Institut für Rationalisierung
der Elektrotechnik/Elektronik
Zentralstelle
für Aus- und Weiterbildung
des Industriebereiches
Elektrotechnik/Elektronik

Lehrmaterial für die
Weiterbildung

Informationsverarbeitung mit Kleincomputern

1

Grundlagen

Kreutzer

Elektrotechnik
Elektronik
Automation
DDR

Steffen Kreutzer

Informationsverarbeitung mit Kleincomputern

1

Grundlagen

Herausgeber
Institut für Rationalisierung der Elektrotechnik/Elektronik
Institutsteil Dresden
Zentralstelle für Aus- und Weiterbildung
des Industriebereiches
Elektrotechnik/Elektronik
8080 Dresden
Karl-Marx-Straße

Autor: FSD Dipl.-Ing. Steffen Kreutzer
Ingenieurschule für Wissenschaftlichen Gerätebau
"Carl Zeiss" Jena

Bearbeiter: Dipl.-Gwl. Heinz Rüdger
Institut für Rationalisierung der
Elektrotechnik/Elektronik
Zentralstelle für Aus- und Weiterbildung

1. Auflage
IR Dresden ZSB
Druckgenehmigungs-Nr.: Ag 682/037 /86 - H.417-3.0
Druck und Herstellung: NOWA DOBA Bautzen
Redaktionsschluß: 28.02.1986
Bestell-Nr.: T.2.04.0001

Vorwort

Die sich unter Führung der SED in der Volkswirtschaft der DDR immer stärker und breiter vollziehende Entwicklung der Mikroelektronik, ihre planmäßige und durchgängige Umsetzung in allen Bereichen führte und führt besonders auch bei der Anwendung der Rechentechnik zu neuen, qualitativ höheren Anforderungen an die Techniker und Ingenieure. Die Entwicklung der Mikrorechentechnik gestattet, den Rechnereinsatz stark zu dezentralisieren. Charakteristisch ist für größere volkswirtschaftliche Einheiten, wie zum Beispiel für Kombinate, die Strukturierung der eingesetzten Rechentechnik zur Lösung von ökonomischen, wissenschaftlich-technischen und leitungsmäßigen Aufgabenstellungen in bezug auf die Einsatzebene. Im wesentlichen wird folgende Struktur angestrebt:

- Kombinatsebene: Einsatz von Großrechnern des ESER (Einheitliches System Elektronischer Rechner der sozialistischen Staaten).
- Betriebsebene: Einsatz von mittleren Rechnern des SKR (System der Klein-Rechner).
- Arbeitsplatzebene: Einsatz von Bürocomputern und spezifisch aus Moduln eines Mikrorechnersystems aufgebauten Rechnerarbeitsplätzen.

Am Arbeitsplatz wird der Rechner unmittelbarer Partner des Technikers und Ingenieurs zur Bewältigung der Routineprozesse. Die arbeitsplatzgebundene Rechentechnik sichert bei ihrem Einsatz das Arbeiten nach einheitlichen Verfahren, ein weitgehend objektives Bewerten von Ergebnissen durch Orientierung an einheitlichen Maßstäben und unterstützt die variantenreiche Suche nach der optimalen Lösung.

Die neuen, qualitativ höheren Anforderungen an den Techniker und Ingenieur erwachsen aus dieser spezifischen Anwendung des Rechners am Arbeitsplatz. Es sind dies Forderungen an das Grundwissen und -können, an die grundlegenden Fähigkeiten und Fertigkeiten zur

- Algorithmierung technisch-technologischer Prozesse
- Programmierung anwendungsspezifischer Algorithmen
- Beherrschung des Dialogverkehrs.

Anliegen dieser Lehrmaterialreihe ist es, eine Einführung in die praktische Anwendung der Arbeitsplatzrechentechnik zu geben und den Umgang mit dieser Technik zur besseren Beherrschung der Routineprozesse in den fachrichtungsspezifischen Aufgabenstellungen vorzubereiten.

Im 1. Teil - Grundlagen - werden die für die Informationsverarbeitung wichtigen Begriffe, wie

Physikalische Größe, Information, Signal, Kode usw.

erläutert, wesentliche Grundlagen der digitalen Signalverarbeitung modellmäßig dargestellt sowie Daten und Datenstrukturen behandelt.

Im 2. Teil - Algorithmierung - steht die schöpferische Bearbeitung einer gegebenen Aufgabenstellung im Mittelpunkt, so daß die Lösung mit einem Digitalrechner erfolgen kann.
Schwerpunkte sind:

- Aufbau und Wirkungsweise von Digitalrechnern
- Darstellung von Algorithmen
- Testen von Algorithmen
- Erstellen von Algorithmen.

Bei der Darstellung von Algorithmen wird vorzugsweise als grafische Form der Programmablaufplan und als sprachliche Form die Notierung in der problemorientierten Programmiersprache BASIC verwendet.

Im 3. Teil - Programmiersprache BASIC - werden die Elemente der Programmiersprache systematisch dargestellt und an einfachen Beispielen erläutert.

Im 4. Teil - Programmierung mit BASIC - wird die Handhabung der Programmiersprache an komplexeren Beispielen gezeigt und einfache Übungen am Rechnerarbeitsplatz vorgestellt.

Die Gestaltung der Reihe erfolgt so, daß jeder Teil in sich abgeschlossen dargestellt ist. Der praktische Gebrauch der Reihe wird notwendige Veränderungen und Ergänzungen deutlich machen. Der Verfasser bittet dazu um einen regen Erfahrungsaustausch.

Inhaltsverzeichnis

		Seite
	Verzeichnis der Abkürzungen und Formelzeichen	6
1.	Einführung	7
2.	Ausgewählte Grundlagen der Informationsverarbeitung	10
2.1.	Physikalische Größe - Information - Signal	10
2.1.1.	Allgemeines	10
2.1.2.	Information	11
2.1.3.	Signale	16
2.1.3.1.	Allgemeines	16
2.1.3.2.	Klassifizierung	16
2.1.3.3.	Binäre Signale	19
2.1.3.4.	Digitale Signale	24
2.2.	Kode - Kodierung - Dekodierung	27
2.2.1.	Allgemeines	27
2.2.2.	Numerische Kode	28
2.2.3.	Alphanumerische Kode	33
2.3.	Zahlen und das Prinzip ihrer Darstellung	35
2.3.1.	BCD-kodierte Dezimalzahlen	35
2.3.2.	Direkt dual kodierte Dezimalzahlen	36
2.3.2.1.	Dezimalsystem	37
2.3.2.2.	Dualsystem	38
2.3.2.3.	Konvertierung	39
2.3.2.4.	Vereinfachte Schreibweise für Dualzahlen im Hexadezimalsystem	40
2.3.3.	Grundschaltungen zur Ablage und Verarbeitung von Zahlen im Rechner	43
2.3.3.1.	Register	44
2.3.3.2.	Adder	50
2.3.3.3.	Addierwerk für Tetraden	54
3.	Sprachen und grundlegende Begriffe der digitalen Rechentechnik	56
3.1.	Mensch - Rechner - Kommunikation	56
3.2.	Daten	60
	Lösungen der Aufgaben	64
	Literaturverzeichnis	66

Verzeichnis der Abkürzungen und Formelzeichen

Abkürzungen

A	Aufgaben
K	Kontrollfragen

Formelzeichen

C	Trigger-(Takt-) Signal
D_i	Dynamischer Triggereingang
H	Informationsgehalt
H_{max}	Maximaler Informationsgehalt
I	Informationsparameter
p	Wahrscheinlichkeit eines Ereignisses
Q_i	Triggerausgang
R_i	Triggerlöscheingang
S_i	Triggersetzeingang, Summenausgang beim Adder
t	Zeit
U	Spannungsamplitude
$\ddot{U}_i$	Übertrag beim Adder
x_i	Binäre Eingangssignale
y_i	Binäre Ausgangssignale

1. Einführung

Im Wissenschaftlichen Gerätebau begegnet dem Techniker und Ingenieur die dezentrale Mikrorechentechnik in Form der rechnergestützten Arbeitsplätze (Bild 1) sowie in Geräten und Anlagen des Gerätebaus (Bild 2).

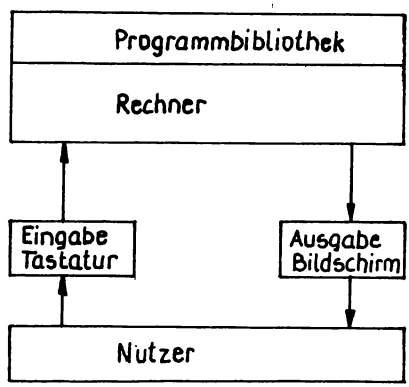


Bild 1: Nutzer-Rechner-Verkehr am rechnergestützten Arbeitsplatz

Das Arbeitsprinzip im Umgang des Technikers und Ingenieurs mit dem Rechnerarbeitsplatz beruht darauf, daß der Rechner unmittelbarer "Partner" ist. In einer speziellen Programmbibliothek stehen programmierte Lösungsalgorithmen für häufig benötigte Aufgabenklassen zur Verfügung. Der Nutzer "sitzt" am Rechner (Sitzung) und "wählt" sein "Menü".

Die Nutzung, ständige Aktualisierung und Erweiterung der bereitgestellten Bibliothek fordert

- Fähigkeiten zur systematischen Problemlösungsfindung
- Fertigkeiten im Dialogbetrieb
- Kenntnisse über die Bibliotheksstruktur
- Fertigkeiten zur Algorithmierung ingenieurtechnischer Aufgabenstellungen
- Fertigkeiten zur Rechnerprogrammierung.

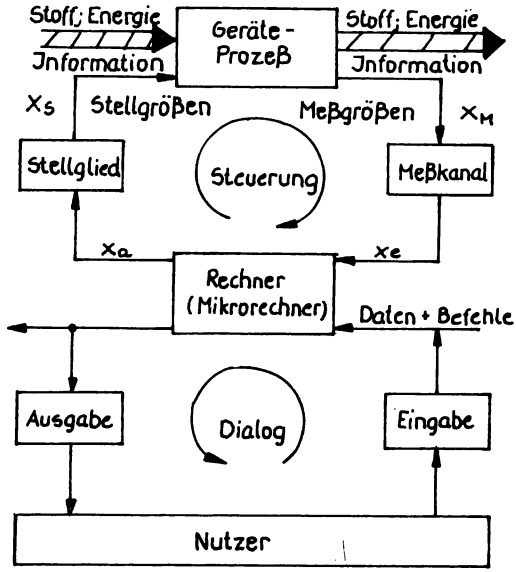


Bild 2: Informationsverarbeitung in einem Gerät des Wissenschaftlichen Gerätebaus

Die Kopplung der arbeitsplatzgebundenen Rechentechnik mit den Klein-, Mittel- und Großrechnern der Betriebs- und Kombinatsebene erfordert darüber hinaus die Beachtung der Paßfähigkeit (Kompatibilität) der Lösungen zum übergeordneten System. Im wesentlichen sind dazu Kenntnisse über den konkreten Dateiaufbau notwendig und deren Folgen für die Gestaltung der Such- und Sortierprogramme.

Zur Informationsverarbeitung werden Digitalrechner zur Lösung von zwei Aufgaben eingesetzt. Zum ersten steuert der Mikrorechner wesentliche Teile des Stoff-, Energie- oder Informationsflusses im Prozeß. Zum zweiten übernimmt der Rechner (häufig auch ein Kleinsteuerrechner, der mit dem Mikrorechner des Steuerkreises gekoppelt ist) die Gestaltung des Dialogverkehrs zwischen Nutzer und Gesamtgerät. Typische Aufgaben sind

- Überprüfung und Signalisierung der Arbeitsfähigkeit des Gerätes
- Unterstützung bei der Fehlersuche
- Organisation der gewählten Betriebsart
- Korrektur, Verdichtung und Aufbereitung der Ergebnis- und Prozeßdaten.

Die Analyse der notwendigen Forderungen an die Befähigung der Techniker und Ingenieure führt, sieht man von den Fragen und Problemen der Hardware ab, auf die grundlegenden Forderungen zur Realisierung des Arbeitsprinzips mit einem Rechnerarbeitsplatz.

Die Erfahrungen lehren, daß der Prozeß des Erlernens und der Aneignung dieser Fähigkeiten, Fertigkeiten und Kenntnisse an einer konkreten, sei es auch nur modellmäßig aufbereiteten Rechnerkonfiguration realisiert werden sollte. Im Zentrum der weiteren Betrachtungen steht deshalb das Modell eines Rechnerarbeitsplatzes (Bild 3), das

- den Dialog-Verkehr realisiert und
- die Eingabe, Korrektur und Testung anwendungsspezifischer Programme ermöglicht.

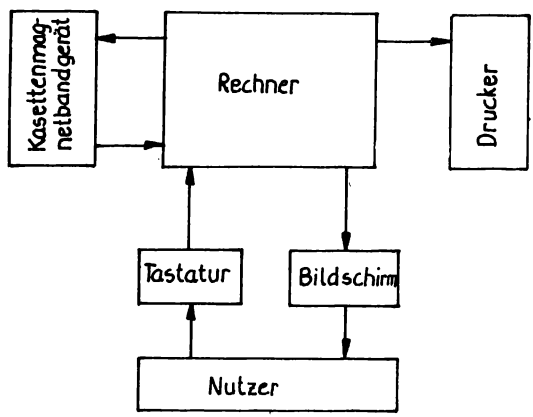


Bild 3: Modell eines speziellen Rechnerarbeitsplatzes

Dieses Modell besteht aus

- dem "eigentlichen" Rechner
- einer Tastatur und einem Bildschirm
- einem Kassettenmagnetbandgerät
- einem Drucker.

Im vorliegenden Heft werden im wesentlichen die hardwareseitigen und begrifflichen Grundlagen gelegt, um später bei der Algorithmierung und Programmierung die Probleme sachkundig einordnen zu können. Einige Kapitel - wie zum Beispiel 2. und 3. - sind so aufgebaut, daß ausgehend von den Darstellungen Probleme der Mikrorechentechnik im Sinne der Prozeßsteuerung weitergehend behandelt werden können.

2. Ausgewählte Grundlagen der Informationsverarbeitung

2.1. Physikalische Größe - Information - Signal

2.1.1. Allgemeines

Beim Umgang mit realen Objekten unserer natürlichen Umwelt haben die Menschen gelernt, die Objekte zu bewerten und sie sich zielgerichtet nützlich zu machen. Um diese Bewertung möglichst von der sie untersuchenden oder vergleichenden Person, also von subjektiven Faktoren, unabhängig zu machen, schuf man möglichst "objektive Maßstäbe" für die verschiedensten Eigenschaften realer Objekte. So entstand im Prozeß der dialektischen Auseinandersetzung des Menschen mit der Natur ein System untereinander abgestimmter Beschreibungs- und Bewertungsgrößen, die aus der Physik als physikalische Größen bekannt sind.

Beispiele dafür sind:

- Zur Beschreibung von Objekten
  - die Masse eines Körpers mit der Einheitenbenennung Kilogramm und dem Einheitenzeichen kg;
  - das Volumen mit der Einheitenbenennung Kubikmeter und dem Einheitenzeichen m^3;
  - die Länge, die Breite, die Höhe mit der Einheitenbenennung Meter und dem Einheitenzeichen m.

- Zur Beschreibung von Zuständen
  - die Temperatur eines Gases mit der Einheitenbenennung Kelvin und dem Einheitenzeichen K;
  - die Ladung eines Kondensators mit der Einheitenbenennung Coulomb und dem Einheitenzeichen C;
  - die Energie eines Körpers mit der Einheitenbenennung Newtonmeter und dem Einheitenzeichen N.m.
- Zur Beschreibung von Vorgängen
  - die Ortsveränderung eines Körpers mit der Einheitenbenennung Meter je Sekunde und dem Einheitenzeichen m/s;
  - die Stromstärke in einem elektrischen Stromkreis mit der Einheitenbenennung Ampere und dem Einheitenzeichen A;
  - der Fluß in einem Magnetfeld mit der Einheitenbenennung Weber und dem Einheitenzeichen Wb.

Allgemein gilt:

Eine physikalische Größe ist ein Merkmal eines physikalischen Objektes, Zustandes oder Vorganges, das meßbar ist. Eine physikalische Größe besteht immer aus dem Produkt von Maßzahl und Maßeinheit.

Beispiel: Masse $m = 9{,}108 \cdot 10^{-31}$ kg
(Ruhemasse eines Elektrons)

2.1.2. Information

Physikalische Größen (eigentlich die damit immer verbundenen Meßvorgänge) bereichern den Menschen. Sein Wissen, seine Kenntnis über das Objekt, über den Zustand und über den Vorgang wächst. Man sagt, es erfolgt ein Zuwachs an Information. So allgemein gesehen ist Information beseitigtes Nichtwissen.
Dabei entsteht nun die Frage, wie groß der Zuwachs an Information ist. Das ist die Frage nach der Meßbarkeit der Information.
Um zur Antwort zu gelangen, wird folgende Begebenheit betrachtet:
Peter, ein Junge im Dorf, ist schon sehr oft beim "Kirschenklauen" ertappt worden. Alle Dorfeinwohner wissen das.
Berichtet Frau Meier beim Einkaufen von einem erneuten Dieb-

stahl, so winken alle Frauen ab, was soviel heißt wie:"was sollte auch anders sein".
Der Zuwachs an Information ist gering. Von Peter hat man nichts anderes erwartet.
Käme nun jedoch Frau Meier und berichtet: "Peter hat der alten Paula die Kirschen abgetan, damit sie diese verkaufen kann". Alle Frauen würden erstaunt dem Bericht folgen. Etwas unerwartetes ist geschehen.
Der Zuwachs an Information ist hoch. Eine neue, bisher nicht bekannte Eigenschaft von Peter wurde erkannt.

Dem Leser werden genügend eigene Beispiele aus dem täglichen Leben einfallen. Das Maß der Information hat, so muß man aus den Beispielen folgern, etwas damit zu tun, ob ein Ergebnis, das gerade eintritt, einer gewissen Erwartung mehr oder weniger entspricht. Was aber erwartet wird, hängt davon ab, wie oft ein gewisses, ganz bestimmtes Ereignis schon eingetreten ist. Das Maß für die Erwartung ist bekannt. Es ist die Wahrscheinlichkeit für das Eintreten eines Ereignisses aus einer Menge der möglichen Ereignisse. Praktisch benutzt man die relative Häufigkeit als Näherung für die Wahrscheinlichkeit. Sie ist definiert durch:

$$\text{Relative Häufigkeit für das Auftreten des Ereignisses A} = \frac{\text{Anzahl des Auftretens von A}}{\text{Anzahl aller unternommenen Versuche}}$$

In dem hier interessierenden Zusammenhang wird zwischen relativer Häufigkeit und Wahrscheinlichkeit nicht unterschieden.
Das betrachtete Ereignis E1 tritt mit der Wahrscheinlichkeit p_{E1} auf, wobei

$$\frac{\text{Anzahl des Auftretens von E1}}{\text{Anzahl aller unternommenen Versuche}} = p_{E1} \quad \text{ist.}$$

Ein Beispiel soll den Gesamtzusammenhang herstellen:

In einem Behälter befinden sich insgesamt 100 Kugeln.
Der Behälter wird geleert und die Kugeln werden nachgezählt.
Das Ergebnis lautet:

25 rote Kugeln
20 blaue Kugeln
54 weiße Kugeln
1 gelbe Kugel

= 100 Kugeln

Die Kugeln werden in den Behälter zurückgelegt und gut gemischt. Stellt man sich nun die Frage, wie hoch die Erwartung (Wahrscheinlichkeit) dafür ist, daß man beim Herausgreifen eine Kugel mit einer bestimmten Farbe erhält, so findet man

- eine rote Kugel mit der Wahrscheinlichkeit $p_{rot} = \frac{25}{100} = 0{,}25$
- eine blaue Kugel mit der Wahrscheinlichkeit $p_{blau} = \frac{20}{100} = 0{,}20$
- eine weiße Kugel mit der Wahrscheinlichkeit $p_{weiß} = \frac{54}{100} = 0{,}54$
- die gelbe Kugel mit der Wahrscheinlichkeit $p_{gelb} \frac{1}{100} = 0{,}01$

Zu beachten ist, daß die Summe aller Wahrscheinlichkeiten

$$p = p_{rot} + p_{blau} + p_{weiß} + p_{gelb} = 1 \text{ ist.}$$

Das Ereignis, das man beim Herausgreifen entweder eine rote, blaue, weiße oder gelbe Kugel erhält, ist das sichere Ereignis. Dieses hat die Wahrscheinlichkeit p = 1.

Der amerikanische Informationstheoretiker SHANNON hat 1947 auf der Basis derartiger Überlegungen folgende Gleichung zur Berechnung des Informationsgehaltes angegeben:

$$H = - \sum_{i=1}^{N} p_i \cdot \text{ld } p_i$$

N = Anzahl aller möglichen Ereignisse

ld = $\log_2 = 3{,}32 \cdot \log_{10}$
= Zweierlogarithmus

[H] = bit (binary digit)

Diese Gleichung soll zur Beurteilung des Inhalts dreier Behälter B1, B2 und B3 herangezogen werden.
Im Behälter B1 sind 100 weiße Kugeln.

Im Behälter B2 sind 100 Kugeln mit der Farbverteilung des genannten Beispiels.
Im Behälter B3 sind 25 rote, 25 blaue, 25 weiße und 25 gelbe Kugeln.

Berechnung des Informationsgehaltes:

Behälter	N	$H = -\Sigma\ p_i \cdot ld\ p_i$	p_i	$ld\ p_i$	H
B1	1	$H1 = -\ p_1 ldp_1$	$p_1 = \frac{100}{100}$	0	H1 = 0 bit
B2	4	$H2 = -(p_1 \cdot ldp_1 +$ $p_2 \cdot ldp_2 +$ $p_3 \cdot ldp_3 +$ $p_4 \cdot ldp_4)$	$p_1 = 0{,}25$ $p_2 = 0{,}20$ $p_3 = 0{,}54$ $p_4 = 0{,}01$	-2,0 -2,32 -0,89 -6,64	H2 = 1,51 bit
B3	4	$H3 = -(p_1 \cdot ldp_1 +$ $p_2 \cdot ldp_2 +$ $p_3 \cdot ldp_3 +$ $p_4 \cdot ldp_4)$	$p_1 = 0{,}25$ $p_2 = 0{,}25$ $p_3 = 0{,}25$ $p_4 = 0{,}25$	-2,0 -2,0 -2,0 -2,0	H3 = 2 bit

Über die Maßeinheit (bit) soll später nachgedacht werden, jetzt geht es nur um den Vergleich der Maßzahlen.

Zusammengefaßt kann ausgewertet werden, daß der Informationsgehalt entsprechend den unterschiedlichen Wahrscheinlichkeiten der Ereignisse unterschiedliche Werte annimmt.
H1 = 0 bit, der Informationsgehalt für Ereignisse des Behälters B1 ist sofort verständlich. Das man aus diesem Behälter nur weiße Kugeln entnehmen kann, ist das sichere Ereignis. Einen Informationszuwachs kann es demnach nicht geben.
Entsprechend der Wahrscheinlichkeitsverteilung der Farben im Behälter B2 liefert die SHANNONsche Gleichung den Informationsgehalt H2 = 1,51 bit. Im Abschnitt 2.1.3.4. wird gezeigt, was dieser Wert bedeutet.
Von besonderer Bedeutung ist das Ergebnis, das der Behälter B3 liefert. Da alle Farben mit der gleichen Wahrscheinlichkeit ge-

zogen werden können, stellt die Information, daß man eine ganz bestimmte zieht, ein Maximum dar. Im Falle der vorliegenden Verteilung der Wahrscheinlichkeiten.

$p_1 = p_2 = p_3 = p_4 = p_i$ gilt $p_i = \frac{1}{N}$, wobei N die

Anzahl der möglichen Ereignisse darstellt. Die SHANNONsche Gleichung vereinfacht sich unter dieser Voraussetzung zu

$$H = - \sum_{i=1}^{N} p_i \cdot \mathrm{ld}\, p_i = - N(\tfrac{1}{N} \mathrm{ld}\, \tfrac{1}{N}) = - \mathrm{ld}\, \tfrac{1}{N} = \mathrm{ld}\, N$$

Im Falle der technischen Übertragung von Information ist es wichtig, die obere Schranke des Informationsgehalts zu erkennen, um die Übertragungseinrichtung dimensionieren zu können. Da dem Ingenieur die Wahrscheinlichkeitsverteilungen der möglichen Ereignisse häufig nicht bekannt sind, kann unter der Annahme der Gleichverteilung der maximale Informationsgehalt mit

$H_{max} = \mathrm{ld}\, N$ berechnet werden.

Information über Objekte, Zustände und Vorgänge wird also durch die Auswertung physikalischer Größen erhalten. Die Auswertung kann so erfolgen, daß man eine physikalische Größe ständig meßtechnisch erfaßt und über der Zeit aufträgt (Bild 4).

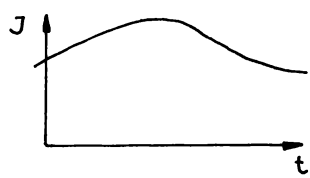


Bild 4: Zeitlicher Verlauf einer physikalischen Größe

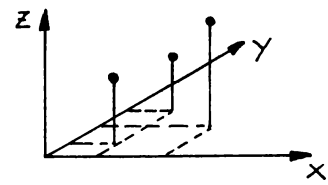


Bild 5: Räumliche Anordnung einer physikalischen Größe

Eine zweite Möglichkeit besteht darin, die physikalische Größe räumlich "anzuordnen"; das ist im Bild 5 gezeigt.

2.1.3. Signale

2.1.3.1. Allgemeines

Nutzt man die zeitliche Veränderung einer physikalischen Größe zur Darstellung von Information, so spricht man von einem Signal. In diesem Zusammenhang bezeichnet man die physikalische Größe als Signalträger. Die Größe, die innerhalb des Signalträgers die Information darstellt, nennt man Informationsparameter I.

Die bekannte Sinusfunktion, beispielsweise eine Sinusspannung,

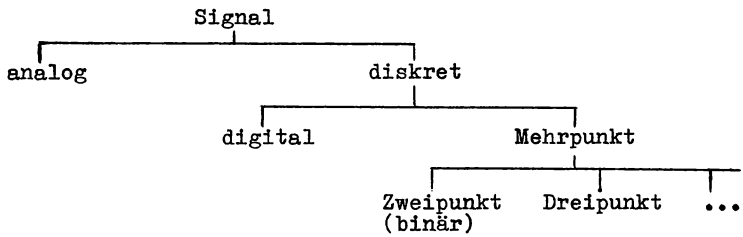
$$u = U \cdot \sin(\omega t + \varphi)$$

als Signal "trägt" 3 Größen, die jeder für sich Information tragen kann. Im einzelnen sind das

- die Amplitude U (Amplitudenmodulation)
- die Frequenz ω (Frequenzmodulation)
- der Phasenwinkel φ (Phasenmodulation).

2.1.3.2. Klassifizierung

Zur eindeutigen Verständigung klassifiziert man die zeitliche Veränderung physikalischer Größen in Signalarten. Man unterscheidet zwei prinzipielle Signalarten, die analogen und die diskreten Signale (s. a. TGL 14 591).



- Analoge Signale:
  Der Informationsparameter I kann in den technisch möglichen Grenzen jeden beliebigen Wert annehmen.

  Ein typisches Beispiel eines analogen Signals ist die Längenänderung der Quecksilbersäule in einem Thermometer.

Innerhalb der Gruppe der analogen Signale unterscheidet man nochmals das zeitliche Verhalten.

- analog kontinuierliches Signal:
  Der Informationsparameter kann seinen Wert zu beliebigen Zeiten ändern (Bild 6).

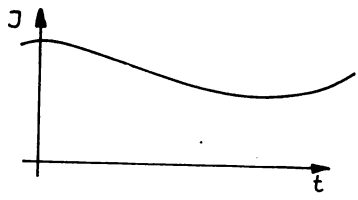


Bild 6: analog kontinuierliches Signal

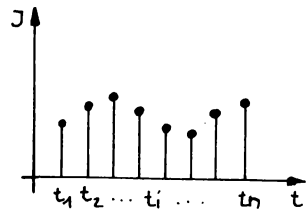


Bild 7: analog diskontinuierliches Signal

- analog diskontinuierliches Signal:
  Der Informationsparameter kann seinen Wert nur zu festgelegten Zeiten $t_1 \ldots t_n$ verändern (Bild 7).

- Diskrete Signale:
  Der Informationsparameter I kann in den technisch möglichen Grenzen nur ganz bestimmte (diskrete) Werte annehmen.

  Ein typisches Beispiel eines diskreten Signals ist das Drei-Punktsignal zur Motorsteuerung Vorwärts (V), Rückwärts (R), Stop (S) (Bild 8).

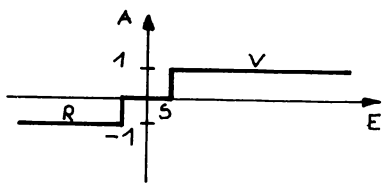


Bild 8: Drei-Punktsignal

Ebenso wie bei den analogen Signalen unterscheidet man in der Gruppe der diskreten Signale nochmals das zeitliche Verhalten

- diskrete kontinuierliche Signale (Bild 9):
- diskrete diskontinuierliche Signale (Bild 10):

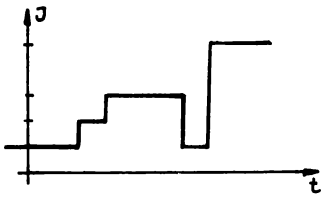


Bild 9: diskretes kontinuierliches Signal

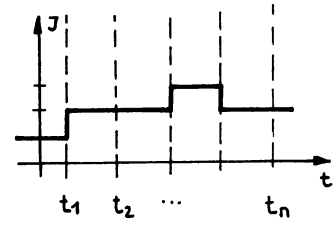
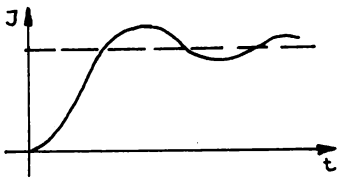


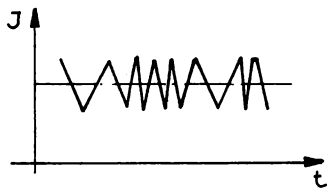
Bild 10: diskretes diskontinuierliches Signal

Übung zur Klassifizierung von Signalen



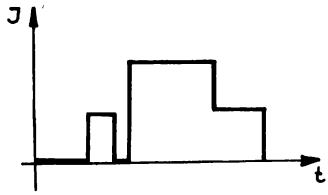
Informationsparameter: Amplitude
Signalart : analog kontinuierlich

Bild Ü1



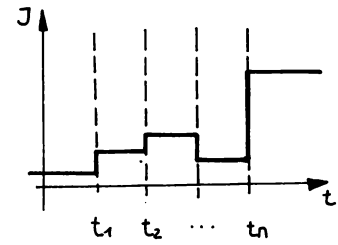
Informationsparameter: Frequenz
Signalart : analog, kontinuierlich

Bild Ü2



Informationsparameter: Amplitude
Signalart : diskret, kontinuierlich

Bild Ü3



Informationsparameter: Amplitude
Signalart : analog, diskontinuierlich

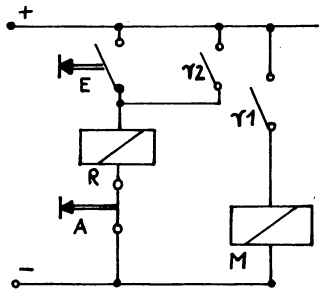
Bild Ü4

Im Zusammenhang mit der Informationsverarbeitung in und mit Rechnern sind zwei diskrete Signale von besonderem Interesse, denen man besondere Namen gegeben hat. Es sind die binären und die digitalen Signale.

2.1.3.3. Binäre Signale

Der Informationsparameter kann nur zwei Werte annehmen. Technische Baugruppen, die solche Signale erzeugen oder übertragen, sind beispielsweise

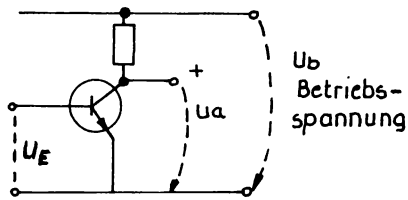
- Relais:



Die Relais R und M können nur 2 Zustände einnehmen. Im gezeichneten Zustand des Bildes 11 sind beide Relais stromlos. Wird E gedrückt, so zieht R an und damit auch M, da r1 geschlossen wird. Über r2 hält sich R selbst.
Die zwei Werte des Signals sind bei R und M: stromlos und stromdurchflossen, nicht gezogen und gezogen; bei E, A und r: offen und geschlossen.

Bild 11: Relaisschaltung zur Ansteuerung eines Motors

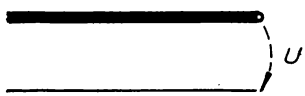
- Transistoren im Schalterbetrieb:



Die Spannung Ua im Bild 12 kann die Werte Ua = Ub annehmen. Der Transistor ist gesperrt und Ua ≈ 0 V, der Transistor ist durchgesteuert.
(Hinweis: Die Schaltung kehrt das Vorzeichen des binären Eingangssignals U_E um.)

Bild 12: Transistorsteuerstufe

- Leitungen:



Beispielsweise kann die Leitung gegenüber einem Bezugspotentional Spannung tragen oder nicht; z. B. +5 V und 0 V (Bild 13).

Bild 13: Leitung als Baugruppe

Trägt man den Verlauf der Spannungsamplitude über der Zeit auf, so ergibt sich das typische Signalbild des binären Signals (Bild 14).

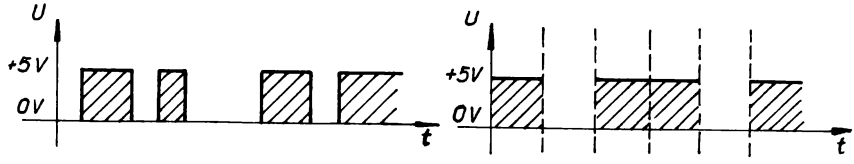


Bild 14: Binäres Signalverhalten

Die Einfachheit der Baugruppen zum Erzeugen und Übertragen binärer Signale als auch die große Sicherheit bei der Unterscheidung beider Zustände (2 Zustände kann man besser unterscheiden als 10, wenn man zum Beispiel an die Aufteilung eines 5 V-Bereiches denkt) haben das binäre Signal zum König der Informationsverarbeitung werden lassen.
Nimmt man die Pulsbreite als Informationsparameter und legt fest, daß es zwei Pulsbreiten gibt, z. B. so wie in Bild 15,

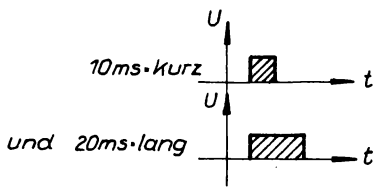


Bild 15: Binäres Signal mit Pulsbreite als Informationsparameter

dann kann man unter Beachtung einer Vereinbarung, wie z. B. dem Morsealphabet, Nachrichten übertragen. Unter Nachricht soll die (technische)Information plus hineingelegte Bedeutung (Semantik) verstanden werden.

Übertragen wird im Bild 16 das Wort "Tag", das nach dem Morsealphabet die Pulsfolge

lang kurz lang lang lang kurz hat.

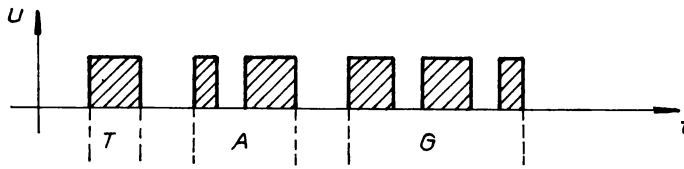


Bild 16: Übertragen einer Nachricht durch ein binäres Signal

In der Informationsverarbeitung zur Steuerung von Maschinen, Geräten und Anlagen hat das binäre Signal eine solche zentrale Stellung erlangt, daß beim Entwurf von Steuerungen von der technischen Realisierungsart (Relaissteuerung oder Transistorsteuerung oder pneumatische, oder hydraulische Steuerung) abstrahiert wird. Man spricht nur von den beiden logischen Werten Null ("0") und Eins ("1"), die das binäre Signal einnehmen kann.

Wenn man logisch 0 sagt, meint man
- das Relais hat nicht gezogen
- der Schließer eines Relais ist offen
- der Transistor ist durchgesteuert (Ua = 0).

Wenn man logisch 1 sagt, meint man
- das Relais hat gezogen
- der Schließer eines Relais ist geschlossen
- der Transistor ist gesperrt (Ua = Ub).

Um die logische Verknüpfung der Signale unabhängig von der technischen Realisierungsart darstellen zu können, schuf man logische Grundglieder. In Bild 17 sind die fundamentalen Logikglieder UND (AND), ODER (OR) und NICHT (NOT) dargestellt und in ihrer Bedeutung erläutert.

Im Hinblick auf die Informationsverarbeitung in Speichern der Rechentechnik wird noch ein Grundelement zur Speicherung binärer Signale besprochen. Aus der Fülle der möglichen Grundschaltungen soll die einfachste, der sogenannte D-Trigger dargestellt werden. Die Bezeichnung D- kommt vom englischen Delay, verzögern. Bild 18 zeigt das Symbol des D-Triggers.

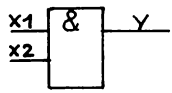
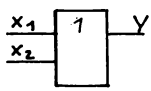
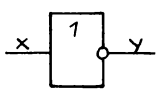
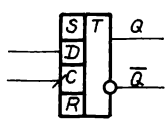
<table>
<tr><th>Symbol</th><th>Bezeichnung</th><th>log Gleichung</th><th>Erläuterung und Schaltbelegungstabelle</th></tr>
<tr><td> 
</td><td>UND
AND
Konjunktor</td><td>$y = x_1 x_2$</td><td>gemeint ist: Das binäre Signal y hat logisch 1, wenn das binäre Signal x1 logisch 1 trägt <u>und</u> das binäre Signal x2 logisch 1 trägt. Ansonsten ist das binäre Signal y logisch 0. Daraus ergibt sich folgende Tabelle:
<table>
<tr><th>x_1</th><th>x_2</th><th>y</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td> 
</td><td>ODER
OR
Disjunktor</td><td>$y = x_1 + x_2$</td><td><table>
<tr><th>x_1</th><th>x_2</th><th>y</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
y = 1, wenn $x_1 = 1$ oder $x_2 = 1$ trägt</td></tr>
<tr><td> 
</td><td>NICHT
NOT
Inverter</td><td>$y = \overline{x}$</td><td><table>
<tr><th>x</th><th>y</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
y = 1, wenn x = 0
y = 0, wenn x = 1</td></tr>
</table>

Bild 17: Logikglieder UND, ODER, NICHT



D : *binäres Eingangssignal*
C : " *Triggersignal*
Q : " *Ausgangssignal*
S : " *Setzsignal*
R : " *Rücksetzsignal*

Bild 18: Symbol des D-Triggers

Die Signalverarbeitung vom Eingangssignal D zum Ausgangssignal Q wird durch das Triggersignal C gesteuert. Im einzelnen ist folgende Verarbeitung festgelegt:

Erscheint am Triggereingang C die Vorderflanke (0 → 1 Flanke) eines binären Signals, so übernimmt der Ausgang Q den logischen Wert von D genau zum Zeitpunkt dieser Vorderflanke.

Die Vorderflanke von C (Bild 19) ist somit der "Pförtner", der

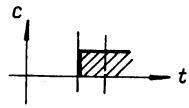


Bild 19: Vorderflanke von C

die Tür für das D-Signal öffnet, damit es zu Q kommen kann. Ohne Vorderflanke von C hat eine Änderung des logischen Wertes von D keinen Einfluß auf Q. In Q bleibt also das D-Signal vom Zeitpunkt der letzten Vorderflanke von C gespeichert.

Neben dieser, wie man sagt, dynamischen Arbeitsweise besteht noch die Möglichkeit, das Ausgangssignal Q direkt (statisch, ohne C-Flanke) über den Setzeingang S auf logisch 1 und über den Rücksetzeingang R auf logisch 0 zu setzen.

Bei S = 1 folgt Q = 1, bei R = 1 folgt Q = 0.

Der Zustand S = 1 und R = 1 muß schaltungstechnisch vermieden werden.

2.1.3.4. Digitale Signale

Als digitales Signal bezeichnet man eine bestimmte Anzahl von binären Signalen, die gemeinsam bewertet werden. Was darunter zu verstehen ist, soll folgendes Beispiel zeigen. Ein Leitungsbündel von 4 Leitungen ist nach Bild 20 gegeben. Jede Leitung überträgt ein binäres Signal (U_1, U_2, U_3, U_4).

Betrachtet man die 4 binären Signale als Einheit, so ergibt sich die Möglichkeit, 16 unterschiedliche Zustände des nun digitalen Signals zu unterscheiden. Im Bild 21 sind die Kombinationen der binären Stellen des digitalen Signals angegeben.

Das digitale Signal hat die "Breite" 4.

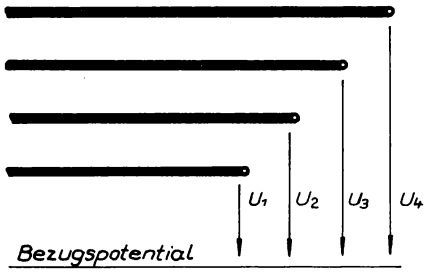


Bild 20: Zusammensetzung binärer Signale zu einem digitalen Signal

U_4	U_3	U_2	U_1	Zustand
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
0	0	1	1	4
0	1	0	0	5
0	1	0	1	6
0	1	1	0	7
0	1	1	1	8
1	0	0	0	9
1	0	0	1	10
1	0	1	0	11
1	0	1	1	12
1	1	0	0	13
1	1	0	1	14
1	1	1	0	15
1	1	1	1	16

Bild 21: Mögliche Unterscheidung von Zuständen der 4 binären Signale im digitalen Signal

An dieser Stelle soll die Brücke zum Begriff der Information und ihrem Maß zurückgeschlagen werden, das sich aus der SHANNONschen Gleichung ergab.
Dort wurde gefunden, daß sich der Informationsgehalt H_{max} der auszuwertenden Objekte, Zustände oder Vorgänge nach der Gleichung H_{max} = ld N berechnet. N war hierbei die Anzahl der überhaupt möglichen Ereignisse und "ld" der Zweierlogarithmus. Die Maßeinheit wurde mit "bit" angegeben. Zur weiteren Verständigung werden für einige N die dazugehörigen H_{max}-Werte berechnet.

N	H_{max}
2	1 bit
4	2 bit
8	3 bit
16	4 bit

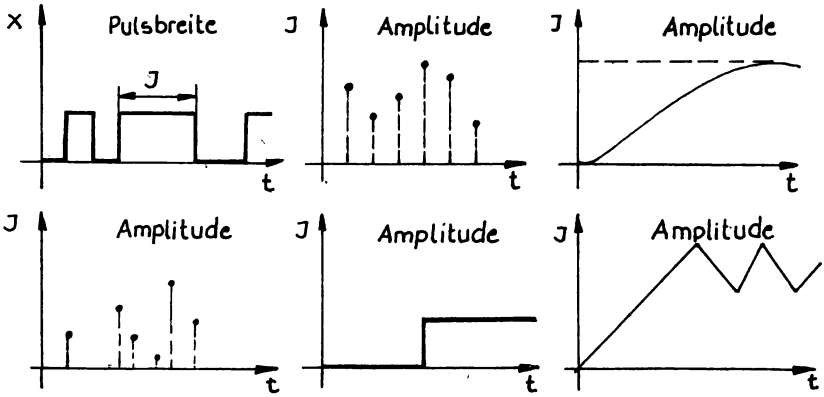
Interessant ist, daß N = 16 einen maximalen Informationsgehalt von 5 bit erbringt. Diese Tatsache korrespondiert sehr genau mit dem 4stelligen digitalen Signal, das 16 Zustände unterscheidbar darstellen kann, womit natürlich 16 Ereignisse eindeutig übertragbar sind.

Nun ist auch klar, was unter der Maßeinheit "bit" (binary digit) technisch zu verstehen ist. Dahinter versteckt sich - die Leitung, das Relais, der Transistor - kurz, die Bitstelle. Sie ist die technische Realisierungsform zur Erzeugung, Übertragung oder Speicherung des Informationsgehaltes von H = 1 bit.
Die Maßzahl des Informationsgehaltes gibt dann an, wie viele Bitstellen benötigt werden, um die Information zu übertragen, zu speichern usw.
Der maximale Informationsgehalt legt also die Breite des digitalen Signals fest.

Kontrollfragen und Aufgaben:

K1. Was versteht man unter dem Begriff Information?
K2. Welcher Zusammenhang besteht zwischen den Begriffen Information und Signal?
K3. Wie teilt man Signale ein?
K4. Charakterisieren Sie ein binäres Signal!
K5. Was versteht man unter dem Begriff "maximaler Informationsgehalt" und welcher Zusammenhang läßt sich zu einem digitalen Signal herstellen?

A1. Klassifizieren Sie folgende Signale!



Bilder A1 bis A6

2.2. Kode - Kodierung - Dekodierung

2.2.1. Allgemeines

Unter einem Kode versteht man ganz allgemein eine Vorschrift, nach der Symbole einer "Sprache" in die Symbole einer anderen "Sprache" überführt werden können.
Im Fall der Kommunikation des Menschen mit einer informationsverarbeitenden Anlage (Rechner, NC-Maschine usw.) liegt der Fall vor, daß die Symbole der "Menschensprache"

- Ziffern (0, 1, ...,9) → 10 numerische Zeichen
- Buchstaben (A, B, ...,Z) → 26 Alpha-Zeichen
- Sonderzeichen → Interpunktion

in Symbole der "Maschinensprache"

- binäre Signalzustände
- digitale Signalzustände

zu überführen sind. Diesen Prozeß nennt man Kodierung, den umgekehrten Prozeß nennt man Dekodierung (Bild 22).

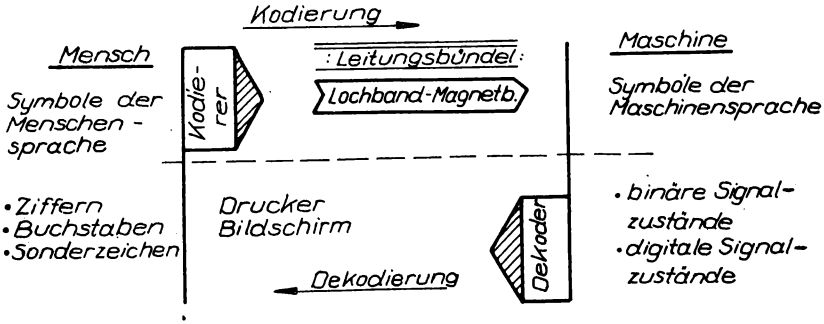


Bild 22: Informationsbeziehung des Menschen mit "intelligenten" Maschinen

2.2.2. Numerische Kode

Beschränkt man sich in bezug auf die Menschensprache mit der Übertragung von Ziffern, so gilt:

Es existieren die Ziffern 0, 1, 2, ..., 9 und somit 10 mögliche Ereignisse. Der maximale Informationsgehalt der zu übertragen ist, ergibt sich damit zu

$$H_{max} = \text{ld } 10 = 3{,}32 \text{ bit.}$$

Man benötigt also ein digitales Signal der Breite 4 zur Übertragung dieser Information.
Bewertet man die einzelnen Bitstellen des digitalen Signals, so daß die Abstufung in Potenzen zur Basis 2 erfolgt, so entsteht die sogenannte BCD (Binary-Coded-Dezimal)-Kodierung der Dezimalziffern.
Die Bewertung der Bitstellen x_0, x_1, x_2 und x_3 erfolgt somit durch

$$x_0 = 2^0 = 1$$
$$x_1 = 2^1 = 2$$
$$x_2 = 2^2 = 4$$
$$x_3 = 2^3 = 8$$

Daraus ergibt sich folgende Kodierungsvorschrift:

Dezimal-ziffer	BCD-Kode x_3 x_2 x_1 x_0
	8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Da mit einem digitalen Signal der Breite 4 insgesamt 16 Zustände unterscheidbar sind, für die Dezimalziffern aber nur 10 benötigt werden, bleiben 6 Kombinationen als sogenannte Pseudotetraden übrig (Tetra - vier).
Pseudotetraden beim BCD-Kode sind:

x_3	x_2	x_1	x_0
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Die Pseudotetraden können zur Verschlüsselung (Kodierung) von notwendigen Trennzeichen, wie z. B. dem Komma und dem Trennzeichen von Zahl zu Zahl verwendet werden. Hier wird ohne Rücksicht auf praktische Belange festgelegt:

Trennzeichen	Tetrade x_3 x_2 x_1 x_0	Bedeutung
Komma	1 0 1 1	trennt den ganzen vom gebrochenen Teil einer Zahl
Punkt	1 0 1 0	trennt zwei Zahlen

Die nach dieser Vorschrift kodierte Information wird auf einem digitalen Datenträger (Magnetband, Lochband, Folienmagnetplatte u. a.) dargestellt und ist somit von intelligenten Maschinen übernehmbar. Bild 23 stellt diesen Sachverhalt dar.

Der Kodierer besteht aus ODER-Gliedern. Die Bitstelle x_3 trägt 1-Signal, wenn
- die Ziffer 8 oder
- die Ziffer 9 oder
- das Komma oder
- der Punkt

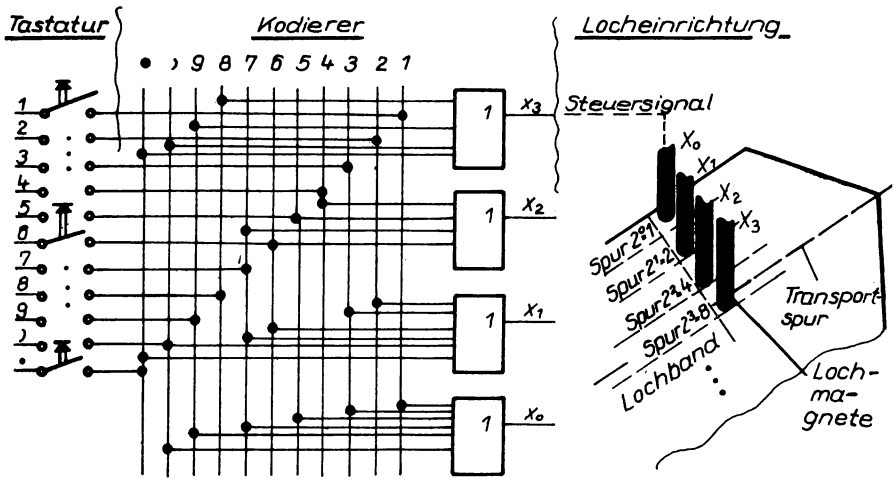


Bild 23: Kodierung der Dezimalziffern und Ablegen der kodierten Ziffer auf einem Lochband

angewählt wurde. Das Signal bewirkt, daß die Locheinrichtung der Spur $2^3 = 8$ in das Band ein Loch stanzt. Die 3 angeren Locheinrichtungen arbeiten entsprechend.
Wird somit beispielsweise die Ziffer 6 angewählt, so folgt

die Spur x_3 wird nicht gelocht
die Spur x_2 wird gelocht
die Spur x_1 wird gelocht
die Spur x_0 wird nicht gelocht

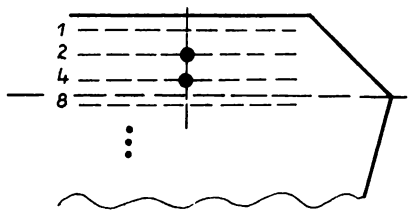


Bild 24: Ablage der Ziffer 6

Die Dezimalzahl 164,183 wird dann wie folgt auf einem digitalen Datenträger dargestellt (Bild 25).

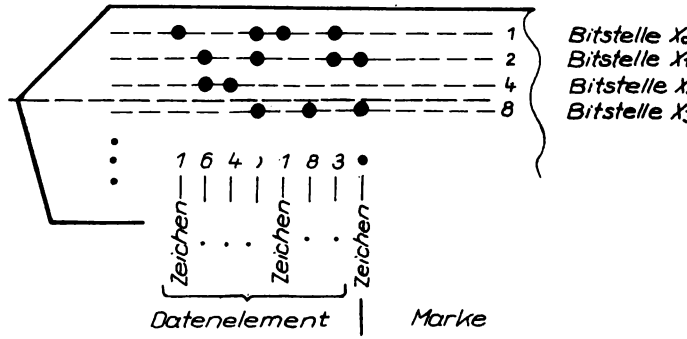


Bild 25: Darstellung von Dezimalzahlen auf einem Lochband

In bezug auf die Darstellung von Informationen zum Zwecke der Verarbeitung in Rechnern spricht man von Daten.
Die im Bild 25 dargestellte Date hat die Struktur

1 Datenelement und 1 Marke.

Das Prinzip der Übernahme der Daten durch den Rechner ist in Bild 26 dargestellt. Die Synchronisation sorgt für die richtige Übernahme der Daten in einen Zwischenspeicher (Puffer). Zur Erinnerung: Der Ausgang Q eines D-Triggers übernimmt den logischen Wert von D bei einer Vorderflanke vom Trigger-Signal C.

Bei der Übertragung von Daten nach dem oben angegebenen Prinzip stellt sich die Frage nach der Störanfälligkeit.
Störungen können zum Beispiel sein:

- Ein Lochmagnet klemmt kurzzeitig.
- Das Lochband hat "dünne" Stellen im Papier.
- Die Schließkontakte sind durch Abrieb kurzzeitig isoliert.

Dadurch würden Fehler in der Kodierung entstehen, die als solche nicht erkannt werden. Aus diesem Grund nimmt man noch eine 5. Bitstelle zu den 4 notwendigen hinzu und setzt diese auf logisch 1, wenn die Lochungen der Date im Lochband

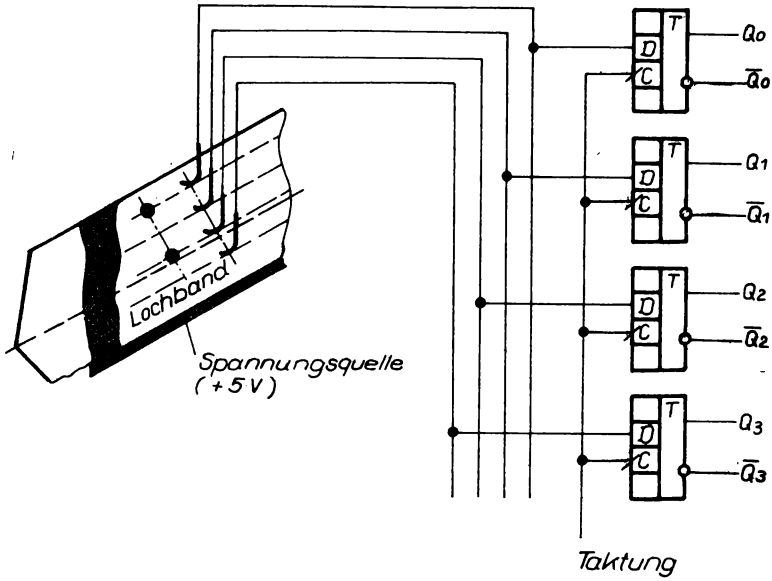


Bild 26: Übernahme der kodierten Daten in einen Puffer aus D-Triggern

eine ungerade Anzahl von Lochungen erfordert,
→ man nennt diese Art die gerade Paritätsbitsetzung
oder
eine gerade Anzahl von Lochungen erfordert,
→ man nennt diese Art die ungerade Paritätsbitsetzung.
Im Rechner wird dann geprüft, ob die Parität (gerade oder ungerade) eingehalten wird. Damit lassen sich Fehler einer Spur erkennen, da durch den Fehler gerade die falsche Parität erzeugt wird.
Neben dem hier benutzten BCD-Kode zur Darstellung von numerischen Daten gibt es zahlreiche andere. Da diese in der Rechentechnik nicht eingesetzt werden, sollen die drei bekanntesten nur mit ihrem Namen erwähnt werden:
- Aiken-Kode
- Gray-Kode
- Dreiexzeß-Kode.

2.2.3. Alphanumerische Kode

Wie der Name schon sagt, handelt es sich um Kode, die den gesamten Umfang der Symbole der "Menschensprache" kodieren können. Mehrere nationale und internationale Kodes wurden entwickelt. Stellvertretend soll der internationale Standardkode ISO-7-Bit-Kode angegeben werden. Er ist mit dem in der Mikrorechentechnik häufig eingesetzten ASCII-Kode (American Standard Code for Information Interchange) identisch. Das 8te Bit des ASCII-Kodes ist immer gleich logisch 0, wenn Bild 27 benutzt wird.

Bit's 7	0	0	0	0	1	1	1	1
Bit's 6	0	0	1	1	0	0	1	1
Bit's 5	0	1	0	1	0	1	0	1
4 3 2 1								
0 0 0 0	NUL	DLE	b	0	@	P	·	p
0 0 0 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	"	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s
0 1 0 0	EOT	STOP	¤	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	'	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	K	Ä	k	ä
1 1 0 0	FF	FS	,	<	L	Ö	l	ö
1 1 0 1	CR	GS	-	=	M	Ü	m	ü
1 1 1 0	SO	RS	.	>	N	^	n	ß
1 1 1 1	SI	US	/	?	O	-	o	DEL

Bild 27: Kode des ISO-7-Bit-Kodes (s. a. Bild 29)

Entsprechend diesem Kode würde das Wort "TAG" auf einem Lochband wie folgt dargestellt werden (Bild 28):

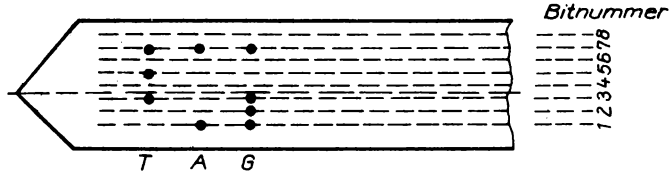


Bild 28: Darstellung des Wortes "TAG"

Die Bedeutung der im Kode angegebenen Kurzzeichen (wie z. B. EOT) ist in Bild 29 beschrieben.

Symbol	Bedeutung	
	englisch	deutsch
NUL	nil	Null
SOH	start of heading	Anfang des Kopfes
STX	start of text	Anfang des Textes
ETX	end of text	Ende des Textes
EOT	end of transmission	Ende der Übertragung
ENQ	enquiry	Antwort von Station gefordert
ACK	acknowledge	positive Rückmeldung
BEL	bell	Klingel
BS	backspace	Rückwärtsschritt
HT	horizontal tabulation	Horizontaltabulator
LF	Line feed	Zeilenvorschub
VT	vertical tabulator	Vertikaltabulator
FF	from feed	Formularvorschub
CR	carriage return	Wagenrücklauf
SO	shift out	Dauerumschaltung
SI	shift in	Rückschaltung
DLE	data link escape	Datenübertragungsumschaltung
DC	device control	Gerätesteuerung
STOP	stop	Halt
NAK	negative acknowledge	negative Rückmeldung
SYN	synchronousidle	Synchronisierung
ETB	end of transmission block	Ende des Datenübertragungsblocks
CAN	cancel	ungültig
EM	end of medium	Ende des Mediums
SUB	substitute character	Substitution
ESC	escape	Umschaltung
FS	file separator	Filetrennzeichen
GS	group separator	Gruppentrennzeichen
RS	record separator	Satztrennzeichen

Symbol	Bedeutung	
	englisch	deutsch
US	unit separator	Trennzeichen für Dateneinheiten
DEL	delete	Löschen

Bild 29: Erläuterung der Kurzzeichen des ISO-7-Bit-Kode

Kontrollfragen und Aufgaben:

K6. Was verstehen Sie unter einem Kode?

K7. Wie werden Zahlen auf digitalen Datenträgern dargestellt?

K8. Was verstehen Sie unter einem Datenwort?

K9. Wie kann man Fehler in Kodeeinrichtungen erkennen?

A2. Geben Sie die BCD-Kodierung für die Dezimalziffern 1 bis 9 an!

A3. Stellen Sie auf einem 8-Kanallochband das Wort "Lernen" dar. Benutzen Sie dazu den ASCII-Kode!

2.3. Zahlen und das Prinzip ihrer Darstellung

2.3.1. BCD-kodierte Dezimalzahlen

Bei der Kodierung der Ziffern des Dezimalsystems in den BCD-Kode und der Darstellung einer Zahl auf einem digitalen Datenträger wurde eine prinzipielle Darstellungsform von Zahlen in informationsverarbeitenden Anlagen schon erläutert.
Die Darstellungsform geht davon aus, daß jede Ziffer der Dezimalzahl für sich in eine Tetrade kodiert wird. Die Anordnung der Tetraden gibt die Stellenwertigkeit der Tetrade an. Diese Form der Zahlendarstellung nennt man

BCD-kodierte Dezimalzahldarstellung.

Ein Beispiel soll diese Form der Zahlendarstellung vertiefen. Dargestellt werden soll die Zahl 183,42:

1. Schritt: Kodierung der Ziffern nach dem BCD-Kode

Ziffer	BCD-Kode
1	0 0 0 1
8	1 0 0 0
3	0 0 1 1
4	0 1 0 0
2	0 0 1 0

2. Schritt: Stellenwertrichtige Darstellung der BCD-kodierten Dezimalzahl. Dabei ist zu berücksichtigen, daß die Darstellung der Dezimalzahl in der Form "183,42" eine Kurzform ist für die vollständige Darstellung der Dezimalzahl

$$Z = 1.10^2 + 8.10^1 + 3.10^0 + 4.10^{-1} + 2.10^{-2}$$

Daraus folgt folgende Darstellungsform:

Stellenwert der Tetrade	10^2	10^1	10^0	10^{-1}	10^{-2}
Wertigkeit in der Tetrade	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1
kodierter Ziffernwert	0 0 0 1	1 0 0 0	0 0 1 1	0 1 0 0	0 0 1 0

Bei der Verarbeitung derartig dargestellter Zahlen ist nun darauf zu achten, daß nur Tetraden gleicher Stellenwerte verknüpft werden.

2.3.2. Direkt dual kodierte Dezimalzahlen

Eine weitere Möglichkeit der Zahlendarstellung in Rechnern basiert darauf, eine Dezimalzahl aus dem dezimalen Zahlensystem in das "duale Zahlensystem" zu konvertieren (zu überführen). So wie das dezimale Zahlensystem, das Zahlensystem des Menschen, ist das duale Zahlensystem, das der "Maschine". "Dual" - deutet darauf hin, daß die Zahl 2 in diesem System genau die gleiche Rolle wie die Zahl 10 im Dezimalsystem spielt. Es ist nützlich, sich noch einmal den Aufbau des Dezimalsystems zu vergegenwärtigen, bevor man sich dem "fremden" Dualsystem zuwendet.

2.3.2.1. Dezimalsystem

Schreibt man die Ziffernfolge 39700,204 auf und fragt nach dem Zahlenwert dieser Zahl, so würde jeder im Rechnen ausgebildete Mensch antworten

Neununddreißigtausendsiebenhundert Komma zwei null vier.

Die ausgeschriebene Form zeigt, daß die oben angegebene Ziffernfolge eine abkürzende Schreibweise für Zahlen der "Menschensprache" ist.

Den Zahlenwert erhält man aus der Ziffernfolge, wenn man den Positionswert der Ziffer mit dem Ziffernwert multipliziert und alle Zwischenergebnisse addiert. Der Zahlenwert ergibt sich also aus:

$$\text{Zahlenwert} = 3.10^4+9.10^3+7.10^2+0.10^1+0.10^0+2.10^{-1}+0.10^{-2}+4.10^{-3}$$

$$\begin{aligned} &= \quad 30000, \\ &+ \quad 9000, \\ &+ \quad 700, \\ &+ \quad 00, \\ &+ \quad 0, \\ &+ \quad ,2 \\ &+ \quad ,00 \\ &+ \quad ,004 \\ &\hline \\ &\quad 39700,204 \\ &========== \end{aligned}$$

Das Komma dient zur Erkennung der Stelle, an jener das Vorzeichen des Exponenten wechselt. Das ist eine sinnvolle Abmachung und führt zur Kurzform der Zahlendarstellung.

Die Zehnerpotenzen 10^4, 10^3, ..., 10^{-1}, 10^{-2}, 10^{-3} sind die Positionswerte des Dezimalsystems. Die "10" nennt man Basis. Die Ziffernwerte sind 0, 1, 2, 3, 4, 5, 6, 7, 8 und 9.

Den Zahlenwert einer Zahl erhält man im Dezimalsystem, indem man die Summe

$$\text{Zahlenwert} = \Sigma\, \text{Ziffernwert} \cdot 10^{\text{Exponent}}$$

errechnet. Das Prinzip der Errechnung des Zahlenwertes gilt für alle Zahlensysteme, die von der Positionierung der Ziffern ausgehen, also Positionssysteme sind. Verändert werden die Basis und die Ziffernwerte. Es gilt für Positionssysteme demnach immer:

$$\text{Zahlenwert} = \Sigma\, \text{Ziffernwert} \cdot \text{Basis}^{\text{Exponent}}.$$

2.3.2.2. Dualsystem

Das Dualsystem ist auf der Zahl 2 aufgebaut. Die Zahl 2 ist die Basis des dualen Zahlensystems. Die Ziffernwerte des Dualsystems sind 0 und 1.

Es wird sofort deutlich, warum das Dualsystem das Zahlensystem der Maschine ist. Kann doch, wie schon gezeigt, der Zustand "0" und der Zustand "1" technisch einfach und gut unterscheidbar realisiert und gegen Störungen gesichert werden.

Zur Bildung von Zahlen stehen im dualen Zahlensystem demnach die Ziffernwerte 0 und 1 und als Positionswerte Potenzen zur Basis 2, z. B. 2^3, 2^4, 2^{-1}, 2^{-3}, zur Verfügung. Eine Zahl im Sinne des Dualsystems ist zum Beispiel 11101,101.

Auch diese Darstellung ist eine Kurzform. Den Zahlenwert erhält man durch Auswertung der Summe

$$\text{Zahlenwert} = \Sigma \text{Ziffernwert} \cdot 2^{\text{Exponent}}.$$

Also:

$$\text{Zahlenwert} = 1.2^4+1.2^3+1.2^2+0.2^1+1.2^0+1.2^{-1}+0.2^{-2}+1.2^{-3}$$

Das Ausrechnen dieser Summe liefert den Wert der Zahl. Dieser ist immer gleich, gleichgültig in welcher Form (Zahlensystem) dieser Wert aufgeschrieben wird. Der Wert einer Zahl ergibt damit immer die Dezimalzahl.

$$\begin{aligned} Z = \; & 1.2^4 = 16 \\ + \; & 1.2^3 = 8 \\ + \; & 1.2^2 = 4 \\ + \; & 0.2^1 = 0 \\ + \; & 1.2^0 = 1 \\ + \; & 1.2^{-1} = 0{,}5 \\ + \; & 0.2^{-2} = 0 \\ + \; & 1.2^{-3} = 0{,}125 \\ \hline (Z)_{10} \; & = 29{,}625 \end{aligned}$$

Zur Unterscheidung der Zahlen in den unterschiedlichen Zahlensystemen gibt man häufig die Basiszahl an (z. B. $(Z)_{10}$).

2.3.2.3. Konvertierung

Die Überführung (Konvertierung) einer Dezimalzahl in das Dualsystem ist letztlich die Beantwortung der Frage, wie oft die Basis 2 des Dualsystems in der vorgegebenen Dezimalzahl enthalten ist.

- Für ganze Zahlen gilt die Regel: Wiederholte ganzzahlige Division der gegebenen Dezimalzahl durch die Basis 2. Der Rest ergibt die Ziffern der Zahl im Dualsystem.

  • Ganzzahlige Division: 3 + 2 : 1 Rest 1
    4 + 2 : 2 Rest 0

  • Beispiel: Konvertierung der Dezimalzahl 29

    29 + 2 : 14 Rest 1
    14 + 2 : 7 Rest 0 $(Z)_2 = 11101$
    7 + 2 : 3 Rest 1
    3 + 2 : 1 Rest 1
    1 + 2 : 0 Rest 1

- Für gebrochene Zahlen gelten die Regeln:

  1. Der ganzzahlige Teil der Zahl ist nach der Regel für ganze Zahlen zu behandeln.
  2. Für den gebrochenen Teil gilt:

     Wiederholte Multiplikation mit der Basis 2.
     Die Vorkommastelle jedes Zwischenergebnisses ist ein Ziffernwert der gesuchten Dualzahl.
     Die Fortführung der Multiplikation erfolgt jeweils nur mit dem gebrochenen Teil des Zwischenergebnisses.
  3. Zusammenfügen des ganzen und gebrochenen Teils der Einzelergebnisse.
     Gebrochener und ganzer Teil sind durch Komma zu trennen.

An einem Beispiel soll das Verfahren erläutert werden.

Zu konvertieren ist die Dezimalzahl 29,625.

1. Der ganzzahlige Anteil "29" wurde schon konvertiert und ergab $(Z)_2 = 11101$.

2. Konvertierung des gebrochenen Teils "0,625"

0,625 . 2 = 1,250
0,250 . 2 = 0,500
0,5 . 2 = 1,0

Der Rest ist nicht mehr zu multiplizieren,

also

$(Z)_2$ = 101 (Aufschreibrichtung von oben nach unten)

3. Die konvertierte Gesamtzahl ist $(Z)_2$ = 11101,101.

Die durch die Konvertierung erhaltene Zahl nennt man Dualzahl oder auch direkt dualkodierte Dezimalzahl.
Ein derartiges Zahlenformat im Rechner (oder anderen informationsverarbeitenden Maschinen) abzulegen, erfordert gerätetechnisch realisierte Bitstellen, die entsprechend dem Dualsystem "bewertet" werden. So kann man beispielsweise in einem Ensemble von 8-Bitstellen bei direkter dualer Bewertung in der Form

$$2^7, 2^6, 2^5, 2^4, 2^3, 2^2, 2^1, 2^0$$

die Dezimalzahlen 0 bis 255 als ganze Zahlen darstellen.
In Bild 30 sind einige Beispiele angegeben.

Wertigkeit / Dezimalzahlen	2^7=128	2^6=64	2^5=32	2^4=16	2^3=8	2^2=4	2^1=2	2^0=1
0	0	0	0	0	0	0	0	0
16	0	0	0	1	0	0	0	0
131	1	0	0	0	0	0	1	1
255	1	1	1	1	1	1	1	1

Bild 30: Beispiele zur Abspeicherung von direkt dualkodierten Dezimalzahlen

2.3.2.4. Vereinfachte Schreibweise für Dualzahlen

Die recht langen Ziffernfolgen der Dualzahlen sind störend und bieten Fehlerquellen durch die Monotonie beim Aufschreiben.
Aus diesem Grund benutzt man zum Notieren von Dualzahlen Kurzformen und zwar die gleichwertigen Oktalzahlen oder Hexadezi-

malzahlen. Es ist zu beobachten, daß sich die Hexadezimaldarstellung immer mehr durchsetzt. Deshalb soll diese Form der Darstellung hier angegeben werden.

Hexadezimalzahlen sind Zahlen im Hexadezimalsystem (auch Sedezimalsystem genannt). Die Basis dieses Zahlensystems ist die "16". Da $16 = 2^4$ ist, kann jede Dualzahl so umgeschrieben werden, daß die neue Basis "16" entsteht. Gezeigt werden soll die Methode an einer 8stelligen Dualzahl:

$$(Z)_2 = ZW_2 \cdot 2^7 + ZW_2 \cdot 2^6 + ZW_2 \cdot 2^5 + ZW_2 \cdot 2^4 + ZW_2 \cdot 2^3 + ZW_2 \cdot 2^2 + ZW_2 \cdot 2^1 + ZW_2 \cdot 2^0$$

$$= (ZW_2 \cdot 2^3 + ZW_2 \cdot 2^2 + ZW_2 \cdot 2^1 + ZW_2 \cdot 2^0) \cdot (2^4)^1 + (ZW_2 \cdot 2^3 + ZW_2 \cdot 2^2 + ZW_2 \cdot 2^1 + ZW_2 \cdot 2^0) \cdot (2^4)^0$$

$$(Z)_{16} = ZW_{16} \cdot 16^1 + ZW_{16} \cdot 16^0$$

Das Prinzip besteht darin, daß Potenzen von $16 = 2^4$ ausgeklammert werden. Dabei ist zu beachten, daß von rechts begonnen wird und somit zuerst 16^0, also "1" ausgeklammert wird. Das zweite Mal ist 16^1 also 2^4 auszuklammern. Wie zu sehen ist, entstehen durch das systematische Ausklammern neue Ziffernwerte ZW_{16}. Diese ergeben sich aus der Summe der Produkte

$$ZW_2 \cdot 2^3 + ZW_2 \cdot 2^2 + ZW_2 \cdot 2^1 + ZW_2 \cdot 2^0 .$$

Die Ziffernwerte ZW_2 des Dualsystems können die Werte 0 und 1 annehmen. Somit ergeben sich für die Ziffernwerte ZW_{16} die in Bild 31 dargestellten Varianten.

Auf den ersten Blick sieht das Verfahren komplizierter aus als es ist.
Praktisch hat man

- die Dualzahl von rechts beginnend in Vierergruppen einzuteilen,
  $(Z)_2 = 11010110 = 1101\ 0110$
- die dualen Wertigkeiten zuzuordnen,

$$(Z)_2 = \underset{2^3\ 2^2\ 2^1\ 2^0}{1\ \ 1\ \ 0\ \ 1} \quad \underset{2^3\ 2^2\ 2^1\ 2^0}{0\ \ 1\ \ 1\ \ 0}$$

Ziffernwert	Bezeichnung der Ziffer im Hexadezimalsystem
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Bild 31:
Ziffernwerte und ihre Bezeichnung im Hexadezimalsystem

- den Wert jeder Vierergruppe zu bestimmen,

$$(Z)_2 = \overset{1}{2^3}\ \overset{1}{2^2}\ \overset{0}{2^1}\ \overset{1}{2^0} \qquad \overset{0}{2^3}\ \overset{1}{2^2}\ \overset{1}{2^1}\ \overset{0}{2^0}$$

$$\begin{array}{rr|rr} & 1.2^3 = 8 & & 0.2^3 = 0 \\ + & 1.2^2 = 4 & & 1.2^2 = 4 \\ + & 0.2^1 = 0 & & 1.2^1 = 2 \\ + & 1.2^0 = 1 & & 0.2^0 = 0 \\ \hline & 13 & & 6 \end{array}$$

- die Werte der Vierergruppen in der Bezeichnung der Hexadezimalziffern aufzuschreiben unter Beibehaltung der Reihenfolge der Vierergruppen

$$(Z)_2 = \overset{1}{2^3}\ \overset{1}{2^2}\ \overset{0}{2^1}\ \overset{1}{2^0} \qquad \overset{0}{2^3}\ \overset{1}{2^2}\ \overset{1}{2^1}\ \overset{0}{2^0}$$

$$= \qquad 13 \qquad\qquad 6$$

$$= \qquad D \qquad\qquad 6$$

$$(Z)_{16} = D6$$

Der umgekehrte Weg führt zur Dualzahl zurück. Nach nur wenig Übung kann man die Darstellungsformen ineinander im Kopf überführen. Einige Beispiele sollen das Erworbene sichern.

- Beispiele:

• $(Z)_2 \rightarrow (Z)_{16}$: $(11001101)_2 = 1100\ 1101$
$12\ \ 13$
$C\ \ D$
$= (CD)_{16}$

$(10111111)_2 = 1011\ 1111$
$11\ \ 15$
$B\ \ F$
$= (BF)_{16}$

• $(Z)_{16} \rightarrow (Z)_2$: $(A9)_{16} = (A\ \ 9)_{16}$
$1010\ 1001$
$=(10101001)_2$

$(87)_{16} = (8\ \ 7)_{16}$
$1000\ 0111$
$=(10000111)_2$

Zur Kennzeichnung von Zahlen, die in unterschiedlichen Systemen dargestellt sind, werden in der Literatur häufig auch folgende Zeichen verwendet:

D = Dezimalzahl	H = Hexadezimalzahl
B = Dualzahl	Q = Oktalzahl

Eine Zahl ohne Kennzeichnung ist immer eine Dezimalzahl.

2.3.3. Grundschaltungen zur Ablage und Verarbeitung von Zahlen im Rechner

In diesem Abschnitt werden Grundschaltungen zur Ablage und Verarbeitung von Zahlen im Rechner behandelt. Ziel ist eine Einführung in die Begriffe und Modelle der Signalverarbeitung im Digitalrechner. Diese Kenntnisse sind bei der Programmierung von Vorteil. Die Schaltungen beziehen sich auf die Verarbeitung von vierstelligen Dualzahlen (Tetraden). Die Verarbeitung breiterer Dualzahlen (allgemein Bitmuster), z. B. 8-, 16- oder 32-Bitstellen, ist als logische Erweiterung der Bitstellenanzahl erklärbar.

2.3.3.1. Register

Ein Register ist die vergegenständlichte Form einer bestimmten Anzahl von Bitstellen, mit der Möglichkeit, eine Date (Zahl) zu speichern.
Beispielsweise kann ein Register aus mehreren D-Triggern bestehen, wie sie im Bild 32 dargestellt sind.

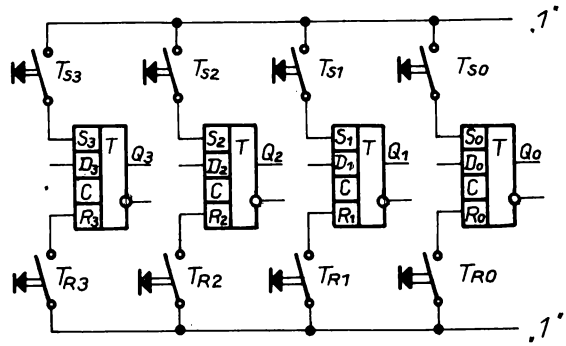


Bild 32: Register zur Speicherung einer Tetrade

Im dargestellten Register besteht die Möglichkeit, eine vierstellige Dualzahl (Tetrade) zu speichern. Im Modell kann es so geschehen, daß zum Beispiel

T_{S3}, T_{S2} und T_{S0} und T_{R1} gedrückt werden.

Als Resultat nehmen die Ausgänge Q folgende logischen Werte an:

$Q_3 = 1$
$Q_2 = 1$
$Q_1 = 0$
$Q_0 = 1$

Vergibt man weiter die Wertigkeiten: $2^3 = 8 = Q_3$
$2^2 = 4 = Q_2$
$2^1 = 2 = Q_1$
$2^0 = 1 = Q_0$,

so wurde im Register der Zahlenwert 13 in dualer Form gespeichert.

- Schieberegister

Nachdem gezeigt wurde, wie in einem Register eine Tetrade gespeichert werden kann, soll eine Schaltung zur Verschiebung der Zahl dargestellt werden. Neu in der Schaltung nach Bild 33 ist

- die Zusammenschaltung der D-Trigger-Taktleitungen zu einem Takt
- die Verbindung des Q-Ausgangs mit dem nachfolgenden D-Eingang.

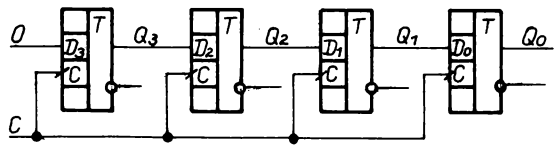


Bild 33: Zusammenschaltung von D-Triggern zu einem Schieberegister

Die Arbeitsweise dieser Art von Schaltungen erläutert man vorteilhaft mit Hilfe von Impulsdiagrammen. Im Register möge als Ausgangszustand die 13 gespeichert sein. Dies sei der Zustand Z_0 der Schaltung (Bild 34).

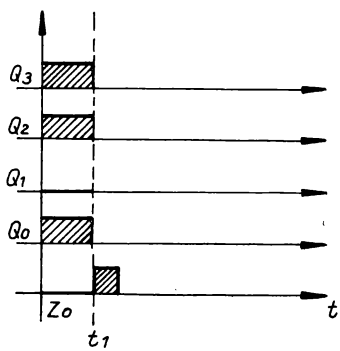


Bild 34: Darstellung des Zustands Z_0 im Impulsdiagramm

Zum Zeitpunkt t_1 erscheint nun eine Vorderflanke von C. Was geschieht?

- Q_3 übernimmt den logischen Wert von D_3 und wird als Folge der Vorderflanke von C den logischen Wert 0 annehmen (siehe Bild 35).
- Q_2 übernimmt den logischen Wert von $D_2 = Q_3$. Zum Zeitpunkt der Vorderflanke ist Q_3 logisch 1, also wird Q_2 den logischen Wert 1 beibehalten (von Q_3 übernehmen).
- Q_1 übernimmt den logischen Wert von $D_1 = Q_2$, wird also logisch 1 werden.
- Q_0 übernimmt den logischen Wert von $D_0 = Q_1$ und wird logisch 0.

In Bild 35 ist der sich neu einstellende Zustand Z_1 der Schaltung dargestellt.

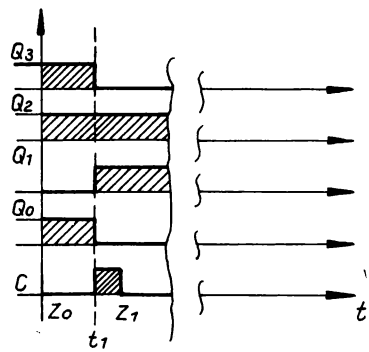


Bild 35: Impulsdiagramm zur Erzeugung des Zustands Z_1

Wie lange dieser Zustand von der Schaltung eingenommen wird, hängt davon ab, wann erneut eine Vorderflanke von C die Schaltung aktiviert. Erscheint nun eine erneute Vorderflanke von C, so reagiert die Schaltung wie Bild 36 zeigt.

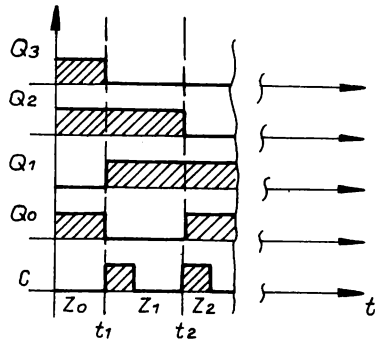


Bild 36: Impulsdiagramm zur Erzeugung des Zustands Z_2

Wie lange dieser Zustand aufrecht erhalten bleibt, hängt wiederum nur vom Erscheinen einer erneuten Vorderflanke von C ab. Bild 37 zeigt die Schaltung im Zustand Z_4, nachdem noch zweimal Vorderflanken von C die Schaltung aktivierten.

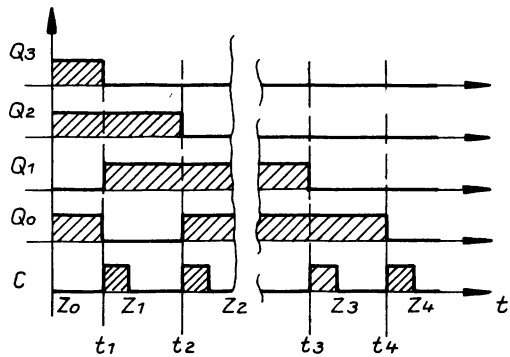


Bild 37: Impulsdiagramm des Schieberegisters nach 4 Vorderflanken von C

Der Zustand Z_3 stellt sich zum Zeitpunkt t_3 ein, indem

Q_3 logisch 0 von D_3 übernimmt
Q_2 logisch 0 von Q_3 übernimmt
Q_1 logisch 0 von Q_2 übernimmt
Q_0 logisch 1 von Q_1 übernimmt und

der Zustand Z_4 ist dadurch gekennzeichnet, daß alle Q-Ausgänge der D-Trigger den logischen Wert 0 tragen. Man sagt, die Null von D_3 wurde durch das Register geschoben und hat es "genullt". Hätte man immer konstant an D_3 eine logisch 1 angelegt, so wäre diese durchgeschoben worden und alle Q-Ausgänge wären im Zustand Z_4 logisch 1.

Aus dem Impulsdiagramm nach Bild 37 läßt sich jedoch noch eine andere Aussage treffen, die in der bisherigen Betrachtung noch keine Rolle spielte. Schaut man sich die Impulsfolge auf der Leitung Q_0 an, so findet man, daß es genau die Impulsfolge ist, die dem Zustand Z_0 des Registers entsprach.
Der Zustand Z_0 war gekennzeichnet durch:

$$Q_0 = 1,\ Q_1 = 0,\ Q_2 = 1,\ Q_3 = 1$$

Die Impulsfolge, die Q_0 beim "Schieben" annimmt, zeigt Bild 38.

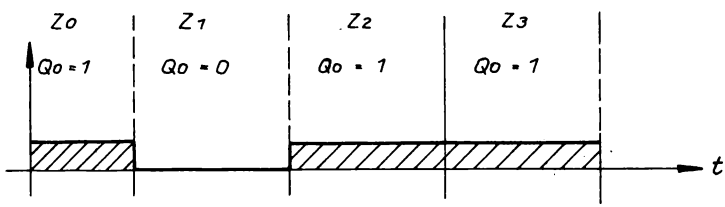


Bild 38: Impulsfolge von Q_0

Damit wurde die Tetrade parallel in das Register eingegeben und seriell aus dem Register herausgeschoben. Man nennt eine solche Arbeitsweise Parallel-Serien-Umsetzung (Bild 39).

In den Bildern 40 und 41 ist gezeigt, was in einem zweiten Schieberegister, dessen Eingang D_3 mit dem Ausgang Q_0 des ersten zusammengeschalten wird, geschieht.

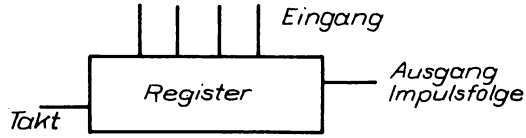


Bild 39: Parallel-Serien-Umsetzer

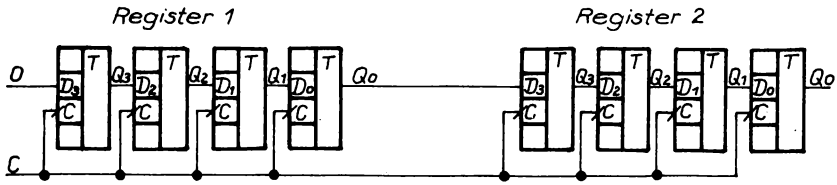


Bild 40: Schaltung zweier Schieberegister

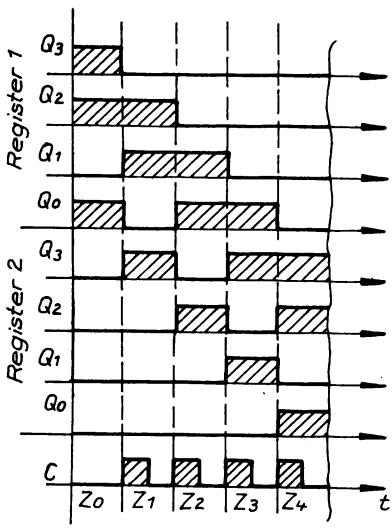


Bild 41: Impulsdiagramm der Zusammenschaltung zweier Schieberegister

Das Impulsdiagramm zeigt, daß der Zustand Z_0 im Register 1 dem Zustand Z_4 des Registers 2 entspricht. Das heißt nichts anderes, als daß die Zahl 13 vom Register 1 in das Register 2 "umgeladen" wurde. Das Register 2 realisiert eine Serien-Parallel-Umsetzung. Über D_3 wird dem Register 2 eine serielle Eingangsimpulsfolge zugeführt. Die Information liegt an den Q-Ausgängen der 4 D-Trigger parallel an.

- Umlaufregister

Will man die Information, die Date, die Zahl, ins Register 2 schieben und trotzdem im Register 1 erhalten, so muß man den Ausgang Q_0 des Registers 1 zum eigenen Eingang D_3 des Registers 1 zurückführen. Die Date "läuft" nun beim Takten (schieben) im Register 1 "um" (Umlaufregister).
Die Bilder 42 und 43 zeigen dazu Schaltung und Impulsdiagramm. Der Zustand Z_4 stimmt mit dem Zustand Z_0 überein. Über Q_0 wurde die Impulsfolge trotzdem übertragen.

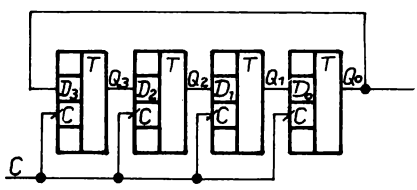


Bild 42: Schaltung eines Umlaufregisters

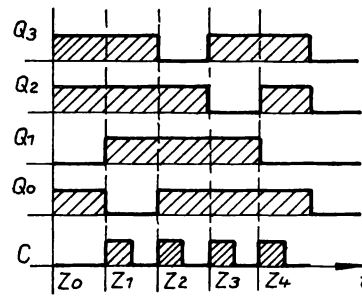


Bild 43: Impulsdiagramm eines Umlaufregisters

2.3.3.2. Adder

Im Bild 44 ist eine logische Schaltung angegeben, die man als Halbadder bezeichnet.

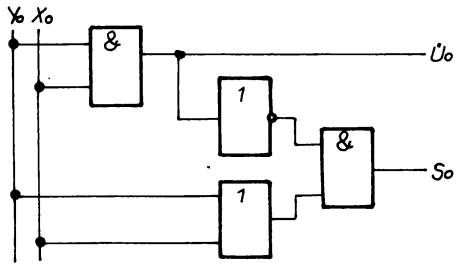


Bild 44: Logikplan eines Halbadders

Zur Analyse dieser Schaltung gibt man sich die möglichen Kombinationen der binären Signale x_o und y_o vor und untersucht, welche Reaktionen die Ausgänge $\ddot{U}_o$ und S_o zeigen. Das binäre Ausgangssignal $\ddot{U}_o$ wird den logischen Wert 1 annehmen, wenn $x_o = 1$ und $y_o = 1$ sind.
Das binäre Ausgangssignal S_o wird den logischen Wert 1 annehmen, wenn

$$(x_o = 1 \text{ oder } y_o = 1) \text{ und } \ddot{U}_o \text{ nicht} = 1$$

$$= (x_o = 1 \text{ und } \ddot{U}_o = 0) \text{ oder } (y_o = 1 \text{ und } \ddot{U}_o = 0).$$

Somit ergeben sich für die Eingangskombinationen von x_o und y_o folgende Ausgangssignale:

x_o	y_o	$\ddot{U}_o$	S_o
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Das sind aber genau die logischen Signalwerte, die bei der dualen Addition entstehen können.

Bitstelle 1. Operand
\+ Bitstelle 2. Operand
= Summe und Übertrag

Folgende Zwischenergebnisse sind möglich:

0	0	1	1
+0	+1	+0	+1
0	1	1	10

Bei der dualen Addition von 1 + 1 = 10 ist es so, daß das Summenergebnis = 0 erscheint und ein Übertrag in die nächste Stelle, so als ob man im Dezimalen über die Zehn hinaus addiert.

- Addition mehrstelliger Dualzahlen

Als Beispiel sollen die Zahlen 9 und 5 addiert werden.
Die dezimale Addition ergibt:

$$\begin{array}{r} 9 \\ +\ 5 \\ \hline 14 \end{array}$$

Die 4 ist das Ergebnis der Summe, die 1 ist der Übertrag in die nächste Spalte.
Die duale Addition der Dualzahlen 1001 + 0101 erfolgt genau so, wie im Dezimalen, nur daß der Übertrag beim Erreichen der 2 erfolgt, also

$$\begin{array}{rl} 1001 & \\ +\ 0101 & \text{Übertrag in die nächste Bitstelle} \\ \hline 1110 & \text{Summe der Bitstelle} \end{array}$$

Sollen also mehrstellige Dualzahlen addiert werden, muß als 3. Eingangsgröße eines Adders der Übertrag der zuvor addierten Bitstelle eingehen. Es reicht demnach die Schaltung des Halbadders nicht aus.

- Volladder

Eine Adderschaltung, die als Eingangsgröße die Bitstellen x_i und y_i der Dualzahlen x und y als auch den Übertrag der Bitstelle i - 1, also $Ü_{i-1}$ verarbeitet, nennt man Volladder. Die Bilder 45, 46, 47 zeigen den Logikplan, die Schaltbelegungstabelle und das Symbol eines Volladders.

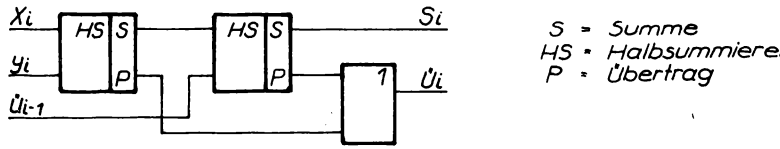
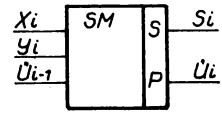


Bild 45: Logikplan des Volladders

Eingang			Ausgang	
x_i	y_i	$\ddot{U}_{i-1}$	S_i	$\ddot{U}_{i-1}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Bild 46: Schaltbelegungstabelle eines Volladders



SM = Summierer

Bild 47: Symbol des Volladders

Technisch realisiert man die Eingangsgröße $\ddot{U}_{i-1}$ als Verzögerung der Ausgangsgröße $\ddot{U}_i$ genau um einen Takt.
Bild 48 zeigt eine mögliche Grundschaltung zur Realisierung der "Verschiebung".
Der D-Trigger schaltet den logischen Wert des D-Eingangs nur durch, wenn eine Vorderflanke von C das Tor öffnet.
Erst dann wird der Übertrag $\ddot{U}_i$ dem Adder wieder zugeführt.
Ist nun gesichert, daß die nächste Bitkombination an x_i und y_i anliegt, so wird der Übertrag von der vorigen Bitkombination mit verarbeitet.

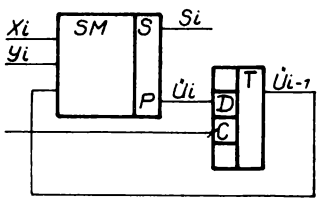


Bild 48: Schaltung zur Realisierung der Übertragungsverschiebung

2.3.3.3. Addierwerk für Tetraden

Ein komplettes Addierwerk regelt neben der direkten Addition die Zuführung der einzelnen Bitkombinationen an den Adder. Die wesentlichen Baugruppen sind

- 1 Schieberegister
- 1 modifiziertes Umlaufregister, genannt Akkumulator
- 1 Volladder mit Übertragsverschiebung

Bild 49 zeigt eine mögliche Schaltung.

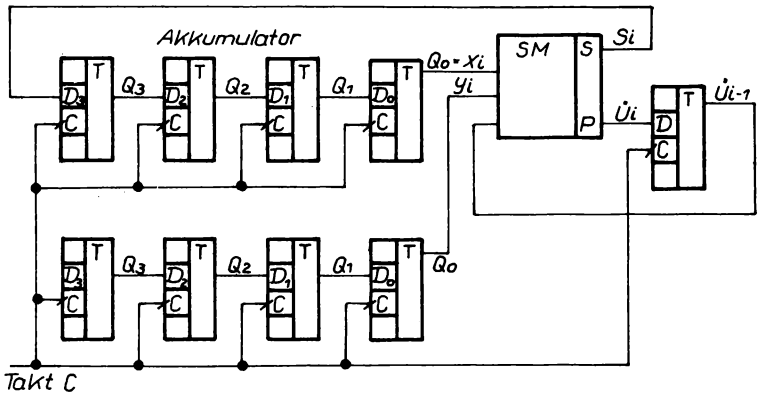


Bild 49: Serienaddierwerk für Tetraden

Neben den schon bekannten Grundschaltungen ist insbesondere bemerkenswert, daß das Additionsergebnis (S_i) in den Akkumulator so hineingeschoben wird (über D_3), wie der dort befindliche eine Operand in den Adder. Somit erfüllt dieses Register zwei Funktionen. Vor der Addition steht in ihm ein Operand, nach der Addition das Ergebnis. Das wesentliche dieser Schaltung ist die synchrone Taktung aller Baugruppen durch den Systemtakt C. Er sichert die serielle Zuführung der Bitstellen der Operanden und die stellenwertrichtige Verschiebung der Überträge.

Die exakte Arbeitsweise ist daran gebunden, daß der Systemtakt genau 4 Vorderflanken von C liefert, wenn die Register aus 4 Bitstellen bestehen. "Verschlucken" oder "Verhaspeln" hat in dieser Schaltung katastrophale Folgen.

In dem dargestellten Addierwerk werden die Bits der Operanden seriell verarbeitet. Deshalb nennt man diese Form der Schaltung ein Serienaddierwerk.

Es gibt auch Paralleladdierwerke. In diesen werden die Q-Ausgänge der D-Trigger der Register direkt den Volladdern zugeführt, die für jede Bitstelle installiert sind. Für die 0-te Bitstelle benötigt man nur einen Halbadder. Hier kann kein Übertrag von der vorherigen Stelle vorliegen.

Kontrollfragen und Aufgaben:

K10. Welcher Unterschied besteht zwischen der Darstellung einer Dezimalzahl als BCD-kodierte oder direkt-dual-kodierte Zahl?

K11. Was ist ein Register und welcher Zusammenhang besteht zwischen seiner Bitstellenzahl und der Datenverarbeitung?

K12. Was kennzeichnet ein Umlaufregister?

A4. Stellen Sie folgende Dualzahlen in Hexadezimalschreibweise dar!

01101100
11001011
00001111
11110000
11111111

A5. Stellen Sie folgende Hexadezimalzahlen als Dualzahlen dar!

BF
3D
A7
47F3

A6. Vervollständigen Sie das folgende Impulsdiagramm eines 3stelligen Schieberegisters! An D_0 ist ständig logisch 1 gelegt.

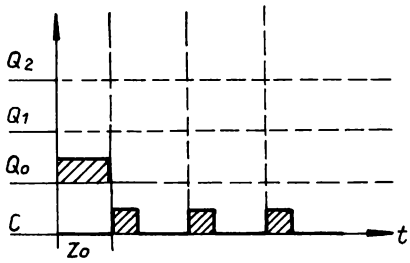


Bild A7

3. Sprachen und grundlegende Begriffe der digitalen Rechentechnik

3.1. Mensch - Rechner - Kommunikation

Die Kommunikation zwischen Nutzer und Rechner erfolgt

- bei der Realisierung des Dialogs am Rechner und
- beim Aufschreiben von Programmen.

Beide Tätigkeiten verlangen eine Verständigungsform zwischen Mensch und Rechner, denn

- der Rechner "versteht" letztlich nur seine Maschinensprache und
- der Mensch möchte sich nicht mit den ihm fremden Nullen und Einsen herumquälen.

Um dieses Problem zu lösen, wurden technische Sprachen entworfen, die eine Formulierung der Programme und Dialogkommandos in einer menschenfreundlichen Form ermöglichen. Solche Sprachen nennt man "Problemorientierte Sprachen", weil der Mensch das ihn interessierende technische oder technologische Problem aufschreiben kann, ohne sich um die Null-Eins-Sprache des Rechners zu kümmern.

Die Null-Eins-Sprache des Rechners nennt man Maschinensprache. Jeder Rechner hat seine eigene Maschinensprache. Programme, die in einer Maschinensprache aufgeschrieben sind, nennt man Maschinenprogramme (auf die Maschine zugeschnitten).

- Maschinensprachniveau

Das Aufschreiben von Programmen mit Hilfe der Maschinensprache (Maschinenprogramm) durch den Menschen hat den

- Nachteil, daß man unbequem und störanfällige Null-Eins-Folgen aufschreiben muß, hat aber den
- Vorteil, daß beim Aufschreiben des Programms die besonderen Eigenheiten des speziellen Maschinenkodes beachtet werden können und damit Programme entstehen, die dem Rechner optimal "liegen". Das zeigt sich dann in einer sehr kurzen Rechenzeit der Programme.

Bei der Gestaltung des Nutzer-Rechner-Dialogs schlägt der schon gezeigte Nachteil voll zu Buche, ohne den Vorteil.

Um den Vorteil voll zu nutzen und den Nachteil möglichst zu beseitigen wurden eingeführt

1. die Zusammenfassung der Null-Eins-Folgen in oktaler bzw. hexadezimaler Form. Der Dialog erfolgt über entsprechende Tastaturen (Oktal-, Hexadezimaltastaturen) und
2. eine neue Sprachform, die Assemblersprache.

- Assemblersprachniveau

Das Aufschreiben der Programme und die Gestaltung des Dialogs wird durch Folgen von festgelegten alphanumerischen Zeichen, sogenannte Mnemoniks (Symbole) realisiert. Für jede Null-Eins-Folge in der Maschinensprache wird im wesentlichen in der Assemblersprache ein Mnemonik definiert.

Beispielsweise für

- die Eingabe von Daten das Symbol: IN
- die Ausgabe von Daten das Symbol: OUT
- das Laden eines Registers das Symbol: LD
- die Addition von Daten das Symbol: ADD

Es leuchtet sofort ein, daß diese Form der Kommunikation schon wesentlich menschenfreundlicher ist, als jene der Null-Eins-Folgen Form. Programme, die in einer Assemblersprache aufgeschrieben sind, nennt man Assembler-Quellprogramme.

Da der Rechner letztlich jedoch nur seine Maschinensprache "versteht", taucht nun das Problem der Überführung des Assembler-Quellprogramms in das Maschinenprogramm auf. Diese formale Arbeit realisiert ein Programm, das im Rechner gespeichert ist. Man nennt es ASSEMBLER. Der ASSEMBLER überführt Assembler-Quellprogramme in Objektprogramme, die dann durch den Rechner realisiert werden können.
Trotz des großen Fortschritts stört noch die Maschinennähe der Programmiersprache. Zu umständlich ist die Formulierung schon einfachster zum Beispiel mathematischer Zusammenhänge, wie c = a + b. Im Prinzip muß man dafür in der Assemblersprache schreiben:

LD A,a	Lade den Akkumulator A mit a
LD B,b	Lade das Register B mit b
ADD	Addiere
LD C,A	Lade das Register C mit dem Inhalt vom Akkumulator (Im Akkumulator steht nach der Addition das Ergebnis von a + b)

- problemorientiertes Sprachniveau

Die Zusammenfassung der genannten Mnemonikfolge zu einer Anweisung der Form c = a + b führt zur problemorientierten Sprache. Beim Aufschreiben der Gleichung c = a + b muß man nicht an einen Rechner denken. Es ist allein wichtig, daß dieser mathematische Zusammenhang zwischen a, b und c das vorliegende Problem löst.
Im Laufe der Zeit entstanden verschiedene problemorientierte Sprachen, die weltweit Verbreitung gefunden haben, so zum Beispiel

- ALGOL : ALGOrytmic Language (algorithmische Sprache)
- FORTRAN: FORmula TRANslation (formale Übersetzung)
- COBOL : COmmon Bussiness Oriendet Language (kommerzielle Sprache - Ökonomie)
- BASIC : Beginners Allpurpose Symbolic Instruction Code (symbolischer Instruktionskode für Anfänger)
- PL1 : Programming Language one (Programmiersprache 1).

Programme, die in einer problemorientierten Sprache geschrieben sind, nennt man Compiler-Quellprogramme. Das Vorwort "Compiler" deutet schon auf das Programm hin, das die Übersetzung des vorliegenden Quellprogramms in eine Form des Assemblersprachniveaus realisiert. Ein solches Programm nennt man COMPILER.
Firmen, die Rechner herstellen oder vertreiben, bieten für den speziellen Rechner stets auch COMPILER für die üblichsten problemorientierten Sprachen an. Somit kann der Nutzer davon ausgehen, daß er, gleichgültig in welcher Sprache er sein Programm notiert hat, das notwendige Übersetzungsprogramm vorliegen hat. Anders ist es der Fall bei der Nutzung von Kleincomputern. Hier einigt man sich aus Gründen der Sparsamkeit, der Einheitlichkeit und des minimierten Speicherbedarfs auf eine Programmiersprache und damit auf ein Übersetzungsprogramm. Die Bereitstellung des Objektprogramms erfolgt wie in Bild 50 gezeigt.

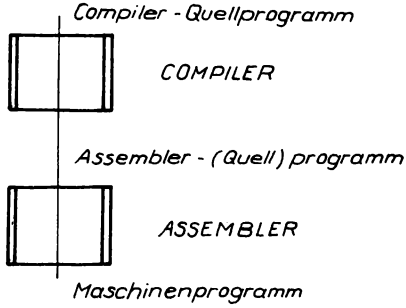


Bild 50: Überführung eines Compiler-Quellprogramms in ein Maschinenprogramm

Für manche problemorientierte Programmiersprache, insbesondere für solche, die den Mensch-Rechner-Dialog ausgezeichnet unterstützen, erfolgt keine geschlossene Überführung vom Quellprogramm in das Objektprogramm. Man nennt eine solche Arbeitsweise "interpretierende" Arbeitsweise und die Programme, die diesen Prozeß realisieren INTERPRETER.
Der Interpreter nimmt Teile des Quellprogramms, überführt diese in die Maschinensprache (Kode) und startet den Rechner zur Ausführung des Teilprogramms. Dann nimmt er sich den nächsten

Teil des Quellprogramms und verfährt entsprechend. Eine typische Interpretersprache ist BASIC.

- Entwicklungstendenzen

Weltweit wird gegenwärtig an zwei Problemen gearbeitet. Erstens geht es darum, die Compiler und Interpreter so zu vervollkommnen, daß in bezug auf die Laufzeit der Objektprogramme solche Zeiten wie bei der Programmierung auf Maschinenniveau erreicht werden. Die Lösung dieses Problems ist besonders wichtig für die effektivere Programmentwicklung in der Mikrorechentechnik zur Steuerung von Echtzeitprozessen in Geräten, Maschinen und Anlagen.

Zweitens geht es darum, die Rechner mit einer sogenannten "intelligenten Oberfläche" auszurüsten. Der grundlegende Gedanke besteht darin, daß zum Beispiel

- Ökonomen
- Technologen
- Konstrukteure
- Meßtechniker usw.

einen ganz bestimmten Wortschatz besitzen, mit dem sie ihre fachlichen Probleme beschreiben. Diese Fachsprache gilt es, dem Rechner anzubieten.
Beide Problemkreise sind in Anfängen bzw. vereinzelt einer Realisierung zugeführt. In den nächsten Jahren ist mit einer verbreiteten Anwendung zu rechnen.

3.2. Daten

Nachdem schon mehrmals Begriffe, wie Daten, Befehle, Anweisungen und Programme benutzt wurden und bewußt die mannigfaltigen Vorstellungen nicht beseitigt wurden, um den übergeordneten Zusammenhang nicht zu zerreißen, ist es nun höchste Zeit, diese Begriffe im Sinne der Informationsverarbeitung durch ihre Arbeitsdefinition festzuschreiben.

Unter Daten soll nach /1/ Information verstanden werden, die zum Zwecke der Verarbeitung auf einem Rechner formuliert wurde.

Dabei unterscheidet man prinzipiell zwei Formen von Daten, den

- Logischen Datenbestand des Menschen und den
- Physischen Datenbestand des Rechners, des Automaten, des Speichermediums.

Das Wort "logisch" deutet darauf hin, daß der Mensch seinen benötigten Datenbestand (seine benötigten Informationen) nach logischen, den Daten innewohnenden Zusammenhängen ordnet.

Das Wort "physisch" deutet darauf hin, daß der Rechner, der Speicher, den physischen Gegebenheiten des Speichermediums Rechnung tragen muß. Der Zusammenhang ist der, daß der physische Datenbestand den logischen Datenbestand, vollständig enthält. Bild 51 zeigt, daß dabei die Art der Ordnung in beiden Datenbeständen nicht einheitlich sein muß.

- Dateneinheiten

In Bild 51 sind äquivalente Dateneinheiten logischer und physischer Datenbestände dargestellt.

Logische Dateneinheiten	Physische Dateneinheiten
Zeichen (Buchstaben, Ziffern, Sonderzeichen)	Byte = Summe mehrerer Bitstellen (häufig: 8 Bitstellen)
Datenelement = Summe von Zeichen mit einer selbständigen semantischen Bedeutung (Attribut und Wert einer Eigenschaft)	Feld = Summe von Byts, die als Gebiet eines reellen physischen Speichermediums ein Datenelement aufnehmen können. Es gibt Felder variabler Länge (vom Datenelement abhängig) und konstanter Länge. Bei Feldern konstanter Länge spricht man bei 2 Byte von Halbwort 4 Byte von Wort 8 Byte von Doppelwort
Logischer Satz = Menge aller Datenelemente, die zu einem gemeinsamen Oberbegriff (Schlüssel) gehören	Physischer Satz: Gebiet eines reellen Speichermediums, in dem mindestens ein logischer Satz abgelegt ist.

	Physischer Satz = Block, wenn mehrere logische Sätze in einem physischen Satz abgelegt ("geblockt") sind
Logischer Datenbestand = Gesamtheit aller logischen Sätze, die im Sinne eines größeren Aufgabenkomplexes zur Lösung benötigt werden.	Physischer Datenbestand = Gebiet eines reellen Speichermediums, in dem ein logischer Datenbestand abgelegt ist.

Bild 51: Dateneinheiten (nach /2/)

Üblich sind auch die Begriffe Datei (bzw. englisch "file") für bestimmte abgeschlossene Gruppen von Daten. Im einzelnen ist jedoch immer zu prüfen, um welche Dateneinheit und welche Form (logisch oder physisch) es sich handelt.
Files sind abgeschlossene Datengruppen, die unter einem Namen abgelegt und aufgefunden werden.

- Datentyp

Entsprechend ihrer Bedeutung unterscheidet man 2 Grundgruppen von Daten

- Zahlen- und Zeichenkettendaten und
- Befehlsdaten.

Beispiele für Zahlen sind:

ganze Zahlen	(1000; -4; +138)
Festkomma-zahlen	(0,438; 18,3)
Gleitkomma-zahlen	$(1{,}37.10^{4}; -4{,}8.10^{-2})$

Beispiele für Zeichenketten sind: OMA, LEIPZIG, BASIC
Beispiele für Befehle sind: Addiere, Eingabe, Ausgabe

Im Sprachgebrauch hat sich eingebürgert, den Begriff Daten im engeren Sinne als Bezeichnung für Zahlen- und Zeichenkettendaten zu verwenden. Obwohl nicht exakt, kann man es doch akzeptieren, da der Begriff im Sinne der "Rechendaten", die einem Programm zur "Bearbeitung" übergeben werden oder der "Ergebnisdaten", die ein Programm erzeugt, gemeint ist.

- Gliederung von Befehlsdaten

Ein Befehl besteht im Sinne der Informationsverarbeitung aus Zeichen:

- Im Maschinensprachniveau sind das die Zeichen "Null" und "Eins".
- Im assembler- und problemorientierten Sprachniveau sind es wieder Buchstaben, Ziffern und Sonderzeichen.

Der Bedeutung nach soll ein Befehl ganz allgemein eine nicht mehr unterteilbare Beauflagung eines Rechners sein. Mit dieser Definition dient der Befehl als Grundelement der Strukturierung der Befehls-Daten in:

Befehl
Anweisung
Programm
Programmsystem.

- Anweisung: Zusammenfassung mehrerer Befehle

  Beispiel:

Befehle	Anweisung
Eingabe a Eingabe b Addiere Speichere	c = a + b

- Programm: Geordnete Folge von Befehlen und Anweisungen zur Lösung einer bestimmten Aufgabenstellung, aufgeschrieben in Maschinen-Assembler- oder problemorientierter Sprache

  Beispiele:

  - Programm zum Sortieren von Zahlen nach ihrer Größe
  - Programm zur Berechnung des arithmetischen Mittelwertes einer Meßwertreihe
  - Programm zur Steuerung eines Motors-Rechtslauf-Linkslauf-Stopp.

  Die Ordnung der Folge der Befehle und Anweisungen richtet sich nach den

  Algorithmen,

  die zur Lösung der Problemstellung genutzt werden. (Eine Definition des Begriffs Algorithmus erfolgt im Teil 2 dieser Lehrmaterialreihe.)

- Programmsystem: Variable Anzahl von Programmen zur Lösung einer bestimmten Aufgabenklasse.

Beispiele:

Programmsystem zur Realisierung der arithmetischen Verknüpfung von Daten

- Addition
- Subtraktion
- Multiplikation
- Division
- Potenzieren

Programmsystem zur Realisierung der statistischen Auswertung von Meßdaten

- Mittelwertbildung
- Standardabweichung
- Korrelationsanalyse

Kontrollfragen und Aufgaben:

K13. Welche Sprachniveaus kennen Sie? Welche Vor- und Nachteile kennzeichnen ihren Zusammenhang?

K14. Was verstehen Sie unter den Begriffen ASSEMBLER, COMPILER und INTERPRETER?

K15. Welcher Zusammenhang besteht zwischen Zeichen, Satz, Datenbestand?

K16. Was kennzeichnet die Begriffe File und Block?

Lösungen der Aufgaben

A 1. Bild A1: analog kontinuierliches Signal
Bild A2: analog diskontinuierliches Signal
Bild A3: analog kontinuierliches Signal
Bild A4: analog kontinuierliches Signal
Bild A5: diskret kontinuierliches Signal (binär)
Bild A6: analog kontinuierliches Signal

A 2.

Ziffern	BCD-Kodierung 8421
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

A 3.

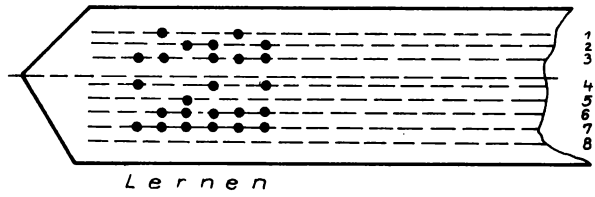


Bild A8

A 4. 6C
CB
OF
FO
FF

A 5. 10111111
00111101
10100111
0100011111110011

A 6.

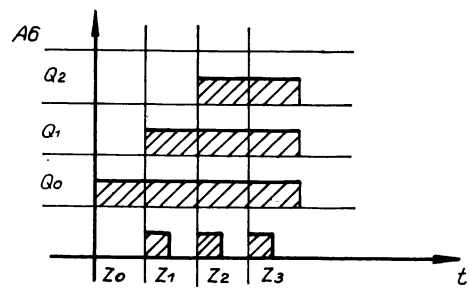


Bild A9

Literaturverzeichnis

/1/ Fritzsch, W.: Prozeßrechentechnik. Berlin: VEB Verlag Technik 1984.

/2/ Autorenkollektiv: Wissensspeicher Datenverarbeitung Berlin: Verlag Die Wirtschaft 1982.

/3/ Schnabel, U.; Bräuning, G.; Heinold, H.: Informationsverarbeitung für Ingenieure. Berlin: VEB Verlag Technik 1982.

/4/ Paulin, G.: Kleines Lexikon der Mikrorechentechnik. Berlin: VEB Verlag Technik 1982.

/5/ Hartmann, G.: Praxis der elektronischen Datenverarbeitung. Berlin: VEB Verlag Technik 1970.

/6/ Schwarz, W.; Meyer, G.; Eckhardt, D.: Mikrorechner. Berlin: VEB Verlag Technik 1980.

Institut für Rationalisierung
der Elektrotechnik/Elektronik
Zentralstelle
für Aus- und Weiterbildung
des Industriebereiches
Elektrotechnik/Elektronik

Lehrmaterial für die
Weiterbildung

Informationsverarbeitung mit Kleincomputern

2

Algorithmierung Teil 1

Kreutzer

Elektrotechnik
Elektronik
Automation
DDR

Steffen Kreutzer

Informationsverarbeitung mit Kleincomputern

2

Algorithmierung, Teil 1

Herausgeber
Institut für Rationalisierung der Elektrotechnik/Elektronik
Institutsteil Dresden
Zentralstelle für Aus- und Weiterbildung
des Industriebereiches Elektrotechnik/Elektronik
Karl-Marx-Straße, Dresden
8080

Autor: FSD Dipl.-Ing. Steffen Kreutzer
Ingenieurschule für Wissenschaftlichen Gerätebau
"Carl Zeiss" Jena

Gutachter: Dr.rer.nat., Dipl.-Math. Siegfried Neuber
Ingenieurschule für Maschinenbau und
Elektrotechnik Berlin

Bearbeiter: Dipl.-Gwl. Heinz Rüdger
Institut für Rationalisierung der
Elektrotechnik/Elektronik
Zentralstelle für Aus- und Weiterbildung

1. Auflage
IR Dresden ZSB
Druckgenehmigungs-Nr.: Ag 682/043/86
Druck und Herstellung: NOWA DOBA Bautzen III-4-9
Redaktionsschluß: 30. 04. 1986
Bestell-Nr.: T.2.04.0002
Vorzugsschutzgebühr: 2,00 M

Vorwort

Im vorliegenden Heft wird auf den Aufbau und die Wirkungsweise von Digitalrechnern sowie auf Grundlagen der Algorithmierung eingegangen. Der Abschnitt "Erstellen von Algorithmen" wird in einem 2. Teil behandelt. Das erschien notwendig, weil in den Lehrveranstaltungen und Weiterbildungskursen diese Aufgabenstellung im Prozeß der Vorbereitung des Rechnereinsatzes die größten Probleme bereitet.

Bei der Beschreibung von Algorithmen wird unter anderem auch auf die problemorientierte Programmiersprache BASIC (Beginners All-purpose Symbolic Instruction Code) zurückgegriffen. Die Programmiersprache BASIC hat sich als eine sehr leistungsfähige und besonders für Lernende leicht annehmbare Sprache in vielen "Dialekten" international weit verbreitet. In der DDR existieren zum Beispiel BASIC-Varianten für

- die Bürocomputer A 5120, A 5130 als Bestandteile der Betriebssysteme SIOS oder SCP,
- die mittlere Rechentechnik auf der Basis des Mikrorechnersystems K 1600,
- die Kleincomputer KC 85/1, KC 85/2 und Z 9001 sowie andere Rechner auf der Basis des Prozessors U 880, wie der MC 80 und seine Nachfolger und eigene Rechnerkonfigurationen vieler Anwender auf der Basis der Module des Mikrorechnersystems K 1520.

Grundlage der hier verwendeten BASIC-Version bildet eine Interpretervariante der Firma Luxor Elektronik GMBH, die unter der Bezeichnung "ABC-80-Basic" in vielen Lernsystemen implementiert ist.

Bei der Beschreibung von Algorithmen wird jedoch soweit wie möglich von Besonderheiten dieser Sprachvariante abgesehen, und es werden solche Sprachelemente ins Zentrum gerückt, die in allen Varianten große Gemeinsamkeiten aufweisen. Abweichungen zur BASIC-Variante des KC 85/2 sind im Text ausgewiesen.

Inhaltsverzeichnis

		Seite
1.	Aufbau und Wirkungsweise von Digitalrechnern	5
1.1.	Gesamtstruktur	5
1.2.	Zentraleinheit	6
1.2.1.	Baugruppen und Struktur	6
1.2.2.	Interner Speicher	7
1.2.2.1.	Speichertyp RAM	7
1.2.2.2.	Speichertyp ROM	10
1.2.2.3.	Nutzung der ROM-Module	12
1.2.2.4.	Nutzung der RAM-Module	12
1.2.3.	Programmiermodell für einen RAM	13
1.2.3.1.	Speicherverwaltung	13
1.2.3.2.	Ablegen von ganzen Zahlen in Integerspeicherplätzen	16
1.2.3.3.	Ablegen von reellen Zahlen in Realspeicherplätzen	18
1.2.3.4.	Ablegen von Zeichen in Zeichenkettenspeicherplätzen	21
1.2.4.	Verarbeitungseinheit	23
1.2.4.1.	Steuerung der Programmabarbeitung	23
1.2.4.2.	Verarbeitung der Daten	25
1.2.5.	Ein-Ausgabe-Tore	26
2.	Grundlagen der Algorithmierung	29
2.1.	Aufgabenstellung - lösender Algorithmus - Programm	29
2.2.	Darstellungsformen von Algorithmen	33
2.3.	Darstellung von Algorithmen mittels PAP	36
2.4.	Darstellung von Algorithmen mittels der problemorientierten Programmiersprache BASIC	39
2.4.1.	Variablenbezeichnung	39
2.4.2.	Zahlen	40
2.4.3.	Mathematische Ausdrücke	41
2.4.4.	Erste Anweisungen	44
2.5.	Analyse von Algorithmen	49
2.5.1.	Drei Algorithmenelemente	49
2.5.1.1.	Summenbildung	49
2.5.2.2.	Zählen	53
2.5.2.3.	Prüfen	54
2.5.2.	Ein Algorithmus zur Mittelwertbildung	56
	Lösungen zu den Aufgaben	59

1. Aufbau und Wirkungsweise von Digitalrechnern

1.1. Gesamtstruktur

Die Gesamtheit der Gerätetechnik eines Digitalrechners bezeichnet man mit dem Begriff Hardware ("harte Ware"). Ausgehend von der Zentraleinheit, die alle Geräte und Baugruppen des "eigentlichen" Rechners umfaßt, gliedert man die weiteren Baugruppen und Geräte in sogenannte Peripherien.

Geräte der 1. Peripherie:	Darunter versteht man die Ein- und Ausgabegerätetechnik, die unmittelbar mit der Zentraleinheit korrespondiert.
Beispiele dafür sind:	Tastatur, Bildschirm, Lochbandleser, Lochbandstanzer, Druckwerke, externe Speicher (Magnetbandspeicher, Folienspeicher, Magnetplattenspeicher usw.), Prozeßkoppelbaugruppen zur unmittelbaren Anbindung eines Rechners an einen technisch-technologischen Prozeß.
Geräte der 2. Peripherie:	Im wesentlichen faßt man darunter die Gerätetechnik zusammen, die maschinenlesbare Datenträger erstellen oder Daten zur Verarbeitung auf dem Rechner in irgend einer Weise vorbereiten.
Beispiele dafür sind:	Lochbanddatenerfassungsgeräte, Lochkartendatenerfassungsgeräte, Sortierautomaten, Fakturierautomaten, Datensammelsysteme
Geräte der 3. Peripherie:	Unter diesem Begriff faßt man die Gerätetechnik zusammen, die den Rechenprozeß maßgeblich unterstützen und Hilfsgeräte darstellen.
Beispiele dafür sind:	Klimatechnik, Stromversorgung usw.

Typisch für die weitere Entwicklung der Gerätetechnik ist, daß auch die Geräte der 1. und 2. Peripherie durch eigene, integrierte Rechnereinheiten gesteuert werden.
Die dargestellte Gliederung in Peripherien ist für den Gesamtbereich der Rechentechnik möglich, sinnvoll erscheint sie unbedingt in großen Rechenzentren. Weniger sinnvoll erscheint die Einteilung für Ingenieurarbeitsplätze. In diesem Bereich spricht man besser von

- Zentraleinheit
- Dateneingabe- und Datenausgabegeräten:
  - Tastatur
  - Bildschirm
  - Drucker
- Externen Speichern:
  - Kassettenmagnetbandgeräte
  - Folienspeicher (Floppy-Disk-Speicher)

1.2. Zentraleinheit

1.2.1. Baugruppen und Struktur

Die Zentraleinheit eines Digitalrechners besteht im wesentlichen aus den Baugruppen

- Interner Speicher
- Verarbeitungseinheit
- Ein-Ausgabetore.

Das Zusammenwirken dieser Baugruppen wird vorwiegend durch ein sogenanntes Bussystem realisiert (Bild 1).

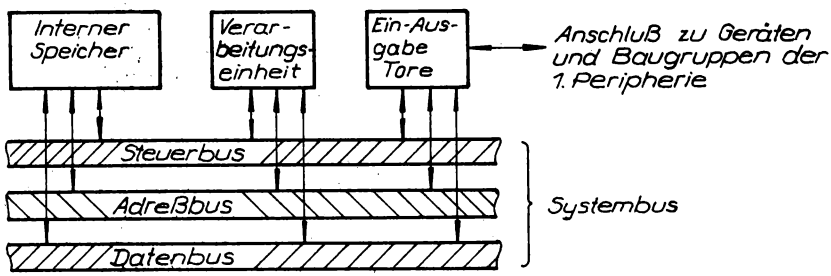


Bild 1: Busstruktur der Zentraleinheit

Unter einem Bus versteht man eine Anzahl von Leitungen zur Übertragung von

- Steuerinformationen	Steuerbus
- Daten	Datenbus
- Adressen	Adreßbus.

Das Strukturprinzip besteht darin, daß jeder Baugruppe, die angeschlossen ist, alle Informationen potentiell zur Verfügung stehen.

1.2.2. Interner Speicher

Interne Speicher werden gegenwärtig mit integrierten Schaltkreisen der Mikroelektronik aufgebaut. Nach dem Wirkungsprinzip unterscheidet man zwei Speichertypen, den Speicher mit wahlfreiem Zugriff (RAM) und den "Nur-Lese"-Speicher (ROM).

1.2.2.1. Speichertyp RAM (Random Access Memory)

Dieser Speichertyp gestattet im laufenden Rechnerbetrieb das Lesen und das Beschreiben der Speicherzellen. Unabhängig von der tatsächlichen technischen Realisierung wird hier die logische Struktur und der modulare Aufbau dieses Speichertyps erläutert.
Als Grundelement eines RAM kann ein dynamischer D-Trigger angenommen werden. Im Bild 2 sind Symbol und Wirkungsweise zur Erinnerung nochmals dargestellt.

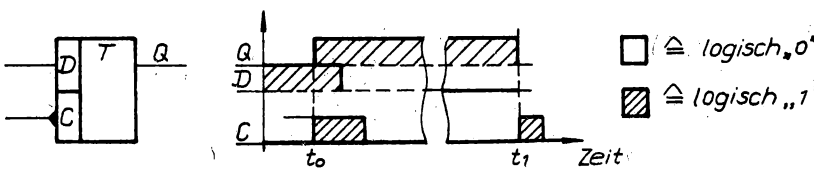


Bild 2: Symbol und Impulsdiagramm eines D-Triggers

Das Impulsdiagramm zeigt, daß

1. der Trigger nur reagiert, wenn C eine Vorderflanke hat;
2. der Ausgang Q infolge der Vorderflanke von C genau den logischen Wert von D annimmt und diesen bis zur nächsten Vorderflanke beibehält.

Damit wird der logische Wert von D im Trigger gespeichert und kann an Q abgerufen werden.

Ausgehend von diesem Speicherelement für 1 Bit (Q kann den logischen Wert 1 oder 0 annehmen) baut man in integrierten Schaltkreisen Matrizen auf, in deren Kreuzungspunkten je ein D-Trigger gedacht werden kann. Oftmals werden Matrizen mit 1024 Bit = 2^{10} Bit = 1 Kilobit = 1 K Bit aufgebaut. Bild 3 zeigt die Zeilen- und Spaltenstruktur eines 1 K Bit-RAM.

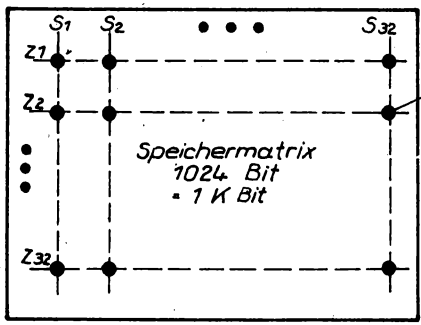


Bild 3: Speichermatrix

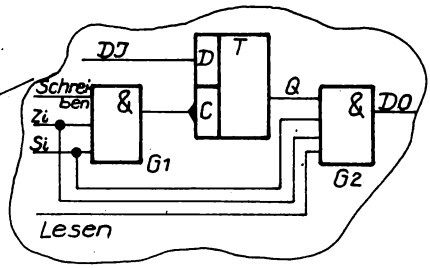


Bild 4: Logik eines Kreuzungspunktes

Aktiviert werden können die Gatter G1 und G2 in Bild 4 nur durch Signale Z_i und S_j, die einer Zeile und einer Spalte entsprechen. Damit wird immer nur ein Kreuzungspunkt aktiviert.

Lesen: Wird auf die Signalleitung "Lesen" eine logische "1" gegeben und ist $Z_i = 1$ und $S_j = 1$, so stellt sich der Datenoutput DO auf den logischen Wert von Q. Der Speicherpunkt wurde gelesen.

Schreiben:

Die logische Funktion des Gatters G1 ist die Und-Verknüpfung der Signale "Schreiben", Z_i und S_j

$$C = \text{"Schreiben"} . Z_i . S_j$$

Werden alle 3 Eingangssignale auf logisch 1 gesetzt, so vollzieht der Ausgang des Gatters G1 eine Vorderflanke von 0 auf 1 und aktiviert den D-Trigger. Der Ausgang Q des Triggers übernimmt den logischen Wert von DJ. Der Speicherpunkt wurde beschrieben.

Entsprechend dem modularen Grundkonzept werden derartige Bitspeichermatrizen zu größeren Einheiten zusammengefaßt. Bild 5 zeigt den prinzipiellen Aufbau eines 1 K Byte RAM, wobei 8 Bit zu einem Byte zusammengefaßt sind.

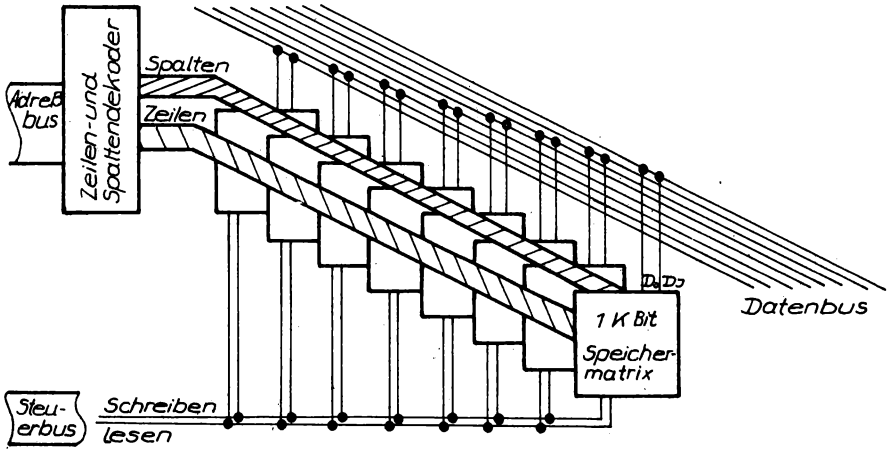


Bild 5: Aufbau eines 1 K Byte RAM

Im dargestellten Beispiel sind 8 Bitspeichermatrizen so zusammengeschalten, daß alle Zeilen der einzelnen Matrizen mit gleichem Index (also beispielsweise alle 8 Zeilen mit dem Index 10 usw.) über eine zentrale Zeile angesprochen werden. Gleiches gilt sinngemeäß für die Spalten.
Wird nun über den Adreßbus der Zentraleinheit eine Adresse eingespeist, so dekodiert der Zeilen- und Spaltendekoder diese Adresse in eine Zeile Z_i und eine Spalte S_j. Die Signale Z_i und S_j aktivieren auf jeder der 8 Matrizen den Kreuzungspunkt i_j gleichzeitig.
Je nachdem, ob über den Steuerbus das Signal "Schreiben" oder "Lesen" kommt, entnehmen die DI (Dateninput) dem Datenbus die 8-Bit breite Date oder geben über die DO (Datenoutput) die 8-Bit breite Date an den Datenbus ab.

1.2.2.2. Speichertyp ROM (Read Only Memory)

Dieser Speichertyp gestattet im laufenden Rechnerbetrieb nur das Lesen der Speicherzellen. Das Beschreiben der Zellen mit Daten erfolgt außerhalb des eigentlichen Rechenbetriebes entweder

- im Produktionsprozeß der Herstellung der Speicherschaltkreise.

  Diese Speicherschaltkreise sind dann fest programmiert. Sie sind somit nur für den programmierten Zweck zu verwenden. Man bezeichnet solche Schaltkreise mit MROM (maskenprogrammierte ROM);

oder

- im Prozeß der Anpassung eines Rechners an zu lösende Aufgaben.
  Das geschieht durch den Rechnerhersteller oder durch den Rechnernutzer. Die Programmierung (Beschreiben) der Schaltkreise erfolgt auf sogenannten Entwicklungssystemen mittels spezieller Programmiergeräte. Solche Schaltkreise bezeichnet man als EPROM (Erasable Programmable-ROM). Wie der Name besagt, sind es umprogrammierbare Speicherschaltkreise. Mittels ultravioletter Strahlung besteht die Möglichkeit, die Daten im Schaltkreis zu löschen. Anschließend kann der Schaltkreis neu programmiert werden.

Obwohl sich die technische Realisierung der ROMs von denen der RAMs wesentlich unterscheidet, ist der modulare Grundaufbau und die logische Struktur als Denkmodell für den Rechnernutzer übernehmbar. Lediglich die Möglichkeit der Datenübernahme vom Datenbus in den Speicherschaltkreis, also das Schreiben, entfällt.
Ein wesentliches Merkmal des Speichertyps ROM besteht darin, daß die einmal eingespeicherten Daten im Nutzerbetrieb ständig erhalten bleiben. Das gilt insbesondere beim Abschalten des Rechners oder bei Netzausfall. Bild 6 zeigt das Modell, das diesem Anliegen gerecht wird.

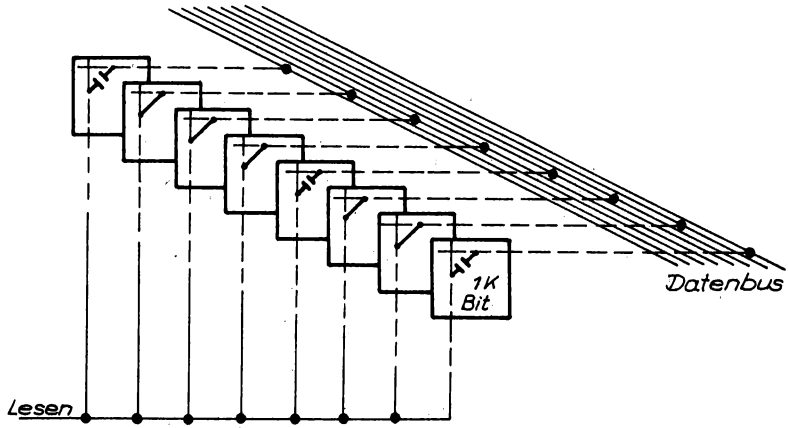


Bild 6: Modellvorstellung zum 1 KByte ROM

Im Kreuzungsbereich der Matrizen liegen Brücken. Bleibt diese Brücke erhalten, so wird die logische Information "1" an die Datenbusleitung übergeben.
Brennt man die Brücke durch, so wird ständig logisch "0" an die Datenbusleitung übergeben.

- Reale Speichermodule und ihre Bewertung
  Die Rechnerindustrie stellt Leiterkarten her, die Speichermodule darstellen. Übliche Speicherkapazitäten sind beispielsweise 4 KByte, 16 KByte, 32 KByte, aber auch 64 KByte und schon 1 MByte. Es besteht die Tendenz, auf der Basis des immer höheren Integrationsgrades in der Mikroelektronik, die Speicherkapazität je Fläche weiter zu erhöhen.
  Somit ist man als Rechnernutzer in der Lage, die Speicherkapazität "seines" Rechners in den gegebenen Abstufungen selbst zu bestimmen.
  Die Speicherkapazität ist dabei ein wesentlicher Faktor zur Bewertung eines Speichers und darüberhinaus einer ganzen Rechenanlage.
  Ein weiteres Bewertungskriterium ist die sogenannte Zykluszeit. Man versteht darunter die Zeit, die beim Lesen und wieder Beschreiben einer Speicherzelle (Byte) vergeht.
  Typische Werte liegen im Bereich von etwa 200 Nanosekunden.

1.2.2.3. Nutzung der ROM-Module

Der ROM-Speicherbereich wird genutzt, um Programme und Konstante im Speicher abzulegen, die beim Rechnereinsatz über eine längere Einsatzzeit oder überhaupt keine Änderung erfahren sollen.

Im wesentlichen sind das

- Programme, die den Rechnerbetrieb an sich im engen Zusammenwirken mit der Hardware sichern. Ein aufeinander abgestimmtes System solcher Programme bezeichnet man als Betriebssystem eines Rechners. Betriebssysteme werden in den Rechnerfirmen erarbeitet und für kleine Rechner in ROMs dem Nutzer zur Verfügung gestellt (MONITOR, COMPILER, ASSEMBLER)
- Programme, die von vielen Nutzern häufig benötigt werden und somit nutzerfreundlichen Charakter tragen. Beispiele dafür sind Programme, die oft gebrauchte mathematische Funktionen realisieren, wie y = sin x usw.. Auch solche Programme werden in Rechnerfirmen erarbeitet und als STANDARD-Programmsysteme bereitgestellt.
- Programme, die vom Nutzer des Rechners selbst geschrieben wurden und sehr häufig genutzt werden müssen.

In ROMs bereitgestellte Programme bezeichnet man als "Residente Software" (Software = weiche Ware, im Gegensatz zu Hardware).

1.2.2.4. Nutzung der RAM-Module

Im RAM-Speicherbereich bringt der Rechnernutzer seine Anwenderprogramme unter. Häufig werden dabei relativ große Datenmengen benötigt, wenn man beispielsweise an Rechnerarbeitsplätze im Bereich der technologischen Vorbereitung der Produktion, der Produktionsüberwachung oder den Bereich der Konstruktion denkt.

Die Eingabe der Programme und Daten erfolgt dabei entweder über die Tastatur oder externe Speichermedien, wie Magnetbändern, Magnetplatten, Disketten oder Lochbändern.

Auch Teile vom Betriebssystem eines Rechners, die weniger häufig benötigt werden oder sehr speicherplatzintensiv sind, werden auf externen Speichermedien abgelegt und bei Bedarf von dort in den RAM-Bereich eingelesen.

Zum dritten wird der RAM-Bereich des internen Speichers vom Betriebssystem des Rechners selbst genutzt, um variable Daten zwischenzuspeichern. Dieser RAM-Bereich wird durch das Betriebssystem für den Nutzer gesperrt.

Insgesamt nennt man die im RAM abgelegten Programme "variable Software" und speziell die Programme, die von Anwendern selbst geschrieben werden → Anwendersoftware.

1.2.3. Programmiermodell für einen RAM

1.2.3.1. Speicherverwaltung

Ausgehend von den Kenntnissen über den Aufbau eines Speichers besteht die Möglichkeit, durch Angabe einer Adresse eine Speicherzelle (im Beispiel besteht die Speicherzelle aus 1 Byte = 8 Bit) "anzusprechen", "aufzurufen". Man ist nun in der Lage, in diese Zelle ein Bitmuster von der Breite 8 Bit abzulegen, oder ein Bitmuster, das früher abgespeichert wurde, über den Datenbus auszulesen (Bitmuster = beliebige Folge von "1" und "0" im Byte).
Beim Auslesen eines Bytes soll folgendes Bitmuster am Datenbus anliegen:

Bitnummer	7	6	5	4	3	2	1	0
Bitwert	0	1	0	0	0	1	1	0

Fragt man nach der Bedeutung dieses Bitmusters, so findet man keine eindeutige Antwort. Viele Bedeutungen sind sinnvoll denkbar.
Beispiele:

- Die Dezimalzahl 86; wenn man den Bitnummern 0 bis 7 die Wertigkeiten 2^n mit $7 \leq n \leq 0$ zuordnet. Dann wäre $Z = 1.2^6 + 1.2^4 + 1.2^2 + 1.2^1 = 86$.
- Der Buchstabe F; wenn die Kodierung nach dem American Standard Code for Information Interchange (ASCII) zugrunde gelegt wird.
- Der Ladebefehl LD B,M; wenn die Maschinenbefehlstabelle des Mikroprozessors U 880 D zugrunde gelegt wird.

Die Beispiele sollen genügen, obwohl weitere Interpretationen möglich sind. Wenn eine eindeutige Rückgewinnung der vollständigen Information aus dem Bitmuster allein nicht möglich ist, so muß gesichert sein, daß "derjenige" der zu einer Speicherzelle "greift" genau weiß, welche Art (Typ) von Date "er" ablegt oder liest.

Allgemein gesagt bedeutet das; der Speicher muß verwaltet werden.
Auf der Ebene der Maschinen - bzw. Assemblerprogrammierung muß der Programmierer diese Verwaltung selbst realisieren. Das erfolgt durch das Anlegen eines Speicherbelegungsplanes. Aus diesem ist zu ersehen, welche Adresse des Speichers mit welcher Art von Date und mit welchem Wert belegt ist (→ Mikrorechner als Steuerrechner).

Auf der Ebene der problemorientierten Programmierung übernimmt die Speicherverwaltung ein Dienstprogramm (LADER, COMPILER). Dieses Programm sichert die getrennte Abspeicherung von Anweisungen und Befehlen eines Programms sowie benötigte Zahlen und Zeichen. Dazu ist es erforderlich, daß sich der Programmierer an Regeln hält, die in einer Sprachsyntax festgelegt sind, damit das Dienstprogramm klar "erkennt", um welchen Datentyp es sich handelt.
Man kann also als Programmierer (Anwender des Rechners) auf dieser Ebene der Rechnerkommunikation von dem Modell ausgehen, das im Bild 7 gezeigt ist.

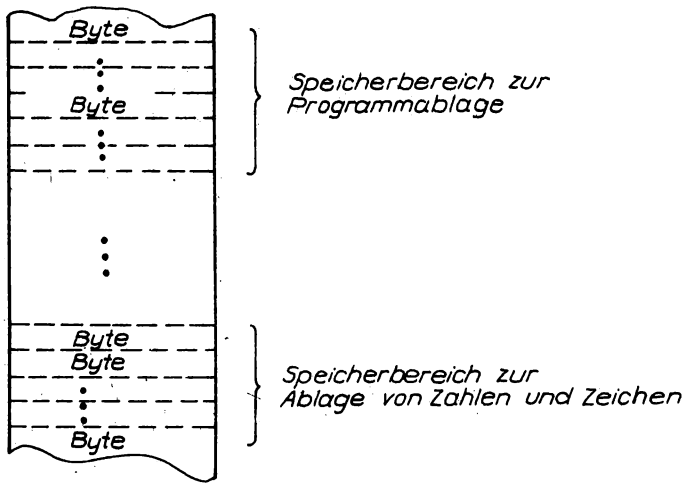


Bild 7: Prinzip der Aufteilung des RAM-Bereiches

Dabei muß dem Nutzer nicht bekannt sein, unter welchen Adressen die Bereiche innerhalb des gesamten Adressumfangs liegen. Das Dienstprogramm regelt es so, daß

- im Speicherbereich für Programme alle Anweisungen und Befehle eines Programmes in geordneter Folge so abgelegt werden, daß dem ersten Befehl der zweite auf der nächst höheren Adresse folgt und so weiter bis alle Befehle abgelegt sind. Als Beispiel soll die Ablage eines kleinen Programms (c = a + b) im Programmspeicherbereich gezeigt werden (Bild 8).

„Programm" in „problemorientierter Sprache		*Befehlsfolge-beschreibung*	*Ablage des „Programms" als Bitmuster im Speicher (U 880 Maschinencode)*		
Anweisungen		*Befehle*	*Byte-Adresse*	*Byte*	
				7 6 5 4	*3 2 1 0*
1. Anweisung	*A = 3*	*Lade A mit 3*	*100* *101*	*0 0 1 1* *0 0 0 0*	*1 1 1 0* *0 0 1 1*
2. Anweisung	*B = 4*	*Lade B mit 4*	*102* *103*	*0 0 0 0* *0 0 0 0*	*0 1 1 0* *0 1 0 0*
3. Anweisung	*C = A+B*	*Addiere A mit B*	*104*	*1 0 0 0*	*0 0 0 0*
		Transportiere das Ergebnis zu C	*105*	*0 1 0 0*	*1 1 1 1*
			⋮	⋮	⋮

Bild 8: Ablage der Anweisung c = a + b

Wie in Bild 8 zu sehen ist, werden für einen Befehl oft 2 Byte zur Abspeicherung benötigt. (Für andere, hier nicht aufgeschrieben, auch 3 und 4 Byte.)
All das ist auf der Ebene der problemorientierten Betrachtung aber unwichtig. Wichtig ist nur, daß das Ordnungsprinzip eingehalten wird, wovon man sich überzeugen konnte.

Das Dienstprogramm regelt es weiterhin so, daß

- im Speicherbereich für Zahlen und Zeichen die im Programm geforderten - Speicherplätze - organisiert werden. Speicherplätze werden in problemorientierten Sprachen durch Buchstaben und Buchstabenfolgen sowie Buchstaben und Ziffernfolgen, kurz durch Variable definiert. Wie die Definition im einzelnen zu erfolgen hat, ist in der problemorientierten Sprache festgelegt.
  Beispiele für Variable sind: MAX A32 BETA ALPHA4 OTTO usw..

Die angegebenen Beispiele sind Namen für Variable (Speicherplätze). Neben dem Namen muß nun noch der Typ der Variablen bekannt sein. Man unterscheidet im wesentlichen zwischen 3 Grundtypen von Variablen:

Numerische Variable:

Integervariable	= Variable, die nur für ganze Zahlen stehen kann
Realvariable	= Variable, die für reelle Zahlen stehen kann
Zeichenkettenvariable	= Variable, die nur für Zeichenfolgen (Buchstaben usw.) stehen kann.

Das Dienstprogramm reserviert nun im Speicherbereich für Zahlen und Zeichen für jede Variable eine bestimmte Anzahl von Bytes und sorgt für das Ablegen des aktuellen Wertes der Variablen (Bild 9).

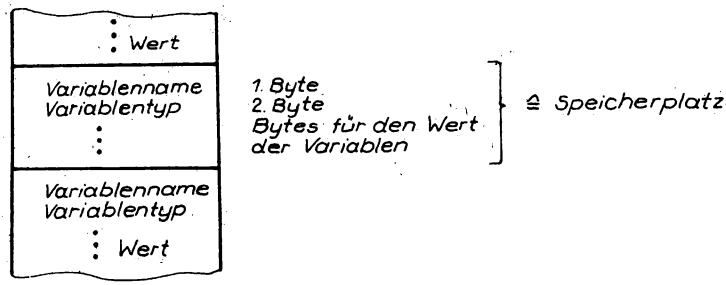


Bild 9: Speicherplatz im RAM

Die Anzahl der Bytes für den Wert der Variablen ist vom Typ der Variablen und von der Programmiersprache sowie dem speziellen Dienstprogramm abhängig. Damit bestimmt die Software eines Rechners im wesentlichen den verwendbaren Zahlenbereich und die Rechengenauigkeit eines Rechners. Beispielsweise kann durch die Software (Betriebssystem) festgelegt sein, daß die im folgenden gezeigte Organisation realisiert wird.

1.2.3.2. Ablegen von ganzen Zahlen in Integerspeicherplätzen

Der Wert der ganzen Zahl wird direkt aus dem Dezimalbereich in den Dualbereich überführt (konvertiert) und in 2 Bytes abgelegt. Damit stehen 16 Bit zur Verfügung, um ganze Zahlen abzuspeichern, wie im Bild 10 gezeigt.

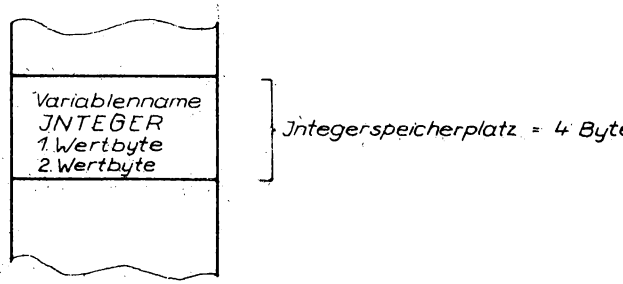


Bild 10: Integerspeicherplatz

Der Wert wird wie im Bild 11 gezeigt abgelegt.

Bytebezeichnung	2. Wertbyte								1. Wertbyte							
Bitnummer	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
festgelegter Stellenwert	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Bild 11: Wertzuordnung im Integerspeicherplatz

Läßt man nur positive ganze Zahlen z zu, so könnte man mit Hilfe dieser 2 Byte einen Zahlenbereich von

$$0 \leqq \text{ganze Zahl} \leqq 65535$$

angeben, wobei die Zahl 0 alle Bitpositionen mit 0 und die Zahl 65535 alle Bitpositionen mit 1 belegen.

Um positive und negative ganze Zahlen abspeichern zu können, wird folgendes festgelegt:

- Für positive Zahlen: Abgelegt wird die dualkodierte Dezimalzahl aus dem Zahlenbereich

  $0 \leqq \text{ganze Zahl} \leqq 32767$ (Bild 12)

Stellenwertigkeit / Beispielzahlen	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Z = 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Z = 230	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0
Z = 32767	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bild 12: Ablegen von positiven Zahlen

Eine positive Zahl wird vom Dienstprogramm in ihrer abgespeicherten Dualform daran erkannt, daß das Bit 2^{15} den Wert Ø trägt.

- Für negative Zahlen: Abgelegt wird der dualkodierte Wert des Ausdrucks

$$A = 2^{16} - |\text{ganze Zahl}|$$

$$A = 65536 - |\text{ganze Zahl}|$$

Die ganze Zahl muß dabei im Zahlenbereich von

$$-32768 \leqq \text{ganze Zahl} \leqq -1$$ liegen (Bild 13)

Stellenwertigkeit / Beispielzahlen	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Z = -1 A = 65535	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Z = -230 A = 65306	1	1	1	1	1	1	1	1	0	0	0	1	1	0	1	0
Z = -32768 A = 32768	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bild 13: Ablegen von negativen Zahlen

Eine negative Zahl wird in ihrer abgespeicherten Dualform daran erkannt, daß das Bit 2^{15} den Wert 1 trägt.

Die hier gezeigte Form der Zahlendarstellung nennt man "Zweierkomplementdarstellung". Sie wird sehr häufig angewendet.

Das angenommene Betriebssystem (und damit der Rechner) würde also in bezug auf ganze Zahlen (Integervariable) einen Zahlenbereich

$$-32768 \leqq \text{ganze Zahlen} \leqq 32767$$

gestatten.

1.2.3.3. Ablegen von reellen Zahlen in Realspeicherplätzen

Zum Aufschreiben von reellen Zahlen sind 2 Formen vorherrschend:

- Der Dezimalbruch, zum Beispiel: 528,27.
- Die halblogarithmische Darstellung, zum Beispiel: $5{,}2827.10^2$.

Allgemein gilt für die halblogarithmische Form die Darstellung

$$Z = \text{Mantisse} \cdot 10^{\text{Exponent}}$$

Die Darstellung des Wertes reeller Zahlen im Speicher des Rechners geht von der halblogarithmischen Darstellung aus. Dabei sind die Mantisse und der Exponent abgelegt. Beispielsweise so, daß 3 Byte für die Mantisse, 1 Byte für das Vorzeichen der Mantisse und 1 Byte für den Exponenten bereitgestellt werden.

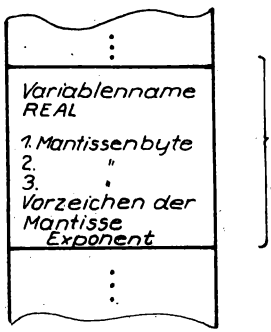


Bild 14: Realspeicherplatz im RAM

Das Ablegen des aktuellen Wertes einer Realvariablen erfolgt dabei nach folgendem Algorithmus:

1. Herstellung der halblogarithmischen Schreibweise in der Form, daß das Komma der Mantisse vor der ersten Ziffer steht (Normalisierung), die größer als Null ist.

   Beispiele: $314{,}73$ $\qquad$ $0{,}31473.10^3$
   $\qquad\quad$ $31{,}473.10^1$ $\qquad$ $0{,}31473.10^3$

2. BCD-Kodierung der Mantissenziffern. Da für jede Ziffer 4 Bit benötigt werden, können in 3 Byte nur insgesamt 6 Ziffern untergebracht werden.

Ziffern	BCD-Kode 8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Beispiel:

1. Mantissenbyte: Ziffern 3 und 1
   0 0 1 1 0 0 0 1

2. Mantissenbyte: Ziffern 4 und 7
   0 1 0 0 0 1 1 1

3. Mantissenbyte: Ziffern 3 und 0
   0 0 1 1 0 0 0 0

Die maximal mögliche Mantisse ist damit 0.999999.

3. Kodierung des Vorzeichens der Mantisse.
   Plus = 0 0 0 0 0 0 0 0
   Minus = 0 0 0 0 0 0 0 1

4. Behandlung des Exponenten. Abgelegt wird der dualkodierte Wert des Ausdrucks:

   A = Exponent + 128

   Der maximal darstellbare Exponent ergibt sich aus:

   $\underline{\text{Exponent}} = A_{max} - 128 = 255 - 128 = 127$

   Der minimal darstellbare Exponent ergibt sich aus:

   $\underline{\text{Exponent}} = A_{min} - 128 = 0 - 128 = -128$

Demnach können folgende Exponenten dargestellt werden:

$= -128 \leq$ ganze Zahl ≤ 127

Im angenommenen Beispiel wird der Exponent = 3 dargestellt als

A = 3 + 128 = 131 1 0 0 0 0 0 1 1

Die Anweisung Z = 314,73 würde vom Dienstprogramm wie folgt erledigt werden:

- Kodierung des Namens der Variablen (z. B. nach ASCII)
- Kodierung des Typs der Variablen
- Abarbeitung des Algorithmus zur Wertbehandlung
- Ablegen im Speicherbereich für Zahlen und Zeichen (siehe Bild 15)

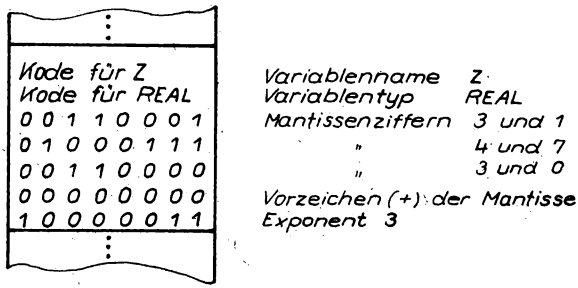


Bild 15: Ablegen der reellen Zahl 314,73

Diese Form der internen Darstellung von Zahlen nennt man Gleitkommadarstellung.
Das angenommene Betriebssystem (und damit der Rechner) verarbeitet reelle Zahlen im Zahlenbereich

$-0.999999 \cdot 10^{127} \leqq$ reelle Zahl $\leqq 0.999999 \cdot 10^{127}$.

Das kleinste Intervall, das dargestellt werden kann, ist 10^{-128}.

1.2.3.4. Ablegen von Zeichen in Zeichenkettenspeicherplätzen

Das Ablegen von Zeichen erfolgt so, daß eine Kodierung vom angegebenen Zeichen in ein durch den Kode festgelegtes Bitmuster der Breite 8 Bit erfolgt. In einem Zeichenkettenspeicherplatz wird meistens eine Folge von Zeichen abgelegt.

OTTO = WIR ␣ WERDEN ␣ BASIC ␣ LERNEN

(OTTO ist der Zeichenkettenvariablenname; ␣ ist das Zeichen für Leerzeichen). Die maximale Länge der Zeichenkette ist im Betriebssystem festgelegt.
Beispiele dafür sind 80 Bytes und 120 Bytes für den "Wert" der Zeichenkettenvariable. Bild 16 zeigt ein Beispiel.

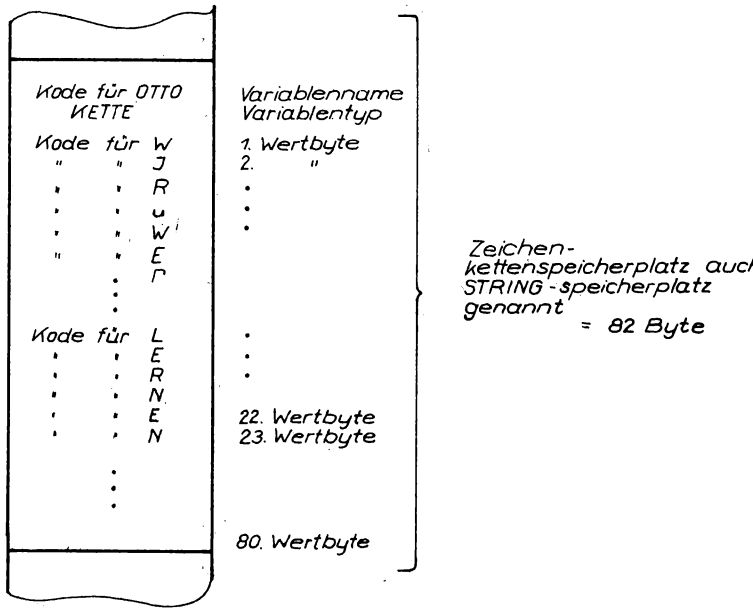


Bild 16: Ablegen von Zeichenkettenvariablen

Abschließend soll zum Ablegen von Zahlen und Zeichen im RAM noch ausgeführt werden, daß die problemorientierten Programmiersprachen zwei Möglichkeiten zeigen, den Typ der Variablen festzulegen:

1. Festlegung des Typs durch Vereinbarungen zu Beginn des Programms hinter festgelegten Kodeworten
   Beispiele: REAL A, BETA, ... , MAX
   INTEGER I, MIN, ... , K
   STRING OTTO, ... , X, Z

(REAL ≙ reelle numerische Variable; INTEGER ≙ ganze numerische Variable; STRING ≙ Zeichenkettenvariable)

2. Festlegung des Typs durch Kennzeichnung nach dem Variablennamen.
   Beispiele: Name ohne Kennzeichen = Real
   Name mit Kennung % = Integer
   Name mit Kennung ¤ = (String) Zeichenkette

1.2.4. Verarbeitungseinheit

Der Name dieser Baugruppe deutet schon auf die Aufgabe hin. Die Verarbeitungseinheit organisiert und realisiert

- die Steuerung der Programmabarbeitung
- die Verarbeitung der Daten.

Gerätetechnischer Kern der Verarbeitungseinheit ist ein Mikroprozessor, der mit seiner Hard- und Softwarestruktur die Steuerung und Verarbeitung bestimmt.

1.2.4.1. Steuerung der Programmabarbeitung

Die Steuerung der Programmabarbeitung geht davon aus, daß das Dienstprogramm zur Speicherverwaltung das Anwenderprogramm im Maschinenkode auf hintereinanderliegende Bytes ablegt.
Die Programmabarbeitung erfolgt durch ein Steuerprogramm, das mit zwei Registern zusammenarbeitet.

- Dem Befehlszeiger (Programmcounter); Ein 16 Bit Register, damit eine vollständige 16 Bit lange Adresse des Speichers darin Platz hat.
- Dem Befehlsregister, in dem der Befehl zur Auswertung zwischengespeichert wird.

Das Steuerprogramm läßt sich vereinfacht durch folgende Schleife beschreiben:

1. Nachdem in den Programmcounter (PC) die Anfangsadresse des Programms, das im Speicher steht, abgelegt wurde, tritt folgender Automatismus in Kraft:
2. Das Bitmuster des PC wird auf den Adreßbus gegeben und damit die Speicherzelle mit dem Befehl des Programms im Speicher gefunden.
3. Der Befehl (das ist der Inhalt der Speicherzelle) wird über den Datenbus in das Befehlsregister transportiert.
4. Der Inhalt des PC wird um 1,2,3 oder 4 erhöht, je nachdem der Befehl 1,2,3 oder 4 Bytes im Speicher beansprucht hatte, somit zeigt der Zeiger auf die Adresse des nächsten Befehls.
5. Auswertung des Befehls, der sich im Befehlsregister befindet.
6. Ausführung des Befehls.
7. Sprung zu 2.

Durch die fortlaufende Erhöhung des Inhalts des Programmcounters (PC) bei jedem Schleifendurchlauf wird gesichert,

daß nacheinander alle Befehle des Programms im Befehlsregister ausgewertet und ausgeführt werden, bis ein STOP-Befehl ausgewertet und ausgeführt wird.

Auswertung des Befehls

Die Auswertung eines Befehls ist im wesentlichen eine Dekodierung. Ausgangspunkt ist das Befehlsregister. Durch die Abarbeitung der Schleife des Steuerprogramms wird dieses Register mit dem aktuellen Befehl geladen. Ausgangsseitig wird das Bitmuster des Registers den Dekodern über ein Ausgabetor zugeführt. Bild 17 zeigt das Logikmodell.

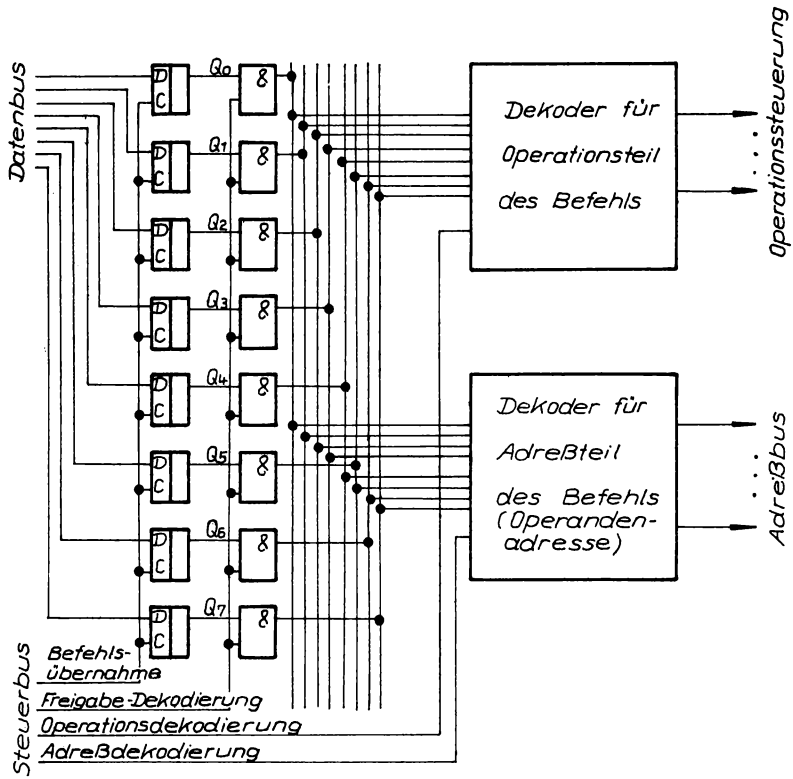


Bild 17: Logikmodell zur Befehlsauswertung

Dabei ist zu beachten, daß ein Befehl immer aus zwei prinzipiellen Teilen besteht. Der eine Teil gibt an, was zu reali-

sieren ist (Operation), der zweite Teil mit wem (Adresse, Operand) etwas zu geschehen hat.
Deshalb müssen diese beiden Teile auch dekodiert werden.
Die Anweisung c = a + b hat beispielsweise aus dieser Sicht

- den Operationsteil - Addition
- den Adreßteil - a, b, c, dabei sind a und b Operandenadressen und c die Ziel-(Ergebnis-) adresse.

Einen Rechner, der diese Befehlsstruktur intern so verarbeiten kann, würde man Dreiadreßmaschine nennen.

Moderne Rechnerkonzepte realisieren jedoch vorwiegend das Einadreßprinzip. Dieses Rechnerkonzept basiert im wesentlichen auf einem speziellen Register, genannt Akkumulator. Der Sinn dieses speziellen Registers besteht darin, daß sich viele Operationen immer direkt auf dieses Register beziehen und vor der Operation der erste Operand und nach der Operation das Resultat dort gespeichert wird. Unter dieser Voraussetzung kann die "3-Adreßanweisung" c = a + b wie folgt zerlegt werden:

1. Einspeichern a - Der Inhalt des Speicherplatzes mit der Adresse a wird in den Akkumulator umgespeichert.
2. Addiere b - Zum Inhalt des Akkumulators wird der Inhalt des Speicherplatzes mit der Adresse b addiert. Das Resultat dieser Addition ist der neue Inhalt des Akkumulators.
3. Ausspeichern c - Der Inhalt des Akkumulators wird in den Speicherplatz mit der Adresse c umgespeichert.

Jeder der drei neuen Befehle hat die Struktur: Operationsteil und Adreßteil mit einer Adresse → Einadreßbefehle.
Derartige Grundbefehle sind in Maschinenbefehlslisten zusammengefaßt dargestellt und dienen dem Programmierer zur Programmierung auf der Ebene der Maschinenprogrammierung (→ Maschinenkode).

1.2.4.2. Verarbeitung der Daten

Die Verarbeitung der Daten erfolgt gerätetechnisch im Mikroprozessor in der Arithmetisch-Logischen-Einheit (ALU- von Arithmetic-Logic-Unit).

Vorwiegend realisieren moderne Mikroprozessoren in ihren ALUs hardwareseitig (z. B. der Mikroprozessor U 880)

- die duale Addition und Subtraktion zweier Bytes
- die bitweise logische Verknüpfung zweier Bytes (AND, OR, XOR)
- den Größenvergleich zweier Bytes (< , = , >)
- die Verschiebung des Bitmusters im Byte eines Registers oder einer Speicherzelle.

Das ist nicht viel, aber ausreichend, Deutlich ist an dieser Stelle der Ausführungen der Stellenwert der Software zu erkennen. Denn alle über diese hardwareseitig zu realisierenden Verknüpfungen hinausgehenden Wünsche der Anwender von Rechnern sind in Programmen zu realisieren. Das beginnt schon bei der einfachen Addition zweier ganzer Zahlen, wenn jede 2 Byte-Breite aufweist. Schon dafür muß es ein STANDARD-Programm geben. STANDARDs werden weiter benötigt, um die arithmetische Verknüpfung von Gleitkommazahlen zu realisieren (Addition, Multiplikation, Division, Potenzieren), um Funktionszuweisungen, z. B. sin, zu realisieren und weiteres mehr. Die Systemsoftware, deren Bestandteil die STANDARDs sind, macht den Rechner erst zu dem, als den der Nichteingeweihte ihn von vornherein ansieht.

1.2.5. Ein-Ausgabe-Tore

Die Ein-Ausgabe-Tore realisieren die hardwaremäßige Kopplung der Zentraleinheit mit der Gerätetechnik der 1. Peripherie. Im wesentlichen sind das

- Tastaturen
- Bildschirme
- Drucker
- externe Speicher (Kassettenmagnetbandspeicher, Floppy-Disk-Speicher).

Aber auch die direkte Kopplung der Zentraleinheit an einen technisch-technologischen Prozeß zum Datenempfang und Datensenden kann über die Ein-Ausgabe-Tore gesichert werden. Gerätetechnisch haben sich 3 wesentliche Typen von Ein-Ausgabe-Tore international durchgesetzt.

- Das Parallele-Ein-Ausgabetor (Parallel-Input-Output), abgekürzt mit PIO.
- Das Serielle-Ein-Ausgabetor (Serial-Input-Output), abgekürzt mit SIO.
- Der Zähler-Zeitgeber-Baustein (Counter-Timer-Circuit), abgekürzt mit CTC.

Die Bezeichnung "Parallel" und "Seriell" bei den Toren PIO und SIO beziehen sich auf die Art des Datenaustausches zwischen der Zentraleinheit und dem peripheren Gerät (Bild 18).
Das Pio gestattet die gleichzeitige (geschlossene) Übergabe eines Datenbyte von 8 Bit. Während das SIO nur 1 Bit zum Übergabezeitpunkt t_o übergeben kann, also 1 Byte nur als 8-Bit-Folge übertragen kann.

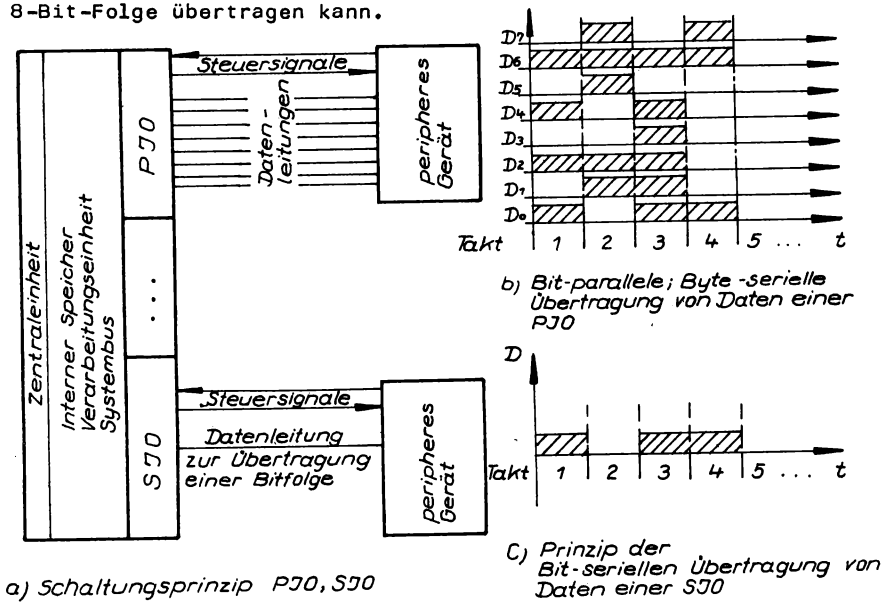


Bild 18: Parallele und serielle Datenübertragung

Die Tore PIO und SIO sind programmierbar. Das heißt, die Tore sind auf den Anwendungsfall spezifizierbar. Das betrifft insbesondere solche Fragen, wie

- Soll der spezielle Baustein als Ein- oder als Ausgabetor arbeiten?

. Darf der Baustein bei Datenangebot des peripheren Gerätes den Programmablauf der Zentraleinheit unterbrechen? (Interrupterlaubnis erteilen oder nicht).

Die Datenübertragung zwischen dem Tor (PIO oder SIO) und dem peripheren Gerät wird in einem Steuersignal (Handshaking) geregelt.

Datenausgabe: - SIO / PIO fordert Geräte zur Datenübernahme
- Gerät meldet die fertige Datenübernahme

Dateneingabe: - Gerät fordert PIO / SIO zur Datenübernahme auf
- PIO / SIO sperrt weitere Eingabe bis interner Datenpuffer (Speicher der SIO, PIO) leer ist.

Der Zähler-Zeitgeber-Baustein (CTC)

Der Baustein kann wahlweise als Zähler oder Zeitgeber arbeiten. Als Zähler programmiert gestattet er Impulse peripherer Einrichtungen aufzunehmen und mit einem einstellbaren Sollwert (Impulsanzahl) zu vergleichen. Beim Erreichen des Sollwertes wird ein Signal an die Verarbeitungseinheit abgegeben, das programmtechnisch ausgewertet werden kann.

Als Zeitgeber programmiert gibt der CTC-Baustein an die angeschlossene periphere Einheit einen Steuerimpuls ab, wenn die eingestellte Solldifferenzzeit vergangen ist.

Kontrollfragen und Aufgaben

K1. Welche Speichertypen kennen Sie? Wodurch unterscheiden sich diese?

K2. Welchen Zusammenhang charakterisieren die Begriffe Speicherplatz und Variable?

K3. Welche Möglichkeiten der Kapazitätsangabe für Speicher kennen Sie? Wie ist ihr Zusammenhang?

K4. Welche Arten der Kennzeichnung von Variablen kennen Sie?

2. Grundlagen der Algorithmierung

2.1. Aufgabenstellung - lösender Algorithmus - Programm

Eine einfache Aufgabenstellung führt zu den Fragen, die in diesem Abschnitt behandelt werden.

Aufgabe: Von zwei reellen Zahlen soll die Summe gebildet und der Betrag der Summe auf dem Bildschirm angezeigt werden.

Es ist gar kein Problem, wenn man vergessen hat, was der Betrag bedeutete. In dem hier betrachteten Fall wäre dieser Zustand sogar wünschenswert, denn dann befinden sich Rechner und Mensch in bezug auf die Lösung dieser konkreten Aufgabe ungefähr auf einer Startlinie. Der Mensch tut nun genau das, was er später dem Rechner zu "sagen" hat:

- Er geht in die Handbibliothek, schlägt nach und liest:

  "Der Betrag einer reellen Zahl x ist gleich x, wenn x größer oder gleich Null ist. Der Betrag von x ist gleich -x, wenn x kleiner als Null ist."

- Er überlegt nun, was praktisch zu tun ist und findet:

  "Die reelle Zahl x habe ich zu prüfen; ist sie nicht negativ (größer oder gleich Null), so ist die Zahl x schon der Betrag. Ist sie negativ (kleiner als Null), so muß ich die Zahl x mit minus Eins multiplizieren.
  Das Ergebnis der Multiplikation ist dann der Betrag der Zahl x.

- Er probiert diese Berechnungsvorschrift an Beispielen aus und bekommt die Bestätigung dafür, daß das Ergebnis unabhängig von der Wahl der reellen Zahlen richtig wird.

Was gefunden wurde, ist ein Algorithmus. Verallgemeinert kann man sagen:

Ein Algorithmus ist ein geordneter Komplex von Regeln, nach denen Eingabegrößen (reelle Zahl) in gewünschte Ausgabegrößen (Betrag der Zahl) zu "überführen" sind. Wichtig ist dabei, daß der geordnete Komplex von Regeln nicht nur für ein Beispiel der Eingabegrößen zum Ergebnis führt, sondern für eine bestimmte Klasse von Eingabegrößen (im Beispiel sind es alle reellen Zahlen) gilt.

Der Algorithmus soll nun noch exakt geordnet aufgeschrieben werden. Die Ordnung soll dabei so exakt sein, daß ein Mensch den Betrag errechnet, ohne das er überhaupt weiß, was der Betrag ist.

In der nachfolgend angegebenen Anweisungsfolge ist zum Beispiel vom Betrag bilden keine Rede mehr. Trotzdem schreibt der den Algorithmus Ausführende den Betrag von x auf das Papier:

1. Wenn x größer oder gleich Null ist, so bilde y = x,
   Wenn x kleiner als Null ist, so bilde y = -x
2. Schreibe y auf ein Blatt Papier
3. Stopp

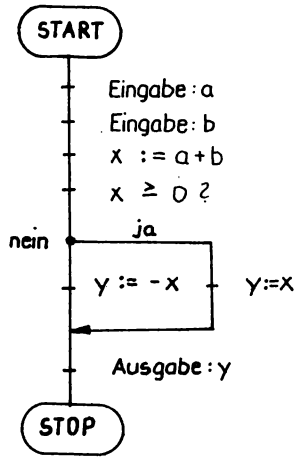
Der Algorithmus ist nun in bezug auf die gegebene Aufgabenstellung dahingehend zu erweitern, daß er die konkrete Aufgabe löst. Die Erweiterung besteht darin, daß von zwei reellen Zahlen erst die Summe zu bilden ist. Von dieser Summe ist dann der Betrag zu bilden. Damit möglichst der gefundene Algorithmus zur Betragsbildung verwendet werden kann, wird für die Summe der Name x verwendet. Der lösende Algorithmus kann dann wie folgt aufgeschrieben werden:

1. Nimm die 1. Zahl
2. Nimm die 2. Zahl
3. Bilde x = 1. Zahl + 2. Zahl
4. Wenn x größer oder gleich Null ist, so ist y gleich x,
   wenn x kleiner als Null ist, so ist y = -x
5. Schreibe y auf ein Blatt Papier
6. Stopp

Im Wissen darum, daß der Algorithmus von einem Rechner bearbeitet werden soll, ist der "Ausdruck" dieser Sachlage anzupassen. Man schreibt deshalb kürzer:

1. Eingabe von a (a: Variable für die 1. Zahl)
2. Eingabe von b (b: Variable für die 2. Zahl)
3. x = a + b
4. Wenn $x \geq 0$ ist, dann y = x, anderenfalls y = -x
5. Ausgabe von y
6. Stopp

Sehr anschaulich kann der Algorithmus grafisch dargestellt werden. Als Beispiel ist in Bild 19 die grafische Darstellung entsprechend der TGL 22451 nach der Programmlinienmethode gezeigt.



In bezug auf die Abarbeitung der Anweisungsfolge gilt:

- Flußlinie von oben nach unten abarbeiten oder
- bei Verzweigung in Pfeilrichtung.

Das Zeichen ":=" bedeutet "ergibt sich aus". Es weist darauf hin, daß z. B. die Anweisung

x := a + b aus zwei Teilen besteht:

1. Teil: Bildung der Summe von a und b
2. Teil: Zuweisung des Ergebnisses zur Variable x.

Bild 19: Grafische Darstellung des Algorithmus Betragsbildung einer Summe

Der im Bild 19 dargestellte Algorithmus löst unsere Aufgabenstellung so, daß die einzugebende 1. Zahl der Variablen a und die 2. Zahl der Variablen b zugewiesen wird. Für den Zahlenbereich gelten keine weiteren Einschränkungen. Es müssen aber reelle Zahlen sein. Die Variablen werden nach den angegebenen Regeln "behandelt", bis der Algorithmus gestoppt wird.
Zur vollständigen Lösung der Aufgabenstellung soll dieser Algorithmus nun in einem Rechner (Rechnerarbeitsplatz) implementiert werden. Implementieren heißt "hineinpflanzen", "anpassen und verankern", aber auch "bereitstellen". Die Implementierung erfolgt durch das Aufschreiben und Eingeben eines Programms.
Das Programm soll in der problemorientierten Sprache BASIC aufgeschrieben werden. Folgende Anweisungsfolge entspricht dieser Forderung:

```
1Ø INPUT A
2Ø INPUT B
3Ø X = A + B
4Ø IF X ≥ Ø THEN Y = X ELSE Y = -X
5Ø PRINT Y
6Ø STOP
```

Hinweis: INPUT bedeutet Eingabe
IF " wenn
THEN " dann
ELSE " sonst
PRINT " Ausgabe

Vergleicht man den Algorithmus mit dem Programm, so wird deutlich, daß

- das Programm den Algorithmus als wesentliches Element enthält und in seiner Struktur realisiert,
- das Aufschreiben des Algorithmus in Form eines Programms jedoch an Regeln der Sprache gebunden ist.

Die im Beispiel verwendeten BASIC-Regeln sind:

- Angabe der Zeilennummer. Sie muß eine positive ganze Zahl sein. Daß im Beispiel die Zeilennummern in 10er Schritten wachsen, ist üblich, aber nicht notwendig (Vorteilhaft bei Einfügungen).
- Verwendung von festgelegten Symbolen:
  INPUT, IF, THEN, ELSE, PRINT, STOP.
- Verwendung von nur großen Buchstaben für Variable:
  A, B, X, Y.

Die wesentliche Einschränkung erfährt der für reelle Zahlen allgemeine Algorithmus jedoch durch die im Interpreter festgeschriebene Verarbeitungsbreite der reellen Zahlen. Bei der Realisierung des dargestellten Programms legt der Interpreter die konkreten Zahlen in Speicherplätzen ab, die die Namen A, B, X, Y tragen.
Obwohl alle 4 Speicherplätze unterschiedliche Adressen im Speicher haben, so haben sie doch gemeinsam, daß in jedem Speicherplatz nur Zahlen im Bereich von

$$\pm 0.999999 \cdot 10^{127}$$

gespeichert werden können. Die spezielle Organisation der Abspeicherung und Verarbeitung von Zahlen durch den Interpreter schränkt die Anwendung des Programmes ein

- auf den Zahlenbereich
- und auf die Genauigkeit von 6 Mantissenstellen.

Der dargestellte Sachverhalt tritt bei jeder Implementierung eines Algorithmus in einem konkreten Rechner auf. Er ist objektiver Natur. Wie stark die Einschränkungen sind, beziehungsweise wie sie den Nutzer stören, hängt vom Compiler, vom Interpreter und von der Syntax der problemorientierten Sprache ab.
Die Bearbeitung einer Aufgabenstellung mit dem Rechner erfordert, so wurde gezeigt, eine Vorbehandlung. Diese Vorbehandlung gliedert sich im wesentlichen in

- die Analyse der Aufgabenstellung: Ziel der Analyse ist es, wesentliche Teilkomplexe der Aufgabe aufzufinden, die in ihrer Gesamtheit die Aufgabe bestimmen.
- die Algorithmierung: Ziel der Algorithmierung ist die Erarbeitung von Algorithmen zur Lösung der Teilkomplexe und die Darstellung des lösenden Gesamtalgorithmus.
- die Programmierung: Ziel der Programmierung ist die Implementierung des lösenden Algorithmus in einem speziellen Rechner.

Die zentrale Aufgabe des Bearbeiters der Aufgabenstellung besteht zweifellos im Erarbeiten und Bereitstellen des lösenden Algorithmus. In den weiteren Ausführungen wird deshalb die Algorithmierung im Mittelpunkt stehen.

2.2. Darstellungsformen von Algorithmen

Zur Darstellung von Algorithmen haben sich im wesentlichen bewährt:

- Die sprachliche Darstellung mit lebenden und künstlichen Sprachen, wie zum Beispiel problemorientierten Rechnersprachen und Pseudosprache.
- Die grafische Darstellung in Form von Programmablaufplänen und Struktogrammen.

Im folgenden werden diese Darstellungsformen für das Beispiel der Summen- und Betragsbildung im Komplex gezeigt.

- sprachliche Darstellung:

  • mit einer lebenden Sprache

  1. Eingabe von a
  2. Eingabe von b
  3. x = a + b
  4. Wenn x ≧ 0 ist, dann y = x ansonsten y = -x
  5. Ausgabe von y
  6. Stopp

  • mit BASIC:

```
1Ø INPUT A
2Ø INPUT B
3Ø X = A + B
4Ø IF X ≧ Ø THEN Y = X
   ELSE Y = -X
5Ø PRINT Y
6Ø STOP
```

  • mit Pseudocode:

```
SEQ                                  SEQ ≙ Symbol für eine Folge,(Se-
                                           quenz) von Anweisungen
 Eingabe: a
 Eingabe: b
 x = a + b
 IF (x ≧ 0) THEN Y = X ELSE Y = -X
 END IF
 Ausgabe: y
ENDSEQ
```

- grafische Darstellung mit Programmablaufplänen (PAP) nach TGL 22451

  • Programmlinienmethode

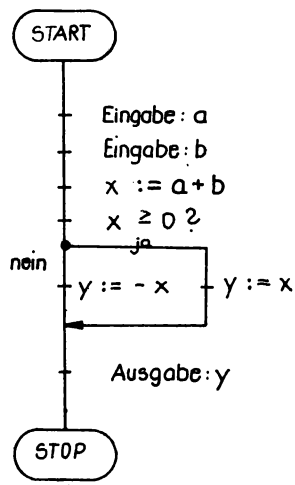


Bild 20: PAP-Programmlinienmethode

  • Kästchenmethode

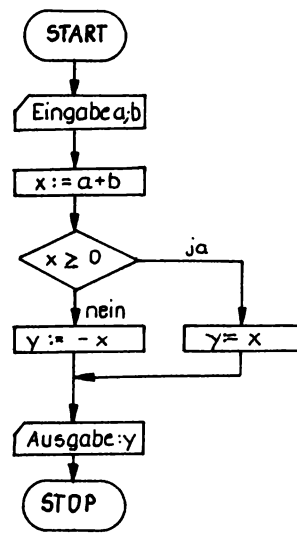


Bild 21: PAP-Kästchenmethode

- grafische Darstellung in Form eines Struktogrammes

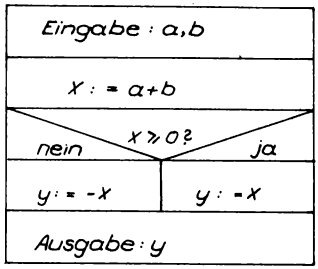


Bild 22:
Struktogramm

Die einzelnen Darstellungsformen von Algorithmen ergänzen sich in ihren Vorteilen. So findet man sie teilweise gleichberechtigt vor. Einige Vor- und Nachteile sollen zeigen, daß alle Darstellungsformen ihre Berechtigung haben.

Darstellungsform	Vorteil	Nachteil
lebende Sprache	Ohne neues Lernen von Symbolen oder Sprachen verständlich	Aufwendige Darstellung bei komplizierteren Algorithmen. Lebendige Sprachen haben oft zwei- oder mehrdeutige Sprachelemente
problemorientierte Sprache (POS)	Kurze, übliche Form des Aufschreibens von Anweisungen in Zeilen, Kann gleichzeitig zur Programmierung verwendet werden	Lernen der Sprache notwendig. Einschränkung des Gültigkeitsbereiches des Algorithmus (Genauigkeit, Zahlenbereich usw.), Einschränkung der Namenwahl für Variable
PAP in Form der Programmlinienmethode	Übersichtlich, änderungsfreundlich, Variablen dürfen beliebige Namen tragen	Neue Symbole müssen gelernt werden
Pseudosprache	Kurze, übliche Form des Aufschreibens von Anweisungen in Zeilen, keine Einschränkungen wie bei POS	Lernen der Sprache notwendig. Kann nicht auf einem Rechner direkt implementiert werden

Darstellungsform	Vorteil	Nachteil
Struktogramm	Vereinigung wesentlicher Vorteile der Sprachbeschreibung in bezug auf - Übersichtlichkeit - Zeilendarstellung - beliebige Namen für Variable	Wenig änderungsfreundlich

In der praktischen Programmierarbeit ist der Trend zu erkennen, daß

im Entwurfstadium der Algorithmierung die grafische Darstellung mittels Programmablaufplänen und die sprachliche Beschreibung mittels Pseudo- oder problemorientierter Sprache erfolgt und

bei der Dokumentation von Algorithmen die Darstellung der Algorithmen durch Struktogramme immer häufiger Anwendung findet.

2.3. Darstellung von Algorithmen mittels PAP (Programmlinienmethode

Im weiteren wird bei der Darstellung der Algorithmen der Programmablaufplan (PAP) nach der Programmlinienmethode und die Sprachbeschreibung mittels der problemorientierten Programmiersprache BASIC verwendet. Dazu sind in Bild 23 wesentliche Sinnbilder als Auszug aus der TGL 22451 dargestellt.
Für die Darstellung der Variablen, Anweisungen und Zeichen im PAP wird vereinbart:

- Namen und Variablen:
  - Alle Bezeichnungen, die in der mathematischen Beschreibung üblich sind und alle Bezeichnungen, die in der Umgangssprache gebräuchlich sind, werden als Namen zugelassen.
    Beispiele: Max als Variablenname für einen Maximalwert
    MIN als Variablenname für einen Minimalwert
    α,ß als Variablenname für einen Winkel usw.
  - Jede Veränderung in der Bezeichnung bedeutete eine andere Variable
    Beispiele: Max ist nicht die Variable max oder MAX, sondern das sind 3 unterschiedliche Variablen.

- Anweisungen:
  - Abkürzungen: "E:" - bedeutet Eingabe; "A:" - bedeutet Ausgabe
    "NL:" - bedeutet "neue Zeile" bei der Ausgabe
  - Alle anderen Anweisungen werden als Frage oder Ergibtanweisung geschrieben.
    Beispiele: Ergibtanweisung: x' := A1 + Max
    Frage : $\alpha > 0$? (Alpha größer Null?)
  - Sollen weitere Anweisungen Verwendung finden, so sind sie im PAP zu erläutern.

- Zeichen
  Erlaubt sind alle mathematisch üblichen Zeichen, wie
  +, -, /, ., sin, arctan, $>$, $<$, $\geq$, $\leq$ u.a.

- Sinnbilder und ihre Bedeutung:

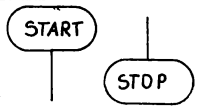
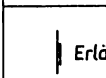
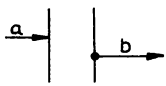
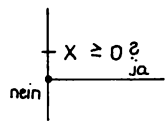
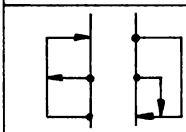
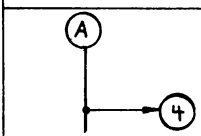
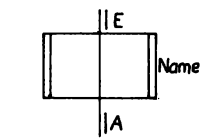
Sinnbild	Bezeichnung und Erläuterung
	Flußlinie mit Zuordnung einer Anweisung. Die Anweisung ist rechts neben die Flußlinie zu schreiben.
	Start- und Stopzeichen
	Nach einem parallel zur Flußlinie verlaufendem Strich können Erläuterungen gegeben werden.
	Zusammenführen (a) und Verzweigen (b) von Flußlinien
	Kennzeichnung der Programmverzweigung nach einer Vergleichsoperation
	Zusammenführen mehrerer Flußlinien von links und rechts
	Zeichenkonnektoren zur Unterbrechung und Fortsetzung der Programmlinie aus zeichentechnischen Gründen.
	Symbol für einen Algorithmus, der komplett in einen anderen integriert wird. Unterprogramm; geschlossener Programmteil. E und A sind Eingangs- und Ausgangsvariable. Rechts neben das Unterprogramm ist der Name zu schreiben.

Bild 23: Sinnbilder zur Darstellung von Programmablaufplänen nach der Programmlinienmethode (Auszug aus TGL 22451)

2.4. Darstellung von Algorithmen mit der problemorientierten Programmiersprache BASIC

(BASIC = Beginners All-purpose Symbolic Instruction Code)
Bei der Darstellung der Algorithmen im Programmablaufplan gelten keine wesentlichen Einschränkungen in bezug auf die Verwendung, z. B. von Namen und mathematischen Symbolen und Zeichen. Das ist bei der Verwendung einer problemorientierten Sprache anders. Hier darf man nur das zur Beschreibung verwenden, was in der Sprache definiert ist. Man könnte nun so vorgehen, daß der gesamt Sprachumfang von BASIC (etwa des Standard-BASIC) angegeben wird, um dann danach zu handeln. Dieser Weg ist jedoch abschreckend und führt nur bei denen zum Ziel, die schon eine Programmiersprache beherrschen. Für den Lernenden ist es wertvoller, mit einem gewissen "Grundwortschaft" zu beginnen und diesen schrittweise auszubauen. So soll verfahren werden.

2.4.1. Variablenbezeichnung

- Namen von einfachen Variablen
  Der Name einer Variablen wird aus einem großen Buchstaben des lateinischen Alphabets oder aus der Folge eines solchen Buchstabens und einer Dezimalziffer 0,1, ...,9 gebildet.
  Beispiele: A; B1; Z9; X; Y3; MØ; O3
  (Um die Ziffer 0 eindeutig vom Buchstaben O zu unterscheiden, schreibt man die Ziffer 0 als Ø).

- Typen von einfachen Variablen
  Die Beispiele der Variablennamen A; B1; Z9; usw. bezeichnen alle Variablen vom Typ "real" (reelle-Variable, im Sinne von Bereich der reellen Zahlen)
  Trägt also der Name der Variable keine weitere "Kennung", so legt der BASIC-Interpreter für diese Variable einen Speicherplatz an,
  . der Beträge von Zahlen im Bereich von $0{,}999999 \cdot 10^{-128}$ bis $0{,}999999 \cdot 10^{127}$ aufnehmen kann. Man nennt diese Variablen auch "Realvariable".
  (Beim KC 85/2-BASIC gilt:
  $9{,}40396 \cdot 10^{-39} \leqq |z| \leqq 1{,}70141 \cdot 10^{38}$

Hinter dem Variablennamen kann auch eine Kennung angegeben werden. Schreibt man z. B.:
A% (A-Prozentzeichen), so legt der BASIC-Interpreter für diese Variable einen Speicherplatz an, der nur ganze Zahlen im Bereich von -32768 bis +32767 aufnehmen kann. Man nennt diese Variablen "Integervariable".
(Beim KC 85/2 gilt $-999999 \leqq z \leqq 999999$)
Die Variablen A und A% dürfen gemeinsam in einem Programm auftreten. Es sind zwei unterschiedliche Variable, und sie bezeichnen unterschiedliche Speicherplätze.
Hinter dem Variablennamen kann auch eine Kennung mit dem Zeichen ¤ angegeben werden. (Beim KC 85/2-BASIC gilt: Kennung ist das Dollar-Zeichen $)

Der BASIC-Interpreter legt für diese Variable einen Speicherplatz an, der eine "ZEICHENFOLGE" aufnehmen kann.
In dem Speicherplatz ist standardmäßig für 80 Zeichen Platz. Eine solche Variable nennt man "Zeichenkettenvariable" oder "Stringvariable". Braucht man weniger oder will man mehr Platz, so kann man mit der Anweisung
Zeilennummer DIM Variablenname¤ = ganze Zahl,
(Beim KC 85/2-BASIC nicht möglich) den Speicherplatz zugeschnitten dimensionieren.
Beispiel: 12Ø DIM A¤ = 42
Der BASIC-Interpreter legt für die Variable A¤ einen Speicherplatz an, der eine Zeichenfolge von maximal 42 Zeichen aufnehmen kann.
Die Variable A¤ darf neben der Variablen A und der Variablen A% im gleichen Programm auftreten.

2.4.2. Zahlen

Der BASIC-Interpreter erkennt
- ganze Zahlen: Sie müssen wie üblich aufgeschrieben werden, dürfen jedoch nur im Bereich von -32768 bis +32767 liegen. Das positive Vorzeichen kann weggelassen werden.
  Beispiele: 123 / 23ØØØ / -43 / +18 /

- reelle Zahlen: Sie müssen im Bereich von $\pm 0.999999 \cdot 10^{\pm 127}$ liegen.
Sie können in zwei Formen aufgeschrieben (notiert) werden, als
Festpunktzahl: 3.14/ -17.348/ 9.437
oder
Gleitpunktzahl: 314 E-2/ -1.7348 E1
9437 E-3
(Dem üblichen Komma entspricht im BASIC der Punkt; E steht für die Basis 10)

2.4.3. Mathematische Ausdrücke

Häufig sind in Algorithmen mathematische Beziehungen darzustellen. Üblicherweise werden sie durch Formeln angegeben, die aus Variablen und mathematischen Verknüpfungsoperatoren bestehen.
- BASIC kennt folgende arithmetische Operatoren:

arithmetischer Operator	Bedeutung	Priorität	Beispiel eines Ausdruckes
+	Addition	3	A + B
-	Subtraktion	3	A - B
*	Multiplikation	2	A * B
/	Division	2	A / B
** oder ↑	Potenzieren	1	A ↑ B

Die Ausführung der Operationen innerhalb eines arithmetischen Ausdrucks erfolgt entsprechend der angegebenen Priorität (Wertigkeit). Die höchste Priorität ist die Priorität 1. Bei gleicher Priorität wird die Abarbeitung von links nach rechts organisiert. Mit Klammern (es sind nur runde Klammern zugelassen) kann die Abarbeitungsfolge verändert werden. Klammern haben die höchste Priorität, die Priorität Ø.

- Beispiele von arithmetischen Ausdrücken

Übliche Schreibweise	BASIC Schreibweise	Erläuterung zur Ausführung der Operation
$\frac{R_1 \cdot R_2}{R_1 + R_2}$	R1*R2/(R1+R2)	Es wird gebildet: (R1+R2), dann R1*R2, dann das Divisionsergebnis. Die Klammer im Nenner muß gesetzt werden, da sonst R1*R2/R1 gebildet wird und dazu R2 addiert wird
$\frac{d^2\pi}{4}$	D↑2*3,14/4	Abarbeitung erfolgt von links nach rechts
$p_o(1+\gamma \cdot \Delta\vartheta)$	P*(1+G*T1)	Es wird gebildet: (1+G*T1) und das Ergebnis wird mit P multipliziert
$\frac{1}{2}m_e \cdot v^2 + h \cdot f_g$	M*V↑2/2+H*F	Es wird gebildet: V↑2, dann mit M multipliziert, dann durch 2 dividiert, dann die Multiplikation H*F und danach die Addition der Teilergebnisse

Achtung! Bei der BASIC-Schreibweise der Variablen ist darauf zu achten, daß bei der Namensbildung nur ein Buchstabe genommen werden darf. Deshalb: $m_e \rightarrow$ M und $f_g \rightarrow$ F. (Es gibt jedoch auch BASIC-Versionen, die als Namen von Variablen mehr als einen Buchstaben zulassen.)

- BASIC kennt folgende Vergleichsoperatoren:

Vergleichs-operatoren	Bedeutung	Priorität
>	größer als	alle Vergleichsoperatoren
>=	größer als oder gleich	haben die Priorität 4
<	kleiner als	
<=	kleiner als oder gleich	
=	gleich	
<>	ungleich	

Werden zwei arithmetische Ausdrücke miteinander verglichen, so besagt die Priorität 4 für die Vergleichsoperatoren, daß erst die arithmetischen Ausdrücke berechnet werden und dann der Vergleich erfolgt.

Bei dem Vergleich

M1 * C1 * T1 = M2 * C2 * T2

werden also erst die Werte M * C * T berechnet und anschließend der Vergleich realisiert.

- Mathematische Standardfunktionen

In jeder problemorientierten Programmiersprache sind gebräuchliche mathematische Funktionen, wie die Sinusfunktion, Cosinusfunktion usw. als sogenannte Standardfunktionen in Form fertiger Programme verankert. Der Nutzer kann sie in seinem Programm dann einfach aufrufen, indem er im Programm z. B. schreibt:

Y = SIN(X).

Es ist nur dafür Sorge zu tragen, daß im Speicherplatz X das Argument (der Wert, von dem der Sinus zu berechnen ist) vorliegt. Im Speicherplatz mit dem Namen Y findet man dann den Sinuswert von X.
Der BASIC-Interpreter kennt folgende mathematische Standardfunktionen:

Standardfunktion	Erläuterung
SIN(A)	sin a; a muß im Bogenmaß angegeben werden
COS(A)	cos a; " " " " " "
TAN(A)	tan a; " " " " " "
ATN(A)	arctan a; Ergebnis im Bogenmaß
LOG(A)[1]	ln a; natürlicher Logarithmus von a > 0
LOG 10(A)[2]	lg a; dekadischer Logarithmus von a > 0
EXP(A)	e^a ;
SQR(A)	$\sqrt{a}$
INT(A)	Größte ganze Zahl kleiner als oder gleich a; INT(2.3)=2
ABS(A)	|a|; absoluter Betrag von a
PI	3.14159
SGN(A)	-1% für a <Ø; Ø% für a = Ø; 1% für a >Ø
RND	Zufallszahl zwischen Ø und Ø.999999
RANDOMIZE[2]	Gibt RND neuen Startwert

1) Beim KC 85/2-BASIC gilt LN(A)

2) Beim KC 85/2-BASIC nicht vorhanden

2.4.4. Erste Anweisungen

Unter einer Anweisung wird die Beauflagung an einen Rechner verstanden, die er im Speicher als Teil eines Programms ablegt. Die geordnete Folge aller Anweisungen bildet dann das geschlossene Programm. Die Ausführung der Anweisungen erfolgt erst nach dem Programmstart.
Der BASIC-Interpreter findet die gewollte Ordnung durch die Suche nach der immer nächst höheren Zeilennummer, hinter der eine Anweisung steht. Er würde also ein Programm

```
1Ø Anweisung 1
25 Anweisung 2
8Ø Anweisung 3
2Ø Anweisung 4
usw.
```

so abarbeiten, als wäre das Programm aufgeschrieben als

```
1Ø Anweisung 1
2Ø Anweisung 4
25 Anweisung 2
8Ø Anweisung 3.
```

Anweisungen müssen demnach als Bestandteil von Programmen in Zeilenform untereinander geschrieben werden. Jede Zeile beginnt mit einer Zeilennummer. Die Zeilennummer muß eine positive ganze Zahl sein.
Die Stufung der Zeilennummern ist beliebig. Eingebürgert hat sich eine Zehnerstufung, da sie sehr einfach Korrekturmöglichkeiten zuläßt.

- Die LET-Anweisung

Durch die Anweisung

```
15Ø LET R3 = R1*R2/(R1 + R2)
```

weist der BASIC-Interpreter den errechneten Wert des arithmetischen Ausdrucks auf der rechten Seite des Gleichheitszeichens der Variablen R3 zu und speichert damit den Wert im Speicherplatz R3. Es erfolgt also eine Wertzuweisung im Sinne von: R3 ergibt sich aus dem Wert des Ausdrucks.

Die allgemeine Form einer LET-Anweisung ist:

```
znr LET Variable = Ausdruck oder Variable oder
                   Konstante oder Standardfunktion
```

Auf der rechten Seite kann demnach statt einem Ausdruck auch eine Variable, eine Konstante oder eine Standardfunktion stehen. LET kann üblicherweise auch entfallen.

Beispiele:
```
12Ø A = B
 13 Z = 4
24Ø X = SIN(Y)
325 A = ATN(Y/X)   Das Argument einer Standardfunktion kann auch ein arithmetischer Ausdruck sein.
1ØØ B = R1*R2/(R1+R2)
```

In den bisherigen Beispielen sind nur Zuweisungen zwischen Realgrößen y gezeigt worden. Die Anweisung

```
N% = K% + 3%
```

ist eine typische Zuweisung zwischen Integergrößen. Diese Art der Anweisung kann der Interpreter schnell abarbeiten. Deshalb ist jeder BASIC-Programmierer gut beraten, wenn er möglichst Integervariable verwendet. Das Kennungszeichen "%" hinter der 3 mutet unsinnig an. Für den Interpreter ist
die 3% eine Zahl, die er sofort mit dem Inhalt von K% verknüpfen kann;
die 3 aber ist eine Zahl, die er erst aus der "real"-Form in die "integer"-Form überführen muß.

Selbstverständlich gilt die Zuweisungsanweisung auch für Stringvariable.
Die Anweisung

```
3Ø G$ = "WIR ␣ LERNEN ␣ BASIC"
```

ist eine Zuweisung einer Zeichenkette zu einer Zeichenkettenvariablen. Diese Anweisung weist dem Speicherplatz G$ die Zeichenkette - WIR ␣ LERNEN ␣ BASIC - zu.

- Steueranweisungen
  - Die IF-THEN-ELSE-Anweisung (bedingte Verzweigung)
    Die Anweisung
    ```
    8Ø IF X>=Ø THEN Y = X ELSE Y = -X
    ```
    steuert die Programmabarbeitung so, daß bei X<Ø dem Wert von Y der Wert von -X zugewiesen wird. Ist X>=Ø, wird dem Wert von Y der Wert von X zugewiesen.
    (IF = Wenn; THEN = Dann; ELSE = Sonst)

Allgemein gilt für diese Art der Anweisung

znr IF Vergleich THEN Anweisung ELSE Anweisung.

Wenn der Vergleich positiv ausgeht, also die Antwort "ja" annimmt, so gilt die Anweisung hinter THEN. In diesem Fall entfällt die Anweisung hinter ELSE.
Ergibt der Vergleich die Antwort "nein", geht er also negativ aus, so gilt die Anweisung hinter ELSE. In diesem Fall entfällt die Anweisung hinter THEN.
Beim KC 85/2 ist vor ELSE ein Doppelpunkt zu setzen, also
znr IF Vergleich THEN Anweisung : ELSE Anweisung

Die GOT-Anweisung (unbedingte Verzweigung)
mit der Anweisung

znr GOTO znr (GOTO = Gehe zu)

kann der Programmierer die übliche Programmabarbeitung verändern. Üblicherweise sucht der Interpreter nach der Abarbeitung einer Anweisung die nächst höhere Zeilennummer.
Anders aber, wenn zum Beispiel die Anweisung

13Ø GOTO 4Ø

realisiert wird. Der Interpreter "springt" von der Zeile 13Ø zurück zur Zeile mit der Zeilennummer 4Ø. Bei der Anweisung

13Ø GOTO 24Ø

würde der Interpreter bei der Anweisung mit der Zeilennummer 24Ø die Programmabarbeitung fortsetzen.

- Einfach Ein-Ausgabe-Anweisungen

BASIC bietet mehrere Eingabemöglichkeiten, insbesondere jedoch vielfältige Formen und Möglichkeiten zur Variation der Ausgabeanweisung.

. Die INPUT-Anweisung:
Die Anweisung 3Ø INPUT K bewirkt, daß der Interpreter an der Zeile 3Ø seine Programmabarbeitung unterbricht.
Er gibt über den Bildschirm ein Fragezeichen aus und wartet auf die Eingabe einer Zahl über die Tastatur.
Der Arbeitsplatzbenutzer wird durch den Interpreter aufgefordert, über die Tastatur eine Zahl einzugeben. Soll beispielsweise die Zahl 2,37 eingegeben werden, so muß

1. die Taste - 2 - gedrückt werden
2. die Taste - . - gedrückt werden (Achtung! "." statt ",")
3. die Taste - 3 - gedrückt werden
4. die Taste - 7 - gedrückt werden.

Die Ziffern und Zeichen werden auf dem Bildschirm hinter dem "?" aufgeschrieben. Damit eindeutig ist, daß keine weitere Ziffer mehr folgt, die Zahl also komplett ist, muß

5. die Taste - ENTER - gedrückt werden. (ENTER ist die Abschlußtaste für Zeilen, Kommandos und Zahlen.)
   Nach dem Drücken der Taste ENTER weist der Interpreter dem Speicherplatz, der hinter INPUT steht, im obigen Beispiel also dem Speicherplatz K, den eingegebenen Wert zu. Das Programm wird mit der Anweisung mit der nächst höheren Zeilennummer fortgesetzt.

Die allgemeine Form der INPUT-Anweisung lautet:

znr INPUT Variable (INPUT = Eingang).

. Die PRINT-Anweisung:

Die Anweisung: znr PRINT Variable
bewirkt, daß der Inhalt des Speicherplatzes der Variablen auf den Bildschirm geschrieben wird. Die Ausgabe erfolgt dabei so, daß auf die Ø-te Position der neuen Zeile das Vorzeichen geschrieben wird, ein "-"-Zeichen bei negativem Wert, ein " ␣ "-Leerzeichen bei positivem Wert der Variablen. Auf die nächsten Positionen der Zeile werden nun die Ziffern geschrieben.
Mit PRINT kann man auch Text auf den Bildschirm schreiben. Mit der Anweisungsfolge

```
1Ø PRINT "EINGABE ␣ VON ␣ X"
2Ø INPUT  X
3Ø PRINT "WERT ␣ VON ␣ X="; X
```

gibt der Interpreter folgendes Bild über den Bildschirm aus:

```
1. Zeile: EINGABE VON X
2. Zeile: ? 3.14              (3.14 wird als Eingabewert
3. Zeile: WERT VON X = 3.14   angenommen).
```

Das Semikolon zwischen "WERT ␣ VON ␣ X =" und X bewirkt, daß der Inhalt von X unmittelbar nach der Textausgabe in der gleichen Zeile erfolgt.

Schreibt man hinter PRINT zwei Variable durch Komma getrennt auf, also beispielsweise

```
125 PRINT A, X
```

so gibt der Interpreter den Wert von A ab der "0"ten Position einer neuen Zeile und den Wert von X ab der "15"ten Position dieser Zeile aus. Da der Bildschirm 80 Positionen je Zeile hat, kann man so maximal 6 Variable in eine Zeile schreiben. Dabei wird die

1. Variable ab Position "0" der Zeile ausgegeben,
die 2. " " " "15" " " " ,
die 3. " " " "30" " " " ,
die 4. " " " "45" " " " ,
die 5. " " " "60" " " " ,
die 6. Variable ab Position "75" der Zeile ausgegeben.

Aufgaben zur Darstellung von Algorithmen mit BASIC

A1. Bestimmen Sie, welche der angegebenen Variablen BASIC-Variable sind!
B3, XZ, Y12, A%, X1, 3, Z, $\bar{X}$, Max
Otto, O, 4Z, G7, B3%, M, M9, BETA

A2. Bestimmen Sie, welche der angegebenen Zahlen BASIC-Zahlen sind!
3,7; 4.37; $3.1\emptyset^4$; 4%; 78,15; 4.31 E-3; -3,7;
14.2^3; 137ØØØ%; 137ØØØ; 37.483757

A3. Bestimmen Sie, welche der angegebenen arithmetischen Ausdrücke in BASIC richtig geschrieben sind!
R2 * 4Z3/3 * X; Z4 * 3% + 4 * K%; XY * 4 * π * 3;
$\frac{3}{4}$ * D↑3 ; A * PI * 3/4 ; M% + 1%

A4. Schreiben Sie für die angegebenen arithmetischen Ausdrücke die BASIC-Version!

$v_o + a.t$; $\frac{v_1 - v_2}{t_1 - t_2}$; $\frac{a.t^2}{2} + v_o.t + s_o$;

$\frac{2\pi . r}{T}$; $\sqrt{v_o^2 + (g . f)^2}$; $\gamma \cdot m_1 \cdot m_2 \cdot (\frac{1}{r_1} - \frac{1}{r_2})$;

2.5 Analyse von Algorithmen

2.5.1. Drei Algorithmenelemente

2.5.1.1. Summenbildung

Es soll folgende Aufgabe gelöst werden:
Ein Algorithmus soll Eingabedaten ständig aufsummieren.
Eine Lösung ist im PAP (Programmablaufplan) Bild 24 dargestellt:

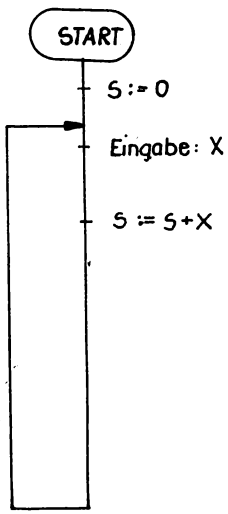


Bild 24: PAP Summe

Wird der Algorithmus gestartet, so nimmt die Variable S den Wert Null an. Das ist der Anfangswert für S.

- Erfolgt nun die Eingabe, so wird die aktuelle Eingabezahl der Variablen X zugeordnet.
- Im nächsten Schritt wird gebildet: S+X; X ist in diesem Zustand die 1. Eingabezahl zugeordnet.
  S ist in diesem Zustand der Wert Null zugeordnet.

Das Ergebnis der Addition O+X wird nun der Variablen S zugeordnet. In diesem Zustand ist S der Wert der 1. Eingabezahl zugeordnet.

- Die Flußlinie führt nun direkt wieder zur Eingabe von X

Der Algorithmus ist "bereit", eine 2. Eingabedate zu verarbeiten.

- Erfolgt nun die 2. Eingabe, so wird die aktuelle Eingabe der Variablen X zugeordnet.
- Im nächsten Schritt wird gebildet S+X; X ist in diesem Zustand die 2. Eingabezahl zugeordnet.
  S ist in diesem Zustand der Wert der 1. Eingabedate zugeordnet.

Das Ergebnis der Addition 1. Eingabedate plus 2. Eingabedate wird der Variablen S zugeordnet; das ist die Summe der bisher eingegebenen Daten.

- Die Flußlinie führt nun direkt wieder zur Eingabe von X.

Der Algorithmus ist "bereit", eine 3. Eingabedate zu verarbeiten.
Der Algorithmus findet selbst kein Ende, sondern "wartet" an der Eingabe.
Deutlich wird der Sinn einer Anweisung der Form S := S + X.

Diese Anweisung realisiert die Summenbildung, weil die Schleife des Algorithmus immer neue Eingabewerte zuführt. Daß das Ergebnis der Summenbildung die Summe aller Eingabewerte ist, sichert die Anfangswertsetzung der Variablen S außerhalb der Schleife auf den Wert Null.

Ein mögliches BASIC-Programm zur Implementierung des Algorithmus wäre:

```
10 S = 0          Hinweis: GOTO bedeutet: Gehe zu.
20 INPUT X                 STOP bedeutet: Stopp des Pro-
30 S = S + X                              gramms
40 GOTO 20
50 STOP
```

Die Schleife des Algorithmus wird im BASIC-Programm realisiert durch die Anweisung

GOTO 20. (Gehe zur Zeile mit der Zeilennummer 20.)

Die Anweisung STOP wird niemals erreicht. Sie dient der Kennzeichnung des Programmendes. Wollte man dieses Programm auf dem Arbeitsplatz testen (ausprobieren), so wäre nichts zu sehen, da keine Ausgaben erfolgen. Gibt man dagegen ein:

```
10 S = 0
20 INPUT X
30 S = S + X
40 PRINT X,S
50 GOTO 20
60 STOP
```

so realisiert man nun in jedem Schleifendurchlauf die Ausgabe des aktuellen Wertes von X und S.

Das Programm soll für die Eingabedaten 2; 3 und -4 getestet werden:

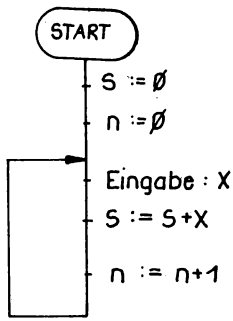
Folge der Anweisungen	Rechnerreaktion und Bildschirmausgabe
S = Ø	Auf dem Speicherplatz S wird eine Ø abgelegt. Es erfolgt keine Bildschirmausgabe.
INPUT X	Auf dem Bildschirm erscheint ein Fragezeichen "?". Der Programmlauf ist unterbrochen. Der Rechner wartet auf die Eingabe einer Date über die Testatur. Diese Date muß eine Zahl sein, da X ein Speicherplatz für Zahlen ist. Es muß nun die erste Testdate "2" eingegeben werden: Taste 2 der Tastatur drücken. Auf dem Bildschirm wird hinter dem Fragezeichen die Ziffer 2 ausgeschrieben. ? 2 Taste ENTER drücken Auf die Taste ENTER reagiert der Rechner mit . Ablegen des Wertes "2" im Speicherplatz X . Fortsetzen des Programmlaufes.
S = S + X	Der aktuelle Wert von S ist "0", dazu wird der aktuelle Wert von X addiert. Das Ergebnis wird nun neuer Wert von S. In S ist jetzt der Wert "2" gespeichert.
PRINT X,S	Auf dem Bildschirm wird ausgegeben: 2 2 Auf Position 16 der Bildschirmzeile die 1. Ziffer des S-Wert Auf Position 1 der Bildschirmzeile die erste Ziffer des aktuellen X-Wert. Der Abstand zwischen den Ausgabepositionen ist die Rechnerreaktion auf das Komma in der PRINT-Anweisung zwischen X und S.
GOTO 2Ø	Die nächste zu bearbeitende Anweisung steht unter der Zeilennummer 2Ø. Das Programm wird dort fortgesetzt.

INPUT X	Reaktion wie schon gezeigt. - Programmunterbrechnung - Bildschirmausgabe: "?" - Bei Eingabe der neuen Testdate wird diese hinter "?" ausgegeben: ? 3 - Nach ENTER: Der Wert "3" wird im Speicherplatz X als neuer Inhalt abgelegt. Der alte ist damit "überschrieben". - Fortsetzung des Programms.
S = S + X	Der aktuelle Wert von S ist "2". Der aktuelle Wert von X ist "3". Die Summe von "2+3" wird neuer Wert von S.
PRINT X,S	Bildschirmausgabe in einer neuen Zeile: 3 5 S-Wert auf Position 16 X-Wert auf Position 1 der neuen Zeile. Insgesamt ist auf dem Bildschirm folgendes Ausgabebild zu sehen: ? 2 2 2 ? 3 3 5
GOTO 2Ø	Fortsetzung des Programms mit der Anweisung der Zeile 2Ø
INPUT X	? -4 (-4 ist die eingegebene Testdate) Zuweisung von "-4" als aktuellen Inhalt des Speicherplatzes X. Fortsetzen des Programms.
S = S + X	Aktueller Inhalt des Speicherplatzes S wird die Summe "5 + (-4)" = 1.
PRINT X,S	Vollständiges Ausgabebild ? 2 2 2 ? 3 3 5 ? -4 -4 1
GOTO 2Ø	Fortsetzung des Programms in Zeile 2Ø.
INPUT X	Bildschirmausgabe: "?"

An dieser Stelle des Programms wartet der Rechner nun "endlos" lange, wenn keine weitere Eingabe erfolgt. Das Programm kann nur durch fehlerhafte Bedienung oder Ausschalten beendet werden.

2.5.2.2. Zählen

In Erweiterung der Aufgabenstellung zur Summenbildung soll zusätzlich gezählt werden, wie viele Eingabedaten in die Summe einbezogen werden.
Der folgende Algorithmus löst diese Aufgabenstellung:



Zusätzlich zum PAP für die Summenbildung sind die Anweisungen n:=Ø und n:=n+1 eingefügt worden. Im Grunde genommen ist das Zählen eine Summenbildung mit der konstanten Date von Eins. Immer wenn eine neue Eingabedate in die Summe gebracht wird, wird der aktuelle Wert der Variablen um "1" erhöht. Die Variable n "zählt" also die Anzahl der in die Summe gebrachten Eingabedaten.

Bild 25: PAP zählen

Das BASIC-Programm kann dafür aus folgenden Anweisungen bestehen:

```
1Ø S = Ø
2Ø N = Ø
3Ø INPUT X
4Ø S = S + X
5Ø N = N + 1
6Ø PRINT X, S, N
7Ø GOTO 3Ø
8Ø STOP
```

Eingefügt ist wieder die Ausgabe der aktuellen Werte auf dem Bildschirm. Das Komma zwischen den Ausgabevariablen X und S und N in der PRINT-Anweisung realisiert die Ausgabe der 1. Ziffer des Wertes von
X auf die Position 1 der Bildschirmzeile
von S auf die Position 16 der Bildschirmzeile,
von N auf die Position 31 der Bildschirmzeile.

Der Test mit den Daten 2, 3 und -4 ergibt die Bildschirmausschrift:

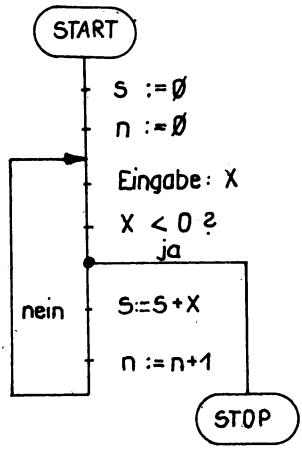
```
"?"2 2      2       1
"?"3 3      5       2
"?"-4-4     1       3
```

Der Rechner wartet wieder bei INPUT X auf eine weitere Eingabe.

2.5.2.3. Prüfen

Die Aufgabenstellung der Summenbildung und des Zählens soll nun so erweitert werden, daß nur positive Eingabedaten in die Summe gebracht und gezählt werden. Die erste negative Eingabedate soll den Stopp des Programms herbeiführen.

Der Algorithmus ist im Programmablaufplan und im BASIC-Programm dargestellt:




```
1Ø  S = Ø
2Ø  N = Ø
3Ø  INPUT X
4Ø  IF X<Ø THEN 9Ø
5Ø  S = S + X
6Ø  N = N + 1
7Ø  PRINT X, S, N
8Ø  GOTO 3Ø
9Ø  STOP
```

Bild 26: PAP Prüfen

Das neue an diesem Algorithmus ist die Untersuchung, ob die gerade eingegebene Date negativ ist, wenn ja, wird der Algorithmus und das Programm gestoppt, wenn nicht, so wird die Date in die Summe gebracht und gezählt.
Der Test mit den Daten 2, 3 und -4 ergibt folgendes Ergebnis, das auf dem Bildschirm ausgeschrieben wird:

```
"?"  2
   2        2         1
"?"  3
   3        5         2
"?" -4
 STOP LINE 9Ø
```

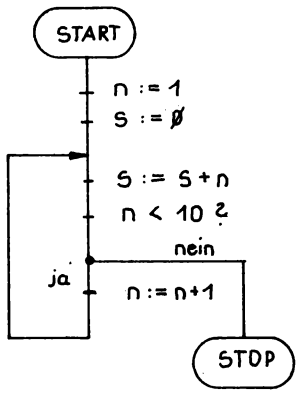
(Beim KC 85/2 steht: BREAK in 9Ø)
Der Rechner gibt auf dem Bildschirm aus, daß ein programmgesteuertes Stopp befohlen ist und in welcher Zeile des Programms die STOP-Anweisung steht.

Neu im BASIC-Programm ist die Anweisung IF X <Ø THEN 9Ø. Sie besagt, daß bei X <Ø mit Zeile 9Ø das Programm fortgesetzt wird. Tritt der Fall nicht ein, ist also X ≥ Ø, dann wird das Programm mit der Anweisung fortgesetzt, die nach der IF-THEN-Anweisung die nächst höhere Zeilennummer trägt.

Aufgaben zum Testen von Algorithmen

A5.

1. Aufgabe:

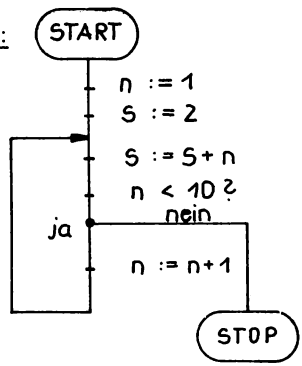



```
1Ø N = 1
2Ø S = Ø
3Ø S = S + N
4Ø PRINT N, S
5Ø IF N<1Ø THEN 7Ø
6Ø GOTO 9Ø
7Ø N = N + 1
8Ø GOTO 3Ø
9Ø STOP
```

Bild A5

A6.

2. Aufgabe:




```
1Ø N = 1
2Ø S = 2
3Ø S = S + N
4Ø PRINT N, S
5Ø IF N>=1Ø THEN 8Ø
6Ø N = N + 1
7Ø GOTO 3Ø
8Ø STOP
```

Bild A6

A7.

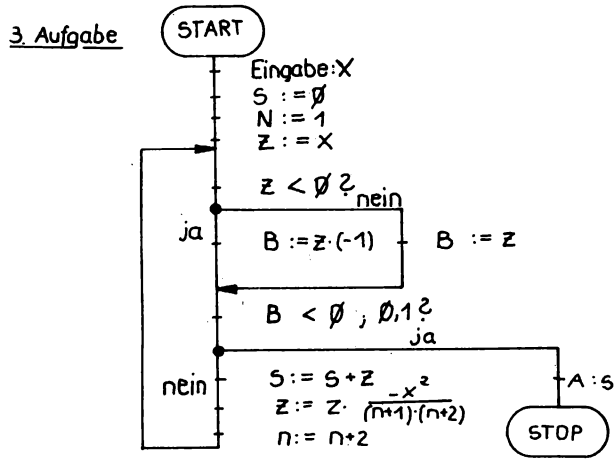


Bild A7

```
1Ø INPUT X
2Ø S = Ø
3Ø N = 1
4Ø Z = X
5Ø IF Z < Ø THEN B = Z * (-1) ELSE B = Z
6Ø IF B < Ø.Ø1 THEN 12Ø
7Ø S = S + Z
8Ø PRINT X, N, Z, S
9Ø Z = -X * X * Z/((N+1) * (N+2))
1ØØ N = N + 2
11Ø GOTO 5Ø
12Ø STOP
```

Das Programm soll für X=Ø.523 getestet werden.

2.5.2. Ein Algorithmus zur Mittelwertbildung

Die Gesamtheit der im vorangegangenen Abschnitt behandelten Algorithmenelemente - Summenbildung - Zählen - Prüfen - ist ein lösender Algorithmus für folgende Aufgabenstellung:

> Von Meßwerten, die dem Rechner einzeln über die Tastatur als Eingabedaten zugeführt werden, soll der arithmetische Mittelwert gebildet werden. Die Anzahl der Meßwerte liegt nicht vor. Alle Meßwerte sind positiv. Das Schlußzeichen der Eingabedaten ist eine negative Zahl.

Der im Bild 27 dargestellte Algorithmus löst diese Aufgabenstellung. Der Algorithmus entspricht genau dem aus Bild 26 bis auf die Anweisungen:

$\bar{X} = S/n$ und A: $\bar{X}$

Die Anweisung $\overline{X}$ = S/n bestimmt aus den Ergebnissen der Summenbildung (S) und des Zählens (n) den Mittelwert aller Eingabedaten durch

$$\overline{X} = \frac{\text{Summe aller Meßwerte}}{\text{Anzahl aller Meßwerte}}$$

Die Anweisung A: $\overline{X}$ sichert die Ausgabe des Mittelwertes als Ergebnis.

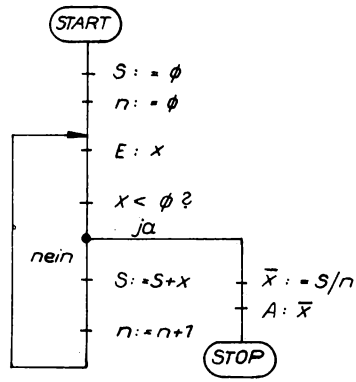


Bild 27: PAP Mittelwertbildung

Erstmals liegt damit der lösende Algorithmus einer in sich geschlossenen, praktischen Aufgabenstellung vor. Bei der Implementierung dieses Algorithmus in einem Rechner möchte man diesem Programm gern einen Namen geben, so daß beim Aufrufen des Programms der Name auf den Inhalt hinweist.
Dazu gibt es in BASIC die REM-Anweisung.
Hinter dem Symbol REM kann der Programmierer beliebige BASIC-Zeichen aufschreiben. Diese Zeichen haben für den Programmlauf keinerlei Bedeutung. Sie kommentieren nur den Programmaufbau.

Beispiel:

```
  1Ø REM  MEIN ␣ ERSTES ␣ BASIC-PROGRAMM
  2Ø REM  IST ␣ DIE ␣ MITTELWERTBILDUNG
  3Ø   .
       .  Anweisungen
       .
 23Ø REM  BEGINN ␣ DER ␣ DATENAUSGABE
       .
       .  Anweisungen
       .
 57Ø REM  X, Y, Z ␣ UND ␣ B ␣ SIND ␣ EINGABEGROESSEN
```

Das BASIC-Programm zur Mittelwertbildung könnte nun wie folgt aufgeschrieben werden:

```
10  REM  MITTELWERTBILDUNG
20  REM  X = MESSWERT; S = SUMME; N% = ZAEHLER
30  REM  M = MITTELWERT
40  S = 0
50  N% = 0%
60  PRINT "MESSWERTEINGABE X"
70  INPUT X
80  IF X < 0 THEN 140
90  REM SUMMENBILDUNG - ZAEHLEN
100 S = S + X
110 N% = N% + 1%
120 GOTO 60
130 REM MITTELWERTBERECHNUNG
140 M = S/N%
150 PRINT "MITTELWERT ␣ M = ␣"; M
160 STOP
```

In den ersten drei REM-Anweisungen wird dem Programmnutzer folgendes mitgeteilt:

1. Das Programm heißt MITTELWERTBILDUNG.
2. Es werden die Variablen X, S, N% und M benutzt.

Die REM-Anweisungen der Zeilen 90 und 130 geben Hinweise dazu, was in dem nachfolgenden Anweisungskomplex im wesentlichen inhaltlich bearbeitet wird.

Der Programmlauf mit den Testdaten

2 | 2 | 3 | 1 | 4 | 3 | 3 | 2 | -1

ergibt dann folgendes Ergebnis auf dem Bildschirm:

```
MESSWERTEINGABE: X
?  2
MESSWERTEINGABE: X
?  2
MESSWERTEINGABE: X
?  3
          usw. bis
MESSWERTEINGABE: X
?  -1
MITTELWERT M =  2,5
```

Damit wird auch der Unterschied zwischen REM und PRINT nochmals deutlich. REM hat mit der Bildschirmausgabe im Programmlauf nichts zu tun. Die REM-Anweisungen kommentieren tatsächlich nur den aufgeschriebenen Programmaufbau.

Lösungen zu den Aufgaben

A1.: - BASIC-Variable sind: B3, A%, X1, Z, O, G7, B3%, M¤, M9¤

- Fehler bei den übrigen Variablen:

XZ	- Zweites Zeichen ist ein Buchstabe
Y12	- besteht aus 3 Zeichen
∝3	- ist unbekanntes Zeichen
$\overline{X}$	- Querstrich nicht zulässig
Max	- 3 Zeichen und kleine Buchstaben
Otto	- 4 Zeichen und kleine Buchstaben
4Z	- 1. Zeichen ist Ziffer
BETA	- 4 Zeichen

A2.: - BASIC-Zahlen sind: 4.37; 4%; 4.31 E-3; 137ØØØ; 37.4837 die Ziffern 5 und 7 entfallen

- Fehler bei den übrigen Zahlen:

3,7	- Komma statt Punkt
$3.1Ø^4$	- Multiplikationszeichen, Zehnerpotenz
78,15	- Komma statt Punkt
-3,7	- Komma statt Punkt
14.2^3	- Multiplikationszeichen, Zweierpotenz
137ØØØ%	- Integerzahl ist größer als 32767

A3.: - BASIC-Ausdrücke sind: Z4 * 3% + 4 * K%

A * PI * 3/4

M% + 1%

- Fehler bei den übrigen Ausdrücken:

R2 * 4Z3/3 * X	- Zwischen 4 und Z3 fehlt ein Operator
XY * 4 * π * 3	- Zwischen X und Y fehlt ein Operator und das Zeichen π gibt es nicht
$\frac{3}{4}$ * D↑3	- falsches Operationszeichen, richtig wäre 3/4 * D↑3

A4.: - Die BASIC-Versionen können sein:

VØ+A * T; (V1-VØ)/(T1-TØ); A * T↑2/2+VØ * T+SØ;

2 * PI * R/T; SQR(VØ↑2+(G * F)↑2);

G * M1 * M2 * (1/R1-1/R2)

A5.: Auf dem Bildschirm wird ausgegeben:

für N	für S
1	1
2	3
3	6
4	1Ø
5	15
6	21
7	28
8	36
9	45
1Ø	55

A6.: Auf dem Bildschirm wird ausgegeben:

für N	für S
1	3
2	5
3	8
4	12
5	17
6	23
7	3Ø
8	38
9	47
1Ø	57

A7.: Auf dem Bildschirm wird ausgegeben:

für X	für N	für Z	für S
Ø.523	1	Ø.523	Ø.523
Ø.523	3	-Ø.0238	Ø.4992
Ø.523	5	Ø.0003	

Institut für Rationalisierung
der Elektrotechnik/Elektronik
Zentralstelle
für Aus- und Weiterbildung
des Industriebereiches
Elektrotechnik/Elektronik

Lehrmaterial für die
Weiterbildung

Informationsverarbeitung mit Kleincomputern

3

Algorithmierung Teil 2

Kreutzer

Elektrotechnik
Elektronik
Automation
DDR

Steffen Kreutzer

Informationsverarbeitung mit Kleincomputern

3

Algorithmierung, Teil 2

Herausgeber
Institut für Rationalisierung der Elektrotechnik/Elektronik
Institutsteil Dresden
Zentralstelle für Aus- und Weiterbildung
des Industriebereiches Elektrotechnik/Elektronik
Karl-Marx-Straße, Dresden
8080

Autor: FSD Dipl.-Ing. Steffen Kreutzer
Ingenieurschule für Wissenschaftlichen Gerätebau "Carl Zeiss" Jena

Gutachter: Dipl.-Ing. Claus Paul
Institut für Rationalisierung der Elektrotechnik/Elektronik

Bearbeiter: Dipl.-Gwl. Heinz Rüdger
Institut für Rationalisierung der Elektrotechnik/Elektronik
Zentralstelle für Aus- und Weiterbildung

1. Auflage
IR Dresden ZSB
Druckgenehmigungs-Nr.: Ag 682/043/86
Druck und Herstellung: NOWA DOBA Bautzen III-4-9
Redaktionsschluß: 31. 7. 1986
Bestell-Nr.: T.2.04.0003
Vorzugsschutzgebühr: 2,- M

Inhaltsverzeichnis Seite

3.	Synthese von Algorithmen	4
3.1.	Ein einführendes Beispiel	4
3.2.	Verallgemeinerung der Lösungsfindung	6
3.3.	Übungen zu einfachverzweigten Algorithmen	7
3.4.	Übungen zu mehrfachverzweigten Algorithmen	17
3.5.	Algorithmen zur Arbeit mit Feldern	26
3.5.1.	Einführende Bemerkungen	26
3.5.2.	Realisierung von Vektoren und Matrizen im Speicher des Rechnerarbeitsplatzes	29
3.5.3.	Laufanweisung	33
4.	Strukturierte Algorithmen	40
4.1.	Grundgedanke	40
4.2.	Logische Grundstrukturen	41
4.2.1.	Folge	42
4.2.2.	Einfache Verzweigung	43
4.2.3.	Bedingte Wiederholung	44
	Lösungen zu den Aufgaben	50

3. Synthese von Algorithmen

Soll eine gegebene Aufgabenstellung mit einem Digitalrechner bearbeitet werden, so ist sofort die Frage nach dem lösenden Algorithmus gestellt. Ohne ihn kann kein Programm aufgeschrieben werden, nach dem der Rechner die Lösung schrittweise erarbeitet.

Wie kommt man von der Aufgabenstellung zu einem lösenden Algorithmus?

3.1. Ein einführendes Beispiel

Das bekannte Beispiel der Mittelwertbildung soll nochmals untersucht werden. Im Abschnitt 2.5. des Heftes 2 dieser Lehrmaterialreihe ergab sich die Lösung durch die Gesamtheit der drei Algorithmenelemente Summenbildung - Zählen - Prüfen. Jetzt soll von der Aufgabenstellung ausgegangen werden. Sie lautet:

Von Meßwerten, die dem Rechner einzeln über die Tastatur als Eingabedaten zugeführt werden, ist das arithmetische Mittel zu bilden. Die Anzahl der Meßwerte liegt nicht vor. Alle Meßwerte sind positiv.
Das Schlußzeichen der Eingabedaten ist eine negative Zahl.

- Analyse der Aufgabenstellung

Die Analyse der Aufgabenstellung erfolgt günstig durch die Beantwortung folgender Fragen:

1. Frage: Gibt es ein zentrales Problem in der Gesamtaufgabenstellung?

Antwort: Ja. Die Bildung des arithmetischen Mittels.

2. Frage: Existiert ein bekannter Algorithmus (Vorschrift) zur Lösung des zentralen Problems?

Antwort: Ja. Das arithmetische Mittel ist die Summe aller Meßwerte geteilt durch die Anzahl aller Meßwerte

$$x = (x_1+x_2+x_3+x_4+ \ldots +x_n)/n$$

$$x = \frac{1}{n} \sum_{i=1}^{n} x_i$$

3. Frage: Welche Forderungen stellt der zentrale Algorithmus?

Antwort: 1. Die Bildung der Summe aller Meßwerte.
2. Er benötigt die Anzahl aller Meßwerte.

Mit der Kenntnis des Algorithmenelementes Summenbildung kann der zentrale Algorithmus aufgeschrieben werden (Bild 28).

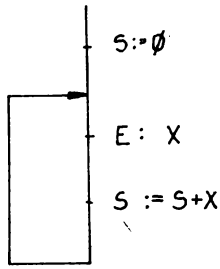


Bild 28:
PAP - zentraler Algorithmus, Summenbildung

Der zentrale Algorithmus kann zum lösenden Algorithmus ergänzt werden, indem die speziellen Bedingungen der Eingabedaten beachtet werden.

- Analyse der Eingabedaten

1. Frage: Ist die Anzahl der Daten begrenzt und damit bekannt?

Antwort: Nein. Es wird deshalb notwendig, die Anzahl durch "Zählen" der Eingabedaten zu bestimmen.

2. Frage: Unterliegen die Eingabedaten bestimmten Einschränkungen?

Antwort: Ja. Meßwerte sind positive Werte. Ein negativer Wert entspricht dem Schlußzeichen. Die Eingabedaten sind also zu prüfen, ob sie negativ sind. Bei negativer Eingabedate ist die Summenbildung abgeschlossen.

- Grobkonzept des lösenden Algorithmus

Das zusammenfassende Durchdenken aller Forderungen führt zu

1. Daten einzeln einlesen;
2. Eingabedate prüfen, ob sie negativ ist;

3. Positive Werte aufsummieren und zählen;
4. Zu 1. zurück.

Diese Form stellt ein grobes Konzept der Lösung dar. Das Aufschreiben des Grobkonzeptes in einem Programmablaufplan (PAP) ist die erste Stufe zum lösenden Algorithmus. Davon ausgehend wird der Algorithmus schrittweise ergänzt, bis er der Aufgabenstellung voll entspricht.

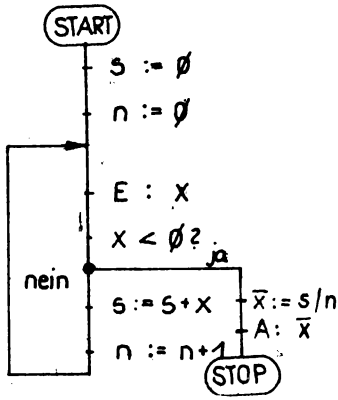


Bild 29:
PAP Mittelwertbildung

Die weiteren Ergänzungen des Grobkonzepts laufen darauf hinaus, daß zu überlegen ist, was zu geschehen hat, wenn der Schlußwert der Daten eingegeben wurde. Nun, dann sind alle positiven Meßwerte schon in die Summe gebracht worden und in n wurde gezählt, wieviele positive Werte in S summiert worten sind. Damit ist im Falle der Erkennung des Schlußwertes der Mittelwert durch

$$\bar{x} = S/n$$ zu berechnen und auszugeben.

Somit ergibt sich der im Bild 29 dargestellte PAP zur Aufgabenstellung Mittelwertbildung als Lösung.

3.2. Verallgemeinerung der Lösungsfindung

Gefunden wurde die Lösung im Abschnitt 3.1. durch zwei Ansätze. Der erste Ansatz besteht darin, daß für das zentrale Problem ein Algorithmus vorliegt. Oftmals findet man ihn in der mathematischen Formulierung des zentralen Problems. Damit

ist dieser Ansatz nicht so unreal, wie er vielleicht erscheinen mag. Für viele praktische Probleme liegt die mathematische Formulierung vor.
Der zweite Ansatz besteht in der notwendigen Kenntnis einiger Algorithmenelemente. Dieser Sachverhalt ist typisch für die rechentechnische Bearbeitung von Aufgabenstellungen. Die Lösungsfindung ist stark vom Können des Programmierers abhängig, von der Kenntnis vieler Algorithmen.

Die nachfolgenden Übungen dienen dem Zweck, möglichst viele Grundalgorithmen kennenzulernen, die dann in komplexere Lösungen einfließen können.

Allgemein kann folgender Verfahrensweg zum Auffinden eines lösenden Algorithmus angegeben werden:

1. Analyse der Aufgabenstellung hinsichtlich ihrer zentralen Probleme und deren mathematischen Darstellung
2. Formulieren der zentralen Algorithmen
3. Analyse der Eingabedaten
4. Darstellung des Grobkonzeptes im PAP
5. Ergänzung des PAP zum lösenden Algorithmus
6. Aufschreiben des Programms.

3.3. Übungen zu einfachverzweigten Algorithmen

Beispiel 1: Einem Rechner werden als Eingabedaten Stückzahlen und Stückpreise eingegeben. Die Reihenfolge ist die, daß zuerst immer die Stückzahl einer Warenposition und danach der zur Warenposition gehörende Stückpreis eingegeben wird. Das Ende der Daten ist eine negative Stückzahl.

Es ist ein Algorithmus zu entwerfen, der nach jeder Warenpositionseingabe diese ausgibt, den Preis der Position berechnet und diesen sowie zum Schluß den Gesamtpreis der Lieferung ausgibt.

- Analyse der Aufgabenstellung

Ein bekannter Algorithmus liegt nicht vor. Man kann sich jedoch schnell die mathematische Formulierung des Problems überlegen.
Der Preis für jede Warenposition ist:

P = Stückzahl mal Stückpreis
P = n * p.

Der Gesamtpreis der Warenlieferung ist dann die Summe aller Warenpositionspreise:

$$G = P_1 + P_2 + P_3 + \ldots + P_i$$

mit P_1 = Preis der 1. Warenposition
P_2 = Preis der 2. Warenposition usw.

- Formulierung des zentralen Algorithmus

Der zentrale Algorithmus ist im wesentlichen durch die Summenbildung bestimmt.

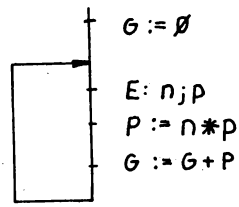


Bild 30:
PAP - zentraler Algorithmus
Rechnungslegung

Mit der Eingabe von n und p, deren Multiplikation zu P und der Summenbildung aller P ist die wesentliche Struktur des Algorithmus bestimmt.

- Analyse der Eingabedaten

Die Stückzahlen und Stückpreise, die als solche verarbeitet werden sollen, sind positive Zahlen. Das Schlußzeichen der Daten bildet eine negative Stückzahl. Somit ist jede Stückzahl daraufhin zu prüfen, ob sie negativ ist. Erst wenn klar ist, daß die aktuell eingegebene Stückzahl positiv ist, darf

ein Stückpreis zur Eingabe kommen.

- Darstellung des Grobkonzepts

1. n einlesen
2. n prüfen, ob negativ
3. Wenn n positiv, p einlesen
4. P bilden
5. G bilden
6. Zu 1. zurück

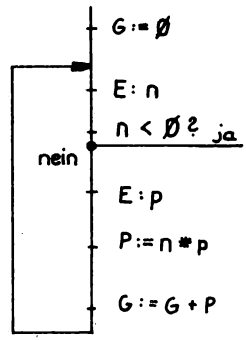


Bild 31:
PAP - Grobkonzept Rechnungslegung

- Ergänzung des PAPs zum lösenden Algorithmus

Die notwendigen Ergänzungen betreffen in diesem Beispiel die Gestaltung der Ausgabe. Laut Aufgabenstellung ist die Eingabe von n und p zu quittieren, die Ausgabe von P vorzunehmen und das Endergebnis von G anzuzeigen.
Die Ausgabe von **n,p und P** muß, da alle Positionen ausgegeben werden sollen, innerhalb der Schleife erfolgen. Die Ausgabe von G darf erst erfolgen, wenn die Summenbildung beendet ist, also muß diese Ausgabeanweisung im ja-Zweig der Programmverzweigung stehen. Die Lösung zeigt Bild 32.

- Das BASIC-Programm

Bevor das zugehörige BASIC-Programm aufgeschrieben werden kann, müssen noch zwei Erweiterungen der im Heft 2, Abschnitt 2.4.1., dargestellten Möglichkeiten des BASIC-Interpreters zur Gestaltung der Ausgabe genannt werden.
Im Prinzip gilt, daß jede neue PRINT-Anweisung in einer neuen

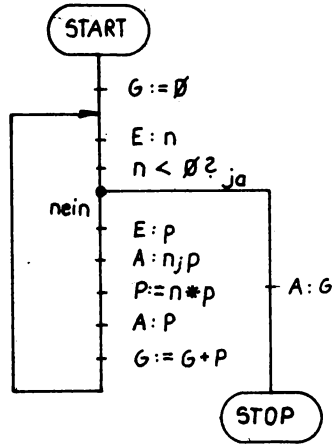


Bild 32:
PAP zum Algorithmus Rechnungslegung

Bildschirmzeile realisiert wird. Dieses Prinzip wird aufgehoben, wenn die letzte PRINT-Anweisung vor der aktuellen mit einem Semikolon abgeschlossen wurde.

Beispiele:
```
11Ø PRINT "BASIC"
12Ø PRINT "INTERPRETER"
 .
 .
 .
```
Dieses Beispiel gibt die Bezeichnung in zwei Zeilen aus, nämlich

1. Zeile BASIC
2. Zeile INTERPRETER

```
11Ø PRINT "BASIC";
12Ø PRINT "-INTERPRETER"
 .
 .
 .
```
Dieses Beispiel gibt die Bezeichnung in einer Zeile aus, nämlich

1. Zeile BASIC - INTERPRETER

Der BASIC-Interpreter gestattet auch die Ausgabe ab einer gewünschten Position innerhalb einer Zeile. Die Zeile des Bildschirms ist in die Positionen 0 ... 64 eingeteilt.
Mit der Anweisung

```
130 PRINT TAB (40); "ABC"
```

wird das A auf die Position 41 der Zeile
das B auf die Position 42 der Zeile
das C auf die Position 43 der Zeile geschrieben.

Das BASIC-Programm für den Gesamtpreis kann nun folgendermaßen aufgeschrieben werden:

```
 10 REM RECHNUNGSLEGUNG
 20 REM EINGABE-VARIABLEN :   N, P
 30 REM AUSGABE-VARIABLEN :   P1, G
 40 G=0
 50 PRINT "EINGABE - N "
 60 INPUT N
 70 IF N<0 THEN 150
 80 PRINT "EINGABE - P "
 90 INPUT P
 100 PRINT N;" STK";P;" M";
 110 P1=N*P
 120 PRINT TAB(30);P1;" M"
 130 G=G+P1
 140 GOTO 50
 150 PRINT "GESAMTPREIS ";TAB(30);G;" M"
 160 STOP
```

Bei der Abarbeitung des Programms entsteht entsprechend den Eingabedaten folgendes Ergebnisprotokoll:

. auf dem Bildschirm ein Eingabe-Ausgabe-Dialog:

```
RUN
EINGABE - N
 1000
EINGABE - P
 .02
 1000 STK .02 M              20 M
EINGABE - N
 130
EINGABE - P
 .02
 130 STK 1.1 M               143 M
EINGABE - N
 200
```

. auf dem Drucker (nach Programmveränderung) eine Ergebnisliste:

```
BASIC

ERGEBNISLISTE

 1000 STK .02 M    20 M
 130 STK 1.1 M     143 M
 200 STK .7 M      140 M
GESAMTPREIS        303 M
```

```
EINGABE - P
 1.1
 200 STK .7 M                140 M
EINGABE - N
-1
GESAMTPREIS                 303 M
STOP LINE 160
```

Beispiel 2: Als Übungsbeispiel soll vom Inhalt eines Speicherplatzes mit dem Namen X der Sinus berechnet werden. Die Berechnung soll nach der unendlichen Reihe

$$y = \sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + - \ldots$$

erfolgen. Das Ergebnis ist im Speicherplatz Y abzulegen. Die Berechnung der Reihe soll abgebrochen werden, wenn ein Summenglied betragsmäßig kleiner als eine vorgegebene Schranke wird.

- Analyse der Aufgabenstellung

In der Aufgabenstellung ist die mathematische Formulierung des Problems schon gegeben. Man könnte nun ausgehend von der Eingabe von x die einzelnen Summenglieder

$$\frac{x}{1!} \; ; \; \frac{x^3}{3!} \; ; \; \frac{x^5}{5!} \; ; \text{ usw.}$$

berechnen und alle Summenglieder vorzeichenbehaftet summieren. Dieses Vorgehen kostet jedoch sehr viel Rechenzeit, da zum Beispiel x^5 berechnet wird, als wäre nicht gerade zuvor x^3 berechnet worden. Somit soll die Aufgabe so gelöst werden, daß der neue Zuwachs aus dem jeweils alten berechnet wird.
Es soll also

$\frac{x^3}{3!}$ aus $\frac{x}{1!}$ berechnet werden und

$\frac{x^5}{5!}$ aus $-\frac{x^3}{3!}$ berechnet werden usw.

Um diese Forderung realisieren zu können, muß man davon ausgehen, daß

1! (Eins-Fakultät) definitionsgemäß gleich 1 ist
2! (Zwei-Fakultät) definitionsgemäß gleich 1 · 2 ist
3! (Drei-Fakultät) definitionsgemäß gleich 1 · 2 · 3 ist also
n! (n-Fakultät) definitionsgemäß gleich 1 · 2 · 3 · ... · n ist.

Somit kann man
3! aus 1! durch Multiplikation mit 2 . 3 erhalten
5! aus 3! durch Multiplikation mit 4 . 5 erhalten
7! aus 5! durch Multiplikation mit 6 . 7 erhalten.

Allgemein erhält man
(n+2)! aus n! durch Multiplikation mit (n+1).(n+2).

Die Potenzen von x errechnet man ähnlich. Man erhält

$-x^3$ aus x durch Multiplikation mit $-x^2$
$+x^5$ aus $-x^3$ durch Multiplikation mit $-x^2$
$-x^7$ aus $+x^5$ durch Multiplikation mit $-x^2$ und
allgemein
$\pm x^{n+2}$ aus $\mp x^n$ durch Multiplikation mit $-x^2$

Um ein neues Summenglied aus dem jeweils alten zu berechnen, muß also

$$Z_{neu} = Z_{alt} \cdot \frac{-x^2}{(n+1).(n+2)}$$ gerechnet werden.

Sorgt man dafür, daß der erste alte Zuwachs (Summenglied) gleich dem Eingabewert x ist und n mit dem Wert 1 beginnt, so ist der erste neue Zuwachs

$$Z_{neu} = x \cdot \frac{-x^2}{(1+1).(1+2)} = -\frac{x^3}{2 \cdot 3} = -\frac{x^3}{3!}$$

- Formulierung des zentralen Algorithmus

Der zentrale Algorithmus besteht darin, daß alle Summenglieder (Zuwachs) aufsummiert werden und jedes neue Summenglied (Z_{neu}) sich aus dem zuvor berechneten (Z_{alt}) ergibt

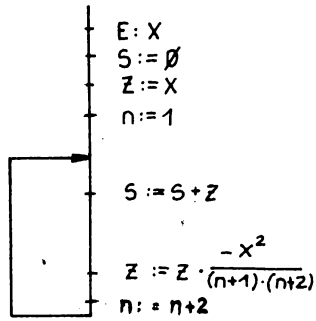


Bild 33:
PAP - zentraler Algorithmus Sinusbildung

- Analyse der Eingabedaten

Als Eingabedaten sind in diesem Beispiel der Wert einzugeben, von dem der Sinus zu bilden ist und der Prüfwert ε (Epsilon). Er wird benötigt, um die Summenbildung abzubrechen, wenn ein neu berechneter Summenzuwachs betragsmäßig kleiner als ε wird. Wählt man ε als Zehnerpotenzen z. B. 0,01; 0,001 oder 0,0001, so wird der Sinuswert als richtig angesehen, wenn keine Veränderungen an der 2., 3. oder 4. Ziffernstelle mehr erfolgt.

- Formulierung des Grobkonzeptes

1. Eingabe von x und ε
2. x-Wert zum ersten Z erklären
3. Prüfen, ob Z-Betrag $< \varepsilon$
4. Summe bilden, wenn $|Z| >$ oder $= \varepsilon$
5. Z_{neu} berechnen
6. Zurück zu 3.

Die notwendigen Ergänzungen zum lösenden Algorithmus sind:

1. Die Erhöhung von n um 2 in jedem Durchlauf, um die Berechnung des neuen Zuwachs vorzubereiten und

2. Das Ablegen der Summe in den Speicherplatz Y, wenn die Prüfung des Z-Betrages $< \varepsilon$ ergibt.

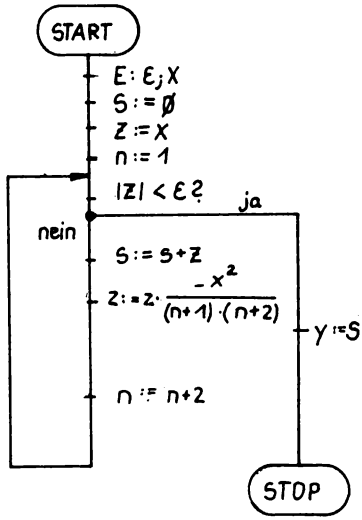


Bild 34:
PAP
Sinusbildung

- BASIC-Programm

```
10 REM SINUSBILDUNG
20 REM E = PRUEFGROESSE EPSILON
30 REM X = EINGANGSVARIABLE
40 REM Y = AUSGANGSVARIABLE
50 PRINT "EINGABE X E"
60 INPUT X,E
70 S=0
80 Z=X
90 N=1
100 A=ABS(Z)
110 IF A<E THEN 160
120 S=S+Z
130 Z=-X*X*Z/((N+1)*(N+2))
140 N=N+2
150 GOTO 100
160 Y=S
170 STOP
```

Aufgaben zu einfachverzweigten Algorithmen

Für die folgenden Aufgaben ist als Lösung immer der Algorithmus in seiner grafischen Darstellungsform als PAP und das BASIC-Programm gefordert.

A 8. Einem Rechner werden laufend über die Tastatur Meßwerte eingegeben. Schlußzeichen ist eine -1. Die Meßwerte sind positiv. In einem Programm soll jedoch nur immer jeder 2. Meßwert zur Bildung des arithmetischen Mittels herangezogen werden.

A 9. Gebildet werden soll n-Fakultät. Die ganze Zahl n wird über die Tastatur eingegeben. Sie ist immer größer oder gleich 2.n!. n-Fakultät soll über Bildschirm ausgegeben werden.
Hinweis: n! = n.(n-1).(n-2). ... .(2).(1)

A 10. Es ist das Skalarprodukt zweier Vektoren X und Y zu bilden. Skalarprodukt = $x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3 + \ldots$
Die Daten x_i, y_i werden dem Rechner über Tastatur eingegeben. Stets werden Wertepaare eingegeben. Nur zum Schluß folgt einzig die Eingabe des Wertes -1 als Schlußzeichen.

A 11. Wie Aufgabe A 10 mit folgender Dateneingabe. Als erste Eingabe erfolgt die Angabe der Anzahl der Wertepaare x_i, y_i. Dann folgen die Wertepaare x_i und y_i ohne Schlußzeichen.

A 12. Wie Aufgabe A 10 mit folgender Dateneingabe. Es ist bekannt, daß nur 10 Wertepaare vorliegen. Die 10 Wertepaare werden über Tastatur eingegeben.

A 13. Mit dem Inhalt des Speicherplatzes X soll jeweils die unten angegebene Funktion berechnet werden. Das Ergebnis ist im Speicherplatz Y abzuspeichern. Die Rechnung soll abgebrochen werden, wenn ein Summenglied betragsmäßig kleiner als 10^{-3} wird.

A 13.1. $y = \cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + - \ldots$

A 13.2. $y = e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$

A 13.3. $y = \arctan x = \frac{x}{1} - \frac{x^3}{3} + \frac{x^5}{5} - + \ldots$ für $|X| < 1$

3.4. Übungen zu mehrfachverzweigten Algorithmen

Mehrfachverzweigte Algorithmen entstehen in der Praxis häufig dann, wenn anfallendes Datenmaterial nach unterschiedlichen Gesichtspunkten untersucht werden soll.

Beispiel 1: Der in Bild 35 gezeigte Algorithmus sortiert die Eingabedaten so, daß alle positiven Eingabedaten eine Ausgabefolge bilden und alle negativen Eingabedaten ebenfalls eine eigene Ausgabefolge bilden. Beim Eingeben einer Ø wird der Algorithmus gestoppt.

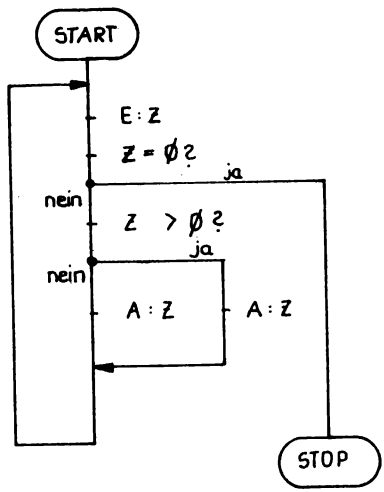


Bild 35:
PAP
Sortieren von Zahlen nach dem Vorzeichen

Im BASIC-Programm ist darauf zu achten, daß die Ausgabe möglichst übersichtlich gestaltet wird. Die sortierten Folgen sollen unter den Überschriften "POSITIV" und "NEGATIV" untereinander angezeigt werden.
Folgendes BASIC-Programm erfüllt diese Forderung:

```
10 REM SORTIEREN VON ZAHLEN
20 REM NACH IHREM VORZEICHEN
30 REM
40 PRINT TAB(40);"POSITIV";TAB(50);"NEGATIV"
50 INPUT Z
60 IF Z=0 THEN 90
70 IF Z>0 THEN PRINT TAB(43);Z ELSE PRINT TAB(53);Z
80 GOTO 50
90 STOP
```

Beispiel 2: Für die Planung von Investitionen interessieren in einer Hauptabteilung die Verteilung der anfallenden Arbeitsgänge. Zu diesem Zweck wird einem Rechner die Arbeitsgangfolge für jedes zu fertigende Einzelteil kodiert eingegeben. Zur Übung sollen nur die Arbeitsgänge

Sägen, Hobeln, Fräsen, Drehen, Bohren und Schleifen

verwendet werden. Die kodierte Eingabe erfolgt durch

SG für Sägen
HO " Hobeln
FR " Fräsen
DR " Drehen
BO " Bohren
SL " Schleifen.

Gesucht ist ein Algorithmus, der die Anzahl jedes Arbeitsganges und seinen prozentualen Anteil an der untersuchten Gesamtzahl aller Arbeitsgänge bestimmt. Die Eingabe von "EN" soll zum STOP führen.

. Analyse der Aufgabenstellung

Das zentrale Problem besteht darin, daß jede Eingabe daraufhin zu untersuchen ist, um welchen Arbeitsgang es sich handelt. Dafür existiert kein bekannter Algorithmus, aber es wird sofort klar, daß mehrmaliges "Nachfragen" die Aufgabe löst.

Der zentrale Algorithmus

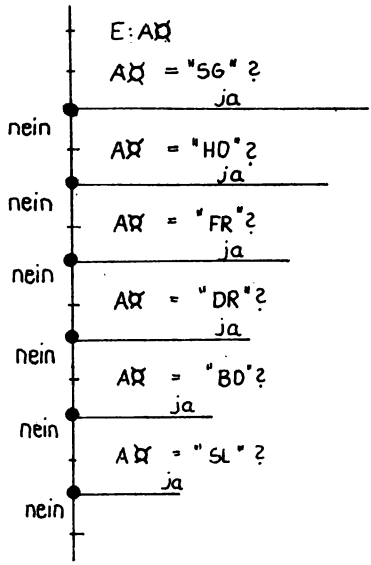


Bild 36:
Zentraler Algorithmus zur Arbeitsgangerkennung

Für jeden Arbeitsgang kann nun im jeweiligen Ja-Zweig des Algorithmus nach Bild 36 sein Auftreten gezählt werden. Somit sind alle Voraussetzungen geschaffen, um auch den prozentualen Anteil jedes Arbeitsganges zu bestimmen, denn es gilt zum Beispiel für den Arbeitsgang Fräsen:

$$\text{prozentualer Anteil FR} = \frac{\text{Anzahl FR}}{\Sigma \text{ aller Arbeitsgänge}} * 100.$$

. Analyse der Eingabedaten

Die Behandlung der Eingabedaten entsprechend dem Algorithmus in Bild 36 ist daraufhin zu ergänzen, daß die Eingabe von "EN" erkannt wird und zum STOP des Algorithmus führt. Fehlerhafte Eingaben werden nicht "behandelt". Findet der Algorithmus keine der definierten Eingabedaten, so werden immer nur die Nein-Zweige durchlaufen. Am Ende der Nein-Zweige ist deshalb eine Ausgabe "Fehleingabe" angebracht.

• Darstellung des Grobkonzepts im PAP

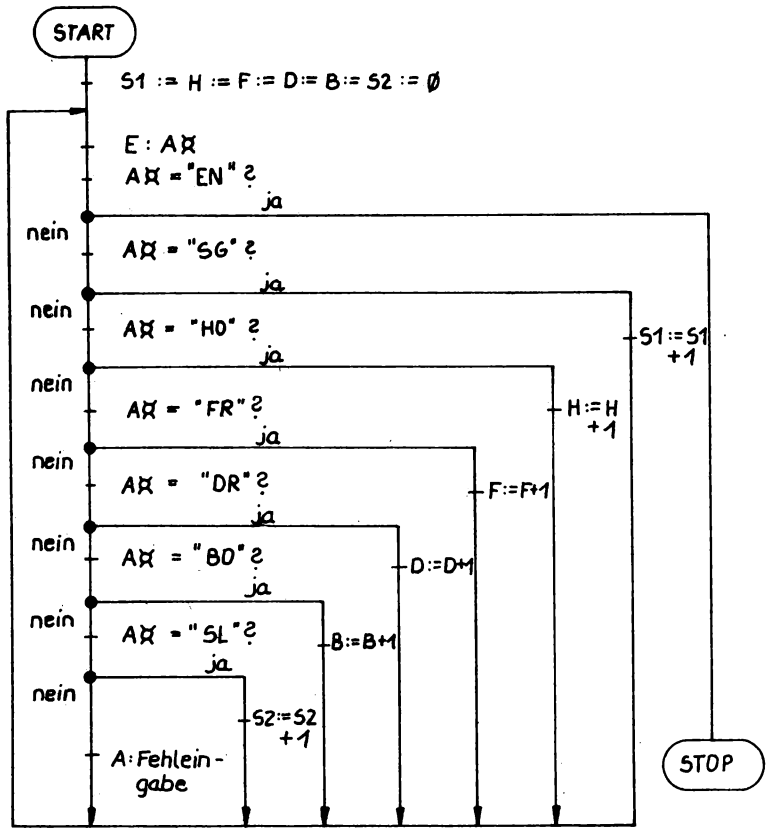


Bild 37: PAP-Grobkonzept Arbeitsgangverteilung

• Ergänzung des PAPs zum lösenden Algorithmus

Noch nicht gelöst ist die Aufgabenstellung hinsichtlich der Berechnung des prozentualen Anteils und der Ausgabe der Ergebnisse. In bezug auf die Ausgabe bestehen zwei prinzipiell unterschiedliche Möglichkeiten. Da die Aufgabenstellung dazu keine Aussage macht, sollen beide erläutert werden.

1. Möglichkeit: Die Ausgabe erfolgt nach jeder bearbeiteten

Eingabe. Das bedeutet, die Berechnung des prozentualen Anteils muß innerhalb der Schleife erfolgen, ebenso die Ausgabe (Bild 38).

2. Möglichkeit: Die Ausgabe erfolgt nur bei Endezeicheneingabe "EN". Dann wird die Berechnung des prozentualen Anteils nur einmal im STOP-Zweig realisiert. Ebenso die Ausgabe. Im lösenden Algorithmus soll diese Variante zum Einsatz kommen (Bild 39).

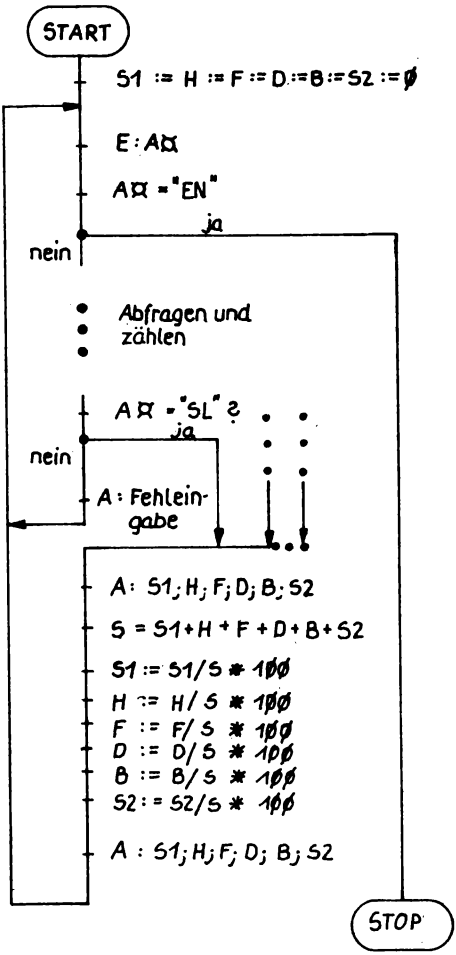


Bild 38:
PAP
Ausgabe in der Schleife

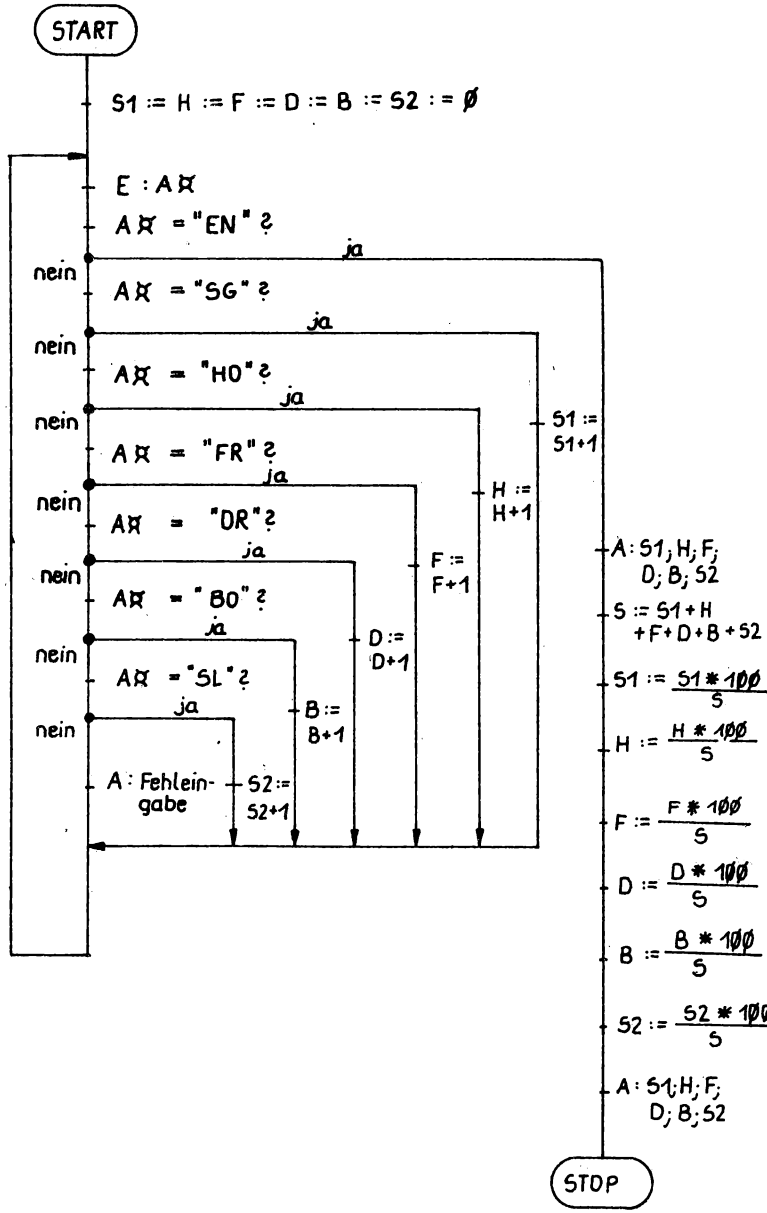


Bild 39: PAP - Lösender Algorithmus Arbeitsgangverteilung

- Das BASIC-Programm

Im BASIC-Programm wird der Gestaltung des Mensch-Rechner-Dialoges besondere Beachtung geschenkt. Aus der Sicht des Programmnutzers stellt sich das Programm in zwei Teilen dar.

1. Teil: Eingabedialog (mit Zählen, aber das ist für den Nutzer schon sekundär)
2. Teil: Ausgabebildgestaltung.

Im PAP nach Bild 39 ist schon auf die Möglichkeit der Fehleingabe eingegangen. Die Aussage des Rechners soll im BASIC-Programm noch verfeinert werden, indem in diesem Fall folgende PRINT-Anweisung eingegeben wird:

```
Znr PRINT "FEHLEINGABE ␣ ! ␣ SIE ␣ HABEN: "; A¤
Znr PRINT "EINGEGEBEN"
Znr PRINT "GEBEN ␣ SIE ␣ ERNEUT ␣ EIN"
```

Da der Eingabedialog auf dem Bildschirm mit verbleibt (protokolliert wird) soll im Falle der Eingabe von "EN" (Schlußzeichen) der Bildschirm gelöscht werden. Das erfolgt durch die Anweisung

```
Znr PRINT CHR ¤ (12)
```

Der Bildschirm wird mit Leerzeichen "vollgeschrieben", also gelöscht. Danach steht nun der gesamte Bildschirm zur Gestaltung der Ausgabe zur Verfügung. Folgendes Ausgabebild wird durch das nachfolgende Programm realisiert.

ARBEITS-GANG	ABSOLUTE ANZAHL	PROZENTUALER ANTEIL
SAEGEN	x	x
HOBELN	x	x
FRAESEN	x	x
DREHEN	x	x
BOHREN	x	x
SCHLEIFEN	x	x

An die Stellen x werden die aktuellen Zahlenwerte geschrieben.

```
10 REM PROGRAMM ARBEITSGANGVERTEILUNG
20 REM EINGABEVARIABLE :   A¤
30 REM MOEGLICHE EINGABEN :     SG, HO, FR, DR,
40 REM ,                        BO, SL, EN
50 REM - EN - IST SCHLUSSZEICHEN
60 REM
70 REM EINGABEDIALOG
80 REM
90 S1=0 : H=0 : F=0 : D=0 : B=0 : S2=0
100 PRINT "EINGABE DER ARBEITSGAENGE"
110 DIM A¤=2
120 INPUT A¤
130 IF A¤="EN" THEN 360
140 IF A¤="SG" THEN 240
150 IF A¤="HO" THEN 260
160 IF A¤="FR" THEN 280
170 IF A¤="DR" THEN 300
180 IF A¤="BO" THEN 320
190 IF A¤="SL" THEN 340
200 PRINT "FEHLEINGABE ! SIE HABEN : ";A¤
210 PRINT "EINGEGEBEN"
220 PRINT "GEBEN SIE NEU EIN"
230 GOTO 120
240 S1=S1+1
250 GOTO 120
260 H=H+1
270 GOTO 120
280 F=F+1
290 GOTO 120
300 D=D+1
310 GOTO 120
320 B=B+1
330 GOTO 120
340 S2=S2+1
350 GOTO 120
360 REM
370 REM AUSGABEGESTALTUNG
375 PRINT CHR¤(12)
380 S=S1+H+F+D+B+S2
390 PRINT "                                                                    "
400 PRINT "|     ARBEITS-      |    ABSOLUTE       |   PROZENTUALER    |"
410 PRINT "|     GANG          |    ANZAHL         |   ANTEIL          |"
420 PRINT "|___________________|___________________|___________________|"
430 PRINT "|";TAB(19);"|";TAB(39);"|";TAB(58);"|"
440 PRINT "|     SAEGEN        |";TAB(25);S1;TAB(39);"|";
450 PRINT TAB(45);S1*100/S;TAB(58);"|"
460 PRINT "|     HOBELN        |";TAB(25);H;TAB(39);"| ";TAB(45);H*100/S;
470 PRINT TAB(58);"|"
480 PRINT "|     FRAESEN       |";TAB(25);F;TAB(39);"|";TAB(45);F*100/S;
490 PRINT TAB(58);"|"
500 PRINT "|     DREHEN        |";TAB(25);D;TAB(39);"|";TAB(45);D*100/S;
510 PRINT TAB(58);"|"
520 PRINT "|     BOHREN        |";TAB(25);B;TAB(39);"|";TAB(45);
530 PRINT B*100/S;TAB(58);"|"
540 PRINT "|     SCHLEIFEN     |";TAB(25);S2;TAB(39);"|";TAB(45);
550 PRINT S2*100/S;TAB(58);"|"
560 PRINT "|";TAB(19);"|";TAB(39);"|";TAB(58);"|"
570 PRINT "                                                                    "
580 STOP
```

Aufgaben zu mehrfachverzweigten Algorithmen

Gefordert wird als Lösung jeweils der lösende Algorithmus in Form des Programmablaufplanes und des BASIC-Programms.

A 14. In einem Sortierprozeß fallen Meßdaten an, die einem Rechner wie folgt zugeführt werden: x_1, x_2, x_3,...,-1.
Es bedeutet:

10,95 $\leq$ Xi $\leq$ 11,05: Qualität 1
Xi > 11,05: Qualität 2
Xi < 10,95: Rücksendung

Zu berechnen ist: a) der prozentuale Anteil der Rücksendungen
b) die Anzahl der Qualität 1 sowie die Anzahl der Qualität 2.

A 15. In einem Rechner werden über die Tastatur Kantenmaße a,b und c von beliebig vielen Quadern eingegeben. Schlußzeichen ist eine -1. Gesucht wird:

- das Gesamtvolumen aller Quader
- die Anzahl aller Quader mit mindestens zwei einander gleichen Kantenmaßen.

A 16. Als Datenmaterial stehen von beliebig vielen Rechtecken die Seitenmaße a und b (reelle Zahlen) bereit. Schlußzeichen ist eine -1. Es soll berechnet und gedruckt werden:

a) die Gesamtfläche F aller Rechtecke
b) die Anzahl k der Rechtecke mit den Seitenmaßen 3 und 5 (dabei kann a = 3 und b = 5, aber auch b = 3 und a = 5 sein).

Datenfolge: a_1, b_1, a_2, b_2, ... , -1.

A 17. Einem Rechner werden die Kartesischen Koordinaten x, y von beliebig vielen Punkten einer Ebene zugeführt. Schlußzeichen ist -10000. Gesucht wird:

- die Anzahl der Punkte innerhalb des 1. Quadranten
- die Anzahl der Punkte, die auf den Achsen liegen.

A 18. Zur Planung von Investitionen wird jede Einweisung auf die Stationen einer Klinik durch die Angabe der Stations-

nummer registriert. Es existieren 5 Stationen, so daß folgendes Datenmaterial entsteht:
1, 1, 3, 4, 1, 5, 5, 4, 2, 2, ... , -1
Gesucht ist die Anzahl der Einweisungen je Station und ihr prozentualer Anteil.

3.5. Algorithmen zur Arbeit mit Feldern

3.5.1. Einführende Bemerkungen

Viele technische und ökonomische Probleme lassen sich durch sogenannte Vektoren und Matrizen sehr übersichtlich darstellen und mit den für diese mathematischen Gebilde definierten Rechenvorschriften elegant bearbeiten. Beispielsweise kann man die morgens, mittags und abends gemessenen Temperaturen in den Räumen R1, R2, R3 und R4 wie folgt übersichtlich aufschreiben:

Zeit / Räume	morgens	mittags	abends
R1	T_{1mo}	T_{1mi}	T_{1a}
R2	T_{2mo}	T_{2mi}	T_{2a}
R3	T_{3mo}	T_{3mi}	T_{3a}
R4	T_{4mo}	T_{4mi}	T_{4a}

Die Ordnung ist derart, daß alle Temperaturen, die zu einem Raum gehören, in einer Zeile geschrieben sind. Der erste Index an T (Temperatur) sagt dabei, zu welchem Raum die Temperatur gehört. Da 4 Räume vorliegen, hat unsere Matrix 4 Zeilen.

Alle Temperaturen, die zu einer Tageszeit gehören, sind zudem in eine Spalte geschrieben. Der zweite Index von T sagt aus, zu welcher Tageszeit die Temperatur gehört.

Um von den Bezeichnungen unabhängig zu werden, kann man schreiben

$$\text{Temperaturmatrix} = \begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \\ T_{41} & T_{42} & T_{43} \end{pmatrix}$$

Es liegt eine Matrix von Temperaturwerten vor. Sie besteht aus 4 Zeilen und 3 Spalten. Der Temperaturwert T_{32} ist beispielsweise jener, der im Raum R3 (Zeilennummer 3) mittags (Spaltennummer 2) gemessen wurde.

Wie übersichtlich es sich mit Matrizen arbeiten läßt, sollen zwei Algorithmen zeigen. Im ersten Algorithmus (Bild 40) wird die Frage nach dem mittleren Temperaturwert des Raumes R2 beantwortet. Das wesentliche dieses Algorithmus besteht darin, daß der Name der Variablen T_{ij} durch konkrete Wertzuweisungen für i und j verändert wird und damit symbolisch durch die Anweisung i = 2 und j = 1 der Temperaturwert T_{21}, also der erste Wert der 2. Zeile in die Summe S gebracht wird. Indem man i konstant auf dem Wert 2 hält und j von 1 bis 3 variiert, wird in die Summe S nacheinander T_{21}, T_{22}, T_{23} eingebracht. Die abschließende Division von S durch j bringt den arithmetischen Mittelwert zur Anzeige.

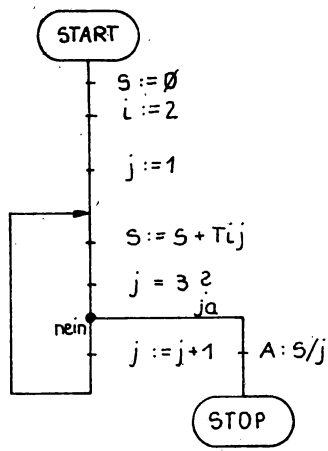


Bild 40:
PAP zur Bildung des Mittelwertes der 2. Zeile

Im zweiten Algorithmus (Bild 41) wird die Summe der 3. Spalte gebildet, so daß der mittlere Temperaturwert der vier Räume am Abend berechnet wird.
Wesentlich ist auch hier, daß der Name der Variablen durch Aktualisierung von i und j verändert wird.
Durch j = 3 und der Variation von i werden die Variablen

T_{31}, T_{32}, T_{33} und T_{34} angesprochen.

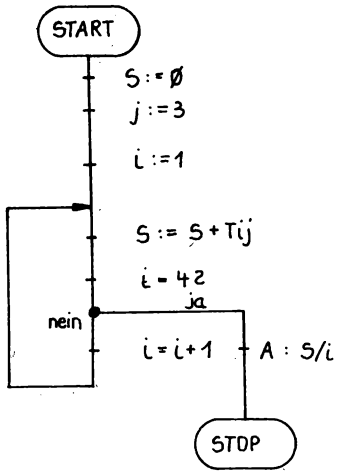


Bild 41:
PAP zur Bildung des Mittelwertes der 3. Spalte

Während Matrizen zweidimensionale mathematische Gebilde sind, sind Vektoren sogenannte eindimensionale Matrizen. Das heißt, eine Zeile (→ Zeilenvektor) oder eine Spalte (→ Spaltenvektor) kann als Vektor betrachtet werden. Die Variation der Variablennamen wird nur durch einen Index realisiert.

$$X = x_1, x_2, x_3, x_4, \ldots, x_i, \ldots, x_n$$

Der Index i des Vektors X zur Bezeichnung der Komponenten läuft von 1 bis n, sagt man.

3.5.2. Realisierung von Vektoren und Matrizen im Speicher des Rechnerarbeitsplatzes

Der BASIC-Interpreter sichert die Ablage von Vektoren und Matrizen im Arbeitsspeicher des Rechnerarbeitsplatzes wenn man im Programm die Größe des Vektors oder der Matrix dimensioniert. Das geschieht mit der Anweisung.

Znr DIM variablenname (größter Zeilenindex, größter Spaltenindex)

Beispielsweise würde die Anweisung

```
2Ø  DIM T (3,2)
```

insgesamt 12 = (3+1) . (2+1) Realspeicherplätze im Arbeitsspeicher reservieren. Die einzelnen Speicherplätze haben folgende Namen

T(Ø,Ø), T(Ø,1), T(Ø,2), T(1,Ø), T(1,1), T(1,2)
T(2,Ø), T(2,1), T(2,2), T(3,Ø), T(3,1), T(3,2).

Der Interpreter beginnt also mit dem Index Ø. Das stiftet zuweilen Verwirrung, da das in der mathematischen Beschreibung unüblich ist. Man muß sich daran erst gewöhnen.
Daß der Interpreter im obigen Beispiel 12 Realspeicherplätze reserviert, liegt an dem Variablennamen.
Die Anweisung

```
2Ø  DIM T% (3,2)
```

würde 12 Integerspeicherplätze mit den Namen T%(Ø,Ø)...T%(3,2) bereitstellen und die Anweisung

```
2Ø  DIM T¤ (3,2)
```

12 Stringplätze mit den Namen

T¤ (Ø,Ø)...T¤ (3,2). Dabei hat jeder Stringplatz eine Kapazität von 120 Byte. Soll diese Kapazität variiert werden, so würde

```
2Ø  DIM T¤ (3,2) = 1Ø
```

die Kapazität jedes T¤ (i,j) auf 1Ø Bytes beschränken und die

Anweisung

```
2Ø  TIM T¤ (3,2) = 18Ø
```

die Kapazität, jedes T¤ (i,j) auf 18Ø Bytes erhöhen.

- Programm zum Einlesen einer Matrix

Die Matrix $T = \begin{pmatrix} 2Ø,3 & 21,Ø & 2Ø,8 \\ 19,7 & 21,2 & 19,8 \\ 18,3 & 2Ø,3 & 2Ø,3 \\ 2Ø,Ø & 22,1 & 21,Ø \end{pmatrix}$

soll im Rechnerarbeitsplatz gespeichert werden; sie besteht aus 4 Zeilen und 3 Spalten. Zu dimensionieren ist also

```
Znr DIM T (3,2).
```

Folgendes Programm realisiert die zeilenweise Eingabe der Matrix:

```
10 DIM T(3,2)
20 I=0
30 J=0
40 PRINT "EINGABE VON T(";I;",";J;")"
50 INPUT T(I,J)
60 IF J=2 THEN 90
70 J=J+1
80 GOTO 40
90 IF I=3 THEN 120
100 I=I+1
110 GOTO 30
120 PRINT "EINGABE BEENDET"
130 STOP
```

Das Programm realisiert folgenden Dialog:

Anforderung des Rechners	Reaktion des Nutzers
EINGABE VON	Eingeben von
T (Ø,Ø)	20,3
T (Ø,1)	21,Ø
T (Ø,2)	2Ø,8
T (1,Ø)	19,7
T (1,1)	21,2
T (1,2)	19,8
T (2,Ø)	18,3
T (2,1)	2Ø,3
T (2,2)	2Ø,3
T (3,Ø)	2Ø,Ø
T (3,1)	22,1
T (3,2)	21,Ø
EINGABE BEENDET	

Nach der Abarbeitung des Einleseprogramms stehen nun die Matrixelemente 2Ø,3 ... 21,Ø in den Feldelementen T(Ø,Ø) ... T(3,2).
Die Feldelemente (Feldvariablen) sind in Programmen genau so verwendbar, wie die schon bekannten einfachen Variablen.
Es ist nur darauf zu achten, daß die Indizes immer einen konkreten Wert haben müssen. In dem Programm

```
10 DIM T(3,2)
20 C=T(I,J)+T(K,M)
30 STOP
```

ist zum Beispiel unklar, wie groß I, J, K und M sind. Demzufolge ist nicht klar, welche Feldelemente zu addieren sind.

Im folgenden Programm dagegen ist alles klar:

```
10 DIM T(3,2)
20 I=1
30 J=2
40 K=0
50 M=2
60 C=T(I,J)+T(K,M)
70 STOP
```

Hier wird die Addition von T(1,2) und T(Ø,2) realisiert und das Ergebnis auf dem Speicherplatz C abgelegt.

Als Beispiel soll im Anschluß an das Einleseprogramm die mittlere Temperatur des Raumes R2 berechnet werden. Der Algorithmus dazu ist im Bild 40 angegeben. Im BASIC-Programm ist zu beachten, daß die Indizes der Feldelemente stets um 1 niedriger sind, als im PAP angegeben. Die Temperaturen des Raumes R2 stehen in den Feldelementen T(1,Ø), T(1,1), T(1,2).

In der Ausgabeanweisung muß die Summe S durch J + 1 dividiert werden, da 3 Temperaturwerte in die Summe gebracht werden.

```
10 DIM T(3,2)
20 I=0
30 J=0
40 PRINT "EINGABE VON T(";I;" ,";J;" )"
50 INPUT T(I,J)
60 IF J=2 THEN 90
70 J=J+1
80 GOTO 40
90 IF I=3 THEN 120
100 I=I+1
110 GOTO 30
120 PRINT "EINGABE BEENDET"
130 S=0
140 I=1
150 J=0
160 S=S+T(I,J)
170 IF J=2 THEN 200
180 J=J+1
190 GOTO 160
200 PRINT "R2-MITTEL =";S/(J+1)
210 STOP
```

- Algorithmus zur Maximumsuche in einem Vektor

Ein Vektor ist eine eindimensionale Matrix. Der Vektor entspricht in der Rechentechnik also einem eindimensionalen Feld. Durch die Anweisung

```
Znr DIM X (1Ø)
```

werden im Arbeitsspeicher des Rechners 11 Realspeicherplätze mit dem Namen

X(Ø), X(1), X(2), X(3), X(4), X(5), X(6), X(7), X(8), X(9) und X(1Ø)

bereitgestellt. Die Namen der Feldelemente des Vektors X dürfen nicht mit dem Namen der einfachen Variablen X1, X2, ..., X9 verwechselt werden. Diese sind neben den Feldelementen zur Nutzung in Programmen zugelassen.

Als Beispiel soll angenommen werden, daß 1Ø beliebige reelle Zahlen vorliegen, von denen das Maximum zu bestimmen ist. Die 1Ø reellen Zahlen sollen in ein Feld eingelesen und anschließend durchgemustert werden. Anzugeben ist der Index des Maximalelements und der Wert des Maximums.

Das Einleseprogramm ist relativ einfach:

```
10 DIM X(9)
20 I=0
30 PRINT "EINGABE VON X(";I;")"
40 INPUT X(I)
50 IF I=9 THEN 80
60 I=I+1
70 GOTO 30
80 PRINT "EINGABE BEENDET"
```

Komplizierter ist das Suchprogramm, um den Maximalwert und dessen Index zu finden. Eine Möglichkeit wäre:

1. Man nimmt die erste Zahl, das ist das Feldelement X(Ø), und sagt: Das ist das Maximum. Dazu schreibt man sich Wert und Index auf ein Blatt Papier.

2. Man nimmt nun X(1) und vergleicht mit dem Maximum, ist X(1) größer als das Maximum, dann streicht man dieses Maximum und sagt: X(1) ist das Maximum. Ist X(1) aber kleiner oder gleich dem Maximum, so behält man X(Ø) als Maximum bei.

3. So verfährt man mit allen Feldelementen, zum Schluß steht der Wert und der Index des Maximalelements auf dem Papier.

In Bild 42 ist der Algorithmus in Form des PAP aufgeschrieben.

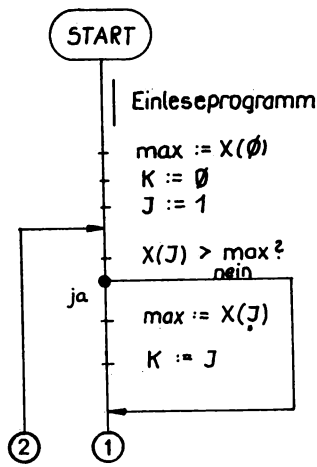
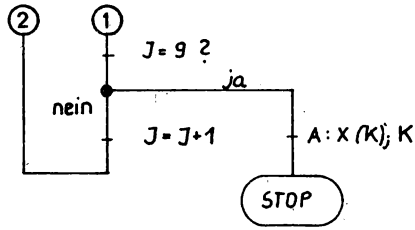


Bild 42:
PAP - lösender Algorithmus zur Maximumsuche in einem eindimensionalen Feld mit 1Ø Feldelementen



Danach lautet das vollständige BASIC-Programm:

```
10 REM MAXIMUMSUCHE IM VEKTOR X(9)
20 DIM X(9)
30 I=0
40 PRINT "EINGABE VON X(";I;")"
50 INPUT X(I)
60 IF I=9 THEN 90
70 I=I+1
80 GOTO 40
90 PRINT "EINGABE BEENDET"
100 M=X(0)
110 K=0
120 I=1
130 IF X(I)>M THEN 150
140 GOTO 170
150 M=X(I)
160 K=I
170 IF I=9 THEN 200
180 I=I+1
190 GOTO 130
200 PRINT "MAXIMALWERT =";X(K);" INDEX =";K
210 STOP
```

3.5.3. Laufanweisung

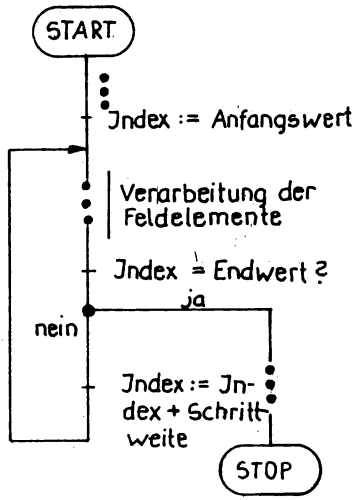
Die bisherigen Überlegungen und Beispiele des Abschnitts 3.5.2. führten auf Algorithmen, die im wesentlichen folgende Grundstruktur hatten:

1. Variable, die als Indizes von Feldelementen dienten, wurden auf einen Anfangswert gesetzt.

2. Mit dem damit festgelegten Feldelement wurde entsprechend der konkreten Aufgabe "gearbeitet".
   Beispiele dazu sind:
   - Mit Wert versehen im Einleseprogramm;
   - vergleichen mit Maximalwert bei der Maximumsuche;
   - in die Summe gebracht beim Mittelwertbilden.

3. Das unter 2. angegebene "Verarbeiten" wurde schrittweise für alle Feldelemente realisiert, bis der Index den Endwert erreicht hat.

Diese allgemeine Struktur ist im Bild 43 nochmals dargestellt.



Für diese Struktur kennen alle problemorientierten Programmiersprachen eine spezielle Anweisung, die sogenannte Laufanweisung.

Sie hat im allgemeinen folgende Form:

```
FÜR Laufvariable = Anfangswert
BIS zum Endwert
IN SCHRITTEN der Schrittweite
MACHE:  Anweisung 1
        Anweisung 2
          :
        Anweisung n
ENDE der Laufanweisung
```

Bild 43: PAP - allgemeine Struktur der Feldverarbeitung

In BASIC hat die Laufanweisung folgende allgemeine Form:

```
Znr FOR Laufvariable = Anfangswert TO Endwert STEP Schrittweite
Znr     Anweisung 1
 :        :
Znr     Anweisung n
Znr NEXT Laufvariable
```

Als Beispiel soll das Einleseprogramm für den Vektor X(9) aus Abschnitt 3.5.2. nochmals angegeben und unter Verwendung einer Laufanweisung umgeschrieben werden.

```
10 REM EINLESEPROGRAMM
20 REM SO WIE BISHER
30 DIM X(9)
40 I=0
50 PRINT "EINGABE VON X(";I;")"
60 INPUT X(I)
```

```
70 IF I=9 THEN 100
80 I=I+1
90 GOTO 50
100 PRINT "EINGABE BEENDET"
```

Nun das gleiche Programm unter Verwendung der Laufanweisung:

```
10 REM EINLESEPROGRAMM
20 REM MIT LAUFANWEISUNG
30 DIM X(9)
40 FOR I=0 TO 9 STEP 1
50 PRINT "EINGABE VON X(";I;")"
60 INPUT X(I)
70 NEXT I
80 PRINT "EINGABE BEENDET"
```

Das Programm wurde kürzer und übersichtlicher. Das, was mit dem Feldelement zu "arbeiten" ist, steht zwischen den Zeilen

```
      4Ø  FOR I=Ø TO 9  STEP 1
      :          :
und   7Ø  NEXT I
```

FOR	bedeutet Für
TO	bedeutet Bis
STEP	bedeutet In Schritten von
NEXT	bedeutet Nächster Wert der Laufvariablen (ergibt sich immer aus altem Wert plus Schrittweite)

Die Anweisungen zwischen den Zeilen 4Ø und 7Ø werden realisiert für die Werte von I = Ø,1,2,3,4,5,6,7,8 und 9. Nach dem die Anweisungen für den Wert I = 9 realisiert sind, bildet der Interpreter abermals den nächsten Wert I. Da I nun jedoch größer als der Endwert ist, werden die Anweisungen zwischen den Zeilen 4Ø und 7Ø nicht mehr ausgeführt und die Laufanweisung wird verlassen. Das Programm wird mit Zeile 8Ø fortgesetzt. Der Wert der Laufvariablen nach Verlassen der Laufanweisung ist jedoch I = Endwert + Schrittweite = 1Ø.

Auch bei der Darstellung von Algorithmen mit dem PAP hat sich die verkürzende Form der Laufanweisung praktisch durchgesetzt. Es muß aber darauf hingewiesen werden, daß diese Darstellungsform nicht der TGL 22451 entspricht (Bilder 44 und 45).

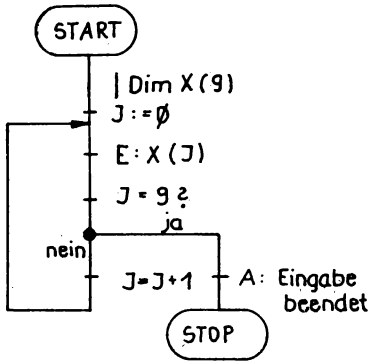


Bild 44: PAP-Einlesealgorithmus nach TGL 22451

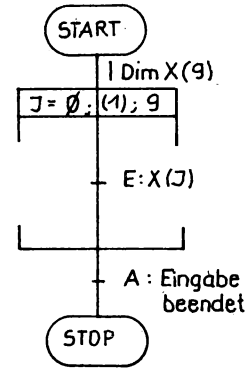
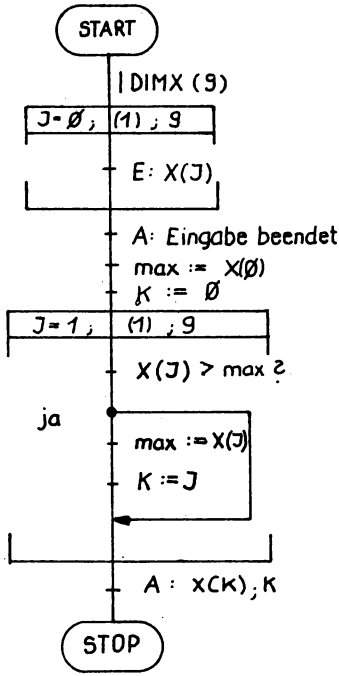


Bild 45: PAP-Einlesealgorithmus mit Darstellung der Laufanweisung

Die Vereinfachung in der Darstellungsvariante mit Laufanweisung leuchtet ein. Das, was innerhalb der Laufanweisung "geschehen" soll, wird in die Klammer geschrieben. Der Klammerkopf gibt an:



Laufvariable = Anfangswert, (Schrittweite), Endwert.

Das Programm MAXIMUMSUCHE IM VEKTOR X(9) des Abschnitts 3.5.2. hat unter Verwendung der Darstellungsform mit Laufanweisung den PAP nach Bild 46.

Bild 46:
PAP-Maximumsuche mit Laufanweisung

Das dem Algorithmus nach Bild 46 entsprechende BASIC-Programm lautet:

```
10 REM MAXIMUMSUCHE IM VEKTOR X(9)
20 DIM X(9)
30 FOR I=0 TO 9 STEP 1
40 PRINT "EINGABE VON X(";I;" )"
50 INPUT X(I)
60 NEXT I
70 PRINT "EINGABE BEENDET"
80 M=X(0)
90 K=0
100 FOR I=1 TO 9 STEP 1
110 IF X(I)>M THEN 130
120 GOTO 150
130 M=X(I)
140 K=I
150 NEXT I
160 PRINT "MAXIMALWERT =";X(K);" INDEX =";K
170 STOP
```

Auch das Einleseprogramm für die Matrix T aus dem Abschnitt 3.5.1. kann mit 2 Laufanweisungen geschrieben werden. Dabei tritt der Fall ein, daß diese Laufanweisungen ineinander geschachtelt sind. Die folgenden Laufanweisungen geben ein Beispiel für die Abarbeitung geschachtelter Laufanweisungen:

```
10 FOR I=0 TO 2 STEP 1
20 FOR J=0 TO 2 STEP 1
30 PRINT I;J
40 NEXT J
50 NEXT I
60 STOP
```

Die I-Laufanweisung stellt I auf Ø

Die J-Laufanweisung gibt nun aus: Ø Ø
Ø 1
Ø 2

Jetzt ist die J-Laufanweisung beendet, die nächste Anweisung ist jedoch Zeile 5Ø NEXT I, also

Die I-Laufanweisung stellt I auf 1

Die J-Laufanweisung beginnt von vorn: 1 Ø
1 1
1 2

Die J-Laufanweisung ist abermals beendet.

Die I-Laufanweisung stellt I auf 2

Die J-Laufanweisung beginnt von vorn: 2 Ø
2 1
2 2

Beide Laufanweisungen sind beendet.

Das Einleseprogramm für die Matrix T kann also wie folgt geschrieben werden:

```
10 DIM T(3,2)
20 FOR I=0 TO 3 STEP 1
30 FOR J=0 TO 2 STEP 1
40 PRINT "EINGABE VON T(";I;",";J;")"
50 INPUT T(I,J)
60 NEXT J
70 NEXT I
80 PRINT "EINGABE BEENDET"
90 STOP
```

Aufgaben zu Algorithmen mit Feldern

A19. Anzugeben sind die BASIC-Programmteile zur Dimensionierung von Feldern:

19.1. Es werden benötigt eine 3 x 4 Matrix und ein Vektor der Dimension 1Ø.

19.2. Zu dimensionieren ist ein eindimensionales Feld. Die Größe des Feldes wird erst durch eine Tastatureingabe aktualisiert.

19.3. Wie 19.2. für ein zweidimensionales Feld.

Als Lösung der nachfolgenden Aufgaben ist jeweils der PAP und das BASIC-Programm aufzustellen:

A20. Eine 4 x 5 Matrix ist spaltenweise einzulesen.

A21. Von einer 3 x 4 Matrix, deren Elemente positive reelle Zahlen sind, sollen gesucht werden:
21.1. alle Spaltenminima,
21.2. alle Zeilenmaxima.

A22. Zwei Felder A(3,3) und B(3,3) sind zu addieren und im Feld C(3,3) abzulegen.
$C_{ij} = A_{ij} + B_{ij}$.
Das Feld C ist auf dem Bildschirm auszugeben.

A23. In einem Feld A(n,m) ist das Maximalelement zu suchen. Auszugeben ist der Wert des Maximalelements und die Indizes. n und m sind über die Tastatur durch das Programm anzufordern.

A24. Zwei Felder A(1ØØØ) und B(1ØØØ) sollen wie folgt gemischt werden:

C(Ø) := A(Ø) ; C(1) := B(Ø)

C(2) := A(1) ; C(3) := B(1) usw.

4. Strukturierte Algorithmen

4.1. Grundgedanke

Wohldurchdachte Ordnung im Sinne der Übersichtlichkeit im Programm erleichtert das Verständnis, die Anwendung, die Wartung und die Änderung. Das ist der Grund dafür, daß beim Entwurf von Programmen und Algorithmen immer stärker auf "Wohlstrukturiertheit" geachtet wird. Raffinesse und besonders ausgeknobelte Programmiertricks treten dagegen in den Hintergrund. Der strukturierte Entwurf geht davon aus, daß

- die Programme und Algorithmen sich aus standardisierten Grundelementen zusammensetzen, wobei
- die Grundelemente jeweils nur einen Eingang und einen Ausgang haben.

Programme sind dann hierarchische Strukturen der Grundelemente (Bild 47), Folgen (Bild 48) von Grundelementen oder aus beiden gemischte Strukturen (Bild 49).

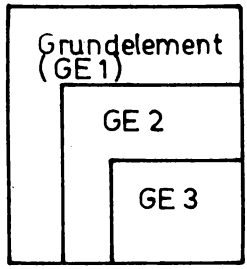


Bild 47: Struktogramm hierarchische Struktur

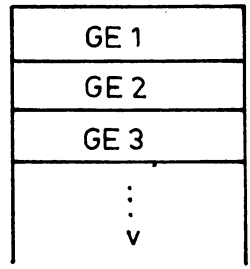


Bild 48: Struktogramm - Folge

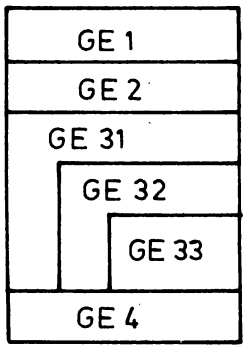


Bild 49: Struktogramm - gemischte Struktur

Die Darstellungsformen von Algorithmen unterstützen das Vorhaben der strukturierten Programmierung mehr oder weniger gut. Besonders gut leisten das
- von den grafischen Darstellungsformen, das Struktogramm
- von den sprachlichen Formen, der sogenannte Pseudocode.

Aber auch einige, besonders fast alle modernen problemorientierten Programmiersprachen unterstützen die strukturierte Programmierung. Als Beispiel sei hier die Sprache PASCAL genannt, die sich einer zunehmenden Verbreitung erfreut. BASIC dagegen ist eine Sprache, die die strukturierte Programmierung weniger unterstützt. Das zeigt sich beispielsweise an der Anweisung

G O T O Zeilennummer.

Diese Anweisung gestattet das "Springen" an jede beliebige Zeilennummer ohne Einschränkung. Somit ist aber unter Umständen ein neuer Eingang in einen Algorithmenabschnitt geschaffen. Trotzdem kann natürlich der Programmierer im Wissen um die Vorteile der strukturierten Programmierung derartige Sprachelemente mit der notwendigen Sorgfalt einsetzen und somit dem Anliegen der Strukturierung Rechnung tragen.

4.2. Logische Grundstrukturen

Als Grundelemente werden in der strukturierten Programmierung logische Grundstrukturen definiert. Drei wesentliche davon sind

- die Folge (Sequenz)
- die einfache Verzweigung (Alternative)
- die bedingte Wiederholung (Iteration).

Nachfolgend soll für diese logischen Grundstrukturen jeweils

- der Programmablaufplan (PAP)
- das Struktogramm (STG)
- der Pseudocode (PSC)
- die BASIC-Formulierung (BASIC)

angegeben werden.

4.2.1. Folge

Die Folge oder Sequenz von Aktionen ist die einfache Aneinandersetzung von Aktionen, die der Computer in eben dieser Reihenfolge abarbeiten soll.

Als Aktion soll dabei eine einfache, nicht mehr unterteilbare Beauflagung eines Computers verstanden werden. In BASIC könnte das zum Beispiel eine Anweisung der Form

```
LET A = B + 3 * C
```

sein.

- Darstellung

Programmablaufplan:

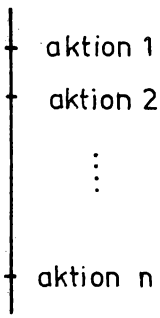


Bild 50: PAP-Folge

Struktogramm:

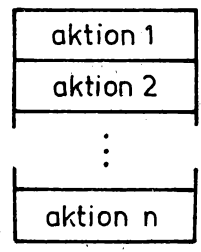


Bild 51: STG-Folge

Pseudocode:

```
SEQ
  aktion_1
  aktion_2
  :
  aktion _n
ENDSEQ
```

BASIC-Variante:

```
Zeilennummer    Anweisung  1
znr             Anweisung  2
 :                  :
znr             Anweisung  n
```

Bei der Darstellung der Folge im Pseudocode (PSC) fällt auf, daß die logische Grundstruktur einen eindeutigen Beginn (Symbol SEQ) und ein eindeutiges Ende (Symbol ENDSEQ) hat. Diese

Form der Darstellung zeigt schon die vollständige Anwendung des Prinzips der Wohlstrukturiertheit. Der Querstrich bei den aktionen und den aktionsnummern bedeutet, daß z. B. der Begriff aktion_1 als ein Begriff gemeint ist. Die Darstellung als aktion 1, also ohne Querstrich, würde zwei Bezeichnungen bedeuten, nämlich "aktion" und "1".

4.2.2. Einfache Verzweigung

Die einfache Verzweigung oder auch Alternative ermöglicht die wahlweise Abarbeitung genau einer Aktion von zwei möglichen in Abhängigkeit von einer Bedingung.

- Darstellung

Programmablaufplan:

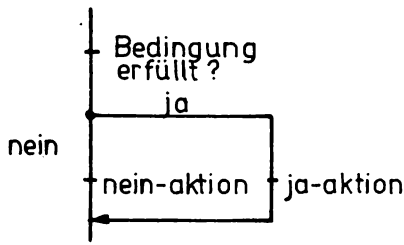


Bild 52: PAP - einfache Verzweigung

Struktogramm:

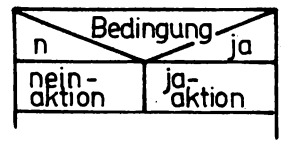


Bild 53: STG - einfache Verzweigung

- Pseudocode:

```
IF (Bedingung) THEN ja_aktion [ELSE nein_aktion]
ENDIF
```

- BASIC-Variante:

```
znr IF bedingung THEN ja-Anweisung [ELSE nein-Anweisung]
```

Die eckigen Klammern deuten an, daß der eingeschlossene Teil auch entfallen kann, wenn ein nein-aktion oder nein-Anweisung nicht gebraucht wird. Es fällt wieder auf, daß in der BASIC-Variante die Darstellung nicht explizit abgeschlossen wird, während dies im Pseudocode mit ENDIF eindeutig geschieht.

4.2.3. Bedingte Wiederholung

Die Grundstruktur Wiederholung oder Iteration kommt zur Anwendung, wenn eine Anweisung oder Aktion in Abhängigkeit von einer bestimmten Bedingung wiederholt ausgeführt werden soll.

Entsprechend der Art der Interpretation der Bedingung unterscheidet man 3 Formen der Grundstruktur Wiederholung:

- die FOR-Form (Indizierte Wiederholung)
- die WHILE-Form (Abweisende Wiederholung)
- die UNTIL-Form (Abbrechende Wiederholung).

- Die FOR-Form der Wiederholung

Die FOR-Form der Wiederholung entspricht der Laufanweisung, die im Abschnitt 3.5.3. dargestellt ist.

Für jeden Wert einer eingeführten Laufvariablen (LV), wird beginnend von einem definierten Anfangswert (AW) in einer festgelegten Schrittweite (SW) bis zu einem Endwert (EW) eine Anweisung oder Aktion genau einmal ausgeführt.

. Darstellung

Programmablaufplan:

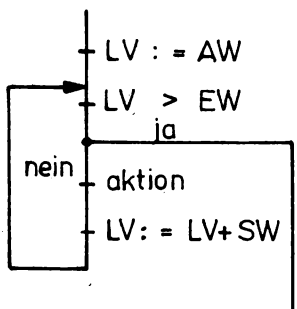


Bild 54: PAP-FOR-Form der Wiederholung vollständig

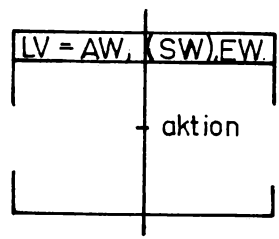


Bild 55: PAP-FOR-Form der Wiederholung verkürzt

Struktogramm:

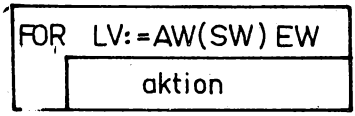


Bild 56: STG-FOR-Form der Wiederholung

Pseudocode:

```
FOR (LV := AW (SW) EW)
    aktion
ENDFOR
```

BASIC-Variante:

```
znr FOR LV = AW TO EW STEP SW
znr Anweisung
znr NEXT LV
```

• Ein Beispiel

Der Algorithmus zur Bestimmung des Maximums eines zweidimensionalen Feldes ist als Lösung der Aufgabe A23 im Lösungsteil als PAP und BASIC-Variante angegeben. Hier soll das Struktogramm und der Pseudocode angegeben werden:

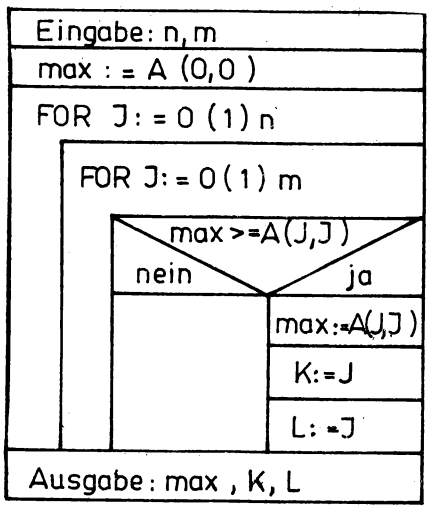


Bild 57:
STG Maximum eines Feldes

Pseudocode:

```
SEQ   Eingabe: n, m
      max := A (0,0)
      FOR I := 0 (1) n
          FOR J := 0 (1) m
             IF  (max := A (I,J)) THEN  max := A (I,J)
                                        K := I
                                        L := J
             ENDIF
          ENDFOR
      ENDFOR
      Ausgabe: max, K, L
 ENDSEQ
```

- Die WHILE-Form der Wiederholung

Eine Anweisung oder Aktion wird solange wiederholt ausgeführt wie eine gegebene Bedingung erfüllt ist.

. Darstellung

Programmablaufplan:

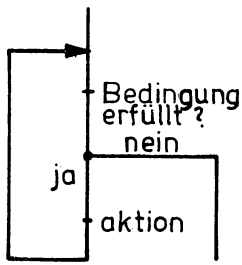


Bild 58: PAP-WHILE-Form der Wiederholung

Struktogramm:

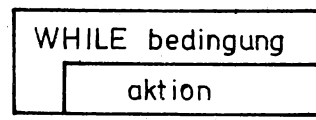


Bild 59: STG-WHILE-Form der Wiederholung

Pseudocode:

```
WHILE  bedingung
     aktion
ENDWHILE
```

BASIC-Variante:

Die WHILE-Form der Wiederholung ist in den meisten BASIC-Versionen nicht definiert.

. Ein Beispiel

Die Lösung der Aufgabenstellung, nach der unendlichen Reihe

$y = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots$ den Sinus zu berechnen, führte im

Abschnitt 3.3. (2. Beispiel) zu dem Algorithmus entsprechend Bild 34.

Der Programmablaufplan und das Struktogramm sowie der Pseudocode zur Darstellung dieses Algorithmus soll nun zum Vergleich gezeigt werden.

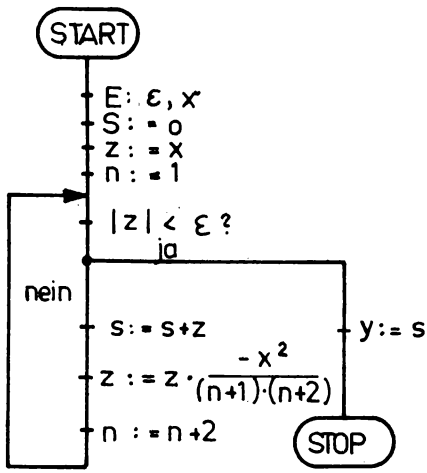


Bild 60: PAP-Sinusbildung

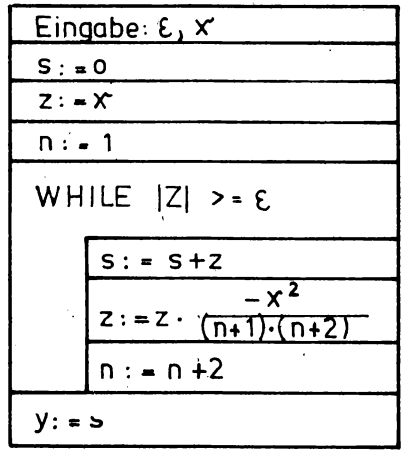


Bild 61:
STG-Sinusbildung

Im Pseudocode erhält der Algorithmus folgende Form:

```
SEQ   Eingabe ε,x
      S := 0
      Z := x
      n := 1
      WHILE (ABS (Z)  >= ε
            S := S+Z
            Z := -(Z * x * x) / ((n+1) * (n+2))
            n := n+2
      ENDWHILE
      y := S
ENDSEQ
```

- Die UNTIL-Form der Wiederholung

Die Anwendung dieser Form der Wiederholung erfolgt durch eine fortlaufende Ausführung einer Anweisung oder Aktion bis eine Bedingung erfüllt ist.

• Darstellung

Programmablaufplan:

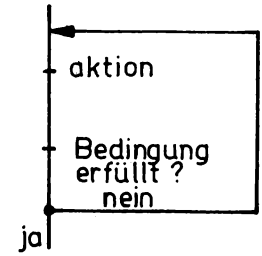


Bild 62: PAP-UNTIL-Form der Wiederholung

Struktogramm:

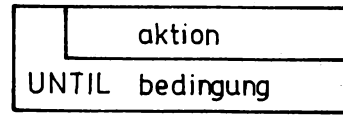


Bild 63: STG-UNTIL-Form der Wiederholung

Pseudocode:

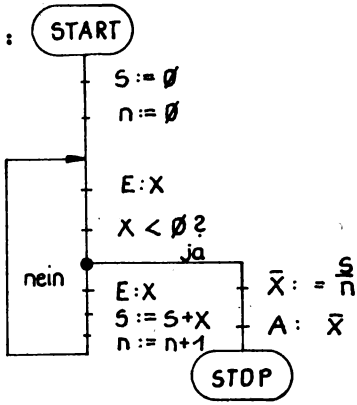
```
UNTIL    (bedingung)
       aktion
ENDUNTIL
```

BASIC-Variante:

Die UNTIL-Form der Wiederholung ist in den meisten BASIC-Versionen nicht definiert.

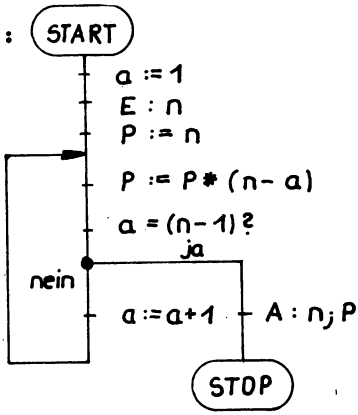
Lösungen zu den Aufgaben

A8.:



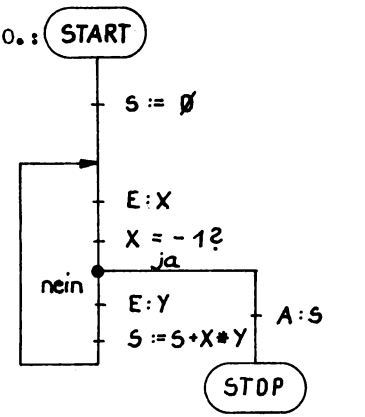

```
10 REM MITTELWERT JEDES
20 REM ZWEITEN EINGABEWERTES
30 S=0
40 N=0
50 INPUT X
60 IF X<0 THEN 110
70 INPUT X
80 S=S+X
90 N=N+1
100 GOTO 50
110 S=S/N
120 PRINT "MITTELWERT =";S
130 STOP
```

A9.:



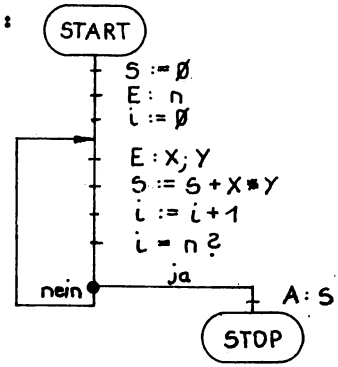

```
10 REM FAKULTAET VON
20 REM N >= 2
30 A=1
40 INPUT N
50 P=N
60 P=P*(N-A)
70 IF A=N-1 THEN 100
80 A=A+1
90 GOTO 60
100 PRINT "N =";N;" N! =";P
110 STOP
```

A10.:



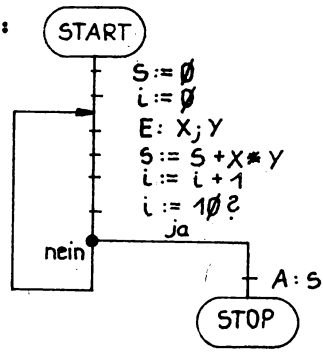

```
10 REM SKALARPRODUKT
20 S=0
30 INPUT X
40 IF X=-1 THEN 80
50 INPUT Y
60 S=S+X*Y
70 GOTO 30
80 PRINT "SKALARP. =";S
90 STOP
```

A11.:



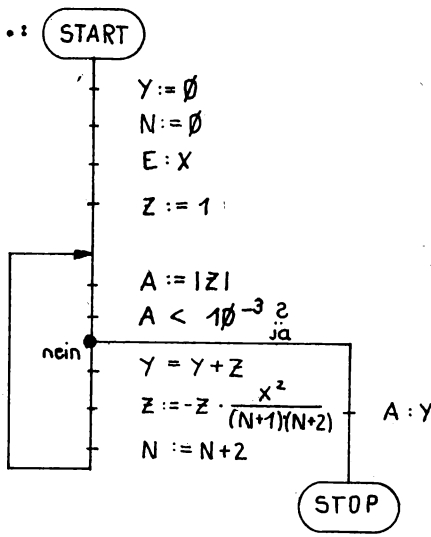

```
10 REM SKALARPRODUKT MIT
20 REM ENDLICHER WERTE-
30 REM PAARANZAHL
40 S=0
50 INPUT N
60 I=0
70 INPUT X,Y
80 S=S+X*Y
90 I=I+1
100 IF I=N THEN 120
110 GOTO 70
120 PRINT "SKALARP. =";S
130 STOP
```

A12.:



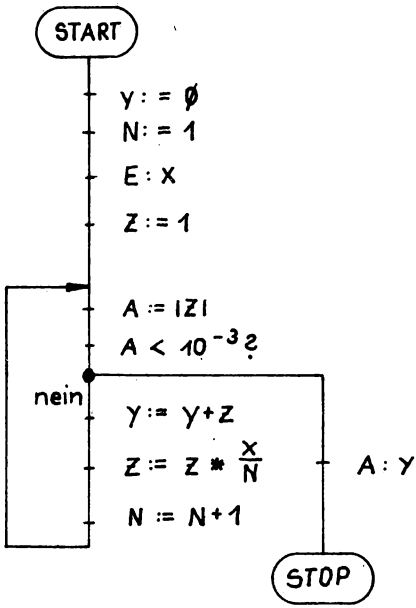

```
10 REM SKALARPRODUKT VON
20 REM 10 WERTEPAAREN
30 S=0
40 I=0
50 INPUT X,Y
60 S=S+X*Y
70 I=I+1
80 IF I=10 THEN 100
90 GOTO 50
100 PRINT "SKALARP. =";S
110 STOP
```

A13.1.:



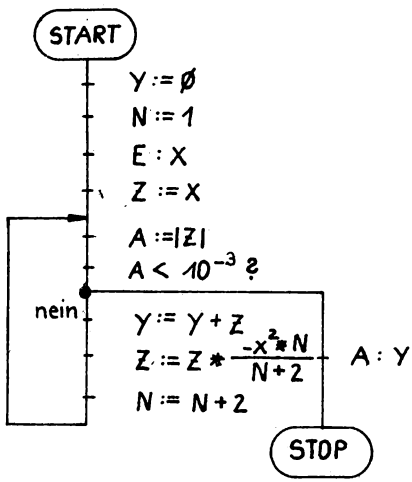

```
10 REM Y=COS(X)
20 Y=0
30 N=0
40 INPUT X
50 Z=1
60 A=ABS(Z)
70 IF A<.001 THEN 120
80 Y=Y+Z
90 Z=-Z*X*X/((N+1)*(N+2))
100 N=N+2
110 GOTO 60
120 PRINT "Y =";Y
130 STOP
```

A13.2.:



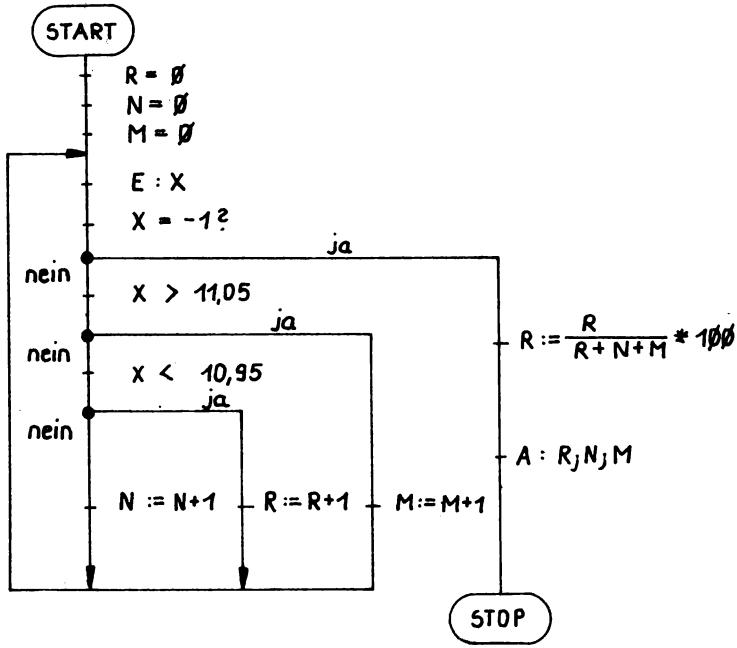

```
10 REM Y = E HOCH X
20 Y=0
30 N=1
40 INPUT X
50 Z=1
60 A=ABS(Z)
70 IF A<.001 THEN 120
80 Y=Y+Z
90 Z=Z*X/N
100 N=N+1
110 GOTO 60
120 PRINT "Y =";Y
130 STOP
```

A13.3.:



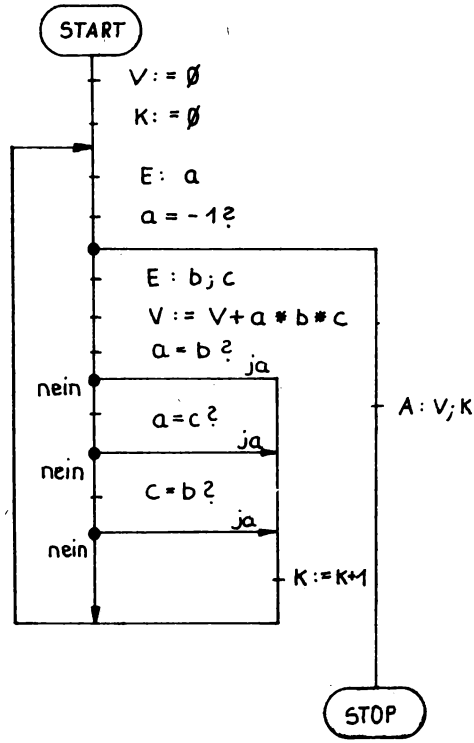

```
10 REM Y= ARCTAN(X)
20 Y=0
30 N=1
40 INPUT X
50 Z=X
60 A=ABS(Z)
70 IF A<.001 THEN 120
80 Y=Y+Z
90 Z=Z*(-X)*X*N/(N+2)
100 N=N+2
110 GOTO 60
120 PRINT "Y =";Y
130 STOP
```

A14.:



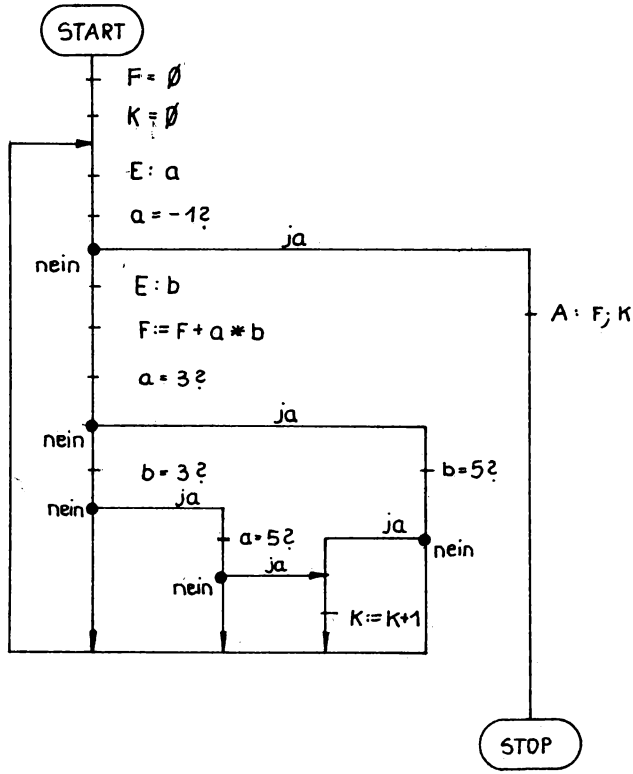

```
10 REM QUALITAET
20 R=0
30 N=0
40 M=0
50 INPUT X
60 IF X=-1 THEN 150
70 IF X>11.05 THEN 130
80 IF X<10.95 THEN 110
90 N=N+1
100 GOTO 50
110 R=R+1
120 GOTO 50
130 M=M+1
140 GOTO 50
150 PRINT "RUECKSENDUNG =";R*100/(R+N+M);"%"
160 PRINT "QUALITAET1 =";N;" STK"
170 PRINT "QUALITAET2 =";M;" STK"
180 STOP
```

A15.:



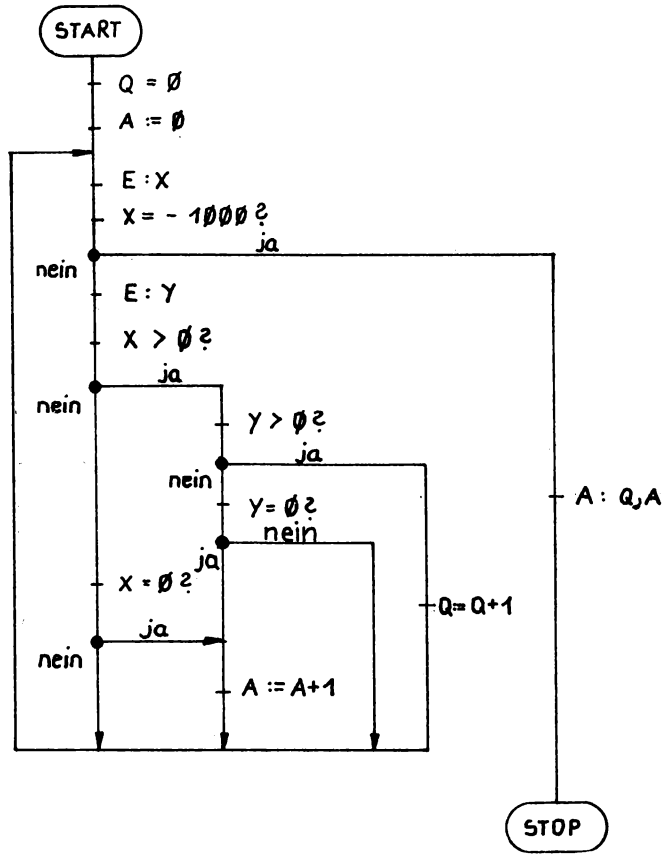

```
10 REM UNTERSUCHUNG VON QUADERN
20 V=0
30 K=0
40 PRINT "EINGABE VON A"
50 INPUT A
60 IF A=-1 THEN 160
70 PRINT "EINGABE VON B UND C"
80 INPUT B,C
90 V=V+A*B*C
100 IF A=B THEN 140
110 IF A=C THEN 140
120 IF C=B THEN 140
130 GOTO 40
140 K=K+1
150 GOTO 40
160 PRINT "GESAMTVOLUMEN =";V
170 PRINT "ANZAHL DER QUADER"
180 PRINT "MIT 2 GLEICHEN KANTEN =";K
190 STOP
```

A16.:



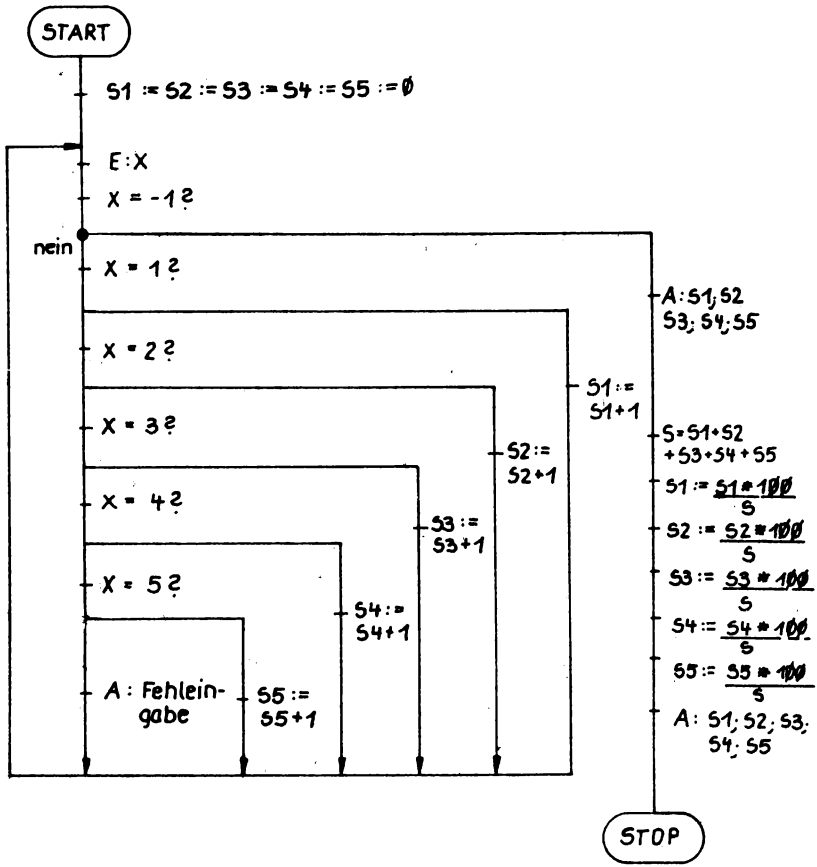

```
10 REM RECHTECKE
20 F=0
30 K=0
40 INPUT A
50 IF A=-1 THEN 170
60 INPUT B
70 F=F+A*B
80 IF A=3 THEN 130
90 IF B=3 THEN 110
100 GOTO 40
110 IF A=5 THEN 150
120 GOTO 40
130 IF B=5 THEN 150
140 GOTO 40
150 K=K+1
160 GOTO 40
170 PRINT "GESAMTFLAECHE =";F;" ANZAHL =";K
180 STOP
```

A17.:




```
10 REM KARTESISCHE KOORDINATEN
20 Q=0
30 A=0
40 INPUT X
50 IF X=-1000 THEN 170
60 INPUT Y
70 IF X>0 THEN 100
80 IF X=0 THEN 130
90 GOTO 40
100 IF Y>0 THEN 150
110 IF Y=0 THEN 130
120 GOTO 40
130 A=A+1
140 GOTO 40
150 Q=Q+1
160 GOTO 40
170 PRINT "IN 1.QUADRANTEN SIND";Q;" PUNKTE"
180 PRINT "AUF DEN ACHSEN LIEGEN";A;" PUNKTE"
190 STOP
```

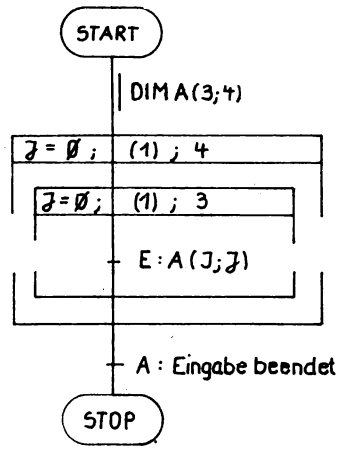
A18.:


START
S1 := S2 := S3 := S4 := S5 := 0
E:X
X = -1 ?
nein
X = 1 ?
X = 2 ?
X = 3 ?
X = 4 ?
X = 5 ?
A: Fehlein-gabe
S5 := S5+1
S4 := S4+1
S3 := S3+1
S2 := S2+1
S1 := S1+1
A: S1; S2 S3; S4; S5
S = S1+S2 +S3+S4+S5
S1 := S1*100 / S
S2 := S2*100 / S
S3 := S3*100 / S
S4 := S4*100 / S
S5 := S5*100 / S
A: S1; S2; S3; S4; S5
STOP

```
10 REM STATIONSPLANUNG
20 S1=0
30 S2=0
40 S3=0
50 S4=0
60 S5=0
70 INPUT X
80 IF X=-1 THEN 260
90 IF X=1 THEN 240
100 IF X=2 THEN 220
110 IF X=3 THEN 200
120 IF X=4 THEN 180
130 IF X=5 THEN 160
140 PRINT "FEHLEINGABE - BITTE NEU EINGEBEN"
150 GOTO 70
160 S5=S5+1
170 GOTO 70
180 S4=S4+1
190 GOTO 70
200 S3=S3+1
210 GOTO 70
220 S2=S2+1
230 GOTO 70
240 S1=S1+1
250 GOTO 70
260 S=S1+S2+S3+S4+S5
270 PRINT "S1 =";S1;" STK";" ENTSPRICHT ";S1*100/S;"%"
280 PRINT "S2 =";S2;" STK";" ENTSPRICHT ";S2*100/S;"%"
290 PRINT "S3 =";S3;" STK";" ENTSPRICHT ";S3*100/S;"%"
300 PRINT "S4 =";S4;" STK";" ENTSPRICHT ";S4*100/S;"%"
310 PRINT "S5 =";S5;" STK";" ENTSPRICHT ";S5*100/S;"%"
320 STOP
```

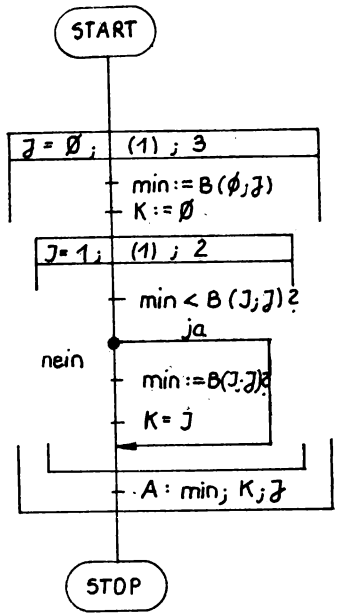
```
A19.1.: 1Ø DIM A (2,3), B(9)

A19.2.: 1Ø PRINT "EINGABE ␣ DER ␣ DIMENSION-N"
        2Ø INPUT N
        3Ø DIM V(N)
             :
A19.3.: 1Ø PRINT "EINGABE ␣ DER ␣ DIMENSIONEN-N,M"
        2Ø INPUT N, M
        3Ø DIM F(N,M)
```

A20.:



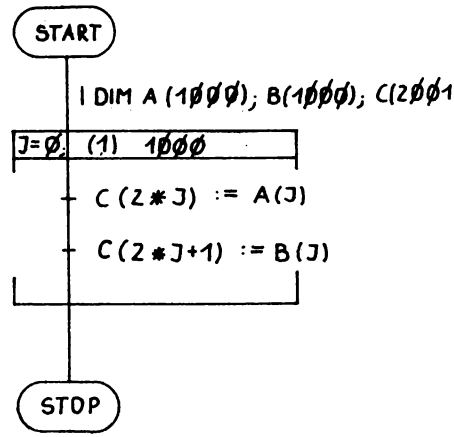

```
10 REM SPALTENWEISES EINLESEN
20 DIM A(3,4)
30 FOR J=0 TO 4 STEP 1
40 FOR I=0 TO 3 STEP 1
50 PRINT "EINGABE A(";I;J;")"
60 INPUT A(I,J)
70 NEXT I
80 NEXT J
90 PRINT "EINGABE BEENDET"
100 STOP
```

A21.1.: Es wird bei der Lösung davon ausgegangen, daß B(2,3) dimensioniert ist.




```
10 REM SPALTENMINIMA
20 FOR J=0 TO 3 STEP 1
30 M=B(0,J)
40 K=0
50 REM INNERE LAUFANWEISUNG
60 FOR I=1 TO 2 STEP 1
70 IF M<B(I,J) THEN 100
80 M=B(I,J)
90 K=I
100 NEXT I
110 PRINT "SPALTENMINIMUM ";J;
120 PRINT " =";M
130 PRINT "ZEILENINDEX =";K
140 NEXT J
150 STOP
```

A24.:




```
10 REM MISCHEN
20 DIM A(1000),B(1000)
30 DIM C(2001)
40 FOR I=0 TO 1000 STEP 1
50 C(2*I)=A(I)
60 C(2*I+1)=B(I)
70 NEXT I
80 STOP
```

Institut für Rationalisierung
der Elektrotechnik/Elektronik
Zentralstelle
für Aus- und Weiterbildung
des Industriebereiches
Elektrotechnik/Elektronik

Lehrmaterial für die
Weiterbildung

Informationsverarbeitung mit Kleincomputern

4

BASIC-Programmierung

Kreutzer
Schuhmann

Elektrotechnik
Elektronik
Automation
DDR

Steffen Kreutzer
Detlef Schuhmann

Informationsverarbeitung mit Kleincomputern

4

BASIC-Programmierung

Herausgeber
Institut für Rationalisierung der Elektrotechnik/Elektronik
Institutsteil Dresden
Zentralstelle für Aus- und Weiterbildung
des Industriebereiches Elektrotechnik/Elektronik
Karl-Marx-Straße, Dresden
8080

Autoren: FSD Dipl.-Ing. Steffen Kreutzer
FSL Dipl.-Ing. Detlef Schuhmann
Ingenieurschule für Wissenschaftlichen Gerätebau
"Carl Zeiss" Jena

Gutachter: Dipl.-Ing. Claus Paul
Institut für Rationalisierung der
Elektrotechnik/Elektronik

Bearbeiter: Dipl.-Gwl. Heinz Rüdger
Institut für Rationalisierung der
Elektrotechnik/Elektronik
Zentralstelle für Aus- und Weiterbildung

1. Auflage
IR Dresden ZSB
Druckgenehmigungs-Nr.: Ag 682/047/87
Druck und Herstellung: NOWA DOBA Bautzen III-4-9-H.958-3.0
Redaktionsschluß: 31.01.1987
Bestell-Nr.: T.2.04.0004
Vorzugsschutzgebühr: 2,50 M

Inhaltsverzeichnis

		Seite
0.	Einführung	5
1.	Grundelemente der Sprache BASIC	5
1.1.	Zeichensatz	5
1.2.	Zahlendarstellung in BASIC	6
1.3.	Konstanten	7
1.4.	Variablen	7
1.4.1.	Erklärung zum Begriff	7
1.4.2.	Variablentypen	8
1.4.3.	Wertzuweisung für Variable	8
1.5.	Aufbau einer Programmzeile	9
2.	Anweisungen	10
2.1.	Kommandos	10
2.2.	Befehle	12
2.2.1.	Erklärung zum Begriff	12
2.2.2.	Ausgabeanweisung	12
2.2.3.	Eingabeanweisung	13
2.2.4.	Programmverzweigung	14
2.2.5.	Zählschleifen	15
2.2.6.	Konstantenfelder	16
2.2.7.	Unterprogramme und Anwenderfunktionen	17
2.2.8.	Felder	19
2.2.9.	Tischrechnermodus	20
3.	Spezielle BASIC-Anweisungen	21
3.1.	Farbe und Grafik	21
3.2.	Bildschirmfenster	22
3.3.	Zeichenketten und Zeichenkettenfunktionen	23
3.4.	Zufallsgenerator	24
3.5.	Sprungverteiler	26
3.6.	Speicherzugriff und Maschinenunterprogramme	27
3.6.1.	Speicherzugriff	27
3.6.2.	Maschinenunterprogramme	28
3.7.	Kursorpositionierung bei Aus- und Eingabe	28
3.7.1.	Ausgabegestaltung	28
3.7.2.	Eingabeanforderung	30
4.	Standardfunktionen	30

Seite

5. Übungen zum Umgang mit dem Kleincomputer KC 85/3 und Programmbeispiele 32
5.1. Übungen zur Programmeingabe und zum Programmtest 32
5.2. Gestaltung von Programmen 36
5.3. Programmbeispiele 40
5.3.1. Sortieren von Datenfolgen 40
5.3.2. Unterprogrammsystem zur Matrizenrechnung 48
5.3.3. Dateiarbeit mit dem KC 85/3 60
6. Befehlsübersicht 65

Literaturverzeichnis 72

0. Einführung

Anliegen dieses Lehrmaterials soll es sein, dem Nutzer ohne spezielle Kenntnis von Rechnersystemen einen Einstieg in die problemorientierte Programmierung mittels BASIC zu ermöglichen. Dabei steht die Nutzung des Rechners als Arbeitsmittel des Ingenieurs im Mittelpunkt. Die im Heft angeführten Beispiele sollten zur Übung sofort am Rechner nachvollzogen werden.

Die Beschreibung der Sprache BASIC erfolgt auf der Basis einer Implementationsvariante auf dem Kleincomputer KC 85/2 oder /3. Eine Übertragung auf andere BASIC-Varianten ist leicht möglich, da auf die Besonderheiten des BASIC auf dem KC 85/2 bzw. /3 in jedem Kapitel hingewiesen wird.

Die Programmiersprache BASIC wurde schon 1960 in den USA für ein Mehrnutzersystem entworfen. BASIC steht für Beginners All-purpose Symbolic Instruction Code, "symbolischer Anweisungskode für Anfänger aller Anwendungsbereiche" und deutet den vorgesehenen Einsatzbereich an. BASIC sollte den "Einstieg" in die Programmierung auf einem niedrigen Niveau realisieren. So enthielt deshalb auch das Ur-BASIC nur 17 Sprachelemente.

Eine weitere Eigenschaft ist die relativ leichte Implementierbarkeit der Sprache BASIC auf beliebige Rechner, was die weite Verbreitung erklärt. Dies führte jedoch auch dazu, daß bei fast jedem Neueinsatz von BASIC Sprachelemente verändert und neue hinzugefügt wurden. Daher ist es notwendig, sich auf das BASIC seines konkreten Rechners zu spezialisieren.

1. Grundelemente der Sprache BASIC

1.1. Zeichensatz

Der Zeichensatz der Sprache BASIC besteht aus:

- den Großbuchstaben des lateinischen Alphabets A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S,T, U, V, W, X, Y, Z
- den Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- den Sonderzeichen

Zeichen	Verwendung in BASIC
.	Trennzeichen zwischen ganzem und gebrochenem Teil einer reellen Zahl
,	Verwendung innerhalb der Anweisung PRINT
;	Verwendung innerhalb der Anweisung PRINT
( )	einzig zugelassener Klammertyp
#	Nummernzeichen, Dateinummer
␣	Leerzeichen
=	Wertzuweisung, Vergleichsoperator
< >	Vergleichsoperatoren
+	Addition
-	Subtraktion, Vorzeichen
*	Multiplikation
/	Division
^	Potenzfunktion
$	Zeichenkettenkennung
" "	Begrenzer für Zeichenketten

1.2. Zahlendarstellung in BASIC

Zahlen werden mit einer vom BASIC-Interpreter festgelegten maximalen Ziffernanzahl dargestellt. Diese Anzahl kann zwischen 6 und 12 Dezimalstellen schwanken. Zur Trennung von ganzem und gebrochenem Anteil einer reellen Zahl wird der Punkt verwendet. Im allgemeinen wird eine vorangestellte Ø bei Zahlen <1 nicht mit ausgegeben, ebenso wie das positive Vorzeichen.
In der Exponentialdarstellung kennzeichnet der Großbuchstabe E mit nachfolgender vorzeichenbehafteter Zahl den Exponenten.

Beispiel:

mathematische Schreibweise	BASIC-Darstellung
765,432	765.432
Ø,345	.345
-33	-33
$1,2 \cdot 1Ø^{-8}$	1.2E-8

1.3. Konstanten

Konstanten sind Zahlen oder Zeichenketten, die bei Erstellung des Programms fest vorgegeben werden.
Konstanten können sein:

- vorzeichenbehaftete Zahlen z. B. -77, 1234E+7, Ø.456
- Zeichenketten z. B. "Mittelwert".

1.4. Variablen

1.4.1. Erklärung zum Begriff

In höheren Programmiersprachen erfolgt die Speicherplatzzuordnung über Variablen. Dies sind Symbole, welche der Anwender definiert. Der BASIC-Interpreter übernimmt es, dafür einen physischen Speicherplatz im Rechner zu reservieren. Es kann somit in BASIC oder auch in einer anderen problemorientierten Sprache programmiert werden, ohne den Speicheraufbau des konkreten Rechners zu kennen. Eine Variable (= symbolische Speicherplatzadresse) wird gebildet

- durch einen Großbuchstaben bzw. eine Folge von Großbuchstaben
- durch einen oder mehrere Buchstaben und Ziffern.

Jede Variable muß mit einem Buchstaben beginnen. Die Länge des Variablennamens wird durch die Eigenschaften des Interpreters bestimmt. Für den KC 85/2-3 gilt:

- Die Länge des Variablennamens ist beliebig, jedoch sind nur die ersten beiden Zeichen signifikant.

- Er darf keine für die Sprache BASIC reservierten Worte enthalten; z. B. LETTER enthält das reservierte Wort LET.

1.4.2. Variablentypen

BASIC unterscheidet je nach abzuspeicherndem Wert verschiedene Typen von Variablen. Der Variablenname schließt deshalb mit einer Typenkennzeichnung ab. Es gibt folgende Variablentypen:

- Gleitpunktvariable einfacher Genauigkeit. Dies ist der Standardvariablentyp. Aus diesem Grund kann eine Kennzeichnung entfallen;
  z. B. A! ist identisch mit A; MAX! entspricht MAX.
- Gleitpunktvariable doppelter Genauigkeit verwenden das Kennzeichen #; z. B. B# oder MIN#.
- Integervariable sind nur ganzzahlig und haben einen kleineren Wertebereich. Gekennzeichnet werden sie durch %; z. B. C% oder ZAHL%.
- Zeichenkettenvariable enthalten beliebige über Tastatur eingebbare Zeichenfolgen (Texte). Gekennzeichnet werden sie durch $ oder $; z. B. D$ oder KETTE$.

Beim KC 85/2-3 gilt:

- Es gibt nur Gleitpunktvariable einfacher Genauigkeit ohne Kennzeichen.
- Zeichenkettenvariable können maximal 255 Zeichen enthalten. Sie müssen mit $ gekennzeichnet werden.

Nur diese beiden Variablentypen sind verfügbar.

1.4.3. Wertzuweisung für Variable

Um einer Variablen einen Wert zuzuweisen, ist die Schreibweise der Ergibtanweisung notwendig, d. h., links vom Ergibtzeichen steht immer die Variable, welcher der Wert zugewiesen werden soll. Eine Ausnahme bildet hierbei die Eingabeanweisung INPUT.

A=1234 heißt z. B., schreibe in den Speicherplatz mit der symbolischen Adresse (= Name) A die Zahl 1234. MAX=A*Ø.7 bedeutet, in dem Speicher mit dem Variablennamen (Adresse) MAX soll das Produkt aus dem Inhalt der Variablen A multipliziert mit der Konstanten Ø.7 gespeichert werden.

1.5. Aufbau einer Programmzeile

BASIC fordert einen bestimmten Programmzeilenaufbau. Eine Zeile, welche in das Programm aufgenommen werden soll, muß immer mit einer Zeilennummer beginnen. Steht keine Zeilennummer am Anfang, so nimmt BASIC an, daß diese Zeile sofort nach Abschluß ausgeführt werden soll. Diese Arbeitsweise wird als Kommandomodus bezeichnet. Soll eine Zeile mehrere Anweisungen enthalten, so ist als Trennzeichen zwischen den einzelnen Anweisungen ein : zu schreiben.

znr anw1 : anw2 : anw3 :.....: anwn

Die Zeilennummern liegen im allgemeinen im Bereich von

1 <= znr <= 65535.

Als Zeilenabschluß werden die Tasten -↵-, -ET-, -ENTER- oder -RETURN- verwendet.

Eine Programmzeile kann 8Ø Zeichen oder mehr enthalten. Die Programmzeilenlänge entspricht nur der auf dem Bildschirm darstellbaren Zeilenlänge. Oft erstreckt sich eine Programmzeile über mehrere Bildschirmzeilen.

Beim KC 85/2-3 wird als Zeilenabschluß die Taste -↵- verwendet. Die Programmzeilenlänge beträgt maximal 8Ø Zeichen. Dies entspricht 2 Bildschirmzeilen.

2. Anweisungen

2.1. Kommandos

Wird eine Programmzeile ohne Zeilennummer eingegeben, so wird diese Zeile als Kommando verstanden und unmittelbar nach dem Zeilenabschluß ausgeführt. Kommandos dienen zur Steuerung des BASIC-Interpreters.

RUN

RUN startet ein BASIC-Programm mit der niedrigsten Zeile. Durch Angabe einer Zeilennummer nach RUN kann ein Programm auch von einer beliebigen Zeile gestartet werden;
z. B. >RUN oder >RUN 5Ø

LIST

LIST läßt das Programm auf dem Bildschirm auflisten. Wird eine Zeilennummer angegeben, so beginnt das Auflisten bei dieser;
z. B. >LIST oder >LIST 1ØØ

EDIT zeilennummer

EDIT veranlaßt BASIC, die angegebene Zeile zur Korrektur auf dem Bildschirm darzustellen. Die einzelnen Korrekturzeichen sind interpreterspezifisch. Ein Überschreiben ist meist durch Stellen des Schreibzeigers (Kursor) auf die entsprechende Stelle und Eingabe des Zeichens möglich. Nach Zeilenabschluß wird die korrigierte Zeile in den Speicher eingetragen.
Die Anwendung von EDIT beim KC 85/2-3 wird im KC-Übungsteil dieses Lehrmaterials dargestellt.

AUTO zeilennummer, schrittweite

AUTO führt zur Programmzeileneingabe, wobei mit der Programmzeile zeilennummer begonnen wird und der Abstand der Zeilennummern durch die Schrittweite festgelegt wird. Es gilt: Wenn keine Zeilennummer und Schrittweite angegeben werden, so beginnt BASIC mit der Zeile 1Ø zu numerieren in einer Schrittweite von 1Ø.
z. B.

```
>AUTO 1Ø
1Ø ...
2Ø ...
3Ø ...
```

CLS

CLS löscht den Bildschirm und setzt den Schreibzeiger auf den Bildschirmanfang.

CONT

CONT setzt ein unterbrochenes Programm mit der nachfolgenden Anweisung fort, wenn keine Veränderung an ihm vorgenommen wurde.

CSAVE "name"

CSAVE speichert ein BASIC-Programm auf einen externen Speicher, meist Kassette, unter dem Namen name. Der Programmname kann aus 8 Zeichen bestehen, darf keine Sonderzeichen enthalten und muß mit einem Buchstaben beginnen.

CLEAR

CLEAR löscht die Werte im Variablenspeicher.

NEW

NEW löscht das im Speicher befindliche Programm und den Variablenspeicher.

CLOAD "name"

CLOAD ladet das BASIC-Programm mit Namen name vom externen Speicher. Die Namenbildung entspricht CSAVE. Der Programmspeicher wird nicht automatisch gelöscht. Das zu ladende Programm wird hinter ein eventuell im Speicher befindliches geladen. Es ist deshalb sinnvoll, den Speicher vor CLOAD mittels NEW zu löschen.

RENUMBER

RENUMBER führt ein Neunumerieren des Gesamtprogramms im Speicher durch.

TRON

TRON schaltet die Anzeige der abgearbeiteten Zeilennummer auf dem Bildschirm ein. Die Anzeige der Zeilennummer erfolgt in eckigen Klammern; z. B.

```
>TRON
>RUN
<10><20><25><26><30><100>.....
```

TROFF

TROFF schaltet die Anzeige der Zeilennummer im Programmlauf ab.

2.2. Befehle

2.2.1. Erklärung zum Begriff

Befehle sind Anweisungen, welche im Programmlauf ausgeführt werden sollen. Um einen Befehl in BASIC in den Programmspeicher aufzunehmen, muß am Anfang einer solchen Befehlszeile eine Zeilennummer geschrieben werden. BASIC arbeitet Programme immer nach steigenden Zeilennummern ab. Befehle werden erst nach Starten des Programmlaufs mittels der Anweisung RUN ausgeführt. Dies schließt nicht aus, daß eine Anzahl von Befehlen auch im Kommandomodus ausführbar ist. Ein Beispiel ist der Tischrechnermodus für die angeführten Ausgabebefehle.

2.2.2. Ausgabeanweisung

PRINT

PRINT ist der Befehl für die Ausgabe auf dem Bildschirm; z. B.

```
>PRINT 2+5
     7
```

Da keine Zeilennummer angegeben wurde, antwortet der Rechner sofort mit dem Ergebnis der Addition. Danach ist die Anweisung vergessen, der Rechner wartet auf ein neues Kommando.

Im folgenden Beispiel ist PRINT Bestandteil eines Programms.

```
1Ø A=2+5
2Ø B=1Ø
3Ø C=A*B
4Ø PRINT C
5Ø END
>RUN
      7Ø
```

Das Beispielprogramm wurde mit dem Kommando RUN gestartet. BASIC führt in der Zeile 4Ø die Ausgabe des Wertes der Variablen C aus. Es erscheint auf dem Bildschirm die Zahl 7Ø.

2.2.3. Eingabeanweisung

znr INPUT variablenname

Diese Form der Wertzuweisung für eine Variable weist der Variablen den Wert zu, welcher von der Tastatur eingegeben wurde. Als Eingabeanforderung setzt BASIC ein ? bei der Abarbeitung von INPUT auf den Bildschirm.

```
1Ø INPUT A
2Ø INPUT B
3Ø C=A*B
4Ø PRINT C
5Ø END
>RUN
? 7
? 1Ø
 7Ø
```

Nach dem Programmstart mit RUN fordert BASIC in Zeile 1Ø die Eingabe eines Wertes für die Variable A an. Als Reaktion des Bedieners auf das ? wurde die Zahl 7 eingegeben. Ähnlich wurde mit dem 2. ? verfahren. BASIC gibt das Produkt der beiden Variablen aus.

2.2.4. Programmverzweigung

znr GOTO zeilennummer

Die Anweisung GOTO veranlaßt BASIC, den linearen Programmlauf zu verlassen und die Abarbeitung an der mit zeilennummer beschriebenen Stelle fortzusetzen. Diese Verzweigung wird ohne Test auf eine Bedingung stets ausgeführt.

znr IF bedingung THEN anweisung1 ELSE anweisung2

Diese Anweisung testet eine Bedingung und führt entsprechend dem Testergebnis die Anweisung1 oder die Anweisung2 aus. Es gilt: Ist die Bedingung erfüllt, so wird mit der Anweisung nach THEN fortgesetzt. Wenn nicht, dann mit der nach ELSE folgenden. Soll bei nicht erfüllter Bedingung mit der nächsten Programmzeile fortgesetzt werden, so kann ELSE... entfallen.
Für den KC 85/2-3 muß vor ELSE ein Doppelpunkt geschrieben werden.
znr IF bedingung THEN anweisung1 :ELSE anweisung2
Beispiel:

```
1Ø INPUT A
2Ø INPUT B
3Ø IF A=B THEN GOTO 5Ø :ELSE GOTO 4Ø
4Ø C=A*B : GOTO 6Ø
5Ø C=A+B
6Ø PRINT C
7Ø END
```

```
>RUN          >RUN
? 7           ? 5
? 1Ø.          ? 5
 7Ø             1Ø
```

Steht hinter THEN bzw. :ELSE nur eine Zahl, so nimmt BASIC diese als die Zeilennummer, bei der das Programm fortzusetzen ist. GOTO hinter THEN und :ELSE kann also entfallen. Im Beispiel kann auch bei gleicher Funktion des Programms :ELSE entfallen.

2.2.5. Zählschleifen

znr FOR variable=anfangswert TO endwert STEP schrittweite

Für häufig auftretende Zählaufgaben gibt es im BASIC die Zählschleife. Sie beginnt mit der Definition des Anfangswertes der Laufvariablen, dem Festlegen des Endwertes des Zählvorganges und der Schrittweite. Abgeschlossen wird eine solche Laufanweisung mit

znr NEXT variable

Beispiel:

Es soll das Quadrat der Zahlen von 1 bis 1Ø ausgegeben werden sowie die Zahl selbst.
Zwei Lösungen sind im BASIC realisierbar:

I.

```
1Ø I=1
2Ø A=I*I
3Ø PRINT I,A
4Ø I=I+1
5Ø IF I<=1Ø THEN 2Ø
6Ø END
```

II.

```
1Ø FOR I=1 TO 1Ø STEP 1
2Ø A=I*I
3Ø PRINT I,A
4Ø NEXT I
5Ø END
```

```
 RUN
1     1
2     4
3     9
:     :
1Ø    1ØØ
```

Die Anweisung NEXT variable führt intern folgendes aus:

1. variable = variable + schrittweite
2. IF variable <= endwert THEN GOTO 1.anweisung nach FOR...

Ein absteigendes Zählen kann erreicht werden durch eine negative Schrittweite und Anfangswert > Endwert. Die Schrittweite wird vorzeichenbehaftet addiert.

Regeln für das Aufstellen von Zählschleifen:

- In eine Zählschleife darf nie hineingesprungen werden.
- Eine Zählschleife darf nur über das zugehörige NEXT verlassen werden.
- Zählschleifen dürfen auch verschachtelt werden, dabei ist auf eine strenge Symmetrie zwischen FOR variable und NEXT variable zu achten.

2.2.6. Konstantenfelder

znr DATA wert1, wert2, wert3, ..., wertn

Die Anweisung DATA mit nachfolgender Datenliste stellt Konstanten eines beliebigen Typs im Speicher bereit.

znr READ variable1, variable2, ...

READ weist Variablen einen Wert aus der Datenliste zu. Diese Zuweisung geschieht in der gleichen Reihenfolge wie sie in der Datenliste steht. Die Datenliste wird über einen Zeiger verwaltet, welcher bei jedem Lesen auf den nachfolgenden Wert weitergestellt wird. Alle im Programm vorhandenen DATA-Zeilen stehen beim Lesen hintereinander, unabhängig von ihrer Zeilennummer. Das Einstellen des Datenzeigers auf eine bestimmte Programmzeile erfolgt mit der Anweisung

znr RESTORE zeilennummer

Das Auslesen beginnt ab der vorgegebenen Zeile.

Beispiel für das Auslesen aus einem Konstantenfeld:

```
10 RESTORE 100
20 FOR I=1 TO 5
30 READ X
40 A=A+X
50 NEXT I
60 PRINT A/5
70 END
100 DATA 1,2,3,4,5
```

Beispiel: Runden eines Eingabewertes auf den nächsten Wert der Reihe E 12.

```
1Ø REM Normierung eines Wertes auf die E-Reihe E 24
2Ø DEF FNLG(X)=LN(X)/LN(1Ø)
3Ø INPUT "Wert:";X
4Ø XØ = LN(X)/LN(1Ø) : E = INT(XØ) : X1 = XØ-E : X1 = 1Ø^X1
5Ø RESTORE 2ØØ
6Ø FOR I=1 TO 24
7Ø READ Y
8Ø IF Y=>X1 THEN I=25
9Ø NEXT I
1ØØ X=Y*1Ø^E
11Ø PRINT "genormter Wert:";X
12Ø END
2ØØ DATA 1,1.1,1.2,1.3,1.5,1.6,1.8,2,2.2,2.4,2.7,3,3.3,3.6
21Ø DATA 3.9,4.3,4.7,5.1,5.6,6.2,6.8,7.5,8.2,9.1
```

2.2.7. Unterprogramme und Anwenderfunktionen

Unterprogrammaufrufe sind Programmverzweigungen mit der Absicht, später an der Verzweigungsstelle die Abarbeitung fortzusetzen. Unterprogramme können von beliebigen Stellen im Programm aufgerufen werden. BASIC organisiert die Rückkehr an die aufrufende Stelle. Dies bedingt einige Regeln:

- Ein Unterprogramm darf nur über die Anweisung GOSUB erreicht werden.
- Unterprogramme dürfen selbst wieder Unterprogramme aufrufen.
- Meist ist es nicht zulässig, daß ein Unterprogramm sich selbst aufruft (rekursive Programmierung).
- Im Unterprogramm gelten die gleichen Variablen wie im übergeordneten, d. h.,bei der Übergabe von Werten ins bzw. vom Unterprogramm können die gleichen Variablen wie im anderen Programmteil benutzt werden.

znr GOSUB zeilennummer

GOSUB ruft ein Unterprogramm auf der angegebenen Zeile auf. Die Stelle im Programm, von welcher das Unterprogramm aufgerufen wurde, hat sich der BASIC-Interpreter gemerkt und setzt die Abarbeitung mit dem Unterprogrammabschlußbefehl dort fort.

znr RETURN

RETURN schließt ein Unterprogramm ab. Unterprogramme müssen immer mit RETURN abgeschlossen werden.

Beispiel: Bildung der Fakultät N

```
1Ø REM FAKULTAET N!
2Ø INPUT "N:";N
3Ø GOSUB 1ØØ
4Ø PRINT "N!:",M
:
1ØØ REM UP FAKULTAETSBILDUNG M=N!
11Ø M=1
12Ø FOR I=2 TO N
13Ø M=M*I
14Ø NEXT I
15Ø RETURN
•
•
```

Eine spezielle Unterprogrammform sind die Anwenderfunktionen. Sie werden vereinbart und dürfen nur eine Programmzeile umfassen. Mit ihrer Hilfe läßt sich bei Bedarf der Wortschatz von BASIC erweitern.

znr DEFFN variablenname (variable) = ausdruck

DEFFN vereinbart eine Nutzerfunktion. Diese kann unter dem Variablennamen aufgerufen werden, wobei die Variable in Klammern den Platz für die konkrete Eingangsvariable freihält.

znr FN variablenname (variable)

Aufruf der Anwenderfunktion, die vorher vereinbart werden muß.

Beispiel: Bildung der Arcussinus im Bogenmaß

```
1Ø REM FUNKTIONSVEREINBARUNG
2Ø DEFFNASIN(X)=ATN(X/SQR(1-X*X))
:
1ØØ REM FUNKTIONSAUFRUF
11Ø Y=FNASIN(X)
:
```

2.2.8. FELDER

Ein Feld enthält eine Menge gleicher Variabler mit einem gemeinsamen Namen. BASIC ermöglicht das Aufstellen ein- und zweidimensionaler Felder, die den Matrizen und Vektoren in der Mathematik entsprechen.

- Zweidimensionale Felder (Matrizen):

$$A = \begin{pmatrix} a & a & a & \ldots\ldots & a \\ a & a & a & \ldots\ldots & a \\ a & a & a & \ldots\ldots & a \\ a & a & a & \ldots\ldots & a \\ a & a & a & \ldots\ldots & a \end{pmatrix}$$

Dieses zweidimensionale Zahlenfeld besitzt (x+1)-Zeilen und (y+1)-Spalten. Die Koordinatenangaben werden auch als Zeilen- bzw. Spaltenindex bezeichnet. BASIC erreicht ein konkretes Matrixelement durch Angabe der Indizes. Es ist zu beachten, daß BASIC mit der Koordinate Ø beginnt.
X(3,7) meint den Wert, der im Feld mit dem Namen X steht an den Koordinatenpunkten 3 und 7.

- Eindimensionale Felder (Vektoren):

A = (a a a .....a)

Ein Vektor entsteht durch Angabe nur einer Koordinate.

Beispiel: Aus einem Vektor V soll der Variablen K der Wert an der Koordinate, welche in der Variablen I steht, zugewiesen werden.

```
2ØØ K=V(I)
```

mit dem Vektoraufbau V = 1 2 4 5.7 3 6
und I=3 würde in K stehen K=5.7

Es ist möglich, die Angabe der Koordinate implizit über eine Variable zu realisieren.

znr DIM variablennamen (anzahl1, anzahl2)

DIM vereinbart den Speicherplatz für ein Feld mit einer Anzahl von Zeilen und Spalten. Die erste Feldkoordinate ist dabei immer Ø; z. B.

```
1Ø INPUT X,Y
2Ø DIM V(3Ø),M(X,Y)
```

Diese Angabe weist dem Vektor V 31 Elemente und der Matrix M X-Zeilen, Y-Spalten zu. Bei der Dimensionierung sind folgende Regeln zu beachten:

- Die Feldvereinbarung muß vor der 1. Feldbenutzung stehen.
- Sie darf für das Feld nur einmal im Programm erfolgen.

Hinweis: Erfolgt ein Zugriff auf ein Feld, ohne dieses vorher in seiner Größe zu vereinbaren, so nimmt BASIC ein Feld der Dimension 1Ø,1Ø an.

2.2.9. Tischrechnermodus

Einige Befehle können auch als Kommandos im sogenannten Tischrechnermodus ausgeführt werden. Es ist jedoch nur eine Zeile mit Anweisungen möglich. Die Befehle im Tischrechnermodus werden sofort nach Zeilenabschluß ausgeführt und sind danach vergessen.
Beispiel:

```
PRINT 4+5
  9
```

3. Spezielle BASIC-Anweisungen

3.1. Farbe und Grafik

BASIC-Interpreter auf Kleincomputern wie dem KC 85/2-3 verfügen meist über erweiterte Möglichkeiten zur grafischen Darstellung und zur Veränderung der Farbdarstellung auf dem Bildschirm. Die aufgeführten Anweisungen gelten für die genannten Rechner, sind jedoch auch auf andere Systeme sinngemäß übertragbar.

znr COLOR vordergrundfarbe, hintergrundfarbe

COLOR stellt die Farbkombination auf dem Bildschirm ein. Den Kode für die Farben enthält die KC-Befehlsübersicht.

znr PAPER hintergrundfarbe

PAPER stellt die Hintergrundfarbe, die Farbe des Papiers ein, auf dem geschrieben werden soll.

znr INK vordergrundfarbe

INK stellt die Vordergrundfarbe, die Farbe der Tinte ein, mit der geschrieben werden soll.

znr PSET x-koordinate, y-koordinate, farbe

PSET setzt einen Grafikpunkt. Für die Koordinaten gilt:

Ø <= x <= 319 : Ø <= y <= 255

Die Farbe des Punktes kann unabhängig von der eingestellten Bildschirmfarbe gewählt werden. Die Hintergrundfarbe bleibt erhalten.
Beispiel: Darstellen der Funktion y=sin(x)

```
1Ø CLS
2Ø FOR X=Ø TO 319
3Ø Y=128+5Ø*SIN(X/159.5*PI)
4Ø Y=INT(Y+Ø.5)
5Ø PSET X,Y
7Ø NEXT X
8Ø END
```

znr PRESET x-koordinate, y-koordinate

PRESET löscht den Punkt mit den angegebenen Koordinaten.

znr LINE x anf., y anf., x ende, y ende, farbe

LINE zeichnet beim KC 85/3 und beim KC 85/2 mit BASIC-Modul eine Linie vom Anfangspunkt der x- und y-Koordinate zu den Endpunkten der angegebenen x- und y-Koordinaten. Die Farbe der Linie kann wie bei PSET gewählt werden.
Beispiel:

```
...
1ØØ line Ø,Ø,319,255
...
```

znr CIRCLE x-koordinate, y-koordinate, radius, farbe

CIRCLE zeichnet einen Kreis um die Mittelpunktkoordinaten mit einem vorgegebenen Radius.
Beispiel:

```
1Ø CLS
2Ø FOR I=1 TO 15
3Ø CIRCLE 154,128,2*I,I
4Ø NEXT I
5Ø END
```

3.2. Bildschirmfenster

znr WINDOW zeilenanfang, zeilenende, spaltenanfang, spaltenende

Mit der Anweisung WINDOW kann der Bildschirm in Bereiche, sogenannte Fenster, unterteilt werden. Ausgaben auf den Bildschirm sind nur im gerade aktuellen Fenster möglich. Damit ist es möglich, daß nur ein Teil des Bildschirms sich infolge von Ausgaben verändert, z. B. nur eine Spalte einer Tabelle, während der gesamte andere Teil unverändert bleibt. Der Löschbefehl CLS ist ebenfalls nur im aktuellen Fenster wirksam.
Beispiel: Erzeugen eines farbigen Fensters

```
...
3Ø WINDOW 2Ø,25,1Ø,15
```

```
40 PAPER 2
50 CLS
...
```

```
>WINDOW 0,31,0,39
```

stellt das größtmögliche Fenster ein.

3.3. Zeichenketten und Zeichenkettenfunktionen

Zeichenketten sind Folgen von beliebigen Zeichen, Groß- oder Kleinbuchstaben, Ziffern und Symbolen, die in unveränderter Form im Speicher abgelegt werden. Zeichenketten können komplette Texte enthalten. Sie werden in der Form ihrer Schreibweise im Speicher abgelegt. Dies begründet den hohen Platzbedarf im Speicher. Für die meisten BASIC-Varianten gilt:

- Die maximale Länge einer Zeichenkette ist vorgegeben, oft 255 Zeichen.
- Eine Veränderung ist mit Hilfe der Anweisung DIM möglich.

Beispiel:

```
...
30 X$="Zeichenkette 1"
40 Y$="Zeichenkette 2"
...
```

Bestimmte Manipulationen sind mit Zeichenketten möglich. Ketten können getrennt und zusammengefügt werden. Sie können, sofern sie numerisch sind, in die Realzahldarstellung überführt werden und umgekehrt.

znr LEFT$ (variable $, position)

LEFT$ liefert eine Anzahl von Zeichen bis zur Position vom Anfang der Kette.

```
50 A$=LEFT$(X$,7)
```

liefert "Zeichen" in der Variablen A$

znr RIGHT$ (variable$, anzahl)

RIGHT$ liefert eine Anzahl von Zeichen ab der Position bis zum Ende der Kette.

60 B$=RIGHT$(X$,8)
liefert "kette 1" in der Variablen B$.

znr MID$ (variable$, position, anzahl)

MID$ liefert Zeichen aus einer Kette ab einer Position.

70 C$=MID$(X$,8,5)
liefert "kette" in der Variablen C$.

znr INSTR$ (variable1$, variable2$)

INSTR$ sucht die Kette 1 in der Kette 2 und meldet die Position des ersten Auftretens.

80 K=INSTR$("e",X$)
K erhält den Wert 2.

znr STRING$ (anzahl, variable$)

Die Funktion STRING$ vervielfacht die angegebene Kette; z. B.
90 D$=(39,"*")
erzeugt 39mal das Zeichen "*" in D$.

3.4. Zufallsgenerator

Für mathematisch-statistische Untersuchungen und auch für Spiele werden oft Zahlen oder Zahlenfolgen benötigt, deren Wert nicht voraussagbar ist (z. B. für ein Würfelspiel). Diese Zufallszahlen kann BASIC erzeugen. Es ist möglich, wie beim Würfeln festzulegen, ob alle Würfel neu zu mischen sind oder ob vom derzeitigen Stand ausgegangen werden soll.

znr RND(x)

Die Funktion RND(x) erzeugt eine Zufallszahl im Bereich von Ø <= RND(x) <1. Das Argument x kann die Bildung der Zufallszahl beeinflussen.

znr RANDOMIZE

Der Aufruf von RANDOMIZE legt fest, daß mit der Bildung der Zufallszahl jedesmal erst ein neuer Startwert gebildet werden soll. Dies ähnelt dem neuen Mischen aller Würfel.

Für den KC 85/2-3 gilt:

- RANDOMIZE kann nicht aufgerufen werden.
- Der Startwert wird durch das Argument von RND festgelegt.
  x > Ø, der Startwert wird neu gebildet, entspricht RANDOMIZE;
  x = Ø, die letzte Zufallszahl wird erneut ausgegeben;
  x < Ø, der Zufallsgenerator wird neu initialisiert.

Beispiel: Test der Verteilung von 1ØØØØ Zufallszahlen

```
1Ø DIM X(1Ø)
2Ø REM Test des Zufallszahlengenerators
3Ø RANDOMIZE
4Ø FOR I=1 TO 1ØØØØ
5Ø A=INT(1Ø*RND(1)+.5)
6Ø X(A)=X(A)+1
7Ø NEXT I
8Ø FOR I=Ø TO 1Ø
9Ø PRINT X(I)/1ØØ:
1ØØ NEXT
11Ø END
```

Wird das Programm mit RUN gestartet, meldet es sich nach einiger Zeit mit der prozentualen Verteilung der Zufallszahlen.

```
>RUN
 1Ø.28  1Ø.Ø3  9.72  1Ø.33  9.91  9.51  9.84  1Ø.37  1Ø.16  5.25
```

3.5. Sprungverteiler - ON-Satz

Um den Aufwand für bedingte Programmverzweigungen gering zu halten, wurde der Sprungverteiler in BASIC aufgenommen. Das folgende Beispiel zeigt die Funktion des Sprungverteilers.

```
...                              ...
...                              ...
10Ø IF A=1 THEN 2ØØ              10Ø ON A GOTO 2ØØ,3ØØ,4ØØ
11Ø IF A=2 THEN 3ØØ              ...
12Ø IF A=4 THEN 4ØØ
...
```

ohne Sprungverteiler — mit Sprungverteiler

Der Sprungverteiler testet also den Wert der Variablen nach ON und verzweigt entsprechend. Die Position der Absprungadresse entspricht dem Wert der Variablen. Der Wert Ø für die Variable ist nicht zugelassen.

znr ON variable GOTO zeile1, zeile2, zeile3, ...

Der Satz ON .. GOTO realisiert einen Sprungverteiler zu absoluten Adressen.

znr ON variable GOSUB zeile1, zeile2, zeile3, ...

Der Satz ON ... GOSUB realisiert einen Unterprogrammsprungverteiler. Es wird das Unterprogramm entsprechend dem Wert der Variablen aufgerufen.

znr ONERRORGOTO zeile

Der Satz ONERRORGOTO läßt eine Beseitigung bestimmter Fehler zu, die im Interpreter festgelegt worden sind. Auf der angewiesenen Zeile muß ein Programm stehen, das den Fehler beseitigt. Es können nur Fehler aufgefangen werden, die nicht zu inneren Konflikten im Interpreter führen. Fehlerhafte Felddimensionen oder Verschachtelungsfehler bei Laufanweisungen können hiermit also nicht beseitigt werden. Beachtet werden muß weiterhin, daß ONERRORGOTO bis zu seinem Aufheben, z. B. durch Angabe einer neuen Zeile, gemerkt wird. Dies kann zu sehr komplizierten Fehlern

führen.
ONERRORGOTO ist beim KC 85/2-3 nicht vorhanden.

3.6. Speicherzugriff und Maschinenunterprogramme

Für die Anwendung der folgenden Befehle ist eine genaue Kenntnis des technischen Aufbaus des Rechners notwendig. Der Schutz, den BASIC dem Nutzer bei Fehlbedienung gibt, ist hier wirkungslos.

3.6.1. Speicherzugriff

znr POKE adresse, byte

Der Befehl POKE schreibt einen dezimalen Zahlenwert auf den mit der Dezimaladresse angegebenen Speicherplatz im Systemspeicher des Rechners.

znr PEEK (adresse)

Der Befehl PEEK liest ein Byte vom Speicherplatz mit dezimal angegebener Adresse.

znr DOKE adresse, byte1byte2

Der Befehl DOKE ähnelt POKE, jedoch wird hier eine zwei Byte lange Zahl auf den dezimal angegebenen Speicherplatz und den folgenden geschrieben.

znr DEEK (adresse)

Der Befehl DEEK entspricht PEEK, angewandt auf eine zwei Byte lange Zahl.

Beim KC 85/2-3 gibt es noch die Befehle:

znr VPOKE adresse, byte

VPOKE schreibt wie POKE ein Byte, jedoch in den Bildwiederholspeicher. Dieser beginnt beim Ansprechen mit VPOKE auf der Adresse Ø und ist 16 KByte lang. Alle Parameter in VPOKE sind dezimal einzugeben.

znr VPEEK (adresse)

VPEEK entspricht dem Befehl PEEK mit der Wirkung auf den Bildwiederholspeicher. Die Adresse beginnt bei Ø.

3.6.2. Maschinenunterprogrammaufrufe

Die Übermittlung von Werten vom - ins Unterprogramm muß der jeweiligen Programmbeschreibung des Rechners entnommen werden.

znr CALL adresse

Der Befehl CALL ruft ein Maschinenunterprogramm auf, welches mit dem Maschinenbefehl "RET" bzw. dem Hexadezimalkode ØC9h abgeschlossen sein muß. Die Adresse des Unterprogramms kann entweder dezimal angegeben werden oder durch Voranstellen eines "*" hexadezimal.
Beispiel:

```
...
345 CALL 256      oder      345 CALL *1ØØ
```

znr USR (x)

Die Funktion USR entspricht weitestgehend dem Befehl CALL. Es ist jedoch möglich, Parameter direkt ins Unterprogramm zu übermitteln. Die Startadresse des Unterprogramms ist auf vereinbarten Speicherplätzen abzulegen.

3.7. Kursorpositionierung bei Aus- und Eingabe

Die Kursorpositionierung ist immer nur im aktuellen Fenster möglich.

3.7.1. Ausgabegestaltung

Um das Druckbild für die Ausgabe gestalten zu können, gibt es eine Reihe von speziellen Funktionen, welche nur innerhalb des Befehls PRINT wirken.

Das Semikolon ";" verhindert den Abschluß einer Ausgabezeile. Die nächste Ausgabe erfolgt somit auf der gleichen Zeile, auf der folgenden freien Schreibposition.

Das Komma "," bewirkt einen Tabulatorsprung auf eine Zeilenposition, die ein Vielfaches von 15 ist. Es steht somit ein einfacher Tabulator mit den Adressen Ø,15,3Ø,... innerhalb einer Zeile zur Verfügung.

TAB (spalte)

Die Funktion TAB bewirkt einen Tabulatorsprung in die angegebene Spalte in der Zeile.

AT (zeile, spalte)

Die Funktion AT bewirkt eine Positionierung des Kursors in die angegebene Zeile auf die Spalte. Es kann mit AT auf dem gesamten Bildschirm gearbeitet werden, jedoch nur im aktuellen Fenster.

Die Funktion CUR (zeile, spalte) entspricht der Funktion AT.

Der Befehl PRINT ... ohne Abschluß mit Semikolon oder Komma schließt immer eine Ausgabezeile ab. Die weiteren Ausgaben erfolgen in der nächsten Bildschirmzeile. Steht PRINT allein, so schließt es entweder eine Zeile ab oder, ist die Zeile schon abgeschlossen, erzeugt er eine Leerzeile.

Beispiele für die Wirkung der Anweisungen innerhalb von PRINT.

```
1Ø FOR I=1 TO 1Ø
2Ø PRINT I
3Ø NEXT I
4Ø END
```

Ausgabe der 1Ø Zahlen untereinander.

```
1Ø FOR I=1 TO 1Ø
2Ø PRINT I;
3Ø NEXT I
4Ø END
```
Ausgabe ohne Zwischenraum

```
1Ø FOR I=1 TO 1Ø
2Ø PRINT I,
3Ø NEXT I
4Ø END
```
mit einem Tabulator auf 15 Position

```
1Ø FOR I=1 TO 1Ø
2Ø PRINT TAB(I*5);I;
3Ø NEXT I
4Ø REM ZEILENABSCHLUSS
5Ø PRINT
6Ø END
```
Ausgabe mit Tabulator im 5 Abstand

```
1Ø FOR I=1 TO 1Ø
2Ø PRINT AT(1Ø,I*5);I;
3Ø NEXT I
4Ø REM ZEILENABSCHLUSS
5Ø PRINT
6Ø END
```
auf Zeile 1Ø mit Tabulator im 5 Abstand

3.7.2. Eingabeanforderung

LOCATE zeile, spalte

Der Befehl LOCATE positioniert den Kursor innerhalb des aktuellen Fensters auf die angegebene Zeile in die Spalte.

INPUT "text"; variable

Bei Eingaben kann der einzugebende Wert mit einem Text auf dem Bildschirm angefordert werden.

4. Standardfunktionen

Im BASIC sind für arithmetische Ausdrücke Standardfunktionen definiert. Diese können durch ihren Namen aufgerufen werden. Das Argument muß in runde Klammern geschrieben werden. Weiterhin ist zu beachten, daß für die trigonometrischen Funktionen die Berechnung im Bogenmaß durchgeführt wird.

SIN(x)	bildet den Sinus von x
COS(x)	bildet den Cosinus von x
TAN(x)	bildet den Tangens von x
ATN(x)	bildet den Arcustangens von x

EXP(x)	entspricht e^x
LOG(x) oder LN(x)	entspricht ln(x)
LOG1Ø(x)	entspricht lg(x)
Y^X	entspricht dem Potenzoperator y^x
SQR(x)	bildet die Quadratwurzel von x (x > Ø)
ABS(x)	bildet den Betrag von x
INT(x)	bildet die größte ganze Zahl von x, welche kleiner oder gleich x ist
FIX(x)	bildet den ganzzahligen Anteil von x
SGN(x)	liefert das Vorzeichen von x SGN(X) =-1 für X < Ø SGN(X) = Ø für X = Ø SGN(X) = 1 für X > Ø
RND(x)	bildet eine Zufallszahl im Bereich von Ø <= x <= 1. Die Bedeutung des Argumentes x ist interpreterspezifisch
RANDOMIZE	startet den Zufallsgenerator bei jedem Aufruf neu
PI	liefert die Konstante pi = 3,14..... mit der Genauigkeit des Interpreters
x AND y	bildet das logische UND zwischen x und y
x OR y	bildet das logische ODER zwischen x oder y
x XOR y	bildet das EXKLUSIVODER zwischen x und y
x IMP y	bildet die IMPLIKATION von x,y

Beispiele:

1. Es soll eine Tabelle der Werte des Sinus im Intervall von Ø < x < 2 * pi in Schritten von Ø.1 auf dem Bildschirm erstellt werden.

```
1Ø FOR I=Ø TO 2*PI STEP .1
2Ø PRINT I,SIN(I)
3Ø NEXT I
4Ø END
```

2. Die Tabelle soll in Grad ausgegeben werden.

```
1Ø FOR I=Ø TO 2*PI STEP .1
2Ø PRINT 18Ø*I/PI,SIN(I)
3Ø NEXT I
4Ø END
```

3. Der natürliche Logarithmus soll im Intervall von 1 < x < 2Ø in Schritten von 1 auf dem Bildschirm tabelliert werden.

```
1Ø FOR I=1 TO 2Ø STEP 1
2Ø PRINT I;LN(I),
3Ø NEXT I
4Ø END
```

4. Es sollen die Zahlen von Ø bis 1Ø quadriert werden.

```
1Ø FOR I=Ø TO 1Ø
2Ø PRINT I,I^2
3Ø NEXT I
4Ø END
```

Das Programm führt im allgemeinen zu einer Fehlermeldung, da die Bildung der Potenzfunktion intern über den Logarithmus realisiert wird. Es empfiehlt sich daher, das Quadrieren als Mehrfachmultiplikation zu schreiben oder vor Aufruf der Potenzfunktion das Argument zu testen.

```
1Ø FOR I=Ø TO 1Ø
2Ø PRINT I,I*I
3Ø NEXT I
4Ø END
```

5. Es soll eine einzugebende Zahl gerundet und auf dem Bildschirm wieder ausgegeben werden.

```
1Ø INPUT X
2Ø X=INT(X+.5)
3Ø PRINT X
4Ø GOTO 1Ø
```

5. Übungen zum Umgang mit dem Kleincomputer KC 85/3 und Programmbeispiele

5.1. Übungen zur Programmeingabe und zum Programmtest

Eine programmierte Einführung in die Arbeit mit dem Kleincomputer KC 85/3 soll den nicht ständig mit dem Rechner arbeitenden Leser den Einstieg in die Nutzung erleichtern. Die Übung ist direkt am Rechner zu realisieren und führt in die Handhabung grund-

legender Kommandos zur Programmeingabe und zum Programmtest ein.

Realisieren Sie folgende Arbeitsgänge:

1. Einschalten des Rechners
   - Fernsehgerät einschalten
   - Taste -POWER- am Grundgerät betätigen.
     (Auf dem Bildschirm erscheint das Grundmenü des Betriebssystems)

2. Aufrufen des BASIC-Interpreters
   - Eingabe der Tastenfolge BASIC und Taste -ENTER-( )
     (Rechner fragt nach dem Speicherende für BASIC)
   - Quittierung mit Taste -ENTER-
     (Der gesamte Speicherbereich wird für BASIC genutzt.
     BASIC meldet sich mit: OK
     >)

3. Programmeingabe
   - Geben Sie ein: 1Ø FR I=1 TO
     Jetzt merken Sie, daß ein Schreibfehler vorliegt (FR statt FOR). Sie müssen ein Zeichen einfügen!
     . Kursor mit Taste -←- auf Zeichen nach der Einfügestelle setzen (also auf -R-)
     . Taste -INS- (insert=einfügen)
     . Buchstabentaste -O-
     . Kursor mit Taste -→- auf alte Schreibposition stellen.
     Es steht nun auf dem Bildschirm: 1Ø FOR I=1 TO
   - Geben Sie weiter ein: ...5 STAP
     Sie merken jetzt, daß Sie einen falschen Buchstaben geschrieben haben (STAP statt STEP). Sie müssen ein Zeichen löschen und ein anderes einfügen!
     Zeichen löschen:
     . Kursor auf falsches Zeichen setzen
     . Taste -DEL- (delete=löschen; die Zeile wird um ein Zeichen verkürzt, Restzeile rückt nach)
     Zeichen einfügen:
     . Taste -INS-
     . Zeichentaste -E-
     . Taste -→- bis Kursor hinter STEP steht.

Auf dem Bildschirm steht nun folgende Zeile:

```
10 FOR I=1 TO 5 STEP
```

- Geben Sie weiter ein:...1
  und betätigen Sie die Taste -ENTER-
  (Hinweis: Mit der Taste -ENTER- haben Sie die Zeile abgeschlossen. Sie wurde dadurch in den Programmspeicher übernommen. Der Kursor stellt sich automatisch an den Anfang der nächsten Zeile.)
- Geben Sie weiter möglichst fehlerfrei ein:

```
20 PRINT I,I+I      Taste -ENTER-
30 NEXT I           Taste -ENTER-
40 GOTO 20          Taste -ENTER-
```

Beim Lesen des Programms auf dem Bildschirm stellen Sie fest, daß Sie eigentlich I*I statt I+I haben wollten. Sie müssen editieren, also eine Programmkorrektur vornehmen!

. Eingabe der Tastenfolge EDIT 20
  (Damit wird die Zeile 20 aus dem Programmspeicher auf dem Bildschirm angezeigt und zur Korrektur zur Verfügung gestellt)
. Kursor auf Zeichen + stellen
. Taste -DEL-
. Taste -INS-
. Taste -*-
. Taste -ENTER- (Zeile ist damit gültig erklärt)
. Taste -BRK- (Abschluß der EDIT-Funktion).

Die Unordnung auf dem Bildschirm stört. Sie möchten den Bildschirm löschen und nur Ihr Programm auf dem Bildschirm sehen.

. Tastenfolge CLS (cleare screen)
. Taste -ENTER-
. Tastenfolge LIST
. Taste -ENTER-

4. Programmtest

- Starten Sie Ihr Programm mit der Tastenfolge RUN und der Taste -ENTER-
- Auf dem Bildschirm erscheint folgende Ausschrift:

```
1             1
2             4
3             9
4             16
5             25
6             36
? NF ERROR IN 30
OK
>
```

Im Programm ist ein logischer Fehler. Die Fehlermeldung NF ERROR IN 30 bedeutet, daß in der 30. Zeile ein NEXT steht, ohne daß zuvor ein FOR steht. Diese Meldung kommt deshalb zustande, weil Sie von der Zeile 40 mit GOTO zur Zeile 20 springen (also mitten in die Laufanweisung hinein). Sie müssen die Zeile 40 editieren und an den Anfang der Laufanweisung springen!

- Editieren Sie:
  - Tastenfolge EDIT 40
  - Taste -ENTER-
  - Kursor auf Zeichen 2 stellen
  - Taste -DEL-
  - Taste -INS-
  - Taste -1-
  - Kursor an das Ende der Zeile stellen
  - Taste -ENTER-
  - Taste -BRK-
- Starten Sie das Programm erneut und beobachten Sie den Bildschirm:
  - Tastenfolge RUN
  - Taste -ENTER-

  Das Programm enthält eine Endlosschleife. Um die Abarbeitung zu unterbrechen, drücken Sie die Taste -BRK-. Der Rechner meldet:

```
BREAK  IN  xx (xx = Zeilennummer, in der der Abbruch
               erfolgt)
OK
>
```

  Um das Programm zu einem Ende zu führen, muß die Zeile 40 verändert werden. Das ist möglich durch:

. Überschreiben der Zeile oder
. Löschen der Zeile und Eingabe einer neuen.

- Überschreiben einer Programmzeile:
  Geben Sie ein: 40 END und drücken Sie die Taste -ENTER-
  Die alte Zeile 40 (GOTO 10) ist nun durch die neue überschrieben.
- Löschen einer Programmzeile:
  Geben Sie ein: 40 und drücken Sie die Taste -ENTER-
  Beachten Sie: Durch Eingabe nur der Zeilennummer wird diese Zeile gelöscht. Jetzt müssen Sie natürlich die Zeile 40 neu eingeben!

- Testen Sie beide Varianten!

5. Löschen des Programms
   - Tastenfolge NEW
   - Taste -ENTER-

   Überprüfen Sie das Löschen durch:
   - Tastenfolge LIST
   - Taste -ENTER-

   Der Rechner meldet sich mit: OK
   >

5.2. Gestaltung von Programmen

Beim Entwurf von Programmen zur rechnerunterstützten Lösung komplexer Aufgabenstellungen tritt das Problem der Übersichtlichkeit sowohl im Entwurfsstadium als auch bei der späteren Nutzung der Programme gegenüber anderen Fragen, wie der Programmlänge oder der Rechenzeit, in den Vordergrund. In der Literatur werden für den Programmentwurf im wesentlichen zwei Wege gezeigt, die "Top down"- und die "Botton up"-Methode. Bei der "Top down"-Methode geht man beim Programmentwurf so vor, wie die spätere Nutzung des Programms erfolgt, also von der allgemeineren, übergeordneten Aufgabenstellung oder Fragestellung zur immer konkreteren, untergeordneten. Die "Botton up"-Methode verfolgt genau den umgekehrten Weg. Es werden Teilprogramme für die Lösung von Teilaufgaben geschaffen, die dann zur Lösung komplexer Aufgaben geeignet zusammengefaßt und verknüpft werden. In der Praxis schlägt

man meist einen Zwischenweg ein. Nach Möglichkeit sollte aber dem "Top down"-Entwurf der Vorzug gegeben werden, weil dadurch übersichtliche, direkt zweckgebundene und zielgerichtete Programmstrukturen entstehen.
Gleichgültig aber, welche Methode angewendet wird, sollte man auch bei der BASIC-Programmierung folgendes beachten:

1. Einfache und übersichtliche Programme und Strukturen realisieren durch
   - Lösung einer Teilaufgabe in einem Programmteil (Modul)
   - Anwendung der strukturierten Programmierung, auch wenn das mit BASIC nicht gut und durchgängig möglich ist.

2. Nutzerfreundliche Programme schaffen durch
   - Führung des Nutzers durch das Programm
   - Ausgabe von Fehlermeldungen und Angebote zu ihrer Beseitigung
   - Vermeidung von Anpassungsproblemen durch Gebrauch einer im Nutzerkreis gebräuchlichen Sprache und Datenschreibweise.

Eine Möglichkeit zum Entwurf übersichtlicher Programmstrukturen ist die Unterprogrammtechnik. Dabei wird für jede zu lösende Aufgabenstellung ein Unterprogramm (UP) geschrieben. In BASIC sind Unterprogramme übliche BASIC-Programme, die als letzte Anweisung die RETURN-Anweisung enthalten müssen (und nicht END oder STOP). Die Einordnung des Unterprogramms zur Lösung einer Teilaufgabe im Komplex einer Gesamtaufgabenlösung regelt (steuert) ein übergeordnetes Programm, das sogenannte Hauptprogramm (HP). Der Zusammenhang zwischen HP und UP ist der, daß das HP das UP aufruft und nach dem Abarbeiten des UPs automatisch (das regelt die RETURN-Anweisung am Schluß des UPs) in das HP zurückgekehrt wird. Das Verlassen des HPs erreicht man durch die Anweisung

```
znrhp GOSUB znrup
```

(znrhp=Zeilennummer im HP, znrup=Zeilennummer, mit der das UP beginnt).
Nach der Abarbeitung des UPs wird an die nächsthöhere Zeilennummer des HPs nach znrhp gesprungen.
Eine Menge von Unterprogrammen faßt man in einem Unterprogrammsystem zusammen. Die einzelnen UPs eines solchen Unterprogrammsystems werden dann je nach Bedarf durch das Hauptprogramm aufgerufen. An folgendem allgemeingültigen Beispiel soll der Zusam-

menhang nochmals verdeutlicht werden:

```
•                                   znr REM HAUPTPROGRAMM
•                                   •
•                                   • Anweisungen im HP
znr1 REM UP1                        •
•                                   
• Anweisungen UP1                   3ØØØ GOSUB znr2
•                                   3Ø1Ø REM von UP2
znr RETURN                          •
znr2 REM UP2                        • Anweisungen im HP
•                                   •
• Anweisungen UP2                   32ØØ GOSUB znr1
•                                   321Ø REM von UP1
znr RETURN                          •
     •                              • Anweisungen im HP
     •                              znr END
znr REM UPn
•
• Anweisungen UPn
•
znr RETURN
```

Durch die GOSUB-Anweisung des HPs in Zeile 3ØØØ wird in das UP2 gesprungen. Die Anweisungen des UP2 werden abgearbeitet und beim Erreichen der RETURN-Anweisung wird zur Zeile 3Ø1Ø der HPs zurückgekehrt, da 3Ø1Ø die nach 3ØØØ nächsthöhere Zeilennummer im HP ist. Nun werden die Anweisungen im HP realisiert, bis bei Zeile 32ØØ der Sprung in das UP1 erfolgt. Nach Rücksprung aus dem UP1 an die Zeile 321Ø des HPs werden die Anweisungen des HPs ausgeführt und das Ende des Hauptprogramms erreicht.
Im hier dargestellten Hauptprogramm erfolgt eine feste Abfolge der Unterprogrammaufrufe. Häufig liegt aber der Fall vor, daß, vom Hauptprogramm ausgehend, der Nutzer entscheiden möchte, welches UP jetzt in der Folge aktuell angesprungen werden soll. Der Nutzer will sich also sein eigenes "Menü" zusammenstellen. Mit der Realisierung dieser Forderung erreicht man eine flexiblere Nutzung und damit breitere Anwendung des UP-Systems.
Im Hinblick auf die nutzerfreundliche Programmgestaltung kann dieses Problem wie folgt gelöst werden:

- Ausgabe eines "Menüs" auf dem Bildschirm, z. B. in folgender Form:

UP-System	Kennung
UP1	1
UP2	2
:	:
UPn	n

Wählen Sie eine Kennung:

- Aufruf des aktuellen UPs durch einen Sprungverteiler, der durch die eingegebene Kennung gesteuert wird:

```
ON variable GOSUB znr1,znr2,znr3,...,znrn
```

Zur Erinnerung: Nimmt die -variable- hinter ON den Wert 1 an, so erfolgt der Aufruf des UPs, welches bei der Zeilennummer znr1 beginnt. Nimmt sie dagegen den Wert 3 an, so wird das UP angesprungen, welches bei der Zeilennummer znr3 beginnt, da znr3 an der 3. Stelle nach GOSUB in der Verteileranweisung steht.

Folgendes Programmbeispiel würde dieser interaktiven Arbeitsweise des Nutzers mit dem Rechner Rechnung tragen:

```
.
.
.
PRINT"UP-MENUE"
PRINT"UP-SYSTEM                KENNUNG"
PRINT"_____________________________"

PRINT"  UP1                    1"
PRINT"  UP2                    2"
  .                            .
  .                            .
  .                            .
PRINT"  UP3                    3"
PRINT"_____________________________"
PRINT
PRINT"WAEHLEN SIE EINE KENNUNG:"
```

```
INPUT N
ON N GOSUB znr1,znr2,...,znrn
•
•
•
```

In den folgenden Programmbeispielen wird die konkrete Anwendung dieser "Menütechnik" gezeigt. Alle Programme sind für den Kleincomputer KC 85/3 geschrieben und auf diesem getestet.

5.3. Programmbeispiele

5.3.1. Sortieren von Datenfolgen

Eine Datenfolge ist eine Menge von Daten, die unter einem Sammelbegriff zusammengefaßt werden können. Beispiele dafür sind

- die Menge aller Zeichnungsnummern der Einzelteile eines Gerätes oder
- die Menge aller Arbeitsgänge zur Herstellung eines Einzelteiles oder
- die Menge aller Körpertemperaturen, Blutdrücke und Herzfrequenzen einer Patientengruppe.

Sortierte Folgen sind Folgen, deren Daten oder deren Ordnungsnummern nach einem Ordnungsprinzip geordnet sind. Beispielsweise kann man eine Zahlenfolge aufsteigend, beginnend mit dem Minimum bis zum Maximum, geordnet in einem Feld im Hauptspeicher (Arbeitsspeicher des Rechners) ablegen. Zur Lösung einer solchen Aufgabenstellung werden in der Literatur mehrere mögliche Algorithmen angegeben. Drei davon sollen beschrieben werden. Es wird davon ausgegangen, daß in einem Feld A im Arbeitsspeicher des Rechners eine ungeordnete Folge vorliegt.

- Sortieren durch Aussondern aus der Folge:
  1. Im Feld A das Minimum suchen.
  2. Das Minimum von A in das erste Feldelement des Feldes B ablegen.
  3. Das Feldelement von A, in dem das Minimum gefunden wurde, mit der größten Zahl des Rechners belegen.

4. Weiter bei 1., bis alle Feldelemente von A bearbeitet sind.

- Sortieren durch Umspeichern in der Folge:

1. Im Feld A das Minimum suchen.
2. Das gefundene Minimum gegen den Wert von A(1) tauschen.
3. Minimumsuche ab Feldelement A(2) fortsetzen und das gefundene Minimum gegen den Wert des Feldelementes A(2) austauschen.
4. Wie 3. für alle Feldelemente von A.

- Sortieren durch Permutation:

1. Beginnend mit I=1 Vergleich von A(I+1) mit A(I) bis die Relation A(I+1)<A(I) auftritt.
2. Austausch von A(I+1) mit A(I).
3. Rückwärts weiter Austauschen bis A(I-1)<A(I) gilt.
4. Weiter bei 1. mit I= Wert von Abbruchstelle.

Nachfolgend sind die drei Algorithmen in drei Unterprogrammen realisiert.

```
500 REM AUSSORTIEREN AUS DER FOLGE
510 PRINT"AUSSORTIEREN AUS DER FOLGE"
520 DIMB(N)
530 FOR J=0 TO N
540 M=1.70141E+38
550 FOR I=6TO N
560 IFA(I)<M THEN M=A(I):K=I
570 NEXTI
580 B(J)=M: A(K)=1.70141E+38
590 NEXTJ
595 RETURN
600 REM UMSPEICHERN IN DER FOLGE
610 PRINT"UMSPEICHERN IN DER FOLGE"
620 FOR J=0 TO N-1
630 M=A(J):K=J
640 FOR I=J TO N-1
650 IF A(I)<M THEN M=A(I):K=I
660 NEXTI
670 A(K)=A(J):A(J)=M
680 NEXTJ
690 RETURN
700 REM PERMUTATION
710 PRINT"PERMUTATION"
720 FOR J=0 TO N-2
730 IFA(J+1)<A(J) THEN 750
740 GOTO 790
```

```
750 I=J+1:K=J+1
760 Z=A(I):A(I)=A(I-1):A(I-1)=Z:I=I-1
770 IF I=0 THEN  790
780 IF A(I)<A(I-1) THEN 760
790 NEXTJ
795 RETURN
#
```

Das angegebene kleine Unterprogrammsystem soll nun mit einem Testprogramm getestet werden. Das nachfolgende Programm "Test-Sortieren" stellt eine mögliche Lösung dieser Aufgabenstellung dar. Von Zeile 1Ø bis Zeile 44Ø steht das Hauptprogramm. Ab Zeile 5ØØ bis Zeile 795 stehen die 3 UPs "Aussortieren", "Umspeichern" und "Permutation".

Das Hauptprogramm realisiert im wesentlichen

- die Dimensionierung des Feldes A und die Handeingabe der Daten (Zeile 3Ø bis Zeile 9Ø);
- die Ausgabe des Menüs auf dem Bildschirm (Zeile 1ØØ bis Zeile 15Ø);
- die Eingabeanforderung der auszuwählenden Kennung für das gewünschte UP und die Prüfung der Kennung auf ihre Zulässigkeit (Zeile 16Ø bis Zeile 18Ø);
  (Fehlerhafte Eingaben werden durch Ausschrift "FEHLEINGABE" angezeigt und führen automatisch zur erneuten Ausgabe des Menü-Angebotes und zur Eingabe einer Kennung zurück.);
- die Verteilung zu den UPs entsprechend der gewählten Kennung (Zeile 19Ø);
- die Verteilung zu zwei möglichen Ausgabeprogrammen, die ab Zeile 3ØØ bzw. Zeile 4ØØ des Hauptprogramms stehen (Zeile 2ØØ). Zu beachten ist, daß nach erfolgter Sortierung in einem UP durch die Anweisung RETURN immer an die Zeile 2ØØ des Hauptprogramms zurückgekehrt wird. Hier findet automatisch die Zuweisung des Ausgabeprogramms statt. Bei Wahl des UPs "Aussortieren" (Kennung=1) wird das Ausgabeprogramm ab Zeile 3ØØ und bei Wahl von UP "Umspeichern" und "Permutation" (Kennung 2 oder 3) das Ausgabeprogramm ab Zeile 4ØØ zugewiesen.

Der Test der Unterprogramme wird nun durch den Start des Programms "Test-Sortieren" vorgenommen. Die nachfolgend dargestellten Testprotokolle zeigen zuerst den Test des UPs "Aussortieren" und anschließend den der Unterprogramme "Umspeichern" und "Permutation". Als Beispiel ist die ungeordnete Zahlenfolge 3,8,7,4, 2,18,12 in das Feld A eingegeben worden. Durch die Wahl der Kennung 1,2 und 3 erfolgt die "Rechnung" in den UPs und die Ausgabe der Ergebnisse auf dem Bildschirm.
Im Protokoll "Aussortieren" ist bemerkenswert, daß die Folge in A in der Ergebnisanzeige nur noch die Zahl 1.7Ø141E+38 enthält. Das ist die größte darstellbare Zahl des KC 85/3. Sie wurde im UP "Aussortieren" nach der jeweiligen Minimumsuche in alle Feldelemente von A gespeichert.

```
10 REM TEST-SORTIEREN
20 CLS:PRINT"TEST-SORTIEREN"
30 PRINT:PRINT"GEBEN SIE LAENGE DER FOLGE EIN":INPUTN
40 PRINT:PRINT"GEBEN SIE DIE FOLGE EIN"
50 DIM A(N-1)
60 FORI=0TON-1
70 PRINT"A(";I+1;")=";:INPUTA(I)
80 NEXTI
90 PRINT:PRINT"EINGABE BEENDET"
100 PRINT:PRINT"SORTIERPROGRAMMAUSWAHL"
110 PRINT:PRINT"PROGRAMM              KENNUNG"
120 PRINT"_____________________________"
130 PRINT:PRINT"AUSSORTIEREN              1"
140 PRINT"UMSPEICHERN             2"
150 PRINT"PERMUTATION             3"
160 PRINT:PRINT:PRINT"WAEHLEN SIE EINE KENNUNG";:INPUTW
170 IF W=1 OR W=2 OR W=3 THEN 190
180 "FEHLEINGABE":PAUSE(30):GOTO100
190 ON W GOSUB 500,600,700
200 ON W GOTO 300,400,400
300 CLS:PRINT:PRINT:PRINT"ANZEIGE DER FOLGEN"
310 PRINT:PRINT"FOLGE A               FOLGE B"
320 PRINT"_____________________________"
330 FOR I=0 TO N-1
340 PRINTTAB(1);A(I);TAB(23);B(I)
350 NEXTI:GOTO 440
400 CLS:PRINT:PRINT"ANZEIGE DER SORTIERTEN FOLGE"
410 FOR I=0 TO N-1
420 PRINTTAB(1);A(I)
430 NEXTI
440 END
```

```
500 REM ÀUSSORTIEREN AUS DER FOLGE
510 PRINT"AUSSORTIEREN AUS DER FOLGE"
520 DIMB(N-1)
530 FOR J=0 TO N-1
540 M=1.70141E+38
550 FOR I=0TO N-1
560 IFA(I)<M THEN M=A(I):K=I
570 NEXTI
580 B(J)=M: A(K)=1.70141E+38
590 NEXTJ
595 RETURN
600 REM UMSPEICHERN IN DER FOLGE
610 PRINT"UMSPEICHERN IN DER FOLGE"
620 FOR J=0 TO N-1
630 M=A(J):K=J
640 FOR I=J TO N-1
650 IF A(I)<M THEN M=A(I):K=I
660 NEXTI
670 A(K)=A(J):A(J)=M
680 NEXTJ
690 RETURN
700 REM PERMUTATION
710 PRINT"PERMUTATION"
720 FOR J=0 TO N-2
730 IFA(J+1)<A(J) THEN 750
740 GOTO 790
750 I=J+1:K=J+1
760 Z=A(I):A(I)=A(I-1):A(I-1)=Z:I=I-1
770 IF I=0 THEN  790
780 IF A(I)<A(I-1) THEN 760
790 NEXTJ
795 RETURN
#
```

```
 RUN
,TEST-SORTIEREN

GEBEN SIE LAENGE DER FOLGE EIN
? 7

GEBEN SIE DIE FOLGE EIN
A( 1 )=? 3
A( 2 )=? 8
A( 3 )=? 7
A( 4 )=? 4
A( 5 )=? 2
A( 6 )=? 18
A( 7 )=? 12

EINGABE BEENDET

SORTIERPROGRAMMAUSWAHL

PROGRAMM                KENNUNG
-----------------------------

AUSSORTIEREN              1
UMSPEICHERN               2
PERMUTATION               3


WAEHLEN SIE EINE KENNUNG? 1
AUSSORTIEREN AUS DER FOLGE
,

ANZEIGE DER FOLGEN

FOLGE A                 FOLGE B
-----------------------------
  1.70141E+38             2
  1.70141E+38             3
  1.70141E+38             4
  1.70141E+38             7
  1.70141E+38             8
  1.70141E+38             12
  1.70141E+38             18
OK
>
```

```
 RUN
TEST-SORTIEREN

GEBEN SIE LAENGE DER FOLGE EIN
? 7

GEBEN SIE DIE FOLGE EIN
A( 1 )=? 3
A( 2 )=? 8
A( 3 )=? 7
A( 4 )=? 4
A( 5 )=? 2
A( 6 )=? 18
A( 7 )=? 12

EINGABE BEENDET

SORTIERPROGRAMMAUSWAHL

PROGRAMM              KENNUNG
-----------------------------

AUSSORTIEREN             1
UMSPEICHERN              2
PERMUTATION              3


WAEHLEN SIE EINE KENNUNG? 2
UMSPEICHERN IN DER FOLGE

ANZEIGE DER SORTIERTEN FOLGE
  2
  3
  4
  7
  8
  12
  18
OK
>
```

```
 RUN
*TEST-SORTIEREN

GEBEN SIE LAENGE DER FOLGE EIN
? 7

GEBEN SIE DIE FOLGE EIN
A( 1 )=? 3
A( 2 )=? 8
A( 3 )=? 7
A( 4 )=? 4
A( 5 )=? 2
A( 6 )=? 18
A( 7 )=? 12

EINGABE BEENDET

SORTIERPROGRAMMAUSWAHL

PROGRAMM                KENNUNG
-----------------------------

AUSSORTIEREN              1
UMSPEICHERN               2
PERMUTATION               3


WAEHLEN SIE EINE KENNUNG? 3
PERMUTATION
*
ANZEIGE DER SORTIERTEN FOLGE
  2
  3
  4
  7
  8
  12
  18
OK
>
```

5.3.2. Unterprogrammsystem zur Matrizenrechnung

Im folgenden wird ein Unterprogrammsystem zur Matrizenrechnung vorgestellt, seine Testung mit einem Testprogramm vorgenommen und die Nutzung am Beispiel der Lösung eines linearen Gleichungssystems gezeigt. Das Unterprogrammsystem besteht aus folgenden UPs:

UP-Name	Erläuterung	1. Zeilennummer
Addition	C=A+B	10000
Substraktion	C=A-B	10100
Transponieren	C=A(T)	10200
Mult. mit Konst.	C=k*A	10300
Multiplikation	C=A*B	10400
A=Nullmatrix	Alle Elemente=0	10500
B=Nullmatrix	Alle Elemente=0	10520
A=Einheitsmatrix	Elemente der HD=1	10600
B=Einheitsmatrix	Elemente der HD=1	10650
Tausch A mit C	C ⟷ A	10700
Tausch B mit C	C ⟷ B	10750
Laden A mit C	A=C	10800
Laden B mit C	B=C	10850
Inversion von A	A=A^(-1)	11000

Das UP-System beginnt bei der Zeilennummer 10000 und endet bei der Zeilennummer 11310. Die Dimensionierung der Felder muß im aufrufenden Hauptprogramm erfolgen. Neben den Feldern A, B und C benutzt das UP-System noch intern das Feld S(1,30). Auch dieses Feld muß im aufrufenden Programm definiert werden. Sollen eindimensionale Felder (Vektoren) zum Einsatz kommen, so sind diese als zweidimensionale Felder mit dem jeweiligen Zeilen- oder Spaltenindex=0 zu dimensionieren (z. B. B(30,0), wenn B ein Zeilenvektor mit 30 Elementen sein soll). Die maximal mögliche Größe der Indizes ist 30.

Das UP-System Matrizenrechnung benutzt (und verbraucht somit) folgende Variablenbezeichnungen:

- Felder: A(Z1,S1),B(Z2,S2),C(Z3,S3),S(1,3Ø)
  Z1,Z2,Z3,S1,S2,S3 sind die jeweils zu dimensionierenden Zeilen- und Spaltenindizes der Felder

- Einfache Variable: Z1,Z2,Z3,Z8,Z9
  S1,S2,S3,S7,S8,S9
  K,L

Nachfolgend ist das komplette Unterprogrammsystem angegeben.

```
10000 REM UP-SYSTEM MATRIZENRECHNUNG
10005 S3=S1:Z3=Z1:REM FELD+FELD(C=A+B)
10010 IF Z1=Z2 AND S1=S2THEN 10020
10015 PRINT"DIMENSIONIERUNGSFEHLER":PAUSE(30):GOTO10050
10020 FOR S9=0 TO S1:FOR Z9=0 TO Z1
10030 C(Z9,S9)=A(Z9,S9)+B(Z9,S9)
10040 NEXTZ9:NEXTS9
10050 RETURN
10100 REM FELD-FELD(C=A-B)
10105 S3=S1:Z3=Z1
10110 IF Z1=Z2 AND S1=S2 THEN 10120
10115 PRINT"DIMENSIONIERUNGSFEHLER":PAUSE(30):GOTO10150
10120 FOR S9=0 TO S1:FOR Z9=0 TO Z1
10130 C(Z9,S9)=A(Z9,S9)-B(Z9,S9)
10140 NEXTZ9:NEXTS9
10150 RETURN
10200 REM TRANSPONIERE FELD A IN FELD C
10205 S3=Z1:Z3=S1
10210 FOR Z9=0 TO Z1:FOR S9=0 TO S1
10220 C(S9,Z9)=A(Z9,S9)
10230 NEXTS9:NEXTZ9:RETURN
10300 REM KONSTANTE*FELD A (C=K*A)
10305 Z3=Z1:S3=S1
10310 FOR S9=0 TO S1:FOR Z9=0 TO Z1
10320 C(Z9,S9)=K*A(Z9,S9)
10330 NEXTZ9:NEXTS9:RETURN
10400 Z3=Z1:S3=S2:REM FELD*FELD(C=A*B)
10410 IF S1=Z2 THEN 10420
10415 PRINT"VERKETTUNGSFEHLER":PAUSE(30):GOTO10440
10420 FOR Z9=0 TOZ2:FOR S9=0 TO S2:C(Z9,S9)=0
10425 FOR K=0 TO Z3:C(Z9,S9)=C(Z9,S9)+A(Z9,K)*B(K,S9)
10430 NEXTK:NEXTS9:NEXTZ9
10440 RETURN
```

```
10500 REM NULLFELD AUF FELD A(A=0)
10510 FOR S9=0 TO S1:FOR Z9=0 TO Z1
10515 A(Z9,S9)=0:NEXTZ9:NEXTS9:RETURN
10520 REM NULLFELD AUF FELD B(B=0)
10530 FOR S9=0 TO S2:FOR Z9=0 TOZ2
10540 B(Z9,S9)=0:NEXTZ9:NEXTS9:RETURN
10600 REM EINHEITSMATRIX AUF A
10610 IF Z1=S1 THEN 10620
10615 PRINT"MATRIX NICHT QUATRATISCH":PAUSE(30):GOTO10645
10620 FOR Z9=0 TO Z1:FOR S9=0 TO S1
10630 IF S9=Z9 THEN A(Z9,S9)=1:ELSE A(Z9,S9)=0
10640 NEXTS9:NEXTZ9
10645 RETURN
10650 REM EINHEITSMATRIX AUF B
10660 IF Z2=S2 THEN 10670
10665 PRINT"MATRIX NICHT QUADRATISCH":PAUSE(30):GOTO10690
10670 FOR Z9=0 TO Z2:FOR S9=0 TOS2
10675 IF S9=Z9 THEN B(Z9,S9)=1:ELSE B(Z9,S9)=0
10680 NEXTS9:NEXTZ9
10690 RETURN
10700 REM TAUSCH FELD A MIT FELD C
10710 IF S1=S3 AND Z1=Z3 THEN 10720
10715 PRINT"MATRIZEN SIND UNTERSCHIEDLICH":PAUSE(30):GOTO10740
10720 FOR Z9=0 TO Z1:FOR S9=0 TO S1
10725 K=A(Z9,S9):A(Z9,S9)=C(Z9,S9):C(Z9,S9)=K
10730 NEXTS9:NEXTZ9
10740 RETURN
10750 REM TAUSCH FELD B MIT FELD C
10760 IF S2=S3 AND Z2=Z3 THEN 10770
10765 PRINT"MATRIZEN SIND UNTERSCHIEDLICH" :PAUSE(30):GOTO10790
10770 FOR Z9=0 TO Z2:FOR S9=0 TOS2
10775 K=B(Z9,S9):B(Z9,S9)=C(Z9,S9):C(Z9,S9)=K
10780 NEXTS9:NEXTZ9
10790 RETURN
10800 REM LADEN FELD A MIT FELD C
10810 IF S1>=S3 AND Z1>=Z3 THEN10820
10815 PRINT"FELD A KLEINER FELD C":PAUSE(30):GOTO10840
10820 FOR S9=0 TO S3:FOR Z9=0 TO Z3
10825 A(Z9,S9)=C(Z9,S9):NEXTZ9:NEXTS9
10830 Z1=Z3:S1=S3
10840 RETURN
10850 REM LADEN FELD B MIT FELD C
10860 IF S2>=S3 AND Z2>=Z3 THEN 10870
10865 PRINT"FELD B KLEINER FELD C":PAUSE(30):GOTO10890
10870 FOR S9=0 TO S3:FOR Z9=0 TO Z3
10875 B(Z9,S9)=C(Z9,S9):NEXTZ9:NEXTS9
10880 Z2=Z3:S2=S3
10890 RETURN
11000 REM MATRIZENINVERSION VON A
11020 IF Z1<>S1 THEN PRINT" A NICHT A(N,N)":PAUSE(30):GOTO11290
11030 FOR Z8=0 TOZ1:S(1,Z8)=Z8:S(0,Z8)=Z8:NEXT Z8
11040 S8=-1
11050 S7=-1
```

```
11060 S9=0:S7=S7+1
11070 IF S7=Z1+1 THEN S7=1:GOTO 11300
11080 IF S(0,S7)=-1 THEN S7=S7+1:GOTO11070
11090 IF S9>=Z1+1 THEN 11060
11100 IF S(1,S9)=-1 THEN S9=S9+1:GOTO11090
11110 Z8=S(1,S9):Z9=S(0,S7)
11120 IF A(Z8,Z9)=0THEN 11180
11130 S(1,Z8)=-1:S(0,Z9)=-1
11140 S8=S8+1
11160 GOSUB 11190
11170 IF S8=Z1 THEN RETURN:ELSE11050
11180 IF S8=Z1-1 THEN 11300:ELSE S9=S9+1:GOTO 11090
11190 A(Z8,Z9)=1/A(Z8,Z9):FOR L=0 TO Z1
11200 IF L=S9 THEN 11210:ELSEA(Z8,L)=A(Z8,L)*((-1)*A(Z8,Z9))
11210 NEXTL
11220 FOR K=0 TO Z1:FOR L=0 TO Z1
11230 IF K=Z8 THEN 11260
11240 IF Z9=L THEN 11260
11250 A(K,L)=A(K,L)+A(K,Z9)*A(Z8,L)
11260 NEXTL:NEXTK
11270 FOR K=0 TO Z1
11275 IF K=Z8 THEN 11280:ELSE A(K,Z9)=A(K,Z9)*A(Z8,Z9)
11280 NEXT K
11290 RETURN
11300 FOR I=0 TO 5:PRINTS(0,I),S(1,I):NEXTI
11310 PRINT"MATRIX SINGULAR":PAUSE(50):GOTO11290
#
```

Mit dem anschließend vorgestellten "Programm zum UP-Systemtest" kann das Unterprogrammsystem "Matrizenrechnung" getestet werden. Das Programm beginnt bei der Zeilennummer 3ØØ und endet mit der Zeilennummer 11ØØ. Es ist als Hauptprogramm geschrieben und realisiert die Steuerung des Testes über zwei Menüs.
Im "Einstiegsmenü" (Zeile 32Ø bis Zeile 41Ø) ist realisiert

- das Laden der Felder A und B mit konstanten Werten, Feld A immer mit:

	1	2	3
	4	5	6
	7	8	9

Feld B immer mit:	9	8	7
	6	5	4
	3	2	1

Das Laden erfolgt über das Auslesen von DATA-Files in den Zeilen 5ØØ bis 55Ø und 6ØØ bis 65Ø;

- das geschlossene Anzeigen der Felder A, B und C auf dem Bildschirm;
- das Aufrufen des Menüs "UP-Menü-Matrizenrechnung";
- die Eingabe der Konstanten K zum Test des UP "Mult. mit Konst.";
- der Abbruch des Testes.

Das nach dem Programm angegebene Testprotokoll zeigt die Realisierung des Testes für

- das Laden des Feldes A (Kennung 1 im Einstiegsmenü);
- das Anzeigen der Felder A, B und C (Kennung 3 im Einstiegsmenü);
- den Aufruf des "UP-Menüs Matrizenrechnung" (Kennung 5 im Einstiegsmenü);
- den Aufruf des UP "Addition" C=A+B (Kennung 1 im UP-Menü Matrizenrechnung);
- das Anzeigen der Felder A, B und C (Kennung 3 im Einstiegsmenü);
- den Testabbruch (Kennung 6 im Einstiegsmenü).

```
300 REM PROGRAMM ZUM UP-SYSTEMTEST
305 REM DIMENSIONIERUNG DER FELDER
306 Z1=2:S1=2:Z2=2:S2=2:Z3=2:S3=2
307 DIM A(Z1,S1),B(Z2,S2),C(Z3,S3),S(1,30)
310 CLS:PRINT:PRINT:PRINT"UP-SYSTEMTEST"
320 PRINT:PRINT"EINSTIEGSMENUE"
330 PRINT:PRINT"PROGRAMM        KENNUNG"
340 PRINT"_____________________________"
350 PRINT
360 PRINT"LADEN FELD A          1"
370 PRINT"LADEN FELD B          2"
380 PRINT"ANZEIGE FELD A,B,C    3"
390 PRINT"EINGABE KONSTANTE     4"
400 PRINT"AUFRUF UP-MENUE       5"
405 PRINT"TESTABBRUCH           6"
410 PRINT:PRINT"WAEHLEN SIE EINE KENNUNG";:INPUT H
420 IF H=1ORH=2ORH=3ORH=4ORH=5ORH=6THEN 440
430 PRINT"FEHLEINGABE":PAUSE(10):GOTO 310
440 ON H GOTO 500,600,700,800,900,1100
```

```
500 DATA1,2,3,4,5,6,7,8,9
510 RESTORE500
520 FOR I=0TO2:FOR J=0TO2
530 READ A(I,J):NEXTJ:NEXTI
540 PRINT"FELD A EINGELESEN VON DATAFILE"
550 PAUSE(10):GOTO310
600 DATA9,8,7,6,5,4,3,2,1
610 RESTORE600
620 FORI=0TO2:FORJ=0TO2
630 READ B(I,J):NEXTJ:NEXTI
640 PRINT"FELD B EINGELESEN VON DATAFILE"
650 PAUSE(10):GOTO 310
700 CLS:PRINT:PRINTTAB(5);"A":PRINT
710 FORI=0TO2:PRINT:FORJ=0TO2:P=J+1:PRINTA(I,J);TAB(P*10);
720 NEXTJ:NEXTI
730 PRINT:PRINT:PRINT:PRINTTAB(5);"B":PRINT
740 FORI=0TO2:PRINT:FORJ=0TO2:P=J+1:PRINTB(I,J);TAB(P*10);
750 NEXTJ:NEXTI
760 PRINT:PRINT:PRINT:PRINTTAB(5);"C":PRINT
770 FORI=0TO2:PRINT:FORJ=0TO2:P=J+1:PRINTC(I,J);TAB(P*10);
771 NEXTJ:NEXTI
772 PRINT:PRINT:PRINT"MIT ENTER-TASTE ZURUECK"
773 PRINT"IN DAS MENUE":INPUT" ";R:GOTO310
800 PRINT"EINGABE EINER REELLEN KONSTANTEN"
810 PRINT"IN DEN SPEICHERPLATZ K:":INPUT K
820 PRINT"UNTER K WURDE DER WERT";K;"ABGELEGT"
830 PAUSE(30):GOTO 310
900 CLS:PRINT"UP-MENUE-MATRIZENRECHNUNG"
910 PRINT:PRINT"UNTERPROGRAMM          KENNUNG"
920 PRINT"_____________________________"
930 PRINT:PRINT"ADDITION C=A+B              1"
940 PRINT"SUBTRAKTION C=A-B             2"
950 PRINT"MULTIPLIKATION C=A*B          3"
960 PRINT"MULT.A MIT KONST. C=K*A       4"
970 PRINT"INVERSION VON A               5"
980 PRINT"TAUSCH A MIT C                6"
990 PRINT"TAUSCH B MIT C                7"
1000 PRINT"LADEN A MIT C                 8"
1010 PRINT"LADEN B MIT C                 9"
1020 PRINT"TRANSPONIEREN A IN C         10"
1025 PRINT"A=0-MATRIX                   11"
1030 PRINT"B=0-MATRIX                   12"
1040 PRINT"EINHEITSMATRIX IN A          13"
1050 PRINT"EINHEITSMATRIX IN B          14"
1060 PRINT:PRINT"WAEHLEN SIE EINE KENNUNG";:INPUTG
1065 IF G<1ORG>14OR INT(G)<>G THENPRINT"FEHLER":PAUSE(10):GOTO900
1070 IF G<8 THEN 1080:ELSE 1090
1080 ONG GOSUB 10000,10100,10400,10300,11000,10700,10750
1085 GOTO310
1090 ON G-7 GOSUB 10800,10850,10200,10500,10520,10600,10650
1095 GOTO310
1100 END
```

```
 RUN300
*

UP-SYSTEMTEST

EINSTIEGSMENUE

PROGRAMM          KENNUNG
--------------------------

LADEN FELD A          1
LADEN FELD B          2
ANZEIGE FELD A,B,C    3
EINGABE KONSTANTE     4
AUFRUF UP-MENUE       5
TESTABBRUCH           6

WAEHLEN SIE EINE KENNUNG? 1
FELD A EINGELESEN VON DATAFILE
*

UP-SYSTEMTEST

EINSTIEGSMENUE

PROGRAMM          KENNUNG
--------------------------

LADEN FELD A          1
LADEN FELD B          2
ANZEIGE FELD A,B,C    3
EINGABE KONSTANTE     4
AUFRUF UP-MENUE       5
TESTABBRUCH           6

WAEHLEN SIE EINE KENNUNG? 3
*
     A


 1          2          3
 4          5          6
 7          8          9


     B


 0          0          0
 0          0          0
 0          0          0
```

```
     C

 0          0          0
 0          0          0
 0          0          0

MIT ENTER-TASTE ZURUECK
IN DAS MENUE

*

UP-SYSTEMTEST

EINSTIEGSMENUE

PROGRAMM          KENNUNG
--------------------------

LADEN FELD A          1
LADEN FELD B          2
ANZEIGE FELD A,B,C    3
EINGABE KONSTANTE     4
AUFRUF UP-MENUE       5
TESTABBRUCH           6

WAEHLEN SIE EINE KENNUNG? 5
*UP-MENUE-MATRIZENRECHNUNG

UNTERPROGRAMM                KENNUNG
-----------------------------------

ADDITION C=A+B                  1
SUBTRAKTION C=A-B               2
MULTIPLIKATION C=A*B            3
MULT.A MIT KONST. C=K*A         4
INVERSION VON A                 5
TAUSCH A MIT C                  6
TAUSCH B MIT C                  7
LADEN A MIT C                   8
LADEN B MIT C                   9
TRANSPONIEREN A IN C           10
A=0-MATRIX                     11
B=0-MATRIX                     12
EINHEITSMATRIX IN A            13
EINHEITSMATRIX IN B            14

WAEHLEN SIE EINE KENNUNG? 1
*
```

```
UP-SYSTEMTEST

EINSTIEGSMENUE

PROGRAMM          KENNUNG
-------------------------

LADEN FELD A          1
LADEN FELD B          2
ANZEIGE FELD A,B,C    3
EINGABE KONSTANTE     4
AUFRUF UP-MENUE       5
TESTABBRUCH           6

WAEHLEN SIE EINE KENNUNG?
                              3
,
     A


 1         2         3
 4         5         6
 7         8         9

     B


 0         0         0
 0         0         0
 0         0         0

     C


 1         2         3
 4         5         6
 7         8         9

MIT ENTER-TASTE ZURUECK
IN DAS MENUE

,

UP-SYSTEMTEST

EINSTIEGSMENUE
```

```
PROGRAMM          KENNUNG
--------------------------

LADEN FELD A          1
LADEN FELD B          2
ANZEIGE FELD A,B,C    3
EINGABE KONSTANTE     4
AUFRUF UP-MENUE       5
TESTABBRUCH           6

WAEHLEN SIE EINE KENNUNG? 6
OK
>
```

Abschließend wird nun die Anwendung des Unterprogrammsystems "Matrizenrechnung" am Beispiel der Lösung des folgenden linearen Gleichungssystems gezeigt:

$$\begin{aligned} X1 + 2*X2 &= 2 \\ 2*X1 + 5*X2 + 3*X3 &= 0 \\ X1 - 5*X3 &= 4 \end{aligned}$$

Bei der Lösung dieser Aufgabe geht man so vor, daß man das Gleichungssystem wie folgt umschreibt:

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 5 & 3 \\ 1 & 0 & -5 \end{pmatrix} * \begin{pmatrix} X1 \\ X2 \\ X3 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 4 \end{pmatrix}$$

In mathematischer Schreibweise ist das eine Matrizengleichung der Form:

$$A * X = B$$

Dabei ist: A die Matrix der Koeffizienten der Variablen
X der Variablenvektor
B der Konstantenvektor.

Den Lösungsvektor X erhält man, indem man auflöst und schreibt:

$$X = A^{(-1)} * B.$$

Der Lösungsvektor ergibt sich also aus der Multiplikation der transponierten Matrix A mit dem Vektor B. Für beide mathemati-

schen Operationen stellt das UP-System Unterprogramme bereit. Somit hat man sich nur noch um die Anpassung des Problems an das UP-System zu kümmern. Im wesentlichen sind 3 Problemkreise zu beachten:

1. Das UP-System kennt kein Feld mit dem Namen X. Das Ergebnis der Matrizenmultiplikation steht automatisch im Feld C. Man wird also das Ergebnis von

   X1 in C(Ø,Ø)
   X2 in C(1,Ø)
   X3 in C(2,Ø)

   finden.

2. Die benötigten Felder sind zu dimensionieren. Das Feld A muß zur Aufnahme der Matrix A aus 3 Zeilen und 3 Spalten bestehen. Das Feld B und C jeweils aus 3 Zeilen und 1 Spalte. Das Feld S dimensioniert man vorsichtshalber so, wie es in der Anwenderinformation steht. Man erspart sich damit eventuell auftretende Fehler durch Unterdimensionierung. Das Feld S erhält also die Dimension (1,3Ø).

3. Schließlich ist noch die Eingabe der aktuellen Werte der Matrizen zu realisieren und eine geeignete Ausgabe vorzusehen.

Das zu schreibende Hauptprogramm besteht somit aus 4 wesentlichen Bestandteilen:

1. Dimensionierung der Felder.
2. Eingabe der Werte der Matrizen.
3. Aufruf der UPs "Invertierung" und "Multiplikation".
4. Ausgabe der Lösungen.

Das anschließende Testprotokoll zeigt die Lösung der gegebenen Aufgabenstellung.

```
10 REM HAUPTPROGRAMM
20 REM LINEARES GLEICHUNGSSYSTEM
30 CLS:PRINT"LINEARES GLEICHUNGSSYSTEM"
40 REM DIMENSIONIERUNG DER MATRIZEN
50 Z1=2:S1=2:Z2=2:S2=0:Z3=2:S3=0
60 DIM A(Z1,S1),B(Z2,S2),C(Z3,S3),S(1,30)
70 PRINT"EINGABE MATRIX A"
80 FOR I=0 TOZ1:FOR J=0 TO S1
90 PRINT"A(";I+1;",";J+1;")=";:INPUTA(I,J)
100 NEXTJ:PRINT
110 NEXTI:PRINT:PRINT
120 PRINT"EINGABE VEKTOR B"
130 FOR I=0 TO Z2
140 PRINT"B(";I+1;")=";:INPUT B(I,0)
150 NEXT I
160 REM RECHNUNG IM UP-SYSTEM
170 PRINT"ES ERFOLGT RECHNUNG IM UP-SYSTEM"
180 GOSUB11000:REM UP-INVERSION VON A
190 GOSUB 10400:REM UP-MULTIPLIKATION  VON A*B
200 REM AUSGABE DES ERGEBNISSES
210 PRINT:PRINT"DAS GLEICHUNGSSYSTEM HAT FOLGENDE ":PRINT"LOE
    SUNGEN:"
220 PRINT:PRINT
230 PRINT"X1 =";C(0,0)
240 PRINT"X2 =";C(1,0)
250 PRINT"X3 =";C(2,0)
260 END
```

```
 RUN
,LINEARES GLEICHUNGSSYSTEM
EINGABE MATRIX A
A( 1 , 1 )=? 1
A( 1 , 2 )=? 2
A( 1 , 3 )=? 0

A( 2 , 1 )=? 2
A( 2 , 2 )=? 5
A( 2 , 3 )=? 3

A( 3 , 1 )=? 1
A( 3 , 2 )=? 0
A( 3 , 3 )=? -5
```

```
EINGABE VEKTOR B
B( 1 )=? 2
B( 2 )=? 0
B( 3 )=? 4
ES ERFOLGT RECHNUNG IM UP-SYSTEM

DAS GLEICHUNGSSYSTEM HAT FOLGENDE
LOESUNGEN:


X1 =-26
X2 = 14
X3 =-6
OK
>
```

5.3.3. Dateiarbeit mit dem KC 85/3

Eine häufig vorkommende Aufgabe besteht darin, anfallende Daten in einer Datei geordnet aufzubewahren. Gleichgültig, ob dienstlich oder privat, liegt das dringende Bedürfnis vor, den Rechner als großen Notizblock zu gebrauchen und dabei den Vorteil der Sortierung nach unterschiedlichen Kriterien und Protokollierung zu nutzen. Beim KC 85/3 wird als externes Speichermedium die Kassette eingesetzt. Damit ergeben sich folgende minimalen Forderungen an ein Dateiprogramm:

- Eingabe einer Datei über die Tastatur
- Ausgabe einer Datei auf die Kassette
- Eingabe einer Datei von Kassette und
- Anzeige einer Datei auf dem Bildschirm.

Das nachfolgend aufgeführte "Kleine Dateiprogramm" ermöglicht die Realisierung dieser Forderungen für eine Minidatei, bestehend aus Name, Geburtsdatum und Telefonnummer zum Beispiel eines Bekanntenkreises. Gegenüber den Programmbeispielen in den Gliederungspunkten 5.3.1. und 5.3.2. ist dieses Programm als geschlossenes Programm geschrieben. Besonders hervorgehoben muß werden, daß bei der Ein- und Ausgabe der Datei von der und auf

die Kassette die einzelnen Felder der Datei auch einzeln im Programm ein- bzw. ausgegeben werden müssen. Bei der Eingabe ist zudem zu beachten, daß jede Eingabe eines Feldes mit einer PRINT-Anweisung abzuschließen ist.

Im Anschluß an das Programm werden zwei Testprotokolle angegeben. Im ersten Testprotokoll ist nach der Eingabe der Kennung 1 die Handeingabe der Datei angedeutet. Sie wurde durch den Bediener nach der Forderung der Daten zur zweiten Person abgebrochen mit dem BREAK-Kommando, da keine neuen Erkenntnisse gewonnen werden. Im zweiten Testprotokoll wurde eine Datei von 10 Personen von Kassette eingelesen und auf dem Bildschirm angezeigt.

```
10 REM KLEINES DATEIPROGRAMM
20 CLS:PRINT:PRINT:PRINT"KLEINES DATEIPROGRAMM"
30 PRINT        "======================"
40 PRINT:PRINT"GEBEN SIE ANZAHL DER PERSONEN AN";:INPUTN
50 DIM N$(N-1),G$(N-1),T(N-1)
60 REM MENUE
70 CLS:PRINT:PRINT:PRINT"MENUE ZUR ARBEIT MIT DER DATEI"
80 PRINT:PRINT:PRINTTAB(5);"PROGRAMM";TAB(30);"KENNUNG"
90 PRINTTAB(5);"--------------------------------"
100 PRINTTAB(5)"HANDEINGABE DER DATEI";TAB(32);1
110 PRINTTAB(5);"AUSGABE AUF KASSETTE";TAB(32);2
120 PRINTTAB(5);"EINGABE VON KASSETTE";TAB(32);3
130 PRINTTAB(5);"ANZEIGE AUF DEM BILDSCHIRM";TAB(32);4
140 PRINTTAB(5);"PROGRAMMENDE";TAB(32);5:PRINT:PRINT:PRINT
150 PRINT"WAEHLEN SIE EINE KENNUNG";:INPUT K
160 IF K=1ORK=2ORK=3ORK=4ORK=5 THEN 190
170 PRINT:PRINT
180 PRINT"FEHLER,KENNUNG IST NICHT DEFINIERT!":PAUSE(50):GOTO 70
190 ON K GOTO 200,300,400,500,600
200 CLS:PRINT"HANDEINGABE DER DATEI"
```

```
210 FOR I=0 TO N-1
220 PRINT:PRINT:PRINT:PRINT"PERSON";I+1
230 PRINT"NAME";:INPUT N$(I)
240 PRINT"GEBURTSDATUM";:INPUT G$(I)
250 PRINT"TELEFON";:INPUT T(I)
260 CLS
270 NEXTI
280 PRINT:PRINT:PRINT"EINGABE IST BEENDET":PAUSE(50):GOTO70
300 PRINT:PRINT:PRINT"AUSGABE AUF KASSETTE"
310 PRINT:PRINT"BEREITEN SIE IHR KASSETTENGERAET VOR."
320 PRINT"QUITTIEREN SIE BEI AUFFORDERUNG MIT DER ENTER-TASTE"
330 INPUT"ENTER";M:CSAVE*"NAME";N$
340 INPUT"ENTER";M:CSAVE*"GEBURTSDATUM";G$
350 INPUT"ENTER";M:CSAVE*"TELEFON";T
360 PRINT:PRINT:PRINT"ENDE DER AUSGABE":PAUSE(50):GOTO 70
400 PRINT:PRINT:PRINT"EINGABE VON KASSETTE"
410 PRINT"BEREITEN SIE IHR KASSETTENGERAET VOR":INPUT"ENTER";M
420 CLOAD*"NAME";N$
430 PRINT
440 CLOAD*"GEBURTSDATUM";G$
450 PRINT
460 CLOAD*"TELEFON";T
470 PRINT
480 PRINT"ENDE DER EINGABE":PAUSE(50):GOTO 70
500 CLS:PRINT:PRINT:PRINT"ANZEIGE AUF DEM BILDSCHIRM":PRINT:PRINT
505 PRINT"NAME";TAB(16);"G.-DATUM";TAB(27);"TELEFON"
506 PRINT"_____":PRINT
510 FOR I=0 TO N-1
520 PRINTN$(I);TAB(16);G$(I);TAB(27);T(I)
530 NEXTI:PRINT:PRINT
540 PRINT"MIT ENTER-TASTE KOMMEN SIE WIEDER"
550 PRINT"IN DAS MENUE
560 INPUT"ENTER";M
570 PRINT"ENDE DER ANZEIGE":PAUSE(10):GOTO 70
600 CLS:PRINT:PRINT:PRINT"PROGRAMMENDE"
610 PRINT"ACHTUNG !!! DIE DATEI WIRD GELOESCHT"
620 PRINT"MIT -MENUE- KOMMEN SIE ZURUECK"
630 PRINT"IN DAS MENUE"
640 PRINT"ALLES ANDERE FUEHRT ZUM PROGRAMMENDE"
650 INPUTA$
660 IF A$="MENUE" THEN 70
670 END
#
```

```
 RUN
'

KLEINES DATEIPROGRAMM
=====================

GEBEN SIE ANZAHL DER PERSONEN AN? 10
'

MENUE ZUR ARBEIT MIT DER DATEI

     PROGRAMM                      KENNUNG
     ---------------------------------
     HANDEINGABE DER DATEI          1
     AUSGABE AUF KASSETTE           2
     EINGABE VON KASSETTE           3
     ANZEIGE AUF DEM BILDSCHIRM     4
     PROGRAMMENDE                   5


WAEHLEN SIE EINE KENNUNG? 1
'HANDEINGABE DER DATEI


PERSON 1
NAME? ANTON
GEBURTSDATUM? 20.01.63
TELEFON? 27142
'


PERSON 2
NAME?
BREAK IN 230
OK
>




 RUN
'

KLEINES DATEIPROGRAMM
=====================

GEBEN SIE ANZAHL DER PERSONEN AN? 10
'

MENUE ZUR ARBEIT MIT DER DATEI
```

```
     PROGRAMM                        KENNUNG
     ---------------------------------------
     HANDEINGABE DER DATEI             1
     AUSGABE AUF KASSETTE              2
     EINGABE VON KASSETTE              3
     ANZEIGE AUF DEM BILDSCHIRM        4
     PROGRAMMENDE                      5


WAEHLEN SIE EINE KENNUNG? 3


EINGABE VON KASSETTE
BEREITEN SIE IHR KASSETTENGERAET VOR
ENTER
ENDE DER EINGABE
*

MENUE ZUR ARBEIT MIT DER DATEI

     PROGRAMM  .                     KENNUNG
     ---------------------------------------
     HANDEINGABE DER DATEI             1
     AUSGABE AUF KASSETTE              2
     EINGABE VON KASSETTE              3
     ANZEIGE AUF DEM BILDSCHIRM        4
     PROGRAMMENDE                      5


WAEHLEN SIE EINE KENNUNG? 4
*

ANZEIGE AUF DEM BILDSCHIRM

NAME              G.-DATUM    TELEFON
-----------------------------------------

ANTON             20.01.63     27142
BERTA             21.04.64     34692
CORNELIA          12.11.64     43563
DORA              01.03.65     27443
EMIL              17.07.62     54666
FRANZ             23.06.61     43229
GUSTAV            31.08.64     22167
HEINRICH          09.10.57     66438
IDA               12.12.61     74623
JUERGEN           07.07.63     68532


MIT ENTER-TASTE KOMMEN SIE WIEDER
IN DAS MENUE
ENTER
```

6. Befehlsübersicht

Monitorkommandos

BASIC	Ruft BASIC-Interpreter auf
REBASIC	Ruft BASIC-Interpreter ohne Zerstörung geladener BASIC-Programme auf
KEY n	Definiert Funktionstaste n
KEYLIST	Zeigt vereinbarte Funktionstastenbelegung an

Kommandos

AUTO [znr] [,sw]	AUTOMATISCHE Zeilennumerierung
BLOAD	Liest Maschinenprogramm ein
BYE	Rückkehr zum Betriebssystem
CLEAR	Löscht Variablenspeicher
CLOAD "name"	Lädt BASIC-Programm von Kassette
CLS	Löscht Bildschirm
CONT	Setzt unterbrochenes Programm fort
CSAVE "name"	Speichert BASIC-Programm auf Kassette
DELETE [anf] [,end]	Löscht Zeile (n)
EDIT zeile	Programmkorrektur in Zeile
LINES zahl	Anzahl der Zeilen für LIST
LIST [zeile]	Listet Programm auf
NEW	Löscht Programm und Variablenspeicher
RENUMBER	Numeriert Programmzeilen neu
RUN	Startet Programm
TROFF	Schaltet Testmodus aus
TRON	Schaltet Testmodus ein
WIDTH länge	Legt Länge für Ausgabezeile fest

Anweisungen

BEEP [dauer]	Erzeugt einen Ton festgelegter Dauer
CALL adresse	Ruft ein Maschinenprogramm auf
CIRCLE xm,ym,r[,farbe]	Zeichnet einen Kreis auf dem Bildschirm mit den Mittelpunktkoordinaten xm,ym und dem Radius r
COLOR v,h	Stellt Vordergrund- und Hintergrundfarbe ein

DATA wert[,wert]	Datenfolge, der durch READ zu lesenden Werte
DEEK (adr)	Liest Inhalt der Speicherplätze adr und adr+1
DEFFNn(x)	Definiert eine Anwenderfunktion n mit dem Parameter x
DIM	Dimensioniert Variablenfelder
DOKE adr,wert	Schreibt Doppelbyte-Wert auf die Speicherplätze adr und adr+1
END	Ende des Programms
FOR lvar=anf TO end [STEP schrittw]	Festlegung einer Programmschleife, lvar läuft von anf bis end in Schritten von schrittw (Standart ist 1)
GOSUB znr	Ruft Unterprogramm auf Zeilennummer znr auf
GOTO znr	Sprung zu Zeilennummer znr
IF ausdruck THEN ausdruck1 [:ELSE ausdruck2]	Bedingter Sprung- oder Handlungsanweisung, wenn Ausdruck wahr, dann THEN-Zweig, sonst nächste Programmzeile oder ELSE-Zweig
INK v	Stellt Vordergrundfarbe ein
INKEY$	Liefert aktuellen Tastencode
INP adr	Liefert das von dem Port adr gelesene Byte
INPUT var[,var1]	Eingabe von Werten über Tastatur in Variable [n] nach Ausgabe von "?" auf Bildschirm
LET ausdruck = ausdruck1	Wertzuweisung ausdruck1 nach ausdruck
LINE xa,ya,xe,ye[,farbe]	Zeichnet eine Linie auf dem Bildschirm von Punkt xa,ya bis Punkt xe, ye
LOCATE znr,spnr	Positioniert den Cursor an Bildschirmposition Zeilennummer, Spaltennummer
NEXT [lvar]	Ende der Programmschleife
ON ausdr GOTO znr[,znr]	Mehrfache Programmverzweigung entsprechend Ausdruck ausdr

ON ausdr GOSUB znr[,znr]	Mehrfache Programmverzweigung in Unterprogramme entsprechend Ausdruck ausdr
OUT adr,wert	Ausgabe des Bytes wert aus dem Port adr
PAPER h	Stellt Hintergrundfarbe ein
PAUSE dauer	Unterbricht Programmabarbeitung für dauer * Ø.1 s
PEEK(adr)	Liest Inhalt von Speicherplatz adr
POKE adr,wert	Schreibt Byte-Wert in den Speicherplatz adr
PRESET x,y	Löscht Bildpunkt an Position x,y
PRINT wert[,wert]	Ausgabe auf Bildschirm
PSET x,y[,farbe]	Setzt Bildpunkt auf Position x,y
PTEST(x)	Test, ob Bildpunkt gesetzt; y gilt aus letzter Grafikanweisung. Als Ergebnis wird für einen Punkt der Wert 1, anderenfalls der Wert Ø erzeugt
RANDOMIZE	Startet Zufallsgenerator mit einem zufälligen Anfangswert
READ var[,var]	Liest Werte aus DATA-Anweisung in Variable
REM text	Textzeile text ist Kommentar
RESTORE [znr]	Setzt DATA-Zeiger auf den Anfang der Liste bzw. auf Zeilennummer znr
RETURN	Beendet ein Unterprogramm
STOP	Stoppt die Programmabarbeitung
WINDOW za,ze,sa,se	Unterteilt den Bildschirm in Fenster Fenster : Zeilenanfang,Zeilenende, Spaltenanfang,Spaltenende

---

Mathematische Funktionen

ABS(x)	absoluter Betrag von x
ATN(x)	arctan x, Resultat im Bogenmaß
COS(x)	cos x, x im Bogenmaß
EXP(x)	e^x, x <= 87.3365
INT(x)	Ganzer Teil von x
LN(x)	ln x, x > Ø

PI	PI = 3.14159
SGN(x)	Signumfunktion : -1 für x < Ø Ø für x = Ø 1 für x > Ø
SIN(x)	sin x, x im Bogenmaß
SQR(x)	Quadratwurzel von x, x > Ø
TAN(x)	tan x, x im Bogenmaß

String-Funktionen

INSTR(a$,b$)	Liefert die Position, ab der die Zeichenkette a$ in der Zeichenkette b$ enthalten ist, Ø wenn nicht enthalten
LEFT$(a$,x)	Liefert die ersten x Zeichen von a$
LEN(x$)	Zeichenlänge des Strings x$
MID$(a$,x,y)	y Zeichen von a$, beginnend mit dem x-ten Zeichen von a$
RIGHT$(a$,x)	Liefert die letzten x Zeichen von a$
STRING$(n,a$)	Vervielfacht a$ n-mal
STR$(x)	Formt den Wert x in einen String um
VAL(a$)	Liefert den numerischen Wert von a$

Sonstige Funktionen

ASC(x$)	Liefert den ASCII-Code des ersten Zeichens von x$
AT(zeile,spalte)	Schreibt Printanweisung an bestimmte Stelle des Bildschirms
CHR$(x)	Liefert das Zeichen des ASCII-Codes x
FRE(Variable)	Gibt die Größe des noch freien Anwenderspeichers an
POS(i)	Liefert die aktuelle Schreibposition in der Zeile i
RND(x)	Erzeugt Zufallszahl zwischen Ø und 1
SPC(i)	Erzeugt i Leerzeichen nach letzter Schreibposition auf dem Bildschirm
TAB(spalte)	Setzt aktuelle Schreibposition auf Spalte
USR(x)	Ruft eine Funktion mit Parametervorgabe auf, die als Maschinenprogramm geschrieben ist

VGET$ Liefert den Inhalt auf Cursorposition als String (1 Zeichen)

Operatoren

Priorität	Symbol	Bedeutung
1	( )	Klammern
2	-	Negative Vorzeichen
3	*	Multiplikation
3	/	Division
4	+	Addition
4	-	Subtraktion
5	=	"gleich"
5	<	"kleiner als"
5	>	"größer als"
5	<=	"kleiner oder gleich"
5	>=	"größer oder gleich"
5	<>	"nicht gleich"
7	NOT	Negation ("nicht")
8	AND	Konjunktion ("und")
9	OR	Disjunktion ("oder")

Die logischen Operatoren wirken bitweise auf 16-Bit-Integerzahlen im Bereich von -32768 bis +32767.
Die Vergleichsoperatoren und die Addition als Verknüpfung sind auch auf Strings anwendbar.

Konstanten

Zahlen- und Stringgrößen:
- Zahlenbereich: 9.4Ø396E-39 <= |x| <= 1.7Ø141E+38
- Stringlänge: Ø <= länge <= 255

Bildschirmgröße für Textdarstellung:
- Spaltenanzahl: Ø <= spalte <= 39
- Zeilenanzahl: Ø <= zeile <= 31

Bildschirmgröße für Grafik:
- Horizontalkoordinate: Ø <= x <= 319
- Vertikalkoordinate: Ø <= y <= 255

Farbwerte:

Der Farbcode für den Vordergrund errechnet sich wie folgt:

Farbcode v = 16 * b + f

mit:

f - Code für Vordergrundfarbe

b - Code zum Blinken der Vordergrundfarbe:

b = Ø für Nicht-Blinken

b = 1 für Blinken

Vordergrundfarbe f	Nummer	Hintergrundfarbe h
Schwarz	Ø	Schwarz
Blau	1	Blau
Rot	2	Rot
Purpur	3	Purpur
Grün	4	Grün
Türkis	5	Türkis
Gelb	6	Gelb
Weiß	7	Grau
Schwarz	8	
Violett	9	
Orange	1Ø	
Purpurrot	11	
Grünblau	12	
Blaugrün	13	
Gelbgrün	14	
Weiß	15	

Beispiele weiterer mathematischer Funktionen

(z. B. Programmierung mittels Anwenderfunktionen)

Funktion	Berechnung in BASIC
SEKANS	FNS(x)=1/COS(x)
COSEKANS	FNC(x)=1/SIN(x)
COTANGENS	FNT(x)=1/TAN(x)
ARCUSSINUS	FNA(x)=ATN(x/SQR(1-x*x))
ARCUSCOSINUS	FNA(x)=-ATN(x/SQR(1-x*x))+PI/2
ARCUSCOTANGENS	FNA(x)=ATN(x)+PI/2

Funktion	Berechnung in BASIC
ARCUSSEKANS	FNA(x)=ATN(x/SQR(x*x-1))
ARCUSCOSECANS	FNA(x)=ATN(x/SQR(x*x-1))+(SGN(x)-1)*PI/2
SINUS HYPERBOLICUS	FNH(x)=(EXP(x)-EXP(-x))/2
COSINUS HYPERBOLICUS	FNH(x)=(EXP(x)+EXP(-x))/2
TANGENS HYPERBOLICUS	FNH(x)=EXP(-x)/(EXP(x)+EXP(-x))*2+1
COTANGENS HYPERBOLICUS	FNH(x)=EXP(-x)/(EXP(x)-EXP(-x))*2+1
SEKANS HYPERBOLICUS	FNH(x)=2/(EXP(x)+EXP(-x))
COSEKANS HYPERBOLICUS	FNH(x)=2/(EXP(x)-EXP(-x))
ARCUSSINUS HYPERBOLICUS	FNH(x)=LN(x+SQR(x*x+1))
ARCUSCOSINUS HYPERBOLICUS	FNH(x)=LN(x+SQR(x*x-1))
ARCUSTANGENS HYPERBOLICUS	FNH(x)=LN((1+x)/(1-x))/2
ARCUSCOTANGENS HYPERBOLICUS	FNH(x)=LN((x+1)/(x-1))/2
ARCUSSEKANS HYPERBOLICUS	FNH(x)=LN((SQR(1-x*x)+1)/x)
ARCUSCOSEKANS HYPERBOLICUS	FNH(x)=LN((SQR(1+x*x)+1)/x)*SGN(x)

Liste der Fehlermeldungen

BS	Feldelement außerhalb des dimensionierten Bereichs aufgerufen
DD	Feld mehrfach dimensioniert
FC	Unzulässiger Funktionsaufruf
ID	Fehlerhafte Eingabe im Direktbetrieb (INPUT und DEF sind im Direktbetrieb unzulässig)
MO	Anweisung unvollständig, Operand fehlt
NF	Variablen von NEXT und FOR passen nicht zusammen
OD	Es wurden durch die DATA-Anweisung zuwenig Daten für eine READ-Anweisung spezifiziert
OM	Vorhandener Speicherplatz im RAM reicht für die Ablage bzw. Abarbeitung eines Programms nicht aus
OV	Ergebnis einer Berechnung ist größer als 1.7Ø141E38

SN	Syntaktischer Fehler
RG	RETURN trat vor GOSUB auf
UL	Es wurde eine nicht existierende Zeilennummer angegeben
/Ø	Division durch Null
CN	Programm kann nicht mit CONT fortgesetzt werden
LS	String länger als 255 Zeichen
OS	Vereinbarter Speicherplatz für Strings reicht nicht aus
ST	String zu lang oder zu komplex
TM	Variablen einer Gleichung indizieren verschiedene Typen, z. B. Zahl und String; oder einer Funktion wurde anstatt einer Zahl ein String übergeben oder umgekehrt
UF	Funktion noch nicht definiert
IO	Falscher Name beim Programmladen
BAD	Fehler beim Laden oder Retten von Feldern
EXTRA IGNORED	Zu viele Daten bei INPUT eingegeben
? REDO FROM START	String anstatt numerischer Wert oder umgekehrt bei INPUT eingegeben
??	Zu wenige Daten bei INPUT eingegeben

Literaturverzeichnis

/1/ Müller, S.: Programmieren mit BASIC. Berlin: VEB Verlag Technik 1985.

/2/ Adler, H.; Karl, H.-U.: Die Programmiersprache BASIC. Lehrbrief für das Hochschulfernstudium. Herausgeber: Zentralstelle für das Hochschulfernstudium des Ministeriums für Hoch- und Fachschulwesen Dresden. Bestell-Nr.: Ø2 1566 Ø1 1.

/3/ Werner, D.: BASIC für Mikrorechner. Berlin: VEB Verlag Technik 1986.

/4/ veb mikroelektronik "wilhelm pieck" mühlhausen: Beschreibung zu CØ111 BASIC-Interpreter.

/5/ Schmidt, J.: UP-System·Matrizenrechnung. IS Jena 1985.