Vorwort

> Was jeder für ausgemacht hält, verdient oft am meisten untersucht zu werden.
>
> Georg Christoph Lichtenberg

> Zum Philosophieren sind die zwei ersten Erfordernisse diese: erstlich, daß man den Mut habe, keine Frage auf dem Herzen zu behalten, und zweitens, daß man alles das, *was sich von selbst versteht*, sich zum deutlichen Bewußtsein bringe, um es als Problem aufzufassen.
>
> Arthur Schopenhauer

> Am vollständigsten und strengsten ist jedoch ein Gedanke begründet, wenn alle Motive und Wege, welche zu demselben geleitet und ihn befestigt haben, klar dargelegt sind.
>
> Ernst Mach

Rechnerarchitektur (Computer Architecture) ist die Lehre von den Schnittstellen zwischen Hardware und Software; mit anderen Worten, von Aufbau und Wirkungsweise des Computers aus der Sicht des Programmierers. Der typische Gegenstand der Rechnerarchitektur ist der programmgesteuerte Universalrechner.

Rechnerarchitekturen im heutigen Sinne gibt es seit mehreren Jahrzehnten. Es ist viel erfunden, geschrieben und gebaut worden. Die heutzutage vorherrschenden Rechnerarchitekturen beruhen auf Forschungs- und Entwicklungsleistungen der 50er bis 80er Jahre des vorigen Jahrhunderts. Nur wenige architekturseitige Verbesserungen waren wirklich entscheidend. In der Praxis waren es vor allem die Technologien, welche die Verarbeitungsleistung um Größenordnungen erhöht haben, die Werkstoffe, Fertigungsverfahren, Algorithmen, Compiler usw.

Die Rechnerarchitektur ist keine exakte Wissenschaft. Viele Entscheidungen der Architekturentwicklung werden auf Grundlage von Erfahrungen und Messungen getroffen, nicht wenige sogar nur gefühlsmäßig. Oft spricht das Marketing ein gewichtiges Wort mit.

Das vorliegende Buch betrifft einen Versuch, diesen Entwicklungsstand zu überwinden. Unser Ansatz beruht darauf, die Rechnerarchitektur aus Sicht der Schaltwerke zu betrachten. Da alle Maschinen Schaltwerke sind, steht zu hoffen, daß sich auf diese Weise ein grundsätzlicher Zugang gewinnen läßt, der neben den verschiedene Auslegungsformen herkömmlicher Universalrechner auch Maschinen mit mehreren Prozessorkernen, Spezialmaschinen, Datenflußmaschinen, Datenstrukturmaschinen usw. einschließen kann.

Es geht hier nicht um grundsätzlich neue Arten des Computing, um grundsätzliche Alternativen zum programmierbaren Universalrechner. Genaues Rechnen und exakte Logik wird man immer brauchen – und sei es, um die künstliche Intelligenz zu beaufsichtigen und das zu tun, was sie nicht kann. Deshalb bleiben wir bei der binären Arbeitsweise und den digitalen Schaltungen.

Unser Entwicklungsweg beginnt nicht an der architekturseitigen Programmschnittstelle (Application Programming Interface, API) zwischen Software und Hardware, sondern führt umgekehrt von den Funktionseinheiten der Hardware zur API, die so zu gestalten ist, daß man die Funktionseinheiten so gut wie möglich ausnutzen kann, um die Anwendungsaufgaben zu erledigen. Die Funktionseinheiten bezeichnen wir als Ressourcen. Es ist ein allgegenwärtiger Ausdruck in der Computerliteratur. Jede Maschine läßt sich als eine Sammlung von Ressourcen auffassen, wie Speichereinrichtungen, Informationswege, Verknüpfungsschaltungen usw. Deshalb ist es folgerichtig, diesen Begriff zu verwenden und eine Theorie der Rechnerarchitektur von den Ressourcen her aufzubauen. Die Ressourcen bestimmen die Architektur, nicht umgekehrt. Ressourcen kann man formal als algebraische Strukturen betrachten. Deshalb wird die hier in Rede stehende Architektur mit dem Begriff der Ressourcen-Algebra bezeichnet. Im Englischen kann sie gar nicht anders heißen als ReAl Computer Architecture[1]. In Band 1 wurden die Grundlagen dargelegt. Hier folgt der nächste logische Schritt. Wir wollen eine hinreichende Erfahrungsbasis gewinnen und für die Ressourcen-Algebra etwas lernen. Hierzu müssen wir den Stand der Technik erfassen und diskutieren, den aktuellen ebenso wie den überlieferten. Der Umfang des Materials, das dabei angefallen ist, hat es als zweckmäßig erscheinen lassen, den im Vorwort des ersten Bandes angekündigten zweiten Band aufzuteilen, in den vorliegenden zweiten Band für die technischen Grundlagen der Universal- und Spezialprozessoren und in einen weiteren (dritten) Band, der die Tiefenstrukturen der logischen Grundlagen und Wirkprinzipien (Principles of Operation) der Architekturen behandelt[2]. Beide Bände gehören zusammen. Es ist ein einziges Thema: die Tiefenstrukturen typischer Rechnerarchitekturen und Maschinen zu ventilieren und dabei nach Verbesserungsmöglichkeiten und Alternativen zu suchen. Der Begriff der Tiefenstruktur kommt aus der Sprachwissenschaft. Er bezeichnet die grundsätzliche Bedeutung, das Allgemeine, das allen jeweils in Rede stehenden Schaltungen, Abläufen, Wirkprinzipien usw. Gemeinsame. Schaltkreise, Interfaces und Standards betrachten wir als Beweise durch effektive Konstruktion. Da das alles offensichtlich funktioniert, können wir sowohl die zugrunde liegenden Technologien nutzen als auch Merkmale und Bestandteile herauslösen und für unsere Zwecke adaptieren.

Der vorliegende Band kann auf zweierlei Weise gelesen werden:

1. Als Fortsetzung der Ausführungen zur Rechnerarchitektur als Technikwissenschaft im allgemeinen und zur Ressourcen-Algebra im besonderen. Hierzu setzen wir den Inhalt des ersten Bandes als bekannt voraus.

2. Als Lehr- und Hilfsbuch der Grundlagen der Rechnerarchitektur, das sowohl einen Überblick über Tiefenstrukturen typischer Architekturprinzipien und Schaltungslösungen gibt als auch Anregungen, Verbesserungsvorschläge und Entwurfsideen enthält. Die Ausführungen zur Ressourcen-Algebra, zu ReAl-Maschinen, Ressourcenvektorbefehlen, Operatoren usw. können dabei übergangen werden.

1. Das Wortspiel ergibt sich von selbst und wird gern mitgenommen (pun intended).
2. In Kurzform: Band 2 = Hardware, Band 3 = Architekturprinzipien.

Der Inhalt im Überblick

Unsere Erläuterungen sind vor allem generische Darstellungen, die das Ziel haben, Grundsätzliches zu beschreiben und Tiefenstrukturen zu veranschaulichen. Sie setzen eine elementare Erfahrungsbasis voraus, die dem Stand der Technik entspricht. Das betrifft die üblichen Rechnerarchitekturen und die Wirkprinzipien (Principles of Operation) typischer Prozessoren. Die Fachkenntnisse können aus dem Studium einschlägiger Standardwerke und Architekturhandbücher gewonnen, aber auch aus der Programmier- und Entwurfspraxis abgeleitet werden.

Kapitel 1: Leistungssteigerung im Einzelprozessor

Das Leistungsvermögen des herkömmlichen Einzelprozessors ist begrenzt. Die prinzipiellen Grenzen ergeben sich aus dem grundsätzlichen Programmiermodell. Wie kann man sie überwinden? Welche Merkmale des Programmiermodells dürfen aufgegeben, welche müssen beibehalten werden? Die Antwort auf die zweite Frage entscheidet, welche Architektur- und Erfindungsgedanken überhaupt in Betracht gezogen werden können. Das eine Extrem wäre, alles über Bord zu werfen und neu anzufangen, das andere, das Vorhandene beizubehalten und die Neuerungen behutsam einfließen zu lassen (strikte Abwärtskompatibilität). Praxislösungen liegen irgendwo dazwischen. Manche grundsätzlichen Merkmale müssen beibehalten werden, weil es gar nicht anders geht. Man denke beispielsweise an das sukzessive Lesen von Steuerangaben[3] aus Speichern und an die Unterstützung der Grundrechenarten. Die Maschine muß rechnen können und sich steuern lassen. Betrachtet man die typische Praxis, die Architekturen, die am Markt vorherrschen, ist leicht zu erkennen, daß der Abwärtskompatibilität eine sehr große Bedeutung beigemessen wird. Der Ehrgeiz, radikal Neues herauszubringen, hält sich in Grenzen. Es bleibt also beim herkömmlichen Programmiermodell und traditionellen Befehlssatzarchitekturen (Instruction Set Architectures, ISAs).

Grundsatzlösungen der Leistungssteigerung bestehen im Vermeiden von Wartezuständen, im Pipelining, in der Erweiterung der Befehlssatzarchitektur auf die Vektorverarbeitung und in der Ausnutzung des innewohnenden Parallelismus (Inherent Parallelism). Im Gegensatz zur Parallelverarbeitung (im nächsten Kapitel) bleiben wir hier bei einem einzigen Befehlsstrom, versuchen aber, mehrere Befehlswirkungen gleichzeitig zu erbringen. Das erfordert ensprechend viele gleichzeitig nutzbare Verarbeitungswerke. Die Frage ist, wie man sie steuert, mit herkömmlichen oder speziellen, außergewöhnlich langen Befehlen.

Kapitel 2: Parallelverarbeitung

Wenn die Verarbeitungsleistung des einzelnen Prozessor nicht ausreicht, liegt es nahe, mehrere einzusetzen. Deshalb – weil der Gedanke naheliegt und es letzten Endes gar nicht anders geht – ist die Parallelisierung seit jeher ein wichtiges Gebiet der Forschung und der erfinderischen Bemühungen. Im Gegensatz zum innewohnenden Parallelismus geht es hier um die Nutzung mehrerer gleichzeitiger Befehlsströme und um Wirkprinzipien, die vom Herkömmlichen grundsätzlich abweichen. Damit strebt man an, die Verarbeitungsleistung zu erhöhen, die Kosten zu senken

3. Um nicht gleich von Befehlen zu reden. Es könnten u. a. auch Datenflußsteuerwörter oder ReAl-Operatoren sein.

und mehrere Anwendungsfunktionen gleichzeitig zu unterstützen. Die Kostensenkung erhofft man sich dadurch, daß man anstelle eines einzigen besonders leistungsfähigen Prozessors mehrere kleine und kostengünstige einsetzt. Auch liegt der Gedanke nahe, dann, wenn man mehrere Anwendungsfunktionen gleichzeitig braucht, jeder einen eigenen kleinen Prozessor zu geben anstatt einen teuren Hochleistungsprozessor im Mehrprogrammbetrieb (Multitasking o. dergl.) einzusetzen. All das ist aber keineswegs so einfach, wie es auf den ersten Blick aussehen mag. Deshalb werden diese Fragen systematisch untersucht.

Kapitel 3: Rechnerarchitektur und Schaltungstechnik
Die Schaltungstechnik bestimmt letzten Endes das Leistungsvermögen der Maschine, vor allem in Hinsicht auf die Operationen je Sekunde, die Übertragungsraten, die Stromaufnahme usw. Wir betrachten die Signalflüsse auf der Register-Transfer-Ebene, grundsätzliche Merkmale der Halbleitertechnologien in Hinsicht auf Verzögerungszeiten und Stromaufnahme, die Nutzung der programmierbaren Schaltkreise, vor allem der FPGAs, sowie Schaltungslösungen der Mehrbefehlsverarbeitung und des Pipelining.
Mit welchen Taktfrequenzen die Maschine betrieben werden kann, ergibt sich daraus, was die Gatter, Flipflops und Signalwege zulassen. Zu den wichtigsten Entwurfsentscheidungen gehört, ob man die Maschine schnell und schmal oder langsamer und breit auslegt, also mit extrem hohen Taktfrequenzen und geringerer Verarbeitungsbreite oder mit größerer bis extremer Verarbeitungsbreite und niedrigeren Taktfrequenzen. Es ist eine Frage der Halbleitertechnologie. Mit den heutzutage vorherrschenden CMOS-Technologien ist die zweite Auslegung oftmals eindeutig überlegen. Eine verblüffende Erkenntnis ist, daß die Informationstransporte mehr Energie benötigen als die Informationsverarbeitung. Das alles führt auf den grundsätzlichen Entwurfsgedanken, nicht um jeden Preis Transistoren zu sparen, sondern die Ressourcen eher großzügig zu dimensionieren, so daß sie soweit wie möglich mit lokalen Daten arbeiten können, es aber mit den Taktfrequenzen nicht zu übertreiben. Besser in längeren Zyklen mehr leisten.
Es liegt nahe, FPGAs einzusetzen, um Maschinen nach eigenen Ideen tatsächlich zu bauen. Dann muß aber alles mit Logikzellen implementiert und über vorgefertigte Signalwege miteinander verbunden werden. Die Taktfrequenzangaben in den Datenblättern der FPGAs betreffen nur die Taktfrequenzen, mit denen die Takterzeugungs- und Verteilungsschaltungen betrieben werden können. Takte in Anwendungsschaltungen sind deutlich langsamer (Slowdown). Mit einer Taktverlangsamung im Verhältnis 1:10 und mehr als zehnmal so vielen Transistoren muß, damit es sich wirklich lohnt, die Schaltung im FPGA sehr viel leistungsfähiger sein als eine fertige Prozessorplattform in vergleichbaren Technologien. In der Praxis wird man wohl kaum herkömmliche Prozessoren mit FPGA-Zellen nachbauen, sondern Verbundlösungen anstreben. Aber auch dabei gibt es einiges zu untersuchen und zu bedenken.
Die letzten Abschnitte des Kapitels betreffen schaltungstechnische Grundsatzlösungen der Mehrfachverarbeitung. Zunächst geht es um einen einzigen Befehlsstrom. Wie kann man den Parallelismus ausnutzen, der ihm innewohnt (Inherent Parallelism)? Dafür braucht man Schaltungslösungen. Deren Beschreibung beginnt mit einer elementaren Taxonomie, die die Vielfalt der Prinzipien in überschaubare Alternativen unterteilt.

Abschließend stellen wir dar, wie Schaltungslösungen dabei helfen können, zwischen verschiedenen Befehlsströmen umzuschalten, die durch eine einzige Maschine laufen.

Kapitel 4: Universal- und Spezialmaschinen

Es ist seit langem üblich, sowohl Universalrechner mit speziellen Einrichtungen zu ergänzen als auch selbsttätig arbeitende Spezialmaschinen zu bauen und dabei auch ungewöhnliche Wirkprinzipien zu implementieren. Man kann nahezu alle Funktionseinheiten eines üblichen Universalrechners mit Zusatzeinrichtungen erweitern und diese über verschiedene Anwendungsprogrammschnittstellen (APIs) ansprechen, beispielsweise über zusätzliche Befehle, verlängerte Befehle, E-A-Befehle oder reservierte Adreßbereiche.

Geht es darum, Maschinen zu bauen, die hochkomplexe Algorithmen ausführen sollen, ist es zweckmäßig, verschiedene alternative Architekturkonzepte zu untersuchen. Alles will genau überlegt sein. Viele kleine Maschinen oder eine große? Ein Hardware-Software-Verbund oder ein großer Universalprozessor, der alles alleine kann? Was ist die bessere Plattform – der Universalrecher, der von Beschleunigungseinrichtungen (Akzeleratoren) umgeben ist, oder die Spezialmaschine, in die Universalprozessoren für Kommunikations- und Unterstützungsfunktionen eingebaut sind?

Sind extreme Leistungsanforderungen zu erfüllen, wird man wohl darüber darüber nachdenken müssen, echte Spezialmaschinen zu bauen. Wir betrachten grundsätzliche Entwurfsentscheidungen, Prinziplösungen, Leistungsklassen und Architekturkonzepte. Zwei grundsätzliche Architekturkonzepte, die für Spezialmaschinen in Betracht kommen, sind die Datenflußmaschine und die Datenstrukturmaschine. Elementare Datenflußschaltungen ergeben sich, wenn man Funktionseinheiten gemäß dem Datenfluß der Anwendungsproblemlösung miteinander verbindet. Man stelle den Datenflußgraphen auf, sehe für jeden Knoten eine Funktionseinheit vor und verbinde alles so miteinander, wie es die Kanten des Graphen vorgeben. Ist alles hart verdrahtet (hard-wired), ergibt sich eine Spezialmaschine zum Lösen des jeweiligen Anwendungsproblems. Ist alles einstell- und veränderbar, ergibt sich das digitale Äquivalent des herkömmlichen Analogrechners. Nun ist das zunächst nur eine Metapher, weil man die Verbindungen zwischen den Funktionseinheiten wohl kaum mit Draht stöpseln wird. Universelle Datenflußmaschinen ergeben sich, wenn man die Funktionseinheiten an programmierbare Koppelschaltungen anschließt, die Verbindungen der Art Jeder-mit-Jedem herstellen können. Die Verwandtschaft mit den ReAl-Maschinen ist offensichtlich. Eine solche Datenflußmaschine ist im Grunde eine vereinfachte ReAl-Maschine.

Das Prinzip der Verbindungen Jeder-mit-Jedem ist nicht schwer zu verstehen, hat aber in der Praxis enge Grenzen. Sind es Direktverbindungen über nichtblockierende Schaltverteiler, wird der Aufwand bald untragbar hoch, sind es Netzwerkverbindungen, ergeben sich lange Latenzzeiten. Ein Ausweg besteht darin, die Datenflußwege hart zu verdrahten. Eine Topologie, die vielen elementaren Datenflußgraphen entspricht, ist der invertierte Binärbaum. Operationswerke mit zwei Eingängen und einem Ausgang werden so miteinander verbunden, daß oben die Operanden eingespeist werden und unten das Ergebnis herauskommt. Die grundsätzliche Alternative besteht im universellen Datenflußprinzip der Theorie. Die Verfügbarkeit der Daten und

die Abhängigkeit der Daten untereinander bestimmt die Verarbeitungsreihenfolge. Vom Datenflußgraphen wird abstrahiert. Er kommt gar nicht vor. Grundlage der universellen Datenflußmaschine ist der assoziative Speicher. Wir beschreiben die Wirkungsweise anhand eines einfachen Beispiels. Anhand dessen zeigen wir durch effektive Konstruktion, daß sich in der assoziativen Datenflußmaschine die gleichen Wertverläufe ergeben wie in Operationswerken, die gemäß dem Datenflußgraphen zusammengeschaltet sind.
Das universelle Datenflußprinzip hat seine Leistungsgrenzen. Wenn man versucht, es in größerem Umfang zu implementieren, wird es aufwendig. Deshalb beschränkt man die Nutzung von Datenflußprinzipien üblicherweise auf die Datenflußsteuerung von Operationswerken in Hochleistungs- und Spezialprozessoren und führt Spezialmaschinen der oberen Leistungsklassen vorzugsweise als Datenstrukturmaschinen aus. Verarbeitungswerke, Speichereinrichtungen und Datenwege werden in Anpassung an den jeweils zu verarbeitenden Datenstrukturen entworfen. Befehle können komplexe Operationen über komplexe Datenstrukturen auslösen, beispielsweise über Datenströme, Vektoren und Matrizen. Die Datenstrukturmaschine ist im Grunde ein massiv erweiterter Universalrechner. Die üblichen Einschränkungen werden fallengelassen. Nicht nur ein Speicher, sondern mehrere, beispielsweise für Operanden, Ergebnisse, Adreßtabellen, Befehle, Deskriptoren und Steuerwörter, nicht nur eine universelle Verarbeitungseinheit (ALU), sondern mehrere Operations- und Adreßrechenwerke usw. Ein typisches Entwurfsziel ist, mindestens die innersten Schleifen der leistungsentscheidenden Algorithmen so mit Hardware zu unterstützen, daß in jedem Maschinenzyklus ein Beitrag zum Ergebnis geliefert wird[4].

Kapitel 5: Universalrechner verbessern
Die Betrachtung grundsätzlicher Alternativen beginnt mit dem prinzipiellen Modell der Informationsverarbeitung, das allen hier in Rede stehenden Maschinen zugrunde liegt, universellen und speziellen. Was begrenzt die Ausführungszeiten der Befehle? Die Dauer der Maschinenzyklen bestimmt die Leistungsgrenzen der Maschine. Wenn man die Einflußgrößen kennt, kann man versuchen, dort anzusetzen, um nach Abwandlungsmöglichkeiten und Alternativen zu suchen.
Es liegt aber auch nahe, das prinzipielle Modell der Informationsverarbeitung grundsätzlich in Frage zu stellen. Womöglich kann man Anwendungsprobleme, die viel Verarbeitungsleistung erfordern, besser auf ganz andere Weise lösen. Die Betrachtungen beginnen zweckmäßigerweise mit der Frage, was wir eigentlich unter Computing verstehen. Alles miteinander zu vermischen, das numerische Rechnen, das Ver- und Entschlüsseln, das Lernen, das Nachbilden der allgemeinen Intelligenz usw. hilft nicht wirklich weiter. Das Schlagwort vom Computing beschreibt eine Vielfalt von Einsatzfällen und Entwicklungszielen, die sich teils beträchtlich voneinander unterscheiden. Wir aber beschränken uns auf die deterministischen, exakt rechnenden programmierbaren Universalmaschinen.

4. Dieses grundsätzliche Entwurfsziel wurde in Band 1 näher begründet.

Die digitale Schaltungstechnik bleibt, die Bits bleiben. Von diesen Vorgaben ausgehend untersuchen wir vielfältige Alternativen, um die Verarbeitungsleistung über die bisher gegebenen Grenzen hinaus zu steigern. Alles wird sozusagen daraufhin abgeklopft, ob etwas verbessert werden könnte, die Befehle, die Programmablaufsteuerung, ja sogar die Speicherorganisation. Es muß nicht bei den üblichen Organisationsformen und Speicherschaltkreisen bleiben. Was die Technologie bereitstellt, wird übernommen, die Speicherzellen, Leseverstärker usw. Damit bauen wir aber andere Speicher. Es handelt sich vor allem darum, die Speicherzellen, das Auswählen der Inhalte und deren Verarbeitung eng miteinander zu verbinden. Wir denken u. a. an Speicher mit eingebauten Verarbeitungsschaltungen, Verarbeitungsressourcen mit eingebauten Speichern, Assoziativspeicher und aktive Zugriffsschaltungen, die direkt mit dem Speicher verbunden sind, um Suchabläufe, Objektzugriffe, das Verschlüsseln, die Datenkompression usw. zu unterstützen. Zu den Entwurfsideen gehört auch, in manchen Speichern auf die herkömmliche Adressierung und Adreßdecodierung zu verzichten.
Im letzten Abschnitt betrachten wir grundsätzliche alternative Wirkprinzipien, wie sie im Lauf der Entwicklungsgeschichte vorgeschlagen und in der Grundlagenforschung untersucht wurden. Womöglich lohnt es sich, solche Ideen erneut zu ventilieren. Die Technologien bieten heute viel mehr Möglichkeiten als seinerzeit. Man kann alles bauen und miteinander kombinieren. Anwendungsprogrammschnittstellen (APIs) auf Grundlage der Ressourcen-Algebra können alles unterstützen. Es ist nur die Frage, was sich lohnt und was nicht.

Anhang 1: Historische Beispiele
Historische Beispiele gehören zur Erfahrungsbasis. Ihr Vorteil liegt in der Überschaubarkeit. Sie wurden seinerzeit vergleichsweise ausführlich beschrieben. Es ist noch möglich, die Prinzipien in kleinen Blockschaltbildern usw. zu veranschaulichen und in wenigen Sätzen zu erläutern. Im Grunde hat sich auch gar nicht viel geändert. Die neueren Typen haben nur von allem mehr und arbeiten mit höheren Taktfrequenzen. Die Schaltungstechnik wird aber zumeist nur kurz beschrieben[5]. Dabei werden die grundlegenden Fachbegriffe als bekannt vorausgesetzt.
Wir betrachten zwei Universalprozessorarchitekturen und eine Datenstrukturmaschine. Das erste Beispiel betrifft die Nutzung des innewohnenden Parallelismus bei Wahrung der Abwärtskompatibilität, das zweite die explizite Nutzung, die in den Befehlen codiert ist und zur Compilierzeit erkannt werden muß. Es sind zwei Architekturen, die von der Fa. Intel entwickelt wurden, IA-32 und IA-64. IA-32 ist die Architektur der historischen Prozessoren der Personalcomputer. IA-64 war seinerzeit als Nachfolge geplant, hat sich aber nicht durchgesetzt. Mit dem dritten Beispiel veranschaulichen wir den Aufbau von Datenstrukturmaschinen anhand einer historischen Familie von Array-Prozessoren der Fa. Floating Point, Inc. Das sind Spezialmaschinen zum Rechnen mit Gleitkommazahlen. Sie führen eigene Programme aus und wirken mit Host-Systemen zusammen.

5. U. a. in einschlägigen Architekturhandbüchern (z. B. [22] und [23]).

Anhang 2: Serielle Verarbeitungsschaltungen
Die Schaltungen betreffen zwei grundsätzliche Architektur- und Entwurfsideen: die serielle Mehrfachverarbeitung von Daten, die sozusagen um 90° gedreht gespeichert werden, und die serielle Verknüpfung von Daten, die ohnehin seriell transportiert werden und mit mehr als einem Bit je Einheitsintervall (Unit Interval, UI) codiert sind.

Anhang 3: Datenflußschaltungen mit Operationswerken
Um rechenintensive Anwendungsprogramme zu unterstützen, liegt es nahe, mehrere Operationswerke im Verbund einzusetzen. Die Frage ist, wie man sie miteinander verbindet und den Verbund steuert. Zu den naheliegenden Ansätzen gehört, sie gemäß den Datenflußgraphen typischer Anwendungsproblemlösungen so zu verbinden, daß Operanden und Zwischenergebnisse von Operationswerk zu Operationswerk weiterfließen können. Die Datenflüsse kann man mit Befehlen steuern, aber auch den Daten Steuerwörter beigeben, die den Datenfluß begleiten. Diese Prinzipien werden an Beispielen von Datenflußstrukturen erläutert, die vorzugsweise als invertierte Binärbäume ausgelegt sind.

Anhang 4: Universelle Datenflußmaschinen
Die universellen Datenflußmaschinen der Theorie sind keine digitalen Analogrechner. Von Verbindungen zwischen Funktionseinheiten oder gar Draht usw. wird abstrahiert. Die Verfügbarkeit der Daten bestimmt die Verarbeitungsreihenfolge. Eine Operation wird dann ausgeführt, wenn alle dafür notwendigen Operanden gültig sind, also mit ihren aktuellen Werten bereitstehen. Diese Arbeitsweise wird mit assoziativen Zugriffen und Assoziativspeichern implementiert. Der assoziative Speicher der Datenflußmaschine entspricht dem Band der Turingmaschine. Wir gehen genauso heran wie (in Band 1) an die Turingmaschine, indem wir versuchen, eine der hypothetischen nahekommende fiktive oder reale Maschine tatsächlich zu bauen.

Die Anhänge 3 und 4 haben vor allem den Zweck, für die Ressourcenvektor- und ReAl-Maschinen etwas zu lernen. Es sind im Grunde Vorarbeiten zu Maschinenentwürfen, die auf der Ressourcen-Algebra beruhen. Wir bleiben aber zunächst bei den herkömmlichen Fachbegriffen. Die Grundlagen – Datenflußgraph und Assoziativspeicher – werden anhand überschaubarer Beispiele erläutert (Feinstrukturanalyse). Operationswerke so miteinander zu verbinden wie es der Datenflußgraph vorgibt, ist eine naheliegende Implementierung des Datenflußprinzips. Die grundsätzliche Metapher ist der digitale Analogrechner, eine Sammlung von Funktionseinheiten, die über ein Steckfeld miteinander verbunden werden. Mit den Verkettungsoperatoren der ReAl-Architektur läßt sich so etwas bauen, zumindest als fiktive Maschine. Wie baut man es aber praktisch, auf Grundlage fest verdrahteter Datenflußwege oder mit programmierbaren Verbindungen der Art Jeder-mit-Jedem? Der Assoziativspeicher ist typisch für akademische = hypothetische Datenflußmaschinen. Assoziativzugriffe können aber auch auf Grundlage üblicher Zugangswege implementiert werden. Daraus könnten sich womöglich Ansätze ergeben, die Verkettung von Ressourcen und objektorientierte Zugriffsprinzipien effektiv zu unterstützen und die unter dem Begriff der Cache-Kohärenz bekannten Probleme auf alternative Weise zu lösen.

Eigentümlichkeiten der Darstellung

Die Darstellung kann nur eine umgangssprachlich-beschreibende sein. Alle Einzelheiten darzustellen wäre viel zu umfangreich. Auch würde man das Grundsätzliche, Gemeinsame nicht erkennen. Daß verstanden wird, was gemeint ist, müssen wir voraussetzen (Komprehensionsaxiom), ebenso das kataleptische Vermögen, aus knappen Funktionsbeschreibungen und Prinzipskizzen die Entwurfsabsichten zu erkennen. Die Ausgestaltung der Einzelheiten muß dem entgegenkommenden Verständnis und der Anwendung einschlägigen Fachwissens überlassen werden. Unsere Fachbegriffe bilden wir, indem wir dem üblichen Sprachgebrauch folgen und, falls erforderlich, den vorgefundenen Ausdrücken eine spezifische Bedeutung unterlegen. Um das Allgemeingültige, Gemeinsame zu beschreiben, brauchen wir neutrale[6] Fachausdrücke. Deshalb führen wir gelegentlich eigene Begriffe ein, die gemäß den Gepflogenheiten der traditionellen deutschen Fachsprache gebildet wurden. Manche Sachverhalte, Zusammenhänge, Entwurfsgedanken, Alternativen usw. lassen sich am besten in kurzen numerierten Abschnitten darstellen, die, ähnlich den Aphorismen, ohne Übergänge aneinandergereiht werden. Hierbei verwenden wir ein hierarchisches Numerierungsprinzip, das von der Dezimalklassifikation und Ludwig Wittgensteins Tractatus angeregt wurde. Das vorliegende Buch wurde ausschließlich mit natürlicher Intelligenz erstellt.

6. Soll heißen, unabhängig von Begriffsbildungen der Hersteller, des Marketing, der sog. Communities usw.