GÜNTSCH · DIGITALE RECHENAUTOMATEN

EINFÜHRUNG IN DIE PROGRAMMIERUNG DIGITALER RECHENAUTOMATEN

VON

DR. FRITZ RUDOLF GÜNTSCH

ZWEITE, ERWEITERTE UND NEUBEARBEITETE AUFLAGE

WALTER DE GRUYTER & CO.

VORMALS G. J. GÖSCHEN'SCHE VERLAGSHANDLUNG – J. GUTTENTAG
VERLAGSBUCHHANDLUNG – GEORG REIMER – KARL J. TRÜBNER – VEIT & COMP.

BERLIN 1963

AM

Vorwort

Die vorliegende zweite Auflage der *Einführung in die Programmierung digitaler Rechenautomaten* ist vollständig umgearbeitet und dabei wesentlich erweitert worden. Dies hat vor allem zwei Gründe: Einmal gewinnen eine Reihe von Programmierungstechniken, wie die Verwendung problemorientierter Sprachen oder die Parallelprogrammierung immer größere praktische Bedeutung, so daß auch ein einführendes Buch mit dem wachsenden Stoff Schritt halten muß. Zum zweiten sind seit dem Erscheinen der ersten Auflage eine Reihe kürzerer und in ihrer Stoffwahl verhältnismäßig begrenzter deutschsprachiger Einführungen in die Programmierungstechnik — teils als selbständiges Buch, teils als Teile von Einführungen in das Gesamtgebiet der Digitalrechenanlagen — erschienen. Daher glaube ich, daß ein gewisses Bedürfnis für ein etwas umfassenderes Programmierungsbuch vorhanden ist. Gleichzeitig konnte ich bei dieser Gelegenheit die von manchen Kritikern zu Recht bemängelte übergroße Knappheit einiger Teile der ersten Auflage mildern.

Als wichtigste Erweiterung gegenüber der ersten Auflage sind einige Kapitel über nicht maschinengebundene Programmsprachen und über Parallelprogrammierung hinzugekommen. Die Darstellung der problemorientierten Sprache wurde konsequent auf ALGOL ausgerichtet, so daß die vorliegende Schrift nicht nur eine elementare Einführung in ALGOL 60 enthält, sondern in ihrer ganzen Anlage und Terminologie auf diese Sprache ausgerichtet wurde. Bei der Auswahl der deutschen Terminologie habe ich versucht, im besonderen Maße die im Fachnormenausschuß für Informationsverarbeitung im Deutschen Normenausschuß bisher erarbeiteten vorläufigen Ergebnisse zu benutzen.

Bei der Darstellung der Parallelprogrammierung kam es mir darauf an, die vielfältigen und neuartigen Probleme dieses noch jungen Arbeitsgebietes, dessen zukünftige praktische Bedeutung kaum überschätzt werden kann, aufzuzeigen. Gerade bei diesen Teilen des Buches zeigt sich jedoch, daß die auch in der zweiten Auflage mit gutem Grund als Beispiel-Maschine beibehaltene Z 22 bei einer weiteren Ausdehnung des Stoffes, insbesondere auf die in der kommerziellen Datenverarbeitung außerordentlich wichtigen Ein- und Ausgabevorgänge, nicht mehr ganz genügt. Daher habe ich die Ein- und Ausgabeorganisation im Rahmen der Parallelarbeit nur grundsätzlich be-

handelt, ohne auf die praktischen Systeme der Ein- und Ausgabe von Magnetbändern, Lochkarten usw. einzugehen. Ich habe mich aber trotzdem bemüht, durch die Wahl der Programmbeispiele (Platzbuchung, Postscheck-Gebühren, Sortierverfahren, Listensuchverfahren u. ä.) auch den an der kommerziellen Datenverarbeitung interessierten Lesern gerecht zu werden.

Bei der Behandlung der Z 22 selbst habe ich gern Herrn Professor *J. Dörrs* Ratschlag befolgt, den Befehlscode von vornherein stärker der echten Z 22 anzupassen. Das äußert sich vor allem in der Behandlung der G-Befehle. Weiter habe ich einem Hinweis Herrn Professor *W. Haacks* folgend bei der symbolischen Adresse die Konvention des *Berliner-Codes* übernommen.

Das vorliegende Buch ist zweifellos kein vollständiges Lehrbuch der Programmierungstechnik, und ich muß daher zur Vertiefung und Ergänzung auf die vielfältige Zeitschriften-Literatur verweisen. Ich möchte jedoch daneben ausdrücklich auf die — zwar sehr knappe — aber außerordentlich präzise und reichhaltige Darstellung von *H. Bottenbruch* im *Taschenbuch der Nachrichtenverarbeitung* [1] hinweisen, in der eine Reihe von hier nur elementar behandelter Techniken genauer und vollständiger beschrieben ist.

Mein besonderer Dank gilt Herrn Dr. *W. Händler* in Saarbrücken und seinen Mitarbeitern, Herrn Dipl.-Math. *J. Schneider* und Herrn cand. math. *Dieter Jurksch*, für viele wichtige Anregungen und Hinweise sowie für die mühevolle Überprüfung und teilweise Neugestaltung sämtlicher Programmbeispiele. Ohne diese wesentliche Hilfe wäre es für mich ein praktisch aussichtsloses Unterfangen gewesen, das Manuskript in einer vernünftigen Zeitspanne einigermaßen fehlerfrei zu erstellen. Mein Dank gilt darüber hinaus Herrn Dipl.-Math. *K. Friedrich* in Berlin für einige interessante Ratschläge, sowie den Herren *P. Aust* und Dipl.-Math. *H. Kampl* in Konstanz für ihre Hilfe beim Korrekturlesen und bei der Fehlersuche in den Programmbeispielen.

Weiter möchte ich der Firma *Telefunken* für die Genehmigung danken, diese Schrift zu veröffentlichen. Es ist mir eine angenehme Pflicht, an dieser Stelle festzustellen, daß nicht zuletzt die vielseitigen und interessanten Aufgaben, mit denen ich es bei dieser Firma zu tun habe, auch für die vorliegende Arbeit Anregung und Grundlagen bilden.

Dem Verlag *de Gruyter* gilt mein besonderer Dank für die überaus angenehme Zusammenarbeit an dieser wegen der nicht enden wollenden Korrekturen und des komplizierten Satzes schwierigen Publikation.

Konstanz, April 1963 F. R. Güntsch

Inhaltsverzeichnis

1 Einleitung

1.1 Die verschiedenen zur Vorbereitung der Rechnung erforderlichen Schritte

Jede umfangreiche numerische Rechnung erfordert eine ausführliche Vorbereitung. Unabhängig von der Art der Aufgabenstellung und unabhängig davon, ob für die Rechnung gewöhnliche Tischrechenmaschinen oder programmgesteuerte Rechenanlagen eingesetzt werden, können wir diese Vorbereitungen folgendermaßen gliedern:

1. Beschreibung der Aufgabenstellung. Der erste Schritt besteht in einer klaren Formulierung der Aufgabe. In vielen Fällen werden wir nicht in der Lage sein, die Aufgaben unmittelbar zu formulieren, sondern wir werden es mit Zusammenhängen zu tun haben, die überhaupt erst mit einem mathematischen Kalkül erfaßt werden müssen. Es muß ein mathematisches Modell gefunden werden, das es erlaubt, die Aufgabe in einer Weise zu beschreiben, die als Grundlage für eine weitere rechnerische Behandlung geeignet ist.

Beispiel: Es soll die Strömung durch das Schaufelgitter einer Turbine berechnet werden. Für die hier vorliegenden physikalisch-technischen Zusammenhänge stehen eine Reihe von mehr oder weniger brauchbaren mathematischen Modellen zur Verfügung. Für bestimmte Zwecke ist es zum Beispiel ausreichend, als mathematisches Bild die ebene, inkompressible, wirbelfreie Strömung zu benutzen. Damit ergibt sich als mathemathische Formulierung eine Randwertaufgabe der ebenen Potentialtheorie.

In anderen Fällen besteht der erste Schritt darin, weitschweifige Beschreibungen, die sich der *natürlichen Sprache*[1]) bedienen, durch Verwendung der üblichen mathematischen Formelzeichen straffer zu fassen. Dies ist insbesondere bei kaufmännischen Aufgaben der Fall.

2. Beschreibung des numerischen Verfahrens. Ist die Aufgabe formuliert, so müssen wir als nächstes ein numerisches Verfahren aussuchen oder auf-

[1]) Wir nennen hier die gewöhnlichen Sprachen, wie Englisch, Französisch und Deutsch, im Gegensatz zu den Programmsprachen *natürliche Sprachen.*

stellen, mit dessen Hilfe die Aufgabe gelöst werden kann. So ist es zum Beispiel nicht damit getan, eine Aufgabe in Form einer gewöhnlichen Differentialgleichung zu beschreiben. Die Berechnung der Lösung verlangt ein numerisches Verfahren, wie zum Beispiel das bekannte Verfahren von *Runge* und *Kutta*. Als Vorbereitung für die spätere Benutzung dieses Verfahrens bei der eigentlichen Rechnung muß das Verfahren genau formuliert werden. Diese, meist formelmäßige, Notierung eines Rechenverfahrens nennen wir einen *Algorithmus*[1]). Er enthält alle zur Lösung der Aufgaben nötigen Angaben, so daß die Rechnung auch durchgeführt werden kann, wenn man die ursprüngliche Aufgabenstellung nicht kennt. Wird beispielsweise das Verfahren von *Runge* und *Kutta* algorithmisch notiert, so folgen daraus die zur Lösung der Differentialgleichung erforderlichen Rechenschritte, ohne daß die Differentialgleichung selbst in Erscheinung tritt.

Die einzelnen Verfahrensschritte werden üblicherweise im Algorithmus in Form von Gleichungen notiert. Diese Gleichungen sind allerdings nicht wie die üblichen mathematischen Gleichungen, die auch bei der Aufgabenformulierung verwendet werden, symmetrisch, sondern sie haben den Charakter der von *Zuse* [3], [4] eingeführten *Plangleichungen* mit einem unsymmetrischen Gleichheitszeichen, dem sogenannten *Ergibtzeichen* (⇐).

Auf diese Weise erhalten die bis dahin statischen Gleichungen einen *dynamischen* Charakter, das heißt sie beschreiben nicht beliebige Zusammenhänge, sondern *Abläufe*.

Beispiel: Ein numerisches Verfahren möge unter anderem die Berechnung von

$$y = \frac{x^3}{(a+x)^2}$$

für $x = 0;\ 0{,}2;\ 0{,}4 \ldots 2{,}0$ erfordern.

Betrachten wir diese Gleichung, so können wir nicht ohne weiteres entscheiden, welche der Größen a, x oder y bekannt sind und welche mittels dieser Gleichung berechnet werden sollen. Mit Hilfe des Ergibtzeichens schreiben wir

$$y \Leftarrow \frac{x^3}{(a+x)^2}.$$

Das soll folgendes bedeuten: Der Wert des auf der rechten Seite des Ergibtzeichens stehenden Ausdruckes $x^3 / (a+x)^2$ soll berechnet und

[1]) Genaueres über Algorithmen und deren Beziehung zu Rechenautomaten ist z. B. bei [2] zu finden.

der links stehenden Variablen y zugeordnet werden. Auf diese Weise ist aus der statischen Gleichung eine Plangleichung geworden.

3. Die eigentlichen Rechenanweisungen. Ist das Verfahren in algorithmischer Form notiert, so können wir die eigentlichen Rechenanweisungen aufstellen. Das sind die Anweisungen, die vom Rechner unmittelbar verstanden werden. Die Form solcher Anweisungen hängt davon ab, ob eine Rechenanlage eingesetzt wird oder ob sich diese Anweisungen an Menschen richten, die die Rechnung mit Tischrechnern durchführen sollen.

Wird die Aufgabe von Hand gerechnet, so bestehen die Rechenanweisungen im allgemeinen aus einem *Rechenformular*, in das zeilenweise alle Rechenschritte eingetragen werden.

Beispiel: Wollen wir den im vorigen Abschnitt (*2*) genannten Ausdruck berechnen, so können wir das Rechenformular auf Seite 12 benutzen.

Auch eine Rechenanlage benötigt detaillierte Rechenanweisungen, die im wesentlichen den einzelnen Zeilen des Rechenformulars entsprechen. Solche Anweisungen an die Rechenmaschine heißen *Maschinenbefehle*, und eine Folge solcher Maschinenbefehle, die eine vollständige Rechenanweisung darstellen, ein *Maschinenprogramm*.

Da Rechenanlagen außerordentlich komplizierte Rechenabläufe durchführen können, gewinnen einige Phasen der Programmherstellung, die beim Handrechnen nicht weiter ins Gewicht fallen, besondere Bedeutung. Das gilt insbesondere für das *Ausprüfen von Programmen*. Beim Handrechnen wird man sich durch einige Stichproben davon überzeugen, daß das Rechenformular keine Fehler enthält. Die komplizierten Programmabläufe in einer Rechenanlage verlangen dagegen sorgfältige Prüfverfahren, und es ist meist sehr mühsam, Programmfehler ausfindig zu machen. Dabei gilt: *Je präziser die Aufgaben und das Verfahren formuliert werden, desto seltener treten Fehler auf.*

1.2 Programmsprachen und Flußdiagramme

Die Vorbereitung der Rechung besteht also im wesentlichen darin, die Formulierung der Anweisungen Schritt für Schritt immer näher an den Rechner heranzuführen.

Diese Anweisungen zur Lösung einer Aufgabe nennt man, wenn Rechenanlagen dazu herangezogen werden — und auf diesen Fall wollen wir uns nunmehr beschränken —, ganz allgemein ein *Programm*. Programme können auf verschiedene Weise formuliert werden, und ein System zur Formulierung

Zeilen Nr.	Rechen-operation	Zahlenwerte der Ergebnisse der Rechenoperationen											Bemerkungen
		Für $i=0$	1	2	3	4	5	6	7	8	9	10	
1	—	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	a = konst.
2	—	0	0,2	0,4	0,6	0,8	1,0	1,2	1,4	1,6	1,8	2,0	x_i
3	<1> + <2>	0,3	0,5	0,7	0,9	1,1	1,3	1,5	1,7	1,9	2,1	2,3	$a + x_i$
4	<3> × <3>	0,09	0,25	0,49	0,81	1,21	1,69	2,25	2,89	3,61	4,41	5,29	$(a + x_i)^2$
5	<2> × <2>	0	0,04	0,16	0,36	0,64	1	1,44	1,96	2,56	3,24	4,00	x_i^2
6	<2> × <5>	0	0,008	0,064	0,216	0,512	1,000	1,728	2,744	4,096	5,832	8,000	x_i^3
7	<6> : <4>	0	0,032	0,1306	0,2666	0,4231	0,5917	0,768	0,9494	1,135	1,322	1,5125	$y_i \leftarrow \frac{x_i^3}{(a + x_i)^2}$
8	<2> + 0,2 →2												x_{i+1}
9	*Zurück nach* 3												

Die Abkürzung $<n>$ bedeutet „Zahl, die in der Zeile mit der Nummer n steht", oder kurz *‚Inhalt von n'*. Die Operation in der dritten Zeile heißt also: Man addiere die Zahlen aus Zeile 1 (a) und Zeile 2 (x_i) und trage das Ergebnis in Zeile 3 ein. Das Zeichen $z \rightarrow n$ bedeutet: Die Zahl z soll in die Zeile n eingetragen werden, oder kurz *‚Transport von z nach n'*.

eines Programmes nennt man eine *Programmsprache*. Wir können also die im vorigen Kapitel geschilderte Vorbereitung einer Rechnung beschreiben, indem wir sagen, das Programm sei in verschiedenen Sprachen formuliert worden.

Bei den Sprachen, die im Umgang mit Rechenanlagen zweckmäßigerweise benutzt werden, unterscheidet man

1. *problemorientierte Sprachen*
2. *maschinenorientierte Sprachen*
3. *Maschinensprachen*.

Die *problemorientierten Sprachen* gestatten es, das Programm ohne Bezug auf einen bestimmten Rechner zu formulieren; sie benutzen vielmehr Sprachelemente, die für die Beschreibung eines bestimmten Problemkreises besonders gut geeignet sind. Ihre Benutzung ist praktisch erst möglich geworden, als *H. Rutishauser* vor etwa zehn Jahren begann, zur Übersetzung zwischen den einzelnen Programmsprachen die Rechner selbst zu benutzen [5].

Die im vorigen Kapitel aufgeführten Vorbereitungsschritte „Beschreibung der Aufgabenstellung" und „Beschreibung des numerischen Verfahrens" benutzen problemorientierte Sprachen. Dabei ergibt sich die Unterscheidung zweier wichtiger Teilbereiche innerhalb der problemorientierten Sprachen, nämlich

a) die *beschreibenden Sprachen* (*deskriptive Sprachen*)
b) die *Verfahrenssprachen* (*operative Sprachen*).

Die beschreibenden Sprachen beschreiben lediglich die Aufgaben (zum Beispiel in Form von statischen Gleichungen), während die Verfahrenssprachen oder *algorithmischen Sprachen* das Rechenverfahren selbst darstellen[1]).

Die *maschinenorientierten* Sprachen nehmen bereits Rücksicht auf die Eigenheiten der Rechenanlagen. Sie sind eine Zwischensprache zwischen den problemorientierten Sprachen und den eigentlichen *Maschinensprachen*, die dem Rechner unmittelbar verständlich sind. Die Maschinensprachen sind meist für den Menschen unhandlich, weil sie ganz bestimmte technische Eigenschaften elektronischer Rechenautomaten berücksichtigen müssen. Die maschinenorientierten Sprachen sind zwar nicht maschinenunabhängig,

[1]) Manche Autoren [6], [7] nennen die operativen problemorientierten Sprachen *Verfahrenssprachen* und die deskriptiven problemorientierten Sprachen *problemorientierte Sprachen*.

sie bieten jedoch eine Reihe von Bequemlichkeiten gegenüber den Maschinensprachen, so daß die Programmierer mit Rechenanlagen meist nicht in der Maschinensprache, sondern in einer maschinenorientierten Sprache verkehren, sofern sie überhaupt eine maschinengebundene Darstellung verwenden.

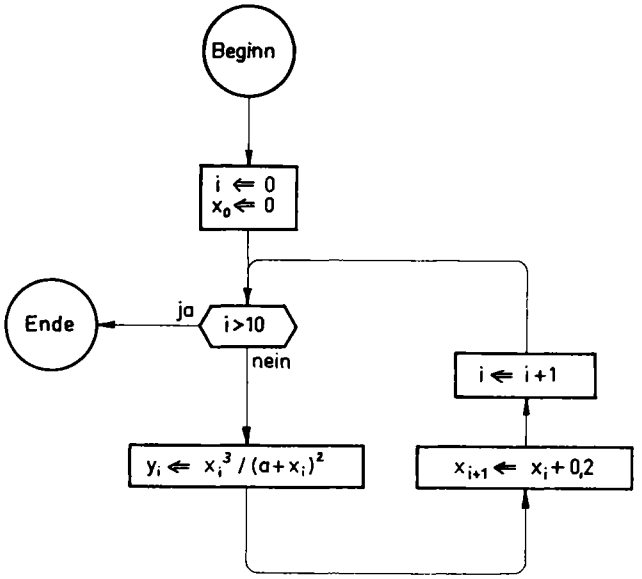
Wir haben soeben, ausgehend vom Begriff *Maschinenprogramm*, ganz allgemein den Begriff *Programm* eingeführt. Je nach der benutzten Sprache können wir dann Spezialfälle, wie Maschinenprogramme, algorithmische Programme usw., unterscheiden.

Wir wollen nun mit dem Begriff *Befehl* entsprechend verfahren. Ein Befehl ist eine innerhalb der betrachteten Sprache elementare Anweisung an den Rechner, das heißt eine Anweisung, die innerhalb der betrachteten Sprache nicht mehr aus einzelnen Anweisungen zusammengesetzt ist. Je nachdem, in welcher Sprache wir uns gerade bewegen, sprechen wir z. B. von *Maschinenbefehlen* oder von Befehlen eines in einer maschinenorientierten Sprache abgefaßten Programms.

Beispiel: Eine Verfahrenssprache möge in ihren Formeln die Benutzung des Funktionsnamens *sin* zulassen. Damit sei die Berechnung der Sinusfunktion nach vorher vereinbarten Regeln gemeint. Eine solche Anweisung kann in der betreffenden algorithmischen Sprache eine elementare Anweisung, also ein Befehl sein. Wird dieses Verfahren nun aber auf einer Rechenanlage durchgerechnet, auf der dieser Sinus-Wert mit Hilfe einer *Tschebyscheffschen* Approximation in mehreren Schritten berechnet wird, so muß das Maschinenprogramm die zur Berechnung des Sinus nötigen Einzelschritte als einzelne Anweisungen enthalten. Das heißt, *sin* ist für das Maschinenprogramm keine elementare Anweisung. Die dafür benötigten einzelnen Anweisungen der Maschinensprache können beispielsweise Additions- und Multiplikationsbefehle sein.

Ein wichtiges Hilfsmittel zum Verständnis der Programme sind die sogenannten *Flußdiagramme*. Das sind graphische Darstellungen der Strukturmerkmale von Abläufen. Wir beschränken uns hier auf die Betrachtung von *Befehlsflußdiagrammen*, also von Flußdiagrammen, die einen Programmablauf verfolgen. Die spezielle Gestalt des Flußdiagramms hängt davon ab, in welcher Sprache das darzustellende Programm notiert ist. Auch Flußdiagramme können also mehr oder weniger *maschinenorientiert* sein.

In der folgenden Abbildung ist ein zu dem Programm von Seite 12 gehöriges Flußdiagramm angegeben.

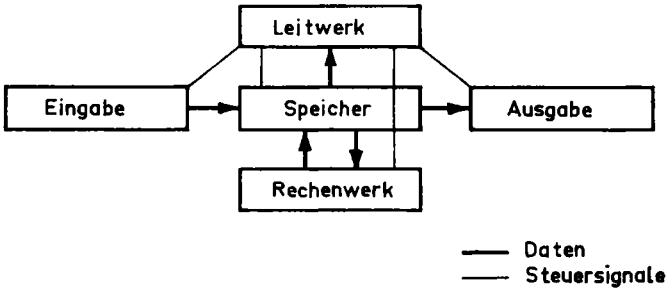


Die einzelnen darin aufgeführten rechteckigen Kästchen enthalten die eigentlichen Rechenschritte, während die beiden Kreise den Beginn und das Ende des Programmes kennzeichnen. Die Pfeile markieren die Reihenfolge der einzelnen Verfahrensschritte. Zu Anfang wird dem Index i und dem ersten Argument x_0 der Wert 0 zugeteilt. Der darauffolgende sechseckige Kasten stellt eine *Verzweigung* dar. Er hat einen Eingang und — im Gegensatz zu den übrigen Kästen — *zwei* Ausgänge. Die in dem Verzweigungskasten angegebene Bedingung prüft, ob der laufende Indexwert bereits größer als zehn ist. Ist dies — wie im Falle $i = 0$ — *nicht* der Fall, so wird der mit *nein* gekennzeichnete Ausgang benutzt. Das heißt, es werden zunächst nacheinander zwei weitere Kästen durchlaufen, in denen der Funktionswert y_i und das folgende Argument x_{i+1} berechnet werden.

Anschließend wird der Kasten mit der Anweisung $i \Leftarrow i + 1$ durchlaufen. Dort wird der nächste Indexwert — beim ersten Durchlauf also der Wert 1 — berechnet und wieder der Indexvariablen i zugeordnet. Dann wird aufs neue geprüft, ob i größer als zehn ist. Beim letzten Durchlauf ist i gleich zehn. Es werden also y_{10} und (überflüssigerweise) x_{11} berechnet, sodann wird i der Wert elf erteilt. Jetzt stellt das Verzweigungselement jedoch fest, daß i größer als zehn ist, und es wird der mit *ja* bezeichnete Ausgang durchlaufen, der zum Programmende führt.

1.3 Kurze Beschreibung einer programmgesteuerten digitalen Rechenanlage

Wir haben bisher eine Rechenanlage nur als ein Gerät gekennzeichnet, das mit Hilfe eines Maschinenprogramms umfangreiche numerische Rechnungen abwickeln kann. Um uns den Zugang zu unserem eigentlichen Thema, der Programmierung solcher Rechner, nicht unnötig zu erschweren, wollen wir uns auf die Betrachtung digitaler Nachrichtenverarbeitungs-Geräte[1]) etwa folgender Struktur beschränken:



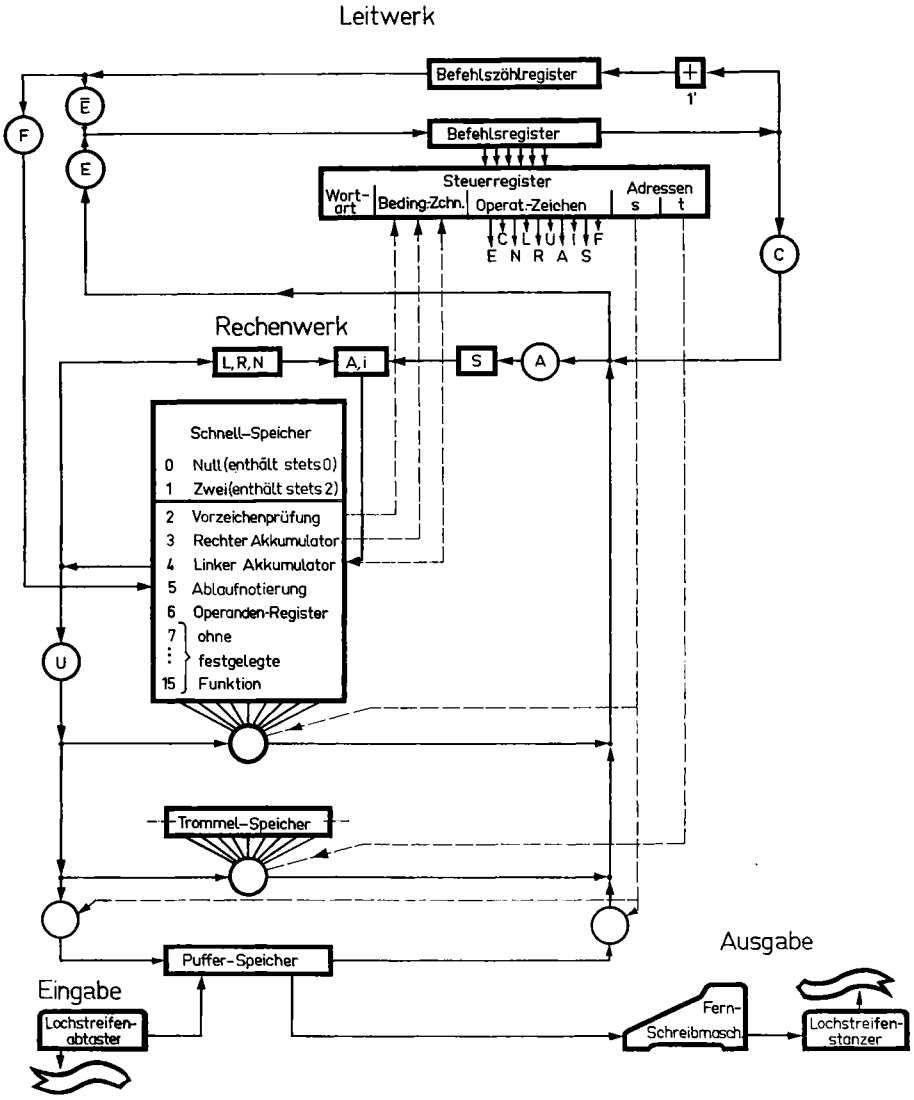
Die in dieser Skizze eingetragenen Verbindungslinien können auch etwas anders verlaufen. So kann zum Beispiel ein direkter Weg vom Rechenwerk zum Leitwerk vorhanden sein. Es soll hier nur eine typische Konfiguration gezeigt werden. Es ist wichtig, zwischen den (stark gezeichneten) Verbindungswegen für *Daten*, wie Operanden und Befehle, und den (schwach gezeichneten) *Steuersignalen* zu unterscheiden.

Der *Speicher* ist in der Lage, beliebige Zahlen- und Zeichenfolgen aufzunehmen und bei Bedarf wieder abzugeben. Er entspricht also dem für die Niederschrift von Rechenanweisungen[2]), Ausgangswerten, Zwischenergebnissen und Endresultaten benutzten Formular und den etwa für die Rechnung benötigten Funktionstabellen. Der Speicher ist in einzelne *Speicherzellen* (oder Speicherplätze) aufgeteilt. In jeder Speicherzelle können wie in einem Kästchen des Formulars Zahlen, Befehle und Klartext untergebracht

---

[1]) Wir sprechen von *digitalen* Geräten, wenn darin nur endlich viele fest vorgegebene Informationszustände auftreten. Solche Automaten können insbesondere Ziffern handhaben, so daß man auch von *Ziffern-Rechenautomaten* spricht.

[2]) Wir betrachten also hier grundsätzlich nur Rechenautomaten mit *gespeichertem Programm*.


Leitwerk
Befehlszählregister
1'
Befehlsregister
Steuerregister
Wort-art
Beding.-Zchn.
Operat.-Zeichen
Adressen
s
t
C L U I F
E N R A S
Rechenwerk
L,R,N
A,i
S
A
Schnell-Speicher
0 Null (enthält stets 0)
1 Zwei (enthält stets 2)
2 Vorzeichenprüfung
3 Rechter Akkumulator
4 Linker Akkumulator
5 Ablaufnotierung
6 Operanden-Register
7 ohne
15 festgelegte Funktion
Trommel-Speicher
Puffer-Speicher
Eingabe
Lochstreifen-abtaster
Ausgabe
Fern-Schreibmasch.
Lochstreifen-stanzer

sein. Jeder Speicherzelle ist eine Nummer, ihre *Adresse*, als Name zugeordnet. Den Inhalt einer Speicherzelle nennen wir ganz allgemein ein *Wort*[1]).

Im *Rechenwerk* werden die eigentlichen Rechenoperationen ausgeführt. Die Zahlen werden aus dem Speicher in das Rechenwerk transportiert, dort miteinander verknüpft, und die Ergebnisse werden zurückgespeichert.

Mit Hilfe der *Ein-* und *Ausgabe* können Zahlen und Befehle in den Automaten eingegeben und Ergebnisse sowie Auskünfte über den Stand der Rechnung ausgeliefert werden. Als technische Hilfsmittel der Ein- und Ausgaben dienen zum Beispiel *Lochstreifen*, *Lochkarten*, *Schreibmaschinen* und *Magnetbänder*.

Das *Leitwerk* steuert die Zusammenarbeit der verschiedenen Maschinenteile. Es analysiert die einzelnen Befehle und veranlaßt, daß die entsprechenden Operationen in der richtigen Reihenfolge abgewickelt werden. Der Funktion des Leitwerks entspricht also die Tätigkeit der Bedienungsperson einer Tischrechenmaschine.

Jeder Befehl enthält einen *Operationsteil* und einen *Adressenteil*. Der Operationsteil bestimmt, *was* zu tun ist, also z. B. „addiere", „multipliziere" usw. Der Adressenteil gibt an, in welchen Speicherzellen die Zahlen zu finden sind, mit denen gerechnet werden soll. Der Aufbau eines Befehls entspricht also recht genau den Rechenanweisungen unseres Rechenformulars von Seite 12. So hat zum Beispiel die Operation in Zeile 3 unseres Beispiels, $<1> + <2>$, den Operationsteil + und die Adressen 1 und 2. Es gibt allerdings auch eine Reihe von Befehlen, für die keine entsprechenden Anweisungen auf dem Formular zu finden sind. Das sind z. B. alle Befehle für die Ein- und Ausgabe von Daten sowie die Befehle, die prüfen, ob die Aufgabe gelöst ist, und je nach dem Ergebnis der Prüfung eine Fortsetzung oder den Abschluß des Programms bewirken.

1.4 Eine kurze Beschreibung der Z 22

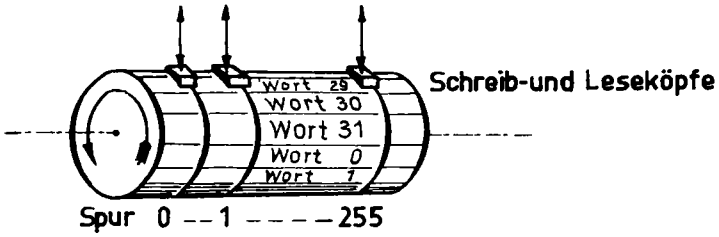
Es soll hier eine kurze Beschreibung der Z 22 [8] gegeben werden, die als kleiner Universalrechner mit gespeichertem Programm wegen ihrer weiten Verbreitung, insbesondere an deutschen Hochschulen und Universitäten, und wegen ihrer Flexibilität besonders gut als Musterrechner für die später folgenden Programmbeispiele geeignet ist. Sie hat außerdem den Vorzug, ein *Dual-Rechner* zu sein, so daß gleichzeitig die Besonderheiten dieser wichtigen Rechnerklasse mit erläutert werden können.

[1]) Bei Rechnern mit zeichenweise adressierbaren Speichern muß ein Wort etwas allgemeiner als eine durch einen Befehl gemeinsam gehandhabte Zeichenfolge definiert werden.

Auf Seite 17 ist ein grobes Blockschaltbild angegeben, aus dem die Benennung der Hauptteile und die Informations-Verbindungswege zu ersehen sind.

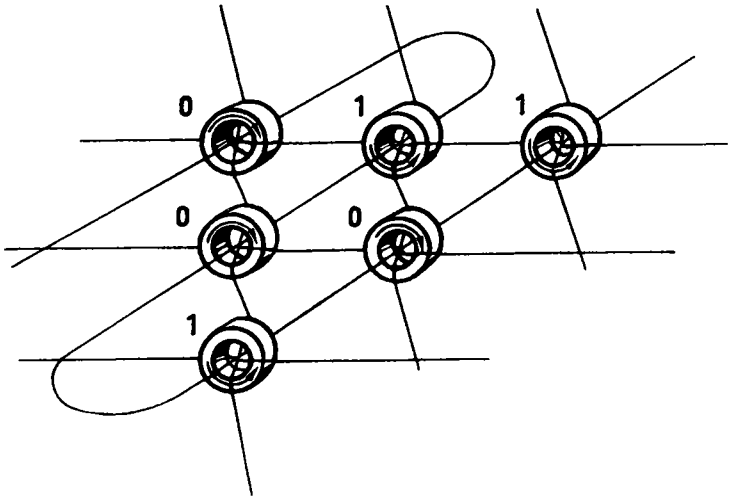
1.41 Die technische Realisierung der Hauptteile der Z 22

Der *Speicher* besteht aus zwei Einheiten, einer Magnettrommel (als Arbeits- oder *Primärspeicher*) und einer Magnetkern-Matrix (als Schnellspeicher).



Die *Magnettrommel* ist ein rotierender Zylinder von etwa 18 cm Durchmesser und 30 cm Länge, auf dessen magnetisierbarer Oberfläche die Informationen in einer magnetischen Impulsnotierung untergebracht sind. Auf jedem Umfang stehen 32 Wörter. Die magnetischen Notierungen längs eines Umfanges werden von einem Magnetkopf geschrieben bzw. gelesen. Den zu einem Kopf gehörigen Trommelbereich nennt man eine *Spur*. In unserem Beispiel sind auf der Trommel 256 Spuren, also insgesamt $32 \cdot 256 = 8192$ Wörter untergebracht. Da die Trommel mit 100 U/sec rotiert, muß man im Mittel 5 ms ($= {}^1/_2$ Umdrehung) warten, bis ein bestimmtes Wort am Kopf der dazugehörigen Spur erscheint und damit zur Verarbeitung erreichbar ist. Die *mittlere Zugriffszeit* beträgt also 5 ms.

Der *Schnellspeicher* besteht aus einer sogenannten *Ferritkern-Matrix*. Das ist eine matrixartige Anordnung von Ringen aus magnetisierbarem Material, bei denen zwei Remanenzzustände, nämlich die beiden gegenläufig ringförmigen Magnetisierungen, stabil sind. Sie werden zur Speicherung von binären Informationen (also etwa der Ziffern 0 und 1) verwendet. Der Kernspeicher enthält 16 Zellen mit sehr geringer Zugriffszeit. Er dient als schneller Zwischenspeicher für oft benutzte Wörter und kann auf diese Weise verhindern, daß die große Zugriffszeit der Trommel den Programmablauf unvernünftig lange aufhält.



Diese 16 Schnellspeicherzellen sind zum Teil durch besondere Inhalte und Funktionen festgelegt. So enthält die Schnellspeicherzelle 0 stets eine 0 und die Zelle 1 eine 2. Die Zelle 2 wird dazu benutzt, um das Vorzeichen einer dorthin transportierten Zahl zu prüfen, während die Zellen 3 und 4 eine besondere Bedeutung für das Rechenwerk besitzen. Die Zelle 5 hat dagegen eine besondere Funktion für das Leitwerk. Zelle 6 ist eng mit der Durchführung der Rechenoperationen in gleitendem Komma verbunden. Die Schnellspeicherzellen 7 bis 15 haben normale Speicherfunktionen.

Über die soeben genannten Schnellspeicherzellen hinaus gibt es eine sogenannte Schnellspeichererweiterung auf 32 Zellen. Aber auch im Adressenbereich 16 bis 31 sind einige Speicher durch Sonderfunktionen festgelegt: Zelle 16 ist mit Zelle 5 identisch, Zelle 17 hat eine besondere Funktion für das Bedienungspult, während die Zellen 19 und 20 mit der Ein- und Ausgabe zu tun haben. Die Zellen 21 bis 31 stehen wieder als normale Schnellspeicher zur Verfügung.

Die Informationsdarstellung ist, wie wir gesehen haben, im Speicher *binär*. Das gilt für die gesamte Rechenmaschine. Informationen, die nicht wie die einzelnen Stellen einer *Dualzahl* ohnehin binär sind, müssen beispielsweise in folgender Form nach einem bestimmten *Code* als Kombination von binären Angaben aufgebaut (*binär verschlüsselt*) werden.

Informationsart		Beispiel für eine binäre Notierung	
Dualzahl	10110		10110
Dezimalziffern:	0	Dualcode (Tetraden):	0000
	1		0001
	2		0010
	3		0011
	4		0100
	5		0101
	6		0110
	7		0111
	8		1000
	9		1001
Buchstaben:	A	Fernschreibcode (Pentaden):	11000
	B		10011
	C		01110
	D		01101
	.		.
	.		.
	.		.

Das *Leitwerk* enthält im wesentlichen die folgenden drei Register: Das *Befehlsregister* (BR), das *Befehlszählregister* (BZ) und das *Steuerregister* (SR). Jedes dieser Register kann einen Befehl aufnehmen. Die Befehle werden aus dem Speicher über den *Schalter* E in das Leitwerk transportiert und im Befehlsregister abgelegt. Zur Ausführung wird der Befehl in das Steuerregister übernommen.

Im Steuerregister werden die verschiedenen zur Ausführung des Befehls benötigten Steuersignale ausgelöst. Insbesondere werden die Schalter E, C, N usw. gestellt und dem Speicher die Adressen der durch den Befehl betroffenen Speicherzellen mitgeteilt. Auf der anderen Seite werden aus dem Rechenwerk Angaben (insbesondere Vorzeichen) über die in den Schnellspeichern 2, 3 und 4 stehenden Zahlen nach SR übertragen. Diese Angaben bedingen die Ausführung des Befehls in einer Weise, die im *Befehlscode* näher beschrieben ist.

Mit Hilfe eines besonderen Befehlswortes, das ständig zwischen dem Befehlsregister und dem Befehlszählregister kreist, wird bewirkt, daß die Be-

fehle in der Reihenfolge, in der sie im Speicher stehen, ins Leitwerk abgerufen werden. Dazu ist ein ständiges Fortschalten der Adresse notwendig. Dies wird von dem zwischen BR und BZ liegenden Addierwerk (+) übernommen (Steuerung der *Befehlsfolge*).

Eine besondere Leitung führt über den Schalter F von BZ in den Schnellspeicher 5, den Speicher für die sogenannte *Ablaufnotierung*. Wie wir später sehen werden, dient dies dazu, bei Programmverzweigungen eine einfache Rückkehr zur Verzweigungsstelle zu ermöglichen.

Bis auf die Übernahme der Befehle von BR nach SR erfolgen alle Transporte in *Serie*; das heißt, die binären Elemente eines Wortes werden zeitlich nacheinander auf *einem* Kanal übertragen. Im Gegensatz dazu werden beim (schnelleren) *Paralleltransport* diese Elemente gleichzeitig auf mehreren Kanälen übertragen.

Das *Rechenwerk* besteht aus dem *Addierwerk* und zwei Registern, die mit den Schnellspeicherzellen 3 und 4 identisch sind. Dem rechten Eingang des Addierwerks (A, I) können über den Schalter A Wörter aus dem Speicher und über den Schalter C Wörter aus dem Befehlsregister zugeführt werden. Dieser Zugang kann über das *Komplementwerk* S negiert werden, so daß im Addierwerk bei Bedarf an Stelle der Addition eine Subtraktion ausgeführt wird. Der linke Zugang des Addierwerks ist im allgemeinen mit dem *Akkumulator* (Schnellspeicher Nr. 4) verbunden. Beim Transport vom Akkumulator zum Addierwerk kann jedes Wort nach links bzw. rechts verschoben oder gelöscht werden (L, R, N). Der Summenausgang des Addierwerks führt immer zum Akkumulator. Ein weiterer Weg führt vom Akkumulator über den Schalter U zum Speicher. Die besonderen Funktionen der Schnellspeicher 2, 3, 5 und 6 werden weiter unten bei der Erläuterung des Befehlscodes deutlich.

Zur *Eingabe* dient ein mechanischer oder photoelektrischer Lochstreifenabtaster, der seine Informationen über einen kleinen Pufferspeicher und von da aus über das Addierwerk in den Akkumulator abliefert.

Die *Ausgabe* erfolgt aus dem Akkumulator über den Schalter U und den Pufferspeicher auf eine Fernschreibmaschine, an die bei Bedarf ein Lochstreifenstanzer angeschlossen werden kann.

1.42 Die Zahlendarstellung im Innern der Z 22

Innerhalb elektronischer Digitalrechner werden vor allem das Dualsystem (Basis $B = 2$) und das Dezimalsystem ($B = 10$), bei dem jedoch die ein-

zelnen Dezimalziffern binär verschlüsselt sind, verwendet. In der Z 22 wird das Dualsystem benutzt[1]).

Die einfachste Zahlendarstellung ergibt sich, wenn jeder Ziffer ein fester Stellenwert zugeordnet wird. Man nennt diese Anordnung eine *Festkomma*-Darstellung (*FK*). Für die Z 22 gilt in diesem Fall:

Stelle	1	2	3	...	36	37	38
Stellenwert	2^{37}	2^{36}	2^{35}	...	2^{2}	2^{1}	2^{0}

Für die Größe der dargestellten Zahlen z gilt die Beschränkung:

$$|z| < 2^{35}$$

Dabei sind mit Rücksicht auf die weiter unten besprochene Darstellung negativer Zahlen die Stellen 1 und 2 nicht benutzt worden.

Da in den verschiedenen von der Maschine zu lösenden mathematischen Problemen Zahlen sehr unterschiedlicher Größenordnung vorkommen, fällt dem programmierenden Mathematiker die unangenehme Aufgabe zu, alle Größen so zu normieren, daß sie während des gesamten Rechenganges in der gleichen vorgeschriebenen Größenordnung bleiben. Diese Aufgabe ist nicht nur mühsam und durch Irrtümer gefährdet, sondern setzt zu ihrer vernünftigen Bewältigung auch im voraus gewisse Kenntnisse über das Verhalten von Größen voraus, die erst berechnet werden sollen. Es müssen also erst geeignete Abschätzungen gemacht werden, die nicht immer einfach durchzuführen sind.

Um den Programmierer dieser Normierungsarbeit zu entheben, führt man die Zahlendarstellung in *gleitendem Komma* (*GK*) ein. Man benutzt im allgemeinen eine sogenannte *halblogarithmische* Notierung; das heißt: Jede Zahl wird in der Maschine als Zahlenpaar $(\hat{z}, \bar{z})$ dargestellt. $\hat{z}$ und $\bar{z}$ haben nun festes Komma und werden *Mantisse* und *Skalenfaktor* (oder *Exponent*) genannt. Mit Hilfe von $\hat{z}$ und $\bar{z}$ können die Zahlen als $z = \hat{z}\, B^{\bar{z}}$ oder in einer ähnlichen Form dargestellt werden. In der Z 22 gilt (mit $B = 2$)

$$z = \hat{z} \cdot 2^{\tilde{z}-64} = \hat{z} \cdot 2^{\bar{z}}.$$

Dabei nennen wir den um 64 verschobenen Skalenfaktor die *Charakteristik* $\tilde{z}$. Durch diese Verschiebung können wir von Skalenfaktoren beiderlei Vor-

[1]) Die Elemente des Rechnens mit Dualzahlen (Addition und Subtraktion) werden hier als bekannt vorausgesetzt. Die Benutzung des Dualsystems in programmgesteuerten Rechnern ist ebenso wie die halblogarithmische Zahlendarstellung zuerst bei *Zuse* zu finden [8], [9].

zeichens auf eine nicht negative Charakteristik übergehen, wenn wir für die Mantisse 31 und für die Charakteristik 7 Stellen nach folgendem Schema vorsehen:

Stelle	1	2	3	4 31	32	33	34	...38
Stellenwert	2^1	2^0	2^{-1}	2^{-2} ... 2^{-29}	2^6	2^5	2^4	... 2^0
	Mantisse $\hat{z}$				Charakteristik $\tilde{z}$			

Beschränken wir die Charakteristik auf den Bereich

$$0 \leqq \tilde{z} < 2^7,$$

so können wir Skalenfaktoren innerhalb der Grenzen

$$-64 \leqq \bar{z} < 64$$

zulassen. In der Z 22 werden die Gleitkommazahlen daher nicht durch $(\hat{z}, \bar{z})$, sondern durch $(\hat{z}, \tilde{z})$ dargestellt.

Beim Rechnen in gleitendem Komma ergeben sich für die einzelnen Rechenoperationen folgende Besonderheiten:

Sollen zwei halblogarithmisch dargestellte Zahlen *addiert* (oder *subtrahiert*) werden, so müssen zuvor die Skalenfaktoren angeglichen werden:

$$z_1 = \hat{z}_1 \cdot B^{\bar{z}_1}$$
$$z_2 = \hat{z}_2 \cdot B^{\bar{z}_1}.$$

Angenommen, es ist $\bar{z}_1 \geqq \bar{z}_2$, dann kann z_2 auch in der Form

$$z_2 = \hat{z}_2' \cdot B^{\bar{z}_1} \text{ mit } \hat{z}_2' = B^{-(\bar{z}_1 - \bar{z}_2)}\, \hat{z}_2$$

geschrieben werden, und es wird

$$z_1 + z_2 = (\hat{z}_1 + \hat{z}_2')\, B^{\bar{z}_1}.$$

Bei der *Multiplikation* und *Division* können die Mantissen direkt verarbeitet werden. Das Resultat erhält dann die Summe bzw. die Differenz der Skalenfaktoren als Skalenfaktor zugeordnet.

$$z_1 \cdot z_2 = (\hat{z}_1 \cdot \hat{z}_2)\, B^{\bar{z}_1 + \bar{z}_2}$$
$$z_1 : z_2 = (\hat{z}_1 : \hat{z}_2)\, B^{\bar{z}_1 - \bar{z}_2}.$$

Lassen wir eine Reihe von Operationen in gleitendem Komma ablaufen, so werden bei der Addition und Subtraktion durch die Exponentenangleichung und durch die Multiplikation von Zahlen, die kleiner als Eins sind,

die von Null verschiedenen Ziffern nach rechts aus dem Zahlenbereich der Mantisse hinauswandern. Es ist daher gebräuchlich, nach jeder Operation die Zahlen in bestimmter Weise zu *normieren*.

Eine einfache Normierungsmethode besteht darin, daß grundsätzlich nach jeder Operation die Mantisse des Ergebnisses ganz nach links geschoben und der Skalenfaktor entsprechend korrigiert wird. Dieses Verfahren wird im folgenden bei der Z 22 verwendet. Wir wollen also verabreden

$$1/2 \leqq |\hat{z}| < 1.$$

Diese Normierung läßt die Tatsache außer acht, daß unter Umständen laufend willkürliche Ziffern nach links gezogen werden, die keinen rechnerischen Wert besitzen. Man darf sich also über die Anzahl der gültigen Stellen eines Ergebnisses nicht täuschen lassen.

Es sind daher auch andere Normierungsverfahren üblich, die das Nachziehen ungültiger Stellen mehr oder weniger vermeiden [10].

Es bleibt noch die Frage zu erörtern, wie *negative Zahlen* dargestellt werden sollen.

Das einfachste ist, die auch sonst gebräuchliche Darstellung durch Betrag und Vorzeichen zu benutzen. Dies bringt aber gewisse Schwierigkeiten beim Addiervorgang mit sich, so daß in vielen Rechenmaschinen sogenannte Komplementdarstellungen benutzt werden. Neben den hier nicht näher erläuterten sogenannten (B-1)-Komplementen interessiert vor allem das *B-Komplement*.

Wir wollen annehmen, daß unsere Zahlen z folgendermaßen begrenzt sind:

$$-B^n \leqq z < B^n.$$

Zur Darstellung positiver z verwenden wir als höchste Stelle B^{n-1}. Nehmen wir noch die Stelle B^n dazu, so können wir (wegen $+0 = -0$) alle Zahlen des oben angegebenen Bereiches darstellen. Am einfachsten geschieht das durch Einführung von sogenannten B-Komplementen:

$$-|z| \sim (B^{n+1} - |z|) \bmod B^{n+1}.$$

Beispiel:

$B = 10$, $n = 0$:

$z = 0{,}3468$

$B^{n+1} - |z| = 10 - |0{,}3468| = 9{,}6532$ also

$-|z| \sim 9{,}6532 \bmod 10 \sim 9{,}6532$

oder

$B = 2, n = 0$:
$z = 0{,}010111$
$B^{n+1} - |z| = 2 - |0{,}010111| = 1{,}101001$
$-|z| \sim 1{,}101001 \bmod 2 \sim 1{,}101001$

Wegen der oben vorausgesetzten Bereichsbeschränkung für z liegt $(B^{n+1} - |z|)$ zwischen B^{n+1} $(= 0 \bmod B^{n+1})$ und $B^{n+1} - B^n$. Das heißt, negative von Null verschiedene Zahlen kann man bei der Darstellung mit B-Komplementen daran erkennen, daß sie in der ersten Stelle (B^n) stets die Ziffer B-1 haben.

Addieren wir grundsätzlich mod B^{n+1}, so werden alle Summen, wie man im folgenden sieht, vorzeichenrichtig.

1. Fall $z_1 < 0, z_2 < 0, |z_1 + z_2| \leqq B^n$.
Es ist mod B^{n+1}
$(B^{n+1} - |z_1|) + (B^{n+1} - |z_2|) \sim$
$B^{n+1} - (|z_1| + |z_2|) \sim B^{n+1} - |z_1 + z_2| \sim z_1 + z_2$

Beispiel: $(B = 10, n = 0)$

z_1:	$-0{,}3468 \sim$	$9{,}6532$
z_2:	$-0{,}1411 \sim$	$9{,}8589$
	$-0{,}4879 \sim$	$1/9{,}5121$

2. Fall $z_1 \geqq 0, z_2 < 0$
a) $z_1 + z_2 \geqq 0$
Es ist dann mod B^{n+1}
$z_1 + B^{n+1} - |z_2| \sim z_1 + z_2$

Beispiel:

z_1:	$0{,}5678 \sim$	$0{,}5678$
z_2:	$-0{,}2310 \sim$	$9{,}7690$
	$0{,}3368 \sim$	$1/0{,}3368$

b) $z_1 + z_2 < 0$
Es ist dann mod B^{n+1}:
$z_1 + B^{n+1} - |z_2| \sim B^{n+1} - (-z_1 + |z_2|) \sim$
$B^{n+1} - |-z_1 + |z_2|| \sim B^{n+1} - |z_1 + z_2| \sim z_1 + z_2$

Beispiel:

z_1: $0{,}5678 \sim 0{,}5678$

z_2: $-0{,}7223 \sim 9{,}2777$

$-0{,}1545 \sim 9{,}8455$

B-Komplemente werden am einfachsten dadurch hergestellt, daß man zuerst das sogenannte (B-1)-Komplement bildet:

$$B^{n+1} - B^m - |z|.$$

Dabei ist B^m der Stellenwert der niedrigsten Stelle von z. Das (B-1)-Komplement entsteht dadurch, daß bei allen Ziffern einzeln das Komplement zu (B-1) gebildet wird. Das ist besonders einfach bei Dualzahlen. Dort werden alle Nullen in Einsen und alle Einsen in Nullen verwandelt. Anschließend muß zum (B-1)-Komplement noch B^m addiert werden, um das B-Komplement zu erhalten.

Beispiel:

$B = 10,\ n = 0,\ m = -6$:

$z = -0{,}734590$

$10^1 - 10^{-6} - |z| = 9{,}265409$

$10^1 - |z| = 9{,}265410$ oder

$B = 2,\ n = 0,\ m = -6$:

$z = -0{,}101101$

$2^1 - 2^{-6} - |z| = 1{,}010010$

$2^1 - |z| = 1{,}010011$

In der Z 22 wird für negative Zahlen in *festem Komma* das B-Komplement mit

$$|z| < 2^{35}$$

und für negative Zahlen in *gleitendem Komma* das B-Komplement mit

$$0{,}5 \leqq |z| < 1$$

benutzt. Dabei wird jedoch für das Vorzeichen nicht nur die nächste Stelle (2^{35} bei *FK* und 2^0 bei *GK*), sondern alle links noch vorhandenen Stellen (2^{35}, 2^{36} und 2^{37} bei *FK*, 2^0 und 2^1 bei *GK*) verwendet. Dies hat den Vorteil, daß eine *Bereichsüberschreitung* (in einer Stelle bei *GK* und in zwei Stellen bei *FK*), wie sie bei manchen arithmetischen Operationen auftritt, einfach (also ohne *Akkumulatorenüberlauf*) aufgefangen werden kann. Außerdem können wir damit Festkommazahlen leicht von Gleitkommazahlen unterscheiden; denn alle positiven Festkommazahlen haben vorn mindestens drei Nullen, während positive Gleitkommazahlen vorn genau zwei Nullen besitzen. Nur Gleitkommanullen bestehen genau wie Festkommanullen aus lauter Nullen. Bei allen von Null verschiedenen positiven Zahlen können wir also an ihrer dritten Stelle erkennen, ob sie in festem oder gleitendem Komma dargestellt sind.

1.43 Der Befehlscode der Z 22

Der Befehlscode eines Rechners ist eine Liste aller im Rechner vorgesehenen Befehle und ihrer Wirkungen. Man unterscheidet dabei verschiedene Typen von Befehls-Strukturen, je nach der Anzahl der Adressen, die im Befehl enthalten sind, und nach dem Aufbau des Operationsteils und sonstiger Bestandteile des Befehls.

Hinsichtlich der *Adresse* spricht man häufig von *(m + n)-Adreß-Befehlen.* Dabei gibt *m* die Anzahl der im Befehl zur Bestimmung der *Operanden* vorgesehenen Adressen an, während *n* bestimmt, wie viele Adressen zur Bestimmung des *Folgebefehls* verwendet werden können (*Operanden-* und *Folgeadressen*).

Wir wollen folgende Typen kurz erläutern:

(3 + 1)-Adreß-Befehle. Am einfachsten zu verstehen sind (3 + 1)-Adreß-Befehle. Da die meisten Befehle zur Verknüpfung zweier Operanden (z. B. $a + b$) dienen, kommt man zur Handhabung der beiden Eingangsoperanden und des Ergebnisses mit drei Adressen bequem aus: Zwei Adressen bestimmen, *aus* welchen Speicherzellen die beiden Eingangsoperanden zu entnehmen sind, und die dritte Operandenadresse kennzeichnet die Speicherzelle, *in* der das Ergebnis abgelegt werden soll.

Zur Bestimmung des Folgebefehls dient jeweils *eine* Adresse.

Beispiel:

In Zelle	steht der Befehl				
	Operation	1. Operanden-Adresse	2. Operanden-Adresse	Ergebnis-Adresse	Folge-Adresse
1013	Addition	2000	2001	2002	1027
.					
.					
.					
1027	Multiplikation	2002	2003	2004	1028
.					
.					

Dieses Programmstück bewirkt, daß $<2000>$ und $<2001>$ addiert werden und das Ergebnis nach 2002 transportiert wird. Als nächster Befehl wird $<1027>$ genommen, der $<2002>$ mit $<2003>$ multipliziert und das Ergebnis in 2004 ablegt.

(1 + 0)-Adreß-Befehle. Hier ist erst einmal zu bemerken, daß für die Bestimmung des Folgebefehls *keine* Adresse bereitsteht. Die Befehle werden vielmehr im allgemeinen in der Reihenfolge ausgeführt, in der sie im Speicher stehen. Dies ist eine vernünftige Maßnahme, weil es das einfachste ist, die Befehle in der Reihenfolge zu speichern, in der sie normalerweise bearbeitet werden müssen. Um nun aber nicht auf solche linearen Abläufe festgelegt zu sein, muß ein Befehlstyp geschaffen werden, mit dessen Hilfe die linearen Befehlsfolgen unterbrochen werden können. Das sind die sogenannten *Sprungbefehle.* Bei Sprungbefehlen wird eine (oder auch mehrere) der Operandenadressen ausnahmsweise zur Bestimmung des Folgebefehls benutzt.

Will man sich auf *eine* Operandenadresse beschränken, so müssen bestimmte Verabredungen über die Behandlung der übrigen Operanden getroffen werden. Normalerweise gilt folgendes: Sollen zwei Operanden durch eine Operation (z. B. Addition) untereinander verknüpft werden, so bestimmt die Operandenadresse einen der beiden Eingangsoperanden (z. B. einen Summanden). Als Aufbewahrungsort für den zweiten Eingangsoperanden wird dagegen ein bestimmtes Rechenwerkregister, der sogenannte *Akkumulator,* vereinbart. Das Ergebnis wird wieder in den Akkumulator transportiert.

Beispiel:

In Zelle	steht der Befehl	
	Operation	Operandenadresse
1013	Bringe Operanden in den Akkumulator	2000
1014	Addition	2001
1015	Speichere Akkumulatorinhalt nach	2002
1016	Sprung nach	1027
.		
.		
1027	Bringe Operanden in den Akkumulator	2002
1028	Multiplikation	2003
1029	Speichere Akkumulatorinhalt nach	2004
.		
.		
.		

Das Programmstück bewirkt das gleiche wie das vorige Beispiel, in dem nur zwei (3 + 1)-Adreß-Befehle vorkommen. Beim (1 + 0)-Adreß-Code wird der Rechner einfacher, aber die Programme werden etwas länger.

(2 + 0)-Adreß-Befehle. Dieser sogenannte *Quellen-Senken-Code* besitzt zwei Operandenadressen. Normalerweise werden sie zur Bestimmung der beiden Eingangsoperanden benutzt. Das Ergebnis wird anschließend anstelle eines der beiden Eingangsoperanden zurückgespeichert. Auf diese Weise kann ein Akkumulator entfallen. Dieser Code empfiehlt sich vor allem beim Rechnen mit *variabler Wortlänge* (vgl. Kap. 9.1).

Die Z 22 besitzt wie die meisten Rechenautomaten einen (1 + 0)-Adreß-Code. Der Umstand, daß zwei Adressen s und t vorhanden sind, ändert nichts daran, daß hier $m = 1$ richtig ist, weil im allgemeinen die durch s und t bezeichneten Operanden nicht miteinander verknüpft werden; s und t werden vielmehr wahlweise zur Bestimmung *eines* Operanden verwendet[1]).

[1]) Eine Variante der Z 22, die Z 22 R, besitzt allerdings die Eigenschaft, daß alle Schnellspeicher als Akkumulator benutzt werden können, so daß s hier neben t einen gesonderten Operanden bezeichnet. Die Z 22 R hat also einen (2 + 0)-Adreß-Code.

Hinsichtlich des *Operationsteiles* wollen wir lediglich auf eine Besonderheit der Z 22 und einiger verwandter Maschinen gegenüber den übrigen Rechnern aufmerksam machen: Im allgemeinen wird der Operationsteil des Befehls *insgesamt verschlüsselt*. So kann man z. B. durch eine zweistellige Dezimalzahl hundert verschiedene Operationen darstellen. Dann kann allerdings den beiden Dezimalziffern, aus denen der Operationsteil aufgebaut ist, einzeln keine besondere Bedeutung mehr zugeordnet werden. Zur Systematisierung und Vereinfachung des Befehlscodes bzw. des Leitwerks sind nun die sogenannten *analytischen Befehlscodes* eingeführt worden. Bei diesen Codes werden alle Operationen nach Möglichkeit so in Teiloperationen aufgelöst, daß die Operation durch eine *Kombination* solcher *Teiloperationen* entsteht. So kann z. B. eine Subtraktion aufgelöst werden in

1. den Transport eines Wortes aus dem Speicher in den Akkumulator,
2. die Einschaltung des Addierwerks mit Einspeisung des bisherigen Akkumulatorinhalts in diesen Transportvorgang und
3. die Vorschaltung eines Komplementwerkes, um das Vorzeichen dem aus dem Speicher in den Akkumulator transportierten Zahl vor dem Addierwerk umzukehren.

Die so erhaltenen Teiloperationen werden nun nach Möglichkeit so in Gruppen geordnet, daß jede Gruppe nur Teiloperationen enthält, die nicht miteinander kombiniert werden können, während Kombinationen mit den Elementen anderer Gruppen erlaubt sind. Jeder Befehl besteht dann aus Kombinationen von Teiloperationen, die alle aus verschiedenen Gruppen stammen [11]. Bei vollständiger Auflösung der Befehle in ihre Elemente kommen wir zu dem von *van der Poel* geschaffenen *Einzelbit-Code*, bei dem jedes Bit[1]) des Operationsteils im Befehlswort eine eigene, von den übrigen unabhängige Funktionen hat [12], [13].

Die Z 22 besitzt im wesentlichen einen solchen Einzelbit-Code. Die einzelnen Binärstellen stellen also kombinierbare Elementarbefehle dar, welche jeweils wichtige Transporte und grundlegende logische Operationen bewirken. Allerdings ist die Handhabung dieses Befehlscodes umständlich. Daher wird grundsätzlich eine maschinenorientierte Sprache und nicht die Maschinensprache benutzt.

---

[1]) Eine Binärstelle nennt man auch ein *Bit*. Das Bit (Plural: Bits) darf nicht mit der *Shannonschen* Informationseinheit *bit* verwechselt werden.

Wir wollen im folgenden den Befehlscode einer solchen maschinenorientierten Sprache einführen und schrittweise vervollständigen. Die eigentliche Maschinensprache der Z 22 werden wir in Kapitel 9.2 kennenlernen.

2 Über die Notierung von Programmen

2.1 Einige oft benutzte Abkürzungen

Hier sollen einige uns zum Teil schon aus der Einleitung bekannte Zeichen zusammengestellt werden, die bei der Notierung von Maschinen-Programmen, insbesondere bei den Programmerläuterungen, und in den Beschriftungen der Flußdiagramme nützlich sind. Die in der algorithmischen Sprache verwendeten Symbole werden jeweils im Laufe der Darstellung eingeführt und später in einem gesonderten Kapitel (11) noch einmal vollständig zusammengestellt.

$<n>$ *Inhalt* des durch n gekennzeichneten Maschinenortes.

Beispiel: $<4096> = \pi$

Inhalt der Speicherzelle mit der Adresse 4096 ist gleich π.

$\{x\}$ Der *Maschinenort*, an dem das Wort x aufbewahrt wird.

Beispiel: $\{\pi\} = 4096$

Die Zahl π steht in der Speicherzelle 4096.

$x \to n$ *Transport* des Wortes x an den Maschinenort n; dabei bleibt, wenn nicht ausdrücklich etwas anderes vorgeschrieben ist, das Wort x an der ursprünglichen Stelle erhalten.

Beispiel: $\pi \to 4$

Die Zahl π wird (etwa aus der Speicherzelle 4096) nach Schnellspeicher 4 transportiert. Sie ist nach wie vor in 4096 verfügbar.

Rechts von $\to$ dürfen auch mehrere durch Komma getrennte Größen stehen. Das bedeutet dann, daß das Wort in alle rechts angegebenen Örter transportiert wird.

$\leftarrow$ Das *Ergibtzeichen*. Sein Gebrauch wird insbesondere in der algorithmischen Sprache genau geregelt. Es wird aber auch außerhalb der eigentlichen algorithmischen Notierung verwendet und hat dort eine entsprechende Bedeutung: Der Wert des (bei der hier gewählten Orientierung) rechts vom

Ergibtzeichen stehenden Ausdrucks wird ermittelt und der links stehenden Größe als Wert zugeordnet. Diese Größe darf dabei auch im rechts stehenden Ausdruck vorkommen. Sie muß dann dort mit dem Wert verwendet werden, den sie vor der durch das Ergibtzeichen bewirkten neuen Wertzuordnung gehabt hat.

Beispiele: $c \leftarrow a + 3$

Angenommen, der Wert von a sei durch eine vorangegangene Wertzuordnung 5, dann wird der Wert des Ausdruckes $5 + 3 = 8$. Dieser Wert 8 wird der Variablen c zugeordnet. Es gilt also fortan $c = 8$.

$$i \leftarrow i + 1$$

Der bekannte Wert von i (z. B. 3) plus 1 ergibt 4. Dieser Wert 4 wird der Variablen i zugeordnet. Es gilt also fortan $i = 4$.

2.2 Der Befehlscode

Will man für einen Rechenautomaten Programme herstellen, so muß man einen bestimmten Befehlscode zugrunde legen. Wir wollen hier als *Beispiel* eine einfache maschinenorientierte Sprache für die Z 22 angeben.

Für die Z 22-Befehle gilt allgemein folgendes:

Alle Befehle werden im Trommelspeicher aufbewahrt und bis zum Auftreten eines Sprungbefehls E in der dort stehenden Reihenfolge ausgeführt.

Jeder Befehl besitzt einen Operationsteil. Zu diesem Operationsteil kann eine Adresse n hinzukommen, nämlich:

entweder eine *Schnellspeicheradresse* s (siehe Blockschaltbild auf Seite 17)

oder eine *Trommelspeicheradresse* $t = 1000 \ldots 8000$.

Wird innerhalb der Maschine mit Adressen gerechnet, so haben diese folgenden Stellenwert:

$$s'' = s \cdot 2^{13} \text{ und } t' = t$$

Die *Striche* ($''$ und $'$) sollen dabei nicht nur den Stellenwert kennzeichnen, sondern auch festlegen, daß diese Zahlen in *festem Komma* dargestellt sind.

Im einzelnen unterscheiden wir folgende Befehlstypen:

1. *Rufbefehle*

Rufbefehle haben *keine Adresse.*

Op.	Wirkung	Bemerkungen
+	$<6> + <4> \to 4,6$	Ausführung in gleitendem Komma
—	$<6> - <4> \to 4,6$	
X	$<6> \times <4> \to 4,6$	
:	$<6> : <4> \to 4,6$	
W	$\sqrt{<4>} \to 4,6$	
M	$- <4> \to 4,6$	
D	$<4>$ *ausdrucken*	Ausführung je nach Darstellung des betreffenden Wortes in *FK* oder *GK*
J	nächstes Wort vom Lochstreifen *lesen* und nach 4 transportieren	

Zur Ausführung der Rufbefehle werden die Schnellspeicher-Zellen 3...10 benötigt.

2. *Grundbefehle*

Grundbefehle haben stets *eine* Adresse.

Bef.	Wirkung	Bemerkungen
An	$<4> + <n> \to 4$	*Addition* in festem Komma
Un	$<4> \to n$	*Speichern ohne* Löschen von $<4>$
Et	$<t> \to BR$	*Sprung* auf Befehl $<t>$, anschließend Ausführung der Befehle in t, $t+1$, $t+2$ ...
In	$<4> \wedge <n> \to 4$	*Intersektion* (stellenweises Produkt von $<4>$ und $<n>$)

Zur Ausführung eines Sprungbefehls wird der Zielbefehl des Sprunges in das Befehlsregister transportiert[1]). Eine genauere Beschreibung der Vorgänge im Befehlswerk wird in Kapitel 9.2 nachgeholt, dort wird auch die Bedeutung der Namen *Rufbefehl* und *Grundbefehl* klar werden.

Die Intersektion zwischen zwei Zahlen $c = a \wedge b$ ist das stellenweise Produkt von a und b. Das heißt, in $c = a \wedge b$ erscheint eine 1 an den Stellen wo sowohl a als auch b eine 1 besitzen, sonst eine 0.

[1]) Es darf statt t auch eine Schnellspeicheradresse stehen. Im allgemeinen befindet sich dann in dem aufgerufenen Schnellspeicher wieder ein Sprungbefehl, der auf eine Trommeladresse weiterführt.

Beispiel:

$$\begin{array}{rl} & a = 1\,0\,1\,1\,0\,0\,1\,1\,1 \\ & b = 0\,0\,0\,1\,1\,1\,1\,1\,1 \\ \hline c = a \wedge & b = 0\,0\,0\,1\,0\,0\,1\,1\,1 \end{array}$$

Sie wird zum Ausblenden bestimmter Teile eines Wortes (etwa a) durch ein in einer *Maske* (etwa b) enthaltenes *Bit-Muster* benötigt. $c = a \wedge b$ enthält dann in den 1-Stellen von b ein genaues Abbild von a, während an allen übrigen Stellen Nullen stehen. Ein häufig auftretendes Beispiel ist das Ausblenden der Charakteristik aus einem Gleitkommawort.

3. *Zusatzzeichen*

Zusatzzeichen treten in den folgenden Kombinationen mit den Operationszeichen der Grundbefehle auf:

	Bef.	Wirkung	Bemerkungen
N	*NEt*	$t \to BR,\ 0 \to 4$	Sprung mit Löschen von $<4>$
	NAn, Bn[1])	$<n> \to 4$	*Bringen*
	NUn, Tn	$<4> \to n,\ 0 \to 4$	Speichern *mit* Löschen von $<4>$
S	*ASn, Sn*	$<4> - <n> \to 4$	*Subtraktion* in festem Komma
	NASn, NSn	$- <n> \to 4$	Bei neg. Zahlen Kap. 1.42
	ISn	$<4> \wedge (-<n>) \to 4$	beachten!
C	*CAt*	$<4> + t \to 4$	Befehle mit *C* als Zusatzzeichen
	CBt	$t \to 4$	benutzen den *Adressenteil als Operanden*
	CSt	$<4> - t \to 4$	In diesem Falle liegt t in dem Be-
	CNSt	$- t \to 4$	reich
	CIt	$<4> \wedge t \to 4$	$0 \leqq t < 2^{13}$
	CISt	$<4> \wedge (-t) \to 4$	
R	*ARn*	$2^{-1} <4> + <n> \to 4$	Multiplikation mit 2^{-1} in *FK*[2])
	SRn	$2^{-1} <4> - <n> \to 4$	(*Rechtsverschiebung*)
L	*ALn*	$2 <4> + <n> \to 4$	Multiplikation mit 2 in *FK*
	SLn	$2 <4> - <n> \to 4$	(*Linksverschiebung*)

Die folgenden *Bedingungszeichen* müssen als spezielle Zusatzzeichen *vor* den anderen Operationszeichen eines Befehls stehen.

[1]) Soll heißen: Anstelle von *NA* kann auch *B* geschrieben werden; Entsprechendes gilt für *T*, *S* usw.
[2]) Beim Rechtsschieben gehen die rechts über die Wortgrenze geratenden Stellen verloren.

Px	Die Befehle x	$<2>$	$\geqq 0$	erfüllt sind. Sonst wird zum folgenden Befehl übergegangen.
Qx	werden nur	$<2>$	< 0	
PPx	ausgeführt,	$<4>$	$\geqq 0$	
QQx	wenn die Bedingungen	$<4>$	< 0	
$PPQQx$		$<4>$	$= 0$	

4. *Stop*

Das Zeichen Z bewirkt Maschinen-Stop. Es wird mindestens einmal am Ende des Programmes geschrieben.

2.3 Befehlsfolgen

Jedes Programm wird, nachdem der Befehlscode festgelegt ist, als Befehlsfolge in ein besonderes Programmformular eingetragen. In diesem Programmformular wird der Programmablauf mit Hilfe der im folgenden Muster angegebenen Zeichen angedeutet.

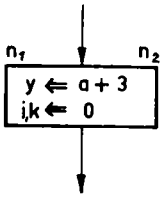
Laufende Nummer	Befehl	Erläuterung
0	Befehl 0	
1	[Befehl 1]	Befehl 1 wird während des Programmablaufes unter Umständen verändert.
2	Befehl 2	
$5 \rightarrow 3$	Befehl 3	Es wird von Befehl 5 unter bestimmten Voraussetzungen auf Befehl 3 gesprungen.
4	Befehl 4	
5	Befehl 5	Befehl 5 ist ein bedingter Sprung.
6	Befehl 6	
7	Befehl 7	Befehl 7 ist ein unbedingter Sprung.
||8	Wort 8||	Diese Position (8) wird während des Programmablaufes nicht erreicht. Sie dient nur als Aufbewahrungsort für das Wort Nr. 8.

2.4 Das Flußdiagramm

Die von *H. H. Goldstine* und *J. von Neumann* [14][1]) eingeführten Flußdiagramme dienen dazu, die Strukturmerkmale eines Programmablaufs graphisch darzustellen. Zu diesem Zweck wird jeder Befehlsfolge eine Linie in der Zeichenebene zugeordnet. In diesen Linien werden „Kästchen" eingefügt, die bestimmten Programmelementen entsprechen.

Wir wollen für den Aufbau von Flußdiagrammen folgende Elemente benutzen:

Allgemeine Anweisung

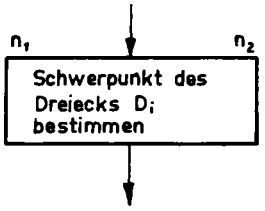


Für allgemeine Anweisungen (insbesondere aber für die weiter unten noch näher zu besprechenden *Ergibt-Anweisungen*) sind rechteckige Kästen mit *einem* Eingangspfeil und *einer* Ausgangslinie, die beliebig am Kasten enden dürfen, vorgesehen. Jeder Kasten enthält eine oder mehrere Anweisungen. Enthält der Kasten mehrere Anweisungen, so sind sie deutlich (z. B. zeilenweise) zu trennen. Sie werden dann nacheinander von oben nach unten ausgeführt (also hier erst $y \Leftarrow a + 3$ und dann $i, k \Leftarrow 0$). Diese Verabredung über die Reihenfolge spielt z. B. eine Rolle, wenn die gleichen Variablen in mehreren Anweisungen eines Kastens und dabei mindestens einmal auf der linken Seite einer Ergibt-Anweisung vorkommen. Die Zahlen n_1 und n_2 am oberen Rand des Kastens bedeuten, daß dem in den Speicherzellen $n_1 \ldots n_2$ stehenden Programmteil die Ausführung dieser Anweisungen obliegt. Entsprechendes gilt auch für die übrigen Flußdiagramm-Symbole. Der für Anweisungen vorgesehene Kasten wird auch benutzt, wenn in sehr groben Flußdiagrammen die einzelnen Verfahrensschritte nur verbal beschrieben werden.

---

[1]) Die klassischen Abhandlungne von *Burks, Goldstine* und *von Neumann* aus den Jahren 1946—48 [14], [15] stellen heute noch ein interessantes Kompendium für jeden dar, der sich mit dem Entwurf und der Programmierung digitaler Rechenautomaten befaßt. Insbesondere ist der das Programmieren betreffende Teil [14] bis heute die genaueste Darstellung der Grundtatsachen der Programmierungstechnik.

Beispiel:



Verzweigung. Eine Verzweigung ist ein Punkt des Flußdiagramms mit *einem* Eingang und *mehreren* Ausgangszweigen, von denen jeweils *einer* durchlaufen wird.

Zu einer Verzweigung gehören stets folgende Elemente:

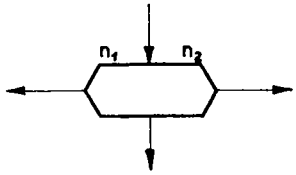
a) *Prüfung*, ob bestimmte Bedingungen, von denen die Auswahl *eines* von mehreren möglichen Zweigen abhängt, erfüllt sind. Diese Prüfung ist unter Umständen mit Rechenoperationen verbunden (z. B. Bedingung $\frac{a-b}{c} > 0$).

b) *Stellen* der Verzweigung in Abhängigkeit von der oben genannten Prüfung.

c) *Durchlauf* durch den eigentlichen Verzweigungspunkt.

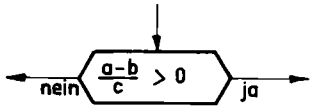
Für jedes dieser Elemente können gesonderte Symbole benutzt werden. Es ist aber häufig zweckmäßiger, statt dessen die folgenden, immer wiederkehrenden Kombinationen zu verwenden:

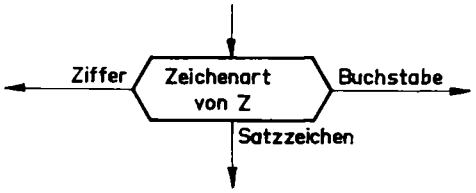
Ein *Durchlauf* durch die Verzweigung wird durch folgendes Symbol bezeichnet:



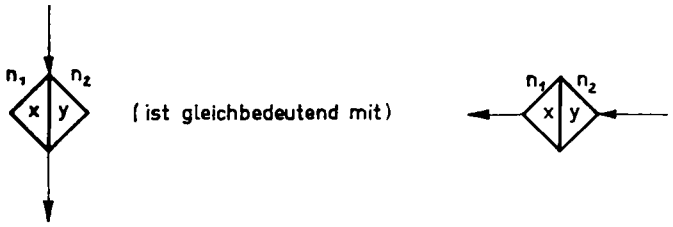
Der Kasten enthält *einen* Eingangspfeil und *mindestens zwei* Ausgangslinien in beliebiger Lage. Der Kasten selbst enthält im allgemeinen die Bedingung, die den Durchlauf durch die Verzweigung bestimmt. Die Zweige werden dazu mit bestimmten Texten gekennzeichnet, die in eindeutigem Zusammenhang mit den Bedingungen stehen.

Beispiele:





In diesem Fall ist das Stellen der Verzweigung mit inbegriffen. Soll das *Stellen* gesondert bezeichnet werden, so wird das Symbol

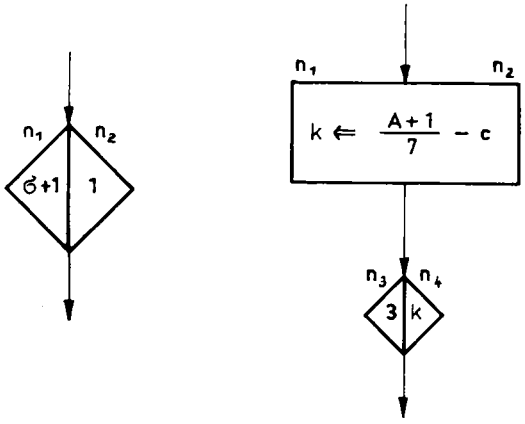


benutzt. Der Kasten mit *einem* Eingangspfeil und *einer* Ausgangslinie enthält ein Namenpaar x/y. Dabei bezeichnet x (links) die zu stellende Verzweigung und y (rechts) den Zweig y, der durchlaufen wird, wenn das Programm dieses Stellsymbol durchlaufen hat und vor dem Erreichen der Verzweigung nicht ein weiteres, die gleiche Verzweigung betreffendes Stellsymbol mit einer der ersten widersprechenden Stellanweisung durchläuft. Wir werden im folgenden hauptsächlich Dezimalzahlen zur Bezeichnung von Verzweigungen und Zweigen benutzen; so heißt also 3/1: Ausgang 1 der Verzweigung 3.

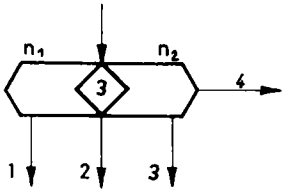
Wir wollen auch zulassen, daß die Namen der Verzweigung und auch des Zweiges nicht fest sind, sondern daß dazu Größen verwendet werden, mit denen gerechnet wird. So kann $3/k$ zum Beispiel bedeuten, daß bei Weiche 3 der Zweig k durchlaufen werden soll, wobei k eine Variable ist, deren Wert im Augenblick des Verzweigungsdurchlaufs (und nicht der Stellung) zu be-

nutzen ist. Wird k selbst nur zur Stellung berechnet, so kann die entsprechende Berechnungsvorschrift (im allgemeinen ein arithmetischer Ausdruck) mit in das Stellsymbol hineingeschrieben werden. Allerdings empfiehlt es sich, aus Platzgründen bei längeren Ausdrücken eine gesonderte Anweisung vorzuschalten. Für die Festlegung der Verzweigung gelten (im Gegensatz zur Bestimmung der Zweige) die Werte beim Durchlaufen des Stellsymbols (und nicht der Verzweigung).

Beispiel:



Auf diese Weise gestellte *Verzweigungen* werden genauso dargestellt wie die Bedingungsverzweigungen. Nur wird diesmal innerhalb des Kastens anstelle der Bedingung das Stellsymbol mit dem Verzweigungsnamen eingetragen. Die Namen der Ausgänge werden jeweils an die Ausgangslinien geschrieben. Eine zum rechts oben gezeichneten Stellsymbol gehörende Verzweigung kann also folgendermaßen aussehen:



Die oben gezeichnete Stellanweisung verbindet hier beispielsweise für $A = 20$ und $c = 1$ den Eingang mit dem Ausgang 2.