Wolfgang Matthes

Minimale Maschinen

Ein Beitrag zu den Grundlagen der Rechnerarchitektur

Logos Verlag Berlin

λογος

Vorwort

In den Grundlagen kann man nicht genau genug sein.
Wir begreifen nur, was wir selbst machen können.

Immanuel Kant

... der Vater aber liebt,
Der über allen waltet,
Am meisten, daß gepfleget werde
Der feste Buchstab, und Bestehendes gut
Gedeutet.

Friedrich Hölderlin

Wenn sich ein Gebiet der Wissenschaft oder Technik über längere Zeit entwickelt hat, sollte man gelegentlich zu den Anfängen zurückkehren. Das wichtigste Motiv zu der vorliegenden Arbeit war, Grundlagen aufzuarbeiten, um Quellen zu erschließen und Anregungen zu gewinnen. Daß man auf dem Gebiet der Rechnerarchitektur nicht aufhört, nach Verbesserungen und neuartigen Lösungen zu suchen, versteht sich von selbst. Die Erörterungen sollen aber auch dazu beitragen, die Rechnerarchitektur als Technikwissenschaft so exakt wie möglich zu begründen.
Mit einfachen Mitteln auszukommen ist immer von Interesse. Was aber als einfach anzusehen ist, ergibt sich aus der jeweiligen Absicht und – wenn man tatsächlich etwas bauen will – aus dem jeweils aktuellen Stand der Technik.
In der Literatur und in der Lehre werden die Grundlagen zumeist nur beiläufig und pauschal behandelt. Gedanken, Lösungsansätze und Einzelheiten, die seinerzeit im Fokus gestanden haben, sind in Vergessenheit geraten.
Die vorliegende Arbeit befaßt sich nicht mit der Biographie und Würdigung der maßgeblichen Persönlichkeiten und auch nicht mit der theoretischen Informatik und der mathematischen Grundlagenforschung. Dazu liegt eine umfangreiche Literatur vor[1]. Vielmehr geht es um eine Rückbesinnung auf die ursprünglichen Entwicklungs- und Erfindungsgedanken, um deren Auswertung und Aufarbeitung mit der Absicht, letzten Endes wirkliche Maschinen zu bauen[2]. Was waren die Absichten, was die entscheidenden Ideen, welche Voraussetzungen und Möglichkeiten waren gegeben? Manches, was seinerzeit ein unüberwindliches Problem war, hat sich mittlerweile erledigt. Die technischen Beschränkungen der Anfangszeit gibt es nicht mehr. Was könnte entstehen, würde man die ursprünglichen Erfindungsgedanken aufgreifen und mit dem heutigen Erkenntnisstand und technischen Möglichkeiten nochmals von Grund auf durcharbeiten?

1. Zur Einführung sei auf [1] bis [4] verwiesen. Grundsätzliches zum erneuten Aufarbeiten von Ideen aus der Pionierzeit in [5].
2. Die anwendungsbrauchbare (fiktive oder reale) Maschine ist das Endziel. Das Studium der Grundlagen und der Entwicklungsgeschichte dient der Begründung und Ideenfindung.

Der Inhalt im Überblick

Kapitel 1: Einführung
Für die nachfolgenden Betrachtungen brauchen wir genaue Begriffsbestimmungen. Den Grundbegriffen begegnet man immer wieder, auch in eher populären Texten und im Internet. Die Grundlagen werden aber oft nur sehr oberflächlich dargestellt. Man weiß nicht immer, was man sich darunter vorzustellen hat, was man bauen oder ausprogrammieren müßte. Wir verfolgen daher den Ansatz, grundsätzlich so zu tun, als ob wir die Maschinen, über die wir reden, tatsächlich bauen wollen. Man muß schon recht tief in die Einzelheiten gehen, sonst kann man die Problemstellungen, Zusammenhänge und Abhängigkeiten nicht richtig erkennen.
Es gibt prinzipielle Unterschiede zwischen der Grundlagenforschung auf dem Gebiet der Berechenbarkeit und der Entwicklung von Maschinen, mit denen man Probleme der Anwendungspraxis lösen kann. Um dies zu erläutern, müssen wir bereits hier mit der Turingmaschine beginnen. Es ist eine sehr überblicksmäßige Darstellung. Wir brauchen sie aber, um die Begriffe der Berechenbarkeit, der Church-Turing-These und der Turing-Vollständigkeit zu erklären. Einzelheiten zur Turingmaschine kommen in den Kapiteln 3 und 4.

Kapitel 2: Minimalprinzipen der Rechentechnik
Wie kann man minimale Maschinen bauen, die Turing-vollständig sind? Immerhin sollen sie – als fiktive Maschinen – wirklich funktionieren, zumindest in Gedanken, wenn nicht sogar in der Emulation. Mit Turings unendlich langem Band wird es offensichtlich nichts. Auch die einfachste Plattform muß um einen adressierbaren Speicher herumgebaut werden. Wir befassen uns zunächst mit Minimalmaschinen der Theorie. Wenn man nicht mit der Turingmaschine, sondern mit dem algorithmischen Schema von Emil Post beginnt, ist man in wenigen überschaubaren Schritten bei den speicherprogrammierbaren Maschinen. Die ganz elementaren hypothetischen Maschinen der mathematischen Grundlagenforschung können nicht rechnen. Also erweitern wir sie mit Rechenschaltungen. Dabei liegt es nahe, so sparsam wie möglich zu bauen und mit wenigen Befehlen auszukommen – womöglich mit einem einzigen. Auch sollte man die Suche nach grundsätzlichen Alternativen nicht aufgeben. Manche Ideen gibt es seit den Anfangstagen der Entwicklungsgeschichte. Man sollte sie von Zeit zu Zeit erneut ventilieren. Vielleicht kommt man dabei auf verblüffende, überraschende Lösungen. Der letzte Abschnitt betrifft Beispiele minimaler Maschinen, die in der ersten Zeit der Entwicklungsgeschichte tatsächlich gebaut wurden.

Kapitel 3: Turingmaschinen
Eine Einführung in Aufbau und Wirkungsweise, die sich an der Originalarbeit Alan Turings orientiert. Es ist eine Art Nacherzählung mit eigenen Worten und mit Fachbegriffen, die heutzutage üblich und allgemein bekannt sind. Um die Grundgedanken zu verstehen und im Hinblick auf unser Ziel, die theoretische Begründung der Rechnerarchitektur, Erkenntnisse zu gewinnen, tun wir so, als wollten wir Alan Turings Beschreibung als Entwurfsvorgabe nehmen und solche Maschinen tatsächlich bauen. Aus den hypothetischen Maschinen sollen fiktive werden.

Kapitel 4: Zur Begründung des Universalrechners
Der praxisbrauchbare Universalrechner soll von der Theorie her systematisch begründet werden. Wir untersuchen zwei Herleitungen: aus der Turingmaschine und aus dem Funktionszuordner. Wie wird die Turing-Vollständigkeit nachgewiesen? Der Speicher ist nicht das Problem; eine Näherungslösung wird man immer finden. Folglich kommt es auf den Steuerautomaten an. Eine Maschine ist dann Turing-vollständig wenn sie den Steuerautomaten einer universellen Turingmaschine emulieren und den zugehörigen Speicher ansteuern kann.
Die Turingmaschine ist überhaupt nicht effektiv. Sie muß es auch nicht sein. Um unsere Begründung zu gewinnen, wollen wir sie aber so abwandeln, daß sie effektiv wird. Zunächst wird der Steuerautomat als fiktive Maschine entworfen, und das Band wird durch einen adressierbaren Speicher ersetzt. In mehreren Schritten wird die Maschine zur Einadreßmaschine weitergebildet. Wie rechnet man mit der Turingmaschine, obwohl sie eigentlich nicht rechnen kann? Es geht nur über die Zustände. Die elementaren numerischen Symbole müssen Bits sein; im Grunde kann man nur bitseriell rechnen. Hat das einzelne Symbol mehr als nur zwei Werte, braucht man bald viel zuviele Zustände (kombinatorische Explosion). Die naheliegende Verbesserung: die Maschine wird mit Rechenwerken ausgerüstet. Anstelle der Skeleton Tables zum Rechnen tritt das Laden der Symbole ins Rechenwerk, das Auslösen der Operation und das Zurückschreiben der Ergebnisse. Schließlich wird die Maschine zum Einadreß-Universalrechner weiterentwickelt.
Der alternative Entwicklungsweg beginnt mit der hypothetischen Vorstellung eines Funktionszuordners, der alle überhaupt möglichen Ergebnisse schon vorberechnet enthält. Wenn man sich ein unendlich langes Band vorstellen kann, darf man sich auch das vorstellen. Offensichtlich kann man so etwas aber nur für überschaubare Aufgaben der Informationswandlung tatsächlich bauen (gespeicherte Funktionstabellen usw.). Aber auch ein Schaltnetz, das addieren kann, ist ein Funktionszuordner. Einen großen Funktionszuordner, der alles auf einmal erledigt, kann man nicht bauen; es gibt vielfältige Einschränkungen. Diese werden einzeln betrachtet und mit entsprechenden Schaltungslösungen überwunden. So wird die Aufgabe gelöst, die Ergebnisbildung darauf zurückzuführen, daß vergleichsweise einfache Funktionen nacheinander durchlaufen werden. Der Übergang zur Universalmaschine ist dann ebenso einleuchtend wie im Falle der Turingmaschine. Am Ende ergibt sich praktisch die gleiche Struktur.

Kapitel 5: Einfachmaschinen
Hier geht es um praxisbrauchbare Universalrechner, die sich mit geringen Schaltungsaufwendungen implementieren lassen. Die Betonung liegt auf der Brauchbarkeit. Sparen ja, die Maschinen soll aber auch etwas leisten. Wir stellen typische Wirkprinzipien (Principles of Operation) und technische Lösungen vor, die im Laufe der Entwicklungsgeschichte gefunden wurden und sich in der Praxis bewährt haben.

Kapitel 6: Architekturbeispiele
Dieses Kapitel dient vor allem dazu, das vorherige Kapitel mit Beispielen zu illustrieren und für das nächste Kapitel Anregungen zu liefern. Wir betrachten typische Beispiele aus der Entwick-

lungsgeschichte, beginnend mit den ersten Arbeiten am Institute for Advanced Study (IAS) in Princeton. Hiermit wurde der grundsätzliche Typus der v. Neumann-Maschine geschaffen. Die TX0 war eine Versuchsmaschine mit wenigen Registern, aber vergleichsweise komplexen Befehlswirkungen. Die CDC 160 und die PDP-8 gehörten zu den ersten der sog. Minicomputer, gekennzeichnet durch ein kurze Wortlänge (12 Bits) und wenige Register. Die Data General Nova war eine 16-Bit-Architektur mit vier Akkumulatorregistern, die wie Universalregister verwendet wurden. Die Operationsbefehle haben nur Akkumulatoren ausgewählt, aber keine Speicherinhalte adressiert (Load-Store-Prinzip). Die CDC 1700 gehört zu den letzten der kleinen Einadreßmaschinen in der Tradition von Princeton (dann kamen die Maschinen mit Universalregistern, zunächst mit komplexen Befehle (CISC), um die Register gut auszunutzen, dann mit vergleichsweise einfachen Befehlen (RISC), um die Taktyklen kurz zu halten). Die Befehle sind kurz (16 Bits), die Befehlswirkungen aber schon ziemlich komplex. Weitere Anregungen, einfache, aber effektive Maschinen zu entwerfen, ergeben sich aus der Technik der Mikroprogrammsteuerwerke. Damit kann man nicht nur die Ausführung der Maschinenbefehle steuern, sondern auch höhere Programmiersprachen interpretieren, verschiedenartige Maschinenarchitekturen emulieren, ja sogar Anwendungsprogramme ausführen. Die Beispiele betreffen kleinere Maschinen, die sog. vertikale Mikrobefehlsformate aufweisen und sich in den soeben genannten Anwendungsfällen bewährt haben. Die Microdata Micro 800 ist eine besonders kleine Maschine zum Einbauen in Anwendungssysteme, im Grunde eine Art Mikrocontroller. Die IBM 360/25 ist eine besonders kostengünstige Maschine des Systems /360, die u. a. in Forschungsprojekten mit anwendungsspezifischen Mikroprogrammen eingesetzt wurde. Die Burroughs B1700 ist eine Datenverarbeitungsanlage, die Programmiersprachen direkt interpretieren und Maschinenbefehle beliebiger Zielarchitekturen emulieren kann. Die International Meta Systems MAX 2 wurde von Grund auf zum Interpretieren höherer Programmiersprachen entwickelt.

Kapitel 7: Neue Einfachmaschinen

Wie könnten einfache Maschinen aussehen, wenn man Anregungen aus der Entwicklungsgeschichte aufgreift, aber nicht um jeden Preis spart? Die Beschränkungen der Vergangenheit gelten nicht mehr.

Was die grundsätzlichen Anforderungen betrifft, orientiert man sich am besten an den heutigen Programmiergepflogenheiten, wie sie sich aus der Nutzung höherer Programmiersprachen ergeben. Wir wollen einfach, aber großzügig bauen, im Aufwand geringer als die typischen CISC- und RISC-Maschinen der mittleren und höheren Leistungsklassen.

Das systematische Entwickeln beginnt mit der Turingmaschine, die mit einem adressierbaren Speicher ausgerüstet und mit Verarbeitungswerken angereichert wird. Was sollen wir an- und einbauen? Um dies näher zu untersuchen, führen wir den an sich üblichen Begriff der Ressource in einer ganz bestimmten, typischen Weise ein. Ressourcen sind Speicher, Verarbeitungsschaltungen, Datenwege usw. Die Maschine arbeitet um so schneller, je mehr Ressourcen sie hat und je leistungsfähiger diese ausgelegt sind. Da wir nicht um jeden Preis sparen müssen, können wir es uns leisten, die Architekturentwicklung mit der Ressourcenauswahl zu beginnen. Zuerst die Ressourcen aussuchen, dann die Verbindungswege zwischen ihnen entwerfen, die Topologie der

Maschine. Auf diese Weise ergibt sich eine zu steuernde Hardware mit soundso vielen Auswahlsignalen, Auslösesignalen, Bedingungssignalen usw. Dann erst werden die Befehlsformate festgelegt, um die Steuerungsaufgaben zu lösen, die sich aus dieser Ressourcenanordnung ergeben.
Solche Maschinen werden als Ressourcenvektormaschinen bezeichnet. Die Betrachtungen bewegen sich auf der Register-Transfer-Ebene (RTL). Wir beginnen damit, die Steuersignale, Auswahlsignale, Direktwertsignale usw. der Ressourcen einfach an ein – zunächst fiktives – langes Befehlsregister anzuschließen, gleichgültig wieviele Bits sich dabei ergeben. Dann erst wird optimiert. Wir beginnen mit Auslegungen, die an herkömmliche Einadreßmaschinen erinnern (aber auch prinzipielle Unterschiede aufweisen). Dann werden Maschinen mit besonders kurzen Befehlen untersucht. Hierbei ergibt sich, daß eine der ersten Pionierleistungen überhaupt, Konrad Zuses Z3, auch heute noch als Anregung dienen kann. Im Grunde ist es eine Art Stackmaschine mit Stacktiefe 2. Wie kann man kurze und lange Befehle sowie kurze Adressen gut ausnutzen? Das wird nachfolgend diskutiert. Das Kapitel endet mit Überlegungen zur Mikroprogrammierung. Was können wir von den mikroprogrammgesteuerten Maschinen und den Wirkprinzipen der Mikroprogrammierung lernen? Das abschließende Beispiel ist eine Ressourcenvektormaschine, deren Mikrobefehle wie Maschinenbefehle genutzt werden können. Damit kann man nicht nur Mikroprogramme der Befehlsablaufsteuerung, Emulation und Interpretation schreiben, sondern auch System- und Anwendungsprogramme.
Alle Untersuchungen beziehen sich auf eine nahezu gleiche Ressourcenausstattung und weitgehend gleichartige Befehlswirkungen. Die Maschinen unterscheiden sich nur in der Schaltungsstruktur und in den Befehlsformaten.

Anhang
In Anhang 1 wird das Stackmodell vorgestellt, das mit der Programmiersprache C und dem Betriebssystem Unix zum Industriestandard geworden ist. Es liegt auch den Ressourcenvektormaschinen von Kapitel 7 zugrunde. Anhang 2 gibt einen Überblick über die architekturseitigen Register des Mikroprozessors Intel 8086. Er ist ein typisches Beispiel für eine Akkumulatormaschine, die mit zusätzlichen Registern erweitert wurde. Unsere Ressourcenvektormaschinen haben eine ähnliche Registerausstattung. Das 8086-Registermodell kann somit als Lehrbeispiel dienen, womöglich sogar zu eigenen Entwicklungen anregen. In Anhang 3 untersuchen wir das Codierungsvermögen einfacher Befehlsformate. Die Betrachtungen beziehen sich auf ein fiktives langes Befehlsregister, das alle Bitpositionen enthält, die erforderlich sind, um die Maschine Takt für Takt zu steuern. Kurze Befehle sind um so effektiver, je weniger von ihnen benötigt werden, um ein solches Befehlsregister zu laden. Anhang 4 ist eine Übersicht über die Befehlswirkungen unserer Ressourcenvektormaschinen, eine Art Befehlsliste. Alle Maschinenvorschläge in Kapitel 7 sollen im Grunde die gleichen Befehlswirkungen aufweisen. Nur die Befehlsformate und Codierungen sind unterschiedlich.

1. Einführung

1.1 Grundbegriffe

"Minimalmaschinen" ist hier ein Sammelbegriff für Universalrechnerarchitekturen, die in Hinsicht auf bestimmte quantifizierbare Merkmale als Minimallösungen ausgelegt sind. Es gibt hypothetische, fiktive und reale Minimalmaschinen. Sie werden zu verschiedenen Zwecken verwendet:

- zum allgemeinen Erkenntnisgewinn in der Mathematik (Berechenbarkeit) und in der theoretischen Informatik,
- als Plattformen für komplexitätstheoretische Untersuchungen,
- als theoretische Grundlage der Untersuchung von Rechnerarchitekturen, vor allem in Hinsicht auf die algorithmische Vollständigkeit und tatsächliche Universalität,
- als theoretische Grundlage zum Entwerfen praktisch brauchbarer Maschinen.

Was soll minimal werden?
Die Frage betrifft grundsätzliche Entwicklungsziele. Abhängig davon ergeben sich unterschiedliche Lösungen, die man nicht ohne weiteres unter dem Begriff des Minimalen zusammenfassen kann[1]. Die folgende Aufzählung gibt einen ersten Überblick über die Minimalkriterien:

- der Funktionsumfang im allgemeinen Sinne,
- die Komplexität im allgemeinen Sinne,
- die Kompliziertheit der Wirkprinzipien (Principles of Operation),
- die Schaltungsaufwendungen,
- die Anzahl der Befehlswirkungen,
- die Kompliziertheit der Befehlsabläufe,
- die Anzahl der Befehle,
- die Länge der Befehle,
- die Kompliziertheit der Befehlsformate.

Maschinen der Theorie, Maschinen der Praxis
Die Unterscheidung ist wichtig. Am Anfang der Entwicklungsgeschichte hatte beides nichts miteinander zu tun. In der Theorie ging es um die mathematische Grundlagenforschung, um die grundsätzliche Klärung des Problems der Berechenbarkeit. In der Praxis ging es darum, Rechenaufgaben großen Umfangs (Aerodynamik, Statik, Ballistik, Kernphysik usw.) automatisiert auszuführen.

1. In der Literatur wird nicht immer genau unterschieden. Man findet u. a. hypothetische Maschinen, die tatsächlich ganz einfach, aber vollkommen unbrauchbar sind, zusammen mit Maschinen abgehandelt, die sehr wenige oder sehr kurze Befehle aufweisen, aber einen vergleichsweise beträchtlichen Schaltungsaufwand erfordern.

Reale, hypothetische und fiktive Maschinen

Reale Maschinen sind als Hardware, also als technische Gebilde vorhanden. Entsprechende Architekturen werden so entwickelt, daß man sie tatsächlich schaltungstechnisch implementieren kann. Hierfür müssen alle Einzelheiten genau spezifiziert werden. Alle Kardinalitäten (Anzahlen) müssen endlich sein. Beispiel: das Maschinenwort hat eine Länge von 32 Bits, ganze Zahlen sind 32 Bits lange Binärzahlen in Zweierkomplementdarstellung.

Hypothetische Maschinen sind Maschinen oder Architekturen, die gar nicht zur Implementierung vorgesehen sind. Sie sollen nicht gebaut, ja nicht einmal emuliert werden. Sie dienen ausschließlich zu theoretischen Untersuchungen, zur Klärung von Grundsatzfragen oder zum reinen Erkenntnisgewinn[2]. Die Beschreibung muß nur soweit in die Einzelheiten gehen, wie es für den jeweiligen Zweck nötig ist. So kann es sein, daß man gar keine Befehlsformate spezifizieren muß, sondern die Beschreibung der Befehlswirkungen ausreicht. Auch dürfen oftmals Kardinalitäten unbestimmt gelassen werden. In manchen Fällen können sie sogar beliebig wachsen. Beispiel: das Maschinenwort darf beliebig lang sein, die Art der Zahlendarstellung ist beliebig, vorausgesetzt, sie erfüllt die Axiome der Algebra der ganzen Zahlen.

Fiktive Maschinen müssen ebenso genau spezifiziert werden wie reale. Sie gibt es aber nicht als Hardware. Manche fiktive Maschinen werden gar nicht implementiert; es sind reine Gedankenmodelle[3]. Programme, die für fiktive Maschinen geschrieben wurden, können durch Emulation (Interpretation) oder durch Übersetzen (Compilierung) ausgeführt werden. Fiktive Maschinen werden vor allem zu Lehr- und Forschungszwecken und als zwischengeordnete Architekturebenen in der Maschinenprogrammfertigung entwickelt, um Anwendungsprogrammschnittstellen anzubieten, die von jeder beliebigen realen Plattform unterstützt werden können[4]. Sie sind zumeist auf Einfachheit und Überschaubarkeit ausgelegt. Solche Maschinen sollen nicht die Spitzfindigkeiten, Einschränkungen und Eigentümlichkeiten aufweisen, die das Programmieren realer Maschinen oftmals beträchtlich erschweren. Datenstrukturen und Befehlswirkungen sollen die Tiefenstrukturen dessen darstellen, was den in Betracht gezogenen realen Maschinen gemeinsam ist. Da ohnehin nichts als Schaltung gebaut werden soll, kommt es auch nicht darauf an, um jeden Preis Schaltmittel einzusparen. Man kann also beim Definieren der Wirkprinzipen und Kardinalitäten großzügig sein. Beispiel: das Maschinenwort hat eine Länge von 256 Bits (damit garantiert alles hineinpaßt), Binärzahlen sind Objekte, die Länge und die Art der Zahlendarstellung sind in Objektdeskriptoren codiert.

2. Manche kann man eigentlich nur als Lösungen von Denksportaufgaben ansprechen. Oftmals wirklich brillant, für jegliche Anwendung – auch in der Theorie – aber vollkommen unbrauchbar.
3. Im Gegensatz zu den hypothetischen Maschinen sind die Einzelheiten aber soweit ausgearbeitet, daß man die Maschine auf Grundlage des aktuellen Fachwissens (State of the Art) emulieren oder als Schaltung bauen könnte. Hierzu gehören auch die fiktiven Maschinen, die in diesem Buch vorgestellt werden.
4. Der übliche Ausdruck: „runs everywhere“. Eine Architektur, die dies anstrebt, darf gar keine Merkmale aufweisen, die mit einer bestimmten Auslegung der Hardware zusammenhängen.