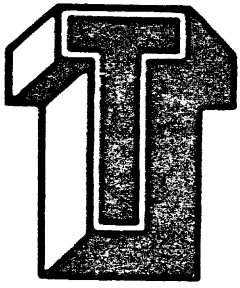
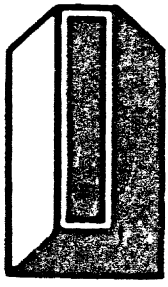
Kleinstrechner



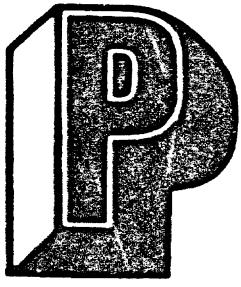
Tendenzen
und Theorien

Programmier-
sprachen



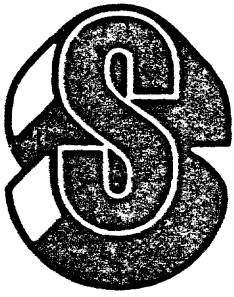
Informationen
und Ideen

Analogeingabe mit
Polycomputer 880



Programme
und Projekte

BASIC-Anwendung



Spaß
und Spiel

Kleinstrechner-TIPS

Heft 4

Mit 33 Bildern

Herausgegeben von
Prof. Dr.-Ing. Hans Kreul
Doz. Dr.-Ing. Wilhelm Leupold
Doz. Dr. sc. techn. Thomas Horn

VEB Fachbuchverlag Leipzig

Inhalt

ISBN 3-343-00128-7

Zum Inhalt

Die ersten drei Hefte dieser Broschürenreihe wurden von einem breiten Leserkreis mit großer Resonanz aufgenommen, wie uns die zahlreichen Leserzuschriften zeigten. Insbesondere fand die Broschürenreihe ein reges Interesse bei Lehrern und Schülerarbeitsgemeinschaften, was Ausdruck einer immer stärker werdenden Durchdringung vieler Bereiche unserer Gesellschaft mit Kleincomputern ist. Zur Förderung dieser Anwendungsbereiche wird die Broschürenreihe in ihren nächsten Heften gerade diesem sich entwickelnden Aspekt besondere Rechnung tragen. Gleichzeitig möchte das Herausgeberkollektiv hier die Möglichkeit nutzen, allen seinen Lesern für die vielen Zuschriften, kritischen Hinweise und Veröffentlichungsangebote zu danken. Wir möchten aber auch um Entschuldigung bitten, wenn wir nicht immer sofort jede Zuschrift beantworten können.

Der Inhalt des vorliegenden Heftes 4 der Broschürenreihe »Kleinstrechner-TIPS« wurde von dem Hauptanliegen geprägt, möglichst vielen Interessenten der Mikrorechentechnik die vielfältigen Probleme der Anwendung und vor allem der Programmierung von Mikrorechnern in einer verständlichen Weise nahezubringen.

In einem umfassenden Artikel von HORN wird dem Anfänger wie auch dem Fachmann ein Überblick über die Entwicklung der Programmiersprachen gegeben. An einem Abstammungsbaum wird zur »Entwirrung des Sprachengewirrs« die Einordnung wichtiger Programmiersprachen gezeigt. Die auf Mikrorechnern verbreiteten Programmiersprachen, wie FORTRAN, BASIC, PASCAL, C und FORTH, werden näher betrachtet.

LEHMANN, SCHUMANN und WALKE zeigen an einem Experimentalaufbau für Studenten der Ingenieurhochschule Dresden, wie man mit einem Kleincomputer analoge elektrische Signale in digitale Werte umsetzen kann.

Dem Problem der begrenzten Darstellungsgenauigkeit von Gleitkommazahlen auf Rechnern, einem Kardinalproblem numerischer Anwendungen, geht GUTZER in seinem Beitrag »Archimedes und die Zahl π« nach.

Im weiteren schlägt SCHÖNFELDER ein Sicherungsverfahren für Speicherinhalte auf Kassette vor. Ein BASIC-Programm für den Chemieunterricht wird von HEINS vorgestellt.

Anschließend möchten die Herausgeber und der Verlag hoffen, daß auch dieses Heft den Interessen unserer Leser entgegenkommt und möglichst viele Anregungen zur intensiveren Beschäftigung mit der Mikrorechentechnik geben möge.

Thomas Horn

Programmiersprachen – ein Vergleich an Hand von Beispielen

1. Einleitung

Mit der Entwicklung und Produktion der ersten Mikroprozessoren vor mehr als zehn Jahren begann zunächst eine Etappe, in der die Mikroprozessoren im Maschinenkode programmiert wurden. Die hexadezimale Verschlüsselung von Maschinenbefehlen, Adressen und Daten war unübersichtlich, wenig änderungsfreundlich und sehr fehleranfällig. Programme mit mehr als einigen hundert Byte waren praktisch nicht mehr beherrschbar.

Sehr bald setzte sich die Erkenntnis durch, daß die Erfahrungen und Programmierverfahren der klassischen Rechentechnik auf die Mikrorechentechnik übertragen werden können und müssen. Zunächst bediente man sich auf Grund der noch relativ geringen Leistungsfähigkeit der Mikrorechner und ihres vorrangigen Einsatzes in der Steuerungs- und Automatisierungstechnik der Assemblerprogrammierung. In der Assemblersprache werden die Maschinenbefehle über symbolische Bezeichnungen notiert. (Vgl. dazu den Beitrag »Zur Programmierung einer einfachen Aufzugssteuerung mit dem Mikroprozessor U 880« in Heft 2.) Die Programme wurden dadurch besser lesbar, änderungsfreundlicher und weniger fehleranfällig. Durch geeignete umfangreiche Kommentare wird das Programm sogar für andere verständlich. Assemblersprachen werden auch als Eins : Eins-Sprachen bezeichnet, weil ein symbolischer Maschinenbefehl (Assemblerzeile) einem echten Maschinenbefehl entspricht. Da jeder Mikroprozessor einen anderen Befehlssatz hat, gibt es folglich verschiedene Assemblersprachen, die miteinander nicht verträglich (inkompatibel) sind. Zum anderen lassen sich verschiedene symbolische Bezeichnungen für ein und denselben Maschinenbefehl einführen, z.B.

LD A, B abgeleitet von LoaD oder
MOV A, B abgeleitet von MOVe,

so daß für einen Mikroprozessor mehrere Assemblersprachen möglich sind. Weitaus mehr Möglichkeiten gibt es in der Festlegung des sonstigen Komforts einer Assemblersprache, wie Regeln zur Symbolbildung, Definitionsanweisungen, bedingte Assemblierung, Makrotechnik u.v.a.m. So gibt es gegenwärtig für den Mikroprozessor U 880 etwa 8 bis 10 Assemblersprachen.
Der wesentliche Nachteil der Assemblersprachen besteht in ihrer Maschinennähe, was bei der Programmierung die Kenntnis des Prozessors voraussetzt und eine Programmierung in elementaren Teilschritten (Befehlen) entsprechend der Arbeitsweise des Mikroprozessors erzwingt. Eine triviale Auf-

```
00001          PN   HA
00002  ;
00003  ;  ADDITION VON 20 ZAHLEN
00004  ;
00005  UPADD:  LD   A,0                ;LOESCHEN DES AKKUMULATORS
00006          LD   B,20               ;ANZAHL DER ELEMENTE
00007          LD   HL,TAB             ;ADRESSE DER TABELLE
00008  M0:     ADD  M                  ;ADDITION
00009          INC  HL                 ;ADRESSE DES NAECHSTEN ELEMENTS
00010          DJNZ M0-#               ;SCHLEIFENORGANISATION
00011          LD   (S),A              ;SPEICHERN DER SUMME
00012          RET                     ;<-- RUECKSPRUNG
00013  ;
00014  S:      BER  1
00015  TAB:    BER  20
00016          END
```

Bild 1. Assemblerprogramm zur Addition von 20 Zahlen

gabe, wie z.B. die Berechnung der Summe S von 20 Werten, die in einer Tabelle TAB stehen, ist in der Assemblersprache SYPS-1520 für den Mikroprozessor U 880 schon ein anspruchsvolles Programm (Bild 1). Da das Assemblerprogramm dem Mikroprozessor in der symbolischen Schreibweise unverständlich ist, muß es vor der Ausführung von einem Übersetzungsprogramm (genannt Assembler) in den internen (hexadezimalen) Maschinenkode übersetzt werden. Es gilt zu beachten, daß das Assemblerprogramm nur auf dem Mikroprozessor U 880 lauffähig ist, da Befehlssatz und Registerstruktur bei anderen Prozessoren anders sind, d.h., Assemblerprogramme sind im allgemeinen nicht auf andere Prozessoren übertragbar (portabel).

Aus Gründen der Portabilität der Programme, der Erhöhung der Effektivität der Programmierung, der drastischen Reduzierung von möglichen Fehlerquellen sowie der Entlastung der Programmierer von unnötigem Detailwissen über die verwendeten Prozessoren wurden schon sehr frühzeitig sogenannte höhere Programmiersprachen entwickelt und eingesetzt. Programme in höheren Programmiersprachen sind im allgemeinen gut lesbar, kompakt in der Darstellung, leicht verständlich und der Denkweise des Menschen in den einzelnen Wissensgebieten angepaßt, aber für die einzelnen Prozessoren nicht verständlich. Aus diesem Grunde müssen Programme für die einzelnen Prozessortypen verfügbar sein, die die Abarbeitung von Programmen in höheren Programmiersprachen zulassen. Es werden hierbei zwei grundsätzlich verschiedene Abarbeitungsverfahren unterschieden:

1. Die Compilation
   Bei der Compilation wird das gesamte Quellprogramm (Programm in der höheren Programmiersprache) geschlossen in die Maschinensprache übersetzt. Anschließend erfolgt die Abarbeitung des generierten Maschinenkodeprogramms. Das Übersetzungsprogramm wird als Compiler bezeichnet. Die prinzipielle Vorgehensweise entspricht etwa der Abarbeitung von Assemblerprogrammen, nur daß aus einer höheren, prozessorunabhängigen Sprache übersetzt wird.
2. Die Interpretation
   Bei der Interpretation wird das Quellprogramm zeilenweise übersetzt und sofort abgearbeitet. Es wird kein Maschinenkodeprogramm erzeugt.
   Die interpretative Abarbeitung kommt insbesondere einer Dialogprogrammierung entgegen, da das

Programm zeilenweise eingegeben, abgearbeitet und somit gleich getestet werden kann, d.h., das komplette Quellprogramm muß zu Beginn der Abarbeitung nicht vorliegen. Es kann im Dialog zeilenweise bei der Abarbeitung fortgeschrieben werden. Das Programm für die Abarbeitung der Quellprogramme wird als Interpreter bezeichnet.

Seit der Entwicklung der ersten höheren Programmiersprachen vor ca. 30 Jahren setzte auf diesem Gebiet eine stürmische Entwicklung ein. Heute sind nach Schätzungen mehr als 1000 Programmiersprachen bekannt. Für den Laien gleicht das Gebiet der Programmiersprachen zweifellos einer Art »babylonischer Sprachverwirrung«.

Mit der weiteren Steigerung der Leistungsfähigkeit der Mikroprozessoren und Vergrößerung der Speicherkapazitäten werden für die verschiedenen Mikrorechner immer mehr Interpreter und Compiler für die verschiedensten Programmiersprachen entwickelt, so daß man heute schon davon ausgehen muß, daß für die leistungsfähigsten Personal- und Kleincomputer die wichtigsten Programmiersprachen verfügbar sind. Der vorliegende Beitrag soll deshalb dem Leser helfen, sich in der Vielzahl der möglichen Programmiersprachen zu orientieren. Auf die wichtigsten Programmiersprachen soll etwas näher eingegangen werden, wobei im Rahmen dieses Beitrages keine Sprachbeschreibung gegeben werden kann. Als ein anspruchsvolles Beispielprogramm soll hier auf die Textverarbeitungsaufgabe eingegangen werden, die im Heft 3 im Beitrag »Wie kann ein Programm systematisch entworfen werden?« mittels der Struktogrammtechnik für die Programmierung vorbereitet wurde.

2. Eine historische Übersicht

Mit der Entwicklung der ersten frei programmierbaren Rechenanlagen Ende der vierziger Jahre begann die Entwicklung sogenannter Systeme für die symbolische Kodierung (Assemblersprachen). Die ersten »höheren« Programmiersprachen wurden aus der ökonomischen Notwendigkeit einer effektiveren Programmierung bereits in den Jahren 1953–1954 entwickelt (Bild 2).

Die bedeutendste Programmiersprache war FORTRAN (FORmula TRANslation). Ihre Haupteinsatzgebiete waren ursprünglich mathematisch-wissenschaftliche Aufgabenstellungen. Sie war als compilierend abzuarbeitende Programmiersprache entstanden, die laufzeiteffektive Maschinenkodeprogramme liefert und somit eine Alternative zur Assemblerprogrammierung darstellt. Unter dem Einfluß von ALGOL-60 und PASCAL wurde FORTRAN später zu moderneren und leistungsfähigeren Sprachversionen (FORTRAN IV, FORTRAN 77) weiterentwickelt. FORTRAN beeinflußte maßgeblich die weitere Entwicklung auf dem Gebiet der zu compilierenden Programmiersprachen, insbesondere die Erarbeitung der Programmiersprachen ALGOL-60 und COBOL.

ALGOL-60 (ALGOrithmic Language 1960) war die erste Sprache, die von einem internationalen Wissenschaftlerkollektiv erarbeitet und auf der IFIP[1]-Hauptkonferenz 1960 von 60 Vertretern aus 7 Ländern angenommen wurde. Hauptanwendungsgebiete von ALGOL-60 sind wissenschaftlich-technische und mathematische Berechnungen. ALGOL-60 zeichnete sich durch Klarheit, Einfachheit und Eleganz in der Sprachstruktur aus. Das

[1] IFIP — International Federation for Information Processing

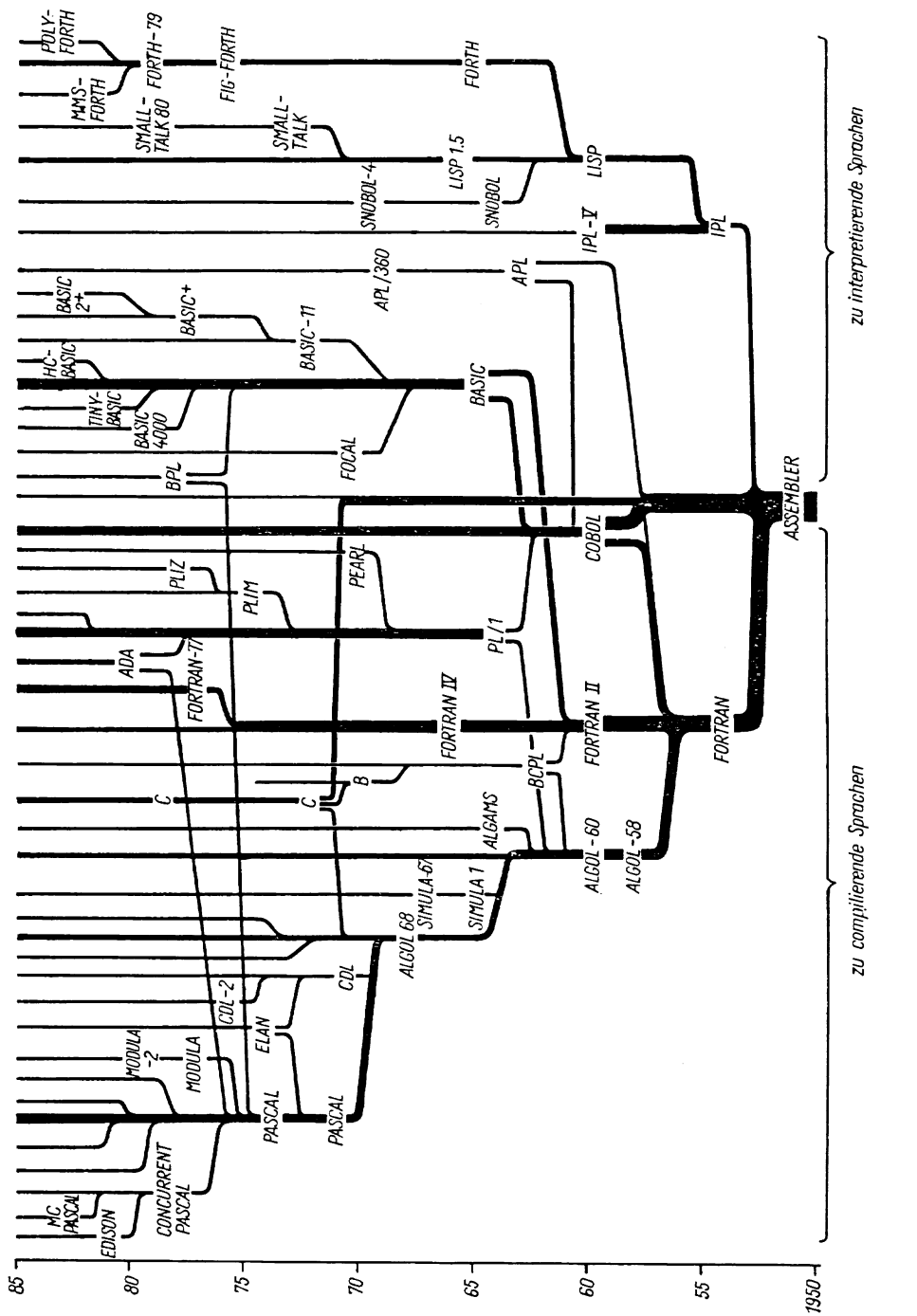


Bild 2. Abstammungsbaum der Programmiersprachen

erste Mal wurde eine formale Grammatik, die sogenannte Backus-Naur-Form (BNF), zur Definition einer Programmiersprache benutzt. Obwohl ALGOL-60 heute an Bedeutung verloren hat, stellte die Definition dieser Sprache eine wichtige Etappe in der Entwicklungsgeschichte höherer Programmiersprachen dar und zeigte einen großen Einfluß auf die Entwicklung von solchen wichtigen Programmiersprachen, wie PASCAL mit seinen vielen Nachkommen, SIMULA und C. Die BNF ist heute das am weitesten verbreitete formale Beschreibungsmittel für Programmiersprachen. Die weiterentwikkelte Sprache ALGOL-68 zeichnet sich durch eine wesentlich vergrößerte, ja sogar allumfassende Anwendungsbreite und eine enorme Komplexität aus. Trotzdem fand ALGOL-68 gegenüber ALGOL-60 eine geringere Verbreitung, was im wesentlichen zwei Ursachen hatte:

- Die Grammatik von ALGOL-68 war für den durchschnittlich gebildeten Programmierer nicht mehr überschaubar.
- Der Aufwand für die Entwicklung eines Compilers war enorm hoch geworden (über 100 Mann-Jahre), und die Abarbeitung des Compilers erforderte ausgesprochene Großrechenanlagen.

Parallel zu ALGOL-60 wurde in den USA unter dem Einfluß des Pentagon die Programmiersprache COBOL (COmmon Business Oriented Language) für die kommerzielle Datenverarbeitung entwickelt. Mit etwa 350 Schlüsselworten stellt die Sprache praktisch eine Teilmenge der englischen Sprache dar, die aber für den Laien nicht mehr verständlich ist. Ihre Anwendung setzt eine umfassende Programmierausbildung voraus. Die Vorteile der Sprache liegen zweifellos in der guten Beschreibung von Datenstrukturen, wie sie zur Verarbeitung und Erstellung von Geschäftspapieren u.ä. benötigt werden. Trotz der außerordentlich großen Verbreitung in den USA und zahlreicher Versuche, die Sprache durch neue Standards zu »modernisieren«, wird international eingeschätzt, daß sie den heutigen Anforderungen nicht mehr genügt. Insbesondere stützen sich solche Aussagen auf fehlende Möglichkeiten der Programmstrukturierung und die enorm hohen Aufwendungen für die Programmpflege und -weiterentwicklung.

Die Programmiersprache PL/1 (**P**rogramming **L**anguage nr. **1**) wurde 1963/64 mit dem Erscheinen der ersten Rechnerfamilie der 3. Generation, IBM/360, gemeinsam von der Fa. IBM mit der Nutzerorganisation SHARE entwickelt, um auf Grund der bekannten Nachteile von FORTRAN, ALGOL-60 und COBOL diese Sprachen abzulösen. Vorteile von PL/1 sind zweifellos der modulare Aufbau der Programme (Blockstruktur), programmierbare Fehlerbehandlungen, Anwendbarkeit für numerische und nichtnumerische Datenverarbeitung, dynamische Speicherverwaltung und eine Makrosprache. Nachteilig wirkt sich aus, daß sehr viele Standardannahmen getroffen werden, kein Deklarationszwang besteht und wenig syntaktische Fehler festgestellt werden, weil das Hauptprinzip ist: »Alles ist machbar!« Praktisch erhielt aber PL/1 außer auf IBM-kompatiblen Rechenanlagen keine größere Verbreitung. In Anlehnung an PL/1 entstanden Mitte der siebziger Jahre die Sprachen PL/M und PL/Z für die Programmierung der Mikroprozessoren Intel 8080 bzw. Z 80. Diese Programmiersprachen stellen keine Subsets (Subset – echte Teilmenge) von PL/1 dar. In den letzten Jahren sind aber auch Versuche unternommen worden,

PL/1-Subsets auf Mikroprozessoren zu implementieren.

PASCAL[1] wurde 1971 von N. WIRTH (ETH Zürich) auf der Grundlage der Erfahrungen mit ALGOL-68 als Lehrsprache erarbeitet. Der Quasi-Standard ist der revidierte Pascal-Bericht aus dem Jahre 1974. PASCAL hat eine sehr einfache, klare und gut definierte Sprachstruktur. Sie fordert eine streng strukturierte Programmierung. Ein neues leistungsstarkes Ausdrucksmittel ist die Typvereinbarung. Die wesentlichen Mängel von PASCAL sind die stark eingeschränkte E/A-Arbeit (Fileverarbeitung), fehlender Hardwarezugang (Echtzeitvorkehrungen) und ein fehlendes Makrokonzept. Daraus resultieren zahlreiche PASCAL-Dialekte (NBS-PASCAL, OMSI-PASCAL, PASCAL-P, PASCAL-2, PASCAL-1600 usw.). Die Bemühungen zur Standardisierung von der ISO[2] und vom IEEE[3] setzten praktisch zu spät ein. Der Einfluß von PASCAL auf die Weiterentwicklung der Programmiersprachen kann aber nicht hoch genug eingeschätzt werden. So sind weitere interessante, aber noch relativ wenig verbreitete Sprachkonzepte als Nachkommen von PASCAL entstanden, wie MODULA-2, CONCURRENT-PASCAL, MC-PASCAL, EDISON u.a. Im Zusammenhang mit PL/1 und ALGOL-68 hat PASCAL auch einen wesentlichen Einfluß auf die neue »Supersprache« ADA gehabt, die auf Initiative des Pentagon in den USA Ende der siebziger Jahre entwickelt wurde. ADA zeigt wie jede sehr große Sprache wenig »Kundenfreundlichkeit«, insbesondere hat ADA einen echten Mangel an einfachen E/A-Operationen. Auch die Notwendigkeit von relativ großen leistungsfähigen Rechenanlagen engt ADA in seiner weiteren Verbreitung sehr stark ein.

Die Programmiersprache C, ein Nachkomme von B bzw. BCPL und ALGOL-68, wurde 1972 von RITCHIE (Bell Telephone Laboratories) zur Implementation des Betriebssystems UNIX auf PDP-11-Kleinrechnern entwickelt. Das Sprachkonzept von C, größtenteils aus der Praxis der Systemprogrammierung geboren, hat sich als sehr lebensfähig erwiesen. Auf Grund seiner Realisierung in C war das Betriebssystem UNIX ausgezeichnet portabel und konnte somit auf alle 16-Bit-Mikrorechner übertragen werden. Durch das Betriebssystem UNIX fand C eine weite Verbreitung auf fast allen Groß-, Klein- und Mikrorechnern. C verfügt über alle modernen Datentypen und Steuerstrukturen sowie über eine leistungsfähige File-E/A sowie ein gutes Makrokonzept. Nachteile von C sind der sehr kompakte Programmtext, der folglich schlecht lesbar ist.

Die Programmiersprache CDL (Compiler Description Language) wurde 1970 als Hilfsmittel zur Implementierung eines ALGOL-68-Compilers (der niemals fertiggestellt wurde!) von C. H. A. KOSTER (TU Berlin (West)) entwikkelt. In ihrer Weiterentwicklung als CDL-2 stellt die Sprache ein effektives Werkzeug zur Entwicklung großer Programmsysteme dar. CDL verwendet zur Erzielung einer hohen Laufzeiteffektivität eine umfangreiche Bibliothek mit Funktionsmoduln, die in der Regel in der Assemblersprache entwickelt werden. CDL organisiert den Steuerfluß und die Parameterübergabe zwischen den Funktionsmoduln. Das CDL-Konzept geht somit von einem Zwei-

---

[1] Benannt nach BLAISE PASCAL (1623 bis 1662). Erfinder der ersten Addiermaschine (1641).

[2] International Organization for Standardization

[3] American Institution of Electrical and Electronical Engineers

sprachniveau aus. Das obere Niveau, in CDL geschrieben, realisiert die rechnerunabhängige Organisationsstruktur des Programms, während das untere Niveau die rechnerabhängigen Funktionsmoduln umfaßt. Die Nachteile von CDL bestehen eben in der Beherrschung dieser zwei Sprachniveaus sowie der umfangreichen Menge von Funktionsmoduln. CDL ist gegenwärtig auf Mikrorechnern noch wenig verbreitet.

Auf der Grundlage des damals vorliegenden Erfahrungsschatzes, insbesondere der Erfahrungen mit CDL, wurde 1974 unter maßgeblicher Beteiligung von C. H. A. KOSTER die Programmiersprache ELAN[1] (Elementary LANguage) als Schulsprache entwickelt. Unter Berücksichtigung der Erfahrungen mit PASCAL als »reine« Lehrsprache wurde von ihm das Konzept der Entwicklung verschiedener Sprachstufen für die Ausbildung in Schulen und Universitäten bis hin zur Anwendung in der Systemprogrammierung realisiert. ELAN-Compiler liegen heute auf einer Vielzahl von Groß-, Klein- und Mikrorechnern (u.a. Z 80) vor. Ihre Verbreitung setzt sich trotz alledem gegenwärtig nur zögernd durch. Diese Situation kann sich aber mit der Einbeziehung des Informatik-Unterrichts in die Schulausbildung ändern. Versuche dazu laufen in einigen Ländern der BRD, in der UdSSR, in der UVR, in der VRB u.a. Allerdings bestehen international noch keine einheitlichen Auffassungen über die Schulsprache der Zukunft. ELAN wäre hier gewiß aber ein aussichtsreicher und die weitere Entwicklung beeinflussender Anwärter.

Betrachten wir nun die interpretativ abarbeitbaren Programmiersprachen im rechten Teil des Sprachbaumes (Bild 2). Die Interpretation ist die Voraussetzung für einen Dialogbetrieb. Deshalb werden die meisten der folgenden Sprachen auch als Dialogsprachen bezeichnet. Die Dialogprogrammierung gestattet vor allem eine effektive Programmentwicklung und -testung. Zur effektiven Abarbeitung ausgetesteter Programme werden oftmals auch Compiler zur Verfügung gestellt, die das Quellprogramm in ein laufzeiteffektives Maschinenkodeprogramm übersetzen.

Drei wichtige Sprachfamilien können hier unterschieden werden: LISP, APL und BASIC. Während die LISP-Familie parallel zu FORTRAN und im wesentlichen unabhängig davon sich entwickelte, stellen APL und BASIC in gewissem Sinne Alternativen zu den compilierend abarbeitbaren Sprachen COBOL bzw. FORTRAN dar.

Die Programmiersprache LISP hat ihren Ursprung in der Programmiersprache IPL (Information Processing Language), die auf Arbeiten von NEWELL, SHAW und SIMON (RAND-Corp./USA) aus dem Jahre 1954 beruht. IPL-V ist eine prozedurorientierte Sprache zur Listenverarbeitung, die praktisch einer Assemblersprache eines hypothetischen Rechners ähnelt. Diese Sprache hat heute fast keine Bedeutung mehr. Die Programmiersprache LISP (LISt Processing) wurde Ende der fünfziger Jahre unter Leitung von MCCARTHY (MIT Cambridge/USA) für die nichtnumerische Datenverarbeitung erarbeitet. Die Probleme werden in Baumstrukturen dargestellt, die intern als Listenstrukturen gespeichert werden. Die Verarbeitung der Listen erfolgt unter Anwendung des Kellerprinzips (Stack). LISP hat eine sehr einfache und übersichtliche Struktur. Für die Operationen mit den Listenstrukturen und -elementen stehen ca. 90 Standard-

[1] Erst SLAN (School LANguage), 1976 Umbenennung in ELAN

operationen zur Verfügung. Das Hauptanwendungsgebiet von LISP ist heute die Programmierung von Problemen der künstlichen Intelligenz. Daraus ergibt sich ihre vorrangige Verbreitung auf Großrechenanlagen.

Die Programmiersprache FORTH (eigentlich FOuRTH – Programmiersprache der »vierten« Generation) wurde von dem Amerikaner CHARLES H. MOORE in den Jahren 1960–1973 als Werkzeug für eine effektivere Programmierung entwickelt. Ähnlich wie in LISP erfolgt die Programmierung in baumartigen Strukturen. Die Verarbeitung der Daten erfolgt auf der Grundlage des Stackprinzips. Für die Programmierung von Ausdrücken wird, wie auch in LISP, das Prinzip der inversen polnischen Notation (s.u.) genutzt. Die FORTH-Funktionen werden in Wörterbüchern zusammengefaßt. FORTH erhielt in den letzten Jahren eine außerordentlich große Verbreitung auf fast allen Mikrorechnern, was insbesondere auf seine Laufzeiteffektivität und die geringe Größe des Interpreters zurückzuführen ist. Der Nachteil von FORTH besteht in der schlechten Lesbarkeit der Programme und dem Fehlen von jeglichen Kontrollen über die auszuführenden Operationen. Die FORTH-Systeme realisieren im allgemeinen den Quasi-Standard FORTH-79 der FORTH Inc. (Los Angeles), der auf dem Vorschlag FIG-FORTH der FORTH Interested Group beruht.

Die Programmiersprache SNOBOL (String Manipulation Language) ist im wesentlichen eine Sprache für die Zeichenkettenverarbeitung, die Anfang der sechziger Jahre bei Bell Telephone Laboratories (USA) entwickelt wurde. Auch hier werden zur Zeichenkettenverarbeitung und Symbolmanipulation gewisse Listentechniken verwendet. SNOBOL ist international wenig verbreitet.

Die Programmiersprache SMALLTALK wurde Anfang der siebziger Jahre als eine LISP-Erweiterung insbesondere für die Verarbeitung von grafischen Informationen bei Xerox in den USA entwickelt. In den letzten Jahren hat die Verbreitung von SMALLTALK auf Mikroprozessoren mit Rastergrafik-Bildschirmsystemen stark zugenommen.

Die Programmiersprache APL (A Programming Language) beruht auf einem Sprachvorschlag von K. IVERSON aus dem Jahre 1962. Sie fand auf Rechenanlagen vom Typ IBM/360 und Tischrechnern IBM 5100 insbesondere für kaufmännische Anwendungen in den USA eine weite Verbreitung. Da APL 61 Sonderzeichen benutzt und somit spezielle APL-Terminals bzw. APL-Tastaturen voraussetzt, ist ihre Verbreitung bisher im wesentlichen auf IBM-Anlagen beschränkt geblieben. Da sich aber gegenwärtig eine Tendenz zum Übergang auf Terminals mit einem multinationalen Zeichensatz und einem erweiterten Vorrat an Sonderzeichen (englisches, deutsches, griechisches Alphabet und alle mathematischen und sonstigen Sonderzeichen) bemerkbar macht, könnte diese Entwicklung zur weiteren Verbreitung von APL führen.

Die Programmiersprache BASIC (Beginners All purpose Symbolic Instruction Code) ist mit ihren zahlreichen Dialekten eine der umfangreichsten Sprachfamilien. BASIC wurde in den Jahren 1963/64 von KURTZ und KEMENY am Dartmouth College in New Hampshire (USA) als Lehrsprache für »Laien« entwickelt. BASIC ist heute die am weitesten verbreitete Programmiersprache. Es wird eingeschätzt, daß etwa 10 bis 15 Millionen Menschen BASIC beherrschen. Da der ursprünglich definierte Sprachkern relativ klein war, wurden unzählige Erweiterungen vorgenommen, so daß die Menge der

BASIC-Dialekte kaum noch überschaubar ist. Der Sprachkern benutzt nur 10 Schlüsselwörter. BASIC ist damit leicht erlernbar und durch die interpretative Abarbeitung im Dialog besonders benutzerfreundlich. Trotzdem muß heute eingeschätzt werden, daß das Sprachkonzept veraltet ist. Die schwächsten Stellen von BASIC sind die Zeilenorientierung, die obligatorische Zeilennumerierung und die Einbeziehung der Zeilennummern in den Logikfluß.

Die Programmiersprache FOCAL (Formulating On-line Calculations in Algebraic Language) ist eine BASIC-verwandte Programmiersprache, die von der Fa. Digital Equipment Corporation (USA) Ende der sechziger Jahre für PDP-8- und später PDP-11-Rechner entwickelt wurde. Auf Grund der weiten Verbreitung der PDP-8- und PDP-11-Rechner erhielt auch FOCAL eine relativ weite Verbreitung. Für die wesentlichen Vor- und Nachteile von FOCAL gilt das gleiche wie für BASIC.

Zur Weiterentwicklung von BASIC wurde Ende der siebziger Jahre die Sprache BPL (Basic-Pascal-Liaison) an britischen Universitäten als eine BASIC-PASCAL-Synthese entwickelt, die die leichte Erlernbarkeit von BASIC mit der guten Strukturierung von PASCAL verbinden sollte. Dieses modernere Sprachkonzept hat sich aber international noch nicht durchgesetzt.

Zum Abschluß der Betrachtung der Programmiersprachen sei noch auf eine große Gruppe von Echtzeitprogrammiersprachen verwiesen. Da die bisher betrachteten Programmiersprachen den Anforderungen für eine Anwendung in der Prozeßsteuerung, Automatisierungstechnik und Robotertechnik nicht genügten, entstanden zahlreiche Echtzeitsprachen durch Erweiterung bekannter Sprachkonzepte mit echtzeittypischen Sprachelementen, z.B. Real-Time-FORTRAN, Real-Time-BASIC, PL/Z-RTC usw. Es wurden aber auch eine Reihe von Echtzeitsprachen mit einem selbständigen Sprachkonzept erarbeitet. Eine der wichtigsten Sprachen, die in der BRD unter maßgeblicher Mitwirkung einiger Hochschulen entwickelt wurde, ist die Programmiersprache PEARL (Process and Experiment Automation Realtime Language). PEARL basiert im wesentlichen auf ALGOL-68 und wurde in der Syntax (Regeln zur Bildung der Anweisungen) an PL/1 angepaßt. PEARL existiert heute für fast alle Prozeß-, Klein- und Mikrorechner.

Damit sei die übersichtsmäßige Betrachtung der wichtigen Vertreter aus der Menge der über 1000 Programmiersprachen und Dialekte beendet. Im weiteren wollen wir uns auf fünf Programmiersprachen beschränken, die auf den Mikrorechnern weit verbreitet sind. Das sind:

- FORTRAN-IV,
- BASIC,
- PASCAL,
- C,
- FORTH.

Die Reihenfolge der Betrachtung der Programmiersprachen wurde aus methodischen Gründen so gewählt und stellt keine Wertung dar.

3. Die Programmiersprache FORTRAN-IV

Die Programmiersprache FORTRAN-IV ist wie alle älteren Programmiersprachen zeilenorientiert. Eine Programmzeile besteht aus einem Feld für eine sechsstellige Zeilennummer, einem Feld für ein Fortsetzungszeichen und einem Feld für die Anweisung, die den Rest der Zeile belegen darf. Ist die Zeile zur Aufnahme der Anweisung nicht lang genug, so darf die folgende Zeile mit einem Fortsetzungszeichen

markiert und die Anweisung fortgeschrieben werden. Die Zeilennummer ist nicht obligatorisch, sie wird nur für Sprunganweisungen (GOTO-Anweisung), für die Schleifenorganisation (DO-Anweisung) und für die FORMAT-Anweisung benutzt.

Ein Programm beginnt mit der PROGRAM-Anweisung, die den Programmnamen festlegt, und endet mit einer END-Anweisung. Nach der PROGRAM-Anweisung werden in der Regel die Variablen und Felder deklariert. Es kann mit ganzzahligen Daten (INTEGER-Anweisung), Gleitkomma-Daten (REAL-Anweisung) und Daten im Byteformat (BYTE-Anweisung) gearbeitet werden. Es besteht aber kein Deklarationszwang. Für nichtdeklarierte Variablen wird, wenn der erste Buchstabe I, J, K, L, M oder N ist, angenommen, daß sie Daten vom Typ INTEGER spezifizieren. Alle anderen symbolischen Namen spezifizieren REAL-Daten.

Die symbolischen Namen werden aus Buchstaben und Ziffern gebildet, wobei das erste Zeichen ein Buchstabe sein muß. In Abhängigkeit vom Compiler können die symbolischen Namen 6 oder 8 Zeichen lang sein.

Felder müssen deklariert werden, damit der Compiler die Feldgröße kennt, z.B. bedeutet BYTE ZP(1ØØ), daß ein eindimensionales Feld mit 100 Bytes vereinbart wird. Der Zugriff auf die Feldelemente erfolgt durch Angabe eines Indexes, der ab eins gezählt wird:

ZP(1) – ist das erste Byte,
ZP(I) – ist das Byte mit der Nummer *i*,
ZP(100) – ist das letzte Byte.

Konstanten können im Programm benutzt werden. INTEGER-Konstanten sind positive oder negative Zahlen ohne Dezimalpunkt. REAL-Konstanten werden mit einem Dezimalpunkt oder Exponenten spezifiziert. Zeichenkonstanten und Zeichenkettenkonstanten werden durch Hochkommas begrenzt, z.B. '/' oder 'OLD'.

In FORTRAN-IV sind folgende Anweisungen zulässig:

a) Zuordnungsanweisungen (Ergibt-Anweisung)

variable = ausdruck

In Ausdrücken können Variablen und Konstanten über die Operatoren +, —, * und / verknüpft werden, z.B. bedeutet L = L — 1, daß der Variablen L der Wert L — 1 zugewiesen werden soll.
Runde Klammern sind zulässig.

b) Sprunganweisung

GOTO marke

Die Marke gibt die Zeilennummer der Anweisung an, bei der das Programm fortgesetzt werden soll.

c) Bedingte Anweisung

IF bedingung marke

Wenn die Bedingung wahr (true) ist, so wird zur angegebenen Marke verzweigt. Modernere Sprachversionen lassen auch

GOTO marke

zu. Die Bedingung stellt einen logischen Ausdruck dar, in dem folgende Operatoren zulässig sind:

.OR. – oder
.AND. – und
.NOT. – nicht
.EQ. – gleich
.NE. – ungleich
.LT. – kleiner als
.GT. – größer als
.LE. – kleiner gleich
.GE. – größer gleich

Zur Steuerung der richtigen Reihenfolge der Berechnung des Ausdrucks sind runde Klammern zulässig.

```
      program text
      integer i,j,w,z,zmax,l,e
      byte c,zp(1ØØ),wp(5Ø),ep(1ØØ)
      zmax=6Ø
c     Eroeffnen der Files
      open(unit=1,name='text.txt',type='OLD')
      open(unit=2,name='text.doc',type='NEW',carriagecontrol='LIST')
      w=1       ! Loeschen des Wort-, Zeilen- und Eingabepuffers
      z=1
      l=Ø
c     Einlesen der Druckzeile
1ØØØ  if (l.gt.Ø) goto 1Ø2Ø
      read (1,1Ø1Ø,end=11ØØ)l,ep
1Ø1Ø  format(q,1ØØa)
      e=1
1Ø2Ø  c=ep(e)
      e=e+1
      l=l-1
      if (c.eq.' ') goto 12ØØ       ! Auswahl
      goto 13ØØ
11ØØ  wp(1)='#'
      wp(2)='/'
      w=3
      c=' '
      goto 14ØØ
12ØØ  if ((wp(1).eq.'#').or.(w.eq.1)) goto 14ØØ
      if (z.eq.1) goto 121Ø
      zp(z)=' '
      z=z+1
121Ø  do 122Ø i=1,w-1
      zp(z)=wp(i)
122Ø  z=z+1
      w=1
      goto 14ØØ
13ØØ  wp(w)=c
      w=w+1
      if (l.eq.Ø) goto 12ØØ
14ØØ  if(((w.ne.3).or.(wp(1).ne.'#')).and.((zmax-z).ge.w)) goto 1ØØØ
c     Aufbereiten der Druckzeile
2ØØØ  do 2Ø1Ø i=1,z-1
      if (zp(i).eq.' ') goto 2Ø2Ø
2Ø1Ø  continue
2Ø2Ø  if ((z.eq.1).or.(i.eq.z).or.(wp(1).eq.'#')) goto 3ØØØ
2Ø3Ø  if (z.gt.zmax) goto 3ØØØ
      j = z-1
2Ø4Ø  if ((z.gt.zmax).or.(j.le.1)) goto 2Ø3Ø
      do 2Ø5Ø j=j,1,-1
      if (zp(j).eq.' ') goto 2Ø6Ø
```

```
2Ø5Ø   continue
2Ø6Ø   if (j.le.1) goto 2Ø3Ø
       z=z+1
       do 2Ø7Ø i=z-1,j+1,-1
2Ø7Ø   zp(i)=zp(i-1)
       do 2Ø8Ø j=j,1,-1
       if (zp(j).ne.' ') goto 2Ø4Ø
2Ø8Ø   continue
       goto 2Ø4Ø
c      Ausgabe der Druckzeile
3ØØØ   if (z.eq.1) goto 3Ø2Ø
       write (2,3Ø1Ø) (zp(i), i=1,z-1)
3Ø1Ø   format (1ØØa)
       z=1
3Ø2Ø   if ((w.ne.3).or.(wp(1).ne.'#')) goto 4ØØØ
       write (2,3Ø1Ø)
       if (wp(2).eq.'%') write (2,3Ø1Ø)
       w=1
c      Test auf Programmende
4ØØØ   if ((wp(1).ne.'#').or.(wp(2).ne.'/')) goto 1ØØØ
c      Schliessen der Files
       close(unit=1)
       close(unit=2)
       end
```

Bild 3. Textverarbeitungsprogramm in FORTRAN-IV

d) Schleifenanweisung

DO marke variable=anfangswert, endwert, schrittweite

Die Schleife beginnt mit der DO-Anweisung und endet mit der Zeile, die mit der angegebenen Zeilennummer (Marke) versehen ist. Die Schleife wird so oft ausgeführt, daß die Schleifenvariable alle Werte vom Anfangswert bis zum Endwert bei der vorgegebenen Schrittweite annehmen kann. Die Schrittweite 1 muß nicht angegeben werden. Ist die letzte Anweisung einer Schleife eine bedingte Anweisung, so muß die Leeranweisung CONTINUE ergänzt werden.

Beispiel: DO 2Ø1Ø I = 1, Z — 1

```
      IF (ZP (I) .EQ. 'b')
          GOTO 2Ø2Ø
2Ø1Ø  CONTINUE
```

Im Feld ZP wird ab der Position 1 beginnend ein Leerzeichen (*b*–blank) gesucht. Wenn es gefunden wird, so wird die Schleife verlassen und zur Marke 2Ø2Ø gesprungen.

Das Beispielprogramm aus Bild 1 würde in FORTRAN-IV wie folgt formuliert sein:

```
      S = Ø
      DO 1Ø I = 1, 2Ø
1Ø    S = S + TAB(I)
```

e) Ein- und Ausgabeanweisungen

READ (gerätenummer, marke) variablenliste

WRITE (gerätenummer, marke)
, variablenliste

Die READ-Anweisung dient zum Einlesen eines Satzes, wobei die eingelesenen Daten den Variablen der Variablenliste zugewiesen werden. Die angegebene Marke gibt die Zeilennummer einer FORMAT-Anweisung an, die die Datenkonvertierungen steuert. Die WRITE-Anweisung organisiert entsprechend die Ausgabe. Zur besseren Stützung der Fileverarbeitung wurde später in moderneren Sprachversionen eine OPEN- und eine CLOSE-Anweisung für das Eröffnen und Abschließen von Datenfiles aufgenommen.

Zum besseren Verständnis kann das Programm mit Kommentarzeilen (C in Position 1) und zusätzlichen Kommentaren (! als Kommentarkennzeichen) zu den Anweisungen erläutert werden. Diese kurzen Erläuterungen sollen hier als Überblick über die Sprache FORTRAN genügen. Die ausführliche Sprachbeschreibung ist der entsprechenden Fachliteratur zu entnehmen.
Bild 3 zeigt das FORTRAN-Programm für die Textverarbeitungsaufgabe aus Heft 3. Es sei noch bemerkt, daß modernere FORTRAN-Versionen auch die durchgängige Kleinschreibung zulassen, während früher ausschließlich auf die Großschreibung orientiert wurde.

4. Die Dialogprogrammiersprache BASIC

Die Programmiersprache BASIC ist ebenfalls wie FORTRAN-IV zeilenorientiert. Die Programmzeile besteht aus einem Feld für eine variabel lange Zeilennummer und einem Feld für die Anweisung, das den Rest der Zeile belegt. Fortsetzungszeilen sind nicht möglich. Die Zeilennummer ist obligatorisch; sie muß in aufsteigender Folge vergeben werden.

Das Programm beginnt mit der Anweisung mit der niedrigsten Zeilennummer und endet mit der END-Anweisung. Es gibt in BASIC keine Deklarationsanweisungen. Ursprünglich waren in BASIC nur REAL-Daten zulässig. Spätere Spracherweiterungen ergänzten die INTEGER-Arithmetik und Zeichenkettenverarbeitung. Die INTEGER-Variablen und -Felder werden am Namen durch ein nachgestelltes %-Zeichen und Zeichenkettenvariablen und -felder durch ein nachgestelltes $-Zeichen erkannt, z. B. I% und ZP$.

Die symbolischen Namen bestehen nur aus einem Buchstaben oder einem Buchstaben und einer Ziffer. Modernere BASIC-Versionen (siehe auch Beitrag von Keller im Heft 2) lassen an zweiter Stelle auch einen Buchstaben zu.

Für Felder muß vor ihrer Benutzung die Feldgröße vereinbart werden (DIMENSION-Anweisung), z. B.

55 DIM TB%(2Ø%)

definiert ein INTEGER-Feld mit 20 Elementen. Der Index für den Zugriff auf die Elemente wird ab eins gezählt:

TB%(1%) – ist das erste Element,
TB%(I%) – ist das Element *i*,
TB%(2Ø%) – ist das letzte Element.

INTEGER-Konstanten werden durch ein nachgestelltes %-Zeichen gekennzeichnet. Alle anderen Zahlen sind REAL-Konstanten. Zeichenkettenkonstanten werden durch Hochkommas begrenzt, z. B. '/' oder 'OLD'.
In BASIC sind folgende Anweisungen zulässig:

a) Zuordnungsanweisung (Ergibt-Anweisung)

LET variable = ausdruck

Das Schlüsselwort LET kann ausgelassen werden. Die Ausdrücke werden wie in FORTRAN-IV gebildet, z. B. bedeutet L%=L%—1%, daß der INTEGER-Variablen L der

INTEGER-Wert L—1 zugewiesen werden soll.

b) Sprunganweisung

GOTO marke

Als Marke wird auch hier wieder eine Zeilennummer angegeben.

c) Bedingte Anweisung

IF bedingung THEN anweisung

Wenn die Bedingung wahr (true) ist, so wird die Anweisung nach dem Schlüsselwort THEN ausgeführt. Ältere BASIC-Versionen lassen nach THEN nur eine Zeilennummer zu, nach der verzweigt werden soll.
Als Vergleichsoperatoren zur Formulierung der Verzweigungsbedingungen sind zulässig:

=	– gleich	>	– größer als
<>	– ungleich	<=	– kleiner gleich
<	– kleiner als	>=	– größer gleich

Logische Operatoren wie OR, AND und NOT sind in der Regel in BASIC nicht definiert.

d) Schleifenanweisung

FOR variable = anfangswert
TO endwert STEP schrittweite

Die Schleife beginnt mit der FOR-Anweisung. Der Schleifenvariablen werden alle Werte vom Anfangs- bis zum Endwert bei vorgegebener Schrittweite zugewiesen. Die Schrittweite 1 kann ausgelassen werden. Das Schleifenende wird durch die Anweisung

NEXT variable

spezifiziert.
Das Beispielprogramm aus Bild 1 würde in BASIC wie folgt lauten:

```
10 S%=0%
20 FOR I%=1% TO 20%
30 S%=S%+TB%(I%)
40 NEXT I%
```

e) Ein- und Ausgabeanweisungen

INPUT filenummer, variablenliste
LINPUT filenummer, variablenliste
PRINT filenummer, variablenliste

INPUT und PRINT dienen zur Ein- bzw. Ausgabe der spezifizierten Liste von Variablen. Ursprünglich war als E/A-Gerät nur das eigene Terminal zulässig. In den letzten Jahren vorgenommene Spracherweiterungen ergänzten die Fileverarbeitung mit den OPEN- und CLOSE-Anweisungen sowie die Filenummer in den INPUT- und PRINT-Anweisungen. Auch die Anweisung LINPUT ist eine spätere Ergänzung zum Einlesen ganzer Zeilen.

Zum besseren Verständnis kann das Programm mit Kommentarzeilen (REMARK-Anweisung) erläutert werden.

Zum Abschluß sind noch einige kurze Bemerkungen zur Zeichenkettenverarbeitung in BASIC erforderlich. In BASIC ist kein Datentyp BYTE implementiert. Deshalb kann eine Zeichenkette nicht als Feld mit Byteelementen dargestellt werden. Den Zeichenkettenvariablen kann aber eine Zeichenkette mit einer Länge zwischen 0 und 80 Bytes zugewiesen werden. Die Zeichenkette wird von BASIC als ein geschlossenes Objekt behandelt. Zur Verarbeitung einzelner Zeichen bzw. von Teilzeichenketten werden in BASIC einige Standardfunktionen angeboten:

LEN (zvariable),
LEFT$ (zvariable, ivariable),
MID$ (zvariable, ivariable 1, ivariable 2),
RIGHT$ (zvariable, ivariable).

LEN liefert die Länge einer Zeichenkettenvariablen. LEFT$ liefert den linken Teil und RIGHT$ den rechten Teil einer Zeichenkettenvariablen bis bzw. ab der Position, die die INTEGER-

```
ØØ1Ø REM Textverarbeitung
ØØ2Ø REM Fileeroeffnung
ØØ3Ø OPEN 'TEXT.TXT' FOR INPUT AS FILE #1
ØØ4Ø OPEN 'TEXT.DOC' FOR OUTPUT AS FILE #2
ØØ5Ø ZM%=6Ø%
ØØ6Ø W%=LEN(WP¤)
ØØ7Ø Z%=LEN(ZP¤)
ØØ8Ø L%=LEN(EP¤)
1ØØØ REM Einlesen der Druckzeile
1Ø1Ø IF L%>Ø% GO TO 11ØØ
1Ø2Ø IF END #1 THEN  GO TO 1Ø7Ø
1Ø3Ø LINPUT #1,EP¤
1Ø4Ø L%=LEN(EP¤)
1Ø5Ø E%=Ø%
1Ø6Ø GO TO 1ØØØ
1Ø7Ø WP¤='#/'
1Ø8Ø W%=LEN(WP¤)
1Ø9Ø GO TO 124Ø
11ØØ E%=E%+1%
111Ø L%=L%-1%
112Ø IF MID¤(EP¤,E%,1%)<>' ' THEN  GO TO 121Ø
113Ø IF LEFT¤(WP¤,1%)='#' THEN  GO TO 124Ø
114Ø IF W%=Ø% THEN  GO TO 124Ø
115Ø IF Z%>Ø% THEN ZP¤=ZP¤+' '
116Ø ZP¤=ZP¤+WP¤
117Ø WP¤=''
118Ø W%=LEN(WP¤)
119Ø Z%=LEN(ZP¤)
12ØØ GO TO 124Ø
121Ø WP¤=WP¤+MID¤(EP¤,E%,1%)
122Ø W%=W%+1%
123Ø IF L%=Ø% THEN  GO TO 113Ø
124Ø IF LEFT¤(WP¤,1%)<>'#' THEN  GO TO 126Ø
125Ø IF W%=2% THEN  GO TO 2ØØØ
126Ø IF (ZM%-Z%)>=W% THEN  GO TO 1ØØØ
2ØØØ REM Aufbereiten der Druckzeile
2Ø1Ø FOR I%=1% TO LEN(ZP¤)
2Ø2Ø IF MID¤(ZP¤,I%,1%)=' ' THEN  GO TO 2Ø4Ø
2Ø3Ø NEXT I%
2Ø4Ø IF Z%=Ø% GO TO 3ØØØ
2Ø5Ø IF I%=Z% GO TO 3ØØØ
2Ø6Ø IF LEFT¤(WP¤,1%)='#' GO TO 3ØØØ
2Ø7Ø IF Z%=ZM% GO TO 3ØØØ
2Ø8Ø J%=Z%-1%
2Ø9Ø IF Z%=ZM% GO TO 2Ø7Ø
21ØØ IF J%=1% GO TO 2Ø7Ø
211Ø IF MID¤(ZP¤,J%,1%)=' ' GO TO 215Ø
212Ø IF J%=1% GO TO 215Ø
```

```
2130 J%=J%-1%
2140 GO TO 2110
2150 IF J%>1% THEN ZP¤=LEFT¤(ZP¤,J%)+' '+RIGHT¤(ZP¤,J%+1)
2160 Z%=LEN(ZP¤)
2170 IF MID¤(ZP¤,J%,1%)<>' ' GO TO 2090
2180 IF J%=1% GO TO 2090
2190 J%=J%-1%
2200 GO TO 2170
3000 REM Ausdrucken der Druckzeile
3010 IF Z%>0% THEN PRINT #2,ZP¤
3020 ZP¤=''
3030 Z%=LEN(ZP¤)
3040 IF W%<>2% GO TO 4000
3050 IF LEFT¤(WP¤,1%)<>'#' GO TO 4000
3060 IF MID¤(WP¤,2%,1%)='/' GO TO 4000
3070 PRINT #2,ZP¤
3080 IF MID¤(WP¤,2%,1%)='%' THEN PRINT #2,ZP¤
3090 WP¤=''
3100 W%=LEN(WP¤)
4000 IF WP¤<>'#/' THEN  GO TO 1000
4010 REM Schliessen der Files
4020 CLOSE #1
4030 CLOSE #2
4040 END
```

Bild 4. Textverarbeitungsprogramm in BASIC

Variable anzeigt. MID$ liefert den mittleren Teil einer Zeichenkettenvariablen ab der Position ivariable 1 in der Länge ivariable 2.
Über den Operator + können Zeichenketten aneinander gekettet und über die Vergleichsoperatoren verglichen werden. Die Suche des ersten Leerzeichens im Zeilenpuffer ZP muß dann wie folgt programmiert werden:

```
2010 FOR I% = 1% TO LEN(ZP$)
2020 IF MID$(ZP$, I%, 1%) = ' '
     THEN GO TO 2040
2030 NEXT I%
```

Es sei hier auch noch darauf hingewiesen, daß ein BASIC-Interpreter neben den Programmieranweisungen eine stattliche Anzahl von Dialogkommandos (beim Kleincomputer KC 85/1: 20 Kommandos) zur Unterstützung der Programmentwicklung und -testung realisiert. Die wichtigsten Kommandos sind:

a) Eingabe einer Programmzeile
   Eintippen der Programmzeile mit Zeilennummer ohne Angabe eines Schlüsselwortes.

b) Löschen einer bzw. mehrerer Programmzeilen
   DELETE marke [, marke]
   Bei der Angabe von 2 Zeilennummern wird der Bereich von – bis gelöscht.

c) Listen des Programms
   LIST [marke [, marke]]
   Wenn keine Zeilennummern angegeben werden, so wird das ganze Programm angezeigt bzw. gedruckt, ansonsten nur eine einzelne Zeile bzw. ein Bereich von – bis.

d) Starten des Programms

RUN [marke [, marke]]

Wenn keine Zeilennummern angegeben werden, so wird das ganze Programm, mit der niedrigsten Zeilennummer beginnend, abgearbeitet, ansonsten wird eine Zeile bzw. ein Bereich von – bis abgearbeitet.

Diese kurzen Erläuterungen sollen hier als Überblick über BASIC genügen. Die ausführliche Sprachbeschreibung ist der einschlägigen Fachliteratur über die entsprechenden BASIC-Dialekte zu entnehmen. Bild 4 zeigt das entsprechende BASIC-Programm für die Textverarbeitungsaufgabe aus Heft 3.

5. Die Programmiersprache PASCAL

In PASCAL wird ein Programm wie in allen ALGOL-ähnlichen Sprachen als eine beliebig lange Programmzeile aufgefaßt. Da die E/A-Geräte eine begrenzte Zeilenlänge haben, darf ein Programm beliebig in Zeilen kürzerer Länge untergliedert werden. Es existieren grundsätzlich keine Zeilennummern (Hauptkritik an BASIC und teilweise an FORTRAN!). Wie alle ALGOL-ähnlichen Sprachen ist PASCAL eine blockstrukturierte Sprache, bei der beliebige Anweisungen mit den Schlüsselwörtern BEGIN und END (BEGIN- und END-Klammer) zu einer Verbundanweisung zusammengefaßt werden können. Alle Aussagen, die für einfache Anweisungen gültig sind, gelten auch für Verbundanweisungen. Damit ergeben sich hervorragende Möglichkeiten für eine Programmstrukturierung. Im Zusammenhang mit einem umfangreichen Angebot an Steuerflußanweisungen ist die GOTO-Anweisung (die in FORTRAN- und BASIC-Programmen eine der wichtigsten Anweisungen ist!) im wesentlichen überflüssig. Der Programmierer wird »gezwungen«, sein Programm gut strukturiert zu entwerfen und zu programmieren (siehe auch »Wie kann ein Programm systematisch entworfen werden?« im Heft 3).

Ein Programm beginnt mit der PROGRAM-Anweisung, die den Programmnamen und die Datenflußströme (Files) festlegt. Anschließend folgt ein Deklarationsteil, in dem Marken (LABEL), Konstanten (CONST), nutzereigene Datentypen (TYPE) und die Variablen (VAR) definiert werden, die im Programm benutzt werden. Nach dem Deklarationsteil kommt der Programmteil, der mit BEGIN und END geklammert wird. Nach dem abschließenden END steht ein Punkt.

Da das Programm eine beliebig lange Programmzeile darstellt, werden die Deklarationen und Anweisungen untereinander, wie in ALGOL-ähnlichen Sprachen üblich, durch ein Semikolon getrennt.

PASCAL stellt ein umfangreiches Angebot an Datentypen zur Verfügung. Als einfache Datentypen zählen:

INTEGER	– Menge der ganzen Zahlen
CHAR	– Menge der ASCII-Zeichen
BOOLEAN	– die logischen Werte TRUE und FALSE
REAL	– Menge der Gleitkommazahlen.

Zusätzlich kann der Nutzer Teilbereichstypen, z. B.

```
TYPE INDEX = 0..100;
```

und explizite Aufzählungstypen, z. B.

```
TYPE FARBE = (WEISS, ROT,
  BLAU, GRUEN, GELB,
  SCHWARZ);
```

definieren. Eine Variable vom Typ INDEX darf dann nur INTEGER-Werte im Bereich von 0 bis 100 annehmen. Eine Variable vom Typ

FARBE darf nur die aufgezählten Werte, wie WEISS, ROT usw., annehmen.
Außer den einfachen Datentypen stellt PASCAL die strukturierten Datentypen Menge (SET), Satz (RECORD), Feld (ARRAY) und File (FILE) sowie Zeiger (Pointertyp) zur Verfügung.
Ein Puffer kann demnach als ein Feld vom Typ CHAR definiert werden, bei dem sich der Index von Ø bis 1ØØ ändern darf:

```
TYPE PUFFER = ARRAY
  [Ø..1ØØ] OF CHAR;
```

Im Variablendeklarationsteil werden alle Variablen vereinbart, indem ihnen nachgestellt und durch einen Doppelpunkt getrennt ihr Typ spezifiziert wird. Die symbolischen Namen der Variablen können in Abhängigkeit vom Compiler 6 bis 31 Zeichen lang sein.

```
Beispiel: I, J, W, Z: INDEX;
          WP, ZP: PUFFER;
          F1, F2: TEXT;
```

(TEXT ist ein Standardtyp für Textfiles). Der Zugriff auf Feldelemente erfolgt durch Angabe des Index in eckigen Klammern, wobei

ZP [Ø] – das erste Element,
ZP [I] – das Element *i* und
ZP [1ØØ] – das letzte Element ist.

Konstanten können im Programm entsprechend dem Datentyp angegeben werden. Zeichenkettenkonstanten werden durch Hochkommas begrenzt. Zweckmäßigerweise sollten für Konstanten im Konstantendeklarationsteil symbolische Namen eingeführt werden.
In PASCAL sind folgende wichtige Anweisungen definiert:

a) Zuordnungsanweisung (Ergibtanweisung)

```
variable := ausdruck;
```

In Ausdrücken werden Variablen und Konstanten durch folgende Operationen, die nach fallender Priorität geordnet sind, verknüpft:

Priorität 1: Funktionsaufrufe, z.B. SIN(x), ABS(x)
Priorität 2: Negation: NOT
Priorität 3: Multiplikationsoperatoren: *, /, DIV, AND (DIV ist die ganzzahlige Division)
Priorität 4: Additionsoperatoren: +, –, OR
Priorität 5: Vergleichsoperatoren: =, <>, <, <=, >, >=

Zur Änderung der Reihenfolge der Ausdrucksberechnung sind runde Klammern zulässig.

Beispiel: Z := Z + 1;

b) Bedingte Anweisung

```
IF bedingung THEN anweisung;
IF bedingung THEN anweisung
  ELSE anweisung;
```

Entsprechend den Regeln für die Ausdrucksberechnung wird der Ausdruck für die Bedingung berechnet. Ein Wert ungleich Ø (TRUE) bewirkt die Ausführung des THEN-Zweiges, ein Wert gleich Ø (FALSE) die Ausführung des ELSE-Zweiges. Wie bereits erwähnt, kann an Stelle der einfachen Anweisung immer eine Verbundanweisung stehen. Als erweiterte IF-Anweisung ist in PASCAL auch die CASE-Anweisung möglich.

c) Schleifenanweisungen

```
FOR variable := anfangswert
  TO endwert DO anweisung;
FOR variable := anfangswert
  DOWNTO endwert DO anweisung;
WHILE bedingung DO anweisung;
REPEAT anweisung UNTIL bedingung;
```

```
PROGRAM TEXT (F2,F1);
LABEL 1;
CONST
   PMAX = 1ØØ;
   NL = 13;
   ZMAX = 6Ø;
TYPE
   PUFFER = ARRAY[Ø..PMAX] OF CHAR;
   INDEX  = Ø..PMAX;
VAR
   I, J, W, Z: INDEX;
   C: CHAR;
   WP, ZP: PUFFER;
   F2, F1: TEXT;
BEGIN
(* FILEEROEFFNUNG *)
RESET(F1);
REWRITE(F2);
W := Ø; Z := Ø;
REPEAT
BEGIN
  (* EINLESEN DER DRUCKZEILE *)
  REPEAT
  IF EOF(F1) THEN BEGIN WP[Ø]:='#';WP[1]:='/';
                        W:=2; C:=' ' END
  ELSE
    BEGIN READ (F1,C);
    IF C = ' ' THEN
      BEGIN IF (WP[Ø]<>'#') AND (W>Ø) THEN
        BEGIN IF Z>Ø THEN
          BEGIN ZP[Z]:=' '; Z:=Z+1 END;
          FOR I:=Ø TO W-1 DO
            BEGIN ZP[Z]:=WP[I]; Z:=Z+1 END;
          W:=Ø
        END;
      END
    ELSE BEGIN WP[W]:=C; W:=W+1 END;
  END
  UNTIL((W=2) AND (WP[Ø]='#')) OR ((ZMAX-Z)<=W);
  (* AUFBEREITEN DER DRUCKZEILE *)
  FOR I:=Ø TO Z-1 DO IF ZP[I]=' ' THEN GOTO 1;
1:IF (Z>Ø) AND (I<Z) AND (WP[Ø]<>'#') THEN
    BEGIN
      WHILE Z<ZMAX DO
        BEGIN
        J:=Z-1;
        WHILE (Z<ZMAX) AND (J>Ø) DO
          BEGIN
```

```
            WHILE (ZP[J]<>' ') AND (J>Ø) DO J:=J-1;
            IF J>Ø THEN
              BEGIN Z:=Z+1;
                FOR I:=Z-1 DOWNTO J+1 DO ZP[I]:=ZP[I-1];
                WHILE (ZP[J]=' ') AND (J>Ø) DO J:=J-1
              END
          END
        END
    END;
  (*  AUSDRUCKEN DER DRUCKZEILE *)
  IF Z>Ø THEN
    BEGIN
      WRITELN (F2,ZP:Z    );
      Z:=Ø
    END;
  IF (W=2) AND (WP[Ø]='#') THEN
    BEGIN
      WRITELN (F2);
      IF WP[1]='%' THEN WRITELN(F2);
      W := Ø
    END;
END
UNTIL (WP[Ø] = '#') AND (WP[1] = '/');
(* SCHLIESSEN DER FILES *)
END.
```

Bild 5. Textverarbeitungsprogramm in PASCAL

Die FOR-Anweisung für abzählbare Schleifen ist nur für die Schrittweiten +1 (TO) und −1 (DOWNTO) realisiert. Außerdem stehen für die Abweisschleife die WHILE-DO-Anweisung und für die Nichtabweisschleife die REPEAT-UNTIL-Anweisung zur Verfügung.
Das Suchen des ersten Leerzeichens im Zeilenpuffer kann wie folgt vorgenommen werden:

- FOR-Anweisung in Verbindung mit GOTO
  FOR I := Ø TO Z − 1 DO IF ZP [I] = '␣' THEN GOTO 1;
  1:
- WHILE-DO-Anweisung
  I := Ø; WHILE ZP [I] ◇ '␣'
  DO I := I + 1;
- REPEAT-UNTIL-Anweisung
  I := Ø, REPEAT I := I + 1
  UNTIL ZP [I] = '␣';
  (ZP [Ø] wird nicht getestet! Da I vom Typ INDEX ist, darf I nicht auf −1 gesetzt werden.)

Das Beispielprogramm aus Bild 1 würde in PASCAL folgende Notation haben:

S := Ø; FOR I := 1 TO 2Ø
DO S := S + TAB [I];

d) Ein- und Ausgabeanweisungen

Für die Fileeröffnung stehen die RESET-Funktion (Eingabe) und REWRITE-Funktion (Ausgabe) zur Verfügung. Der Fileabschluß wird automatisch bei Programmende vorgenommen oder durch RESET er-

zwungen. Filenamen können im Programm nicht spezifiziert werden (keine OPEN-Anweisung!).
Für die Eingabe stehen die Anweisungen

READ (filevariable, variablenliste)
READLN (filevariable)
GET (filevariable)

und für die Ausgabe die Anweisungen

WRITE (filevariable, variablenliste)
WRITELN (filevariable)
PUT (filevariable)

zur Verfügung. Die Ausführung der E/A-Anweisungen ist von der Definition des Files und der Variablenliste abhängig. Wird bei TEXT-Files eine CHAR-Variable spezifiziert, so erfolgt eine zeichenweise Eingabe. Die Anweisungen READLN und WRITELN dienen zum Positionieren auf die nächste Zeile bzw. zum Ausgeben eines Zeilenvorschubes (New Line).

Zur Erhöhung der Lesbarkeit der Programme können zusätzliche Kommentare aufgenommen werden, die durch (* eingeleitet und *) abgeschlossen werden.

Die weiteren Möglichkeiten der Programmiersprache PASCAL sind der einschlägigen Fachliteratur sowie den für die einzelnen Compiler spezifischen Programmieranleitungen zu entnehmen, die in der Regel gewisse Einschränkungen aufweisen oder Erweiterungen zulassen.

Bild 5 zeigt das entsprechende PASCAL-Programm für die Textverarbeitungsaufgabe aus Heft 3.

6. Die Programmiersprache C

Die Programmiersprache C ist mit ALGOL-68 in gewissem Grade verwandt. Daraus ergeben sich zwangsläufig Parallelen zu PASCAL. Allerdings entstand C als Systemprogrammiersprache für moderne Kleinrechner vom Typ PDP-11, die als einer der Wegbereiter der Mikroprozessortechnik gelten. Dieser Sachverhalt erklärt, warum die Programmiersprache C für die Mikrorechnerprogrammierung besonders geeignet und auch folglich auf Mikrorechnern international weit verbreitet ist.
Ein C-Programm ist wieder eine beliebig lange Programmzeile, die beliebig in Zeilen kürzerer Länge untergliedert werden darf. C ist ebenfalls eine blockstrukturierte Sprache, wobei allerdings zur Erhöhung der Kompaktheit des Programms die BEGIN- und END-Klammern durch geschweifte Klammern im Schriftbild ersetzt werden. Analog zu PASCAL ist die Verbundanweisung in C ebenfalls ein wichtiges Strukturelement. C verfügt genauso über alle wichtigen Steuerflußanweisungen, wobei sogar die meisten Anweisungen (switch/case u.a.) entweder gegenüber PASCAL erweitert oder verallgemeinert (z.B. for) sind. Für eine verbesserte Schleifensteuerung gibt es zusätzlich eine break-Anweisung (vorzeitiger Schleifenabbruch) und eine continue-Anweisung (vorzeitiger Übergang zum nächsten Schleifendurchlauf).
Eine wichtige Eigenschaft von C ist die konsequente Ausschöpfung des vollen ASCII-Zeichensatzes (96 Zeichen). So werden nicht nur alle Sonderzeichen benutzt, sondern es werden auch die lateinischen Groß- und Kleinbuchstaben zugelassen, die bei der Bildung symbolischer Namen unterschieden werden, z.B. sind AØ und aØ unterschiedliche Symbole. Die Schlüsselwörter werden in C prinzipiell klein geschrieben, da wir die Kleinschreibung von unseren natürlichen Sprachen gewohnt sind. Gleichzeitig können durch

die Klein- und Großschreibung mehr semantische Unterschiede sichtbar gemacht werden, wie z.B. Großbuchstaben für Konstantensymbole, Kleinbuchstaben für Variablensymbole usw.
In C ist jedes Programm grundsätzlich eine Funktion. Funktionen sind etwa den Begriffen SUBROUTINE bei FORTRAN, PROCEDURE bei PASCAL oder Unterprogramm in der Assemblersprache äquivalent. (Diese Möglichkeiten von FORTRAN und PASCAL blieben hier unberücksichtigt.) Jedes Programm wird daher in C als Funktion definiert und ist daher als ein Baustein von anderen Programmen her aufrufbar. Eine Funktion hat folgenden prinzipiellen Aufbau:

```
typ funktionsname (liste der formalen parameter)
deklaration der formalen parameter;
{
deklarationen;
anweisungen;
}
```

wobei typ den Datentyp des Funktionswertes festlegt. Die Funktion, die vom Betriebssystem gestartet wird, trägt den Namen main, ihre Parameterliste ist in der Regel leer.
Im Deklarationsteil werden für die Variablen der Datentyp und die Speicherklasse vereinbart. Folgende einfache Datentypen sind zulässig:

```
int, long int, unsigned int, float,
long float.
```

Darüber hinaus gibt es die höheren Datentypen Felder (arrays), Funktionen (functions), Zeiger (pointers), Strukturen (structures) und Vereinigungen (unions). Mit der typedef-Anweisung können in C neue Datentypen auf der Basis der einfachen oder höheren Datentypen vereinbart werden. Teilbereichs- und explizite Aufzählungstypen gibt es in C nicht.

Als Speicherklassen sind zulässig: auto, static, register und extern. Standard ist auto, d.h., die Variablen werden beim Eintritt in den BEGIN-END-Block definiert und im Speicher reserviert sowie beim Austritt wieder vergessen und aus dem Speicher gestrichen.
Die symbolischen Namen der Variablen können beliebig lang sein, jedoch sind nur die ersten 8 Zeichen signifikant.

Beispiel für Deklarationen:

```
int i, j, w, z; char c, wp [5Ø],
zp [1ØØ];
```

Die Variablen wp und zp sind als Feld vom Typ char vereinbart, wobei in Klammern die Dimension angegeben wird. Der Feldindex wird bei C, wie in der Systemprogrammierung üblich, immer ab Null beginnend gezählt, wobei

```
zp [Ø]  – das erste Element,
zp [i]  – das Element i + 1 und
zp [99] – das letzte Element ist.
```

Konstanten können im Programm entsprechend dem Datentyp angegeben werden. Zeichenkonstanten werden durch Hochkommas und Zeichenkettenkonstanten durch Anführungszeichen begrenzt, z.B. '/', "Text" usw. Durch die #define Anweisung (Bestandteil der Makrosprache von C) können für Konstanten auch Symbole definiert werden, z.B.

```
# define ZMAX 6Ø
```

In C sind folgende wichtige Anweisungen definiert:

a) Zuordnungsanweisung
(Ergibtanweisung)

```
variable = ausdruck;
```

In Ausdrücken können auf Variablen (bzw. Konstanten) folgende einstellige Operatoren angewendet werden:

```
#include <stdio.h>
main()
{
int i, j , w , z , zmax ;  char c , zp[1ØØ] , wp[5Ø];
FILE  *fin, *fout; /* Definition File-Deskriptor-Zeiger */
zmax = 6Ø;

/* Eroeffnen der Files */
if ((fin=fopen("TEXT.TXT","r"))==NULL) error("File nicht vorhanden");
if ((fout=fopen("LP1:","w"))==NULL) error("Drucker nicht bereit");
w=z=Ø; /* Loeschen des Wort- und Zeilenpuffers */
do {

/* Einlesen der Druckzeile */
   do { c=getc (fin);
      switch (c) {
         case EOF:   wp[Ø]='#'; wp[1]='/'; w=2; c=' '; break;
         case '\n':
         case ' ':    if ((wp[Ø]!='#') && (w>Ø)) {
                         if (z>Ø) zp[z++]=' ';
                         for (i=Ø; i<w; ) zp[z++]=wp[i++]; w=Ø;
                      } break;
         default:     wp[w++]=c; break;
      }
   } while (((w!=2) || (wp[Ø]!='#')) && ((zmax-z)>w));

/* Aufbereiten der Druckzeile */
   for (i=Ø; (i<z) && (zp[i]!=' '); i++);
   if ((z>Ø) && (i<z) && (wp[Ø]!='#')) {
      while (z<zmax) { j=z-1;
         while ((z<zmax) && (j>Ø)) {
            while ((zp[j]!=' ') && (j>Ø)) j--;
            if (j>Ø) {
               for (i=z++; i>j; i--) zp[i]=zp[i-1];
               while ((zp[j]==' ') && (j>Ø)) j--;
            }
         }
      }
   }
/* Ausdrucken der Druckzeile */
   if (z>Ø) {
      for (i=Ø; i<z; ) putc (zp[i++], fout);
      putc ('\n', fout); z = Ø;
   }
   if ((w==2) && (wp[Ø] =='#')) {
      putc ('\n', fout);
      if (wp[1] == '%') putc ('\n', fout); w = Ø;
```

```
} while ((wp[Ø]!='#') || (wp[1]!='\n'));

/* Schliessen der Files */
fclose (fin); fclose (fout) ;
```

Bild 6. Textverarbeitungsprogramm in C

— – Zweierkomplement
~ – Einerkomplement
! – logische Negation
+ – positives Vorzeichen
++ – Inkrementoperator
—— – Dekrementoperator

Die Inkrement- und Dekrementoperatoren führen je nach Stellung in bezug auf die Variable entweder vor oder nach der Operation zur Erhöhung oder Erniedrigung ihres Wertes, z. B.

s = s + tab [i + +]; Index *i* wird nach der Operation erhöht,

s = s + tab [— — i]; Index *i* wird vor der Operation erniedrigt.

Durch zweistellige Operatoren können Variablen und Konstanten in Ausdrücken verknüpft werden. Folgende Operatoren, nach fallender Priorität geordnet, sind möglich:

- Funktionsaufrufe
- einstellige Operatoren
- Multiplikationsoperatoren: *, /, % (% – Rest der ganzzahligen Division)
- Additionsoperatoren: +, —
- Verschiebeoperatoren: >>, <<
- Verhältnisoperatoren: <, <=, >, =>
- Gleichheitsoperatoren: ==, != (ungleich)
- Bitorientierte logische Operatoren: &, ^, |
- Logisches UND: &&
- Logisches ODER: ||
- Zuweisungsoperatoren: =, +=, —=, *=, /=, %=, <<=, >>=, &=, ^=, |=

Durch die zusätzlichen Zuweisungsoperatoren lassen sich insbesondere die häufig vorkommenden Operationen mit einer Variablen vereinfacht schreiben. Die oben spezifizierte Anweisung zur Summenbildung verkürzt sich wie folgt:

s + = tab [i + +];

Sind in einer Anweisung mehrere Zuweisungsoperatoren enthalten, so wird die Anweisung von rechts nach links abgearbeitet, z. B.

i = j = w = z = Ø;

Zur Änderung der Reihenfolge der Ausdrucksberechnung sind runde Klammern zulässig.

b) Bedingte Anweisung

```
if (bedingung) anweisung;
if (bedingung) anweisung;
   else  anweisung;
```

Die if-Anweisung entspricht der PASCAL-IF-Anweisung. Es ist nur zu beachten, daß das Schlüsselwort THEN nicht realisiert ist, dafür muß aber die Bedingung in runde Klammern eingeschlossen werden. Die Anweisung des THEN-Zweiges muß immer mit einem Semikolon abgeschlossen werden. Als erweiterte if-Anweisung ist in C die switch-case-Anweisung realisiert.

c) Schleifenanweisungen

```
for (ausdruck 1; ausdruck 2;
     ausdruck 3) anweisung;
while (bedingung) anweisung;
do anweisung while (bedingung);
```

Die for-Anweisung für abzählbare Schleifen ist gegenüber anderen Sprachen eine stark verallgemeinerte Schleife. Ausdruck 1 wird einmalig zu Beginn der Schleife ausgeführt (Festlegung des Anfangswertes). Ausdruck 2 legt die Wiederholungsbedingung fest. Ausdruck 3 organisiert die Schrittweitensteuerung. (Der Begriff des Ausdrucks ist in C gegenüber anderen Sprachen erweitert, da auch Zuweisungsoperatoren zulässig sind!) Das Beispielprogramm aus Bild 1 kann durch folgende Anweisung beschrieben werden:

```
for (i = s = 0; i < 20; i++)
    s += tab [i];
```

oder

```
for (s = i = 0; i < 20;)
    s += tab [i++];
```

Das Suchen des ersten Leerzeichens im Zeilenpuffer kann auch wie folgt geschrieben werden:

```
for (i = 0;
(i < z) && (zp [i] != 'b'); i++);
```

Die runden Klammern im Ausdruck 2 sind redundant, da das logische UND && die niedrigste Priorität im Ausdruck hat. Aus Gründen der Verbesserung der Lesbarkeit sind sie aber zweckmäßig.
Die for-Anweisung ist im allgemeinen folgender while-Schleife äquivalent:

```
ausdruck 1;
while (ausdruck 2)
  {anweisung; ausdruck 3;}
```

d) Ein- und Ausgabeanweisungen
In C stehen umfangreiche Funktionen zur Fileverarbeitung für sequentielle und Direktzugriffsdatenträger zur Verfügung. Für jedes File muß in C ein Filedeskriptorblock fdb definiert werden:

```
FILE *fdb;
```

Ein File kann mit fopen eröffnet und mit fclose abgeschlossen werden:

```
fdb = fopen (filename, attribut);
fclose (fdb);
```

Die wichtigsten Verarbeitungsfunktionen sind:

- zeichenweise Verarbeitung
  ```
  c = getc (fdb);
  putc (c, fdb);
  ```
- satzweise Verarbeitung
  ```
  fgets (puffer, maxlänge, fdb);
  fputs (puffer, fdb);
  ```

Außerdem gibt es Anweisungen für die formatgesteuerte Ein- und Ausgabe: fscanf, scanf, fprintf und printf.

Zur Erhöhung der Lesbarkeit der Programme können zusätzliche Kommentare aufgenommen werden, die durch /* eingeleitet und */ abgeschlossen werden.
Die umfangreichen weiteren Möglichkeiten der Programmiersprache C sind der Standardliteratur zu entnehmen. Bild 6 zeigt abschließend das entsprechende C-Programm für die Textverarbeitungsaufgabe aus Heft 3.

7. Die Programmiersprache FORTH

Die Programmiersprache FORTH ist eine leistungsfähige Dialogsprache, die syntaktisch und semantisch im Dialog erweiterbar ist. Es können neue Operanden und Operationen sowie neue Klassen von Operanden und Operationen definiert werden.
Es gibt zwei grundsätzliche Arbeitsweisen, die Interpretation und die interaktive Compilierung bzw. Assemblierung. Daraus folgt, daß die Anwendung der Programmiersprache

FORTH an ein spezielles FORTH-Betriebssystem gebunden ist, welches den Interpreter, Dialog-Compiler, Dialog-Assembler und Dialog-Editor enthält. Compiler, Assembler und Editor sind zum großen Teil selbst in FORTH geschrieben.
Das Erlernen der Programmiersprache FORTH setzt, sofern nicht die Kenntnis einer verwandten Programmiersprache wie LISP vorhanden ist, ein völliges Umdenken voraus. In FORTH gibt es keine Anweisungen, sondern nur FORTH-Wörter, die in einem Wörterbuch gespeichert sind. Die elementaren FORTH-Wörter werden über Assemblerroutinen realisiert. Das Wörterbuch kann durch den Dialog-Compiler mit neuen FORTH-Wörtern erweitert werden, die auf der Grundlage vorhandener FORTH-Wörter definiert werden. Der Dialog-Assembler gestattet die Definition von FORTH-Wörtern auf der Grundlage von bereits bekannten FORTH-Wörtern und Assembleranweisungen, die beliebig gemischt werden können.
Der Dialog-Editor gestattet die Eingabe und Korrektur von FORTH-Programmen auf einem Hintergrundspeicher (Diskette, Platte usw.), wobei unter einem FORTH-Programm eine Menge von Definitionen für neue FORTH-Wörter verstanden wird. Die FORTH-Programme können mit dem Dialog-Compiler geladen werden, so daß ein effektives Mittel zur Erweiterung des aktuellen Wörterbuches vorhanden ist. Der Editor organisiert die Verwaltung der Programme in Form von 1-KByte-Blöcken, die als 16 Zeilen zu je 64 Zeichen interpretiert werden. Das entspricht der nutzbaren Bildschirmfläche der meisten Mikrorechner. Deshalb werden die 1-KByte-Blöcke auch als SCREENs bezeichnet. Im Prinzip stellt aber ein Screen eine lange Programmzeile mit 1024 Zeichen dar.

FORTH-Programme müssen streng strukturiert aufgebaut werden, d.h., das Programm muß nach der Methode der schrittweisen Verfeinerung von »Oben nach Unten« (Top-Down) entworfen werden. Die Programmierung hat aber in umgekehrter Richtung von »Unten nach Oben« (Bottom-Up) zu erfolgen, da neue FORTH-Wörter nur auf der Grundlage bekannter FORTH-Wörter definiert werden dürfen. Daraus ergibt sich, daß zuerst die Variablen deklariert werden müssen und dann erst das FORTH-Wort für das Programm definiert werden darf. Da große FORTH-Programme schlecht lesbar sind, unübersichtlich wirken und sich nur mit großem Aufwand testen lassen, sollte man wirklich nach der Bottom-Up-Methode zuerst FORTH-Wörter für einfache Operationen definieren, die dann wieder zu komplexeren FORTH-Wörtern zusammengefaßt werden, bis schließlich ein FORTH-Wort das gesamte Programm repräsentiert.
Das FORTH-Wort

VOCABULARY name

legt den Namen des neuen Wörterbuches fest.
Das FORTH-Wort

wert VARIABLE name

definiert die Variable name und weist ihr den Anfangswert wert zu. Alle Variablen sind 16-Bit-Worte. Ist die Variable ein Feld, so ist nach der Variablen ein zusätzlicher Speicherbereich entsprechend der Feldgröße mit dem FORTH-Wort ALLOT zu reservieren, z.B.

Ø VARIABLE ZP 98 ALLOT

definiert ein Feld ZP mit 100 Byte.
Mit dem FORTH-Wort CONSTANT können entsprechend Konstanten vereinbart werden, z.B.

6Ø CONSTANT ZMAX

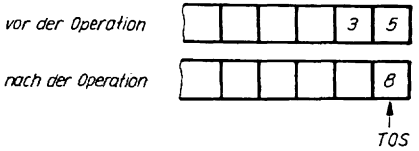


Bild 7. Prinzip der Stackoperationen in FORTH

definiert das Wort ZMAX als Konstante mit dem Wert 60.
FORTH-Wörter für neue Operationen werden über die Doppelpunktdefinition vereinbart:

: name definition;

wobei name das neue FORTH-Wort festlegt und definition die Erklärung für das neue FORTH-Wort auf der Grundlage bekannter FORTH-Wörter gibt, z.B.

: TEXT FOPEN
TEXTAUFBEREITUNG FCLOSE;

Hinweis: In FORTH wird als universelles Trennzeichen das Leerzeichen verwendet; es muß zwischen allen syntaktischen Einheiten (FORTH-Wörtern) stehen.
Betrachten wir im weiteren die wichtigsten Operationen, die im FIG-FORTH definiert und somit Bestandteil des Standard-Wörterbuches sind. Zum Verständnis der Arbeitsweise ist es wichtig zu wissen, daß FORTH von einem Stackcomputer ausgeht, d.h., alle Operationen werden nur mit einem Stackspeicher ausgeführt.
Bild 7 erklärt das Prinzip am Beispiel der Operation $3 + 5 = 8$. Bei einem Stackcomputer sind die beiden Operanden zuerst in den Stack zu bringen: 3 und 5. Der zweite Operand steht an der Stackspitze (TOS – Top Of Stack). Nun wird die Addition ausgeführt. Dabei werden die beiden Operanden gelöscht und das Ergebnis anschließend in den Stack eingetragen.

Bei der Programmierung sind die Formeln entsprechend der Arbeitsweise des Stackcomputers aufzubereiten. Man nennt diese Schreibweise inverse polnische Notation, da der Operator den Operanden immer nachgestellt wird, z.B. 3 5 +.

a) Arithmetische und logische Operationen
- \+ Addition
- — Subtraktion
- * Multiplikation
- / Division
- MOD Rest der Division
- AND Logisches UND
- OR Logisches ODER
- XOR Logisches exclusives ODER

b) Speicheroperationen
- @ Lesen eines Integerwertes
- ! Schreiben eines Integerwertes
- C@ Lesen eines Bytes
- C! Schreiben eines Bytes

Beispiel:

L := L — 1 in FORTH:

L @ 1 — L !

c) Stackoperationen
- DUP Oberste Zahl im Stack wird noch einmal eingetragen.
- DROP Löschen der obersten Zahl im Stack
- SWAP Vertauschen der beiden obersten Zahlen

d) Steuerworte
- DO ... LOOP
  Abzählbare Schleife
- BEGIN ... UNTIL
  Nichtabweisschleife
- BEGIN ... WHILE ... REPEAT
  Abweisschleife
- IF ... THEN
  Unvollständige Alternative
- IF ... ELSE ... THEN
  Vollständige Alternative

```
******** ZKI-FIG-FORTH REL 2.1 ********

SCR #  24Ø
  Ø ( TEXTVERARBEITUNG I )
  1 VOCABULARY TEXT
  2 TEXT DEFINITIONS HEX
  3
  4 ( VARIABLE, KONSTANTEN )
  5 Ø VARIABLE ZP 98 ALLOT ( PUFFER 1ØØ BYTE )
  6 Ø VARIABLE WP 48 ALLOT ( DTO )
  7
  8 Ø VARIABLE Z ( ZEIGER NACH ZP )
  9 Ø VARIABLE W ( ZEIGER NACH WP )
 1Ø Ø VARIABLE C ( ZEICHENPUFFER FUER KEY )
 11 Ø VARIABLE J ( HILFSVARIABLE )
 12 23 CONSTANT '#   2F CONSTANT '/   25 CONSTANT '%
 13 DECIMAL   6Ø CONSTANT ZMAX   13 CONSTANT NL
 14 8 LOAD ( CASE-ANWEISUNG )
 15 —>



SCR #  241
  Ø ( TEXTVERARBEITUNG II )
  1 ( EINLESEN DER DRUCKZEILE )
  2 : ZP_VOLL    W @ 2 = WP C@ '# = AND ZMAX Z @ -
  3              W @ < OR  ;
  4 ( ZEICHEN IN WORTPUFFER EINTRAGEN )
  5 : ZEICHEN C C@ WP W @ + C! 1 W +!  ;
  6 ( WORT AUS WORTPUFFER IN ZEILENPUFFER KOPIEREN )
  7 : WORTCOPY
  8       Z @ Ø >
  9      IF
 1Ø        BL ZP Z @ + C! 1 Z +!
 11      THEN    W @ Ø
 12      DO
 13         WP I + C@ ZP Z @ + C! 1 Z +!
 14      LOOP    Ø W !  ;
 15 —>



SCR #  242
  Ø ( TEXTVERARBEITUNG III )
  1 ( KOP. NUR, WENN WORT IM WORTPUFFER UND KEIN STEUERZEICHEN )
  2 : WORT WP C@ '# = Ø= W @ Ø > AND
  3       IF
  4         WORTCOPY
```

```
 5       THEN ;
 6 ( EINLESEN EINES ZEICHENS )
 7 : GETC RDR C C! ;
 8 ( FORTH-WORT ZUM EINLESEN EINER DRUCKZEILE )
 9 : READ_ZEILE
10     BEGIN GETC C C@ CASE
11       BL OF WORT ENDOF
12       NL OF WORT ENDOF
13       ZEICHEN ENDCASE
14       ZP_VOLL
15      UNTIL ; -->

SCR #  243
 0 ( TEXTVERARBEITUNG IV )
 1 : FLATTERRANDAUSGLEICH
 2     BEGIN Z @ ZMAX <
 3     WHILE Z @ 1 - J !
 4        BEGIN Z @ ZMAX < J @ 0 > AND
 5        WHILE
 6           BEGIN ZP J @ + C@ BL = 0= J @ 0 > AND
 7           WHILE -1 J +!
 8           REPEAT  J @ 0 >
 9             IF 1 Z +! J @  Z @ 1 -
10               DO ZP I + DUP 1 - C@ SWAP  C! -1 +LOOP
11             THEN
12          BEGIN ZP J @ + C@ BL = J @ 0 > AND
13          WHILE -1 J +! REPEAT
14       REPEAT
15    REPEAT ; -->

SCR #  244
 0 ( TEXTVERARBEITUNG V )
 1 ( AUFBEREITUNG DER DRUCKZEILE )
 2 ( SUCHEN EINES LEERZEICHENS )
 3 : BL_SUCHEN    Z @ 0
 4    DO
 5      ZP I + C@ BL = IF I LEAVE THEN
 6    LOOP ;
 7 ( AUFBEREITUNG DER DRUCKZEILE MIT FLATTERRANDAUSGLEICH )
 8 : FLATTERRAND Z @ 0 > SWAP Z @ < AND WP C@ '# = 0= AND
 9       IF  FLATTERRANDAUSGLEICH THEN ;
10 ( FORTH-WORT FUER DIE AUFBEREITUNG EINER DRUCKZEILE )
11 : RAND BL_SUCHEN FLATTERRAND ;
12
13
14
15 -->
```

```
SCR #  245
  Ø ( TEXTVERARBEITUNG VI )
  1 ( AUSGABE EINER ZEILE )
  2 : ZEILE Z @ Ø
  3     DO
  4       ZP I + C@ EMIT
  5     LOOP CR Ø Z ! ;
  6 ( VORSCHUBSTEUERUNG FUER DEN DRUCKER )
  7 : VORSCHUB  CR WP 1+ C@ '% =
  8      IF CR THEN
  9      Ø W ! ;
 1Ø ( FORTH-WORT ZUR AUSGABE DER KOMPLETTEN DRUCKZEILE )
 11 : OUTPUT Z @ Ø > IF ZEILE THEN
 12      W @ 2 = WP C@ '# = AND
 13      IF VORSCHUB THEN ;
 14
 15 -->

SCR #  246
  Ø ( TEXTVERARBEITUNG VII )
  1 ( FILEEINGABE UEBER LOCHBAND )
  2 : FOPEN ATT-PR: ; ( ATTACH PRØ: )
  3 : FCLOSE DTA-PR: ; ( DETTACH PRØ: )
  4 ( TEST AUF ENDE DER FILEVERARBEITUNG )
  5 : FILEENDE WP C@ '# = WP 1+ C@ '/ = AND ;
  6 ( HAUPTPROGRAMM )
  7 : TEXT1
  8      FOPEN
  9      BEGIN
 1Ø         READ_ZEILE RAND OUTPUT
 11         FILEENDE
 12      UNTIL
 13       FCLOSE ;
 14
 15 ;S
```

Bild 8. Textverarbeitungsprogramm in FORTH

I Zählparameter in Schleifen
Das Beispielprogramm aus Bild 1 könnte in FORTH wie folgt programmiert werden:
Programmieren wir zuerst die Teilsumme S := S + TAB [I]:

```
S @ TAB  I + @ + S !
```

Nachdem dieses Teilprogramm im Dialog ausgetestet ist, wird ein neues FORTH-Wort definiert:

```
: TEILSUMME  S @
  TAB  I + @ + S ! ;
```

Nun wird die Schleife programmiert. DO verlangt im Stack den Endwert + 1 und den Anfangswert:

```
2Ø Ø DO TEILSUMME LOOP
```

Nach der Testung wird definiert:

```
: SUMME  2Ø Ø DO
  TEILSUMME  LOOP ;
```

Ähnlich verhält es sich mit der »bedingten Anweisung«: Vor dem Wort IF muß der Ausdruck für die Bedingung berechnet werden. Ist der Wert ungleich Ø (true), so werden die Worte zwischen IF und THEN ausgeführt. THEN schließt die bedingte Anweisung ab.

e) Ein- und Ausgabeoperationen
Es stehen eine Reihe von Worten für die verschiedenen Formen der Terminalein- und -ausgabe zur Verfügung. Eine Fileverarbeitung ist im FIG-FORTH nicht definiert. Deshalb sind in den verschiedenen FORTH-Systemen spezielle Wörterbücher für erweiterte E/A-Operationen vereinbart, die bei Bedarf mit dem Wort LOAD von einem Screen geladen werden.

Zur Erhöhung der Lesbarkeit der FORTH-Programme sollte man Kommentare ergänzen. Kommentare werden durch (eingeleitet und) abgeschlossen.
Diese kurzen Erläuterungen sollen hier als kleine Einführung in FORTH genügen. Die ausführliche Sprachbeschreibung ist der entsprechenden Fachliteratur zu entnehmen. (Informationen darüber können beim Autor eingeholt werden.)
Bild 8 zeigt das FORTH-Programm für die Textverarbeitungsaufgabe aus Heft 3. Das Wörterbuch heißt TEXT. Das Programm wird mit TEXT 1 gestartet. Das Programm wurde auf der Grundlage des entsprechenden C-Programms unter Verwendung der elementaren Byteoperationen entwickelt.

8. Zusammenfassung

Ausgehend von der Darstellung der historischen Entwicklung, wurden fünf für die Mikrorechnerprogrammierung wichtige Sprachen etwas ausführlicher betrachtet. Das Beispiel zur Textverarbeitung aus Heft 3 wurde in allen Sprachen implementiert.
Aus den Darlegungen ist zu ersehen, daß es die ideale Programmiersprache noch nicht gibt und in der nächsten Zukunft auch nicht geben wird. Durch die Vielfalt der objektiven und subjektiven Bedingungen hat jede Programmiersprache ihre Existenzberechtigung, und es lohnt sich, sich mit jeder Programmiersprache näher zu beschäftigen. Dieser Beitrag soll dabei dem Amateur auch eine kleine Hilfe geben, »seine« Programmiersprache auszuwählen.

Autor:
Doz. Dr. sc. techn. Thomas Horn

Hochschuldozent
an der Sektion Informationsverarbeitung
der Ingenieurhochschule Dresden

Mikrorechnergesteuerte Analog-Digital-Umsetzung mit dem Polycomputer 880

Einleitung

Mit dem Polycomputer 880 ist ein vollständiger Mikrorechner mit der CPU U 880 mit 2 KByte ROM-Kapazität (U 505) für das Betriebssystem, 1KByte RAM-Kapazität (U 202), mit zwei Peripheriebausteinen PIO U 855 und der CTC U 857 auf dem Markt, der sich als Mikrorechner-Lernsystem für eigene Programmentwicklungen sowie für einfache Prozeßsteuerungsaufgaben sinnvoll einsetzen läßt.

In der Literatur sind erste Erfahrungen [1], [2], [3] über die Möglichkeiten des Einsatzes dieses ersten in der DDR industriell gefertigten Mikrorechner-Lernsystems angegeben. Während in [4] Zusatzgeräte für den Polycomputer 880 vorgestellt sind, wurde im Heft 3 dieser Broschürenreihe [5] dieses Gerät hardwaremäßig eingehend beschrieben. Im folgenden soll die Möglichkeit der Anwendung des Polycomputers 880 für eine Analog-Digital-Umsetzung und deren grafische Darstellung auf einem Sichtgerät für Demonstrationszwecke der Funktionsweise einfacher ADUs unter Verwendung einer einfachen Programmierung der Umsetzerfunktion dargestellt werden. Im einzelnen werden die Funktionen des Stufen- oder Folge-ADUs und des ADUs nach dem Prinzip der sukzessiven Approximation vorgestellt.

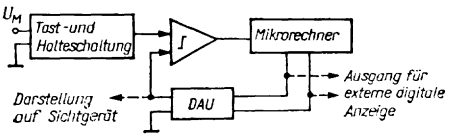


Bild 1. Blockschaltbild der Hardwarebaugruppen zur Realisierung der ADU-Funktionen

In Bild 1 ist das prinzipielle Blockschaltbild des Aufbaus für die Realisierung der ADU-Funktionsweisen gegeben.

Die Wirkungsweise dieses Aufbaus ist folgende: Die zu messende und darzustellende analoge Spannung U_M wird über eine Tast- und Halteschaltung auf den Eingang eines Komparators gegeben und mit einem vom Mikrorechner bereitgestellten und über die PIO ausgegebenen 8-bit-Datenwort, das mittels eines DAU zu einer Vergleichsspannung U_V gewandelt wird, verglichen. Im Ergebnis des Spannungsvergleichs steht am Ausgang des Komparators die Information $U_M \geqq U_V$ durch H bzw. $U_M \leqq U_V$ durch L als Eingangsinformation für den Mikrorechner zur Verfügung, die als Auswertekriterium für die softwaremäßige Realisierung der entsprechenden ADU-Algorithmen Verwendung findet. Nach später dargelegten Algorithmen liefert der Mikrorechner ein neues Datenwort,

das über den DAU erneut in eine äquivalente Vergleichsspannung gewandelt und wiederum an den Komparatoreingang gelegt wird. Dieser Zyklus wird softwaremäßig gesteuert und so lange wiederholt, bis die Vergleichsspannung U_V vom DAU innerhalb der kleinsten Digitalisierungsstufe mit der Meßspannung U_M übereinstimmt.
Die einzelnen Schritte bzw. Stufen der Umsetzung können durch geeignete Wahl der Umsetzungsgeschwindigkeit sowohl digital auf der Anzeige des Mikrorechners als auch als grafische Darstellung auf dem Sichtgerät verfolgt werden.

Funktionsweise der AD-Wandlung nach dem Prinzip der Stufenumsetzung bzw. nach dem Verfahren der sukzessiven Approximation

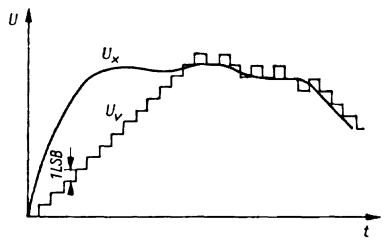


Bild 2. Prinzip des Stufen- oder Folge-ADU

Bei der AD-Wandlung nach dem Stufen- oder Folgeverfahren (Bild 2) wird eine angelegte Spannung mit einer stufenweise (schrittweise) veränderlichen Vergleichsspannung U_V verglichen, wobei die Spannungsstufung durch den Spannungsbereich, der umgesetzt werden soll, und durch die Anzahl der Bits des Datenworts vorgegeben ist. Die Anstiegsgeschwindigkeit bzw. Umsetzgeschwindigkeit ist dabei durch die Größe der kleinsten Spannungsstufe und durch die Taktfrequenz bestimmt.

Nachdem die stufenförmig ansteigende Vergleichsspannung U_V die Meßspannung U_M erreicht hat, wechselt bei konstanter Meßspannung das Signal am Komparatorausgang ständig zwischen H und L, d.h., es wird abwechselnd eine Spannungsstufe zu- bzw. abgeschaltet. Ändert sich zeitlich die Meßspannung, so »folgt« der Folge-ADU unterhalb der hardware- bzw. softwarebedingten Grenzfrequenz der angelegten Spannung. Um zu verhindern, daß bei konstant bleibender Eingangsspannung die Vergleichsspannung ständig um die kleinste Spannungsstufung schwankt, kann einerseits der Komparator mit einer Schalthysterese von $\pm$ 1/2 Spannungsstufe ΔU versehen werden bzw. andererseits ein DA-Wandler verwendet werden, der eine um 1 bit höhere Auflösung besitzt, als digitale Ausgänge herausgeführt sind.

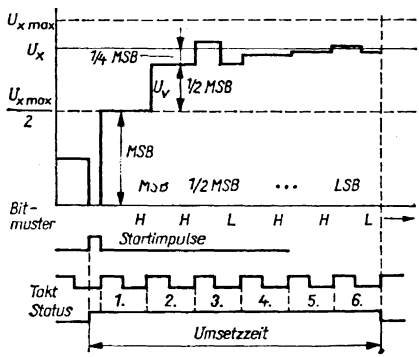


Bild 3. Prinzip der sukzessiven Approximation

In Bild 3 ist das Prinzip der AD-Wandlung nach der sukzessiven Approximation verdeutlicht. Der Wandler besteht wieder – wie in Bild 1 gezeigt – aus einem Komparator, dem internen DA-Wandler und einer im Mikrorechner softwaremäßig realisierten Approximationslogik. Die vorgeschaltete Tast-

und Halteschaltung ist nur erforderlich, wenn sich U_M während der Umsetzzeit um $\geq 1/2$ LSB ändert (LSB = least significant bit = kleinste unterscheidbare Amplitudenstufe).
Mit dem Startimpuls wird ein MSB (MSB = most significant bit = Stufe mit der höchsten Wertigkeit = $U_{max/2}$) gesetzt und damit das Vergleichssignal auf $U_{max/2}$ eingestellt. Der Komparator vergleicht U_M und U_V. Da für das MSB, wie in Bild 3 dargestellt, $U_V < U_M$ ist, liegt der Ausgang auf H, und mit der nächsten H/L-Flanke des Taktes wird das MSB »verriegelt«, d.h., es bleibt gesetzt. Mit dem zweiten Taktimpuls wird MSB/2 = $U_{max/4}$ dazugeschaltet, und der Komparator vergleicht jetzt U_M mit U_V = MSB + MSB/2 = $U_{max/2} + U_{max/4}$.
Solange die Vergleichsspannung $U_V < U_M$ ist, steht am Komparatorausgang H an, und die entsprechenden Spannungsstufungen werden »verriegelt« bzw. bleiben gesetzt. Sobald sich bei der bitweisen Aufsummierung der gestuften, immer weiter halbierten Spannungswerte eine Vergleichsspannung $U_V > U_M$ ergibt, steht am Komparatorausgang L an, und das letzte bit wird zurückgesetzt. Dieser Vorgang wiederholt sich, bis das LSB abgearbeitet ist.
Diese Wandlerprinzipien werden durch ein kleines Unterprogramm des Anwenderprogramms des Mikrorechners realisiert, wobei beim Folge-ADU in Abhängigkeit vom Vergleich U_M mit U_V eine kleinste Spannungsstufe ΔU aufsummiert oder subtrahiert werden muß. Beim ADU nach der sukzessiven Approximation erfolgt diese Aufsummation oder Subtraktion mit gestuften (d.h. ausgehend von der maximalen Umsetzspannung U_{max} jeweils halbierten) Spannungswerten.

Schaltungsbeschreibung

Durch den Einsatz des Polycomputer 880 als Steuer- und Verarbeitungssystem kann der Hardwareaufwand relativ minimal gehalten werden. So enthält die Schaltung nach Bild 4, die auf einer Leiterkarte realisiert an die entsprechende Steckerleiste des Polycomputers angeschlossen wird, im wesentlichen nur zwei Baugruppen:

- einen 8-bit-DAU mit Spannungsverstärker
- einen Komparator.

Damit lassen sich u.a. folgende Funktionsweisen realisieren:

- Digital-Analog-Umsetzung
- digital steuerbarer Funktionsgenerator
- Analog/Digital-Umsetzung (Einquadrantenumsetzung) mit verschiedenen Umsetzverfahren
- Nullpunkttrigger bzw. Trigger mit Offsetkompensation
- Fensterdiskriminator
- Zeit- und Frequenzmessung.

Bedingt durch den vordergründigen Einsatz in Demonstrations- und Lernsysteme wurden die einzelnen Schaltungen unkompliziert und übersichtlich gehalten und weisen keine speziellen Besonderheiten auf.
Der Digital-Analog-Umsetzer arbeitet nach dem Prinzip der Stromsummation, wobei die einzelnen Stromquellen durch die Widerstände $R_{V1} \cdots R_{V8}$ realisiert werden und die Stromsenke der Widerstand R_s bildet. Die Steuerung der Stromquellen wird über Kurzschlußschalter (Variante mit Open-Kollektor-Inverter) bzw. über Serienkurzschlußschalter (Variante mit Gegentakt-Inverter) erreicht. Diese einfache Schaltungsvariante besitzt den Vorteil, daß durch Parameteränderungen der einzelnen Widerstände

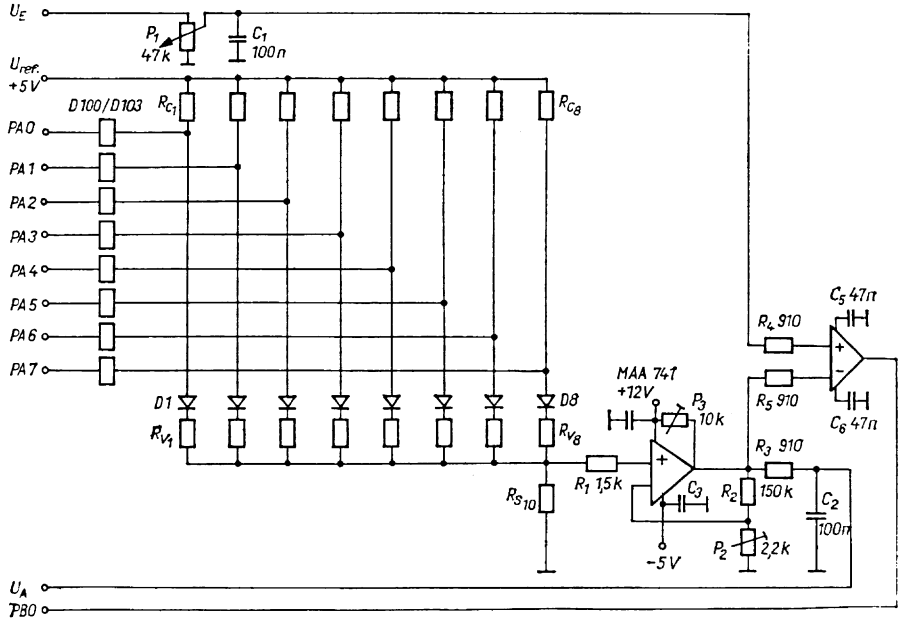


Bild 4. 8-bit-DAU mit Spannungsverstärker und Komparator

$R_{V1} \cdots R_{V8}$ bzw. R_S deren Einfluß auf Kennlinien- und Linearitätsfehler dargestellt werden kann. Weiterhin ist die Möglichkeit gegeben, durch Änderung der Wichtung der Widerstandswerte $R_{V1} \cdots R_{V8}$ nichtlineare Umsetzcharakteristiken zu erzeugen.

Dem Digital-Analog-Wandler schließt sich ein Spannungsverstärker (MAA 741) an, mit dem die Größe der Ausgangsspannung U_A variiert werden kann. Zur Unterdrückung von Glitsches wurde der Tiefpaß R_3/C_2 am Ausgang des DAU vorgesehen.

Der zweite Schaltungsteil beinhaltet einen Komparator A 110, wobei ein Eingang mit dem DAU-Ausgang verbunden ist, und der andere über einen Spannungsteiler das Eingangssignal erhält (siehe auch Bild 1).

Die Steuerung des DAU bzw. die Informationsübernahme des Komparatorzustandes erfolgt über die PIO U 855 D, die bereits im Polycomputer 880 enthalten ist. Desgleichen erfolgt die Stromversorgung der Zusatzschaltungen durch den Mikrorechner über den Peripheriesteckverbinder.

Programmbeschreibung

Hauptprogramm

Vor dem Programmstart sind die Register A mit der Zeitkonstanten für die CTC, B mit der Schrittweite und C mit der Anfangsbitbelegung für den DAU zu laden. Während der Anfangsinitialisierung des Programms werden diese Registerinhalte auf Merkzellen für die spätere Bearbeitung gerettet. Falls im Akku eine "Ø" übergeben wurde, wird diese in eine "1" übergeführt, da die CTC die "Ø" als "ØFFH" interpretieren würde, also die maximal mögliche Zeitkonstante. Das I-Register

und damit der höherwertige Teil des Interruptvektors wird mit 42 H geladen sowie der Interruptmode 2 eingestellt.
Für die Datenbyteausgabe an den DAU wird, wie im Stromlaufplan ersichtlich, Port A der PIO genutzt und deshalb für Mode Ø »Byte-Ausgabe« initialisiert. Das Komparatorbit ist an Bit Ø des Port B der PIO angeschlossen. Port B wird hier für Mode 3 »Bitgesteuerte E/A« initialisiert, wobei alle Leitungen als Eingänge definiert sind.
Der PIO-Initialisierung folgt die Ausgabe des Programmnamens »ADU« auf dem LED-Display. Bis hierher kann man die Befehlsfolge als Initialisierungsteil des Programms bezeichnen. Es schließt sich der eigentliche Meßzyklus an. Dazu wird die aktuelle Bitbelegung des Ausgabebytes von der Zelle »Merk1« geladen und über die PIO an den DAU ausgegeben sowie die CTC für die Realisierung des Zeitrasters initialisiert. Der Wert des Ausgabebytes ⟨C⟩ wird zur Kompensation der Wirkung des Eingangsspannungsteilers mit 4 multipliziert, in eine Dezimalzahl gewandelt und zur Anzeige gebracht.
Die CPU begibt sich nun in eine Schleife und wartet den CTC-Interrupt ab. Innerhalb der Interruptwarteschleife wird die LED-Anzeige durch zyklische Ausgabe ständig aufgefrischt. Die Betriebsart des ADUs wird innerhalb der CTC-Interrupt-Serviceroutine mit der Befehlsfolge des »USER-Programms« festgelegt (siehe Bild 5).

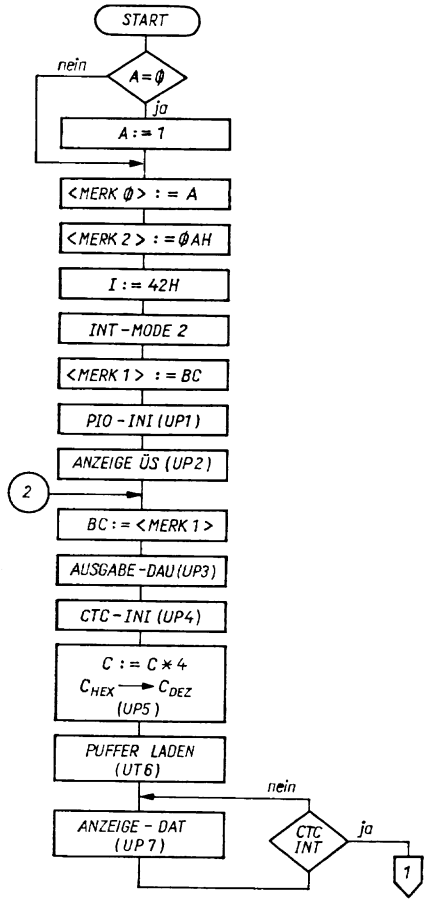


Bild 5. Hauptprogramm

CTC-Interrupt-Serviceroutine

Empfängt die CPU einen CTC-Interrupt, so wird der Programmzeiger der CPU auf die Interrupt-Serviceroutine gestellt und diese abgearbeitet. Die CTC wird abgeschaltet und das Vergleichsergebnis des Komparators in den Akku eingelesen. Anschließend werden das aktuelle Ausgabebitmuster und die Schrittweite von der Zelle »Merk1« in das BC-Register geladen. Damit ist das »USER-Programm« mit den notwendigen Daten versorgt. Für die sukzessive Approximation tritt als Grenzfall der Wert $B = 1$ auf. Damit auch bei dieser Betriebsart nach Erreichen des Komparatorpunkts weitere Messungen ausgelöst werden, bewirkt das Programm nach Abarbeiten einer Hilfszeitschleife das Rücksetzen

der Arbeitszellen und einen Neubeginn der Approximation.
Nachdem im eingelesenen Vergleichsergebnis des Komparators die überflüssigen Bits ausgeblendet wurden, kann mit dem »USER-Programm« die gewünschte Arbeitsweise des ADU realisiert werden. Vor dem anschließenden Rücksprung aus der Interruptroutine wird die Zieladresse unter Zuhilfenahme von Austauschbefehlen in den Stack geladen (Bild 6).

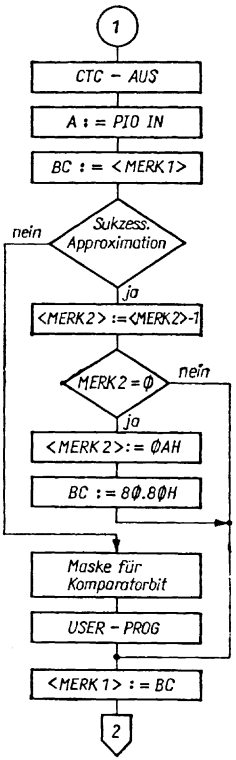


Bild 6. CTC-Interrupt-Serviceroutine

PIO-Initialisierung

Mit diesem Unterprogramm (Bild 7) wird die am E/A-Stecker angeschlossene PIO des Polycomputers so initialisiert, daß über Port A die Ausgabe des Bitmusters an den DAU erfolgen und über Port B das Vergleichsergebnis des Komparators in den Rechner eingelesen werden kann. Da ein Interruptbetrieb der PIO unnötig ist, wird dieser verboten.

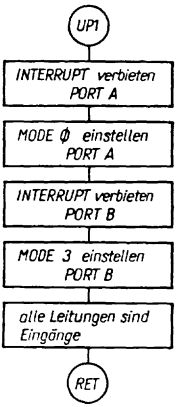


Bild 7. Unterprogramm 1
PIO-Initialisierung

CTC-Initialisierung

Zur Erzeugung des Zeitrasters, unter welchem der DAU arbeiten soll, wird der Kanal 2 als Zeitgeber und der Kanal 3 als Zähler initialisiert. Diese Kombination ist möglich, weil im Polycomputer der Zählerausgang des Kanals 2 mit dem Triggereingang des Kanals 3 verbunden ist. Damit kein vorzeitiger Interrupt ausgelöst wird, wird die CPU vor der Initialisierung in den Dissable-Interrupt-Zustand gebracht.
Bei der Wahl des Interruptvektors ist zu beachten, daß die Interrupttabelle der CTC auf einer durch acht teilbaren Adresse beginnen muß und dieser nur über Kanal Ø der CTC übermittelt werden kann (Bild 8).

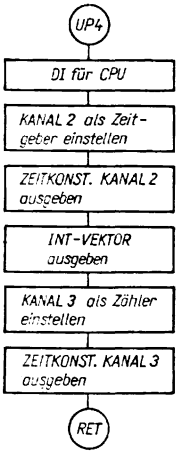


Bild 8.
Unterprogramm 4
CTC-Initialisierung

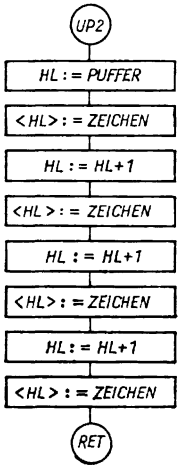


Bild 9.
Unterprogramm 2
Anzeige der
Überschrift

Anzeige der Überschrift

Um den Programmnamen »ADU« auf der LED-Anzeige des Polycomputers ausgeben zu können, ist es zweckmäßig, den Anzeigepuffer des Monitors zu benutzen, da auf diese Weise gleich das interne Treiberprogramm für die Ausgabe genutzt werden kann. Dazu wird die Adresse des Anzeigepuffers in das HL-Register geladen und unter Weiterstellen dieses Zeigers der Puffer mit den entsprechenden Zeichen gefüllt (Bild 9). Zu beachten ist, daß im Polycomputer 880 keine ASCII-Zeichen benutzt werden.

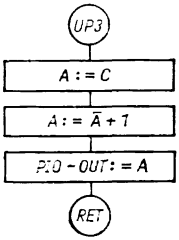


Bild 10.
Unterprogramm 3
Ausgabe DAU

Ausgabe DAU

Die aktuelle Bitbelegung des Ausgabebytes an den DAU steht an dieser Stelle des Programms im C-Register. Als Treiber zur Ansteuerung des Widerstandsnetzwerkes des DAU wird der Schaltkreis D100 benutzt. Da dieser die Signale negiert, für die Anschaulichkeit der Arbeitsweise ein highaktives Signal aber am PIO-Ausgang günstiger ist, wird der Inhalt des C-Registers vor der Ausgabe negiert (Bild 10).

Multiplikation × 4 und Konvertierung Hexadezimal — Dezimal

Nachdem das B-Register zum Auffangen des Überlaufs der sich anschließenden Linksrotation gelöscht ist, werden die Bits 0, 1 und 2 aus dem C-Register ausgeblendet. Diese Maßnahme ist notwendig, da erstens die zwei niederwertigsten Bits im DAU nicht verdrahtet sind und zweitens ohne die Ausblendung von Bit 2 die LED-Anzeige ständig zwischen zwei Werten wechseln würde. Die zweimalige Linksrotation des C-Registers bewirkt eine Multiplikation mit vier. Damit wird im Zusammenhang mit der verwendeten Hardwarelösung der Meßbereich 0 ··· 10 V eingestellt.
Da der so erhaltene Meßwert jedoch auf Grund der Arbeitsweise des Mikrorechners und des DAU als Hexadezimalzahl im BC-Register steht, der Mensch aber gewohnt ist, den Meßwert im Dezimalsystem zu erfassen, übernimmt eine Konvertierungsroutine die Wandlung in eine Dezimalzahl (Bild 11). Der Algorithmus der Konvertierung beruht auf der zyklischen Subtraktion von Zehnerpotenzen, beginnend mit der höchsten Wertigkeit, wobei die mitgezählten Zyklusanzahlen die gesuchte Dezimalzahl darstellen.

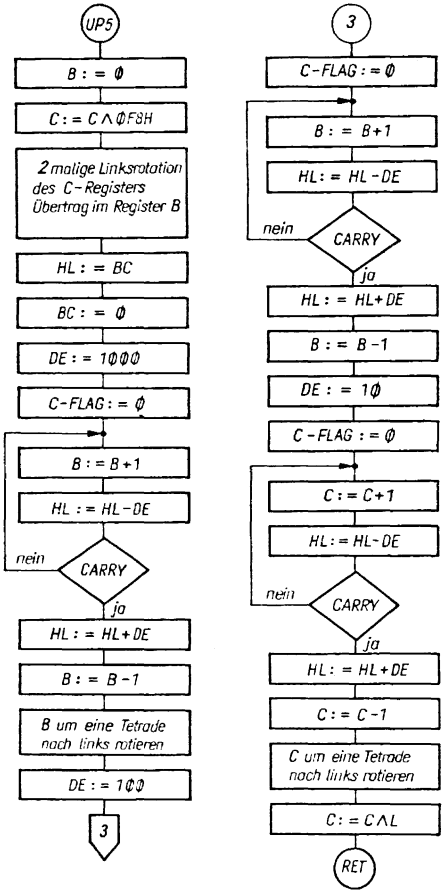


Bild 11. Unterprogramm 5
Multiplikation × 4 und Konvertierung Hexadezimal-Dezimal

Anzeigepuffer laden

Da der Polycomputer nicht wie normalerweise üblich mit ASCII-Zeichen arbeitet, wurde zur Wandlung des Meßwerts in den Anzeigecode der 7segmentanzeige die interne Referenztabelle benutzt. Der höherwertige Teil der Tabellenadresse liegt mit "3" fest. Der niederwertige Teil der Adresse ergibt sich aus dem jeweiligen Zahlenwert plus einem Festwert von 1Ø H. Der unter dieser Adresse aufgefundene Anzeigecode wird fortlaufend in den Anzeigepuffer eingeschrieben. Bei der ersten und dritten Stelle muß vor dem Abspeichern noch der Dezimalpunkt ausgeblendet werden (Bild 12).

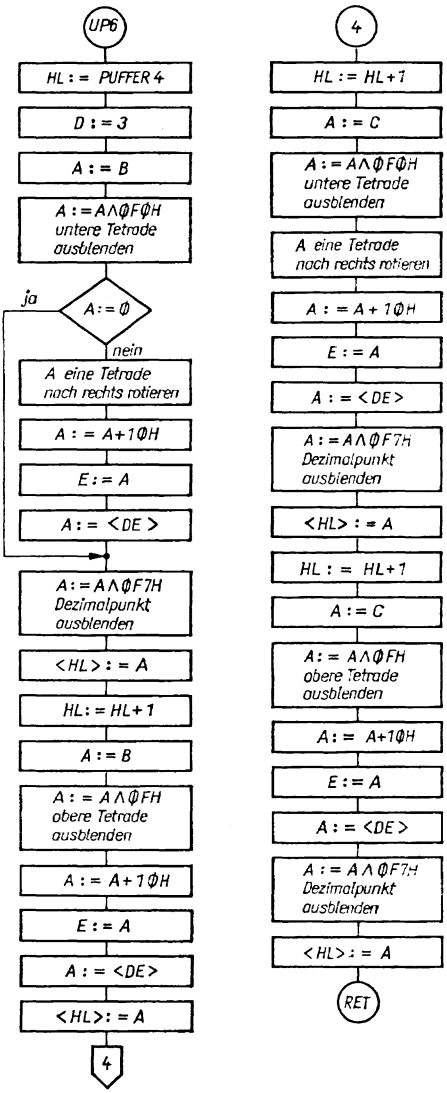


Bild 12. Unterprogramm 6
Anzeigepuffer laden

Anzeige der Daten auf dem LED-Display

Dieses Programm existiert bereits im Polycomputer als Unterprogramm des Monitors und wird vom ADU-Programm mitgenutzt. Eine Programmbeschreibung ist in der Dokumentation zum Polycomputer zu finden [6].
Das detaillierte Anwenderprogramm zur Realisierung der ADU-Funktionen mit Hilfe des Polycomputer 880 finden Sie auf S. 44 bis 48.

Inbetriebnahme der mikrorechnergesteuerten ADU

Nachdem das auf S. 44 dargestellte Anwenderprogramm mittels Tastatur eingegeben bzw., falls es bereits auf einer Tonbandkassette abgespeichert ist, in den RAM-Bereich des Mikrorechners eingelesen ist (Anfangsadresse 4000, Endadresse 4300), kann das folgende Unterprogramm für den Stufen- oder Folge-ADU über die Tastatur im Hexadezimalcode ab Adresse 4002 eingegeben werden:

Mnemonische Beschreibung		Adresse	Maschinenprogramm	
	CMP A	4002	FE	00
	JPZ M1	4004	CA	0B 40
	LD A, C	4007	79	
	SUB B	4008	90	
	LD C, A	4009	4F	
	RET	400A	C9	
M1:	LD A, C	400B	79	
	ADD B	400C	80	
	LD C, A	400D	4F	
	RET	400E	C9	

In diesem Unterprogramm wird jeweils in einer Additions- bzw. Subtraktionsschleife der aktuelle Inhalt des C-Registers in den Akkumulator A geladen und mit dem im B-Register abgespeicherten Wert der Spannungsstufung (im Fall des Folge-ADU wird die kleinste Spannungsstufe $\Delta U = 04H$ gewählt, da im realisierten DAU die beiden niederwertigen Bit 01H und 02H nicht belegt sind) erhöht bzw. erniedrigt. Diese stufenweise Annäherung der »Vergleichsspannung« an die Meßspannung U_M kann auch einfach dadurch realisiert werden, daß man anstelle der Additionsschleife den Befehl INC C = 0CH bzw. für die Subtraktionsschleife DEC C = 0DH einfügt. Dabei erhöht dann der Rechner die »Vergleichsspannung« in 01H-Stufen, während extern auf dem Sichtgerät die Spannungsstufung in 04H-Schritten angezeigt wird.
Zur Realisierung der Funktion des ADU nach dem Verfahren der sukzessiven Approximation, die gleichfalls durch eine Additions- und Subtraktionsschleife verwirklicht werden kann, ist nur zu gewährleisten, daß der im Register B abgespeicherte Anfangswert $U_{max/2}$ vor jeder Schleifenabarbeitung halbiert wird. Dies ist einfach durch den Befehl RRC B = CB 08, der auf die Adresse 4000 und 4001 abgespeichert wird, zu erreichen, da durch die Rechtsrotation eines Bits die mathematische Division durch 2 der Spannungsstufung im Register B realisiert wird. Durch ein einfaches Ersetzen dieses RRC-Befehls durch NOP-Befehle wird die Funktion des Stufen- oder Folge-ADU's wieder hergestellt.
Während vor dem Start des Programms und damit der Spannungsumsetzung im Register C ein beliebiger Anfangswert (z. B. 04H für den Folge-ADU bzw. 80H für die sukzessive Approximation) eingegeben wird, ist durch die Eingabe einer Hexadezimalzahl in das A-Register die Umsetzgeschwindigkeit wählbar. Je größer diese Zahl gewählt wird, desto größer ist die Zeitschleife und desto kleiner die Umsetzgeschwindigkeit. Bei Eingabe von

```
                        ABS. ASSEMBLER  MEOS 1521 V3.2/4
  LOC        OBJ.CODE     STMT     SOURCE PROGRAM
=HEX==DEC=============================================================================18/09/84==001==============
                         00001          PN   A0U
                         00002  ;
                         00003  ;
                         00004  ;       ANALOG - DIGITALWANDLUNG MITTELS POPLVCOMP - 880
                         00005  ;       UMSETZZEITGRENZE IST SOFTWAREBEDINGT
                         00006  ;
                         00007  ;       10.09.84
                         00008  ;
                         00009          ORG  4100H
                         00010  ;
                         00011  ;
4100 16640  FE 00        00012  START: CMP  0
4102 16642  20 02        00013          JRNZ ST1-#
4104 16644  3E 01        00014          LD   A,1
4106 16646  32 62 42     00015  ST1:    LD   (MERK0),A          ;ZEITKONSTANTE FUER CTC
4109 16649  3E 0A        00016          LD   A,0AH
410B 16651  32 65 42     00017          LD   (MERK2),A          ;HILFSZEITKONSTANTE
410E 16654  3E 42        00018          LD   A,42H
4110 16656  ED 47        00019          LD   I,A                ;HWT - INTERRUPTVEKTOR
4112 16658  ED 5E        00020          IM2                     ;INTERRUPT - MODE 2
4114 16660  ED 43 63 42  00021          LD   (MERK1),BC         ;ANFANGSWERT + SCHRITTWEITE FUER UMSETZUNG
4118 16664  CD 6C 41     00022          CALL PIOI               ;PIO - INITIALISIEREN
411B 16667  CD 98 41     00023          CALL ANZLO              ;ANZEIGE LOESCHEN
411E 16670  ED 4B 63 42  00024  M0:     LD   BC,(MERK1)
4122 16674  CD A7 41     00025          CALL PIOAU              ;AUSGABE AN DAU
4125 16677  CD 81 41     00026          CALL CTCI               ;CTC - INIT.
4128 16680  CD E8 41     00027          CALL KONV               ;MULTIPLIKATION * 4 UND KONVERTIERUNG HEX --> DEZ
412B 16683  CD AD 41     00028          CALL ANZLA              ;ANZEIGEPUFFER LADEN
412E 16686  11 58 42     00029  M1:     LD   DE,ANZBE          ;<DE> = ANZEIGEPUFFER
4131 16689  F3           00030          DI                      ;INTERRUPT FUER CPU VERBIETEN
4132 16690  CD 4E 01     00031          CALL ANZEI              ;ANZEIGEN DES AKTUELLEN WERTES.
4135 16693  FB           00032          EI                      ;INTERRUPT FUER CPU ERLAUBEN
4136 16694  18 F6        00033          JR   M1-#
                         00034  ;
                         00035  ;       CTC - INTERRUPT - SERVICEROUTINE
                         00036  ;
4138 16696  3E 27        00037  INT:    LD   A,27H
413A 16698  D3 89        00038          OUT  89H                ;CTC - AUSSCHALTEN
413C 16700  DB 86        00039          IN   86H                ;KOMPARATOR (BIT 0)
413E 16702  ED 4B 63 42  00040          LD   BC,(MERK1)
4142 16706  57           00041          LD   D,A                ;AKKU RETTEN
4143 16707  78           00042          LD   A,B
4144 16708  FE 01        00043          CMP  1
```

```
4146 16710  20 13          00044          JRNZ INT1-#
4148 16712  3A 65 42       00045          LD   A,(MERK2)
414B 16715  3D              00046          DEC  A                    ;HILFSZEITSCHLEIFE
414C 16716  32 65 42       00047          LD   (MERK2),A
414F 16719  20 10          00048          JRNZ INT2-#
4151 16721  01 80 80       00049          LD   BC,8080H
4154 16724  3E 0A          00050          LD   A,0AH
4156 16726  32 65 42       00051          LD   (MERK2),A
4159 16729  18 06          00052          JR   INT2-#
415B 16731  7A              00053  INT1:   LD   A,D
415C 16732  E6 01          00054          AND  1                    ;KOMPARATORBIT AUSBLENDEN
415E 16734  CD 00 40       00055          CALL USER
4161 16737  ED 43 63 42    00056  INT2:   LD   (MERK1),BC          ;NEUES AUSGABE - BYTE AN DAU
4165 16741  E3              00057          EX   (SP),HL
4166 16742  21 1E 41       00058          LD   HL,M0                ;RUECKSPRUNGADRESSE IN STACK LADEN
4169 16745  E3              00059          EX   (SP),HL
416A 16746  ED 4D          00060          RETI
                            00061          EJEC
                            00062  ;
                            00063  ;        PIO - INITIALISIERUNG
                            00064  ;
416C 16748  3E 03          00065  PIOI:   LD   A,3                  ;INT. SPERRE,KANAL A
416E 16750  D3 85          00066          OUT  85H
4170 16752  3E 0F          00067          LD   A,0FH                ;BYTE - AUSGABE
4172 16754  D3 85          00068          OUT  85H
4174 16756  3E 03          00069          LD   A,3                  ;INT. SPERRE,KANAL B
4176 16758  D3 87          00070          OUT  87H
4178 16760  3E CF          00071          LD   A,0CFH               ;MODE - 3
417A 16762  D3 87          00072          OUT  87H
417C 16764  3E FF          00073          LD   A,0FFH               ;ALLE LEITUNGEN SIND EINGAENGE
417E 16766  D3 87          00074          OUT  87H
4180 16768  C9              00075          RET
                            00076  ;
                            00077  ;        CTC - INITIALISIERUNG
                            00078  ;
4181 16769  F3              00079  CTCI:   DI
4182 16770  3E 07          00080          LD   A,7                  ;STEUERWORT
4184 16772  D3 8A          00081          OUT  8AH
4186 16774  3E FF          00082          LD   A,0FFH               ;ZEITKONSTANTE FUER KANAL 2
4188 16776  D3 8A          00083          OUT  8AH
418A 16778  3E 50          00084          LD   A,50H                ;NWT - INTERRUPTVEKTOR FUER CTC
418C 16780  D3 88          00085          OUT  88H                  ;(KANN NUR UEBER KANAL - 0 AUSGEGEBEN WERDEN)
418E 16782  3E F7          00086          LD   A,0F7H               ;STEUERWORT
4190 16784  D3 8B          00087          OUT  8BH
4192 16786  3A 62 42       00088          LD   A,(MERK0)            ;ZEITKONSTANTE
4195 16789  D3 8B          00089          OUT  8BH
4197 16791  C9              00090          RET
```

```
                                00091  ;
                                00092  ;       ANZEIGE LOESCHEN
                                00093  ;
4198 16792  21 58 42      00094  ANZLO: LD   HL,ANZBE              ;ANZEIGEPUFFER
419B 16795  36 F5         00095         LD   M,0F5H               ;A
419D 16797  23            00096         INC  HL
419E 16798  36 97         00097         LD   M,97H                ;D
41A0 16800  23            00098         INC  HL
41A1 16801  36 C7         00099         LD   M,0C7H               ;U
41A3 16803  23            00100         INC  HL
41A4 16804  36 00         00101         LD   M,0                  ;SPACE
41A6 16806  C9            00102         RET
                          00103  ;
                          00104  ;       AUSGABEBYTE AN ADU
                          00105  ;
41A7 16807  79            00106  PIOAU: LD   A,C
41A8 16808  ED 44         00107         NEG
41AA 16810  D3 84         00108         OUT  84H
41AC 16812  C9            00109         RET
                          00110         EJEC
                          00111  ;
                          00112  ;       LADEN DES ANZEIGEPUFFERS
                          00113  ;
41AD 16813  21 5C 42      00114  ANZLA: LD   HL,ANZBE+4
41B0 16816  16 03         00115         LD   D,3                  ;HWT - TABELLENADRESSE FUER  7-SEGMENTANZEIGE
41B2 16818  78            00116         LD   A,B
41B3 16819  E6 F0         00117         AND  0F0H
41B5 16821  FE 00         00118         CMP  0
41B7 16823  28 08         00119         JRZ  AN1-#
41B9 16825  0F            00120         RRCA
41BA 16826  0F            00121         RRCA
41BB 16827  0F            00122         RRCA
41BC 16828  0F            00123         RRCA
41BD 16829  C6 10         00124         ADD  10H                  ;TABELLENADRESSE BILDEN
41BF 16831  5F            00125         LD   E,A                  ;NWT - TABELLENADRESSE
41C0 16832  1A            00126         LD   A,(DE)
41C1 16833  E6 F7         00127  AN1:   AND  0F7H                 ;DEZIMALPUNKT AUSBLENDEN
41C3 16835  77            00128         LD   M,A
41C4 16836  23            00129         INC  HL
41C5 16837  78            00130         LD   A,B
41C6 16838  E6 0F         00131         AND  0FH
41C8 16840  C6 10         00132         ADD  10H                  ;TABELLENADRESSE BILDEN
41CA 16842  5F            00133         LD   E,A                  ;NWT - TABELLENADRESSE
41CB 16843  1A            00134         LD   A,(DE)
41CC 16844  77            00135         LD   M,A
41CD 16845  23            00136         INC  HL
```

```
41CE 16845  79          00137          LD   A,C
41CF 16847  E6 F0       00138          AND  0F0H
41D1 16849  0F          00139          RRCA
41D2 16850  0F          00140          RRCA
41D3 16851  0F          00141          RRCA
41D4 16852  0F          00142          RRCA
41D5 16853  C6 10       00143          ADD  10H                    ;TABELLENADRESSE BILDEN
41D7 16855  5F          00144          LD   E,A                    ;NWT - TABELLENADRESSE
41D8 16856  1A          00145          LD   A,(DE)
41D9 16857  E6 F7       00146          AND  0F7H                   ;DEZIMALPUNKT AUSBLENDEN
41DB 16859  77          00147          LD   M,A
41DC 16860  23          00148          INC  HL
41DD 16861  79          00149          LD   A,C
41DE 16862  E6 0F       00150          AND  0FH
41E0 16864  C6 10       00151          ADD  10H                    ;TABELLENADRESSE BILDEN
41E2 16866  5F          00152          LD   E,A                    ;NWT - TABELLENADRESSE
41E3 16867  1A          00153          LD   A,(DE)
41E4 16868  E6 F7       00154          AND  0F7H                   ;DEZIMALPUNKT AUSBLENDEN
41E6 16870  77          00155          LD   M,A
41E7 16871  C9          00156          RET
                        00157  ;
                        00158  ;       MULTIPLIKATION * 4 DES C - REGISTERS
                        00159  ;
41E8 16872  06 00       00160  KONV:   LD   B,0
41EA 16874  79          00161          LD   A,C
41EB 16875  E6 F8       00162          AND  0F8H                   ;BITS 0,1,2 AUSBLENDEN
41ED 16877  4F          00163          LD   C,A
41EE 16878  CB 21       00164          SLA  C
41F0 16880  CB 10       00165          RL   B
41F2 16882  CB 21       00166          SLA  C
41F4 16884  CB 10       00167          RL   B
                        00168          EJEC
                        00169  ;
                        00170  ;       WANDLUNG HEX --> DEZ
                        00171  ;
41F6 16886  60          00172          LD   H,B
41F7 16887  69          00173          LD   L,C
41F8 16888  01 00 00    00174          LD   BC,0
41FB 16891  11 E8 03    00175          LD   DE,1000
41FE 16894  AF          00176          XOR  A
41FF 16895  04          00177  H1:     INC  B
4200 16896  ED 52       00178          SBC  HL,DE
4202 16898  30 FB       00179          JRNC H1-#
4204 16900  19          00180          ADD  HL,DE
4205 16901  05          00181          DEC  B
4206 16902  CB 20       00182          SLA  B
```

```
4208 16904  CB 20        00183          SLA  B
420A 16906  CB 20        00184          SLA  B
420C 16908  CB 20        00185          SLA  B
420E 16910  11 64 00     00186          LD   DE,100
4211 16913  AF           00187          XOR  A
4212 16914  04           00188  H0:     INC  B
4213 16915  ED 52        00189          SBC  HL,DE
4215 16917  30 FB        00190          JRNC H0-#
4217 16919  19           00191          ADD  HL,DE
4218 16920  05           00192          DEC  B
4219 16921  11 0A 00     00193          LD   DE,10
421C 16924  AF           00194          XOR  A
421D 16925  0C           00195  H3:     INC  C
421E 16926  ED 52        00196          SBC  HL,DE
4220 16928  30 FB        00197          JRNC H3-#
4222 16930  19           00198          ADD  HL,DE
4223 16931  0D           00199          DEC  C
4224 16932  CB 21        00200          SLA  C
4226 16934  CB 21        00201          SLA  C
4228 16936  CB 21        00202          SLA  C
422A 16938  CB 21        00203          SLA  C
422C 16940  79           00204          LD   A,C
422D 16941  B5           00205          OR   L
422E 16942  4F           00206          LD   C,A
422F 16943  C9           00207          RET
                         00208  ;
                         00209          ORG  4250H
                         00210  ;
                         00211  ;        INTERRUPTTABELLE FUER CTC - BAUSTEIN
                         00212  ;
4250 16976  38 41        00213  INTA:   DA   INT                    ;KANAL 0
4252 16978  38 41        00214          DA   INT                    ;KANAL 1
4254 16980  38 41        00215          DA   INT                    ;KANAL 2
4256 16982  38 41        00216          DA   INT                    ;KANAL 3
                         00217  ;
4258 16984               00218  ANZBE:  BER  10                     ;ANZEIGEPUFFER
4262 16994               00219  MERK0:  BER  1                      ;ZEITKONSTANTE - CT(
4263 16995               00220  MERK1:  BER  2                      ;AUSGABEBYTE AN DAU
4265 16997               00221  MERK2:  BER  1                      ;HILFSZEITKONSTANTE
4266 16998               00222          BER  0
                         00223  ;
                         00224  ANZEI:  EQU  14EH
                         00225  USER:   EQU  4000H
4266 16998               00226          END
                        PROGRAM CONTAINS 00000 ERRORS
```

ØØH sind allein die im Anwenderprogramm programmierten Zeitschleifen wirksam.
Abschließend sei festgestellt, daß das Auflösungsvermögen dieser mikrorechnergestützten Analog-Digital-Wandler durch die Bit-Zahl der Datenworte vorgegeben ist und die Genauigkeit durch Verwendung der nicht speziell stabilisierten Rechnerbetriebsspannung zur Realisierung der Vergleichsspannung bestimmt wird. In [4] sind industrielle Zusatzbaugruppen zur Realisierung einzelner ADU mit hoher Genauigkeit vorgestellt. Ziel dieses Beitrages war es, zu zeigen, wie unter Verwendung des Polycomputers 880 mit sehr geringem Hardwareaufwand und mit einfachen Softwaremitteln die Funktionsweisen einzelner ADU realisiert und demonstriert werden können.

Literatur

[1] Arnold, H.; Pilz, W.: Polycomputer 880. – In: r f e. – Berlin **31** (1982) 6, – S. 385—386

[2] Jakubaschk, H.: Erfahrungen mit dem Polycomputer PC 880. – In: r f e. – Berlin **32** (1983) 8, – S. 492—493

[3] Burkhardt, S.; Hübner, U.: Technik und Anwendung des Mikrorechnerlernsystems Polycomputer 880. – In: r f e. – Berlin **33** (1984) 5, – S. 282—287

[4] Hübner, U.: Zusatzgeräte für Polycomputer 880. – In: r f e. – Berlin **33** (1984) 7, – S. 415—419

[5] Hübner, U.: Polycomputer 880 – Anwendung und Erweiterungsmöglichkeiten. – In: Kleinstrechner-TIPS, H. 3. – Leipzig, 1985

[6] Dokumentation des Polycomputer 880 (5 Hefte)

Autoren:

Dr.-Ing. Kurt Lehmann
Wissenschaftlicher Oberassistent an der Ingenieurhochschule Dresden

Dipl.-Ing. Lothar Schumann
Wissenschaftlicher Assistent an der Ingenieurhochschule Dresden

Dipl.-Ing. Peter Walke
Problemanalytiker
Bezirkshygieneinspektion und -institut Dresden

Rechentechnische Begriffe für den Laien erklärt

Kode Eineindeutige Abbildung einer Menge von Informationen auf eine Menge von Zahlen. Ein weit verbreitetes Kodesystem ist der ASCII-Kode.

SP Leerzeichen (SPACE). Als Leerzeichen wird ein Leerschritt bei der Ein- und Ausgabe von Informationen (Texten) bezeichnet.

CR Wagenrücklauf (CARRIAGE RETURN). Das Steuerzeichen 'Wagenrücklauf' bewirkt bei Druckern die Rückführung des Druckkopfes an den linken Blattrand und bei Bildschirmgeräten die Rückführung des Cursors an den linken Bildschirmrand.

LF Zeilenvorschub (LINE FEED). Das Steuerzeichen 'Zeilenvorschub' bewirkt auf Druckern und Bildschirmgeräten den Übergang auf die nächste Zeile. Der nächste Zeilenanfang wird eingestellt durch Ausgabe der Steuerzeichen CR und LF, bei manchen Geräten ist dafür auch das Steuerzeichen NL (Neue Zeile – NEW LINE) realisiert.

Archimedes und die Zahl Pi

Fährt man auf der Fernverkehrsstraße von Halle an der Saale nach Eisleben, so stößt man in der Ortslage Halle-Neustadt auf ARCHIMEDES' (etwa 287 bis 212 v.u.Z.) Spuren, denn hier ist eine Archimedische Wasserschraube zur Absenkung des Grundwasserspiegels installiert. Allerdings ist diese Erfindung nur eine von vielen Leistungen des griechischen Mathematikers ARCHIMEDES von Syrakus. Am meisten scheint hingegen die Legende um die Berechnung der Wichte bemüht worden zu sein, nach der ARCHIMEDES, im Bade sitzend, die Lösung fand, eine vermeintlich goldene Krone auf minderwertige Metallbestandteile zu untersuchen. Er soll nach diesem Erkenntnisblitz unbekleidet und »Heureka« (Ich hab's) rufend nach Hause geeilt sein. Belegt ist der Vorgang nicht, er paßt aber zu den Berichten des griechischen Schriftstellers PLUTARCH. Er schreibt, daß ARCHIMEDES über seinem Grübeln oft Essen und Trinken vergessen habe und gewaltsam zum Baden und Salben geschleppt werden mußte, wobei er sogar noch auf seinen gesalbten Körper, ebenso wie in Sand oder Asche, geometrische Figuren gezeichnet haben soll.

Die Leistungen von ARCHIMEDES auf den Gebieten der Mathematik, Physik und Astronomie sind vielfältig. Neben den schon genannten Beispielen führte er den Begriff des Schwerpunktes ein und bestimmte ihn praktisch an einfachen Körpern. Er stellte die Hebelgesetze auf und wandte sie praktisch an. Er erweiterte den zu seiner Zeit üblichen Zahlenbereich, der nur bis 10^4 ($10^4 = 1$ Myriade) reichte. Er schuf einen Himmelsglobus und entwarf eine Methode zur Ausmessung krummliniger Flächen. Zwei seiner Bücher trugen den Titel »Über Kugel und Zylinder«. Seine Erkenntnis, daß die Volumina eines Zylinders und der ihn umschließenden Kugel im Verhältnis 2:3 stehen, scheint seinen Wunsch begründet zu haben, daß Kugel und Zylinder auf seinem Grabstein dargestellt sein sollten.

Das Wort Kreis hat im Zusammenhang mit ARCHIMEDES eine tragische und eine wissenschaftliche Seite. Tragisch deshalb, weil einer Legende zufolge die letzten Worte von ARCHIMEDES lauteten: Bringe mir meine Kreise nicht durcheinander. Gesagt haben soll er sie zu einem römischen Legionär (die Römer hatten Syrakus nach zweijähriger Belagerung geplündert), der ihn daraufhin erschlagen haben soll.

Die wissenschaftliche Seite beinhaltet Messungen am Kreis, bei denen ARCHIMEDES durch einbeschriebene Vielecke den Kreisumfang rechnerisch ermitteln konnte. Damit schuf er eine der frühesten Methoden zur Bestimmung der transzendenten Zahl π.

Wir wollen die Gedanken von ARCHIMEDES zunächst nachvollziehen, uns dann aber, wenn es um die rechnerische Ausführung zur Bestimmung von π geht, der modernen Rechentechnik bedienen. Dabei wird sich zeigen, daß wir auch hier vor Überraschungen nicht sicher sind. Ausgangspunkt der ARCHIMEDischen Gedanken ist der Einheitskreis in Bild 1. Zunächst ist in Bild 1 ersichtlich, daß sich im Kreis ein reguläres n-Eck (zu Beginn $n = 3$) mit der Seite c und ein reguläres $2n$-Eck (zu Beginn $2n = 6$) mit der Seite a befinden. Wir suchen jetzt eine Gleichung $a = f(c)$. Damit wird es möglich, uns schrittweise über einbeschriebene Vielecke (3-, 6-, 12-, 24-, 48-, 96eck usw.) dem Kreisumfang u zu nähern. Es gilt ja $u = 2 \cdot \pi \cdot r$, und im Einheitskreis mit $r = 1$ gilt $u = 2 \cdot \pi$ und damit $\pi = \frac{u}{2}$.

Beim regulären 6eck beträgt $\alpha = 60°$. Aus Bild 1 ist ersichtlich, daß $\alpha = \alpha'$ ist. Daraus folgt weiter $a = r$, im Einheitskreis also $a = r = 1$.

Zur Bestimmung von b nutzte ARCHIMEDES die Erkenntnisse seines Landsmannes PYTHAGORAS, der rund 300 Jahre vor ihm gelebt hatte.

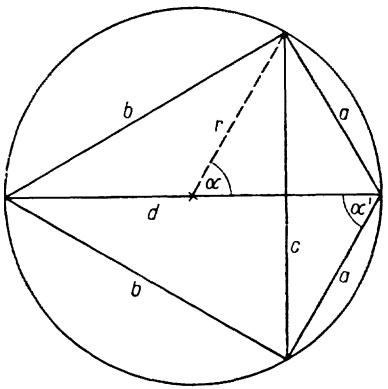


Bild 1. Einheitskreis mit $r = 1$

Nach dem Satz des PYTHAGORAS gilt:

$$a^2 + b^2 = d^2$$

$$b^2 = d^2 - a^2 \quad \text{im Einheitskreis } d = 2$$

$$\curvearrowright b^2 = 4 - a^2$$

$$b = \sqrt{4 - a^2} \quad \text{(nur positive Wurzel ist sinnvoll)}$$

Über die Berechnung des Flächeninhaltes A des »Drachens« mit den Seiten $abba$ wird nun versucht, die Gleichung $a = f(c)$ herzuleiten. Dazu werden 2 Berechnungsarten für die Fläche A genutzt.

1. Berechnung der Fläche A:

$$A = \frac{d \cdot \frac{c}{2}}{2} \cdot 2$$

$$A = d \cdot \frac{c}{2} \quad \text{im Einheitskreis } d = 2$$

$$\curvearrowright A = c$$

2. Berechnung der Fläche A:

$$A = \frac{a \cdot \sqrt{4 - a^2}}{2} \cdot 2$$

$$A = a \cdot \sqrt{4 - a^2}$$

Wir setzen beide Flächenberechnungen gleich und erhalten:

$$c = a \cdot \sqrt{4 - a^2}$$

$$\frac{c}{a} = \sqrt{4 - a^2}$$

Die Gleichung ist nun nach a aufzulösen.

$$\frac{c^2}{a^2} = 4 - a^2$$

$$c^2 = 4a^2 - a^4 \quad \text{Substitution } a^2 = s$$

$$c^2 = 4s - s^2$$

$$c^2 = -s^2 + 4s$$

$$-c^2 = s^2 - 4s$$

$$-c^2+4 = s^2-4s+4$$

$$-c^2+4 = (s-2)^2$$

$$(s-2)^2 = 4-c^2$$

$$s-2 = \pm\sqrt{4-c^2}.$$

Es gilt $s = a^2$

$$a^2-2 = \pm\sqrt{4-c^2}$$

$$a^2 = 2 \pm \sqrt{4-c^2}$$

Da a im Einheitskreis nicht größer als 1 werden kann, ist nur die negative Wurzel sinnvoll.

$$a = \sqrt{2-\sqrt{4-c^2}}$$

Damit liegt die gewünschte Gleichung $a = f(c)$ vor. Betrachten wir nun den Kreisumfang u, so gilt:

$$u = 2 \cdot \pi \cdot r \text{ im Einheitskreis } r = 1$$

$$\curvearrowright u = 2 \cdot \pi$$

$$\curvearrowright \pi = \frac{u}{2}$$

Wir werden die Berechnung mit der in

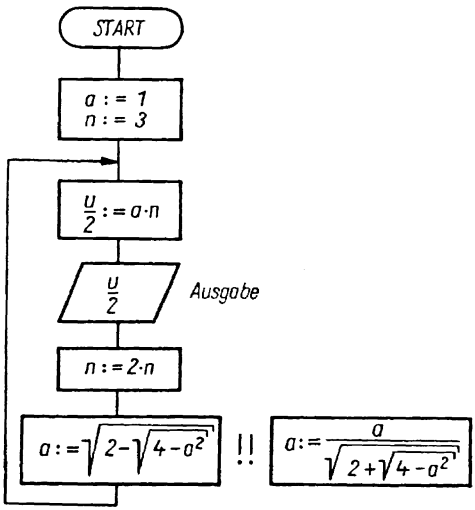


Bild 2. Programmablaufplan zur Berechnung von $\frac{u}{2}$

Bild 1 gezeigten Situation beginnen. Danach ist

$$b = c = \sqrt{4-a^2} \qquad a = 1$$

$$b = c = \sqrt{3}.$$

Die Anzahl der Ecken des regulären n-Ecks beträgt $n = 3$. Aus der Gleichung $a = \sqrt{2-\sqrt{4-c^2}}$ ergibt sich für $c = \sqrt{3}$ der Wert $a = 1$.
Damit folgt für $a = 1$ und $n = 3$ nach der Gleichung

$$\frac{u}{2} = a \cdot n$$

$$\frac{u}{2} = 3, \text{ also } \pi \approx 3.$$

Mit diesen Anfangswerten werden wir, ebenso wie ARCHIMEDES, zu rechnen beginnen. Im zweiten Rechengang verdoppeln wir die Anzahl der Ecken und setzen den Wert $a = 1$ als neuen Wert c in die Gleichung $a = \sqrt{2-\sqrt{4-c^2}}$ ein. Das Ganze dürfte für ARCHIMEDES ohne Taschenrechner, Rechenstab oder Quadratwurzeltabelle recht mühselig gewesen sein.
Wir wollen die Sache jetzt mit den uns heute zur Verfügung stehenden Rechenhilfsmitteln weiter verfolgen. Zu diesem Zweck soll ein Rechenprogramm für einen programmierbaren Tisch- oder Taschenrechner erarbeitet werden. Dazu stellen wir zunächst einen Programmablaufplan (PAP) auf (Bild 2). Wie schon erwähnt, beginnen wir mit $a := 1$ und $n := 3$, errechnen dann $\frac{u}{2} := a \cdot n$ und geben $\frac{u}{2} \approx \pi$ aus. Die Verdoppelung des n-Ecks geht aus der Ergibtanweisung $n := 2 \cdot n$ hervor. Mit einer Ergibtanweisung errechnen wir dann auch das »neue a« aus dem »alten a«, das wir bisher mit c bezeichnet hatten. Es gilt also

$$a := \sqrt{2-\sqrt{4-a^2}}.$$

```
0000 LRK    0028 TEX    0056   9
0001 STM    0029 DRU    0057   8
0002 CL     0030   1    0058 TEX
0003 TXR    0031   0    0059 ZS.
0004 MOD    0032   .    0060 2
0005 1      0033        0061 TXR
0006 TXR    0034   N    0062 MUL
0007 1      0035   =    0063 2
0008 MOD    0036 DRU    0064 TRX
0009 3      0037   X    0065 1
0010 TXR    0038   7    0066 X↑2
0011 2      0039   0    0067 1
0012 MRK    0040 TEX    0068 +/-
0013 1      0041 TRX    0069 MUL
0014 TRX    0042 1      0070 4
0015 1      0043 TEX    0071 ADD
0016 TRX    0044   A    0072 QWX
0017 2      0045   =    0073 1
0018 MUL    0046 DRU    0074 +/-
0019 TXR    0047   9    0075 MUL
0020 3      0048   8    0076 2
0021 MOD    0049 TEX    0077 ADD
0022 1      0050 TRX    0078 QWX
0023 TXR    0051 3      0079 TXR
0024 ADD    0052 TEX    0080 1
0025 TRX    0053   -π   0081 STM
0026 2      0054   =    0082 1
0027 TRX    0055 DRU    0083 END
```

Bild 3. Nichtkonvergierendes Rechenprogramm für den K 1003

Dann wird im PAP zur Berechnung von $\frac{u}{2} \approx \pi$ zurückgesprungen. In Bild 3 ist das entsprechende Rechenprogramm für den programmierbaren Tischrechner K 1003 (Kombinat Robotron, DDR) angegeben. Bild 4 zeigt den zugehörigen Rechnerausdruck. Die erste Druckzeile enthält die laufende Nummer des Rechendurchlaufes und den Wert n. Die zweite Zeile enthält den entsprechenden Wert a und die dritte Zeile den Näherungswert für die transzendente Zahl π. Während wir bis zum 9. Rechendurchlauf noch recht optimistisch dreinblicken, verfinstert sich unser Blick in den Folgeschritten zunehmend, da von einer sinnvollen Näherung an π keine Rede mehr sein kann.

Was ist geschehen? Einen Irrtum von ARCHIMEDES schließen wir von vornherein aus. Die Überprüfung des Rechenprogramms in Bild 3 ergibt auch keinen Fehler. Für den erfahrenen Programmierer ist es aber stets verdächtig, wenn im Rechenprogramm sehr große Zahlen (für n) und sehr kleine Zahlen (für a) auftreten und diese dann auch noch, wie in unserem Fall, multipliziert werden. So ändert sich in den Rechendurchläufen 19 bis 23 der Wert für a überhaupt nicht mehr, während n fleißig weiter verdoppelt wird. Wir stoßen damit an die Grenze der Genauigkeit des Rechners, da der Wert a offenbar nicht schnell genug kleiner wird und sich schließlich gar nicht mehr ändert. Übrigens wird diese Katastrophe bei den Rechnern TI 58/59 vom 11. Rechendurchlauf an deutlich, der Wert für π steigt aber wegen der höheren Rechnergenauigkeit viel langsamer an.

Die Beseitigung dieser Unannehmlichkeit fällt natürlich nicht mehr in den Verantwortungsbereich von ARCHIMEDES. Uns bleibt nur der Versuch, den Ausdruck $\sqrt{2-\sqrt{4-a^2}}$ in eine »rechnerfreundliche« Form zu bringen. Eine Möglichkeit bietet das Erweitern.

$$\sqrt{2-\sqrt{4-a^2}} = \frac{\sqrt{2-\sqrt{4-a^2}} \cdot \sqrt{2+\sqrt{4-a^2}}}{\sqrt{2+\sqrt{4-a^2}}}$$

Für den Zähler schreiben wir

$$\sqrt{(2-\sqrt{4-a^2}) \cdot (2+\sqrt{4-a^2})}$$

$$\curvearrowright \sqrt{4+2\sqrt{4-a^2}-2\sqrt{4-a^2}} \cdots$$

$$\cdots \sqrt{-(\sqrt{4-a^2})^2}$$

$$\curvearrowright \sqrt{4-(4-a^2)} = \sqrt{a^2} = a\,.$$

Wir ersetzen jetzt im PAP (Bild 2) die Gleichung zur Berechnung von a durch

$$a := \frac{a}{\sqrt{2+\sqrt{4-a^2}}}\,.$$

```
 1. N=       3
A= 1,00000000
π= 3,00000000

 2. N=       6
A= 0,51763809
π= 3,10582854

 3. N=      12
A= 0,26105238
π= 3,13262861

 4. N=      24
A= 0,13080626
π= 3,13935021

 5. N=      48
A= 0,06543817
π= 3,14103196

 6. N=      96
A= 0,03272346
π= 3,14145251

 7. N=     192
A= 0,01636228
π= 3,14155772

 8. N=     384
A= 0,00818121
π= 3,14158400

 9. N=     768
A= 0,00409061
π= 3,14159128

 10. N=    1536
A= 0,00204531
π= 3,14160254

 11. N=    3072
A= 0,00102266
π= 3,14160630

 12. N=    6144
A= 0,00051135
π= 3,14174147

 13. N=   12288
A= 0,00025570
π= 3,14198177

 14. N=   24576
A= 0,00012791
π= 3,14342216

 15. N=   49152
A= 0,00006411
π= 3,15109941

 16. N=   98304
A= 0,00003209
π= 3,15493054

 17. N=  196608
A= 0,00001673
π= 3,28988189

 18. N=  393216
A= 0,00000894
π= 3,51703082

 19. N=  786432
A= 0,00000548
π= 4,30746546

 20. N= 1572864
A= 0,00000548
π= 8,61493093

 21. N= 3145728
A= 0,00000548
π= 17,22986185

 22. N= 6291456
A= 0,00000548
π= 34,45972371

 23. N= 12582912
A= 0,00000548
π= 68,91944742
```

Bild 4. Ergebnisdruck des nichtkonvergierenden Rechenprogramms

Das auf diese Weise geänderte Rechenprogramm für den programmierbaren Tischrechner K 1003 zeigt Bild 5. Der Ergebnisdruck im Bild 6 macht deutlich, daß sich vom 13. Rechendurchlauf an der Wert π in den neun ausgedruckten Stellen nicht mehr ändert. Intern arbeitet der Rechner mit insgesamt 12 Stellen. Vergleicht man die entsprechenden Werte für a im nichtkonvergierenden (Bild 4) und im konvergierenden Fall (Bild 6), so kann a eindeutig als »Übeltäter« identifiziert werden.

Der Programmablaufplan (Bild 2) mit der umgestellten Gleichung zur Berechnung von a wurde auch für den Taschenrechner TI 58, allerdings ohne Drucker, programmiert. Das Programm (Bild 7) wird bei A gestartet und zeigt nur den jeweils errechneten Wert für π an. Bringt man den rechnerinternen Wert für π in das t-Register, so ist nach jedem Rechendurchlauf mit der Taste $x \rightleftarrows t$ ein Vergleich möglich. Das Programm wird dann wieder mit R/S gestartet. Vom 17. Rechendurchlauf an liefert der Rechner auf 10 Stellen genau den Wert, den er selbst über die Tasten 2nd π bereitstellt.

Für Kleincomputerfreunde soll das Programm noch in der Programmiersprache BASIC vorgestellt werden.

```
0000 MRK     0029 DRU     0058 TEX
0001 STM     0030   1     0059 ZS
0002 GL      0031   0     0060 2
0003 TXR     0032   .     0061 TXR
0004 MOD     0033         0062 MUL
0005 1       0034   N     0063 2
0006 TXR     0035   =     0064 TRX
0007 1       0036 DRU     0065 1
0008 MOD     0037   X     0066 X↑2
0009 3       0038   7     0067 1
0010 TXR     0039   0     0068 +/-
0011 2       0040 TEX     0069 MUL
0012 MRK     0041 TRX     0070 4
0013 1       0042 1       0071 ADD
0014 TRX     0043 TEX     0072 QWX
0015 1       0044   A     0073 2
0016 TRX     0045   =     0074 ADD
0017 2       0046 DRU     0075 QWX
0018 MUL     0047   9     0076 1/X
0019 TXR     0048   8     0077 TRX
0020 3       0049 TEX     0078 1
0021 MOD     0050 TRX     0079 MUL
0022 1       0051 3       0080 TXR
0023 TXR     0052 TEX     0081 1
0024 ADD     0053   π     0082 STM
0025 TRX     0054   =     0083 1
0026 2       0055 DRU     0084 END
0027 TRX     0056   9
0028 TEX     0057   8
```

Bild 5. Konvergierendes Rechenprogramm für den K 1003

```
 1. N=       3
A= 1,00000000
π= 3,00000000

 2. N=       6
A= 0,51763809
π= 3,10582854

 3. N=      12
A= 0,26105238
π= 3,13262861

 4. N=      24
A= 0,13080626
π= 3,13935020

 5. N=      48
A= 0,06543817
π= 3,14103195

 6. N=      96
A= 0,03272346
π= 3,14145247

 7. N=     192
A= 0,01636228
π= 3,14155761

 8. N=     384
A= 0,00818121
π= 3,14158389

 9. N=     768
A= 0,00409061
π= 3,14159046

 10. N=    1536
A= 0,00204531
π= 3,14159211

 11. N=    3072
A= 0,00102265
π= 3,14159252

 12. N=    6144
A= 0,00051133
π= 3,14159262

 13. N=   12288
A= 0,00025566
π= 3,14159265

 14. N=   24576
A= 0,00012783
π= 3,14159265

 15. N=   49152
A= 0,00006392
π= 3,14159265

 16. N=   98304
A= 0,00003196
π= 3,14159265

 17. N=  196608
A= 0,00001598
π= 3,14159265

 18. N=  393216
A= 0,00000799
π= 3,14159265

 19. N=  786432
A= 0,00000399
π= 3,14159265

 20. N= 1572864
A= 0,00000200
π= 3,14159265

 21. N= 3145728
A= 0,00000100
π= 3,14159265

 22. N= 6291456
A= 0,00000050
π= 3,14159265

 23. N= 12582912
A= 0,00000025
π= 3,14159265
```

Bild 6. Ergebnisdruck des konvergierenden Rechenprogramms

Grundlage ist wiederum der Programmablaufplan im Bild 2. Im BASIC-Programm (Bild 8) wurde auf rechnerspezifische Kürzungen (z.B. LET weglassen) bewußt verzichtet, um die Gültigkeit für möglichst viele Computertypen mit ihren BASIC-Dialekten zu gewährleisten. Aus diesem Grunde wurde auch auf »schmückendes

```
2nd Lbl    √x
    A      +
    1      2
    STO    =
    00     √x
    3      1/x
    STO    x ⇄
    1      RCL
2nd Lbl    0 ⇄
    B      = ⇄
    RCL    STO
    0      0
    x      B
    RCL
    1
    =
    R/S
    2
2nd Prd
    1
    RCL
    0
    x²
    +/-
    +
    4
    =
```

Bild 7. Konvergierendes Rechenprogramm für den TI 58/59

Beiwerk« (z.B. Quer- und Längsstriche, grafische Darstellung des jeweiligen n-Ecks) verzichtet. Hier bietet ein Kleincomputer mit Vollgrafik schöne Darstellungsmöglichkeiten.

Die Berechnung von a für den nichtkonvergierenden und den konvergierenden Fall wurde in Zeile 90 untergebracht. Diese Zeile läßt sich im Programm auf einfache Weise austauschen. Die meisten BASIC-Interpreter geben 8 oder 6 signifikante Stellen aus. Intern wird dann mit einer Genauigkeit von 9 bis 10 oder 7 bis 8 Stellen gerechnet. Das ist viel weniger als bei den vorhin erwähnten Rechnern K 1003 und TI 58. Aus diesem Grunde wird die uner-

```
10 REM BERECHNUNG VON PI
20 PRINT"NR.";TAB(5);"N";TAB(15);"A";TAB(27);"PI"
30 PRINT
40 LET A=1: LET N=3
50 FOR I=1 TO 19
60 PRINT I; TAB (4); N; TAB (12); A;
70 LET U 2=A*N: PRINT TAB (25); U 2
80 LET N=2*N
90 LET A=SQR (2—SQR (4—A ↑ 2))
100 NEXT I
110 END
```

für konvergierenden Fall Zeile 90 ersetzen durch:

```
90 LET A=A/SQR (2+SQR (4—A ↑ 2))
```

Bild 8. BASIC-Programm zur Berechnung von $\frac{u}{2} \approx \pi$

wünschte Abweichung beim nichtkonvergierenden Fall mit einem 8-Stellen-Interpreter schon nach dem 8. Durchlauf sichtbar. Beim konvergierenden Fall ändert sich bereits nach dem 12. Durchlauf der Wert für π mit 8 Stellen nicht mehr.

ARCHIMEDES von Syrakus fand mit seiner Methode, die wir hier nachvollzogen haben, einen Näherungswert für π zwischen 3,1408 und 3,1428. Andere Quellen geben eine weitaus bessere Näherung zwischen 3,14159 und 3,14160 an. Unabhängig von seiner erreichten Genauigkeit steht fest, daß Grundgedanke und Methode von ihm stammen und damit über 2000 Jahre alt sind. Ein Rechner, damals wie heute eingesetzt, hätte die Geistesarbeit nicht abgeschafft, sondern nur Schnelligkeit und Genauigkeit gefördert. Eine gedankenlose Anwendung hätte damals wie heute zu bösen Überraschungen geführt. Hätte ARCHIMEDES einen Taschenrechner gehabt, er hätte das gleiche und sicherlich noch viel, viel mehr geschaffen.

Autor:

Dr. Hannes Gutzer
Wissenschaftlicher Mitarbeiter an der Akademie der pädagogischen Wissenschaften Forschungszentrum Halle-Neustadt

Rechentechnische Begriffe für den Laien erklärt

FF Formularwechsel (FORM FEED). Der Formularwechsel bewirkt einen Übergang auf den nächsten Blattanfang.

HT Horizontaler Tabulatorsprung (HORIZONTAL TABULATOR). Das Steuerzeichen HT bewirkt den Übergang zu einer vorgegebenen Ausgabeposition. Das Steuerzeichen wird in der Regel bei der Ein- und Ausgabe von Tabellen benutzt. Sind mehrere Tabulatorpositionen vorgegeben, so wird in Abhängigkeit von der aktuellen Position des Druckkopfes oder Cursors die nächst mögliche Tabulatorposition angewählt.

Den Speicherinhalt retten – aber wie?

Der Besitzer eines Kleincomputers hat sehr bald das Interesse, die von ihm eingegebenen Programme auf einem Massenspeicher zu sichern. Dazu steht ihm zumeist ein Anschluß für Kassettenrecorder zur Verfügung, und das Betriebssystem des Rechners bietet die entsprechenden Leistungen an. Dabei werden die Daten bei der Aufzeichnung meist gepackt, also mit 8 Bit Daten je aufgezeichnetes Wort, und kaum strukturiert abgelegt, was den Nachteil bringt, daß sich bei auftretenden Fehlern diese schwer lokalisieren und korrigieren lassen. Die dafür notwendigen Programme sind jedoch relativ kurz, und man nutzt die Aufzeichnungsdichte des Bandgerätes fast vollständig aus. Um die Datensicherheit bei Maschinencodeaufzeichnungen zu erhöhen, hat sich international das für Lochbänder von Intel eingesetzte Format durchgesetzt, welches man auch für Magnetbandaufzeichnungen verwenden kann. Der Nachteil dieses Formates besteht darin, daß sich die aufgezeichnete Datenmenge gegenüber der aus dem Speicher zu rettenden etwa verdoppelt bis verdreifacht, was sich auch in den Ladezeiten bemerkbar macht. Dieses Format hat allerdings auch einige Vorteile. So erfolgt die Aufzeichnung im ASCII-Code, das heißt in lesbarem Text. Daraus ergibt sich die Möglichkeit, eine fehlerhafte Aufzeichnung als Text in den Speicher zu lesen und mit einem Editierprogramm den Fehler zu suchen und zu beseitigen. Außerdem werden zu diesem Format von den 128 möglichen Zeichen nur 20 genutzt, so daß bei Bitfehlern das Zeichen meistens rekonstruierbar ist. Zum anderen ist in dem verwendeten Blockaufbau – außer der Partitätsprüfung innerhalb jedes einzelnen Zeichens – auch eine Prüfsummenbildung über den ganzen Block enthalten. Damit besteht schon beim Laden die Möglichkeit, Fehler auf Blockgröße genau einzugrenzen.
Nun zum praktischen Teil, dem Blockaufbau, selbst.

Man unterscheidet im wesentlichen drei, prinzipiell gleich aufgebaute Maschinencodeformate. Das sind:

a) der absolute Maschinencode (MC), bei dem die Adresse, wo das Programm abzulegen ist, fest vereinbart ist;
b) der relative Maschinencode (Reloc-MC), bei dem während des Ladevorgangs die eigentliche Lage im Speicher erst errechnet wird, und
c) der Programmverbinder-Maschinencode (Linking-MC), der undefinierte Adressen enthält und erst durch das Laden mehrerer MC-Programme ein vollständiges Programm ergibt.

Hier von Interesse sind nur a) und b), da a) dem üblichen Aufbau von MC-

Lochbändern entspricht und in der internationalen Literatur oft Programme als Reloc-Textliste wiedergegeben werden.
Der Maschinencode im Intel-Format hat folgenden allgemeinen Aufbau, wobei für jedes Druckzeichen ein Byte im Block zu setzen ist:

S BB AAAA RR VV HL HL ···PS

Beim absoluten MC haben die Zeichen folgende Bedeutung:

S Blockanfangszeichen (Doppelpunkt)
BB Anzahl der Datenbytes (Paare HL und VV)
AAAA Adresse des ersten Datenbytes
RR Relocalisierungsinformation – hier immer ØØ
VV entfällt
H Bit 4 ···7 des Datenbytes (als ASCII-Zeichen Ø ···9, A ···F)
L Bit 0 ···3 des Datenbytes (als ASCII-Zeichen Ø ···9, A ···F)
PS Prüfsumme aller Bytes ab BB bis einschließlich PS muß ØØ ergeben.

Zwischen der Prüfsumme und dem nächsten Blockanfangszeichen können beliebige Zeichen stehen. Üblich ist eine Zeilenschaltung, um im Fall der Bearbeitung mit einem Textsystem keine endlose Zeichenkette zu haben.
Beim Reloc-MC ändert sich der Aufbau wie folgt:

S Blockanfangszeichen (Semikolon)
RR Reloc-Information = Ø1, daraus folgt, daß die Adresse AAAA zu modifizieren ist.
VV Vektor für im Maschinencode zu modifizierende Bytes innerhalb der nächsten 8 Byte. Bit 7 ist für das folgende Byte zuständig und Bit Ø für das letzte der 8 Byte. Ein gesetztes Bit in VV bedeutet eine Modifikation des Bytes. Ist das darauf folgende Bit Ø, so handelt es sich um eine Adresse, und es sind beide Bytes zu modifizieren. Sind zwei aufeinander folgende Bits gesetzt, so ergibt sich eine Aussage für das Linking-Format.

Die anderen Zeichen bleiben in ihrer Bedeutung unverändert. Das Byte VV kehrt bei diesem Format nach jeweils 8 Byte wieder. Auch ist hier zu beachten, daß die Blocklänge nicht mehr so gleichbleibend wie beim absoluten MC ist, da man Adressen nicht auf zwei MC-Blöcke aufteilen kann.
Es gibt noch drei Sonderfälle für den Maschinencode.

Der eine ist ein Block mit der Länge (BB) gleich Null, was das Ende des Programms kennzeichnet. Bei diesem Block ist es noch möglich, den Wert von RR auf eins zu setzen. In diesem Fall wird die Adresse als Startadresse gedeutet, und der Lader startet das Programm automatisch.
Die anderen beiden Fälle betreffen die Blockanfangszeichen »Ausrufezeichen« und »negativer Schrägstrich«. Diese Blöcke enthalten Informationen für das Linking-Format.
Abschließend noch ein Beispiel für einen Reloc-MC.
Ausgangspunkt sei folgendes kleines Programm:

```
                        PN    BSP
ØØØØ  3E 1E             LD    A, 1EH
0002  77          M1:   LD    M, A
0003  23                INC   HL
0004  C3 02 ØØ          JMP   M1
0007                    END
```

Welches den folgenden Maschinencode ergäbe

```
; Ø8 ØØØØ Ø1 04 3E 1E 77 23 C3 Ø2 ØØ38
; ØØ ØØØØ ØØ
```

Literatur

[1] KLEIN, R. D.: Mikrocomputer Hard- und Softwarepraxis. – München: Franzis-verlag, 1981. – S. 140f.

Autor:
Dr.-Ing. Gert Schönfelder

Problemanalytiker im Rechenzentrum der Sektion Informationsverarbeitung der Ingenieurhochschule Dresden

Rechentechnische Begriffe für den Laien erklärt

Hardware Aus dem Englischen stammende Bezeichnung für die Gesamtheit aller technischen Einheiten und Geräte, die zur Ausstattung eines Computers erforderlich sind. Dazu gehören die Chips, die Schaltkreisplatinen, das Display, die Speicher, Kassetten, Disketten u.a.m.

Software Gesamtheit der zu einem Computer gehörenden Programme, die die Arbeit mit dem Computer ermöglichen, erweitern bzw. vereinfachen. Dazu gehören u.a. die Betriebssysteme, die Assembler, die Sprachcompiler bzw. -interpreter u.a.m.
Die von den Nutzern eines Computers für spezielle Anwendungszwecke selbst erarbeiteten Programme werden auch als *Anwendersoftware* bezeichnet.

Bug (engl.) die Wanze.
Man versteht darunter einen Fehler in einem Programm.

LIFO Last in – first out. Arbeitsprinzip eines Stackspeichers (Kellerspeicher). Es kann verglichen werden mit dem Aufsuchen von Gegenständen in einem Schubkasten. Um zu den weiter unten liegenden Gegenständen gelangen zu können, müssen zuerst diejenigen Gegenstände entnommen werden, die zuletzt obenauf in den Schubkasten gelegt worden sind.

ASCII Amerikanischer Standardkode für den Informationsaustausch (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)
Bei Kleincomputern häufig verwendeter Kode für die Speicherung und Übertragung von Informationen. Die meisten peripheren Geräte benutzen den ASCII-Kode für die Ein- und Ausgabe von Informationen. Der ASCII-Kode verwendet für die Darstellung eines Zeichens 7 Bit. Damit können insgesamt 128 Zeichen kodiert werden, die sich wie folgt aufschlüsseln:
32 Steuerzeichen (Zeilenvorschub, Blattvorschub usw.)
10 Ziffernzeichen 0 · · · 9
26 lateinische Großbuchstaben A · · · Z
26 lateinische Kleinbuchstaben a · · · z
32 Sonderzeichen, wie ! " % & ' ( ) + — , . / ; : = ? usw.
1 Leerzeichen (Space oder blank)
1 Korrekturzeichen (Delete).

BASIC-Tricks und -Kniffe

Es ist schon erstaunlich, wie unüberlegt oder computergläubig eine unnötige Anzahl von Nachkommastellen bei Ergebnissen aus BASIC-Programmen akzeptiert wird. So gibt der 6stellige BASIC-Interpreter für die Aufprallgeschwindigkeit der Mondlandefähre zum Beispiel 8.23498km/h aus. Ein achtstelliger Interpreter liefert noch zwei unnütze Stellen mehr. Da viele BASIC-Interpreter für Kleincomputer nicht die WRITE-FORMAT-Anweisung und nicht die PRINT-USING-Anweisung kennen, müssen wir uns selbst etwas einfallen lassen. Das Runden der Ergebnisse auf eine sinnvolle Anzahl von Kommastellen erledigen wir mit folgendem Mini-Unterprogramm:

```
1000 REM UP-RUNDEN
1010 LET KO = 2
1020 LET A = INT
     (A * 10 ↑ KO + 0.5)/10 ↑ KO
1030 RETURN
```

Wir haben hier als Beispiel für die Anzahl der Kommastellen KO = 2 gewählt. Die zu rundende Zahlenvariable heißt A.

Dieses Runden ersetzt aber noch nicht vollständig eine PRINT-USING-Anweisung zur formatierten Darstellung von Zahlen. Wollen wir mit einem BASIC-Interpreter für Kleincomputer Zahlenwerte ordentlich Komma unter Komma anordnen, so zeigt er uns die kalte Schulter. Besser gesagt die linke, denn er gibt jeden Zahlenwert, unabhängig von der Stellenzahl, linksbündig aus. Deshalb hier die folgende PRINT-Anweisung mit Trick zur Ausgabe des Inhaltes von A:

```
PRINT TAB
(7-LEN(STR$(INT(A)))); A
```

Bei einem sechsstelligen Interpreter werden hier alle Zahlen, die keine Angabe eines Exponenten erfordern, kommabündig untereinander geschrieben. Eine Ausnahme bildet der Fall, daß der eine oder andere Interpreter beispielsweise für

```
LEN(STR$(INT(0.06)))
```

eine 1 liefert, aber nur .06 auf dem Bildschirm ausgibt. Für einen achtstelligen Interpreter muß die 7 durch eine 9 ersetzt werden. Natürlich können auch größere Zahlen gewählt werden, um die Zahlenkolonne an der gewünschten Stelle des Bildschirms zu positionieren. Bei einem »stattlichen« Interpreter, z. B. dem BASIC-1520 für die Bürocomputer A 5120/5130, würde man hierzu die PRINT-USING-Anweisung und die SPC- oder SPACE$-Funktion verwenden.

Dr. Hannes Gutzer

Berechnung des Molekulargewichts einer chemischen Verbindung – ein BASIC-Programm für die Schule

Mit diesem Beitrag sollen Anregungen für die Nutzung von Mikro- oder Kleinstrechnern in der Schule gegeben werden.
Bei der Erarbeitung des Programms wurden folgende Punkte berücksichtigt:

1. Der verwendete Befehlsumfang entspricht dem des Kleincomputers KC 85/1
2. Die LET-Anweisung zur Belegung der Variablen wurde immer verwendet, obwohl sie beim KC 85/1 entfallen kann. Dadurch ist das vorgestellte Programm mit relativ geringem Aufwand auch auf andere Rechnertypen übertragbar.
3. Aus Gründen der Übersichtlichkeit des Programms wurde nur ein Befehl pro Zeile angegeben.
4. Die Etappen der Programmerstellung (Problemanalyse, Aufstellen des Algorithmus, Erarbeitung des BASIC-Programms) werden relativ ausführlich beschrieben, so daß auch Leser, die noch keine Erfahrung in der Programmierung von Kleincomputern haben, das Programm verstehen und die verwendeten Programmiertechniken auch für ihre eigenen Programme verwenden können.

Programm zur Berechnung des Molekulargewichts einer chemischen Verbindung (Programm »MG«)

Problemanalyse

Das Molekulargewicht einer beliebigen chemischen Verbindung setzt sich aus den relativen Atommassen der enthaltenen Elemente unter Berücksichtigung der Häufigkeit der einzelnen Elemente in der untersuchten Verbindung

zusammen. Als Beispiel sei das Molekulargewicht von Wasser betrachtet:

H_2O
2 Atome Wasserstoff $= 2 \times 1{,}008$
$+$ 1 Atom Sauerstoff $= 15{,}999$
MG (H_2O) $= 18{,}015$

Damit beträgt das Molekulargewicht von Wasser 18,015.
Das von uns aufzustellende BASIC-Programm für den Kleincomputer sollte folgende Eigenschaften besitzen:

1. Die Symbole der Elemente und ihre relativen Atommassen sind im Rechner gespeichert.
2. Die Elemente und ihre Häufigkeit in einer Verbindung werden in einer Programmschleife nacheinander eingegeben.

3. Der Computer »sucht sich« entsprechend dem Symbol des Elements die richtige Atommasse aus den gespeicherten Daten heraus und berechnet das Molekulargewicht der analysierten Verbindung.

Algorithmus und Programm »MG«

Das Blockschema des Programms »MG« ist in Bild 1 gezeigt. Dieses Schema wird im folgenden an Hand der Programmbeschreibung erklärt.
Nach dem Start des Programms werden die Daten in die entsprechenden Datenfelder eingelesen, wobei die Symbole der Elemente in dem String-Variablenfeld M$ und die relativen Atommassen in dem Feld M gespeichert werden. In Bild 2 ist die Befehlsfolge gezeigt.

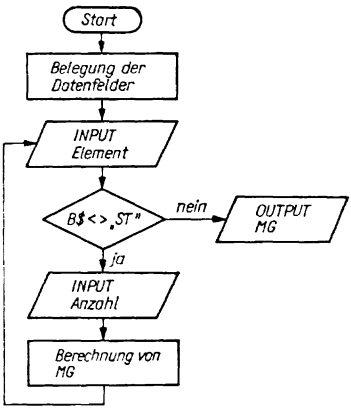


Bild 1. Blockschema des Programms »MG«

```
10 REM PROGRAMM "MG"
15 LET N=20
20 DIM M$(N)
25 DIM M(N)
30 FOR I=1 TO N
35 READ M$(I),M(I)
40 PRINT M$(I),M(I)
45 NEXT I
```

Bild 2

In der vorliegenden Variante des Programms »MG« sind 20 Elemente gespeichert. Das wird deutlich durch die Programmzeile 15.

15 LET N = 20

Es ist jedoch durchaus möglich, das gesamte Periodensystem abzuspeichern. In diesem Falle müßte die Zeile 15 lauten:

15 LET N = 103

Durch die Programmzeile 40 werden die durch die READ-Anweisung eingelesenen Daten auf dem Bildschirm des Fernsehgerätes angezeigt. In Tabelle 1 sind die gespeicherten Elemente und die relativen Atommassen zusammengestellt.

Tabelle 1. Daten der im Programm »MG« gespeicherten Elemente

Element	relative Atommasse
H	1,008
C	12,01
N	14,007
O	15,999
F	18,998
Na	22,989
Mg	24,31
P	30,97
S	32,06
Cl	35,45
K	39,10
Ca	40,08
Fe	55,85
Cu	63,54
Zn	65,37
Br	79,91
Ag	107,87
Au	196,97
Hg	200,59
Pb	207,19

Aus Bild 3 ist das eigentliche Programm zur Berechnung der Molekulargewichte ersichtlich.
In dem Programm kennzeichnet die Zeile 115

115 IF B$ = "ST" THEN GOTO 150

```
100 REM HAUPTPROGRAMM "MG"
105 LET MG=0
110 INPUT"ATOM: ";B$
115 IF B$="ST" THEN GOTO 150
120 INPUT"ANZAHL: ";M
130 FOR I=1 TO N
135 IF B$=M$(I) THEN LET MG=MG+M*M(I)
140 NEXT I
145 GOTO 110
150 PRINT"MG=",MG
155 PAUSE
160 GOTO 105
```

Bild 3

die Abbruchbedingung, d.h., nach Eingabe sämtlicher Elemente erfolgt dann die Ausgabe des berechneten Molekulargewichts, wenn die String-Variable B$, die sonst das Symbol des jeweiligen Elements enthält, mit »ST« belegt ist.
In der Programmschleife 130—140 wird das Symbol des eingegebenen Elements mit den in M$ gespeicherten Symbolen verglichen und das Molekulargewicht als Zwischenergebnis berechnet, wenn B$ = M$ (I) ist. Das berechnete Molekulargewicht wird durch die Programmzeile 150 angezeigt, die Zeile 155 bewirkt, daß das Programm anhält und durch »CONT« erneut gestartet werden kann.
In den Programmzeilen 200—215 (Bild 4) sind die Daten, die durch die READ-Anweisung in Zeile 35 eingelesen werden, abgespeichert.

```
200 DATA "H", 1.008, "C", 12.01, "N", 14.007
"O", 15.999, "F", 18.998
205 DATA "Na", 22.898, "Mg", 24.31, "P", 30.97,
"S", 32.06, "Cl", 35.45
210 DATA "K", 39.10, "Ca", 40.08, "Fe", 55.85,
"Cu", 63.54, "Zn", 65.37
215 DATA "Br", 79.91, "Ag", 107.87, "Au", 196.97,
"Hg", 200.59, "Pb", 207.19
```

Bild 4

Nach erfolgter Programmierung des Kleincomputers und Abspeicherung des Programms auf der Kassette kann das Programm durch »RUN« gestartet werden. Als Beispiel soll das Molekulargewicht der **Phosphorsäure** berechnet werden. Auf dem Bildschirm erscheint folgende Anzeige:

ATOM:
H
ANZAHL:
3
ATOM:
P
ANZAHL:
1
ATOM:
O
ANZAHL:
4
ATOM:
ST
MG = 97.9915

Zum Abschluß dieses Programms sollen die verwendeten Variablen und ihre Belegung zusammengestellt werden:

Variable	Belegung
N	Anzahl der gespeicherten Elemente
M$ (N)	Symbole der Elemente
M (N)	relative Atommassen
MG	berechnetes Molekulargewicht und Zwischenergebnis bei der Molekulargewichtsberechnung
B$	Symbol des eingegebenen Elements bzw. »ST« als Abbruchgröße
I	Schleifenzähler

Autor:
Dr. Jochen Heins

Wissenschaftlicher Mitarbeiter an der
Martin-Luther-Universität Halle
Sektion Biowissenschaften

Berichtigungen zu Heft 3

Beitrag „Eine Mondlandung mit dem K 1003“

Im Beitrag wurde die Wirkungsrichtung der Beschleunigung a mit falschem Vorzeichen angegeben. Aus diesem Grunde ist folgendes richtigzustellen:

- Gl. (1) lautet richtig:

$$h_t = h_0 - v_0 \cdot t - \frac{a}{2} \cdot t^2$$

- Gl. (5) lautet richtig:

$$h_t = h_0 - v_0 - \left(\frac{1{,}62 - T_{ein}}{2}\right)$$

- Gl. (7) lautet richtig:

$$h_t = h_0 - \frac{v_0}{3{,}6} - 0{,}81 + \frac{T_{ein}}{2}$$

Diese Gleichung (7) ist auch im Programmablaufplan und im Programm zu ändern. Die richtigen Befehle im Programm lauten: 0407 SUB und 0413 ADD. Auch zur Berechnung der Aufprallgeschwindigkeit v_A auf Seite 51 müssen die Vorzeichen entsprechend vertauscht werden. Wir wollen das hier nicht tun, da unser Leser Herr GEROLD GÜNTNER aus Rostock eine elegante und einfachere Ableitung gefunden hat. Seine Gleichung lautet:

$$v_A = \sqrt{v_0^2 + 25{,}92 \cdot h_0 \cdot (1{,}62 - T_{ein})}.$$

Das entsprechende Programm hat er auf den Programmzeilen 0446 bis 0469 untergebracht (Bild 1). Die Programmzeilen können so direkt überschrieben werden. Allerdings sind dann noch die Programmzeilen 0470 bis 0521 »kampfunfähig« zu machen. Dazu kann auf diese Zeilen zum Beispiel die Taste M (der Drucker gibt dann MOD aus) gebracht werden.
Wir bitten, den Fehler zu entschuldigen, und hoffen auf weitere Leserideen.

Dr. Hannes Gutzer

```
0446 MOD      0458 5
0447 1        0459 DP
0448 DP       0460 9
0449 6        0461 2
0450 2        0462 MUL
0451 TRX      0463 TRX
0452 4        0464 2
0453 SUB      0465 X↑2
0454 TRX      0466 ADD
0455 1        0467 CHX
0456 MUL      0468 KNO
0457 2        0469 KNO
```

Bild 1. Korrektur der Zeilen 446—469

Beitrag „Polycomputer 880“

Im Stromlaufplan auf S. 28 sind folgende Korrekturen vorzunehmen:

1. Der Ausgang des D 120 ist mit $\overline{E2}$ des DS 8205 und $\overline{CS}$ der beiden DS 8216 zu verbinden.
2. Der Ausgang 0Ø des DS 8205 ist mit $\overline{CS}$ der beiden ersten RAM (von links gesehen) zu verbinden.
3. Am Ausgang des D 120 und des Gatters am WRB-Signal sind die Negationspunkte nachzutragen.

Dr. Uwe Hübner

ISBN 3-343-00128-7

1. Auflage
Lizenznummer 114-210/107/86
LSV 1083
Verlagslektor: Helga Fago
Gestaltung: Lothar Gabler
Printed in GDR
Satz und Druck:
Messedruck Leipzig, Bereich Borsdorf
III-18-328
Redaktionsschluß: 15. 5. 1986
Bestellnummer: 547 011 0
00780

Kleinstrechner-TIPS / Hrsg. von Hans Kreul u. a. - Leipzig: Fachbuchverl., H. 4. - 1. Aufl. - 1986. - 64 S. : 33 Bild.

Anschrift des Verlages:
VEB Fachbuchverlag
PSF 67
DDR – Leipzig
7031

Vorschau auf die nächsten Hefte

Schilling: Ein Formalismus zur Beschreibung von Problemlösungen

Hübner: Spracheingabe für den Mikrocomputer

Girlich: Iterationen und der Feigenbaum

Michel: Wieso rechnet (m)ein Taschenrechner $\sqrt{8} \times \sqrt{8} = 8$?

Schönfelder: Computerspiele — mehr als eine Spielerei

Schönfelder: Master Mind — gegen den Rechner gespielt

Die Broschürenreihe

KLEINSTRECHNER-TIPS

behandelt

- Tendenzen und Theorien
- Informationen und Ideen
- Programme und Projekte
- Spaß und Spiel

und stellt sich das Ziel

- den Nutzer der Mikrorechentechnik aus allen Bereichen der Volkswirtschaft und dem Bildungswesen bei der Einarbeitung in die Informatik und Computertechnik zu unterstützen
- Entwicklungstendenzen der Informatik und Computertechnik vorzustellen und zur Erweiterung des Grundwissens beizutragen
- Anregungen für den Computereinsatz zu geben und Beispielprogramme für Kleincomputer zu veröffentlichen

um somit einem großen Kreis von Freunden der Informatik und Computertechnik zu helfen, sich moderner Hilfsmittel und Methoden zu bedienen.

ISBN 3-343-00128-7