

Inhalt

1.	Einführung	5
2.	Überblick	6
2.1.	Rechnerarchitektur als Technikwissenschaft	6
2.2.	Grundlagen	9
2.2.1.	Modelle der Informationsverarbeitung	9
2.2.2.	Leistungsgrenzen eines elementaren Systems	11
2.2.3.	Alternative Prinzipien	14
2.3.	Entwicklungswege zum heutigen Stand der Rechnerarchitektur	17
2.3.1.	Universalrechner	17
2.3.1.1.	Herkömmliche Architekturen (CISC)	17
2.3.1.2.	Architekturen mit reduzierten Befehlslisten (RISC)	18
2.3.1.3.	Interne Parallelarbeit	19
2.3.1.4.	Architekturen mit Orientierung auf höhere Programmiersprachen	20
2.3.1.5.	Universelle Emulatoren	21
2.3.2.	Vektorrechner	22
2.3.3.	Parallelverarbeitung	22
2.3.4.	Datenflußmaschinen	25
2.3.5.	Datenstruktur- bzw. Spezialmaschinen	25
2.4.	Die eigene Arbeitsrichtung	27
2.4.1.	Vorhaben und Methodik	27
2.4.2.	Einordnung in den Stand von Wissenschaft und Technik	31
3.	Das elementare Modell der sequentiellen Informationsverarbeitung	32
3.1.	Von der Verarbeitungsschaltung zum Universalrechner	32
3.2.	Formale Darstellung	42
3.2.1.	Allgemeine Vereinbarungen	42
3.2.2.	Algorithmen	44
3.2.3.	Ressourcen	44
3.2.4.	Schaltungsanordnungen	46
3.2.5.	Abbildungsfragen	47
4.	Grundlagen der Bewertung	48
4.1.	Bewertung der Schaltungsanordnungen	48
4.2.	Bewertung der Algorithmen	52
4.3.	Bewertung der Aufwendungen	57
4.4.	Bewertung des Wirkungsgrades	60
5.	Tiefenstrukturen des Verarbeitungsmodells	61
5.1.	Datenstrukturen	62
5.1.1.	Numerische Datenstrukturen	62
5.1.2.	Nichtnumerische Datenstrukturen	68
5.2.	Programmstrukturen	69
5.2.1.	Operatoren	70
5.2.2.	Selektoren	70
5.2.3.	Iteratoren	76
5.2.4.	Aktivatoren	80
5.3.	Speicherung	85

6.	Vergegenständlichte Abstraktionen	88
6.1.	Prinzipien der Codierung	91
6.2.	Objekte	95
6.3.	Zugriffsorganisation	100
6.3.1.	Objektbeschreibungen und -zugriffe	100
6.3.2.	Ablauforganisation	103
6.3.3.	Programmstrukturen	104
6.4.	Speicherorganisation	104
6.4.1.	Organisation der Ressourcen	104
6.4.2.	Organisation der Speicherebenen	106
6.4.3.	Speicherverwaltung	110
6.5.	Datenstrukturen	114
6.5.1.	Numerische Datenstrukturen	115
6.5.2.	Nichtnumerische Datenstrukturen	115
6.6.	Programmstrukturen	117
6.6.1.	Operatoren	117
6.6.2.	Selektoren	118
6.6.3.	Iteratoren	126
6.6.4.	Aktivatoren	129
6.6.5.	Befehlsgestaltung	130
7.	Wirkprinzipien und Schaltungsstrukturen	132
7.1.	Grundlagen des Zusammenfassens von Vergegenständlichungen	132
7.2.	Ablaufsteuerprinzipien und deren Codierung	136
7.3.	Schaltungsstrukturen eines Hochleistungs- rechners	143
7.3.1.	Speicherstrukturen	145
7.3.2.	Operationswerke	145
7.3.3.	Selektions- und Iterationswerke	148
7.3.4.	Ausnutzung der Schaltmittel	156
7.3.5.	Übersicht	156
7.4.	Leistungsbetrachtungen	158
7.4.1.	Absolute Grenzen	158
7.4.2.	Vergleich mit herkömmlichen Architekturen	158
7.4.3.	Vergleich mit RISC- Architekturen	160
7.4.4.	Vergleich mit VLIW- Architekturen	161
7.4.5.	Vergleich mit Sondermaschinen	162
8.	Empfehlungen und Ausblicke	165
9.	Zusammenfassung	169
10.	Literaturverzeichnis	170

1. Einführung

Die Entwicklung der Rechentechnik ist im wesentlichen darauf gerichtet, das absolute Leistungsvermögen zu erhöhen, das Preis- Leistungs- Verhältnis zu verbessern und neuartige Gebrauchseigenschaften bereitzustellen.

Vorliegende Arbeit soll auf dem Gebiet der Rechnerarchitektur dazu einen Beitrag leisten. Gegenstand ist der einzelne Universalrechner. Dessen Architekturprinzipien und Schaltungsstrukturen sollen so gestaltet werden, daß im Rahmen bestimmter Aufwandsvorstellungen (vom Mikrocontroller bis zum Supercomputer) ein jeweils optimales Leistungsvermögen verwirklicht werden kann.¹ Im folgenden geht es darum, für einschlägige Forschungsarbeiten eine Arbeitsrichtung zu begründen, erste Vorstellungen zu umreißen und Anregungen für das wissenschaftlich- technische Handeln zu vermitteln.

Abschnitt 2 gibt - aus technikwissenschaftlicher Sicht - einen Überblick über jene Grundlagen der Informationsverarbeitung, die für dieses Ziel wesentlich sind, und erläutert die eigene Arbeitsrichtung: Um die Möglichkeiten moderner Technologien in überlegene Gebrauchswerte umsetzen zu können, werden Algorithmen, Datenstrukturen, Sprachkonstrukte und Schaltungsstrukturen aus einer gleichsam ganzheitlichen Sicht betrachtet, wobei das Ziel darin besteht, technische Mittel, also Hardware- Strukturen, so leistungsfähig und zweckmäßig wie möglich gestalten zu können.

Dem liegt ein elementares Verarbeitungsmodell zugrunde, das in Abschnitt 3 erklärt wird.

In Abschnitt 4 werden die Grundlagen der Bewertung von Schaltungsstrukturen, Algorithmen und Aufwendungen dargestellt.

Abschnitt 5 gibt einen Überblick über wesentliche Tiefenstrukturen des Verarbeitungsmodells anhand einer Erfahrungsbasis, die durch eingeführte Maschinenarchitekturen und Programmiersprachen gegeben ist. Diese Darstellung liefert grundlegende, universell nutzbare Abstraktionen für Datenstrukturen, Operationen und Ablaufprinzipien, und sie vermittelt Anregungen für das Vorgehen bei weiteren systematischen Untersuchungen.

In Abschnitt 6 werden bestimmte Abstraktionen für die technische Umsetzung ausgewählt und näher beschrieben.

In Abschnitt 7 wird die Ausgestaltung der Wirkprinzipien und Schaltungsstrukturen von Universalrechnern auf Grundlage der Abstraktionen diskutiert, die in den Abschnitten 5 und 6 erläutert wurden.

Abschließend werden in Abschnitt 8 Empfehlungen für künftige Arbeiten gegeben.

¹ Konkrete technisch- ökonomische Überlegungen, die diese Zielstellung näher begründen, enthält /245/.

2. Überblick

2.1. Rechnerarchitektur als Technikwissenschaft

Eine hinreichend entwickelte Technikwissenschaft bildet die theoretische Grundlage des jeweiligen Gebietes der Technik; sie liefert Richtlinien, Methodenlehren, Berechnungsverfahren und Grundsatzlösungen für das Analysieren und Vervollkommen gegebener und für das Schaffen neuer technischer Gebilde bzw. Verfahren. Als Beispiele seien die Aerodynamik, die Angewandte Mechanik und die Theoretische Elektrotechnik genannt. Sie stellen Mittel bereit, mit denen Gebilde der Technik, wie Flugzeuge, Brücken, Motore, Hochfrequenzschaltungen klassifiziert, bewertet und rechnerisch behandelt werden können. (Beispielsweise konnte bereits vor 1920 der Verbrennungsmotor in allen entscheidenden Parametern berechnet werden.)

Das Gebiet der Rechnerarchitektur ist von einem vergleichbaren Entwicklungsstand noch weit entfernt.

Der Mangel an wissenschaftlicher Grundlegung wird offensichtlich besonders stark von jenen Fachleuten empfunden, die unmittelbar mit der Schaffung hochleistungsfähiger Maschinen befaßt sind.

So schreibt Patt in der Einführung zu einer Übersichtsdarstellung, die industriell gefertigte Hochleistungsrechner betrifft (/272/), daß Rechnerarchitektur keine Wissenschaft sei.

Lincoln (/220/) verweist darauf, daß gerade auf dem Gebiet der Supercomputer viele wesentliche Entscheidungen gefühlsmäßig ("gut feel") getroffen werden und daß es in der Regel besser ist, einen einzelnen hochqualifizierten Bearbeiter mit der Architekturdefinition zu betrauen als ein Kollektiv.

Colwell schreibt in /146/: "Computer systems work not only lacks a formal foundation, it has not even progressed to the point where a taxonomy or other means of codifying existing knowledge can be constructed. Proofs cannot be expected in computer systems work, for they presuppose some set of axioms that has yet to be created."

Die meisten Standardwerke beschreiben gegebene Architekturen, vergleichen diese unter verschiedenen Gesichtspunkten und geben allgemeine Hinweise. So gibt das Buch von Myers (/86/), das als Beispiel für einschlägige Publikationen (wie etwa /50/, /56/, /61/) angesehen werden kann, folgende Empfehlungen für die Ausarbeitung von Rechnerarchitekturen:

- konzeptionelle Einheitlichkeit
- Orthogonalität
- Adäquatheit zu den Nutzer- Anforderungen
- Optimierung der technischen Mittel gemäß der Nutzungshäufigkeit
- Vorkehrungen für Erweiterungen
- Unabhängigkeit von Implementierung und Technologie.

Weitere Bemühungen um Systematisierung und Begriffsbildung betreffen formale Beschreibungen (wie ISP zur Beschreibung von Befehlslisten; /4/) sowie taxonomische bzw. morphologische Schemata, z. B. die Taxonomie der Rechnerarchitektur nach

Giloi (/177/), die Klassifizierung der Rechnerstrukturen nach Flynn (/167/) oder das Computer-Spektrum nach Hockney (/196/).

Daneben wurden in den letzten Jahren umfangreiche meßtechnische Untersuchungen betrieben (Beispiele: /120/, /147/, /153/, /159/-/161/, /207/, /283/. Sie haben wichtige Erkenntnisse gebracht: zur Nutzungshäufigkeit von Befehlen, zu Trefferraten bei verschiedenen Cache-Organisationen und zum innewohnenden Parallelismus in Programmen. Von dieser Erfahrungsbasis aus wurden neue, an den Meßergebnissen orientierte Architekturen definiert. So wurde die Vorgehensweise bei der Schaffung einer solchen Architektur ausdrücklich als "measurement oriented approach" bezeichnet.¹

Gegenwärtig ist die Wissenschaft von der Rechnerarchitektur also vorwiegend eine Erfahrungswissenschaft: "Computer science is an empirical discipline... Each new machine that is built is an experiment. Actually constructing the machine poses a question to nature: and we listen for the answer by observing the machine in operation and analyzing it by all analytical and measurement means available."²

In diesem Wechselspiel zwischen Schaffen und Bewerten neuer technischer Lösungen ist das systematische versuchsweise Entwickeln mit dem Ausarbeiten von Versuchsanordnungen in den Naturwissenschaften vergleichbar. Um den Grad an Exaktheit zu erhöhen, werden in der Literatur Vorstellungen diskutiert, die im wesentlichen auf folgende Ansätze zurückführbar sind:

1. Axiomatisierung. Das ist die klassische Methode zur Begründung der Mathematik. Sie geht in ihrer modernen Form auf Hilbert zurück: "Wenn es sich darum handelt, die Grundlagen einer Wissenschaft zu untersuchen, so hat man ein System von Axiomen aufzustellen, welche eine genaue und vollständige Beschreibung derjenigen Beziehungen enthalten, die zwischen den elementaren Begriffen jener Wissenschaft stattfinden. Die aufgestellten Axiome sind zugleich die Definition jener elementaren Begriffe, und jede Aussage innerhalb des Bereiches der Wissenschaft, deren Grundlage wir prüfen, gilt uns nur dann als richtig, fall sie sich mittels einer endlichen Anzahl logischer Schlüsse aus den aufgestellten Axiomen ableiten läßt" (/191/). Demgemäß werden die Anforderungen an die Architektur axiomatisch formuliert, und es ist zu beweisen, daß die Axiome selbst widerspruchsfrei sind und daß die Festlegungen der Architektur die Axiome erfüllen.³

2. Algebraische Modellierung. Als Beispiel und Anregung sei der algebraische Ansatz zur Leistungsbewertung nach /261/ genannt. Damit wird das Leistungsverhalten von (realen) Super-

1 Es handelt sich um die Precision Architecture von Hewlett-Packard (/120/, /226/, /229/).

2 Aus /87/; zit. nach /146/.

3 Die Anregung wurde /224/ entnommen. Weiterhin sei z. B. auf /97/, /194/ und /195/ verwiesen. Diese Arbeiten gelten Fragen der Programmierung; die Nutzbarkeit für das Gebiet der Architekturprinzipien bedarf weiterer Untersuchungen.

rechnern mit einer Leistungsalgebra P, einer Anwendungsalgebra H und einer Abbildung $F: H \rightarrow P$ untersucht. P und H sind geordnete Halbgruppen, womit arithmetische Operationen, Transporte und Verzögerungen (die z. B. durch das Warten auf Speicherzugriffe bedingt sind) modelliert werden können.

3. Linearoptimierung: "If it were possible to somehow enumerate all of the constraints...and then assign appropriate relative weights to them, architectural design might be reduced to a linear programming problem" (/146/).

In der Literatur herrscht Übereinstimmung darüber, daß solche Ansätze derzeit und in nächster Zukunft nicht für das Ausarbeiten konkreter Architekturen nutzbar sind. Man wird sich also weiterhin auf empirische Grundlagen stützen müssen und kann lediglich versuchen, durch Nutzung bewährter Methoden des wissenschaftlichen Arbeitens Anteil und Einfluß "gefühlsmäßiger" Entscheidungen so gering wie möglich zu halten. Das kann aber im folgenden nur anhand überschaubarer Sachverhalte demonstriert werden, da allein die bloße Beschreibung einer neuen Rechnerarchitektur den Umfang der vorliegenden Arbeit übersteigt¹, umso mehr die ausführliche Darlegung aller Grundgedanken, Einflüsse, Probeentwürfe, Variantenvergleiche und Entscheidungen. Die folgende Aufzählung gibt einen kurzen Überblick über die Methoden, die der Arbeit hauptsächlich zugrunde liegen:

1. Systematisches Vorgehen in nachvollziehbaren Schritten.
2. Rückführung von Neuem auf Bekanntes bzw. von Kompliziertem auf Einfaches. Es wird ein Modell der elementaren Informationsverarbeitung zugrunde gelegt, das auf kombinatorische Zuordnungen, also auf aussagenlogische Verknüpfungen, die durch Boolesche Gleichungen beschreibbar sind, und auf binäre Speichermittel zurückgeht. Es wird gezeigt (in Abschnitt 3.1.), wie spezielle, für die Implementierung eines einzigen Algorithmus ausgelegte Schaltmittel schrittweise in die bekannte v. Neumann-Rechnerstruktur überführt werden können. Jede Rechnerstruktur wird als Sammlung von Ressourcen aufgefaßt, woraus die Aufgabe ersichtlich ist, diese Ressourcen so zweckmäßig wie möglich zu nutzen.
3. Systematisches Aufarbeiten der Erfahrungsgrundlagen. Das betrifft zunächst bewährte Maschinenarchitekturen und Programmiersprachen.
4. Metasprachliche Betrachtungen, also Untersuchungen der Eigenschaften von Beschreibungsmitteln der fraglichen Sachverhalte. Diese auf Tarski (/300/) zurückgehende Methode wird sowohl für die Formalisierung bestimmter Aspekte des Verarbeitungsmodells (Abschnitt 3.2.) als auch für die Auslegung codierter Beschreibungen von Informationsstrukturen (strukturrell-deskriptive Angaben; Abschnitte 5 und 6) genutzt.

¹ Vgl. den Umfang von Architekturhandbüchern (/36/, /38/). Als ersten Überblick über eigene Vorstellungen s. /248/.

2.2. Grundlagen

2.2.1. Modelle der Informationsverarbeitung

Hier geht es ausschließlich um Informationsverarbeitung mit technischen Mitteln. Eine entsprechende Einrichtung kann ganz allgemein als "black box" dargestellt werden, die über Ein- und Ausgänge mit ihrer Umgebung verbunden ist (Bild 1).

Technische Mittel sind stets für bestimmte Zwecke vorgesehen; den Belegungen der Ein- und Ausgänge kommen folglich bestimmte Bedeutungen zu. Die Informationsverarbeitung besteht darin, aus aktuellen Eingangswerten, die Träger gewisser Bedeutungen sind, in endlicher Zeit Ausgangswerte zu ermitteln, die andere Bedeutungen haben. Fragen der bedeutungserhaltenden Informationsübertragung und -wandlung werden hier nicht behandelt¹; Fragen der Informationsspeicherung sind nur von Interesse, sofern es um Speicherung zu Verarbeitungszwecken geht.²

Jede informationsverarbeitende Einrichtung ist gekennzeichnet durch ihre Funktionsweise, ihre innere Struktur und ihre technischen Grundlagen.

Die technischen Grundlagen umfassen die Elemente, aus denen die Einrichtung aufgebaut ist, deren Wirkprinzipien sowie die entsprechenden Technologien.

Die Struktur wird durch die Anordnung und Verbindung der Elemente bestimmt.

Die Funktionsweise bezeichnet die Prozesse der Informationsverarbeitung im einzelnen. Dabei ist zwischen äußerer und innerer Funktionsweise zu unterscheiden: erstere beschreibt die Informationswandlung zwischen den Ein- und Ausgängen der "black box" ohne, letztere mit Bezug auf die innere Struktur.

Es sind verschiedene Modelle der Informationsverarbeitung denkbar. Jedes Modell steht für eine bestimmte Betrachtungsweise; es beschreibt einen grundlegenden Ansatz bzw. eine konzeptionelle Auffassung, wobei jeweils bestimmte Gesichtspunkte hervorgehoben und andere vernachlässigt werden. So können beispielsweise im Vordergrund stehen:

- die technischen Grundlagen
- die Art und Weise der Informationsdarstellung
- die strukturelle Gestaltung
- die innere Funktionsweise
- die äußere Funktionsweise
- die Anwendungsgebiete.

Dem Anliegen der Arbeit gemäß soll sich die weitere Betrachtung auf digital arbeitende, vorzugsweise programmgesteuerte Einrichtungen auf mikroelektronischer Grundlage beschränken.

¹ Das ist beispielsweise Gegenstand der Informationstheorie, der Nachrichtentechnik, der Theorie der Informationswandler usw.

² Das betrifft vorzugsweise Speicher mit wahlfreiem Zugriff für binär codierte Information (Flipflops und RAM- bzw. ROM-Zellen).

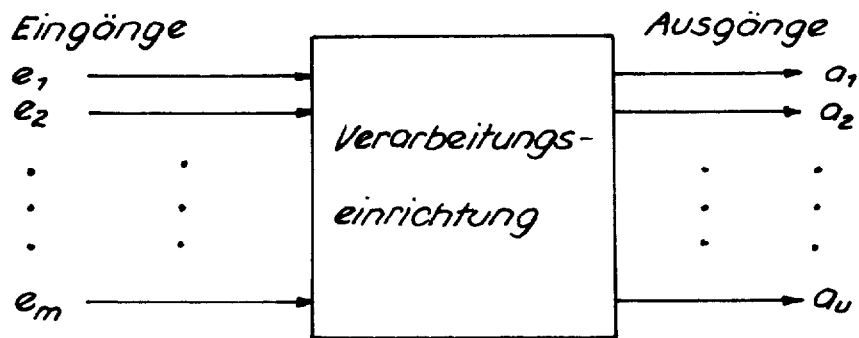
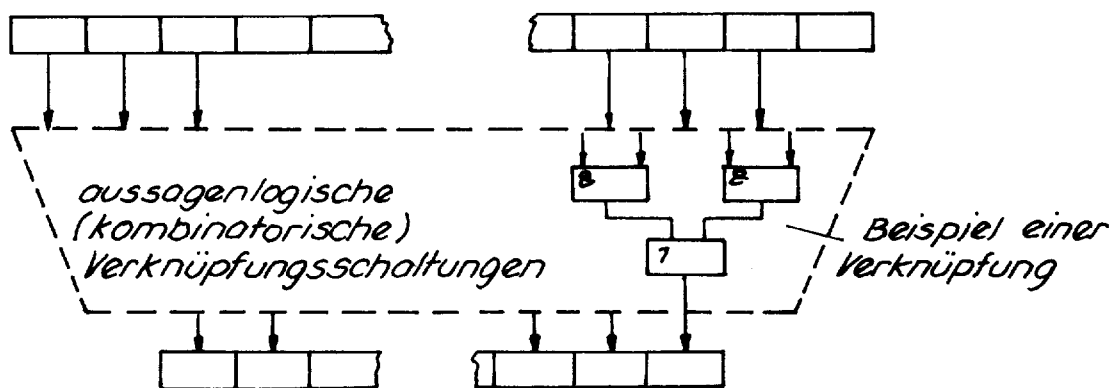


Bild 1 "Black box" Darstellung einer techn. Einrichtung zur Informationsverarbeitung

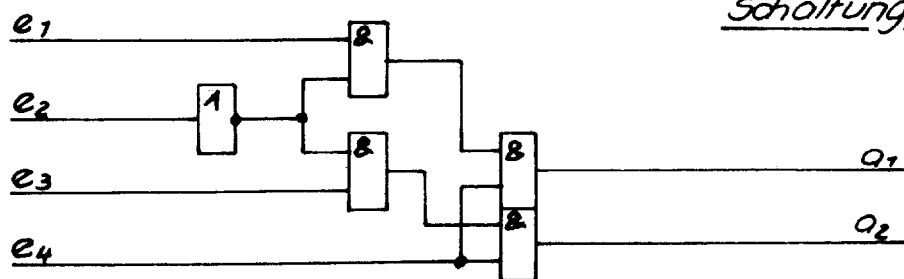
Binäre Speicher (Argumente)



Binäre Speicher (Resultate)

Bild 2 Allgemeines Schema der binären Informationsverarbeitung

Bild 3 Beispiel einer Verknüpfung mit Schaltungstiefe 3



Folgende Prinzipien haben sich im Ergebnis der bisherigen Entwicklung durchgesetzt:

1. Informationsdarstellung durch binäre Codierung
2. Verknüpfungen gemäß den Prinzipien der Aussagenlogik
3. algorithmische, deterministische Arbeitsweise
4. getaktete Arbeitsweise (diskrete Zeitschritte)
5. Programmsteuerung
6. weitgehende Flexibilität bis hin zur Universalität, so daß durch Wechseln des steuernden Programms eine Einrichtung beliebige Algorithmen ausführen kann (Beschränkung lediglich durch Zeitbedarf bzw. Umfang der vorhandenen Ressourcen)
7. Aufbau mit mikroelektronischen Schaltungen (Grundlagen: Schaltungen für elementare aussagenlogische Verknüpfungen (Gatter) sowie Speichermittel (Flipflops, RAM- und ROM-Zellen).

Dafür werden absolute Leistungsgrenzen diskutiert; im Anschluß soll gezeigt werden, welche Konsequenzen sich ergeben, wenn die genannten Prinzipien geändert bzw. durch andere ersetzt werden.

2.2.2. Leistungsgrenzen eines elementaren Systems

Das Leistungsvermögen betrifft sowohl den Zeitbedarf für die jeweilige Verarbeitungsaufgabe als auch die Möglichkeit, eine solche Aufgabe überhaupt ausführen zu können. Es wird bestimmt durch:

- die Kompliziertheit der Verarbeitungsaufgabe
- die Ausbreitungsgeschwindigkeit der informationstragenden Signale und die Länge der Signalwege
- die Arbeitsgeschwindigkeit der Verarbeitungseinrichtungen
- die vorgesehenen Aufwendungen.

Zunächst sollen die Leistungsgrenzen der elementaren binären Informationsverarbeitung mit aussagenlogischen Verknüpfungen untersucht werden. Dem liegt das allgemeine Schema nach Bild 2 zugrunde: gespeicherte Bits werden miteinander verknüpft; die entstehenden Resultate werden ebenfalls gespeichert. Die Leistungsfähigkeit wird durch die konkrete Ausgestaltung dieses Schemas bestimmt. Die Zeit für einen Verknüpfungszyklus ergibt sich zu:

$$t_c = s \cdot t_p + t_A + t_{SETUP} + t_L + t_{TOL}. \quad (2.1)$$

Bedeutung der Symbole:

- t_c : Zykluszeit
 t_p : Verzögerungszeit des einzelnen Gatters (Richtwerte moderner Technologien: CMOS 1 - 2 ns, ECL 350 ps, GaAs 150 ps)¹
 s : Schaltungstiefe ($s = 1, 2, \dots, n$); der Wert drückt aus, wieviele Gatter im ungünstigsten Fall nacheinander durchlaufen werden müssen (Bild 3 zeigt ein Beispiel; dort ist $s = 3$)
 t_A : Zugriffszeit zu den gespeicherten Bits (Richtwert: $4 t_p$)²
 t_{SETUP} : Vorhaltezeit für die Resultatübernahme (Richtwert: $4 t_p$)³
 t_L : Summe der Laufzeiten durch alle Verbindungsleitungen (für jedes Resultatbit zu bestimmen, der ungünstigste Wert wird zugrunde gelegt); Näherungswert: 1 ns/15 cm.
 t_{TOL} : technisch bedingte Zugabe (Toleranz- Ausgleich); Richtwert: wenige t_p , stark von technologischen Gegebenheiten abhängig.

Betrachtet man Bild 2 als Illustration für das Operationswerk eines Universalrechners und nimmt man idealisierte Betriebsbedingungen an, so ergibt

$$P = \frac{1}{t_c} \quad (2.2)$$

die Maximalleistung in "Befehlen pro Zeiteinheit". Die idealisierten Betriebsbedingungen bestehen darin, daß eine lückenlose Aneinanderreihung von Operandenverknüpfungen angenommen wird, also mit zusätzlichen Schaltmitteln parallel die steuernden Befehle gelesen, die Operanden herangeschafft und die Resultate abtransportiert werden.

Welche Möglichkeiten gibt es, die Parameter zwecks Leistungssteigerung zu beeinflussen?

t_p : Die Schaltungstechnologie hat nach wie vor beachtliche Auswirkungen auf die Hardwarekosten: je "schneller" die Technologie, um so bedeutsamer die Zusatzaufwendungen (Kühlung, Stromversorgung usw.), um so geringer der fertigungstechnisch beherrschbare Integrationsgrad. t_p kann aus physikalischen Gründen nicht beliebig vermindert werden. Für elektronische Technologien werden bis zu 30 ps angestrebt⁴, bei optischen Prinzipien hofft man, 1 ps noch "auf dem Wege normaler Ingenieurarbeit" erreichen zu können.⁵

¹ Zu GaAs s. /305/; /278/ beschreibt ein neues Logikprinzip auf Si-Basis, das besser als ECL sein soll (190 ps).

² Zeit zwischen Taktflanke und Verfügbarkeit der Information.*

³ Zeit zwischen Bereitstellung und Übernahme der Information.*

⁴ Vgl. /206/.

⁵ Nach /315/; $t_p = 100$ fs erscheint noch möglich (absolut kürzeste Impulsdauer: 10 fs).

* Richtwert nach Datenblättern üblicher Logikbaureihen.

t_A , t_{SETUP} , t_{TOL} hängen ebenfalls direkt mit der Technologie zusammen (vgl. die Richtwerte auf S. 12). Die Verminderung von t_{TOL} erfordert hohe fertigungstechnische Aufwendungen (Präzision der Leiterplattenfertigung, aufwendige Meßtechnik, ggf. Abgleichvorgänge).

t_L : Die Schaltungstechnologie bestimmt in erster Linie die physische Größe. Wenn das Schema gem. Bild 2 auf einen Schaltkreis paßt, ist $t_L < 1$ ns und üblicherweise vernachlässigbar. Kritisch sind die Übergänge zwischen den Schaltkreisen. Die notwendigen Koppelstufen verlangsamen den Informationstransport, und zwar um so mehr, je leistungsfähiger die Basistechnologie ist (bei GaAs hat das Laufzeitverhältnis zwischen internen Verbindungen und solchen, die Schaltkreisgrenzen überschreiten, die Größenordnung 1:10). Optische Verbindungen sind schneller (bis 30 cm/ns), wobei die Laufzeit nicht durch induktive, kapazitive und ohmsche Belastung verlängert wird.

s : Die Schaltungstiefe wird vom Gatter-Sortiment bestimmt. Entscheidende Parameter sind:

- Eingangszahl: übliche Werte liegen zwischen 2 und 8; bei großer Zahl an Eingängen wird die Schaltungsstruktur größer und daher langsamer.
- Funktionsvielfalt. Die Verknüpfungen werden mit Transistorstrukturen realisiert. Komplizierte Verknüpfungen bedingen ausgedehntere Anordnungen mit entsprechend längeren Verzögerungszeiten, deshalb gibt es in den meisten Technologien nur einen Typ elementarer Verknüpfungen (NAND bzw. NOR).
- Anzahl der nachschaltbaren Eingänge ("fan out").

Alle Verknüpfungen werden letztlich durch Boolesche Gleichungen beschrieben, die für jedes Resultatbit die Abhängigkeit von den jeweiligen Argumentbits angeben. Eine bestimmte Schaltungstiefe s ist dann erreichbar, wenn sich für jede Boolesche Gleichung eine Dekomposition für das jeweilige Gattersortiment¹ derart angeben läßt, daß für kein Netzwerk s überschritten wird. Beim Streben nach höchstem Leistungsvermögen hat man somit nur 2 Alternativen:

1. Beschränkung von s im Hinblick auf ein bestimmtes kleines t_c , indem nach jeweils s Gatter-Stufen Speichermittel eingefügt werden. Kompliziertere Operationen werden so durch eine Kaskaden-Anordnung von Schaltungen gem. Bild 2 verwirklicht: das ist das bekannte Pipelining-Prinzip.

2. Anordnung kombinatorischer Schaltmittel für die gewünschten komplizierten Verknüpfungen, wobei t_c nach dem notwendigen s gewählt wird; das Ziel besteht darin, eine Verarbeitungsaufgabe statt in n Schritten mit $t_{c,1}$ in einem Schritt mit $t_{c,2}$ auszuführen.

Beim Pipelining wird das einzelne Resultat nicht schneller gebildet als bei kombinatorischer Zuordnung, sondern es wird

¹ Zur Theorie s. /52/; zur rechenpraktischen Nutzung /164/.

mehr Zeit benötigt, nämlich für jede eingefügte Speicherstufe ein Taktzyklus. Der Vorteil besteht vielmehr darin, daß bei n Speicherstufen n-1 Operationen parallel mit einem Zeitversatz von jeweils t_c ausgeführt werden können. Das Prinzip lohnt sich also nur, wenn eine Vielzahl gleichartiger Operationen auszuführen ist ("Vektorrechner").¹ Der 2. Ansatz läuft darauf hinaus, sehr komplexe Verknüpfungen vorzusehen. Diese sind aber nicht immer universell nutzbar ("Spezialrechner"). Für die einzelne Operation ist dieser Ansatz überlegen, wenn gilt:

$$t_c \cdot 2 < z t_c \cdot 1, \quad (2.3)$$

wobei z die Zahl der Taktzyklen angibt, die notwendig sind, um die Operation mit den einfacheren Verknüpfungen auszuführen. Läßt sich beim Pipelining die Parallelverarbeitung ausnutzen ("Vektorisierung"), so müßte gelten²

$$t_c \cdot 2 < t_c \cdot 1/n, \quad (2.4)$$

wenn die kombinatorische Zuordnung überlegen sein soll.

2.2.3. Alternative Prinzipien

Einen Anwender interessiert an sich nur die reine Verarbeitungszeit für sein Problem und nicht die interne Struktur der Verarbeitungseinrichtung. Es ist deshalb gerechtfertigt, die allgemein üblichen Prinzipien (S. 11) in Frage zu stellen:

1. keine binäre Codierung, also mehrwertige Codierung, Analogdarstellung u. a.
2. andere als aussagenlogische Verknüpfungen. Beispiele: mehrwertige Logik, Wahrscheinlichkeitslogik, Schwellwertlogik, assoziative Verknüpfungen, neuronale Prinzipien
3. Verzicht auf algorithmische Arbeitsweise, also assoziative Arbeitsweise, Selbstorganisation usw.
4. keine getaktete Arbeitsweise, stattdessen kontinuierliche Zeitabläufe
5. keine Programmsteuerung. Alternativen: entweder Zwangssteuerung für einen bestimmten Zweck oder Prinzipien des Lernens und der Selbstorganisation
6. Verzicht auf Universalität und Flexibilität, also: Einzwecksysteme
7. andere technische Grundlagen, z. B. optische oder biochemische Prinzipien.

¹ Deshalb sind manche Hochleistungsschaltkreise (bes. solche für Gleitkommaoperationen) entsprechend umschaltbar.

² Näherungsweise Ausdruck; n: Anzahl der Pipeline-Stufen. Mit praktischen Werten um 4...16 ist (2.4) kaum zu erfüllen.

Manche dieser Alternativen sind in der Technik bereits verwendet worden. Sie sind aber mit spezifischen Problemen verbunden (Beispiel: Genauigkeit bei der analogen Informationsverarbeitung).

Andere Alternativen sind technisch ohne weiteres durchführbar; ihre Verwirklichung ist eine Frage von Kosten- Nutzen- Rechnungen (z. B. Einzweckmaschinen im Vergleich zu Universalmaschinen).

Einige Prinzipien haben - für sich gesehen - durchaus Vorteile. Aus der Sicht des Technikers ist aber zu prüfen, ob im nutzbaren Erzeugnis Kosten- bzw. Leistungsvorteile tatsächlich verwirklicht werden können. Beispiele:

- Mehrwertige Logik oder Schwellwertlogik: Fragen der Störsicherheit, der Schaltgeschwindigkeit und der Fertigungstoleranzen
- asynchroner Betrieb: Beherrschbarkeit in den Fertigungs- und Prüfprozessen der Schaltkreise und Systeme.

Manche Prinzipien bedürfen noch umfangreicher Forschungsarbeiten, bevor sie für eine Nutzung in Betracht gezogen werden können. Die Forschungen betreffen sowohl die Erhöhung der Verarbeitungsgeschwindigkeit im Rahmen eingeführter Modelle der Informationsverarbeitung als auch vollkommen neuartige Lösungsansätze. Tafel 1 vermittelt einen Eindruck von den vielfältigen Möglichkeiten.

Wichtige Leistungen der Informationsverarbeitung lebender Systeme konnten bisher nur sehr unvollkommen oder gar nicht technisch nachgeahmt werden. Die bekannten Schaltmittel haben aber bereits einen Geschwindigkeitsvorteil von $1:10^3 \dots 1:10^6$ gegenüber den Nervenzellen.¹ Folglich kann bloße Geschwindigkeitssteigerung nicht der einzige Weg sein; vielmehr sind künftig neue Ansätze zur technischen Informationsverarbeitung notwendig, die über den Gedankenkreis "berechenbare Funktionen und Algorithmen" hinausgehen (man muß in einem bestimmten Zeitabschnitt t_c ²) offenbar wesentlich mehr leisten, als Verknüpfungsergebnisse gemäß der Aussagenlogik zu bilden).

Für eine technische Umsetzung solcher Vorstellungen gibt es noch keine gesicherte Grundlage.³ Es ist deshalb gerechtfertigt, Bemühungen um bessere technische (also: beherrschbare und anwendbare) Lösungen auf die Mikroelektronik und die bekannten Prinzipien der digitalen Informationsverarbeitung zu stützen. Wegen der wissenschaftlichen, technischen, anwendungspraktischen und wirtschaftlichen Bedeutung ist die weitere Beschränkung auf programmgesteuerte Rechner zweckmäßig. Nachfolgend werden die wesentlichen Entwicklungswege dieser Technik kurz skizziert.

1 Bezogen auf die gängigen Modellvorstellungen zur Arbeitsweise der Neuronen.

2 Nach /28/ erkennt der Mensch ein Bild in 20-30 ms, d. h. mit höchstens wenigen hundert Verarbeitungsschritten.

3 Trotz aller Bemühungen dürften solche Prinzipien für die nächsten Produkt- Zyklen (Forschung -> Entwicklung -> Markteinführung -> Effekte beim Anwender) nicht wirksam werden.

Ziel	Gegenstand der Bemühungen		
	Struktur		Technologie
höhere Geschwindigkeit	<ul style="list-style-type: none"> ● einfachere Strukturen für kurze Zykluszeiten (RISC) ● Pipelining ● mehrere Verarbeitungswerke: <ul style="list-style-type: none"> - Scoreboard - VLIW - Datenflußprinzipien 	<p>In jeder Hinsicht:</p> <p><u>Massive Parallelisierung</u></p>	<ul style="list-style-type: none"> ● GaAs ● HEM- Transistoren ● COL (/278/) ● Josephson- Effekt ● optische Prinzipien
komplexere Funktionen	<ul style="list-style-type: none"> ● Sondermaschinen ● mehrwertige Logik ● zelluläre Systeme ● neuronale Systeme 		<ul style="list-style-type: none"> ● optische Prinzipien (z. B. holographische) ● biochemische Prinzipien ● mehrwertige bzw. Analog- Prinzipien (z. B. Schwellwertlogik)

Tafel 1 Grundsätzliche Möglichkeiten zur Verbesserung der Verarbeitungsleistung

2.3. Entwicklungswege zum heutigen Stand der Rechnerarchitektur

2.3.1. Universalrechner

2.3.1.1. Herkömmliche Architekturen (CISC)

Die Entwicklung der Rechnerarchitektur war in den 70er Jahren zu einem gewissen Abschluß gekommen (Beispiele: IBM/370, CDC 6600 und 7600, DEC PDP 11 und VAX 11). Die seinerzeitigen Lösungen wurden maßgeblich von folgenden Sachverhalten bestimmt:

1. Speichermittel mit wahlfreiem Zugriff waren kostbare Ressourcen. Große Speicherkapazitäten konnten praktisch nur mit Ferritkernspeichern realisiert werden; das führte aus technischen Gründen zur Trennung zwischen zentralen Speichern und angeschlossenen Verarbeitungseinrichtungen.
2. Der Integrationsgrad der Schaltmittel war vergleichsweise gering, die Geschwindigkeit aber bereits recht hoch (CDC 6600: mit diskreten Transistoren 100 ns Zykluszeit bereits 1965).
3. Es bestand eine Diskrepanz zwischen der Zykluszeit der Verarbeitung und der des Speichers (Richtwert: 1:5...1:10).
4. Schnelle Registerspeicher waren teuer. Übliche technische Lösungen: Flipflops (360/75, EC 1040), Kondensatorspeicher (360/65), schnelle Kernspeicher (360/50, Siemens 4004), Dünnschichtspeicher (Univac).
5. Große Festwertspeicher für Mikroprogramme waren vergleichsweise kostengünstig (z. B. nach dem Transformatorprinzip arbeitend, einige tausend Worte mit über 100 bit, 200 - 400 ns Zyklus).¹
6. Es wurde überwiegend mit Maschinenbefehlen programmiert (Assembler-Programmierung). Das führte dazu, bei der Gestaltung der Architektur nach möglichst leistungsfähigen und flexibel nutzbaren Maschinenbefehlen zu suchen (die langsame Verbindung zwischen Speicher und Prozessor mußte gut ausgenutzt werden, die Mikroprogrammsteuerung erlaubte es, komplizierte Verarbeitungsabläufe zu implementieren). Auch waren die Befehle so festzulegen, daß sie vom Programmierer bequem und wirkungsvoll genutzt werden konnten. Wichtige Kriterien waren Vollständigkeit, Symmetrie und Orthogonalität sowie die weitgehende Unabhängigkeit der Architekturdefinition von technischen Gegebenheiten, um programmkompatible Familien von Rechenanlagen anbieten zu können.

Die meisten der heutzutage üblichen Architekturprinzipien wurden seinerzeit geschaffen: Mikroprogrammsteuerung, universelle Bussysteme, Universalregister, Adressierungsverfahren, virtuelle Speicher, das 8-bit-Byte als grundlegende Datenstruktur usw.

¹ Beispiel: Transformatorspeicher der EC 1040 (/40/).

Die Mikroprozessoren, die in den 70er Jahren aufkamen, waren keine architekturseitigen, sondern technologische Neuerungen. Es ist erst in den 80er Jahren gelungen, die Vielfalt der eingeführten Architekturprinzipien in Mikroprozessoren anzubieten (ein typisches Beispiel ist die Schaltkreisfamilie Motorola 68 000...68 040).

2.3.1.2. Architekturen mit reduzierten Befehlslisten (RISC)

Maschinen mit vergleichsweise elementaren Befehlslisten gibt es seit der Frühzeit der Rechentechnik. In den 70er Jahren begannen systematische Untersuchungen, wobei folgende Prinzipien im Zusammenhang betrachtet wurden:

1. das System ist ausschließlich auf die Programmierung in höheren Programmiersprachen orientiert
2. es wird ein elementarer Befehlssatz vorgesehen, der vollständig "hart verdrahtet" werden kann (kein Rückgriff auf Mikroprogrammsteuerung); Ziel ist, die meisten Befehle jeweils in einem Zyklus auszuführen
3. hochentwickelte Compiler zur praktischen Nutzung der Prinzipien 1 und 2.

Diese Forschungsrichtung wurde maßgeblich durch folgende Sachverhalte angeregt:

- Es wurden zunehmend höhere Programmiersprachen verwendet.
- Untersuchungen zur Nutzungshäufigkeit von Befehlen in einer Vielzahl kompilierter Programme haben ergeben, daß elementare Befehle weitaus am häufigsten benutzt werden.
- Mit dem Einsatz der Halbleiterspeichern konnten die Zykluszeiten von Speicher und Verarbeitungslogik zunehmend angeglichen werden (im besonderen wurden große Cache-Speicher realisierbar, die bei akzeptablen Trefferraten mit derselben Zykluszeit wie die Verarbeitungslogik betrieben werden konnten).
- Große Registerspeicher wurden kostengünstig realisierbar.¹

Die entscheidende Überlegung besteht darin²: Die elementaren Befehle (LOAD, STORE, ADD, COMPARE, BRANCH usw.) werden weitestgehend am häufigsten benutzt. Solche Befehls seien beispielsweise mit einer Schaltungstiefe $s = 10$ zu implementieren, wodurch sich eine entsprechend kurze Zykluszeit t_c verwirklichen läßt. Wird durch Hinzufügen komplexerer Befehle die Schaltungstiefe nur um 1 erhöht, so werden alle Abläufe um 10% langsamer. Die Nutzungshäufigkeit und Leistungsfähigkeit der komplexen Befeh-

¹ Auf demselben Schaltkreis wie die Verarbeitungslogik (oft 32 Register zu 32 bit; eine Architektur hat 192 Register).

² Die Darstellung folgt der "klassischen" Veröffentlichung von Radin (/283/). Eine weitere grundlegende Arbeit ist /273/.

le muß dann diesen Verlust überkompensieren und die zusätzlichen Kosten rechtfertigen.

2.3.1.3. Interne Parallelarbeit

Elementare Formen der internen Parallelarbeit sind gegeben durch die Überlappung verschiedener Phasen des Befehlsablaufs: Holen der nachfolgenden Befehle, Heranschaffen von Operanden, deren Verknüpfung, Abtransport der Resultate. Es wird also der innewohnende Parallelismus der Befehlsablaufsteuerung ausgenutzt.

Um den innewohnenden Parallelismus in Programmen zu nutzen, sind mehrere Operationswerke erforderlich, so daß Operationen, die nicht voneinander abhängen, gleichzeitig ausgeführt werden können.

Bei beiden Formen der Nutzung des internen Parallelismus ist es wesentlich, wie dieser erkannt und gesteuert wird.

Das ist zum einen ausschließlich mit schaltungstechnischen Vorkehrungen möglich. Befehle werden nacheinander automatisch vorbeugend gelesen, Operandenadressen berechnet usw. Sonderfälle, wie Verzweigungen, Schreiben auf Adressen, von denen bereits voreilend Operanden gelesen wurden usw. werden von besonderen Schaltmitteln erkannt und automatisch korrigiert.¹

Auf ähnliche Weise sind mehrere Verarbeitungswerke nutzbar.²

Ein neuerer Ansatz besteht darin, auf Schaltmittel zum Erkennen und Beheben von Konflikten zu verzichten und diese Aktivitäten dem Compiler zu übertragen. Schaltungstechnisch sind nur sehr elementare Formen der überlappenden Arbeitsweise vorgesehen (z. B. das voreilende Befehlslesen). Der Compiler kennt das Taktschema der Zielmaschine. Er muß potentielle Konflikte erkennen und durch Umstellen bzw. Ändern der Befehlsfolge auflösen. Bei Anordnung mehrerer Operationswerke wird dem Compiler deren Ausnutzung übertragen. Dazu sind die Befehlsformate so gestaltet, daß alle Werke gleichzeitig gesteuert werden können. Solche Befehle sind extrem lang (VERY LONG INSTRUCTION WORD VLIW).³ Der Vorteil des compiler-gestützten Ansatzes liegt zum einen darin, daß die Hardware einfach gehalten werden kann (hohe Geschwindigkeit durch geringe Schaltungstiefe), und zum anderen darin, daß man hoffen kann, durch Analyse des gesamten Programmtextes mehr Möglichkeiten zur Parallelarbeit zu erkennen als durch Auswertung des Befehlsstromes zur Laufzeit. Allerdings sind auch schwerwiegende Nachteile nicht zu übersehen:⁴

- solche extrem optimierenden Compiler werden langsam und unzuverlässig
- es ist nicht mehr möglich, programmkompatible Rechnerfamilien zu schaffen: jede Änderung etwa der Taktverhältnisse an der Befehlspipeline erfordert eine Neucompilierung, auch bei vollständig identischer Befehlsliste.

1 Eine konkrete Lösung ist in /40/ beschrieben.

2 Das wurde bereits bei CDC 6600 verwirklicht ("scoreboard").

3 Eine solche Maschine ist in /147/ beschrieben.

4 Darauf wird z. B. von Wirth in /330/ hingewiesen.

2.3.1.4. Architekturen mit Orientierung auf höhere Programmiersprachen

Mit zunehmender Nutzung höherer Programmiersprachen erschien es sinnvoll, Maschinenarchitekturen so auszubilden, daß sie direkt an bestimmte Sprachen angepaßt sind. Dazu hat es viele Vorschläge und auch einige ausgeführte Maschinen gegeben.¹ Solchen Lösungen ist bisher der entscheidende Durchbruch versagt geblieben. Im besonderen wurde das Leistungsvermögen bemängelt. Dazu einige Anmerkungen:

1. Das Preis- Leistungs- Verhältnis verschlechtert sich grundsätzlich, wenn Aktivitäten, die ein Compiler erledigen kann, zur Laufzeit von Schaltmitteln ausgeführt werden.

2.. Die Ausrichtung der Architektur auf ein einziges Sprachkonzept ist eine fast sicher Garantie für Erfolglosigkeit in wirtschaftlicher Hinsicht; zumindest ein beachtliches Risiko (Änderung der Sprachumgebung während der Entwicklung; Marktsituation im Fertigungszeitraum usw.).

3. Bei manchen Sprachkonzepten (z. B. Smalltalk) können bestimmte Aktivitäten nicht vom Compiler übernommen werden; sie sind grundsätzlich zur Laufzeit auszuführen. Dann ist natürlich eine schaltungstechnische Unterstützung von Vorteil.

4. Wenn der Compiler alle Aktivitäten erledigen muß (Typkontrolle, Variablenbindung usw.), kann bei hochentwickelten Programmiersprachen und großen Programmkomplexen die Compilierzeit untragbar werden; das hat z. B. einen führenden Anbieter von Ada- Systemen dazu veranlaßt, eigene Hardware (mit Vorkehrungen zur Laufzeit- Unterstützung) zu entwickeln.

5. Es geht grundsätzlich um die Frage, welche funktionelle Anforderung auf welcher Architekturebene (Hardware, Mikroprogramm, Laufzeitsystem usw.) am besten zu implementieren ist ("function to level mapping"). Um dafür eine Methodenlehre angeben zu können, stehen noch nicht genügend experimentelle Daten zur Verfügung (/146/).²

6. Um überzeugende Leistungsvorteile zu erzielen, ist es notwendig, unkonventionelle Schaltungsanordnungen vorzusehen, die gewisse Mindestaufwendungen erfordern.³ In /146/ wurde im einzelnen nachgewiesen, daß Leistungsverluste des iAPX 432 gegenüber herkömmlichen Architekturen ihre Ursache in unzureichender Hardwarestruktur bzw. Ressourcenausstattung haben. (Der iAPX 432 ist 4...20 mal langsamer als der 8086. Aber: er ist 10 mal schneller als ein softwareseitig implementiertes Laufzeitsystem.)

1 Beispiele u. a. in /79/, /86/, /88/, /89/, /186/.

2 Erklärlich, weil bisher die meisten Forschungsarbeiten auf Leistungsverbesserungen der eingeführten Architekturen, auf RISC- Konzepte und auf Parallelverarbeitung gerichtet waren.

3 D. h. die über den Aufwandsrahmen von Mikroprozessoren hinausgehen.

2.3.1.5. Universelle Emulatoren

Seit schnelle ladbare Mikroprogrammspeicher großer Kapazität verfügbar sind, gibt es Versuche, die Ebene der Mikroprogrammierung dem Anwender zugänglich zu machen, teils um auf einer Hardware unterschiedliche Befehlslisten implementieren zu können, teils um die Geschwindigkeitsvorteile von Mikroprogrammen gegenüber üblichen Maschinenprogrammen wirksamer zu nutzen.¹

Solche Maschinen eignen sich gut für Sonderanwendungen und dazu, existierende Software, die nicht ohne weiteres umgeschrieben werden kann, durch Emulation der jeweiligen Befehlsliste weiterhin abzuarbeiten.

Die Geschwindigkeitsvorteile beim Übergang vom Maschinenprogramm zum Mikroprogramm liegen erfahrungsgemäß bei 1:2 bis etwa 1:20 (nach /146/ 1:8...1:15). Wodurch kommen sie zustande?

1. Das Mikroprogramm nutzt unmittelbar die Hardware; ein Mikrobefehl wird - in den weitaus meisten Fällen - in einem Taktzyklus abgearbeitet.

2. Mit leistungsfähigen Mikrobefehlsformaten lassen sich Datenwege und Verknüpfungsschaltungen direkt steuern, so daß, wenn immer möglich, alle Schaltmittel parallel ausgenutzt werden können. Des weiteren kann man Mikroprogrammsteuerungen so auslegen, daß Verzweigungen (auch in mehrere Richtungen) keine zusätzlichen Zyklen erfordern.

Die Beziehungen zu RISC- bzw. VLIW-Konzepten sind offensichtlich. RISC-Befehle entsprechen den "vertikalen" Mikrobefehlsformaten (kurze Mikrobefehle, zumeist nur eine Wirkung pro Mikrobefehl); VLIW-Befehle entsprechen den "horizontalen" Mikrobefehlsformaten (lange Mikrobefehle mit mehreren parallelen Wirkungen).²

Zwischenzeitlich wird es offenbar beherrscht, aus Programmtexten in üblichen Programmiersprachen gute Mikroprogramme zu compilieren: - eine sehr notwendige Voraussetzung für die praktische Nutzung, denn bisher haben Anwender nur selten von der Möglichkeit Gebrauch gemacht, eigene Befehlslisten zu definieren und zeitkritische Abläufe direkt in Mikroprogramme umzusetzen.³

1 Beispiele: Emulator- und Assist-Vorkehrungen (Burroughs 1700, S/360, S/370, ESER); dem Nutzer zugängliche Mikroprogrammspeicher (PDP 11/70); anwenderseitig mikroprogrammierbare Maschinen (Interdata, MLP 9000, WISC (/254/)).

2 Praktische Unterschiede: Mikroprogramme werden in besonderen Speichern oder dem Nutzer unzugänglichen Speicherbereichen (z. B. bei 360/25) gehalten. Bei den meisten Maschinen sind die Mikrobefehlsformate nicht mit dem Ziel einer eigenständigen regulären Architektur, sondern einer kostengünstigen Emulation der Zielarchitektur ausgelegt worden.

3 Ein Programmsystem wird z. B. in /334/ beschrieben. Ohne Unterstützung beträgt die Produktivität oft nur einige hundert Mikrobefehle je Programmierer und Jahr.

2.3.2. Vektorrechner

Die konsequente Verwirklichung des Pipeline-Prinzips ermöglicht es, eine Vielzahl gleichartiger Operationen mit geringem Zeitversatz (jeweils 1 Taktzyklus) parallel mit denselben Schaltmitteln auszuführen.

Damit lassen sich sehr kurze Taktzyklen verwirklichen (Richtwert derzeit 4- 10 ns).¹

Erfahrungsgemäß lohnt es sich nicht, die Anzahl der Pipeline-Segmente über 6- 8 zu erhöhen; mehr als durchschnittlich 2 Funktionen lassen sich nicht sinnvoll nutzen (z. B. Multiplikation und Addition). Die Anlaufzeit der Pipeline ("vector start up overhead") hat maßgeblichen Einfluß auf die Leistung, und zwar um so mehr, je kürzer die Vektoren sind. So stehen dem Nutzer oft nur 5...15% der Maximalleistung zur Verfügung.²

2.3.3. Parallelverarbeitung

An Problemen der Parallelverarbeitung wird seit mehr als 20 Jahren gearbeitet. Die verschiedenen Ansätze lassen sich unterscheiden:

- nach der Ausgestaltung der einzelnen Verarbeitungseinheiten (Struktur, Umfang, Leistungsvermögen)
- nach der Größenordnung der Anzahl an zusammenwirkenden Verarbeitungseinheiten
- nach den Verbindungsprinzipien (z. B. Bussysteme, Gitterstrukturen mit Direktverbindungen zu den benachbarten Modulen, gemeinsame Speicher)
- nach den Steuerprinzipien des gesamten Systems (bekannteste Unterscheidung: SIMD und MIMD).

Dem Gegenstand der Arbeit gemäß ist von Bedeutung, wie leistungsfähig bzw. aufwendig die einzelne Verarbeitungseinrichtung gestaltet ist.

Lohnt es sich weiterhin, die Leistung des Einzelprozessors zu verbessern, oder sind Forderungen nach extremen Verarbeitungsleistungen eher durch Anordnungen aus einer sehr großen Anzahl vergleichsweise einfacher Verarbeitungseinrichtungen zu erfüllen?

Der zuletzt genannte Ansatz ist vorrangig in 2 Richtungen untersucht worden:

1. SIMD- Anordnungen aus einer sehr großen Zahl (4k...64k) von äußerster elementaren Verarbeitungseinrichtungen, die in den meisten Fällen bitseriell arbeiten
2. Anordnungen, die aus einer großen Zahl (128...4k) zueinander ("off the shelf"-) Mikroprozessoren aufgebaut sind.

¹ Beispiele sind die kommerziell verfügbaren Supercomputer und Vektorprozessoren verschiedener Hersteller (vgl. u. a.: /137/, /163/, /192/, /256/, /290/).

² Die Darstellung folgt /212/; vgl auch Tafel 4 (S. 50).

Die Massenanwendung beider Ansätze ist bisher ausgeblieben, und es gibt Gründe dafür, daß sich diese Entwicklungsrichtungen auch in Zukunft nicht durchsetzen werden, zumindest was den Bereich der Universalrechner angeht.¹

Die Ausführungszeit vieler elementarer Algorithmen (dazu gehört z. B. die Addition) läßt sich durch Parallelisierung in n Verarbeitungseinrichtungen günstigstenfalls in der Größenordnung $O(\log n)$ verringern.² Selbst wenn es gelingt, einen komplexen Algorithmus zu parallelisieren, wird die Gesamtlaufzeit durch die vergleichsweise langsame Bearbeitung der Teilalgorithmen bestimmt. Zudem ist eine Vielzahl von Synchronisations- und Kommunikationsabläufen notwendig, die die Ausführungszeit weiter verlängern. Sollte auch das alles beherrscht werden: die Hardware wird aus sehr vielen Komponenten bestehen; das bringt elementare technische Schwierigkeiten mit sich (Zuverlässigkeit, Strombedarf, Kühlung, Fehlersuche usw.). Der Vorteil zuhandener Schaltmittel³ verschwindet also, wenn sehr viele davon einzusetzen sind; auch wird die einfache betriebswirtschaftliche Rechnung zeigen, daß - bei Fertigung in größeren Stückzahlen - die Entwicklungsaufwendungen für leistungsfähigere Hardwarekomplexe bei weitem aufgewogen werden durch die geringere Anzahl der Funktionseinheiten, die für ein gewünschtes Leistungsvermögen vorgesehen werden müssen.

Im Gegensatz dazu sind Parallelverarbeitungssysteme mit einer mittleren Anzahl an Verarbeitungseinrichtungen sowohl von der technischen als auch von der algorithmischen Seite aus wesentlich besser beherrschbar. Es hat sich gezeigt, daß die Schwierigkeiten der Programmierung bisher überschätzt und die Vielfalt der Anwendungsmöglichkeiten bisher unterschätzt wurden. Für manche Probleme hat es sich erwiesen, daß es einfacher ist, eine parallele Formulierung zu finden als eine sequentielle. Tafel 2 gibt einen Überblick über die Entwicklung solcher Parallelverarbeitungssysteme in den nächsten Jahren.⁴

1 Die Darstellung folgt /111/. Für einen Überblick s. u. a. /3/, /8/, /58/, /201/. Alle maßgeblichen Autoren verweisen darauf, daß Versuche mit voll ausgebauten Maschinen notwendig sind, um die Grundfragen entscheiden zu können (zu den Aufwendungen vgl. etwa /117/, /275/).

2 Dazu gibt es viele Untersuchungen. Für arithmetische Operationen vgl. etwa /91/. Anregung: für Elementaroperationen dürfte sich die Parallelisierbarkeit anhand der Booleschen Gleichungen, die die Informationswandlungen beschreiben, exakt untersuchen lassen (Ausbau des Ansatzes von /243/).

3 Die üblichen Mikroprozessoren sind nicht ausdrücklich für Parallelverarbeitungssysteme entworfen worden. Für einen solchen Einsatzfall sind andere Auslegungen erforderlich (spezifische Kompromisse bei der Nutzung der Siliziumfläche für Verarbeitungs-, Speicher-, Speicheranschluß- und Kommunikationshardware). Ein bekanntes Beispiel für einen auf Einsatz in Parallelverarbeitungssystemen optimierten Schaltkreis ist der "Transputer" T 800 (/162/).

4 Die Darstellung (einschließlich Tafel 2) folgt /111/. Beispiele für solche Systeme: Suprenum, GF 11, RP 3.

Generation	1. 1983...87	2. 1988...92	3. 1993...97
T y p i s c h e r K n o t e n			
Leistung in MIPS	1	10	100
MFLOPS skalar	0,1	2	40
MFLOPS vektoriell	10	40	200
Speicher (MBytes)	0,5	4	32
T y p i s c h e s S y s t e m			
Knoten	64	256	1024
Leistung in MIPS	64	2560	100 K
MFLOPS skalar	6,4	512	40 K
MFLOPS vektoriell	640	10 K	200 K
Speicher (MBytes)	32	1024	32 K
Latenzzeit für Nachricht aus 100 Bytes			
a) zum Nachbarn (μ s)	2000	5	0,5
b) nicht lokal (μ s)	6000	5	0,5

Tafel 2

Übersicht: Parallelverarbeitungssysteme für mittleren Parallelisierungsgrad ("medium grain")

2.3.4. Datenflußmaschinen

Die Prinzipien der Datenflußsteuerung sind bisher vorwiegend theoretisch bzw. in Form von Versuchsmaschinen bearbeitet worden. Neuerdings werden solche Prinzipien in Mikroprozessoren zur Signalverarbeitung angewendet.¹ Seit längerem ist bekannt, einen Befehlsstrom zur Laufzeit in Angaben zur Datenflußsteuerung umzuschlüsseln, um mehrere Verarbeitungswerke parallel betreiben zu können.² Auch sind gewisse Prinzipien der Datenflußsteuerung in Maschinen mit langen Befehlsworten (VLIW) angewendet worden.³

Ein Grund für das Ausbleiben von Erfolgen in großem Maßstab besteht darin, daß von Neumann-Rechner sequentielle Programme recht effizient abarbeiten können und dazu wesentlich weniger Speicherbandbreite brauchen als vergleichbare Datenflußmaschinen.⁴

2.3.5. Datenstruktur- bzw. Spezialmaschinen

Ehe digitale Universalrechner kostengünstig verfügbar waren, war es geradezu selbstverständlich, für jede Klasse von Aufgaben der Informationsverarbeitung (im weitesten Sinne) spezifische Einrichtungen zu entwerfen. Dazu wurden zuhandene technische Mittel der vielfältigsten Art (mechanische, hydraulische, elektrische usw.) zweckgerichtet kombiniert.

Heutzutage denkt man eher an ein Programm für einen Universalrechner als an eine zweckgebundene technische Sonderlösung.

Um so überraschender sind die Ergebnisse, wenn man den altbewährten Ansatz auf die modernen technologischen Grundlagen überträgt - schließlich hat man mit Gattern, Flipflops und RAMs noch weit mehr Freizügigkeit als mit Schaltwalzen, Zahnrädern und Hydraulikzylindern -: ist ein einzelner, genau abgegrenzter Komplex von Algorithmen und Datenstrukturen gegeben, so bereitet es oft keine grundsätzlichen Schwierigkeiten, dafür spezielle Schaltungsanordnungen auszuarbeiten, die ohne weiteres technisch realisierbar sind und jeden Universalrechner im Leistungsvermögen übertreffen. Die Bezeichnung "Datenstrukturmaschine" soll genau dies zum Ausdruck bringen: die zweckgerechte Nutzung von Schaltmitteln, um bestimmte Operationen über bestimmte Datenstrukturen auszuführen.⁵

Probeentwürfe haben gezeigt, daß Beschleunigungsfaktoren von 100...>2000 gegenüber technologisch vergleichbaren Universalrechnern mittlerer Leistung ohne weiteres mit üblichen Schaltmitteln (TTL, CMOS) und beherrschbaren Aufwendungen (100...4000 Schaltkreise) zu erreichen sind.⁶

1 Beispiel: NEC μ PD 7281 (/306/).

2 Z. B. verwirklicht in 360/91 (/76/).

3 Z. B. die "directed dataflow architecture" (/284/).

4 Die Darlegung folgt /108/; neuere Ergebnisse, die Verbesserungen bringen sollen (/271/), sind noch nicht zugänglich.

5 Die Bezeichnung "Datenstrukturmaschine" geht auf Giloi zurück (/61/, /176/).

6 Die einschlägige Erfahrungen sind in /243/ zusammengefaßt.

Die Erfahrung wird von anderen Autoren bestätigt. So wird in /176/ angemerkt, daß Datenstrukturarchitekturen den Datenflußarchitekturen an Effizienz und Kosteneffektivität deutlich überlegen sind, da die explizite Parallelität in den Algorithmen viel wirksamer technisch genutzt werden kann.

Es ist deshalb naheliegend, für anwendungspraktisch bedeutsame Algorithmenkomplexe solche Maschinen zu entwerfen und diese mit einem Universalrechner zusammenzuschalten, beispielsweise durch Anschluß an ein universelles Bussystem.

Es hat sich aber gezeigt, daß die Transporte zwischen den einzelnen Verarbeitungseinrichtungen die Leistung begrenzen, wenn die wesentlichen Verarbeitungsalgorithmen erst einmal beschleunigt sind.¹ Man stelle sich beispielsweise vor, es seien Relationen zu durchmustern (in einem speziellen Datenbasisprozessor), und die gefundenen Tupel enthielten numerische Angaben zur Verarbeitung in einem Numerikprozessor. Dann sind fortlaufend Transporte zwischen den beiden Spezialprozessoren notwendig, die zudem vom Universalrechner (d. h. vom übergeordneten Anwenderprogramm) gesteuert werden müssen.

Somit ist zu prüfen, ob eine Sammlung von Spezialmaschinen in Verbindung mit Universalrechnern mittleren Leistungsvermögens grundsätzlich besser ist als ein universeller Hochleistungsrechner, namentlich dann, wenn Operationen über umfangreiche, komplizierte und vielfältige Datenstrukturen im Verbund auszuführen sind.

Dabei darf nicht nur die Geschwindigkeit gesehen werden: die meisten der Algorithmen, die für eine hardwareseitige Beschleunigung in Frage kommen, sind in sich nicht trivial, und sie sind häufig in umfangreiche Programmkomplexe eingebettet. Die Schnittstellen zur speziellen Hardware werden von den eingeführten Programmiersprachen nur unvollkommen unterstützt (Einfügen von Maschinencode, Assembler-Unterprogramme). Bei Wechsel der Hardware-Konfiguration sind diese recht aufwendigen Arbeiten stets von neuem erforderlich.²

Daraus ergibt sich, daß es zweckmäßig ist, zunächst das Leistungsvermögen von Universalrechnern weiter zu verbessern, und es erweist sich als naheliegend, dafür nach Schaltungslösungen zu suchen, die bei universeller Nutzbarkeit ähnlich leistungsfähig sind wie Datenstrukturmaschinen für wichtige Anwendungsgebiete.

¹ Das ist in /243/ mit vielen Zahlenbeispielen belegt.

² Es geht hier grundsätzlich um die Zukunftssicherheit der Spezialhardware, um die Übertragbarkeit der Anwendungsprogrammkomplexe auf Nachfolgesysteme. Dieser Gesichtspunkt wird z. B. in /310/ kritisch gewertet.

2.4. Die eigene Arbeitsrichtung

2.4.1. Vorhaben und Methodik

Der Universalrechner, dessen Prinzipien auf Babbage, Zuse und v. Neumann zurückgehen, konnte bisher nicht von anderen informationsverarbeitenden Einrichtungen verdrängt werden. Er wird noch viele Jahre seine überragende Bedeutung behalten: grundsätzlich andere Lösungen, die ihn ersetzen können, sind in absehbarer Zeit nicht zu erwarten, und bereits die derzeitigen Investitionen in Anwendungslösungen sind so hoch, daß die weitere massenhafte Nutzung einfach eine Folge wirtschaftlicher Zwänge ist.¹

Die Architektur von Universalrechnern ist weiterhin Gegenstand umfassender Forschungsarbeiten, die sowohl den Einzelprozessor als auch Parallelverarbeitungssysteme aller Art betreffen.

Wesentliche Triebkräfte dafür bestehen in den Forderungen der Anwender nach Leistungssteigerung und Kostensenkung, in der Verfügbarkeit leistungsfähiger Technologien, in der Nutzbarkeit einer umfassenden Erfahrungsbasis und darin, daß auf Grund der wirtschaftlichen Bedeutung Mittel für Forschung und Entwicklung vergleichsweise problemlos bereitgestellt werden.² Es ist nach wie vor entscheidend, die Verarbeitungsgeschwindigkeit zu erhöhen. Rechner, bei denen auf Kosten der reinen Geschwindigkeit der Befehlsausführung ("low level performance") andere Gebrauchseigenschaften (Unterstützung bestimmter Programmiersprachen, gute Compiler, Zuverlässigkeit usw.) vorrangig weiterentwickelt wurden, konnten sich, Sonderanwendungen ausgenommen, nicht durchsetzen. Auch künftig werden solche Eigenschaften kein Ersatz für unzulängliche Verarbeitungsgeschwindigkeit sein.³

Die eigene Arbeitsrichtung betrifft daher folgendes Ziel: leistungsentscheidende Wirkprinzipien und Strukturen aufzufinden und zu studieren, sowie Richtlinien, Aufgabenstellungen und Bewertungskriterien für die Gestaltung leistungsoptimierter Universalrechner anzugeben. Die Vorgehensweise ist eine gleichsam experimentelle⁴: durch konstruktives Handeln, durch Ausarbeiten neuer Vorschläge soll versucht werden, bedeutsame Verbesserungen zu erreichen.

Dabei ist es ein wichtiges methodisches Prinzip, den Fragen der "Kompatibilität" auf der Ebene der Maschinenbefehle nicht jenen Rang einzuräumen, den sie derzeit in der Praxis innehaben. Nur so ist herauszufinden, welche absoluten Verbesserungen noch zu erwarten sind. Sind solche Verbesserungen hinreichend bedeutsam, beispielsweise eine Verzehnfachung des Durch-

1 Derzeit sind z. B. weltweit 20 Millionen Personalcomputer im Einsatz, und es wird abgeschätzt, daß der Markt noch weitere 80 Millionen Stück aufnehmen kann.

2 Dongarra nennt im Vorwort zu /9/ folgende Sachverhalte, die neue Architekturentwicklungen anregen: Verfügbarkeit von leistungsfähigen Mikroprozessoren, standardisierten Bussystemen, gate-array-Technologien und von Risikokapital.

3 Die Darstellung folgt /146/; sie wird durch die Verkaufsstatistiken, Geschäftsberichte usw. bestätigt.

4 Vgl. Abschnitt 2.1. (S. 7).

satzes im Rahmen bestimmter Aufwendungen, so muß man darüber nachdenken, sie in die Praxis einzuführen; sind sie es nicht, so ist es recht wahrscheinlich, daß der Stand der Technik einem Optimum bereits sehr nahe kommt, so daß die nächsten Ziele von Forschung und Entwicklung eher in Verfeinerungen, Verbesserungen der Technologie usw. zu sehen sind. Mit diesen Überlegungen läßt sich der Gegenstand der Arbeit folgendermaßen eingrenzen:

1. Es geht um den einzelnen programmierbaren Universalrechner. Sein Anwendungsgebiet sind Algorithmen aller Art, also alle Aufgabe der Informationsverarbeitung, die sich letztlich auf berechenbare Funktionen zurückführen lassen, vorzugsweise auf Operationen über binär codierte numerische und nichtnumerische Informationsstrukturen unter Einbeziehung des logischen Schließens.

2. Es sollen Überlegungen zu technisch- ökonomisch leistungs-optimalen Universalrechnern angestellt werden; Ziel ist das höchste Leistungsvermögen in der jeweiligen Größenklasse (vom Mikrocontroller bis zum Supercomputer).

3. Hier werden - ohne Einschränkung der Allgemeinheit - jene Größenordnungen bevorzugt betrachtet (aus naheliegenden Gründen der Nutzbarkeit), die für die Breitenanwendung in Frage kommen: vom Mikrocontroller bis zum Hochleistungs- OEM- Rechner, der beispielsweise als Maschinensteuerung, als "workstation", aber auch als Verarbeitungsmodul in Supercomputern¹ nutzbar ist.

4. Dem Vorhaben liegt das Prinzip der Ingenieurarbeit zugrunde, Bewährtes und Neues im Hinblick auf ein optimales Ergebnis zu vereinen und zuhandene technische Mittel für einen bestimmten Zweck in bestmöglicher Weise zu nutzen. Als zuhanden werden angesehen²:

- die Prinzipien der binären Informationsverarbeitung als technischer Umsetzung der Aussagenlogik
- die Technik der integrierten Schaltkreise bis hin zur "wafer scale integration" (WSI)
- der Stand der Technik hinsichtlich der konstruktiven Gestaltung von Digitalrechnern (mechanischer Aufbau, Kühlung, periphere Einrichtungen usw.)
- die bekannten Verfahren, Prinzipien und Algorithmen der numerischen und nichtnumerischen Informationsverarbeitung.

Das heißt, von den Grundlagen her wird die bisher bewährte Hauptrichtung der Entwicklung weiter verfolgt: keine optische Informationsverarbeitung, keine Wahrscheinlichkeitslogik, keine Schwellwertelemente, keine neuronalen Strukturen.

¹ Das betrifft Supercomputer, die als Parallelverarbeitungssysteme aus vergleichsweise leistungsfähigen Modulen aufgebaut sind, also Strukturen ähnlich Suprenum, RP 3 u. a.

² Mit Blick auf die Zeit der Einführung und Nutzung.

Die Arbeit gilt ausschließlich dem einzelnen Rechner, dessen struktureller Vervollkommnung. Fragen der Parallelverarbeitung und der Nutzung extrem leistungsfähiger Technologien werden damit keineswegs gegenstandslos; aus der Sicht der vorliegenden Arbeit sind sie vielmehr Gegenstand künftiger Aktivitäten: erst wird der Einzelprozessor strukturell auf höchstes Leistungsvermögen gebracht, bevor man sich damit befaßt, eine Vielzahl davon zusammenzuschalten¹ oder sehr kostenaufwendige Technologien einzusetzen. Das allgemeine Ziel soll durch folgenden methodischen Ansatz erreicht werden:

1. Für bedeutsame Anwendungsgebiete der Rechentechnik werden grundsätzliche und leistungsbestimmende Abstraktionen (Datentypen + Operationen + Ablaufprinzipien) gesucht. Diese werden exakt beschrieben.
2. Es werden Hardwarestrukturen entwickelt, um diese Abstraktionen so effektiv wie möglich implementieren zu können.
3. Auf Grundlage dieser Strukturen werden Rechnerarchitekturen definiert, die die erforderliche Universalität gewährleisten.
4. Der Auswahl der Abstraktionen und der Ausgestaltung der Schaltungslösungen werden die Tiefenstrukturen der jeweiligen Prozesse der Informationsverarbeitung zugrunde gelegt und nicht die Oberflächenstrukturen, wie sie durch konkrete Programmiersprachen, Betriebssystem-Umgebungen, Standards für die Datendarstellung usw. gegeben sind. Es wird also versucht, das Wesen der Verarbeitungsprozesse zu erfassen und die technischen Lösungen hinreichend allgemeingültig auszulegen, das heißt eine einseitige Orientierung, z. B. auf ein bestimmtes Sprachkonzept, zu vermeiden.
5. Für jede Abstraktion wird versucht, die Schaltmittel so leistungsfähig wie möglich auszulegen. Dafür wird - im Rahmen der technischen Beherrschbarkeit - Hardware ohne Rücksicht auf übliche Gepflogenheiten oder Konventionen eingesetzt; sowohl hinsichtlich der Aufwendungen als auch der strukturellen Gestaltung.²
6. Allen technischen Lösungsvorschlägen, vom einzelnen Schaltungskomplex bis zur Architekturdefinition, wird der Stand der Technologie zugrunde gelegt, der zu Beginn ihrer Nutzung erwartet werden kann. Bis zur Einführung einer neuen Architektur sind etwa 4 Jahre zu rechnen³; erfolgreiche Architekturen haben eine Nutzungsdauer von mehr als 20 Jahren. Derzeit werden auf Schaltkreisen von etwa $10 \times 15 \text{ mm}^2$ mit über 10^6

¹ Für die umfassende Bearbeitung dieses Problems sind unbedingt Ergebnisse der laufenden (sehr aufwendigen) Versuche (Suprenum, GF 11, RP 3 u. a.) auszuwerten.

² Die meisten der eingeführten Architekturkonzepte wurden unter der Bedingung grundsätzlich knapper Hardware-Ressourcen (Speicher, Verarbeitungswerke usw.) ausgearbeitet.

³ Vgl. einschlägige Erfahrungsberichte, z. B. in /4/, /7/, /19/, /113/, /150/, /200/.

Transistoren sehr leistungsfähige Einzelprozessoren verwirklicht.¹ Die Verarbeitungsleistung wird vor allem deshalb erreicht, weil ein ingenieurmäßig sinnvoller Kompromiß bekannter Konzepte (Cache, RISC-Prinzipien, parallele Verarbeitungswerke) auf einem einzigen Schaltkreis untergebracht werden kann, wodurch sehr kurze Zykluszeiten realisierbar sind (Taktfrequenzen um 40 MHz). Will man diesen Stand der Technik übertreffen, sind neue Konzepte notwendig, die von Grund auf die Möglichkeiten der Technologie nutzen, um dem Anwender überlegene Leistungen² bereitzustellen (Realisierungsbasis: Anordnungen aus mehreren derart hochintegrierten Schaltkreisen³, WSI-Technologien). Wenn man Forschungen zu neuen Architekturen nicht nur des Erkenntnisgewinns willen betreibt, sondern auf Erfolge präbendiert, ist grundsätzlich zu vermeiden, daß zeitweilige technologische Einschränkungen Beschränkungen bei den Architekturkonzepten zur Folge haben.⁴

Allein der Umfang des Vorhabens schließt eine umfassende Behandlung in der vorliegenden Arbeit aus. Hier geht es vielmehr darum, die Grundlagen zu erörtern, Bewertungskriterien aufzustellen und Methoden vorzuschlagen. Das wird beispielhaft vorgeführt, indem verbreitete und bekannte Maschinenarchitekturen und Programmiersprachen als Erfahrungsbasis für das Auffinden von Abstraktionen verwendet werden.

Rechnerarchitekturen und Schaltungslösungen auf dieser Grundlage werden noch der weiteren Vervollkommnung bedürfen; sie werden aber aufzeigen, in welcher Größenordnung die Leistungsvorteile liegen können, die von künftigen optimierten Rechnerstrukturen zu erwarten sind.

1 Vgl. Intel 80860 (/325/, /327/), 80486 (/328/), Motorola 68040 (/329/).

2 Als weiteres Ziel verbleibt die Verbesserung des Preis-Leistungs-Verhältnisses, wo Erfolge auch mit deutlich geringeren Anforderungen an die Technologie demonstriert werden können (z. B. bei Mikrocontrollern).

3 Das erfordert sicherlich neue Vorstellungen zur Funktionsaufteilung, die über den Stand der Technik (vgl. etwa /105/) hinausgehen.

4 Für die ersten Hardwaremodelle sind dann eher Mehraufwendungen (z. B. Realisierung mit mehreren Standardzellen-Schaltkreisen) und gewisse Leistungsverluste (durch niedrigere Taktfrequenzen; ggf. auch durch mikroprogrammtechnische Emulation) in Kauf zu nehmen. Der Nutzer wird eine neue Architektur nur dann akzeptieren, wenn die Umstellungsaufwendungen nicht ins Gewicht fallen oder durch perspektivische Aussichten (Leistungsverbesserung, Zukunftssicherheit, künftige weite Verbreitung) gerechtfertigt werden. Vergleichsweise geringfügige Verbesserungen, die Umstellungen erfordern, haben gegenüber dem Stand der Technik kaum eine Chance.

2.4.2. Einordnung in den Stand von Wissenschaft und Technik

Die vorgeschlagene Arbeitsrichtung geht von folgenden Ansätzen aus:

1. Die Optimierung der Maschinenbefehle. Der RISC- Ansatz ist in Betracht zu ziehen, soll aber durch andere Überlegungen, beispielsweise zur bestmöglichen Ausnutzung von Datenwegen, ergänzt werden. Die Übergänge zwischen RISC und Konzepten der Mikroprogrammsteuerung sind fließend und sollen näher untersucht werden (RISC- Befehle lassen sich als "vertikale" Mikrobefehle auffassen, die im allgemeinen Speicheradressenraum abgelegt sind).

Auch sind Konzepte zur schaltungstechnischen Unterstützung höherer Programmiersprachen (HLL- Architekturen) von neuem zu durchdenken (Nutzung der zwischenzeitlich gewonnenen Erfahrungen und der Möglichkeiten der modernen Schaltungstechnik). Wesentlich ist, daß die Untersuchung nicht vordergründig auf statistische Analysen gegebener Programme gestützt wird, sondern auf eine weitgehend analytische Betrachtung der leistungsbestimmenden Sachverhalte.¹

2. Die Nutzung des innewohnenden Parallelismus in üblichen Programmen. Die Konzepte der Anordnung und Steuerung mehrerer Operationswerke ("scoreboard"- Prinzip, VLIW, "horizontale" Mikrobefehle, Datenflußprinzipien) sind dafür in Betracht zu ziehen.

3. Der Entwurf von Spezialmaschinen. Es ist wirtschaftlich nicht durchführbar, für jeden anwendungspraktisch bedeutsamen Algorithmenkomplex eine Sondermaschine bereitzustellen, die leistungsmäßige Überlegenheit zweckgerichtet entworfener Sonderschaltungen ist aber beeindruckend: es lohnt sich also zu untersuchen, wie solche Schaltmittel in die Struktur eines Universalrechners eingefügt werden können.

4. Die gleichsam ganzheitliche Betrachtungsweise, die verschiedene Ebenen im Zusammenhang untersucht: von der Anweisung in einer höheren Programmiersprache über die compilierte Befehlsfolge bis zum Ablauf in der Hardware. Dieses Vorgehen wird beispielsweise in /233/ als notwendig angesehen, um das Leistungsvermögen von Rechnerstrukturen deutlich zu verbessern. Hier wird zunächst auf eine Erfahrungsbasis Bezug genommen, die durch eingeführte Programmiersprachen, Maschinenarchitekturen und typische Schaltungslösungen gegeben ist.

¹ Auf Basis von Messungen und statistischen Auswertungen allein können lediglich bestehende Konzepte in sich optimiert, aber keine neuen gefunden werden (abgesehen davon, daß aus solchen Ergebnissen Ziele für erfinderisches Handeln ersichtlich bzw. ableitbar sind).

3. Das elementare Modell der sequentiellen Informationsverarbeitung

3.1. Von der Verarbeitungsschaltung zum Universalrechner

Im folgenden geht es ausschließlich um die Verarbeitung binär codierter Information, die in Speichermitteln mit wahlfreiem Zugriff gespeichert ist. Dabei entstehen binär codierte Ergebnisse; diese werden ebenfalls in Speichermitteln mit wahlfreiem Zugriff abgelegt. (Der Stand der Technik erlaubt es, Fragen der Ein- und Ausgabe als grundsätzlich lösbar anzusehen, beispielsweise durch Anschluß der Speichermittel an ein universelles Bussystem, so daß übliche Mikrorechner- Baugruppen für die E/A- Funktionen nutzbar sind.)

Die Ergebnisse werden stets durch Anwendung von Wandlungsvorschriften mit eindeutiger Wirkung in diskreten Zeitschritten (Taktzyklen) gebildet; aus gleichen Angaben entstehen durch Anwendung gleicher Wandlungsvorschriften stets gleiche Resultate. Jede solche Wandlungsvorschrift heißt im folgenden Algorithmus.¹ Ein Algorithmus kann beispielsweise implementiert werden durch einen Programmkomplex, ein einzelnes Programm, ein Unterprogramm, einen Maschinenbefehl, eine Schaltungsanordnung usw. Das Ziel besteht darin, universell nutzbare und überdurchschnittlich leistungsfähige Schaltungsanordnungen im Zusammenhang mit den ihnen zugeordneten Informationsstrukturen zweckgerichtet auszubilden. Deshalb werden hier lediglich Beziehungen zwischen Algorithmen und technischen Mitteln zu deren Implementierung untersucht; das Gebiet der Programmierung bleibt außer Betracht. Um Klarheit über grundlegende technische Sachverhalte und Zusammenhänge zu gewinnen, soll, mit dem einfachsten Fall beginnend, beschrieben werden, wie einzelne Verarbeitungsschaltungen systematisch in Universalrechnerstrukturen überführt werden können. Der einfachste Fall betrifft die Implementierung eines einzigen Algorithmus, der aus n Argumenten m Resultate erzeugt.² Das grundsätzliche Schema zeigt Bild 4. Es sind folgende technische Mittel vorge-
sehen:

1. Speichermittel für Argumente und Resultate
2. Verknüpfungsschaltungen
3. Verbindungen zwischen 1. und 2.

Dieses Schema hat keinerlei Erkenntniswert, wenn zugelassen wird, daß die Verknüpfungsschaltungen in beliebiger Weise ausgestaltet sein können. Im folgenden sind Verknüpfungsschaltungen deshalb ausschließlich rückwirkungsfreie kombinatorische Schaltungen bzw. Zuordner, die aus allen Argumenten in einem einzigen Zeitintervall alle Resultate bilden.³

1 Zur Algorithmentheorie vgl. etwa /46/, /47/, /74/, /274/.

2 In den Bildern ist der Einfachheit halber - ohne Beschränkung der Allgemeinheit - $n = 2$ und $m = 1$.

3 Solche Überlegungen sind bereits in /243/ zu finden; allerdings ausdrücklich auf Spezialmaschinen beschränkt. Demgegenüber wird hier - zwecks Kenntniskennntnisgewinn - zunächst die bekannte v. Neumann- Struktur angestrebt.

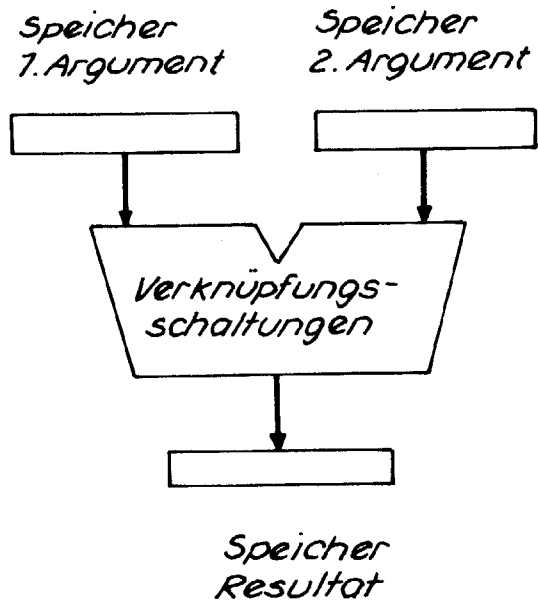


Bild 4

Allgemeines Schema der unmittelbaren Ausführung eines Algorithmus

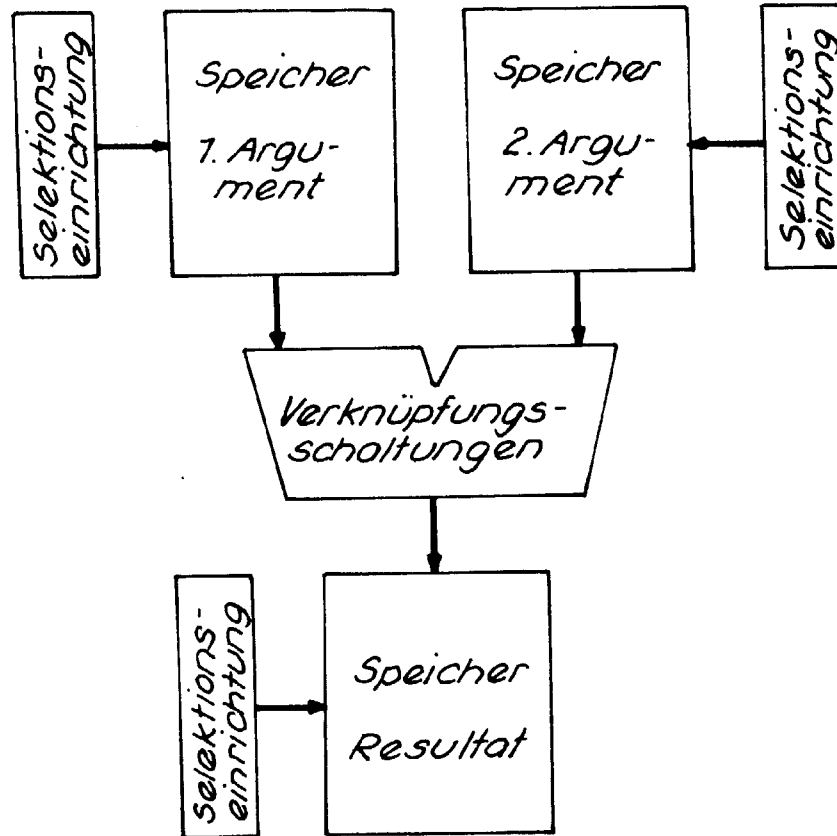


Bild 5

Abschnittsweise unmittelbare Ausführung eines Algorithmus

Dieses Zeitintervall entspricht einem Taktzyklus vom Auslesen der Argumentspeicher bis zum Laden der Resultatspeicher. Eine solche Implementierung eines Algorithmus ist in Bezug auf die Ausführungsgeschwindigkeit die wünschbarste überhaupt¹, und diese Vorstellung wird den weiteren Betrachtungen gleichsam als Richtbild zugrunde gelegt, - es ist aber klar, daß der technischen Durchführbarkeit der unmittelbaren Zuordnung enge Grenzen gesetzt sind.²

Algorithmen, die nicht für die unmittelbare Zuordnung geeignet sind, müssen gewissermaßen stückweise in mehreren Taktzyklen ausgeführt werden. Die Informationswandlungen in diesen Zyklen werden hier allgemein als Aktionen bezeichnet. Den Verknüpfungsschaltungen werden in den einzelnen Zyklen Teile der Argumente zugeführt, und es werden Teile der Resultate gebildet. Aktionen, die die jeweils benötigten Abschnitte der Argumente auswählen bzw. Abschnitte von Resultaten entsprechend einordnen, heißen Selektionen; Aktionen, die Resultat-Abschnitte erzeugen, heißen Operationen. Bild 5 zeigt die Modifikation des Schemas von Bild 4. Alle Speichermittel sind zur abschnittswisen Speicherung der Argumente bzw. Resultate ausgebildet. Sie sind an weitere Schaltmittel (Selektionseinrichtungen) angeschlossen, so daß in jedem Taktzyklus ein Zugriff zu einem Abschnitt in jeder Speichereinrichtung möglich ist. Dieses Schema ist nicht für alle Algorithmen anwendbar, sondern nur für solche, die unter 2 Einschränkungen implementiert werden können:

1. In allen Taktzyklen ist nur eine einzige Operation auszuführen.
2. Es ist nicht notwendig, Teile von Resultaten in Operationen einzubeziehen (keine Rückführung von Resultaten).

Um die erste Einschränkung aufzuheben, müssen die Verknüpfungsschaltungen gemäß Bild 6 so erweitert werden, daß Schaltmittel für mehrere Operationen verfügbar sind. Soll in einem bestimmten Taktzyklus eine bestimmte Operation ausgeführt werden, wird das betreffende Operationswerk aktiviert. Aktionen, die Operationen auswählen, heißen Aktivierungen. In technischer Hinsicht ist dazu notwendig, Auswahlsteuerleitungen für die Resultate vorzusehen und deren Auswahlsteuerleitungen an eine Ablaufsteuerung anzuschließen. Diese besteht im Beispiel aus einem Zähler, der die Nummer des aktuellen Zyklus liefert und aus einem nachgeschalteten Zuordner, der die Auswahlsteuerleitungen erregt.

- 1 Der besagte Zyklus muß natürlich hinreichend kurz sein; aus praktischen Erwägungen heraus sind etwa 2 μ s als Obergrenze anzusetzen: damit ist die Zuordnung oft schneller als die sequentielle Verarbeitung, und es ist technisch auch mit umfangreichen Zuordnungsschaltungen problemlos erreichbar (Beispiel: Tabelle der Winkelfunktionen in ROM- oder DRAM-Speichern mit nachgeordnetem Interpolationsnetzwerk).
- 2 Bei vergleichsweise wenigen Argumentbits spielt die Kompliziertheit keine Rolle (ROM bzw. RAM als Zuordner); ansonsten müssen sich die Verknüpfungen mit Gatternetzwerken verwirklichen lassen (Näheres s. S. 52 ff.).

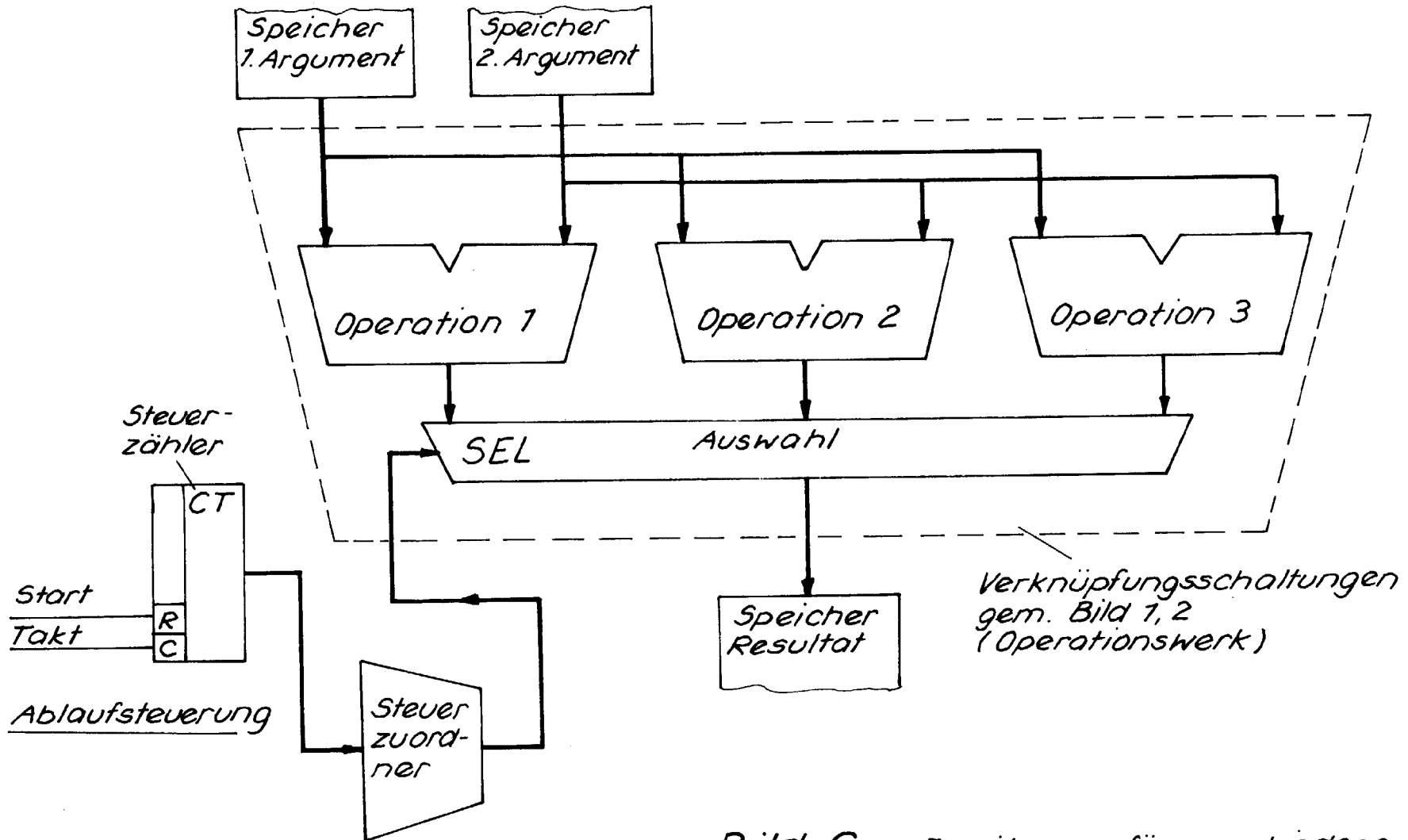


Bild 6 Erweiterung für verschiedene Operationen

Die zweite Einschränkung läßt sich beseitigen durch gemeinsame Speichermittel für alle Angaben des Algorithmus: Übergang zur Registermaschine nach Bild 7. Es werden zusätzliche Taktzyklen benötigt, da die Abschnitte von Argumenten und Resultaten nur nacheinander selektiert werden können; die Register sind notwendig, um die selektierten Abschnitte zu halten. Diese Anordnung ist allerdings nur für Algorithmen geeignet, die nicht erfordern, in bestimmten Zyklen zwischen verschiedenen Selektionen bzw. Operationen zu wählen.¹ Um diese Einschränkung aufzuheben, sind die Steuerschaltungen zu einem Steuerautomaten nach Bild 8 weiterzubilden, der zusätzlich über Bedingungsleitungen an Teile der Ausgänge von Operationswerken und Selektionseinrichtungen angeschlossen ist. Von hier aus erfordert es nur die folgenden Schritte, um zur wirklichen Universalmaschine zu kommen:

1. Anordnung eines ladbaren Speichers im Steuerautomaten (Bild 9 zeigt ein Ausführungsbeispiel).
2. Gestaltung der Operationswerke und Selektionseinrichtungen derart, daß anstelle der speziellen Verknüpfungen für den jeweiligen Algorithmus elementare, allgemein nutzbare Verknüpfungen² für Operationen und Selektionen vorgesehen werden.
3. Anordnung eines einzigen gemeinsamen Speichers für Argumente, Resultate und Steuerinformation (Bild 10). Damit laufen alle Aktionen in mehreren aufeinanderfolgenden Zyklen ab (Lesen der Steuerinformation -> Lesen der Argumente -> Verknüpfung -> Schreiben der Resultate), und es müssen Register vorgesehen werden, um die in den einzelnen Zyklen benötigten Angaben zu halten.
4. Nutzung der Operationswerke, um die Selektionsangaben für Argumente und Resultate zu bestimmen. Statt der Selektionseinrichtungen gibt es lediglich ein Adressenregister, das in jedem Zyklus mit jeweils einer Selektionsangabe (Adresse) geladen werden kann.
5. Weiterbildung der Steuermittel für die beschriebene zyklusweise sequentielle Nutzung der Schaltungsstruktur (Befehlsablaufsteuerung; "Sequencer").

Der Übergang zum klassischen v. Neumann-Rechner ist damit vollzogen. Bild 11 zeigt die Struktur; die Benennung der Schaltmittel ist bereits auf die allgemein übliche Ausdrucksweise umgestellt. Man gelangt also von der zweckgebundenen Schaltung zum Universalrechner, indem die technischen Mittel

¹ In Abhängigkeit von bestimmten Teilresultaten (Bedingungen).
² Welche Verknüpfungen unbedingt notwendig sind, ist umfassend untersucht worden; im besonderen haben van der Pohl und Fromme gezeigt, daß ein einzige Verknüpfungsart ausreicht (/74/, /279/, /280/). Gemäß dem Stand der Technik sind solche Minimalprinzipien kaum mehr von Interesse; vielmehr kann eine umfangreiche Erfahrungsbasis im Hinblick auf Zweckmäßigkeit und Nutzungshäufigkeit ausgewertet werden.

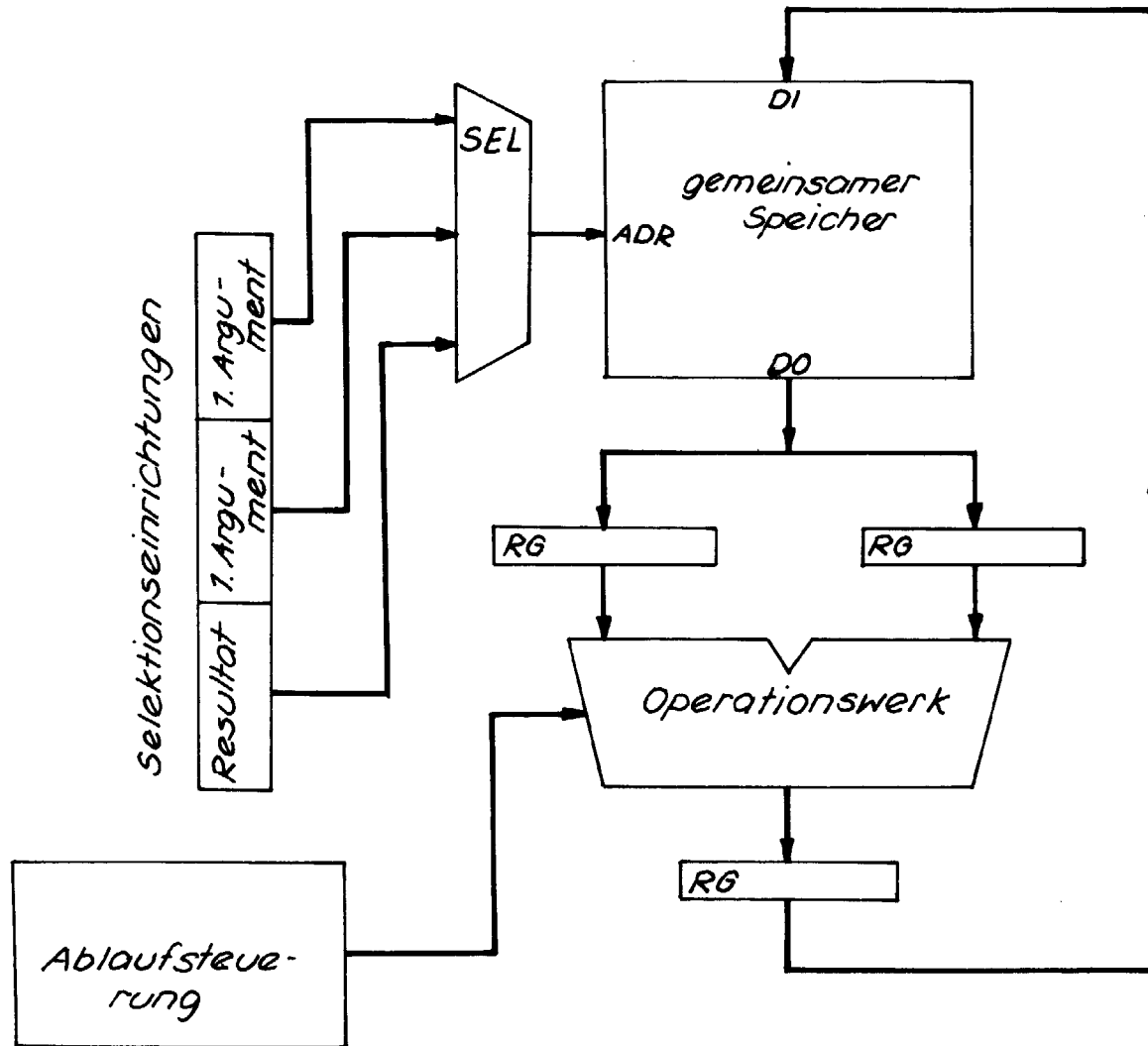


Bild 7

Registermaschine mit gemeinsamen Speicher für Argumente und Resultat

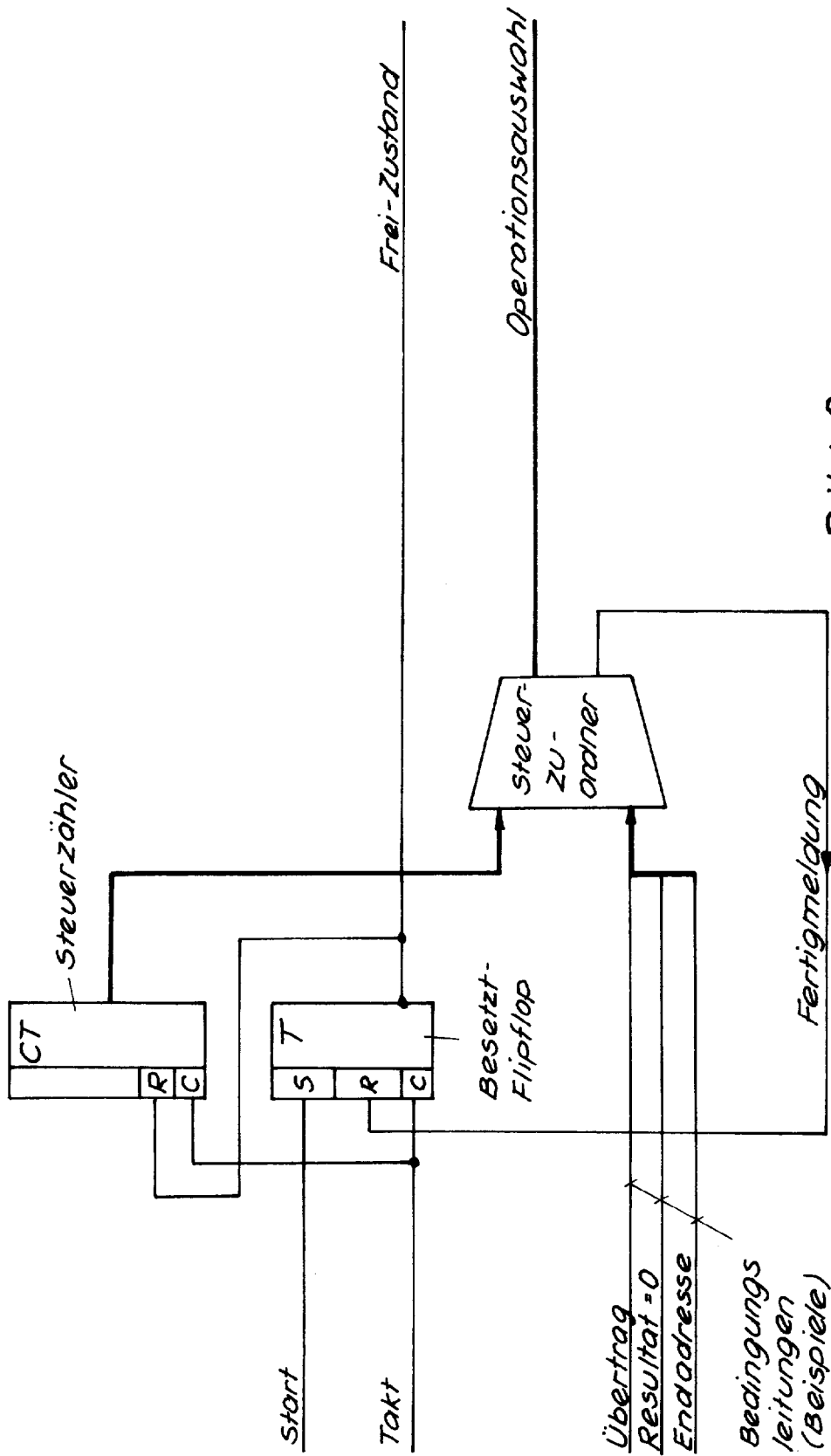
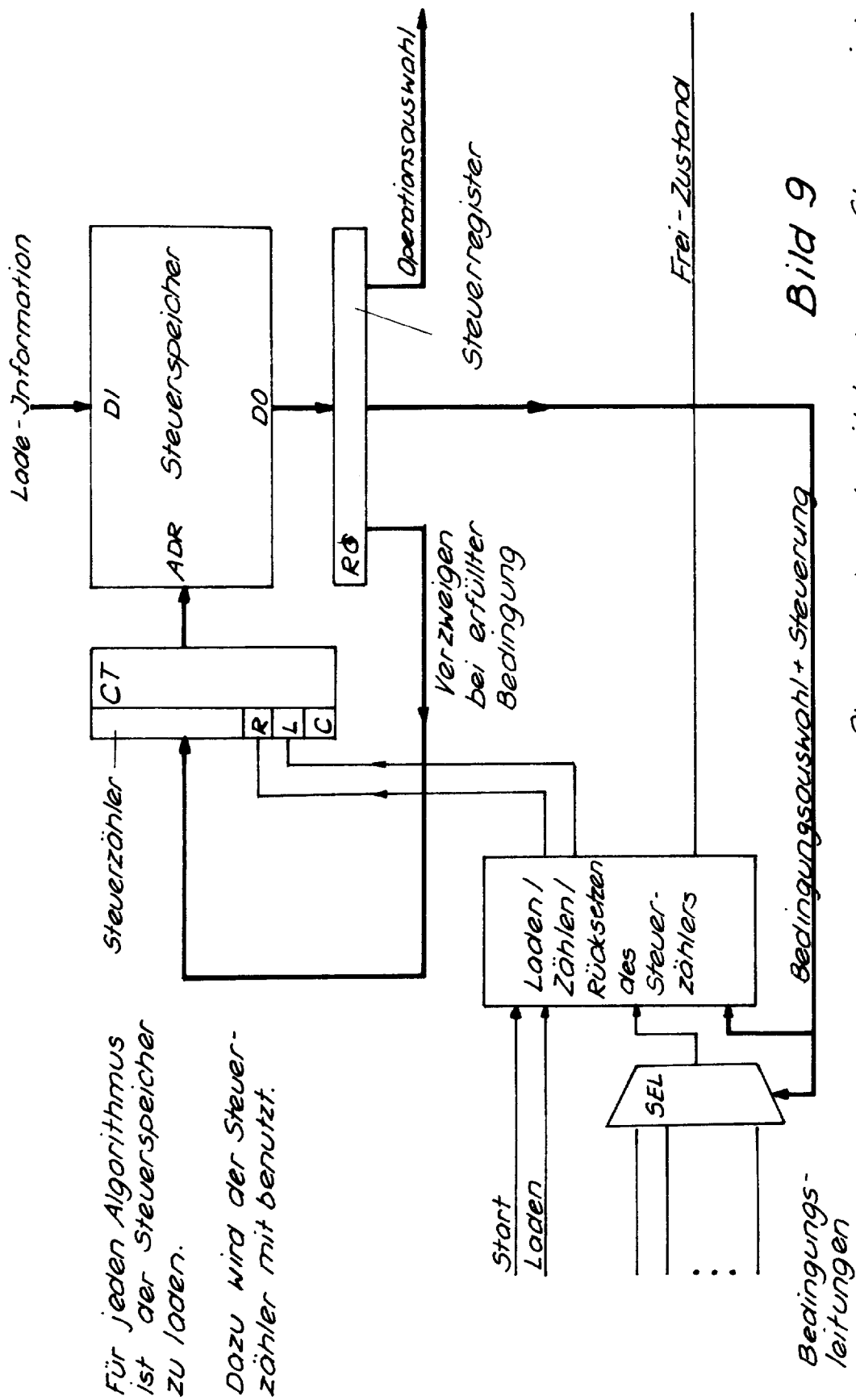


Bild 8 steuerautomat mit Bedingungs-
auswertung



Für jeden Algorithmus ist der Steuerspeicher zu laden.

Dazu wird der Steuerzähler mit benutzt.

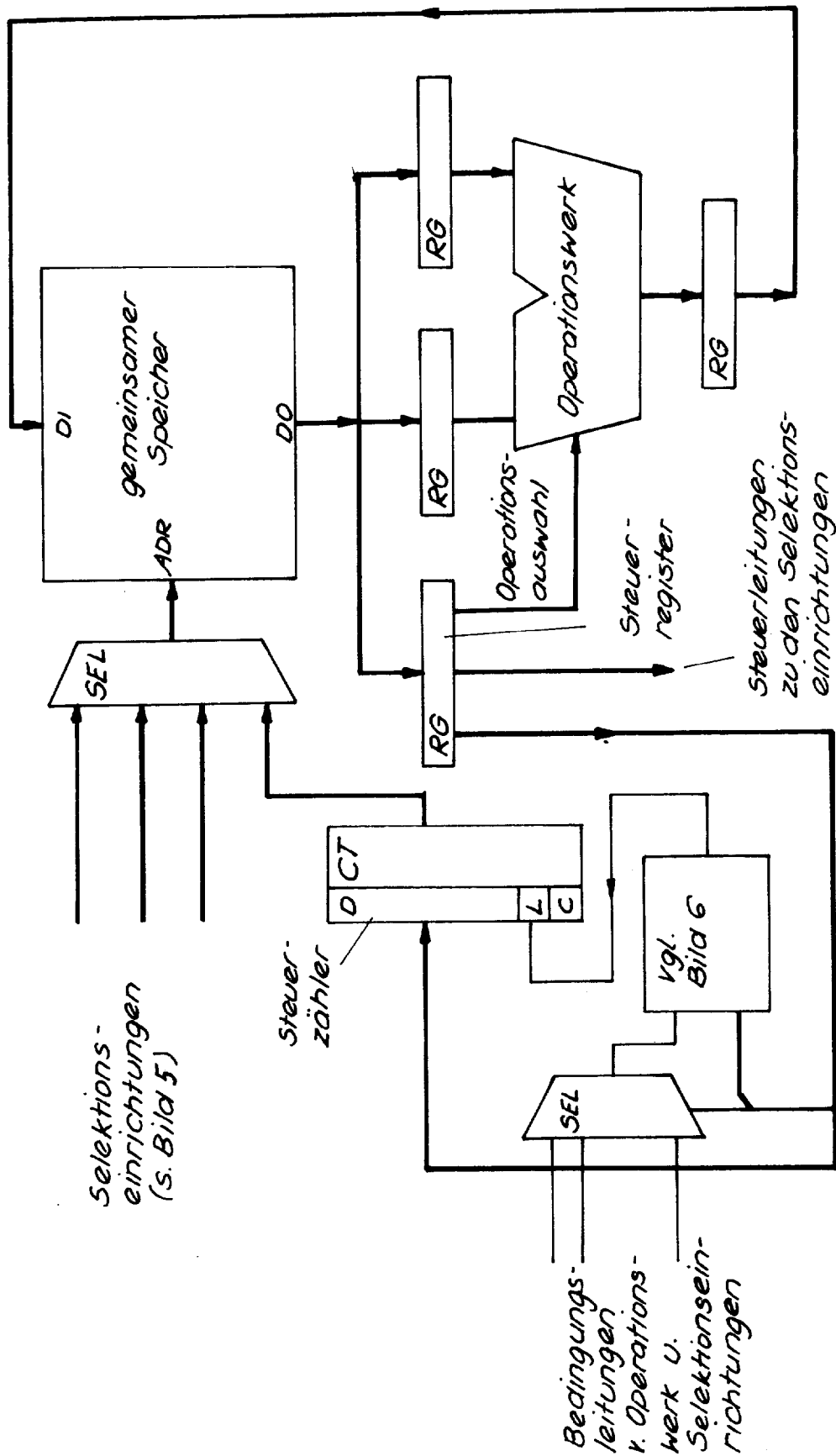
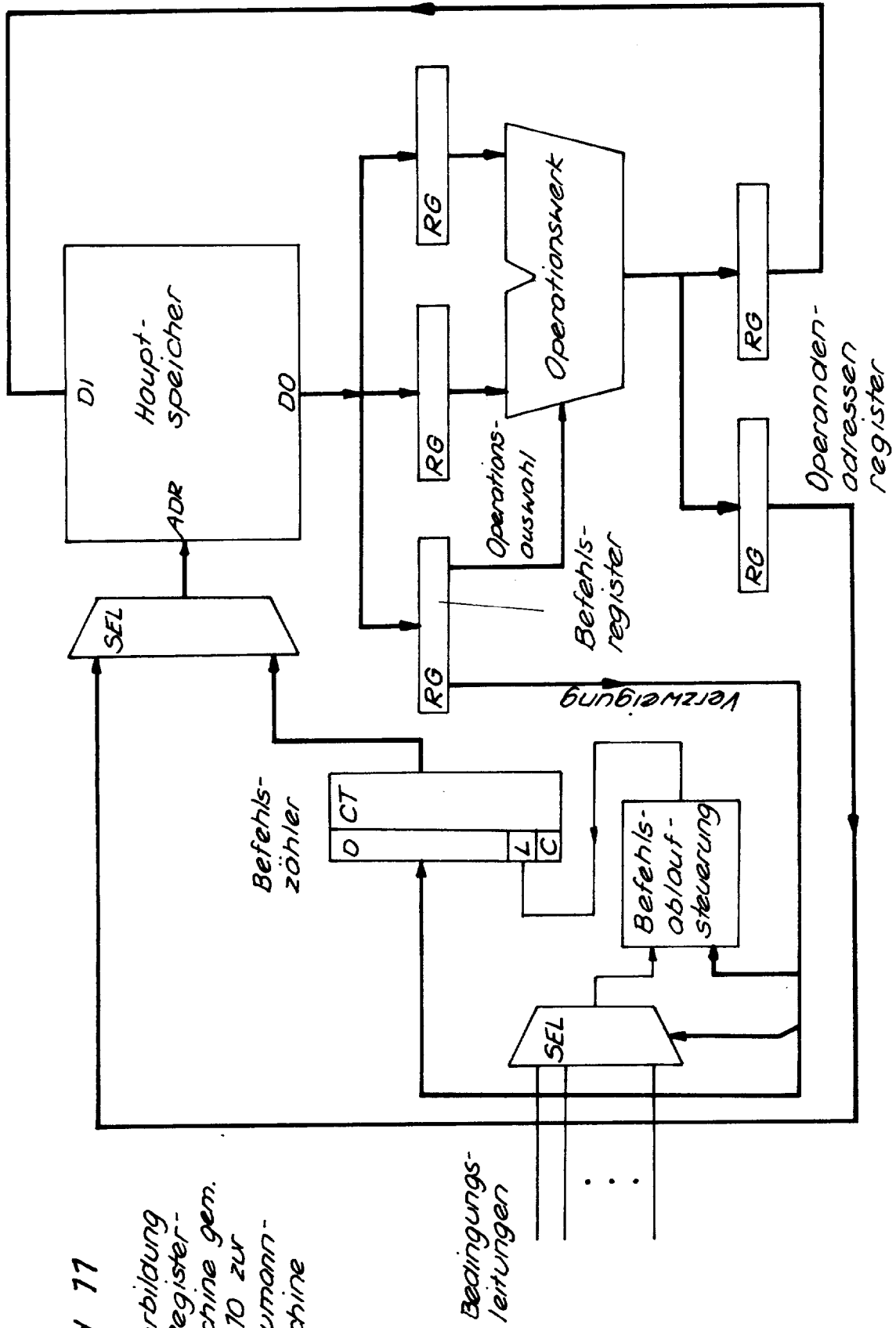


Bild 10 Registermaschine mit gemeinsamen Speicher für Argumente, Resultate und Steuerinformation

Bild 11

Weiterbildung
der Register-
maschine gem.
Bild 10 zur
v. Neumann-
Maschine



zur Implementierung von Algorithmen systematisch zunehmend universeller ausgebildet werden. Ein solcher Weg zeigt wichtige Sachverhalte auf:

1. Mit zunehmender Universalität der Schaltungsstruktur vermindert sich die Ausführungsgeschwindigkeit des einzelnen Algorithmus, da in leistungsbestimmende Signalwege Auswahl-schaltungen eingefügt (vgl. Bild 6) und alle Abläufe auf Folgen elementarer Aktionen zurückgeführt werden müssen.

2. Jede technische Einrichtung bedarf grundlegender Steuermit-tel. Diese werden mit zunehmender Universalität komplizierter; sie reichen vom einfachen Taktgeber für den Resultatspeicher in Bild 4 bis zur Befehlsablaufsteuerung (Sequencer) der v. Neumann- Maschine gemäß Bild 11.

3.2. Formale Darstellung

Um die Beziehungen zwischen Algorithmen und Schaltungsstruktu-ren zu untersuchen, wird nachfolgend eine formale Notation eingeführt. Eine Formalisierung kann gegebene Sachverhalte eindeutiger, bestimmter, genauer ausdrücken als eine auf An-schauung beruhende Darstellungsweise; sie kann aber selbst keine neuen Erkenntnisse hervorbringen, also auch beim Ausar-beiten neuartiger Schaltungslösungen und Architekturkonzepte die erfinderische Tätigkeit, die ohne ganzheitliche Anschauung schwer vorstellbar ist, nicht ersetzen. Deshalb wird die For-maldarstellung hier nur soweit ausgebildet, daß sie im heuristischen Sinne nutzbar ist.¹

3.2.1. Allgemeine Vereinbarungen

Alle Strukturen, die hier betrachtet werden, sind gekennzeich-net durch einen Eigennamen, eine Beschreibung ihres inneren Aufbaus und eine Beschreibung ihrer Bedeutung bzw. Wirkungs-weise.² Deshalb wird eine beliebige Menge aus n einzelnen Strukturen als Tripel gleichmächtiger Teilmengen definiert:

$$\mathcal{M} = \{N^m, D^m, M^m\} \quad (3.1)$$

¹ Es sei darauf hingewiesen, daß sich eine algebraische Model-lierung zur Verfeinerung und exakteren Fassung des hier skizzierten Ansatzes anbietet. Vgl. S. 2, Punkt 2. Um ein formales System für konkrete Architekturen, Schaltungen usw. wirklich nutzen zu können, ist dessen programmseitige Imple-mentierung unerläßlich.

² Das ist methodisch durch die von Tarski (/300/) begründete Vorgehensweise geprägt: es wird untersucht, wie die sprach-lichen Mittel zur Beschreibung der in Frage stehenden Sach-verhalte beschaffen sein müssen.

Darin bedeuten:

$N^{m_i} = \{N_1^{m_i} \dots N_v^{m_i}\}$	die Menge der Eigennamen
$D^{m_i} = \{D_1^{m_i} \dots D_v^{m_i}\}$	die Menge der internen Strukturbeschreibungen
$M^{m_i} = \{M_1^{m_i} \dots M_v^{m_i}\}$	die Menge der Beschreibungen der Bedeutung bzw. Wirkungsweise.

Die Eigennamen dienen lediglich der eindeutigen Bezeichnung, ihnen kommt sonst keine weitere Bedeutung zu.

Informationsstrukturen sind bis aufs Bit ausdefiniert¹; Schaltungsstrukturen werden letztlich durch Verbindungen von Speichermitteln und kombinatorischen Schaltungen beschrieben.

Die Bedeutung bzw. Wirkungsweise wird, je nach Zweckmäßigkeit, in einer geeigneten Metasprache ausgedrückt bzw. durch Rückführung auf Abläufe in wohlbekannten Schaltungsanordnungen erklärt.

Als geeignete Metasprache ist beispielsweise die Kombination von umgangssprachlicher, formaler und bildlicher Darstellung anzusehen, in der üblicherweise Rechnerarchitekturen gegenüber dem Nutzer dokumentiert werden.² Es sei bemerkt, daß es keineswegs um triviale Systeme geht, so daß pragmatische Gesichtspunkte nicht außer Acht gelassen werden sollten. So hat die Erfahrung gezeigt, daß es unerlässlich ist, komplexe Rechnerarchitekturen in vollem Umfang umgangssprachlich zu beschreiben.³

Die Rückführung auf wohlbekannte Schaltungsanordnungen ist ihrem Wesen nach eine Methode der operationalen Semantikdefinition.⁴ Die Absicht besteht darin, eine bis auf die einzelne Boolesche Gleichung ausgearbeitete Schaltungsbeschreibung einer solchen Semantikdefinition zugrunde zu legen. Die Schaltungsanordnungen sollen mit Evidenzkriterien verifizierbar sein; sie sollen deshalb die Algorithmen und Datenstrukturen möglichst direkt mit überschaubaren technischen Mitteln⁵ widerspiegeln. Beispielsweise wird für die Multiplikation von Binärzahlen eine Anordnung geschichteter Addierwerke vorgesehen (Schaltungsanordnung MULTIPLY). Die Bedeutung eines Ausdruckes $C := A * B$ wird dann so erklärt, daß C jenes Resultat-Bitmuster zugeordnet erhält, das an den Ausgängen von MULTIPLY entsteht, wenn die Werte von A, B an deren Eingängen anliegen (das Resultat ist durch die Booleschen Gleichungen, die MULTIPLY beschreiben, eindeutig bestimmt).

1 Diese Absicht wurde bereits von Zuse mit dem Plankalkül verfolgt (/100/).

2 Als Beispiele können die bekannten Architekturhandbücher angesehen werden, z. B. /36/, /38/.

3 Umfassende Erfahrungen eines führenden Anbieters werden in /332/ vermittelt.

4 Eine Übersicht über verschiedene Möglichkeiten zur formalen Definition von Bedeutungen ist z. B. in /90/ zu finden.

5 Überschaubarkeit geht vor Aufwand!

3.2.2. Algorithmen

Ein Algorithmus \mathcal{A} wird durch ein geordnetes Paar aus Daten \mathcal{V} und einer Wandlungsvorschrift \mathcal{W} dargestellt:

$$\mathcal{A} = \{ \mathcal{V}, \mathcal{W} \} \quad (3.2)$$

Das entspricht direkt der Formulierung eines Algorithmus in einer hinreichend ausgestatteten Programmiersprache.¹ Die Menge \mathcal{V} der Daten ist in die Menge der Argumente \mathcal{V}^a und in die Menge der Resultate \mathcal{V}^r zerlegbar:

$$\mathcal{V} = \{ \mathcal{V}^a, \mathcal{V}^r \}$$

Die Wandlungsvorschrift \mathcal{W} ist gegeben durch eine Ressourcenbeschreibung \mathcal{R} und eine Ablaufbeschreibung \mathcal{P} :

$$\mathcal{W} = \{ \mathcal{R}, \mathcal{P} \}$$

So wird deutlich, daß es stets notwendig ist, zu jeder Ablaufbeschreibung \mathcal{P} (die man sich z. B. als den Quelltext eines Programms vorstellen kann), die jeweiligen Voraussetzungen mitzudenken: wird Hardware direkt programmiert, ist \mathcal{R} die Beschreibung der Maschinenarchitektur; wird \mathcal{P} in einer höheren Programmiersprache formuliert, ist \mathcal{R} die Sprachbeschreibung.

3.2.3. Ressourcen

Ressourcen sind hier allgemein die Mittel zur Implementierung von Algorithmen, also die unmittelbar gegebenen grundlegenden Datenstrukturen, elementaren Algorithmen und explizit nutzbaren technischen Einrichtungen. Die Ressourcenbeschreibung \mathcal{R} wird folgendermaßen definiert:

$$\mathcal{R} = \{ d, c, h, r, s, b \} \quad (3.3)$$

Die Bedeutung der Symbole ist in Tafel 3 erklärt. Die Mengen sind gemäß (3.1) zerlegbar; dabei gilt für jede Menge q ; $q = (d, c, b, r, s)$:

D^q beschreibt die Kardinalitäten (Anzahl der Bits bzw. der elementarerer Strukturen)

M^q beschreibt die Wirkungen.

Beispielsweise beschreibt D^d die Datenformate, D^c die Befehlsformate und M^c die Wirkungen der Befehle. b darf nur Ausdrücke aus d, c, h, r, s sowie allgemein-logische² und Ausdrücke des zeitlichen Folgens enthalten. Im besonderen müssen sich alle Wirkungen durch Nennung der jeweili-

1 Die Programmiersprache muß Ausdrucksmittel für Datentypen, Verknüpfungen und Abläufe enthalten. Beispiele: C, Clu, Ada.

2 Die Ausdrucksweise folgt /300/. Allgemein-logische Ausdrücke sind UND, ODER, NICHT, ELEMENT VON, FÜR ALLE...GILT, WENN...SO usw.

Symbol	Bedeutung	praktische Entsprechung	
		in Maschinenarchitekturen	in höheren Programmiersprachen
$d = \{d_1 \dots d_r\}$	Menge der Datenstrukturen	Datenformate (Byte, Maschinenwort usw.)	elementare Datentypen (Integer, Real usw.)
$c = \{c_1 \dots c_p\}$	Menge der Steuerstrukturen	Befehle und Steuerworte	elementare Operatoren
$h = \{h_1 \dots h_s\}$	Menge der technischen Einrichtungen	architekturseitig definierte Register u. ä.	- (leere Menge)
$r = \{r_1 \dots r_e\}$	Menge der Arten des Datenzugriffs	Adressierungsprinzipien	Konstrukte für Zugriffe zu Datenstrukturen
$s = \{s_1 \dots s_g\}$	Menge der Prinzipien der Ablauforganisation	Steuerung der Befehlsfolge, Verzweigungen, Unterbrechungsbehandlung usw.	Konventionen der Abarbeitungsreihenfolge, Verzweigungen, Prozeduraufruf, Ausnahmebehandlung usw.
b	Beschreibung des Zusammenwirkens	beschreibt alle Einzelheiten, die nicht im Rahmen von $d \dots s$ erklärbar sind	

Tafel 3 Symbolerklärungen für die
Ressourcendefinition

gen Elemente von M^c bzw. h und der zulässigen Aktivitäten (aus r , s) ausdrücken lassen. Man beachte, daß keine physischen Verbindungen definiert sind: (3.3) ist eine funktionelle Beschreibung, keine strukturelle.

Ist ein Algorithmus \mathcal{A} mit gegebenen Ressourcen \mathcal{R} zu implementieren, so sind alle Datenstrukturen \mathcal{V} aus Datenstrukturen aufzubauen, die in \mathcal{R} definiert sind, und alle Informationswandlungen von \mathcal{A} sind auf Folgen von elementaren Algorithmen zurückzuführen, die in c , r , s bzw. b vorgesehen sind.¹

Die Ablaufbeschreibung von \mathcal{A} muß also mit \mathcal{R} auskommen, sie darf höchstens noch Eigennamen von Daten und Ausdrücke aus deren Strukturbeschreibung $D^{\mathcal{V}}$ enthalten.

3.2.4. Schaltungsanordnungen

Für die weiteren Betrachtungen liefert ein Strukturgraph $\Sigma = \{E, V, \Gamma, \Gamma^L, \Gamma^b, \Pi\}$ ein hinreichend genaues Modell einer Schaltungsanordnung.² Im einzelnen bezeichnet:

E die Liste der Schaltelemente (Knoten des Strukturgraphen), wofür entweder Speicher (S-Knoten) oder kombinatorische Schaltungen (K-Knoten) in Frage kommen. E ist gemäß (3.1) zerlegt; dabei beschreibt $D^E = \{D_1^E \dots D_\mu^E\}$ die Art des Knotens (S oder K) und $M^E = \{M_1^E \dots M_\mu^E\}$ dessen Ausgestaltung im einzelnen.

V die Liste der Verbindungen (Kanten des Strukturgraphen). In der Zerlegung nach (3.1) beschreibt $D^V = \{D_1^V \dots D_\nu^V\}$ die jeweilige Anzahl an Verbindungsleitungen, und $M^V = \{M_1^V \dots M_\nu^V\}$ kennzeichnet deren Verwendung (z. B. Taktleitungen, Datenleitungen, Adressenleitungen usw.).

Γ die Verbindungsmatrix. Es ist eine quadratische Matrix mit μ Zeilen. Besteht eine Verbindung von einem Knoten i zu einem Knoten j , so ist an der Position Γ_{ij} die Ordinalzahl in V eingetragen, die die Art der Verbindung kennzeichnet, sonst eine \emptyset .

Γ^L die Leitungszahlmatrix. Sie hat dieselbe Struktur wie Γ und enthält für jede Verbindung von einem Knoten i zu einem Knoten j in der Position Γ_{ij}^L die Anzahl der Verbindungsleitungen.

Γ^b die binäre Verbindungsmatrix. Sie hat dieselbe Struktur wie Γ und enthält für jede Verbindung von einem Knoten i zu einem Knoten j in der Position Γ_{ij}^b eine 1.

1 Die elementaren Algorithmen (s. Tafel 3) sind innerhalb von \mathcal{R} nicht weiter zurückführbar. Ihre Ressourcen sind die Mittel, mit denen \mathcal{R} implementiert ist.

2 In der Praxis sind zur Dokumentation komplexer Schaltungsanordnungen umfangreiche Datenmassive erforderlich. Σ entspricht dem Schaltplan auf der Ebene der Funktionalelemente.

Π die Funktionsbeschreibung der Anordnung.¹ Sie enthält neben Ausdrücken aus E und V nur noch allgemein-logische Ausdrücke und solche des zeitlichen Folgenseins.

An sich bestimmen E , V , Π den Strukturgraphen vollständig. Die Matrizen Γ , Γ^a , Γ^b wurden eingeführt, um bestimmte Eigenschaften von Σ elegant auswerten zu können.²

3.2.5. Abbildungsfragen

Um einen Algorithmus \mathcal{U} tatsächlich auszuführen, ist eine Schaltungsanordnung Σ erforderlich, die die Anforderungen der Ressourcenbeschreibung \mathcal{R} erfüllt.³ Dazu muß die Schaltungsanordnung Σ folgenden Anforderungen entsprechen:

1. Σ muß S-Knoten enthalten, die zur Aufnahme aller in \mathcal{R} beschriebenen Informationsstrukturen geeignet sind, und zwar in dem Umfang, wie er durch die in D^r , D^s angegebenen Kardinalitäten erforderlich ist.
2. Σ ist so auszugestalten, daß alle in M^c , M^r , M^s beschriebenen Wirkungen durch zeitliche Folgen von Signalflüssen eintreten, die sich im Rahmen von Π beschreiben lassen, sowie durch Verknüpfungen in K-Knoten und Informationsspeicherung in S-Knoten.

Gelingt es, einen Algorithmus \mathcal{U} so in eine Schaltungsanordnung Σ abzubilden, daß alle Daten \mathcal{U} eindeutig S-Knoten zugeordnet werden können und die Ablaufbeschreibung ρ der Wandlungsvorschrift \mathcal{W} nur eine einzige Aktivierung umfaßt (die Anwendung einer einzigen Steuerstruktur $c; e c$ im Rahmen eines einzigen Prinzips $p; e p$ der Ablauforganisation), so heißt der Algorithmus \mathcal{U} vergegenständlicht in der Schaltungsanordnung Σ .

Erfüllt eine Schaltungsanordnung Σ die Anforderungen einer Ressourcenbeschreibung \mathcal{R} , so sind alle in \mathcal{R} definierten Informationswandlungen in Σ vergegenständlichte Algorithmen. Von nun an wird ausdrücklich zwischen Vergegenständlichung und Implementierung von Algorithmen unterschieden, das heißt zwischen der direkten Umsetzung in Schaltungsanordnungen und der Umsetzung in zeitliche Folgen anderer Algorithmen.

- 1 Dafür reichen die Mittel der Automatentheorie an sich aus. Die Praxis zeigt aber, daß ein einzelnes Beschreibungsmittel (Graph, Übergangsmatrix, Phasenliste usw.) nicht allen Anforderungen gerecht wird, so daß verschiedene Darstellungsweisen im Verbund genutzt werden müssen (vgl. /51/).
- 2 Das wird z. B. in Abschnitt 4.3. genutzt.
- 3 Die Schaltungsstruktur darf einen größeren Funktionsumfang aufweisen als gemäß \mathcal{R} erforderlich ist. Nur sind bei gegebenem \mathcal{R} die zusätzlichen Möglichkeiten nicht nutzbar, um \mathcal{U} zu implementieren (typisches Beispiel: zusätzliche Spezialhardware, die von einer höheren Programmiersprache (\mathcal{R}) aus nicht zugänglich ist).

4. Grundlagen der Bewertung

Um leistungsfähige und technisch-ökonomisch sinnvolle Schaltungsanordnungen zu schaffen, sind verschiedene Lösungsansätze zu erarbeiten und zu bewerten. Teillösungen, die sich als zweckmäßig erwiesen haben, sind zu Systemlösungen zusammenzufassen. Dafür sind Bewertungsmaßstäbe erforderlich: Schaltungsanordnungen sind nach ihrer Leistungsfähigkeit zu bewerten, Algorithmen nach der Eignung zur Vergegenständlichung und Aufwendungen nach ihrer Nützlichkeit. Weiterhin sind die verschiedenen Lösungsansätze untereinander zu vergleichen.

4.1. Bewertung der Schaltungsanordnungen

Zunächst wird das absolute Leistungsvermögen von Hardware untersucht; die Aufwendungen bleiben also außer Betracht. Allgemein übliche Bewertungsgrundlagen sind:¹

1. Ausführungszeiten bestimmter Algorithmen (Anwendungsprogramme)
2. Ausführungszeiten "repräsentativer" Algorithmen ("benchmark"-Programme)
3. Ausführungszeiten vergegenständlichter Algorithmen (bei üblichen Rechnern in herkömmlicher Redeweise: die Ausführungszeiten der Maschinenbefehle).

Es ist klar, daß brauchbare Angaben nur selten analytisch oder durch einmalige Probeläufe zu erhalten sind, sondern daß Mittelwerte bzw. Erwartungswerte gebildet werden müssen.

Die Messung der Ausführungszeiten von Anwendungsprogrammen liefert einem Nutzer, der nur an bestimmten Algorithmen interessiert ist, gut auswertbare Vergleichsdaten über die Eignung verschiedener Maschinen. Sie hat aber folgende Nachteile:

- Anwendbarkeit nur auf fertige Hardware-Software-Komplexe (Maschinen und Programme müssen vorhanden sein); kaum geeignet für die Bewertung von Lösungsansätzen neuer Schaltungsstrukturen (Simulation ist aufwendig und langsam)
- sehr beschränkte Aussagekraft hinsichtlich des allgemeinen Leistungsvermögens
- vergleichsweise hohes Bewertungsrisiko: geringe Änderungen in den Algorithmen, Programmen bzw. Compilern können sich erfahrungsgemäß in manchen Maschinen deutlich auf die Laufzeiten auswirken, in anderen nicht.

Derartige Messungen testen nicht nur die Hardware, sondern auch den Compiler und die Kunstfertigkeit des Programmierers.

¹ Derzeit gibt es noch kein gesichertes Fachwissen darüber, wie ausgewogene ("well-balanced") Hardware-Software-Systeme zu entwerfen sind, die hohe Leistungen für normale Nutzer liefern. Man kann sich für die Leistungsbewertung nur auf empirische Resultate verlassen. (Nach: /213/.)

So ist in /264/ dargestellt, daß auf der CRAY-1 die rechts angegebene Schleife um 25% schneller läuft als die linke¹:

```
DO 10 I = 1,N
  Y(I) = A*X(I)+Y(I)
10 CONTINUE
```

```
DO 10 I = 1,N
  Y(I) = A*X(I)+(Y(I))
10 CONTINUE
```

Praktische Erfahrungen zeigen, daß allein ein Wechsel des Compilers das Laufzeitverhalten der Programme beträchtlich verändern kann.

Repräsentative Algorithmen, die eindeutig dokumentiert sind (z. B. in einer verbreiteten Programmiersprache), sind für das überschlägige Vergleichen zweckmäßiger; sie gestatten es, die Eignung von Maschinen für bestimmte Klassen von Algorithmen zu beurteilen ("benchmark"-Tests). Mit einigen Tests kann man das Leistungsvermögen bezüglich bestimmter Operationen recht genau erfassen.² So kann man beim bekannten LINPACK-Benchmark (Lösung linearer Gleichungssysteme) die Gleitkommaoperationen (Addition und Multiplikation) auszählen: ein System aus n Gleichungen erfordert

$$\frac{2}{3}n^3 + 2n^2 + O(n)$$

solche Operationen.

Die gemessenen Werte weichen erheblich von den Angaben der Maximalleistung ("peak performance") ab (Tafel 4).³

Hingegen lassen sich die Ausführungszeiten der vergegenständlichten Algorithmen ("Maschinenbefehle") an sich recht einfach aus einem hinreichend detaillierten Schaltungsentwurf bestimmen (durch Auszählen der Taktzyklen und einige statistische Annahmen, z. B. hinsichtlich der Vermittlungszeiten von Speicherzugriffen und der Trefferraten bei Cache-Speichern). Besonders beliebt ist in der Praxis die Annahme der jeweils günstigsten Verhältnisse: so kommen die meisten der üblichen MIPS- bzw. MFLOP-Angaben zustande.

Ein Blick in Tafel 4 zeigt die Fragwürdigkeit solcher Angaben. Etwas bessere Vergleichswerte liefern die bekannten statistischen Befehlsverteilungen (Mix-Werte, z. B. Gibson-Mix, GPO-Mix usw.). Sie können zu Vergleichszwecken recht einfach berechnet werden; man sollte aber nicht errechnete Zahlen mit gemessenen vergleichen, und die betreffenden Maschinenarchitekturen sollten einigermaßen vergleichbar sein.⁴

Des weiteren hat sich gezeigt, daß eine auf gute Mix-Leistung hin entworfene Maschine in der Anwendungsleistung nicht immer im erwarteten Maße überlegen ist.⁵

1 Gilt für Übersetzung mit Fortran-Compiler Level 1.09.

2 Auch dieses Verfahren ist ohne fertige Maschine (mit Betriebssystem und Compiler) kaum anwendbar.

3 Die Darstellung (einschließlich Tafel 4) stammt aus /161/; s. weiterhin /159/, /160/.

4 Man vergleiche also nur CISC-Maschinen, RISC-Maschinen, Vektorprozessoren usw. jeweils untereinander.

5 Z. B. EC 1055 im Vergleich mit EC 1040.

lfd. Nr.	Maschine	Zyklus (ns)	Prozessoren
1	Culler PSC	200	1
2	Multiflow TRACE 7/200	130	1
3	Convex C-1	100	1
4	SCS-40	45	1
5	FPS 264	38	1
6	Alliant FX/8	170	8
7	Amdahl 500	7,5	1
8	CRAY-1	12,5	1
9	CRAY X-MP-1	9,5	1
10	IBM 3090/VF-200	18,5	2
11	Amdahl 1100	7,5	1
12	NEC SX-1E	7	1
13	CDC CYBER 205	20	1
14	CRAY X-MP-2	9,5	2
15	IBM 3090/VF-400	18,5	4
16	Amdahl 1200	7,5	1
17	NEC SX-1	7	1
18	CRAY X-MP-4	9,5	4
19	Hitachi S-810/20	14	1
20	NEC SX-2	6	1
21	CRAY-2	4,1	4

lfd. Nr.	Leistung (MFLOP)		Effizienz
	maximal	LINPACK	
1	5	2	0,4
2	15	6	0,4
3	20	3	0,15
4	44	8	0,18
5	54	5,6	0,1
6	94	7,6	0,08
7	133	14	0,11
8	160	12	0,075
9	210	24	0,11
10	216	12*	0,11 (0,056)
11	267	16	0,06
12	325	35	0,11
13	400	17	0,043
14	420	24*	0,11 (0,057)
15	432	12	0,11 (0,028)
16	533	18	0,034
17	650	39	0,06
18	840	24*	0,11 (0,029)
19	840	17	0,02
20	1300	46	0,035
21	2000	5*	0,03 (0,0075)

* Angabe für einen Prozessor

Tafel 4

Das Leistungsvermögen von Hochleistungsrechnern; mit dem LINPACK- "benchmark" gemessen

Hier sollen nicht nur bekannte bzw. von vornherein leicht vergleichbare Prinzipien untersucht werden, und es ist wünschenswert, Schaltungslösungen in einem frühen Bearbeitungsstand¹ überschlagsmäßig beurteilen zu können. Dafür wird ein Verfahren vorgeschlagen, das auf folgenden Überlegungen beruht:

Für einen einzelnen Algorithmus interessiert an sich nur die reine Ausführungszeit; die Zeit, die die Maschine braucht, um die gewünschten Resultate zu liefern.

Weiterhin kann man sich vorstellen, für den besagten Algorithmus eine Sondermaschine zu bauen. Daß programmgesteuerte Universalmaschinen verwendet werden, hat unter diesem Gesichtspunkt nur den Grund, daß es nicht durchführbar ist, für jeden Algorithmus eine Sondermaschine zu bauen. Die programmgesteuerte sequentielle Ausführung eines Algorithmus ist also nur ein Notbehelf, eben wegen der technisch-ökonomischen Gegebenheiten. Man kann deshalb die Abarbeitung von Befehlen eher als Störfaktor auffassen und nur die Argumente und Resultate der Algorithmen betrachten. Diese Werte sind binär codiert, und sie werden in aufeinanderfolgenden Taktzyklen von Speichermitteln zu Speichermitteln über Verbindungsleitungen und kombinatorische Netzwerke bewegt.

Für eine Sondermaschine ist die Frage nach "MIPS" an sich gegenstandslos. Solche Angaben sind nur sinnvoll, wenn es darum geht, den Geschwindigkeitszuwachs einer Sondermaschine im Vergleich zur Nutzung einer Universalmaschine für den selben Zweck zu beurteilen. Beispiel: Der leistungsentscheidende Ablauf eines Algorithmus erfordere 50 Befehle einer geläufigen Rechnerarchitektur. Eine Sondermaschine leiste dasselbe in 1 μ s. Es müßte also ein Universalrechner mit 50 MIPS beschafft werden, um das Leistungsvermögen der Sondermaschine zu erreichen² (an diese Überlegung wird sich üblicherweise eine Kostenabschätzung anschließen). Solche Betrachtungen wurden z. B. in /243/ angestellt, um die Sinnfälligkeit konkret entworfener Sondermaschinen im Vergleich zu Universalrechnern beurteilen zu können. Hier stellt sich die Aufgabe gerade anders herum: das Leistungsvermögen der Sondermaschine ist bekannt, und es ist die Universalmaschine zu bewerten. Die Sondermaschine verkörpert die leistungsfähigste technisch beherrschbare Vergegenständlichung des Algorithmus. Sie erzeugt die Resultate in einer minimalen Anzahl von Maschinenzyklen.

Um ein Maß für die Verarbeitungsleistung zu gewinnen, ist es mithin ausreichend, zu zählen, wieviele Bits an Nutz-Information (Argumente und Resultate der jeweils betrachteten Algorithmen) in einer bestimmten Zeiteinheit verarbeitet bzw. erzeugt werden. Bezeichnungsvorschlag: "Effektive Bits je Sekunde" (EB/s; mit Faktoren 10^6 bzw. 10^9 dann MEB/s bzw. GEB/s).

1 Z. B. nach Ausarbeitung der Funktionsprinzipien und des Blockschaltbildes bzw. der Register-Transfer-Struktur.

2 Die Aussage "die Sondermaschine leistet x MIPS" ist falsch! Zutreffend ist vielmehr: "Die Sondermaschine ersetzt für den betreffenden Algorithmus einen Universalrechner von x MIPS".

Um dieses Leistungsmaß (im folgenden mit PM bezeichnet) zu bestimmen, wird in einem Intervall t_x gezählt, wieviele Zugriffe zur Nutz- Information (die durch \mathcal{V} in (3.2) beschrieben wird) dabei ausgeführt werden (die Anzahl sei r).¹ Die Anzahl der Nutzbits in Zugriff i ($1 \leq i \leq r$) sei $CARDB_i$. Dann gilt:

$$PM = \frac{1}{t_x} \sum_{i=1}^r CARDB_i. \quad (4.1)$$

Da die Ausführungszeiten der meisten Algorithmen datenabhängig sind, müssen in der Praxis Mittelwerte bzw. Erwartungswerte gebildet werden. Die Angaben müssen stets auf den jeweiligen Algorithmus bzw. Komplex von Algorithmen bezogen werden.

Für überschlägige Abschätzungen und Vergleiche eignet sich beispielsweise der Vorrat an elementaren Operatoren (numerische und nichtnumerische), der in eingeführten Programmiersprachen (z. B. Fortran, C, Ada) definiert ist.² Oft lassen sich die Operatoren direkt auf Maschinenbefehle abbilden; gelegentlich erfordert ein Operator eine Folge von Maschinenbefehlen (z. B.: Multiplikation in RISC- Architekturen). Es wird von Verzweigungen usw. abgesehen und eine lückenlose Folge von Operatoren angenommen, die gespeicherte Argumente verarbeiten und die Resultate wieder in einem allgemein zugänglichen Speicher ablegen.³ Dem eigentlichen Vergleich wird dann eine sinnfällige Folge ("Mix") solcher Operatoren zugrunde gelegt, wobei die Ausführungszeit (t_x) und die unbedingt notwendigen Argument- und Resultattransporte ($CARDB_i$) bestimmt werden.⁴ Bei Hochleistungssystemen reicht es bisweilen aus, den Idealfall anzunehmen, daß alle Datenpfade und Verarbeitungswerke voll ausgelastet sind: es wird dann die maximale Datenrate betrachtet.⁵

4.2. Bewertung der Algorithmen

Um Algorithmen zur Vergegenständlichung auszuwählen, ist deren Eignung zu bewerten. Dabei geht es nicht um die Nützlichkeit oder universelle Anwendbarkeit - eine solche Vorauswahl wird als gegeben angenommen -, sondern um die grundsätzliche Möglichkeit, leistungsmäßig überlegene Schaltmittel entwerfen zu können. Die zeiteffektivste Form der Vergegenständlichung ist dann gegeben, wenn die Argument- und Resultatbits nur jeweils einmal transportiert werden müssen.⁶

1 Für künftige Maschinen sollte auch an technische Vorkehrungen zur Leistungsmessung gedacht werden.

2 Vgl. etwa Tafel 8 (S. 72).

3 Die Speicher müssen für andere Prozessoren bzw. für die Ein- und Ausgabe zugänglich sein. Register-Register-Verknüpfungen verlängern t_x , zählen aber nicht für $CARDB_i$.

4 Datentransporte aus Emulationsgründen (z. B. zu Hilfsbereichen im RAM) verlängern t_x , zählen aber nicht für $CARDB_i$.

5 Zugriffe zu Befehlen, Deskriptortabellen usw. verlängern manchmal (wenn nicht parallelisierbar) t_x , zählen aber nicht für $CARDB_i$.

6 Dieser Ansatz wurde in /243/ erstmals beschrieben.

Um diesen Sachverhalt zu bewerten, wird der Begriff der Implementierungseffizienz¹ e_i wie folgt eingeführt:

$$e_i = \frac{\sum_{i=1}^n \text{CARDB}(A_i) + \sum_{j=1}^m \text{CARDB}(R_j)}{z \cdot (\text{ARG_LINES} + \text{RES_LINES})} \quad (4.2)$$

Bedeutung der Symbole:

- $\text{CARDB}(A_i)$, $\text{CARDB}(R_j)$: Anzahl der Bits, mit denen die Argumente A_i ($1 \leq i \leq n$) bzw. Resultate R_j ($1 \leq j \leq m$) jeweils codiert sind
- ARG_LINES , RES_LINES : Anzahl der genutzten Leitungen für die Zuführung der Argumente bzw. den Abtransport der Resultate
- z : Anzahl der Maschinenzyklen, die für die Bildung aller Resultate benötigt werden ($z \geq 1$).

Wird kein Resultat gespeichert (wenn beispielsweise der Algorithmus lediglich eine Bedingung prüft), so sind die Resultat-Terme gemäß der jeweiligen binären Codierung (z. B.: einzelne Bedingungsleitung, 2-bit-Bedingungscode o. ä.) anzusetzen. Ist eine technisch gegebene Leitungszahl (z. B. aus Γ^L ablesbar) größer als die betreffende Anzahl an Bits, so gilt letztere.

Die Implementierungseffizienz ist eine dimensionslose Zahl im Intervall $0 < e_i \leq 1$, und es ist ersichtlich, daß im günstigsten Fall $e_i = 1$ ist. Um das zu erreichen, müssen die beiden folgenden Sätze erfüllt sein:

1. $e_i = 1$ kann grundsätzlich nur dann erreicht werden, wenn für jeden Abschnitt des Resultats gilt, daß dieser durch Zuordnung (d. h. gemäß dem Schema von Bild 4) aus den jeweils aktuellen Abschnitten der Argumente erzeugt werden kann, wobei in diese Zuordnung höchstens noch Zustands-Information einbezogen ist, die eindeutig durch die zuvor verarbeiteten Abschnitte bestimmt ist.

2. Um $e_i = 1$ praktisch verwirklichen zu können, müssen die Abläufe des Algorithmus so zerlegbar sein, daß jeder Abschnitt der Resultate sowie erforderlichenfalls die Zustands-Information im Sinne von Satz 1 ausschließlich durch kombinatorische Verknüpfungen gemeinsam selektierbarer Abschnitte der Argumente sowie der besagten Zustands-Information des jeweils vorausgegangenen Verarbeitungszyklus gebildet wird.

Satz 1 muß stets erfüllt sein, da nur so gewährleistet ist, daß zur Gewinnung eines Resultatabschnittes keine zusätzlichen Zustandswechsel erforderlich sind. Man braucht hingegen solche Zustandswechsel, wenn auch nur ein Resultatabschnitt von Argumentabschnitten abhängt, die nicht im jeweils aktuellen Zyklus zugänglich sind oder in vorausgegangenen Zyklen bereits verarbeitet wurden: diese Argumentabschnitte sind nur mit

¹ Die Bezeichnung aus /243/ wird zunächst beibehalten; sie wäre ggf. künftig durch Vergegenständlichungseffizienz ("efficiency of incarnation") zu ersetzen.

zusätzlichen Zugriffen erreichbar. Für $e_i = 1$ ist es aber nicht notwendig, daß ein Resultatabschnitt nur von den aktuellen Argument-Abschnitten abhängt: es können auch Abhängigkeiten von zuvor verarbeiteten Abschnitten bestehen; sofern diese über die Zustands-Information vermittelt werden können. Bild 12 zeigt, wie dafür das Schema von Bild 4 bzw. 5 zu erweitern ist

Die praktische Konsequenz von Satz 2 besteht darin, daß $e_i = 1$ nur von bestimmten Verarbeitungsbreiten an realisierbar ist, d. h. nur im Rahmen technischer Auslegungen, die gewährleisten, daß die jeweils erforderlichen Argument- und Zustandsbits parallel in den Speichermitteln selektiert und den Verknüpfungsnetzwerken zugeführt werden können.

Grundsätzlich ist die Implementierungseffizienz kleiner als 1, wenn:

1. die abschnittsweise Zuordnung technisch nicht zu verwirklichen ist (Aufwand, Kompliziertheit), so daß bestimmte Folgen von Maschinenzuständen notwendig sind, um einen Resultatabschnitt zu erzeugen

2. Resultatbits von Argumentbits abhängen, die sich in verschiedenen Abschnitten befinden können, so daß Zugriffe zu mehreren Argumentabschnitten nötig sind, um diese Resultatbits zu bestimmen

3. bestimmte Argumentbits veranlassen, daß Teile bereits erzeugter Resultatabschnitte geändert werden müssen, so daß erneute Zugriffe zu diesen Abschnitten auszuführen sind.

Ein Sachverhalt nach 1. ist nicht immer ein unüberwindliches Hindernis: es ist letztlich eine Ermessensfrage, was man als "zu aufwendig" oder "zu kompliziert" ansieht.

Hingegen bezeichnen die Sachverhalte 2. oder 3. objektive Grenzen: solche Algorithmen können nie mit $e_i = 1$ vergänglich werden. Ein plausibles Beispiel dafür ist das Umordnen eines Vektors gemäß den Angaben einer Indexliste: jede Argumentposition kann grundsätzlich in jede Resultatposition transportiert werden, so daß es notwendig sein kann, bereits abgespeicherte Resultatabschnitte nochmals aufzurufen, um neue Werte einzufügen.

Alle diese Überlegungen gehen von der grundsätzlichen Wünschbarkeit der direkten Zuordnung aus. Durch abschnittsweise Zuordnung soll dieser Ansatz technisch verwirklicht werden, und es ist klar, daß damit Schaltmittel in bestmöglicher Weise ausgenutzt werden können, nämlich durch lückenlose Folgen von Nutzoperationen. Läßt sich $e_i = 1$ von einer gewissen Mindest-Verarbeitungsbreite an verwirklichen, so heißt das, daß jede so ausgelegte Schaltung in jedem ihrer internen Zyklen einen Anteil zum Endresultat beiträgt, der ihrer Verarbeitungsbreite entspricht; man erhält also das Maximum an Verarbeitungsleistung, das mit den vorgesehenen Aufwendungen überhaupt zu verwirklichen möglich ist.

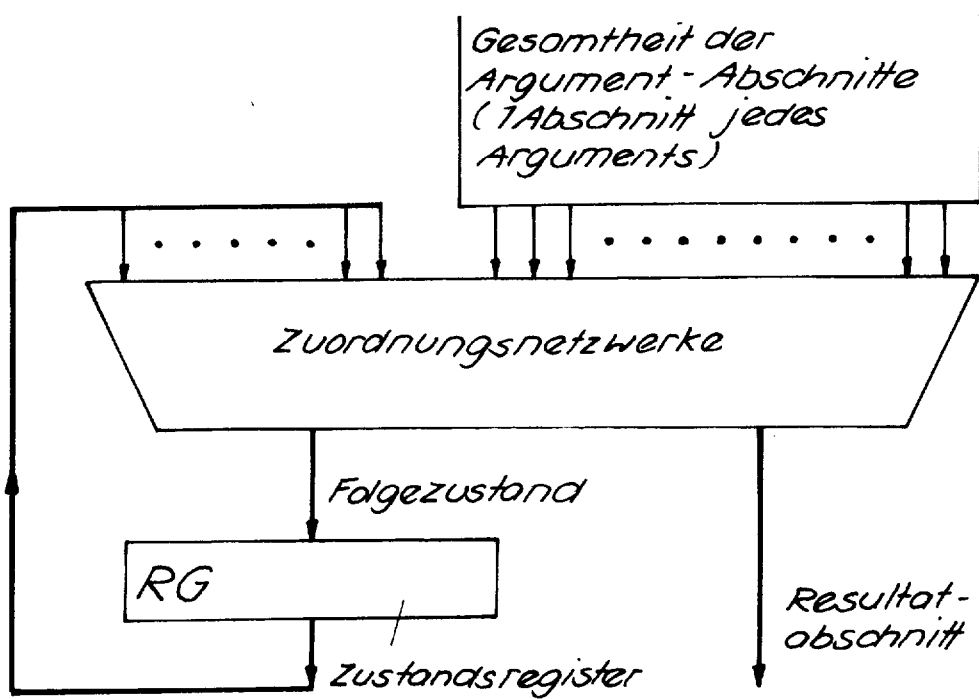
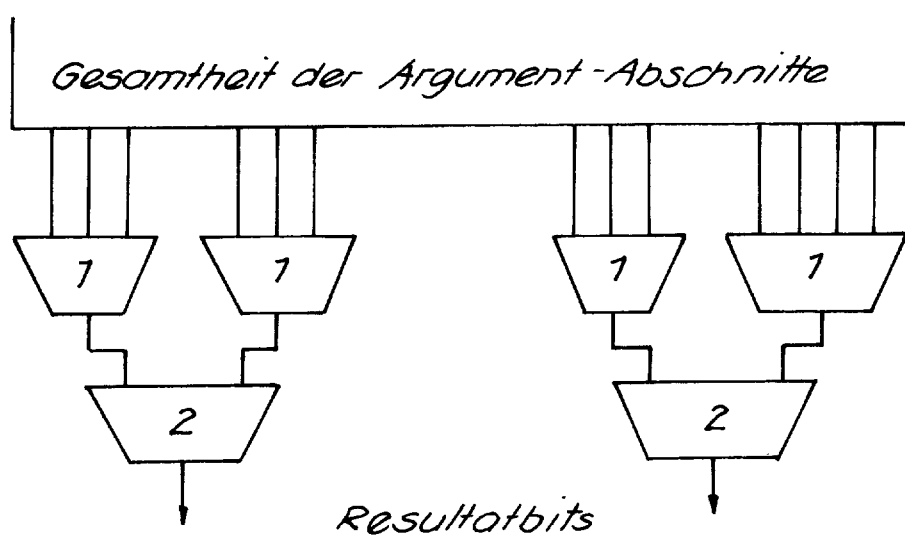


Bild 12 Prinzip der abschnittswisen Zuordnung unter Einbeziehung von Zustands-Information



*1: Netzwerke zur Verknüpfung unabhängiger Gruppen von Argumentbits
2: Netzwerke zur kombinatorischen Zusammenfassung*

Bild 13 Illustration der gruppenweisen Zerlegbarkeit

Die Frage, ob für einen gegebenen Algorithmus eine technisch-ökonomisch sinnvolle Vergegenständlichung gelingt, ist letzten Endes nur durch zielgerichtete erfinderische Bemühungen entscheidbar, also durch Versuche, entsprechende Schaltungsanordnungen auszuarbeiten. Im besonderen sollte man sich nicht darauf beschränken, einen prozedural beschriebenen Algorithmus lediglich schaltungstechnisch umzusetzen, sondern man sollte gleichsam vorurteilsfrei nach Lösungen suchen, die die gewünschten Wirkungen hervorbringen.¹ Ein solcher Weg kann recht langwierig sein; deshalb seien einige Kriterien angeführt, die oft ein überschlagsmäßiges Urteil ermöglichen:

1. Beherrschbare Leitungszahlen liegen in der Größenordnung von 32 bis etwa 512. Dem Schema von Bild 4 entsprechen direkt arithmetisch-logische Einheiten, die aus 2 Argumenten zu 32 oder 64 bit ein Resultat gleicher Länge erzeugen und zusätzlich einige Bedingungsleitungen erregen (z. B. "Übertrag", "Resultat = 0"). Die Grenzen sind im wesentlichen durch die beschränkten Kontaktzahlen an Schaltkreisen und durch Störeinflüsse gegeben (kapazitive und induktive Kopplung; impulsförmige Beeinflussung der Speisespannung, wenn viele Ausgangstreiber gleichzeitig schalten).²

2. Ob sich ein Zuordner technisch verwirklichen läßt, hängt von der Anzahl der Verknüpfungen, den technologischen Voraussetzungen und der Kompliziertheit der Booleschen Gleichungen ab, die die Verknüpfungen beschreiben.

3. Beliebig komplizierte Verknüpfungen können mit ROM- oder RAM- Zuordnern vergegenständlicht werden; die Variablenzahlen sind allerdings beschränkt:

- 8 bit/Verknüpfung: unproblematisch
- 12 bit/Verknüpfung: noch beherrschbar
- 16 bit/Verknüpfung: noch möglich.

Die absolute Obergrenze (wenn Kosten keine Rolle mehr spielen) liegt vielleicht bei 20...24 bit/Verknüpfung.³

4. Die Chancen der Realisierbarkeit steigen an, wenn es gelingt, die notwendigen Verknüpfungen in Gruppen zu zerlegen, die jeweils für sich realisierbar sind und die entweder voneinander unabhängig sind oder mit einfachen Mitteln verknüpft werden können.⁴ Das ist in Bild 13 veranschaulicht.

1 Das wurde in /241/ bzw. /243/ gezeigt. Man vergleiche die prozedurale Beschreibung des Algorithmus (/281/, /297/), deren technische Umsetzung in /123/ und die Lösung nach /241/, die durch abschnittsweise Parallelarbeit mit $e_i = 1$ deutlich (wenigstens 8-16 mal) schneller ist.

2 512-bit-Datenpfade sind schon ausgeführt worden (/151/).

3 1...16 Mbit je Resultat-Bitposition sind technisch durchaus noch beherrschbar (z. B. mit 1 bzw. 4Mbit DRAM).

4 Für Weiteres s. /243/. Beispiel: die Schaltung nach /123/, die aus einem beliebigen Binärvektor einen Vektor erzeugt, der nur die erste Eins enthält (Indexvektor).

Die Verhältnisse lassen sich genauer untersuchen, wenn man für den betreffenden Algorithmus eine Abhängigkeitsmatrix $|D|$ aufstellt. Das ist eine binäre Matrix, deren Zeilen den Bitpositionen aller Argumente entsprechen und deren Spalten den Bitpositionen aller Resultate. Hängt ein Resultatbit i von einem Argumentbit j ab (andernfalls: übt das Argumentbit j Einfluß auf das Resultatbit i aus), so steht in der Position ij der Matrix $|D|$ eine Eins, sonst eine Null. ($|D|$ repräsentiert nicht eine aktuelle Abhängigkeit, sondern alle grundsätzlich möglichen Abhängigkeiten bei beliebigen Werten.) Beispielsweise sieht die Matrix einer bitweise unabhängigen Verknüpfung von zwei 3-bit-Argumenten zu einem 3-bit-Resultat so aus:

$$|D| = \begin{array}{c} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{array} \begin{array}{|c|c|c|} \hline r_1 & r_2 & r_3 \\ \hline 1 & & \\ & 1 & \\ & & 1 \\ \hline 1 & & \\ & 1 & \\ & & 1 \\ \hline \end{array}$$

Die Spaltensummen von $|D|$ geben für jedes Resultatbit an, von wievielen Argumentbits es abhängt. Damit läßt sich abschätzen, ob direkte Zuordnungen realisierbar sind. Die Zeilensummen von $|D|$ geben für jedes Argumentbit an, auf wieviele Resultatbits es Einfluß hat. Damit läßt sich abschätzen, welche Aufwendungen für die Informationswege bzw. für das Selektieren der Argumentbits zu veranschlagen sind. Bildet man die elementweise Konjunktion zwischen 2 Spalten und erhält dabei keine einzige Eins, so können die beiden Resultatbits unabhängig voneinander gebildet werden: folglich lohnt es sich, die Schaltung so auszulegen, daß die benötigten Argumentbits parallel für die Verknüpfungen zugänglich sind bzw. abschnittsweise parallel selektiert werden können.

4.3. Bewertung der Aufwendungen

Aufwendungen, wie Datenwege, Speicher, Verknüpfungsschaltungen, sind nach ihrer Nützlichkeit zu bewerten. Ein Schaltungsentwurf wird auch danach beurteilt, wie gut die Schaltmittel im praktischen Betrieb ausgelastet werden. Eine bestimmte Aufwendung ist um so nützlicher, je mehr sie zum Leistungsvermögen der gesamten Anordnung beiträgt. Um diese Nützlichkeit beurteilen zu können, wird der Begriff der Mehraufwandseffizienz eingeführt. Dieser geht auf /243/ zurück, wo gezeigt wurde, daß damit plausible Überschlagsbetrachtungen möglich sind, um in der Konzeptionsphase schnell Entscheidungen zu treffen.¹ Zunächst wird für die einzelne Schaltungsanordnung eine Hardware-Effizienz HE wie folgt definiert:

$$HE = \frac{\text{Leistungsangabe}}{\text{Aufwandsangabe}}$$

¹ Z. B. auf Grundlage von Probeentwürfen bis zur Blockschaltbild- oder Register- Transfer- Ebene.

In der gesamten Anordnung sollte jede Komponente irgendeinen Zweck erfüllen, d. h. die elementweise disjunktive Verknüpfung aller Bedeckungsmatrizen sollte der binären Verbindungsmatrix Γ^b gleich sein:

$$\bigvee_{\nu=1}^n |C_{\nu}| = \Gamma^b$$

Ist das nicht der Fall, enthält die Anordnung überflüssige Verbindungen¹; diese sind durch elementweise Antivalenzverknüpfung sofort ersichtlich.

Für einen bestimmten Algorithmus \mathcal{U}_{ν} ist die Schaltung dann gut ausgenutzt, wenn $|C_{\nu}|$ möglichst viele Einsen enthält. Das wird durch den Ausnutzungsgrad ψ_{ν} für den jeweiligen Algorithmus \mathcal{U}_{ν} gekennzeichnet:

$$\psi_{\nu} = \frac{\sum_{i,j} |c_{ij}^{\nu}|}{\sum_{i,j} \Gamma_{ij}^b} ; 0 < \psi_{\nu} \leq 1$$

Diese Bewertungsweise allein wird ausgesprochenen Hochleistungsschaltungen nicht gerecht: jede kombinierte Auslegung bzw. Mehrfachnutzung von Schaltmitteln erhöht zwar den Ausnutzungsgrad, ist aber grundsätzlich mit Leistungsbeschränkungen verbunden. Für höchste Leistungsanforderungen sind erfahrungsgemäß die Vergegenständlichungen der einzelnen Algorithmen getrennt voneinander auszuführen² und in sich so auszubilden, daß das jeweils höchste Leistungsvermögen erreicht wird. Deshalb wird aus der Häufigkeit der Nutzung der einzelnen Algorithmen eine Ausnutzungsmatrix $|UE|$ über den Erwartungswert bestimmt:

$$|UE| = \sum_{\nu=1}^n p_{\nu} |C_{\nu}| ; \sum_{\nu=1}^n p_{\nu} = 1$$

Aus $|UE|$ lassen sich wichtige Schlüsse für die weitere Verfeinerung der Schaltungslösung ziehen. Dazu wird der Mittelwert \overline{UE} gebildet, und es wird eine Abweichungsmatrix $|\Delta|$ aufgestellt:

$$\overline{UE} = \frac{\sum_{i,j} |UE_{ij}|}{i,j} ; |\Delta| = |UE| - \overline{UE} \quad (\text{elementweise Subtraktion}).$$

Komponenten, die sich durch eine große negative Abweichung hervorheben (bzw. absolut gesehen: die in $|UE|$ \emptyset nahekommen), sind in ihrer Nützlichkeit grundsätzlich fragwürdig: es lohnt sich zu überprüfen, ob die so gekennzeichneten Schaltungsteile weggelassen werden können, wobei deren Aufgaben durch andere, an sich gegebene Einrichtungen übernommen werden.³

1 Fehlende Verbindungen sind ebenso erkennbar; dieser (praktisch bedeutsame) Gesichtspunkt wird hier vernachlässigt.

2 Meist werden gewisse Algorithmen zusammengefaßt und jeweils gemeinsame Schaltmittel vorgesehen (z. B. für Gleitkommaoperationen, Adressenrechnung usw.).

3 Z. B. durch mikroprogrammtechnische Emulation.

Komponenten mit außergewöhnlich großer positiver Abweichung (bzw. absolut: in $|UE|$ nahe bei 1) haben eine besondere Bedeutung für das Leistungsvermögen des Systems. Es lohnt sich deshalb zu untersuchen, ob durch Verfeinerungen oder gezielten Einsatz zusätzlicher Mittel die Gesamtleistung weiter verbessert werden kann.

4.4. Bewertung des Wirkungsgrades

Das Ziel ist die universelle Maschine, die vergegenständlichte Algorithmen als Ressourcen bereitstellt, um damit eine Vielfalt von - üblicherweise recht komplexen - Anwendungsalgorithmen implementieren zu können. Mit den bisher vorgeschlagenen Bewertungsgrundlagen läßt sich das absolute Leistungsvermögen angeben, es läßt sich beurteilen, welche Algorithmen sich zur Vergegenständlichung eignen, und es läßt sich bewerten, wie gut die Schaltmittel ausgenutzt sind. Hingegen ist der Anwender ausschließlich daran interessiert, wie schnell seine Aufgaben praktisch bearbeitet werden. Solche Angaben sind durch Schätzungen, Simulation oder Probeläufe zu ermitteln. Damit sind beispielsweise Preis- Leistungs- Vergleiche mit anderen Maschinen möglich.

Wenn man universelle Maschinen mit höchstem Leistungsvermögen schaffen will, braucht man aber den Vergleich mit absoluten Leistungsgrenzen. Anhand solcher Vergleichswerte ist dann die Sinnfälligkeit des Lösungsvorschlages beurteilbar. Im besonderen wird sich herausstellen, ob die vergegenständlichten Algorithmen tatsächlich zweckmäßig ausgewählt wurden. Dazu wird vorgeschlagen:

Es wird für jeden wesentlichen Algorithmenkomplex eine fiktive Sondermaschine ausgearbeitet (so detailliert, daß deren Leistungsvermögen beurteilbar ist). Unterstellt man für die Sondermaschine eine technisch gerade noch beherrschbare Auslegung¹, so wird deren Leistungsvermögen den absoluten Leistungsgrenzen sehr nahe kommen.

Dann läßt sich ein Wirkungsgrad η einführen, der das Verhältnis der Leistungen der zu beurteilenden Universalmaschine und der fiktiven Sondermaschine repräsentiert²:

$$\eta = \frac{P_{UNIV}}{P_{SPEZ}}$$

Damit läßt sich ein Ziel für das Entwerfen neuartiger Universalmaschinen angeben: ihr Leistungsvermögen sollte für die anwendungspraktisch wichtigsten Algorithmenkomplexe dem von einschlägigen Sondermaschinen soweit wie möglich entsprechen.

1 Es bietet sich an, dafür die Technologien und Aufwendungen der jeweils leistungsfähigsten kommerziell verfügbaren Supercomputer anzusetzen (in /243/ bereits ausgeführt).

2 Je nach Zweckmäßigkeit kann der Wirkungsgrad auf eine bestimmte Technologie oder Größenordnung des Aufwandes bezogen werden; es ist dann für die fiktive Sondermaschine dieselbe Technologie oder ein ähnlicher Kostenrahmen wie bei der zu vergleichenden Universalmaschine anzunehmen.

5. Tiefenstrukturen des Verarbeitungsmodells

Im folgenden werden wesentliche Einzelheiten des Verarbeitungsmodells Überblicksmäßig beschrieben, um jene Tatsachen, Zusammenhänge und Notwendigkeiten zu erkennen, die den technischen Ausgestaltungen zugrunde zu legen sind.

Bekannte, weit verbreitete Programmiersprachen und Rechnerarchitekturen liefern dafür das Erfahrungsmaterial¹; sie sind gleichsam Ausdruck der Oberflächenstrukturen, woraus die Tiefenstrukturen abstrahiert werden.² Dazu werden die Erfahrungsbereiche der numerischen und nichtnumerischen Informationsverarbeitung mit Universalrechnern genutzt, und es werden Datenstrukturen, Programmstrukturen sowie Prinzipien der Informationsspeicherung betrachtet.³

Das ist auf den ersten Blick eine eher konservative Grundlage; deshalb sind einige Anmerkungen zur Einbeziehung neuerer Forschungsrichtungen notwendig:

1. Für elementare ("low level") Operationen gibt es leistungsfähige und flexible Prinzipien der Befehlsgestaltung und Steuerung.⁴ Mit darauf beruhenden optimierten technischen Lösungen dürften sich Konzepte wie Lisp, Prolog oder Smalltalk wenigstens so effizient implementieren lassen, wie es dem Stand der Technik entspricht.

2. In Weiterführung der Arbeiten lassen sich künftig einschlägige Forschungsergebnisse (z. B. zu Prolog-Maschinen) bewerten und ggf. einbeziehen.

¹ Nach /146/ kommen als Erfahrungsbasis für neue Rechner-Befehlslisten in Betracht: Programmiersprachen, Anwendungslösungen, Betriebssysteme und die elementaren Befehle ("low level instructions"), die in fast allen Systemen zu finden sind. Betriebssysteme werden hier über hinreichend ausgestattete Programmiersprachen (z. B. Ada) mit erfaßt. Die Analyse von Anwendungslösungen bleibt weiterführenden Arbeiten vorbehalten.

² Die Begriffe stammen aus der Sprachwissenschaft; sie gehen u. a. auf Wittgenstein und Chomsky zurück (vgl. etwa die Erläuterungen in /45/, /54/). Die Oberflächenstrukturen sind durch die jeweilige Sprache bzw. durch das jeweilige konkrete Repräsentationssystem gegeben; die Tiefenstrukturen bringen das Wesen, die innewohnenden - bei Transformationen invarianten - Bedeutungen zum Ausdruck.

³ Für diesen Abschnitt wurden hauptsächlich folgende Quellen verwendet: /10/, /22/, /34/-/38/, /59/-/61/, /76/, /77/, /86/, /88/, /91/, /92/, /129/, /169/, /229/, /285/ zu Rechnerarchitekturen; /33/, /55/, /62/, /70/-/72/, /102/, /225/ zu Programmiersprachen.

Architektur- und Sprachbeschreibungen haben einen beachtlichen Umfang, so daß es unmöglich ist, in den folgenden Übersichten auf Einzelheiten einzugehen; dafür sei auf die jeweilige Original-Dokumentation verwiesen.

⁴ Beispiele: VLIW-Architekturen; Mikroprogrammsteuerungen großer EDV-Anlagen (370/168, 3081 u. a.).

3. Zur Vergegenständlichung von Funktionen der "Künstlichen Intelligenz" in Universalrechnern wird folgende vorläufige Arbeitshypothese vertreten:

Die Beschränkung auf den Prädikatenkalkül 1. Stufe (vgl. Prolog) ist völlig unzureichend, wenn man den Begriff ("KI") auch nur einigermaßen wörtlich nimmt. Vielmehr erscheint es notwendig, wenigstens einen n-stufigen Prädikatenkalkül zu implementieren¹ und darüber hinaus Vorkehrungen zu treffen, um auf einen gegebenen Informationsbestand ("Wissensbasis") verschiedene nichtklassische Logiken, wie modale, deontische, mehrwertige usw.² anwenden zu können.

Als Grundlage für die Implementierung eines n-stufigen Prädikatenkalküls eignen sich Operationen über binär codierte halbgeordnete Mengen. Dazu ist die Isomorphie zwischen dem Booleschen Verband $(B^k, \wedge, \vee, \neg)$ und dem Potenzmengenverband $(P(M), \cap, \cup, \bar{})$ sowie zwischen dem Booleschen Ring (B^k, \wedge, \oplus) und dem Potenzmengenring $(P(M), \cap, \triangle)$ technisch ausnutzbar.³

Wesentlich ist, daß sich die innersten Schleifen aller leistungsentscheidenden Algorithmen mit $e_i = 1$ vergegenständlichen lassen.⁴ International wird diese Richtung (Durchmustern von Mengen statt Anwenden von Regeln) mit dem Stichwort "memory based reasoning" bezeichnet.⁵

5.1. Datenstrukturen

5.1.1. Numerische Datenstrukturen

In Maschinenarchitekturen sind nur wenige elementare Datenstrukturen definiert:

- natürliche bzw. ganze Binärzahlen fester Länge; Bild 14 zeigt einige Beispiele
- Näherungsdarstellungen für reelle Zahlen; zumeist als binäre Gleitkommazahlen (Bild 15)
- stellenweise binär codierte Dezimalzahlen (Bild 16).

Daraus können Vektoren, Matrizen und andere zusammengesetzte Strukturen gebildet werden; das wird aber von den meisten

1 Ein entsprechender Vorschlag ist in /244/ bzw. in /245/, Anhang 5, näher skizziert.

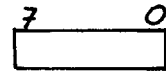
2 Zu nichtklassischen Logiken s. /5/, /18/, 93/.

3 Zu den theoretischen Grundlagen s. etwa /52/.

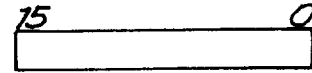
4 Das ist bereits in /240/ bzw. /243/ gezeigt worden. Für das hier skizzierte Vorhaben sind die Datenstrukturen, Algorithmen und Schaltungslösungen natürlich in den Einzelheiten zweckgerecht abzuwandeln.

5 Eine solche Lösung (mit einfacherer Zielstellung) wurde auf der Connection Machine implementiert (/311/). Ähnliche Ansätze zum parallelen Suchen in semantischen Netzen auf Grundlage spezieller Parallelverarbeitungs-Strukturen werden von anderen Forschungsgruppen bearbeitet (/190/, /193/). Die bisherigen Datenbasismaschinen bzw. Assoziativprozessoren sind praktisch als elementare Vorstufen für solche Lösungen anzusehen (vgl. z. B. /119/, /127/, /288/).

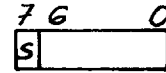
Natürliche Binärzahl, 8bit



Natürliche Binärzahl, 16bit



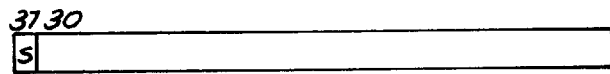
Ganze Binärzahl, 8bit



Ganze Binärzahl, 16bit



Ganze Binärzahl, 32bit



S=0: Null bzw. positive Zahl; S=1: negative Zahl

Bild 14 Beispiele für Formate natürlicher und ganzer Binärzahlen

*: 1. Mantissen-Stelle ist implizit stets 1

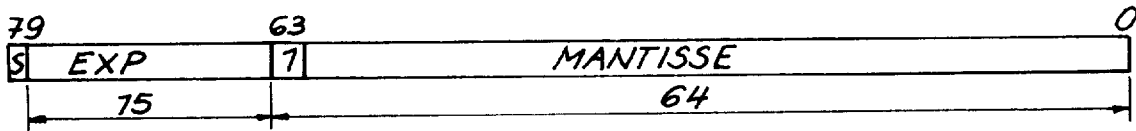
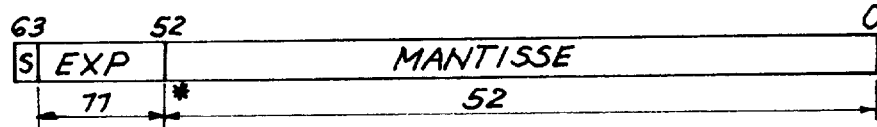
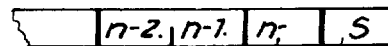
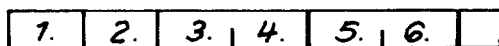


Bild 15 Beispiele für Gleitkommaformate (gemäß IEEE 754)



Stelle Vorzeichen

Je Byte 2 Dezimalstellen,
Länge 1-16 Bytes

Bild 16 Beispiel für das Format binär codierter Dezimalzahlen

Maschinenarchitekturen nicht direkt unterstützt. Nur bei Superrechnern ist die Datenstruktur "Vektor" in der Architektur definiert. Die Verarbeitungsgeschwindigkeiten unterscheiden sich beträchtlich:

- natürliche oder ganze Binärzahlen werden am schnellsten verarbeitet (zumeist in allen Stellen parallel; Addition bzw. Subtraktion in einem Taktzyklus)
- Gleitkommazahlen werden oft in allen Stellen parallel verknüpft, die Operationen erfordern aber mehrere Verarbeitungsschritte; höhere Leistungen sind nur mit speziellen Hardwarestrukturen zu erreichen (im Rahmen von Superrechnern oder als ergänzende Beschleunigungseinrichtungen)
- parallel wirkende Schaltmittel für binär codierte Dezimalzahlen sind zwar vorgeschlagen worden (so in /35/), sie haben sich aber nicht durchgesetzt. Manche Architekturen bieten lediglich Hilfsbefehle, um die Implementierung einer Dezimalarithmetik zu unterstützen; sie wirken aber auch bei großen Verarbeitungsbreiten oft nur für eine Dezimalstelle. Selbst in Maschinen mit ausgebauter Dezimalarithmetik sind die Grundoperationen meist nur seriell (byte- oder tetradenweise) implementiert.¹
- Vektoren werden in üblichen Rechnern sequentiell mittels Programmschleifen verarbeitet. In Superrechnern bzw. speziellen Zusatzprozessoren wird das Pipelining-Prinzip genutzt, so daß in jedem Maschinenzklus eine Komponente als Resultat erzeugt wird.² Bei manchen Architekturen sind die vergegenständlichten Vektor-Strukturen starken Beschränkungen unterworfen³, um die Maschinenzyklen extrem kurz halten zu können; bei anderen wurde hingegen Wert auf sehr flexible Strukturen gelegt⁴.

Tafel 5 zeigt eine Übersicht über numerische Datenstrukturen, die in eingeführten Programmiersprachen vorgesehen sind.

Tiefenstrukturen

Numerische Datenstrukturen sind sowohl für das eigentliche numerische Rechnen als auch für strukturell-deskriptive Angaben⁵ von Bedeutung, so beispielsweise für:

- Angaben zur Beschreibung und Lokalisierung von Datenstrukturen (Adressen, Längen, Grenzen usw.)
- Kardinalzahlen (Mächtigkeit von Mengen bzw. Anzahlen)
- Ordinalzahlen.

¹ Beispiele: 360/65, 370/158, EC 1040 und Nachfolger.

² Die Vektor-Komponenten sind binäre Gleitkommazahlen.

³ Beispiel: Cray-1 (maximal 64 Komponenten je Vektor).

⁴ Beispiel: Matrixmodul zu EC 1055.

⁵ Der Begriff wurde aus /300/ entlehnt, hat aber eine gewandelte Bedeutung: er bezeichnet codierte Angaben, die Strukturen anderer Angaben beschreiben (z. B. Felder nach Typ, Speicheradresse und Größe).

PL/1

Numerische Daten können nach Modus, Basis, Skala und Genauigkeit spezifiziert werden.

Modus: REAL oder COMPLEX
Basis: BINARY oder DECIMAL
Skala: FIXED oder FLOAT

Die Genauigkeit wird durch die Stellenzahl beschrieben (Gesamtanzahl und Stellen nach dem Komma).

Ada

Es können ganze Zahlen sowie Gleitkomma- und Festkommazahlen deklariert werden. Für alle Datentypen ist eine Bereichsangaben (range) möglich; für Gleit- und Festkommazahlen auch eine Genauigkeitsangabe:

- digits (relative Genauigkeit; Gleitkomma)
- delta (absolute Genauigkeit; Festkomma).

C

Es sind ganze (int), natürliche (unsigned int) und Gleitkommazahlen (float) verschiedener Länge vorgesehen:

- | | | |
|-------------|----------------------|----------|
| • short int | • unsigned short int | • float |
| • int | • unsigned int | • double |
| • long int | • unsigned long int | |

Tafel 5

Übersicht: Numerische Datenstrukturen in eingeführten Programmiersprachen

Dafür reichen natürliche Binärzahlen angemessener Länge aus.

Für das eigentliche numerische Rechnen sind Anforderungen zu erfüllen hinsichtlich der Genauigkeit, des Umfangs der Zahlenbereiche, des Speicherbedarfs und der Verarbeitungsgeschwindigkeit. Aus der Mathematik sind folgende grundlegende Zahlenbereiche bekannt:

1. die natürlichen Zahlen
2. die ganzen Zahlen
3. die rationalen Zahlen
4. die reellen Zahlen
5. die komplexen Zahlen.

Über jeden Zahlenbereich sind alle 4 Grundrechenarten definiert. Die entscheidende Aufgabe besteht darin, diese Bereiche auf finite Mengen binärer Codierungen abzubilden. Das ist mit 2 Beschränkungen verbunden:

1. man kann aus jedem Bereich nur ein endliches Intervall abbilden
2. die Intervalle der rationalen, reellen und komplexen Zahlen können nicht völlig exakt abgebildet werden.

Die Unzulänglichkeiten der üblichen Implementierungen waren Anlaß zu umfassenden Forschungsarbeiten (/22/, /77/, /134/). Deren Ergebnisse sollen im folgenden genutzt werden.

Es ist notwendig, numerische Rechnungen in folgenden Zahlenbereichen ausführen zu können:

1. natürliche sowie ganze Zahlen
2. reelle Zahlen
3. komplexe Zahlen
4. Intervalle über 2. und 3.
5. Vektoren und Matrizen über 2., 3. und 4.

In Tafel 6 sind die Bereiche 2. - 5. zusammen mit den grundlegenden Operationen dargestellt.

Jede Operation # ($\# \in \{+, -, *, /\}$) in jedem Zahlenbereich M (gemäß Tafel 6) wird auf eine korrespondierende maschineninterne Operation \square im jeweiligen Bereich der Maschinendarstellung N unter Wahrung folgender Eigenschaften abgebildet:

$$(5.1) \quad x \square y = \square(x\#y) \quad \text{für alle } x, y \in N.$$

$$(5.2) \quad \square x = x \quad \text{für alle } x \in N \text{ (Rundung)}.$$

$$(5.3) \quad x \leq y \text{ impliziert } \square x \leq \square y \text{ für alle } x, y \in M \text{ (Monotonizität)}.$$

$$(5.4) \quad \square(-x) = -\square x \text{ für alle } x \in M \text{ (Antisymmetrie)}.$$

Zahlenbereich	Struktur	Operationen
reell	Skalar	+ - * /
	Vektor	+ - *
	Matrix	+ - *
reelles Intervall	Skalar	+ - * /
	Vektor	+ - *
	Matrix	+ - *
komplex	Skalar	+ - * /
	Vektor	+ - *
	Matrix	+ - *
komplexes Intervall	Skalar	+ - * /
	Vektor	+ - *
	Matrix	+ - *

Tafel 6 Zahlenbereiche, Strukturen und Operationen für das numerische Rechnen

- 1) $\boxed{+}$ $\boxed{-}$ $\boxed{*}$ $\boxed{/}$ $\boxed{\cdot}$ (Semimorphismus)
- 2) \triangleup \triangleleft $\triangle*$ $\triangle/$ $\triangle\cdot$ (Aufwärtsgerichtete monotone Rundungen)
- 3) \triangledown \triangledown $\triangledown*$ $\triangledown/$ $\triangledown\cdot$ (Abwärtsgerichtete monotone Rundungen)

Skalarprodukt $\sum a_i b_i$

Bild 17

Grundlegende Operationen als Voraussetzung des numerischen Rechnens

Ist M eine Menge von Intervallen so bezeichnet \llcorner die mengentheoretische Inklusion \subseteq , und die Rundung hat die zusätzliche Eigenschaft

(5.5) $x \llcorner \square x$ für alle $x \in M$ (aufwärtsgerichtete Rundung).

Eine solche Abbildung \square stellt eine Annäherung an einen Homomorphismus dar. (5.2) ist eine natürliche Eigenschaft, die jede Rundung haben sollte. Es kann gezeigt werden, daß (5.1), (5.3) und (5.5) notwendige Bedingungen für einen Homomorphismus zwischen geordneten algebraischen Strukturen sind. Eine solche Abbildung \square wird deshalb als Semimorphismus bezeichnet.

Bild 17 zeigt, welche elementaren Operationen notwendig sind, um alle Verknüpfungen gemäß Tafel 6 mit der erforderlichen Genauigkeit ausführen zu können. Wesentlich ist, daß zwischen dem korrekten Resultat und seiner Annäherung durch die maschineninterne Darstellung kein weiterer Wert der maschineninternen Darstellung vorkommt. Werden diese Anforderungen erfüllt, so sind an sich beliebige Codierungen zulässig. Die Codierung kann also ausschließlich im Hinblick auf eine hohe Implementierungseffizienz gewählt werden.

5.1.2. Nichtnumerische Datenstrukturen

In den verbreiteten Maschinenarchitekturen sind nur Zeichenketten und Binärvektoren (Bitfelder) vorgesehen. Zeichen sind üblicherweise in 8-bit-Bytes codiert¹ (wichtige Codes sind ASCII und EBCDIC bzw. DKOI II). Zeichenketten sind Aneinanderreihungen von Bytes.²

Ein Binärvektor (Bitfeld) entspricht normalerweise einem Maschinenwort (z. B. von 32 bit Länge), das als Ansammlung einzelner Bits interpretiert wird.

Höhere Programmiersprachen bieten darüber hinaus Aufzählungstypen an und gestatten es, vielfältige zusammengesetzte Datenstrukturen (records) zu deklarieren.

Tiefenstrukturen

An sich sind nur Binärvektoren variabler Länge (vom einzelnen Bit an) und Aufzählungstypen zu betrachten. Datenstrukturen vom Aufzählungstyp sind geordnete endliche Mengen. Ist eine solche Menge einmal beschrieben, läßt sich jedes ihrer Elemente durch seine Ordinalzahl codieren.

Auch Zeichen können als Datenstrukturen vom Aufzählungstyp angesehen werden; die geordnete endliche Menge ist dann das jeweilige Alphabet (so ist beispielsweise der übliche ASCII-Zeichensatz im Standard-Package der Sprache Ada definiert). Mit diesem Konzept ist man nicht zwingend an eine starre Byte-

¹ In älteren Architekturen (z. B. ICL 1900, CDC 3600) waren 6-bit-Codes üblich; einige Maschinen haben bei 36 bit Wortlänge 9-bit-Bytes (Univac 1100, S1).

² Die Länge der Zeichenkette ist üblicherweise in Deskriptoren (iAPX 432) oder direkt im Befehl (S/360) angegeben; früher waren auch Endemarken üblich (1401, R 300).

Struktur gebunden, sondern man kann für jedes Alphabet eine jeweils angemessene Länge des Zeichencode-Formates festlegen.¹ Dies dürfte die praktischen Schwierigkeiten mit mehrsprachigen bzw. landesspezifischen Zeichensätzen radikal und elegant beheben.²

Alle komplexeren Strukturen (records) sind heterogener Art. Es ist eine Angelegenheit der Implementierung, ob die innere Struktur ausschließlich durch programmseitige Zugriffe repräsentiert wird (prozedurale Darstellung³) oder durch zusätzliche strukturell-deskriptive Angaben (deklarative Darstellung⁴).

5.2. Programmstrukturen

In Verallgemeinerung bekannter Ansätze lassen sich alle Programmstrukturen auf 4 Abstraktionen zurückführen:

1. Operatoren liefern Resultate durch Verarbeitung gegebener Argumente
2. Selektoren wählen bestimmte Angaben aus gespeicherten Informationsstrukturen aus
3. Iteratoren liefern nacheinander Folgen von Angaben
4. Aktivatoren bestimmen, unter welchen Bedingungen und in welcher zeitlichen Reihenfolge Operatoren, Selektoren und Iteratoren nacheinander zur Wirkung kommen.

Der Begriff des Operators ist in Mathematik und Informatik seit langem üblich (vgl. beispielsweise /74/).

Der Begriff des Selektors wird in grundlegenden Arbeiten zu abstrakten Objekten verwendet (so in /335/). Hier wird er so aufgefaßt wie in der Programmiersprache Clu: als Abstraktion jeglicher Auswahlvorgänge.

Der Begriff des Iterators wurde in der Programmiersprache Clu als verallgemeinerte Abstraktion für Programmschleifen (for-loops) eingeführt.

Sinngemäß wird der Begriff des Aktivators als zusammenfassende Bezeichnung angewendet, die beispielsweise Verzweigungen, Unterprogrammaufrufe und Unterbrechungen einschließt.

1 Es geht hier um die Effizienz der internen Speicherung und Verarbeitung. Wandlungen zu Ein- und Ausgabezwecken bereiten keine Schwierigkeiten; sie sind z. B. durch Blocktransporte mit Tabellenzugriffen ("Move Translated") zu implementieren.

2 Einen Eindruck von diesen Schwierigkeiten gewinnt man anhand der Firmenschriften einschlägiger Gerätesysteme, wie IBM 3270 oder EC 7920.

3 Der Compiler erzeugt mit den Angaben der record-Deklaration unmittelbar die Befehlsfolgen.

4 Der Compiler erzeugt aus der record-Deklaration Deskriptoren (andere Bezeichnung: Dope-Vektoren, z. B. beim PL/1-Laufzeitsystem), die von Unterprogrammen oder von Befehlen (z. B. bei iAPX 432) genutzt werden.

5.2.1. Operatoren

In Tafel 7 sind die Operatoren einer verbreiteten Maschinenarchitektur angeführt.¹

Tafel 8 zeigt die Operatoren der Programmiersprache Ada.

Tiefenstrukturen

Verglichen mit Programmiersprachen enthalten Maschinenarchitekturen oft mehr elementare Operatoren.² Die Tiefenstrukturen werden also durch die Befehlslisten der bewährten Rechner gut repräsentiert. Die Vielfalt der Operatoren läßt sich folgendermaßen ordnen:

1. arithmetische Operatoren (+, -, *, /, Divisionsrest, Vorzeichenwechsel, Vergleich)
2. logische Operatoren, die bitweise unabhängig wirken (AND, OR, XOR usw.)
3. sonstige (einschließlich Transporte, Bittests, Setzen von Einzelbits, Transporte mit tabellengesteuerter Umcodierung, Ermitteln der ersten Eins in einem Bitfeld usw.).

5.2.2. Selektoren

In den üblichen Programmiersprachen ist die Selektion durch Benennung bzw. Indizierung (bei Array-Strukturen) gegeben. Es lassen sich nur Informationsstrukturen selektieren, die in der Sprache definiert sind.

In den Maschinenarchitekturen wird die Selektion durch die Adressierung verwirklicht. Im allgemeinen ist das Byte die kleinste adressierbare Einheit; Wort- oder Bitadressierung sind wenig verbreitet. In Bild 18 und Tafel 9 sind eingeführte Adressierungsprinzipien dargestellt. Es gibt eine beachtliche Vielfalt von Auffassungen darüber, welche Adressierungsprinzipien in einer Rechnerarchitektur vorgesehen werden sollten; die Verfahren nach Bild 18 bzw. Tafel 9 repräsentieren nur eine dieser Auffassungen. Beachtenswerte Extreme sind:

1. Beschränkung auf die Form "Basisregister + vorzeichenbehafteter 16-bit-Offset". Das wird z. B. in /139/ als ausreichend angesehen, da viele Programme von 0 verschiedene ("non zero") Offsets verwenden.³

2. Adressierung unter Nutzung eines Kellerspeichers, wobei ein vollständiger Satz von Adressierungsmodi definiert ist (Tafel 10). Das Ziel besteht darin, Konstrukte höherer Programmiersprachen so direkt wie möglich in die Befehlsliste abzubilden: Beziehungen zwischen Namen im Befehl und temporären Speicherplätzen sollen redundanzfrei und ohne Zwang zur Einführung zusätzlicher Befehle hergestellt werden können.⁴

1 VAX 11 (CISC mit besonders reichhaltigem Befehlsvorrat).

2 Deshalb ist es manchmal unumgänglich, auf die Maschinensprache zurückzugreifen.

3 Es ist eine Basisadresse von 32 bit vorgesehen.

4 Das Konzept geht namentlich auf Flynn zurück (/168/, /169/).

Add
 Add packed decimal string
 Add with carry
 Add one and branch
 Arithmetic shift
 Arithmetic shift and round packed decimal string
 Bit clear
 Bit set
 Bit test
 Clear
 Compare numeric
 Compare character string
 Compare packed decimal string
 Compare variable bit field to integer
 Convert data types
 Decrement
 Divide
 Divide packed decimal string
 Extended Divide
 Extended Multiply
 Extract variable bit field
 Find first bit in variable bit field
 Increment
 Index (calculate array index)
 Insert entry into queue
 Insert integer into variable bit field
 Locate character
 Match character string
 Move complemented
 Move negated
 Move numerical quantities
 Move address
 Move character string
 Move packed decimal string
 Move translated characters
 Move zero extended numerical quantities
 Multiply
 Multiply packed decimal string
 Remove entry from queue
 Rotate longword
 Scan character string
 Span character string
 Subtract
 Subtract packed decimal strings
 Subtract with carry
 Subtract one and branch
 Test
 Exclusive OR

Tafel 7

Operatoren einer verbreiteten Maschinenarchitektur (Maschinenbefehle, die Daten verändern bzw. Bedingungen abfragen)

<p><u>Logische Operatoren</u></p> <p>and or xor and then: verkürzte Konjunktion¹ or else: verkürzte Disjunktion¹</p> <hr/> <p>1 Der 2. Operand wird erst dann ausgewertet, wenn das Ergebnis nicht bereits durch den 1. Operanden zu bestimmen ist.</p>
<p><u>Relationale Operatoren</u></p> <p>= /= < <= > >= in not in: Prüfung, ob ein Wert zu einem Typ oder Subtyp gehört bzw. in einem Bereich (range) liegt.</p>
<p><u>Zweistellige additive Operatoren</u></p> <p>+ - &: Konkatenation von 2 arrays; Erweiterung eines array mit einem Element; Bilden eines array durch Konkatenation von 2 Elementen</p>
<p><u>Einstellige additive Operatoren</u></p> <p>+ -</p>
<p><u>Multiplikative Operatoren</u></p> <p>* / mod rem</p> <p><u>Operandenkombination</u> für * und /: integer/integer; float/float; fixed/integer; integer/fixed; fixed/fixed mod und rem: integer/integer</p>
<p><u>Operatoren mit höchstem Vorrang</u></p> <p>abs: Absolutwert not: logische Negation **: Potenzierung; Operandenkombinationen: integer/positive integer; float/integer</p>

Tafel 8

Operatoren der Programmiersprache Ada

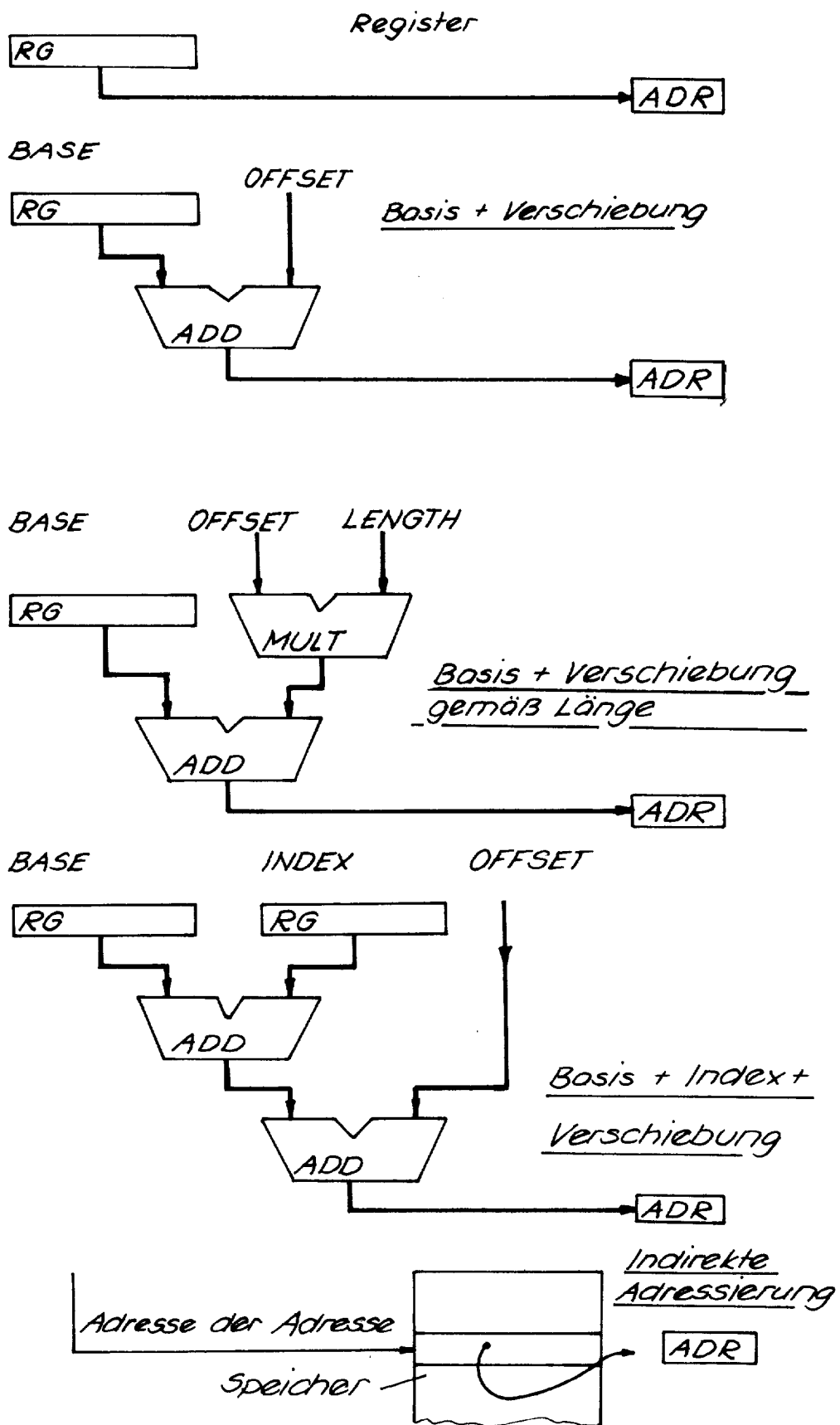


Bild 78 Adressierungsprinzipien

Bezeichnung	Der Operand ist gegeben durch
Short Literal	N (Direktwert im Befehl)
Index	$\langle (b + s * \langle Rn \rangle) \rangle$
Register	$\langle Rn \rangle$
Register deferred	$\langle \langle Rn \rangle \rangle^{\#}$
Autodecrement	$\langle \langle Rn \rangle \rangle$; zuvor $\langle Rn \rangle := \langle Rn \rangle - s^{\#}$
Autoincrement	$\langle \langle Rn \rangle \rangle$; danach $\langle Rn \rangle := \langle Rn \rangle + s^{\#}$
Autoincrement deferred	$\langle \langle \langle Rn \rangle \rangle \rangle$; danach $\langle Rn \rangle := \langle Rn \rangle + s^{\#}$
Displacement	$\langle (\langle Rn \rangle + D) \rangle^{\#}$
Displacement deferred	$\langle \langle (\langle Rn \rangle + D) \rangle \rangle^{\#}$
- Adressierung auf Befehlszähler bezogen (R 15) -	
Immediate	N
Absolute	$\langle A \rangle$
Relative	$\langle (\langle PC \rangle + D) \rangle$
Relative deferred	$\langle \langle (\langle PC \rangle + D) \rangle \rangle$

$\langle \dots \rangle$ Inhalt von...
Rn Register
N Direktwert
A Absolutadresse (im Befehl)
s Operandengröße (1, 2, 4, 8, 16 Bytes)
D Displacement (Byte, Wort, Langwort)
b Basisadresse des Index Mode
 $\#$ An die Adressenangabe im Befehl kann noch ein Indexregister (für den Index Mode) angefügt werden.

Tafel 9

Adressierungsverfahren einer verbreiteten Maschinenarchitektur

Format	Verknüpfung	Explizite Operanden
A B C	A <u>op</u> B -> C	3
A B B	A <u>op</u> B -> B	2
A B A	A <u>op</u> B -> A	2
A A A	A <u>op</u> A -> A	1
A B T	A <u>op</u> B -> T	2
A T B	A <u>op</u> T -> B	2
T A B	T <u>op</u> A -> B	2
T A A	T <u>op</u> A -> A	1
A T A	A <u>op</u> T -> A	1
A A T	A <u>op</u> A -> T	1
A T T	A <u>op</u> T -> T	1
T A T	T <u>op</u> A -> T	1
T T A	T <u>op</u> T -> A	1
T U A	T <u>op</u> U -> A	1
T U T	T <u>op</u> U -> T	Ø

A, B, C explizite Operanden
T erste Stack- Position (Top of Stack)
U zweite Stack- Position (auf T folgend)

Besondere PUSH- und POP- Befehle sind nicht nötig:

- T als Resultat veranlaßt automatisch PUSH
- T als Argument veranlaßt automatisch POP.

Tafel 10

Ein vollständiger Satz von
Adressierungsmodi (nach Flynn)

3. objektorientierte Zugriffsorganisation; in den Bildern 19 und 20 ist ein Beispiel veranschaulicht.¹ Zur Implementierung höherer Programmiersprachen sind solche Zugriffsverfahren in irgendeiner Form stets notwendig (je nach Sprachkonzept mit unterschiedlicher Bedeutung und Nutzungshäufigkeit): wenn die Maschinenarchitektur sie nicht zur Verfügung stellt, müssen sie im Laufzeitsystem programmtechnisch vorgesehen werden.²

Tiefenstrukturen

Letztlich geht es stets darum, für jeden Zugriff eine einzige Angabe zum Speicher zu liefern, die faktisch die Ordinalzahl der gewünschten elementaren Informationsstruktur darstellt. Diese Ordinalzahl kann nach verschiedenen Algorithmen ermittelt werden, wozu vornehmlich Additionen und Tabellenzugriffe gehören.

5.2.3. Iteratoren

Einen Überblick über Iterator-Konstrukte in verbreiteten Programmiersprachen gibt Tafel 11.

In der Sprache Clu ist die Iterator-Abstraktion explizit enthalten. So kann in den eigentlichen Programmschleifen davon abstrahiert werden, wie die Angaben bei aufeinanderfolgenden Durchläufen herangeschafft bzw. abgespeichert werden.

In Maschinenarchitekturen sind meist nur recht elementare Iteratoren vorgesehen, z. B.:

- Blockoperationen (Transporte und Vergleiche)
- Wiederholungsbefehle³
- Blocktransporte für Graphik-Bitmuster: Bitblock-Transporte BITBLT, CLIP (nur Null-Werte werden geschrieben), DRAW (nur von Null verschiedene Werte werden geschrieben), Transporte mit Hintergrund-Unterdrückung (Werte, die kleiner sind als der aktuelle Inhalt der jeweiligen Zielposition, werden nicht geschrieben) usw.⁴
- automatische Adressenrechnungen im Rahmen der Befehlsausführung, wobei die Inhalte von Adressenregistern geändert werden (Autoincrement, Autodecrement usw.). Besonders bedeutsam sind Zugriffe zu Kellerspeichern (Stacks): meist ist PUSH ein "Pre-decrement" vor dem Schreiben, POP ein "Postincrement" nach dem Lesen.⁵

1 Die Bilder betreffen iAPX 432. Solche Prinzipien gibt es auch in anderen Architekturen (z. B. B 5500...6900).

2 Zur Geschwindigkeit vgl. S. 20, Punkt 6.

3 Beispiel: Befehl DJNZ des Mikroprozessors Z 80.

4 Neuerdings in verschiedenem Umfang vergegenständlicht (Intel 80860, "Transputer" T 800, Stellar u. a.).

5 Bei manchen Architekturen mit jedem Universalregister ausführbar, bei manchen nur mit den Stackpointer-Registern. Die Überwachung der Bereichsgrenzen ist erst neuerdings vergegenständlicht (68 030).

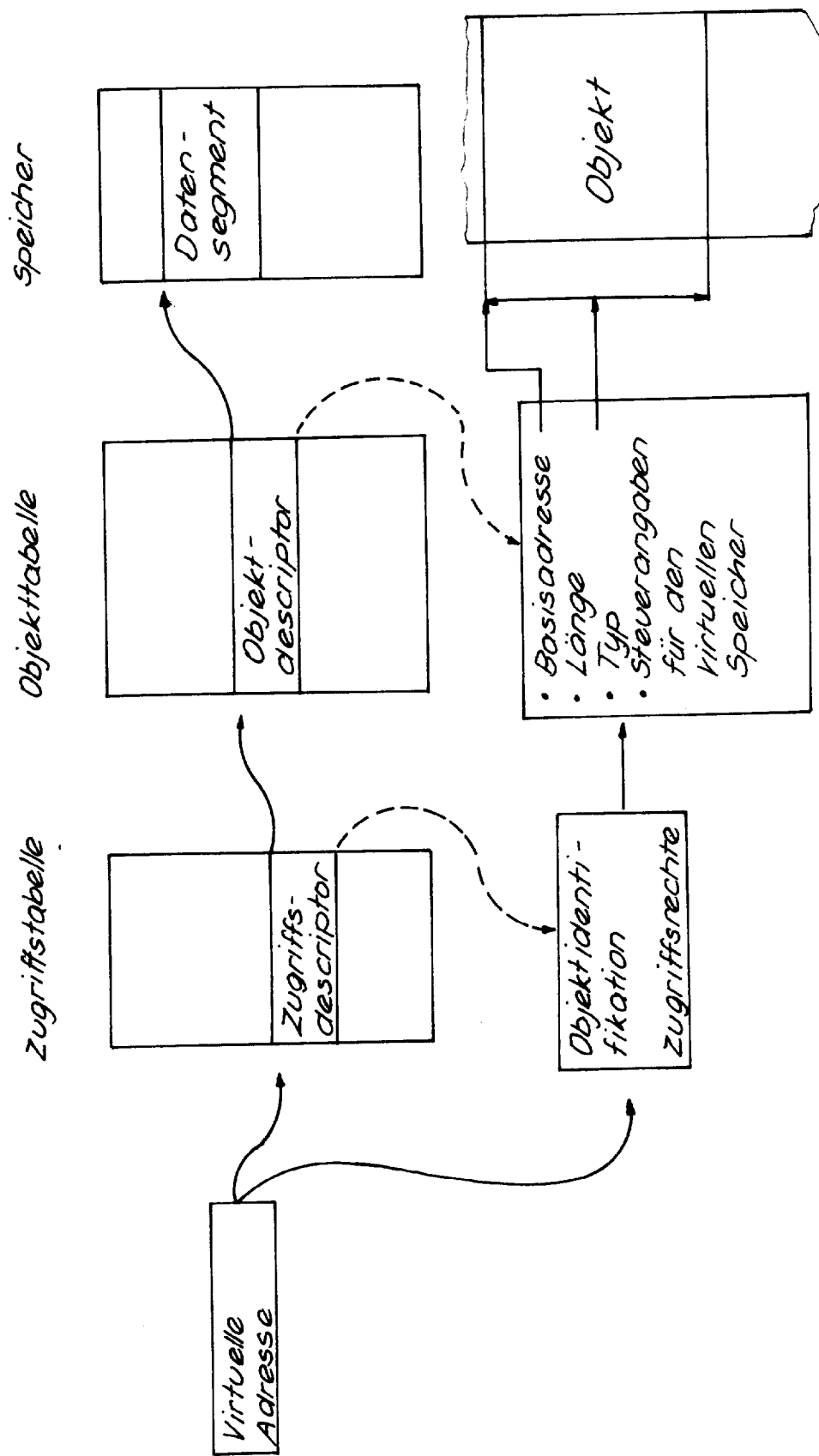


Bild 19 Prinzip der objektorientierten Zugriffsorganisation
(Bsp. iAPX 432)

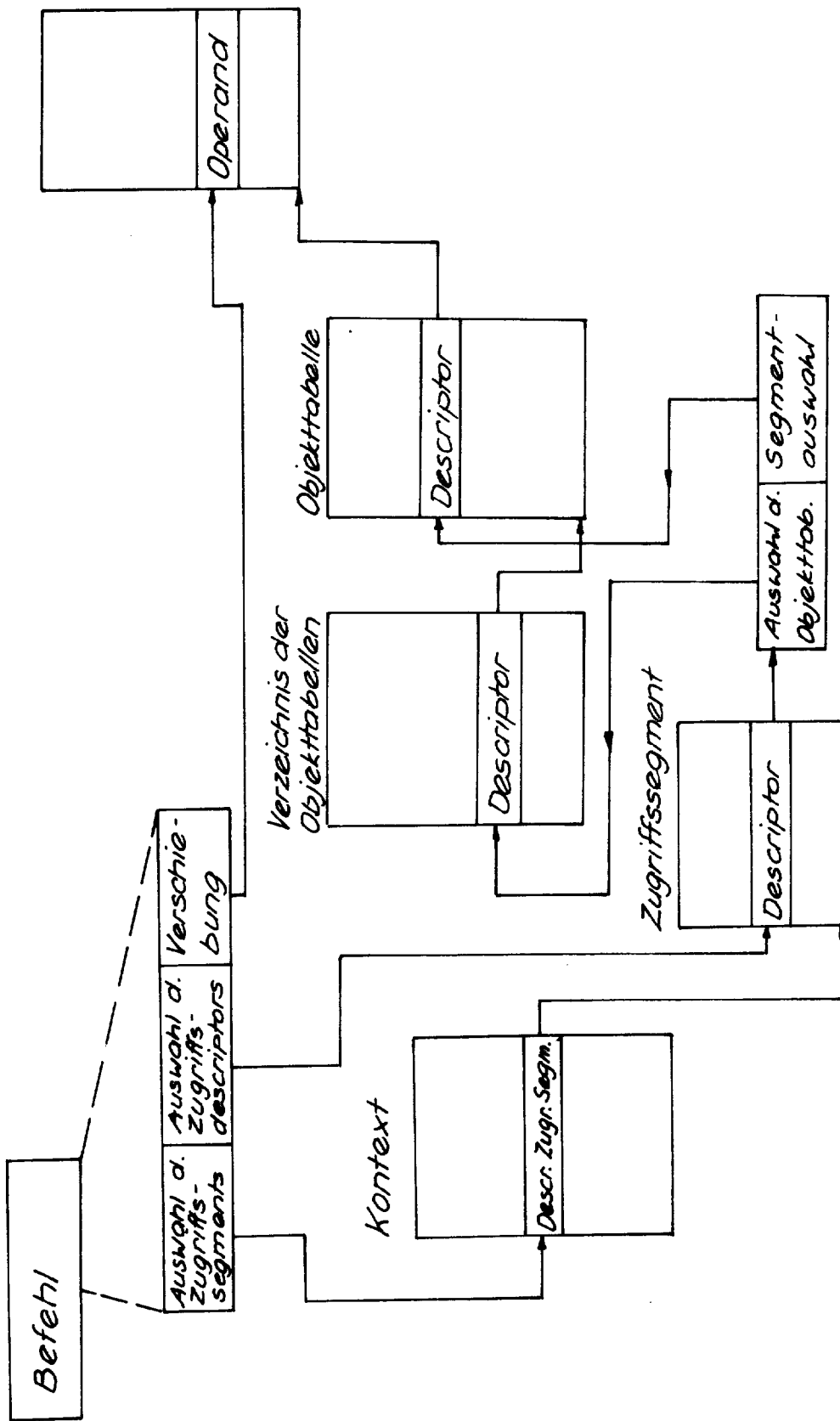


Bild 20 Schema der Adressierung eines Operanden beim Mikroprozessor iAPX 432

PL/1

DO

DO WHILE (Bedingung)

DO Variable = Anfangswert TO Endwert BY Schrittweite

Ada

while Boolescher Ausdruck loop

.....

end loop

for Variable in (reverse) Bereich loop

.....

end loop

loop

.....

end loop

C

while (Ausdruck)
Anweisung...

for (Initialisierung, Endebedingung, Zählweisung)
Anweisung...

do
Anweisung...
while (Ausdruck)

● Vektoroperationen. Tafel 12 gibt einen Überblick über solche Operationen, die in einem Spezialprozessor¹ vorgesehen sind; weitere Anregungen sind aus den Tafeln 13 - 15 ersichtlich. Tafel 13 zeigt einige elementare Schleifen, die beim numerischen Rechnen häufig verwendet werden (nach /290/). In Tafel 14 sind jene Muster für den Zugriff auf zusammengesetzte Strukturen aufgezählt, die in der numerischen Analysis am meisten gebraucht werden (nach /199/).² Tafel 15 wurde nach den Vorschlägen in /61/ zusammengestellt, die von der Sprache APL beeinflusst sind.

Tiefenstrukturen

Für einen Iterator ist kennzeichnend, auf welche Weise die aufeinanderfolgenden Angaben selektiert werden; der Iterator ist praktisch ein Algorithmus zum Erzeugen von Selektionsangaben. Solche Algorithmen können aus der Sicht des Anwenders recht komplex ausfallen, so daß tatsächlich nur ganz wesentliche Abläufe vergegenständlicht werden können. Diese betreffen in der Regel das Erhöhen bzw. Vermindern von Adressenangaben, wobei die Endbedingung durch das Erreichen eines Endwertes, durch Ausschöpfen eines Zählwertes oder eine in der Schleife ermittelte Bedingung (z. B. durch arithmetischen Vergleich) gegeben ist. Als elementare Iteratoren bieten sich an:

- Blockoperationen mit konstanter Adressenerhöhung bzw. -verminderung (auch besondere; beispielsweise Zeichenketten-Transporte mit Umcodierung, Transporte für Graphik- Bitmuster, Blockvergleiche)
- elementare Schleifen, wie sie beim numerischen Rechnen gebräuchlich sind (vgl. die Tafeln 12- 15)
- Zugriffsprinzipien für Kellerspeicher (LIFO) und Warteschlangen (FIFO).

5.2.4. Aktivatoren

Es ist zwischen synchronen und asynchronen Aktivatoren zu unterscheiden. Synchrone Aktivatoren sind Verzweigungen verschiedener Art, einschließlich der Unterprogrammaufrufe, der Supervisoraufrufe und der programmseitig ausgelösten Unterbrechungen; asynchrone Aktivatoren sind externe Unterbrechungen oder Maßnahmen zur Behandlung unvorhersehbarer Ausnahmefälle.

In Programmiersprachen sind synchrone Aktivatoren in Form der Prozedur- bzw. Funktionsaufrufe und in Form von Konstrukten wie IF...THEN...ELSE, CASE..., GOTO... u. a. gegeben. Asynchrone Aktivatoren sind in vielen Programmiersprachen nicht ohne weiteres zugänglich. Manche Sprachen (z. B. PL/1 und Ada)

¹ IBM 2938.

² Tafel 14 soll dazu anregen, sowohl die Speicherorganisation als auch die vergegenständlichen Iteratoren anwendungsgerichtet auszugestalten.

1. Elementweise Multiplikation von Vektoren
$Y(i) := X(i) * U(i)$
2. Elementweise Addition von Vektoren
$Y(i) := X(i) + U(i)$
3. Skalarprodukt von Vektoren
$Y := \sum_i X(i) * U(i)$
4. Summe der Elemente eines Vektors
$Y := \sum_i X(i)$
5. Summe der Quadrate der Elemente eines Vektors
$Y := \sum_i X(i) * X(i)$
6. Multiplikation mit Faltung
$Y(i) := \sum_j U_j * X(i + j - 1)$

Tafel 12

Elementare Vektoroperationen eines Spezialprozessors

$A(i) := 1$	$A(i) := \sin(B(i))$
$A(i) := B(i)$	$A(i) := \arcsin(B(i))$
$A(i) := B(i) + 10$	$A(i) := \text{abs}(B(i))$
$A(i) := B(i) + C(i)$	Umordnen:
$A(i) := B(i) * 10$	$\{C(i) := A(i);$
$A(i) := B(i) / 10$	$A(i) := B(i);$
$A(i) := B(i) / C(i)$	$B(i) := C(i).\}$
$A(i) := \max(B(i), C(i))$	
$A(i) := B(i) * C(i) + D(i)$	
$A(i) := B(i) * C(i) + D(i) * E(i)$	

Tafel 13

Elementare Schleifen des numerischen Rechnens

1. eine Zeile einer Matrix
2. eine Spalte einer Matrix
3. die Hauptdiagonale einer quadratischen Matrix
4. der Zeilenabschnitt einer quadratischen Matrix, der der oberen Dreiecksmatrix entspricht ("half row")
5. der Zeilenabschnitt einer quadratischen Matrix, der der unteren Dreiecksmatrix entspricht ("half row")
6. der Vektor, der durch die geraden Elemente eines Vektors gebildet wird
7. der Vektor, der durch die ungeraden Elemente eines Vektors gebildet wird
8. die transponierte Matrix
9. die "Fläche" eines Würfels
10. eine Matrix, die durch Extraktion der ungeraden Elemente der ungeraden Zeilen einer Matrix gebildet wird
11. Untermatrizen

Tafel 14

Wichtige Zugriffsmuster für
das numerische Rechnen