Firma Intel, Gordon Moore, der diese Prognose 1965 formuliert hatte. Es handelt sich dabei nicht um ein

Naturgesetz, sondern um eine empirische Beobachtung bei der Produktion von Mikrochips, die er in die Zukunft extrapoliert hatte. Man kann das Mooresche Gesetz in drei Ausprägungen formulieren:

- ✔ Die Anzahl der Transistoren auf einem Prozessorchip verdoppelt sich alle zwei Jahre.
- ✔ Die Verarbeitungsleistung von Prozessoren verdoppelt sich alle 18 Monate.
- ✔ Für den gleichen Preis bekommt man einen Prozessor mit doppelter Leistung in weniger als zwei Jahren.

Diese Vorhersage bezüglich der Integrationsdichte von Transistoren auf einem Prozessorchip war viele Jahrzehnte gültig. Zu Beginn der Hochintegration von mikroelektronischen Komponenten, man spricht dabei auch von *VLSI (Very Large Scale Integration),* konnte man gerade mal einige Tausend Transistoren auf einem Chip unterbringen. Heute ist man in der Größenordnung von Milliarden Transistoren angekommen. So kann man heute (2022) auf einem Chip von 150 Quadratmillimetern Fläche etwa 50 Milliarden Transistoren integrieren.

Die zeitliche Entwicklung der Transistordichte auf einem Prozessorchip im Verlauf der Zeit ist in Abbildung 1.4 dargestellt. Bezüglich der Anzahl der Transistoren steht M dabei für 10^6 und G steht für 10^9.

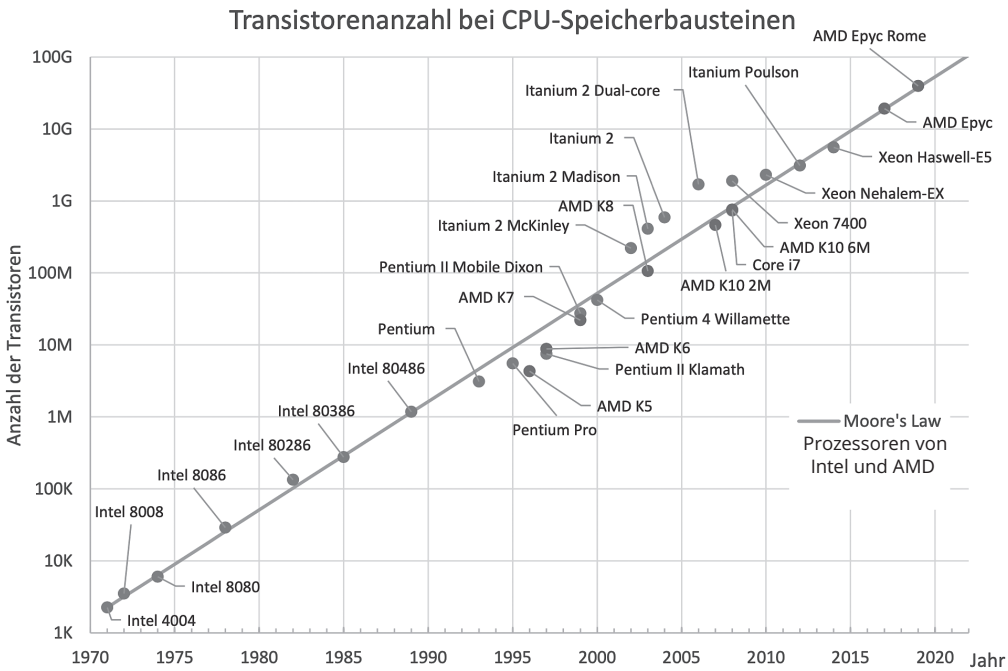

Abbildung 1.4: Anzahl der Transistoren auf einem Prozessorchip im Verlauf der Zeit

Um zu veranschaulichen, welche enorme Steigerungsrate das Mooresche Gesetz für die Integration von Transistoren auf einem Siliziumchip ermöglicht hat, wird im Kasten »Das Mooresche Gesetz in der Automobilindustrie« das Mooresche Gesetz auf ein Beispiel aus dem täglichen Leben angewendet.

Das Mooresche Gesetz in der Automobilindustrie

Wie hoch waren der Spritverbrauch und die Höchstgeschwindigkeit eines PKW im Jahre 1970? Nehmen Sie an, der Spritverbrauch betrug damals 15 l auf 100 km und die maximale Geschwindigkeit wäre 120 km/h gewesen. Dabei handelt es sich sicher nicht um das sparsamste und schnellste Auto. Aber was wäre das Ergebnis, wenn für die Entwicklung eines sparsameren und schnelleren Autos im Jahre 2020 für die Automobilindustrie das Mooresche Gesetz gegolten hätte?

Von 1970 bis 2020 sind es 50 Jahre. Nach dem Mooreschen Gesetz ergibt sich eine Verdopplung der Leistungsfähigkeit alle 18 Monate. Daraus folgt, dass in diesem Zeitraum rund 33 Verdopplungsschritte erfolgt sind, was eine Leistungssteigerung um den Faktor $8{,}6 \times 10^9$ (genau 8589934592) bedeutet.

Für die Höchstgeschwindigkeit des Fahrzeugs resultiert daraus der ungeheure Wert von $1{,}03 \times 10^{12}$ km/h. Damit wäre das Fahrzeug rund 950-mal schneller als das Licht, was natürlich physikalisch unmöglich wäre. Der Verbrauch des Fahrzeugs sinkt von 15 l pro 100 km auf 1,75 Nanoliter. Für den Tank müsste man damit die Größe eines Stecknadelkopfes vorsehen. Zur Veranschaulichung: Etwa 28571 Nanoliter Tropfen ergeben die Größe eines Tropfens, der aus einem Tropfenzähler zum Dosieren von Medikamenten herauskommt.

Wenn man die Höchstintegration der mikroelektronischen Komponenten jedoch genauer analysiert, stellt man fest, dass diese exponentielle Entwicklung nicht ganz so kontinuierlich verlaufen ist, sondern in Sprüngen vonstatten ging. So gab es Phasen, in denen die Integrationsdichte schneller wuchs und es gab Phasen, in denen es für eine Zeitlang fast eine Stagnation gab. Aber im Mittel betrachtet, gilt das Mooresche Gesetz.

Die exponentielle Zunahme der Integrationsdichte war nur deshalb möglich, weil die Strukturgröße der Transistoren stetig verkleinert werden konnte. Es gibt aber eine minimale physikalische Strukturgröße, die heutzutage fast erreicht wird. Diese liegt bei der gegenwärtigen Technologie etwa bei 2 Nanometer (nm). Wenn nämlich die auf dem Chip integrierten Strukturen in die Größe der Atome kommen, treten sogenannte Quanteneffekte auf. Es macht sich dann beispielsweise der *Tunneleffekt* bemerkbar. Dadurch kann es vorkommen, dass leitende Elektronen mit hoher Wahrscheinlichkeit aus den Transistorbahnen »hinaustunneln«, was dazu führt, dass die Prozesse auf dem Chip nicht mehr nach den vorher festgelegten Regeln ablaufen. Aber das wäre dann das Ende der klassischen Miniaturisierung von Prozessorchips, da dadurch kein deterministisches Verhalten des Rechners mehr garantiert werden kann. Prozessoren, die nur mit einer bestimmten Wahrscheinlichkeit richtig funktionieren, dürfen nicht in Rechner eingebaut werden.

Die größten Abweichungen vom Mooreschen Gesetz finden sich aber nicht bei den Prozessoren, sondern bei den Halbleiterspeichern, wie beispielsweise den statischen und dynamischen Speicherbausteinen, also den SRAM- und den DRAM-Bausteinen. Bei diesen Speicherbausteinen kann man heute beobachten, das schon relativ stark vom Mooreschen Wachstumsgesetz abgewichen wird; es stagniert sogar teilweise. Es werden dabei

Strukturgrößen in der Größenordnung von 10 bis 5 Nanometer (nm) auf dem Chip integriert. Als Beispiel: Ein menschliches Haar ist gegenüber einem Transistor der Größe 5 nm etwa 4000-mal so dick.

Da eine zweidimensionale Realisierung von Speicherbausteinen an die Wachstumsgrenze gekommen ist, gehen Halbleiterhersteller heute dazu über, 3D-Chips herzustellen. Es wird also in die dritte Dimension ausgewichen. In diesen dreidimensionalen Chips werden verschiedene Layer oder Schichten vertikal übereinandergestapelt und miteinander verdrahtet.

Der ultimative Prozessor

Um zu demonstrieren, dass kein Wachstumsgesetz in der realen Welt auf ewig angelegt ist, hat die Zeitschrift »Nature« im Jahre 2000 einen Beitrag mit dem Titel »Ultimate physical limits to computation« veröffentlicht, der aufzeigt, was man erhalten würde, wenn das Mooresche Gesetz weitere 250 Jahre gültig wäre. Folgende Ergebnisse kämen dabei heraus:

- ✔ Ein Prozessor könnte dann 10^{51} Maschinenbefehle pro Sekunde ausführen.
- ✔ Dieser Prozessor wäre dann einhundert Sextillionen mal schneller als heutige Hochleistungsprozessoren.
- ✔ Die Packungsdichte der Transistoren wäre so dicht wie die Materie in einem Schwarzen Loch.
- ✔ Beim Einschalten würde der Prozessor heißer als die Sonne sein.

An diesen Konsequenzen können Sie erkennen, wie absurd es wäre, das Mooresche Wachstumsgesetz ad infinitum weiter zu extrapolieren. 10^{51} ist eine unvorstellbar große Zahl, die etwa der Anzahl aller Elementarteilchen unseres Planeten entspricht. Die Schnelligkeit von einhundert Sextillionen entspricht einhundert Millionen Quintillionen, also 10^{36} (eine Eins mit 36 Nullen); auch dies können wir uns nicht vorstellen. Dass durch diesen Prozessor ein schwarzes Loch auf unserem Planeten entstehen würde, wäre für Physiker zwar interessant, aber nicht wirklich wünschenswert, was alle Star-Wars-Fans sicher verstehen. Und bei einer Einschalttemperatur, die heißer als die Sonne ist, wäre die prognostizierte Lebensdauer des Prozessors sicher auch extrem stark limitiert. Aber all dies sollte nur zeigen, dass Wachstumsgesetze in der realen Welt eben immer auf zweierlei Art enden: Entweder das Wachstum geht in eine Sättigung über (Stagnation) oder das System zerstört sich selbst.

Der Performance-Gap

Dass die Integration von Halbleiterbausteinen bei Prozessoren und bei Speicherbausteinen unterschiedlich verläuft, hat leider auch Konsequenzen für die praktische Abwicklung von Programmen im Rechner. In Abbildung 1.5 geht es nicht um die Integrationsdichte – diese ist bei Halbleiterspeichern sogar höher als bei den Prozessoren –, sondern um die Performance.

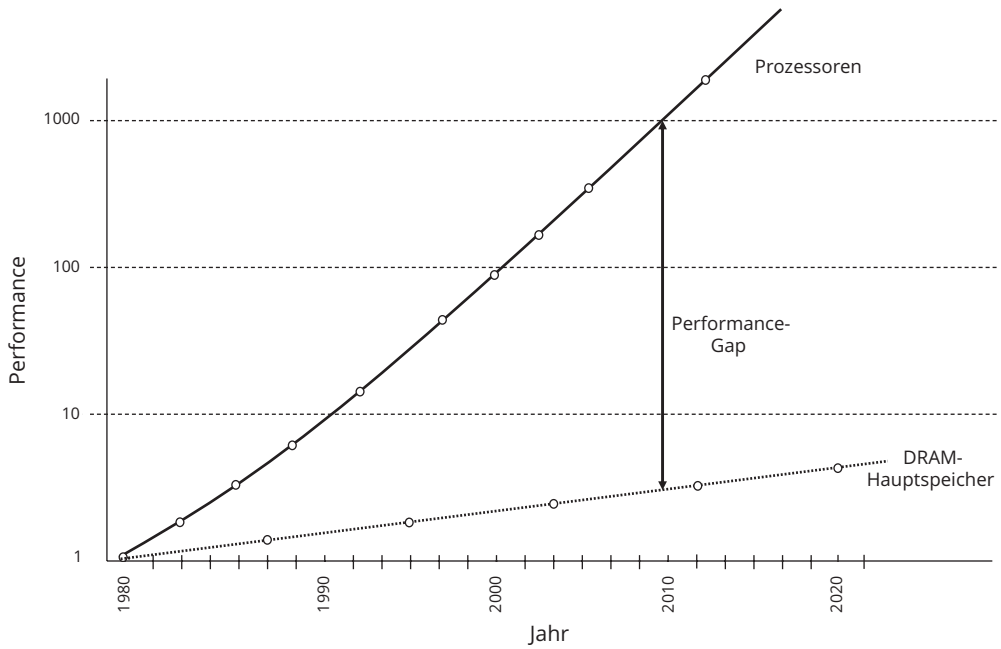

Abbildung 1.5: Der Performance-Unterschied zwischen Prozessoren und DRAMs

So ist in Abbildung 1.5 die Entwicklung der Performance von Prozessoren und von dynamischen RAM-Bausteinen in den letzten Jahrzehnten dargestellt. DRAM-Speicherbausteine werden für die Realisierung von Hauptspeichern in Rechnern eingesetzt.

Performance ist dabei wie folgt zu verstehen:

- ✔ Bei den Prozessoren beschreibt Performance die Geschwindigkeit, mit der prozessorintern Befehle verarbeitet werden können.
- ✔ Bei den DRAM-Speicherbausteinen beschreibt die Performance die Anzahl der Speicherzugriffe pro Zeiteinheit, um bei einem Speicherzugriff des Prozessors die geforderten Daten zur Verfügung zu stellen.

Die Performance der Prozessoren hat sich etwa alle 18 Monate verdoppelt. Die Performance der DRAM-Speicher verdoppelt sich etwa alle 10 Jahre. Man erkennt an der Grafik, dass im Laufe der Jahre die Performance der DRAM-Bausteine zwar gestiegen ist und damit die Zugriffszeit auf den DRAM-Speicher kleiner wurde, aber lange nicht in dem Maße, wie sich die Performance der Prozessoren entwickelt hat. Daraus resultiert nun eine sich stetig entwickelnde Lücke zwischen der Performance der Prozessoren und der DRAM-Speicher, der sogenannte *Performance Gap*. Konkret bedeutet dies, dass die Zeit, die ein Prozessor für den Zugriff auf den DRAM-Speicher aufwenden muss, von Jahr zu Jahr steigt. Oder, anders ausgedrückt, je schneller die Prozessoren werden, umso länger müssen diese beim Speicherzugriff auf die Reaktion des DRAM-Speichers warten.

Alternativen zur Steigerung der Taktrate

Eine Steigerung der Leistungsfähigkeit von Prozessoren erreicht man mit der Erhöhung der Taktfrequenz. Während in den Anfängen der Mikroprozessortechnik die Taktraten im Bereich einiger Megahertz (MHz) lagen, stiegen sie in den Folgejahren drastisch an. Zu Beginn der 2000er-Jahre war man im Bereich von Gigahertz (GHz) angelangt. Damals war man sehr optimistisch und prognostizierte Taktfrequenzen, die bis 12 GHz reichen sollten. Aber es kam ganz anders. Als man bei etwa 4 GHz angelangt war, stellte man fest, dass die Wärmeentwicklung des Prozessorchips überproportional stieg und zwar derart, dass der Chip nicht mehr betriebsfähig war. Denn je höher die Taktfrequenz des Prozessors ist, desto mehr Strom benötigt der Prozessor, in der Konsequenz, dass mehr Energie auf dem Chip produziert wird. Diese muss man effizient abführen, sonst stirbt der Chip den »Wärmetod«. Bei einer hochratigen Taktung von 5 GHz kann man den Chip zwar mit flüssigem Stickstoff kühlen, aber diese Lösung funktioniert nicht bei Personal Computern oder gar bei mobilen Notebooks. Deshalb ließ sich die Taktfrequenz ab 4 GHz nicht mehr steigern.

Nun mussten andere Möglichkeiten gefunden werden, um die Prozessorleistung weiter anzukurbeln. Aufgrund der fortschreitenden Höchstintegration war es möglich, die interne Verarbeitungsbreite von 32 auf 64 Bit zu vergrößern, es entstanden die 64-Bit-Architekturen. Ein weiteres Potential zur Erhöhung der Leistungsfähigkeit bietet die Fähigkeit zur Parallelverarbeitung. Bei einem einzelnen Prozessor ließ sich diese durch die Einführung der Fließbandtechnik, auch Pipelining genannt, umsetzen. Die grundlegende Idee dabei ist, die Befehle eines Maschinenprogramms überlappt auszuführen. Während ein Befehl noch nicht vollständig ausgeführt ist, beginnt man schon die folgenden Befehle abzuarbeiten, sodass die Anzahl der pro Zeiteinheit abgewickelten Befehle deutlich steigt. Allerdings funktioniert dies nur, wenn zwischen den Befehlen keine direkten Abhängigkeiten bestehen. Man spricht hier vom *Instruction Level Parallelism (ILP)*. Dies werden Sie ausführlich in Kapitel 10 kennenlernen. In der weiteren Entwicklung zur Nutzung der Parallelverarbeitung untersuchte man Anwendungsprogramme daraufhin, ob sich in ihnen unabhängige Programmsequenzen, sogenannte Threads finden lassen, die parallel zueinander ausgeführt werden können. Dies nennt man *Thread Level Parallelism (TLP)*. Die Prozessoren wurden hardwaremäßig weiterentwickelt, um dies auch technisch zu unterstützen. Ein Beispiel dafür ist das sogenannte Hyperthreading bei den Intel-Prozessoren. Auch dies führte zu einer weiteren Leistungssteigerung. Die Details zum TLP werden Sie in Kapitel 11 kennenlernen.

Der weitere Ausweg aus dem Dilemma des zu hohen Energieverbrauchs war die Einführung von *Multicore-Prozessoren*, also Prozessoren, die mehr als einen Rechenkern besitzen. Zwei Rechenkerne mit 3 GHZ produzieren weniger Wärme als ein Prozessor mit 4 GHz. Heutige CPUs besitzen beispielsweise 8 oder mehr Kerne. Das neue Problem, das dadurch entsteht, ist allerdings, dass man die Rechenkerne möglichst alle auslasten sollte, was eine größere Parallelisierung der Anwendungssoftware erfordert. Der Programmfluss sollte dabei derart aufteilbar sein, dass man die parallelen Stränge in einem Programm auf die einzelnen Kerne aufteilen kann und damit insgesamt eine Parallelisierung der Programmabwicklung möglich wird. Dies ist aber keine einfache Aufgabe für die Programmierer und Compiler und funktioniert auch nicht immer.

Des Weiteren wurde versucht, die Leistung von Prozessoren durch Architekturänderungen zu verbessern. So ist die Multicore-Technologie ein Ausweg aus dem Dilemma, dass die

Taktfrequenz für einen Prozessor im kommerziellen Einsatz nicht weiter gesteigert werden kann. Eine zusätzliche Möglichkeit besteht darin, dass man bestimmte Funktionen oder Berechnungen, etwa von Gleitkommazahlen oder die Nutzung spezieller Multimediabefehle, in sogenannte *Spezialprozessoren* oder Coprozessoren auslagert. Dies sind Add-ons zu den Standardprozessoren. Die Abwicklung eines Algorithmus wird dann durch den normalen Prozessor und den Spezialprozessor im Zusammenspiel gemeinsam durchgeführt. Hier hängt die Effizienz einer solchen Verarbeitung auch davon ab, inwieweit sich eine Parallelisierung der Verarbeitung mit den beteiligten Prozessoren erreichen lässt.

Fortschritte beim automatisierten Entwurf solcher Chips

Fortschritte im Bereich der Realisierung von Prozessoren durch hochintegrierte Schaltungen wurden durch die Automatisierung erreicht. Man spricht hier von *Electronic Design Automation (EDA)*. Die ideale Wunschlösung wäre die Realisierung eines sogenannten *Silicon Compilers*, also eines Compilers, der eine funktionale Spezifikation in das Hardwarelayout einer elektronischen Schaltung automatisch umsetzt. Leider bleibt dies in der geschilderten Form eine Wunschvorstellung, da zu viele Randbedingungen für einen automatisierten Gesamtentwurf beachtet werden müssen, die nicht alle in formale Regeln gefasst werden können. In vielen Teilen ist die Automatisierung eines Entwurfs allerdings umgesetzt worden.

Man muss bedenken, dass es bei diesem Entwurf ja um Systeme mit Milliarden von Komponenten, meist Transistoren, geht. Deshalb ist aufgrund der Komplexität der Aufgabe ein vollständig automatisierter Entwurf mit einer Garantie der Korrektheit und ohne Interaktion eines Benutzers auch heutzutage nicht umsetzbar. In der Praxis teilt man den Entwurfsprozess in einzelne konsekutive Entwurfsschritte auf. Dabei verfolgt man in der Regel einen Mix aus einem *Top-down-Entwurf* und einem *Bottom-up-Entwurf*. Der Top-down-Entwurf ist das natürliche, aber eher akademische, Vorgehen beim Entwurf komplexer Systeme, indem man das System sukzessiv in kleinere Teilsysteme aufteilt. So wächst der Detaillierungsgrad eines zu entwerfenden Systems systematisch von oben nach unten. Der Bottom-up-Entwurf ist immer dann sinnvoll, wenn es bereits existierende Teilsysteme gibt, die man auf einer bestimmten Entwurfsebene wiederverwenden kann. Dies spart Aufwand und Kosten. An jeden Teilentwurf muss sich eine Verifikation als notwendiger Prüfschritt anschließen. So kann der gesamte Entwurfsprozess als eine kontinuierliche Folge von synthetisierenden und verifizierenden Schritten angesehen werden, der so weit wie möglich durch automatisierte Prozesse unterstützt wird.

Der Entwurf eines hochintegrierten digitalen Bausteins ist ein sehr komplexer Prozess, der auf verschiedenen Ebenen stattfindet. Um die Vielfalt der Sichten und Ebenen zu verdeutlichen, sehen Sie sich bitte die Abbildung 1.6 genauer an.

Es gibt verschiedene Modelle, um die Aspekte beim Entwurf hochintegrierter Schaltungen in systematischer Weise zu einem Entwurfsmodell miteinander zu verknüpfen. Das am besten geeignete Diagramm dazu ist das in Abbildung 1.6 dargestellte Y-Modell für digitale Schaltungen. Im Y-Modell sind die Abstraktionsgrade eines Entwurfs in sogenannten Entwurfsebenen als konzentrische Kreise angeordnet. Die verschiedenen Repräsentationen eines Entwurfs auf einer Ebene sind in drei unterschiedliche Sichten (funktionell, strukturell,

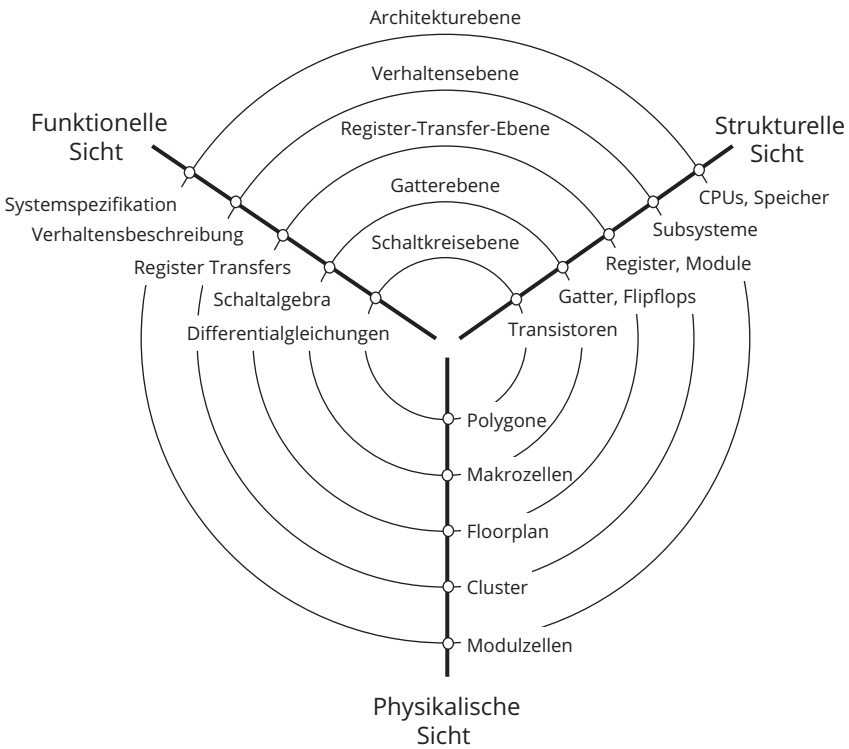

Abbildung 1.6: Die verschiedenen Sichten und Entwurfsebenen bei hochintegrierten Schaltungen

physikalisch) aufgeteilt. Der Entwurf selber gliedert sich in verschiedene Sichten und auch Schichten. Die Gesamtheit aller Sichten liefert eine vollständige Beschreibung aller relevanten Aspekte des zu entwerfenden Systems.

Die *funktionelle Sicht* umfasst im Wesentlichen die Funktionsbeschreibung, also was soll das System überhaupt leisten sowie eine Verhaltensbeschreibung des Systems; ferner Operationen und Prozesse, die vom zu entwerfenden System ausgeführt werden sollen, außerdem sein Zeitverhalten. Für jede dieser verschiedenen Entwurfsebenen existiert eine funktionelle Beschreibung, mit der das Verhalten des Systems auf dieser Ebene beschrieben werden kann. Ein typisches Beschreibungsmittel dieser Entwurfsebene sind dabei mathematische Gleichungen.

Die *strukturelle Sicht* beschreibt die logische Struktur des zu entwerfenden Systems sowie eine abstrakte Implementierung in Form der topologischen Anordnung von Komponenten und deren Verbindungen. Diese erstreckt sich dabei von der obersten Ebene, auf der beispielsweise CPUs und Speicher zu finden sind, bis hinunter zur untersten Ebene; das ist in diesem Fall die Schicht der Transistoren. Das dazu passende Modell sind Graphen, die häufig durch sogenannte Netzlisten repräsentiert werden.

Für das Layout des hochintegrierten Chips ist die *physikalische Sicht* die entscheidende. Sie beschreibt die konkrete Implementierung der hochintegrierten Schaltung, das heißt die Realisierung der (strukturellen) Komponenten mithilfe realer physischer Objekte. Dies

beinhaltet die exakte geometrische Ausdehnung und Anordnung aller Komponenten und Verbindungsstrukturen. Die Kernfrage lautet hier: Wie kann das gesamte System auf einem Chip platziert und integriert werden. Auf der untersten Ebene sind es sogenannte Polygone, die das zweidimensionale Layout des Systems beschreiben. Hier spielt die Geometrie eine entscheidende Rolle, und zwar sowohl für die Platzierung der einzelnen Subsysteme als auch für deren gegenseitige Verdrahtung. Gerade die Verdrahtung ist allerdings oft nicht mehr in einer Ebene durchführbar, sondern erfordert zusätzliche Layer (Schichten), um die Millionen von leitenden Verbindungen kreuzungsfrei durchführen zu können.

Wie bereits beschrieben ist die Verifikation nach jedem Entwurfsschritt eines hochintegrierten Systems, wie einem Prozessor, ein unbedingtes Erfordernis. Das vollständige Testen eines fertigen Chips stellt leider eine quasi unlösbare Aufgabe dar. Der klassische Fall zum Funktionstest eines digitalen Systems wäre der, dass man das System als Black-Box mit In- und Output auffasst. Diese wird dann mit den vorkommenden Eingabewerten beaufschlagt und es werden dann die Ausgabewerte des Systems analysiert. Dies ist das sogenannte *Exhaustive Testing*. Dabei sollte sich die Zuordnung von Eingangs- zu Ausgangswerten gemäß einer funktionalen Spezifikation verhalten. Diese Vorstellung funktioniert allerdings nur bei sehr kleinen digitalen Systemen. Bei einem komplexen System, wie einem Prozessor, ist diese Art des Funktionstests nicht möglich.

An einem einfachen Beispiel können Sie sich dies klarmachen. Ein Prozessorchip besitzt heute mehr als 300 Pins (Anschlussbeine). Nehmen Sie zur Vereinfachung an, dass der betrachtete Prozessorchip 150 Eingangspins und 150 Ausgangspins hätte. Nach dem eben vorgestellten Testmodell des Exhaustive Testings würde das bedeuten, dass 2^{150} binäre Eingangswertekombinationen an den Eingangspins anzulegen wären. Nehmen Sie weiter an, es existiere ein Testautomat, der das Testen vollautomatisiert durchführen könnte. Ein Test besteht dann darin, alle Eingangswertekombinationen mit den zugeordneten Ausgangswertekombinationen auf eine korrekte Zuordnung gemäß einer funktionalen Spezifikation zu testen. Wenn der Automat ein einziges vollständiges Testmuster innerhalb einer Nanosekunde (10^{-9} Sekunden) ausführen könnte, dann würde dieses System 10^{32} Jahre benötigen, um den Chip nach dieser Methode vollständig zu testen. Das bedeutet, dass das Alter des uns bekannten Universums lange nicht ausreichen würde, um alle Tests durchzuführen.

An diesem Beispiel erkennen Sie, dass eine solche Art des Testens völlig illusorisch ist. Leider ist es so, dass trotz anderer aufwendiger Testverfahren, die bei der industriellen Fertigung von Prozessoren eingesetzt werden, solche Chips nicht zu einhundert Prozent vollständig getestet sein können. Fehlschläge, die auf eine nicht vollständige Testung der Prozessorchips zurückzuführen sind, wie beispielsweise der Absturz der ersten Ariane-5-Rakete, kommen immer wieder mal vor, glücklicherweise aber nicht so häufig.

Modellierungstechnik

In diesem Buch wird der Aufbau und die Funktion von Rechnern systemtechnisch beschrieben. Das bedeutet, dass ein ganzheitlicher Ansatz für die Beschreibung der Rechnersysteme gewählt wird. Allgemein gilt, dass sich die Funktion eines dynamischen Systems oder

Teilsystems, wie beispielsweise eines Rechners oder einer Komponente des Rechners, durch ein Verhaltensmodell beschreiben lässt. Dabei muss der innere Aufbau des Systems nicht bekannt sein; es handelt sich dann um eine sogenannte Black-Box. Wenn man davon ausgeht, dass das Verhalten durch das Zusammenwirken seiner Systemteile erzeugt wird, kann man beim Entwurf eines solchen Systems sukzessiv Komponenten einführen, die dann so verknüpft werden, dass das beobachtbare Verhalten produziert wird. So kommt man schließlich zu einer White-Box, die die innere Struktur des Systems offenbart. Diese lässt sich wiederrum durch ein Aufbaumodell beschreiben. Rechner sind sogenannte informationelle oder informationsverarbeitende Systeme. Ihre Aufgabe ist also nicht die Verarbeitung von Energie oder Materie, sondern von Information. Zur Modellierung informationeller Systeme hat sich eine Modellierungstechnik als sehr zweckmäßig herausgestellt: die FMC-Methodik.

Es handelt sich dabei um die *Fundamental Modeling Concepts (FMC)*. Mitte der 1970er-Jahre wurde diese Methodik von Prof. Dr. S. Wendt für die kommunikative Beherrschung der Komplexität großer Softwaresysteme entwickelt und danach kontinuierlich an mehreren Stellen weiterentwickelt, beispielsweise am Hasso-Plattner Institut in Potsdam. Kernpunkt dieser Modellierungstechnik sind *grafische Pläne* zur Visualisierung des Aufbaus von Systemen, der Abläufe in einem oder zwischen Systemen sowie von Datenplänen, die zeigen, aus welchen Komponententypen eine Datenstruktur aufgebaut ist und wie diese Komponententypen zueinander in Beziehung stehen. Die Modellierung informationeller Systeme erfolgt also mit drei Plantypen:

- ✔ Aufbauplänen,
- ✔ Ablaufplänen und
- ✔ Datenplänen.

Wenn in Rechnersystemen eine Aktion initiiert oder durchgeführt werden soll, muss stets ein *Akteur,* oder auch *Instanz* genannt, dafür verantwortlich sein. Eine Instanz kann ein Mensch sein oder eine technische Komponente, beispielsweise eine Hardware- oder eine Softwarekomponente. Für eine Modellierung auf Systemebene spielt diese Unterscheidung noch keine Rolle. Die Begriffe »Instanz« und »Akteur« werden synonym verwendet. In diesem Buch wird jedoch der Begriff »Instanz« verwendet.

Erwähnt werden soll hier noch, dass heutzutage in der Softwareentwicklung mit der UML-Methodik (*Unified Modeling Language*) eine analoge, aber weiter ausdifferenzierte Menge an Darstellungsmitteln verwendet wird.

Aufbaupläne (Instanzennetze)

Um den Aufbau eines Systems aus seinen Komponenten zu dokumentieren, werden Aufbaupläne verwendet. Für die *Aufbaupläne* soll in diesem Buch der Begriff *Instanzennetze* verwendet werden. Graphentheoretisch gesehen sind Instanzennetze bipartite gerichtete Graphen. *Bipartit* bedeutet, dass das Netz aus zwei verschiedenen Typen von Knoten aufgebaut ist und dass Verbindungen grundsätzlich nur zwischen zwei Knoten unterschiedlichen Typs vorkommen dürfen. Beispiele dazu werden Sie gleich kennenlernen.

Modelltechnisch werden Instanzen durch rechteckige Kanten begrenzt; im einfachsten Fall handelt es sich dabei um ein Rechteck. Instanzen sind die aktiven Komponenten eines Systems. Gibt es mehrere Instanzen, die miteinander kommunizieren, nennt man dies ein Instanzennetz. Die Kommunikation zwischen Instanzen kann nur über *Speicher* oder *Kanäle* erfolgen. Modellierungstechnisch werden die Speicher oder Kanäle durch abgerundete Kanten begrenzt; im einfachsten Fall kann dies ein Kreis sein. Exemplarische Beispiele dazu sehen Sie in Abbildung 1.7 a).

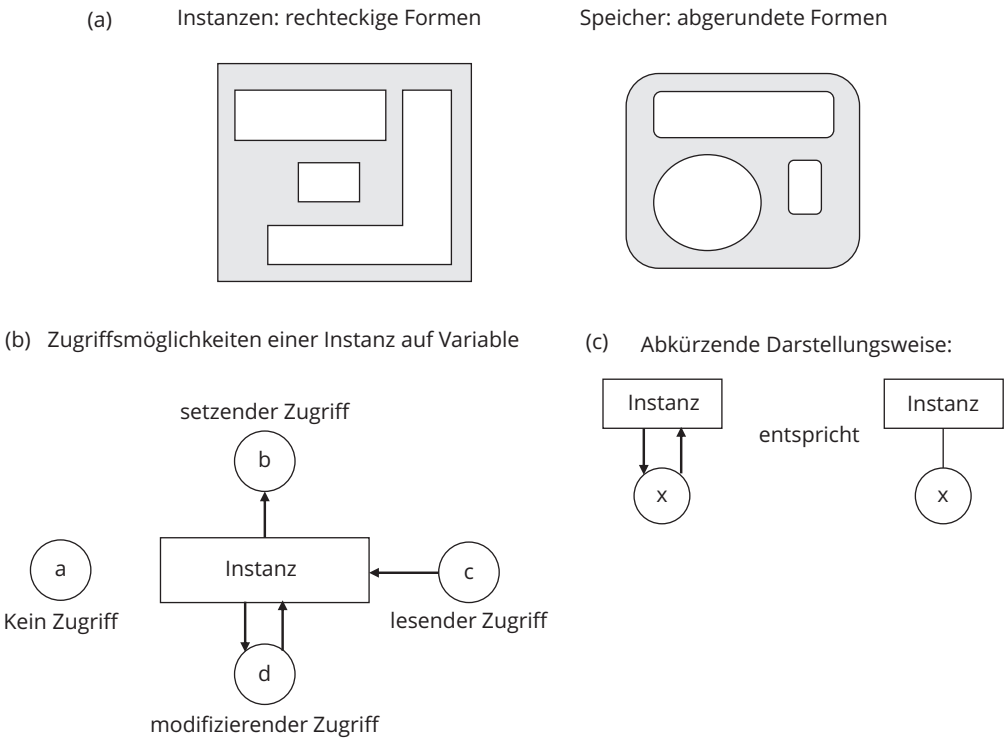

Abbildung 1.7: Darstellung von Instanzen und Speichern

Instanzennetze bestehen also aus zwei Bausteintypen, Instanzen (Akteure) und Speichern (oder Kanälen). Instanzen modellieren die aktiven und Speicher die passiven Komponenten im System. Die Speicher sind die Modellgebilde für die im Rechnersystem auftretenden Variablen. Wertänderungen von Variablen in einem Rechnersystem kann man als Ereignis auffassen. Für das Auftreten eines Ereignisses muss es in einem Rechnersystem immer einen Verursacher geben. Eine Instanz ist ein gedachtes oder reales Gebilde, welches für die Produktion solcher Ereignisse zuständig ist.

Zum Aufbau von Instanzennetzen müsse gewisse formale Regeln eingehalten werden. Der Ausgang einer Instanz darf nur mit einem Speichereingang verbunden und der Eingang einer Instanz darf nur mit einem Speicherausgang verbunden werden. Eine Instanz liefert an einen Speicher beispielsweise den Auftrag »Setze den Speicherinhalt auf einen neuen Wert«. Ein solcher Auftrag ist das Ergebnis einer Instanzenaktion, die eine Funktion der

Instanzeneingänge und des inneren Instanzenzustands ist. Die modellierten Speicher selbst haben keine Verarbeitungsfähigkeit. Instanzen können lesenden, setzenden oder modifizierenden Zugriff auf einen Speicher (Variable) haben. Diese verschiedenen Möglichkeiten sehen Sie in Abbildung 1.7 b).

Beim *lesenden Zugriff* kann die Instanz den Wert einer Variablen nicht beeinflussen. Im Aufbauplan wird dies durch einen Pfeil vom Speicher zur Instanz dargestellt. Vergleichbar wäre dies mit einem Menschen, der vor einer Glasscheibe sitzt und den Text eines Buchs hinter der Glasscheibe lesen kann, aber keinerlei Zugriff auf das Buch hat. Beim *setzenden Zugriff* wird der neue Wert einer Variablen unabhängig vom alten Wert neu geschrieben. Symbolisiert wird dies durch einen Pfeil, der von der Instanz zum Speicher zeigt. So kann beispielsweise ein Rechner eine Zahl in eine Speicherzelle schreiben, unabhängig davon, welche Zahl vorher in der Speicherzelle stand. Der *modifizierende Zugriff* zeichnet sich dadurch aus, dass der neue Wert einer Variablen vom alten Wert abhängig sein darf. Dies wird im Plan durch zwei gegensätzliche Pfeile zwischen einer Instanz und dem zugeordneten Speicher modelliert. Beispielsweise könnte ein Rechner den Inhalt einer Speicherzelle abhängig vom gespeicherten (alten) Wert ersetzen: Steht eine 0 in der Speicherzelle, wird der neue Wert 100 eingeschrieben; steht eine 1 in der Speicherzelle, wird der Wert 50 eingeschrieben. Oftmals wird der modifizierende Zugriff auch vereinfacht durch eine Kante ohne Pfeile dargestellt, so wie in Abbildung 1.7 c) gezeigt.

Die Festlegung dessen, was eine Instanz tun darf, geschieht durch das Programm der Instanz. Dieses Programm legt also das *Verhalten* oder die *Rolle der Instanz* fest. Ein Beispiel eines Instanzennetzes sehen Sie in Abbildung 1.8. Um es etwas konkreter zu gestalten, können Sie sich dabei eine Nachrichtenagentur vorstellen.

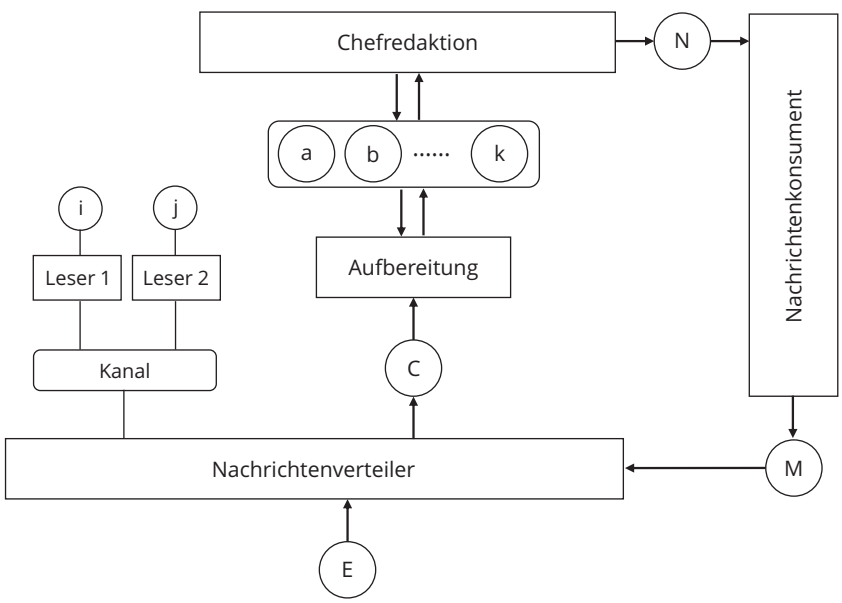

Abbildung 1.8: Beispiel eines Instanzennetzes

Es kommen insgesamt sechs verschiedene Instanzen vor. Die verschiedenen Rollen dieser Instanzen sind der »Nachrichtenverteiler«, die »Aufbereitung« der Nachrichten, die »Chefredaktion« sowie der »Nachrichtenkonsument«. Der Nachrichtenverteiler, der neue Nachrichten von der Umgebung empfängt (Variable E) oder direkt von Konsumenten informiert wird (Variable M), sendet diese Nachrichten an die Nachrichtenaufbereitung (Variable C) und verteilt zusätzlich auch Nachrichten über einen separaten Kanal direkt an Leser 1 und Leser 2, die ihrerseits auch wieder Rückmeldung geben können. Dabei könnte man an Twitter als möglichen Nachrichtenkanal denken. Die Nachrichtenaufbereitung speichert und bereitet die Meldungen (a, b, … k) auf, die anschließend die Chefredaktion nochmals überarbeitet und gegebenenfalls weiter modifiziert. Der Nachrichtenkonsument kann dann die von der Chefredaktion aufbereiteten Nachrichten (Variable N) zur Kenntnis nehmen.

Ablaufpläne (Petri-Netze)

Das Verhalten einer Instanz wird durch ihr Programm festgelegt. Da Programme auch von nicht-sequenzieller Natur sein können, werden sie durch *Ablaufpläne* in Form von Petri-Netzen modelliert. *Petri-Netze* sind ebenfalls spezielle bipartite Graphen. Die Knotenmenge des Graphen zerfällt in die Menge der *Transitionen* oder *Aktionen*, die durch Rechtecke symbolisiert werden und in die Menge der *Stellen* oder *Plätze*, die durch Kreise dargestellt werden. Diese Grundelemente sehen Sie in Abbildung 1.9 a).

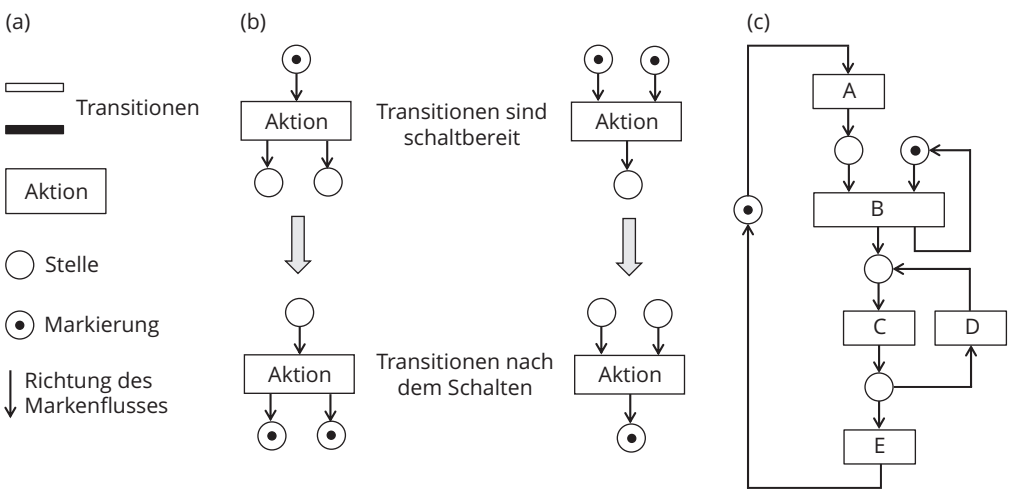

Abbildung 1.9: Modellierung von Abläufen durch Petri-Netze

Transitionen, die durch kleine schmale Rechtecke dargestellt werden, beschreiben Ereignisse die auftreten, wie beispielsweise die Wertänderung einer Variablen (0 ➔ 1) oder etwa das Auftreten einer Unterbrechungsmeldung im Prozessor. Handelt es sich um eine *Aktion*, die eine Anzahl von Transitionen umfassen kann, verwendet man ein größeres Rechteck, das mit einer Bezeichnung beschriftet wird. Manchmal benötigt man nur aus formalen Gründen im Petri-Netz eine Transition, mit der kein reales Ereignis verbunden ist; dann kommt die dritte Variante, das kleine schwarze Rechteck, zum Einsatz.

Zu den in diesem Buch betrachteten Petri-Netzen gehört stets eine *Anfangsmarkierung*. Das bedeutet, dass mindestens ein Platz im Netz durch eine Marke belegt ist. Die Dynamik oder die Abläufe in einem zu beschreibenden System werden durch einen *Markenfluss* simuliert. Markenfluss entsteht, wenn eine Marke von einer Stelle zu einer anderen Stelle fließt. Dazu müssen aber Transitionen aktiv werden; man sagt, sie müssen *schalten* oder *feuern*. Eine Transition kann schalten, wenn alle Eingangsplätze der Transition markiert und alle Ausgangsplätze der Transition, die nicht gleichzeitig auch Eingangsplätze sind, nicht markiert sind. Man nennt dies die *starke Schaltregel*. Sie erahnen, dass es auch andere Schaltregeln für Petri-Netze gibt, die aber hier nicht betrachtet werden müssen. Beim Schalten einer Transition wird von jeder Eingangsstelle eine Marke weggenommen und auf jede Ausgangsstelle eine Marke hingelegt. Dabei darf es bei der hier betrachteten Variante von Netzen nicht vorkommen, dass mehr als eine Marke auf einer Stelle liegt. Beispielhaft sind in Abbildung 1.9 b) zwei mögliche Fälle wiedergegeben. Im linken Teil wird beim Schalten der Transition eine Eingangsmarke eingesammelt und beide Ausgangsplätze mit einer Marke belegt. Während im rechten Teil zwei Eingangsmarken eingesammelt werden, aber nur ein Ausgangsplatz belegt wird. Es gilt also keine Markenerhaltung in den hier betrachteten Petri-Netzen, das heißt, die Anzahl der Marken im Netz muss nicht konstant sein.

In Abbildung 1.9 c) ist beispielhaft ein Petri-Netz mit Anfangsmarkierung dargestellt. Die Schaltreihenfolge der Transitionen, also die Dynamik im zugeordneten System, die durch dieses Petri-Netz modelliert wird, ergibt sich durch den möglichen Markenfluss. In diesem Petri-Netz ist nur die Transition A schaltbereit.

Eine mögliche zeitliche Abfolge des Markenflusses sehen Sie in Abbildung 1.10. Im Teil a) dieser Abbildung ist die Situation wiedergegeben, wenn die Transition A geschaltet hat. Dadurch entsteht dann die Schaltbereitschaft der Transition B.

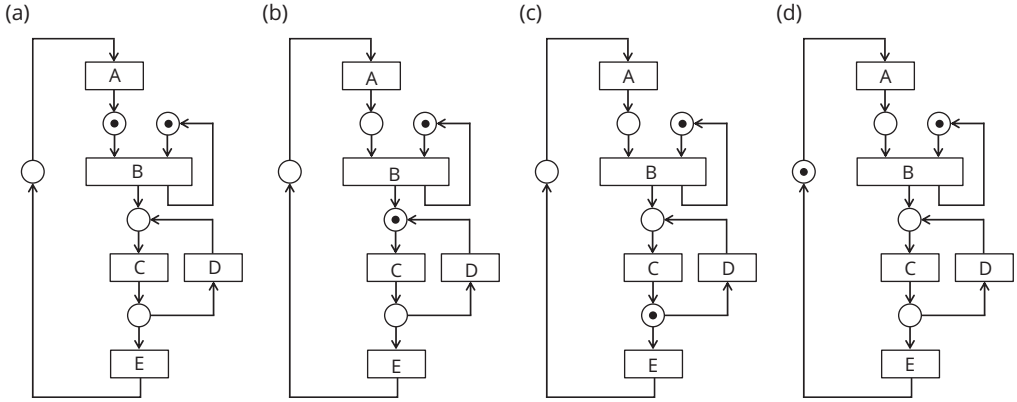

Abbildung 1.10: Markenfluss durch ein Petri-Netz

Interessant ist dabei die Abbildung 1.10 c). Dort liegt die untere Marke auf einem sogenannten *Konfliktplatz*. Im Konfliktfall darf nur eine der am Konflikt beteiligten Transitionen schalten. In der gezeigten Situation ist die Transition D als auch die Transition E schaltbereit. Welche

dieser beiden Transitionen als erstes schaltet und damit die Marke von der Stelle abzieht, kann aus dem Petri-Netz nicht abgeleitet werden, sondern hängt von äußeren Einflüssen des simulierten Systems ab.

Eine mögliche Reihenfolge des Schaltens der Transitionen in diesem Netz wäre: A-B-C-E, dann wäre die Ausgangssituation von Abbildung 1.9 c) wiederhergestellt. In Abbildung 1.10 d) ist genau die Situation wiedergegeben, dass Transition E als erstes geschaltet hat. Das Schalten der Transition E verhindert dabei das Schalten der Transition D. Dieser Konfliktfall ist vergleichbar mit der Situation, dass ein 5 Euro-Stück auf einem Tisch liegt und die zwei Personen **D**etlev und **E**rika am Tisch sitzen und sich um die Münze »streiten«. Wer zuerst zugreift und die Münze wegnimmt, hat gewonnen und die andere Person geht dann leer aus. Wenn **D**etlev schneller ist als **E**rika, dann wäre die Schaltreihenfolge in diesem Fall die Schaltreihenfolge: A-B-C-D. Danach wird durch Schalten der Transition C erneut der Konfliktplatz belegt. Eine mögliche weitere Schaltfolge wäre dann: D-C-E, womit die initiale Netzkonfiguration wieder erreicht wäre.

Wie bereits ausgeführt, wird das Schalten einer Transition als ein Ereignis interpretiert. In informationsverarbeitenden Systemen ist mit einem Ereignis immer die Wertänderung einer Variablen verbunden. Anweisungen zur Wertänderung von Variablen sind den Transitionen zugeordnet und diese können damit beschriftet werden. In Abbildung 1.10 sind die Transitionen mit den Buchstaben A bis E beschriftet, ohne dass in diesem Beispiel damit eine Interpretation verbunden wäre. Aber im Buch werden Sie viele Beispiele kennenlernen, bei denen dem Schalten einer Transition eine bestimmte Ursache zugeordnet wird. Weitere Erläuterungen, auch zu möglichen Konfliktfällen, die in den Ablaufplänen auftreten können, finden Sie insbesondere in Kapitel 9 bei der Diskussion paralleler Abläufe.

Der Wert der FMC-Modellierungstechnik liegt in der klaren Trennung zwischen der Aufbaustruktur (Instanzennetz) eines Systems und der darauf ablaufenden Prozessdynamik (Petri-Netz). Die beim Entwurf eines Systems entstehenden Aufbau- und Ablaufpläne können im weiteren Verlauf eines Entwurfsprozesses kontinuierlich präzisiert und detailliert werden. Für die Realisierung eines Systems kann man ab einem bestimmten Detailierungsgrad der Pläne entscheiden, welche Komponenten des Systems in Hardware und welche in Software implementiert werden sollen. Zur FMC-Begriffswelt gehören auch die Datenpläne wie weiter vorne erläutert. Diese kommen in diesem Buch aber nicht vor, da keine komplexen Datenstrukturen behandelt werden müssen.