Vorwort

> Sichere Wahrheit werden wir in den Fundamenten nicht gewinnen, ohne alle Möglichkeiten ernsthaft durchdacht zu haben. Ganz ernst denkt eine Möglichkeit aber nur derjenige durch, der an sie glaubt.
>
> Edmund Husserl

> Wir dürfen vor ungewohnten Auffassungen, wenn sie auf sicheren Grundlagen ruhen, nicht zurückschrecken.
>
> 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 haben bisher immer die Technologien die Leistung gebracht. Die eigentlichen Wundertaten werden von den Werkstoffen, Fertigungsverfahren, Compilern und Algorithmen verrichtet ...

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

Im folgenden handelt es sich um einen Versuch, diesen Entwicklungsstand zu überwinden. Es geht sowohl um die Grundlagenforschung als auch um die Anwendungspraxis. Das eine Ziel ist, die Rechnerarchitektur als wohlbegründete Technikwissenschaft aufzubauen, zumindest aber echt wissenschaftliche Begründungen für Architekturentscheidungen zu finden. Das andere ist letzten Endes die leistungsoptimierte Maschine, die beliebige Programme ausführen kann[1]. Abb. 1 veranschaulicht beide Ziele und die dorthin führenden Entwicklungswege.

Eine echte Wissenschaft der Rechnerarchitektur sollte nicht danach fragen, was vorhanden und leicht zu beschaffen ist, sondern erforschen, wie optimale technische Lösungen aussehen könnten. Wir wollen versuchen, diese Aufgabe durch eine entschiedene Abkehr von der traditionellen Prozessorarchitektur zu lösen. Die bewährten Grundlagen der Informationsverarbeitung sollen jedoch beibehalten werden. Das betrifft die elementaren Datenstrukturen und Operationen ebenso wie die herkömmlichen Programmiermodelle. Programmierabsichten und Programmiergepflogenheiten werden als gegeben angenommen. Die Auffassungen vom Programmieren verbleiben im Rahmen des Bekannten und Üblichen[2].

---

1. Und sich womöglich sogar verkaufen läßt ...
2. Mit anderen Worten: Alles, was bisher auf Universalrechnern gelaufen ist, soll nach wie vor laufen.

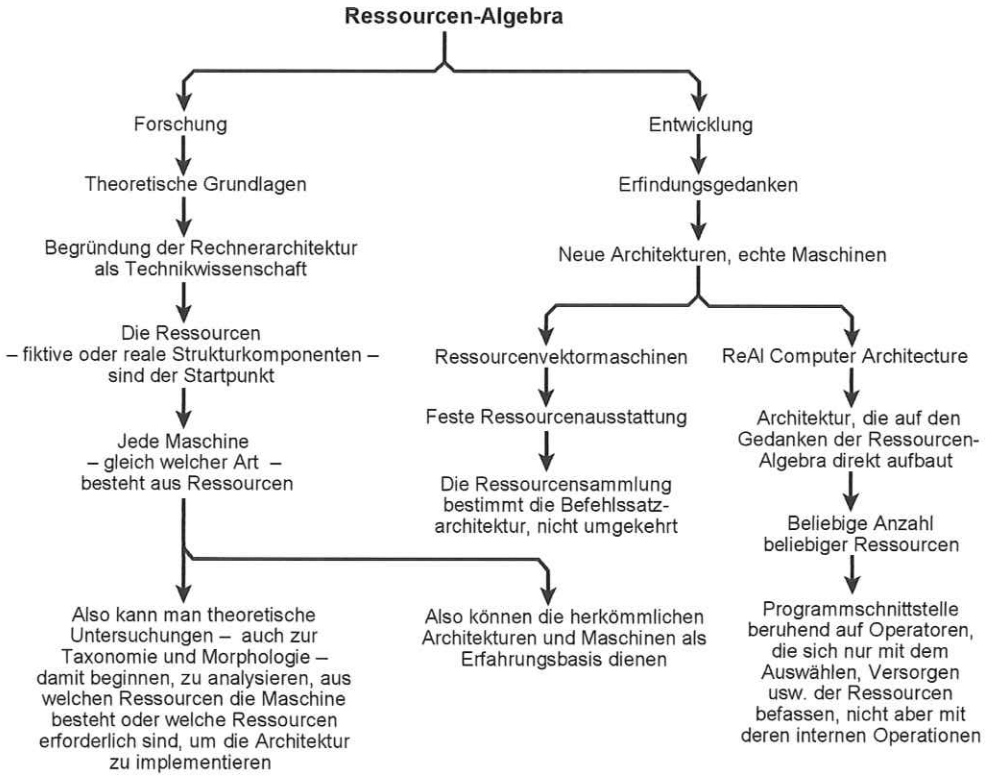

Abb. 1 Eine Theorie, zwei Ziele. Zum einen die Rechnerarchitektur als Technikwissenschaft, die beliebige Architekturen analysiert und beschreibt (sowohl gegebene als auch künftige), zum anderen das Bestreben, neuartige Architekturen und Maschinen zu erfinden und zu entwickeln. Die Begriffe werden wir nach und nach erläutern.

Grundsätzliche Ziele der Architekturentwicklung bestehen darin, das Leistungsvermögen zu erhöhen, die Energieaufnahme zu verringern, das Verhältnis von Leistung und Kosten zu verbessern und weiteren Anforderungen gerecht zu werden, die seitens der Programmierung und der Anwendungspraxis gestellt werden (Sicherheit, Zuverlässigkeit, Bequemlichkeit der Nutzung usw.). Es liegt nahe, die Möglichkeiten auszunutzen, die sich aus den modernen Technologien ergeben. Heute steht im Grunde beliebig viel Hardware zur Verfügung. Bei Schaltungen und Speicherkapazitäten muß man nicht mehr um jeden Preis sparen. Somit kann man auch daran denken, von üblichen Maschinenbefehlen und Prozessorkernen grundsätzlich abzugehen. Wenn man es ernst meint, ist immer eine Maschine zu bauen. Diese Maschine ist im Grunde ein Schaltwerk. Nur das Schaltwerk erbringt die Verarbeitungsleistung. Es sind alles Schaltwerke, gleich welche Architekturgedanken ihnen zugrunde liegen und mit welchen Schlagworten oder Marketingbegriffen sie bezeichnet werden. Somit liegt es nahe, die Maschine von Grund auf als Schaltwerk so zu entwerfen, wie es jeweils zweckmäßig ist.

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.
Zur Implementierung von Programmierabsichten sind Schaltwerke zu schaffen, die bei hinreichend hohem Leistungsvermögen zu annehmbaren Kosten angeboten werden können, ganz gleich, wie sie heißen und ob sie zu irgend etwas Gewohntem kompatibel sind oder nicht. Wir wollen aber auch über den herkömmlichen Begriff des Schaltwerks hinausgehen, uns nicht auf Gebilde aus Gattern, Flipflops, Registen, Speichermatrizen usw. beschränken, sondern auch mit hypothetischen, fiktiven usw. Maschinen arbeiten und womöglich andere technische Prinziplösungen außerhalb der digitalen Schaltungstechnik berücksichtigen.
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. Dabei nehmen wir die Grundlagen der Berechenbarkeit und die Anforderungen der Anwendungspraxis als gegeben an.
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.
Eine Architekturdefinition umfaßt eine Menge von Ressourcen und eine Menge von Datenstrukturen. Ressourcen führen bestimmte Operationen über Daten aus, die bestimmten Datentypen entsprechen. Wird dieser Sachverhalt mit den Ausdrucksmitteln der Mathematik formal beschrieben, so ergeben sich algebraische Strukturen, universelle Algebren. Unser Ansatz der Architekturentwicklung beginnt von Grund auf bei den Ressourcen, die wir als algebraische Strukturen betrachten. Deshalb wird die hier in Rede stehenden Architektur mit dem Begriff der Ressourcen-Algebra bezeichnet. Im Englischen kann sie gar nicht anders heißen als ReAl Computer Architecture[3]. Unsere Darlegungen umfassen drei Bände:

- Band 1: Rechnerarchitektur als Technikwissenschaft.
- Band 2: Tiefenstrukturen.
- Band 3: ReAl Computer Architecture.

Der vorliegende Band 1 dient der Einführung in die Grundlagen. Wie soll die Rechnerarchitektur als echte Technikwissenschaft aufgebaut werden? Hierzu sind die ganz grundsätzlichen, ohne Zweifel offenkundigen (apodiktisch evidenten) Sachverhalte aufzusuchen und formal dar-

3. Das Wortspiel ergibt sich von selbst und wird gern mitgenommen (pun intended).

zustellen, ähnlich wie bei der Axiomatisierung in der Mathematik. Auch geben wir einen ersten Überblick über die Ressourcen-Algebra und die ReAl Computer Architecture. Tiefere Einzelheiten werden in den weiteren Bänden nach und nach erörtert werden. Band 2 gibt einen Überblick über den Stand der Technik sowie über alternative Ansätze und Entwurfsgedanken. Es geht darum, das Grundsätzliche der vielfältigen Architekturlösungen aufzufinden und daraus Entwicklungsziele abzuleiten. In diesem Sinne sprechen wir von Tiefenstrukturen und vergegenständlichten Abstraktionen. Mit Tiefenstrukturen meinen wir das Allgemeingültige, allen jeweils entsprechenden Architekturprinzipien Gemeinsame. Denken wir beispielsweise an das Addieren von Binärzahlen und an das bedingte Verzweigen. In der einen Maschine sind es 12 Bits, in einer anderen 16, in einer weiteren 32. Die Entscheidung, ob ein Programmstück ausgeführt wird oder nicht, mag IF ... THEN heißen oder BC = Branch on Condition usw. Die Operanden werden aber immer auf gleiche Weise miteinander verknüpft, der auszuführende Programmabschnitt auf gleiche Weise ausgewählt. Solche allgemeinen Prinzipien brauchen wir, um Ansätze zur Gestaltung unserer Ressourcen zu gewinnen. Alles, was nützlich erscheint oder Erfolg verspricht, kann als Ressource implementiert und über die ReAl-API angesprochen werden. Vergegenständlichung heißt, Ressourcen zu bauen, die bestimmten funktionellen Spezifikationen entsprechen. Mit Abstraktion meinen wir das Grundsätzliche, Allgemeingültige der jeweiligen Funktion. Die Vergegenständlichung der Abstraktion ergibt eine Ressource mit einer entsprechend allgemeinen Programmschnittstelle (API). In Band 3 beschreiben wir tiefere Einzelheiten von Architekturlösungen, die auf der Ressourcen-Algebra beruhen.

Die Darstellung setzt eine Erfahrungsbasis voraus, die dem Stand der Technik entspricht. Das betrifft die üblichen Rechnerarchitekturen und die Wirkprinzipien (Principles of Operation) der typischen Prozessoren. Die Fachkenntnisse können aus dem Studium einschlägiger Standardwerke und Architekturhandbücher gewonnen, aber auch aus der Programmier- und Entwurfspraxis abgeleitet werden. Das begründende Denken und das Entwickeln von Anfang an erfordert beträchtliche Anstrengungen. Es liegt nahe, solche Bemühungen zu unterlassen, weil es schon alles gibt, weil alles so großartig ist und man es einfach kaufen oder noch einfacher herunterladen kann. Daß solchen Überlegungen hier kein Raum gegeben wird, versteht sich von selbst. Abb. 2 betrifft eine typische Einwendung gegen diese Art des Vorgehens.

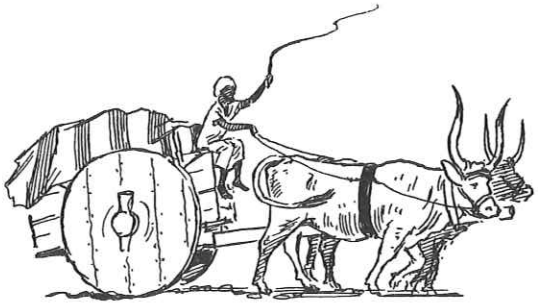

Abb. 2 "Man soll das Rad nicht neu erfinden." Schön und gut. Wenn sich aber ALLE daran gehalten hätten – wie würden die Räder wohl heute aussehen? Vielleicht so ...

Der Inhalt im Überblick

Der vorliegende Band hat propädeutischen Charakter. Die Ansätze zur Begründung der Rechnerarchitektur als Technikwissenschaft können nur skizziert werden. Das betrifft vor allem die Formalbeschreibung und die Nutzung der Methoden und des Werkzeugkastens der Mathematik. Architektur- und Schaltungslösungen auf Grundlage der Ressourcen-Algebra werden soweit beschrieben, wie es erforderlich ist, um die Wirkprinzipien darzulegen und zu zeigen, daß sich auf diese Weise tatsächlich universelle Maschinen bauen lassen. Um die Architektur- und Entwurfsgedanken abzusichern, werden sowohl der Universalrechner als auch die ReAl-Architektur auf mehreren Wegen hergeleitet. Abschließend werden Leistungsmaße und Bewertungskriterien vorgestellt.

Kapitel 1: Einführung

Wir beginnen mit dem Gegenstand der Rechnerarchitektur und dem verallgemeinerten Architekturbegriff. Jede wohldefinierte Schnittstelle ist eine Architektur. Erfinden, entwerfen und formal beschreiben kann man viel. Manches könnte man sogar bauen. Wie aber steht es um die Technikwissenschaft, auf der all diese Gedanken beruhen, wie beispielsweise die Flugzeugkonstruktion auf der Aerodynamik und der Angewandten Mechanik? Seit den Anfangszeiten der Computertechnik ist sie ein Desiderat geblieben. Die Rechnerarchitektur ist nach wie vor eine Erfahrungswissenschaft, obwohl immer wieder Ansätze vorgeschlagen wurden, um diesen Zustand zu überwinden. Auf solchen Gedanken wollen wir aufbauen. Was wir anstreben, ist eine echt wissenschaftliche Begründung von Architekturentscheidungen, im Gegensatz zu Gefühlsentscheidungen und Entscheidungen, die auf Erfahrungen, Meßergebnissen und Vorgaben des Marketing beruhen. Unsere Vermutung ist, daß sich das Problem lösen läßt, indem wir sozusagen am anderen Ende beginnen, mit den Schaltungen bzw. – allgemein gesprochen – mit den Ressourcen, und die technische Erfahrungsbasis mit der Theorie zusammenführen. Dabei wollen wir in Analogie zu den wohlbegründeten Technikwissenschaften und zur Mathematik vorgehen, aber auch in Analogie zu induktiv arbeitenden Naturwissenschaften. Der Rest des Kapitels betrifft die Erläuterung von Grundbegriffen.

Kapitel 2: Rechnerarchitektur als Ressourcen-Algebra

Sucht man nach Ansätzen, die Rechnerarchitektur als Technikwissenschaft zu begründen, wird man u. a. Gedanken zur algebraischen Modellierung vorfinden. Sucht man nach einer Bezeichnung für Mittel, welcher Art auch immer, die funktionelle Absichten implementieren, ist der Begriff der Ressource sicherlich angemessen. Daraus haben sich die erste Architekturgedanken ergeben: Maschinen aus Ressourcen aufzubauen und die Ressourcen-Algebra als formales Beschreibungsmittel. Die Befehle richten sich nach der Ressourcenauswahl, nicht umgekehrt. Das ergibt nach wie vor herkömmliche Universalrechner, die aber nicht so recht zu den Marketing-Schlagworten (RISC, CISC, VLIW usw.) passen wollen. Wir bezeichnen sie als Ressourcenvektormaschinen. Die Befehlsformate ähneln jenen der herkömmlichen Mikrobefehle, aber ohne deren Einschränkungen. Sie sind jedoch nicht maschinenunabhängig; es wäre also keine richtige Architektur. Deshalb brauchen wir eine abstrakte Programmschnittstelle. Diese Pro-

grammschnittstelle direkt auf der Ressourcen-Algebra aufzubauen und mit einer unbegrenzten Anzahl beliebiger Ressourcen zu arbeiten, hat sich als spontaner Erfindungsgedanke ergeben. Das Maschinenprogramm ist eine Fertigungsanweisung, die Hardware eine Werkhalle mit Werkzeugmaschinen. Maschinenprogramme werden so generiert, als ob beliebig viele Werkzeugmaschinen = Ressourcen zur Verfügung stünden. Optimieren und sparen können wir später. Dieser Gedankenkreis wird im Überblick dargelegt. Begründungen und Einzelheiten folgen nach. Hier zeigen wir zunächst, daß man die Wirkprinzipien der v. Neumann-Architektur mit strukturell und aufwandsseitig plausiblen Ressourcen darstellen kann, einschließlich der Sprungvorhersage und der Sprungzielpufferung. In den weiteren Bänden werden wir u. a. typische Verfahren der Speicheradressierung und die in den üblichen Programmiersprachen vorgesehenen Kontrollstrukturen betrachten und zeigen, daß sich hinreichend kompakte, praxisbrauchbare Maschinencodes angeben lassen.

Kapitel 3: Rechnerarchitektur und Programmierung
Architekturen vorschlagen und Maschinen erfinden genügt nicht. Man muß sie auch bauen und programmieren können. Seit es Computer gibt, unterscheidet man zwischen Software und Hardware. Software ist etwas Ideelles, das beliebig geändert und erweitert werden kann, Hardware ist physisch vorhanden und starr. Die grundsätzliche Frage ist, wie man Anwendungsprobleme löst – indem man eine passende Maschine erfindet oder indem man Programme schreibt und sie auf fertigen programmierbaren Maschinen laufen läßt. Im Laufe der Entwicklungsgeschichte wurden Softwarelösungen mehr und mehr bevorzugt. Möglichst alles soll programmierbar sein. Mittlerweise sind aber auch die Schaltungslösungen programmierbar (FPGAs), und sie lassen sich durch Programmieren entwerfen (Verhaltensbeschreibung). Manches Problem läßt sich einfacher lösen, indem man eine Schaltung baut anstatt Programme zu schreiben, die im Grunde nichts anderes tun als einen begrenzten Ressourcenvorrat trickreich auszunutzen. Programmieren kann man auf verschiedene Weise und auf verschiedenen Ebenen. Architekturen sind Programmschnittstellen, und zwar nicht nur zur Hardware, sondern auch zu anderen Programmen und zu fiktiven Maschinen, die es als Hardware gar nicht gibt. Wir betrachten die Unterschiede zwischen Hard- und Software, die Ebenen der Programmierung, die grundsätzlichen Maschinentypen, die Maschinenprogrammfertigung und die Architekturentwicklung. Unser alternativer Ansatz besteht darin, nicht mit dem Befehlssatz oder dem Programmiermodell zu beginnen, sondern mit den Schaltwerken, in unserer Redeweise mit den Ressourcen der Hardware.

Kapitel 4: Rechnerarchitektur und Mathematik
Es liegt nahe, sich soweit wie möglich an die Mathematik anzulehnen, Verfahren und Gepflogenheiten zu übernehmen oder wenigstens eine weitgehende Anähnlichung anzustreben. Die Rechnerarchitektur ist kein Teilgebiet der Mathematik. Vielmehr ist die Mathematik eine Grundlagenwissenschaft, aus der man Begründungen, Methoden und Verfahren übernehmen kann. Im Grunde ist sie eine Art Werkzeugkasten. In diesem Sinne diskutieren wir Vorschläge, die Formalisierung und theoretische Begründung der Rechnerarchitektur auf die Axiomatisierung, die algebraische Modellierung und die lineare Optimierung zu stützen. Wir zeigen, daß es

möglich ist, wenngleich mit Abweichungen und Einschränkungen, die durch die Spezifik des Gegenstands bedingt sind. Im Bereich der formalen Verifizierung (Formal Verification) gehört die Axiomatisierung zum Stand der Technik. Darüber hinaus sind wir bestrebt, ganz grundsätzliche Axiome aufzusetzen und so die Rechnerarchitektur ähnlich zu begründen wie mathematische Theorien. Die algebraische Modellierung hat sich als geeignet erwiesen, Architekturgedanken zu begründen und formal zu beschreiben. Dabei wird die technische Erfahrungsbasis einbezogen und gelegentlich als Startpunkt angesetzt. Entwurfsziele sind Ressourcen als fiktive oder reale Schaltungslösungen sowie die zugehörigen Anwendungsprogrammschnittstellen (APIs). Begriffserklärungen enden mit dem Verweisen auf Bestehendes. Die Bedeutung eines Ausdrucks ist seine Implementierung (operationale Semantikdefinition). Die lineare Optimierung ist in den Grenzen und mit den Einschränkungen, die wir hier skizzieren, für manche Optimierungsaufgaben anwendbar. Es ist nur die Frage, ob sich der Aufwand lohnt. In der Entwurfspraxis jenseits der theoretischen Grundlagenforschung dürfte das Verfahren nur selten praktikabel sein. Alternative Ansätze, sich an näherungsweise optimale Entwürfe heranzutasten, werden wir in Kapitel 7 vorstellen.

Kapitel 5: Grundlagen der Formalbeschreibung

Die Ressourcen-Algebra und die darauf beruhenden Architekturen und Maschinen kann man auch auf herkömmliche Weise beschreiben. Den in Abb.1 rechts dargestellten Entwicklungsweg könnte man so dokumentieren – in Patentschriften, Zeichnungssätzen, Handbüchern usw. – wie bisher in der Architekturentwicklung und im Schaltungsentwurf üblich. Das genügt aber nicht, um die Rechnerarchitektur als Technikwissenschaft zu begründen. Hier soll alles exakt und vollständig formal beschrieben werden, und zwar im lückenlosen Verbund und so, daß man es auch viel (Jahrzehnte und mehr) später noch verstehen kann. Im lückenlosen Verbund heißt, daß auch Übergänge formal beschrieben werden, vor allem die zwischen Funktion und Struktur. Typischerweise sind Strukturen zu entwerfen, um funktionelle Absichten zu implementieren. Erfahrene Praktiker wissen, daß das manchmal ohne wirkliche Schwierigkeiten gelingen kann, sozusagen routinemäßig, daß es manchmal aber auch notwendig sein kann, erst einmal herauszufinden, was eigentlich beabsichtigt ist und getan werden soll. Unser Hauptpunkt ist die Begründung. Wozu ist es gut? Dieses Schaltnetz dient zum Multiplizieren von Gleitkommazahlen, dieser Sequencer erledigt den Punkt 123 des Interfacestandards XYZ, dieses Register dient der Maschinenfehlerbehandlung usw. All das wollen wir lückenlos formal erfassen. Anschaulich können wir uns das so vorstellen, daß jedem Funktionselement im Schaltplan, jeder Anweisung in einer Programmier- oder Beschreibungssprache usw. ein unverlierbares Etikett beigegeben wird.

Eine ernsthafte durchgehende Formalbeschreibung praxisbrauchbarer Rechnerarchitekturen scheint ohne Nutzung maschineller Mittel nicht durchführbar. Zumindest wird sie sehr umfangreich werden. Somit können wir nur grundsätzliche Gedanken skizzieren. Unsere Betrachtungen bewegen sich auf der Ebene der Metasprache. Was sollen die Beschreibungen ausdrücken, aus welchen Komponenten bestehen sie? Das erfassen wir durch Aufzählung entsprechender Mengen. Es ist eine sehr weitgehende Abstraktion. Ein Mengensymbol kann einen Schaltplan

bedeuten, eine Beschreibung herkömmlicher Art, eine Programmdatei, eine Satz von Entwurfsdateien (Design Master Files) usw. Wir erläutern, wie Funktionen und Strukturen, Begründungen, Algorithmen, Informationsstukturen, Ressourcen, Architekturen und Schaltungen formal beschrieben werden.

Kapitel 6: Zur Begründung des Universalrechners
Die Architekturentwicklung soll im Sinne einer echten Technikwissenschaft betrieben werden. Somit ist sie in der Theorie so genau wie möglich zu begründen. Unser Begründungsziel ist zunächst der praxisbrauchbare Universalrechner. Wir untersuchen drei Herleitungen: aus der Turingmaschine, aus dem Funktionszuordner und aus elementaren Universalmaschinen der mathematischen Grundlagenforschung, den sog. GOTO-Maschinen. Das nächste Begründungsziel ist die ReAl-Architektur. Sie kann aus der Turingmaschine, der GOTO-Registermaschine und dem herkömmlichen Universalrechner hergeleitet werden. Eine Gegenprobe besteht darin, von der Ressourcen-Algebra aus in die umgekehrte Richtung zu schauen. Mit unseren Ressourcen kann man nahezu alles bauen, auch Turingmaschinen und herkömmliche Universalrechner.

Kapitel 7: Grundlagen der Bewertung
Um leistungsfähige, praxisbrauchbare und wirtschaftlich sinnfällige[4] Architekturen und Maschinen zu entwickeln, sind verschiedene Lösungsansätze zu erarbeiten, zu bewerten und untereinander zu vergleichen. Teillösungen, die sich als zweckmäßig erwiesen haben, sind zu Systemlösungen zusammenzufassen. Dafür sind Bewertungsmaße erforderlich. Schaltungen sind nach ihrer Leistungsfähigkeit zu bewerten, Algorithmen nach der Eignung zur Vergegenständlichung und Aufwendungen nach ihrer Nützlichkeit.
Am Anfang steht ein kurzer Überblick über übliche Leistungskennwerte von Prozessoren. Diese werden sinngemäß auf die Ressourcen übertragen. Das genügt aber nicht. Wie bewerten wir Maschinen, die gar keine Befehle haben? Wir fragen, wozu die Befehle und Daten eigentlich gut sind. Sind sie an der Lösung der Anwendungsaufgaben beteiligt oder dienen sie nur zu Verwaltungszwecken oder gar nur dazu, Unzulänglichkeiten und Sparlösungen der Architektur oder des Laufzeitsystems zu umgehen? Unser Ansatz: die Befehle verbrauchen zwar Maschinenzyklen, erbringen aber keine Leistung. Was zählt, sind nur die sog. Nutzbits, die, indem sie durch die Maschine fließen, zur Lösung der Anwendungsaufgaben beitragen.
Nachfolgend untersuchen wir, welches Leistungsvermögen überhaupt erreicht werden kann. Die unmittelbare Zuordnung, das bloße Abrufen vorberechneter Ergebnisse, ergibt die höchste Leistung überhaupt (Axiom der absolut höchsten Verarbeitungsleistung). Unter welchen grundsätzlichen Bedingungen ist sie erreichbar? Wenn man dieses Wirkprinzip nicht implementieren kann, ergibt sich das maximale Leistungsvermögen dann, wenn die Maschine in jedem Maschinenzyklus einen Beitrag zum Endergebnis liefert, der der Verarbeitungsbreite entspricht (Axiom der maximalenVerarbeitungsleistung). Auch dieser Ansatz wird näher untersucht. Es zeigt sich, daß das potentielle Leistungsvermögen nicht nur von der Auslegung der Maschine

4. Wir beziehen uns auf die Kosten über alles, die Total Cost of Ownership (TCO).

abhängt, sondern auch von den auszuführenden Algorithmen. Das erfassen wir mit dem Maß der Implementierungseffizienz. Mit weiteren Maßen bewerten wir Entwurfslösungen und Aufwendungen. Wie effektiv werden die Aufwendungen eingesetzt, wie verhalten sich Aufwand und Leistung, wenn wir erweitern oder sparen? Unsere Ansätze der quantitativen Bewertung sind vor allem dazu gedacht, sich anhand von Probeentwürfen usw. an optimale Auslegungen heranzutasten, wenn eine lineare Optimierung nicht möglich ist.

Anhang 1: Anmerkungen und Zitate
Eine Sammlung zu den Themen Rechnerarchitektur als Technikwissenschaft, Mathematik und Praxis, an die Nachwelt überliefern und Pasigraphie.

Anhang 2: Einzelheiten fiktiver Maschinen
Der Anhang enthält Einzelheiten zu den fiktiven Maschinen des Kapitels 6. Auch beweisen wir durch effektive Konstruktion[5], daß die aus der Turingmaschine umgebaute Einadreß-Universalmaschine und die Eindreß-Akkumulatormaschinen Turing-vollständig sind.

Anhang 3: Funktionszuordner
Der Funktionszuordner – ein Speicher, der alle Ergebnisse schon vorab berechnet enthält –, ist unser Paradigma für Schaltungslösungen, die ihre Ergebnisse sozusagen auf eine Schlag liefern. Ein solches Leistungsvermögen kann grundsätzlich nicht weiter überboten werden (Axiom der absolut höchsten Verarbeitungsleistung). Hier zeigen wir zwei Praxisbeispiele.

Anhang 4: Befehle, Register, Ressourcen
Es geht um die Befehle herkömmlicher Architekturen. Zu unseren Absichten gehört es, Befehlsbeschreibungen auszuwerten und in die Erfahrungsbasis einfließen zu lassen. Der Anhang enthält Beispiele von Befehlsbeschreibungen und zur Einschränkung der Registernutzung sowie erste Überlegungen, wie man eine solche Erfahrungsbasis nutzen kann, um Ressourcenvektor- und ReAl-Maschinen zu gestalten.

Eigentümlichkeiten der Darstellung
Die Darstellung kann nur eine umgangssprachlich-beschreibende sein, mit gelegentlichem Rückgriff auf die Beschreibungsmittel der Mathematik. Konkrete Architektur- und Schaltungsbeschreibungen wären viel zu umfangreich. Auch würde man das Grundsätzliche, Gemeinsame nicht erkennen. Deshalb müssen wir abstrahieren und vielen Fragen auf der Ebene der Metasprache erörtern, beispielsweise darlegen, was bestimmte Beschreibungen enthalten und wie die Beschreibungsmittel beschaffen sein müssen. Dafür nutzen wir die Symbole und Notation der Mathematik und borgen uns weitere Darstellungs- und Gliederungsmittel. Daß verstanden wird, was gemeint ist, müssen wir voraussetzen (Komprehensionsaxiom). Die Formalbeschreibung kann hier nur skizzenhaft angedeutet werden. Eine durchgehende formale Darstellung ist am Anfang noch nicht möglich, weil die Probleme erst einmal erfaßt und erläutert werden müssen.

5. Mit anderen Worten, indem wir entsprechende Programmstücke angeben. Mehr zu diesen Maschinen in [10].

Zunächst ist die Erfahrungsbasis zu gewinnen und aufzuarbeiten. Dann erst ist es sinnvoll, die formale Notation zu spezifizieren. Eine den Aphorismen angeähnliche Darstellung – in kurzen Abschnitten ohne Übergänge – hat sich als gelegentlich zweckmäßig erwiesen. Hierbei verwenden wir ein hierarchisches Numerierungsprinzip, das vom Prinzip der Dezimalklassifikation und Ludwig Wittgensteins Tractatus angeregt wurde. Unsere Fachbegriffe bilden wir, indem wir dem üblichen Sprachgebrauch folgen und, falls erforderlich, vorgefundenen Ausdrücken eine spezifische Bedeutung unterlegen. Das vorliegende Buch wurde ausschließlich mit natürlicher Intelligenz erstellt.