

Germann
Jörg
Waldvogel

**Das große
Pocket
Computer
Buch**

SHARP

PC1401

PC1402

PC1421

PC1350

Ein DATA BECKER Buch

ISBN 3-89011-177-7

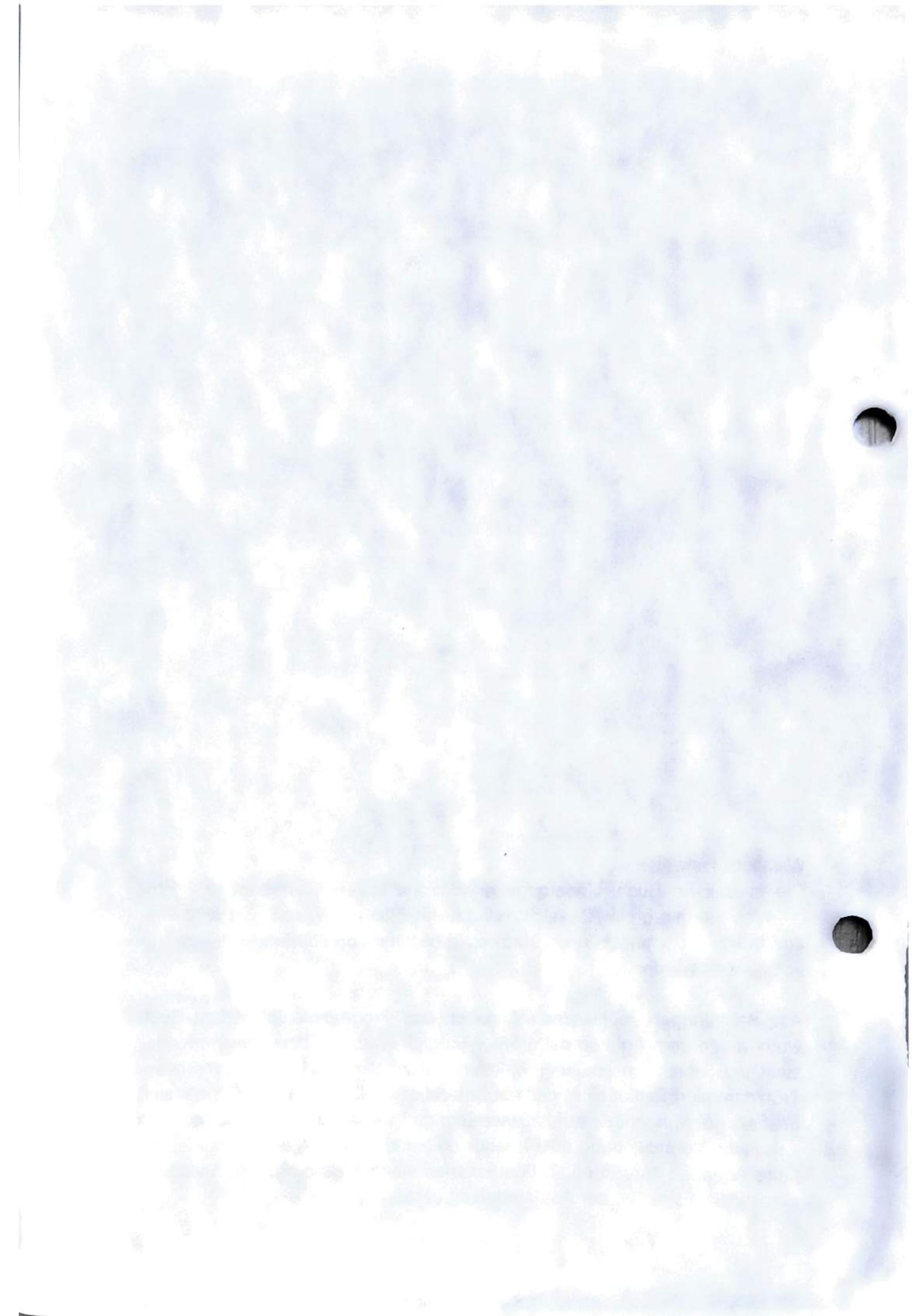
**Copyright © 1995 DATA BECKER GmbH
Marowingerstraße 30
4000 Düsseldorf**

**Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form
(Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung
der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer
Systeme verarbeitet, vervielfältigt oder verbreitet werden.**

Wichtiger Hinweis:

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit großer Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.



Vorwort

Vielleicht ging es Ihnen auch so: Da haben Sie sich einen kleinen programmierbaren Rechner gekauft und mit der Zeit festgestellt, daß er ja noch viel mehr kann, als Sie ursprünglich annahmen.

Sie probierten ein wenig herum; manchmal mit Erfolg, manchmal auch ohne. Langsam fanden Sie sich auch in dieser neuen Welt des BASICs zurecht. Doch irgendwann merkten Sie, daß Sie so nicht mehr weiterkommen würden. Auch wenn Sie sich in BASIC schon recht gut auskannten und von Bits und Bytes einiges verstanden: diese Kenntnisse reichten nicht!

Uns ging es genauso. Wir hatten dieselben Probleme, mit denen auch Sie sich jetzt herumschlagen. Als Einsteiger hat man es nicht leicht. Wir kauften uns damals ein Systemhandbuch und brüteten darüber: Bits, Bytes, HEX-Zahlen, PEEK und POKE wurden einem wild an den Kopf geworfen. Schließlich wollten wir uns auch noch mit Maschinensprache beschäftigen. Aber es war nicht leicht, sich in diesem Chaos von Flags, Registern, Zeigern und Ports zurechtzufinden.

Als wir uns soweit durchgekämpft hatten, entschlossen wir uns, die mühsam erworbenen Kenntnisse auch für andere Einsteiger zugänglich zu machen. Wir faßten den Entschluß, ein Buch zu schreiben; ein Buch in dem wirklich alles Wichtige stehen sollte. Es sollte sowohl von Systeminformationen als auch Maschinensprache handeln. Aber alles so verständlich, daß man ohne schlaflose Nächte oder abgestürzte PCs in diese neue Materie einsteigen konnte und Stück für Stück, Byte für Byte mehr dazulernte. Heute, ein Jahr später, ist es soweit.

Das Buch läßt sich folgendermaßen aufteilen:

Im ersten Teil lernen Sie die Hintergründe von BASIC kennen; außerdem erfahren Sie viel über den Aufbau des PCs und wie er arbeitet. Aber auch, wie ein BASIC-Programm gespeichert wird oder wie Sie direkt auf Speicherzellen zugreifen können. Sie interessieren sich vielleicht auch für spezielle BASIC-Probleme;

wie man z.B. das PASS-Wort knackt oder wie man ein verlorenes Programm rettet, nachdem man versehentlich NEW eingegeben hat.

Im zweiten Teil steigen wir in eine neue, faszinierende Sprache ein: die Maschinensprache. Sie bietet ungleich viel mehr Möglichkeiten als Sie von BASIC her kennen, da sie direkt den Mikroprozessor anspricht. Sie lernen Befehle kennen, mit denen Sie eine Aufgabe bis zu 1000 mal schneller als in BASIC lösen können. Und auch eigene BASIC-Befehle zu kreieren, ist dann für Sie kein Problem mehr.

Und schließlich bekommen Sie im dritten Teil auch noch eine großen Softwaresammlung: BASIC-Programme von finanzmathematischen Anwendungen bis hin zur professionellen Adreßverwaltung. Oder Maschinenprogramme, mit denen Sie auf Ihrem Pocket-Computer Musik spielen oder auf dem Display eigene Zeichen entwerfen können. Auch gute Spiele fehlen nicht: Wer würde nicht gerne einmal Mastermind oder Hangman auf seinem Sharp spielen?

Außerdem liegen auch Anleitungen zum Bau von professioneller Hardware bei. Wenn Sie einmal Lust haben, ein Cassetteninterface oder einen Wecker zu bauen, oder auch externe Geräte zu steuern: alles ist vorbereitet.

Und was Sie vor allem sehen werden: Sie werden mit Ihrem PC nicht nur einen riesigen Schritt weiter kommen, sondern das Ganze macht auch Spaß!

Stephan Germann

Matthias Jörg

Daniel Waldvogel

Inhaltsverzeichnis

Teil 1: BASIC im Detail: Erklärungen, Systeminformationen, Tips und Tricks

1.	Der Speicher: Das Innenleben des PC	14
1.1	Aufbau des Speichers.....	14
1.2	ROM, RAM und CPU	14
1.3	Speicherbelegung	17
1.4	Das High/Lowbyte-System.....	21
1.5	Bits und Bytes.....	23
1.6	Logik mit AND und OR	25
1.7	Spezielle BASIC-Befehle.....	27
1.8	Wie wird ein BASIC-Programm gespeichert.....	32
1.9	ASCII- und Interpretercodes	37
2.	Variablen: Speicherung - eine vielfältige Sache	45
2.1	Die Verwaltung von Variablen.....	45
2.2	Das BCD-System	45
2.3	Die Standardvariablen	48
2.4	Selbstdefinierte Variablen	57
3.	Systemadressen: Die interne Organisation	67
3.1	Interessante Systemadressen.....	67
3.2	Übersicht über die Systemvariablen	67
3.3	Nützliche Maschinenprogrammaufrufe.....	76
3.4	Der BASIC-Interpreter	78
3.5	Der Zeichengenerator.....	87

4.	Tips und Tricks: BASIC professionell	91
4.1	Programmiertips für BASIC-Programme.....	91
4.2	Betriebszustandsanzeigemarken	93
4.3	Der Ein/Ausgabepuffer	97
4.4	Der Tastaturspeicher	100
4.5	Rückgängigmachen von NEW	102
4.6	Ersetzen von Codes	103
4.7	Sonderzeichen	105
4.8	Nützliches über Programmzeilen	107
4.9	Renumber.....	108
4.10	Knacken des PASS-Worts.....	110
4.11	Hochauflösende Grafik.....	113

Teil 2: Einstieg in die Maschinensprache: Die Grenzen von BASIC sprengen

5.	Maschinensprache: Ein erster Überblick	124
5.1	Die CPU: Das Herz des Rechners.....	124
5.2	Externes und internes RAM.....	125
5.3	Die Register	126
5.4	Der CPU-Stack.....	136
5.5	Flags	139
5.6	Maschinensprache und BASIC.....	143
5.7	Mnemonics	143
5.8	Wie man Maschinenprogramme eingibt.....	144
5.9	Der ASSEMBLER - Installation und Bedienung	148
6.	Befehlssatz: Was die CPU so alles kann	167
6.1	Ladebefehle.....	169
6.2	8-Bit-Addition und -Subtraktion	176
6.3	Unterprogramme und der CPU-Stack	180
6.4	Arbeiten mit dem Akku	183
6.5	Sprungbefehle	185
6.6	Vergleichsbefehle und Bittestbefehle.....	194
6.7	Schleifen.....	200

6.8	Bearbeiten ganzer Speicherteile.....	205
6.9	Blockbefehle	210
6.10	16-Bit-Addition und -Subtraktion	219
6.11	Das BCD-System in der Anwendung	223
6.12	Logische Befehle.....	228
6.13	Bit-Verschiebebefehle	233
6.14	Unterprogrammaufruf über Tabellen.....	239
6.15	Sonstige Befehle der CPU	243
7.	Ports: Die Verbindungen zur Außenwelt	249
7.1	Befehle zur Steuerung der Ports.....	250
7.2	Tastaturabfrage über den IA- und IB-Port.....	251
7.3	Der Control-Port (C-Port).....	259
7.4	Der Anschluß externer Geräte	266
7.5	Der TEST-Befehl	273
8.	Insiderwissen: Die Ausnutzung des Interpreters	279
8.1	Die drei Rechnerebenen	279
8.2	Fehlermeldungen, Funktionen und weitere Tricks.....	282
8.3	Eigene BASIC-Befehle.....	286

Teil 3: Soft- und Hardware

9.	Software: Den PC ganz ausnutzen	296
9.1	Mathematik	296
9.2	Finanzmathematik	306
9.3	Datenverarbeitung	310
	- Wörterlernprogramm	311
	- Adreßverwaltung	316
9.4	Spiele	320
	- Mastermind	321
	- Siebzehn und vier.....	323
	- Reaktionstest.....	326
	- Hangman	328

9.5 Utilities - eigene BASIC-Befehle	330
- Dataline	331
- RENEW	333
- MERGE	334
- LINECALC	336
- PSAVE	338
- FIND	342
- SOUND	347
9.6 Grafik: Turbo Graphics	350
9.7 Musik: Soundbox	360
9.8 Wecker	369
10. Hardware: Mit Draht und Lötzinn	381
10.1 Pinbelegung der Schnittstelle	382
10.2 Cassetteninterface	383
10.3 Datenaustauschkabel für zwei Rechner	386
10.4 Elektronisches Relais	387
10.5 Quarzzeitbasis	390

Anhang - Tabellen, Listen, Übersichten

A Maschinenbefehle - eine Übersicht	394
B Mnemonics und Gedankenstützen	399
C Wichtige ROM-Unterprogramme	406
D Druckercodes	412
E Umwandlung dezimal-hexadezimal	414
F Literaturverzeichnis	416

Teil 1: BASIC im Detail - Erklärungen, Systeminformationen, Tips und Tricks

Kapitel 1

Der Speicher: Das Innenleben des PC

In diesem ersten Kapitel möchten wir Ihnen einen groben Überblick über das Innenleben Ihres Sharp Pocket Computers vermitteln. Neben einer Einführung in die Grundbegriffe der Computersprache werden im ersten Teil vor allem die Zusammenhänge aufgedeckt, die mit der Programmiersprache BASIC zu tun haben. Wie man mit Maschinensprache arbeitet, werden Sie im zweiten Teil erfahren. Jetzt geht's zuerst einmal los in die Welt der Bits und Bytes.

1.1 Aufbau des Speichers

Wie verarbeitet unser PC überhaupt Daten, und wie speichert er sie? Ein Computer kann ja bekanntlich nur zwei Zustände unterscheiden: Strom ein (1) und Strom aus (0). Vielleicht fragen Sie sich, wie ein kompliziertes Programm letztlich aus zwei Betriebszuständen aufgebaut ist.

Auf den folgenden Seiten sind diese Zusammenhänge klar aufgezeigt.

1.2 ROM, RAM und CPU

Die PCs von SHARP sind natürlich prinzipiell gleich aufgebaut wie jeder andere Computer. Sie bestehen aus mehreren kompakten Grundbausteinen.

Wie bei einem Lebewesen ein Gehirn vorhanden sein muß, damit seine Existenz gewährleistet ist, so benötigt auch der Computer eine Steuereinheit. Sie muß alle Abläufe kontrollieren und koordinieren. Auf diese Weise sichert sie das 'Überleben' des Computers.

Die Steuereinheit des Computers wird CPU (Central Processing Unit = Zentrale Steuereinheit) genannt. Dieser Mikroprozessor überwacht alle Vorgänge. Er rechnet, steuert, verwertet Tastatureingaben und anderes mehr.

 Der Prozessor ist aber nur das ausführende, steuernde Organ. Um einen Ablauf zu steuern, braucht er erst einmal Daten, die er verarbeiten kann. Ohne 'fremde' Hilfe von außen versteht die CPU nämlich nichts von BASIC, Trigonometrie oder Anzeigesteuerung. Das Wissen zur richtigen Ausführung dieser Dinge erhält sie von einem anderen Chip, dem ROM (Read Only Memory = Nur Lesespeicher). In diesem Chip sind alle Informationen zum Betrieb des Rechners enthalten. Das ROM besteht aus einer langen Kette von Anweisungen, die als Ganzes ein großes und vollständiges Computerprogramm ergeben und der CPU die nötigen Informationen für den Betrieb zur Verfügung stellen. Falls eine Eingabe die Kompetenz dieses Betriebssystems übersteigt, wird eine ERROR-Meldung ausgegeben.

Das Programm des ROM ist in Maschinensprache, der Sprache der CPU, abgefaßt. Es enthält alle notwendigen Routinen zur Verarbeitung Ihres BASIC-Programms, zur Verwertung von Tastatureingaben usw. Dieses grosse Maschinenprogramm wird nach dem Einschalten des Rechners automatisch gestartet. Es besteht aus einer langen Folge von einzelnen Zahlen, die jeweils einen Wert zwischen 0 und 255 besitzen. Eine einzige solche Zahl nennt man ein Byte.

 Der Computer besitzt noch ein drittes, für den Anwender sehr wichtiges Chip, das RAM (Random Access Memory = Wahlfreier Zugriffspeicher oder einfach Schreib- und Lesespeicher). Zwischen RAM und ROM besteht folgender Unterschied: Das ROM ist ein Speicher, dessen Daten nur gelesen werden. Es wird schon bei der Chip-Produktion mit Daten beschrieben, die dann nie mehr verändert werden können. Das RAM dagegen bietet beide Möglichkeiten: Daten können sowohl aus dem Speicher gelesen, als auch in den Speicher geschrieben werden. Während beim ROM die Befehlsfolge fest und unveränderlich ist, können

Daten, die im RAM abgespeichert sind, geändert oder auch gelöscht werden.

Und wo sind wohl Ihre BASIC-Programme gespeichert? Sie haben es erraten: Ihre BASIC-Programme sind natürlich im RAM abgelegt. Sie können Ihr ganzes Programm ja nach Belieben abändern, ergänzen oder auch löschen. Während bei den größeren Computern das RAM beim Ausschalten gelöscht wird, bleibt es beim PC erhalten. Die Pocket Computer von SHARP verwenden nämlich sogenannte batteriegepufferte CMOS-Speicher-Chips. Fließt auch nur ein sehr kleiner Strom, behalten diese Chips ihren Speicherinhalt.

Im RAM sind aber nicht nur Ihre Programme abgelegt, sondern auch die Werte aller Variablen und Systemadressen. Die Daten der Systemadressen geben Auskunft über den aktuellen Betriebszustand des Rechners. Es muß z.B. die Nummer der gerade zu bearbeitenden BASIC-Zeile gespeichert werden, damit sich der Rechner im Gewirr von Daten überhaupt orientieren kann.

Speicherzellen

Sowohl das ROM als auch das RAM besteht, wie wir gesehen haben, aus langen Zahlenfolgen, sogenannten Bytes. Die Gesamtzahl der Bytes des Rechners gibt immer an, wieviele Zeichen maximal im Speicher abgelegt werden können, d.h. wieviele Speicherzellen der Computer überhaupt besitzt. Der BASIC-Befehl 'MEM' gibt Auskunft darüber, wieviele Speicherzellen des BASIC-Speichers noch nicht von Programmen belegt sind.

Eine Speicherzelle entspricht nun eben gerade einem solchen Byte. So wie im BASIC-Programm die Zeilen zu deren Unterscheidung verschieden numeriert sind, müssen auch die einzelnen Bytes des Rechners unterschieden werden können. Jedes Byte besitzt deshalb eine spezifische Adresse. Über diese Adresse kann dann die entsprechende Speicherzelle mit speziellen

BASIC-Befehlen, die wir in Kapitel 1.7 noch behandeln werden, angesprochen werden.

Die Bytes Ihres Rechners sind von 0 bis 65535 durchgehend numeriert (wobei aber gewisse Speicherbereiche doppelt oder mehrfach adressiert sind). Aus dem folgenden Speicherbelegungsplan erhalten Sie eine grobe Übersicht über die Funktionen der einzelnen Speicherzellen.

1.3 Speicherbelegung

Die folgenden Tabellen zeigen die grobe Verteilung der Speicherzellen und deren Belegung mit verschiedenen 'Aufgaben'.

Anmerkung: 1K = 1024 Bytes

PC 140X:

0-8191	Internes ROM (8K)
8192-14335	PC 1402: BASIC-Speicher, PC 1401: Doppelbelegungen
14336-17871	BASIC-Speicher
17872-18079	Standardvariablen
18080-32767	Systemvariablen, Mehrfachbelegungen und nicht belegte Adressbereiche
32768-65535	Externes ROM (32K)

PC 1421:

0-8191	Internes ROM (8K)
8192-14335	Doppelbelegungen
14336-17791	BASIC-Speicher
17792-18079	Standardvariablen (auch i,n usw.)
18080-32767	Systemvariablen, Mehrfachbelegungen und nicht belegte Adressbereiche
32768-65535	Externes ROM (32K)

PC 126X:

0-8191	Internes ROM (8K)
8192-16383	Systemvariablen, Mehrfachbelegungen
16384-22527	PC 1261: BASIC-Speicher, PC 1260: Mehrfachbelegungen
22528-25855	BASIC-Speicher
25856-26063	Standardvariablen
26064-32767	Systemvariablen, Mehrfachbelegungen
32768-65535	Externes ROM (32K)

*PC 1350:*

0-8191	Internes ROM (8K)
8192-16383	RAM-CARD, erste 8 K (20K RAM: BASIC-Speicher, 12K RAM: Mehrfachbelegung)
16384-24575	RAM-CARD, zweite 8 K
24576-27695	BASIC-Speicher (4K RAM: Grundausstattung)
27696-27903	Standardvariablen
32904-32767	Systemvariablen, Mehrfachbelegungen
32768-65535	Externes ROM (32K)

Anfang des BASIC-Speichers:

- A1 Beginn der Systemvariablen (48 Bytes)**
A2 Beginn des BASIC-Speichers

Adressen für die verschiedenen Modelle:

Kapazität RAM	A1	A2
4 K	24576	24624
12 K	16384	16432
20 K	8192	8240

Wichtig: Die Systemvariablen direkt vor dem BASIC-Speicher (A1 bis A2) dürfen nicht verändert werden; andernfalls führt der Rechner einen ALL RESET aus.

Verschiedene Speicherbereiche des Rechners, z.B. die Systemvariablen, sind mehrfach abgespeichert. Manchen Leser wird das sicher verwundern. Auf den ersten Blick könnte man glauben, daß gerade der PC 1401 oder PC 1260 durch einfaches Umlöten von ein paar Drähten auf 24K RAM (24K RAM, 40K ROM) ausgebaut werden könnte. In der Tat aber besitzen diese Rechner in der Grundversion nicht ein schlecht ausgenutztes 24K-RAM, sondern nur zwei 2K-RAM Chips. Die CPU kann aber volle 64K adressieren und ansteuern. Da jedoch im Gesamten nur 4K RAM vorhanden sind, haben die meisten Speicherzellen mehrere, verschiedene Adressen erhalten.

In den folgenden Kapiteln werden wir von vielen dieser Speicherzellen im Direktzugriff Gebrauch machen.

Der Aufbau des RAM

Was uns in diesem ersten Teil des Buches vor allem interessieren wird, das sind die Speicherzellen des RAM. Es gibt spezielle BASIC-Befehle, mit denen wir diese Speicherzellen verändern können. Deshalb sind im Folgenden Übersichten speziell für die RAM-Bereiche abgedruckt.

Man unterteilt das RAM in drei Teilbereiche:

Der BASIC-Programmspeicher

Modell	BASIC-Anfang	BASIC-Ende	BASIC-Speicher (Anzahl Bytes)
PC 1401:-	14336	17871	3534
PC 1402:	8192	17871	9678
PC 1421:	14336	17791	3454

PC 1260:	22656	25855	3198
PC 1261:	16512	25855	9342
(nach EQU#=0)			

PC 1350:

4K RAM:	24624	27695	3070
12K RAM:	16432	27695	11262
20K RAM:	8240	27695	19454

Wie aus dieser Darstellung ersichtlich ist, wird das BASIC-Programm im unteren Teil des Programmspeichers abgelegt und zu den höheren Adressen hin erweitert. Selbstdefinierte Variablen beschneiden den BASIC-Speicher, indem sie im oberen Teil des Speichers Speicherzellen beanspruchen. Die Differenz zwischen dem Ende des BASIC-Programms und dem Anfang der selbstdefinierten Variablen ergibt dann die Anzahl freier Bytes, die mit der Funktion 'MEM' abgefragt werden kann. Nur diese freien Bytes stehen dem Anwender tatsächlich noch für eigene BASIC-Programme zur Verfügung.

Der StandardvariablenSpeicher

Adresse				
PC 140X	PC 1421	PC 126X	PC 1350	Bemerkung
17872	17892	25856	27696	Z=A(26) Z\$=A\$(26)
208	208	208	208	belegte Bytes
18079	17999	26063	27903	A=A(1) A\$=A\$(1)

Für die Standardvariablen ist im Gegensatz zu den selbstdefinierten Variablen ein fester Platz im RAM reserviert. Nach welchem System diese Variablen abgespeichert werden, erfahren Sie in Kapitel 2.

Der SystemvariablenSpeicher

Mit den Systemvariablen werden wir uns noch genauer befassen. Durch eine direkte Veränderung dieser Bytes können viele interessante Effekte hervorgerufen werden; z.B. das Löschen des PASS-Wortes. Mehr dazu in Kapitel 4.

1.4 Das High/Lowbyte-System

Wie wir bereits gesehen haben, kann eine Speicherzelle (ein Byte) einen Wert zwischen 0 und 255 fassen. Wir wissen aber, daß der Rechner auch sehr viel größere Zahlen speichern kann. Die Speicheradressen selbst (ROM und RAM) gehen ja schon von 0 bis 65535.

Damit Zahlen, die größer als 255 sind, überhaupt dargestellt werden können, müssen sie mittels bestimmter Zahlensysteme codiert und auf mehrere Bytes verteilt werden. Unser Rechner besitzt zwei solche Zahlensysteme:

- Das High/Lowbyte-System, für ganze Zahlen bis 65535 geeignet, findet zum Beispiel bei der Codierung der Zeilenummern seine Anwendung. Je zwei Bytes zusammen codieren eine Zahl zwischen 0 und 65535.
- Das BCD-System, das bei den numerischen Variablen benötigt wird, codiert mit je acht Bytes eine Zahl zwischen $-9.99 \cdot 10^9$ und $9.99 \cdot 10^9$. Wir werden das BCD-Zahlensystem in Kapitel 2.2 gründlich unter die Lupe nehmen.

Nun zum High/Lowbyte-System. Nach diesem System werden die ganzen Zahlen zwischen 0 und 65535 bearbeitet. Wie geht dies vor sich?

Der Rechner nimmt eine Unterteilung der Zahl in High- und Lowbyte vor:

Beispiel:

7325 HB LB

$$7325 = 256 * 28 + 157$$

Die Zahl 7325 wird in 28 Zweihundertsechsundfünfziger (HB) und 157 Einer (LB) aufgeteilt.

Das Highbyte: 28 (256er) $256 * 28 = 7168$

Das Lowbyte: 157 (1er) $1 * 157 = \underline{157}$

7325

Mit diesem System kann als höchste Zahl $256 * 255 + 255 = 65535$ gespeichert werden.

Das High/Lowbyte-System wird z.B. für die Codierung der BASIC-Zeilenummern oder des WAIT-Intervalls (z.B. WAIT 3000) usw. verwendet. Zum Auslesen der verschiedenen Systemvariablen (Kapitel 3.1) werden wir noch häufig von diesem Zahlensystem Gebrauch machen. In den Tabellen der folgenden Kapitel sind die Speicherzellen, die das Highbyte einer Zahl enthalten, mit HB gekennzeichnet, das Lowbyte besitzt die Abkürzung LB.

Noch einmal: Um den wirklichen Wert einer Zahl zu erfahren, die im High/Lowbyte-System gespeichert ist, rechnen Sie so:

$$\text{Zahl} = \text{HB} * 256 + \text{LB}$$

1.5 Bits & Bytes

Ganz am Anfang haben wir festgestellt, daß der Computer eigentlich nur zwei Betriebszustände kennt: Strom ein (1) und Strom aus (0). Folglich ist auch der Zahlenwert eines Bytes (0 bis 255) bereits eine codierte Zahl. Irgendwie muß der Computer eine solche Zahl ja in seiner eigenen 'Sprache', Strom ein/Strom aus, verstehen.

 Nehmen wir zum Beispiel die Zahl 147. Wie kann sie mit einzelnen Ja/Nein-Entscheidungen (0 oder 1) dargestellt werden? Die folgende Darstellung zeigt, wie die Aufteilung in 0 und 1 und somit in Bits zustande kommt:

Bit Nr.	7	6	5	4	3	2	1	0
Wertigkeit:	1	0	0	1	0	0	1	1
Auswertung:	2 ⁷			+2 ⁴			+2 ¹	+2 ⁰ = <u>147</u>

 Die Darstellung zeigt, daß jedes Byte in 8 Bits (Bits 0 bis 7) unterteilt ist. Jedes dieser Bits kann nur die zwei Werte 0 oder 1 speichern. Werden 8 Bits zu einem Byte zusammengefaßt, so können damit 2^8 (=256) verschiedene Zahlen dargestellt werden. Bedingung ist natürlich auch hier, daß ein bestimmtes Bit von jedem andern Bit unterschieden werden kann. Jedes Bit des gesamten Speicherbereichs (65535 Bytes) muß eindeutig ange- sprochen werden können. Die 'Lage' eines bestimmten Bits wird deshalb durch die Adresse seines Bytes, in dem es mit 7 anderen Bits zusammengefasst ist, und durch seine eigene Bitnummer (0 bis 7) innerhalb seines Bytes eindeutig festgelegt. Damit nun also die Zahl 147 dargestellt werden kann, müssen die Bits 7 (2^7), 4 (2^4), 1 (2^1) und 0 (2^0) den Wert 1 erhalten.

Mit den für die Maschinensprache benutzen BASIC-Befehlen, die wir in Kapitel 1.7 ausführlich behandeln werden, können die Werte der einzelnen Bytes und Bits gelesen und verändert wer-

den. Die nächste Darstellung zeigt, warum ein Byte als Maximum die Zahl 255 speichern kann:

Bit Nr.	7	6	5	4	3	2	1	0	
Wertigkeit:	1	1	1	1	1	1	1	1	
Auswertung:	27	+26	+25	+24	+23	+22	+21	+20	= <u>255</u>

Erstaunlich einfach ist auch die Bitbelegung für eine Zahl, die im High/Lowbyte-Format dargestellt wird. Sie wird bekanntlich durch 2 Bytes oder $2^8 = 16$ Bits dargestellt. Anhand eines weiteren Beispiels wollen wir die Codierung der High/Lowbyte-Zahl 7325 betrachten:

Highbyte:

Bit Nr.	7	6	5	4	3	2	1	0
oder:	15	14	13	12	11	10	9	8
Wertigkeit:	0	0	0	1	1	1	0	0
Auswertung:				212	+211	+210		= <u>7168</u>

Lowbyte:

Bit Nr.	7	6	5	4	3	2	1	0
Wertigkeit:	1	0	0	1	1	1	0	1
Auswertung:	27			+24	+23	+22	+20	= <u>157</u>

Zusammen:

$$7168 + 157 = \underline{\underline{7325}}$$

Die Bits des Highbytes werden, im Anschluß an die Bitnummern des Lowbytes, ganz einfach weiter numeriert:

- Bitnummern des Lowbytes: 0 bis 7
- Bitnummern des Highbytes: 8 bis 15 oder (0 bis 7)+8

1.6 Logik mit AND und OR

Da wir nun wissen, daß der Rechner im Grunde genommen aus einzelnen Bits ('Ja/Nein-Speichern') aufgebaut ist, können wir auch besser verstehen, wie die BASIC-Befehle AND und OR funktionieren, denn auch diese Befehle arbeiten nur mit den beiden Werten 0 und 1.

AND-Verknüpfung zweier Bits

A	B	
0	AND	0 -> 0
1	AND	0 -> 0
0	AND	1 -> 0
1	AND	1 -> 1

Nur wenn Bit A und Bit B gesetzt sind, erscheint als Resultat eine Eins.

Beispiel: 153 AND 165

Bit Nr.	7	6	5	4	3	2	1	0
<hr/>								
1. Byte Wertigkeit:	1	0	0	1	1	0	0	1
= 153								
AND								
2. Byte Wertigkeit:	1	0	1	0	0	1	0	1
= 165								
<hr/>								
AND-Verknüpfung:	1	0	0	0	0	0	0	1
<hr/>								
Auswertung:	27						+20 = <u>129</u>	
<hr/>								

Geben Sie dieses Beispiel im PC ein: 153 AND 165

Ergebnis: 129

OR-Verknüpfung zweier Bits

A	B	
0	OR	0 -> 0
1	OR	0 -> 1
0	OR	1 -> 1
1	OR	1 -> 1

Wenn Bit A oder Bit B gesetzt ist, erscheint als Resultat eine Eins.

Beispiel: 153 OR 165

Bit Nr.:	7	6	5	4	3	2	1	0
1. Byte	1	0	0	1	1	0	0	1
Verteiligkeit:	1	0	0	1	1	0	0	1
2. Byte								0E
Verteiligkeit:	1	0	1	0	0	1	0	1
0X-Verknüpfung:	1	0	1	1	1	1	0	1
Auswertung:	z^7	$\cdot z^6$	$\cdot z^5$	$\cdot z^4$	$\cdot z^3$	$\cdot z^2$	$\cdot z^1$	$\cdot z^0$
								= 189

Geben Sie dieses Beispiel im PC ein: 153 OR 165
Ergebnis: 189

Mit diesen Befehlen ist es uns nun möglich, einzelne Bits direkt zu setzen oder zu löschen.

Wenn z.B. Bit 5 gesetzt werden soll, während alle andern Bits unverändert bleiben, müssen Sie den alten Bytewert mit OR 32 (25) verknüpfen. Oder soll Bit 3 gelöscht werden, ohne daß die restlichen Bits ihren Zustand ändern, so ist der alte Bytewert mit AND 247 (255 - $2^3 = 247$) zu verknüpfen.

Diese Beispiele mögen fürs erste sehr theoretisch und kompliziert erscheinen. Die Befehle AND und OR können uns jedoch bei der Programmierung von hochauflösender Grafik von großem Nutzen sein.

1.7 Spezielle BASIC-Befehle

Vielleicht ist es Ihnen schon bekannt: Fünf BASIC-Befehle Ihres Rechners bilden die absolute Grundlage für das direkte Arbeiten mit Speicherzellen und Maschinensprache. Diese Befehle haben es in sich!

Ohne sie wäre es nicht möglich, die Werte einzelner Bytes direkt zu verändern. Auch die ganze Maschinensprachprogrammierung könnten wir vergessen, würden diese Befehle nicht existieren.

Im Folgenden wird die Funktionsweise dieser BASIC-Befehle anhand von Beispielen erklärt. Vor allem in Kapitel 3.1 ('Interessante Systemadressen') werden wir dann sehen, wieviel man mit diesen Befehlen aus dem Rechner holen kann. Alle Befehle funktionieren sowohl im PRO-, als auch im RUN-Modus.

PEEK Byteadresse

Mit dem PEEK-Befehl haben Sie die Möglichkeit, den Inhalt einer bestimmten Speicherzelle auszulesen. Mit der auf den Befehl folgenden Byteadresse geben Sie an, welches der 65536 Bytes gelesen werden soll.

Syntax:

```
PEEK x      (0 <= x <= 65535)
            (x = Byteadresse)
```

POKE Byteadresse, Wert (,Wert,Wert,...)

POKE speichert den angegebenen Wert in das Byte mit der angegebenen Byteadresse. Dem POKE-Befehl müssen also mindestens zwei Zahlen folgen, die durch ein Komma voneinander getrennt werden.

Syntax:

```
POKE x,y    (0 <= x <= 65535; 0 <= y <= 255)
            (x = Byteadresse, y = Wert)
```

Nach der Byteadresse des POKE-Kommandos können aber auch mehrere Werte gleichzeitig angeknüpft werden, wenn ganze Zahlenfolgen in den Speicher gePOKEd werden sollen. Die einzelnen Werte werden durch Komma voneinander getrennt.

Syntax:

```
POKE x,y,z,... (0 <= x <=65536)
  (x = Byteadresse, y=Wert 1
   z = Wert 2)
```

y wird in der Speicherzelle x abgelegt, z in der Speicherzelle (x+1) usw.

CALL Byteadresse

Der CALL-Befehl ruft ein Maschinenprogramm auf, das an der angegebenen Byteadresse beginnt. Damit kann man Maschinenprogramme aufrufen, die man vorher mit POKE eingegeben hat.

Syntax:

```
CALL x      (0 <= x <= 65535)
  (x = Byteadresse)
```

Beispiele zu diesen drei Befehlen:

Schalten Sie den Rechner ein und schalten Sie in den RUN-Modus. Geben Sie ein: Z\$="A". Nun geben Sie ein:

PC 140X:	PEEK 17873
PC 1421:	PEEK 17793
PC 126X:	PEEK 25857
PC 1350:	PEEK 27697

Auf der Anzeige erscheint die Zahl 65. Da wir das erste Zeichen der Variable Z\$ auslesen, kann man sagen, daß "A" den Code 65 hat. Wir können mit der nächsten Eingabe probieren, welches Zeichen für den Code 66 erzeugt wird:

PC 140X:	POKE 17873,66
PC 1421:	POKE 17793,66

PC 126X: POKE 25857,66
PC 1350: POKE 27697,66

Wenn Sie nun Z\$ nachprüfen, werden Sie sehen, daß diese Variable einen neuen Inhalt hat: Z\$="B".

Noch ein Beispiel zum CALL-Befehl. Geben Sie nun CALL 49321¹⁾ ein. Der Rechner gibt einen BEEP von sich. An der Byteadresse 49321¹⁾ beginnt nämlich das Maschinenprogramm des BEEP-Befehls. Diese Routine wird intern aufgerufen, wenn Sie in BASIC den Befehl BEEP eingeben. Mit der für uns neuen Anweisung CALL 49321¹⁾ können wir diese Routine direkt anspringen.

Mit den beiden Befehlen POKE und CALL können Sie Ihren Rechner manipulieren. Darum dürfen POKE und CALL nur gezielt eingesetzt werden. Ein unüberlegter, zufälliger Einsatz kann zum Absturz des Computers führen.

CSAVE M ("Name");Startadresse, Endadresse

Der normale CSAVE-Befehl zur Abspeicherung von BASIC-Programmen auf Cassette ist Ihnen sicher bekannt. Das Betriebssystem Ihres Rechners kennt aber auch noch eine Variation davon: den CSAVE M-Befehl. Mit diesem Befehl haben Sie die Möglichkeit, Maschinenprogramme oder ganze Speicherteile auf Band zu speichern.

Der durch Start- und Endadresse festgelegte Speicherbereich wird auf Cassette gespeichert. Diesem Datenblock können Sie zusätzlich noch einen Namen geben.

1) PC 1421: 55539, PC 126X: 48908, PC 1350: 49430

Syntax:

```
CSAVE M x,y      (x > y; 0 <= x,y <= 65535)
                  (x = Startadresse,
                   y = Endadresse)

CSAVE M "Name";x,y  (Speicherung mit Name)
```

Beispiel: CSAVE M "BSP1";25900,25910

Die Inhalte der Speicherzellen mit den Adressen 25900 bis 25910 werden der Reihe nach auf Cassette gespeichert. Der Name des Datenblocks, in unserem Fall 'BSP1', wird auch auf der Cassette abgelegt.

CLOAD M ("Name:") Startadresse

Der CLOAD M-Befehl ist das Gegenstück zum CSAVE M-Befehl. Mit CLOAD M können Daten, die zuvor mit CSAVE M abgespeichert wurden, wieder in den Speicher geladen werden.

CLOAD M lädt einen Datenblock von Cassettenband an die angegebene Startadresse.

Syntax:

```
CLOAD M x          (0 <= x <= 65535)
                  (x = Startadresse)

CLOAD M "Name";x    (Laden mit Namen)

CLOAD M "Name"      (Laden mit Namen, das Programm
                  wird in den gleichen Speicher-
                  bereich geladen, von dem aus
                  es gespeichert wurde)
```

Beispiel: CLOAD M "BSP1"

Der Datenblock mit dem Namen "BSP1" wird geladen und automatisch in den Speicherbereich zurückgeladen, von dem aus er schon gespeichert wurde. Das erste Byte des Datenblocks auf Cassette wird in Speicherzelle 25900 abgelegt, das zweite in Speicherzelle 25901 usw.

Mit den beiden Befehlen CSAVE M und CLOAD M können Sie Ihre eigenen Maschinenprogramme speichern und laden.

I.5 Wie wird ein BASIC-Programm abgespeichert?

In diesem Kapitel wollen wir anhand eines Beispiels betrachten, wie überhaupt ein BASIC-Programm im Speicher abgelegt wird. Wir editieren das folgende BASIC-Programm nicht wie gewohnt mit dem LIST-Kommando, sondern wir wollen die Werte der entsprechenden Speicherzellen, die durch das Programm belegt werden, mit dem PEEK-Befehl auslesen und die ausgelesenen Zahlen untersuchen. Folgendes Programm soll untersucht werden:

```
100:INPUT S0  
200:FOR I=1 TO 25  
300:PRINT I  
400:NEXT I:END
```

Löschen Sie den Speicher Ihres Rechners zuerst mit NEW (beim PC 126X löschen Sie auch noch den ESP-Speicher mit NEW#EQU#=0). Geben Sie dann dieses kurze Programm ein. Da wegen der Vielfalt der Anfangsadressen nicht Adresspalten für alle Rechner aufgeführt werden können, werden diese Adressen mit der Standardvariablen A angegeben. Für die einzelnen Rechnermodelle geben Sie für A bitte folgende Werte ein:

PC 1401/21: A = 14336
PC 1402: A = 8192

PC 1260: A = 22656
PC 1261: A = 16512

PC 1350:
4K RAM A = 24624
12K RAM A = 16432
20K RAM A = 8240

Lesen Sie die Werte unseres Beispiels auch selbst mit PEEK (A+n) auf Ihrem Rechner aus. Sie werden sehen, daß die angegebenen Codes mit denen Ihres Rechners übereinstimmen.

Adresse	Wert	Bedeutung
(A)	255	255 bedeutet: Programmanfang
(A+1)	HB 0	Highbyte 1. Zeilennummer: $0 * 256 = 0$
(A+2)	LB 100	Lowbyte 1. Zeilennummer: $100 * 1 = \underline{100}$ 100
(A+3)	4	Länge der 1. Zeile in Bytes
		Startadresse der nächsten Zeile: $(A+3)+4+1$
(A+4)	197	WAIT: Interpretercode für WAIT
(A+5)	53	5 ASCII-Code für '5'
(A+6)	48	0 ASCII-Code für '0'
(A+7)	13	Ende der ersten Programmzeile
(A+8)	HB 0	Highbyte 2. Zeilennummer: $0 * 256 = 0$
(A+9)	LB 200	Lowbyte 2. Zeilennummer: $200 * 1 = \underline{200}$ 200
(A+10)	8	Länge der 2. Zeile in Bytes
		Startadresse der nächsten Zeile: $(A+10)+8+1$
(A+11)	213	FOR Interpretercode für FOR
(A+12)	73	I ASCII-Code für 'I'
(A+13)	61	= ASCII-Code für '='

(A+14)	49	1	ASCII-Code für '1'
(A+15)	208	TO	Interpretercode für TO
(A+16)	50	2	ASCII-Code für '2'
(A+17)	53	5	ASCII-Code für '5'
(A+18)	13		Ende der 2. Programmzeile
<hr/>			
(A+19)	HB	1	Highbyte 3. Zeilennummer: $1 * 256 = 256$
(A+20)	LB	44	Lowbyte 3. Zeilennummer: $44 * 1 = \underline{44}$ 300
(A+21)		3	Länge der 3. Zeile in Bytes Startadresse der nächsten Zeile: (A+21)+3+1
(A+22)		222	PRINT Interpretercode für PRINT
(A+23)		73	I ASCII-Code für 'I'
(A+24)		13	Ende der 3. Programmzeile
<hr/>			
(A+25)	HB	1	Highbyte 4. Zeilennummer: $1 * 256 = 256$
(A+26)	LB	144	Lowbyte 4. Zeilennummer: $144 * 1 = \underline{144}$ 400
(A+27)		5	Länge der 4. Zeile in Byte Startadresse der nächsten Zeile: (A+27)+5+1
(A+28)		217	NEXT Interpretercode für NEXT
(A+29)		73	I ASCII-Code für 'I'
(A+30)		58	:
(A+31)		216	END Interpretercode für END
(A+32)		13	Ende der 4. Zeile
<hr/>			
(A+33)		255	255 bedeutet: Ende des Programms

Aus unserem Beispiel können wir schließen:

Ein BASIC-Programm beginnt und endet mit der Zahl 255.

Der eigentliche Programmtext steht zwischen der ersten und der zweiten 255.

Die ersten beiden Bytes einer Programmzeile enthalten die Zeilennummer. Diese Zeilennummer wird im High/Lowbyte-Format codiert. Das erste Byte ist das Highbyte, das zweite das Lowbyte. Die Zeilennummer läßt sich also wie folgt berechnen:

$$1. \text{ Byte} * 256 + 2. \text{ Byte} = \text{Zeilennummer}$$

Auf die Zeilennummer folgt die Zeilenlänge dieser Zeile in Bytes.

Erst jetzt erscheint der eigentliche Text dieser Programmzeile, der nach bestimmten Zeichencodetabellen (nächstes Kapitel) codiert wird. Jedem Befehl, jeder Zahl, jedem Buchstaben und jedem Sonderzeichen ist eine bestimmte Zahl, ein Code, zugeordnet.

Da der Computer eigentlich nur Zahlen speichern kann, müssen alle Zeichen und Befehle in Form von Zahlen abgespeichert sein. Eine vollständige Tabelle dieser Zeichencodes finden Sie im nächsten Kapitel.

Alle BASIC-Befehlsworte werden durch eine einzige Zahl codiert. Ein Befehl wird also nicht Buchstabe für Buchstabe gespeichert, sondern dem Befehl als Ganzes wird ein Code zugeordnet. PRINT z.B. nimmt im BASIC-Speicher nicht 5 Bytes in Anspruch, sondern wird lediglich durch den Code 222 dargestellt. Auf diese Art und Weise kann sehr viel Speicherplatz eingespart werden. Auch die Bearbeitungsgeschwindigkeit wird dadurch erhöht, da weniger Zeichen gelesen werden müssen.

Das Ende einer Programmzeile wird durch die Zahl 13 dargestellt.

Weiter ist zu bemerken, daß die Anfangsadresse des BASIC-Programms in den Speicherzellen 18145/18146¹⁾ im High/Lowbyte-System gespeichert ist. In der Speicherzelle 18145¹⁾ steht das

1) PC 126X: 26337/38, PC 1350: 28417/18

Low-, in 18146¹⁾ das Highbyte. Die Endadresse des BASIC-Programms steht in den Speicherzellen 18147/18148²⁾. Die aktuellen Anfangs- und Endadressen eines BASIC-Programms lassen sich dann wie folgt berechnen:

$$\text{Anfangsadresse} = \text{PEEK } 18145\text{ }^1) + \text{PEEK } 18146\text{ }^1) * 256$$

$$\text{Endadresse} = \text{PEEK } 18147\text{ }^2) + \text{PEEK } 18148\text{ }^2) * 256$$

Lesen Sie diese Werte aus und vergleichen Sie sie mit der Tabelle des aufgelisteten BASIC-Programms, das wir eben betrachtet haben.

Beispiel:

Der PRINT-Befehl in Zeile 300 soll in einen LPRINT-Befehl umgewandelt werden, damit die Ausgabe auf den Drucker erfolgt. PRINT wird mit der Zahl 222 codiert, LPRINT besitzt den Code 226. Wenn Sie also den Code 222 durch 226 ersetzen, sollte danach in der Zeile 300 nicht mehr 300:PRINT I, sondern 300:LPRINT I erscheinen.

Wir probieren das gleich aus:

Ersetzen Sie den Code für PRINT durch die Eingabe von:

POKE A+22,226

Durch diesen POKE-Befehl wird die Speicherzelle A+22, wo bisher 222 gespeichert war, auf den Wert 226 gesetzt. Wenn Sie das Programm mit LIST editieren, erscheint anstelle von PRINT der LPRINT-Befehl.

1) PC 126X: 26337/38, PC 1350: 28417/18

2) PC 126X: 26339/40, PC 1350: 28419/20

Mit Hilfe der Zeichencodetabelle im nächsten Kapitel können Sie noch andere Änderungen in Ihrem Programm bewirken. Sie können z.B. Sonderzeichen (z.B. das Hochkomma), die über die Tastatur nicht abrufbar sind, in PRINT-Anweisungen einfügen.

1.9 ASCII- und Interpretercodes

In diesem Kapitel folgt eine Gesamtübersicht über die einzelnen Zeichen und wie sie vom Rechner codiert werden.

Alle Buchstaben, Ziffern und Satzzeichen werden (mit gewissen Abweichungen) nach einem internationalen Schlüssel codiert. Die Codes dieses internationalen Schlüssels nennt man ASCII-Codes (von 'American Standard Code for Information Interchange', 'Amerikanischer Standardcode für Informationsaus tausch').

Eigentlich handelt es sich hierbei um eine amerikanische Codierung, aber dieser Zeichencode wird heute international und so auch in den nicht amerikanischen Geräten zur Darstellung von Information benutzt. Wie bereits gesagt wurde, benutzen auch die SHARP Pocket-Computer diesen ASCII-Code für Buchstaben, Ziffern und Satzzeichen.

Die Sonderzeichen und BASIC-Befehle werden nach einem andern Schlüssel codiert. Fast jeder Computer hat für seine BASIC-Befehle eigene Codes, die dann vom BASIC-Interpreter bearbeitet und (de-)codiert werden. Die Codes der BASIC-Befehle werden deshalb Interpretercodes genannt.

Alle BASIC-Befehle werden mit einer einzigen Zahl codiert. Einen solchen Code eines BASIC-Befehls nennt man wie gesagt Interpretercode oder auch Token (das Token des Befehls LIST zum Beispiel heißt 180). Wenn Sie in Ihren Rechner eine Programmzeile eingeben und die Eingabe mit ENTER abschliessen, so werden sofort alle Befehle (z.B. PRINT) und Funktionen (z.B. SIN) in Tokens umgewandelt. Befehle mit Namen, die aus meh-

eren Buchstaben bestehen, verbrauchen so nur eine einzige Speicherzelle, ein Byte im Programmspeicher.

Eine gesamte ASCII- und Interpretercodetabelle ist im Rechner-ROM abgespeichert, damit der Computer die Zeichen, die ihm eingegeben werden, sofort selbst codieren kann.

Der Drucker CE-126P kennt keine BASIC-Tokens. Bei der Erstellung eines Listings (LLIST) muß daher ein BASIC-Token in die einzelnen Zeichen zerlegt werden, z.B. das Wort 'TAN' in die einzelnen Buchstaben 'T', 'A' und 'N'. Diese Buchstaben werden dann als ASCII-Codes ausgegeben.

Zeichencodetabellen

Hier finden Sie eine Tabelle der ASCII-Codes und Sonderzeichen. Diese Tabellen sind nach den verschiedenen Rechnermodellen geordnet. Suchen Sie deshalb zuerst die richtige Tabelle für Ihren Rechner.

ASCII- und Interpretercodetabelle füer den PC-140X

ASCII-Codes:

Interpretercodes:

0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	SPC		0	0	P	96	~	~	FACT	RND	RUN	RANDOM	TD	60SUB	~
1	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241
		!	1	A	Q	~	~	REC	LN	AMD	MEM	DEGREE	STEP	AREAD	~
2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242
C-CE	*		2	B	R	~	~	POL	LOG	DR	CONT	RADIAN	THEM	LPRINT	~
3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243
CA		0	3	C	S	~	~	ROT	EXP	MOT	PASS	GRAD	ON	RETURN	~
4	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244
+		\$	4	D	T	~	~	DEC1	SQR	ASC	LIST	EEP	IF	RESTORE	~
5	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245
+	Z	5	E	U	~	~	~	HEX	SIN	VAL	L LIST	WAIT	FOR	~	~
6	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246
P6+NP	*	6	F	V	~	~	~	TEN	COS	LEM	CSAVE	GOTO	LET	~	~
7	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247
	*	7	6	W	~	~	~	RCP	TAN	PEEK	CLOAD	TRON	REM	~	~
8	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248
BASIC	(B	H	X	~	~	~	SQU	INT	CHR\$	~	TROFF	END	~	~
9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249
CAL)	9	I	Y	~	~	~	CUR	ABS	STRG	~	CLEAR	NEXT	~	~
10	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250
	*	:	J	Z	~	~	~	HSM	SGN	MIDS	~	USING	STOP	~	~
11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251
INS	+	1	K	L	~	~	~	HCS	DEG	LEFT\$	~	DIM	READ	~	TT
12	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252
DEL	,	<	L	Y	~	~	~	HTN	DMS	RIGHT\$	~	CALL	DATA	~	✓
13	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253
ENTER	-	=	M	J	~	~	~	AHS	ASM	INKEY\$	~	POKE	PAUSE	~	□
14	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254
►	*	>	N	^	~	~	~	AHC	ACS	PI	~	~	PRINT	~	■
15	31	47	63	79	95	111	127	143	159	175	191	207	223	239	255
	/	?	0	-	~	~	~	AHT	ATH	MEM	~	~	INPUT	~	□

ASCII- und Interpretercodetabelle für den PC-1421

ASCII-Codes:

#	16	32 SPC	48 0	64 P	80 R	96 X	112 Y	128 ~	144 FACT	160 RND	176 RUN	192 RANDOM	208 TO	224 GOSUB	240 PMT
1	17	JJ !	49 1	65 A	81 D	97 i	113 Y	129 REC	145 LN	161 AND	177 NEW	193 DEGREE	209 STEP	225 AREAD	241 NPV
2	18	JF "	50 2	66 D	82 R	98 n	114 Z	130 POL	146 LOG	162 OR	178 CONT	194 RADIAN	218 THEN	226 LPRINT	242 IRR
3	19	JS "	51 3	67 C	83 S	99 I	115 E	131 ROT	147 EXP	163 NOT	179 PASS	195 GRAD	211 ON	227 RETURN	243 PRM
4	20	J6 \$	52 4	68 D	84 T	100 "	116 Y	132 DEC1	148 SDR	164 ASC	180 LIST	196 BEEP	212 IF	228 RESTORE	244 INTE
5	21	J7 %	53 5	69 E	85 U	101 "	117 1	133 HEX	149 SIN	165 VAL	181 LLIST	197 WAIT	213 FOR	229 ERASE	245 BAL
6	22	J8 k	54 6	70 F	86 V	102 "	118 ^	134 TEH	150 COS	166 LEN	182 CSAVE	198 GOTO	214 LET	230 FIN	246 SPRN
7	23	J9 '	55 7	71 W	87 "	103 "	119 -	135 RCP	151 TAN	167 PEEK	183 CLOAD	199 TRON	215 REM	231 ~	247 SINTE
8	24	J0 (56 8	72 H	88 X	104 "	120 -	136 SDU	152 INT	168 CHR\$	184 ACC	200 TROFF	216 END	232 ~	248 NI
9	25	J1)	57 9	73 I	89 Y	105 i	121 i	137 CUR	153 ABS	169 STR\$	185 ARMT	201 CLEAR	217 NEXT	233 ~	249 CF1
10	26	J2 t	58 1	74 J	90 2	106 R	122 n	138 HSM	154 SGN	170 MID\$	186 COMP	202 USING	218 STOP	234 CST	250 ~
11	27	J3 +	59 1	75 K	91 L	107 S	123 ~	139 HCS	155 DEG	171 LEFT\$	187 MDF	203 DIM	219 READ	235 SEL	251 TT
12	28	J4 ,	60 -	76 L	92 V	108 T	124 II	140 HTN	156 DMS	172 RIGHT\$	188 EFF	204 CALL	220 DATA	236 MAR	252 V
13	29	J5 -	61 =	77 M	93 J	109 U	125 ~	141 AHS	157 ASN	173 INKEY\$	189 APR	205 POKE	221 PAUSE	237 MU	253 □
14	30	J6 ▶	62 .	78 N	94 ^	110 n	126 ~	142 AHC	158 ACS	174 PI	190 DAYSI	206 BGNON	222 PRINT	238 PV	254 ■
15	31	J7 ◀	63 ?	79 O	95 -	111 W	127 ~	143 AHT	159 ATN	175 MEM	191 DAYSI	207 BGNOFF	223 INPUT	239 FV	255

→ Sonderzeichen, die ohne Bedeutung sind.

ASCII- und Interpretercodetabelle für den PC-126X

ASCII-Codes:

Interpretercodes:

#	16 SHIFT	32 SPC	48	64	80	96	112	128	144	160	176	192	208	224	240
1	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241 SHIFT SPC
2	18 DEF	34	50	66	82	98	114	130	146	162	178	194	210	226	242
3	19 CA	35	51	67	83	99	115	131	147	163	179	195	211	227	243
4	20 HELP	36	52	68	84	100	116	132	148	164	180	196	212	228	244
5	21 SML	37	53	69	85	101	117	133	149	165	181	197	213	229	245
6	22 P++NP	38	54	70	86	102	118	134	150	166	182	198	214	230	246 SHIFT V
7	23 BRK	39	55	71	87	103	119	135	151	167	183	199	215	231	247
8	24 MODE	40	56	72	88	104	120	136	152	168	184	200	216	232	248 SHIFT X
9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249 ■
10	26 OFF	42	58	74	90	106	122	138	154	170	186	202	218	234	250 SHIFT Z
11	27 INS	43	59	75	91	107	123	139	155	171	187	203	219	235	251 TT
12	28 DEL	44	60	76	92	108	124	140	156	172	188	204	220	236	252 ~
13	29 ENTER	45	61	77	93	109	125	141	157	173	189	205	221	237	253 ^
14	30 ▶	46	62	78	94	110	126	142	158	174	190	206	222	238	254
15	31 ◀	47	63	79	95	111	127	143	159	175	191	207	223	239	255

** Auf der Anzeige: C

Die japanischen Zeichen sind unter den Codes 161-248 gespeichert. Diese Codes haben also offenbar eine Doppelbelegung:

1. Nur Zeichencode: Es erscheinen die BASIC-Befehle.
2. Dem eigentlichen Zeichencode wird der Code 254 vorangestellt: Nun erscheinen die japanischen Zeichen. Ein japanisches Zeichen wird also durch zwei Bytes codiert.

ASCII- und Interpretercodetabelle fuer den PC-1350

ASCII-Codes:

Interpretercodes:

0	16 SHIFT	32 SPC	48 0	64 P	80 '	96 p	112 `	128 SHIFT A	144 LN	160 AND	176 RUM	192 RANDOM	208 TO	224 60SUB	240
1	17 !	33 !	49 1	65 A	81 Q	97 x	113 q	129 SHIFT B	145 LN	161 AND	177 MEM	193 DEGREE	209 STEP	225 AREAD	241 BPRINT SPC
2	18 CLS	34 DEF	50 "	66 R	82 b	98 r	114 SHIFT C	130 SHIFT D	146 LN	162 OR	178 CONT	194 RADIAN	210 THEN	226 LPRINT	242
3	19 CA	35 SHIFT <	51 0	67 C	83 S	99 c	115 %	131 SHIFT C	147 EXP	163 NOT	179 PASS	195 GRAD	211 ON	227 RETURN	243 SHIFT S
4	20 +	36 SHIFT ▶	52 4	68 D	84 T	100 d	116 t	132 SHIFT D	148 SDR	164 ASC	180 LIST	196 BEEP	212 IF	228 RESTORE	244 SHIFT *
5	21 +	37 SML	53 1	69 E	85 U	101 *	117 u	133 SHIFT E	149 SIN	165 VAL	181 LLIST	197 WAIT	213 FOR	229 CHAIN	245
6	22 P+NP	38 \$	54 6	70 F	86 V	102 f	118 v	134 SHIFT F	150 COS	166 LEN	182 CSAVE	198 GOTO	214 LET	230 GCURSOR	246 SHIFT V
7	23 BRK	39 '	55 7	71 G	87 W	103 g	119 w	135 SHIFT G	151 TAN	167 PEEK	183 CLOAD	199 TRON	215 REM	231 BPRINT	247
8	24 MODE	40 (56 8	72 H	88 X	104 h	120 x	136 SHIFT H	152 INT	168 CHR\$	184 MERGE	200 TROFF	216 END	232 LINE	248 SHIFT I
9	25)	41 9	57 I	73 Y	89 i	105 y	121 Y	137 SHIFT I	153 ABS	169 STR\$	185 I85	201 CLEAR	217 NEXT	233 POINT	249 ■
10	26 OFF	42 :	58 J	74 Z	90 j	106 z	122 z	138 SHIFT J	154 S6N	170 MID\$	186 USING	202 STOP	218 PSET	234 SHIFT Z	250 ,"
11	27 INS	43 +	59 K	75 [91 k	107]	123]	139 SHIFT K	155 DEG	171 LEFT\$	187 OPEN	203 DIM	219 READ	235 PRESET	251 TT
12	28 DEL	44 ,	60 <	76 L	92 \	108 1	124 1	140 SHIFT L	156 DMS	172 RIGHT\$	188 CLOSE	204 CALL	220 DATA	236 BASIC	252 √
13	29 ENTER	45 -	61 =	77 M	93)	109 "	125)	141 SHIFT M	157 ASN	173 INKEY\$	189 SAVE	205 POKE	221 PAUSE	237 TEXT	253
14	30 ▶	46 .	62 >	78 N	94 ^	110 n	126 ~	142 SHIFT N	158 ACS	174 PI	190 LOAD	206 CLS	222 PRINT	238 OPEN\$	254
15	31 ◀	47 /	63 ?	79 0	95 -	111 o	127)	143 ATN	159 MEM	175 CONSOLE	191 CURSOR	207 INPUT	223 239	255	

→ Auf der Anzeige: □

Nun geben Sie folgende BASIC-Zeile ein:

1:REM

Indem Sie in diese BASIC-Zeile direkt über POKE einen Code eingeben, können Sie sehen, wie ein solcher Code interpretiert wird. Wollen Sie das Zeichen oder den Befehl eines Codes anschauen, geben Sie deshalb ein:

PC 14XX:	POKE 8196, Code
PC 126X:	POKE 16388, Code
PC 1350:	POKE 8244, Code (12 oder 20K RAM)
PC 1350:	POKE 24625, Code (4K RAM)

Nun schauen Sie sich Ihr BASIC-Programm an! Der erste Punkt der REM-Zeile wurde durch das Zeichen mit dem von Ihnen eingegebenen Code ersetzt.

Kapitel 2

Variablen: Speicherung – eine vielfältige Sache

2.1 Die Verwaltung von Variablen

Wir wollen jetzt einmal anschauen, wo und wie der Computer die verschiedenen Variablen (also z.B. A\$, Z(3,2) usw.) speichert und verarbeitet. Jeder Computerbenutzer, der die Arbeitsweise seines Rechners besser verstehen will, sollte sich dieses Wissen aneignen.

2.2 Das BCD-System

Neben dem High/Lowbyte-System, das ja nur für natürliche Zahlen bis 65535 geeignet ist, kennt der Rechner auch noch ein anderes System, das er für alle numerischen Variablen anwendet. Mit diesem System ist es möglich, Zahlen zwischen $-9.99 \cdot 10^{99}$ und $9.99 \cdot 10^{99}$ darzustellen. Auch nicht ganzzahlige Werte können gespeichert werden (z.B. 1.4275).

Diese Form der Speicherung sieht beim ersten Hinschauen recht kompliziert aus, zumal nicht jeder das Arbeiten mit Hexadezimal- und BCD-Zahlen gewohnt ist. Wir wollen aber die Verwaltung der Variablen gleich anhand von einfachen Beispielen erläutern.

Doch zuerst zum BCD-System: BCD ist eine Abkürzung für 'Binary Coded Decimals', das heißt 'Binär codierte Dezimalzahlen'. Jede Ziffer einer Dezimalzahl wird als 4-stellige Binärzahl gespeichert. Je 4 Bits werden benötigt, um eine Ziffer zwischen 0 und 9 darzustellen. Eine Ziffer nimmt also ein Halbbyte oder Nibble in Anspruch. In einem Byte können zwei binär codierte Dezimalziffern voneinander unabhängig untergebracht werden. Die folgende Tabelle zeigt, wie die Ziffern 0 bis 9 abgespeichert sind.

Dezimal	Binär
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001



Beispiel:

Wie wird die Zahl 39 im BCD-System abgespeichert? Die Zahl 39 wird aufgeteilt in die Ziffern 3 und 9, wobei jede Ziffer in einem Halbbyte (= 4 Bits = 1 Nibble) abgespeichert wird.

Ziffer 3:	Ziffer 9:
1. Nibble	2. Nibble
Bit Nr.: <u>3</u> 2 1 0	<u>3</u> 2 1 0
Wertigkeit: 0 0 1 1	1 0 0 1
Auswertung: $2^1 + 2^0 = \underline{\underline{3}}$	$2^3 + 2^0 = \underline{\underline{9}}$



Nun werden diese beiden 4-stelligen Binärzahlen zu einer 8-stelligen Binärzahl zusammengefaßt, die dann mit 8 Bits dargestellt und in einem Byte untergebracht werden kann.

Bit Nr.: 7 6 5 4 3 2 1 0	
Wertigkeit: 0 0 1 1 1 0 0 1	
Auswertung: $2^5 + 2^4 + 2^3 + 2^0 = \underline{\underline{57}}$	

Speichern Sie in der Variablen Z den Wert 39 (Z=39). Die Zahl 39 wurde nun als binär codierte Dezimalzahl abgespeichert. Für die Variable Z sind die Speicherzellen 17872 bis 17879¹⁾ reserviert. Mit PEEK 17874²⁾ erhalten Sie nun den Wert 57, was wieder der binär codierten Dezimalzahl 39 entspricht. Wenn Sie nämlich die ausgelesene Dezimalzahl 57 ins Hexadezimale Zahlensystem umrechnen, erhalten Sie wieder die Zahl 39, die Sie vorher in Z gespeichert haben. Auch eine hexadezimale Ziffer wird ja mit 4 Bits dargestellt.

 Der Unterschied zwischen dem BCD- und dem hexadezimalen Zahlensystem besteht einzig darin, daß die hexadezimalen Zahlen Ziffern zwischen 0 und 15 besitzen, die hexadezimal mit den Buchstaben A bis F dargestellt werden. BCD-Zahlen hingegen besitzen nur Zahlen zwischen 0 und 9.

Für das Umrechnen können Sie beim PC 14XX die Funktionen HEX und DECI in Anspruch nehmen. Bei einigen Rechnern fehlt die Funktion DECI. Hier hilft ein kurzer Blick auf die Umrechnungstabelle im Anhang E.

Alle Zahlen der numerischen Variablen werden nach diesem Zahlensystem codiert.

Zusammenfassung

 Wollen wir eine Zahl zwischen 0 und 99 auslesen, die im BCD-System abgespeichert wurde (was bei den Variablen immer der Fall ist), müssen wir den ausgelesenen Zahlenwert zuerst in eine hexadezimale Zahl umrechnen, diese umgerechnete hexadezimale Zahl dann aber als Dezimalzahl interpretieren.

-
- 1) PC 1421: 17792 bis 17799, PC 126X: 25856 bis 25863, PC 1350: 27696 bis 27703
 - 2) PC 1421: 17794, PC 126X: 25858, PC 1350: 27698

2.3 Die Standardvariablen

Für die Standardvariablen (z.B. G, D\$) ist ein fester Speicherbereich reserviert. Dem Speicherbelegungsplan in Kapitel 1.3 können Sie entnehmen, daß dieser feste Speicherbereich bei Adresse 17872¹⁾ beginnt und bei Adresse 18079¹⁾ endet.

Unter den Standardvariablen versteht man diejenigen Variablen, deren Namen nur aus einem Buchstaben bestehen (so z.B. C für eine numerische, C\$ für eine Textstandardvariable). Es gibt demzufolge genau 26 Standardvariablen (A-Z), die entweder als Text- oder als numerische Variablen verwendet werden können. Jede dieser Standardvariablen belegt einen bestimmten Speicherbereich von je 8 Bytes.

Die folgende Tabelle gibt eine Übersicht über alle Standardvariablen, deren Namen, Anfangs- und Endadressen:

Variable	PC 140X Adresse von - bis	PC 1421 Adresse von - bis	PC 126X Adresse von - bis	PC 1350 Adresse von - bis
Z, Z\$	17872-17879	17792-17799	25856-25863	27696-27703
Y, Y\$	17880-17887	17800-17807	25864-25871	27704-27711
X, X\$	17888-17895	17808-17015	25872-25879	27712-27719
W, W\$	17896-17903	17816-17823	25880-25887	27720-27727
V, V\$	17904-17911	17824-17831	25888-25895	27728-27735
U, U\$	17912-17919	17832-17839	25896-25903	27736-27743
T, T\$	17920-17927	17840-17847	25904-25911	27744-27751
S, S\$	17928-17935	17848-17855	25912-25919	27752-27759
R, R\$	17936-17943	17856-17863	25920-25927	27760-27767
Q, Q\$	17944-17951	17864-17871	25928-25935	27768-27775
P, P\$	17952-17959	17872-17879	25936-25943	27776-27783
O, O\$	17960-17967	17880-17887	25944-25951	27784-27791
N, N\$	17968-17975	17888-17895	25952-25959	27792-27799
M, M\$	17976-17983	17896-17903	25960-25967	27800-27807

1) Diese Werte gelten für den PC 140X. Gültige Adressen für die andern Rechner finden Sie in der nächsten Tabelle.

L, LS	17984-17991	17994-17999	25968-25971	27808-27815
K, KS	17992-17997	17912-17919	25976-25983	27816-27823
J, JS	18008-18007	17920-17927	25984-25991	27824-27831
I, IS	18008-18015	17928-17935	25992-25999	27832-27839
H, HS	18018-18023	17936-17943	26000-26007	27840-27847
G, GS	18024-18031	17944-17951	26008-26015	27848-27855
F, FS	18032-18039	17952-17959	26016-26023	27856-27863
E, ES	18040-18047	17960-17967	26024-26031	27864-27871
D, DS	18048-18055	17968-17975	26032-26039	27872-27879
C, CS	18056-18063	17976-17983	26040-26047	27880-27887
B, BS	18064-18071	17984-17991	26048-26055	27888-27895
A, AS	18072-18079	17992-17999	26056-26063	27896-27903
STACK	18080-18087	18080-18087	-	-
CAL	18088-18094	18088-18094	-	-

Speicher

Der STACK- und CAL-Speicher des PC 14XX

Obwohl die Bytes des STACK- und CAL-Speichers eigentlich zu den Systemvariablen gehören, schließen wir sie dennoch in diese Tabelle ein, da auch diese Speicher numerische Zahlenwerte speichern, die nach dem BCD-System codiert werden.

Der CAL-Speicher ist im CAL-Modus aktiv. Er kann dann über die Tasten X->M, RM und M+ bedient werden. Durch die Befehle PEEK und POKE aber können wir diesen Speicher auch im RUN-Modus auslesen und verändern!

Im STACK-Speicher bleibt das Resultat der zuletzt ausgeführten Berechnung oder der letzten numerischen Eingabe gespeichert, bis wieder eine neue Rechenoperation ausgeführt wird. Der Inhalt des STACK-Speichers kann mittels der Tasten '↑' und '↓' im RUN-Modus auf die Anzeige gebracht werden (dies geht nur, wenn das PASS-Wort nicht aktiv ist).

Standardtextvariablen

Der Text, der in einer Textvariablen gespeichert werden soll, wird vom Rechner anhand der sich im ROM befindlichen (ASCII-) Zeichencodetabelle codiert. Diese Tabelle finden Sie in Kapitel 1.9, sie ist dort bereits ausführlich beschrieben.

Alle Zeichen einer Textvariablen werden nach dieser Zeichencodetabelle verschlüsselt; sie sind alle im ASCII-Code abgespeichert.

Beispiel:

Geben Sie ein: **P\$="ABC"**

Die Variable P (oder hier P\$) nimmt einen festen Platz im Bereich der Standardvariablen ein. Die dabei maßgebenden Adressen finden Sie in der nächsten Tabelle. Mit dem PEEK-Befehl können Sie auch die unterstehenden Daten selbst auslesen und nachprüfen.

Also: **P\$="ABC"**

Adresse

1400	1421	1260	1350	Wert	Bedeutung
17952	17872	25936	27776	245	Eine 245 in der ersten Speicherzelle einer Standardvariable weist diese als Textvariable aus.

Nun folgt der eigentliche Text:

17953 17873 25937 27777 65 A ASCII-Code für 'A'
17954 17874 25938 27778 66 B ASCII-Code für 'B'
17955 17875 25939 27779 67 C ASCII-Code für 'C'
17956 17876 25940 27780 0 Eine Null zeigt das vor-

zeitige Ende eines
Textes an (nicht alle 7
Bytes werden benutzt).

17959 17879 25943 27783 0 Letzte Stelle von P\$,
hier nicht benutzt.

17960 17880 25944 27784

Hier beginnt nun die Variable
0 bzw. 0\$. Die Adressen 17957
bis 17959 gehören noch zu P,
werden in diesem Beispiel aber
nicht benutzt.

Da für jede Standardvariable 8 Bytes reserviert sind, fassen
Standardtextvariablen nur Texte mit maximal 7 Zeichen (2. -
8. Speicherzelle der Variable).

Mit dem POKE-Befehl können Sie die Variable P\$ nun direkt
verändern: Der Buchstabe 'B' des Textes 'ABC' soll durch ein
Ausrufezeichen (!) ersetzt werden. Das Ausrufezeichen besitzt
den Code 33 (Kapitel 1.9). Die 66 in Speicherzelle 179541) muß
folglich durch eine 33 ersetzt werden. Geben Sie deshalb ein:

POKE 17954¹⁾,33

Schauen Sie jetzt nach, was in P\$ gespeichert ist: Geben Sie 'P\$'
ein und drücken Sie ENTER!

1) PC 1421: 17874, PC 126X: 25938, PC 1350: 27778

Numerische Standardvariablen

Nun kommen wir zu den numerischen Standardvariablen. Bevor wir die 'Speicherlandschaft' dieser Variablen erforschen können, teilen wir sie, dem besseren Verständnis zuliebe, in drei Untergruppen ein:

1. Zahlen ohne Exponent
2. Zahlen mit positivem Exponent
3. Zahlen mit negativem Exponent

Zahlen ohne Exponent

Beispiel:

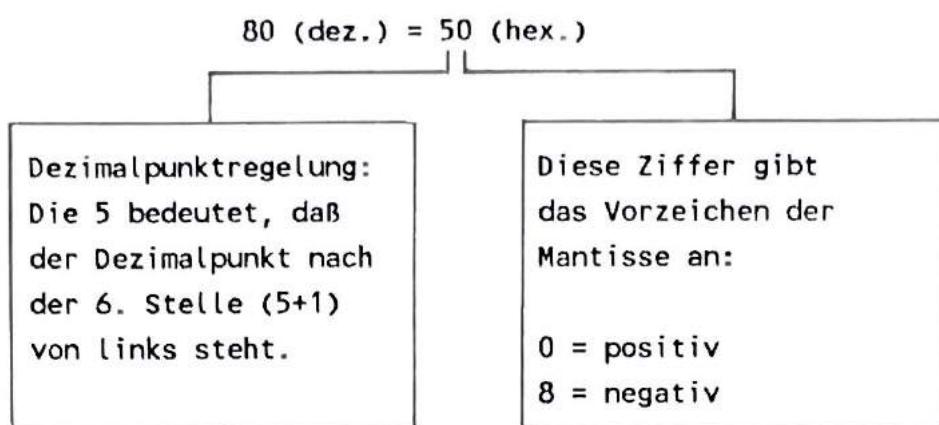
$$P = 153464$$

	Speicherzelle			Wert	Wert hexadezimal (oder BCD)
	140X	1421	126X	1350 dezimal	
1.	17952	17872	25936	27776	00
2.	17953	17873	25937	27777	80
3.	17954	17874	25938	27778	21
4.	17955	17875	25939	27779	52
5.	17956	17876	25940	27780	100
6.	17957	17877	25941	27781	00
7.	17958	17878	25942	27782	00
8.	17959	17879	25943	27783	00

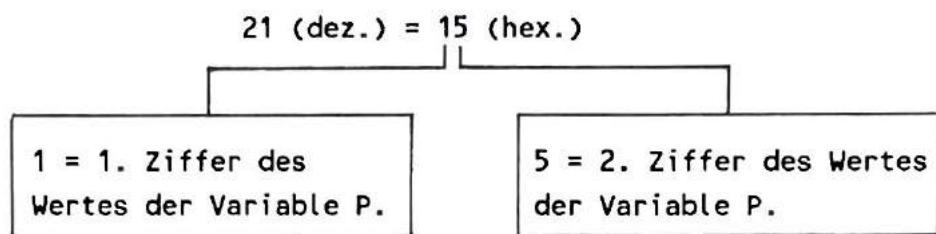
Wie aber müssen diese Zahlenfolgen interpretiert werden, damit wir für P den Wert 153464 erhalten? Das wollen wir nun Schritt für Schritt durchgehen. Dabei sollten Sie noch wissen: Unter einer Mantisse versteht man allgemein die Basis einer Zehnerpotenz; in unserem Fall also die Ziffern vor dem Exponent.

P = 153464

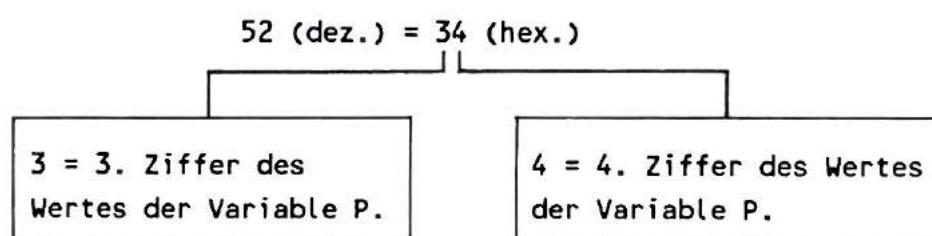
1. Adresse: 0 Es ist kein Exponent vorhanden
2. Adresse: 80 Diese Zahl muss, da die Variablen prinzipiell im BCD-Code verarbeitet werden, ins hexadezimale Zahlensystem umgerechnet werden (siehe Tabelle).



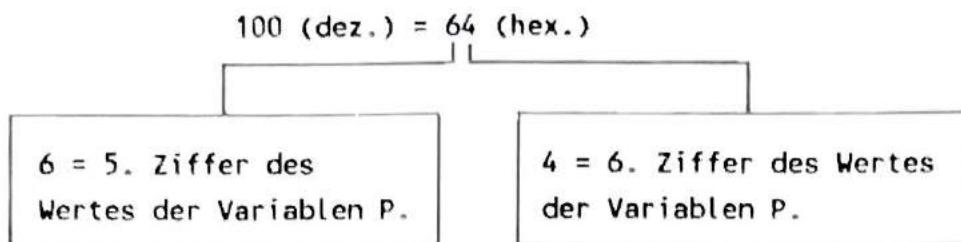
3. Adresse: 21 In dieser und den folgenden Adressen folgt der eigentliche Variablenwert.



4. Adresse: 52



5. Adresse: 100



6. Adresse: 0 Die Zahl 153464 besitzt keine weiteren Ziffern, also keine Nachkommastellen mehr.

Bei den numerischen Variablen hat die 8. Adresse normalerweise immer den Inhalt 0. Diese Speicherzelle wird für diese Variablen nicht verwendet!

Hat eine Zahl weniger als 10 Ziffern, z.B. A = 23, so werden ja nur die Adressen 1, 2 und 3 benötigt, die restlichen (4-8) werden mit Nullen aufgefüllt.

Nun können Sie mit dem POKE-Befehl die Bytes der Variablen P direkt verändern:

Wenn Sie zum Beispiel das Vorzeichen der Variablen ändern wollen, so müssen Sie ganz einfach den Inhalt der 2. Adresse von 50 (hex.) auf den Wert 58 (hex.) setzen (die zweite Ziffer dieser Zahl ist ja für das Vorzeichen der Mantisse zuständig).

Damit der Rechner erkennen kann, daß die Zahl 58 hexadezimal zu verstehen ist, müssen Sie vor diese Zahl ein '&' -Zeichen setzen, also:

PC 140X:	POKE 17953,&58
PC 1421:	POKE 17873,&58
PC 126X:	POKE 25937,&58
PC 1350:	POKE 27777,&58

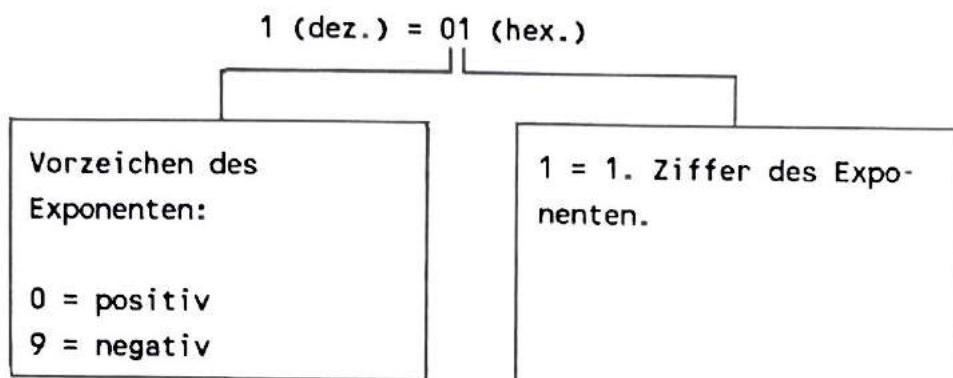
Geben Sie nun 'P' ein und drücken Sie die ENTER-Taste!
Es erscheint: -153464

Zahlen mit positivem Exponent

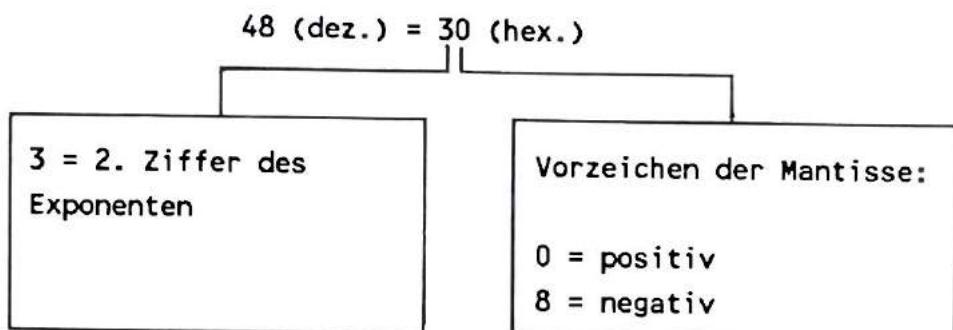
Wenn Zahlen mit Exponenten eingegeben werden, erfolgt eine etwas andere Abspeicherung:

Zum Beispiel: P = 2.5 E13 (2.5 * 10¹³)

1. Adresse: 1



2. Adresse: 48



In der 3. bis 7. Adresse stehen die entsprechenden Ziffern der Mantisse. Wir wollen hier auf eine Auflistung dieser Zahlen verzichten, da deren Abspeicherung identisch ist mit denjenigen

der numerischen Variablen ohne Exponent, und die Verarbeitung dieser Zahlen wurde ja schon detailliert behandelt.

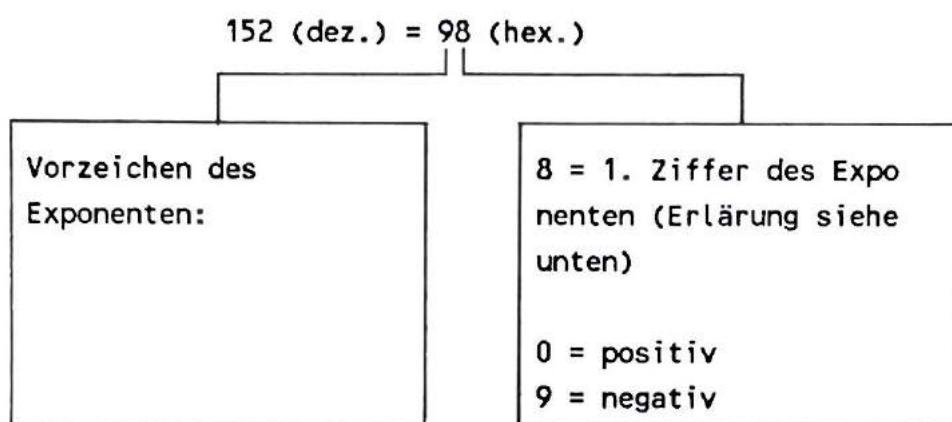
Bei der Darstellung von Zahlen mit Exponenten steht der Dezimalpunkt immer nach der ersten Stelle von links. Es wird deshalb kein zusätzliches Nibble (1 Nibble = 4 Bit) als Positionszeiger für den Dezimalpunkt benötigt.

Zahlen mit negativem Exponent

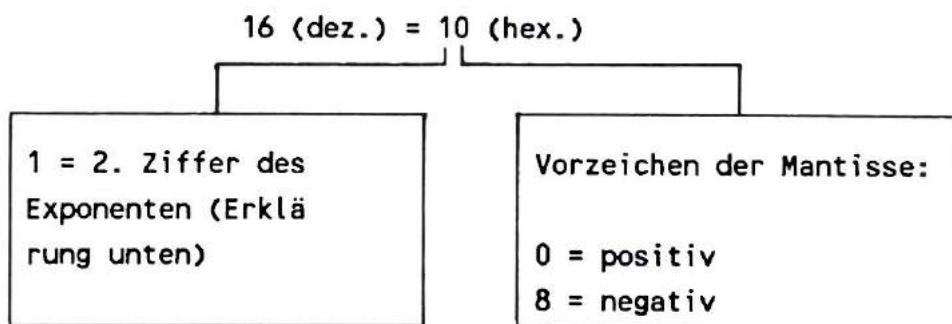
Der negative Exponent ist wieder etwas anders aufgebaut als der positive:

Beispiel: P = 4.56 E-19

1. Adresse: 152



2. Adresse: 16



Auch hier folgen in der 3. bis 7. Adresse die Ziffern der Mantisse. Die Abspeicherung ist gleich mit derjenigen der numerischen Variablen ohne Exponent.

Erklärung zu den Ziffern mit negativen Exponenten:

Auf dem ersten Blick mögen die ausgelesenen Werte etwas befremdend wirken, zumal wir ja für den Exponenten den Wert -19 eingeben, aber den Wert 81 ausgelesen haben.

Um an den wirklichen Exponenten von -19 heranzukommen, muß so gerechnet werden:

$$\begin{array}{l} 1. \text{ Ziffer} = 8 \\ 2. \text{ Ziffer} = 1 \quad \text{also der Wert } 81 \end{array}$$

Daraus folgt: der Exponent lautet $81 - 100 = -19$

Experimentieren Sie mit eigenen Beispielen, um die erworbenen Kenntnisse noch zu vertiefen!

2.4 Selbstdefinierte Variablen

Zu den selbstdefinierten Variablen zählen wir alle diejenigen Variablen, denen kein fester Speicherplatz im RAM zugewiesen ist (also alle dimensionierten Variablen und alle Variablen, deren Namen aus zwei und mehr Buchstaben bestehen).

Während die 8 Bytes einer bestimmten Standardvariablen immer die gleichen Speicherzellen besetzen, so ist die Lage einer selbstdefinierten Variable nicht genau festgelegt. Sie hängt von der Größe und Anzahl der bereits verwendeten oder dimensionierten selbstdefinierten Variablen ab.

Die Art der Verwaltung von selbstdefinierten numerischen Variablen oder Textvariablen ist identisch mit derjenigen der numerischen Standard- oder Standardtextvariablen. Auch selbst-

definierte Textvariablen werden nach der ASCII-Code-Tabelle codiert, numerische selbstdefinierte Variablen werden auch in dem uns schon bekannten BCD-Code abgespeichert.

Da kein fester Speicherbereich für diese Art von Variablen reserviert ist, muß mindestens irgendwo im Systemvariablenbereich angegeben sein, wo denn der Block der selbstdefinierten Variablen beginnt. Wie Sie aus Kapitel 2 ('Der BASIC-Programmspeicher') ersehen können, breiten sich die selbstdefinierten Variablen von der Speicherzelle 1787¹⁾ an zu den niedrigeren Adressen hin aus. Das obere Ende der selbstdefinierten Variablen befindet sich also immer bei der Adresse 1787¹⁾.

Wie aber finden wir die Anfangsadresse dieses Speicherblocks? Er steht in den Speicherzellen 18172/73²⁾, codiert nach dem Low/Highbyte-Format. In der ersten dieser beiden Speicherzellen steht also das Lowbyte der Anfangsadresse, in der zweiten das Highbyte.

Anfangsadresse der selbstdefinierten Variablen:

$$\text{PEEC } 18172^{\text{2)}} + \text{PEEC } 18173^{\text{2)}} * 256$$

Kommen wieder neue, noch nie verwendete Variablen dazu, so werden diese 'unten' angehängt (d.h. gegen die niedrigeren Adressen hin), und die Anfangsadresse für die selbstdefinierten Variablen in den oben erwähnten Speicherzellen wird wieder auf den aktuellen Stand gebracht. Durch das ständige Ändern der Anfangsadresse beim Hinzukommen neuer Variablen ist es natürlich schwierig, die Lage einer selbstdefinierten Variable vorzusagen. Sie können aber davon ausgehen, daß eine einmal benutzte Variable ihre Lage im RAM nicht mehr ändert, solange nicht ein CLEAR-, NEW- oder RUN-Befehl ausgeführt wird.

1) PC 1421: 17791, PC 126X: 25858, PC 1360: 27055

2) PC 126X: 26384/65, PC 1360: 28423/24

wodurch alle Variablen gelöscht werden (bei RUN werden nur die selbstdefinierten Variablen gelöscht, alle Standardvariablen bleiben erhalten).

Im Gegensatz zu den Standardvariablen, die ganz bestimmte Plätze und Namen haben, können Sie ja die Namen der selbstdefinierten Variablen frei bestimmen. Damit der Computer einmal verwendete Variablen wiederfindet, müssen also auch deren Namen mitgespeichert werden, was bei den Standardvariablen ja nicht der Fall ist. Obwohl der Name einer selbstdefinierten Variable mehr als zwei Buchstaben haben kann, werden immer nur die ersten beiden Zeichen des Namens gespeichert. Achten Sie deshalb darauf, daß Ihr Programm nicht zwei Variablen enthält, deren erste beiden Zeichen gleich sind (z.B. die Variablen MONAT und MONTAG kann der Rechner nicht unterscheiden).

'Normale' selbstdefinierte Variablen

Darunter verstehen wir diejenigen Variablen, die einen Namen aus zwei oder mehr Buchstaben besitzen, die aber nicht durch einen DIM-Befehl vorbereitet werden müssen. Also z.B. PRS, TAG, aber nicht TXS(2,2). An zwei Beispielen wollen wir Ihnen die Verwaltung der normalen selbstdefinierten Variablen erklären:

1. Beispiel: selbstdefinierte Textvariablen

Geben Sie CLEAR ein, um alle noch bestehenden Variablen zu löschen. Füllen Sie darauf die Variable PRS mit dem Wort 'ELEFANT' (also: PRS = 'ELEFANT').

Lesen Sie aus den Adressen 18172/18173¹⁾ die Anfangsadresse der selbstdefinierten Variablen aus. Als Anfangsadresse sollten Sie erhalten:

1) PC 194X: 24864/65, PC 1950: 18418/24

PC 140X:	17849
PC 1421:	17769
PC 126X:	25833
PC 1350:	27673

Somit ergibt sich für die Variable PR\$ folgende Auflistung:

Adresse

140X	1421	1260	1350	Wert	Bedeutung
17849	17769	25833	27673	80	ASC "P"
17850	17770	25834	27674	146	ASC "R" + 64 Name der Variablen
17851	17771	25835	27675	0	HB Ende der Variablen PR\$
17852	17772	25836	27676	19	LB bei Adresse: 17852+19+0*256 = 1787 ¹⁾
17853	17773	25837	27677	0	hier unbenutzt
17854	17774	25838	27678	0	hier unbenutzt
17855	17775	25839	27679	16	maximal zulässige Länge des Variablen textes von PR\$

Nun folgt der eigentliche Text:

17856	17776	25840	27680	69	E	ASCII-Code für 'E'
17954	17874	25841	27681	66	L	ASCII-Code für 'L'
17955	17875	25842	27682	67	E	ASCII-Code für 'E'

Und so weiter ...

In einer normalen selbstdefinierten Textvariable können maximal 16 Zeichen gespeichert werden. Werden nicht alle 16 Zeichen benötigt, so werden die restlichen, ungebrauchten Speicherzellen mit Nullen gefüllt.

1) Entsprechend andere Werte für die anderen Rechner

Brauchen Sie Variablen, die mehr Platz bieten, so müssen Sie Feldvariablen dimensionieren, z.B. DIM E\$(0)*45. Die somit erstellte Variable E\$(0) bietet Platz für maximal 45 Zeichen.

2. Beispiel: numerische selbstdefinierte Variablen

Füllen Sie die Variable TAG mit dem Wert 345.6 (Tag=345.6). Geben sie dabei keinen CLEAR-Befehl ein; die Variable PR\$ von vorhin sollte gespeichert bleiben. Lesen Sie die neue Anfangsadresse der selbstdefinierten Variablen aus den Adressen 18172/18173¹⁾. Da sich die Variable PR\$ noch immer im Speicher befindet, wird die neudefinierte Variable TAG direkt 'darunter' angehängt.

Adresse						
140X	1421	126X	1350	Wert	Bedeutung	
17834	17754	25818	27658	84	ASC "T"	
17835	17755	25819	27659	65	ASC "A" Name der	
					Variable. Nur die ersten	
					zwei Buchstaben des Na-	
					mens werden gespeichert.	
17836	17756	25820	27660	0	HB	Ende der
						Variable TAG
17837	17757	25821	27661	11	LB	bei Adresse:
						$17837+11+0*256 = 17848$ ²⁾
17838	17758	25822	27662	0	hier unbenutzt	
17839	17759	25823	27663	0	hier unbenutzt	
17840	17760	25824	27664	8	Länge der Variable:	
						Maximal 8 Bytes.

Und so weiter ...

1) PC 126X: 26364/65, PC 1350: 28423/24

2) Entsprechend andere Werte für die anderen Rechner

Die nachfolgenden 8 Bytes enthalten nun die in der Variable TAG gespeicherte Zahl 345.6. Diese Zahl wird nach dem BCD-Schlüssel codiert, wobei die Funktionen der einzelnen Adressen der Variable gleich wie bei den Standardvariablen sind. Genaueres darüber erfahren Sie aus Kapitel 2.2 und 2.3.

Feldvariablen

Mit den Feldvariablen behandeln wir eine zweite Gruppe von selbstdefinierten Variablen, nämlich diejenigen, die mit dem DIM-Befehl vorbereitet werden müssen.

Also zum Beispiel: T\$(1,1)*2; FB(0,1).

Auch hier wollen wir die Funktionsweise gleich anhand von zwei Beispielen erklären:

1. Beispiel: Textfeldvariablen

Geben Sie CLEAR ein, um den Variablen Speicher zu löschen. Dimensionieren Sie die Variable T\$ folgendermaßen:

DIM T\$(1,1)*2

Geben Sie folgende Werte ein:

1. Feld: T\$(0,0) = "AB"
2. Feld: T\$(0,1) = "CD"
3. Feld: T\$(1,0) = "EF"
4. Feld: T\$(1,1) = "GH"

Ermitteln Sie die Anfangsadresse der selbstdefinierten Variablen aus den Speicherzellen 18172/18173¹⁾.

1) PC 126X: 26364/65, PC 1350: 28423/24

Wir können die folgende Tabelle zusammenstellen:

Adresse				Wert		Bedeutung
140X	1421	1260	1350			
17857	17777	25841	27681	84	ASC "T"	
17858	17778	25842	27682	160	160	kennzeichnet eine Textfeldvariable
17859	17779	25843	27683	0	HB	Ende der Variablen
17860	17780	25844	27684	11	LB	T\$(1,1) bei Adresse:
						$17860+11+0*256 = 17871^1)$
17861	17781	25845	27685	1	Höchste Variable:	
17862	17782	25846	27686	1	T\$(1,1)	
17863	17783	25846	27687	2	Maximale Anzahl Zeichen pro Feld: 2	

Nun folgt der eigentliche Text:

17864	17784	25847	27688	65	A	ASCII-Code für 'A'
17965	17885	25848	27689	66	B	ASCII-Code für 'B'
17966	17886	25849	27690	67	C	ASCII-Code für 'C'
17867	17787	25850	27691	68	D	ASCII-Code für 'D'
17968	17888	25851	27692	69	E	ASCII-Code für 'E'
17969	17889	25852	27693	70	F	ASCII-Code für 'F'
17970	17890	25853	27694	71	G	ASCII-Code für 'G'
17971	17891	25854	27695	72	H	ASCII-Code für 'H'

Und so weiter ...

1) Entsprechende Werte für die anderen Rechner

2. Beispiel: numerische Feldvariablen

Geben Sie CLEAR ein und dimensionieren Sie danach die Variable FB mit dem Befehl DIM FB(0,1).

Geben Sie folgende Werte ein:
 1. Feld: FB(0,0)=-375
 2. Feld: FB(0,1)=1748

Daraus ergibt sich folgende Tabelle:

Adresse				Wert	Bedeutung
140X	1421	1260	1350		
17849	17769	25833	27673	70	ASC "F"
17850	17770	25834	27674	194	ASC "B"+128 Name der Variable
17851	17771	25835	27675	0	HB Ende der Variable
17852	17772	25836	27676	19	LB FB(0,1) bei Adresse: 17852+19+0*256 = 17871 ¹⁾
17853	17773	25837	27677	1	
17854	17774	25838	27678	0	Höchste Variable:FB(0,1)
17855	17775	25839	27679	8	Anzahl Bytes pro Feld: 8

Hier beginnen und enden die einzelnen Felder:

17856	17776	25840	27680	
bis	bis	bis	bis	FB(0,0)
17863	17783	25847	27687	

17864	17784	25848	27688	
bis	bis	bis	bis	FB(0,1)
17871	17791	25855	27695	

1) Entsprechend andere Werte für die anderen Rechner

Auch bei den numerischen Feldvariablen benötigen die Felder je 8 Bytes, und die Zahlen werden im BCD-System abgespeichert.

Es ist Ihnen sicher aufgefallen, daß die selbstdefinierten Variablen relativ viel Speicherplatz fressen, zumal eben auch Name, Länge, Matrix usw. als "Vorspann" abgespeichert werden müssen. Wenn Sie also platzsparend programmieren wollen (oder müssen), so benutzen Sie wenn möglich nur Standardvariablen.

Kapitel 3

Systemadressen: Die interne Organisation

In diesem Kapitel werden wir tiefer auf die verschiedenen Systemvariablen eingehen und die Funktionsweise der wichtigsten davon detailliert in Tabellenform erklären.



3.1 Interessante Systemadressen

Der SystemvariablenSpeicher wird zur Speicherung äußerst wichtiger Daten benutzt. Diese Daten geben nämlich Auskunft über den aktuellen Betriebszustand des Rechners, Betriebszustand im weitesten Sinn des Wortes. In diesem Speicherbereich sind alle Werte abgespeichert, die während des Betriebs oft geändert werden müssen und die der Computer zur Ausführung eines Programms benötigt (z.B. die Nummer der aktuellen Unterprogrammebene, die Adresse der zuletzt benutzten Variablen, Zwischenspeicherung von Resultaten usw.). Auch das PASS-Wort wird in diesem Speicherbereich abgelegt.

Mit unüberlegten POKE-Befehlen innerhalb des SystemvariablenSpeichers können Sie die Organisation des Rechners derart verändern und stören, daß er nur durch Drücken der ALL-RESET-Taste wieder in den Normalzustand zurückversetzt werden kann.



Aber gerade durch gezielte 'Mutationen' können sehr interessante Effekte hervorgerufen werden, die man für eigene Zwecke in Programmen nutzen kann.

3.2 Übersicht über die Systemvariablen

Aus der großen Auswahl von Systemadressen haben wir im Folgenden nur diejenigen aufgeführt, die auch einen praktischen Nutzen haben. Doch diese Systemadressen, das werden Sie bald feststellen, haben es in sich!

LB = Lowbyte

HB = Highbyte

Wichtige Systemadressen für den PC 14XX

Adresse	Funktion/Erklärung
18101 LB	Dieses Adreßpaar zeigt auf die Speicherzelle und direkt nach dem zuletzt ausgeführten BASIC-
18102 HB	Befehl im Speicher.
	Nach dem Starten eines Programms: Zeigt auf den BASIC-Speicher. Im Direktmodus: Zeigt auf den Ein/Ausgabepuffer.
18113 bis 18119	An dieser Stelle ist das PASS-Wort abgelegt. Es wird nach dem ASCII-Schlüssel decodiert.
18136	2: WAIT-Intervall eingeschaltet 6: WAIT-Intervall ausgeschaltet
18137	0-31: PASS-Wort unwirksam, kein Programmschutz 32-63: PASS-Wort wirksam, Programmschutz
18145 LB	Zeiger auf den Anfang des BASIC-Programms
18146 HB	
18147 LB	Zeiger auf das Ende des BASIC-Programms
18148 HB	
18149 LB	Dauer des WAIT-Intervalls
18150 HB	
18163	Nummer der aktuellen FOR-NEXT-Schleife: 6=0; 24=1; 42=2; 60=3; 78=4; 96=5

18164		Nummer des aktuellen Unterprogramms: 104=0; 106=1; 108=2; ...; 124=10
18172	LB	Zeiger auf die Anfangsadresse der selbstdefinierten Variablen
18173	HB	
18176	LB	Adresse des Cursors nach BREAK oder Fehler
18177	HB	
18182	LB	1. FOR-NEXT-Schleife: Zeiger auf das Byte direkt vor der Anfangsadresse der Schleifenvariable
18183	HB	
18184		1. FOR-NEXT-Schleife: Endwert der Schleife, codiert dem nach BCD-System
bis		
18190		
18191		1. FOR-NEXT-Schleife: STEP-Wert der Schleife, codiert nach dem BCD-System
bis		
18197		
18198	LB	1. FOR-NEXT-Schleife: Zeiger auf das Byte direkt nach dem 1. Schleifenausdruck
18199	HB	
18200		2. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT-Schleife
bis		
18217		
18218		3. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT-Schleife
bis		
18235		
18236		4. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT-Schleife
bis		
18253		

18254	5. FOR-NEXT-Schleife: Aufteilung der Adressen:
bis	Wie bei 1. FOR-NEXT-
18271	Schleife
18352	Ein/Ausgabepuffer (80 Byte)
bis	
18431	
24576	Erste 8 Ziffern des Displaypuffers
bis	(hochauflösend, vgl. Kapitel 4.11)
24615	
24634	Tastaturspeicher
23637	1. Teil der Betriebszustandsanzeigemarken
24640	Zweite 8 Ziffern des Displaypuffers
bis	
24679	
24680 LB	Rücksprungadresse aus der 1. Unterprogrammebene
24681 HB	
24682	Rücksprungadressen der 2.-10. Unterprogramm-
bis	ebene (Aufbau gleich wie bei den Adressen
24699	24680/24681)
24700	2. Teil der Betriebszustandsanzeigemarken

Wichtige Systemvariablen für den PC 126X

Adresse Funktion/Erklärung

8252	0: Umschaltung auf groß/klein gesetzt 255: Umschaltung auf groß/japanisch gesetzt
------	--

8253 1. Teil der Betriebszustandsanzeigemarken

8316 2. Teil der Betriebszustandsanzeigemarken

26064 RSV-Speicher
bis
26111

26145 LB Startadresse ESP

26146 HB

26147 LB Endadresse ESP

26148 HB

26156 Tastaturspeicher

26165 LB Zeigt auf die Speicherzelle direkt nach dem

26166 HB zuletzt ausgeführten BASIC-Befehl:

Nach Starten eines Programms: Zeigt auf den
BASIC-Speicher. Im Direktmodus: Zeigt auf den
Ein/Ausgabepuffer.

26177 An dieser Stelle ist das PASS-Wort abgelegt.

bis Es wird nach dem ASCII-Schlüssel decodiert.

26183

26192 Anzeigepuffer (ASCII-Codes)

bis

26239

26243 LB Rücksprungadresse aus der 1. Unterprogrammebene

26244 HB

26245 Rücksprungadressen der 2.-10. Unterprogramm-

bis ebene (Aufbau gleich wie bei den Adressen

26262 26243/26244)

- 26328 2: WAIT-Intervall eingeschaltet
 6: WAIT-Intervall ausgeschaltet
- 26329 0-31: PASS-Wort unwirksam, kein Programmschutz
 32-63: PASS-Wort wirksam, Programmschutz
- 26337 LB Zeiger auf den Anfang des BASIC-Programms
- 26338 HB
- 26339 LB Zeiger auf das Ende des BASIC-Programms
- 26340 HB
- 26341 LB Dauer des WAIT-Intervalls
- 26342 HB
- 26355 Nummer der aktuellen FOR-NEXT-Schleife:
 6=0; 24=1; 42=2; 60=3; 78=4; 96=5
- 26356 Nummer des aktuellen Unterprogramms:
 131=0; 133=1; 135=2; ...; 151=10
- 26364 LB Zeiger auf die Anfangsadresse der selbstdefinierten Variablen
- 26365 HB
- 26374 LB 1. FOR-NEXT-Schleife: Zeiger auf das Byte direkt vor der Anfangsadresse der Schleifenvariable
- 26375 HB
- 26376 1. FOR-NEXT-Schleife: Endwert der Schleife, bis codiert nach dem BCD-System
- 26382 26383 1. FOR-NEXT-Schleife: STEP-Wert der Schleife, bis codiert nach dem BCD-System
- 26389 26390 1. FOR-NEXT-Schleife: Zeiger auf das Byte direkt nach dem 1. Schleifenausdruck
- 26391 HB

26392 bis 26409	2. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT-Schleife
26410 bis 26427	3. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT-Schleife
26428 bis 26445	4. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT-Schleife
26446 bis 26463	5. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT-Schleife
26544 bis 26523	Ein/Ausgabepuffer (80 Byte)
8192 bis 8251	Erste 12 Ziffern des Displaypuffers (hochauflösend, vgl. Kapitel 4.11)
10240 bis 10299	Zweite 12 Ziffern des Displaypuffers
8256 bis 8315	Dritte 12 Ziffern des Displaypuffers
10304 bis 10363	Vierte 12 Ziffern des Displaypuffers

Wichtige Systemvariablen für den PC 1350

Adresse	Funktion/Erklärung
28166 LB	1. FOR-NEXT-Schleife: Zeiger auf das Byte
28167 HB	direkt vor der Anfangs- adresse der Schleifen- variable
28168 bis 28174	1. FOR-NEXT-Schleife: Endwert der Schleife, codiert nach dem BCD- System
27904 bis 27999	Anzeigepuffer (ASCII-Codes)
28175 bis 28181	1. FOR-NEXT-Schleife: STEP-Wert der Schleife, codiert nach dem BCD- System
28182 LB	1. FOR-NEXT-Schleife: Zeiger auf das Byte
28183 HB	direkt nach dem 1. Schleifenausdruck
28184 bis 28201	2. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT- Schleife
28202 bis 28219	3. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT- Schleife
28220 bis 28237	4. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT- Schleife
28238 bis 28255	5. FOR-NEXT-Schleife: Aufteilung der Adressen: wie bei 1. FOR-NEXT- Schleife

28336		Ein/Ausgabepuffer (80 Byte)
bis		
28415		
28417	LB	Zeiger auf den Anfang des BASIC-Programms
28418	HB	
28419	LB	Zeiger auf das Ende des BASIC-Programms
28420	HB	
28423	LB	Zeiger auf die Anfangsadresse der selbstdefinierten Variablen
28424	HB	
28426		An dieser Stelle ist das PASS-Wort abgelegt.
bis		Es wird nach dem ASCII-Schlüssel decodiert.
28432		
28435		2: WAIT-Intervall eingeschaltet 6: WAIT-Intervall ausgeschaltet
28436		0-31: PASS-Wort unwirksam, kein Programmschutz 32-63: PASS-Wort wirksam, Programmschutz
28448	LB	Zeiger auf die Speicherzelle direkt nach dem zuletzt ausgeführten BASIC-Befehl:
28449	HB	Nach dem Starten eines Programms: Zeigt auf den BASIC-Speicher. Im Direktmodus: Zeigt auf den Ein/Ausgabepuffer
28459		Nummer der aktuellen FOR-NEXT-Schleife: 6=0; 24=1; 42=2; 60=3; 78=4; 96=5
28460		Nummer des aktuellen Unterprogramms: 144=0; 146=1; 148=2; ...; 164=10
28503		Tastaturspeicher
28527		RSV-Speicher bis 28671

28816 LB Rücksprungadresse aus der 1. Unterprogrammebene
 28817 HB
 28818 Rücksprungadressen der 2.-10. Unterprogramm-
 bis ebene (Aufbau gleich wie bei den Adressen
 28835 26243/26244)
 28851 LB Dauer des WAIT-Intervalls
 28852 HB
 30780 Betriebszustandsanzeigemarken



3.3 Nützliche Maschinenprogrammaufrufe

Die unten aufgelisteten Programme sind für die Verwendung in BASIC-Programmen gedacht. Maschinenroutinen für die Verwendung in Maschinenprogrammen finden Sie im Anhang C.

Wichtige Maschinenprogramme für den PC 14XX

CALL 1434 Schaltet den PC 140X softwaremäßig aus.
 CALL 1479 Schaltet den PC 1421 softwaremäßig aus.
 CALL 1442 Schaltet das Display des PC 140X (für Grafik) ein.
 CALL 1448 Schaltet das Display des PC 1421 (für Grafik) ein.
 CALL 5208 Schaltet das Display (für Grafik) ein und wartet auf einen Tastendruck zur Fortführung des Programmes. Der Code dieser Taste wird im Tastaturspeicher (Adresse 24634) abgelegt.
 CALL 2755 Erzeugt einen kurzen Piepton. Wird dieser CALL-Befehl in einer FOR-NEXT-Schleife verwendet, können beliebig lange Töne erzeugt werden.
 (PC 140X)



CALL 2700 Erzeugt beim PC 1421 einen kurzen Piepton.
(PC 1421)

CALL 49321 Erzeugt beim PC 140X einen BEEP
CALL 55539 Erzeugt beim PC 1421 einen BEEP

Wichtige Maschinenprogramme für den PC 126X

- 
- CALL 1112 Schaltet den PC 126X softwaremäßig aus.
 - CALL 4414 Schaltet das Display ein und wartet auf einen Tastendruck zur Fortführung des Programmes. Der Code dieser Taste wird im Tastaturspeicher (Adresse 26156) abgelegt.
 - CALL 2094 Erzeugt einen kurzen Piepton. Wird dieser CALL-Befehl in einer FOR-NEXT-Schleife verwendet, können beliebig lange Töne erzeugt werden.
 - CALL 48908 Erzeugt einen BEEP

Wichtige Maschinenprogramme für den PC 1350

- 
- CALL 1240 Schaltet den PC 1350 softwaremäßig aus.
 - CALL 4618 Wartet auf einen Tastendruck zur Fortführung des Programmes. Der Code dieser Taste wird im Tastaturspeicher (Adresse 28503) abgelegt.
 - CALL 2379 Erzeugt einen kurzen Piepton. Wird dieser CALL-Befehl in einer FOR-NEXT-Schleife verwendet, können beliebig lange Töne erzeugt werden.
 - CALL 49430 Erzeugt einen BEEP

In Kapitel 4 (Tips und Tricks) werden wir wichtige Systemadressen und Maschinenprogrammaufrufe noch detailliert behan-

dehn und anhand von Beispielen auch zeigen, wie Sie diese Adressen und Maschinenprogramme in Ihre BASIC-Programme einbeziehen können.

3.4 Der BASIC-Interpreter

Schon in Kapitel 1.8 haben wir gesehen, wie der Rechner ein BASIC-Programm abspeichert.

Auf den folgenden Seiten wollen wir uns damit beschäftigen, wie der Rechner ein solches Programm umsetzt, damit er es dann auch ausführen kann.

Ein BASIC-Programm kann vom Computer nämlich nicht direkt so verwertet werden, wie es im Speicher abgelegt wurde, denn Ihr Rechner arbeitet intern nicht mit der Programmiersprache BASIC, sondern mit der sogenannten Maschinensprache; einer Sprache, deren Befehle direkt den Mikroprozessor ansprechen. Auch diese Befehle sind durch Zahlen zwischen 0 und 255 codiert.

Aber Vorsicht: Die Funktionen der Maschinensprachbefehle sind nicht identisch mit denen der BASIC-Befehle, obwohl auch das BASIC-Programm aus Codes zwischen 0 und 255 besteht (ASCII- und Interpretercodes). Also entspricht auch das nach der Eingabe in ASCII-Codes und Tokens umgewandelte BASIC-Programm nicht der Maschinensprache Ihres Computers. Versuchen Sie deshalb nie, ein BASIC-Programm zum Beispiel mit CALL BASIC-Anfangsadresse zu starten. Ihr Rechner wird mit allerhöchster Wahrscheinlichkeit 'abstürzen'.

Mit einem CALL-Befehl können und dürfen nur Maschinenprogramme aufgerufen werden!

Da also Ihr Rechner nicht direkt in BASIC arbeiten kann, müssen alle BASIC-Befehlsschritte zuerst übersetzt oder interpretiert werden. Sie müssen so umgeformt werden, damit sie der Prozessor direkt verstehen und ausführen kann.

Es gibt zwei Möglichkeiten, wie ein BASIC-Programm für den Prozessor zugänglich gemacht werden kann:

Übersetzung (Compilierung)

Das gesamte BASIC-Programm wird durch ein Compilerprogramm in die Maschinensprache übersetzt. Das aus dem BASIC-Programm entstandene Maschinenprogramm kann durch den Prozessor direkt ausgeführt werden und ist somit äußerst schnell. Dieses Verfahren ist v.a. bei größeren Rechnern, also bei Home und Personal Computern, sehr beliebt.

Interpretierung

Bei der Interpretierung wird das BASIC-Programm nicht in die Maschinensprache übersetzt, sondern die fertigen Maschinenspracheübersetzungen der einzelnen BASIC-Befehle befinden sich bereits im ROM des Rechners. Der BASIC-Interpreter greift nun der Reihe nach auf die einzelnen Bytes des BASIC-Programms zu. Findet er im Programm den Code eines BASIC-Befehls, so wird der Befehl nicht in die Maschinensprache übersetzt, sondern der Prozessor erhält anhand der Interpretcodetabelle die Anweisung, welches dieser bereits im ROM vorhandenen Maschinenprogramme er nun ausführen soll. Für jeden BASIC-Befehl braucht der Interpreter also eine Einsprungadresse in einen bestimmten ROM-Bereich.

Diese zweite Methode ist wesentlich langsamer als die der Compilierung, aber Interpreterprogramme lassen sich auch einfacher entwickeln als Compilerprogramme. Aus diesem und weiteren Gründen benutzen die SHARP Pocket Computer die zweite Methode. Ihr Computer besitzt also einen BASIC-Interpreter, keinen Compiler.

Aufbau des Interpreters

Der BASIC-Interpreter besteht zuerst einmal aus einem "Lexikon", worin die Namen der einzelnen BASIC-Befehle buchstabenweise und die dazugehörigen Interpretercodes gespeichert sind. Auf den Namen und den Interpretercode folgen zwei Bytes (High/Lowbyte), die die Anfangsadresse des Maschinenprogramms angehen, das aufgerufen wird, wenn der Rechner im BASIC-Programm den entsprechenden Interpretercode gefunden hat.

Die Maschinenprogramme, die bei diesen Adressen beginnen, bieten das Programm zur Ausführung der BASIC-Befehle in Maschinensprache.

Beispiel:

In der Befehlstabelle des Interpreters finden wir unter anderem folgende Angaben: RADIAN, 194, 49819 (PC 140X).

Name des Befehls:	RADIAN
Interpretercode:	194
Einsprungadresse des zu RADIAN gehörenden Maschinenprogramms:	49819 ¹⁾

Bei der Einsprungadresse von RADIAN beginnt das Maschinenprogramm, das die Winkelfunktion auf RADIAN umschaltet.

Dieses Maschinenprogramm wird dann aufgerufen, wenn der Interpreter im BASIC-Programmtext den Code 194 findet, der für RADIAN steht. Diese Einsprungadressen werden normalerweise nur intern benutzt. Mit dem BASIC-Befehl CALL aber können Sie diese Routine auch direkt aufrufen.

Geben Sie CALL 49819¹⁾ ein!

Resultat: Der Rechner schaltet auf RADIAN.

1) PC 1421: 88044, PC 136X: 49447, PC 1350: 49891

Findet der Rechner für einen Code des BASIC-Programms keine Einsprungadresse (weil dieser Code in der Tabelle des Rechners nicht vorhanden ist), so wird eine Fehlermeldung ausgegeben. Da der Computer aber bei der Eingabe einer Programmzeile einen Fehlertest durchführt, kann ein solcher Fehler überhaupt nicht auftreten, es sei denn, man benutzte den POKE-Befehl, um entsprechend falsche Interpretercodes in ein Programm zu setzen.

BASIC-Einsprungadressen im ROM

Bevor wir die Auflistung aller BASIC-Befehle der Interpretercodetabelle betrachten, sei hier noch ein detaillierteres Beispiel abgedruckt, das den Aufbau dieser Tabelle im ROM verdeutlichen soll:

Beispiel: Der BASIC-Befehl RUN

Adresse

140X	1421	126X	1350	Wert		Bedeutung
44139	50018	42003	42663	82		ASC "R"
44140	50019	42004	42664	85		ASC "U"
44141	50020	42005	42665	78		ASC "N"
44142	50021	42006	42666	176		Interpretercode für RUN
44143	50022	42007	42667	188	HB ¹⁾	Einsprungadresse von
44144	50023	42008	42668	252	LB ¹⁾	RUN: 48380 ¹⁾ 48380=256*128+252
44145	50024	42009	42669	198		Funktion nicht bek.
44146						Auflistung der Daten des nächsten Befehls

1) PC 1421: 54433 (212,161), PC 1350: 47664 (186,48) PC 126X: 47793 (186,177)

Vollständige Befehlstabelle des BASIC-Interpreters

PC 140X: 90 *

(Für BEEP verwendet man besser die Einsprungadresse 49321).

43666 AREAD	225	49930	43962 INKEY\$	173	3331
43675 AND	161	3331	43972 LIST	180	50272
43682 ABS	153	3331	43980 LLIST	181	50574
43689 ATN	159	3331	43989 LPRINT	226	50541
43696 ASN	157	3331	43999 LOG	146	3331
43703 ACS	158	3331	44006 LN	145	3331
43710 ASC	164	3331	44012 LET	214	49154
43717 AHS	141	3331	44019 LEN	166	3331
43724 AHC	142	3331	44026 LEFT\$	171	3331
43731 AHT	143	3331	44035 MEM	175	3331
43738 BEEP	196	49294	44042 MID\$	170	3331
43746 CONT	178	50085	44050 NEXT	217	48780
43754 CLEAR	201	49346	44058 NOT	163	3331
43763 CLOAD	183	40460	44065 NEW	177	48308
43772 CSAVE	182	40192	44072 ON	211	49835
43781 COS	150	3331	44078 OR	162	3331
43788 CHR\$	168	3331	44084 PRINT	222	50406
43796 CUR	137	3331	44093 PASS	179	49954
43803 CALL	204	49368	44101 PI	174	3331
43811 DIM	203	48471	44107 POL	130	3331
43818 DEGREE	193	49811	44114 PEEK	167	3331
43828 DEG	155	34058	44122 POKE	205	49397
43835 DMS	156	34065	44130 PAUSE	221	49740
43842 DECI	132	3331	44139 RUN	176	48380
43850 DATA	220	49930	44146 RETURN	227	49692
43858 END	216	49719	44156 READ	219	49462
43865 EXP	147	3331	44164 RESTORE	228	49587
43872 FOR	213	48618	44175 RND	160	3331
43879 FACT	144	3331	44182 RANDOM	192	49291
43887 GOTO	198	48999	44192 RIGHT\$	172	3331
43895 GOSUB	224	49046	44202 RADIAN	194	49819
43904 GRAD	195	49827	44212 REM	215	49685
43912 HEX	133	3331	44219 REC	129	3331
43919 HSN	138	3331	44226 ROT	131	3331
43926 HCS	139	3331	44233 RCP	135	3331
43933 HTN	140	3331	44240 STOP	218	49728
43940 INPUT	223	50117	44248 SQR	148	3331
43949 IF	212	49635	44255 SIN	149	3331
43955 INT	152	3331	44262 SGN	154	3331

44269	STR\$	169	3331	44315	TROFF	200	49736
44277	STEP	209	4788	44324	TO	208	4788
44285	SQU	136	3331	44330	TEN	134	3331
44292	THEN	210	4788	44337	USING	202	50362
44300	TAN	151	3331	44346	VAL	165	3331
44307	TRON	199	49732	44353	WAIT	197	49775

PC 1421:

49360	AREAD	225	56155	49630		229	17473
49369	AND	161	3303	49634	S	124	48908
49376	ABS	153	3303	49639		243	17742
49383	ATN	159	3303	49643		216	55944
49390	ASN	157	3303	49647	EXP	147	3303
49397	ACS	158	3303	49654	EFF	188	3303
49404	ASC	164	3303	49661	ERASE	229	41873
49411	AHS	141	3303	49670	FOR	213	54678
49418	AHC	142	3303	49677	FACT	144	3303
49425	AHT	143	3303	49685	FY	239	4784
49432	ACC	184	41956	49691	FIN	230	4784
49439	AMRT	185	41927	49698	GOTO	198	55102
49447	APR	189	3303	49706	GOSUB	224	55150
49454	BEEP	196	55512	49715	GRAD	195	56052
49462	BAL	245	4784	49723	HEX	133	3303
49469	BGNON	206	41863	49730	HSN	138	3303
49478	BGNOFF	207	41868	49737	HCS	139	3303
49488	CONT	178	56310	49744	HTN	140	3303
49496	CLEAR	201	55564	49751	INPUT	223	56343
49505	CLOAD	183	46070	49760	IF	212	55860
49514	CSAVE	182	45802	49766	INTE	244	4784
49523	COS	150	3303	49774	IRR	242	4784
49530	CHR\$	168	3303	49781	INT	152	3303
49538	CUR	137	3303	49788	INKEY\$	173	3303
49545	CALL	204	55594	49798	LIST	180	56497
49553	COMP	186	3303	49806	LLIST	181	56809
49561	CST	234	4784	49815	LPRINT	226	56776
49568	CFI	249	4784	49825	LOG	146	3303
49575	DIM	203	54530	49832	LN	145	3303
49582	DEGREE	193	56036	49838	LET	214	55258
49592	DEG	155	34137	49845	LEN	166	3303
49599	DMS	156	34144	49852	LEFT\$	171	3303
49606	DECI	132	3303	49861	MEM	175	3303
49614	DATA	220	56155	49868	MID\$	170	3303
49622	DAY\$	123	48652	49876	MAR	236	4784

49883 MU	237	4784	50071 RIGHTS	172	3303
49889 MDF	187	4784	50081 RADIAN	194	56044
49996 NEXT	217	54839	50091 REM	215	55910
49904 NOT	163	3303	50098 REC	129	3303
49911 NEW	177	54360	50105 ROT	131	3303
49918 NPV	241	4784	50112 RCP	135	3303
49925 NI	248	4784	50119 STOP	218	55953
49931 DN	211	56060	50127 SQR	148	3303
49937 OR	162	3303	50134 SGN	154	3303
49943 PRINT	222	56632	50141 STR\$	169	3303
49952 PASS	179	56179	50149 STEP	209	4784
49960 PI	174	3303	50157 SQU	136	3303
49966 POL	130	3303	50164 SEL	235	4784
49973 PEEK	167	3303	50171 SPRN	246	4784
49981 POKE	205	55623	50179 SINTE	247	4784
49989 PAUSE	221	55965	50188 SIN	149	3303
49998 PY	238	4784	50195 THEN	210	4784
50004 PMT	240	4784	50203 TAN	151	3303
50011 PRN	243	4784	50210 TRON	199	55957
50018 RUN	176	54433	50218 TROFF	200	55961
50025 RETURN	227	55917	50227 TO	208	4784
50035 READ	219	55688	50233 TEN	134	3303
50043 RESTORE	228	55812	50240 USING	202	56588
50054 RND	160	3303	50249 VAL	165	3303
50061 RANDOM	192	55509	50256 WAIT	197	56000

PC 126X:

41558 AREAD	225	49558	41691 CURSOR	207	48822
41567 AND	161	2687	41701 DIM	203	47909
41574 ABS	153	2687	41708 DEGREE	193	49439
41581 ATN	159	2687	41718 DEG	155	34899
41588 ASN	157	2687	41725 DMS	156	34906
41595 ACS	158	2687	41732 DATA	220	49558
41602 ASC	164	2687	41740 END	216	49307
41609 BEEP	196	48881	41747 EXP	147	2687
41617 CONT	178	49723	41754 EQU#	185	50070
41625 CLEAR	201	48933	41762 FOR	213	48057
41634 CLOAD	183	38268	41769 GOTO	198	48494
41643 CSAVE	182	37896	41777 GOSUB	224	48548
41652 COS	150	2687	41786 GRAD	195	49455
41659 CHR\$	168	2687	41794 INPUT	223	49756
41667 CALL	204	48955	41803 IF	212	49223
41675 CLS	206	48804	41809 INT	152	2687
41682 CHAIN	229	38193	41816 INKEY\$	173	2687

41826 LIST	180	49928	42003 RUN	176	47793
41834 LLIST	181	50501	42010 RETURN	227	49280
41843 LPRINT	226	50424	42020 READ	219	49049
41853 LOG	146	2687	42028 RESTORE	228	49175
41860 LN	145	2687	42039 RND	160	2687
41866 LET	214	48656	42046 RANDOM	192	48801
41873 LEN	166	2687	42056 RIGHTS	172	2687
41880 LEFT\$	171	2687	42066 RADIAN	194	49447
41889 MEM	175	2687	42076 REM	215	49273
41896 MEM#	186	2687	42083 STOP	218	49331
41904 MID\$	170	2687	42091 SQR	148	2687
41912 MERGE	184	38306	42098 SIN	149	2687
41921 NEXT	217	48220	42105 SGN	154	2687
41929 NOT	163	2687	42112 STR\$	169	2687
41936 NEW	177	47644	42120 STEP	209	4033
41943 ON	211	49463	42128 THEN	210	4033
41949 OR	162	2687	42136 TAN	151	2687
41955 PRINT	222	50259	42143 TRON	199	49335
41964 PASS	179	49582	42151 TROFF	200	49339
41972 PI	174	2687	42160 TO	208	4033
41978 PEEK	167	2687	42166 USING	202	50026
41986 POKE	205	48984	42175 VAL	165	2687
41994 PAUSE	221	49343	42182 WAIT	197	49403

PC 1350:

42124 AREAD	225	50108	42277 CHAIN	229	38348
42133 AND	161	2954	42286 CURSOR	207	55441
42140 ABS	153	2954	42296 DIM	203	48466
42147 ATN	159	2954	42303 DEGREE	193	49983
42154 ASN	157	2954	42313 DEG	155	35307
42161 ACS	158	2954	42320 DMS	156	35314
42168 ASC	164	2954	42327 DATA	220	50108
42175 BEEP	196	49403	42335 END	216	49897
42183 BASIC	236	47228	42342 EXP	147	2954
42192 CONT	178	50136	42349 FOR	213	48612
42200 CONSOLE	191	64143	42356 GOTO	198	49075
42211 CLEAR	201	47787	42364 GOSUB	224	49137
42220 CLOAD	183	38439	42373 GPRINT	231	59439
42229 CSAVE	182	38055	42383 GCURSOR	230	60134
42238 COS	150	2954	42394 GRAD	195	49999
42245 CHR\$	168	2954	42402 INPUT	223	50168
42253 CLOSE	188	64131	42411 IF	212	49736
42262 CALL	204	49455	42417 INT	152	2954
42270 CLS	206	55564	42424 INKEY\$	173	2954

42434 LIST	180	50355	42644 PRESET	235	60044
42442 LLIST	181	50605	42654 PAUSE	221	52835
42451 LPRINT	226	50526	42663 RUN	176	47664
42461 LOG	146	2954	42670 RETURN	227	49807
42468 LN	145	2954	42680 READ	219	49563
42474 LOAD	190	62275	42688 RESTORE	228	49695
42482 LINE	232	58795	42699 RND	160	2954
42490 LET	214	49255	42706 RANDOM	192	49400
42497 LEN	166	2954	42716 RIGHT\$	172	2954
42504 LEFT\$	171	2954	42726 RADIAN	194	49991
42513 MEM	175	2954	42736 REM	215	49796
42520 MID\$	170	2954	42743 STOP	218	49934
42528 MERGE	184	38480	42751 SQR	148	2954
42537 NEXT	217	48782	42758 SIN	149	2954
42545 NOT	163	2954	42765 SGN	154	2954
42552 NEW	177	48040	42772 STR\$	169	2954
42559 ON	211	50007	42780 SAVE	189	62055
42565 OPEN	187	63806	42788 STEP	209	4332
42573 OPEN\$	238	2954	42796 THEN	210	4332
42582 OR	162	2954	42804 TAN	151	2954
42588 PRINT	222	53217	42811 TEXT	237	47434
42597 PASS	179	47838	42819 TRON	199	49938
42605 PI	174	2954	42827 TROFF	200	49942
42611 PEEK	167	2954	42836 TO	208	4332
42619 POKE	205	49498	42842 USING	202	50481
42627 POINT	233	2954	42851 VAL	165	2954
42636 PSET	234	60053	42858 WAIT	197	49946

Kurze Erklärung zum Listing (hier für PC 140X):

44139	RUN	176	48380
-------	-----	-----	-------

Adresse, bei der die In- formationen für den ent- sprechenden Befehl im ROM beginnen.	Name des Befehls	Token	Einsprungadresse des zugehörigen Maschinenprogramms im ROM
---	------------------------	-------	---

Anstatt mit RUN können Sie also Ihr erstes BASIC-Programm ins Speicher auch direkt mit CALL 48380¹⁾ starten. Probieren Sie es aus!

Man sieht natürlich sofort, daß man nur sehr wenige dieser Einsprungadressen direkt und sinnvoll einsetzen kann. Die meisten BASIC-Befehle benötigen nach dem Namen ja noch weitere Parameterangaben, die einem CALL-Aufruf nicht einfach ohne weiteres mitgegeben werden können.

Es gibt allerdings Möglichkeiten, einem CALL-Aufruf mehrere Parameter anzuhängen und diese durch die Maschinensprache auszuwerten. Sie können somit Ihre eigenen BASIC-Befehlserweiterungen schreiben. Mehr erfahren Sie über diese interessante Möglichkeit in Kapitel 8.3 ('Eigene BASIC-Befehle').

3.5 Der Zeichengenerator

Neben dem BASIC-Interpreter gibt es noch weitere interessante Speicherzellen im ROM: Es sind diejenigen des Zeichengenerators. Von diesem Zeichengenerator erhält der Rechner die Informationen, wie die verschiedenen Zeichen auf der Anzeige aussiehen sollen.

Der Speicherblock des Zeichengenerators umfaßt alle diejenigen Zeichen, die auch in der ASCII-Zeichencodetabelle in Kapitel 1.9 enthalten sind; das ganze Alphabet, die Ziffern 0 bis 9, die Satz- und Sonderzeichen. Die Zeichen sind alle in der 5x7 Matrix der Anzeige abgespeichert.

Legte des Zeichengenerators:

PC 140X	32870 - 33219
PC 1421	32888 - 33261
PC 126X	32837 - 33676
PC 1350	32926 - 33779

1) PC 1431: 54433, PC 1360: 47084, PC 126X: 47793

Ein Beispiel:

Wie das Zeichen '2' gespeichert ist, zeigt folgende Abbildung:

<u>ADRESSEN:</u>					
PC 140X:	32985	32986	32987	32988	32989
PC 1421:	33003	33004	33005	33006	33007
PC 126X:	32927	32928	32929	32930	32931
PC 1350:	32996	32997	32998	32999	33000
BIT NR.:	↓	↓	↓	↓	↓
0 = $2^0 \rightarrow$					
1 = $2^1 \rightarrow$					
2 = $2^2 \rightarrow$					
3 = $2^3 \rightarrow$					
4 = $2^4 \rightarrow$					
5 = $2^5 \rightarrow$					
6 = $2^6 \rightarrow$					
WERTE:	70	73	81	97	66

Bit 7: Keine Funktion

Der Zeichengenerator kopiert nun, falls ein Zeichen auf die Anzeige gebracht werden soll, die entsprechenden Werte des geforderten Zeichens in den Displaypuffer. Aus der Darstellung ist ersichtlich, daß die Zeichen offenbar spiegelverkehrt abgespeichert sind.

Das folgende Beispielprogramm gibt einen Überblick über die Anfangsadressen der Zeichen. Nach dem Starten des Programms erscheint links in der Anzeige das erste Zeichen, wie es im

ROM abgespeichert ist, also spiegelverkehrt, in der Mitte wird das Zeichen so dargestellt, wie man es normalerweise auf der Anzeige sieht, und rechts steht die Anfangsadresse des betreffenden Zeichens: Nach einem Tastendruck erscheinen die 'Parameter' des nächsten Zeichens.

FUER PC 140X:

```

10:WAIT 0: PRINT ""
20:FOR I=32870 TO 33215
    STEP 5: PRINT I:
    FOR J=0 TO 4: POKE 2
        4606+J, PEEK (I+J):
        POKE 24675+J, PEEK (
            I+J): NEXT J
30:FOR J=24576 TO 24596
    STEP 5: POKE J,0,0,0
    ,0,0: NEXT J: CALL 5
208: NEXT I

```

FUER PC 1421:

```

10:WAIT 0: PRINT ""
20:FOR I=32888 TO 33281
    STEP 5: PRINT I:
    FOR J=0 TO 4: POKE 2
        4606+J, PEEK (I+J):
        POKE 24675+J, PEEK (
            I+J): NEXT J
30:FOR J=24576 TO 24596
    STEP 5: POKE J,0,0,0
    ,0,0: NEXT J: CALL 5
208: NEXT I

```

FUER PC 126X:

```

10:WAIT 0: PRINT ""
20:FOR I=32837 TO 33676
    STEP 5: PRINT I: FOR
        J=0 TO 4: POKE 8197+
        J, PEEK (I+J): POKE
        8212+(4-J), PEEK (I+
        J): NEXT J
30:CALL 4414: NEXT I

```

FUER PC 1350

```

10:WAIT 0: PRINT ""
20:FOR I=32906 TO 33779
    STEP 5: CLS : PRINT
    I: FOR J=0 TO 4:
        POKE 28672+J, PEEK (
            I+J): POKE 28687+(4-
            J), PEEK (I+J):
        NEXT J
30:CALL 4618: NEXT I

```

Vielleicht ist es Ihnen noch ein Rätsel, wie dieses Programm funktioniert. Trotzdem: Schon jetzt können Sie sehen, daß Ihr Rechner grafikfähig ist. Die spiegelverkehrten Zeichen zeigen dies ganz klar. Wie Grafik programmiert wird, zeigen wir Ihnen detailliert in Kapitel 4.11.

Kapitel 4

Tips und Tricks: BASIC professionell

Nachdem wir nun relativ viel Theorie behandelt haben, wollen wir den Schwerpunkt auf die Praxis legen. Wir möchten Ihnen verschiedene Tips und Tricks weitergeben, die es ermöglichen, Ihre Programme interessanter und effizienter zu gestalten. Verstehen Sie deshalb die abgedruckten Anwendungsbeispiele auch als eine Anregung zum Selberprogrammieren.

4.1 Programmertips für BASIC-Programme

Mit den folgenden Tips wird der PC optimal ausgenutzt. Sie können schnellere, übersichtlichere und bessere Programme schreiben, wenn Sie sich an folgende Punkte halten:

- Schreiben Sie möglichst viele Anweisungen in eine Zeile. Auf diese Weise können Sie Speicherplatz sparen. Trotzdem sollten Sie aber darauf achten, daß Ihr Programm übersichtlich und strukturiert bleibt.
- Entfernen Sie alle unnötigen REM-Anweisungen, sie verlangsamen die Bearbeitungsgeschwindigkeit unnötig.
- An der Stelle von REM können Sie auch das Anführungszeichen ("") verwenden. Dieses Zeichen benötigt vom Anfang an nur eine Speicherstelle. Das macht es einfacher, REM-Zeilen ganz aufzufüllen.
- Zur IF-THEN-Anweisung: Sicher haben Sie schon bemerkt, daß eine Anweisung wie z.B. 100 IF J=12 THEN Z=X+1 nicht so abläuft, wie Sie es sich gewünscht haben. Ist nämlich J=12, berechnet der PC nicht die Gleichung Z=X+1, sondern er gibt die Fehlermeldung ERROR 4 aus. Der Ausdruck Z=X+1 wird nämlich nicht als Gleichung, sondern als Zeilennummer interpretiert. Damit der Rechner nach der THEN-Anwei-

sung die Berechnung $Z=X+1$ ausführt, muß deshalb unbedingt noch der Zuweisungsbefehl LET eingeschoben werden. Also:

```
100: IF J=12 THEN LET Z=X+1
```

Diese Anweisung kann aber noch einfacher geschrieben werden. Es ist nämlich nicht nötig, den THEN-Befehl zu verwenden:

```
100: IF J=12 LET Z=X+1
```

- Wie Sie vielleicht gemerkt haben, können die meisten BASIC-Befehle bei der Programmeingabe abgekürzt werden. Zum Beispiel:

```
PRINT = P.  
CALL = CA.  
RUN = R.  
DEGREE = DE.  
usw.
```

Der Rechner nimmt dabei einfach den Befehl, der in der Interpretercodetabelle als erster mit den angegebenen Zeichen beginnt.

Es ist folglich nicht schwer, die Abkürzungen der anderen BASIC-Befehle herauszufinden.

- Einfache Regel zum High/Lowbyte-System:
Im ROM und im BASIC-Speicher (z.B. Zeilennummern und Länge der Variablenfelder) sind die Adressen im High/ Lowbyte-Format gespeichert:

1. Speicherzelle: Highbyte
2. Speicherzelle: Lowbyte

Im SystemvariablenSpeicher dagegen werden die Adressen (z.B. das WAIT-Intervall) im Low/Highbyte-Format gespeichert, also gerade umgekehrt:

1. Speicherzelle: Lowbyte
2. Speicherzelle: Highbyte

Anwendungsbeispiele

Mehrere Systemvariablen (Kapitel 3.1) und Maschinenprogrammaufrufe (Kapitel 3.2) erfordern noch eine genauere Beschreibung.

 Anhand einiger Beispiele wollen wir zeigen, wie Sie diese Kenntnisse in BASIC-Programmen verwenden können.

4.2 Betriebszustandsanzeigemarken

Am oberen und unteren Rand des Displays befinden sich mehrere kleinere Anzeigefelder, die den momentanen Betriebszustand des Rechners beschreiben (z.B. CAL, RUN, DEF, usw.).

Für jede einzelne Anzeigeeinheit ist ein bestimmtes Bit zuständig. Ist dieses Bit gesetzt, so leuchtet die dazugehörige Anzeige. Das Setzen eines Bits (1) bedeutet also 'Anzeige ein', das Löschen (0) bewirkt 'Anzeige aus'.



PC 140X:

SPEICHERZELLE : 24636

7	6	5	4	3	2	1	0
KEINE FUNKT.	KEINE FUNKT.	KEINE FUNKT.	KEINE FUNKT.	STAT	KEINE FUNKT.	KEINE FUNKT.	KEINE FUNKT.

SPEICHERZELLE : 24637

7	6	5	4	3	2	1	0
KEINE FUNKT.	CRL	RUN	PRO	HYP	SHIFT	DEF	BUSY

SPEICHERZELLE : 24700

7	6	5	4	3	2	1	0
KEINE FUNKT.	PRINT	DE	G	PRO	()	H	E

PC 1421:

SPEICHERZELLE: 24636

7	6	5	4	3	2	1	0
KEINE FUNKT	KEINE FUNKT	KEINE FUNKT	KEINE FUNKT	RUN	KEINE FUNKT	KEINE FUNKT	KEINE FUNKT

SPEICHERZELLE: 24637

7	6	5	4	3	2	1	0
KEINE FUNKT	PRINT	FIN	STAT	BGM	SHIFT	DEF	BUSY

SPEICHERZELLE: 24700

7	6	5	4	3	2	1	0
KEINE FUNKT	PRO	Σ	PRN	INT	BAL	H	E

PC 126X:

SPEICHERZELLE: 8253

7	6	5	4	3	2	1	0
KEINE FUNKT.	DEF	SHIFT	SHALL	JAPAN	KEINE FUNKT.	PRINT	BUSY

DAS BIT "JAPAN" SOLLTE NICHT GESETZT WERDEN!

SPEICHERZELLE: 8316

7	6	5	4	3	2	1	0
KEINE FUNKT.	FLAG	ERROR	KEINE FUNKT.	KEINE FUNKT.	GRAD	RAD	DEG

PC 1350:

SPEICHERZELLE: 36780

7	6	5	4	3	2	1	0
SM	JAPAN	PRO	RUN	KEINE FUNKT.	KEINE FUNKT.	DEF	SHIFT

Beispiel:

Mit diesen Kenntnissen können wir z.B. die DEF-Anzeige programmgesteuert einschalten. Durch einen darauffolgenden Tastendruck, z.B. auf die Taste A, wird danach das Teilprogramm mit dem Label A aufgerufen.

```
10: POKE 246371), 2^1+2^5
20: WAIT: PRINT "A, B ODER C":END
100: "A" PRINT "PROGRAMM A":END
200: "B" PRINT "PROGRAMM B":END
300: "C" PRINT "PROGRAMM C":END
```

Mit AND und OR können Sie so natürlich auch einzelne Bits setzen oder löschen (vgl. Kapitel 1.6).



4.3 Der Ein/Ausgabepuffer

Wenn Sie eine Eingabe tätigen (Programmzeile oder Direktbefehle), dann wird diese Eingabe im Eingabepuffer zwischengespeichert. Da die Größe dieses Puffers 80 Bytes beträgt, können auch die Zeilen, die Sie eingeben oder editieren, maximal 80 Zeichen lang sein.

Wo liegt der Sinn dieser Zwischenspeicherung? Solange eine Zeile editiert wird, befindet sie sich im Eingabepuffer. Nach Abschluß einer Eingabe mit der ENTER-Taste wird zuerst getestet, ob die Anweisung fehlerfrei eingegeben wurde. Wenn ja, so werden die Befehle ins Programm eingebaut oder direkt ausgeführt.

Mit dem folgenden Programm können Sie maximal 60 Zeichen lange mathematische Ausdrücke und Formeln direkt vom RUN-Modus aus in eine Programmzeile übernehmen.



Tippen Sie nun das Programm 'Eingabepuffer' ein und schalten Sie nach der Eingabe wieder in den RUN-Modus zurück!

Wenn Sie nun z.B. die Funktion X*LOG 12+17 eingeben und die Eingabe mit der ENTER-Taste abschließen, danach mit '' wieder zurückholen und mit DEF F das Programm starten, so wird das Programm bei Zeile 10 ausgeführt, während sich Ihre Eingabe X*LOG 12+17 immer noch im Eingabepuffer befindet. In

1) PC 1421: POKE 24637,2^1, PC 126X: POKE 8253,2^6, PC 1350:
POKE 30780,2^1+2^4

der Zeile 10 wird nun der Puffer in einer FOR-NEXT-Schleife mittels PEEK zeichenweise ausgelesen und nach Zeile 1 kopiert. Sobald der Rechner im Eingabepuffer eine 13 (13 bedeutet Ende der Eingabe), wird zu Zeile 50 gesprungen.

Falls keine Funktion eingegeben wurde, bleibt die Zeile 1 unverändert, andernfalls wird der Funktion in Zeile 1 nun der Zusatz "REM FUNKTION" angehängt. Dieser REM-Befehl bewirkt, daß die nachfolgenden Sternchen (*), die den Platz für die Funktion reservieren, ignoriert werden.

Programm "Eingabepuffer"

(Zeile 1: Ganze Zeile mit "*" auffüllen!)

FÜR PC 140X1

```

J1:Y=*****+*****+*****+
      +*****+*****+*****+
      +*****+*****+*****+
      +*****+*****+*****+
50:RETURN
10: "F" FOR I=0 TO 59:J=
      PEEK (18352+I): IF J
      =13 THEN 50
20:POKE 8198+I,J: NEXT
   I
50:IF I POKE 8198+I,58:
      215,70,85,78,75,84,7
      3,79,78
100:REM HAUPTPROGRAMM

```

FÜR PC 126X1

```

J1:Y=*****+*****+*****+
      +*****+*****+*****+
      +*****+*****+*****+
      +*****+*****+*****+
50:RETURN
10:"F" FOR I=0 TO 59:J=
      PEEK (26544+I): IF J
      =13 THEN 50
20:POKE 16518+I,J:
      NEXT I
50:IF I POKE 16518+I,58:
      ,215,70,85,78,75,84,
      73,79,78
100:REM HAUPTPROGRAMM

```

FUER PC 1350 (4K):

```
1: Y=*****  
*****  
*****  
*****  
*****  
5:RETURN  
10:"F" FOR I=0 TO 59:J=  
PEEK (28336+I): IF J  
=13 THEN 50  
20:POKE 24630+I,J:  
NEXT I  
50:IF I POKE 24630+I,58,  
,215,70,85,78,75,84,  
73,79,78  
100:REM HAUPTPROGRAMM
```

FUER PC 1350 (12&20K):

```
1: Y=*****  
*****  
*****  
*****  
*****  
5:RETURN  
10:"F" FOR I=0 TO 59:J=  
PEEK (28336+I): IF J  
=13 THEN 50  
20:POKE 8246+I,J: NEXT  
I  
50:IF I POKE 8246+I,58,  
,215,70,85,78,75,84,7  
3,79,78  
100:REM HAUPTPROGRAMM
```

Geben Sie X*LOG 12+17 ein, drücken Sie ENTER, dann die ',-Taste, und starten Sie das Programm mit DEF F. In Zeile 1 steht nun folgendes:

```
1: Y=X*LOG 12+17: REM FUNKTION*****
```

Das Unterprogramm in den Zeilen 1 bis 5 kann vom Hauptprogramm (ab Zeile 100) aufgerufen werden.

4.4 Der Tastaturspelcher

Bei Tastatureingaben wird der ASCII- oder Interpretercode der zuletzt gedrückten Taste in einer bestimmten Speicherzelle abgelegt. Beim PC 14XX ist das die Adresse 24634, beim PC 126X die Adresse 26156 und beim PC 1350 schließlich die Adresse 28503. Mit Hilfe dieser Systemvariable kann man elegante Tastaturabfragen programmieren, und zwar ohne INKEY\$.

Das Maschinenprogramm, das beim PC 14XX mit CALL 5208, beim PC 126X mit CALL 4414 und beim PC 1350 mit CALL 4618 aufgerufen wird, schaltet die Anzeige ein und wartet, bis eine beliebige Taste gedrückt wird. Der Code dieser Taste wird dann in der oben erwähnten Speicherzelle abgespeichert und kann mit dem PEEK-Befehl ausgelesen werden. Mit dieser Kombination können im Gegensatz zur INKEY\$-Anweisung fast alle Tasten, also auch solche wie SIN, BASIC, CAL usw. abgefragt werden. Die Codes der einzelnen Tasten finden Sie in der ASCII- und Interpretercodetabelle in Kapitel 1.9.

Auf der folgenden Abbildung sind die Adressen für die einzelnen Rechner nochmals übersichtlich wiedergegeben:

Rechner	Tastaturabfrage (beginnend bei Speicherzelle)	Tastencode in Speicherzelle
PC 14XX	5208	24634
PC 126X	4414	26156
PC 1350	4618	28503

Beispiel:

```
10: WAIT 0:PRINT "TASTE?":CALL 52081)
    WAIT: PRINT PEEK 246341):GOTO 10
```

Wenn Sie das Programm starten, zeigt der Rechner den Displayinhalt an und wartet auf einen Tastendruck. Drücken Sie nun eine beliebige Taste (z.B. 'A') und der Rechner zeigt Ihnen den Code dieser Taste (65 für 'A').

Einfaches Beispiel einer Tastaturabfrage:

```
200:WAIT 0: PRINT "S ODE
    R C?": CALL 5208:X=
    PEEK 24634
210:IF X= ASC "S" GOSUB
    250: PRINT "S GEDRUE
    CKT": END
220:IF X= ASC "C" GOSUB
    250: PRINT "C GEDRUE
    CKT": END
230:GOTO 200
250:PAUSE "SIE HABEN... "
    : WAIT : RETURN
```

Bitte beachten Sie die für bestimmten Rechner zu ändernde Zeile²⁾

Dieses Abfragesystem hat den Vorteil, daß die Anzeige auch während der Eingabe weder erlischt noch blinkt, wie dies sonst beim INKEY\$-Befehl der Fall ist; und daß die Reaktion auf den Tastendruck blitzartig erfolgt.

-
- 1) Adresse nur für den PC 14XX gültig; vergleichen Sie mit der Tabelle oben.
 - 2) Zeile 200: PC 126X: 4414, PC 1350: 4618 (CALL)
Zeile 200: PC 126X: 26156, PC 1350: 28503 (PEEK)

4.5 Rückgängigmachen von NEW

Der BASIC-Befehl NEW löscht den ganzen BASIC-Speicher und bereitet diesen somit auf die Eingabe eines neuen Programms vor. Wenn Sie den Befehl NEW eingeben, geschieht folgendes:

- In die erste Speicherzelle des BASIC-Programms (High-byte d. Zeilennummer) wird eine 255 geschrieben. Wie Sie bereits wissen, beginnt und endet ein BASIC-Programm mit der Zahl 255. So existiert nach NEW für den Rechner kein Programm mehr, da die beiden 255 unmittelbar aufeinanderfolgen.
- Die Endadresse des BASIC-Programms ist beim PC 14XX in den Systemadresse 18147 (HB) und 18148 (HB), beim PC 126X in den Adressen 26339 und 26340 und beim PC 1350 in den Adressen 28419 und 28420 gespeichert. Diese Endadresse bekommt den Wert der Anfangsadresse des BASIC-Programms plus 1. Sie zeigt somit immer auf die zweite 255.

Das Programm ist nach der Eingabe von NEW jedoch immer noch vorhanden, es wurde nämlich gar nicht gelöscht, sondern der Rechner 'kennt' es nur nicht mehr, weil eine 255 dazwischengesetzt wurde.

Es ist deshalb möglich, ein Programm, das versehentlich mit NEW gelöscht wurde, durch Überschreiben der zweiten 255 mit 0 wieder zurückzuholen.

Geben Sie deshalb für die Wiederherstellung Ihres Programms folgendes ein:

PC 14XX:	POKE 8193,0
PC 126X:	POKE 16385,0 (EQU#=0)
PC 1350:	POKE 24625,0 (4K RAM)
	POKE 16433,0 (12K RAM)
	POKE 8241,0 (20K RAM)

Vorsicht: Da die Zeiger auf das BASIC-Ende falsche Werte haben, kann es unter Umständen zu einem Systemabsturz kommen (vor allem dann, wenn Zeilen editiert und verändert werden). Das Programm kann auch nicht mehr auf Band gespeichert werden. Es kann aber problemlos mit RUN, GOTO oder DEF gestartet werden.

Beim PC 1350 wird beim Ausschalten die 255 wieder zurückgeschrieben. Durch die gleiche Methode wie oben kann jedoch das Programm nach dem Einschalten erneut zurückgeholt werden.

Natürlich können die Systemvariablen, in denen das Ende des BASIC-Programms steht, auch wiederhergestellt werden. Genau das macht dann auch das Maschinenprogramm RENEW, das ein BASIC-Programm völlig restauriert. Sie finden dieses Programm in Kapitel 9.5.

4.6 Ersetzen von Codes

Mit dem folgenden Hilfsprogramm ist es möglich, bestimmte Zeichen oder auch BASIC-Befehle durch beliebige andere Zeichen oder Befehle zu ersetzen:

FUER PC 14XX:

```
59999:@  
60000:"H"X=8196: INPUT "ALTES ZEICHEN: ";A  
,"NEUES ZEICHEN: "  
;N  
60010:IF PEEK X=64 END  
60020:FOR I=X TO PEEK (X  
-1)+X: IF PEEK I=A  
POKE I,N  
60030:NEXT I:X=I+2:  
GOTO 60010
```

FUER PC 126X:

```
59999:@  
60000:"H"X=16516: INPUT "ALTES ZEICHEN: ";  
A,"NEUES ZEICHEN: "  
;"N  
60010:IF PEEK X=64 END  
60020:FOR I=X TO PEEK (X  
-1)+X: IF PEEK I=A  
POKE I,N  
60030:NEXT I:X=I+2:  
GOTO 60010
```

FUER PC 1350 (4K):

```

59999:@
60000:"H"X=24628: INPUT
    "ALTES ZEICHEN: ";
    A,"NEUES ZEICHEN:
    ";N
60010:IF PEEK X=64 END
60020:FOR I=X TO PEEK (X
    -1)+X: IF PEEK I=A
    POKE I,N
60030:NEXT I:X=I+2:
    GOTO 60010

```

FUER PC 1350 (12&20K):

```

59999:@
60000:"H"X=8244: INPUT "
    ALTES ZEICHEN: ";A
    , "NEUES ZEICHEN: "
    ;N
60010:IF PEEK X=64 END
60020:FOR I=X TO PEEK (X
    -1)+X: IF PEEK I=A
    POKE I,N
60030:NEXT I:X=I+2:
    GOTO 60010

```



Geben Sie nun ein kurzes Zusatzprogramm ein, dessen Zeilennummern kleiner sind als die des oben gelisteten Hilfsprogramms. In Ihrem Zusatzprogramm sollten Sie ein paar PRINT-Anweisungen verwenden.

Das Zusatzprogramm könnte z.B. so aussehen:

```

200:PRINT "ZUSATZPROGRAMM":PRINT "ZUM"
210:PRINT "HILFSPROGRAMM":PRINT "ZUM ERSETZEN"
220:PRINT "VON CODES":END

```

Alle diese PRINT-Anweisungen sollen nun durch das Hilfsprogramm in PAUSE-Befehle umgewandelt werden.

Starten Sie das Hilfsprogramm mit DEF H.

Das 'alte' Zeichen, das ersetzt werden soll, ist PRINT. Geben Sie den Interpretercode von PRINT (222) ein. Das neue Zeichen ist PAUSE; die PRINT-Befehle sollen ja alle durch PAUSE-Befehle ersetzt werden. Geben Sie den Interpretercode von PAUSE (221) ein.



Wenn Sie nun nach Beendigung des Hilfsprogramms Ihr Zusatzprogramm betrachten, so werden Sie sehen, daß alle PRINT-Anweisungen in PAUSE-Befehle umgewandelt wurden. Der Klammeraffe in Zeile 59999 ('@') stellt das Abbruchkriterium

für den Austauschprozeß dar. Der Rechner fährt mit dem Austausch also nur solange fort, bis er das '@'-Zeichen findet.

4.7 Sonderzeichen

Wie aus der ASCII-Codetabelle (Kapitel 1.9) ersichtlich ist, existieren noch einige Sonderzeichen, die nicht über die Tastatur und manchmal nicht einmal über die CHR\$-Funktion sichtbar gemacht werden können. Mit dem POKE-Befehl aber kann dieser Mangel überbrückt werden.

Beispiel:

Die Variable Z\$ soll mit dem Sonderzeichen ']' (Code 91) gefüllt werden.

Die Variable Z beginnt bei Speicherzelle 17872¹⁾. Z muß nun aber als Textvariable Z\$ interpretiert werden, damit Sonderzeichen gespeichert werden können. Damit Z als Textvariable gedeutet wird, schreiben wir also in die erste Speicherzelle von Z eine 245:

POKE 17872¹⁾,245

Nun müssen Sie nur noch die restlichen Speicherzellen von Z\$ mit dem Code 91 auffüllen, der für ']' steht:

POKE 17873²⁾,91,91,91,91,91,91,91

Schauen Sie nach, was nun in Z\$ gespeichert ist: Geben Sie 'Z\$' ein und drücken Sie die ENTER-Taste!

Auf dieselbe Weise können Sie auch BASIC-Befehle in Variablen speichern. Geben Sie z.B. ein:

POKE 17873²⁾,222,0

1) PC 1421: 17792, PC 126X: 25856, PC 1350: 27696

2) PC 1421: 17793, PC 126X: 25857, PC 1350: 27697

In Z\$ befindet sich nun der Interpretercode für PRINT.

ASCII-Codes

Mit dem nächsten kurzen Hilfsprogramm können Sie ein Zeichen oder ein BASIC-Befehlswort ermitteln, dessen ASCII- oder Interpretercode Sie zuvor eingegeben haben:

FUER PC 140X:

```
60100:"C" INPUT "CODE: "
;C: POKE 17872,245
,C,0: PRINT C;""
"+Z$: GOTO 60100
```

FUER PC 1421:

```
60100:"C" INPUT "CODE: "
;C: POKE 17872,245
,C,0: PRINT C;""
"+P$: GOTO 60100
```

FUER PC 126X:

```
60100:"C" INPUT "CODE: "
;C: POKE 25856,245
,C,254,C,0: PRINT
C;" "+Z$: GOTO 6
0100
```

FUER PC 1350:

```
60100:"C" INPUT "CODE: "
;C: POKE 27696,245
,C,0: PRINT C;""
"+Z$: GOTO 60100
```

Der Code, den Sie eingegeben haben, wird als ASCII- oder Interpretercode mittels POKE in die Variable Z\$ hineingeschrieben. Der Code mit dem zugehörigen Zeichen wird dann angezeigt.

Starten Sie das Programm mit DEF C. Geben Sie einen Code ein, z.B. 201, und drücken Sie ENTER!

Ergebnis: 201 ist der Interpretercode von CLEAR.

4.8 Nützliches über Programmzeilen

Mit dem folgenden Hilfsprogramm können Sie die Anfangsadresse jeder beliebigen Programmzeile ermitteln:

FUER PC 14XX:

```
60200:"Z" INPUT "ZEILENN  
UMMER: ";Z:I=8193  
60210:X=256* PEEK I+  
PEEK (I+1): IF X<Z  
LET I=I+3+ PEEK (I  
+2): GOTO 60210  
60220:PRINT "ZEILENNR.:  
";X: PRINT "ADRESS  
E: ";I: END
```

FUER PC 126X:

```
60200:"Z" INPUT "ZEILENN  
UMMER: ";Z:I=16513  
60210:X=256* PEEK I+  
PEEK (I+1): IF X<Z  
LET I=I+3+ PEEK (I  
+2): GOTO 60210  
60220:PRINT "ZEILENNR.:  
";X: PRINT "ADRESS  
E: ";I: END
```

FUER PC 1350 (4K):

```
60200:"Z" INPUT "ZEILENN  
UMMER: ";Z:I=24625  
60210:X=256* PEEK I+  
PEEK (I+1): IF X<Z  
LET I=I+3+ PEEK (I  
+2): GOTO 60210  
60220:PRINT "ZEILENNR.:  
";X: PRINT "ADRESS  
E: ";I: END
```

FUER PC 1350 (12&20K):

```
60200:"Z" INPUT "ZEILENN  
UMMER: ";Z:I=8241  
60210:X=256* PEEK I+  
PEEK (I+1): IF X<Z  
LET I=I+3+ PEEK (I  
+2): GOTO 60210  
60220:PRINT "ZEILENNR.:  
";X: PRINT "ADRESS  
E: ";I: END
```

Dieses Hilfsprogramm liest mittels PEEK in einer Schleife die Zeilennummern der im Programm vorkommenden Zeilen und vergleicht die ausgelesenen Daten mit dem vorgegebenen Wert der von Ihnen eingegebenen Zeilenummer. Stimmen beide Werte überein, so wird die Anfangsadresse der betreffenden Zeilenummer ausgegeben.

Starten Sie das Hilfsprogramm mit DEF Z.

Wir wollen z.B. die Anfangsadresse der Zeile 60220 des Hilfsprogramms ermitteln:

Geben Sie also 60220 ein!

Als Ergebnis gibt der Rechner nun die Adresse I aus, bei der die Programmzeile 60220 beginnt.

Zeile 60220 soll nun in Zeile 60225 umgewandelt werden:

Adresse I: 235 Highbyte
Adresse I+1: 60 Lowbyte der Zahl 60220
($235 \cdot 256 + 60 = 60220$)



Um diese Zeilennummer auf 60225 zu erhöhen, müssen Sie lediglich das Lowbyte um 5 vergrössern, geben Sie also ein:

POKE I+1,65

Anstelle der Zeile 60220 finden Sie nun die Zeile 60225. Schauen Sie im Programm nach!

Dieser Trick, das programmgesteuerte Ändern der Zeilennummern, ist für das nächste Anwendungsbeispiel von großer Bedeutung.



4.9 Renumber

Das folgende Hilfsprogramm renumeriert die Zeilennummern eines BASIC-Programms im Speicher. Dies ist dann sinnvoll, wenn Sie noch weitere Zeilen in Ihr Programm einfügen wollen oder müssen, aber die Intervalle zwischen den Zeilennummern zu klein sind:

z.B. 11:PRINT "RESULTAT 1=";X1
12:PRINT "RESULTAT 2=";X2

Zwischen diese beide Zeilen können Sie keine dritte mehr einfügen, außer Sie renumerieren die Zeilennummern dieses Pro-

gramms, so daß größere Intervalle zwischen den einzelnen Zeilennummern vorhanden sind.

FUER PC 14XX:

```
59999:@  
60300:"M"X=8193: INPUT "  
    ZEILENNUMMER: ";Z,  
    "SCHRITTWEITE: ";S  
60310:IF PEEK (X+3)=64  
    END  
60320:HB= INT (Z/256):LB  
    =Z-256*HB: POKE X,  
    HB,LB  
60330:Z=Z+S:X=X+3+ PEEK  
    (X+2): GOTO 60310
```

FUER PC 126X:

```
59999:@  
60300:"M"X=16513: INPUT "  
    ZEILENNUMMER: ";Z,  
    "SCHRITTWEITE: ";  
    S  
60310:IF PEEK (X+3)=64  
    END  
60320:HB= INT (Z/256):LB  
    =Z-256*HB: POKE X,  
    HB,LB  
60330:Z=Z+S:X=X+3+ PEEK  
    (X+2): GOTO 60310
```

FUER PC 1350 (4K):

```
59999:@  
60300:"M"X=24625: INPUT "  
    ZEILENNUMMER: ";Z,  
    "SCHRITTWEITE: ";  
    S  
60310:IF PEEK (X+3)=64  
    END  
60320:HB= INT (Z/256):LB  
    =Z-256*HB: POKE X,  
    HB,LB  
60330:Z=Z+S:X=X+3+ PEEK  
    (X+2): GOTO 60310
```

FUER PC 1350 (12&20K):

```
59999:@  
60300:"M"X=8241: INPUT "  
    ZEILENNUMMER: ";Z,  
    "SCHRITTWEITE: ";S  
60310:IF PEEK (X+3)=64  
    END  
60320:HB= INT (Z/256):LB  
    =Z-256*HB: POKE X,  
    HB,LB  
60330:Z=Z+S:X=X+3+ PEEK  
    (X+2): GOTO 60310
```

Nach dem Starten des Hilfsprogramms mit DEF M geben Sie zuerst die Zeilennummer ein, mit der das zu renumerierende Programm dann beginnen soll. Bei der Abfrage der Schrittweite wird der Abstand zwischen zwei aufeinanderfolgenden Zeilennummern eingegeben.

Der Klammeraffe ('@') in Zeile 59999 verhindert, daß sich das Renumberprogramm selbst neu numeriert.

Wichtig: Bei diesem Hilfsprogramm werden lediglich die Zeilenummern, nicht aber die Sprungbefehle (GOTO, GOSUB und ON..GOTO) neu numeriert. Sie müssen dies entweder nachträglich manuell ändern oder andernfalls konsequent nur Sprungbefehle mit Labels verwenden (z.B. GOTO "START").

4.10 Knacken des PASS-Worts

Das PASS-Wort, mit dem Sie Ihre Programme gegen den Zugriff Unbefugter schützen können, ist gar keine so sichere Sache, wie es vielleicht scheinen mag. Schließlich muß ja auch dieses Codewort irgendwo im RAM abgespeichert sein.

Für BASIC-Anwender oder für Laien, die nichts über die Bedeutung der Systemvariablen wissen, ist der PASS-Befehl ein wirksamer Schutz gegen Softwareraub oder Manipulation an eigenen Programmen. Sehr leicht aber läßt sich dieses PASS-Wort abändern oder auch ganz beseitigen, selbst wenn Sie nicht wissen, mit welchem Namen das Programm gesichert wurde.

Das PASS-Wort wird in den folgenden Speicherzellen abgespeichert:

Rechner	Speicherzellen	
PC 14XX	18113	18119
PC 126X	26177	26183
PC 1350	28426	28432

Die Bedeutung dieser Adressen wollen wir nun anhand eines kleinen Beispiels erläutern:

Schützen Sie ein beliebiges Programm (dasjenige, das sich gerade im Speicher Ihres Rechners befindet) mit PASS "ABC". Lesen Sie

nun mit dem PEEK-Befehl die oben aufgelisteten Speicherzellen aus.

Das ergibt diese Tabelle:

Adresse				
14XX	126X	1350	Wert	Bedeutung
18113	26177	28426	65	ASC "A"
18114	26178	28427	66	ASC "B"
18115	26179	28428	67	ASC "C"
18116	26180	28429	0	0 bedeutet: Ende des PASS-Worts
...	
18119	26183	28432	0	Letzte Stelle des PASS-Worts, hier nicht verwendet.

Geben Sie nun die folgende Anweisung ein:

Zuerst: PC 14XX: A = 18113
PC 126X: A = 26177
PC 1350: A = 28426

Und dann: POKE A,72,65,76,76,79,0

H A L L O

Um das Programm, das Sie soeben mit PASS "ABC" geschützt haben, wieder sichtbar zu machen, müssen Sie nun anstelle von PASS "ABC" das neue PASS-Wort PASS "HALLO" eingeben.

Das PASS-Wort kann also mit PEEK aus den oben genannten Speicherzellen gelesen und mit POKE verändert werden. Auch das PASS-Wort wird nach dem ASCII-Schlüssel (Kapitel 1.9) codiert. Es wird also ähnlich wie die Textvariablen abgespeichert.

Um ein Programm zu sichern oder zu entsichern, müssen Sie aber immer noch manuell, d.h. im Direktmodus, PASS "HALLO" eingeben, weil der Befehl PASS im Programm nicht benutzt werden kann.

Es gibt noch einen zweiten, weit komfortableren Weg, das PASS-Wort einz- oder auszuschalten: Es existiert nämlich eine bestimmte Speicherzelle, deren Inhalt angibt, ob ein Programm geschützt ist oder nicht. Mittels eines POKE-Befehls, der den Wert dieser Adresse ändert, kann das PASS-Wort auf einfache Weise aktiviert oder ausgeschaltet werden:

Adresse		Wert	Bedeutung
14XX	126X	1350	
18137	26329	28436	0-31 PASS-Wort nicht aktiv
18137	26329	28436	32-63 PASS-Wort aktiv

Schützen Sie das Programm, das sich gerade in Ihrem Rechner befindet, mit irgendeinem beliebigen PASS-Wort. Geben Sie daraufhin folgendes ein:

PC 14XX: POKE 18137,16
 PC 126X: POKE 26329,16
 PC 1350: POKE 26329,16

Der Programmschutz ist nun aufgehoben.

Achtung: Achten Sie unbedingt darauf, daß Sie nicht aus Versagen das Bit 1 (21) dieser Adresse setzen: Dies könnte unter Umständen zu einem ALL RESET des Systems führen!

Wenn Sie in die gleiche Speicherzelle wieder eine 32 hineinschreiben, ist Ihr Programm wieder geschützt. Es spielt keine Rolle, ob im Speicherbereich, wo das PASS-Wort abgelegt ist, wirklich ein sinnvoller Name steht. Das PASS-Wort kann durchaus auch aus lauter Nullen (Code 0) oder andern unsinnigen

Codes bestehen. Es gilt: Wenn in Adresse 18137¹⁾ das Bit 5 (25=32) gesetzt ist, kann das Programm nicht gelistet werden.

4.11 Hochauflösende Grafik

Wie man sofort sieht, werden alle Zeichen, die auf der Anzeige dargestellt werden, aus einzelnen Punkten zusammengesetzt. Diese Zusammensetzung wird intern vom Zeichengenerator (vgl. Kapitel 3.5) vorgenommen. Der Zeichengenerator setzt alle nötigen Punkte, damit das von Ihnen gewünschte Zeichen auf der Anzeige erscheint.

Da im Rechner nur beschränkt viele verschiedene Zeichenformen gespeichert sind, sind auch Ihrer Phantasie enge Grenzen gesetzt.

Mit bestimmten Befehlen aber, mit denen Sie die einzelnen Punkte direkt ansteuern können, haben Sie die Möglichkeit, den Zeichengenerator im ROM zu umgehen und beliebige eigene Zeichen zu definieren.

Vorerst müssen wir aber noch ein bißchen Theorie betreiben: Jeder Spalte der Anzeige (ein Zeichen besteht aus 5 Spalten (vertikal) und 7 Zeilen (horizontal)) ist eine bestimmte RAM-Adresse zugeordnet. Der Inhalt einer solchen Speicherzelle bestimmt eindeutig, welche Punkte der entsprechenden Anzeigespalte ein- oder ausgeschaltet sind.

Die einzelnen Punkte der betreffenden Spalte sind direkt mit den Bits der zugehörigen Spaltenadresse verbunden: Ist Bit 0 gesetzt, so leuchtet der oberste Punkt der Spalte, ist Bit 1 gesetzt, so ist der zweitoberste Punkt eingeschaltet usw. Durch eine sinnvolle Kombination können Sie nun beliebige Zeichen oder Figuren entwerfen.

1) PC 126X: 26329, PC 1350: 28436

Beispiel zur Berechnung des Wertes einer Anzeigespalte:

BIT NR.:		DISPLAY- SPALTE:
0	$\rightarrow 2^0 = 1$	$\rightarrow 1 \rightarrow$
1	$\rightarrow 2^1 = 2$	$\rightarrow 2 \rightarrow$
2	$\rightarrow 2^2 = 4$	
3	$\rightarrow 2^3 = 8$	$\rightarrow 8 \rightarrow$
4	$\rightarrow 2^4 = 16$	$\rightarrow 16 \rightarrow$
5	$\rightarrow 2^5 = 32$	
6	$\rightarrow 2^6 = 64$	$\rightarrow 64 \rightarrow$
		
SUMME:		<u>91</u>

Anmerkung: Bit 7 hat nur beim PC 1350 eine Funktion.

Mit POKE Spaltenadresse,91 können Sie nun also die obenstehende Spalte auf der Anzeige sichtbar machen (bei verschiedenen Rechnern muß zusätzlich noch das Display eingeschaltet werden).

Die folgenden Darstellungen zeigen die Speicherbelegungen der Anzeigen der verschiedenen Rechnertypen. Soll zum Beispiel beim PC 14XX der oberste Punkt der ersten Spalte des sechsten Zeichens aufleuchten, so muß das Bit 0 der Adresse 24601 gesetzt werden (POKE 24601,1).

Die 48stellige Anzeige des PC 126X:

1	2	3	4	5	6	7	8	9	10	11	12
8256	8257-										
8258	8259										
8260											
8261-											
8262											
8263-											
8264											
8265											
8266											
8267											
8268											
8269											
8270											
8271											
8272											
8273											
8274											
8275											
8276											
8277											
8278											
8279											
8280											
8281											
8282											
8283-											
8284											
8285											
8286											
8287											
8288											
8289											
8290											
8291											
8292											
8293											
8294											
8295											
8296											
8297-											
8298											
8299											
8300											
8301											
8302											
8303											
8304											
8305											
8306											
8307											
8308											
8309											
8310											
8311-											
8312											
8313-											
8314											
8315-											
8350											
8351-											



Die 96stellige Anzeige des PC 1350:

1. Zeile, 1. Hälfte

286772	23572
286774	23572
286776	29575
286778	28577
286780	29581
286782	28583
286784	28695
286786	28687
286788	28689
286790	28691
286792	28693
286794	28695
286796	28697
286798	28699
286700	28699
29184	28701
29186	29185
29188	29187
29190	29191
29192	29193
29194	29195
29196	29197
29198	29197
29200	29199
29202	29201
29204	29202
29206	29205
29208	29207
29210	29209
29212	29211
29214	29213
29596	29597
29698	29699
29700	29701
29702	29701
29703	29704
29704	29705
29706	29707
29708	29708
29710	29710

1. Zeile, 2. Hälfte

29711	29712	29713	29714	29715	29716	29717	29718	29719	29720	29721	29722	29723	29724	29725	30208	30209	30210	30211	30212	30213	30214	30215	30216	30217	30218	30219	30220	30221	30222	30223	30224	30225	30226	30227	30228	30229	30230	30231	30232	30233	30234	30235	30236	30237	30720	30721	30722	30723	30724	30725	30726	30727	30728	30729	30730	30731	30732	30733	30734	30735	30736	30737	30738	30739	30740	30741	30742	30743	30744	30745	30746	30747	30748	30749
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

2. Zeile, 1. Hälfte

288736	28737-
288738	28739-
288740	28741-
288742	28743-
288744	28745-
288746	28747-
288748	28749-
288750	28751-
288752	28753-
288754	28755-
288756	28757-
288758	28759-
288760	28761-
288762	28763-
288764	28765-
292448	29249-
29250	29251-
29252	29253-
29254	29255-
29256	29257-
29258	29259-
29260	29261-
29262	29263-
29264	29265-
29266	29267-
29268	29275-
29270	29277-
29272	29271-
29274	29273-
29276	29275-
29278	29277-
29280	292768
29282	292761-
29284	292762
29286	292763-
29288	292764
29290	292765-
29292	292766
29294	292767-
29296	292768
29298	292769-
29300	292770
29302	292771-
29304	292772
29306	292773-

2. Zeile, 2. Hälfte

29775	29776	29777	29778	29779	29780	29781	29782	29783	29784	29785	29786	29787	29788	29789	30272	30273	30274	30275	30276	30277	30278	30279	30280	30281	30282	30283	30284	30285	30286	30287	30288	30289	30290	30291	30292	30293	30294	30295	30296	30297	30298	30299	30300	30301	30784	30785	30786	30787	30788	30789	30790	30791	30792	30793	30794	30795	30796	30797	30798	30799	30800	30801	30802	30803	30804	30805	30806	30807	30808	30809	30810	30811	30812	30813
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

3. Zeile, 1. Hälfte

29702	29703
29742	29741
29768	28767
28770	28769
28772	28771
28774	28773
28776	28775
28778	28777
28780	28779
28782	28781
28784	28783
28786	28785
28788	28787
28790	28789
28792	28791
28794	28793
28796	28795
28798	28797
28799	28798
30240	30239
30242	30241
30244	30243
30246	30245
30248	30247
30250	30249
30252	30251
30254	30253
30256	30255
30258	30257
30260	30259
30262	30261
30264	30263
30266	30265
30268	30267
30750	30751
30752	30753
30754	30755
30756	30757
30758	30758
30760	30759
30762	30761
30764	30763
30766	30765
30768	30767
30770	30769
30772	30771
30774	30773
30776	30775
30778	30777
30779	30779
29704	29743
29744	29745
29746	29747
29748	29749
29750	29750
29752	29751
29754	29753
29756	29755
29758	29758
29760	29711
29762	29712
29764	29713
29766	29714
29768	29715
29770	29716
29772	29717
29774	29718
29776	29719
29778	29720
29780	29721
29782	29722
29784	29723
29786	29724
29788	29725
29790	29726
29792	29727
29794	29728
29796	29729
29798	29730
29800	29731
29802	29732
29804	29733
29806	29734
29808	29735
29810	29809
29811	29811
29812	29813
29813	29814
29814	29815
29815	29816
29816	29817
29817	29818
29818	29819
29819	30302
30302	30303
30303	30304
30304	30305
30305	30306
30306	30307
30307	30308
30308	30309
30309	30310
30310	30311
30311	30312
30312	30313
30313	30314
30314	30315
30315	30316
30316	30317
30317	30318
30318	30319
30319	30320
30320	30321
30321	30322
30322	30323
30323	30324
30324	30325
30325	30326
30326	30327
30327	30328
30328	30329
30329	30330
30330	30331
30331	30814
30814	30815
30815	30816
30816	30817
30817	30818
30818	30819
30819	30820
30820	30821
30821	30822
30822	30823
30823	30824
30824	30825
30825	30826
30826	30827
30827	30828
30828	30829
30829	30830
30830	30831
30831	30832
30832	30833
30833	30834
30834	30835
30835	30836
30836	30837
30837	30838
30838	30839
30839	30840
30840	30841
30841	30842
30842	30843

C

3. Zeile, 2. Hälfte

29726	29725
29728	29727
29730	29729
29732	29731
29734	29733
29736	29735
29738	29737
29740	29739
29742	29741
29744	29743
29746	29745
29748	29747
29750	29749
29752	29751
29754	29753
29756	29755
29758	29756
29760	29759
29762	29758
29764	29757
29766	29756
29768	29755
29770	29754
29772	29753
29774	29752
29776	29751
29778	29750
29780	29749
29782	29748
29784	29747
29786	29746
29788	29745
29790	29744
29792	29743
29794	29742
29796	29741
29798	29740
29800	29739
29802	29738
29804	29737

C

4. Zeile, 1. Hälfte

29221	29220
29216	29217
29218	29219
29220	29221
29222	29221
29223	29222
29224	29223
29225	29226
29226	29227
29228	29229
29230	29231
29232	29233
29234	29235
29236	29235
29238	29237
29240	29239
29242	29241
29244	29243
29246	29245
29248	29247
29250	29249
29252	29251
29254	29253
29256	29255
29258	29257
29260	29259
29262	29261
29264	29263
29266	29265
29268	29267
29270	29266
29272	29271
29274	29273
29276	29275
29278	29277
29280	29279
29282	29281
29284	29283
29286	29285
29288	29287
29290	29289
29292	29291
29294	29293
29296	29295
29298	29297
29300	29299
29302	29301
29304	29303
29306	29305
29308	29307
29310	29309
29312	29311
29314	29313
29316	29315
29318	29317
29320	29319
29322	29321
29324	29323
29326	29325
29328	29327
29330	29329
29332	29331
29334	29333
29336	29335
29338	29337
29340	29339
29342	29341

C

4. Zeile, 2. Hälfte

29751	29750
29753	29752
29755	29754
29757	29756
29759	29758
29761	29760
29763	29762
29765	29764
29767	29766
29769	29768
29771	29770
29773	29772
29775	29774
29777	29776
29779	29778
29781	29780
29783	29782
29785	29784
29787	29786
29789	29788
29791	29790
29793	29792
29795	29794
29797	29796
29799	29798
29801	29800
29803	29802
29805	29804
29807	29806
29809	29808
29811	29810
29813	29812
29815	29814
29817	29816
29819	29818
29821	29820
29823	29822
29825	29824
29827	29826
29829	29828
29831	29830
29833	29832
29835	29834
29837	29836
29839	29838
29841	29840
29843	29842

Die beiden wichtigen CALL-Befehle zur Grafiksteuerung:
(ausser PC 1350)

Das Setzen der Bits des Displaybuffers allein genügt beim PC 14XX nicht, damit die selbstdefinierten Zeichen auf der Anzeige erscheinen. Dies verhält sich deshalb so, da während der Programmausführung die Anzeige nur dann eingeschaltet wird, wenn der BASIC-Interpreter gerade einen INPUT-, PRINT- oder PAUSE-Befehl ausführt. Bei den Rechnern PC 126X und PC 1350 dagegen bleibt das Display immer eingeschaltet.

Zusätzlich zum Setzen der gewünschten Bits des Displaybuffers müssen Sie also auch noch die Anzeige einschalten. Dies kann grundsätzlich auf folgende zwei Arten geschehen:

CALL-Adresse	140X	1421	126X	Funktion beim Aufruf durch CALL Adresse
1442 1448 -				Schaltet die Anzeige ein und fährt sofort mit dem Programm fort. Beim PC 126X bleibt die Anzeige stets eingeschaltet.
5208 5208 4414				Schaltet die Anzeige ein und unterbricht das Programm, bis eine Taste zur Fortführung des Programms gedrückt wird. Kann auch für Tastaturabfragen verwendet werden (siehe Kapitel 4.4)

Ein Grafikbeispiel:

Mit diesen ROM-Unterprogrammaufrufen der Tabelle oben ist es auch möglich, bewegte Grafiken zu erstellen, denn diese Unterprogramme schalten das Display ein und kehren ins BASIC zurück (bei CALL 5208 (PC 126X: CALL 4414) muß noch eine Taste betätigt werden). Alle folgenden Displaymanipulationen

werden direkt sichtbar. Damit kann man z.B. Werbeschriften, Trickfilmszenen usw. programmieren. Lassen Sie Ihrer Phantasie freien Lauf!

FUER PC 14XX:

```

10:WAIT 0:PRINT "":CALL
  5208:FOR I=1 TO 3:
    RESTORE
20:FOR J=24576 TO 24611
  STEP 5:GOSUB 200:
  NEXT J:FOR J=24640
  TO 24675 STEP 5:
  GOSUB 200:NEXT J
30:FOR J=24581 TO 24615
  :READ A:POKE J,A:
  NEXT J:FOR J=24645
  TO 24679:READ A:POKE
  J,A:NEXT J:NEXT I
40:CALL 5208:END
100:DATA 38,73,73,73,50,
  127,8,4,4,120,32,84,
  84,84,120,124,8,4,4,
  8,124,36,36,36,24
110:DATA 0,0,0,0,0,0,62,65
  ,65,65,34
120:DATA 8,4,4,8,124,24,
  84,84,84,56,0,68,127
  ,4,0,124,32,64,64,60
  ,24,36,36,36,124
130:DATA 120,4,8,4,120,5
  6,68,68,68,56
200:POKE J,12,18,36,18,1
  2:RETURN

```

FUER PC 126%:

```

10:WAIT 0: PRINT "":CALL
  4414: FOR I=1
  TO 3: RESTORE
20:FOR J=8212 TO 8251
  STEP 5: GOSUB 200:
  NEXT J: FOR J=10275
  TO 10240 STEP -5:
  GOSUB 200: NEXT J
30:FOR J=8217 TO 8251:
  READ A: POKE J,A:
  NEXT J: FOR J=10274
  TO 10240 STEP -1:
  READ A: POKE J,A:
  NEXT J: NEXT I
40:CALL 4414: END
100:DATA 38,73,73,73,50,
  127,8,4,4,120,32,84,
  84,84,120,124,8,4,4,
  8,124,36,36,36,24
110:DATA 0,0,0,0,0,0,62,65
  ,65,65,34
120:DATA 8,4,4,8,124,24,
  84,84,84,56,0,68,127
  ,4,0,124,32,64,64,60
  ,24,36,36,36,124
130:DATA 120,4,8,4,120,5
  6,68,68,68,56
200:POKE J,12,18,36,18,1
  2: RETURN

```

Zum PC 14XX: Erschrecken Sie nicht, wenn bei Ihren eigenen Grafikprogrammen zuweilen auf der linken Seite der Anzeige softwarebedingte Störungen auftreten: Der Displaypuffer wird

eben auch für interne Zwischenspeicherungen benutzt. Aus diesem Grund wird die Anzeige bei laufenden Berechnungen normalerweise auch ausgeschaltet. Sie können diese störenden Zeichen mit POKE Spaltenadresse,0 löschen.

Zum PC 1350: Hier wird es komfortabler und einfacher sein, wenn Sie zur Grafiksteuerung die bereits im BASIC implementierten Grafikbefehle (PSET, PRESET und LINE) benutzen.

Die hier vorgestellten Techniken haben wir auch bei der Entwicklung des Programms 'Turbo Graphics' (Kapitel 9.6) angewandt. Mit diesem fertigen Maschinenprogramm können Sie direkt auf das Display zugreifen und auch ganze Bildschirmschäfte abspeichern.

Für den PC 1350 sind in einem speziellen Programm auch noch weitere Funktionen wie Linien ziehen, Kreise zeichnen usw. vorhanden.



Teil 2: Einstieg in die Maschinensprache: Die Grenzen von BASIC sprengen

Kapitel 5

Maschinensprache: Ein erster Überblick

In diesem zweiten Teil kommen wir zum eigentlichen Kern der Sache. Die Programmiersprache BASIC und ihre Leistungsmarken haben wir in den letzten vier Kapiteln ausführlich besprochen. Um die weiteren Möglichkeiten des PC zu nutzen, brauchen Sie die Maschinensprache. Nur mit ihr können Sie direkt auf den Prozessor zugreifen und dessen Befehlsatz ausnutzen. Auf den folgenden Seiten finden Sie einen ersten Überblick über die Maschinensprache und alles, was damit zu tun hat.

Danach zeigen wir Ihnen auch, wie Sie auf dem PC Maschinenprogramme eingeben können und worauf Sie acht geben müssen. Doch zuerst ist es einmal nötig, die CPU, die Steuereinheit des Rechners, ein wenig genauer unter die Lupe zu nehmen.

5.1 Die CPU: Das Herz des Rechners

Jeder Computer besitzt ein 'Herz', die CPU (Central Processing Unit = Zentrale Steuereinheit). Sie führt fast alle Operationen im Computer durch und ist also so etwas wie eine Kommandozentrale.

Auf sich allein gestellt kann die CPU jedoch gar nichts. Um zu wissen, was sie tun soll, benötigt sie detaillierte Anweisungen von außen. Im PC sind schon einige tausend Anweisungen in Maschinensprache (im ROM) für die CPU enthalten, ohne die wir nicht einmal etwas eingeben, geschweige denn ein BASIC-Programm laufen lassen könnten.

Die CPU unseres PC ist die von SHARP entwickelte SC 61860. Von ihrer Struktur her ist sie eigentlich mit keinem der bekannten 8-Bit-Prozessoren wie dem Z80 oder 6502 vergleichbar.

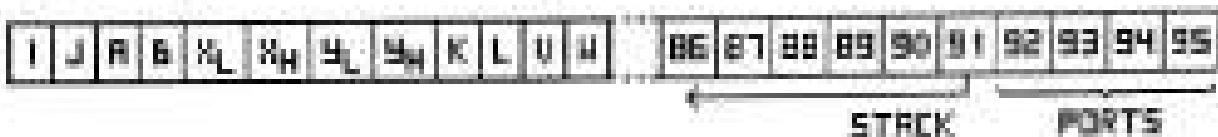
5.2 Externes und internes RAM

Das externe RAM kennen Sie ja bereits vom ersten Teil; alle BASIC-Beispiele betrafen nur diesen Bereich. Der Einfachheit halber haben wir immer nur von einem RAM gesprochen. Es war dabei auch klar, daß es meistens um den BASIC-Speicher, manchmal auch um den Standardvariablen Speicher ging. Auf jeden Fall um Speicherzellen, auf die wir mit PEEK und POKE direkten Zugriff hatten.

Nun gibt es aber noch ein zweites RAM: Das interne RAM, das RAM der CPU. Intern heißt es darum, weil es sich in der CPU selbst befindet, und nicht etwa in Bauteilen, die der CPU angeschlossen sind.

Das interne RAM ist auch wesentlich kleiner als das externe. Es umfaßt nur 96 Byte. Neu ist auch, daß gewisse Speicherzellen des internen RAM eine ganz bestimmte Funktion haben, ähnlich wie dies bei den Standardvariablen des externen RAM der Fall ist. Diese speziellen Speicherzellen nennt man Register. Die folgende Skizze gibt Auskunft über den Aufbau des internen RAM.

Abbildung: Intern benutzte Speicherzellen

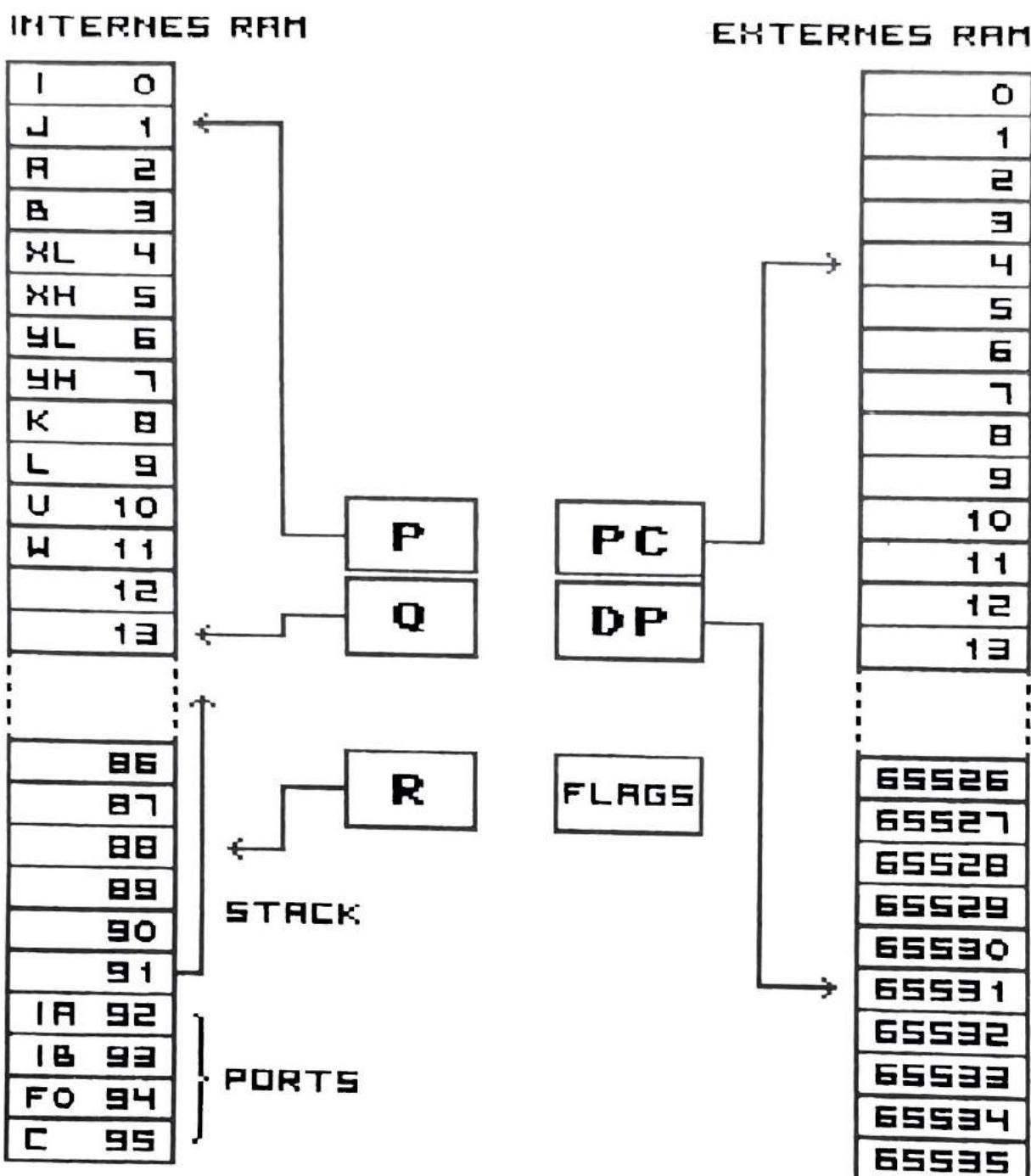


Register sind für uns sehr wichtig. Deshalb ist es gut, wenn wir uns nun die einzelnen Register und ihren Aufgabenbereich genauer anschauen.

5.3 Register

Wie jede CPU enthält auch unsere CPU SC 61860 eine Reihe von speziellen internen Speicherzellen und Zeigern. Wie Sie bereits wissen, werden diese Speicherzellen Register genannt. Die folgende Abbildung zeigt den Aufbau der CPU und die verschiedenen Zeiger und Register:

Abbildung: Die CPU



Unsere Register haben auch besondere Namen:

DP, PC, P, Q, R sind Zeiger

I, J, A, B, X, Y, K, L, V, W sind Speicherzellen

Daneben gibt es noch die Flags; das Carry- und das Zero-Flag. Flags bestehen aus einem einzigen Bit; sie können also nur 0 oder 1 speichern. Sie zeigen folglich immer einen bestimmten Zustand an. Wir werden später auf sie zurückkommen.

Im Folgenden werden wir uns genauer mit den Registern aus-einandersetzen. Über sie läuft nämlich die Verarbeitung aller hereinkommenden Daten. Die Register werden gebraucht, um diese Daten kurzzeitig zu speichern, um mit ihnen dann Opera-tionen durchzuführen.

Ein typisches Maschinenprogramm, das mit diesen Registern operiert, könnte zum Beispiel so aussehen:

- Lade den Datenzeiger DP mit 15000
 - Lade das A-Register mit dem Inhalt der Speicherzelle, auf die der Datenzeiger DP weist
 - Addiere 10 zu A
 - Speichere den Inhalt von A wieder in der Speicherzelle, auf die der Datenzeiger DP weist.
 - Gehe ins BASIC zurück

Entsprechendes Programm in BASIC:

```
DP = 15000  
A = PEEK(DP)  
A = A+10  
POKE DP,A  
RETURN
```

Wir werden Ihnen im Folgenden so oft wie möglich solche BASIC-Entsprechungen bereitstellen, damit Sie die Struktur eines Maschinenprogramms besser und schneller verstehen lernen.

Der Datenzeiger DP (Data Pointer)

Wenn Sie mit PEEK Daten auslesen, muß nach dem PEEK-Befehl die Adresse einer ganz bestimmten Speicherzelle stehen.

Immer wenn Sie auf Daten zugreifen wollen, muß die Adresse der betroffenen Speicherzelle vorgegeben sein.

Der Datenzeiger DP 'merkt' sich eine bestimmte Adresse des externen RAM, auf die nachher zugegriffen wird. Wenn Sie z.B. den Inhalt des Akkumulators (Register A) in einer bestimmten Speicherzelle des externen RAM ablegen wollen, müssen Sie vorher die Adresse dieser Speicherzelle im Datenzeiger DP speichern und dann einen speziellen Speicherbefehl benutzen.

Ein typisches Maschinenprogramm könnte z.B. so aussehen:

- Lade den Datenzeiger DP mit 25900 DP = 25900
- Lade A mit dem Inhalt der Speicher- A = PEEK DP
- zelle, auf die der Datenzeiger DP weist.
- Lade den Datenzeiger mit 25901 DP = 25901
- Speichere den Inhalt des Akkumulators POKE DP,A
- A in der Speicherzelle, auf die der
 Datenzeiger DP weist.

Der Datenzeiger kann neben dem externen RAM auch das externe BASIC-ROM (32 K) auslesen. Er hat jedoch keinen Zugriff auf das interne 8K-ROM, das sich in der CPU befindet.

Immer wenn wir den Ausdruck 'DP' (DP ohne Klammern) verwenden, meinen wir nur den Datenzeiger als gewöhnliches 16-Bit-Register, das direkt angesprochen wird.

Bei '(DP)' (DP in Klammern) dagegen ist der Inhalt der Speicherzelle gemeint, auf die der Datenzeiger weist, also PEEK DP.

Die Zeigerregister P und Q

Was der DP für das externe RAM ist, das sind die Zeiger P und Q für das interne RAM. Sie zeigen auf Speicherzellen des internen RAM, mit denen eine Operation durchgeführt werden soll.

Wir verwenden für unsere Maschinenprogramme fast ausschließlich den Zeiger P. Der Zeiger Q ist zwar genauso wichtig, arbeitet aber eher im Hintergrund. P und Q sind also genau aufeinander abgestimmt. Jeder dieser beiden Zeiger hat seine spezielle Aufgabe.

Immer wenn wir von 'P' oder 'Q' sprechen, sind diese Zeiger nur als gewöhnliche 7-Bit-Register gemeint. Dagegen ist bei '(P)' oder '(Q)' der Inhalt der internen Speicherzellen gemeint, auf die diese Register zeigen.

Steht zum Beispiel im Zeiger P eine 2 (2 ist die Adresse des A-Registers), so sind (P) und der Inhalt des A-Registers identisch. P zeigt dann also auf die Speicherzelle des internen RAM mit der Adresse 2. Wie aus der Abbildung der CPU in Kapitel 5.3 ersichtlich ist, befindet sich eben das A-Register in dieser Speicherzelle.

Dieser Zusammenhang ist äußerst wichtig. Wer die Funktionsweise der Zeigerregister begriffen hat, ist auf dem Weg zum Verständnis der CPU schon einen großen Schritt vorwärts gekommen. Ein mehr oder weniger historisches Beispiel mag diesen Zusammenhang noch verdeutlichen.

Eine Verdeutlichung:

DIE SCHATZKAMMER DES GROSSEN SULTANS

Der mächtige arabische Sultan Sindbad von Bagdad besaß eine große unterirdische Schatzkammer, ja, ein ganzes System von Schatzkammern. Nur wenige Zeitgenossen hatten sie je gesehen.

Nach dem streng bewachten Eingang führte eine steile Treppe hinab in einen langen Gang, an dessen rechter Wand lauter Türen zu Schatzkammern führten. Auf jeder Türe war ein altes Stück Pergament angebracht, auf der eine Nummer stand. Auf der ersten Türe '0', auf der zweiten Türe '1', auf der dritten '2'; so waren alle Kammern bis ans Ende des Ganges nummeriert. Auf der letzten Kammer stand '95'. In diesen 96 Kammern waren gewaltige Schätze untergebracht, allesamt in der Form von Säcken, gefüllt mit Gold und Diamanten. Insgesamt konnten in einer Kammer 255 volle Säcke untergebracht werden; dann war sie aber zum Bersten gefüllt. Man munkelte, einige der Kammern seien prall gefüllt, andere hingegen ganz leer.

Der Sultan selbst hatte sich dieses raffinierte System ausgedacht. Jede Kammer war mit einer starken Eichentür verriegelt. Potentielle Diebe konnten nicht wissen, welche Kammern nun gefüllt waren, und mehrere Kammern auf einmal konnten sie auch nicht aufbrechen.

Der große Sindbad hielt sich oft in diesen Kammern auf. Darum gab er seinen Lieblingskammern auch Namen. Die Bedeutung der meisten Namen blieb bis heute ein Geheimnis, da sie der Sultan nie aussprach. Er sprach immer nur von den Kammern I und J, den Kammern A und B und natürlich seinen Doppelkammern X und Y, die aus zwei Kammern bestanden und deshalb besonders geräumig waren.

Weil er so oft von der Kammer A sprach, vermutete man, daß sie als Zwischenlager für Goldeingänge und -ausgänge von und an fremde Königshäuser diente. Wahrscheinlich wurden auch die neuen Schätze aus seinen zahlreichen Eroberungskriegen zuerst hier, in der Kammer A, untergebracht, um später vom Schatzmeister überprüft und aufgeteilt zu werden.

Seinem Schatzmeister und seinen engsten Vertrauten verriet der Sultan, nach welchem System er seine Lieblingskammern benannt hatte. Bei I sei Kammer 0 gemeint, bei J die Kammer 1, bei A und B die Kammern 2 und 3, entsprechend für X die Kammern 4 und 5 und für Y die Kammern 6 und 7. Seinem Koch hinterließ er oft die Mitteilung, er befindet sich in Kam-

mer A. Wenn das Essen zubereitet war (und der Sultan Sindbad war ein großer Feinschmecker), mußte ihn der Schatzmeister jeweils in der Kammer 2 aufsuchen, um die Mitteilung des Kochs zu überbringen.

Als der König dann gestorben war, wurden seine Schätze in eine neue, zentrale Schatzkammer gebracht. Im Laufe der Zeit vergaßen die Nachkommen die raffinierte Technik des alten Sultans. Sie blieb der Weltöffentlichkeit verborgen, bis sie in jüngster Zeit in alten Handschriften aus Bagdad wieder entdeckt wurde und durch ihre Raffinesse bald Eingang in die modernste Mikroprozessortechnologie fand.

Soweit zur Schatzkammer dieses legendären Sultans.

Laden wir also einen bestimmten Wert ins P-Register, so wählen wir eine 'Tür' an, ein bestimmtes Register. Dieses Register ist eine Speicherzelle des internen RAM, welche die Adresse P hat. Man sagt auch: P zeigt auf dieses Register. Unter (P) ist dann auch nie die Adresse der Speicherzelle gemeint, sondern immer ihr Inhalt. (P) entspricht also der Anzahl Goldsäcke, die sich in der Kammer mit der Nummer P befinden.

Zählerregister I und J

Unsere CPU besitzt einige Befehle, mit denen man ganze Speicherabschnitte verändern kann. Die Größe eines solchen Speicherabschnitts, den man auch 'Block' nennt, wird durch die Register I oder J festgelegt. Bei Blockoperationen werden immer I+1 bzw. J+1 Speicherzellen verändert.

Beispiel:

- Lade A mit 42 (CHR\$ (42)="*") A=42
- Lade I mit 6 I = 6
- Lade den Datenzeiger mit 17873 DP = 17873
- Fülle im externen RAM ab Adresse 17873 FOR N = 0 TO I
I+1 Bytes mit dem Inhalt von A. POKE DP+N, A
NEXT N

Dieses Programm füllt beim PC 140X die Standardvariable Z\$ mit "*****".

Wir werden für unsere Maschinenprogramme ausschließlich das I-Register verwenden, da eine falsche Verwendung des J-Registers zu Störungen oder gar zu einem ALL RESET führen kann.

Register A (Akkumulator)

Sehr viele Operationen der CPU sind auf den Akkumulator oder kurz Akku abgestimmt. Wie die meisten anderen Register der CPU faßt auch er 1 Byte. Der Akku ist ein sehr wichtiges Register, eine Art 'Hauptspeicher', der sehr oft für den Datenaustausch zwischen Speicherzellen zuständig ist.

Beispiel:

- Lade den Datenzeiger mit 15000 DP = 15000
- Lade den Akku mit dem Inhalt der Speicherzelle, auf die der Datenzeiger DP weist A = PEEK DP
- Addiere 5 zum Inhalt des Akku A = A+5
- Speichere den Inhalt des Akku in der Speicherzelle, auf die der Datenzeiger weist POKE DP,A

Das B-Register

Wie der Akku faßt auch das B-Register 1 Byte. Zusammen mit dem Akku kann man in diesen beiden Registern einen 16-Bit-Wert speichern und damit z.B. einen Adreßwert von 0 bis 65535 bearbeiten. In B steht dann das Highbyte, in A das Lowbyte. Diese Möglichkeit wird bei der 16-Bit-Addition und -Subtraktion und beim Auslesen des internen ROM eine Rolle spielen.

Natürlich kann das B-Register auch als normales 8-Bit-Register verwendet werden, was unter anderem zur Zwischenspeicherung eines Akkumulatorinhalts dienen kann.

Beispiel:

- Lade das B-Register mit 1 $B = 1$
 - Lade den Akkumulator mit 10 $A = 10$
 - Lade P mit 8 (P zeigt nun auf das K-Register) $P = 8$
 - Lies die Speicherzelle mit der Adresse, die B als High- und A als Lowbyte hat, und schreibe das Ergebnis ins K-Register $K = \text{PEEK}(256 * B + A)$

Das X- und Y-Register

Diese beiden Register fassen je 16 Bits (2 Bytes) und können daher einen Adreßwert von 0 bis 65535 aufnehmen.

Das ist auch ihre wichtigste Funktion: Bei der Bearbeitung eines Speicherteils steht im X- oder Y-Register die Adresse der Speicherzelle, mit der gerade eine Operation durchgefrt wird.

Dabei können in Verbindung mit dem Akkumulator ganze Speicherteile bearbeitet werden, wobei das Lesen spezielle Aufgabe des X-Registers, das Schreiben spezielle Aufgabe des Y-Registers ist.

Typische Befehlsfolge:

- Erniedrige X um 1 $X = X - 1$
 - Lade den Datenzeiger mit dem Inhalt des X-Registers $DP = X$
 - Lade A mit dem Inhalt der Speicherzelle, auf die der Datenzeiger weist $A = PEEK(DP)$
 - Addiere 10 zu A $A = A + 10$
 - Erniedrige Y um 1 $Y = Y - 1$
 - Lade den Datenzeiger mit dem Inhalt des Y-Registers $DP = Y$
 - Speichere nun den Inhalt des Akkus in der Speicherzelle, auf welche der Datenzeiger weist $POKE DP, A$

Natürlich kann man auch diese beiden Register anders verwenden, indem man ihre einzelnen Bytes separat benutzt.

Das Register X setzt sich dann aus den Registern XH (X-Highbyte) und XL (X-Lowbyte) zusammen, das Register Y aus YH (Y-Highbyte) und YL (Y-Lowbyte).

Die Zählregister K und L, V und W

Diese vier Register fassen je 8 Bits. Oft werden sie einfach zur Speicherung von Ergebnissen benutzt. Ihre eigentliche Funktion ist aber eine andere. Diese Zählregister eignen sich, wie schon der Name sagt, hervorragend zum Zählen, wie oft ein einzelner Programmabschnitt (Schleife) durchlaufen wurde.

Beispiel:

- Lade den Zeiger P mit 9. P zeigt nun auf das L-Register $P = 9$
- Lade 20 ins L-Register $L = 20$
- Subtrahiere 1 von L $L = L - 1$
- Vergleiche den Inhalt von L mit 0 $L = 0 ?$
- Ist L ungleich 0, springe wieder zurück zu Punkt 3 IF $L \neq 0$ GOTO 3

Diese einfache Schleife subtrahiert solange 1 von L, bis L den Wert 0 hat. Insgesamt wird diese Schleife also 20 mal durchlaufen.

Programmzähler PC

Die CPU verwendet den Programmzähler PC, um sich während der Ausführung von Befehlen zu orientieren. Der PC merkt sich, wo es im Maschinenprogramm weitergeht.

Die gleiche Arbeitsweise haben Sie, wenn Sie z.B. beim Ausprobieren eines Kochrezepts Schritt für Schritt im Kochbuch weiterlesen und immer mit dem Zeigefinger auf dem Punkt bleiben, den Sie gerade bearbeiten.

Zunächst lesen Sie diesen Punkt. Wenn Sie mit dem Lesen fertig sind (z.B. 'Drei Pfund Mehl dazugeben und gut rühren'), führen Sie die Anweisung aus. Dann geht der Zeigefinger weiter zur nächsten Anweisung.

Genauso arbeitet die CPU: Der Programmzähler ist also der Zeigefinger der CPU!

Im Unterschied zu den Kochrezepten, die im Kochbuch verzeichnet sind, stehen die Befehle eines Maschinenprogrammes im Speicher (RAM) des Computers. Darum enthält der Pro-

grammzähler immer die Adresse der Speicherzelle, in welcher der Befehl steht, der gerade bearbeitet wird.

Dieser Befehl wird gelesen und ausgeführt. Dann zählt der Programmzähler weiter bis zum nächsten Befehl. Für jeden Befehl wiederholt die CPU diesen Zyklus.

5.4 Der CPU-Stack

Der CPU-Stack oder kurz Stack (Stapel) ist ein spezieller Bereich des internen RAM, der bei der Adresse 91 beginnt und von dort nach unten verwaltet wird. Die theoretische Größe des CPU-Stacks beträgt also 92 Bytes, doch davon werden normalerweise nur wenige Bytes benutzt.

Die CPU verwendet den Stack, um sich bei Unterprogrammaufrufen zu merken, bei welcher Adresse es bei der Rückkehr aus einem Unterprogramm weitergeht. Jedesmal, wenn ein Unterprogramm aufgerufen wird, wird die entsprechende Rücksprungadresse auf dem Stack abgelegt.

Der CPU-Stack arbeitet also wie ein etwas wirrer Personalchef, der seine täglichen Besprechungen folgendermaßen organisiert hat:

Für jedes Gespräch verwendet er ein Blatt Papier, um sich Notizen zu machen. Unterbricht ihn jemand in einem Gespräch, legt er das gerade bearbeitete Blatt auf einen Stapel, schickt seinen Gesprächspartner in den Warterraum und beginnt das neue Gespräch, für das er auch ein neues Blatt benutzt. Falls ihn in diesem Gespräch wieder jemand unterbricht, legt er auch dieses Blatt auf den Stapel und schickt seinen Gesprächspartner in den Warterraum.

Hat er dagegen ein Gespräch ganz bearbeitet, begleitet er den Gesprächspartner zum Ausgang, nimmt dann das oberste Blatt vom Stapel, ruft die entsprechende Person wieder zu sich herein und fährt mit seinem Gespräch fort. Diese Person ist natürlich

immer diejenige, die er als letzte herausgeschickt hat. So fährt er fort, bis er alle Blätter fertig bearbeitet hat und nach Hause gehen kann.

Mit diesem Verfahren kann man Unterbrechungen beliebig ineinander verschachteln. Man findet so auch stets an den Ausgangspunkt zurück und bearbeitet alle Vorgänge zu Ende.

Der Stack ist als "last-in-first-out" Speicher organisiert. Der zuletzt auf dem Stack abgelegte Wert wird als erster wieder zurückgeholt. In einem Unterprogramm kann also ein weiteres Unterprogramm aufgerufen werden, in diesem wieder ein neues usw. Nachdem alle Unterprogramme abgearbeitet sind, befindet sich die CPU wieder an der richtigen Stelle im Hauptprogramm.

Der Stack wird jedoch nicht nur zur Speicherung von Rücksprungadressen verwendet. Es gibt spezielle Befehle, mit denen man auch den Inhalt des Akkumulators auf dem Stapel ablegen und bei Bedarf wieder zurückholen kann.

Die Art der Speicherung von Daten auf dem Stack erfolgt von 'oben' nach 'unten', also von den höheren Adressen zu den niedrigeren Adressen hin. Konkret heißt das, dass der zuletzt auf dem Stack abgelegte Wert in den niedrigeren Adressen gespeichert ist, also weiter unten als die früher gespeicherten Werte, die sich folglich weiter oben, also in den höheren Adressen befinden.

Der Stackzeiger R

Die CPU merkt sich mit Hilfe des Stackzeigers R, welche Speicherzelle des Stacks zuletzt bearbeitet wurde. Der Stackzeiger enthält also immer die Adresse der letzten belegten Stackposition.

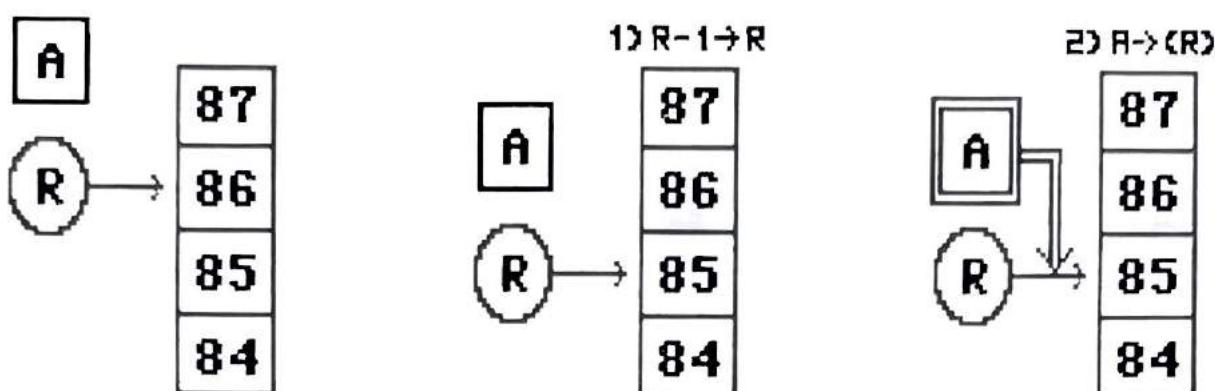
Beim Ablegen eines Wertes auf den Stack wird der Stackzeiger auch dementsprechend vermindert. Beim Zurückholen dieses Wertes wird R dann wieder auf den alten Stand gebracht, also erhöht.

Die folgende Skizze mag diesen Zusammenhang verdeutlichen:

Speicherung des Akkumulatorinhalts:

INTERNES RAM: ADRESSEN 84-87

PUSH



Ausgangsposition:
Der Stackzeiger
R weist auf die
zuletzt belegte
Speicherzelle.

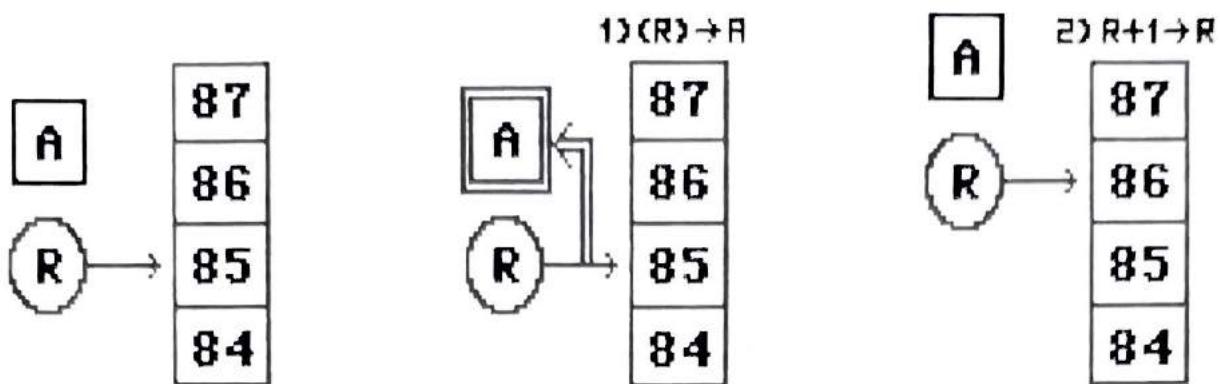
Der Stackzeiger
wird vermindert.
Er zeigt auf die
nächste freie
Speicherzelle.

Der Inhalt des
Akkumulators
wird in der
Speicherzelle
abgelegt, auf
die der Stack
zeiger R weist.

Zurückholen des gespeicherten Akkumulatorinhalts

INTERNES RAM: ADRESSEN 84-87

POP



Ausgangsposition:
Der Stackzeiger R
weist auf die zu-
letzt belegte
Speicherzelle.

Die zuletzt beleg-
te Speicherzelle
wird zurück in den
Akkumulator ge-
laden.

Der Stackzeiger
R wird erhöht.
Er weist nun
auf die nächste
belegte Spei-
cherzelle.

Analog dazu, jedoch mit zwei Bytes (High/Lowbyte) wird eine Rücksprungadresse gespeichert und wieder zurückgeholt.

5.5 Flags

Flag bedeutet Flagge, Fahne. Dies lässt erkennen, daß es sich bei Flags um eine Ja-/Nein-Entscheidung handelt. Eine Fahne ist entweder gehisst oder nicht.

Die Maschinensprache unseres 8-Bit-Prozessors kann nur Zahlen zwischen 0 und 255 bearbeiten. Alle arithmetischen Operationen

mit 8-Bit-Registern bewegen sich in diesem Zahlenbereich. Was geschieht aber, wenn Sie 10 zu 255 addieren wollen? Stürzt dann der Rechner ab, weil sich die Zahl 265 einfach nicht in 8 Bits quetschen läßt?

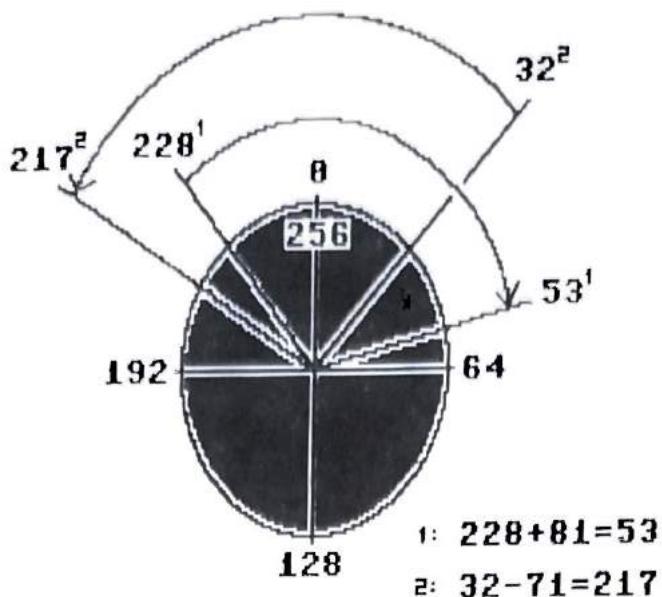
Oder ergreift die CPU wohl besondere Gegenmaßnahmen wie BEEPen und Blinken, um auf diese Weise auf den inneren Notstand aufmerksam zu machen?

Natürlich nicht. In Tat und Wahrheit besitzt sie ein einfaches, aber äußerst effizientes System, um solche "kritischen" Additionen und Subtraktionen zu verarbeiten. Jedesmal, wenn die 255er- bzw. 0er-Grenze überschritten wird, fängt sie wieder bei 0 bzw. 255 an.

Jede Uhr mit Zeigern macht genau dasselbe. Die Zeiger bewegen sich immer in einer 12 Stunden-Periode. Drehen Sie bei der Zeit 11'55 Uhr den Minutenzeiger um 10 Minuten vor, zeigt die Uhr die Zeit 00'05 Uhr an. Sie wissen natürlich, dass es im Wirklichkeit 12'05 Uhr ist.

Ähnlich werden Überträge bei der CPU gehandhabt. Auch die CPU muß wissen, ob eine Periode über- bzw. unterschritten worden ist. Dazu werden die Flags entsprechend gesetzt. Beträgt ein Ergebnis genau 0, wird das Z-Flag gesetzt. Bei einer Addition mit einem Ergebnis größer als 255 wird das C-Flag gesetzt, ebenso bei einer Subtraktion mit einem Ergebnis kleiner 0.

Skizze: CPU-Register innerhalb des 255er-Kreislaufs.



Ein paar numerische Beispiele:

Operation	Status der Flags:	C	Z
$200 + 56 = 0$		*	*
$10 - 10 = 0$		-	*
$0 + 0 = 0$		-	*
$200 + 57 = 1$		*	-
$240 + 16 = 0$		*	*
$20 - 21 = 255$		*	-
$255 - 254 = 1$		-	-

*: gesetzt (1)

-: nicht gesetzt (0)

Wie man aus den obigen Beispielen ersehen kann, besitzt die CPU der SHARP PCs zwei verschiedene Flags: Das Zero-Flag (Z-Flag) und das Carry-Flag (C-Flag).

Das Zero-Flag wird gesetzt, wenn eine Operation ein Ergebnis von 0 hat; nicht gesetzt ist es dann, wenn ein Resultat ungleich 0 erscheint. Das Carry-Flag wird dann gesetzt, wenn ein Übertrag stattfindet (Carry=tragen); wenn z.B. die 255er-Grenze überschritten wird.

Neben den arithmetischen Operationen gibt es noch weitere Befehle, die mit den Flags arbeiten: Die Vergleichsbefehle.

Bei einer Vergleichsoperation führt die CPU einfach eine Subtraktion der beiden Vergleichswerte durch, die jedoch nicht gespeichert wird: Das Ergebnis ist 0 bzw. das Zero-Flag ist gesetzt, wenn die beiden verglichenen Register den gleichen Wert haben.

Aus dem Status der Flags kann auch geschlossen werden, ob der zweite Wert größer oder kleiner als der erste war. Dafür ist dann das Carry-Flag verantwortlich.

Direkt kann man die Flags allerdings nicht abfragen. Es gibt jedoch sehr viele bedingte Sprungbefehle, die man benutzt, um ihren Zustand auszuwerten.

Eine typische Befehlsfolge in Maschinensprache sieht darum so aus:

- Vergleiche XL mit A A = XL ?
 - Wenn das Zero-Flag gesetzt ist
(d.h., wenn A=XL), springe zu
Adresse .. IF Z = 1
GOTO ..

Flags bleiben solange gesetzt, bis sie durch weitere Vergleichsbefehle oder arithmetische Befehle wieder neu gesetzt oder gelöscht werden. Interne Berechnungen wie z.B. die Veränderung des Stackzeigers R oder des Zeigers P, verändern den Status der Flags nicht!

Jede CPU hat ihre Flags. Leider gelten bei jedem Mikroprozessor wieder etwas andere Regeln, wie diese Flags eingesetzt werden. Andere Prozessoren haben noch weitere Flags, z.B. für Vorzeichen. Die CPU SC 61860 ist in dieser Beziehung recht einfach. Nur wenige Befehle beeinflussen den Zustand der beiden Flags. Deshalb fällt es auch nicht schwer, den Überblick zu behalten.

5.6 Maschinensprache und BASIC

BASIC-Programme werden vom Interpreter, einer Befehlstabelle im ROM, als einzelne Maschinenprogramme ausgeführt. Diese Maschinenprogramme sind relativ umfangreich. Dafür sind dann die BASIC-Befehle umso vielseitiger und komfortabler. Die zur Verfügung stehenden Variablen machen BASIC noch einfacher und bedienerfreundlicher.

 Die Befehle der Maschinensprache sind viel einfacher als BASIC-Befehle. Dafür werden sie auch wesentlich schneller ausgeführt. Das BASIC des PC's schafft in der Sekunde höchstens 100 Befehle, die CPU des PC's hingegen führt in dieser Zeit bis zu 96'000 Maschinenbefehle aus!

Diese Einfachheit bedingt natürlich auch, daß man mehr Befehle braucht, um mit Maschinenbefehlen die gleiche Wirkung zu erzielen, wie mit einem einzigen BASIC-Befehl.

 Die meisten Befehle der Maschinensprache beziehen sich direkt auf die CPU oder auf Speicherzellen. Will man z.B. den Inhalt zweier Speicherzellen des externen RAM vergleichen, so muß man zunächst den Inhalt der ersten Speicherzelle in den Akkumulator laden, danach den Inhalt der zweiten Speicherzelle ins interne RAM laden. Erst dann kann die Vergleichsoperation durchgeführt werden.

5.7 Mnemonics

 In der Maschinensprache hat jeder Befehl einen Code, der für die CPU bestimmt ist. Für den Benutzer bestimmt ist dagegen der Name eines Befehls. Ein solcher Name heißt Mnemonic. Mnemonics sind englische Abkürzungen für die Wirkung eines Befehls. Gegenüber Codes haben sie einen großen Vorteil: Diese Abkürzungen kann man sich viel leichter merken als die abstrakten Codes. Man kann deshalb ein Flußdiagramm, das in Mnemonics erstellt ist, leicht durchgehen und den Programmablauf noch einmal überprüfen, ohne jeden Befehl als Code in einer Tabelle suchen zu müssen.

Beispiel:

'Speichere den Inhalt des Akkumulators in der Speicherzelle, auf die der DP zeigt'

Befehl	Code	Mnemonic	Bedeutung
A -> (DP)	82	STD	Store A To Data

Eine Liste mit der Bedeutung aller Mnemonics der CPU und einige Tips, wie man sich diese Befehle am einfachsten merkt, finden Sie in Anhang B.

Die Befehle eines Maschinenprogramms werden jedoch als Zahlenkode gespeichert; im Speicher steht nicht etwa der Name eines Befehls, sondern der jeweilige Code, der sogenannte Operation-Code oder Op-Code.

Beispiel:

Mnemonic	Op-Code
LIDP 25900	16,44,101
INCA	66
STD	82
LIB 20	3,20

Der Code für einen Befehl nimmt jeweils ein Byte in Anspruch. Die zugehörigen Zahlenwerte können noch zusätzliche Bytes beanspruchen.

5.8 Wie werden Maschinenprogramme eingegeben?

Alle eigenen Maschinenprogramme sind also sinnvoll angeordnete Befehlscode, die von der CPU direkt verstanden werden.

Eigene Maschinenprogramme sind natürlich im externen RAM abgelegt. Meistens legt man sie im BASIC-Speicher ab. Man könnte dazu auch den Standardvariablen Speicher verwenden. Doch das ist nicht so günstig, da somit ein Maschinenprogramm mit der Benutzung der Standardvariablen wieder überschrieben wird. Außerdem entfällt diese Möglichkeit bei der Benutzung des ASSEMBLERS, da dieser sämtliche Standardvariablen benutzt.

 Die Codes eines Maschinenprogramms müssen also zuerst in die betreffenden Speicherzellen des externen RAM geschrieben werden. Um den POKE-Befehl kommt man nicht herum. PEEK, POKE und CALL bilden die absolute Grundlage für die Entwicklung eigener Maschinenprogramme auf jedem PC. Das ist auch der Grund, weshalb Rechner wie z.B. der PC-1430 nicht für Maschinensprache eingesetzt werden können: Diese Rechner besitzen diese wichtigen Befehle nicht.

Es gibt mehrere Möglichkeiten, Codes über den POKE-Befehl einzugeben. Diese Möglichkeiten wollen wir Ihnen nun anhand eines einfachen Programmbeispiels zeigen:

'Lade 10 in den Akkumulator und kehre ins BASIC zurück'

Die Codes für dieses Maschinenprogramm lauten: 2,10,55

Einfaches POKEn

 Codes werden in POKE-Zeilen aufgeteilt, wobei eine Zeile natürlich maximal 80 Zeichen (zirka 20 Zahlen zwischen 0 und 255) umfaßt. Unser Programmbeispiel würde so eingegeben:

(10:) POKE Anfangsadresse,2,10,55

Das ist die einfachste Art der Eingabe und bei kleinen Programmen durchaus angebracht. Sobald aber Maschinenprogramme länger als 20 Bytes sind, wird diese Methode sehr umständlich, weil eine POKE-Zeile einfach zu kurz ist. Eine zweite POKE-Zeile muß eingeschaltet werden. Vor allem die Fehlersuche und

auch die Berechnung der Anfangsadresse für die jeweils neue POK E-Zeile sind zeitraubend und machen das Programm unübersichtlich. Die zweite Methode hilft hier weiter.

DATA-Zeilen

Die BASIC-Befehle DATA und READ ermöglichen das Anlegen von Codetabellen. Wir können unsere Codes also in BASIC-Programmzeilen, in DATA-Zeilen, ablegen:

10: DATA 2,10,55

Die Eingabeschleife sieht dann wie folgt aus:

20: RESTORE:FOR I = Anfangsadresse TO Anfangsadresse +
Programmlänge -1:READ N:POKE I,N:NEXT I

Diese Form der Eingabe hat den großen Vorteil, daß bei größeren Programmen nicht für jede DATA-Zeile erneut eine Anfangsadresse berechnet werden muß. Bei Änderungen (z.B. Einfügen oder Löschen) kann einfach eine DATA-Zeile eingefügt oder gelöscht werden. Das POKEn, also die eigentliche Eingabe, wird über eine simple, übersichtliche FOR-NEXT-Schleife abgewickelt.

Der ASSEMBLER

Wie wir schon gesehen haben, kann man sich Mnemonics viel leichter merken als die relativ abstrakten Befehlscodes. In Mnemonics ausgedrückt sieht unser Beispiel so aus:

LIA 10	Lade 10 in den Akkumulator
RTN	Kehre ins BASIC zurück

Mit unserem Programm-ASSEMBLER können Sie jedes Maschinenprogramm direkt in Mnemonics eingeben. Der

ASSEMBLER nimmt Ihnen dabei einen Haufen Arbeit ab: Die Übersetzung der Mnemonics in die entsprechenden Codes und die Speicherung des lauffähigen Programms im externen RAM. Auch um die jeweilige Adresse brauchen Sie sich nicht mehr zu kümmern. Der ASSEMBLER rechnet sie selbst aus.

Der ASSEMBLER arbeitet nach folgendem Grundprinzip: In einer Befehlstabelle wird nach dem entsprechenden Mnemonic gesucht. In dieser Tabelle stehen auch alle Informationen über Code und Befehlslänge. Der ASSEMBLER legt die ausgelesenen Codes zusammen mit den zugehörigen Zahlenwerten an der richtigen Adresse ab. Dann addiert er die Befehlslänge zur Anfangsadresse und hat damit die Adresse, an der die nächste Eingabe abgelegt werden soll. Nachdem er das alles erledigt hat, zeigt er diese neue Adresse an und wartet auf weitere Eingaben.

Da die Suchroutine in Maschinensprache geschrieben ist, dauert die ganze Umwandlung und Speicherung nur etwa 2 Sekunden. Mit dem ASSEMBLER können auch bereits bestehende Maschinenprogramme gelesen und auf das Display oder den Drucker ausgegeben werden.

Die Beispielprogramme zu den Maschinenbefehlen im nächsten Kapitel sind sowohl in Mnemonics als auch in Codes angegeben. Sie können Maschinenprogramme also sowohl mit dem ASSEMBLER, als auch mit dem POKE-Befehl eingeben.

Will man wirklich intensiv mit der Maschinensprache arbeiten, so ist die Installation des ASSEMBLERS eine unbedingte Voraussetzung. Durch das einfache Eingeben und Editieren sparen Sie bei der Entwicklung von Programmen nicht nur viele Stunden kostbare Zeit, sondern das Programmieren macht auch mehr Spaß, da harte Knochenarbeit entfällt. Flußdiagramme und Entwürfe schreiben Sie direkt in Mnemonics. Die mühsame Zeit der Code-Programmierung mit POKE ist damit ein für allemal vorbei.

5.9 Der ASSEMBLER - Installation und Bedienung

Der ASSEMBLER entstand aus dem Wunsch, möglichst schnell und einfach Maschinensprache auf dem PC zu programmieren. Dabei sollte die mühsame Umwandlung Mnemonics - Codes und umgekehrt umgangen werden. Für die SHARP PCs kam dabei nur eine Lösung in Frage: ein komfortabler Zeilenssembler. Vor gut einem Jahr machten wir uns also an die Arbeit. Ein Teilprogramm in Maschinensprache sollte die Suche nach Codes und Befehlen in der Befehlstabelle übernehmen, ein BASIC-Teil die anspruchsvolleren Funktionen.

Ein reiner Maschinensprache-ASSEMBLER, ohne BASIC-Bestandteile, wäre zwar schneller gewesen und hätte etwas weniger Speicherplatz gebraucht. Von Anfang an kam aber so etwas nicht in Frage: Der Benutzer hätte allein für die Eingabe eines so riesigen Maschinenprogramms Stunden benötigt; und ein einziger Tippfehler hätte die ganze Arbeit wieder zunichte gemacht. Außerdem wollten wir einen ASSEMBLER, der auf allen PCs lauffähig sein sollte. Während einem halben Jahr verbesserten wir die ursprüngliche Version immer wieder. Schließlich war er fertig. Wir hatten nun unseren leistungsstarken ASSEMBLER für alle SHARP PCs.

Der ASSEMBLER bietet Ihnen folgende Möglichkeiten:

- *Assemblieren*

Sie bestimmen eine Anfangsadresse und geben dann Ihr Maschinenprogramm in reinen Mnemonics ein. Das letzte Mnemonic - Sie sind fertig! Das Maschinenprogramm ist nun im externen RAM des PC gespeichert und kann jederzeit aufgerufen werden.

- *Disassemblieren*

Angenommen, das Programm läuft nicht oder Sie wollen sich das Programm einfach anschauen. Wiederum geben Sie

die Anfangsadresse ein. Zeile für Zeile können Sie nun Ihr Programm listen. Finden Sie einen Fehler, überschreiben Sie die falsche Zeile und drücken ENTER. Damit ist der Fehler behoben.

- *Erstellen von Listings*

Oder Sie wollen sich ein Listing von einem Maschinenprogramm anfertigen. Sie betätigen die Tastenkombination für Drucken und geben die Endadresse ein, also die Adresse des letzten Befehls, der noch zum Programm gehört. Nach dem Drücken der ENTER-Taste wird das Listing vollautomatisch von Ihrem Drucker erstellt. Für größere Listings spart man sich damit viel Zeit und Mühe.

Noch etwas: Falls Sie ein Musterprogramm für strukturiertes BASIC suchen, nehmen Sie bitte nicht den BASIC-Teil des ASSEMBLERs. Dieser ist nämlich, milde ausgedrückt, sehr unübersichtlich. Wir haben bei der Entwicklung und Verbesserung des ASSEMBLERs nur auf möglichst geringen Speicherbedarf geachtet und sämtliche Regeln für strukturierte Programmierung außer Acht gelassen. In der Tat kämpften wir um jedes gesparte Byte, um Ihnen noch möglichst viel freien Speicher für eigene Maschinenprogramme zu überlassen.

Die Installation des ASSEMBLERs

Um es gleich vorweg zu nehmen: Ohne Cassettenrecorder mit funktionsfähigem Interface können Sie nicht sinnvoll mit dem ASSEMBLER arbeiten. Beim ersten Absturz geht das ganze Programm unweigerlich verloren. Außerdem ist es oft von Vorteil, vor dem Ausprobieren eines Maschinenprogramms das Programm auf Cassette zu speichern, um im Falle eines Absturzes nicht mehr das ganze Programm manuell eingeben zu müssen.

Der ASSEMBLER besteht aus zwei Teilen: einem Maschinenteil für die Suchroutine und einem BASIC-Teil für andere Aufgaben. Nachdem wir den Maschinenteil eingegeben haben, versetzen wir die Anfangsadresse des BASIC-Speichers nach oben, so daß das Maschinenprogramm dadurch 'unsichtbar' und somit auch geschützt ist. Danach geben Sie den BASIC-Teil ein. Und denken Sie daran: Speichern Sie den gesamten ASSEMBLER, bevor Sie auch nur irgend etwas ausprobieren.

Diese oben beschriebene Konstellation von Maschinen- und BASIC-Teil (Maschinenteil vor dem BASIC-Teil), die wir verwenden, hat einen gewaltigen Vorteil: Der BASIC-Speicher ist gegen oben, also gegen das Ende des BASIC-RAM, völlig frei. Sie können weitere BASIC-Programme eingeben oder auch Teile des ASSEMBLERS löschen, z.B. die Druckroutine. Dabei sind Sie nur durch den noch verfügbaren Speicher eingeschränkt. Die großen Speicherkapazitäten des PC 1402, PC 1261 und PC 1350 können damit optimal ausgenutzt werden. Neben dem ASSEMBLER können Sie bei diesen Rechnern noch viele andere BASIC-Programme im Speicher haben, ohne daß hier die geringsten Konflikte auftreten würden.

Geben Sie zuerst den Maschinenteil für Ihren Rechner mit POKE ein; die entsprechenden Anfangsadressen sind angegeben. Benutzen Sie auf jeden Fall diese angegebenen Adressen und achten Sie darauf, daß Ihnen keine Tippfehler unterlaufen!

Alte Modelle des PC 1401 oder PC 1260

Für einige ältere Versionen des PC 1401 (Kaufdatum vor mehr als 1.5 Jahren) ist der ASSEMBLER nicht lauffähig, da die BASIC-Adresse nicht nach oben verschoben werden kann. Erkennungszeichen für die ältere PC 1401-Version: Statt FACT für die Fakultätsfunktion (!) haben diese Rechner FAC. Außerdem haben sie auch eine andere Adresse für die BEEP-Routine, so daß CALL 49321 keinen BEEP bewirkt.

Auch beim PC 1260 gab es eine Urversion, bei der der zweite Teil des ROM nicht mit dem aktuellen ROM übereinstimmt. ROM-Unterprogramme stimmen in diesem Falle nicht mit den angegebenen Adressen überein!

Maschinenteil PC 140X

8192	132	2	176	219	8280	128	134	219	2
8196	2	32	80	219	8284	70	80	219	36
8200	2	2	148	219	8288	38	103	62	43
8204	120	32	137	103	8292	5	5	7	2
8208	58	43	6	148	8296	0	38	2	95
8212	99	1	126	32	8300	134	219	2	70
8216	71	120	32	137	8304	80	219	2	245
8220	103	50	41	19	8308	38	36	38	2
8224	148	99	2	126	8312	0	38	2	136
8228	32	71	120	32	8316	134	219	36	38
8232	137	103	65	41	8320	103	58	59	5
8236	32	148	99	3	8324	7	2	0	38
8240	126	32	71	120	8328	55	36	132	99
8244	32	137	103	0	8332	30	40	6	133
8248	41	45	148	99	8336	99	61	56	2
8252	4	126	32	71	8340	55	2	86	50
8256	120	32	137	103	8344	121	192	169	2
8260	0	41	58	2	8348	80	219	2	108
8264	58	103	58	40	8352	3	66	0	2
8268	30	37	103	58	8356	148	53	16	32
8272	59	4	37	103	8360	155	148	25	55
8276	62	43	4	2					

Maschinenteil PC 1421

14336	132	2	176	219	14388	32	137	103	0
14340	2	32	80	219	14392	41	45	148	99
14344	2	2	148	219	14396	4	126	32	71
14348	120	32	137	103	14400	120	32	137	103
14352	58	43	6	148	14404	0	41	58	2
14356	99	1	126	32	14408	58	103	58	40
14360	71	120	32	137	14412	30	37	103	58
14364	103	48	41	19	14416	59	4	37	103
14368	148	99	2	126	14420	62	43	4	2
14372	32	71	120	32	14424	48	134	219	2
14376	137	103	0	41	14428	70	80	219	36
14380	32	148	99	3	14432	38	103	62	43
14384	126	32	71	120	14436	5	5	7	2

14440	0	38	2	15		14476	30	40	6	133
14444	134	219	2	70		14480	99	61	56	2
14448	80	219	2	245		14484	55	2	86	50
14452	38	36	38	2		14488	121	192	169	0
14456	0	38	2	56		14492	0	0	2	254
14460	134	219	36	38		14496	3	67	0	2
14464	103	58	59	5		14500	148	53	16	32
14468	7	2	0	38		14504	155	148	25	55
14472	55	36	132	99						

Maschinenteil PC 126X

16384	132	2	176	219		16472	176	134	219	2
16388	2	64	80	219		16476	101	80	219	36
16392	2	2	148	219		16480	38	103	62	43
16396	120	64	137	103		16484	5	5	7	2
16400	58	43	6	148		16488	0	38	2	143
16404	99	1	126	64		16492	134	219	2	101
16408	71	120	64	137		16496	80	219	2	245
16412	103	48	41	19		16500	38	36	38	2
16416	148	99	2	126		16504	0	38	2	184
16420	64	71	120	64		16508	134	219	36	38
16424	137	103	53	41		16512	103	58	59	5
16428	32	148	99	3		16516	7	2	0	38
16432	126	64	71	120		16520	55	36	132	99
16436	64	137	103	56		16524	30	40	6	133
16440	41	45	148	99		16528	99	93	56	2
16444	4	126	64	71		16532	55	2	86	50
16448	120	64	137	103		16536	121	191	12	0
16452	0	41	58	2		16540	0	0	2	110
16456	58	103	58	40		16544	3	100	0	2
16460	30	37	103	58		16548	148	53	16	64
16464	59	4	37	103		16552	155	148	25	55
16468	62	43	4	2						

Maschinenteil PC 1350

8240	132	2	224	219		8260	99	1	126	32
8244	2	32	80	219		8264	119	120	32	185
8248	2	2	148	219		8268	103	48	41	19
8252	120	32	185	103		8272	148	99	2	126
8256	58	43	6	148		8276	32	119	120	32

8280	185	103	65	41	8348	134	219	2	108
8284	32	148	99	3	8352	80	219	2	245
8288	126	32	119	120	8356	38	36	38	2
8292	32	185	103	0	8360	0	38	2	232
8296	41	45	148	99	8364	134	219	36	38
8300	4	126	32	119	8368	103	58	59	5
8304	120	32	185	103	8372	7	2	0	38
8308	0	41	58	2	8376	55	36	132	99
8312	58	103	58	40	8380	78	56	6	133
8316	30	37	103	58	8384	99	61	56	2
8320	59	4	37	103	8388	55	2	86	50
8324	62	43	4	2	8392	233	75	78	0
8328	224	134	219	2	8396	0	0	2	204
8332	108	80	219	36	8400	3	99	0	2
8336	38	103	62	43	8404	148	53	16	32
8340	5	5	7	2	8408	203	148	25	55
8344	0	38	2	191					

Für den PC 1350 ohne RAM-Erweiterung, also mit nur 4K, ist die Installation des ASSEMBLERs nicht möglich. Wir raten Ihnen dringend zum Kauf einer RAM-Erweiterung. Diese Ausgabe lohnt sich für einen so teuren und leistungsstarken Rechner auf jeden Fall.

Sind Sie mit dem POKE fertig? Überprüfen Sie diesen Teil mit dem Maschinenprogrammaufruf:

PC 14XX: CALL 8192
PC 126X: CALL 16384
PC 1350: CALL 8240

Falls der Rechner stillschweigend ins BASIC zurückkehrt, ist die Wahrscheinlichkeit groß, daß das Maschinenprogramm richtig eingegeben worden ist.

Ändern Sie nun die BASIC-Anfangsadresse:

PC 1401/21: POKE 18145,172,56
NEW

PC 1402: POKE 18145,172,32
NEW

PC 1260: POKE 26337,172,88
NEW

PC 1261: POKE 26337,172,64
NEW

PC 1350: POKE 28417,220,64
(12K RAM) NEW

PC 1350: POKE 28417,220,32
(20K RAM) NEW

Der Maschinenteil ist nun 'unsichtbar', er befindet sich in den Adressen vor dem BASIC-Speicher. Jetzt können Sie den BASIC-Teil ohne Bedenken eingeben, da das Maschinenprogramm nicht mehr überschrieben werden kann.

BASIC-Teil für alle Rechner

5: " LII20LIJ21LIA22LIB	M259?160?161?162
23IX14DX15IY16DY17MV	30: " ?163INCI164DECI165
W18EXW19MVB110EXB111	INCA166DECA167ADM168
ADN112SBN113ADW114	SBM169ANMA1700RMA171
10: " ADB120LIDP316LIDL2	INCK172DECK173
17SBB121LIQ219RC1209	35: " INCV174DECY175INA1
SR1210JP3121?123MVWD	76NDPW177WAIT278?179
124EXWD125MVBD126	INCP180DECP181STD182
15: " EXBD127SRW128SL190	MVDM183READ186
FILM130FILD131LDP132	40: " MVMD185LDD187SWP18
LDQ133LDR134?135IXL1	8LDM189SLW129POP191?
36DXL137IYS138	1920UTA193?1940UTF19
20: " DYS139JRNZP240JRNZ	5ANIM2960RIM297
M241JRNCP242JRNCM243	45: " TSIM298CPIM299ANIA
JRP244JRM245?146LOOP	21000RIA2101TSIA2102
247STP148STQ149	CPIA2103?1104?1106TE
25: " STR150?151PUSH152D	ST2107CAL2224
ATA153?154RTN155JRZP	50: " ?1110?1111ADIM2112
256JRZM257JRCP258JRC	SBIM2113?1114?1115AD

IA2116SBIA2117?1118?	201INCW1202
1119CALL3120	65:" DECW1203INB1204?12
55:" 21123JPNZ3124JPNC3	05NOPT1206?1207SC120
125JPZ3126JPC3127LP1	85BW115LIP218WRIT121
128?1190?1191INCJ119	1ANID22120RID2213
2DECJ1193?174	70:" TSID2214LEAVE1216?
60:" INCB1194DECB1195AD	1217EXAB1218EXAM1219
CM1196SBOM1197?1198C	?1220OUTB1221?1222OU
PMA1199INCL1200DECL1	TC1223READM184

BASIC-Teil für den PC 14XX

```

75:CLEAR :WAIT 0:INPUT
  "ANFANGSADR: ";A:0=1
  10:P=100:IF A<17872-
    MEM PAUSE "ROM"
85:Z0$="" :IF A>17846
  PAUSE "END"
90:GOTO 0
100:BEEP 1
110:G=0:PRINT STR$ A+" "
  +Z0$:CALL 5208:PRINT
  "" :E=PEEK 24634
115:IF E>31 AND E<91 LET
  Z0$=Z0$+CHR$ E:GOTO
  0
120:IF E=15 LET Z0$=
  LEFT$ (Z0$,LEN Z0$-1
  ):GOTO 0
125:IF E=13 THEN 175
135:IF E=2 THEN 75
140:IF E=5 LET A=A+H:J=H
  :GOTO 265
145:IF E=4 LET A=A-J:J=0
  :GOTO 265
150:IF E=180 INPUT "DRUC
  K BIS: ";J:0=390:
  GOTO 380
170:GOTO 0
175:IF Z0$<"A" GOTO P
180:G=G+1:IF MID$ (Z0$,G
  ,1)>"@" THEN 180

```

```

185:D$=LEFT$ (Z0$,G-1):K
  =LEN D$:FOR I=1 TO K
  :A(16+I)=ASC MID$ (D
  $,I,1):NEXT I:IF K<2
  GOTO P
186:IF D$="PTC" GOTO 360
187:IF D$="ETC" POKE A,1
  05:A=A+1:GOTO 370
190:Y=1:GOSUB 301:Y=0:C=
  VAL C$:POKE 18163,6:
  B$="":IF H=0 GOTO P
205:FOR I=G TO LEN Z0$:B
  $=B$+MID$ (Z0$,I,1):
  NEXT I:IF LEN B$=0
  AND H>1 GOTO P
210:B=VAL B$:IF C=128
  LET C=C+B:IF B>63
  GOTO P
225:IF C=224 LET I= INT
  (B/256):B=B-I*256:C=
  C+I:IF I>31 GOTO P
230:IF H=3 LET M= INT (B
  /256):POKE A+2,B-M*2
  56:B=M
235:IF B>255 GOTO P
240:IF H=1 LET B=PEEK (A
  +1)
245:POKE A,C,B:A=A+H:
  GOTO 85
265:IF X=0 GOTO 271

```

```

266:I=I-1:GOSUB 350:H=3:
    IF W=0 LET X=0
267:IF I=-1 LET Z0$=STR$(
    (256*F+PEEK (Z+1)):H
    =2:X=0
268:GOTO 0
271:Z=8347:H= INT (A/256
    ):POKE Z+3,2,A-256*H
    ,3,H:CALL Z+3
280:F=PEEK Z:B$=" " :IF Y
    =1 RETURN
281:IF F=122 LET V=PEEK
    (Z+1):W=0:I=1:X=1:H=
    1:Z0$="PTC":GOTO 0
282:IF F=105 LET I=V:X=1
    :H=1:W=1:Z0$="ETC":
    GOTO 0
285:IF F>127 AND F<192
    LET D$="LP":B$=B$+
    STR$ (F-128):H=1:
    GOTO 325
290:IF F>223 LET D$="CAL
    ":B$=B$+STR$ (PEEK (
    Z+1)+(F-224)*256):H=
    2:GOTO 325
300:F$=STR$ F:Q=58:K=LEN
    F$:FOR I=1 TO K:A(17
    +I)=ASC MID$ (F$,I,1
    ):NEXT I:K=K+1
301:C$="":D$="":I=8201:
    POKE I,K:POKE I+7,Q:
    POKE I+20,R:POKE I+3
    S:POKE I+46,T:M=41
    :IF Q=58 LET M=43
302:I=8260:POKE I,U:POKE
    I+6,Q:POKE I-51,M:
    CALL I-68:I=PEEK (Z+
    1):H=VAL H$ :IF Y=1
    RETURN
305:IF D$=? LET D$=D$+
    F$
310:IF H=1 LET B$="":
    GOTO 325
315:IF H=2 LET B$=B$+
    STR$ I:GOTO 325
320:B$=B$+STR$ (I*256+
    PEEK (Z+2))
325:Z0$=D$+B$ :GOTO 0
350:Y=1:GOSUB 271:Y=0:Z0
    $=STR$ F+", "+STR$ (
    PEEK (Z+1)*256+PEEK
    (Z+2)):RETURN
360:INPUT "ANZ. VGL: ";Y
    :INPUT "ENDADR: ";C:
    N= INT (C/256):POKE
    A,122,V,N,C-N*256:A=
    A+4:GOTO 85
370:FOR I=1 TO V:INPUT "
    WENN AKKU= ";M:INPUT
    "SPRINGE ZU: ";C:N=
    INT (C/256):POKE A,M
    ,N,C-256*N:A=A+3:
    NEXT I
375:INPUT "SONST: ";C:N=
    INT (C/256):POKE A,N
    ,C-256*N:A=A+2:GOTO
    85
380:IF A>J LET 0=110
385:A=A+H:GOTO 265
390:LPRINT STR$ A;" ";Z0
    $:GOTO 380

```

BASIC-Teil für den PC 126X

```

75:CLEAR : WAIT 0:
    INPUT "Anfangsadr: "
    ;A=0=110:P=100: IF A
    <25856- MEM PAUSE "R
    OM
85:Z0$="" : IF A>25833
    PAUSE " END
90:GOTO 0
100:BEEP 1
110:G=0: CURSOR 0: PRINT
    STR$ A+" "+Z0$: CALL
    4414:E= PEEK 26156:
    CLS
115:IF E>31 AND E<91 LET
    Z0$=Z0$+ CHR$ E:
    GOTO 0
120:IF E=15 LET Z0$=
    LEFT$ (Z0$, LEN Z0$-
    1): GOTO 0
125:IF E=13 THEN 175
135:IF E=2 THEN 75
145:IF E=5 CURSOR 24:
    PRINT STR$ A;" ";Z0$
    :A=A+H:J=H: GOTO 265
150:IF E=140 INPUT "Druc
    k bis: ";J:0=390:
    GOTO 380
160:IF E=4 LET A=A-J:J=0
    : GOTO 265
170:GOTO 0
175:IF Z0$<"A" GOTO P
180:G=G+1: IF MID$ (Z0$,
    G,1)>"@" THEN 180
185:D$= LEFT$ (Z0$,G-1):
    K= LEN D$: FOR I=1
    . TO K:A(16+I)= ASC
    MID$ (D$,I,1): NEXT
    I: IF K<2 GOTO P
186:IF D$="PTC" GOTO 360
187:IF D$="ETC" POKE A,1
    05:A=A+1: GOTO 370
190:Y=1: GOSUB 301:Y=0:C
    = VAL C$: POKE 26355
    ,6:B$="": IF H=0
    GOTO P
205:FOR I=G TO LEN Z0$:B
    $=B$+ MID$ (Z0$,I,1)
    : NEXT I: IF LEN B$=
    0 AND H>1 GOTO P
210:B= VAL B$: IF C=128
    LET C=C+B: IF B>63
    GOTO P
225:IF C=224 LET I= INT
    (B/256):B=B-I*256:C=
    C+I: IF I>31 GOTO P
230:IF H=3 LET M= INT (B
    /256): POKE A+2,B-M*
    256:B=M: IF B>65335
    GOTO P
235:IF B>255 GOTO P
240:IF H=1 LET B= PEEK (
    A+1)
245:POKE A,C,B:A=A+H:
    GOTO 85
265:IF X=0 GOTO 271
266:I=I-1: GOSUB 350:H=3
    : IF W=0 LET X=0
267:IF I=-1 LET Z0$=
    STR$ (256*F+ PEEK (Z
    +1)):H=2:X=0
268:GOTO 0
271:Z=16539:H= INT (A/25
    6): POKE Z+3,2,A-256
    *H,3,H: CALL Z+3
280:F= PEEK Z:B$=" ": IF
    Y=1 RETURN
281:IF F=122 LET V= PEEK
    (Z+1):W=0:I=1:X=1:H=
    1:Z0$="PTC": GOTO 10
    5
282:IF F=105 LET I=V:X=1
    :H=1:W=1:Z0$="ETC":
```

```

GOTO 105
285: IF F>127 AND F<192
    LET D$="LP": B$=B$+
    STR$ (F-128): H=1:
    GOTO 325
290: IF F>223 LET D$="CAL
    " : B$=B$+ STR$ ( PEEK
    (Z+1)+(F-224)*256): H
    =2: GOTO 325
300: F$= STR$ F: Q=58: K=
    LEN F$: FOR I=1 TO K
    : A(17+I)= ASC MID$ (
    F$,I,1): NEXT I: K=K+
    1
301: D$="": D$="": I=16393:
    POKE I,K: POKE I+7,Q
    : POKE I+20,R: POKE
    I+33,S: POKE I+46,T:
    M=41: IF Q=58 LET M=
    43
302: I=16452: POKE I,U:
    POKE I+6,Q: POKE I-5
    1,M: CALL I-68:H=
    VAL H$: IF Y=1
    RETURN
305: IF D$=? LET D$=D$+
    F$
310: IF H=1 LET B$="":
    GOTO 325
315: IF H=2 LET B$=B$+
    STR$ PEEK (Z+1):
    GOTO 325
320: B$=B$+ STR$ ( PEEK (
    Z+1)*256+ PEEK (Z+2)
    )
325: Z0$=D$+B$: GOTO 0
350: Y=1: GOSUB 271: Y=0: Z
    0$= STR$ F+", "+ STR$(
    PEEK (Z+1)*256+
    PEEK (Z+2)): RETURN
360: INPUT "ANZ VGL: "; V:
    INPUT "Endaddr: "; C:N
    = INT (C/256): POKE
    A,122,V,N,C-N*256:A=
    A+4: GOTO 85
370: FOR I=1 TO V: INPUT
    "Wenn Akku= "; M:
    INPUT "Springe zu: "
    ; C:N= INT (C/256):
    POKE A,M,N,C-256*N:A
    =A+3: NEXT I
375: INPUT "Sonst: "; C:N=
    INT (C/256): POKE A,
    N,C-256*N:A=A+2:
    GOTO 85
380: IF A>J LET O=110
385: A=A+H: GOTO 265
390: LPRINT STR$ A;" "; Z0
    $: GOTO 380

```

BASIC-Teil für den PC 1350

```

75: CLEAR : WAIT 0:
    INPUT "ANFANGSADR: "
    ; A:O=110:P=100: IF A
    <27696- MEM PRINT "R
    OM
85: Z0$="": IF A>27673
    PRINT "END
90: GOTO 0
100: BEEP 1

```

```

110: G=0: PRINT STR$ A;" "
    "; Z0$: CALL 4618:E=
    PEEK 28503: IF E<95
    AND E>31 LET Z0$=Z0$+
    + CHR$ E: CLS :
    GOTO 110
120: IF E=15 LET Z0$=
    LEFT$ (Z0$, LEN Z0$-
    1): GOTO 0

```

```

125:IF E=13 THEN 175
135:IF E=2 THEN 75
145:IF E=5 LET A=A+H:J=H
    : GOTO 265
146:IF E=4 LET A=A-J:J=0
    : GOTO 265
150:IF E=140 INPUT "DRUC
    K BIS: ";J:0=390:
    GOTO 380
170:GOTO 0
175:IF Z0$<"A" GOTO P
180:G=G+1: IF MID$(Z0$,G,1)>"@" THEN 180
185:D$= LEFT$(Z0$,G-1):
    K= LEN D$: FOR I=1
    TO K:A(16+I)= ASC
    MID$(D$,I,1): NEXT
    I: IF K<2 GOTO P
186:IF D$="PTC" GOTO 360
187:IF D$="ETC" POKE A,1
    05:A=A+1: GOTO 370
190:Y=1: GOSUB 301:Y=0:C
    = VAL C$: POKE 28459
    ,6:B$="": IF H=0
    GOTO P
205:FOR I=G TO LEN Z0$:B
    $=B$+ MID$(Z0$,I,1)
    : NEXT I: IF LEN B$=
    0 AND H>1 GOTO P
210:B= VAL B$: IF C=128
    LET C=C+B: IF B>63
    GOTO P
225:IF C=224 LET I= INT
    (B/256):B=B-I*256:C=
    C+I: IF I>31 GOTO P
227:IF E=3 LET B= ABS (B
    -A-1)
230:IF H=3 LET M= INT (B
    /256): POKE A+2,B-M*
    256:B=M
235:IF B>255 GOTO P
240:IF H=1 LET B= PEEK (
    A+1)
245:POKE A,C,B:A=A+H:
    GOTO 85
265:IF X=0 GOTO 271
266:I=I-1: GOSUB 350:H=3
    : IF W=0 LET X=0
267:IF I=-1 LET Z0$=
    STR$(256*F+ PEEK (Z
    +1)):H=2:X=0
268:GOTO 0
271:Z=8395:H= INT (A/256
    ): POKE Z+3,2,A-256*
    H,3,H: CALL Z+3
280:F= PEEK Z:B$="":
    IF Y=1 RETURN
281:IF F=122 LET V=
    PEEK (Z+1):W=0:I=1:X
    =1:H=1:Z0$="PTC":
    GOTO 0
282:IF F=105 LET I=V:X=1
    :H=1:W=1:Z0$="ETC":
    GOTO 0
285:IF F>127 AND F<192
    LET D$="LP":B$=B$+
    STR$(F-128):H=1:
    GOTO 325
290:IF F>223 LET D$="CAL
    ":B$=B$+ STR$(F-
    224)*2
    56):H=2: GOTO 325
300:F$= STR$ F:Q=58:K=
    LEN F$: FOR I=1 TO K
    :A(17+I)= ASC MID$(F$,
    I,1): NEXT I:K=K+
    1
301:C$="":D$="":I=8249:
    POKE I,K: POKE I+7,Q
    : POKE I+20,R: POKE
    I+33,S: POKE I+46,T:
    M=41: IF Q=58 LET M=
    43
302:I=8308: POKE I,U:
    POKE I+6,Q: POKE I-5
    1,M: CALL I-68:I=
    PEEK (Z+1):H= VAL H$:
    : IF Y=1 RETURN
305:E=10: IF D$="?" LET
    D$=D$+F$
```

```

310:IF H=1 LET B$="":
    GOTO 325
315:IF H=2 LET B$=B$+
    STR$ I: GOTO 325
320:B$=B$+ STR$ (I*256+
    PEEK (Z+2))
325:Z$=D$+B$: GOTO 0
350:Y=1: GOSUB 271:Y=0:Z
    $= STR$ F+", "+"
    STR$ ( PEEK (Z+1)*25
    6+ PEEK (Z+2)):
    RETURN
360:INPUT "ANZ. VGL: ";V
    : INPUT "ENDADR: ";C
    :N= INT (C/256):
    POKE A,122,V,N,C-N*2

56:A=A+4: GOTO 85
370:FOR I=1 TO V: INPUT
    "WENN AKKU= ";M:
    INPUT "SPRINGE ZU: "
    ;C:N= INT (C/256):
    POKE A,M,N,C-256*N:A
    =A+3: NEXT I
375:INPUT "SONST";C:N=
    INT (C/256): POKE A,
    N,C-256*N:A=A+2:
    GOTO 85
380:IF A>J LET D=110
385:A=A+H: GOTO 265
390:LPRINT STR$ A;" ";Z0
    $: GOTO 380

```

Besitzen Sie keinen Drucker oder sind Sie besonders stark auf Speicherplatz angewiesen, so können Sie die Zeilen 150, 380, 385 und 390 löschen. Damit sparen Sie 85 Bytes. Diese Zeilen sind für die Druckroutine bestimmt und nicht unbedingt notwendig, da Listings auch 'von Hand' erstellt werden können, indem man mit SHIFT ENTER die Ausgabe auf den Drucker festlegt.

Sind Sie mit der Eingabe fertig? Speichern Sie nun den ASSEMBLER unbedingt auf Cassette ab, bevor Sie irgendetwas ausprobieren!

Speichern des ASSEMBLERS

PC 1401/21: CSAVE M "ASSEMBLER";14336,18148

PC 1402: CSAVE M "ASSEMBLER";8192,18148

PC 1260: CSAVE M "ASSEMBLER";22528,26340

PC 1261: CSAVE M "ASSEMBLER";16384,26340

PC 1350: siehe weiter unten

Laden des ASSEMBLERs

Für alle PCs (außer PC 1350): CLOAD M "ASSEMBLER"

Sicher haben Sie bemerkt, daß Sie bei dieser Art der Speicherung immer den ganzen BASIC-Teil zusammen mit den Systemvariablen speichern. Das geht zwar relativ langsam, ist dafür aber einfach und narrensicher.

Profis und alle PC 1350-Benutzer können (müssen) das Programm natürlich auch separat als BASIC- und Maschinenprogramm speichern. Aber Achtung: Nach dem Laden des Maschinenprogramms muß die BASIC-Anfangsadresse wieder nach oben versetzt werden, erst dann können Sie das BASIC-Programm laden!

Diese Methode lohnt sich jedoch nur für PCs mit einem grossen Speicher (10K oder mehr):

PC 1402:

Speichern CSAVE M "ASSEMBLER";8192,8364

 CSAVE "ASSEMBLER"

Laden CLOAD M "ASSEMBLER"

 POKE 18145,172,32

 NEW

 CLOAD "ASSEMBLER"

PC 1261:

Speichern	CSAVE M "ASSEMBLER";16384,16556
	CSAVE "ASSEMBLER"
Laden	CLOAD M "ASSEMBLER"
	POKE 26337,172,64
	NEW
	CLOAD "ASSEMBLER"

PC 1350:

Speichern	CSAVE M "ASSEMBLER";16384,16556
	CSAVE "ASSEMBLER"
Laden	CLOAD M "ASSEMBLER"
12K RAM:	POKE 28417,220,64
	NEW
20K RAM:	POKE 28417,220,32
	NEW
	CLOAD "ASSEMBLER"

Sobald der ASSEMBLER auf Cassette gespeichert ist, können Sie mit dem Austesten beginnen.

Bedienung des ASSEMBLERS

Folgende Möglichkeiten stehen Ihnen zur Verfügung:

Funktion	Tastenkombination
Neue Anfangsadresse	C-CE bzw. CL,CLS
Eingabe von Mnemonics	'A'...'Z', '1'...'9'
Abschließen der Eingabe	ENTER
Auflisten der nächsten Programmzeile	↓
Auflisten der letzten Programmzeile (nur einmal)	↑

Rücklaufstaste (Korrektur)	<
Ausdrucken von Listings	SHIFT+'L'
Abbrechen des ASSEMBLERS	BRK

Wir wollen diese Funktionen anhand eines kleinen Beispiels ausprobieren:

PC 14XX	PC 126X	PC 1350	Mnemonics
17400	25500	25900	LIA 10
17402	25502	25902	LIDP 25910
17405	25505	25905	STD
17406	25506	25906	RTN

Starten Sie den ASSEMBLER mit RUN. Falls Sie später Listings erstellen wollen (wir werden diese Funktion auch benutzen), müssen Sie Ihren Drucker schon vor dem Starten des ASSEMBLERS korrekt anschließen. Nun geben Sie die Anfangsadresse an, für unser Beispiel also '17400' (für den PC 126X und PC 1350 nehmen Sie die oben angegebenen Adressen). Nach dem Betätigen von ENTER wird diese Adresse angezeigt. Nun können Sie mit dem ASSEMBLER arbeiten.

- *Eingabe*

Geben Sie 'LIA 10' ein. Auf dem Display steht nun:

17400 LIA 10

Betätigen Sie nun die ENTER-Taste. Nach etwas mehr als zwei Sekunden hat der ASSEMBLER den Befehl in Codes umgesetzt, und es erscheint die neue Adresse, also 17402. Diese Adresse wird Ihnen immer vorgegeben. Sie dient zur Kontrolle. Versuchen Sie deshalb nicht, Sie zu verändern; Sie können damit höchstens einen Fehler erzeugen.

Das Leerzeichen zwischen 'LIA' und '10' ist nicht unbedingt notwendig, Sie können es also auch ruhig weglassen.

Bei der zweiten Zeile geben wir absichtlich einen Fehler ein:

17402 LIDP 259100

Die Adresse nach LIDP ist natürlich zu groß. Der Rechner merkt das, gibt einen BEEP von sich und kehrt ins Anzeigefeld zurück. Der ASSEMBLER erkennt praktisch jeden Tipp- oder Syntaxfehler und kehrt dann unverrichteter Dinge mit einem BEEP ins Anzeigefeld zurück.

Nun berichtigen wir diesen Fehler: Betätigen Sie einfach die '<-Taste und schon ist der Fehler korrigiert! Drücken Sie jetzt ENTER. Sie sehen: Der Befehl wird nun akzeptiert. Geben Sie auch noch die restlichen zwei Zeilen ein:

17405 STD

17406 RTN

Sobald die Adresse '17407' erscheint, ist das ganze Programm in lauffähigen Codes gespeichert.

- *Neue Anfangsadresse*

Wie Sie schon wissen, wird die Anfangsadresse vorgegeben und kann mit der Rücklauftaste nicht verändert werden. Sie können aber die rote C-CE-Taste bzw. CL-Taste betätigen und dann eine neue Anfangsadresse wählen.

Diese neue Anfangsadresse wird mit Hilfe der Funktion MEM zuerst auf ihre Gültigkeit hin überprüft. Besteht die Gefahr, daß ein BASIC-Programm überschrieben wird, erscheint die Meldung 'ROM'. Sind Sie dagegen am oberen Ende des BASIC-Speichers angelangt, meldet sich der ASSEMBLER mit 'ENDE'.

- *Listen*

Wir wollen unser Programm noch einmal anschauen, um uns zu versichern, ob sich kein Fehler eingeschlichen hat. Schauen wir also unser Programm an! Drücken Sie die rote C-CE-Taste und geben Sie die neue Anfangsadresse ein (17400, 25500 oder 25900). Drücken Sie dann die ' \downarrow '-Taste, den Pfeil nach unten. Gut zwei Sekunden später erscheint die erste Befehlszeile. Nach erneutem Drücken der ' \downarrow '-Taste erscheint die zweite Zeile. Gehen Sie Ihr Programm nun Zeile um Zeile durch und überprüfen Sie es auf Fehler.

Beim Listen können Sie auch eine Zeile zurückgehen. Dies geschieht über die ' \uparrow '-Taste. Geben Sie also ' \uparrow ' ein, und die letzte Befehlszeile erscheint wieder. Weiteres Drücken der ' \uparrow '-Taste hat aber keine Wirkung mehr: Sie können nur einen Befehl zurückgehen. Weiteres Zurückgehen ist unmöglich, da der ASSEMBLER nicht wissen kann, welche Codes Maschinenbefehle und welche nur Zahlenwerte sind.

- *Erstellen von Programmlistings*

Nun wollen wir unser Beispiel auf den Drucker ausgeben. Sie haben hoffentlich den Drucker schon vor dem Start des ASSEMBLERS eingeschaltet. Andernfalls brechen Sie einfach den ASSEMBLER mit BRK ab, schließen Sie den Drucker korrekt an, schalten ihn ein, und starten den ASSEMBLER erneut mit RUN.

Wir wählen nochmals die Anfangsadresse 17400, 25500 oder 25900 mit C-CE. Zuerst schauen Sie sich mit der ' $'$ -Taste an, was in dieser Zeile steht. Damit können Sie sich überzeugen, daß Sie sich wirklich am Anfang Ihres Programms befinden. Nun geben Sie SHIFT und 'L' ein. Der ASSEMBLER fragt Sie nach der Endadresse, der letzten Zeile Ihres Programms. In unserem Fall ist das die Zeile 17406. Nach der Eingabe von '17406' und ENTER fährt der Rechner vollautomatisch fort. Bei

größeren Listings können Sie hier also ruhig eine Kaffeepause einlegen, um danach das fertige Listing vom Drucker in Empfang zu nehmen.

Jetzt müssen wir unser fertiges Programm aber auch noch laufen lassen. Wir verlassen den ASSEMBLER mit BRK und geben je nach Rechner ein: CALL 17400, CALL 25500 oder CALL 25900. Lesen Sie die Speicherzelle 25910 aus! Ergebnis: PEEK 25910='10'.

Kapitel 6

Befehlssatz: Was die CPU so alles kann

Der ganze nächste Teil dieses Buches dreht sich um die Befehle der Maschinensprache. Der Aufbau dieses Kapitels richtet sich nach dem Schwierigkeitsgrad der zu behandelnden Befehle. Das erleichtert Ihnen den Einstieg: Sie können sich Schritt für Schritt mit der neuen Materie bekannt machen und werden dabei nicht überfordert. Wichtig ist ja, daß Sie diese neue Programmiersprache von Grund auf kennenlernen und verstehen.

Deshalb sind auch zu allen Beschreibungen anschauliche Beispiele aus der Praxis angegeben, wo Sie die erworbenen Kenntnisse gerade anwenden können. Systemadressen, die für Unterprogramme wie BEEP-Routine oder Tastaturabfrage und für das Arbeiten mit den Standardvariablen wichtig sind, orientieren sich am PC 1401/02 (PC 140X). Besitzen Sie einen anderen PC, z.B. einen PC 1421, einen PC 1260/61 (PC 126X) oder einen PC 1350, so halten Sie sich einfach an die Anmerkungen direkt nach den Programmbeispielen. Dann kann nichts schiefgehen.

Vielleicht wäre es noch gut, einmal einen Blick auf den Anhang C zu werfen. Dort finden Sie nämlich eine Übersicht über sämtliche verwendeten ROM-Unterprogramme.

Viele Beispiele arbeiten mit einem kleinen Ein-/Ausgabe-Puffer im externen RAM, über den Werte ins interne RAM geladen werden und Ergebnisse zurückgespeichert werden. Als gemeinsamer Puffer für alle Rechner wird der RAM-Bereich 25900 bis 25912 verwendet. Natürlich ist er bei den verschiedenen Rechnern auch unterschiedlich belegt:

RAM-Bereich 25900 - 25912

PC 14XX:	Systemvariablen
PC 126X:	Standardvariablen
PC 1350:	BASIC-Speicher

Der Einsatz des Puffers unterliegt dadurch bestimmten Einschränkungen. Beim PC 14XX wird er gelöscht, sobald Sie in den CAL-Modus wechseln. Der ASSEMBLER überschreibt den Puffer des PC 126X, da er mit den Standardvariablen arbeitet. Und beim PC 1350 sollten Sie darauf achten, daß sich BASIC-Programme und Puffer nicht überschneiden.

Bei allen Maschinenprogrammen sind die Adressen absichtlich nur unvollständig angegeben. Für die ersten drei Ziffern einer Adresse stehen nur Punkte. Das hat den Vorteil, daß Sie nicht an einen bestimmten Adressbereich gebunden sind, sondern die Beispiele an beliebigen RAM-Adressen ablegen können. Die beiden letzten Ziffern sind jedoch konsequent angegeben, da sie für die Eingabe und das Ändern des Programms sehr nützlich sind.

Beispiel:

...04 LIA 45

Beliebiger RAM-Bereich Zur Bearbeitung, Änderung

Alle Programmbeispiele können Sie mit dem ASSEMBLER eingeben.

Steht kein ASSEMBLER zur Verfügung, ist auch eine Eingabe über POKE-Befehle möglich. Die dazu nötigen Codes sind angegeben. Aber passen Sie gut auf, wenn Sie keinen PC 140X besitzen: Im Code-Teil sind die Abweichungen von den Systemadressen des PC 140X nicht mehr speziell erwähnt. Hier gelten die Anmerkungen des Mnemonic-Teil; die entsprechenden Codes für Ihren PC sind unter dem Listing in Klammern angegeben.

Kommen wird also zum Befehlssatz unserer CPU.

6.1 Ladebefehle

*LII, LIJ, LIA
LIB, LIP, LIQ*

Sie kennen die Register I, J, A, B, P und Q ja bereits aus dem letzten Kapitel. Wir haben sie dort ähnlich wie BASIC-Speicher (A, B, C usw.) behandelt. Wollen Sie in BASIC die Zahl 234 im Speicher A ablegen, geben Sie 'A=234' ein. Ganz ähnlich funktionieren auch die Ladebefehle der Maschinensprache. Mit LIA 234 erhält das A-Register den Wert 234, LII 168 weist dem I-Register den Wert 168 zu. Alle Ladebefehle beginnen jeweils mit 'LI' ('Load immediate', 'Lade unmittelbar'). Wir beschreiben diese Art der Speicherung mit dem Zeichen ' \rightarrow '. Dieser Pfeil wird immer dann verwendet, wenn Werte direkt in Register oder Speicherzellen abgelegt werden.

I, J, A und B sind 8-Bit-Register. Sie fassen also maximal 8 Bits; die Bits 0 bis 7. Folglich ist 255 die höchste Zahl, die wir in einem solchen Register speichern können. Dagegen sind die Register P und Q nur 7-Bit-Register. Sie fassen maximal den Wert 127.

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	
LII n	0,n	n \rightarrow I	- -	I=n
LIJ n	1,n	n \rightarrow J	- -	J=n
LIA n	2,n	n \rightarrow A	- -	A=n
LIB n	3,n	n \rightarrow B	- -	B=n
LIP n	18,n	n \rightarrow P	- -	P=n
LIQ n	19,n	n \rightarrow Q	- -	Q=n
LP n	128+n	n \rightarrow P	- -	P=n
LIDP nm	16,n,m	nm \rightarrow DP	- -	DPH=n:DPL=m
LIDL m	17,m	m \rightarrow DPL	- -	DPL=m

*Der LP-Befehl**LP*

Einen Spezialfall stellt der LP-Befehl dar. Eigentlich hat er dieselbe Funktion wie der LIP-Befehl; beide Befehle laden ja das P-Register. Mit LP brauchen wir dafür aber nicht zwei, sondern nur ein Byte. Befehlscode und Argument sind nämlich verknüpft. LP hat den Befehlscode 128. Den endgültigen Code (Op-Code) erhält man, indem man das Argument n zum Befehlscode addiert.

$$\text{Op-Code} = \text{Befehlscode} + \text{Argument} = 128 + n$$

Der Zahlenwert n darf höchstens 63 betragen; n liegt also immer im Bereich von 0 bis 63. Sobald P auf eine interne Speicherzelle zeigen soll, deren Adresse 64 oder mehr beträgt, muß der LIP-Befehl verwendet werden. Keine Sorge: Mit dem Assembler müssen Sie die Op-Codes ja nicht selbst berechnen: Sie geben einfach das Mnemonic ein, z.B. LP 10. Um den Rest brauchen Sie sich nicht mehr zu kümmern.

Beispiele:

Mnemonic	Codes	Kommentar
LP 1	128+1=129	1->P
LP 10	138	10->P
LP 0	128	0->P
LIP 91	18,91	91->P

*Arbeiten mit dem Datenzeiger DP**LIDP, LIDL*

Der Datenzeiger DP weist auf eine Speicherzelle des externen RAM. Der DP ist ein 16-Bit-Adreßregister: In ihm steht eine bestimmte Adresse, die sich im Bereich von 0 bis 65535 bewegt.

Mit LIDP wird also eine neue 16-Bit-Adresse in den DP geladen. Dem Datenzeiger DP wird ein ganz neuer Wert zugewiesen; sowohl das Highbyte n als auch das Lowbyte m des DP wird verändert. Mit dem LIDL-Befehl dagegen kann man nur das Lowbyte m des Datenzeigers verändern, während das Highbyte unverändert bleibt. Sie können ein Byte sparen, wenn Sie beim Laden von Adressen mit gleichem Highbyte LIDL verwenden.

Beispiele:

Mnemonic	Codes	Kommentar
LIDP 25900	16,101,44	25900->DP
LIDL 45	17,45	45->DPL. Der DP zeigt jetzt auf die Speicherzelle 25901
LIDP 20000	16,78,32	20000->DP
LIDL 132	17,132	132->DPL. Der DP zeigt jetzt auf die Speicherzelle 20100

Ein- und Ausgabe, Byteaustausch

*LDD, MVMD, LDM
STD, MVDM, EXAM*

Auf die internen RAM-Register (A, B, I usw.) können wir von BASIC aus (mittels PEEK) ja nicht direkt zugreifen. Um diese Register auszulesen, müssen wir ihren Inhalt in einer Speicherzelle des externen RAM ablegen. Es kommt auch vor, daß wir Werte aus dem externen RAM ins interne RAM kopieren müssen.

Für solche Operationen brauchen wir Befehle, die den Datentransfer zwischen internem und externem RAM regeln. Sie sind in der folgenden Tabelle aufgeführt:

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	

Eingabebefehle

LDM	89	(P)->A	- -	A=M(P)
LDD	87	(DP)->A	- -	A=PEEK DP
MVMD	85	(DP)->(P)	- -	M(P)=PEEK DP

Ausgabebefehle

STD	82	A->(DP)	- -	POKE DP,A
MVDM	83	(P)->(DP)	- -	POKE DP,M(P)

Byteaustauschbefehle

EXAM	219	A <-> (P)	- -	-
EXAB	218	A <-> B	- -	-

Leider existiert kein Befehl, der nur den Inhalt des Akkus in (P) ablegt. Dazu müssen wir den EXAM-Befehl verwenden. EXAM vertauscht den Inhalt des Akkus mit dem Inhalt der Speicherzelle, auf die P zeigt. Nicht nur (P), sondern auch der Akku wird verändert.

Ein kleiner Tip: Brauchen Sie den Inhalt des Akkus noch, hängen Sie dem EXAM-Befehl einfach einen LDM-Befehl an. Das ist zwar umständlich, aber es funktioniert.

Ganz ähnlich wie EXAM funktioniert auch EXAB. Dieser Befehl vertauscht die Registerinhalte von A und B.

Beispiele:

Mnemonic	Codes	Kommentar
LIDP 25900	16,101,44	25900->DP
LDD 87	(DP)->A.	Im Akkumulator steht nun der Inhalt der Speicherzelle 25900
LIDP 25900	16,101,44	25900->DP
LP 9	137	9->P. P zeigt auf L
MVMD	85	(DP)->(P). Im L-Register steht nun der Inhalt der Speicherzelle 25900
LIA 20	2,20	20->A
LIDP 25900	16,101,44	25900->DP
STD	82	A->(DP). PEEK 25900 = 20
LIB 10	3,10	10->B
LIDP 25900	16,101,44	25900->DP
LP 3	136	3->P. P zeigt auf B
MVDM	83	(P)->(DP). PEEK 25900 = 10
LIA 20	2,20	20->A
LIB 10	3,10	10->B
EXAB	218	A<->B. A = 10, B = 20

Mit diesen Befehlen besitzen wir endlich auch das nötige Rüstzeug, um unsere Kenntnisse anzuwenden: Wir schreiben unser erstes Maschinenprogramm!

Das erste Maschinenprogramm

Wie Sie vielleicht schon festgestellt haben, wird jedes Maschinenprogramm mit RTN (Code 55) abgeschlossen. RTN kommt natürlich von 'RETURN', 'Kehre zurück'. Diesen Befehl verwendet der Rechner, um aus einem Unterprogramm ins Hauptprogramm zu springen oder ins BASIC zurückzukehren. Verges-

sen Sie also nie Ihr Programm mit RTN abzuschließen! Sonst geschieht folgendes: Der Rechner hat natürlich keine Ahnung, wo das Programm fertig ist. Deshalb sucht er nach einem RTN-Befehl. Brav macht er also weiter und führt die nachfolgenden Codes einfach als Befehle aus. Es liegt auf der Hand, daß er sich bald in irgendeiner Routine verfängt und nicht mehr ins BASIC zurückfindet: Er stürzt ab.

Auch ein Unterprogramm muß immer mit RTN abgeschlossen werden, damit der Rechner wieder zurück ins Hauptprogramm findet.

Geben Sie das folgende Programm mit dem ASSEMBLER ein. Falls Sie ihn noch nicht auf Ihrem PC installiert haben, raten wir Ihnen dringend, das zuvor noch zu erledigen. Die Anleitung dazu finden Sie in Kapitel 5.9. Wie schon erwähnt, können Sie aber auch die angegebenen Op-Codes verwenden und die Beispiele mit POKÉ eingeben.

Sie bestimmen die Anfangsadresse. Dazu suchen Sie einfach einen freien Adressbereich des externen RAM. Folgende Adressen gehen auf jeden Fall, wenn Sie nur den ASSEMBLER im BASIC-Speicher haben:

PC 1401:	ab 17400
PC 1402:	ab 11300
PC 1260:	ab 25500
PC 1261:	ab 19600
PC 1350:	ab 19600 (12K RAM)
PC 1350:	ab 11400 (20K RAM)

Nachdem Sie den ASSEMBLER mit 'RUN' gestartet haben, geben Sie Ihre Anfangsadresse ein. Keine Angst, Sie können Ihre BASIC-Programme gar nicht überschreiben: Falls ein Speicherbereich für Maschinenprogramme nicht geeignet ist, warnt Sie der ASSEMBLER durch die Meldung 'ROM' (Adresse zu niedrig) oder 'END' (Adresse zu hoch). Drücken Sie dann einfach die rote 'C-CE'-Taste (beim PC 126X und 1350 die rote 'CL'-Taste) und versuchen Sie es noch einmal mit einer besseren Adresse.

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP	25900	16,101,44 25900->DP
...03	LDI	87	(DP)->A
...04	LIDL	45	25901->DP
...06	STD	82	A->DP
...07	RTN	55	Zurück ins BASIC

Sobald der ASSEMBLER die Adresse '...08' anzeigt, ist auch der RTN-Befehl gespeichert. Jetzt drücken Sie einfach die 'BRK/ON'-Taste. Damit verlassen Sie das Programm. Mit PEEK können Sie nachschauen, ob die Codes mit denen des Listings übereinstimmen (diese Überprüfung ist allerdings nicht notwendig).

Das Programm kopiert den Inhalt der Speicherzelle 25900 in die Speicherzelle 25901. Dazu muß der Rechner das eine Byte in den Akku laden und an der neuen Adresse wieder zurückspeichern. Belegen Sie die Speicherzelle 25900 zuerst noch mit einem neuen Wert, damit Sie auch sehen, ob Ihr Maschinenprogramm funktioniert.

Jetzt starten Sie das Programm mit 'CALL ...00'. Mit PEEK 25901 schauen Sie nach, ob alles geklappt hat. Die Speicherzellen 25900 und 25901 sollten nun den gleichen Wert haben.

Genauso, wie man Daten zwischen dem Akkumulator und dem externen Speicher bewegen kann, funktioniert dies auch innerhalb des internen RAM. Nur ist der Austausch dabei nicht so offensichtlich, da man ja keinen direkten Zugriff auf die internen Register hat. Deshalb müssen in unseren Beispielen die internen Register auch oft ins externe RAM kopiert werden.

Adresse	Mnemonic	Codes	Kommentar
...00	LP 10	138	10->P
...01	LDM	89	(P)->A
...02	LP 11	139	11->P
...03	EXAM	219	A<->(P)
...04	LIDP 25901	16,101,45	25901->DP
...07	MVDM	83	(P)->(DP)
...08	RTN	55	Zurück ins BASIC

Hier wird ein Byte aus der internen Speicherzelle 10 in die Speicherzelle 11 kopiert und im externen RAM gespeichert (Adresse 25901).

6.2 8-Bit-Addition und -Subtraktion

Inkrement und Dekrement

*INCI, INCJ, INCA, INCB, INCP
INCK, INCL, INCV, INCW*

*DECI, DECJ, DECA, DECB, DECP
DECK, DECL, DECV, DECW*

Recht oft müssen wir zum Inhalt eines Registers nur 1 addieren oder subtrahieren. Eine Addition mit 1 nennt man Inkrementieren, eine Subtraktion von 1 Dekrementieren.

Die ersten drei Buchstaben aller Inkrement- bzw. Dekrementbefehle sind immer gleich: 'INC' steht natürlich für 'Increment' (Zuwachs) und 'DEC' für 'Decrement' (Abnahme). Der letzte Buchstabe kennzeichnet das betroffene Register; DECB z.B. dekrementiert das B-Register.

Mnemonic	Code	Wirkung	Flags C Z	BASIC
INCL	64	I+1->I	- -	I+1+1
DECL	65	I-1->I	- +	I-1+1
INCJ	192	J+1->J	- -	J+J+1
DECJ	193	J-1->J	- -	J-J-1
INCA	66	A+1->A	- -	A+A+1
DECA	67	A-1->A	- -	A-A-1
INCB	194	B+1->B	- -	B+B+1
DECB	195	B-1->B	- -	B-B-1
INCK	72	K+1->K	- -	K+K+1
DECCK	73	K-1->K	- -	K-K-1
INCL	200	L+1->L	- -	L+L+1
DECL	201	L-1->L	- -	L-L-1
INCV	74	V+1->V	- -	V+V+1
DECV	75	V-1->V	- -	V-V-1
INCW	202	W+1->W	- -	W+W+1
DECW	203	W-1->W	- -	W-W-1
INCP	80	P+1->P	--	P+P+1
DEC P	81	P-1->P	--	P-P-1

Die Inkrement- und Dekrementbefehle für das P-Register werden nicht für arithmetische Operationen verwendet; sie haben daher auch keinen Einfluß auf die CPU-Flags.

Wie Sie sehen, ist der Code eines Subtraktionsbefehls immer um 1 größer als der des entsprechenden Additionsbefehls. Diese Faustregel gilt übrigens für sämtliche arithmetischen Befehle.

Beispiel:

Die Standardvariable Z\$ beginnt beim PC 140X an der Adresse 17872¹⁾. Bei allen Textvariablen (Strings) steht im ersten Byte der Code 245, der sie als Textvariablen markiert. Das erste Byte

1) PC 140X: 17792 (69,128), PC 150X: 25804 (101,8), PC 1550: 27856 (108,48)

einer Textvariablen steht also erst nach diesem Code. Das folgende Programm verändert das erste Zeichen der Textvariable Z\$.

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 17873 ¹⁾	16,69,209	17873 ¹⁾ ->DP. Der DP zeigt auf das 1. Byte von Z\$
...03	LDI	87	(DP)->A. A = ASCII-Code des ersten Bytes
...04	INCA	66	A+1->A
...05	STD	82	A->(DP). Akku zurück-schreiben
...06	RTN	55	

Geben Sie für Z\$ zuerst einen Begriff oder Namen ein, z.B. Z\$= "GASE". Die Veränderung nach dem Aufrufen des Programms ist unverkennbar: In unserem Fall handelt es sich um ein bekanntes Nagetier.

Addition und Subtraktion

ADIA, ADIM, ADM
SBIA, SBIM, SBM

Mit diesen Befehlen sind wir schon in der Lage, mit 8-Bit-Zahlen zu rechnen. Die Additions- und Subtraktionsbefehle sind einfach: Zu einem Register wird einfach ein Wert n addiert oder subtrahiert. Die Flags werden wie üblich gesetzt. Sind Sie sich hier nicht mehr sicher, schauen Sie am besten nochmals in Kapitel 5.5 nach.

1) PC 1421: 17792 (69,128), PC 126X: 25856 (101,0), PC 1350: 27696 (108,48)

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	
ADIA n	116,n	A+n->A	* *	A=A+n
SBIA n	117,n	A-n->A	* *	A=A-n
ADIM n	112,n	(P)+n->(P)	* *	M(P)=M(P)+n
SBIM n	113,n	(P)-n->(P)	* *	M(P)=M(P)-n
ADM	68	(P)+A->(P)	* *	M(P)=M(P)+A
SBM	69	(P)-A->(P)	* *	M(P)=M(P)-A

Die Funktion eines Befehls können Sie auch hier leicht aus den Mnemonics ablesen: 'AD' steht für Addieren, 'SB' für Subtrahieren.

Beispiele:

Adresse	Mnemonic	Codes	Kommentar
...00	LII 80	0,80	80->I
...02	LP 0	128	0->P. P zeigt auf I
...03	SBIM 30	113,30	(P)-30->(P). I=50
...05	LIDP 25900	16,101,44	25900->DP
...08	MVDM	83	(P)->(DP)
...09	RTN	55	

$$I - 30 = 50$$

Ergebnis: PEEK 25900 = 50

Adresse	Mnemonic	Codes	Kommentar
...00	LIB 240	3,240	240->B
...02	LIA 60	2,60	60->A
...04	LP 3	131	3->P. P zeigt auf B
...05	ADM	68	(P)+A->(P). B=44 (Übertrag)

```

...06      LIDP 25900   16,101,44  25900->DP
...09      MVDM          83           (P)->(DP)
...10      RTN           55

```

$$B + A = 300 = 44$$

Ergebnis: PEEK 25900 = 44

6.3 Unterprogramme und der CPU-Stack

Mnemonic	Code	Wirkung	Flags C Z	BASIC
CALL nm	120,n,m	PC+3->(R-1,R-2) R-2->R n->PCH, m->PCL	- -	GOSUB nm (0<=nm<65536)
CAL nm	224+n,m	PC+2->(R-1,R-2) R-2->R n->PCH, m->PCL	- -	GOSUB nm (0<=nm<8192)
RTN	55	(R-1,R-2)->PC	- -	RETURN
PUSH	52	R-1->R, A->(R)	- -	
POP	91	(R)->A, R+1->R	- -	

Unterprogramme in der Maschinensprache

CALL, CAL

RTN

Den BASIC-Befehl CALL kennen Sie ja schon gut. Bis jetzt wußten Sie nur nicht, daß dieser Befehl auch im Befehlssatz der CPU vorkommt: Innerhalb eines Maschinenprogramms kann man mit 'CALL nm' ein weiteres Maschinenprogramm aufrufen. Sobald der Rechner beim Abarbeiten dieses Unterprogramms auf RTN stößt, kehrt er wieder ins Hauptprogramm zurück. CALL und RTN spielen in der Maschinensprache also etwa so zusammen wie GOSUB und RETURN in BASIC.

Die Rücksprungadresse, die den Weg zurück ins Hauptprogramm weist, wird auf dem Stack abgelegt. Die Kapazität des Stacks ist groß genug, um auf diese Weise Dutzende von Unterprogrammen zu verschachteln.

Aber wie arbeitet der CPU-Stack überhaupt? Um diese Frage zu beantworten, müssen wir den CALL-Befehl einmal gründlich unter die Lupe nehmen:

Wie Sie schon aus dem letzten Kapitel wissen, ist der Stackzeiger R für die Verwaltung des Stacks verantwortlich. Beim Aufrufen eines Unterprogramms durch CALL oder CAL wird die momentane Adresse+3 (PC+3) auf den Stack geladen (bei CAL: PC+2).

Da der Stack von oben nach unten verwaltet wird, wird das Highbyte dieser Adresse in der internen Speicherzelle (R-1), das Lowbyte in (R-2) abgelegt. Vom Stackzeiger R wird dann 2 subtrahiert. Alle Vorkehrungen für die spätere Rückkehr ins Hauptprogramm sind nun getroffen.

Der Rechner arbeitet jetzt an der Adresse weiter, die dem CALL- bzw. CAL-Befehl folgt. Sie können sich denken, was beim nächsten RTN-Befehl geschieht: Die CPU lädt die Rücksprungadresse vom Stack in den PC zurück und addiert 2 zum Stackzeiger R. Auch der Zustand des CPU-Stacks ist wieder wie vor dem Aufruf des Unterprogramms. Der Rechner fährt im Hauptprogramm weiter, als ob nie etwas gewesen wäre. So einfach ist das!

Der CAL-Befehl hat die gleiche Funktion wie sein großer Bruder CALL. Nur wurde er für ein spezielles Einsatzgebiet geschaffen: CAL kann man nur für das Aufrufen von Unterprogrammen des internen ROM (Adressen 0 bis 8191) verwenden. Ähnlich wie beim LP-Befehl wird der Op-Code aus dem Befehlscode und dem Highbyte der anzuspringenden Adresse (max. 31) gebildet. Damit kann man beim Aufrufen von ROM-Subroutinen ein Byte einsparen:

$$\text{Op-Code} = \text{Befehlscode} + \text{Highbyte}, \text{Lowbyte} = 224 + n,m$$

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	CAL 5208 ¹⁾	244,88	Tastatureabfrage. Code der gedrückten Taste -> Akku
...02	LIDP 25900	16,101,44	25900->DP
...05	STD	82	A->(DP)
...06	RTN	55	

Mit PEEK 25900 erfahren Sie den Code der gedrückten Taste.

Direkte Nutzung des Stacks

PUSH, POP

PUSH und POP bieten Ihnen die Möglichkeit, direkt auf den Stack zuzugreifen. PUSH dekrementiert den Stackzeiger R und legt dann den Inhalt des Akkus in der Speicherzelle (R) auf dem Stack ab. POP macht genau das Gegenteil: Der Inhalt der Speicherzelle (R) wird wieder in den Akku geladen und der Stackzeiger R wird inkrementiert.

Das Befehlspaar PUSH und POP kann man zum Beispiel benutzen, um Zwischenergebnisse des Akkus auf dem Stack abzulegen. Dazu ein Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 25900	16,101,44	25900->DP
...03	LDD	87	(DP)->A
...04	PUSH	52	R-1->R, A->(R)
...05	LIA 0	2,0	0->A. Akku löschen
...07	POP	91	(R)->A, R+1->R
...08	STD	82	A->(DP)
...09	RTN	55	

1) PC 126X: CAL 4414 (241,62), PC 1350: CAL 4618 (242,10)

Der Inhalt der Speicherzelle 25900 wird in den Akku geladen. Da der Inhalt des Akkus auf dem Stack zwischengespeichert wird, wirkt sich 'LIA 0' (Zeile ...05) nicht aus. Der Akku bekommt zwar für den Moment den Inhalt 0, aber schon der nächste Befehl lädt den alten Inhalt vom Stack zurück.

6.4 Arbeiten mit dem Akku

Auslesen und Manipulation der Register

LDP, LDQ, LDR

STP, STQ, STR

Außer den folgenden Befehlen gibt es keine Möglichkeit, die intern benutzten Zeigeregister P, Q und R auszulesen:

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	

Ladebefehle

LDP	32	P->A	- -	A=P
LDQ	33	Q->A	- -	A=Q
LDR	34	R->A	- -	A=R

Speicherbefehle

STP	48	A->P	- -	P=A
STQ	49	A->Q	- -	Q=A
STR	50	A->R	- -	R=A

Sie haben es sicher schon erraten: 'LD' kommt von 'Load' und bedeutet 'Laden', 'ST' kommt von 'Store' und bedeutet 'Speichern'. Mit diesen Befehlen können Sie über den Akku die Zeigeregister P, Q und R auslesen und verändern.

Anhand des R-Registers soll allgemein demonstriert werden, wie man diese Befehle einsetzt. Auf dieselbe Art und Weise können Sie auch die entsprechenden Befehle für das P- und Q-Register verwenden.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	CALL...07	120,LB,HB	Aufrufen des Unterprogramms ...07
...03	CALL 49321 ¹⁾	120,192,169	BEEP
...06	RTN	55	Zurück ins BASIC
...07	LIA 86	2,86	86->A
...09	STR	50	A->R. R = 86
...10	RTN	55	Zurück ins BASIC

Das Aufrufen dieses Maschinenprogramms nimmt der Rechner ruhig und gelassen hin. Kein BEEP ertönt, kein Ton, nichts! Und das, obwohl wir als zweites Unterprogramm die BEEP-Routine aufrufen!

Die Erklärung ist einfach: Der Rechner kehrt aus diesem Unterprogramm nicht zum Hauptprogramm, sondern direkt ins BASIC zurück. Dazu muß nur 86 ins R-Register geladen werden. In den internen Speicherzellen 86 und 87 steht nämlich die Rückprungadresse, die dem Rechner den Weg zurück ins BASIC weist.

Wir werden diese Behauptung gleich im nächsten Beispiel überprüfen:

1) PC 1421: 55539 (216,243), PC 126X: 48908 (191,12)
PC 1350: 49430 (193,22)

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LDR	34	R->A
...01	LIDP 25900	16,101,44	25900->DP
...04	STD	82	A->(DP)
...05	RTN	55	

Mit PEEK 25900 erhalten Sie den Inhalt des R-Registers. Diesen Inhalt (den Wert 86) hat also das R-Register, bevor irgend ein Unterprogramm aufgerufen wird. Und diesen Inhalt muß es auch haben, wenn wir das Maschinenprogramm mit RTN beenden, um ins BASIC zurückzukehren.

Die Speicher- und Ladebefehle für das P- und Q-Register funktionieren analog zu STR und LDR.

6.5 Sprungbefehle

Ist es Ihnen auch schon passiert: Sie benutzten beim Schreiben eines BASIC-Programms viele GOTOS, GOSUBs und ähnliches. Doch mit jedem weiteren GOTO wurde der Ablauf des Programms verschwommener und undurchsichtiger. Aber Sie bissen sich durch und vollendeten das Programm. Geschafft! Der große Schreck kam später, als Sie das Programm ein paar Wochen danach ändern wollten. Was ist denn das für ein Chaos? In dem Wirrwarr von GOTOS verloren Sie vollends den Durchblick. Das Programm war nicht mehr zu retten.

In der Tat ist es so, daß GOTO aus diesem Grund bei vielen Programmierern geradezu verpönt ist.

Die Sprungbefehle der Maschinensprache entsprechen in ihrer Struktur etwa dem GOTO-Befehl in BASIC. In der Maschinensprache können Sie Verzweigungen nur über Sprungbefehle realisieren. Sie sind das wichtigste Element zur Steuerung von Programmabläufen. Alle Verzweigungen und IF...THEN-Verknüpfungen werden über Sprungbefehle abgewickelt. Außer ih-

nen gibt es in der Maschinensprache keine Möglichkeit, Entscheidungen zu fällen.

Maschinenprogramme werden deshalb sehr schnell ungar in der Form von Mnemonics unübersichtlich. Damit Sie trotzdem den Überblick bewahren können und Ihnen nicht etwas Ähnliches wie das oben geschilderte passiert (wie es uns nämlich genau mit einem fast perfekten Renumberprogramm von 400 Bytes in Maschinensprache geschah, das wir inzwischen aufgegeben haben), halten Sie sich am besten an folgende Tips:

- Programmieren Sie nicht einfach drauf los.
- Sparen Sie bei der Erstellung von Flußdiagrammen und Entwürfen nicht an Bemerkungen.
- Werfen Sie diese Vorbereitungen und Entwürfe nicht weg, auch wenn das Programm schon steht.

Mit GOTO setzen Sie in BASIC den Programmverlauf an einer neuen Programmzeile fort. Ganz ähnlich arbeiten die Sprungbefehle: Die CPU fährt an der neuen Adresse mit dem Abarbeiten des Programms.

Absolute Sprungbefehle

JP, JPC, JPZ
JPNC, JPNZ

Bei den JP-Befehlen ("Jump", "Sprung") wird die absolute Systemadresse nm (0 bis 65535) des Sprungziels angegeben. In der Speicherzelle mit der Adresse nm steht der Befehl, mit dem das Maschinenprogramm nach dem Sprung weiterfahren soll.

Da diese Sprungadressen wie schon gesagt absolut, also fest sind, ist ein Programm mit absoluten Sprungbefehlen fast immer an eine bestimmte Adresse gebunden.

Die meisten Sprungbefehle sind mit einer Bedingung verknüpft: Der Zustand eines der CPU-Flags wird getestet. Je nach dem

Ausgang des Tests wird dann entweder ein Sprung durchgeführt oder nicht.

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	
JP nm	121,n,m	nm->PC	- -	GOTO nm
JPC nm	127,n,m	IF C=1 nm->PC	- -	IF C=1 GOTO nm
JPZ nm	126,n,m	IF Z=1 nm->PC	- -	IF Z=1 GOTO nm
JPNC nm	125,n,m	IF C=0 nm->PC	- -	IF C=0 GOTO nm
JPNZ nm	124,n,m	IF Z=0 nm->PC	- -	IF Z=0 GOTO nm

Anmerkung: n = Highbyte der Zieladresse

m = Lowbyte der Zieladresse

Ist eine Sprungbedingung nicht erfüllt, führt der Rechner den Sprungbefehl nicht aus, sondern fährt direkt mit dem nächsten Befehl fort. Das nächste Beispiel zeigt, wie man mit JP-Befehlen Endlosschleifen bildet: Nach der Tastaturabfrage springt der Rechner sofort wieder an den Programmanfang (Adresse ...00).

Probieren Sie dieses Programm lieber nicht aus, denn einmal gestartet, kann es nur über ALL RESET (zusammen mit einer beliebigen Taste des Tastenfelds, z.B. <C-CE>) verlassen werden:

Adresse	Mnemonic	Codes	Kommentar
...00	CAL 5208 ¹⁾	244,88	Tastaturabfrage
...02	JP ...00	121,n,m	Sprung an Programmanfang

Beim nächsten Beispiel wird eine Addition durchgeführt. Ein bedingter Sprungbefehl sorgt dafür, daß sich kein Fehler einschleichen kann:

1) PC 126X: CAL 4414 (241,62), PC 1350: CAL 4618 (242,10)

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 25900	16,101,44	25900->DP
...03	LDI	87	(DP)->A
...04	ADIA 100	116,100	A+100->A
...06	JPC 49321 ¹⁾	121,192,169	C=1: BEEP und zurück ins BASIC
...09	STD	82	A->(DP)
...10	RTN	55	

POKEn Sie eine Zahl in die Speicherzelle 25900 und starten Sie das Programm. Zu Ihrer Zahl wird 100 addiert. Ist das Ergebnis größer als 255, springt der Rechner in die BEEP-Routine und von dort direkt ins BASIC zurück. Andernfalls wird das Ergebnis in die Speicherzelle 25900 zurückgespeichert. Verändern Sie dieses Programm, indem Sie andere bedingte Sprungbefehle, z.B. JPNC oder JPZ verwenden. Wann reagiert der Rechner bei diesen Befehlen mit einem BEEP?

Relative Sprungbefehle

*JRP, JRM, JRCP
JRCM, JRZP, JRZM
JRNCP, JRNCM,
JRNZP, JRNZM*

Die JR-Befehle ('Jump relative', 'relativer Sprung') unterscheiden sich von den JP-Befehlen eigentlich nur dadurch, daß nicht direkt eine Adresse als Sprungziel angegeben wird, sondern um wieviele Bytes vor- oder zurückgesprungen werden soll. Die Zählung hat das Byte n, das dem JR-Befehlsbyte folgt, als Nullpunkt. Von dort aus kann bis 255 vorwärts oder rückwärts gesprungen werden. Ein Sprung vorwärts um 1 bedeutet also, daß der Rechner gerade auf die nächste Programmzeile springt.

1) PC 1421: 55539 (216,243), PC 126X: 48908 (191,12)
PC 1350: 49430 (193,22)

Da also die relativen Sprungbefehle nur festlegen, an welche Stelle innerhalb des Programms gesprungen wird, ist das Programm nicht an eine bestimmte Adresse gebunden. Im Gegenteil: Es kann frei im externen RAM verschoben werden; es ist relokativ. Sie können relative Sprungbefehle für Sprünge bis zu 255 Bytes vor oder zurück verwenden. Für Sprünge, die über 255 Bytes hinausgehen, bleibt Ihnen jedoch nichts anderes übrig, als absolute Sprungbefehle zu verwenden.

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	
JRP n	44,n	PC+1+n->PC	- -	GOTO PC+1+n
JRM n	45,n	PC+1-n->PC	- -	GOTO PC+1-n
JRCP n	58,n	IF C=1 PC+1+n->PC	- -	IF C=1 GOTO PC+1+n
JRCM n	59,n	IF C=1 PC+1-n->PC	- -	IF C=1 GOTO PC+1+n
JRNCP n	42,n	IF C=0 PC+1+n->PC	- -	IF C=0 GOTO PC+1+n
JRNCM n	43,n	IF C=0 PC+1-n->PC	- -	IF C=0 GOTO PC+1-n
JRZP n	56,n	IF Z=1 PC+1+n->PC	- -	IF Z=1 GOTO PC+1+n
JRZM n	57,n	IF Z=1 PC+1-n->PC	- -	IF Z=1 GOTO PC+1-n
JRNZP n	40,n	IF Z=0 PC+1+n->PC	- -	IF Z=0 GOTO PC+1+n
JRNZM n	41,n	IF Z=0 PC+1-n->PC	- -	IF Z=0 GOTO PC+1-n

Insgesamt gibt es 10 relative und 5 absolute Sprungbefehle. Sie kommen in den Programmen sehr häufig vor. Darum lohnt es sich auch, ihre Funktion auswendig zu wissen. Mit den Mnemonics geht das spielend:

'JP' bzw. 'JR' sagt aus, um welche Art von Sprungbefehlen es sich handelt. Das nächste Zeichen steht für die Sprungbedingung: Bei 'C' lautet die Sprungbedingung 'Carry' (IF C=1). Bei 'NC' lautet sie 'No Carry' (IF C=0). Entsprechendes gilt für das Z-Flag. Bei den JR-Befehlen gibt schließlich das letzte Zeichen Auskunft darüber, ob es sich um einen Sprung vorwärts ('P'=Plus) oder rückwärts ('M'=Minus) handelt.

Die Funktion eines Sprungbefehls lässt sich also einfach aus dem betreffenden Mnemonic ablesen. Bei JRNZM z.B. lesen Sie einfach 'Jump Relative If No Zero Minus' oder zu deutsch: Relativer Sprungbefehl um n Bytes zurück. Bedingung: Z=0. Und das alles, ohne auch nur einen Blick auf die Tabelle zu werfen!

Zwei Beispiele:

Art des Sprungbefehls: 'JR' = relativ	Sprungbedingung: 'No Zero' IF Z = 0	Sprungrichtung: 'M' = Minus
--	---	--------------------------------

JRNZM n

Art des Sprungbefehls: 'JP' = absolut	Sprungbedingung: 'Carry' IF C = 1	Sprungrichtung: absolute Adresse
--	---	--

JPC nm

Eine Tabelle mit den Mnemonics aller Befehle und weiteren Gedankenstützen finden Sie auch im Anhang B.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	JRP 3	44,3	PC+1+3->PC. Springe zu ...04
...02	CAL 5208 ¹⁾	244,88	Tastaturabfrage
...04	RTN	55	

 Der Sprungbefehl unmittelbar vor der Tastaturabfrage in Zeile ...02 sorgt dafür, daß diese einfach übersprungen wird: Der Rechner springt somit direkt zu RTN.

Beim nächsten Beispiel werden zwei Zahlen addiert:

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 25900	16,101,44	25900->DP
...03	LDD	87	(DP)->A
...04	LIDL 45	17,45	25901->DP
...06	LP 3	131	3->P. P zeigt auf B
...07	MVMD	85	(DP)->B
...08	ADM	68	(P)+A->(P). Da (P)=B: A+B->B
...09	JRNZP 3	40,3	IF Z=0 GOTO PC+1+3
...11	CAL 5208 ¹⁾	244,88	Tastaturabfrage
...13	JRNCP 3	42,3	IF C=0 GOTO PC+1+3
...15	CAL 5208 ¹⁾	244,88	Tastaturabfrage
...17	RTN	55	

 Zwei Zahlen, die in den Speicherzellen 25900 und 25901 stehen, werden zusammengezählt. Ist das Ergebnis 256 bzw. 0, wird die Tastaturabfrage zweimal aufgerufen (nach dem ersten Tastendruck bleibt der Displayinhalt erhalten). Ist das Resultat größer

1) PC 126X: CAL 4414 (241,62), PC 1350: CAL 4618 (242,10)

als 256, wird die Tastatur nur einmal abgefragt und bei einem Ergebnis kleiner als 256 kehrt der Rechner sofort ins BASIC zurück.

Probieren Sie das Programm aus, indem Sie die Addition mit verschiedenen Zahlenpaaren durchführen.

Wieviele Bytes vor oder zurück?

Die größte Mühe bei den relativen Sprungbefehlen haben die meisten Einsteiger bei der Anzahl Bytes, die vor- oder zurückgesprungen wird. Wie berechnet man das Argument eines relativen Sprungbefehls? Mit dem folgenden Demonstrationsbeispiel zeigen wir Ihnen, wie man diese Anzahl Bytes am einfachsten berechnet:

Adresse	Mnemonic	Codes	Kommentar
...00	JRP 5	44,5	Springe zu JRM 5 vor (Zeile ...06)
...02	LIA 100	2,100	100->A
...04	LIB 50	3,50	50->B
...06	JRM 5	45,5	Springe zu LIA 100 zurück (Zeile ...02)

oder anders dargestellt:

Adresse	00	01	02	03	04	05	06	07
Mnemonics	JRP	5	LIA	100	LIB	50	JRM	5
JRP 5	0	1	2	3	4	5		
JRM 5		5	4	3	2	1	0	

Wie man auf der zweiten Darstellung ganz deutlich sieht, liegt der Nullpunkt der Zählung immer auf dem Argument n. Eine allgemeine Methode, mit der man n berechnen kann, sieht darum so aus:

PC: JRP 5
LIA 100 NM:

... ..

... ..

... ..

NM: JRM 5 PC:

pos. Sprungbefehl:

$n = NM - PC + 1$

neg. Sprungbefehl:

$n = PC + 1 - NM$

PC: Adresse des Sprungbefehls

NM: Adresse des Sprungziels

n: Anzahl Bytes

Oder in Worten: Das Argument n ist immer so groß wie die Differenz zwischen der Adresse des Sprungbefehls+1 und der Adresse des Sprungziels.

Absolute oder relative Sprungbefehle

Welche Art von Sprungbefehlen soll man nun in welcher Situation verwenden? Das hängt stark von der jeweiligen Situation ab. Die folgende Aufstellung vermittelt einen groben Überblick:

Absolute Sprungbefehle:

- feste Adressen, z.B. Programmanfang, Unterprogramme
- Sprünge größer als 255 Bytes

Relative Sprungbefehle:

- Kleinere Sprünge vor oder zurück
- einfache Schleifen

Im einzelnen Fall ist dann meistens die Größe des Sprungs ausschlaggebend. Für kleinere Sprünge, die nur über relativ wenige Bytes gehen, nimmt man fast nur relative Sprungbefehle. Nicht nur weil Sie einfacher sind, sondern sie sind auch kürzer: Im Gegensatz zu den absoluten Sprungbefehlen brauchen sie nur 2 Bytes.

Es liegt auf der Hand, daß für relokatile Programme nur relative Sprungbefehle verwendet werden sollten.

Dagegen werden bei größeren Sprüngen die absoluten Sprungbefehle oft bevorzugt. Sie sind bequemer zu handhaben, da man nichts berechnen muß. Außerdem sind sie nicht auf eine Sprunglänge von 256 Bytes beschränkt.

6.6 Vergleichs- und Bittestbefehle

Vergleich von Speicherzellen

CPIA, CPIM, CPMA

Mit diesen Vergleichsbefehlen können Sie Speicherzellen miteinander vergleichen. Wie bei der Addition und Subtraktion zeigen auch hier die CPU-Flags das Ergebnis eines Vergleichs an.

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	
CPIA n	103,n	A-n	* *	-
CPIM n	99,n	(P)-n	* *	-
CPMA	199	(P)-A	* *	-

Sie können sich sicher denken, warum alle Vergleichsbefehle mit 'CP' beginnen. 'CP' kommt von 'Compare', 'Vergleiche'. Bei Vergleichen führt die CPU einfach eine Subtraktion durch und setzt die Flags dem Ergebnis entsprechend. Das Ergebnis der Subtraktion wird jedoch nirgends gespeichert; deshalb bleibt auch das verglichene Register unverändert.

Werden zwei Werte verglichen, z.B. A mit dem Wert n (CPIA n), signalisiert das Zero-Flag, ob Gleichheit vorliegt ($Z=1$) oder nicht ($Z=0$). Das Carry-Flag dagegen zeigt an, ob das verglichene Register kleiner ($C=1$) oder größer oder gleich ($C=0$) als der Vergleichswert ist. Das Carry-Flag ist aber gelöscht, wenn beide Vergleichswerte identisch sind. Die folgende Situation aus der Praxis zeigt, wie das konkret aussieht:

CPIA 10

liefert die folgenden Ergebnisse:

- | | |
|--------|----------|
| A < 10 | Z=0, C=1 |
| A = 10 | Z=1, C=0 |
| A > 10 | Z=0, C=0 |

IF-THEN-Verknüpfungen

Um den Zustand der Flags für einen bestimmten Fall zu ermitteln, führen Sie am besten in Gedanken eine Subtraktion durch und stellen sich vor, wie die Flags dabei gesetzt werden. Sie können aber auch die folgende allgemeine Tabelle zu Hilfe nehmen:

CPIA n	CPIM n	CPMA	Flags
$A < n$	$(P) < n$	$(P) < A$	$C=1, Z=0$
$A = n$	$(P) = n$	$(P) = A$	$C=0, Z=1$
$A > n$	$(P) > n$	$(P) > A$	$C=0, Z=0$
$A \neq n$	$(P) \neq n$	$(P) \neq A$	$Z=0$

Um einen Vergleich auszuwerten, verwendet man die bedingten Sprungbefehle. Vergleichsbefehle und bedingte Sprungbefehle können zusammen einfache IF-THEN-Verknüpfungen bilden:

Beispiel:

CPIA 10

Befehlsfolge	BASIC-Analogien
JPC 15120	IF A < 10 GOTO 15120
JPNC 15060	IF A >= 10 GOTO 15060
JPNZ 15070	IF A <> 10 GOTO 15070
JPZ 15140	IF A = 10 GOTO 15140

Will man sicher sein, daß A echt grösser (und nicht gleich) 10 ist, vergleicht man A einfach mit 11 oder testet nicht nur das Carry-, sondern auch das Zero-Flag.

Eine praktische Anwendung ist das nächste Beispiel. Wieder einmal arbeiten wir mit der Tastaturabfrage. Sie soll diesmal so eingesetzt werden, daß nur Buchstaben und BASIC-Befehle akzeptiert werden, deren Code mindestens 65 beträgt (ASC "A" = 65). Bei der Eingabe eines Zeichens, dessen Code unter 65 liegt, z.B. einer Zahl, soll der Rechner das Programm über die BEEP-Routine abrupt verlassen. Bei der Eingabe eines Buchstabens oder eines BASIC-Befehls soll er wieder zur Tastaturabfrage zurückspringen, womit alles wieder von vorne beginnt.

Adresse	Mnemonic	Codes	Kommentar
...00	CAL 5208 ¹⁾	244,88	Tastaturabfrage. Code der gedrückten Taste -> Akku
...02	CPIA 65	103,65	Ist A ein Buchstabe?

1) PC 126X: CAL 4414 (241,62), PC 1350: CAL 4618 (242,10)

...04	JPC 49321 ¹⁾	127,192,169	Nein: Sprung in BEEP-Routine und von dort direkt ins BASIC
...07	JRM 8	45,8	Ja: Sprung an Programmanfang

Ein RTN-Befehl erübrigts sich hier, da die Rückkehr ins BASIC über die BEEP-Routine erfolgt. Machen Sie Tastaturabfragen mit eigenen Kriterien! Die Abfrage könnte z.B. nur BASIC-Befehle annehmen, bei Zahlen einen BEEP von sich geben und beim Drücken der '\$'-Taste (Code 36) ins BASIC zurückkehren. Weitere Tastencodes finden Sie in Kapitel 1.9.

Bittest-Befehle

TSIA, TSIM

TSID

Mit diesen Befehlen können Sie den Zustand eines einzelnen Bits testen. Die CPU führt dabei einfach eine AND-Verknüpfung durch. Wie bei allen Vergleichsbefehlen wird das Ergebnis nirgends gespeichert, sondern nur die Flags (hier nur das Zero-Flag) werden entsprechend gesetzt. Besitzen die zu vergleichenden Werte keine gemeinsamen Bits, so ist das Zero-Flag gesetzt, da ja das Resultat einer AND-Verknüpfung zweier Werte, die keine gemeinsamen Bits haben, 0 ergibt.

Beispiel:

16 AND 32	Resultat: 0 -> Z=1
= 2 ⁴ AND 2 ⁵	(Das Z-Flag ist immer dann gesetzt, wenn ein Resultat 0 ergibt.)

1) PC 1421: 55539 (216,243), PC 126X: 48908 (191,12)
PC 1350: 49430 (193,22)

Ist jedoch mindestens ein gemeinsames Bit bei beiden Vergleichswerten gesetzt, so ist das Zero-Flag gelöscht, da ja das Resultat einer AND-Verknüpfung zweier Werte mit mindestens einem gemeinsamen Bit nie 0 ergibt.

Beispiel:

7

16 AND 24	Resultat: 16 -> Z=0
= 24 AND ($2^4 + 2^3$)	(Das Z-Flag ist nur dann gesetzt, wenn ein Resultat 0 ergibt.)

Mnemonic	Code	Wirkung	Flags
			C Z
TSIA n	102,n	A AND n	- *
TSIM n	98,n	(P) AND n	- *
TSID n	214,n	(DP) AND n	- *

Alle Mnemonics beginnen mit TSI ('Test immediate', 'Teste unmittelbar'). Wie schon gesagt, führen die TSI-Befehle eine AND-Verknüpfung durch, speichern aber das Ergebnis nirgends ab.

Das Zero-Flag wird scheinbar etwas seltsam gesetzt:

8

Gemeinsame Bits vorhanden:	Z=0
Alle Bits unterschiedlich:	Z=1

Tip: Wie merkt man sich diese etwas komplizierte Setzung des Z-Flags am einfachsten? Die folgende Eselsbrücke hilft Ihnen vielleicht dabei:

Gibt es bei einem Vergleich gemeinsame Bits, hat die CPU keinerlei (0) Einwände vorzubringen: Z=0. Sind jedoch in den

beiden Vergleichswerten keine gemeinsamen Bits vorhanden, so hat sie durchaus Grund zu einem (1) Einwand: Z=1.

Auch bei anderen Befehlen wird das Zero-Flag auf diese Art gesetzt; so z.B. beim TEST-Befehl.

Beispiel:

Wir wollen mit unseren neuen Befehlen den Inhalt der Speicherzelle 25900 auf bestimmte Bits hin überprüfen. Ist das 4. Bit dieser Zahl gesetzt (also z.B. bei 16=24 oder auch bei 24=24+23), soll ein BEEP ertönen. Sonst soll der Rechner stillschweigend ins BASIC zurückkehren.

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 25900	16,96,61	25900->DP
...03	TSID 16	214,16	4. Bit gesetzt: Z=0
...05	JPNZ 49321 ¹⁾	124,192,169	IF Z=0 GOTO 49321 ¹⁾
...08	RTN	55	

Auf diese Art und Weise können Sie auch Systemvariablen oder die Betriebszustandsanzeigemarken (vgl. Kapitel 4.2) auswerten. Sie können z.B. in einem Programm, das Werte an den Drucker ausgeben soll, zuerst überprüfen, ob der Drucker überhaupt eingeschaltet ist.

1) PC 1421: 55539 (216,243), PC 126X: 48908 (191,12)
PC 1350: 49430 (193,22)

6.7 Schleifen

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	
LOOP n	47,n	(R)-1->(R) * * IF C=0 PC+1-n->PC	C Z	(R)=0(R)-1; IF C=0 LET PC=PC+1-n
LEAVE	216	0->(R)	C Z	(R)=0

Die LOOP-Schleife

LOOP

Schleifen nehmen bei der Entwicklung eines Programms einen großen Stellenwert ein. Sie sind überall dort anzutreffen, wo ein Programmteil wiederholt ausgeführt wird. Aus dem BASIC kennen wir die FOR...NEXT-Schleife, im anderen Programmiersprachen findet man noch mehr Schleifen, so z.B. in PASCAL die WHILE <Bedingung> DO...- und die REPEAT...UNTIL <Bedingung>-Schleife. Unsere CPU kennt nur eine einzige Schleife: LOOP.

Diese Zählschleife wiederholt einen Programmteil höchstens 256 mal. Dieser Wert, der aussagt wie oft ein Programmteil wiederholt wird, muß vor dem Aufruf dieser Zählschleife auf dem CPU-Stack abgelegt werden. Dazu laden Sie diesen Wert zuerst in den Akku und legen ihn dann mit PUSH auf dem Stack ab. Dieser Wert ist nun der Zähler '(R)'. Jedesmal, wenn die CPU nun auf LOOP stößt, wird dieser Zähler dekrementiert. Die Schleife wird erst verlassen, wenn beim Dekrementieren das Carry-Flag gesetzt wird; wenn also (R) von 0 auf 255 dekrementiert wird. Bis es soweit ist, springt der Rechner bei LOOP jedesmal n+1 Bytes zurück.

Übersicht über den LOOP-Befehl:

```
LIA n  
PUSH  
Programmteil  
...  
...  
...  
IF C=0  
    LOOP n  (R)-1->(R)  
IF C=1  
    R+1->R  
    Nächster Befehl nach LOOP  
...  
...
```

Insgesamt wird der Programmteil, den LOOP einschließt, $(R)+1$ mal durchlaufen. Das hat seinen Grund darin, daß das C-Flag erst bei der Dekrementierung einer Null gesetzt wird. Wenn wir z.B. die Zahl 5 mehrmals dekrementieren, erhalten wir folgende Zahlenfolge: 5,4,3,2,1,0,255. (R) wird also insgesamt 6 mal dekrementiert. Ist nach einer Dekrementierung das C-Flag gesetzt (nach 0-1=255), wird die Schleife verlassen. Der Stackzeiger R wird inkrementiert und der Stack hat damit wieder denselben Zustand wie vor der LOOP-Schleife.

Dazu ein Beispiel:

Wir wollen eine Zahl n mal inkrementieren. Eigentlich wird dabei ja nur die Addition Zahl+n->Zahl in Einzelschritten ausgeführt. Wir legen dazu den Faktor n in der Speicherzelle 25900, die Zahl in der Speicherzelle 25901 ab. Achten Sie darauf, daß das Resultat von Zahl+n->Zahl unter 256 liegt, da sonst das Ergebnis verfälscht wird!

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 25900	16,101,44	25900->DP
...03	LDI	87	(DP)->A. A = Faktor n
...04	LIDL 45	17,45	25901->DP
...06	LP 3	131	3->P. P zeigt auf B
...07	MVMD	85	(DP)->(P). B = Zahl
...08	DECA	67	A-1->A. Eine LOOP-Schleife wird ja (R)+1 mal durchlaufen
...09	PUSH	52	Vorbereitung LOOP: R-1->R, A -> (R)
...10	INCB	194	B+1->B
...11	LOOP 2	47,2	(R)-1->(R), IF (R)>=0: Sprung zurück zum INCB-Befehl (2 Bytes zurück)
...13	LIDL 46	17,46	25902->DP
...15	MVDM	83	(P)->(DP). Ergebnis der Inkrementierung steht in der Speicherzelle 25902
...16	RTN	55	

Mit PEEK 25902 erfahren Sie das Ergebnis!

Gegenüber anderen Schleifen, die z.B. mit bedingten Sprungbefehlen konstruiert werden, hat LOOP gewichtige Vorteile: Der Akkumulator selbst kann in der Schleife verwendet werden; zum Zählen wird kein Register verbraucht. Außerdem können mehrere LOOPS ineinander verschachtelt werden. Dadurch kann man beliebig lange Schleifen erzeugen. Wie man LOOP-Schleifen verschachtelt, sehen Sie im folgenden Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LP 22	150	22->P
...01	LIA 1	2,1	1->A
...03	EXAM	219	A<->(P). (P) initia-

			lisieren
...04	LIA 1	2,1	1->A
...06	PUSH	52	Vorbereitung 2. LOOP
...07	LIA 3	2,3	3->A
...09	PUSH	52	Vorbereitung 1. LOOP
...10	L1DP 24576 ¹⁾	16,96,0	24576 ¹⁾ ->DP. Anfang des Displayspeichers
...13	LP 22	150	22->P
...14	MVDM	83	(P)->(DP)
...15	LDM	89	(P)->A
...16	ADM	68	(P)+A->(P)
...17	CAL 5208 ²⁾	244,88	Tastaturabfrage. Display- inhalt wird sichtbar
...19	LOOP 10	47,10	1. LOOP
...21	LOOP 15	47,15	2. LOOP
...23	RTN	55	

Das Programm besteht aus zwei verschachtelten LOOP-Schleifen. Beachten Sie, daß der Rechner nach dem zweiten LOOP zu Zeile ...07 zurückspringt und die erste LOOP-Schleife damit wieder startet. Insgesamt wird die Schleife $8=(3+1)*(1+1)$ mal durchlaufen. Da wir mit dem Startwert (P)=1 (20) beginnen, wird bei jeder Multiplikation ($2*1=2$, $2*2=4$, $2*4=8$, usw.) das nächste Bit gesetzt.

Diese Multiplikation können Sie nun direkt auf dem Display mitverfolgen! Nach jedem Tastendruck verschiebt sich der Punkt an der ersten Stelle des Displays nach unten.

Vielleicht fragen Sie sich, weshalb wir gerade die interne Speicherzelle 22 verwenden. Das hat folgenden Grund: Zur Tastaturabfrage wird das ROM-Unterprogramm INKEY benutzt. Wir müssen deshalb zur Speicherung unseres Zahlenwerts (P) eine Speicherzelle verwenden, die dabei nicht verändert wird. Wie Sie im Anhang C sehen können, bleibt die Speicherzelle 22 unverändert.

1) PC 126X: 8192 (32,0), PC 1350: 28672 (112,0)

2) PC 126X: CAL 4414 (241,62), PC 1350: CAL 4618 (242,10)

Etwas fehlt uns noch: Befindet sich der Rechner erst einmal in einer LOOP-Schleife, wird diese in jedem Fall fertig abgearbeitet. Es wäre von Vorteil, die Schleife in bestimmten Situationen auch vorzeitig abbrechen zu können. Dazu gibt es einen speziellen Befehl: LEAVE.

Abbruch einer LOOP-Schleife

LEAVE

Auf deutsch bedeutet LEAVE 'Verlassen'. Genau das geschieht, wenn der Rechner in einer LOOP-Schleife auf LEAVE stösst. In den Zähler (R) wird eine Null geschrieben. Der Rechner beendet zwar die begonnene Schleife noch; sobald er jedoch auf den LOOP-Befehl stößt, wird 0 dekrementiert, das C-Flag gesetzt und die Schleife verlassen.

Soll auch der Rest der angefangenen Schleife nicht mehr ausgeführt werden, muß nach LEAVE ein Sprungbefehl stehen, der direkt zur Zeile mit dem LOOP-Befehl springt.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 255	2,255	255->A
...02	PUSH	52	Vorbereitung LOOP
...03	CAL 5208 ¹⁾	244,88	Tastaturabfrage
...05	CPIA 48	103,48	Ist A=48? (Taste '0')
...07	JRNZP 4	40,4	Nein: Springe zu ...12
...09	LEAVE	216	0->(R)
...10	JRP 5	44,5	Springe zu LOOP

1) PC 1421: 17793 (69,129), PC 126X: 25857 (101,1), PC 1350: 27697 (108,49)

...12	LIDP 17873 ¹⁾	16,69,209	17873 ¹⁾ ->DP. DP=Erstes Zeichen von Z\$
...15	STD	82	A->(DP)
...16	LOOP 14	47,14	LOOP (IF C=0 GOTO ...03)
...18	RTN	55	

Dieses Beispiel verändert so lange (höchstens 256 mal) das erste Zeichen der Textvariable Z\$, bis Sie die Taste '0' (Null) betätigen. Diese Null dient nur als Abbruchkriterium und soll keinesfalls in die Variable Z\$ aufgenommen werden. Deshalb brechen wir nicht nur die LOOP-Schleife ab, sondern wir springen auch direkt zum LOOP-Befehl, um die Speicherung des Akkumulators mit dem Inhalt '0' zu überspringen.

Initialisieren Sie vor dem Starten die Textvariable Z\$, z.B. mit Z\$="ABC". Nachdem Sie das Programm über '0' verlassen haben, steht an der ersten Stelle von Z\$ das zuletzt vor Null eingegebene Zeichen.

LEAVE und LOOP lassen sich also nicht nur für Zählschleifen einsetzen. Sogar komplizierte Schleifenstrukturen können wir ohne Probleme realisieren!

6.8 Bearbeiten ganzer Speicherteile

Das X- und Y-Register

Wie Sie wissen, führt der Rechner Operationen nur im internen RAM durch, denn dort hat die CPU direkten Zugriff auf die Register. Speicherzellen des externen RAM können Sie nur bearbeiten, indem Sie ihren Inhalt ins interne RAM laden, die gewünschte Operation durchführen und den Inhalt wieder zurückspeichern. Bei einzelnen Bytes geht das ja noch; wir haben schon öfters so gearbeitet. Aber wie sollen wir damit Aufgaben bewältigen, die nicht einige Bytes, sondern gleich Dutzende betreffen?

1) PC 126X: CAL 4414 (241,62), PC 1350: CAL 4618 (242,10)

Dazu gibt es spezielle Adressregister, X und Y. Schauen Sie auf dem Plan der CPU in Kapitel 5.1 nach. X und Y sind zwei 16-Bit-Register, die jeweils in High- und Lowbyte unterteilt sind. In ihnen kann man eine Adresse speichern (0 bis 65535) und mit dieser Adresse sehr effizient arbeiten. Es existieren sogar Befehle, die im gleichen Zuge das betreffende Adreßregister X oder Y in- oder dekrementieren und Daten austauschen.

Die Aufgabenteilung zwischen X und Y

IX, DX, IXL, DXL

IY, DY, IYS, DYS

X und Y haben getrennte Aufgabenbereiche. X ist für das Laden von Bytes in den Akku zuständig, während Y das Zurückspeichern ins externe RAM übernimmt.

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	
IX	4	X+1->X. X->DP	- -	X=X+1:DP=X
DX	5	X-1->X. X->DP	- -	X=X-1:DP=X
IXL	36	X+1->X. X->DP. (DP)->A	- -	X=X+1:DP=X: A=PEEK DP
DXL	37	X-1->X. X->DP. (DP)->A	- -	X=X-1:DP=X: A=PEEK DP
IY	6	Y+1->Y. Y->DP	- -	Y=Y+1:DP=Y
DY	7	Y-1->Y. Y->DP	- -	Y=Y-1:DP=Y
IYS	38	Y+1->Y. Y->DP. A->(DP)	- -	Y=Y+1:DP=Y: POKE DP,A
DYS	39	Y-1->Y. Y->DP. A->(DP)	- -	Y=Y-1:DP=Y: POKE DP,A

Wir wollen ein Programm schreiben, das alle Codes der Textvariablen Z\$ inkrementiert. Mit den Befehlen, die uns bis jetzt zur Verfügung stehen, sähe das für den PC 1401 etwa so aus:

LIDP 17873

LDI

INCA

STD

...

...

LIDP 17879

LDI

INCA

STD

RTN

Eine mühsame Sache! Für dieses simple Programm brauchten wir mehr als 40 Bytes. Die Bilanz sieht wesentlich besser aus, wenn wir mit dem X- und Y-Register arbeiten:

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 208 ¹⁾	2,208	208 ¹⁾ ->A. A=Lowbyte von 17872 ²⁾
...02	LP 4	132	4->P. P zeigt auf XL
...03	EXAM	219	A<->(P)
...04	LIA 69 ³⁾	2,69	69 ³⁾ ->A. A=Highbyte von 17872 ²⁾
...06	INCP	80	P+P->P. P zeigt auf XH
...07	EXAM	219	A<->(P). X ist geladen
...08	LIA 208 ¹⁾	2,208	208 ¹⁾ ->A. Dasselbe für Y
...10	INCP	80	P+P->P. P zeigt auf YL
...11	EXAM	219	A<->(P)
...12	LIA 69 ³⁾	2,69	69 ³⁾ ->A

1) PC 1421: 128 (128), PC 126X: 0 (0), PC 1350: 48 (48)

2) PC 1421: 17792, PC 126X: 25856, PC 1350: 27696

3) PC 126X: 101 (101), PC 1350: 108 (108)

...14	INCP	80	P+P->P. P zeigt auf YH
...15	EXAM	219	A<->(P). Y ist geladen
...16	LIA 6	2,6	6->A
...18	PUSH	52	Vorbereitung LOOP
...19	IXL	36	X+1->X, X->DP, (DP)->A. Code von Z\$ lesen
...20	INCA	66	A+1->A
...21	IYS	38	Y+1->Y, Y->DP, A->(DP). Code zurückschreiben
...22	LOOP 4	47,4	LOOP. (R)>=0: Springe zu ...19
...24	RTN	55	

Belegen Sie Z\$ vor dem Starten des Programms mit 6 Zeichen, z.B. Z\$="ABCDEF". Nachdem Sie das Programm aufgerufen haben, geben Sie Z\$ ein und drücken ENTER.

Auch dieses Programm schluckt noch 25 Bytes. Aber davon werden allein 16 für das Laden des X- und Y-Registers verwendet.

Man sieht sofort, daß man für dieselbe Bearbeitung einer weiteren Variable nur noch wenige zusätzliche Bytes braucht.

Sind die Adreßregister X und Y erst einmal geladen, kann man damit äußerst praktisch und platzsparend arbeiten.

Eine Besonderheit des X-Registers

Haben Sie sich schon einmal überlegt, welchen Inhalt X eigentlich hat, bevor wir es für unsere Zwecke einsetzen? Nein? Dann müssen wir diese Frage aber einmal gründlich unter die Lupe nehmen. Jedes Maschinenprogramm wird ja mit CALL aufgerufen. Die CALL-Routine bereitet den Rechner auf das Maschinenprogramm vor. Dabei wird natürlich der Ein/Ausgabepuffer ausgelesen. Wie alle BASIC-Befehle benutzt die CALL-Routine dazu das X-Register. Deshalb zeigt das X-Register beim Start eines Maschinenprogramms im Direktmodus auf das zuletzt

ausgelesene Byte des Ein/Ausgabepuffers. Das ist das letzte Byte einer Eingabe, z.B. die Zahl '1' bei 'CALL 49321'.

Diese Eigenschaft des X-Registers werden wir in einem späteren Kapitel benutzen, um eigene BASIC-Befehle zu kreieren. Auf diese Weise können wir nämlich Parameter direkt an das Maschinenprogramm übergeben, indem wir sie an den CALL-Befehl anhängen.

Im folgenden Beispiel sehen Sie, wie man Ergebnisse direkt in den Ein/Ausgabepuffer schreibt:

Adresse	Mnemonic	Codes	Kommentar
...00	LP 4	132	4->P. P zeigt auf XL
...01	LDM	89	(P)->A
...02	LP 6	134	6->P. P zeigt auf YL
...03	EXAM	219	A<->(P). YL=XL
...04	LP 5	133	5->P. P zeigt auf XH
...05	LDM	89	(P)->A
...06	LP 7	135	7->P. P zeigt auf YH
...07	EXAM	219	A<->(P). YH=XH.
...08	LIDP 25900	16,101,44	25900->DP
...11	LDI	87	(DP)->A
...12	IYS	38	Y+1->Y, Y->DP, A->(DP)
...13	RTN	55	

POKEn Sie zuerst den ASCII-Code eines bestimmten Zeichens in die Speicherzelle 25900 (z.B. 65 für 'A'). Am besten rufen Sie das Programm mit "CALL ...00." auf, also mit einem Punkt nach dem CALL-Befehl. Dies widerspricht der Syntax des PC's; er meldet sich aber erst nach der Ausführung unseres Maschinenprogramms mit einer Fehlermeldung. Schauen Sie dann mit der '<' -Taste nach: Nach dem Punkt steht das Zeichen mit dem Code, den Sie in der Speicherzelle 25900 abgelegt haben.

Das Programm geht hier folgendermaßen vor: Zunächst wird das X-Register ins Y-Register kopiert. Dann wird der ASCII-Code

des erwünschten Zeichens in den Akku geladen und über das Y-Register im Ein/Ausgabepuffer abgelegt.

6.9 Blockbefehle

Wollen Sie ganze Speicherbereiche bearbeiten, bietet unsere CPU eine Fülle von Möglichkeiten. Bereiche zwischen internem und externem RAM (und umgekehrt) oder innerhalb des internen RAM können kopiert und vertauscht werden. Sie werden staunen, wieviel man oft mit erstaunlich wenig Befehlen zustande bringen kann.

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	

Blockfüllbefehle

FILD	31	$A \rightarrow (DP) \dots (DP+I)$	- -	FOR N=DP TO DP+I: POKE DP,A:NEXT N
FILM	30	$A \rightarrow (P) \dots (P+I)$	- -	FOR N=P TO P+I: $M(N)=A:NEXT N$

Blocktransferbefehle

MVW	8	$(Q) \dots (Q+I) \rightarrow (P) \dots (P+I)$	- -	FOR N=0 TO I: $M(P+N)=M(Q+N):$ NEXT N
MVB	10	$(Q) \dots (Q+J) \rightarrow (P) \dots (P+J)$	- -	FOR N=0 TO J: $M(P+B)=M(Q+N):$ NEXT N
MVWD	24	$(DP) \dots (DP+I) \rightarrow (P) \dots (P+I)$	- -	FOR N=0 TO I: $M(P+N)=PEEK$ $(DP+N):NEXT N$
MVBD	26	$(DP) \dots (DP+J) \rightarrow (P) \dots (P+J)$	- -	FOR N=0 TO J: $M(P+N)=PEEK$ $(DP+N):NEXT N$

DATA	53	(BA)...(BA+1)->	- -	FOR N=0 TO 1:
		(P)...(P+I)		M(P+N)=PEEK
				(BA+N):NEXT N

Blockaustauschbefehle

EXW	9	(P)...(P+I)<->	- -	-
		(Q)...(Q+1)		
EXB	11	(P)...(P+J)<->	- -	-
		(Q)...(Q+J)		
EXWD	25	(P)...(P+I)<->	- -	-
		(DP)...(DP+I)		
EXBD	27	(P)...(P+J)<->	- -	-
		(DP)...(DP+J)		

Was ist ein Block?

Unter einem Block versteht man eine Anzahl Bytes, die bearbeitet werden soll. Blockbefehle sind also Befehle, die nicht nur ein Byte, sondern gerade mehrere (maximal 256) Bytes bearbeiten. Die Blocklänge wird durch die Zählregister I bzw. J bestimmt. Wie Sie in der Befehlsübersicht oben sehen können, existieren von vielen Blockbefehlen zwei Versionen. Sie unterscheiden sich nur dadurch, daß eine Version mit dem I-Register arbeitet, die andere nur das J-Register verwendet. Wir werden uns konsequent auf die Version beschränken, die mit dem Zählregister I arbeitet.

Verwenden Sie auf keinen Fall Befehle, die J als Zählregister verwenden! Hat das J-Register beim Aufrufen von ROM-Unterprogrammen oder bei der Rückkehr ins BASIC nicht den Inhalt 1, so kommt es mit großer Wahrscheinlichkeit zu einem Systemabsturz.

Die Länge eines Blocks wird also durch das I-Register bestimmt. I kann man auf zwei Arten bestimmen: Geht es um eine bestimmte Anzahl Bytes, nimmt man einfach:

$$I = \text{Anzahl Bytes} - 1$$

Ist dagegen eine Anfangs- und Endadresse (Zeiger P oder DP) vorgegeben, die einen Block umfassen, rechnet man I am besten so aus:

$$I = \text{Endadresse} - \text{Anfangsadresse}$$

Die folgende Darstellung verdeutlicht diesen Zusammenhang:

Feste Adressierung:	10	11	12	13	14	15	16	17	18	19	20
Anzahl bearbeiteter Bytes:	1	2	3	4	5	6	7	8	9	10	11
Resultierendes I:	0	1	2	3	4	5	6	7	8	9	10



Blockfüllbefehle

FILD, FILM

Mit den Blockfüllbefehlen können Bereiche des externen (FILD) oder internen RAM (FILM) mit einem konstanten Wert gefüllt werden. Das Zählregister I bestimmt, wieviele Bytes davon betroffen sind. Da diese Befehle sehr leicht verständlich sind, fangen wir gleich mit einem Beispiel an:

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 17873 ¹⁾	16,69,209	17873 ¹⁾ ->DP. (Erstes Zeichen von Z\$)
...03	LII 6	0,6	6->I. 7 Bytes sollen bearbeitet werden
...05	LIA 90	2,90	90->A. CHR\$(90)="Z"
...07	FILD	31	A->(DP)...(DP+I)
...08	RTN	55	



1) PC 1421: 17793 (69,129), PC 126X: 25857 (101,1), PC 1350: 27697 (108,49)

Initialisieren Sie zuerst die Standardvariable Z als Textvariable Z\$ ($Z\$ = ""$). Nachdem Sie dieses Programm aufgerufen haben, überprüfen Sie Z\$. Das Programm braucht stolze 9 Bytes, um Z\$ mit 7 Buchstaben zu füllen.

Sie können sich jetzt auch denken, wie der CLEAR-Befehl, der alle Variablen löscht, im Fall der Standardvariablen funktioniert: Der Speicherbereich, in dem die Standardvariablen abgelegt sind, wird mittels FILD mit Nullen überschrieben.

Auf dieselbe Art und Weise können wir auch die Speicherzellen des internen RAM manipulieren. Beim folgenden Beispiel wird zuerst das W-Register geladen. Eine Anzahl Speicherzellen bis und mit W-Register wird durch FILM mit 100 überschrieben.

Adresse	Mnemonic	Codes	Kommentar
...00	LP 11	139	11->P. P zeigt auf W
...01	LIA 50	2,50	50->A
...03	EXAM	219	A<->(P). W=50
...04	LP 3	131	3->P. P zeigt auf B
...05	LIA 100	2,100	100->A
...07	LII 8	0,8	8->I. Bearbeitung Register 3 bis 11
...09	FILM	30	A->(P)...(P+I)
...10	LP 11	139	11->P. P zeigt auf W
...11	LIDP 25900	16,101,44	25900->DP
...14	MVDM	83	(P)->(DP)
...15	RTN	55	

Lesen Sie die Speicherzelle 25900 aus. Da auch das W-Register durch den FILM-Befehl verändert wurde, hat diese Speicherzelle nun den Inhalt 100. Ändern Sie die Zeile ...09, indem Sie FILM durch LP 11 (Code: 139) oder einen anderen Befehl ohne Einfluß auf das W-Register ersetzen. Damit behält das W-Register seinen ursprünglichen Wert (50).

Veränderung der Zeiger P und DP

Bei allen Blockbefehlen werden die beteiligten Zeiger verändert! Der Datenzeiger DP bekommt den Wert DP+I, zeigt also auf das zuletzt bearbeitete Byte. Der neue Inhalt des Zeigers P heißt P+I+1. Damit zeigt P auf das Byte, das dem zuletzt bearbeiteten folgt.

Veränderung durch die Blockbefehle:

$DP+I \rightarrow DP$ $P+I+1 \rightarrow P$



Blocktransferbefehle

MVW, MVB
MVWD, MVBD

'MV' stammt von 'Move', 'Bewege'. Mit den MV-Befehlen wird ein Speicherteil in einen anderen kopiert. Als zusätzlicher, zweiter Zeiger wird neben P der Zeiger Q verwendet. Q ist ähnlich wie P für die Adressierung des internen RAM bei Blockbefehlen verantwortlich. Beim nächsten Beispiel kopieren wir zuerst mit Blocktransferbefehlen das X- ins Y-Register; dann manipulieren wir zwei Zeichen, die mit einem Komma an den CALL-Befehl angehängt wurden:

Adresse	Mnemonic	Codes	Kommentar
...00	IX	4	Komma überspringen
...01	LIQ 4	19,4	4->Q. Der Zeiger Q zeigt auf XL
...03	LP 6	134	6->P. P zeigt auf YL
...04	LII 1	0,1	1->I
...06	MVW	8	(Q)...(Q+I)-> (P)...(P+I). Y=X
...07	IXL	36	X+1->X, X->DP, (DP)->A
...08	INCA	66	A+1->A
...09	IYS	38	Y+1->Y, Y->DP, A->(DP)



...10	IXL	36	Dasselbe mit dem 2. Byte
...11	INCA	66	
...12	IYS	38	
...13	RTN	55	

Starten Sie das Programm z.B. mit 'CALL ...00,AB'. Nachdem der Rechner das Maschinenprogramm ausgeführt hat, meldet er sich mit einer Fehlermeldung. Mit der '<'-Taste schauen Sie nach, was los ist: Die ASCII-Codes der beiden angehängten Zeichen wurden inkrementiert (aus 'AB' wurde 'BC'). Bekanntlich zeigt ja das X-Register beim Aufruf eines Maschinenprogramms auf das letzte Byte der Anfangsadresse des CALL-Befehls. Durch das Inkrementieren des X-Registers überspringen wir das Trennzeichen, das Komma. Diese neue Adresse von X wird nun ins Y-Register kopiert. X und Y wirken nun so zusammen, daß die Codes für 'AB' manipuliert werden.

Genausogut können wir mit den Blockbefehlen auch Teile des externen RAM in das interne kopieren. Das folgende Beispiel kopiert die BASIC-Anfangsadresse (im Low/Highbyte-Format) aus der Systemadresse 181451)/18146 ins X- und Y-Register. Um diesen Kopiervorgang zu demonstrieren, inkrementieren wir die Zeilennummer der ersten BASIC-Programmzeile.

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 18145 ¹⁾	16,70,225	18145 ¹⁾ ->DP
...03	LP 4	132	4->P. P zeigt auf XL
...04	LII 1	0,1	1->I
...06	MVWD	24	(DP)...(DP+I)-> (P)...(P+I)
...07	LP 6	134	6->P. P zeigt auf YL
...08	LIDP 18145 ¹⁾	16,70,225	DP zurücksetzen
...11	MVWD	24	(DP)...(DP+I)-> (P)...(P+I)

1) PC 126X: 26337 (102,225), PC 1350: 28417 (111,1)

...12	IX	4	X+1->X. X zeigt auf das Highbyte der ersten BASIC-Zeilenummer
...13	IXL	36	X+1->X. X->DP. (DP)->A. A=Lowbyte der ersten Zeilenummer
...14	INCA	66	A+1->A
...15	IY	6	Y+1->Y. Y zeigt auf das Highbyte der ersten Zeilenummer
...16	IYS	38	Zurückspeichern des Lowbytes
...17	RTN	55	



Schauen Sie nach dem Aufrufen des Programms im PRO-Modus nach: Die erste Programmzeilenummer wurde inkrementiert!

Sie können diese Programmzeile wieder zurücksetzen, indem Sie INCA in Zeile ...14 durch DECA ersetzen und dann das Programm erneut aufrufen.



Auslesen des internen ROM

DATA

Vielleicht haben Sie auch schon versucht, das interne ROM (Adressen 0 bis 8191) mit PEEK auszulesen. Aber dieses Vorhaben funktioniert nicht! Bei PEEK 1 erschien nur 0, ebenso bei PEEK 2 oder PEEK 3. Aber weshalb klappt das nicht?

Das interne ROM ist ein Teil CPU selbst, deshalb kann es mit PEEK nicht ausgelesen werden. Versucht man es trotzdem, wird nur das Highbyte der gewünschten Adresse ausgegeben. Nicht einmal in der Maschinensprache kann man das interne ROM über den Datenzeiger DP lesen.

Der DATA-Befehl ('Read Data from internal ROM', 'Lies Daten aus dem internen ROM') ist in der Tat der einzige Befehl der CPU, der das interne ROM auslesen kann.

Mit DATA wird gerade ein ganzer Bereich des internen ROM ins interne RAM kopiert. Das Lowbyte der Adresse, die Sie auslesen wollen, laden Sie ins A-Register, das Highbyte ins B-Register. Die Register B und A bilden hier ein 16-Bit-Adreßregister; B und A spielen so zusammen wie z.B. XH und XL. I bestimmt die Länge des Blocks, der aus dem internen ROM kopiert wird. Im nächsten Beispiel lesen wir die ROM-Speicherzellen mit den Adressen 10 und 11 aus und kopieren sie zuerst ins interne und dann ins externe RAM.

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 10	2,10	10->A. 10=Lowbyte von 10
...02	LIB 0	3,0	0->B. 0=Highbyte von 10
...04	LII 1	0,1	1->I. 2 Bytes sollen ja ausgelesen werden
...06	LP 10	138	10->P. P zeigt auf V
...07	DATA	53	(BA)...(BA+I)-> (P)...(P+I)
...08	LP 10	138	P zurücksetzen
...09	LIDP 25900	16,101,44	25900->DP
...12	EXWD	25	(P)...(P+I)<-> (DP)...(DP+I)
...13	RTN	55	

Mit PEEK 25900 und PEEK 25901 können Sie nun auslesen, was in Wirklichkeit in den Speicherzellen 10 und 11 des internen ROM steht. Auch der ASSEMBLER benutzt natürlich den DATA-Befehl, um das interne ROM auslesen zu können.

Blockaustauschbefehle

EXW, EXB

EXWD, EXBD

'EX' kommt von 'Exchange', 'Austauschen'. Mit dieser Befehlsgruppe können Sie ganze Speicherteile austauschen.

Das nächste Beispiel zeigt, wie man Speicherzellen des internen RAM mit denen des externen austauscht. Wir füllen nämlich die ersten Zeichen des Displays mit einer schwarzen Fläche. Im Grafikmodus ist das der Code 127 (127=alle 7 Bits gesetzt, vgl. Kapitel 4.11).

Da beim Austausch der alte Displayinhalt im internen RAM gespeichert wird, kann er zurückgeholt werden. Wir können auf diese Weise ständig zwischen zwei Bildschirminhalten wechseln. Das Programm wird durch die Betätigung der roten 'C-CE'-Taste (PC 126X und PC 1350: CL-Taste) abgebrochen.

Adresse	Mnemonic	Codes	Kommentar
...00	LP 20	148	20->P
...01	LII 29	0,29	29->I
...03	LIA 127	2,127	127->A. 127 bedeutet: Alle Punkte der Matrix gesetzt
...05	FILM	30	A->(P)...(P+I)
...06	LP 20	148	20->P
...07	LIDP 24576 ¹⁾	16,96,0	24576 ¹⁾ ->DP. Anfang des Displayspeichers
...10	EXWD	25	(P)...(P+I)<-> (DP)...(DP+I)
...11	CAL 5208 ²⁾	244,88	Tastaturabfrage

1) PC 126X: 8192 (32,0), PC 1350: 28672 (112,0)

2) PC 126X: CAL 4414 (241,62), PC 1350: CAL 4618 (242,10)

...13	L11 29	0,29	I zurücksetzen; wurde durch Tastaturabfrage verändert
...15	CPIA 2	103,2	Wurde die rote C-CE-Taste gedrückt?
...17	JRNZM 12	41,12	Nein: Springe zu ...06
...19	RTN	55	Ja: Zurück ins BASIC

Durch das Verändern der Zeile ...03 können Sie auch andere Muster erzeugen. Interessant ist z.B. auch '...03 LIA 85'.

6.10 16-Bit-Addition und -Subtraktion

ADB, SBB

ADCM, SBCM

Auf den ersten Blick ist man vielfach enttäuscht, wie wenig so ein 8-Bit-Prozessor überhaupt kann. Der 'Manager' des Rechners, die CPU, scheint ein recht kümmerlicher Apparat zu sein. Bei den folgenden Befehlen werden Sie sehen, daß die Stärke der CPU in ihren vielseitigen Befehlen steckt.

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	

16-Bit-Addition und -Subtraktion

ADB	20	(P)+A->(P)	* *	$M(P)=M(P)+A:$ $(P+1)+B+C->(P+1)$ $M(P+1)=M(P+1)+B+C$
SBB	21	(P)-A->(P)	* *	$M(P)=M(P)-A$ $(P+1)-B-C->(P+1)$ $M(P+1)=M(P+1)-B-C$

Addition und Subtraktion mit Berücksichtigung des Übertrags

ADCM	196	(P)+A+C->(P)	* *	$M(P)=M(P)+A+C$
SBCM	197	(P)-A-C->(P)	* *	$M(P)=M(P)-A-C$

Mit der ersten Befehlsgruppe können wir eine direkte 16-Bit-Addition bzw. -Subtraktion mit Übertrag durchführen. Das Lowbyte der Zahl, die addiert bzw. subtrahiert wird, steht im A-Register, das Highbyte im B-Register.

Wie sie bereits wissen, liest die CALL-Routine den Ein/Ausgabepuffer mit Hilfe des X-Registers. Das X-Register zeigt also (unverändert) immer auf das zuletzt gelesene Byte. Bei einem CALL-Befehl nach dem Muster 'CALL ...00?' zeigt X auf die letzte Ziffer der Anfangsadresse. Das Fragezeichen soll beim nächsten Beispiel einen Fehler generieren und damit ermöglichen, daß wir den Inhalt des Ein/Ausgabepuffers durch Drücken der Taste '◀' wieder anschauen können.

Beim nächsten Beispiel ändern wir den Ein/Ausgabepuffer:

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 4	2,4	4->A. A = Lowbyte
...02	LIB 0	3,0	0->B. B = Highbyte
...04	LP 4	132	4->P. P zeigt auf XL
...05	SBB	21	(P)-A->(P). (P+1)-B-C-> (P+1). X=X-4
...06	LP 6	134	6->P. P zeigt auf YL
...07	LIQ 4	19,4	4->Q. Q zeigt auf XL
...09	LII 1	0,1	1->I
...11	MVW	8	(Q)...(Q+1)-> (P)...(P+1). Y=X
...12	LIA 196	2,196	196->A. Code für BEEP
...14	DYS	39	Y-1->Y. Y->DP. A->(DP). CALL durch BEEP ersetzen
...15	RTN	55	

In diesem Beispiel wird 4 vom X-Register subtrahiert. Damit zeigt das X-Register auf das zweite Byte des Ein/Ausgabepuffers. Das X-Register wird ins Y-Register kopiert und Y wird

dekrementiert. Y zeigt nun also auf das erste Byte des Ein-/Ausgabepuffers. In diesem Byte steht der Interpretercode von CALL. Nun wird CALL durch BEEP ersetzt.

Geben Sie 'CALL ...00?' ein. Der Rechner meldet sich mit ERROR 1. Mit '◄' schauen Sie nach, was los ist: Der CALL-Befehl wurde durch den Befehl BEEP ersetzt. Drücken Sie nun ENTER: BEEP ertönt.

Bei einer Uhr haben Sie zwei Möglichkeiten, die Zeit von 12'00 Uhr auf 11'00 Uhr zu verstellen. Sie können die Zeiger 1 Stunde zurück- oder 11 Stunden vordrehen. Die Wirkung ist in beiden Fällen dieselbe. Genauso auch hier: Anstatt 4 von X zu subtrahieren, können Sie auch 65532 zu X addieren.

Ändern Sie dazu folgende Programmzeilen:

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 252	2,252	252->A. 252=LB von 65532
...02	LIB 255	3,255	255->B. 255=HB von 65532
...04	LP 4	132	4->P
...05	ADB	20	(P)+A->(P). (P+1)+B+C-> (P+1). X=X+65532

Es ist klar, daß ein 8-Bit-Prozessor intern nie zwei 16-Bit-Werte auf einen Schlag addieren oder subtrahieren kann. Dazu besitzt die CPU ein ganz bestimmtes internes Programm, das die Operation in Schritte zerlegt, die die CPU ausführen kann.

Intern rechnet die CPU also mit 8-Bit-Zahlen. Bei einer 16-Bit-Addition addiert bzw. subtrahiert sie zuerst die beiden Lowbytes. Tritt dabei ein Übertrag auf, wird das Carry-Flag gesetzt. Das Carry-Flag wird nun in die zweite Berechnung, die Addition oder Subtraktion der Highbytes, miteinbezogen. Das Ergebnis ist schließlich dasselbe, wie wenn die beiden 16-Bit-Zahlen direkt addiert oder subtrahiert worden wären.

Was die CPU hier im Hintergrund erledigt, können wir in Einzelschritten nachvollziehen. Dazu verwenden wir die Additions- und Subtraktionsbefehle, die das Carry-Flag in die Rechnung miteinbeziehen:

ADCM führt eine 8-Bit-Addition durch, wobei auch das Carry-Flag addiert wird. Indem Sie mehrere einzelne ADCM-Additionen durchführen, können Sie 16-, ja sogar 32-Bit-Zahlen addieren.

Der Befehl SBCM funktioniert analog zu ADCM. Bei ihm wird das Carry-Flag in eine Subtraktion miteinbezogen.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LP 4	132	4->P. P zeigt auf XL
...01	LDM	89	(P)->A. A=XL
...02	LP 6	134	6->P. P zeigt auf YL
...03	ADM	68	(P)+A->(P). YL=YL+XL
...04	LP 5	133	5->P. P zeigt auf XH
...05	LDM	89	(P)->A. A=XH
...06	LP 7	135	7->P. P zeigt auf YH
...07	ADCM	196	(P)+A+C->(P). YH=YH+XH
...08	RTN	55	

Bei diesem Beispiel wird das X-Register zum Y-Register addiert. Auf die gleiche Art und Weise könnte man sogar 32-Bit-Zahlen addieren oder subtrahieren.

Wie bei der schriftlichen Addition fängt man auch hier hinten an und addiert zuerst einmal die Lowbytes. Als nächstes werden dann die Highbytes addiert, wobei der Übertrag, der bei der Addition der Lowbytes auftrat, hier wieder in die Addition der Highbytes miteinbezogen wird.

6.11 Das BCD-System in der Anwendung

Bis jetzt haben wir nur mit Dualzahlen gearbeitet. Der höchste Zahlenwert lag bei 255, allenfalls bei 65535 mit dem High/Lowbyte-System. Unsere CPU kann aber auch dezimale Additionen durchführen. Dabei verwendet sie die BCD-Darstellung ("Binary Coded Decimal", "Dual codierte Dezimaldarstellung"). Wie Sie schon aus dem Kapitel 2.2 wissen, werden dezimale Ziffern separat in Halbbytes dual codiert:

Ziffer (dezimal)	Halbbyte (binär)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Auf diese Weise werden in einem Byte Zahlen von 0 bis 99 dargestellt. Wir wollen dieses Zahlensystem am Beispiel von 58 betrachten:

Dezimal:	5	8	58
Binär:	0101	1000	01011000

*Rechnen mit BCD-Zahlen***ADN, ADW****SBN, SBW**

Der PC behandelt Dezimalzahlen bei Berechnungen wie Dualzahlen. Die Flags, die Addition und Subtraktion zweier Werte funktionieren analog. Nur faßt ein Byte im BCD-System eben maximal hexadezimal 99 (dezimal also 153) und nicht mehr 255. Das Carry-Flag wird also schon gesetzt, wenn ein Register mit dem Inhalt 153 inkrementiert wird.

Mnemonic	Code	Wirkung	Flags
			C Z
BCD-Addition			
ADN	12	(P)+A->(P)... * *	
		(P-I)+C->(P-I)	
ADW	14	(P)+(Q)->(P)... * *	
		(P-I)+(Q-I)+C->	
		(P-I)	
BCD-Subtraktion			
SBN	13	(P)-A->(P) * *	
		(P-I)-C->(P-I)	
SBW	15	(P)-(Q)->(P) * *	
		(P-I)-(Q-I)-C->	
		(P-I)	

Sie sehen: Bei der BCD-Addition oder -Subtraktion erfolgt der Transport des Übertrags automatisch.

Für Beispiele eignen sich die Standardvariablen natürlich bestens, da die Zahlen hier ja im BCD-System abgelegt werden. Folgende Tabelle zeigt, wie diese Speicherung aussieht:

Byte Nr.	Bedeutung	Bemerkungen
1	Exponent: Vorzeichen ¹⁾ Stelle 1	
2	Exponent: Stelle 2	
3	Mantisse: Vorzeichen	0:positiv 8:neg.
3	Mantisse: Stelle 1+2	BCD-Format
4	Mantisse: Stelle 3+4	BCD-Format
5	Mantisse: Stelle 5+6	BCD-Format
6	Mantisse: Stelle 7+8	BCD-Format
7	Mantisse: Stelle 9+10	BCD-Format
8	nicht benutzt, immer 0	

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 17874 ²⁾	16,69,210	17874 ²⁾ ->DP. 3. Byte von Z
...03	LP 4	132	4->P
...04	LII 4	0,4	4->I
...06	MVWD	25	(DP)...(DP+I)-> (P)...(P+I)
...07	LP 8	136	8->P. P zeigt auf das letzte kopierte Byte
...08	LIA 17	2,17	17->A. &11=17
...10	ADN	12	(P)+A->(P)...(P-I)+C-> (P-I)
...11	LP 4	132	4->P. P zurücksetzen
...12	LIDP 17874 ²⁾	16,69,210	17874 ²⁾ ->DP. DP zurücksetzen

-
- 1) Falls kein Exponent: Byte Nr. 1 = 0 Byte Nr. 2 = Dezimalpunkt der Mantisse und Vorzeichen der Mantisse
 - 2) PC 1421: 17794 (69,130), PC 126X: 25858 (101,2), PC 1350: 27698 (108,50)

...15	EXWD	25	(DP)...(DP+1)<-> (P)...(P+1)
...16	RTN	55	

Dieses Beispiel addiert 11 zur Standardvariablen Z. Für die Zahl Z=9876543270 z.B. resultiert Z=9876543281. Rufen Sie das Maschinenprogramm mehrere Male auf. Sie werden sehen, daß der Übertrag der zweitletzten Ziffer auch auf die drittletzte übertragen wird. Damit das Beispiel funktioniert, muß die Standardvariable Z ganz gefüllt (alle 10 Stellen belegt) sein!

Probieren Sie das Programm auch für den Wert Z=1999999999 aus. Hier sehen Sie besonders deutlich, wie der Übertrag jeweils an das vorhergehende Byte weitergegeben wird.

Was geschieht wohl, wenn Sie ADN (Zeile ...10) durch SBN ersetzen?

Wichtig: Standardvariablen, bei denen nicht alle Stellen belegt sind, können nicht für solche BCD-Additionen und -Subtraktionen verwendet werden.

Rechnen mit Standardvariablen

Mit Hilfe des BCD-Systems können wir Standardvariablen direkt zusammenzählen. Auch hier müssen beide Variablen ganz belegt sein, da sonst falsche Ergebnisse resultieren.

Das Rechnen mit Standardvariablen ist nicht besonders schwierig. Darum wollen wir gleich mit einem Beispiel anfangen: Wir nehmen dazu die Rechnung Z=Z+Y.

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 17874 ¹⁾	16,69,210	17874 ¹⁾ ->DP. 3. Byte von Z
...03	LP 4	134	4->P
...04	LII 4	0,4	4->I
...06	MVWD	25	(DP)...(DP+I)-> (P)...(P+I)
...07	LIDP 17882 ²⁾	16,69,218	17882 ²⁾ ->DP. 3. Byte von Y
...10	LP 10	138	10->P
...11	MVWD	25	(DP)...(DP+I)-> (P)...(P+I)
...12	LP 8	136	8->P. P zeigt auf das letzte von Z kopierte Byte
...13	LIQ 14	19,14	14->Q. Q zeigt auf das letzte von Y kopierte Byte
...15	ADW	14	(P)+(Q)->(P)... (P-I)+(Q-I)+C->(P-I)
...16	LIDP 17874 ¹⁾	16,69,210	17874 ¹⁾ ->DP. 3. Byte von Z
...19	LP 4	132	4->P. P zeigt auf das erste kopierte Byte von Z
...20	EXWD	25	(DP)...(DP+I)<-> (P)...(P+I)
...21	RTN	55	

Ergebnis: Z=Z+Y

Füllen Sie dazu zuerst beide Variablen ganz (alle 10 Stellen); z.B.
 Z=1111111111 und Y=1888888889. Das Programm macht folgendes:

1) PC 1421: 17794 (69,130), PC 126X: 25858 (101,2), PC 1350: 27698 (108,50)

2) PC 1421: 17802 (69,138), PC 126X: 25866 (101,10), PC 1350: 27706 (108,58)

$$\begin{array}{r} 111111111 \\ + \underline{1888888889} \\ = \underline{3000000000} \end{array}$$

Nun verstehen Sie auch, warum alle Stellen der Variablen besetzt sein müssen: Zum Beispiel für den Fall Z=1 und Y=300000000 kommt folgendes heraus:

$$\begin{array}{r} 1 \\ + \underline{3000000000} \\ = \underline{4000000000} \end{array}$$



Der Subtraktionsbefehl SBW funktioniert analog zu ADW.

6.12 Logische Befehle

Mit den logischen Befehlen können Sie einzelne Bits setzen, löschen oder aus einem Byte herausfiltern. Ja, Sie können damit sogar relativ komplizierte Entscheidungen fällen.

Alle Befehle verändern das Zero-Flag entsprechend dem Resultat der Operation. Das Carry-Flag wird bei keiner Operation beeinflußt.

Wie die logischen Operationen AND und OR funktionieren, wurde schon in Kapitel 1.6 ausführlich behandelt; sie sollen hier nicht nochmals erwähnt werden.



*AND-Befehle**ANIA, ANIM**ANMA, ANID*

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	
ANIA n	100,n	A AND n->A	- *	A=A AND n
ANIM n	96,n	(P) AND n->(P)	- *	M(P)=M(P) AND n
ANMA	70	(P) AND A->(P)	- *	M(P)=M(P) AND A
ANID n	212,n	(DP) AND n->(DP)	- *	POKE DP, PEEK DP AND n

Die AND-Befehle ('AND immediate', 'Unmittelbare AND-Verknüpfung) sind besonders wichtig, wenn man gezielt einzelne Bits löschen will. Unsere Befehle erlauben sogar AND-Operationen direkt mit einer externen Speicherzelle.

Die Wirkung eines AND-Befehls wird am besten bei dualer Darstellung klar. Sie wollen z.B. das dritte Bit des Akkus (nach der üblichen Zählung 0. bis 7. Byte) löschen. Dazu geben Sie ein:

ANIA 247	76543210		
247 dual:	11110111	247	
Akku z.B. 104	AND 01101000	AND 104	
Ergebnis:	= 01100000	= 96	

Bei der AND-Operation behalten also alle Bits, die mit '1' logisch verknüpft werden, ihren alten Wert. Dagegen werden die Bits gelöscht, die logisch mit '0' verknüpft werden, gelöscht. Wenn Sie Werte aus dem externen RAM übernehmen wollen, stellt sich oft ein Problem: Die Umwandlung einer ASCII-Ziffer

in einen (einstelligen) BCD-Wert. Wollen Sie z.B. von einer Textvariablen Ziffern übernehmen, müssen Sie zuerst von jeder Ziffer 48 subtrahieren. Mit den AND-Befehlen wird das ganze viel einfacher:

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 17873 ¹⁾	16,69,209	17873 ¹⁾ ->DP. 1. Byte von Z\$
...03	ANID 207	212,207	4. und 5. Bit (Zählung ab 0) werden gelöscht
...04	RTN	55	

Legen Sie zuerst eine Ziffer in Z\$ ab, z.B. Z\$="5". Nach dem Aufruf des Maschinenprogramms schauen Sie mit PEEK 17873¹⁾ nach, was das Programm bewirkt hat. Statt dem ASCII-Code für '5', nämlich 53, steht in dieser Speicherzelle der Code 5.

Beim nächsten Beispiel manipulieren wir die ersten Bytes des Displays:

Adresse	Mnemonic	Codes	Kommentar
...00	LP 6	134	6->P. P zeigt auf YL
...01	LIA 255	2,255	255->A
...03	EXAM	219	A<->(P)
...04	LP 7	135	7->P. P zeigt auf YH
...05	LIA 95 ²⁾	2,95	95 ²⁾ ->A
...07	EXAM	219	A<->(P)
...08	LIA 28	2,28	28->A
...10	PUSH	52	Vorbereitung LOOP
...11	IY	6	Y+1->Y. Y->DP
...12	ANID 126	212,126	A AND n->A

1) PC 1421: 17793 (69,129), PC 126X: 25857 (101,1), PC 1350: 27697 (108,49)

2) PC 126X: LIA 31 (2,31), PC 1350: LIA 111 (2,111)

...14	LOOP 4	47,4	LOOP. (R)>=0: Springe zu ...11
...16	CAL 5208 ¹⁾	244,88	Tastatureabfrage. Die Veränderung wird angezeigt
...18	RTN	55	

Durch ANID 126 in Zeile ...12 wird die oberste Punktereihe des Displays gelöscht.

OR-Befehle

*ORIA, ORIM
ORMA, ORID*

Mit AND löschen Sie einzelne Bits innerhalb eines Bytes. Der restliche Inhalt des Bytes wird dabei nicht beeinflußt. Die OR-Befehle machen genau das Gegenteil: Bestimmte Bits werden gesetzt, ohne die restlichen zu beeinflussen.

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	
ORIA n	101,n	A OR n->A	* -	A=A OR n
ORIM n	97,n	(P) OR n->(P)	* -	M(P)=M(P) OR n
ORMA	71,n	(P) OR n->(P)	* -	M(P)=M(P) OR A
ORID n	213,n	(DP) OR n->(DP)	* -	POKE DP, PEEK DP OR n

Die Wirkung eines OR-Befehls erkennt man am besten bei duality Darstellung. Sie wollen z.B. alle Bits des Akkus vom 0. bis zum 4. Bit setzen, ohne das die restlichen Bits (5 bis 7) verändert werden. Für den Akku nehmen wir einen beliebigen Zahlenwert, z.B. 104:

1) PC 126X: CAL 4414 (241,62), PC 1350: CAL 4618 (242,10)

ORIA 31	76543210		
31 dual:	00011111	31	
Akku, z.B. 104	OR	01101000	OR 104
Ergebnis:	*	01111111	= 127

Bei der OR-Operation werden alle Bits gesetzt, die in einem der beiden Bytes gesetzt sind. Aus diesem Grund setzt eine OR-Verknüpfung mit 255 alle Bits, während eine OR-Verknüpfung mit 0 keinerlei Auswirkungen zeigt.

Eine (einstellige) BCD-Zahl können wir deshalb leicht in eine ASCII-Ziffer überführen. Initialisieren Sie zuerst Z\$ mit Z\$=" ". Dann POKEn Sie einen Wert (0 bis 9) in die Speicherzelle 17873¹⁾.

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 17873 ¹⁾	16,69,209	17873 ¹⁾ -> DP. 1. Byte von Z\$
...03	ORID 48	213,48	4. und 5. Bit werden gesetzt
...04	RTN	55	

Um ORID auch auf dem Display zu demonstrieren, ersetzen Sie beim letzten Programmbeispiel (blättern Sie etwa zwei Seiten zurück) einfach ANID 126 (Zeile ...12) durch z.B. ORID 85.

1) PC 1421: 17793 (69,129), PC 126X: 25857 (101,1), PC 1350: 27697 (108,49)

6.13 Bit-Verschiebebefehle

SR, SL

SRW, SLW

SWP

Mit diesen Befehlen können Sie den Inhalt des Akkumulators um 1 Bit nach links oder rechts verschieben, ganze Speicherbereiche des internen RAM um ein halbes Byte verschieben oder die beiden Halbbytes des Akkus vertauschen. Wir werden Ihnen auch zeigen, wie man über die Bit-Verschiebung sogar multiplizieren kann.

Mnemonic	Code	Wirkung	Flags
			C Z
SR	210	C->A7...A0->C	* -
SL	90	C->A0...A7->C	* -
SWP	88	A0...A3<-> A4...A7	- -
SRW	28	(P)...(P+1) 4 Bit SR (SR=Shift right)	- -
SLW	29	(P)...(P-1) 4 Bit SL (SL=Shift left)	- -

Die Befehle SR und SL

SR, SL

Mit den Befehlen SR ('Shift right', 'Verschiebe nach rechts') und SL ('Shift left', 'Verschiebe nach links') wird von der einen Seite des Bytes der alte Inhalt des Carry-Flags hereingeschoben, während das aus dem Akkumulator hinausgeschobene Bit ins Carry-Flag kommt.

Mit SL und SR kann man multiplizieren bzw. dividieren. Setzt man vorher das C-Flag auf 0, wirkt SL wie eine Multiplikation mit 2, SR wie eine Division durch 2. Dabei wird allerdings nur der ganzzahlige Wert in A geliefert.

Bei ungeraden Zahlen ist das 0. Bit gesetzt, bei geraden Zahlen ist dieses Bit nicht gesetzt. Um herauszufinden, ob der Inhalt des Akku gerade oder ungerade ist, überprüfen wir nach einem SR-Befehl einfach den Zustand des C-Flags. Beim SR-Befehl wird ja das erste Bit des Akkus ins Carry-Flag geschoben. War die Zahl also ungerade, ist das C-Flag gesetzt.

Wir können diese Tatsache ausnutzen, um Multiplikationen mit kleinen konstanten Werten zu programmieren.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP	25900	16,101,44 25900->DP
...03	LP 3	131	3->P. P zeigt auf B
...04	MVMD	85	(DP)->(P)
...05	LDI	87	(DP)->A
...06	RC	209	0->C. C muss zuvor 0 sein
...07	SL	90	C->A0...A7->C. A=2*A
...08	RC	209	0->C
...09	SL	90	C->A0...A7->C. A=2*A
...10	ADM	68	(P)+A->(P)
...11	MVDM	83	(P)->(DP). B zurück- speichern
...12	RTN	55	

Dieses Beispiel lädt den Inhalt der Speicherzelle 25900 ins B- und A-Register. Der Akku wird daraufhin 2 mal mit 2 multipliziert und dann zu B addiert. So bewirken wir eine Multiplikation mit 5.

Mulitplikationen, die über die Bit-Verschiebebefehle abgewickelt werden, haben den Vorteil, unübertroffen schnell zu sein. Wird aber der Inhalt der Speicherzelle 25900 zu groß (grösser als 255), liefert die Mulitplikation nur Unsinn.

Durch das Vertauschen aller SL-Befehle durch SR und ADM durch SBM lässt sich dieses Programm leicht zu einem Divisionsprogramm verwandeln.

Der Akku und der SWP-Befehl

SWP

Der Befehl SWP ('Swap', 'Vertausche') vertauscht die beiden Hälften des Akkumulators, die Bits 0 bis 3 mit den Bits 4 bis 7. Wir wollen dies gleich anhand eines Beispiels mit einem Inhalt des Akkumulators von 190 zeigen:

Akku = 190 (binär)	Inhalt des Akku nach SWP: 235
10111110	11101011

Der Befehl SWP kann man z.B. auch verwenden, um einen BCD-Wert umzukehren:

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 17874 ¹⁾	16,69,210	17874 ¹⁾ ->DP
...03	LDI	87	(DP)->A
...04	SWP	88	A0...A3<-> A4...A7
...05	STD	82	A->(DP)
...06	RTN	55	

1) PC 1421: 17794 (69,130), PC 126X: 25858 (101,2), PC 1350: 27698 (108,50)

Speichern Sie einen zweistelligen Wert, z.B. 86, in der Standardvariablen Z. Nachdem Sie das Programm aufgerufen haben, überprüfen Sie Z. Die in Z gespeicherte Zahl 86 wurde umgekehrt: Z=68!

Verschieben von Halbbytes

SRW, SLW

Das Befehlspaar SRW und SLW wird auch für die Operationen mit den Standardvariablen verwendet. SRW verschiebt innerhalb eines durch das Register I festgelegten Bereichs alle Bytes um ein Halbbyte nach rechts, SLW nach links. Das erste Halbbyte oder Nibble wird dabei durch 0 ersetzt, das letzte Halbbyte geht verloren.

Mit den drei folgenden Bytes wird eine SRW-Operation durchgeführt (dazu muß I den Inhalt 2 haben):

						SRW						
10101110	10101001	11110101	->	00001010	11101010	10011111						
1.	2.	3.	4.	5.	6.	->	neu	1.	2.	3.	4.	5.

Bei SRW erhält hier das erste Nibble den Inhalt 0, während das sechste Nibble überschrieben wird. Nun wollen wir uns dieselbe Situation anschauen, aber diesmal mit dem SLW-Befehl:

						SLW						
10101110	10101001	11110101	->	11101010	10011111	01010000						
1.	2.	3.	4.	5.	6.	->	2.	3.	4.	5.	6.	neu

Bei SLW wird das erste Nibble überschrieben, während das letzte den Inhalt 0 erhält.

Zum Schluß wollen wir diese Befehle noch anhand eines konkreten Beispiels betrachten:

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP 17875 ¹⁾	16,69,211	178751) ->DP
...03	L11 2	0,2	2->I. 3 Bytes sollen kopiert werden
...05	LP 12	140	12->P
...06	EXWD	25	(DP)...(DP+I)<->(P)...(P+I)
...07	LP 14	142	14->P. P zeigt auf das letzte kopierte Byte
...08	SRW	29	(P)...(P-1) 4 Bits SL
...09	LIDP 17875 ¹⁾	16,69,211	DP zurücksetzen; wurde durch EXWD verändert
...12	LP 12	140	12->P. P zurücksetzen
...13	EXWD	25	(DP)...(DP+I)<->(P)...(P+I). Veränderte Bytes zurückschreiben
...14	RTN	55	

Das Beispiel manipuliert Stelle 3 bis 8 der Standardvariablen Z. Geben Sie vor dem Aufruf des Maschinenprogramms ein: Z=123456789. Nach dem Maschinenprogramm lautet das Ergebnis: Z=123456790. Sie sehen: Das letzte Nibble wurde mit 0 ersetzt, das erste (das dritte) wurde überschrieben.

16-Bit-Verschiebung

Mit den Verschiebebefehlen kann man auch eine Verschiebung über mehr als 8 Bit durchführen. Über das C-Flag wird dann jeweils ein Bit von einem zum nächsten Byte transportiert. Das folgende Beispiel verschiebt den Inhalt von BA (B=HighbYTE, A=Lowbyte) nach links und bewirkt dadurch eine Multiplikation mit 2.

1) PC 1421: 17795 (69,131), PC 126X: 25859 (101,3), PC 1350: 27699 (108,51)

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP	25900	16,101,44 25900->DP
...03	LII 1	0,1	1->I
...05	LP 2	131	2->P
...06	MVWD	8	(DP)...(DP+1)-> (P)...(P+1)
...07	RC	209	0->C
...08	SL	90	C->A0...A7->C. A=2*A
...09	EXAB	218	A<->B. B in den Akku laden
...10	SL	90	C->A0...A7->C. A=2*A
...11	EXAB	218	A<->B. A zurück ins B- Register laden
...12	LIDP	25900	16,101,44 5900->DP. DP zurücksetzen
...15	LP 2	131	2->P. P zurücksetzen
...16	EXWD	25	(DP)...(DP+1)<-> (P)...(P+1)
...17	RTN	55	

Legen Sie eine Zahl (kleiner als 32600) im Low/Highbyte-Format in den Speicherzellen 25900/01 ab. Das Programm multipliziert diese Zahl bei jedem Aufruf mit 2 und speichert das Ergebnis zurück. PEEK 25900+PEEK 25901*256 liefert nun das Ergebnis.

8-Bit-Multiplikation

Wir wollen Ihnen noch zeigen, wie man mit Hilfe von Addition und Bitverschiebung zwei 8-Bit-Zahlen allgemein multipliziert. Man geht dabei eigentlich gleich vor wie bei der schriftlichen Multiplikation. Wir benutzen die Verschiebebefehle, um zu testen, ob an der jeweiligen Stelle des einen Faktors eine 1 oder 0 steht. Um dafür zu sorgen, daß die Zwischenergebnisse richtig 'untereinanderstehen', haben wir den ADB-Befehl eingesetzt. Im folgenden Beispiel wird A mit W multipliziert; das Resultat wird in Y gespeichert:

Adresse	Mnemonic	Codes	Kommentar
...00	LIB 0	3,0	0->B. Initialisierung
...02	LP 6	134	6->P. P zeigt auf YL
...03	ANIM 0	96,0	0->YL. Initialisierung
...05	LP 7	135	7->P. P zeigt auf YH
...06	ANIM 0	96,0	0->YH. Initialisierung
...08	LP 11	139	11->P. P zeigt auf W
...09	EXAM	219	A<->(P)
...10	RC	209	0->C
...11	SR	210	C->A7...A0->C
...12	EXAM	219	A<->(P)
...13	JRNCP 3	42,3	C=0: Keine Addition. Springe zu ...17
...15	LP 6	134	6->P. P zeigt auf YL
...16	ADB	20	C=1: Y=Y+BA Zwischenresultat
...17	LP 11	139	11->P. P zeigt auf W
...18	CPIM 0	99,0	Ist W bereits 0?
...20	JRZP 5	56,5	Ja: Springe zu ...26 (Programmende)
...22	LP 2	130	2->P. P zeigt auf A
...23	ADB	20	Nein: (P)+A->(P). BA verschieben, damit bei nächster Addition alles richtig 'untereinander' steht
...24	JRM 17	45,17	Springe wieder zu ...08
...26	RTN	55	

Sie können dieses Programm natürlich auch für Ihre ganz persönlichen Erfordernisse umschreiben.

6.14 Unterprogrammaufruf über Tabellen

PTC, ETC

Mit den Befehlen PTC ('Prepare Table Call', 'Bereite Unterprogrammaufruf nach Tabelle vor') und ETC ('Execute Table Call',

'Führe Unterprogrammaufruf nach Tabelle aus') können in Abhängigkeit des Akkumulatorinhalts verschiedene Unterprogramme angesprungen werden.

Mnemonic	Code	Wirkung	Flags
			C Z
PTC	122	n->H, nm->	- -
n,nm	n,nm	(R-1,R-2)	
		R-2->R	
ETC	105	For i = 1 to H	- *
n,nm	n,nm	If A=n let PC=nm	
...	...	Next i	
nm	nm	nm->PC	

P

PTC bereitet die Tabelle mit den Unterprogrammen vor:

PTC	Prepare Table Call
n,nm	n Vergleiche, Rücksprungadresse nm

Die Anzahl Vergleichspositionen in der Tabelle nach ETC wird hier mit n angegeben. Die Adresse nm wird auf dem Stack abgelegt. Alle aufgerufenen Unterprogramme kehren schließlich an diese Stelle zurück.

Nach ETC wird dann die vorbereitete Unterprogrammtabelle ausgeführt:

ETC	Execute Table Call
n,nm	IF A=n GOTO nm
n,nm	IF A=n GOTO nm
...	
...	



n,nm	IF A=n GOTO nm
nm	ELSE GOTO nm

Der Akkumulatorinhalt wird der Reihe nach mit den Werten n verglichen. Stimmt er mit einem dieser n überein, springt der Rechner zum Unterprogramm, das an der zugehörigen Adresse 'nm' steht. Im Fall 'nicht gefunden' springt der Rechner an die letzte Adresse nm. Diese Adresse kann z.B. die Adresse der BEEP-Routine sein. Eine unkorrekte Antwort auf eine Abfrage würde der Rechner somit mit einem BEEP quittieren. Für die Anzahl Vergleichspositionen zählen Sie alle Zeilen 'n,nm' zusammen. Die letzte Adresse nm gilt nicht als Vergleichsposition.

Sie haben sicher schon erkannt, worin der Nachteil von ETC und PTC besteht: Für die Vergleichspositionen können nur absolute Sprünge angegeben werden. Darum sind Programme mit ETC und PTC fest an einen bestimmten Speicherbereich gebunden. Das ist auch der Grund, warum z.B. das Musik-Programm "Soundbox" nur in einem bestimmten Adressbereich installiert werden kann.

Der ASSEMBLER bietet Ihnen für ETC und PTC sehr bequeme Eingaberoutinen. Um eine Unterprogrammtabelle vorzubereiten, geben Sie einfach 'PTC' ein. Der Rechner fragt Sie nun nach der Anzahl Vergleiche und der Rücksprungadresse. Diese Parameter speichert er in Standardvariablen. Sobald Sie dann 'ETC' eingeben, nimmt er diese Parameter und fragt Sie in einer komfortablen Eingabeschleife jeweils nach Akku und entsprechender Sprungadresse.

Wichtig: Da die Informationen von PTC in den Standardvariablen versorgt werden, müssen PTC und ETC direkt aufeinanderfolgend eingegeben werden. Der ASSEMBLER darf nicht verlassen werden, weil sonst dabei alle Variablen gelöscht werden.

Sie können Unterprogrammtabellen zwar anschauen, aber nicht verändern. Bei Änderungen muß PTC und ETC nochmals ganz neu eingegeben werden.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	PTC	122	Vorbereiten einer Unterprogrammtabelle
...01	2,5208 ¹⁾	2,20,88	2 Vergleichspositionen, Endadresse: Tastaturabfrage
...04	LIDP 25900	16,101,44	25900->DP
...07	LDD	87	(DP)->A
...08	LIDL 45	17,45	25901->DP
...10	ETC	105	Unterprogrammtabelle aktivieren
...11	0,...19	0,HB,LB	1. Vergleichsposition: IF A=0 GOTO ...19
...14	255,...22	255,HB,LB	2. Vergleichsposition: IF A=255 GOTO ...22
...17	49321 ²⁾	192,169	ELSE GOTO 49321 ²⁾

Nun folgen die aufzurufenden Unterprogramme:

...19	INCA	66	A+1->A
...20	STD	82	A->(DP)
...21	RTN	55	
...22	DECA	67	A-1->A
...23	STD	82	A->(DP)
...24	RTN	55	

POKEn Sie entweder 0 oder 255 in die Speicherzelle 25900. Einen andern Wert quittiert der Rechner nach dem Aufruf des

1) PC 126X: CAL 4414 (241,62), PC 1350: CAL 4618 (242,10)

2) PC 1421: 55539 (216,243), PC 126X: 48908 (191,12), PC 1350: 49430 (193,22)

Programms mit einem BEEP. Bei '0' wird der Akku inkrementiert (1) und in der Speicherzelle 25901 abgelegt. Bei '255' wird der Akku dekrementiert und ebenfalls in der Speicherzelle 25901 gespeichert.

Eine weitere Anwendung finden Sie im Softwareteil: Das Programm "Soundbox" (Kapitel 9.7) arbeitet im Zusammenhang mit der Tastaturabfrage sehr stark mit ETC und PTC.

6.15 Sonstige Befehle der CPU

Wartebefehle

NOPW, NOPT

WAIT

Die Wartebefehle sind die simpelsten Befehle unserer CPU. Sie lassen die CPU einfach für eine bestimmte Anzahl Taktzyklen warten. Ein Taktzyklus ist die kleinste Zeiteinheit, die die CPU überhaupt kennt. Die Ausführungszeit jedes Befehls wird in Taktzyklen angegeben (vergleiche Anhang A).

Kennonik	Code	Wirkung	Flags C Z
NOPW	77	No Operation 2 Taktzyklen	- -
NOPT	206	No Operation 3 Taktzyklen	- -
WAIT n	78,n	No Operation bis n Taktzyklen	- -

Die verschiedenen Wartebefehle unterscheiden sich nur durch ihre unterschiedlichen Wartezeiten. Bei NOPW und NOPT sind das 2 bzw. 3 Taktzyklen, bei WAIT kann eine gewünschte

Anzahl Taktzyklen (max. 255) angegeben werden. Der Taktzyklus der CPU wird vom Counter der CPU bestimmt.

Dieser Counter macht nichts anderes, als einfach die Geschwindigkeit anzugeben, mit der die einzelnen Befehle ausgeführt werden. Den Takt nimmt der Counter von einem externen Taktgeber. Bei Mikroprozessoren handelt es sich dabei um einen Quarz. Ein Taktzyklus dauert bei unserer CPU zirka 5,2 us (1/192'000 Sekunden). In einer Sekunde können also zirka 96'000 NOPW-Befehle abgearbeitet werden. Zum Vergleich: ein Tag hat zirka 86'000 Sekunden!



Wenn Sie auf der Befehlsliste im Anhang A nachschauen, finden Sie jeweils unter 'Zyklen' die Zeit, die die CPU zur Ausführung eines Befehls benötigt.

Die NOP-Befehle werden oft eingesetzt, um in einem Programm z.B. bei der Fehlersuche die Wirkung eines Befehls auszuschalten, ohne das ganze Programm verschieben zu müssen. Oder auch, um freien Platz für spätere Einfügungen zu reservieren. Da sich NOPT und NOPW nur durch ihre unterschiedlichen Ausführungszeiten unterscheiden, sollte der kürzere NOPW-Befehl für diese Zwecke bevorzugt werden.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LIDP	25900	16,101,44 25900->DP
...03	LDI	87	(DP)->A
...04	INCA	66	A+1->A
...05	STD	82	A->(DP)
...06	RTN	55	



Dieses Programm inkrementiert den Inhalt der Speicherzelle 25900. Ersetzen Sie INCA (Zeile ...04) durch NOPW. Ergebnis?

Den WAIT-Befehl benutzt man oft, um den Programmablauf etwas zu verzögern. Damit kann man den Rechner für eine bestimmte Zeit anhalten oder sogar Programme zur Zeitmessung erstellen! Das nächste Beispiel zeigt, wie man ein solches Programm anpackt.

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 9	2,9	9->A. 9+1=Wartezeit in Sekunden
...02	PUSH	52	Vorbereitung 3.LOOP
...03	LIA 15	2,15	15->A
...05	PUSH	52	Vorbereitung 2.LOOP
...06	LIA 45	2,45	45->A
...08	PUSH	52	Vorbereitung 1.LOOP
...09	WAIT 255	78,255	Warte 255+6 Taktzyklen
...11	LOOP 3	47,3	1.LOOP
...13	LOOP 8	47,8	2.LOOP
...15	LOOP 13	47,13	3.LOOP
...17	JP 49321 ¹⁾	121,192,169	BEEP und zurück ins BASIC

Der Rechner wartet 10 Sekunden, bis er sich mit einem BEEP meldet. Diese 10 Sekunden kommen wie folgt zustande:

$$t = \frac{(9+1)(15+1)(45+1)(255+6)}{192000} = 10 \text{ Sekunden}$$

Natürlich können Sie Ihren PC damit nicht gerade als äußerst präzises Meßinstrument einsetzen, aber für Verzögerungen oder kleinere Zeitmessungen sind diese Befehle bestens geeignet.

1) PC 1421: 55539 (216,243), PC 126X: 48908 (191,12), PC 1350: 49430 (193,22)

*Manipulation der Flags**RC, SC*

Bei jeder Addition bzw. Subtraktion werden die Flags automatisch gesetzt. Es gibt jedoch Befehle, die direkt auf die Flags zugreifen: SC (Set Carry) setzt das Carry-Flag und RC (Reset Carry) löscht es. In beiden Fällen wird zusätzlich das Zero-Flag gesetzt.

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	
SC	208	1->C, 1->Z	* *	C=1:Z=1
RC	209	0->C, 1->Z	* *	C=0:Z=1

Mit diesen beiden Befehlen können wir den Zustand der Flags ganz schön beeinflussen. RC und SC werden aber nur selten eingesetzt; so z.B. bei den Schiebe-Befehlen (SL,SR) und bei der Erzeugung von Fehlermeldungen.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 3	2,3	3->A
...02	LP 52	180	52->P
...03	EXAM	219	A<->(P)
...04	LIA 88	2,88	88->A
...06	STR	50	A->R
...07	SC	208	1->C
...08	RTN	55	

Nach dem Aufrufen des Programms erscheint die Fehlermeldung ERROR 3. Nicht etwa, weil sich im Maschinenprogramm irgend

ein Fehler eingeschlichen hätte, sondern weil die Fehlermeldung bewußt erzeugt wird. Sie können diese Aussage überprüfen, indem Sie die Zeile 1 ('LIA 3') z.B. durch 'LIA 4' ersetzen und damit die Fehlermeldung ERROR 4 erzeugen.

Eine Fehlermeldung erscheint nur, wenn das Carry-Flag gesetzt ist. Ohne den SC-Befehl geht da gar nichts!

Für Fehlermeldungen verwenden viele Computer das gleiche oder ein ähnliches Prinzip: Ein gesetztes Carry-Flag weist auf einen Fehler hin. In unserem Beispiel wird ein solcher Fehler simuliert.

Wie man Fehlermeldungen erzeugt und weitere Effekte hervorruft, werden wir noch ausführlich in Kapitel 8.2 behandeln.

Ausblick: Die Port-Befehle

Die Ports sind etwas komplexer als das, was wir nun schon besprochen haben. Aber sie sind keineswegs nur Bitfummern, Elektronikfreaks oder Bastlernaturen vorbehalten, sondern man kann sie sehr gut auch in einfacheren Maschinenprogrammen, z.B. zur Tastaturabfrage, benutzen. Da alle Port-Befehle eingehend und mit vielen Beispielen erklärt werden, lohnt es sich auf jeden Fall, das nächste Kapitel zu studieren.

Viele unserer Programme basieren auf den Port-Befehlen, z.B. das Programm 'Soundbox'. Ohne Ports steht man sonst vielfach auf verlorenem Posten. Wie soll man ohne Ports externe Geräte steuern oder Daten, z.B. Meßwerte, von außen empfangen?

Im nächsten Kapitel sehen Sie, wie man solche interessanten Probleme anpackt.



Kapitel 7

Ports: Die Verbindung zur Außenwelt

Sie haben sich vielleicht schon gefragt, wie denn die CPU externe Daten aufnimmt oder interne Daten ausgibt. Zuerst einmal: Was versteht man überhaupt unter internen und externen Daten? Intern sind diejenigen Daten, die in der CPU, im RAM oder im ROM gespeichert sind und nach 'außen' gesendet werden können. Bei den externen Daten verhält es sich gerade umgekehrt: extern sind diejenigen Daten, die von 'außen', also von der Tastatur, vom Cassettenrekorder oder von anderen Peripheriegeräten zur CPU und dann ins RAM gelangen.

Jetzt stellt sich uns die Frage, auf welche Art und Weise interne und externe Daten gesendet oder empfangen werden können.

Für diesen Datentransfer benötigt die CPU spezielle Schnittstellen: die Ports. Die deutsche Übersetzung für 'Port' lautet 'Hafen'. Und genau das ist auch die Aufgabe der Ports: sie sind für den Datenverkehr der CPU mit der Umwelt zuständig. Sie sind also die Zugänge von der Außenwelt zur CPU und umgekehrt.

Die CPU SC 61860 besitzt insgesamt 4 Ports, die für diesen Datenverkehr verwendet werden können: Sie heißen IA-Port, IB-Port, F0-Port und Control-Port. Den Control-Port nennen wir der Einfachheit halber auch C-Port. Jeder Port besteht aus 8 Bits, ist also gleich aufgebaut wie eine normale Speicherzelle.

In den folgenden Kapiteln wollen wir zuerst die Befehle zur Steuerung der Ports behandeln, dann erst werden wir die Funktionen der einzelnen Portbits anhand mehrerer Beispiele detailliert behandeln.

7.1 Befehle zur Steuerung der Ports

Senden von internen Daten (Ausgabebefehle)

**OUTA OUTB
OUTC OUTD**

Zur Ausgabe von Daten an die Ports dienen die internen RAM-Register 92 bis 95. Anstatt dass diese Register spezielle Namen haben, wie z.B. das A- oder B-Register der CPU, sprechen wir sie lediglich mit ihren Nummern an. Z.B. hat ja das A-Register die interne Registernummer 2. Genauso werden auch die internen RAM-Register 92 bis 95, die für die Ports zuständig sind, immer mit dem Zeiger P adressiert.

Das Register 92 bildet die Verbindung zum IA-Port, das Register 93 zum IB-Port, das Register 94 zum F0-Port, und das Register 95 schließlich ist mit dem C-Port verbunden.

Soll zum Beispiel der Wert 34 auf den IA-Port ausgegeben werden, so muß zuerst das interne RAM-Register 92 mit dieser Zahl geladen werden (LIA 34, LIP 92, EXAM). Der Befehl OUTA bewirkt nun, daß die Zahl, die in Register 92 gespeichert ist (34) zum IA-Port gesendet wird.

Empfangen von externen Daten (Lesebefehle)

INA INB

Über den IA- und IB-Port können Daten nicht nur gesendet, sondern auch empfangen werden. Mit diesen beiden Ports ist es also auch möglich, externe Daten zu lesen, während der F0- und der C-Port nur für die Ausgabe von Daten verwendet werden können. Die Lesebefehle INA und INB ermöglichen eine direkte Abfrage des IA- und IB-Ports; durch diese beiden Befehle können die Daten, die am entsprechenden Port anliegen, in den Akku geladen werden.

Die folgende Tabelle gibt über Codes und andere Einzelheiten Auskunft:

Mnemonic	Code	Wirkung	Flags	BASIC
			C Z	

Ausgabebefehle

OUTA	93	(92)->IA-Port	- -	IA=M(92)
OUTB	221	(93)->IB-Port	- -	IB=M(93)
OUTF	95	(94)->FO-Port	- -	FO=M(94)
OUTC	223	(95)-> C-Port	- -	C=M(95)

Lesebefehle

INA	76	IA-Port->A	- -	A=IA
INB	204	IB-Port->A	- -	A=IB

7.2 Tastaturabfrage über den IA- und IB-Port

Bis jetzt war es uns nur möglich, die Tastatur in einem Programm indirekt abzufragen, z.B. mit INKEY\$ oder CALL 5208, was zuweilen eine relativ lange Ausführungszeit zur Folge hatte. Durch die Abfrage der Ports eröffnen sich nun aber ganz neue Dimensionen der schnellen und effizienten Programmierung von Tastaturabfragen. Die elektrischen Kontakte der Tasten sind nämlich direkt mit gewissen Bits des IA- und IB-Ports verbunden. Diese Zusammenhänge sind im folgenden elektronischen Schaltbild aufgezeigt:

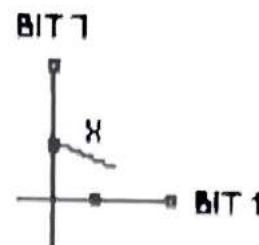
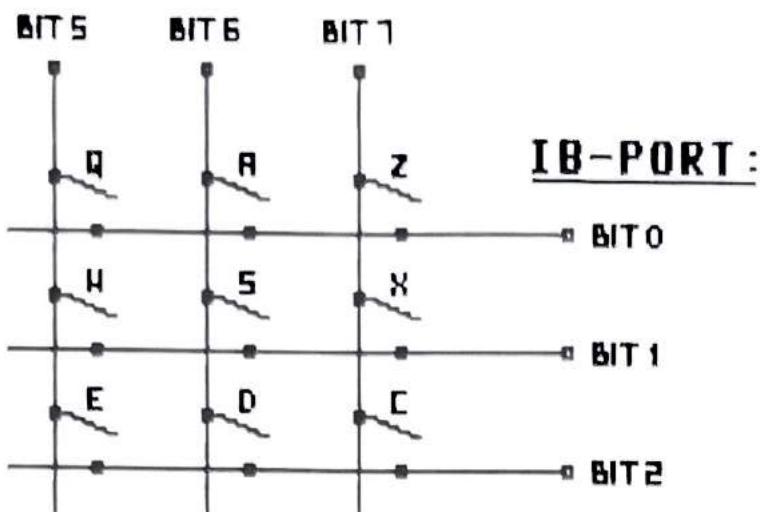
IA-PORT:

ABB. 2

ABB. 1

Wie man aus Abbildung 1 sofort ersehen kann, sind alle Tasten in einer sogenannten Matrix angeordnet (Abb. 1 zeigt nur einen Ausschnitt der gesamten Tastaturmatrix). Die Erklärung dazu ist deshalb relativ einfach: Wird zum Beispiel die Taste 'X' (Abb. 2) gedrückt, so wird das Bit 1 des IB-Ports mit dem Bit 7 des IA-Ports verbunden, bzw. kurzgeschlossen. Liegt nun eine elektrische Spannung (+5V) an Bit 1 des IB-Ports, so ist diese Spannung auch an Bit 7 des IA-Ports vorhanden (falls eben die Taste 'X' gedrückt ist). Der Strom fließt also von Bit 1 des IB-Ports zu Bit 7 des IA-Ports. Wenn wir also eine Tastaturabfrage für die Taste 'X' programmieren wollen, müssen wir zuerst wissen, wie wir softwaremäßig eine elektrische Spannung auf das Bit 1 des IB-Ports geben können und wie wir danach testen können, ob diese Spannung auch an Bit 7 des IA-Ports anliegt.

Eine Abfrage der Taste 'X' wird folgendermaßen programmiert:

Durch das Setzen des 1. Bits des IB-Ports mit dem OUTB-Befehl liegt nun an demselben Bit eine positive Spannung von 5 Volt. Indem wir nun den IA-Port mit dem INA-Befehl abfragen, können wir testen, ob die Taste 'X' gedrückt ist oder nicht. Wenn nämlich der Strom bei gedrückter Taste auch an Bit 7 des IA-Ports anliegt, so finden wir, daß dieses 7. Bit gesetzt (1) ist.

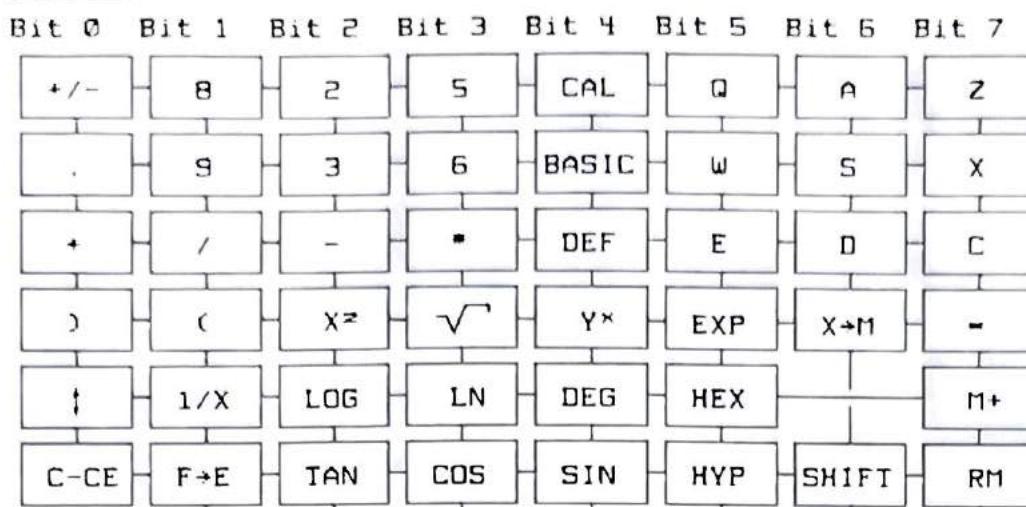
Maximal können auf diese Weise acht Tasten gleichzeitig abgefragt werden. Dazu nur ein kleines Beispiel: Ist das 1. Bit des IB-Ports gesetzt, und wird sowohl die Taste 'X', als auch die Taste 'S' gedrückt, so finden wir bei der Abfrage des IA-Ports, daß sowohl Bit 6, als auch Bit 7 gesetzt ist (im Akku befindet sich nach dem INA-Befehl also der Wert $192=26+27$).

Zusammenfassung

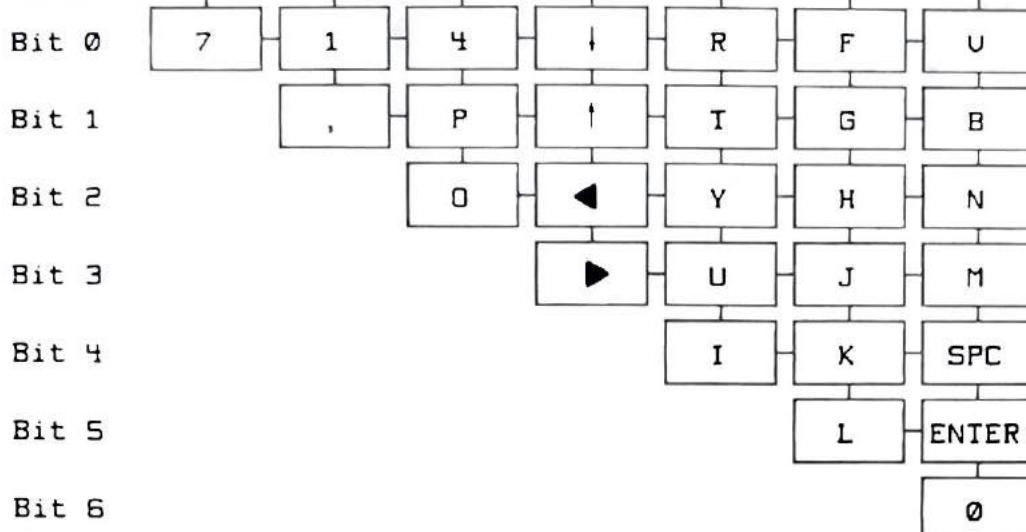
Eine bestimmte Taste kann abgefragt werden, indem man dasjenige Bit setzt, das an den ersten Kontakt der gewünschten Taste geführt ist. Daraufhin fragt man dasjenige Bit ab, welches mit dem zweiten Kontakt der Taste verbunden ist. Zum Setzen eines solchen Portbits verwendet man die Befehle OUTA und OUTB. Zur Abfrage eines Portbits werden die Befehle INA und INB benötigt. Da die Tasten in einer Matrix angeordnet sind, kann jede einzelne Taste einfach bestimmt und abgefragt werden.

Auf den folgenden Seiten finden Sie die vollständigen Tastatormatrizen der verschiedenen Rechner. Beim PC 1350 ist der größte Teil der Tasten weder mit dem IA-, noch mit dem IB-Port verbunden. Leider konnten wir bis jetzt noch nicht ermitteln, mit welchem System die restlichen Tasten abgefragt werden können.



*Tastaturmatrix des PC 140X*IA-PortIB-Port

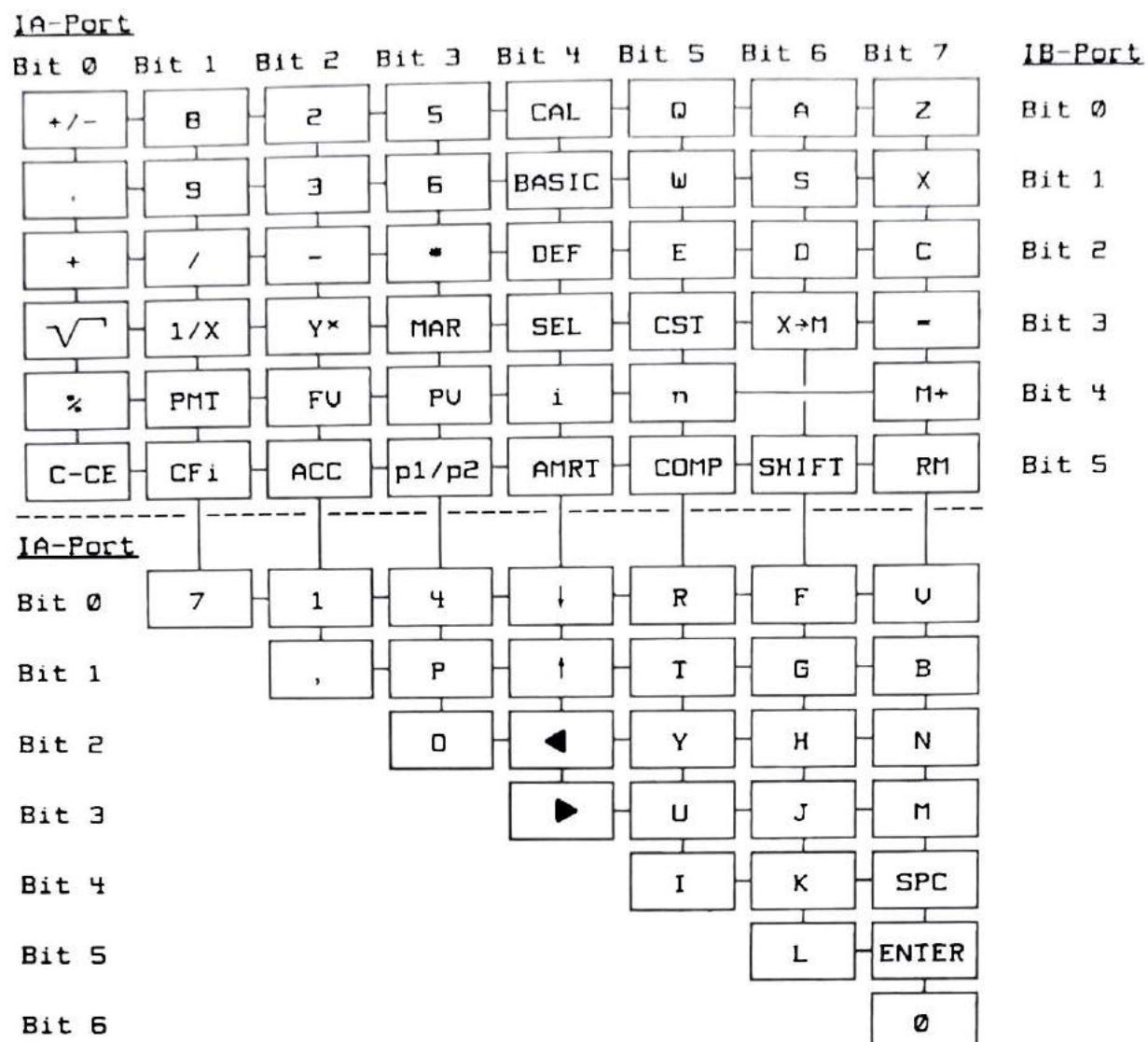
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5

IA-Port

Abfrage des ON/OFF-Schalters:

In der Stellung 'OFF' ist das Bit 0 des IB-Ports gesetzt.

Tastaturmatrix des PC 1421

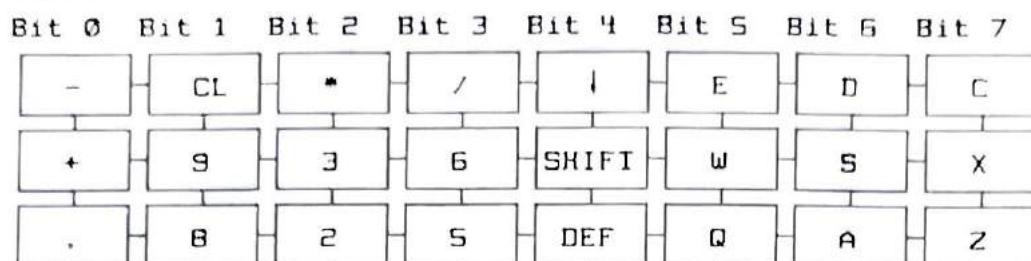


Abfrage des ON/OFF-Schalters:

In der Stellung 'OFF' ist das Bit 0 des IB-Ports gesetzt.

Tastaturmatrix des PC 126X

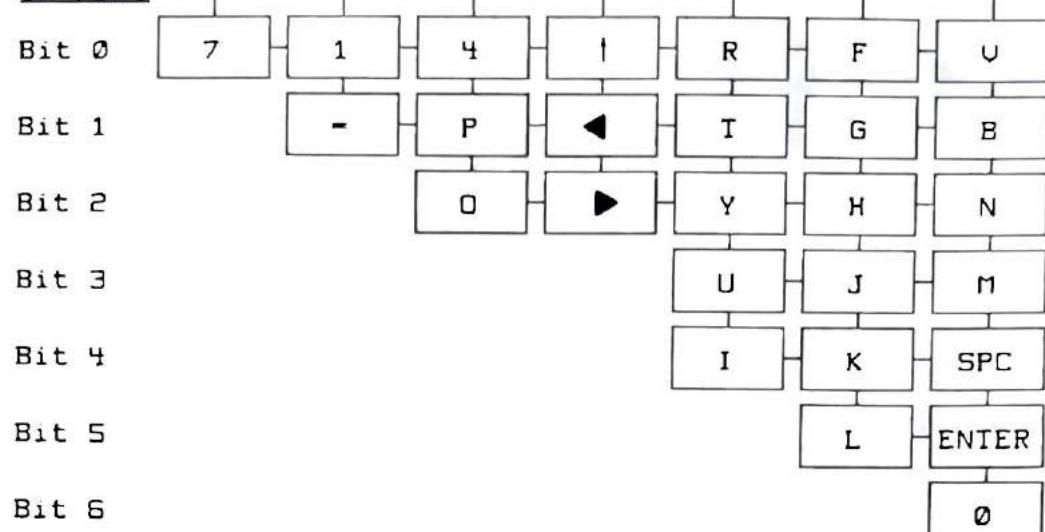
IA-Port



IB-Port

Bit 0
Bit 1
Bit 2

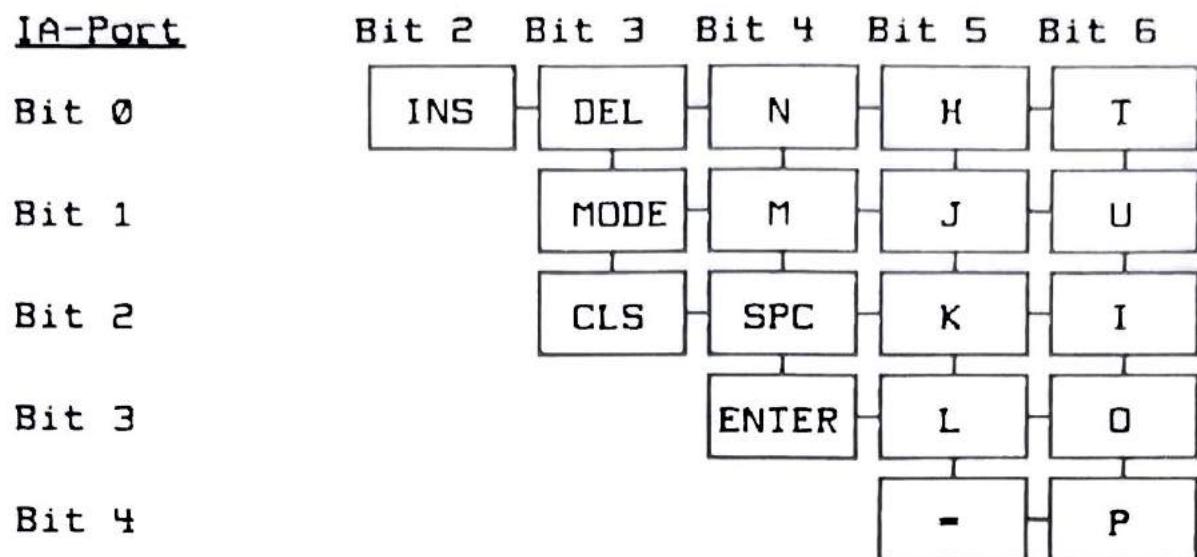
IA-Port



MODE-Schalter:

- OFF: 7. Bit des IA-Ports¹⁾ mit dem 3. Bit des IB-Ports²⁾ verbunden
RUN: nichts verbunden
PRO: 1. Bit des IB-Ports²⁾ mit dem 3. Bit des IB-Ports¹⁾ verbunden
RSV: 0. Bit des IB-Ports²⁾ mit dem 3. Bit des IB-Ports¹⁾ verbunden

Tastaturmatrix des PC 1350



1) Lesen

2) Schreiben

Abfrage des ON/OFF-Schalters:

In der Stellung 'OFF' ist das Bit 6 des IA-Ports mit Bit 7 des IA-Ports verbunden.

Beim PC 1350 kann der 1B-Port zur Tastaturabfrage nicht eingesetzt werden.

Die Ausgabe- und Lesebefehle für die Ports (OUTx, INx) wollen wir nun zum ersten Mal in einem praktischen Beispiel im Zusammenhang mit einer Tastaturabfrage anwenden.

Das folgende Programm soll nur auf die Tasten 'G', 'H' und 'J' (beim PC 1350: Tasten 'U', 'I' und 'O') reagieren. Wird 'G' (beim PC 1350: Taste 'U') gedrückt, kehrt der Rechner sofort ins BASIC zurück. Werden die Tasten 'H' und 'J' (beim PC 1350: Tasten 'I' und 'O') gleichzeitig gedrückt, so ertönt ein BEEP-Signal:

Adresse	Mnemonic	Codes	Kommentar
...00	LIP 92	18,92	92->P. P zeigt auf IA
...02	LIA 64	2,64	64->A
...04	EXAM	219	A<->(P), d.h. 64->(92)
...05	OUTA	93	(92)->IA-Port, d.h. 64->IA, Bit 6 gesetzt $64=2^6$
...06	INA	76	IA-Port->A, der Wert des IA-Port setzt sich aus den einzelnen Bits der gedrückten Tasten zu- sammen Keine Taste: IA-Port=0, sonst: IA-Port=2^Bit- nummer

...07	CPIA 12	103,12	Ist A=12? $12=2^2+2^3$ Bit 2: Taste 'H' (PC 1350: Taste '1') Bit 3: Taste 'J' (PC 1350: Taste '0')
...09	JRNZP 3	40,3	Ist A<>12: Springe zu ...13
...11	CAL 2755 ¹⁾	234,195	Sonst: BEEP
...13	CPIA 2	103,2	Ist A=2? $2=2^1$ Bit 1: Taste 'G' (PC 1350: Taste 'U')
...15	JRNZM 16	41,16	Ist A<>2: Zurück zum Pro- grammanfang
...17	RTN	55	Sonst: Zurück ins BASIC

7.3 Der Control-Port (C-Port)

Der Control-Port (deutsch 'Kontrollhafen') dient zur Steuerung des Displays, des Summers und zum softwaremäßigen Ausschalten des Rechners. Auch der C-Port besteht aus 8 Bits, davon sind jedoch nur 7 Bits belegt. Beim Arbeiten mit dem C-Port muß man vorsichtig sein; das Setzen falscher Bits kann zu Störungen in der Anzeige führen.

1) PC 1421: CAL 2700 (234,140), PC 126X: CAL 2094 (232,46), PC 1350:
CAL 2379 (233,75)

Belegung der einzelnen Bits des Control-Ports:

Bit	Funktion	
0	Anzeige ein/aus	(gesetzt = ein)
1	Counter-Reset	(gesetzt = Reset)
2	CPU-Halt	(gesetzt = Halt)
3	Rechner ein/aus	(gesetzt = aus)
4	Frequenz des Summers: 2/4 kHz	(gesetzt = 4 kHz)
5	Summer ein/aus	(gesetzt = ein)
6	Steuerung des Summers	
7	Keine Funktion	

Über die Funktion des 1. und 2. Bits besitzen wir keine schlüssigen Informationen. Bei Tests reagierte die CPU weder auf das Setzen noch das Löschen dieser Bits, sondern sie arbeitete ganz normal weiter.

Bit 0 des C-Ports: Steuerung der Anzeige

Wenn Sie das 0. Bit des C-Ports setzen, d.h. den Wert 1 auf den C-Port ausgeben, so wird die Anzeige eingeschaltet und kann so z.B. für die Darstellung von Grafik verwendet werden. Sobald Sie das 0. Bit wieder löschen, d.h. wenn Sie 0 auf den C-Port ausgeben, wird die Anzeige ausgeschaltet, die Zeichen verschwinden.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 1	2,1	1->A
...02	LIP 95	18,95	95->P. P zeigt auf den C-Port
...04	EXAM	219	A<->(P); d.h. 1<->(95)

...05	OUTC	223	(P)->C-Port; d.h. 1->C- Port, Bit 0 setzen: Anzeige ein
...06	LIA 255	2,255	255->A
...08	PUSH	52	Vorbereitung 4. LOOP
...09	PUSH	52	Vorbereitung 3. LOOP
...10	PUSH	52	Vorbereitung 2. LOOP
...11	PUSH	52	Vorbereitung 1. LOOP
...12	WAIT 255	78,255	6+255 Taktzyklen warten
...14	LOOP 3	47,3	1. LOOP
...16	WAIT 255	78,255	6+255 Taktzyklen warten
...18	LOOP 3	47,3	2. LOOP
...20	LIA 0	2,0	0->A
...22	EXAM	219	A<->(P)
...23	OUTC	223	(P)->C-Port; d.h., 0-> C-Port, Anzeige aus
...24	WAIT 255	78,255	6+255 Taktzyklen warten
...26	LOOP 3	47,3	3. LOOP
...28	WAIT 255	78,255	6+255 Taktzyklen warten
...30	LOOP 3	47,3	4. LOOP
...32	RTN	55	

Wenn Sie das Programm starten, bleibt der Displayinhalt zuerst für eine halbe Sekunde erhalten. Danach erlischt die Anzeige, und nach einer weiteren halben Sekunde kehrt der Rechner ins BASIC zurück. Die halbe Sekunde Wartezeit wird jeweils durch zwei hintereinander liegende Warteschleifen erreicht.

Bit 3 des C-Ports: Rechner ein/aus

Wird das 3. Bit des C-Ports gesetzt, also $8=2^3$ an den C-Port ausgegeben, so wird damit der Rechner ausgeschaltet.

Vorsicht: Einschalten sollten Sie nur über die BRK/ON-Taste: Beim Betätigen des ON/OFF-Schalters wird ein ALL RESET ausgeführt.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	L1P 95	18,95	95->P; P zeigt auf den C-Port
...02	L1A 8	2,8	8->A
...04	EXAM	219	A<->(P); d.h. 8->(95)
...05	OUTC	223	(P)->C-Port; d.h. 8->C-Port, Bit 3 gesetzt: 8=23, Rechner wird ausgeschaltet

Der übliche RTN-Befehl ist hier nicht nötig, da der Rechner gleich nach der Ausführung des OUTC-Befehls ausgeschaltet wird und gar nicht mehr zur Ausführung des RTN-Befehls kommt.

Wegen der Gefahr eines Systemabsturzes kommt diese Art des Ausschaltens nicht in Frage. Soll der Rechner trotzdem vom Programm aus ausgeschaltet werden, so empfiehlt es sich, das Unterprogramm OFF des internen ROM (Anhang C) zu verwenden, da der Rechner auf diese Weise ohne die Gefahr eines Datenverlustes auch mit dem ON/OFF-Schalter wieder eingeschaltet werden kann. Dieses ROM-Unterprogramm ist bei den einzelnen Rechnern an folgenden Adressen lokalisiert:

PC 140X:	1434
PC 1421:	1479
PC 126X:	1112
PC 1350:	1240

Bit 4 und 5 des C-Ports: Steuerung des Summers

Mit dem 5. Bit des C-Ports wird der Summer eingeschaltet. Durch das 4. Bit kann nun die Frequenz des Summers bestimmt werden. Ist das Bit 4 nicht gesetzt, ertönt ein Ton mit der Frequenz 2 kHz, ist dieses Bit hingegen gesetzt, so hören Sie die

doppelte Frequenz, nämlich 4 kHz. Das ist auch die Frequenz des BASIC-BEEP.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LIP 95	18,95	95->P. P zeigt auf den C-Port
...02	LIA 48	2,48	48->A
...04	EXAM	219	A<->(P); d.h. 48<->(95)
...05	OUTC	223	(P)->C-Port; d.h. 48->C-Port, Bit 4/5 gesetzt: 48=25+24, Ton von 4 kHz ertönt
...06	RTN	55	Zurück ins BASIC, Summer wird automatisch ausgeschaltet

Nach dem Starten ertönt ein kurzer, hoher Ton. Ersetzen Sie probehalber den LIA 48-Befehl durch LIA 32. Was geschieht nun? Da dabei das 4. Bit nicht mehr gesetzt wird, wird die Tonfrequenz auf 2 kHz festgelegt. Bei einem zweiten Start des Programms hört man sofort, daß der Ton deutlich tiefer ist.

Der Summer bleibt nach dem Einschalten bis zum Ende des Programms eingeschaltet. Das kann zuweilen sehr unangenehm sein, besonders wenn das Programm nach dem BEEP-Ton noch nicht beendet ist, und der Summer während des ganzen Programmverlaufs eingeschaltet bleibt. Schalten Sie deshalb in solchen Fällen den Summer wieder aus, indem Sie den Wert 0 an den C-Port ausgeben.

Einsatz des Summers für variable Frequenzen

Mit dem Summer können Sie nicht nur 2 oder 4 kHz-Töne erzeugen, der Summer kann mit einem kleinen Trick auch für variable Frequenzen eingesetzt werden.

Ist nämlich das 5. Bit des C-Ports nicht gesetzt, kann der Strom für den Summer auch über das 4. Bit des C-Ports ein- und ausgeschaltet werden.

Der Summer verhält sich nun wie die Membran eines gewöhnlichen Lautsprechers: wird Strom auf die Membran gegeben (indem man Bit 4 setzt), so gibt sie einen 'Knack' von sich; die Membran wird herausgedrückt. Schaltet man den Strom wieder aus (indem man Bit 4 löscht), gibt die Membran erneut einen Knack von sich und kehrt in ihre Ruhestellung zurück. Wird der Strom nun sehr schnell ein- und ausgeschaltet, entsteht auf diese Weise ein Ton, dessen Frequenz von der Geschwindigkeit des Ein- und Ausschaltens abhängig ist. Die CPU arbeitet schnell genug, daß man damit auch hohe Töne erzeugen kann.

Im folgenden Programm wird der Summer über das 4. Bit des C-Ports periodisch ein- und ausgeschaltet. Mit dem WAIT-Befehl läßt sich die Ein- und Ausschaltzeit und damit die Tonhöhe verändern.

Beispiel:

Adresse	Mnemonic	Codes	Kommentar
...00	LIB 0	3,0	0->B
...02	LIA 0	2,0	0->A
...04	LIP 95	18,95	95->P. P zeigt auf den C-Port
...06	EXAM	219	A<->(P); d.h. 0->(95)
...07	LIA 16	2,16	16->A. 16=24
...09	OUTC	223	(P)->C-Port. (95) ist abwechselnd 0 oder 16
...10	WAIT 255	78,255	6+255 Taktzyklen warten

...12	EXAM	219	A<->(P); d.h. 16->(95), 0->A oder 0->(95), 16->A
...13	INCB	194	B+1->B
...14	JRNCM 6	43,6	Ist C=0 (da B<=255), dann zurück zum OUTC-Befehl. EXAM (vor dem INCB- Befehl) bewirkt, daß abwechselnd 0 oder 16 auf den C-Port ausgegeben wird, daß der Summer also ein- oder ausge- schaltet wird.
...16	RTN	55	

Der Inhalt des B-Registers bestimmt die Anzahl Durchläufe des Programms und somit die Länge des Tones. Die Tonfrequenz wird durch das Argument des WAIT-Befehls bestimmt. Wenn Sie das Programm starten, ertönt ein ziemlich tiefer Ton. Setzen Sie beim WAIT-Befehl andere Werte ein! Sie können auf diese Weise die Tonfrequenz ändern. Sicher haben Sie das Prinzip längst durchschaut: je kleiner das Argument des WAIT-Befehls, desto schneller wird der Strom für die Membran ein- und ausgeschaltet. Dies wiederum bewirkt eine Erhöhung der Tonfrequenz.

Leider wird bei Veränderung der Tonfrequenz auch die Tonlänge verändert. Tiefe Töne sind deshalb länger (das WAIT-Intervall ist größer) als hohe Töne.

Wir weisen noch einmal darauf hin: Haben Sie in einem Maschinenprogramm nach diesem Verfahren einen Ton erzeugt, sollten Sie unbedingt vor dem weiteren Programmablauf noch den Wert 0 an den C-Port ausgeben. Wie Sie wissen, bleibt der Summer sonst während der ganzen weiteren Programmausführung eingeschaltet. Der Stromverbrauch ist dann um ein Vielfaches größer als bei 'normalen' Maschinenprogrammen. Natürlich wird der Summer automatisch ausgeschaltet, wenn man ins BASIC zurückspringt.

Ein komfortables Programm zur Benutzung des PC als Musikcomputer finden Sie in Kapitel 9.7 unter dem Titel 'Soundbox'.

7.4 Der Anschluß externer Geräte

Alle SHARP Pocket Computer besitzen auf der linken Gehäuseseite eine Anschlußleiste, eine Schnittstelle, die normalerweise zum Laden, Speichern (Cassette) und Ausdrucken von Daten verwendet wird. Diese Schnittstelle kann aber durchaus auch zweckentfremdet werden; wir können sie sehr wohl auch zur Steuerung anderer externer Geräte benutzen. Warum sollten Drucker und Cassettengerät unsere einzigen Peripheriegeräte sein, wo wir doch ebensogut andere Geräte wie z.B. Lampen oder elektronische Schaltungen ansteuern können?

Da einzelne Anschlüsse der Schnittstelle auch zum Einlesen von Daten genutzt werden können, bietet sich uns die Möglichkeit, nicht nur externe Geräte zu steuern, sondern auch Meßwerte und Steuersignale von externen Geräten zu empfangen.

Das elektronische Relais

In Kapitel 10.4 dieses Buches finden Sie eine komplette Bauanleitung für ein sogenanntes elektronisches Relais.

Dieses spezielle für die SHARP-Pocket-Computer entwickelte Peripheriegerät erlaubt Ihnen den Anschluß zweier getrennt schaltbaren 220-Volt-Geräte. Weiter können über das elektronische Relais auch zwei verschiedene Steuerspannungen auf die Portleitung des Rechners gegeben werden.

Mit Hilfe des elektronischen Relais können Sie erstens also programmgesteuert beliebige Netzspannungsgeräte steuern, zweitens können über dieses Relais beliebige Steuerspannungen direkt in den Rechner geladen werden.

Die Bauanleitungen mit weiteren Detailangaben finden Sie, wie schon erwähnt, in Kapitel 10.4.

Der FO-Port

Wie der Control-Port ist auch der FO-Port nur ein Ausgabeport; über diesen Port können also nur Daten gesendet, nicht aber empfangen werden. Der Rechner verwendet den FO-Port zur Ausgabe von Daten an den Drucker, zur Steuerung des Anzeigeprozessors und zur Ansteuerung der verschiedenen RAM-Chips.

Für uns sind am FO-Port eigentlich nur die Bits 0 und 1 von Interesse. Beim PC 1350 sind es die Bits 2 und 1. Anstelle des Bits 0 ist beim PC 1350 also das Bit 2 an die Schnittstelle herangeführt. Wenn diesbezüglich im Folgenden keine Anmerkungen stehen, gilt für den PC 1350 immer: Bit 2 des FO-Ports umstelle des Bit 0.

Die Bits 0 und 1 sind nämlich an die seitliche Anschlußleiste des Rechners geführt. Es sind die beiden Bits, über die wir mittels des elektronischen Relais die beiden Netzspannungsgeräte ansteuern können. Ist z.B. das Bit 0 (PC 1350: Bit 2) des FO-Ports gesetzt, so liegt an Pin 4 der Anschlußleiste eine Spannung von zirka +5 Volt an. Diese Spannung wird im elektronischen Relais verstärkt, was bewirkt, daß das erste Netzspannungsgerät eingeschaltet wird. Wird Bit 0 wieder gelöscht, so schaltet auch das Gerät wieder ab. Merken Sie sich also:

Bit 0 (PC 1350: Bit 2) des FO-Ports: Mit Pin 4 der Schnittstelle verbunden.

Bit 1 des FO-Ports: Mit Pin 5 der Schnittstelle verbunden.

Die restlichen Bits des FO-Ports wollen wir nicht detaillierter behandeln, da sie nur für interne Steuerungen des Rechners zuständig sind. Für die Verwendung in eigenen Maschinenprogrammen haben sie keine Bedeutung.

Nun aber zurück zu den Bits 0 (PC 1350: Bit 2) und 1. Das folgende Programm soll Ihnen ein Beispiel geben, wie man externe Geräte mit diesen beiden Bits steuert.

Beispiel:

Mit dem folgenden Programm können Sie in Verbindung mit dem elektronischen Relais zwei externe Geräte ein- und ausschalten. Durch das Drücken der Taste 'H' bei den Rechnern PC 14XX und PC 126X, Taste 'I' beim PC 1350 wird Gerät 1 eingeschaltet, über die Taste 'J' bei den Rechnern PC 14XX und PC 126X, Taste 'O' beim PC 1350 kann Gerät 2 eingeschaltet werden. Beide Geräte werden wieder ausgeschaltet, wenn Sie die Taste 'K' bei den Rechnern PC 14XX und PC 126X, Taste 'P' beim PC 1350 drücken. Der Rechner kehrt dann ins BASIC zurück.

Adresse	Mnemonic	Codes	Kommentar
...00	LIP 92	18,92	92->P. P zeigt auf den IA-Port
...02	LIA 64	2,64	64->A
...04	EXAM	219	A<->(P); d.h. 64<->(92)
...05	OUTA	93	(92)->IA-Port; d.h. 64->IA, Bit 6 gesetzt: 6=26
...06	INA	76	IA-Port->A, der Wert des IA-Ports setzt sich aus den einzelnen Bits der gedrückten Tasten zusammen.
...07	CPIA 4	103,4	Ist A=4? 4=22 Bit 2: Taste 'H' (beim PC 1350: Taste 'I')
...09	JRNZP 8	40,8	Ist A>4: Springe zu ...18
...11	LIP 94	18,94	94->P. P zeigt auf F0

...13	ORIM 1 ¹⁾	97,1	(P) OR 1 ¹⁾ ->(P)
...15	OUTF	95	(94)->F0-Port; Gerät 1 wird eingeschaltet
...16	JRM 17	45,17	Zurück zum Programmanfang
...18	CPIA 8	103,8	Ist A=8? 8=23 Bit 3: Taste 'J' (beim PC 1350: Taste '0')
...20	JRNZP 8	40,8	Ist A<>8: Springe zu ...29
...22	LIP 94	18,94	94->P. P zeigt auf F0
...24	ORIM 2	97,2	(P) OR 2->(P)
...26	OUTF	95	(94)->F0-Port; Gerät 2 wird eingeschaltet
...27	JRM 28	45,28	Zurück zum Programmanfang
...29	CPIA 16	103,16	Ist A=16? 16=24 Bit 4: Taste 'K' (beim PC 1350: Taste 'P')
...31	JRNZM 32	41,32	Ist A<>16: Zurück zum Programmanfang
...33	LIP 94	18,94	94->P. P zeigt auf F0
...35	ANIM 0	96,0	0->(P); d.h. 0->(94)
...37	OUTF	95	(94)->F0-Port; beide Geräte werden ausgeschaltet
...38	RTN	55	

Schließen Sie das elektronische Relais an die seitliche Schnittstelle an und schalten Sie das Relais ein. An den Ausgang F0 0 des Relais schließen Sie z.B. eine Lampe an, an den Anschluß F0 1 des Relais wird ein zweites Gerät angeschlossen, z.B. ein Radio. Schalten Sie nun auch Lampe und Radio ein; die beiden Geräte sollten auf dieses Einschalten überhaupt nicht reagieren. Andernfalls ist entweder der Rechner abgestürzt, oder das elektronische Relais ist defekt.

1) PC 1350: ORIM 4 (97,4)

Nun starten Sie das Programm. Immer noch sollten die Geräte ausgeschaltet bleiben. Schalten Sie nun trotzdem ein, so würde wahrscheinlich das Maschinenprogramm fehlerhaft eingegeben.

Wenn bisher alles korrekt funktioniert hat, so drücken Sie nun die Taste 'H' (PC 1350: Taste 'T'); Gerät 1 (Lampe) schaltet ein. Nach Drücken der Taste 'J' (PC 1350: Taste 'O') wird auch Gerät 2 (Radio) eingeschaltet. Um die Geräte wieder auszuschalten und ins BASIC zurückzukehren, betätigen Sie einfach die Taste 'K' (PC 1350: Taste 'P').

Steuerung des Programms mit externen Spannungen über den IB-Port

Wie Sie vielleicht schon festgestellt haben, werden für Tastaturabfragen nur 6 Bits des IB-Ports benutzt (beim PC 1350 ist der IB-Port überhaupt nicht mit der Tastatur verbunden). Trotzdem besitzt natürlich auch der IB-Port ganze 8 Bits. Welche Aufgaben haben also die beiden Bits 6 und 7 des IB-Ports?

Diese beiden 'freien' Bits sind wie diejenigen des F0-Ports an die Anschlußleiste auf der linken Seite des Rechners geführt. Sie dienen normalerweise dazu, dem Rechner Fehlermeldungen des Druckers zu melden. Beide Bits lassen sich aber auch zur externen Steuerung von beliebigen Maschinenprogrammen verwenden. Bit 6 des IB-Ports ist mit Pin 9 der Schnittstelle verbunden, Bit 7 mit Pin 8. Auch das elektronische Relais ist mit diesen Anschlüssen der Schnittstelle verbunden. Auf dem Schaltbild des elektronischen Relais (Kapitel 10.4) sehen Sie die Anschlüsse IB 6 INPUT1 und IB 7 INPUT. Die Ziffern 6 und 7 stehen für das entsprechende Bit des IB-Ports. Beide Bits können also von 'außen' her über das elektronische Relais gesetzt werden.

Bit 6 des IB-Ports: Mit Pin 9 der Schnittstelle verbunden.
Bit 7 des IB-Ports: Mit Pin 8 der Schnittstelle verbunden.

Ein Bit ist gesetzt, wenn am entsprechenden Pin der Anschlußleiste eine Spannung von +5 Volt anliegt; das Bit ist gelöscht, wenn keine (0 Volt) oder eine negative Spannung anliegt.

Auch bei diesen Experimenten empfehlen wir, den Rechner nicht direkt anzusteuern, sondern das elektronische Relais als Interface dazwischenzuschalten. Das Relais besitzt nämlich eine Schutzschaltung, die Spannungen auch bis zu +/- 50 Volt über längere Zeit zulassen würde, ohne daß der Rechner dabei Schaden nähme.

Es spielt natürlich keine Rolle, von welcher Art Stromquelle die Spannung stammt.

Beispiel:

Sobald das 6. Bit des IB-Ports gesetzt wird, soll ein Dauer-BEEP ertönen. Wird dann auch noch das 7. Bit gesetzt, soll das Gerät, das am Anschluß F0 1 des elektronischen Relais eingesteckt ist, eingeschaltet werden. Durch das Drücken der ENTER-Taste soll das Verlassen des Programms ermöglicht werden.

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 0	2,0	0->A
...02	LIP 93	18,93	93->P. P zeigt auf den IB-Port
...04	EXAM	219	A<->(P); d.h. 0<->(93)
...05	OUTB	221	(P)->IB-Port; d.h. 0->IB, alle Bits gelöscht
...06	INB	204	IB-Port->A
...07	CPIA 64	103,64	Ist A=64? 64=26 Ist Bit 6 von IB gesetzt?
...09	JRZP 17	56,17	Ja: Springe zu ...27
...11	CPIA 128	103,128	Ist A=128? 128=27 Ist Bit 7 von IB gesetzt?
...13	JRZP 17	56,17	Ja: Springe zu ...31

...15	LIA 128 ¹⁾	2,128	128 ¹⁾ ->A
...17	LIP 92	18,92	92->P. P zeigt auf IA
...19	EXAM	219	A<->(P); d.h. 1281)<->(92)
...20	OUTA	93	(P)->IA-Port d.h. 128->IA
...21	INA	76	IA-Port->A
...22	CPIA 32 ²⁾	103,32	Ist A=32 ²⁾ ? 32=25 Ist Bit 5 von IA gesetzt?
...24	JRNZM 25	41,25	Nein: Springe zum Programmanfang
...26	RTN	55	Ja: Zurück ins BASIC
...27	CAL 2755 ³⁾	234,195	Aufruf Dauer-BEEP
...29	JRM 30	45,30	Zurück zum Programmanfang
...31	LIA 2	2,2	2->A
...33	LIP 94	18,94	94->P. P zeigt auf F0
...35	EXAM	219	A<->(P); d.h. 2<->(94)
...36	OUTF	95	(P)->F0-Port; d.h. 2->F0, Gerät 2 wird eingeschaltet
...37	JRM 38	45,38	Zurück zum Programmanfang

Schließen Sie das elektronische Relais an und starten Sie das Programm. Beim Setzen des 6. Bits des IB-Ports durch Anlegen einer elektrischen Spannung von mindestens +5 Volt am Anschluß IB 6 INPUT des elektronischen Relais gibt der Rechner einen Dauerton von sich. Wird das 7. Bit des IB-Port gesetzt (auf die gleiche Art wie das 6. Bit), so schaltet der Rechner das am zweiten Anschluß (F0 1) angeschlossene Gerät ein. Durch Drücken der ENTER-Taste kann das Programm beendet werden.

Nun gibt es noch eine andere Möglichkeit, wie die beiden Bits 6 und 7 des IB-Ports benutzt werden können.

1) PC 1350: LIA 16 (2,16)

2) PC 1350: CPIA 8 (103,8)

3) PC 1421: CAL 2700 (234, 140), PC 126X: CAL 2094 (232,46), PC 1350: CAL 2379 (233,75)

Der IB-Port ist ja ein Ein- und Ausgabeport. Bei den obigen Erklärungen und Beispielen über das 6. und 7. Bit wurde aber nur das Empfangen und Auswerten von externen Spannungen betrachtet.

Selbstverständlich kann der Wert dieser beiden Bits nicht nur gelesen, sondern auch vom Programm her geändert werden. Somit kann nämlich auch der IB-Port als Ausgabeport zur Steuerung externer Geräte funktionieren (wie der F0-Port).

 Für den Fall der Steuerung externer Geräte über das 6. und 7. Bit des IB-Ports sind am elektronischen Relais zwei Ausgänge vorgesehen, die auf dem Schaltbild in Kapitel 10.4 mit IB 6 OUTPUT und IB 7 OUTPUT bezeichnet sind. Wird das 6. bzw. das 7. Bit des IB-Ports von einem Programm aus gesetzt, so ist am entsprechenden Anschluß des Relais eine Spannung von zirka 4 bis 5 Volt vorhanden. Mit dieser Spannung können somit direkt z.B. TTL-ICs oder auch andere elektronische Schaltungen angesteuert werden.

Vorsicht: Schließen Sie an den Ausgängen der IB-Ports nie Glühlämpchen oder ähnliches direkt an, da diese Geräte zuviel Strom verbrauchen!

7.5 Der TEST-Befehl

 Mit diesem speziellen Maschinensprachebefehl können die BRK/ON-Taste und die Cassetteneingangsleitung (bei CLOAD) abgefragt werden.

Mnemonic	Code	Wirkung	Flags	BASIC
TEST n	107,n	TEST-Byte AND n	C Z	- * -

Der TEST-Befehl führt eine AND-Verknüpfung durch, ähnlich wie beim TSIA-Befehl wird das Ergebnis auch mit dem Zero-Flag angezeigt. Im Unterschied zu TSIA prüft der TEST-Befehl aber nicht ein bestimmtes Register, sondern die AND-Verknüpfung wird mit speziellen 8 Leitungen der CPU gemacht. Diese 8 Leitungen wollen wir zu einem Byte zusammenfassen und dieses 'neue' Byte im folgenden mit dem Namen TEST-Byte bezeichnen, da sein Zustand ja mit dem TEST-Befehl abgefragt werden kann.

Belegung der einzelnen Bits des TEST-Bytes:

Bit	Funktion
0	Systemtakt 512 ms
1	Systemtakt 2 ms
2	Keine Funktion
3	BRK/DN-Taste
4	Keine Funktion
5	Keine Funktion
6	RESET-Taste
7	Cassetteneingangsleitung (LOAD)

Bit 0 und 1 der TEST-Bytes: Taktsignale 2 ms/512 ms

Diese beiden Bits werden im angegebenen Zeitraster gesetzt und wieder gelöscht. Das Bit 0 des TEST-Bytes wird also alle 512 ms (1 ms = 1 Tausendstel Sekunde) gesetzt bzw. gelöscht. Der Zustand des 1. Bits wird sogar alle 2 ms geändert. Das Bit 1 wollen wir nicht weiter behandeln, da das Zeitintervall nur sehr kurz und deshalb von geringerer praktischer Bedeutung ist.

Mit dem 0. Bit hingegen könnte man auf einfache Art Zufallszeiten erzeugen; zu diesem Zweck programmieren Sie eine Schleife, die der Rechner immer wieder durchläuft, bis das 0. Bit gesetzt wird. Je nachdem zu welchem Zeitpunkt die

Schleife begonnen hat, dauert es mehr oder weniger lange, bis das Bit 0 gesetzt wird.

Auf ein Programmbeispiel wollen wir hier verzichten; diese beiden Bits haben in nur sehr wenigen Fällen eine wirklich praktische Bedeutung.

Bit 3 des TEST-Bytes: Abfrage der BRK/ON-Taste

Wird die BRK/ON-Taste gedrückt, so ist während der Dauer des Tastendrucks das 3. Bit des TEST-Bytes gesetzt. Mit TEST 8 ($8=2^3$) können Sie deshalb den Zustand dieses Bits abfragen. Ist es gesetzt (Z-Flag=0), so wissen Sie, daß die BRK/ON-Taste gerade betätigt wird. Nach dem Überprüfen des TEST-Bytes mit dem TEST-Befehl ist das Z-Flag, das das Ergebnis der AND-Verknüpfung mit dem getesteten Bit des TEST-Bytes angibt, gesetzt, wenn das getestete Bit des TEST-Bytes gelöscht ist.

Dazu schauen wir uns am besten gleich ein Beispiel an:

Der Rechner soll einen Dauerton erzeugen, der nur mit der BRK/ON-Taste wieder abgebrochen werden kann.

Adresse	Mnemonic	Codes	Kommentar
...00	LIP 95	18,95	95->P. P zeigt auf den C-Port
...02	LIA 32	2,32	32->A
...04	EXAM	219	A<->(P); d.h. 32<->(95)
...05	OUTC	223	(P)->C-Port; d.h. 32->C-Port, Einschalten des Summers (2 kHz) durch Setzen des 5. Bits des C-Ports
...06	TEST 8	107,8	Ist die BRK/ON-Taste gedrückt?
...08	JRZM 3	57,3	Nein: Springe zu ...06
...10	RTN	55	Ja: Zurück ins BASIC

Nach dem Starten des Programms ertönt ein Dauer-BEEP. Erst wenn Sie die BRK/ON-Taste drücken, bricht der Ton ab, und der Rechner kehrt ins BASIC zurück.

Bit 6 des TEST-Bytes: Abfrage der RESET-Taste

Dieses Bit ist nur dann gesetzt, wenn die RESET-Taste betätigt wird. Theoretisch könnte man also mit dem TEST-Befehl die RESET-Taste abfragen; in der Praxis ist dies aber nicht möglich. Sobald nämlich die RESET-Taste gedrückt wird, bricht der Rechner das laufende Programm ab. Dieses Bit hat deshalb für uns keinen praktischen Wert.

Bit 7 des TEST-Bytes: Cassetteneingangsteigung (CLOAD)

Denjenigen Anschluß der zeitlichen Schnittstelle, über den die Daten und Programme von der Cassette in den Rechner gelangen, nennt man Cassetteneingangsleitung. Dieser Anschluß ist direkt mit dem 7. Bit des TEST-Bytes verbunden. Mit TEST 128 kann man deshalb den logischen Zustand (0 oder 1) dieses Bits überprüfen.

Wichtig: TEST 128 funktioniert nur, wenn das 6. Bit des C-Ports gesetzt ist. Setzen Sie also jeweils das 6. Bit des C-Ports, bevor Sie TEST 128 benutzen wollen.

Ist das Z-Flag nach dem TEST 128-Befehl nicht gesetzt, so liegt eine positive Spannung (4-5 Volt) an Pin 6 (Cassetteneingangsleitung) der Schnittstelle an. Ist das Z-Flag gesetzt, so ist an Pin 6 keine Spannung vorhanden.

Alle Daten, die von der Cassette geladen werden, werden auf diese Weise, also mit dem TEST-Befehl, in den Rechner transferiert.

Mit dem Bit 7 des TEST-Bytes könnte man zum Beispiel auch Ladebefehle mit höherer Übertragungsgeschwindigkeit oder Baudrate (1 Baud = 1 Bit/s) programmieren. Solche "Highspeederweiterungen" kommen jedoch höchstens für den Daten austausch zwischen zwei Rechnern in Frage. Die Übertragungsgeschwindigkeit ließe sich soweit steigern, daß 10 KByte in wenigen Sekunden kopiert wären. Bei Cassettenbetrieb ist die normale Übertragungsgeschwindigkeit jedoch schnell genug; eine Steigerung der Baudrate hätte hier nur vermehrte Schreib- und Lesefehler zur Folge.

Wenn Sie ein Programm von Cassette laden, hören Sie die Tons folge im Summer. Das müßte nicht unbedingt so sein, denn ob der Ton hörbar ist oder nicht, hängt vom Zustand des 5. und 6. Bit des C-Ports ab. Bis jetzt haben wir dieses 5. Bit immer nur zum Ein- oder Ausschalten des Summers für feste Frequenzen benutzt. Ist nun aber das 6. Bit des C-Ports gesetzt (und das ist ja eine Voraussetzung, damit der TEST 128-Befehl überhaupt funktioniert), so bekommt das 5. Bit des C-Ports eine neue Funktion:

Ist nämlich sowohl das 5. als auch das 6. Bit des C-Ports gesetzt, so ist der Summer des Rechners direkt mit der Cassetten eingangsleitung verbunden. Die Impulse, die nun von der Cassette in den Rechner gelangen, werden gleichzeitig durch den Summer wiedergegeben.

Allgemein gilt (Voraussetzung: 6. Bit des C-Ports gesetzt):

5. Bit des C-Ports gesetzt:	Ton auf dem Summer hörbar
5. Bit des C-Ports nicht gesetzt:	Ton auf dem Summer nicht hörbar

Beispiel:

Wir wollen ein Programm schreiben, das auf dem Summer hörbar macht, was vom Cassetteninterface her kommt. Es kann mit der BRK/ON-Taste unterbrochen werden.

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 96	2,96	96->A. 96=25+26
...02	LIP 95	18,95	95->P. P zeigt auf C-Port
...04	EXAM	219	A<->(P); d.h. 96<->(95)
...05	OUTC	223	(P)->C-Port; d.h. 96-> C-Port
...06	TEST 8	107,8	Ist BRK/ON-Taste ge- drückt?
...08	JRZM 3	57,3	Nein: Springe zu ...06
...10	RTN	55	Ja: Zurück ins BASIC



Wenn Sie den LIA 96-Befehl in Zeile ...00 durch LIA 64 ersetzen, ist das, was vom Cassetteneinterface her kommt, auf dem Summer nicht hörbar.

Sie haben in diesem Kapitel einen tiefen Einblick in die hardwaremäßig bedingte innere Struktur und das System des PC erhalten. Im nächsten Kapitel können Sie mehr über die softwaremäßigen internen Zusammenhänge zwischen BASIC und Maschinensprache, sowie weitere Einzelheiten über das Betriebssystem des PCs erfahren.



Kapitel 8

Insiderwissen: Die Ausnutzung des Interpreters

8.1 Die drei Rechnerebenen

Wir wollen uns im folgenden die internen Vorkehrungen anschauen, die vor dem Start eines Maschinenprogrammes getroffen werden. Der Rechner bewegt sich dabei in drei Betriebs-ebenen:

1. *Die Eingaberoutine*

Wenn Sie Ihren Rechner einschalten, wird ein umfangreiches Maschinenprogramm, die Eingaberoutine, gestartet. Sie überprüft ständig, ob eine Taste gedrückt ist. Sobald dann eine Taste betätigt wird, stellt diese Routine das entsprechende Zeichen auf dem Display dar und speichert es im Ein/Ausgabepuffer. Auch alle weiteren Eingaben werden auf diese Weise verwertet. Sobald jedoch die ENTER-Taste betätigt wird, legt der Rechner die Adresse der Eingaberoutine auf dem Stack ab und springt in den Interpreter.

2. *Der Interpreter*

Nun wird der Inhalt des Ein/Ausgabepuffers, in dem ja alle Zeichen stehen, die bis zu ENTER eingegeben wurden, mit Hilfe des X-Registers Zeichen für Zeichen ausgelesen. Bei Zeichenfolgen, die nicht mit einem Anführungszeichen als Text gekennzeichnet sind, sucht der Interpreter in seiner BASIC-Befehlsliste nach dem zugehörigen Maschinenprogramm. Der Interpreter richtet sich auch nach dem momentanen Betriebszustand (RUN oder PRO). Denn für jeden Betriebszustand existieren spezifische Befehle, die nur in diesem oder jenem Modus akzeptiert werden, z.B. LIST oder RUN. Nehmen wir einmal an, daß mit unserer Eingabe ein Maschinenprogramm aufgerufen werden soll. Der Rechner findet dann den BASIC-Befehl CALL und

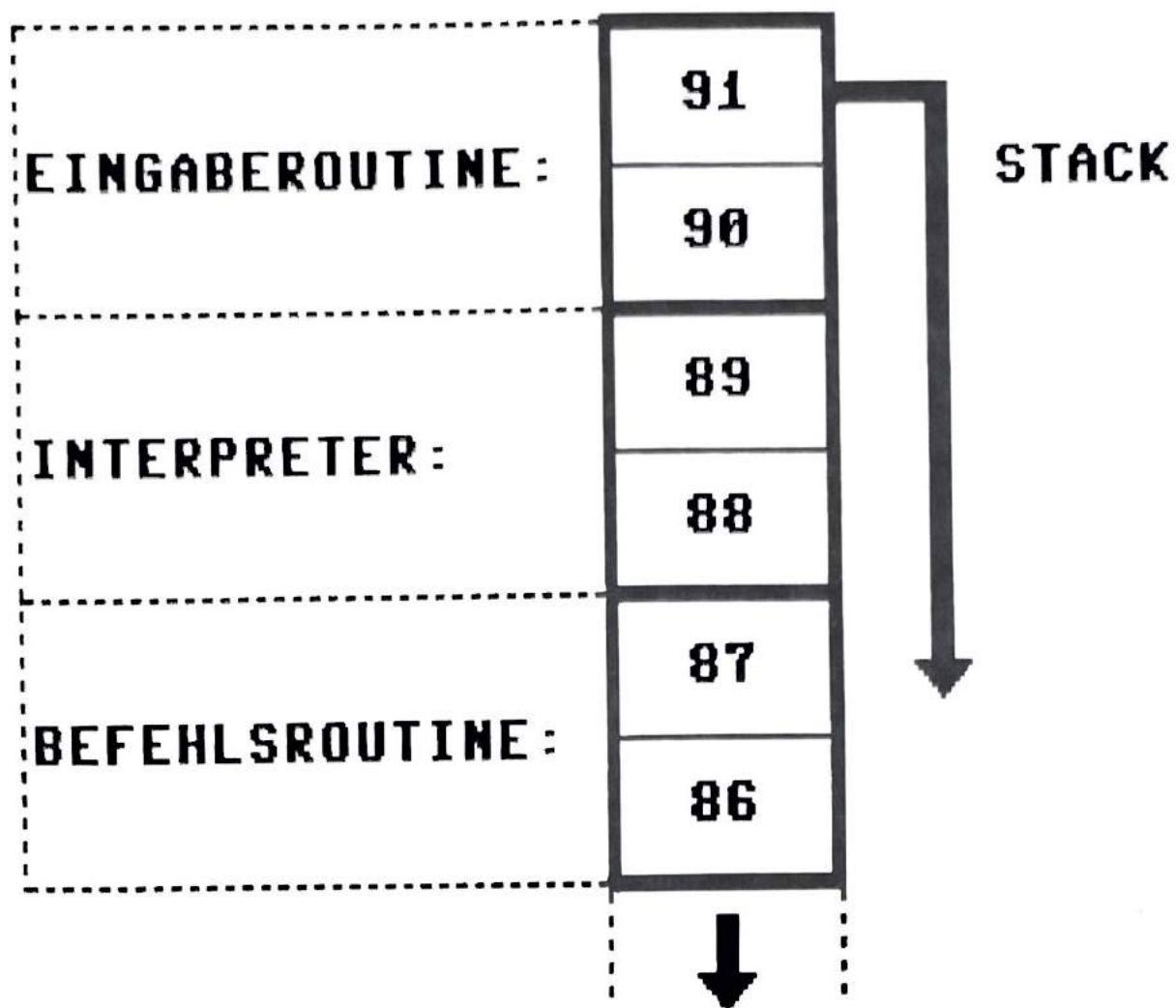
springt in das zugehörige Maschinenprogramm, die CALL-Routine. Die Rücksprungadresse für den Interpreter wird dabei wiederum im Stack gespeichert; die momentane Adresse des X-Registers (es zeigt auf das letzte Zeichen des CALL-Arguments) wird in einer speziellen Systemvariable (PC 14XX: 18101/02, PC 126X: 26165/66 und PC 1350: 28448/49) im Low/Highbyte-Format abgelegt.

3. *Die CALL-Routine*

Nun befinden wir uns in der CALL-Routine, einem ROM-Unterprogramm, das die nötigen Vorkehrungen trifft, um dann unser Maschinenprogramm aufzurufen. Dabei wird natürlich die Adresse der CALL-Routine auf dem Stack abgelegt.

Wird ein Maschinenprogramm im Direktmodus, also nicht in einem BASIC-Programm aufgerufen, so springt der Rechner aus dem Interpreter, der den Inhalt des Ein/Ausgabepuffers auswerten soll, direkt in die CALL-Routine. Wird ein Maschinenprogramm aus einem BASIC-Programm heraus aufgerufen, so führt der Interpreter den CALL-Befehl wie jeden andern BASIC-Befehl aus.

Die nächste Abbildung zeigt die auf dem Stack abgelegten Rücksprungadressen, die im Normalfall stets in dieser Reihenfolge abgelegt sind.



Gehen wir nun weiter. Das Maschinenprogramm wird aufgerufen; schließlich kommt der Rechner zum letzten Befehl des Maschinenprogramms, zum RTN-Befehl. Nun geschieht folgendes:

1. Rückkehr in die *CALL*-Routine

Der Rechner springt in die CALL-Routine zurück. Hier wird das Carry-Flag gelöscht und das X-Register mit seinem ursprünglichen Inhalt, also mit dem Wert, den das X-Register vor dem Aufruf des Maschinenprogramms hatte, geladen. Diese Adresse nimmt der Rechner aus den Systemvariablen, wo sie vor dem Aufruf des CALL-Befehls auch abgelegt wurde. Dann kehrt er in den Interpreter zurück.

2. Rückkehr in den Interpreter

Da das X-Register wieder seinen ursprünglichen Wert hat, kann der Interpreter mit dem Auslesen des BASIC-Programms oder des Ein-/Ausgabepuffers fortfahren. Sobald das BASIC-Programm zu Ende ist oder der Ein-/Ausgabepuffer fertig ausgelesen wurde, springt der Rechner zurück in die Eingaberoutine. Der Ein-/Ausgabepuffer wird gelöscht.

3. Rückkehr in die Eingaberoutine



Nun ist der Rechner wieder am Ausgangspunkt angelangt. Neue Eingaben können getätigt werden.

Diese Beschreibung zeigt, was alles geschieht, bis schließlich ein Maschinenprogramm ausgeführt wird. Durch bestimmte Tricks kann man dieses System nun soweit für eigene Zwecke einsetzen, daß einem damit ganz neue Möglichkeiten offen stehen. Das Hervorrufen eigener Fehlermeldungen oder das Programmieren kleiner Funktionen sind dann kein Problem mehr.

8.2 Fehlermeldungen, Funktionen und weitere Tricks

Platzreservation im internen RAM



Sie können durchaus eine Reihe von Speicherzellen des internen RAM für den Eigengebrauch reservieren. Diese reservierten Speicherzellen können dann als interner Speicher verwendet werden; z.B. bei der Übergabe von Zwischenergebnissen an ein zweites Maschinenprogramm.

Zur Reservierung von n Bytes versetzen wir einfach den Stackzeiger R um n Bytes nach unten. Danach müssen auch noch die drei Rücksprungadressen, die auf dem Stack abgelegt sind, um n Bytes nach unten kopiert werden.

Konkret sieht die ganze Sache so aus: Der Stackzeiger R wird mit seinem neuen Wert 86-n geladen. Dann werden die 6 Speicherzellen (86) bis (91) an die Adressen (86-n) bis (91-n) kopiert.

Nun können Sie über die Speicherzellen von (91-n+1) bis (91) frei verfügen.

Ein auf diese Weise einmal nach unten versetzter Stackzeiger wird vom Rechner nicht mehr nach oben versetzt. Deshalb ist diese Platzreservierung, einmal angelegt, bis auf weiteres aktiv. Sie wird erst mit dem Ausschalten des Rechners aufgehoben.

Das folgende Beispiel reserviert 5 Bytes des internen RAM:

Adresse	Mnemonic	Codes	Kommentar
...00	LIA 81	2,81	81=86-5
...02	STR	50	A->R
...03	LIP 86	18,86	86->P
...05	LIQ 81	19,81	81->Q
...07	LII 5	0,5	5->I
...09	EXW	9	(P)...(P+I)-> (Q)...(Q+I)
...10	RTN	55	Zurück ins BASIC

Erzeugen von Fehlermeldungen

Wenn Sie den Stackzeiger R um 2 von 86 auf 88 erhöhen, wird nicht mehr die Adresse der CALL-Routine, sondern die Adresse des Interpreters als Rücksprungadresse verwendet. Daraus folgt, daß weder das C-Flag gelöscht wird, noch das X-Register mit seinem ursprünglichen Inhalt (vor dem CALL-Befehl) geladen wird. Falls das X-Register in dem mit CALL aufgerufenen Maschinenprogramm nicht verändert wurde, fährt der Interpreter sofort mit dem Auslesen des Ein/Ausgabepuffers oder des BASIC-Programms weiter.

Wurde aber das X-Register im Programm benutzt und damit auch verändert, so muß es nun vor dem Rücksprung in den Interpreter unbedingt wieder mit seinem ursprünglichen Inhalt aus den Adressen 18101/02¹⁾ geladen werden.

Auch das C-Flag bleibt unverändert. Bei der Rückkehr aus einer Routine, die einen BASIC-Befehl bearbeitet (also z.B. CALL, PRINT) bedeutet ein gesetztes C-Flag stets, daß ein Fehler aufgetreten ist. Der zugehörige Fehlercode steht in der Speicherzelle 52 des internen RAM.

Wir können somit eigene Fehlermeldungen erzeugen! Dazu speichern wir zuerst den Fehlercode in dezimaler Form (0 bis 9) in der Speicherzelle 52 des internen RAM und laden den Stackzeiger R mit 88. Nun setzen wir das C-Flag und beenden das Maschinenprogramm mit RTN.

Bei einer vorgenommenen Platzreservierung im internen RAM müssen Sie den Stackzeiger R nicht mit 88, sondern mit 88-n laden, wobei mit n die Anzahl reservierter Speicherzellen gemeint ist.

Das folgende Beispiel erzeugt die Fehlermeldung ERROR 4:

Adresse	Mnemonic	Codes	Kommentar
...00	LP 52	180	52->P
...01	LIA 4	2,4	4->A
...03	EXAM	219	A<->(P)
...04	LIA 88	2,88	88->A
...06	STR	50	A->R
...07	SC	208	1->C
...08	RTN	55	

1) PC 126X: 26165/66, PC 1350: 28448/49

Funktionen

Für die Bearbeitung von Funktionen im Direktmodus, also nicht in einem BASIC-Programm, müssen wir den Stackzeiger R sogar um 4 erhöhen (also auf 90 setzen). Dadurch bleibt der Ein-/Ausgabepuffer auch nach der Rückkehr aus dem Maschinenprogramm erhalten: Das Display wird nicht gelöscht.

Dabei kann das X-Register einen beliebigen Wert haben.

Bei einer vorgenommenen Platzreservierung von Speicherzellen im internen RAM muß der Stackzeiger R nicht mit 90, sondern mit 90-n geladen werden.

Da der Ein-/Ausgabepuffer nicht gelöscht wird, können Sie im Direktmodus einfache Funktionen von Ihren eigenen Maschinenprogrammen abarbeiten lassen. Das nächste Beispiel zeigt, wie eine solche Funktion programmiert werden kann:

Addresse	Mnemonic	Codes	Kommentar
...00	LIA 90	2,90	90->A
...02	STR	50	A->R
...03	IX	4	Komma des CALL-Befehls überspringen
...04	IXL	36	Lesen der Zahl, die dem CALL-Befehl angehängt wurde
...05	ANIA 207	100,207	Umwandlung ASCII-Code-> Zahl
...07	SR	210	Akku/2 -> Akku. Wenn Zahl ungerade, ist C gesetzt.
...08	JPC 49321 ¹⁾	126,192,169	Ja: BEEP
...11	RTN	55	Nein: Zurück ins BASIC

1) PC 1421: 55539 (216,243), PC 48908 (191,12), PC 1350: 49430 (18,10)

Anmerkung: Das Programm läuft nur im Direktmodus (also nicht in BASIC-Programmen).

Diese Funktion überprüft eine Zahl X darauf, ob sie gerade (durch 2 teilbar) oder ungerade (nicht ohne Rest durch 2 teilbar) ist. Eingabeformat:

DX1) ...00,X (0<X<9)

Ist X ungerade, gibt der Rechner einen DEEP von sich, andernfalls kehrt er einfach ins BASIC zurück. Da der Ein/Ausgabepuffer erhalten bleibt, können Sie nach der Ausführung des Maschinenprogramms einfach mit dem Cursor zurückfahren und eine neue Zahl zwischen 0 bis 9 über die alte schreiben. Dann brauchen Sie nur noch die ENTER-Taste zu betätigen und schon wissen Sie auch von dieser neuen Zahl, ob sie gerade oder ungerade ist.

Im nächsten Kapitel werden Sie auch sehen, wie man die WAIT-Routine für die Umrechnung von dezimalen Zahlen (codiert nach BCD oder ASCII) in binäre Werte von 16 Bit Länge (High-/Lowbyte) einsetzt. Mit dieser Umrechnung können Sie dann sogar anspruchsvolle mathematische Funktionen programmieren.

8.3 Eigene BASIC-Befehle

Der vorhandene BASIC-Befehlssatz der SHARP PCs ist recht groß und für den täglichen Gebrauch reicht er auch gut aus. Trotzdem gibt es einiges, was recht unbefriedigend oder überhaupt nicht gelöst worden ist.

Es wäre also sehr nützlich, wenn man die BASIC-Befehlsbibliothek durch eigene Befehle ergänzen könnte.

Schön wäre z.B. auch ein PSAVE-Befehl für das Speichern von Teilprogrammen oder ein SOUND-Befehl für das Erzeugen von beliebigen Tönen. Oder Grafikbefehle für den PC 14XX oder

PC 126X, mit denen man die Grafikfähigkeiten dieser Rechner optimal ausnutzt.

Wir wollen uns zuerst einmal mit dem Aufbau eines solchen neuen BASIC-Befehls befassen:

Übergabe von Werten an ein Maschinenprogramm

Viele neue BASIC-Befehle brauchen noch weitere Informationen, die irgendwie an das Programm übermittelt werden müssen. Da ist es am besten, wenn man diese Werte an den CALL-Befehl anhängt.

Um diese angehängten Werte vom CALL-Befehl abzugrenzen, brauchen wir ein Trennzeichen. Dazu kommen folgende Zeichen in Frage: . ; : @ # % \$! ?

Es ist am einfachsten, wie beim POK.E-Befehl das Komma als Trennzeichen zu benutzen. Das Eingabeformat sieht somit folgendermaßen aus:

CALL Anfangsadresse,xx

Wie wir schon aus Kapitel 6.5 wissen, zeigt das X-Register immer auf das zuletzt ausgelesene Byte, also auf die letzte Ziffer der Anfangsadresse.

z.B.

CALL 52000,xx

Das X-Register zeigt auf die letzte Ziffer der Adresse 52000, also auf '0'.

Nun müssen wir nur noch das Trennzeichen (also das Komma) überspringen, und schon können wir das Argument (mit dem IXL-Befehl oder über die WAIT-Routine) auslesen.

Textargumente müssen in Anführungszeichen stehen, da der Rechner sonst versucht, den Text in einzelne BASIC-Befehle zu zerlegen. Und manchmal hat dies unerwünschte Folgen, wie der Fall unten deutlich zeigt:

:R

CALL 52000,ANDREA

wird nach ENTER zu:

CALL 52000,AND REA

Geben Sie deshalb ein:

CALL 52000,"ANDREA"

Während man Texteingaben direkt verwenden kann, muß man die meist im ASCII- oder BCD-Codes gespeicherten numerischen Argumente zuerst noch in verwertbare Binärzahlen umsetzen. Im nächsten Abschnitt sehen Sie, wie man dieses Problem elegant mit der WAIT-Routine löst.

Umrechnung von numerischen Argumenten

Es liegt auf der Hand, daß auch das Betriebssystem des PC numerische Ausdrücke, die entweder nach BCD oder ASCII codiert sind, in 16-Bit-Zahlen umwandeln muß. Wir benutzen deshalb für diese Umwandlung ein Maschinenprogramm, das schon im ROM des Rechners existiert: die WAIT-Routine.

Ist dabei ein numerisches Argument inkorrekt, kehrt der Rechner mit C=1 von der WAIT-Routine zurück, ansonsten mit C=0.

Das können wir geschickt ausnutzen: Sollen mehrere Argumente umgerechnet werden, trennt man sie einfach durch Kommas. Nach dem Aufruf der WAIT-Routine wird ein numerischer Ausdruck zwar umgerechnet, dann bemerkt der Rechner jedoch

das Komma und kehrt mit C=1 ins Hauptprogramm zurück. Auch das X-Register wird durch die WAIT-Routine etwas verändert: Es zeigt auf die letzte Stelle des Arguments. Und das wichtigste: In den Systemadressen für das WAIT-Intervall steht dann der 16-Bit-Wert (im Low/Highbyte-Format) des Arguments.

Nachdem wir diesen 16-Bit-Wert ausgewertet haben, überspringen wir das Komma wieder und rufen erneut die WAIT-Routine auf, um den nächsten numerischen Ausdruck umzuwandeln. Für die verschiedenen Rechner gelten folgende Systemadressen:

PC-Modell	WAIT-Routine	Argument des WAIT-Befehls (WAIT-Intervall)
PC 140X	49775	18149/18150
PC 1421	56000	18149/18150
PC 126X	49403	26341/26342
PC 1350	49946	28851/28852

Damit können wir also numerische Ausdrücke umwandeln. Sie können einen Befehl folgendermaßen eingeben:

CALL Anfangsadresse,Argument
oder: CALL Anfangsadresse,num. Ausdruck oder Variable

Jetzt gibt es da leider nur noch einen Schönheitsfehler: Bei der Rückkehr ins BASIC meldet sich der Rechner mit einer Fehlermeldung. Diese Fehlermeldung wird verursacht durch das Komma nach der Anfangsadresse des CALL-Befehls. Sie lässt sich jedoch mit einem kleinen Trick leicht unterdrücken.

Unterdrücken der Fehlermeldung

Es gibt zwei Möglichkeiten, 'sauber' ins BASIC zurückzukehren. Beide erlauben eine Verwendung des neuen BASIC-Befehls sowohl im Direktmodus als auch integriert in einem BASIC-Programm:

1. X-Register wurde nicht benutzt

Das X-Register wurde im Maschinenprogramm höchstens indirekt durch BASIC-Unterprogramme (WAIT-Routine) verändert. Da das X-Register dadurch aber keinen neuen, unzulässigen Wert erhält, können wir das Programm folgendermaßen beenden:

```
RC  
LIA88  
STR  
RTN
```

Das C-Flag wird gelöscht, um Fehlermeldungen zu verhindern, die bei einem gesetzten C-Flag auftreten. Da der Stackzeiger R auf 88 gesetzt wird, übernimmt der Rechner die Rücksprungadresse direkt aus dem X-Register.

2. X-Register wurde benutzt

Es wurde z.B. ein ROM-Unterprogramm (vgl. Anhang C) benutzt, das das X-Register verändert (vgl. Anhang C), oder man verwendet das X-Register für das eigene Programm. Im Grunde genommen können wir für diesen Fall die gleiche Methode wie oben anwenden, doch zuvor muß noch das X-Register seinen ursprünglichen Wert erhalten.

Dazu kopieren wir einfach die BASIC-Rücksprungadresse, die beim Start des Programms in den Adressen 18101/02¹⁾ abgelegt wurde, zurück ins X-Register. Dann suchen wir einen Doppelpunkt (Code 58) oder ein ENTER-Zeichen (Code 13). Sobald eines dieser beiden Zeichen gefunden wurde, dekrementieren wir das X-Register, setzen den Stackzeiger auf 88 und kehren ins BASIC zurück:

```
LIDP 181011)
LP 4
LII 2
MVWD
IXL
CPIA 13
JRZP 5
CPIA 58
JPNZM 8
DX
RC
LIA 88
STR
RTN
```

Beispiel:

Wir schreiben ein Maschinenprogramm, das den Inhalt der Standardvariable Z\$ mit dem Code eines beliebigen Zeichens füllt.

1) PC 126X: 26165, PC 1350: 28448

Adresse	Mnemonic	Codes	Kommentar
...00	IX	36	Komma überspringen
...01	CALL 49775 ¹⁾	120,194,111	Aufruf der WAIT-Routine
...04	LIDP 18149 ²⁾	16,70,181	18149 ²⁾ ->DP
...07	LDI	87	(DP)->A. Lowbyte des WAIT-Intervalls in den Akku laden
...08	LII 6	0,6	6->I
...10	LIDP 17873 ³⁾	16,69,209	178733)->DP
...13	FILD	31	A->(DP)...(DP+1)
...14	RC	209	C-Flag löschen
...15	LIA 88	2,88	88->A
...17	STR	50	A->R
...18	RTN	55	

Da es sich bei der auszuwertenden Zahl nur um eine 8-Bit-Zahl handeln soll, brauchen wir nur das Lowbyte des WAIT-Intervalls weiterzuverwerten (das Highbyte beträgt ja auf jeden Fall 0).

Das numerische Argument kann in beliebiger Form eingegeben werden:

oder: CALL ...00,91
 oder: CALL ...00,50+41
 oder: CALL ...00,X (X = 91)
 oder: CALL ...00,ASC "Z"

Fügen Sie unseren neuen BASIC-Befehl probehalber auch in ein kleines Programm ein, z.B.

1) PC 1421: 56000 (218,192), PC 126X: 49403 (192,251), PC 1350: 49946 (195,26)

2) PC 126X: 26341 (102,229), PC 1350: 28851 (112,179)

3) PC 1421: 17793 (69,129), PC 126X: 25857 (101,1), PC 1350: 27697 (108,49)

```
500 INPUT "ZEICHEN :";A$; X$="";CALL 100,ASC A$;
PRINT Z$;END
```

Eine eigene Namensgebung der neuen Befehle ist leider nicht möglich, da der Interpreter im ROM lokalisiert ist und deshalb keine Manipulationen zuläßt.

Für weitere Beispiele verweisen wir Sie auf das Kapitel 9.5 ("Utilities"). Dort sind viele eigene BASIC-Befehle beschrieben, die Ihnen vielleicht auch Anregungen für weitere neue BASIC-Befehle geben können.



Teil 3: Soft- und Hardware

Kapitel 9

Software: Den PC ganz ausnutzen

Wir stellen Ihnen in diesem Kapitel eine Software-Sammlung zur Verfügung. Diese Programme sollen Ihnen einerseits als Musterprogramme, andererseits aber auch als nützliche Anwendungen dienen. Dieses breite Spektrum von Programmen gibt Ihnen auch einen Überblick über die vielen Möglichkeiten, die Ihnen der PC bietet.

9.1 Mathematik

Quadratische Gleichung

Bei solchen kurzen, häufig gebrauchten Programmen zeigt es sich, wie nützlich ein Computer sein kann. Vielleicht haben Sie sich auch schon darüber geärgert, wie mühsam es doch ist, diese Gleichung für neue Parameter immer wieder neu auszuwerten.

Das vorliegende Programm ersetzt eine solche mühsame Rechnerei. Es besitzt zudem eine komfortable Fehleranzeige.

Die Formel:

$$A \cdot X^2 + B \cdot X + C = 0$$

$$X_1 = \frac{SQR(B^2 - 4 \cdot A \cdot C) - B}{2 \cdot A}$$

$$X_2 = \frac{-SQR(B^2 - 4 \cdot A \cdot C) - B}{2 \cdot A}$$

Nach der Eingabe der Parameter A, B und C erscheint zuerst die erste Lösung X1, dann die zweite Lösung X2.

FUER ALLE RECHNER:

```
10: INPUT "A= ";AX;"B= "
    ;BX;"C= ";CX
15: IF AX=0 OR BX=0
    PRINT "FALSCHE EINGA
    BE!":GOTO 10
20: DX=BX*BX-4*AX*CX:IF
    DX<0 PRINT "DISKR. N
    EGATIV!":GOTO 10
25: X1=(+DX-BX)/2/AX:X2=
    (-DX-BX)/2/AX
30: PAUSE "X1=...":PRINT
    X1:PAUSE "X2=...":
    PRINT X2:END
```

Primzahltest

Dieses ebenso kurze wie auch einfache Programm zeigt, ob es sich bei einer eingegebenen Zahl um eine Primzahl handelt oder nicht.

z.B. Zahl: 31
Ergebnis: 31 ist eine Primzahl!

oder Zahl: 45
Resultat: 45 ist keine Primzahl

FUER ALLE RECHNER:

```
10: INPUT "ZAHL: ";Z:X=
    INT Z:IF Z<4 AND Z>
    0 THEN 30
15: IF Z/2- INT (Z/2)=0
    THEN 25
20: FOR I=3 TO X STEP 2:
    IF Z/I- INT (Z/I)<>0
    NEXT I:GOTO 30
```

```

25:PAUSE Z;" IST...":
    PRINT "KEINE PRIMZAH
    L":END
30:PAUSE Z;" IST...":
    PRINT "EINE PRIMZAHL
    !":END

```

Berechnung eines Wochentags

Dieses Programm enthält die Formel zur Berechnung des Wochentags (Montag bis Sonntag) eines beliebigen Datums. Sie können ein beliebiges Datum ab 1600 verwenden. Damit können Sie z.B. herausfinden, an welchem Tag Sie geboren wurden.

z.B. 4.3.1988
 Resultat: Freitag

FUER ALLE RECHNER:

```

10:"D" WAIT :INPUT "TAG
= ";T:INPUT "MONAT=
";M:INPUT "JAHR= ";J
15:I= INT (J/100):J=J-I
*100:IF MK3 LET M=M+
12:J=J-1
20:Z=(T+ INT (2.6*(M+1)
)+ INT (1.25*J)+ INT
(I/4)-2*I-1)
25:X=Z/7:Y=Z- INT X*7
30:IF Y=0 PRINT "SONNTA
G"
35:IF Y=1 PRINT "MONTAG
"
40:IF Y=2 PRINT "DIENST
AG"
45:IF Y=3 PRINT "MITTWO
CH"
50:IF Y=4 PRINT "DONNER
STAG"
55:IF Y=5 PRINT "FREITA
G"

```

```
60:IF Y=6 PRINT "SAMSTA  
G"  
65:GOTO 10
```

Anzahl Tage zwischen zwei Daten

Das folgende Programm funktioniert ähnlich wie das Programm 'Wochentag'. Während Sie mit 'Wochentag' für ein beliebiges Datum den zugehörigen Wochentag berechnen können, so können Sie mit diesem Programm nun die Anzahl Tage zwischen zwei Daten berechnen. Dieses Programm kommt vor allem auch für finanzmathematische Programme in Betracht, da dieses Problem hier oft auftritt (z.B. in der Zinsrechnung).

Gestartet wird das Programm mit RUN.

Beispiel:

```
Eingabe: 1. Datum: Tag: 12  
Monat: 3  
Jahr: 1986  
2. Datum: Tag: 25  
Monat: 7  
Jahr: 1987
```

```
Ausgabe: Anzahl Tage  
zwischen den  
beiden Daten: 500
```

FUER ALLE RECHNER:

```
10:DIM T(2),M(2),J(2)  
15:FOR I=1 TO 2:PAUSE I  
;" DATUM...":INPUT "  
TAG= ";T(I),"MONAT="  
";M(I),"JAHR= ";J(I)  
:NEXT I  
20:T=T(1):M=M(1):J=J(1)  
:GOSUB 30:N=A:T=T(2)  
:M=M(2):J=J(2):GOSUB  
30:N=A-N
```

```

25:PRINT "ANZ. TAGE: ";
  STR$ N:GOTO 15
30:RESTORE 60:FOR I=1
   TO M:READ A:NEXT I:A
   =A+J*365+ INT (J/4)*
   T+1- INT (J/100)*
   INT (J/400)
35:IF INT (J/4)<>J/4
   RETURN
40:IF INT (J/400)=J/400
   THEN 50
45:IF INT (J/100)=J/100
   THEN 55
50:IF M>2 RETURN
55:A=A-1:RETURN
60:DATA 0,31,59,90,120,
    151,181,212,243,273,
    304,334

```

Rationale Zahlen in Brüche umwandeln und Brüche kürzen

Hier handelt es sich um ein komplizierteres Programm. Sämtliche rationalen Zahlen können damit in ihre Brüche zurückgeführt werden. Unter rationalen Zahlen versteht man Zahlen, bei denen sich eine bestimmte Zahlenfolge, die sogenannte 'Periode', bis ins Unendliche wiederholt.

Z.B.: $3.296296296\dots = 3.\overline{296} = \frac{89}{27}$

Das Programm zur Umrechnung rationaler Zahlen in Brüche wird mit DEF B gestartet. Wie wird eine solche rationale Zahl eingegeben?

Wir wollen das gleich am letzten Beispiel erklären: Die rationale Zahl $3.296296296\dots$ soll in einen Bruch zurückverwandelt werden.

Starten Sie das Programm und geben Sie ein:

3.P296 (ENTER)

Das P kennzeichnet den Anfang der Periode. 'P' muß deshalb immer vor der ersten Ziffer der Periode stehen. Leider dauert die Umrechnung einige Zeit, da der errechnete Bruch auch noch gekürzt werden muß.

Resultat in der Anzeige: 89 27, das heißt: 89/27

Ein paar weitere Beispiele zum Ausprobieren:

18.3838383 = 18.3

Eingabe: 1P8.3 (ENTER)

Resultat: 1820/99

0.95833333 = 0.9583

Eingabe: 0.958P3 (ENTER)

Resultat: 23/24

121.810810 = 121.810

Eingabe: 121.P810 (ENTER)

Resultat: 4507/37

Anmerkung: Falls Dezimalpunkt und Anfang der Periode an der gleichen Stelle stehen, so muß zuerst der Dezimalpunkt eingeben werden, erst dann das P, das den Anfang der Periode kennzeichnet; also:

121.P810 und nicht etwa: 121P.810

Brüche kürzen

Das Programm zur Umwandlung rationaler Zahlen verwendet auch ein Unterprogramm zum Kürzen von Brüchen. Praktisch alle aus rationalen Zahlen errechneten Brüche müssen nämlich zuerst noch gekürzt werden. Dieses Teilprogramm 'Brüche kürzen' kann separat mit DEF K aufgerufen werden.

Einige Beispiele zum Ausprobieren:

Zähler: 1400	56
—————	Resultat: —
Nenner: 1025	41

Zähler: 4005	45
—————	Resultat: —
Nenner: 5251	59

Wollen Sie nur das Programm 'Brüche kürzen' benutzen (ohne das Programm zur Umwandlung rationaler Zahlen), so geben Sie einfach nur die Programmzeilen nach 'K' ein (ab Zeile 90).

FÜR ALLE RECHNER:

```

10:"B":WAIT :CLEAR :DIM
A$(0)*22:I=1:PRINT "
RAT.ZAHL - BRUCH"
15:INPUT "ZAHL= ";A$(0)
20:IF A$(0)="" BEEP 1:
PRINT "KEINE EINGABE
":GOTO 15
25:X$=MID$(A$(0),I,1):
IF X$="P" LET J=1:
GOTO 40
30:I=I+1:IF I=LEN (A$(0)
)) BEEP 1:PRINT "P
VERGESSEN":GOTO 10
35:GOTO 25
40:X$=MID$(A$(0),J,1):
IF X$=". " THEN 50
45:J=J+1:IF J>LEN (A$(0))
THEN 40

```

```

50:AA$=LEFT$ (A$(0),I-1
):BB$=RIGHT$ (A$(0),
LEN (A$(0))-I)
55:AA=VAL (AA$):BB=VAL
(BB$)
60:IF I<J LET AA=AA*10^
(J-LEN (AA$)-2):K=1:
GOTO 70
65:BB=BB*10^(-(LEN (A$(0))-J-1)):K=0
70:NQ=1-10^(K-LEN (BB$))
):ZQ=AA*NQ+BB
75:IF ZQ+1- INT (ZQ+1)<
>0 LET ZQ=ZQ*10:NQ=N
Q*10:GOTO 75
80:IF NQ+1- INT (NQ+1)<
>0 LET ZQ=ZQ*10:NQ=N
Q*10:GOTO 80
85:GOTO 110
90:"K":WAIT :PRINT "BRU

```

```

      ECHE KUERZEN"
95: INPUT "ZAEHLER= "; ZQ
    : IF ZQ<1 BEEP 1:
    PRINT "ZAEHLER MUSS
    > 0": GOTO 95
100: INPUT "NENNER= "; NQ:
    IF NQ<1 BEEP 1: PRINT
    "NENNER MUSS > 0":
    GOTO 100
105: IF ZQ=0 OR NQ=0 BEEP
    1: PRINT "KEINE EINGA
    BE": GOTO 95
110: K=0: IF ZQ>NQ LET DD=
    NQ: NQ=ZQ: ZQ=DD: K=1
115: DD=2: QQ=1: L=1: XQ=ZQ:
    YQ=NQ
120: IF DD>ZQ THEN 145
125: IF DD=3 LET L=2
130: IF ZQ- INT (ZQ/DD)*D
    D<>0 LET DD=DD+L:
    GOTO 120
135: IF NQ- INT (NQ/DD)*D
    D<>0 LET DD=DD+L:
    GOTO 120
140: ZQ=ZQ/DD: NQ=NQ/DD: QD
    =QQ*DD: IF QD<>XQ
    THEN 130
145: IF K=1 LET DD=NQ: NQ=
    ZQ: ZQ=DD
150: BEEP 1: PRINT ZQ, NQ:
    END

```

Lineare Gleichungssysteme mit 2 bis 5 Unbekannten

Ein lineares Gleichungssystem könnte z.B. so aussehen:

	Parameterspalte			Resultatspalte			
	A	B	C				
1. Reihe:	1x	+	3y	+	5z	=	1
2. Reihe:	2x	+	15y	-	2z	=	3
3. Reihe:	-1x	+	2y	-	10z	=	-2

Lineare Gleichungssysteme kann man natürlich 'von Hand' lösen.
Wesentlich einfacher geht es aber mit diesem Programm, das die Resultate über Determinanten berechnet.

Maximal können lineare Gleichungssysteme mit 5 Unbekannten gelöst werden.

Das Programm wird mit DEF L gestartet. Zuerst wird die Art des Gleichungssystems abgefragt. Geben Sie einfach die Zahl der vorkommenden Unbekannten ein (2-5 Unbekannte). Bei unserem Beispiel oben wären das also 3 Unbekannte.

Nach dem Abschluß dieser ersten Eingabe mit ENTER erscheint 'Parameterspalte, 1. Reihe' auf der Anzeige. Nun geben Sie der Reihe nach die Parameter der ersten Reihe ein. Danach werden die Parameterwerte der 2. Reihe abgefragt und schließlich die Werte für die 3. Reihe.

Wenn Sie alle Werte der Parameterspalte eingegeben haben, erscheint 'Resultatspalte' auf der Anzeige. Sie müssen nun die verschiedenen Resultate der Reihe nach eingeben (von oben nach unten).

Nach dieser Eingabe werden die Eingaben ausgewertet (es kann recht lange dauern!). Die Resultate für die Unbekannten werden der Reihe nach angezeigt. Zudem sind sie in den Standard-Variablen A bis E abgelegt.

Wenn Sie das Programm eingegeben haben, prüfen Sie dessen Funktionstüchtigkeit mit dem folgenden Testgleichungssystem:

	Parameterspalte					Resultatspalte	
	A	B	C	D	E	=	
1. Reihe:	0	0	3	4	6	=	1
2. Reihe:	10	12	-10	20	30	=	2
3. Reihe:	-5	-6	9	10	11	=	3
4. Reihe:	500	600	1500	100	12	=	4
5. Reihe:	3	12	16	17	18	=	5

Richtige Resultate:

A = -0.356
B = 0.180
C = 0.064
D = 0.091
E = 0.073

FUER ALLE RECHNER:

```
10:"L" WAIT 60:PRINT "L  
INEARES":PRINT "GLEI  
CHUNGSSYSTEM"  
15:CLEAR :INPUT "ORDNUN  
G: ";N:IF N>5 OR N<2  
THEN 15  
20:DIM B(N,N),Y(N),Z(N)  
25:FOR I=1 TO N:PRINT "  
PARAMETERSPALTE":  
PRINT I;" REIHE":FOR  
J=1 TO N:INPUT B(I,J)  
:NEXT J:NEXT I  
30:GOSUB 55  
35:GOSUB 110:Q=X:FOR I=  
1 TO N:FOR J=1 TO N  
40:Z(J)=B(J,I):B(J,I)=Y  
(J):NEXT J:GOSUB 55:  
A(I)=X/Q  
45:FOR J=1 TO N:B(J,I)=  
Z(J):NEXT J:NEXT I  
50:WAIT :PRINT "RESULTA  
TE...":FOR I=1 TO N:  
PAUSE CHR$ (64+I)+"  
...":PRINT A(I):NEXT  
I:END  
55:ON N-2 GOTO 65,70,75  
60:X=B(1,1)*B(2,2)-B(1,  
2)*B(2,1):RETURN  
65:L=1:M=1:O=2:P=3:  
GOSUB 95:X=F:RETURN  
70:S=1:R=1:T=2:U=3:V=4:  
GOSUB 85:X=G:RETURN  
75:R=2:S=2:T=3:U=4:V=5:  
X=0:W=1:FOR H=1 TO 5  
:GOSUB 85  
80:X=X+W*B(H,1)*G:W=-W:  
A(18+H)=H:NEXT H:  
RETURN  
85:M=R+1:L=T:O=U:P=V:  
GOSUB 95:G=B(S,R)*F:  
L=S:GOSUB 95:G=G-B(T  
,R)*F:O=T  
90:GOSUB 95:G=G+B(U,R)*  
F:P=U:GOSUB 95:G=G-B  
(V,R)*F:RETURN  
95:F=B(P,M+2)*(B(0,M+1)  
*B(L,M)-B(0,M)*B(L,M  
+1))  
100:F=F+B(0,M+2)*(B(P,M)  
*B(L,M+1)-B(P,M+1)*B  
(L,M))  
105:F=F+B(L,M+2)*(B(0,M)  
*B(P,M+1)-B(P,M)*B(0  
,M+1)):RETURN  
110:PRINT "RESULTATSPALT  
E":FOR I=1 TO N:  
INPUT Y(I):NEXT I:  
RETURN
```

9.2 Finanzmathematik

Wir stellen Ihnen hier ein paar einfache Programme vor, die Ihnen in ihren finanziellen Angelegenheiten vielleicht nützlich sein können. Es erübrigt sich darauf hinzuweisen, daß diese und weitere Funktionen beim PC 1421 schon als feste Programme im ROM vorhanden sind.

Erklärung der finanzmathematischen Begriffe

- E: allgemein Zahlung (Einzahlung, Auszahlung, Rate)
Z: Zinssatz in %
K: allgemein Kapital (Anfangskapitel, Endkapital, Schuld)

Vor- und nachschüssig

Unter vorschüssig versteht man eine Zahlung, die am Anfang einer bestimmten Zeitperiode erfolgt und darum in dieser Zeit dem Empfänger noch Zins einbringt. Bei der üblichen Zeitperiode von einem Jahr muß die Zahlung folglich am 1. Januar erfolgen.

Unter nachschüssig versteht man dagegen eine Zahlung, die am Ende einer bestimmten Zeitperiode erfolgt. Sie bringt dem Empfänger keinen Zins mehr ein. Bei einer Zeitspanne von einem Jahr muß die Zahlung folglich am 31. Dezember erfolgen.

Sparerformel

Durch die Sparerformel kann das Kapital berechnet werden, das aus der regelmäßigen, jährlichen Einzahlung eines bestimmten Geldbetrages E bei einem Zinssatz Z nach J Jahren resultiert. Ein Anfangskapital kann angegeben werden, das sich während dieser Zeit auch vermehrt.

Beispiel:

Jedes Jahr zahlt jemand mit einem Anfangskapital von DM 10000.- am Jahresende DM 1000.- auf sein Konto ein. Wie groß ist sein Kapital nach der 10. Einzahlung (am Jahresanfang) bei einem Zinssatz von 5 Prozent?

Antwort: DM 28866,83. Hätte er die DM 1000.- jeweils schon am Jahresanfang einzuzahlt (vorschüssig), so beträfe er heute DM 29495,73.

Rentnerformel

Die Rentnerformel berechnet das pure Gegenteil der Sparerformel: Wieviel Geld bleibt nach J Jahren übrig, wenn jedes Jahr der Betrag E ausbezahlt wird, das Anfangskapital K und der Zinssatz Z beträgt?

Beispiel:

Ein Vater zahlt seinem studierenden Sohn während 4 Jahren jeweils am Jahresanfang (vorschüssig) DM 2500.- aus. Er hat ein Vermögen von DM 20000.-, und der Zinssatz beträgt 5 Prozent. Wie groß ist das Vermögen des Vaters nach 4 Jahren?

Antwort: DM 12996,04

Tilgungsplan

Der Tilgungsplan ist grundsätzlich nachschüssig, eine Rate wird ja am Schluß einer Zeitperiode zurückbezahlt. Mit ihm kann der jährlich zu bezahlende Betrag E (Annuität) errechnet werden, der nötig ist, um eine Schuld K in J Jahren und bei einem Zinssatz von Z Prozent abzuzahlen.

Beispiel:

Herr Y hat eine Hypothek von DM 100000.- Er muß sie zu einem Zinssatz von 6 Prozent verzinsen und möchte gern innerhalb von 10 Jahren seine ganze Schuld zurückbezahlt haben. Wieviel muß er am Ende jedes Jahres bezahlen, damit er sein Ziel erreicht?

Antwort: DM 13586,79

Amortisation

Mit dieser Formel kann berechnet werden, in wievielen Jahren sich eine Investition K bei jährlichen Mehreinnahmen E bei einem Zinssatz von Z Prozent amortisiert hat.

Beispiel:

Die Firma F. hat für eine neue Produktionsanlage rund DM 50000.- investiert. Jedes Jahr werden dadurch Uukosten in der Höhe von DM 10000.- eingespart. Der Zinssatz liegt bei 6 Prozent. Nach wie vielen Jahren hat sich die Anlage amortisiert?

Antwort: Nach 6,12 Jahren

Barwert einer Rente

Durch dieses Programm kann der heutige Wert einer Ratenzahlung E während T Jahren bei einem Zinssatz von Z Prozent ermittelt werden.

Beispiel:

Herr X kauft sich einen Fernseher. Er möchte ihn mit 4 Jahresraten von je DM 500.- bezahlen. Wieviel bezahlt er für diesen 'Service', wenn der Zinssatz bei 6 Prozent und der Neupreis des Farbfernsehers bei DM 1700.- liegt?

Antwort: DM 136,50. Der Barwert dieser Ratenzahlung beträgt DM 1836,50.

Zinssatz

Mit diesem Programm kann ermittelt werden, welcher Wert aus einem zu einem Zinssatz Z angelegten Anfangskapital E nach J Jahren resultiert.

Beispiel:

Frau A zahlt einen Grundstock von DM 10000,- auf ihr neu eröffnetes Konto ein. Welchen Betrag wird sie in 5 Jahren abheben können, wenn der Zinssatz 4,5 Prozent beträgt und der Zins automatisch gutgeschrieben und wieder verzinnt wird?

Antwort: DM 12461,82

FÜR ALLE RECHNER:

```

101;"S":PAUSE "SPARERFOR
    MEL...":GOSUB 160
15:INPUT "ANFANGSKAP.:";
    "IK,"EINZAHLUNG:";IE
    ;"JAHRE:";IJ;"ZINS I
    N %";IZ
20:GOSUB 180:E=K*R*A/J*B-
    E*B*(R^J-1)/(R-1):VS
    =A+B:MS=A+B/R
25:PAUSE "ENDKAPITAL=..";
    ."ION H GOTO 30,55
30:PRINT MS:END
35:PRINT VS:END
40;"V":PAUSE "RENTHERFO
    RMEL...":GOSUB 160
45:INPUT "ANFANGSKAP.:";
    "IK,"AUSZAHLUNG:";IE
    ;"JAHRE:";IJ;"ZINS I
    N %";IZ
50:GOSUB 180:A=K*R*A/J:B-
    E*B*(R^J-1)/(R-1):VS
    =A-B:MS=A-B/R

```

```

55:PAUSE "ENDKAPITAL=..";
    ."ION H GOTO 60,65
60:PRINT MS:END
65:PRINT VS:END
70;"G":PAUSE "TILGUNGSP
    LAN...:"
75:INPUT "SCHULD:";IK;""
    JAHRE:";IJ;"ZINS IN
    %";IZ
80:GOSUB 180:E=K*R*A/J*(R-
    -1)/(R^J-1)
85:PAUSE "AMMUTAET=...":
    :"PRINT E:END
90;"A":PAUSE "AMORTISAT
    ION...:"
95:INPUT "SCHULD:";IK;""
    ABZAHLUNG:";IE;"ZINS
    IN %";IZ
100:GOSUB 180:J=LOG (E/(
    E-K*(R-1)))/LOG R
105:PAUSE "JAHRE=...":
    PRINT J:END
110;"B":PAUSE "BARWERT..";
    ."":GOSUB 160

```

```

115:INPUT "RATE: ";E,"JA
      HRE: ";J,"ZINS IN %:
      ";Z
120:GOSUB 180:A=E*(R^J-1
      )/(R-1):VS=A/R^(J-1)
      :NS=A/R^J
125:PAUSE "BARWERT=...":
      ON N GOTO 130,135
130:PRINT NS:END
135:PRINT VS:END
140:"Z":PAUSE "ZINSESZIN
      S... "
145:INPUT "ANFANGSKAP.:
      ";E,"JAHRE: ";J,"ZIN
      S IN %: ";Z
150:GOSUB 180:K=E*R^J
155:PAUSE "ENDKAPITAL=..
      .":PRINT K:END
160:PAUSE "VOR- ODER... "
      :INPUT "NACHSCHUESSI
      G: ";N$
165:IF LEFT$(N$,1)="N"
      LET N=1:RETURN
170:IF LEFT$(N$,1)="V"
      LET N=2:RETURN
175:GOTO 160
180:R=1+Z/100:RETURN

```

Tastaturbelegung

```

DEF S: Sparerformel
DEF V: Rentnerformel
DEF G: Tilgungsplan
DEF A: Amortisation
DEF B: Barwert einer Rente
DEF Z: Zinseszins

```

9.3 Datenverarbeitung

Die folgenden Programme zeigen Ihnen, wie man auf dem PC Daten verarbeiten kann. Da man mit dem PC keine Dateien bearbeiten kann (wie auf größeren Systemen), sind die Daten in Feldvariablen abgelegt und können von da auf Band gespeichert werden.

Wörterlernprogramm Wordstar

'Wordstar', in der Personal-Computer-Welt auch anderweitig bekannt, steht hier für ein komfortables Programm zum Lernen der Wörter einer Fremdsprache. Wordstar bietet nicht nur Funktionen zum Speichern von Wörtern und deren Übersetzung, sondern auch sinnvolle Methoden zum Üben des neuen Vokabulars. Das vorliegende Programm wurde für das Lernen von englischen Wörtern entworfen. Natürlich können Sie auch andere Sprachen (z.B. Latein) damit lernen.

 Vielleicht fragen Sie sich, welchen Nutzen es hat, auf einem kleinen Taschencomputer das Vokabular einer Fremdsprache aufzubessern oder Adressen zu verwalten. Der Vorteil liegt vor allem in der Mobilität des Taschencomputers. Da er sehr handlich ist, kann er überall mitgenommen werden. Sie können so z.B. in der Straßenbahn, im Zug oder auch bei sonstigen Wartezeiten etwas sinnvolles mit der Zeit anfangen!

Bedienungsanleitung

- Starten Sie das Programm mit RUN.
- Auf der Anzeige erscheinen die einzelnen Menüpunkte:

1 = EINGEBEN	5 = DRUCKEN
2 = UEBERSETZEN	6 = SPEICHERN
3 = LISTEN	7 = LADEN
4 = UEBEN	E = ENDE

 Die verschiedenen Teilprogramme werden mit einem Tastendruck aufgerufen. Damit der Rechner reagiert, müssen Sie die Taste ca. 1 Sekunde lang drücken.

1: EINGEBEN

Als erstes erfahren Sie, wie viele Wörter bereits eingegeben worden sind. Das Maximum der Wörter hängt vom verfügbaren freien Speicher ab; beim PC 1401 sind es z.B. maximal 90 Wörter und deren Übersetzung.

Zuerst wird das englische Wort (in der Anzeige steht: ENG. WORT:) eingegeben, darauf das deutsche (DEU. WORT:). Ein Wort darf aus maximal 11 Buchstaben bestehen. Um ins Hauptmenü zurückzukehren, drücken Sie die Taste 'N' (Nein), wenn Sie sich in der Anzeige WEITERE EINGABEN (J/N)? befinden. Sollen noch weitere Wörter gespeichert werden, geben Sie einfach 'J' (Ja) ein.

2: UEBERSETZEN

Wollen Sie vom Englischen ins Deutsche übersetzen? Oder lieber umgekehrt? Sie können wählen:

- E: Englisch - Deutsch
- D: Deutsch - Englisch

Ihre Eingabe wird quittiert. Geben Sie daraufhin das Wort ein, dessen Übersetzung Sie gerne wüßten. Falls sich das gesuchte Wort im Speicher befindet, wird die Übersetzung angezeigt, andernfalls erscheint "NICHT GEFUNDEN".

3: LISTEN

Mit dieser Funktion können sämtliche Wörter, die sich im Speicher befinden, gelistet werden. Die Listgeschwindigkeit wird über die Tasten 1, 2 oder 3 angewählt:

- 1 - 3
- schnell - langsam

Das Listen kann durch Drücken der Taste S unterbrochen werden.

4: UEBEN

Damit haben Sie die Möglichkeit, sich vom Rechner abfragen zu lassen. Auch hier können Sie wählen, ob der Rechner in deutscher oder englischer Sprache abfragen soll. Lediglich die Übersetzung muß dann eingegeben werden. Alle Wörter werden zufällig ausgewählt.

5: DRUCKEN

Wenn Sie einen Drucker besitzen, so können Sie sich mit dieser Funktion eine Liste der gespeicherten Wörter ausdrucken lassen.

6: SPEICHERN

Alle Wörter werden auf Band gesichert. Natürlich muß das Cassettengerät vor dem Anwählen dieser Funktion auf die Speicherung vorbereitet werden.

7: LADEN

Einmal gespeicherte Wörter können auch jederzeit wieder geladen werden. Nach dem Drücken der Taste 7 wartet der Rechner auf Daten vom Cassettengerät, um die neuen Wörter zu laden.

E: ENDE

Das Programm wird abgeschlossen.

Warnung: Starten Sie das Programm nie mit RUN, wenn Sie schon Wörter gespeichert haben. Durch RUN werden alle Feldvariablen gelöscht und damit alle Ihre gespeicherten Wörter. Es ist darum zu empfehlen, jeweils gerade nach der Eingabe der Wörter eine Sicherungskopie auf Band zu machen.

Statt mit RUN können Sie das Programm jedoch mit DEF A starten. Alle Wörter bleiben dann erhalten. Wenn sich hingegen noch keine Wörter im Speicher befinden, muß das Programm mit RUN gestartet werden, damit die Feldvariablen überhaupt dimensioniert werden.

Das Programm 'Wordstar' ist so ausgelegt, daß es den gesamten Speicher Ihres PCs beansprucht. Sie sollten darum im Direktmodus nur noch Standardvariablen verwenden, denn diese benötigen keinen Platz im Programmspeicher, der fast bis auf das letzte Byte gefüllt ist.

PC 14XX:

```

10:CLEAR :M= INT (MEM /
23):DIM N$(M)*11:DIM
M$(M)*11:Q=-1
15:"A":PAUSE "*** WORDS
TAR ***":WAIT 40
20:PRINT "1= EINGEBEN":
GOSUB 30:PRINT "2= U
EBERSETZEN":GOSUB 30
:PRINT "3= LISTEN":
GOSUB 30:PRINT "4= U
EBEN":GOSUB 30
25:PRINT "5= DRUCKEN":
GOSUB 30:PRINT "6= S
PEICHERN":GOSUB 30:
PRINT "7= LADEN":
GOSUB 30:PRINT "E= E
NDE":GOSUB 30:GOTO 2
0

```

```

30:A$$=INKEY$
35:IF A$$="1" WAIT 80:
      GOSUB 85
40:IF A$$="2" GOSUB 100
45:IF A$$="3" GOSUB 140
50:IF A$$="4" GOSUB 175
55:IF A$$="5" GOSUB 225
60:IF A$$="6" GOSUB 235
65:IF A$$="7" GOSUB 240
70:IF A$$="E" END
75:WAIT 40:RETURN
80:WAIT 30
85:Q=Q+1:IF Q=M PAUSE "
SPEICHER VOLL!":Q=Q-
1:RETURN
90:PRINT Q;" WOERTER...
":PRINT "GESPEICHERT
":PRINT "NEUE EINGA
BE..."
```

```

95: INPUT "ENG. WORT: ";  

    N$(Q), "DEU. WORT: ";  

    M$(Q): GOSUB 245: GOTO  

    80  

100: GOSUB 265: GOSUB 275  

105: INPUT "WORT: "; N$(M)  

    : WAIT  

110: FOR X=0 TO Q: ON J  

    GOTO 115, 125  

115: IF N$(M)=N$(X) PRINT  

    M$(X): GOTO 135  

120: GOTO 130  

125: IF N$(M)=M$(X) PRINT  

    N$(X): GOTO 135  

130: NEXT X: BEEP 1: PAUSE  

    "NICHT GEFUNDEN!"  

135: GOSUB 245: GOTO 105  

140: GOSUB 265: INPUT "TEM  

    P0 (1-3): "; A$  

145: IF A$<"1" OR A$>"3"  

    THEN 140  

150: IF A$="1" WAIT 25  

155: IF A$="2" WAIT 50  

160: IF A$="3" WAIT 100  

165: FOR X=0 TO Q: PRINT N  

    $(X)+" =": PRINT M$(X)  

    : FOR Y=1 TO 8: A$=  

    INKEY$: IF A$="S"  

    RETURN  

170: NEXT Y: NEXT X: BEEP 1  

    : RETURN  

175: GOSUB 265: GOSUB 275  

180: RANDOM : X=RND (Q+1)-  

    1: WAIT 80: ON J GOTO  

    185, 195  

185: PRINT N$(X): GOSUB 22  

    0: IF N$(M)=M$(X)  

    THEN 210  

190: GOSUB 205: PRINT M$(X)  

    : GOTO 215  

195: PRINT M$(X): GOSUB 22  

    0: IF N$(M)=N$(X)  

    THEN 210  

200: GOSUB 205: PRINT N$(X)  

    : GOTO 215  

205: BEEP 1: PRINT "FALSCH  

    !": WAIT 100: RETURN  

210: PRINT "RICHTIG!"  

    GOTO 215  

215: GOSUB 245: GOTO 180  

220: INPUT "UEBERSETZUNG:  

    "; N$(M): RETURN  

225: GOSUB 265: FOR X=0 TO  

    Q: M$(M)="" : FOR Y=1  

    TO 12-LEN N$(X): M$(M  

    )=M$(M)+" "; NEXT Y  

230: LPRINT N$(X)+M$(M)+M  

    $(X): NEXT X: RETURN  

235: GOSUB 265: PRINT #0, N  

    $(*), M$(*): RETURN  

240: INPUT #0, N$(*), M$(*)  

    : RETURN  

245: WAIT 40: PRINT "WEITE  

    RE...": INPUT "EINGAB  

    EN J/N? "; A$  

250: IF A$="J" RETURN  

255: IF A$<>"N" THEN 245  

260: POKE 18164, 108:  

    RETURN  

265: IF Q=-1 PAUSE "SPEIC  

    HER LEER!": POKE 1816  

    4, 108: RETURN  

270: RETURN  

275: INPUT "ENG-DEU/DEU-E  

    NG?", A$: IF A$="E"  

    LET J=1: PRINT "ENGLI  

    SCH": RETURN  

280: IF A$="D" LET J=2:  

    PRINT "DEUTSCH":  

    RETURN  

285: GOTO 275

```

AENDERUNGEN FUER
PC 126X:

```
260:POKE 26356,133:  
      RETURN  
265:IF Q=-1 PAUSE "SPEIC  
HER LEER!":POKE 2635  
6,133:RETURN
```

AENDERUNGEN FUER
PC 1350:

```
260:POKE 28460,146:  
      RETURN  
265:IF Q=-1 PAUSE "SPEIC  
HER LEER!": POKE 284  
60,146: RETURN
```

Adresseverwaltung

Das Programm 'Adresseverwaltung' ist ähnlich aufgebaut wie 'Wordstar'. Auch dieses Programm darf nur das erste Mal mit RUN gestartet werden, da sonst alle gespeicherten Adressen gelöscht werden. Erneutes Aufstarten wird mit DEF A bewerkstelligt.

Auch hier hängt die Anzahl speicherbarer Adressen vom verfügbaren Speicher ab; z.B. beim PC 1401 haben 18 Adressen Platz. Die Speichersättigung wird angezeigt.

Adressebestandteile:

Vorname:	max. 10 Zeichen
Name:	max. 12 Zeichen
Strasse:	max. 16 Zeichen
PLZ:	max. 4 Zeichen
Ort:	max. 16 Zeichen
Telefon:	max. 14 Zeichen

Bedienungsanleitung:

- Starten Sie das Programm mit RUN.
- Auf der Anzeige erscheinen die einzelnen Menüpunkte:

1 = EINGEBEN	5 = DRUCKEN
2 = AENDERN	6 = SPEICHERN
3 = SUCHEN	7 = LADEN
4 = LISTEN	

Die verschiedenen Teilprogramme werden mit einem Tastendruck aufgerufen. Sie müssen eine Taste zirka 1 Sekunde lang drücken.

1: EINGEBEN

Wie bei 'Wordstar'.

2: AENDERN

Geben Sie ein, welche Adresse Sie ändern wollen. Falls Sie die Nummer einer bestimmten Adresse nicht wissen, sollten Sie zuerst die Funktion 'Suchen' oder 'Listen' aufrufen.

3: SUCHEN

Sie haben die Möglichkeit, nach Namen oder Orten zu suchen:

N = Name
O = Ort

Geben Sie den genauen Namen oder die Ortsangabe der gesuchten Adresse ein. Falls sich diese Adresse im Speicher befindet, wird sie darauf vollständig angezeigt (inkl. Nummer).

4: LISTEN

Alle Adressen werden mit der zugehörigen Nummer der Reihe nach gelistet.

5: DRUCKEN

Durch Eingabe der Adreßnummer können einzelne Adressen ausgedruckt werden (z.B. zum Aufkleben auf Briefe etc.).

6 und 7: SPEICHERN (6) bzw. LADEN (7)

Die im Rechner abgelegten Adressen können alle auf Band gespeichert und auch wieder geladen werden. Vor dem Aufrufen dieser Funktionen sollten Sie das Cassettengerät für die Aufnahme oder Wiedergabe vorbereiten.

Wichtig: Der Save- und Loadvorgang darf auf keinen Fall unterbrochen werden, da sonst der Rechner abstürzen kann.

Bitte im Direktmodus nur noch Standardvariablen verwenden!

PC 14XX:	PRINT "6= SPEICHERN" :GOSUB 40 35:PRINT "7= LADEN": GOSUB 40:PRINT "E= E NDE":GOSUB 40:GOTO 2 5 40:A\$=INKEY\$ 45:IF A\$="1" GOSUB 90 50:IF A\$="2" GOSUB 115 55:IF A\$="3" GOSUB 170 60:IF A\$="4" GOSUB 250 65:IF A\$="5" GOSUB 215 70:IF A\$="6" GOSUB 240 75:IF A\$="7" GOSUB 245 80:IF A\$="E" END 85:WAIT 40:RETURN 90:WAIT 40:Q=Q+1:IF Q>1 8 WAIT 100:PRINT "SP
10:CLEAR :M= INT (MEM / 81) 15:DIM V\$(M)*10,N\$(M+1) *12,S\$(M)*16,P\$(M)*4 ,0\$(M+1)*16,T\$(M)*14 :Q=-1 20:"A":PAUSE "* ADRESSD ATEI *":WAIT 40 25:PRINT "1= EINGEBEN": GOSUB 40:PRINT "2= A ENDERN":GOSUB 40: PRINT "3= SUCHEN": GOSUB 40 30:PRINT "4= LISTEN": GOSUB 40:PRINT "5= D RUCKEN":GOSUB 40:	

```

EICHER WOLL!":Q=0-1:
RETURN
95:PRINT Q+1;" ADRESSE:
"
100:INPUT "VORNAME: ";V$(
Q),"NAME: ";N$(Q),"STRASSE: ";S$(Q),"PL
Z: ";P$(Q)
105:INPUT "ORT: ";O$(Q),
"TELEFON: ";T$(Q):
WAIT 40
110:GOSUB 265:GOTO 90
115:GOSUB 285:PRINT "ES
SIND...":PRINT Q+1;" ADRESSEN...":PRINT
"GESPEICHERT."
120:PRINT "WELCHE WOLLEN
...":PRINT "SIE AEND
ERN?...":WAIT 100:
PRINT "1. BIS ";Q+1;
"ADR.?"
125:INPUT "IHRE WAHL: ";
X$:X=VAL (X$):X=X-1
130:IF X<0 OR X>Q THEN 1
25
135:WAIT 40:PRINT "VORNA
ME (ALT)...":PRINT V
$(X):INPUT "VORNAME
(NEU)":";V$(X)
140:PRINT "NAME (ALT)...":
PRINT N$(X):INPUT
"NAME (NEU)":";N$(X)
145:PRINT "STRASSE (ALT)
...":PRINT S$(X):
INPUT "STRASSE (NEU)
":";S$(X)
150:PRINT "PLZ (ALT)...":
PRINT P$(X):INPUT "
PLZ (NEU)":";P$(X)
155:PRINT "ORT (ALT)...":
PRINT O$(X):INPUT "
ORT (NEU)":";O$(X)
160:PRINT "TELEFON (ALT)
...":PRINT T$(X):
INPUT "TELEFON (NEU)
":";T$(X)
165:GOSUB 265:GOTO 115
170:GOSUB 285:PRINT "WOL
LEN SIE...":PRINT "N
ACH NAMEN...":PRINT
"ODER ORTEN..."
175:PRINT "SUCHEN?"
180:INPUT "NAME/ORT? ";A
$:IF A$="N" THEN 205
185:IF A$<>"O" THEN 180
190:INPUT "ORT: ";O$(M+1
):FOR X=0 TO Q:IF O$(
M+1)=O$(X) GOSUB 20
0
195:GOTO 210
200:WAIT 60:PRINT X+1;" ADRESSE...":WAIT :
PRINT V$(X):PRINT N$(
X):PRINT S$(X):
PRINT P$(X):PRINT O$(
X):PRINT T$(X):
RETURN
205:INPUT "NAME: ";N$(M+
1):FOR X=0 TO Q:IF N
$(M+1)=N$(X) GOSUB 2
00
210:NEXT X:GOSUB 265:
GOTO 170
215:GOSUB 285
220:PRINT "WELCHE ADRESS
E...":PRINT "WOLLEN S
IE...":PRINT "DRUCKE
N?":PRINT "1. BIS ";
Q+1;" ADR.?"
225:INPUT "IHRE WAHL: ";
X$:X=VAL (X$):X=X-1:
IF X<0 OR X>Q THEN 2
25
230:LPRINT V$(X)+" "+N$(
X):LPRINT S$(X):
LPRINT P$(X)+" "+O$(
X):LPRINT "TEL.: ";T
$(X):LPRINT "":
LPRINT ""
235:GOSUB 265:WAIT 40:
GOTO 215
240:GOSUB 285:PRINT #Q,V

```

```

$(*) , N$(*) , S$(*) , P$(*)
*, O$(*) , T$(*) ;
RETURN
245: INPUT #Q, V$(*) , N$(*)
, S$(*) , P$(*) , O$(*) , T
$(*) : RETURN
250: GOSUB 285: FOR X=0 TO
Q: PRINT X+1;" ADRESS
E...": PRINT V$(X):
PRINT N$(X): PRINT S$(
X): PRINT P$(X):
PRINT O$(X): PRINT T$(
X)
255: A$=INKEY$: IF A$="S"
RETURN
260: NEXT X: BEEP 1: RETURN
265: WAIT 40: PRINT "WEITE
RE...": INPUT "EINGAB
EN J/N? "; A$
270: IF A$="J" RETURN
275: IF A$<>"N" THEN 265
280: POKE 18164,108:
RETURN
285: IF Q<0 PAUSE "SPEICH
ER LEER!": POKE 18164
,108: RETURN
290: RETURN

```

AENDERUNGEN FUER
PC 126X:

```

280: POKE 26356,133:
RETURN
285: IF Q<0 PAUSE "SPEICH
ER LEER!": POKE 26356
,133: RETURN

```

AENDERUNGEN FUER
PC 1350:

```

280: POKE 28460,146:
RETURN
285: IF Q<0 PAUSE "SPEICH
ER LEER!": POKE 2846
0,146: RETURN

```

9.4 Spiele

Wir haben uns bemüht, in dieser Spalte nur phantasievolle und interessante Spiele aufzuführen.

'Mastermind' oder 'Hangman' sind Denkspiele, bei denen es darum geht, möglichst schnell etwas Unbekanntes herauszufinden.

Andere Spiele wiederum zeigen, wie man BASIC und Maschinensprache kombinieren kann, so z.B. das Programm 'Reaktions-test'.

Etwas Spezielles bietet auch das Spiel 'Siebzehn und vier'. Hier kann man nämlich auch gegen den Computer spielen. Der Computer ist so programmiert, daß er eine Strategie wählt, die am wahrscheinlichsten zum Sieg führt. Umso reizvoller ist es natürlich, trotzdem gegen den Computer zu gewinnen.



Mastermind

Mastermind ist ein spannendes Denkspiel, bei dem Sie eine von Ihrem Rechner ermittelte Zufallszahl durch logisches Kombinieren mit möglichst wenig Versuchen herausfinden sollten. In einem gewissen Rahmen können Sie den Schwierigkeitsgrad des Spiels selbst bestimmen: Nach dem Starten des Programms mit DEF M müssen Sie eingeben, aus wievielen Ziffern die zu erratende Zahl bestehen soll (ANZ. STELLEN:). Der Rechner akzeptiert bei dieser Abfrage Werte von 2 bis 6, d.h., die Zufallszahl muß mindestens aus 2, kann aber höchstens aus 6 Ziffern bestehen. Bei der Abfrage 'LEVEL 1/2:' haben Sie die Möglichkeit, zwei verschiedene Spielarten zu wählen:

Level 1: Jede Ziffer der Zufallszahl kommt nur einmal vor; es ist z.B. nicht möglich, daß in einer Zahl zweimal die Ziffer 2 vorkommt.

Level 2: Die einzelnen Ziffern der zu erratenden Zahl können doppelt, dreifach, kurz mehrfach vorkommen. Level 2 ist also eindeutig die schwierigere Spielart.

Bei der Abfrage 'ZIFFERN 1-' können Sie wählen, aus welchen Ziffern die Zufallszahl bestehen soll. Die Zahl muß mindestens aus zwei verschiedenen Ziffern bestehen (Eingabe: 2), maximal können aber die Ziffern 1 bis 9 verwendet werden (Eingabe: 9).

Wer einen Drucker besitzt, lässt sich mit Vorteil die Spiel-situation auf den Drucker ausgeben. Eine entsprechende Abfrage wurde ins Programm einbezogen. Die Ausgabe auf den Drucker ermöglicht ein wesentlich übersichtlicheres Spiel.

Der Rechner bewertet Ihre Tips, also Ihre Eingaben, mit zwei verschiedenen Zeichen:

- Eine Ziffer der von Ihnen vorgeschlagenen Zahl kommt auch in der Zahl vor, die der Rechner sich 'ausgedacht' hat, sie befindet sich aber noch nicht an der richtigen Stelle.
- * Eine Ziffer hat nicht nur den richtigen Wert, sondern sie steht auch an der richtigen Stelle.

*Beispiel: **-*

Zwei Ziffern (**) Ihres Tips haben nicht nur den richtigen Wert, sondern sie stehen auch an der richtigen Stelle. Eine weitere Ziffer (-) hat zwar den richtigen Wert, aber sie steht an der falschen Stelle.

Die restlichen Ziffern kommen in der zu erratenden Zahl nicht vor.

Wenn Sie anstelle eines Tips den Buchstaben E (=Ende) eingeben, so wird das laufende Spiel abgebrochen und die zu erratende Zahl angezeigt.

PC 14XX:

```
500:"M" CLEAR :INPUT "AN
Z. STELLEN: ";S:IF S
<2 OR S>6 THEN 500
505:DIM X$(S)*1,Y$(S)*1
510:INPUT "LEVEL 1/2: ";
L:IF L<1 OR L>2 THEN
510
515:INPUT "ZIFFERN 1-";K
:IF K<S OR K>9 IF L=
```

```
1 OR K<2 THEN 515
520:PRINT = PRINT :PAUSE
"AUSGABE AUF...":
INPUT "DRUCKER J/N?":
";A$:IF A$="J" PRINT
=LPRINT
525:FOR I=1 TO S
530:RANDOM :R$=STR$ RND
K:POKE 18163,24
535:FOR J=1 TO LEN Z$:IF
R$=MID$(Z$,J,1) AND
```

```

L=1 THEN 538
540:NEXT J:Z1=Z1+R1:NEXT
    1
545:RS=":":INPUT "IHR TIP
    :":TAKE:IF AS="E"
    PRINT "ZAHL":A2$:
    GOTO 590
550:FOR J=1 TO 9:X$(J)=
    RIDE(Z1,J,1):Y$(J)=
    RIDE(AS,J,1)
555:IF X$(J)=Y$(J) LET R
    S=R$+"-":X$(J)="X":Y
    $(J)="Y"
560:NEXT J
565:FOR I=1 TO S:FOR J=1
    TO S
570:IF X$(I)=Y$(J) LET R
    S=R$+"-":X$(I)="X":Y
    $(J)="Y"
575:NEXT J:NEXT I:V=V+1:
    PRINT STR V;"":TAKE
    :":RS
580:IF AS>2$ THEN 545
585:BEER 1:PRINT = PRINT
    :PAUSE "ERRATEN NACH
    ...":PRINT STR V:

```

VERSUCHEN!"
 590:INPUT "HOCHMASS J/N?":
 TAKE:IF AS="J" THEN
 580
 595:END

AENDERUNGEN FUER
 PC 128K:

530:RANDOM :R\$=STR\$ RND
 :KPOKE 26355,24

AENDERUNGEN FUER
 PC 1350:

530:RANDOM :R\$=STR\$ RND
 :KPOKE 28459,24

Siebzehn und vier

Sicher kennen Sie das bekannte Gesellschaftsspiel "Siebzehn und vier". Nun können Sie es auch auf Ihrem PC spielen. Für dieses Spiel gelten folgende Spielregeln:

Es gibt Karten zu 2, 3, 4, 6, 7, 8, 9 und 10 Punkten. Einen Sonderfall stellt das As dar: Es kann nämlich sowohl 11 Punkte als auch nur 1 Punkt wert sein, je nachdem, was gerade günstiger ist.

Um zu gewinnen, muß man eine möglichst große Punktzahl zwischen 17 (Minimum) und 21 (Maximum) erzielen. Bei einer Punktzahl unter 17 hat man auf jeden Fall verloren. Das Punktemaximum kann nur dann über 21 Punkten liegen, wenn

es sich bei beiden gezogenen Karten um Asse handelt ($2 \cdot 11 = 22$).

Das Programm wird mit RUN gestartet. Dann erscheint '1/2 SPIELER' auf der Anzeige. Bei der Eingabe von 2 spielen Sie das Spiel zu zweit, also mit einem Spielpartner. Geben Sie dagegen 1 ein, so müssen Sie gegen die Spielstrategie des Computers spielen. Dieser orientiert sich an der Wahrscheinlichkeitsrechnung und verfolgt daher eine optimale Strategie.

Nun können Sie ein Startkapital eingeben, das beiden Spielern gutgeschrieben wird. Mit diesem Kapital müssen Sie auskommen, bis das Spiel abgebrochen wird.

Danach geben Sie den Einsatz für diese Runde ein. Gewinnen Sie diese Runde, so wird Ihnen dieser Einsatz gutgeschrieben und Ihrem Spielpartner abgeschrieben. Verlieren Sie eine Spielrunde, so müssen Sie für den Einsatz aufkommen.

Das Spiel kann beginnen.

Der 1. Spieler kommt an die Reihe. Der Rechner zeigt die Punktzahl der ersten beiden gezogenen Karten an; ebenso die Anzahl Asse, die sich darin befinden. Wurden z.B. die Karten mit 2 und 10 Punkten gezogen, zeigt der Rechner an: '1: 12 Punkte 0 Asse'. Wollen Sie fortfahren, so drücken Sie ENTER.

Jetzt erscheint die Frage 'KARTE J/N?', und Sie können eingeben, ob Sie weitere Karten ziehen wollen. Wenn Sie 'J' für Ja eingeben, zieht Ihnen der Rechner eine weitere Karte. Kommt dabei eine Punktzahl von mehr als 21 zustande, so wird ein eventuell vorhandenes As automatisch als 1 Punkt gewertet und bei der Angabe über den Bestand von Assen nicht mehr aufgeführt.

Sind Sie schließlich mit Ihrer Punktzahl zufrieden oder haben Sie verspielt, drücken Sie bei 'KARTE J/N?' die Taste 'N'. Damit kommt der zweite Spieler an die Reihe. Spielen Sie gegen

den Computer, so erscheint keine Meldung; nach einigen Sekunden hat der Computer gezogen.

Anschließend wird die Punktzahl beider Spieler ausgewertet, der Sieger wird angezeigt und schließlich auch die Kapitalbilanz.

Wollen Sie mit den aktuellen Vermögensverhältnissen weiterspielen, so geben Sie bei der Abfrage 'NOCHMALS J/N?' einfach 'J' ein. Der Rechner fragt Sie erneut nach Ihrem Einsatz, und schon können Sie es nochmals versuchen.

FÜR ALLE RECHNER:

```

200:CLEAR :DIM K(35):
      INPUT "SPIELER (1/2)
      :";B:INPUT "STARTKA
      PITAL: ";L:M=L
205:"G"J=1:T=2:U=2:A=0:Z
      =0:Y=0:V=0:W=0:FOR X
      =2 TO 11:IF X=5 NEXT
      X
210:FOR I=0 TO 3:K(A)=X:
      A=A+1:NEXT I:NEXT X:
      INPUT "EINSATZ: ";D
215:GOSUB 315:GOSUB 315:
      PRINT "1. SPIELER!!"
220:PRINT "1: "+STR$ Y+
      PTE "+STR$ V+" ASSE
      ":INPUT "KARTE J/N?
      ";S:IF S=1 GOSUB 305
      :T=T+1:Y=Y+E:IF E=11
      LET V=V+1
225:IF Y>21 AND V>0 AND
      T>2 LET V=V-1:Y=Y-10
230:IF S=1 GOTO 220
235:IF B=1 GOTO 265
240:PRINT "2. SPIELER!!"
245:PRINT "2: ";STR$ Z+
      PTE.";STR$ W;" ASSE
      ":INPUT "KARTE J/N?
      ";S:IF S=1 GOSUB 305
      :U=U+1:Z=Z+E:IF E=11
      LET W=W+1
250:IF Z>21 AND W>0 AND

```

```

U>2 LET W=W-1:Z=Z-10
255:IF S=1 GOTO 245
260:GOTO 280
265:IF Z<18 GOSUB 305:Z=
      Z+E:U=U+1:IF E=11
      LET W=W+1
270:IF Z>21 AND W>0 AND
      U>2 LET W=W-1:Z=Z-10
275:IF Z<17 GOTO 265
280:IF (Y>16 AND (Y<22
      OR (Y=22 AND T=2))
      AND (Y>Z OR (Z>21
      AND U>2)) BEEP 1:
      PRINT "SIEGER: SPIEL
      ER1":L=L+D:M=M-D
285:IF (Z>16 AND (Z<22
      OR (Z=22 AND U=2))
      AND (Z>Y OR (Y>21
      AND T>2)) BEEP 2:
      PRINT "SIEGER: SPIEL
      ER2":M=M+D:L=L-D
290:PRINT "PTE 1: ";STR$ Y;
      2: ";STR$ Z:
      PRINT "1: ";STR$ L;
      US$":PRINT "2: ";
      STR$ M; US$"
295:INPUT "NOCHMALS J/N?
      ";S$:IF S$="J" THEN
      205
300:END
305:RANDOM :X=RND 36-1:
      IF K(X)=0 GOTO 305
310:E=K(X):K(X)=0:RETURN

```

```
315:GOSUB 305:Y=Y+E:IF E  
    =11 LET V=V+1  
320:GOSUB 305:Z=Z+E:IF E  
    =11 LET W=W+1  
325:RETURN
```

Reaktionstest

Dieses Programm ist zum größten Teil in BASIC geschrieben. Ein Teil in Maschinensprache ermöglicht die Anzeige auf dem Bildschirm, während gleichzeitig die Tastatur abgefragt und die Zeit bis zum ersten Tastendruck gezählt wird.

Starten Sie das Programm mit RUN. Der Titel 'REAKTIONSTEST' erscheint auf dem Display. Nach dem nächsten Druck auf die ENTER-Taste erscheint '*** LOS ***'. Sobald Sie das nächste Mal ENTER betätigen, erscheint ein Zeichen auf der Anzeige.

Die Schwierigkeit dieses Spiels besteht nun darin, dieses Zeichen möglichst schnell in eine von sechs gegebenen Gruppen einzutragen und dann schnell den Code für diese Gruppe einzugeben.

Bei einem falschen Code wird der richtige Code und ein passender Kommentar (z.B. 'SCHWACH! 2') angezeigt. War der Code richtig, so erfolgt die Meldung 'RICHTIG: ' und die Reaktionszeit gemessen in Hundertstelsekunden. Warten Sie nicht zu lange: Nach zirka 1.5 Sekunden wird die Tastaturabfrage automatisch abgebrochen.

Aus der folgenden Tabelle erfahren Sie den Code, den Sie für eine bestimmte Zeichengruppe eingeben müssen:

1 = Sonderzeichen	(?, @, ", !, \$, etc.)
2 = Buchstaben	(A, B, C, ..., Z)
3 = Negative Zahlen	(-0, -1, ..., -8, -9)
4 = BASIC-Befehle	(PRINT, END, READ, etc.)
5 = String-Befehle	(MIDS, RIGHTS, LEFTS, etc.)
6 = Positive Zahlen	(0, 1, ..., 8, 9)

Sicher ist es am Anfang relativ schwierig, diese Codes in der zur Verfügung stehenden Zeit (1.5 Sekunden) einzutippen. Aber da Sie bei einem Fehler sofort sehen, welches der richtige Code war, haben Sie diese Codes schon bald im Kopf.

PC 140X:

```

400:"S" WAIT :RESTORE 46
 0:PRINT "REAKTIONSTE
  ST":Z$=""":Y=1
405:FOR I=17800 TO 17837
  :READ A:POKE I,A:
  NEXT I
410:PRINT "*** LOS ***":
  WAIT 0
415:RANDOM :E=RND 6:
  RANDOM :IF E=6 OR E=
  3 LET F=RND 10:Z$=
  CHR$ (47+F):IF E=3
  LET Z$="-"+Z$
420:IF E=1 LET F=RND 15:
  Z$=CHR$ (32+F)
425:IF E=2 LET F=RND 26:
  Z$=CHR$ (64+F)
430:IF E=4 LET F=RND 21:
  POKE 17873,207+F,0
435:IF E=5 LET F=RND 6:
  POKE 17873,167+F,0
440:PRINT "      ";Z$:
  CALL 17800:WAIT
445:IF E+48=PEEK 25901
  LET P=.6*PEEK 25902:
  PRINT "RICHTIG: ";
  STR$ P:G=G+P:GOTO 45
  5

```

```

450:PRINT "SCHWACH! ";
  STR$ E
455:Z$=""":D=0:Y=Y+1:GOTO
  410
460:DATA 229,162,0,0,2,4
  4,134,219,2,101,135,
  219,2,0,38,7,64,42,4
  ,55,206,206,237
465:DATA 59,16,101,45,87
  ,103,0,57,15,16,101,
  46,128,83,55

```

AENDERUNGEN FUER
PC 1421:

```

405:FOR I=17700 TO 17737
  :READ A:POKE I,A:
  NEXT I
430:IF E=4 LET F=RND 21:
  POKE 17773,207+F,0
435:IF E=5 LET F=RND 6:
  POKE 17773,167+F,0
440:PRINT "      ";Z$:
  CALL 17700:WAIT
460:DATA 229,168,0,0,2,4
  4,134,219,2,101,135,
  219,2,0,38,7,64,42,4
  ,55,206,206,237

```

AENDERUNGEN FUER
PC 126X:

```

405:FOR I=25800 TO 25837
    :READ A:POKE I,A:
    NEXT I
430:IF E=4 LET F=RND 21:
    POKE 25857,207+F,0
435:IF E=5 LET F=RND 6:
    POKE 25857,167+F,0
440:PRINT "      ";Z$:
    CALL 25800:WAIT
460:DATA 229,162,0,0,2,4
    4,134,219,2,101,135,
    219,2,0,38,7,64,42,4
    ,206,206,206,234
465:DATA 169,16,101,45,8
    7,103,0,57,15,16,101
    ,46,128,83,55

```

AENDERUNGEN FUER
PC 1350:

```

405:FOR I=27650 TO 27687
    :READ A:POKE I,A:
    NEXT I
430:IF E=4 LET F=RND 21:
    POKE 27697,207+F,0
435:IF E=5 LET F=RND 6:
    POKE 27697,167+F,0
440:PRINT "      ";Z$:
    CALL 27650:WAIT
445:IF E+48=PEEK 27694
    LET P=.6*PEEK 27695:
    PRINT "RICHTIG: ";
    STR$ P:G=G+P:GOTO 45
    5
460:DATA 229,162,0,0,2,4
    5,134,219,2,108,135,
    219,2,0,38,7,64,42,4
    ,206,206,206,235
465:DATA 187,16,108,46,8
    7,103,0,57,15,16,108
    ,47,128,83,55

```

Hangman

'Hangman' ist ein Spiel für zwei Personen.

Der 1. Spieler gibt ein möglichst schwieriges Wort (max. 16 Zeichen) ein.

Der 2. Spieler muß nun Stück für Stück ausfindig machen, wie dieses Wort heißt. Er weiß nur, wie lang das zu erratende Wort ist. Mit den schwarzen Feldern wird die Anzahl Buchstaben des gesuchten Wortes angezeigt.

Der 2. Spieler, der ja das Wort erraten soll, gibt in einem ersten Versuch einen Buchstaben ein, von dem er vermutet, daß er im

gesuchten Wort enthalten ist. Ist der Buchstabe tatsächlich in diesem Wort vorhanden, so wird er direkt eingefügt und anstelle der schwarzen Felder gezeigt. Falls Sie herausgefunden haben, wie das gesuchte Wort heißt, können Sie auch das ganze Lösungswort direkt eingeben.

Die aktuelle Anzahl Fehlversuche wird natürlich angezeigt. Nach 8 Fehlversuchen ist das Spiel verloren.

PC 140X:

```

10:CLEAR :WAIT 80:I=0:
    DIM A$(2)*16:POKE 18
    064,245,254,0:PRINT
    "*** HANG-MAN ***"
15:PRINT "EINGABE...":
    PRINT "MAX. 16 ZEICH
EN."
20:INPUT "WORT: ";A$(0)
    :FOR J=1 TO LEN A$(0)
    ):A$(2)=A$(2)+B$:
    NEXT J
25:WAIT :PRINT A$(2):
    INPUT "IHR TIP: ";A$
(1):WAIT 100:A=0:IF
    LEN A$(1)=LEN A$(0)
    THEN 55
30:FOR J=1 TO LEN A$(0)
    :IF A$(1)<>MID$(A$(0),J,1) NEXT J:GOTO
    40
35:A$(2)=LEFT$(A$(2),J
    -1)+A$(1)+RIGHT$(A$(2),LEN A$(2)-J):A=A
    +1:NEXT J
40:IF A=0 THEN 70
45:IF A$(2)=A$(0) THEN
    60
50:BEEP @:GOTO 25
55:IF A$(1)<>A$(0) THEN
    75
60:FOR K=1 TO 3:FOR J=1
    TO 10:CALL 2755:NEXT
    J:BEEP 1:NEXT K

```

```

65:WAIT 120:GOSUB 100:
    PRINT "GEWONNEN MIT.
    ..":PRINT I;" FEHLVE
    RSUCHEN!":GOTO 80
70:I=I+1:WAIT 80:PRINT
    "*FALSCH GERATEN*":
    FOR J=1 TO 20:CALL 2
    755:NEXT J:IF I<8
    WAIT 100:GOTO 25
75:WAIT 120:PRINT "***VERLOREN ***":GOSUB
    100
80:INPUT "NOCH EINMAL?
    ";A$
85:IF A$="J" THEN 10
90:IF A$<>"N" THEN 80
95:END
100:PRINT "DAS WORT HEIS
    ST":PRINT A$(0):
    RETURN

```

AENDERUNGEN FUER
PC 1421:

```

10:CLEAR :WAIT 80:I=0:
    DIM A$(2)*16:POKE 17
    984,245,254,0:PRINT
    "*** HANG-MAN ***"
60:FOR K=1 TO 3:FOR J=1
    TO 10:CALL 2700:NEXT
    J:BEEP 1:NEXT K
70:I=I+1:WAIT 80:PRINT
    "*FALSCH GERATEN*":
    FOR J=1 TO 20:CALL 2

```

```

700:NEXT J:IF I<8          ÄENDERUNGEN FUER
WAIT 100:GOTO 25           PC 1350:

ÄENDERUNGEN FUER
PC 126X:

10:CLEAR :WAIT 80:I=0:
DIM A$(2)*16:B$=CHR$249:PRINT "*** HANG-
MAN ***"
60:FOR K=1 TO 3:FOR J=1
TO 10:CALL 2379:NEXT
J:BEEP 1:NEXT K
70:I=I+1:WAIT 80:PRINT
"*FALSCH GERÄTEN*":
FOR J=1 TO 20:CALL 2
379:NEXT J:IF I<8
WAIT 100:GOTO 25

```

9.5 Utilities

Unter diesem Kapitel drucken wir eine Reihe interessanter, kurzer Hilfsprogramme (Utilities) ab. Fast alle diese Programme sind in reiner Maschinensprache abgefaßt und bilden eigene BASIC-Befehle; Sie können also Ihre BASIC-Programme mit diesen Utilities ergänzen.

Diese Hilfsprogramme zeigen auf praktische Weise, mit welch geringem Aufwand 'neue' BASIC-Befehle programmiert werden können (vgl. Kapitel 8.3).

Ist bei der Beschreibung der einzelnen Programme nichts besonderes vermerkt, so werden alle Programme mit der Anfangsadresse, d.h. mit der Adresse des ersten Befehls des Programms, gestartet. Ohne speziellen Vermerk sind die Programme durchaus auch in einem anderen Adreßbereich lauffähig.

Dataline - Automatische Datazeilen-Erzeugung

'Dataline' wird Ihnen sicher sehr von Nutzen sein. Mit diesem Hilfsprogramm haben Sie nämlich die Möglichkeit, die Inhalte ganzer Speicherbereiche in BASIC-DATA-Zeilen umzuwandeln. Sie geben einfach Anfangs- und Endadresse des gewünschten Speicherblocks ein und schon werden diese Daten ausgelesen und in Form von DATA-Zeilen als BASIC-Programm gespeichert.

Im Gegensatz zu allen andern Utilities ist dieses Programm rein in BASIC geschrieben und stellt daher keinen eigenen BASIC-Befehl dar. Es kann mit RUN oder DEF D gestartet werden.

Folgende Eingaben werden gleich nach dem Starten des Programms verlangt:

Anfangsadresse: Geben Sie die Anfangsadresse des gewünschten Speicherblocks ein.

Endadresse: Hier wird nach der Endadresse des umzuwandelnden Speicherblocks gefragt.

Zeilenummer: Geben Sie die Zeilenummer ein, die die erste DATA-Zeile haben soll. Diese Nummer muß natürlich größer sein als diejenige der letzten Zeile Ihrer Programme im Speicher.

Schrittweite: Abstand zwischen zwei DATA-Zeilen:
Zeilenummer der nächsten DATA-Zeile =
Zeilenummer der vorhergenden Zeile +
Schrittweite.

Haben Sie ein wenig Geduld, während der Rechner mit der Umwandlung beschäftigt ist; bei größeren Maschinenprogrammen (mehr als 500 Bytes) können Sie sich schon eine Kaffee-

pause gönnen (die Umwandlung von 500 Bytes in DATA-Zeilen dauert zirka 5 Minuten).

Ist der PC mit der Umwandlung fertig, meldet er sich mit einem BEEP-Signal. Falls hingegen zwei BEEP-Signale ertönen, ist für die Umwandlung zu wenig freier Speicherplatz vorhanden: Es werden dann soviele DATA-Zeilen generiert, bis die Speichersättigung eintritt.

Wichtig: Unterbrechen Sie das Programm auf keinen Fall, nachdem es einmal gestartet ist; dies könnte unter Umständen zu einem Systemabsturz führen.

Das Utility 'Dataline' ist natürlich ideal für die Umwandlung von Maschinenprogrammen in DATA-Zeilen; Ihre Maschinenprogramme können Sie dann als BASIC-Programme abspeichern, was vor allem dann sinnvoll ist, wenn ein Programm nur teilweise in Maschinensprache geschrieben wurde (z.B. unser Programm 'Reaktionstest' in Kapitel 9.4).

PC 14XX:

```

10:"D" INPUT "ANFANGSAD
R.: ";A,"ENDADR.: ";
E,"ZEILENNUMMER: ";Z
,"SCHRITTWEITE: ";S
15:V=PEEK 18172+PEEK 18
    173*256-5:X=PEEK 181
    47+PEEK 18148*256
20:B=X+2:H= INT (Z/256)
    :L=Z-H*256:POKE X,H,
    L,0,220:X=X+3
25:W$=STR$ PEEK A:FOR I
    =1 TO LEN W$:POKE X+
    I,ASC MID$ (W$,I,1):
    NEXT I:X=X+I:A=A+1
30:IF X>V LET X=B-2:
    POKE X,255:H= INT (X
    /256):L=X-H*256:POKE
    18147,L,H:BEEP 2:END
35:IF A>E POKE X,13,255
    :POKE B,X-B:X=X+1:H=
    INT (X/256):L=X-H*25
    6:POKE 26339,L,H:
    BEEP 1:END

```

BEEP 1:END

```

40:IF X-B>70 POKE X,13:
    POKE B,X-B:X=X+1:Z=Z
    +S:GOTO 20
45:POKE X,44:GOTO 25

```

AENDERUNGEN FUER
PC 126X:

```

15:V=PEEK 26364+PEEK 26
    365*256-5:X=PEEK 263
    39+PEEK 26340*256
30:IF X>V LET X=B-2:
    POKE X,255:H= INT (X
    /256):L=X-H*256:POKE
    26339,L,H:BEEP 2:END
35:IF A>E POKE X,13,255
    :POKE B,X-B:X=X+1:H=
    INT (X/256):L=X-H*25
    6:POKE 26339,L,H:
    BEEP 1:END

```

AENDERUNGEN FUER
PC 1350:

```
15:V=PEEK 28423+PEEK 28
    424*256-5:X=PEEK 284
    19+PEEK 28420*256
30:IF X>V LET X=B-2:
    POKE X,255:H= INT (X
    /256):L=X-H*256:POKE
    28419,L,H:BEEP 2:END
```

```
35:IF A>E POKE X,13,255
    :POKE B,X-B:X=X+1:H=
    INT (X/256):L=X-H*25
    6:POKE 28419,L,H:
    BEEP 1:END
```

RENEW - Zurückholen von gelöschten Programmen

Dieses Programm ist außerordentlich nützlich. Mit diesem Utility kann nämlich der ganze BASIC-Speicher, nachdem er versehentlich oder auch absichtlich mit NEW gelöscht wurde, wiederhergestellt werden. Auch die Endadresse des BASIC-Programms wird wieder rekonstruiert.

Nach dem Durchlaufen dieses Programms verhält sich der Rechner fast so, als ob NEW nie eingegeben worden wäre. Sie können also nach wie vor das 'zurückgeholt' BASIC-Programm editieren, verändern, Zeilen löschen oder einfügen.

Doch Vorsicht: Da nur die wichtigsten Systemvariablen wiederhergestellt werden, bleibt das wiederhergestellte Programm nur solange intakt, bis ein neues BASIC-Programm eingegeben wird oder zu viele Feldvariablen definiert werden. In diesem Fall kann das 'alte' BASIC-Programm nicht mehr zurückgeholt werden und bleibt verloren.

Das Programm ist wie üblich für den PC 14XX gelistet. Für die andern Rechner müssen gewisse Adressen geändert werden¹⁾. Die nötigen Änderungen finden Sie in den Fußnoten.

1) Adresse 17700 und 17722: PC 126X: 26337 (102,225), PC 1350: 28417 (111,1)
Adresse 17718: PC 126X: 26339 (102,227), PC 1350: 28419 (111,3)

17700 LIDP 18145	16, 70, 225
17703 LP 4	132
17704 MVBD	26
17705 IX	4
17706 IX	4
17707 IXL	36
17708 LIB 0	3, 0
17710 LP 4	132
17711 ADB	20
17712 IXL	36
17713 CPIA 255	103, 255
17715 JRNZM 10	41, 10
17717 LP 4	132
17718 LIDP 18147	16, 70, 227
17721 EXBD	27
17722 LIDP 18145	16, 70, 225
17725 LP 6	134
17726 MVBD	26
17727 LIA 0	2, 0
17729 IYS	38
17730 RTN	55



MERGE - Zuladen von Programmen (nur PC 14XX)

Mit diesem Utility können mehrere BASIC-Programme nacheinander von der Cassette in den Rechner geladen werden, ohne daß das sich bereits im Speicher befindliche Programm gelöscht wird. Sie können also mit diesem Hilfsprogramm verschiedene auf Cassette gespeicherte Programme im Speicher miteinander verknüpfen.

Die Zeilenummern des hinzuladenden Programms sollten natürlich größer sein als diejenigen des bereits im Programmspeicher vorhandenen Programms; andernfalls werden Sie bei der Handhabung dieser Zeilen große Mühe haben.

Bei den Rechnern PC 126X und PC 1350 ist MERGE bereits als BASIC-Befehl fest eingebaut.

Der Programmname wird einfach an den CALL-Befehl angehängt:

CALL Startadresse,"Programmname"

Dieser Befehl funktioniert praktisch gleich wie der CLOAD-Befehl, einzig der Programmname wird mit einem Komma von der Anfangsadresse des Maschinenprogramms abgetrennt. Nur so akzeptiert der Rechner die Eingabe. Fehler werden nicht angezeigt; bei Ladefehlern oder zuwenig Speicherplatz kehrt der Rechner einfach ins BASIC zurück. Kontrollieren Sie deshalb nach dem Ladevorgang, ob sich das gewünschte Programm wirklich im Speicher befindet. Mit BRK/ON kann jederzeit unterbrochen werden.

Auch hier muß wieder eine Adresse für den PC 1421 geändert werden¹⁾.

17600	LIDP	18145	16,	70,	225
17603	LP	10	138		
17604	MVBD		26		
17605	LP	10	138		
17606	LIDP	17872	16,	69,	208
17609	MVDM		83		
17610	INCP		80		
17611	LIDP	17873	16,	69,	209
17614	MVDM		83		
17615	LIDP	18147	16,	70,	227
17618	LP	20	148		
17619	MVBD		26		
17620	LIB	0	3,	0	
17622	LIA	1	2,	1	
17624	LP	20	148		
17625	SBB		21		
17626	LIDP	18145	16,	70,	225
17629	LP	20	148		
17630	EXBD		27		
17631	IX		4		

1) Adresse 17632: PC 1421: 46070

17632 CALL 40460	120, 158, 12
17635 LIDP 17872	16, 69, 208
17638 LP 10	138
17639 MVBD	26
17640 LIDP 18145	16, 70, 225
17643 LP 10	138
17644 EXBD	27
17645 LIDP 18352	16, 71, 176
17648 LIA 13	2, 13
17650 LII 79	0, 79
17652 FILD	31
17653 RTN	55

17872

LINECALC - Berechnung der Anfangsadresse einer Programmzeile

Dieses Hilfsprogramm ermöglicht es Ihnen, die Anfangsadresse einer beliebigen BASIC-Programmzeile in Sekundenschnelle zu berechnen. Als Anfangsadresse einer Programmzeile gilt die Adresse, in der das Highbyte der Zeilennummer steht (das Lowbyte steht erst in der nächsthöheren Adresse).

Auch bei diesem Befehl wird die Nummer der gesuchten Programmzeile einfach mit einem Komma an den CALL-Befehl angehängt.

Eingabemöglichkeiten:

CALL Startadresse,Zeilenummer

CALL Startadresse,numerische Variablen oder Ausdrücke

17873

Existiert eine gesuchte Programmzeile nicht, so ertönt ein BEEP-Ton. Wurde die Zeile aber gefunden, so wird die Anfangsadresse dieser Zeile in den ersten beiden Bytes der Standardvariable Z abgelegt. Das Lowbyte der Adresse steht vor dem Highbyte. Um die richtige Anfangsadresse zu erhalten, muß vom Lowbyte noch 1 subtrahiert werden:

$$\text{Anfangsadresse} = \text{Lowbyte} + 256 * \text{Highbyte} - 1$$

17872 17873

Wird dieses Utility in einem BASIC-Programm aufgerufen, so fährt der Rechner nach der Ausführung des Maschinenprogramms mit dem nächsten BASIC-Befehl fort.

Wichtig: Das WAIT-Intervall wird verändert! Als WAIT-Intervall steht nun der Wert der eingegebenen Zeilennummer.

Auch dieses Hilfsprogramm zeigt keine Fehlermeldungen an. Ein ungültiges Argument wird meistens mit 0 ersetzt.

Auch hier sind wieder in einigen Fällen Adressen zu ändern¹⁾.

17400	IX	4
17401	CALL 49775	120, 194, 111
17404	LP 8	136
17405	LIDP 18149	16, 70, 229
17408	MVBD	26
17409	LIDP 18145	16, 70, 225
17412	LP 4	132
17413	MVBD	26
17414	IXL	36
17415	LP 9	137
17416	CPMA	199
17417	JRNZP 13	40, 13
17419	IXL	36
17420	LP 8	136
17421	CPMA	199
17422	JRNZP 9	40, 9
17424	LIDP 17872	16, 69, 208

-
- 1) Adresse 17401: PC 1421: 56000, (218,192), PC 126X: 49403 (192,251),
 PC 1350: 49946 (195,26)
 Adresse 17405: PC 126X: 26341 (102,229), PC 1350: 28851 (112,179)
 Adresse 17409: PC 126X: 26337 (102,225), PC 1350: 28417 (111,1)
 Adresse 17424: PC 1421: 17792 (69,128), PC 126X: 25856 (101,0),
 PC 1350: 27696 (108,48)
 Adresse 17447: PC 126X: 26165 (102,53), PC 1350: 28448 (111,32)
 Adresse 17468: PC 1421: 55539 (216,243), PC 126X: 48908 (191,12),
 PC 1350: 49430 (193,22)

17427 LP 4	132
17428 EXBD	27
17429 JRP 17	44, 17
17431 IX	4
17432 IXL	36
17433 LIB 0	3, 0
17435 LP 4	132
17436 ADB	20
17437 IXL	36
17438 CPIA 255	103, 255
17440 JRNZP 3	40, 3
17442 JRP 25	44, 25
17444 DX	5
17445 JRM 32	45, 32
17447 LIDP 18101	16, 70, 181
17450 LP 4	132
17451 MVBD	26
17452 IXL	36
17453 CPIA 58	103, 58
17455 JRZP 5	56, 5
17457 CPIA 13	103, 13
17459 JRNZM 8	41, 8
17461 DX	5
17462 LIDP 18101	16, 70, 181
17465 LP 4	132
17466 EXBD	27
17467 RTN	55
17468 CALL 49321	120, 192, 169
17471 JRM 25	45, 25

PSAVE - Speichern von Teilprogrammen

Mit diesem Hilfsprogramm können Sie einzelne BASIC-Programmzeilen und somit auch Teilprogramme auf Cassette speichern.

Eingabe:

CALL Startadresse, erste Zeilennr., letzte Zeilennr., "Name"

Wichtig: Ein auf diese Weise auf Band abgespeichertes Teilstprogramm kann nicht mit CLOAD? verifiziert werden. Fehlermeldungen werden nicht ausgegeben. Mindestens zwei Zeilen müssen gespeichert werden; eine erste und eine letzte Zeile.

Bitte beachten Sie die eventuell zu ändernden Adressen¹⁾.

17400 IX	4
17401 CALL 49775	120, 194, 111
17404 LIDP 18149	16, 70, 229
17407 LP 10	138
17408 MVBD	26
17409 LP 10	138
17410 LIDP 17872	16, 69, 208
17413 EXBD	27
17414 IX	4
17415 CALL 49775	120, 194, 111
17418 LP 4	132
17419 LIDP 17874	16, 69, 210
17422 EXBD	27
17423 LIDP 18145	16, 70, 225
17426 LP 4	132
17427 LII 3	0, 3

-
- 1) Adresse 17401: PC 1421: 56000 (218,192), PC 126X: 49403 (192,251),
PC 1350: 49946 (195,26)
Adressen 17404 und 17477: PC 126X: 26341 (102,229), PC 1350: 28851
(112,179)
Adressen 17410 und 17440: PC 126X: 25856 (101,0), PC 1350: 27696
(108,48)
Adressen 17419 und 17516: PC 126X: 25858 (101,2), PC 1350: 27698
(108,50)
Adressen 17423, 17472 und 17482: PC 126X: 26337 (102,225), PC 1350:
28417 (111,1) A. 17524
Adressen 17430, 17435 und 17876: PC 126X: 28860 (112,188)
PC 1350: 27700 (108,52)
Adresse 17510: PC 126X: 26339 (102,227), PC 1350: 28419 (111,3)
Adresse 17521: PC 1421: 45802 (178,234), PC 126X: 37896 (148,8),
PC 1350: 38055 (148,167)
Adresse 17536: PC 126X: 26544 (103,176), PC 1350: 28336 (110,176)

17429 MVWD	24
17430 LIDP 17876	16, 69, 212
17433 LP 4	132
17434 EXWD	25
17435 LIDP 17876	16, 69, 212
17438 LP 4	132
17439 MVBD	26
17440 LIDP 17872	16, 69, 208
17443 LP 8	136
17444 MVBD	26
17445 IXL	36
17446 LP 9	137
17447 CPMA	199
17448 JRNZP 8	40, 8
17450 IXL	36
17451 LP 8	136
17452 CPMA	199
17453 JRNZP 4	40, 4
17455 JRP 14	44, 14
17457 IX	4
17458 IXL	36
17459 LIB 0	3, 0
17461 LP 4	132
17462 ADB	20
17463 IXL	36
17464 CPIA 255	103, 255
17466 JRNZM 21	41, 21
17468 JRP 67	44, 67
17470 DX	5
17471 DX	5
17472 LIDP 18145	16, 70, 225
17475 LP 4	132
17476 EXBD	27
17477 LIDP 18149	16, 70, 229
17480 LP 8	136
17481 MVBD	26
17482 LIDP 18145	16, 70, 225
17485 LP 4	132
17486 MVBD	26
17487 IX	4
17488 IX	4
17489 IXL	36

17490 LP 4	132
17491 ADB	20
17492 IXL	36
17493 CPIA 255	103, 255
17495 JRZP 40	56, 40
17497 LP 9	137
17498 CPMA	199
17499 JRNZM 12	41, 12
17501 IXL	36
17502 LP 8	136
17503 CPMA	199
17504 JRNZM 16	41, 16
17506 IXL	36
17507 LP 4	132
17508 ADB	20
17509 IX	4
17510 LIDP 18147	16, 70, 227
17513 LP 4	132
17514 EXBD	27
17515 LP 4	132
17516 LIDP 17874	16, 69, 210
17519 MVBD	26
17520 IX	4
17521 CALL 40192	120, 157, 0
17524 LIDP 17876	16, 69, 212
17527 LP 10	138
17528 LII 3	0, 3
17530 MVWD	24
17531 LP 10	138
17532 LIDP 18145	16, 70, 225
17535 EXWD	25
17536 LIDP 18352	16, 71, 176
17539 LIA 13	2, 13
17541 LII 79	0, 79
17543 FILD	31
17544 RTN	55

FIND - das Bytesuchprogramm

Mit Hilfe dieses Programms können Sie eine beliebige Bytesfolge von maximal 10 Bytes Länge im gesamten Adressbereich von 0 bis 65535 suchen. Ein ganzer Speicherdurchlauf (65536 Bytes) dauert ca. 40 Sekunden.

Das Programm kann natürlich mit BRK/DN abgetrochen werden. Der Rechner gibt dann einen BEEP-Ton mit einer Frequenz von 2 kHz (halb so hoch wie normal) von sich und führt, falls das Maschinenprogramm von einem BASIC-Programm aus aufgerufen wurde, mit der Ausführung des nächsten Befehls fort. Existiert die gesuchte Bytesfolge im Speicher nicht, so kehrt der Rechner mit einem normalen BEEP (4 kHz) und ERROR 1 ins BASIC zurück.

Das Programm wird folgendermaßen aufgerufen:

```
CALL Startadresse,Suchanfang,Azu. Bytes-1,Byte 1, Byte 2,...
```

Unter dem Namen 'Suchanfang' ist die Adresse gemeint, bei der die Bytesuche im Speicher beginnen soll. Für alle Parameter können sowohl numerische Ausdrücke als auch numerische Variablen gesetzt werden.

Das Resultat, die Anfangsadresse der Bytesfolge, wird in den ersten zwei Bytes der Standardvariablen Z im Low/Highbyte-Format abgelegt. Dort kann es dann auch ausgelesen werden. Diese Adresse erhalten Sie also mit:

Gesuchte Adresse = PEKK 17872¹⁾ + 256*PEKK 17873²⁾

1) PC 1421: 17792, PC 136X: 25856, PC 1380: 27696
2) PC 1421: 17793, PC 136X: 25857, PC 1380: 27697

Hinweise:

- Das WAIT-Intervall wird verändert.
- Für die Verwendung in Programmen: Werden mehr Bytes eingegeben, als mit der Anzahl der Bytes festgelegt wurde, so müssen die überzähligen Bytes den Wert 0 haben, sonst wird die Befehlsfolge nicht gefunden.
- Dieses Utility kann nicht von einem andern Maschinenprogramm aus aufgerufen werden, da der Stackpointer R verstellt wird. Der Rechner geht nach der Ausführung dieses Hilfsprogramms immer ins BASIC zurück.

Warnung: Der Parameter für die Anzahl der zu suchenden Bytes darf unter keinen Umständen den Wert 9 (also 10-1) überschreiten. Andernfalls besteht akute Absturzgefahr.

Auch hier sind wieder einige Änderungen von Adressen zu beachten¹⁾.

-
- 1) Adressen 16001 und 16026: PC 1421: 56000 (218,192), PC 126X: 49403 (192,251), PC 1350: 49946 (195,26)
Adressen 16004 und 16029: PC 126X: 26341 (102,229), PC 1350: 28851 (112,179)
Adresse 16010: PC 126X: 25856 (101,0), PC 1350: 27648 (108,0)
Adresse 16056: PC 126X: 25858 (101,2), PC 1350: 27650 (108,2)
Adresse 16136: PC 126X: 25856 (101,0), PC 1350: 27696 (108,48)
Adresse 16155: PC 1421: 55539 (216,243), PC 126X: 48908 (191,12),
PC 1350: 49430 (193,22)
Adresse 16164: PC 126X und PC 1350: ersetzen durch WAIT 0, NOPW (78,0,77)
Adressen 16158 bis 16163: PC 126X und PC 1350: Speicherzellen mit NOPW füllen
Adresse 16167: PC 126X: CAL 4414 (241,62), PC 1350: CAL 4618 (18,10)
Adresse 16174: PC 1421: CAL 2700 (234,140), PC 126X: CAL 2094 (232,46), PC 1350: CAL 2379 (233,75)
Adressen 16183 und 16198: PC 126X: 26165 (102,53), PC 1350: 28448 (111,32)

16000 IX	4
16001 CALL 49775	120, 194, 111
16004 LIDP 18149	16, 70, 229
16007 LP 10	138
16008 MVBD	26
16009 LP 10	138
16010 LIDP 24576	16, 96, 0
16013 EXBD	27
16014 LIA 1	2, 1
16016 LIB 96	3, 96
16018 PUSH	52
16019 EXAB	218
16020 PUSH	52
16021 IXL	36
16022 CPIA 44	103, 44
16024 JRNZP 22	40, 22
16026 CALL 49775	120, 194, 111
16029 LIDP 18149	16, 70, 229
16032 POP	91
16033 EXAB	218
16034 POP	91
16035 LP 6	134
16036 LIQ 2	19, 2
16038 MVB	10
16039 LDD	87
16040 IYS	38
16041 LP 2	130
16042 LIQ 6	19, 6
16044 MVB	10
16045 JRM 28	45, 28
16047 LIA 86	2, 86
16049 STR	50
16050 LP 10	138
16051 LIA 0	2, 0
16053 LII 40	0, 40
16055 FILM	30
16056 LIDP 24578	16, 96, 2
16059 LP 0	128
16060 MVMD	85
16061 LIDL 0	17, 0
16063 LP 4	132
16064 MVBD	26

16065 LIDL 3	17,	3
16067 LP 10	138	
16068 EXWD	25	
16069 LP 5	133	
16070 LDM	89	
16071 DECP	81	
16072 EXAB	218	
16073 LDM	89	
16074 LP 30	158	
16075 DATA	53	
16076 LP 30	158	
16077 LDM	89	
16078 LP 10	138	
16079 CPMA	199	
16080 JRNZP 66	40,	66
16082 LP 31	159	
16083 LDM	89	
16084 LP 11	139	
16085 CPMA	199	
16086 JRNZP 60	40,	60
16088 LP 32	160	
16089 LDM	89	
16090 LP 12	140	
16091 CPMA	199	
16092 JRNZP 54	40,	54
16094 LP 33	161	
16095 LDM	89	
16096 LP 13	141	
16097 CPMA	199	
16098 JRNZP 48	40,	48
16100 LP 34	162	
16101 LDM	89	
16102 LP 14	142	
16103 CPMA	199	
16104 JRNZP 42	40,	42
16106 LP 35	163	
16107 LDM	89	
16108 LP 15	143	
16109 CPMA	199	
16110 JRNZP 36	40,	36
16112 LP 16	144	
16113 LDM	89	
16114 LP 36	164	

16115	CPMA	199
16116	JRNZP 30	40, 30
16118	LP 37	165
16119	LDM	89
16120	LP 17	145
16121	CPMA	199
16122	JRNZP 24	40, 24
16124	LP 18	146
16125	LDM	89
16126	LP 38	166
16127	CPMA	199
16128	JRNZP 18	40, 18
16130	LP 19	147
16131	LDM	89
16132	LP 39	167
16133	CPMA	199
16134	JRNZP 12	40, 12
16136	LIDP 17872	16, 69, 208
16139	LP 4	132
16140	EXBD	27
16141	JRP 41	44, 41
16143	WAIT 0	78, 0
16145	WAIT 0	78, 0
16147	LIB 0	3, 0
16149	LIA 1	2, 1
16151	LP 4	132
16152	ADB	20
16153	JRNCP 16	42, 16
16155	CALL 49321	120, 192, 169
16158	LP 16	144
16159	LIA 254	2, 254
16161	LII 15	0, 15
16163	FILM	30
16164	CALL 32789	120, 128, 21
16167	JP 5208	121, 20, 88
16170	TEST 8	107, 8
16172	JRZM 104	57, 104
16174	CAL 2755	234, 195
16176	LIA 255	2, 255

16178	PUSH	52
16179	WAIT 255	78, 255
16181	LOOP 3	47, 3
16183	LIDP 18101	16, 70, 181
16186	LP 4	132
16187	MVBD	26
16188	IXL	36
16189	CPIA 13	103, 13
16191	JRZP 5	56, 5
16193	CPIA 58	103, 58
16195	JRNZM 8	41, 8
16197	DX	5
16198	LIDP 18101	16, 70, 181
16201	LP 4	132
16202	EXBD	27
16203	RTN	55

SOUND - Töne nach Belieben

Mit dem neuen BASIC-Befehl 'SOUND' können Sie dem Rechner einen Ton von beliebiger Tonhöhe und Tonlänge entlocken.

Dieser neue BASIC-Befehl wird folgendermaßen eingegeben:

CALL Startadresse, Tonhöhe, Tonlänge

Für Tonhöhe und Tonlänge sind Werte von 0 bis 65535 zugelassen:

Tonhöhe: 0: hoher Ton
65535: tiefer Ton

Tonlänge: 0: kurzer Ton
65535: langer Ton

In Abhängigkeit von der Tonhöhe dauert ein Ton mit der Länge 65535 20 Sekunden bis 10 Minuten.

Wichtig: Ein hoher Ton erklingt bei gleichem Wert für die Tonlänge wesentlich weniger lang als ein tiefer Ton. Achten Sie deshalb darauf, bei hohen Tönen den Wert für die Tonlänge etwas größer zu wählen.

Hinweis: Das Argument des WAIT-Befehls und auch das des darauffolgenden LIA-Befehls wird beim Programmaufruf verändert. Da sich das Programm diese Werte für einen bestimmten Speicherbereich selbst sucht, ist es durchaus relokatibel und muß in einem andern Speicherbereich als dem im Listing angegebenen nicht neu installiert werden.

Das Programm darf unter keinen Umständen vor diesen beiden Befehlen WAIT- und LIA abgeändert werden. Es muß auch berücksichtigt werden, daß das J-Register der CPU während dem Programmablauf verändert wird.

Beachten Sie auch hier wieder die zu ändernden Adressen!¹⁾

17500	DXL	37
17501	CPIA 204	103, 204
17503	JRNZM 4	41, 4
17505	CALL 49775	120, 194, 111
17508	LIDP 18149	16, 70, 229
17511	LP 60	188
17512	MVBD	26
17513	LIDP 18101	16, 70, 181
17516	LP 4	132
17517	MVBD	26
17518	IX	4

1) Adressen 17505, 17519 und 17528: PC 1421: 56000 (218,192), PC 126X: 49403 (192,251), PC 1350: 49946 (195,26)
 Adresse 17508 und 17523: PC 126X: 26341 (102,229), PC 1350: 28851 (112,179)
 Adressen 17513 und 17579: PC 126X: 26165 (102,53), PC 1350: 28448 (111,32)

17519	CALL 49775	120, 194, 111
17522	LP 62	190
17523	LIDP 18149	16, 70, 229
17526	MVBD	26
17527	IX	4
17528	CALL 49775	120, 194, 111
17531	LP 60	188
17532	LIQ 4	19, 4
17534	EXB	11
17535	LIB 0	3, 0
17537	LIA 62	2, 62
17539	LP 4	132
17540	ADB	20
17541	LP 62	190
17542	LDM	89
17543	IX	4
17544	STD	82
17545	IX	4
17546	IX	4
17547	LP 63	191
17548	MVDM	83
17549	LIDP 18149	16, 70, 229
17552	LP 0	128
17553	MVBD	26
17554	LIP 95	18, 95
17556	DRIM 16	97, 16
17558	OUTC	223
17559	ANIM 239	96, 239
17561	OUTC	223
17562	WAIT 255	78, 255
17564	LIA 255	2, 255
17566	PUSH	52
17567	LOOP 1	47, 1
17569	LP 0	128
17570	LIA 1	2, 1
17572	LIB 0	3, 0
17574	SBB	21
17575	JRNCM 22	43, 22
17577	LIJ 1	1, 1
17579	LIDP 18101	16, 70, 181
17582	LP 4	132
17583	MVBD	26
17584	IXL	36
17585	CPIA 58	103, 58

17587 JRZP 5	56,	5
17589 CPIA 13	103,	13
17591 JRNZM 8	41,	8
17593 RC	209	
17594 DX	5	
17595 LIA 88	2,	88
17597 STR	50	
17598 RTN	55	

9.6 Grafik: Turbo Graphics

Wir stellen Ihnen in diesem Kapitel zwei Grafikprogramme vor, die die grafischen Fähigkeiten Ihres Rechners optimal ausnutzen.

Grafikprogramm für den PC 14XX und den PC 126X

Die Grafikmöglichkeiten der Rechner PC 14XX und PC 126X sind leider recht beschränkt. Dennoch bieten wir Ihnen mit dem folgenden Maschinenprogramm ein Hilfsmittel, mit dem Sie auf ganz einfache Art beliebige Grafiken und spezielle Zeichen generieren können.

Nach dem Start des Programms (CALL Anfangsadresse) stehen Ihnen beim PC 14XX acht Felder, also das halbe Display, beim PC 126X ganze zwölf Felder für die Grafikerstellung zur Verfügung.

Auf dem Display befindet sich ein Cursor, ein einzelner Bildpunkt mit halber Helligkeit, der Ihnen anzeigt, an welcher Stelle Sie sich gerade befinden. Dieser Cursor kann beliebig nach oben und unten, nach links und rechts bewegt werden. Derjenige Punkt, an dessen Stelle sich der Cursor gerade befindet, wird auf das Drücken bestimmter Tasten gesetzt oder gelöscht. Sie können also auf einfache Art Grafiken erzeugen.

Beim Verlassen von 'Turbo Graphics' (d.h. bei der Rückkehr ins BASIC), wird der ganze Displayinhalt automatisch im Standardvariablenbereich gespeichert. Genauso wird auch nach dem

Starten des Programms der Inhalt der Standardvariablen in den Displayspeicher kopiert, damit Sie diejenigen Grafiken und Zeichen, die Sie vielleicht schon früher erstellt haben, weiter ergänzen, verändern und vervollständigen können. Wollen Sie ein neues 'Bild' auf einer blanken Anzeige zeichnen, so müssen Sie vor dem Start von 'Turbo Graphics' nur die Variablen mit dem BASIC-Befehl CLEAR löschen. So erscheint dann ein 'leeres' Display, bereit, Ihre neuen Ideen festzuhalten.

Da ja beim Verlassen des Programms alle Grafikpunkte in den Standardvariablenbereich kopiert werden, können Sie Ihre Grafiken mit

CSAVE M Anfangsadresse,Endadresse

Anfangsadresse = Anfangsadresse der Standardvariablen
Endadresse = Endadresse der Standardvariablen

auf Cassette speichern.

Tastaturbelegung

R: Cursor abwärts
7: Cursor auf
1: Cursor links
4: Cursor rechts

F: Punkt setzen
V: Punkt löschen

V und F: Zurück ins BASIC

Beachten Sie auch hier wieder die zu ändernden Adressen¹⁾.

-
- 1) Adresse 17507: PC 1421: CAL 1448 (229,168), PC 126X: WAIT 0 (78,0)
 Adressen 17510 und 17703: PC 1421: 17792 (69,128), PC 126X: 25856 (101,0)
 Adressen 17518 und 17698: PC 126X: 8192 (32,0)
 Adresse 17513: PC 126X: 59 (59)

17500 LIA 8	2,	8
17502 LP 8	136	
17503 EXAM	219	
17504 WAIT 0	78,	0
17506 PUSH	52	
17507 CAL 1442	229,	168
17509 NOPT	206	
17510 LIDP 17872	16,	69 128
17513 LII 39	0,	39
17515 LP 10	138	
17516 MVWD	24	
17517 LP 10	138	
17518 LIDP 24576	16,	96 0
17521 EXWD	25	
17522 JRP 9	44,	9
17524 WAIT 124	78,	124
17526 WAIT 124	78,	124
17528 WAIT 124	78,	124
17530 NOPT	206	
17531 NOPT	206	
17532 LIA 96	2,	96
17534 LP 5	133	
17535 EXAM	219	
17536 LIA 0	2,	0
17538 LP 4	132	
17539 EXAM	219	
17540 LIP 92	18,	92
17542 LIA 1	2,	1
17544 EXAM	219	
17545 OUTA	93	
17546 INA	76	
17547 WAIT 0	78,	0
17549 CPIA 32	103,	32
17551 JRZP 83	56,	83
17553 CPIA 2	103,	2
17555 JRZP 95	56,	95
17557 CPIA 4	103,	4
17559 JRZP 55	56,	55
17561 CPIA 8	103,	8
17563 JRZP 31	56,	31
17565 NOPT	206	
17566 CPIA 64	103,	64
17568 JRZP 107	56,	107

17570	CPIA	128	103, 128
17572	JRZP	112	56, 112
17574	CPIA	192	103, 192
17576	JRZP	121	56, 121
17578	IX		4
17579	DXL		37
17580	LP	8	136
17581	EXAM		219
17582	ORMA		71
17583	EXAM		219
17584	STD		82
17585	WAIT	100	78, 100
17587	EXAM		219
17588	SBM		69
17589	EXAM		219
17590	STD		82
17591	WAIT	25	78, 25
17593	JRM	54	45, 54
17595	POP		91
17596	STD		82
17597	LIA	255	2, 255
17599	PUSH		52
17600	WAIT	80	78, 80
17602	LOOP	3	47, 3
17604	IXL		36
17605	PUSH		52
17606	LP	4	132
17607	WAIT	0	78, 0
17609	JRM	70	45, 70
17611	WAIT	0	78, 0
17613	WAIT	0	78, 0
17615	POP		91
17616	STD		82
17617	LIA	255	2, 255
17619	PUSH		52
17620	WAIT	80	78, 80
17622	LOOP	3	47, 3
17624	DXL		37
17625	PUSH		52
17626	LP	4	132
17627	WAIT	100	78, 100
17629	JRM	90	45, 90
17631	WAIT	132	78, 132

17633	WAIT 132	78, 132
17635	POP	91
17636	STD	82
17637	PUSH	52
17638	LP 8	136
17639	EXAM	219
17640	SL	90
17641	EXAM	219
17642	LIA 255	2, 255
17644	PUSH	52
17645	WAIT 110	78, 110
17647	LOOP 3	47, 3
17649	JRM 110	45, 110
17651	LP 8	136
17652	CPIM 0	99, 0
17654	JRZP 17	56, 17
17656	POP	91
17657	STD	82
17658	PUSH	52
17659	LP 8	136
17660	EXAM	219
17661	SR	210
17662	EXAM	219
17663	LIA 255	2, 255
17665	PUSH	52
17666	WAIT 110	78, 110
17668	LOOP 3	47, 3
17670	JRM 131	45, 131
17672	ORIM 128	97, 128
17674	JRM 135	45, 135
17676	POP	91
17677	LP 8	136
17678	EXAM	219
17679	ORMA	71
17680	EXAM	219
17681	STD	82
17682	PUSH	52
17683	JRM 144	45, 144
17685	LP 8	136
17686	LDM	89
17687	LIB 255	3, 255
17689	LP 3	131

17690	SBM	69
17691	POP	91
17692	ANMA	70
17693	EXAB	218
17694	PUSH	52
17695	STD	82
17696	JRM 157	45, 157
17698	LIDP 24576	16, 96 0
17701	LP 10	138
17702	MWWD	24
17703	LIDP 17872	16, 69 128
17706	LP 10	138
17707	EXWD	25
17708	POP	91
17709	RTN	55

Grafikprogramm für den PC 1350

Da man auf dem PC 1350 mit seiner vierzeiligen Anzeige viel bessere und umfangreichere Grafiken erstellen kann, haben wir für diesen Rechner ein spezielles Programm entwickelt, mit dem Sie die grafischen Möglichkeiten des PC 1350 nun wirklich voll ausnutzen können.

Damit Sie das Programm überhaupt betreiben können, muß Ihr PC 1350 mit einer RAM-Erweiterung ausgestattet sein. Auch müssen mindestens 6030 freie Bytes für das Programm (BASIC und Maschinensprache) und für die Speicherung des Bildes im RAM vorhanden sein.

'Turbo Graphics' für den PC 1350 bietet folgende Möglichkeiten:

- Punkte setzen und löschen
- Punkte miteinander verbinden, d.h. Linien ziehen
- Kreise und Ellipsen zeichnen
- Speicherung von Bildern im RAM und auf Cassette

Bedient wird das Programm folgendermaßen:

Start:

DEF D: Das Programm wird gestartet; ein im RAM gespeichertes Bild wird gelöscht. Von der Cassette kann ein Bild geladen werden.

DEF G: Start des Programms; ein bereits vorhandenes Bild im RAM bleibt erhalten. Es kann kein Bild von der Cassette geladen werden. DEF G funktioniert auch, nachdem CLEAR eingegeben wurde.

Tastenfunktionen:

L: Laden des Bildes aus dem RAM in die Anzeige

Q: Speichern des Bildes ins RAM und Verlassen des Programms

K: Speichern des Bildes, und Kreis oder Ellipse zeichnen

S: Anfangspunkt einer Linie bestimmen

E: Endpunkt einer Linie bestimmen und Linie zeichnen

+: Punkt setzen

-: Punkt löschen

4: Cursor links

6: Cursor rechts

2: Cursor ab

8: Cursor auf

.: Punkt, gefolgt von einer Ziffer: Schrittweite des Cursors in jede Richtung (1-9 Punkte)

Festlegung der Schrittweite des Cursors

Drücken Sie die Taste '*' so lange, bis der Cursor nicht mehr blinks. Anschließend können Sie durch Anwählen einer Ziffer (1-9) die Schrittweite des Cursors bestimmen.

Cursor bewegen

Mit den Tasten 2, 3, 4 und 6 kann der Cursor vertikal oder horizontal verschoben werden. Er springt dabei um soviele Punkte weiter, wie unter dem Programmfpunkt 'Schrittweite' angegeben wurde.

Punkte setzen oder löschen

Durch Drücken der Taste '+' wird derjenige Punkt, auf dem sich der Cursor gerade befindet, gesetzt (schwarz). Mit der Taste '-' kann ein Punkt gelöscht werden.

Linien ziehen

Um Linien zu ziehen, wird der Cursor an den Anfangspunkt der gewünschten Linie gesetzt. Befindet sich der Cursor an der richtigen Stelle, so drücken Sie die Taste 'S', um diesen Anfangspunkt zu markieren. Bewegen Sie nun den Cursor an den Endpunkt der Linie. Auf Drücken der Taste 'E' wird die Linie gezeichnet. Sollen mehrere Linien vom gleichen Anfangspunkt aus gezeichnet werden, so genügt es, diesen Anfangspunkt einmal zu definieren (Taste 'S').

Kreise und Ellipsen zeichnen

Damit Kreise oder Ellipsen gezeichnet werden können, muß zuerst der Mittelpunkt des Kreises bzw. der Ellipse mit dem Cursor festgelegt werden. Bewegen Sie den Cursor also an die gewünschte Stelle. Drücken Sie nun die Taste 'K'. Das Bild auf

der Anzeige wird nun im RAM zwischengespeichert und der Rechner fragt Sie nach dem Radius des Kreises in Anzahl Punkten. Als zweites müssen Sie das X:Y-Verhältnis der Radien festlegen. Wie man sofort sieht, wird ein Kreis gezeichnet, wenn dieses Verhältnis 1 beträgt (X- und Y-Radius sind dann gleich lang). Wird ein Verhältnis ungleich 1 eingegeben, so wird eine Ellipse mit den Radien $r(x)=r \cdot V$ und $r(y)=r$ gezeichnet, wobei V das Verhältnis zwischen der X- und Y-Koordinate beschreibt. Wenn Sie diese Eingaben abgeschlossen haben, wird das im RAM zwischengespeicherte Bild in die Anzeige zurückgeladen, und der Kreis oder die Ellipse wird gezeichnet. Der Zeichenvorgang dauert etwa eine Minute. Haben Sie also ein wenig Geduld!

Laden eines Bildes aus dem RAM

Wird die Taste 'L' gedrückt, wird das Bild, das sich im RAM befindet, ins Anzeigefeld kopiert.

Vorsicht: Wurde 'Turbo Graphics' mit DEF D gestartet, erscheint nach Drücken der Taste 'L' eine Fehlermeldung, wenn kein Bild von der Cassette geladen oder kein Kreis gezeichnet wurde. In einem solchen Fall müssen Sie 'CLS', dann 'MODE' drücken und das Programm nochmals starten.

Programm verlassen

Um das Programm zu verlassen, müssen Sie den Cursor aus dem Bild hinausschieben, weil er sonst auch als ein normaler Bildpunkt mitgespeichert wird. Danach wird die Taste 'Q' (=Quit) gedrückt. Das Bild wird automatisch im RAM abgespeichert. Der Rechner fragt Sie nun, ob das Bild zusätzlich noch auf Cassette gespeichert werden soll. Mit den Tasten 'J' (=Ja) oder 'N' (=Nein) können Sie diese Entscheidung treffen. Im Falle 'J' wird das Bild auf den Cassettenrecorder ausgegeben und das Programm verlassen. Mit 'N' kann das Programm direkt verlassen werden.

```

2000:"D" CLEAR : DIM K$      2070:PSET (X,Y):A$=
(255,16)*1:CLS:           INKEY$: IF A$="4"
WAIT 0:PRINT "Bil          GOSUB 2200:X=X-Z:
d laden (J/N)?            GOTO 2060
2005:A$= INKEY$: IF A$      2080:IF A$="6" GOSUB 22
="J" PRINT "Laden          00:X=X+Z: GOTO 206
laeuft ...":              0
INPUT #K$(*):             2090:IF A$="8" GOSUB 22
GOTO 2015                  00:Y=Y-Z: GOTO 206
                           0
2010:IF A$<>"N" GOTO 20      2100:IF A$="2" GOSUB 22
                           00:Y=Y+Z: GOTO 206
                           0
2015:"G" CLS : GOSUB 23      2110:IF A$="+" LET P=1:
10: WAIT 0: PRINT           GOTO 2070
" **** MENU                2120:IF A$="-" LET P=0:
*****":X=0,Y=0:Z=          GOTO 2070
1
2020:PRINT " Cursorste      2130:IF A$="S" LET S=X,
uerung": PRINT "           T=Y: GOTO 2070
2: nach unten":           2140:IF A$="E" LINE (S,
WAIT : PRINT "           T)-(X,Y): GOTO 206
8: nach oben               0
2025:WAIT 0: PRINT "       2150:IF A$=". " GOTO 222
6: nach rechts":           0
PRINT " 4: nach
liks": WAIT                 2160:IF A$="Q" GOTO 224
2030:PRINT "L: neues Bi    0
ld laden"
2040:WAIT 0: PRINT "+:      2170:IF A$="L" CALL 231
Punkt setzen":             28
PRINT "-: Punkt lo        2180:IF A$="K" GOTO 229
eschen": PRINT "S:          0
Start einer Linie
"
2045:WAIT : PRINT "E: E      2190:PRESET (X,Y):
nde einer Linie":         GOTO 2070
WAIT 0
2050:PRINT "Q: Bild spe      2200:IF P=1 PSET (X,Y):
ichern und Progr          RETURN
amm verlassen":           2210:PRESET (X,Y):
PRINT ".<Ziffer>:          RETURN
neue Schritte
2055:WAIT : CURSOR 23,1     2220:A$= INKEY$: IF A$<
: PRINT ":" CLS:           ">"1" OR A$>"9" OR
                           A$="" GOTO 2220
                           2230:Z= VAL A$: GOTO 20
                           70
                           2240:CLEAR : DIM K$(255
                           ,16)*1: CALL 23100
                           2250:CLS : PRINT "Bild
                           auf Kassette spei-

```

```

        chern (J/N)?"
2260:A$= INKEY$ : IF A$      NEXT A: RETURN
      ="N" END
2270:IF A$="J" PRINT "S      3000:DATA 2,255,132,219
      peichern laeuft ..      ,2,111,133,219,2,1
      ." : PRINT #K$(*):
      CLS : END
2280:GOTO 2260
2290:CALL 23100: CLS :
      INPUT "Radius=";R,
      "X:Y - Verhaeltnis
      =";J: CALL 23128
2300:DEGREE : FOR A=0      3010:DATA 52,2,255,52,3
      TO 360 STEP 3:B=
      SIN A:C= COS A:
      PSET (X+B*R*R,J,Y+C*
      R): NEXT A: GOTO 2
      070
2310:WAIT 0: PRINT "Lad
      en des Maschinener
      og-rammes laeuft .
      ..
2320:RESTORE : FOR A=23
      100 TO 23155:
      READ B: POKE A,B:

```

9.7 Musik: Soundbox

Mit Hilfe von diesem Programm wird Ihr PC zu einer elektronischen Orgel, auf der Sie wahlweise entweder manuell spielen (die Tastatur wird zur Klaviatur) oder eine gespeicherte Melodie abrufen und anhören können.

Den Startpunkt, also die Anfangsadresse der Tontabelle für die gespeicherte Melodie, können Sie bei der Erstellung des Programms 'Soundbox' frei wählen. Er richtet sich nach der verfügbaren Speicherkapazität und nach der Länge der jeweiligen Musikstücke.

Das Programm 'Soundbox' muß an seine Programmanfangsadresse angeglichen werden. Es ist also nicht möglich, das mit der Anfangsadresse X erstellte Programm später bei Anfangsadresse Y zu laden und dort laufen zu lassen.

Aus diesem Grunde sollten Sie sich gründlich überlegen, in welchem Speicherbereich Sie das Programm installieren wollen.

Wichtig: Das Programm kann nicht für jeden beliebigen Adressbereich installiert werden; es gibt da gewisse Einschränkungen: Das Lowbyte der Anfangsadresse des Programms muß entweder kleiner als 176 oder aber größer als 232 sein.

Installation

1. Festlegung der Anfangsadresse des Programms (Lowbyte beachten!). Günstige Werte sind zum Beispiel:

PC 14XX: 17300
PC 126X: 25450
PC 1350: 26000

2. Anfangsadresse für die Musiktabelle der gespeicherten Melodie festlegen. Das Lowbyte dieser Anfangsadresse hat den Wert 81. Die Lage der Musiktabelle kann also nur durch das Highbyte bestimmt werden. Bei Rechnern, die nur 4 KByte Speicher besitzen, sollte als Anfangsadresse dieser Tontabelle das Highbyte der BASIC-Anfangsadresse gewählt werden.

3. Falls Sie den Wunsch haben, das Programm in einem andern als dem im Listing angegebenen Adressbereich zu speichern, so müssen alle Adressen des Datapointers DP, der Unterprogrammaufrufe und der absoluten Sprungbefehle an die 'neuen Umstände' angeglichen werden.

4. Besonderheiten: Einige andere Argumente müssen bei andern Rechnern oder andern Startadressen ebenfalls geändert werden. Diese 'faulen' Stellen sind im Listing mit Ziffern gekennzeichnet.

17300	25994	LIDP	27696	16, 108, 48
17303	25997	LIA	0	2, 0
17305	25999	STD		82
17306	26000	LIA	108 69	2, 108
17307	26002	LP	7	135
17308	26003	EXAM		219
17309	26004	LP	6	134
17311	26005	LIA	47 208	2, 47
17313	26007	EXAM		219
17314	26008	CAL	3003 3387	235, 187
17316	26010	LIDP	27696 17872	16, 108, 48
17319	26013	LDD		87
17320	26014	JP	26276 17582	121, 102, 164
17323	26017	NOPW		77
17324	26018	PTC		122
17325	26019	16, 26209 17515		16, 102, 97
17326	26022	ETC		105
17329	26023	90, 26131 17437		90, 102, 19
17332	26026	88, 26137 17443		88, 102, 25
17335	26029	67, 26143 17449		67, 102, 31
17336	26032	86, 26146 17452		86, 102, 34
17341	26035	66, 26152 17458		66, 102, 40
17344	26038	78, 26158 17464		78, 102, 46
17347	26041	77, 26164 17470		77, 102, 52
17350	26044	32, 26167 17473		32, 102, 55
17353	26047	65, 26134 17440		65, 102, 22
17356	26050	83, 26140 17446		83, 102, 28
17359	26053	70, 26149 17455		70, 102, 37
17362	26056	71, 26155 17461		71, 102, 43
17365	26059	72, 26161 17467		72, 102, 49
17368	26062	80, 26250 17556		80, 102, 138
17371	26065	79, 1240 1434		79, 4, 216
17374	26068	82, 26127 17473		82, 102, 15
17377	26071	26122 17428		102, 10
17379	26073	PTC		122
17380	26074	14, 26209 17515		14, 102, 97
17381	26077	ETC		105
17384	26078	90, 26167 17473		90, 102, 55
17387	26081	88, 26173 17479		88, 102, 61
17390	26084	67, 26179 17485		67, 102, 67
17397	26087	86, 26182 17488		86, 102, 70
17396	26090	66, 26188 17494		66, 102, 76
17399	26093	78, 26194 17500		78, 102, 82

17402	-26096	77, 26200	17506	77, 102, 88
17403	-26099	32, 26203	17509	32, 102, 91
17404	-26102	65, 26170	17476	65, 102, 58
17405	-26105	83, 26176	17482	83, 102, 64
17406	-26108	70, 26185	17491	70, 102, 73
17407	26111	71, 26191	17497	71, 102, 79
17408	26114	72, 26197	17507	72, 102, 85
17409	26117	80, 26250	17556	80, 102, 138
17410	26120	26122	17478	102, 10
17411	26122	LIA	86	2, 86
17412	26124	STR		50
17413	26125	JRM	132	45, 132
17414	26127	LIA	86	2, 86
17415	26129	STR		50
17416	26130	RTN		55
17417	26131	LIA	237	2, 237
17418	26133	RTN		55
17419	26134	LIA	225	2, 225
17420	26136	RTN		55
17421	26137	LIA	211	2, 211
17422	26139	RTN		55
17423	26140	LIA	195	2, 195
17424	26142	RTN		55
17425	26143	LIA	184	2, 184
17426	26145	RTN		55
17427	26146	LIA	168	2, 168
17428	26148	RTN		55
17429	26149	LIA	155	2, 155
17430	26151	RTN		55
17431	26152	LIA	148	2, 148
17432	26154	RTN		55
17433	26155	LIA	139	2, 139
17434	26157	RTN		55
17435	26158	LIA	128	2, 128
17436	26160	RTN		55
17437	26161	LIA	120	2, 120
17438	26163	RTN		55
17439	26164	LIA	110	2, 110
17440	26166	RTN		55
17441	26167	LIA	102	2, 102
17442	26169	RTN		55
17443	26170	LIA	95	2, 95

26172	RTN	55
26173	LIA 87	2, 87
26175	RTN	55
26176	LIA 79	2, 79
26178	RTN	55
26179	LIA 73	2, 73
26181	RTN	55
26182	LIA 67	2, 67
26184	RTN	55
26185	LIA 61	2, 61
26187	RTN	55
26188	LIA 56	2, 56
26190	RTN	55
26191	LIA 51	2, 51
26193	RTN	55
26194	LIA 45	2, 45
26196	RTN	55
26197	LIA 41	2, 41
26199	RTN	55
26200	LIA 37	2, 37
26202	RTN	55
26203	LIA 32	2, 32
26205	RTN	55
26206	WAIT 0	78, 0
26208	RTN	55
26209	CALL 2621417520	120, 102, 102
26212	JRM 219	45, 219
26214	LIDP 2624617552	16, 102, 134
26217	STD	82
26218	LIB 0	3, 0
26220	LIA 200	2, 200
26222	LP 8	136
26223	EXAM	219
26224	LIA 255	2, 255
26226	INCP	80
26227	EXAM	219
26228	LIA 0	2, 0
26230	LIP 95	18, 95
26232	EXAM	219
26233	LIA 16	2, 16
26235	EXAM	219
26236	OUTC	223
26237	PUSH	52

17544	26238 LIA 1	2, 1
	26240 LP 8	136
	26241 ADB	28
	26242 POP	91
	26243 LIP 95	18, 95
	26245 WAIT 0	78, 0
	26247 JRNCM 13	43, 13
	26249 RTN	55
	26250 LP 4	132
	26251 EXAM	219
	26252 LP 5	133
	26253 LIA 9656	2, 96
	26255 EXAM	219
	26256 IX	4
	26257 LIB 0	3, 0
	26259 LP 8	136
	26260 MVBD	26
	26261 IXL	36
	26262 IXL	36
	26263 LIDP 26246 17552	16, 102, 134
17572	26266 STD	82
	26267 CALL 26228 17534	120, 102, 116
	26270 IXL	36
	26271 CPIA 0	103, 0
	26273 JRNZM 17	41, 17
	26275 RTN	55
	26276 TEST 8	107, 8
	26278 JRNZP 4	40, 4
	26280 JP 26018 17324	121, 101, 162
	26283 LIDP 26282 17568	16, 102, 170
	26286 LDD	87
	26287 CPIA 162-172	103, 162
	26289 JRZP 10	56, 10
	26291 LIA 162	2, 162
	26293 STD	82
	26294 LIA 255	2, 255
	26296 PUSH	52
	26297 LOOP 1	47, 1
	26299 JP 26018 17324	121, 101, 162
	26302 LIA 217 227	2, 217
	26304 STD	82
	26305 JRM 12	45, 12

Das Programm ist *ausnahmsweise für den PC 1350* gelistet. Bitte beachten Sie die für die anderen Rechner zu ändernden Adressen¹⁾!

Bedienung des Programms

Das fertig installierte Programm wird mit

CALL Anfangsadresse

gestartet. Nun können Sie mit den untersten Buchstabenreihen Ihres Rechners Melodien spielen. Wird die BRK/ON-Taste kurz gedrückt, so wird die Tastatur entweder auf die erste oder die

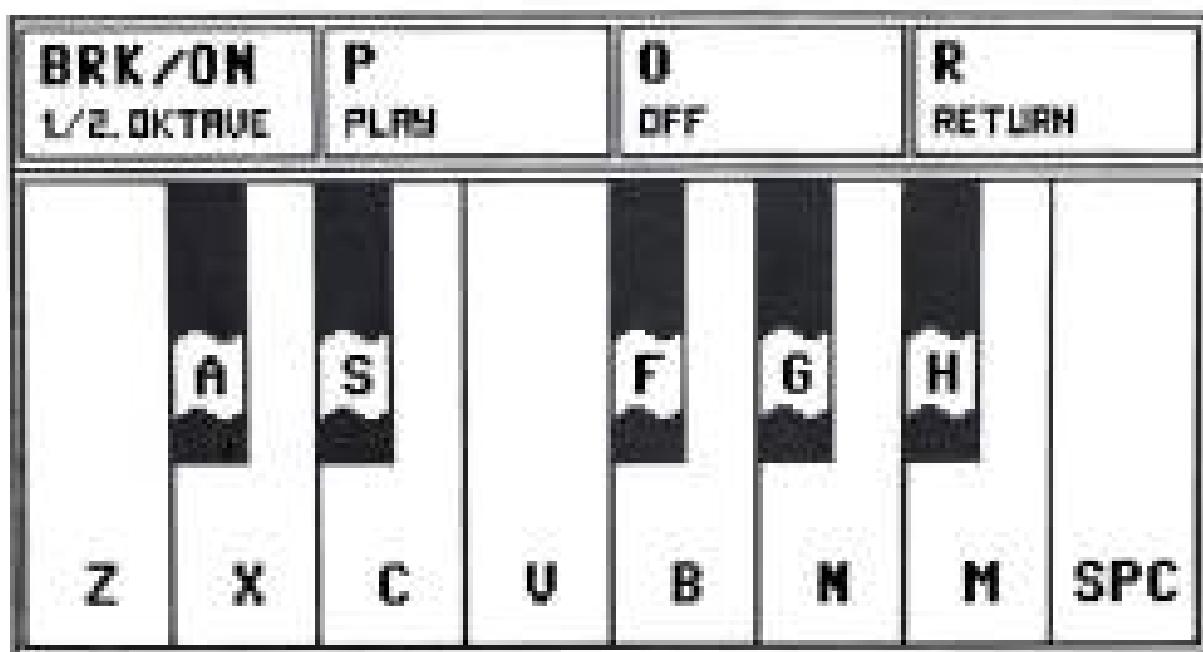
-
- 1) ↘ Adresse 26287: Dieser Wert ist das Lowbyte derjenigen Adresse, die im Listing als 26018 wiedergegeben ist.
- ↘ Adresse 26302: Dieser Wert ist das Lowbyte derjenigen Adresse, die im Listing als 26073 wiedergegeben ist.
- 17324 ↘ Adresse 26018: Diese Adresse sollten Sie sich notieren, das Lowbyte wird später noch gebraucht.
- 17325 ↘ Adresse 26073: Wie Adresse 26018.
- Adresse 26280: Wird das Programm mit ALL RESET (bei gewissen Rechnern inklusive Drücken der C-CE-Taste) abgebrochen, kann dieses Argument nachher unter Umständen einen andern Wert haben.
- 56 ↗ Adresse 26253: Highbyte der Anfangsadresse der Tontabelle für die gespeicherte Melodie.
- ↘ Adresse 26000: Highbyte des Standardvariablenbeginns.
- ↘ Adresse 26005: Lowbyte des Standardvariablenbeginns.
- Adresse 26065: Einsprungadresse des Unterprogramms OFF (im ROM).
- PC 140X: CAL 1434 (229,154), PC 1421: CAL 1479 (229,199), PC 126X: CAL 1112 (228,88)
- Adresse 26008: Einsprungadresse des Unterprogramms INKEY (im ROM).
- PC 140X: CAL 3387 (237,59), PC 1421: CAL 3359 (237,31), PC 126X: 3003 (235,187)
- Adresse 26010: Datapointer DP muß auf Standardvariablenbeginn zeigen.

zweite Oktave umgeschaltet. Nach dem Starten des Programms wird automatisch die erste Oktave eingeschaltet.

Hier nun die Tastaturregelung:

- Z, X, C, V, B, M, H, SPC: Tonläufe (C-Dur, eine Oktave)
- A, S, F, G, H: Halbtöne (immer über den entsprechenden Ganztönen)
- BRK/ON: Umschaltung erste/zweite Oktave
- P: PLAY, eine zuvor abgespeicherte Melodie kann angehört werden
- O: OFF, Ausschalten des Rechners
- R: RETURN, zurück ins BASIC

Die Tasten 'P', 'O' und 'R' sind nur benutzbar, wenn die tiefere Oktave gewählt ist.



Programmieren von Melodien

Um Melodien zu programmieren, die dann vom Programm aus über die Taste P angehört werden können, müssen die Tonlänge in Low- und Highbyte und die Tonhöhe in den Rechner gePOKEd werden. Aus Gründen der Übersichtlichkeit ist es wohl das Beste, diese Tontabelle zuerst auf einem Stück Papier zu notieren und dann erst im Rechner zu speichern. Falls Sie im Speicher noch genügend freien Speicherplatz haben, können Sie ja ein Hilfsprogramm schreiben, mit dem Sie die Tabelle eingeben können. Achten Sie aber darauf, daß Sie mit solchen Hilfsprogrammen nicht das Programm "Soundbox" überschreiben.

Speichern Sie die Tontabelle einer Melodie in jenen Speicherbereich, dessen Anfangsadresse Sie beim Installieren des Programms festgelegt hatten.

Nun das Format der Tontabelle:

1. Lowbyte der Tonlänge
2. Highbyte der Tonlänge
3. Höhe des Tones (0 = sehr hoher Ton, 255 = tiefer Ton)

Eine Melodientabelle muß an ihrem Ende immer mit drei aufeinanderfolgenden Nullen abgeschlossen werden.

Wichtig: Ein hoher Ton erklingt bei gleichem Wert für die Tonlänge wesentlich weniger lang als ein tiefer Ton. Achten Sie deshalb darauf, bei hohen Tönen den Wert für die Tonlänge etwas größer zu wählen.

Tabelle für die Tonhöhe

C	237	4	c	102	1
C#	225	5	c#	95	2
D	211	6	d	87	3
D#	195	7	d#	79	4
E	184	8	e	73	5
F	168	9	f	67	6
F#	155	10	f#	61	7
G	148	11	g	56	8
G#	139	12	g#	51	9
A	128	13	a	45	10
A#	120	14	a#	41	11
H	110	15	h	37	12

Neben diesen Werten für die Tonhöhe können selbstverständlich auch noch andere Zahlen verwendet werden, um feinere Abstufungen zu ermöglichen. Je nach Rechner muß man eine Melodie unter Umständen noch rein stimmen, also kleine Korrekturen der Tonhöhenwerte vornehmen.

Nun viel Spaß beim Musizieren!

9.8 Wecker

Mit dem Maschinenprogramm 'Wecker' bieten wir Ihnen einen äußerst komfortablen Wecker für den universellen Haus- oder Bürobetrieb: Dieses Weckerprogramm ist besser als manche teure Schaltuhr. Es ist nämlich möglich, 25 (!) Ein- oder Ausschaltzeiten für zwei getrennt schaltbare Netzspannungsgeräte vorzuprogrammieren.

Ein Beispiel, wie man den Wecker verwenden könnte:

Angenommen, Sie wollen die Beleuchtung Ihres Zimmers und Ihre Stereoanlage über unsere SHARP-PC-Schaltuhr steuern. Um 6.30 Uhr soll die Stereoanlage eingeschaltet werden, damit Sie sanft aus dem Schlaf geweckt werden, eine Viertelstunde später soll dann auch die Zimmerbeleuchtung einschalten, um Ihnen

anzuzeigen, daß es nun etwa Zeit zum Aufstehen wäre. Um 7.30 Uhr müssen Sie Ihr Heim spätestens verlassen, also sollen sowohl die Stereoanlage, als auch das Licht um 7.30 Uhr ausschalten. Wenn Sie dann um die Mittagszeit zum Mittagessen nach Hause kommen, soll Ihre Stereoanlage für die Zeit der neuesten Nachrichten, also für zirka 10 Minuten, eingeschaltet werden. Am Abend, wenn Sie um 18.00 Uhr von der Arbeit heimkommen, soll dann die Beleuchtung automatisch einschalten, da es um diese Zeit bereits dunkel ist. Wenn Sie sich um 22.00 Uhr zur Nachtruhe legen, so soll natürlich auch das Licht ausgeschaltet werden. Die Stereoanlage schaltet noch für eine halbe Stunde ein und wiegt Sie mit ein wenig Musik in den Schlaf.

Das wäre also ein praktisches Beispiel einer Anwendung dieses Weckerprogramms. Wie Sie sehen, ist es durchaus möglich, sehr viele verschiedene Ein- und Ausschaltzeiten zu programmieren. Maximal können zwei Geräte gesteuert werden.

Damit unser Weckerprogramm auch einwandfrei funktioniert, sind noch einige Probleme bezüglich der Hardware zu lösen. Die passenden Hardwarelösungen finden Sie in Kapitel 10.

Für den Bau des Weckers brauchen Sie also nebst dem eigentlichen Programm noch folgendes:

1. Das elektronische Relais

Die Bauanleitung dazu finden Sie in Kapitel 10.4. Dieses elektronische Relais wird an der seitlichen Schnittstelle des Rechners angeschlossen und sorgt für die Verstärkung der Steuersignale des Rechners, um die beiden Netzspannungsgeräte einzuschalten.

2. Die Quarzzeitbasis

Die Bauanleitung dazu finden Sie in Kapitel 10.5. Die Quarzzeitbasis garantiert den genauen Gang der Uhr. Die Quarzzeitbasis wird über das elektronische Relais mit dem Rechner ver-

bunden. Somit erhält der Computer die quarzgenauen Sekundenzählimpulse.

Installation des Programms

Diejenigen Unterprogrammaufrufe, die Unterprogramme im RAM anspringen, müssen geändert werden, wenn das Programm in einem andern Speicherbereich abgelegt wird, als dem im Listing angegebenen. Achten Sie also darauf, das Programm sorgfältig einzugeben.

Beachten Sie bitte auch hier wieder die zu ändernden Adressen¹⁾.

17400 LP 8	136
17401 LDM	89
17402 SWP	88
17403 LP 16	144
17404 ANIA 15	100, 15
17406 ORIA 48	101, 48
17408 EXAM	219
17409 LP 8	136
17410 ANIM 15	96, 15

1) Adresse 17464: PC 126X und PC 1350: Anfangsadresse des Zusatzprogramms

Adressen 17472 und 17618: PC 1421: 17795 (69,131), PC 126X: 25859 (101,3), PC 1350: 27699 (108,51)

Adresse 17485: PC 126X: 101 (101), PC 1350: 108 (108)

Adresse 17489: PC 1421: 128 (128), PC 126X: 10 (10), PC 1350: 48 (48)

Adresse 17520: PC 1421: 82 (82), PC 126X: 210 (210), PC 1350: 2 (2)

Adresse 17531: PC 1350: 4 (4)

Adresse 17535: PC 1350: 2 (2)

Adresse 17539: PC 1350: 8 (8)

Adresse 17548: PC 1350: 0 (0)

Adressen 17552 und 17553: PC 1350: LDM durch LIA 2 (2,2) ersetzen. An die Stelle von INCA in der Speicherzelle nach LDM tritt das Argument (also 2) des LIA-Befehls.

17412 ORIM 48	97,	48	
17414 EXAM	219		
17415 LP 17	145		
17416 EXAM	219		
17417 LP 9	137		
17418 LDM	89		
17419 SWP	88		
17420 LP 19	147		
17421 ANIA 15	100,	15	
17423 ORIA 48	101,	48	
17425 EXAM	219		
17426 LP 9	137		
17427 ANIM 15	96,	15	
17429 ORIM 48	97,	48	
17431 EXAM	219		
17432 LP 20	148		
17433 EXAM	219		
17434 LP 10	138		
17435 LDM	89		
17436 SWP	88		
17437 LP 22	150		
17438 ANIA 15	100,	15	
17440 ORIA 48	101,	48	
17442 EXAM	219		
17443 LP 10	138		
17444 ANIM 15	96,	15	
17446 ORIM 48	97,	48	
17448 EXAM	219		
17449 LP 23	151		
17450 EXAM	219		
17451 LP 24	152		
17452 LIA 32	2,	32	
17454 LII 8	0,	8	
17456 FILM	30		
17457 LIA 58	2,	58	
17459 LP 18	146		
17460 EXAM	219		
17461 LDM	89		
17462 LP 21	149		
17463 EXAM	219		
17464 CALL 32789	120,	128,	21
17467 CALL 17472	120,	68,	64
17470 JRP 13	44,	13	

17472 LIDP 17875	16,	69,	211
17475 LII 2	0,	2	
17477 LP 8	136		
17478 MWWD	24		
17479 LIA 1	2,	1	
17481 WAIT 0	78,	0	
17483 RTN	55		
17484 LP 5	133		
17485 LIA 69	2,	69	
17487 EXAM	219		
17488 LP 4	132		
17489 LIA 218	2,	218	
17491 EXAM	219		
17492 LP 8	136		
17493 LIA 2	2,	2	
17495 PUSH	52		
17496 IXL	36		
17497 CPMA	199		
17498 JRNZP 11	40,	11	
17500 INCP	80		
17501 LOOP 6	47,	6	
17503 IXL	36		
17504 LIP 94	18,	94	
17506 EXAM	219		
17507 OUTF	95		
17508 JRP 15	44,	15	
17510 POP	91		
17511 ADIA 5	116,	5	
17513 LP 4	132		
17514 WAIT 0	78,	0	
17516 LIB 0	3,	0	
17518 ADB	20		
17519 LP 4	132		
17520 CPIM 162	99,	162	
17522 JRNZM 31	41,	31	
17524 LIA 16	2,	16	
17526 LIP 92	18,	92	
17528 EXAM	219		
17529 OUTA	93		
17530 INA	76		
17531 CPIA 128	103,	128	
17533 JRZP 11	56,	11	
17535 CPIA 64	103,	64	

17537 JRZM 14	57,	14
17539 CPIA 32	103,	32
17541 JRZP 4	56,	4
17543 JRP 18	44,	18
17545 RTN	55	
17546 LIP 94	18,	94
17548 CPIM 3	99,	3
17550 JRZP 7	56,	7
17552 LDM	89	
17553 INCA	66	
17554 EXAM	219	
17555 OUTF	95	
17556 JRP 5	44,	5
17558 LIA 0	2,	0
17560 EXAM	219	
17561 OUTF	95	
17562 INB	204	
17563 TSIA 128	102,	128
17565 JRNZP 8	40,	8
17567 INB	204	
17568 TSIA 128	102,	128
17570 JRZM 4	57,	4
17572 JRP 6	44,	6
17574 INB	204	
17575 TSIA 128	102,	128
17577 JRNZM 4	41,	4
17579 LP 10	138	
17580 LIA 1	2,	1
17582 LII 0	0,	0
17584 ADN	12	
17585 LP 10	138	
17586 CPIM 96	99,	96
17588 JRNZP 29	40,	29
17590 LP 10	138	
17591 LIA 0	2,	0
17593 EXAM	219	
17594 LIA 1	2,	1
17596 LP 9	137	
17597 ADN	12	
17598 LP 9	137	
17599 CPIM 96	99,	96
17601 JRNZP 16	40,	16
17603 LIA 0	2,	0

17605 EXAM	219
17606 LP 8	136
17607 LIA 1	2, 1
17609 ADN	12
17610 LP 8	136
17611 CPIM 36	99, 36
17613 JRNZP 4	40, 4
17615 LIA 0	2, 0
17617 EXAM	219
17618 LIDP 17875	16, 69, 211
17621 LII 2	0, 2
17623 LP 8	136
17624 EXWD	25
17625 CALL 17472	120, 68, 64
17628 LIP 95	18, 95
17630 ORIM 1	97, 1
17632 DUTC	223
17633 JRM 234	45, 234

Damit die Anzeige der Zeit einwandfrei funktioniert, brauchen die Rechner PC 1260/61 einerseits, der PC 1350 andererseits noch ein zusätzliches Unterprogramm, während bei den Rechnern PC 1401/02/21 direkt in das Unterprogramm im ROM gesprungen werden kann. Im Folgenden drucken wir diese zusätzlichen Unterprogramme für den PC 1350 und PC 1260/61 ab. Bei den Rechnern PC 1401/02/21 werden diese Zusätze nicht benötigt und können deshalb weggelassen werden.

Es folgen die Listings der zusätzlichen Unterprogramme.

PC 126X:

25700	LIDP 26192	16, 102, 80
25703	LII 47	0, 47
25705	LIA 32	2, 32
25707	FILD	31
25708	LIDP 26200	16, 102, 88
25711	LP 16	144
25712	LII 10	0, 10
25714	EXWD	25
25715	CALL 32783	120, 128, 15
25718	RTN	55

PC 1350:

25700	LIDP 27904	16, 109, 0
25703	LII 95	0, 95
25705	LIA 32	2, 32
25707	FILD	31
25708	LIDP 27910	16, 109, 6
25711	LP 16	144
25712	LII 10	0, 10
25714	EXWD	25
25715	LP 8	136
25716	LIA 0	2, 0
25718	EXAM	219
25719	LIA 4	2, 4
25721	CALL 53958	120, 210, 198
25724	RTN	55

Bedienung des Programms

1. Anschluß des Rechners

Ihr Rechner muß an das elektronische Relais angeschlossen werden. Über den Anschluß IB7 INPUT des Relais wird das Taktsignal für die Uhr in den Rechner geleitet. Schließen Sie deshalb diesen Anschluß (IB7 INPUT) mit dem 0.5 Hz-Ausgang (A3) der Quarzzeitbasis zusammen.

2. Eingabe der Zeit

Alle Eingaben können nur in BASIC durchgeführt werden. Die aktuelle Zeit wird in der Standardvariable Z gespeichert, und zwar in der folgenden Form:

Z = 99103625

Stunden: 10

Minuten: 30

Sekunden: 25

Der Dezimalpunkt kann an jeder beliebigen Stelle stehen.

3. Eingabe der Ein- und Ausschaltzeiten

Die verschiedenen Ein- und Ausschaltzeiten werden vor dem Start des Programms in den Variablen A-Y (die Reihenfolge ist beliebig) gespeichert. Wie Sie sofort sehen, können Sie also maximal 25 Ein- oder Ausschaltzeiten vorprogrammieren.

Hat eine dieser Variablen (A-Y) den Wert 0, so bedeutet das, daß beide Netzespannungsgeräte, die am elektronischen Relais angeschlossen sind, um Mitternacht ausgeschaltet werden. Wenn Sie dies vermeiden wollen, müssen Sie das entsprechende Gerät über eine Variable um Mitternacht einschalten. Wenn Sie so

vorgehen, entsteht kein Unterbruch; Ihr Gerät bleibt auch über Mitternacht eingeschaltet.

Wichtig: Mitternacht ist bei dieser Uhr um 0.00 Uhr, und nicht etwa um 24.00 Uhr.

Die Eingabe einer Ein- oder Ausschaltzeit sieht folgendermaßen aus (hier am Beispiel der Variable X gezeigt; selbstverständlich könnte auch eine andere Standardvariable (außer Z) verwendet werden):

X = 9918300501

Stunden: 18

Minuten: 30

Sekunden: 05

Gerät: 01

Hinweis zu den beiden Ziffern, die das Gerät betreffen:

PC 1350: Werden die beiden Geräteziffern weggelassen, so schalten beim angegebenen Zeitpunkt beide Geräte aus. Für das Einschalten gilt:

02: 2. Gerät

04: 1. Gerät

06: beide Geräte

Für alle übrigen Rechner gilt:

Auch hier werden beide Geräte beim angegebenen Zeitpunkt ausgeschaltet, wenn die beiden Geräteziffern weggelassen werden. Für das Einschalten sieht es so aus:

01: 1. Gerät

02: 2. Gerät

03: beide Geräte

4. Starten des Programms

Das Programm wird mit derjenigen Adresse gestartet, die im Listing mit einem Sternchen gekennzeichnet ist. Nach dem Start des Programms erscheint die Anzeige der Zeit in der folgenden Form:

15:33:17

Stunden: 15
Minuten: 33
Sekunden: 17

Wenn Sie alles richtig eingegeben und die Peripheriegeräte richtig angeschlossen und eingeschaltet haben, so sollte die Zeit nun laufen. Falls dies nicht funktioniert, prüfen Sie zuerst die Anschlüsse des elektronischen Relais, falls der Fehler nicht dort zu finden ist, sollten Sie das Programm durch das gleichzeitige Drücken von C-CE und des ALL-RESET-Knopfes abbrechen und überprüfen.

Wenn das Programm fehlerfrei läuft, können Sie Ihren Wecker mit folgenden Tasten beeinflussen:

Tastaturbelegung beim PC 1350:

M: Anhalten der Uhr
SPC: Rückkehr ins BASIC
ENTER: Durch Drücken der ENTER-TASTE können Sie die angeschlossenen Geräte manuell ein- oder ausschalten.

Tastaturbelegung bei den übrigen Rechnern:

K: Anhalten der Uhr
SPC: Zurück ins BASIC
I: Manuelles Ein- oder Ausschalten der Geräte

Hinweise:

Bei der Rückkehr ins BASIC bleiben sowohl die Uhrzeit, als auch alle Ein- und Ausschaltzeiten in den Variablen erhalten.

Die Tasten werden nur nach dem Eintreffen eines Taktimpulses (IB7 INPUT) abgefragt. Die Tasten müssen deshalb genügend lange gedrückt werden (circa 1 Sekunde).

Falls, aus welchen Gründen auch immer, keine Taktimpulse eintreffen, kann man nur durch gleichzeitiges Drücken der roten C-CE-Taste und der ALL-RESET-Taste ins BASIC zurückgelangen.

Kapitel 10

Hardware: Mit Draht und Lötzinn

Hinweise zum Löten

Wer keine Übung im Umgang mit einem Lötkolben hat, sollte diesen Abschnitt zuerst durchlesen, bevor er sich ans Löten wagt.

1. Verwendet werden sollte Elektroniklot mit 60% Zinn und 40% Blei mit Kolophoniumseele, keinesfalls aber Lötfett oder andere 'Löthilfen'.
2. Der Lötkolben sollte eine Leistung von 10-50 Watt bei einer Temperatur von zirka 350 Grad Celsius haben. Die Lötspitze muß sauber sein.
3. Ein frisch eingelötetes Bauteil darf nicht bewegt werden, bis das Lot hart ist.
4. An einer Lötstelle darf höchstens 5 Sekunden gelötet werden, sonst lösen sich die Leiterbahnen ab, oder die Bauteile werden zerstört.
5. Für die ICs CD 4040, CD 4060 (Quarzzeitbasis) und LM 393 (Cassetteninterface) sollten Sie unbedingt IC-Sockel verwenden.

 Wenn Sie sich nicht sicher fühlen, probieren Sie das Löten zuerst an wertlosem Material aus.

10.1 Pinbelegung der Schnittstelle

Besprechung der einzelnen Anschlüsse (Pins) der Schnittstelle (von oben nach unten numeriert):

Pin 2: Speisespannung +6 Volt

Pin 3: Masseleitung

Pin 4: Bit 0 des F0-Ports (F0 0); Bit gesetzt: +5 Volt
(beim PC 1350: Bit 2) Bit gelöscht: 0 Volt

Pin 5: Bit 1 des F0-Ports (F0 1)

Pin 6: LOAD; über diesen Pin gelangen die Daten von der Cassette
in den

Rechner

Pin 7: SAVE; über diesen Pin gelangen die Daten vom Rechner auf
das

Cassettenband

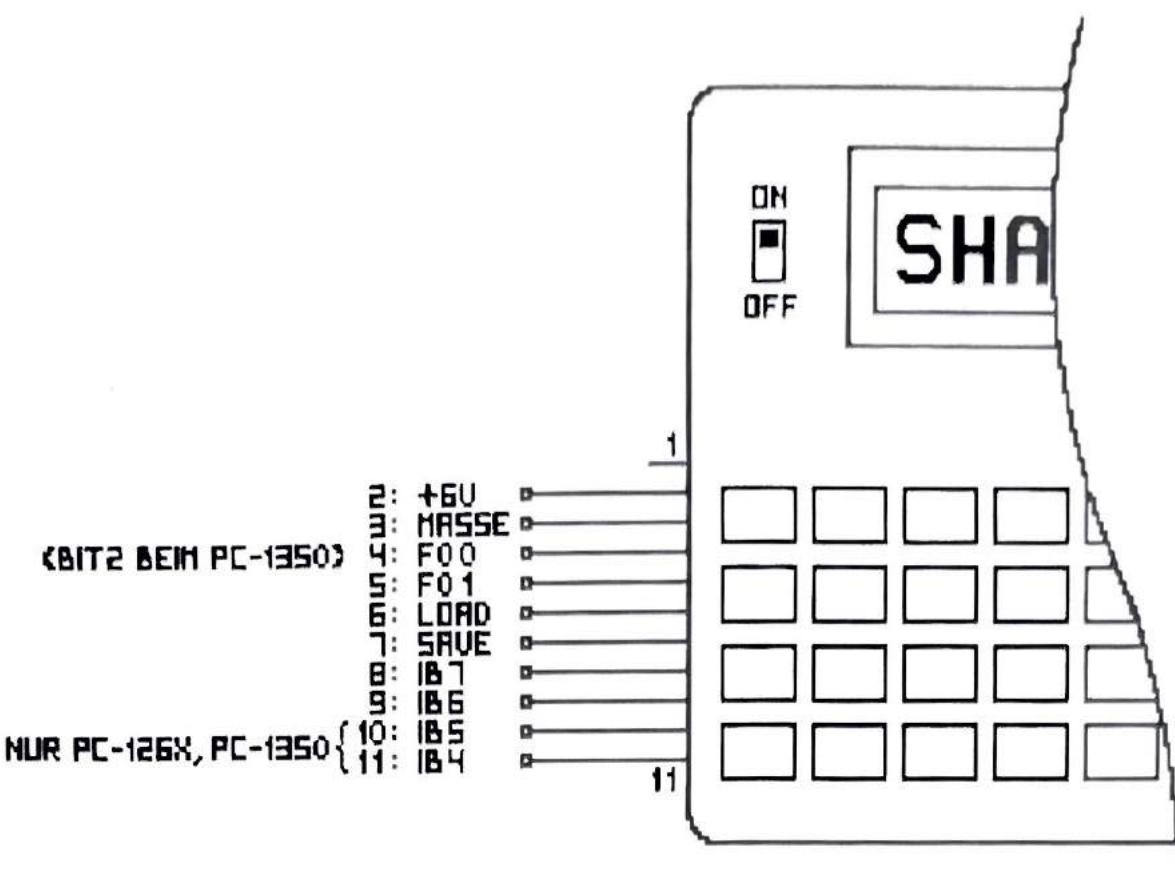
Pin 8: Bit 7 des IB-Ports (IB 7); Bit gesetzt: +5 Volt
Bit gelöscht: 0 Volt

Pin 9: Bit 6 des IB-Ports (IB 6)

Pin 10: Bit 5 des IB-Ports (IB 5) nur bei den Rechnern

Pin 11: Bit 4 des IB-Ports (IB 4) PC 126X und PC 1350

Wichtig: Beim PC 1350 ist Pin 4 mit dem 2. Bit des F0-Ports
verbunden.



10.2 Cassetteninterface

Wer keine Lust oder Zeit hat, lange (oder auch kurze) Programme immer wieder auf's neue einzutippen, wer auch kein Geld für den Drucker mit eingebautem Cassetteninterface hat, dem können wir hier folgende Alternative anbieten:

Bauen Sie selbst ein Cassetteninterface! Sie können damit alle Ihre Programme auf einem handelsüblichen Cassettenrecorder abspeichern.

Ist das Interface eingeschaltet, so wird Ihr Rechner über Pin 2 der Schnittstelle von der externen Batterie des Interfaces gespeist. Es lohnt sich nämlich enorm, dem Rechner die Speisespannung von außen her zuzuführen, da die 9-Volt-Blockbatterie, mit der das Interface gespeist wird, wesentlich billiger ist als die eingebauten, teuren Lithiumbatterien des Rechners.

Betreiben Sie Ihren Rechner deshalb immer mit eingeschaltetem Interface. Der Stromverbrauch der Interfaceschaltung selbst ist relativ gering, sodaß er kaum ins Gewicht fällt.

Falls Sie das Cassetteninterface zusammen mit dem elektronischen Relais in dasselbe Gehäuse bauen wollen, so kann der Spannungsregler des Interfaces weggelassen werden; die Speise spannung für das Interface wird dann vom Anschluß (A) des elektronischen Relais abgezapft.

Das Cassetteninterface besteht grundsätzlich aus drei verschiedenen Teilschaltungen:

1. Spannungsregler (uA 7806)
2. Komparator (LM 393) zum Verstärken der Informationen von der Cassette (CLOAD)
3. Schaltung zur Umwandlung der Computerinformationen in Cassetteninformationen (CSAVE)

Bedienungsanleitung:

- Verbinden Sie das Interface mit dem Computer und mit dem Cassettenrecorder. Wie das Interface angeschlossen wird, können Sie dem Schaltplan entnehmen.
- Schalten Sie das Interface, den Rechner und falls notwendig auch den Recorder ein.
- Da dieses Interface, im Gegensatz zum SHARP-Interface im Drucker CE-126P, den Recorder nicht automatisch startet, müssen Sie das Cassettengerät kurz vor dem Speichern oder Laden starten und erst nach dem CSAVE-oder CLOAD-Vorgang wieder anhalten.
- Der Lautstärkeregler am Cassettengerät sollte auf Mitte bis Maximum eingestellt sein.

Speichern:

Starten Sie das Cassettengerät und geben Sie danach sofort CSAVE und ENTER ein. CSAVE funktioniert nur, wenn ein Programm im Speicher ist.

Laden:

Zum Laden muß auf der Cassette zuerst das richtige Programm gesucht werden. Es ist deshalb von Vorteil, einen Recorder mit Bandzählwerk zu verwenden. Starten Sie zuerst den Rechner mit CLOAD und ENTER, dann erst das Cassettengerät.

Tritt beim Laden ein Fehler auf, wird die Fehlermeldung ERROR 8 angezeigt.

Stückliste zum Cassetteninterface

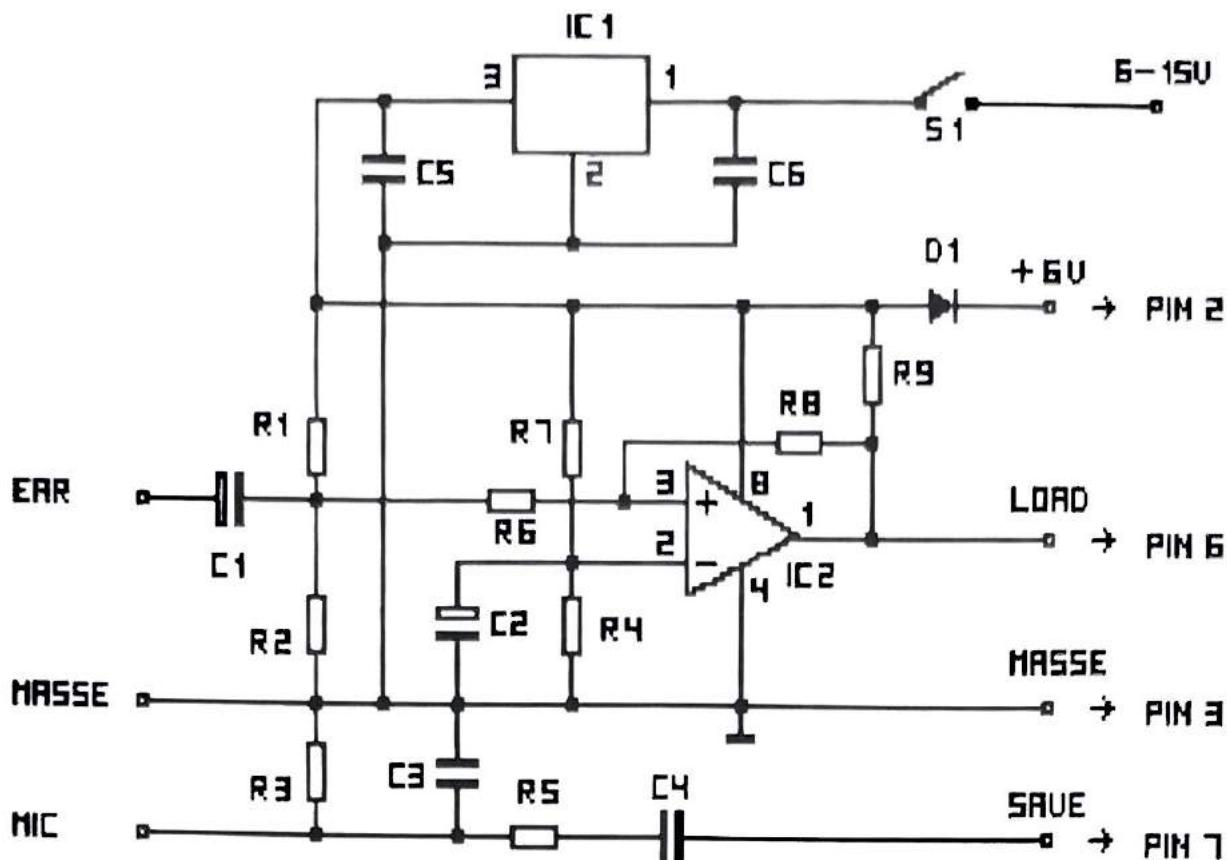
ICs: IC1 uA 7806
 IC2 LM 393

Dioden: D1 1N4148

Widerstände: R1, R2 15 kOhm
 R3 2.2 kOhm
 R4, R7, R9 12 kOhm
 R5 27 kOhm
 R6 3.3 kOhm
 R8 470 kOhm

Kondensatoren: C1 10 uF
 C2 22 uF
 C3, C5 100 nF
 C4 4.7 nF
 C6 330 nF

Sonstiges: 2 Klinkenstecker 3.5 mm
1 IC-Fassung DIL 8



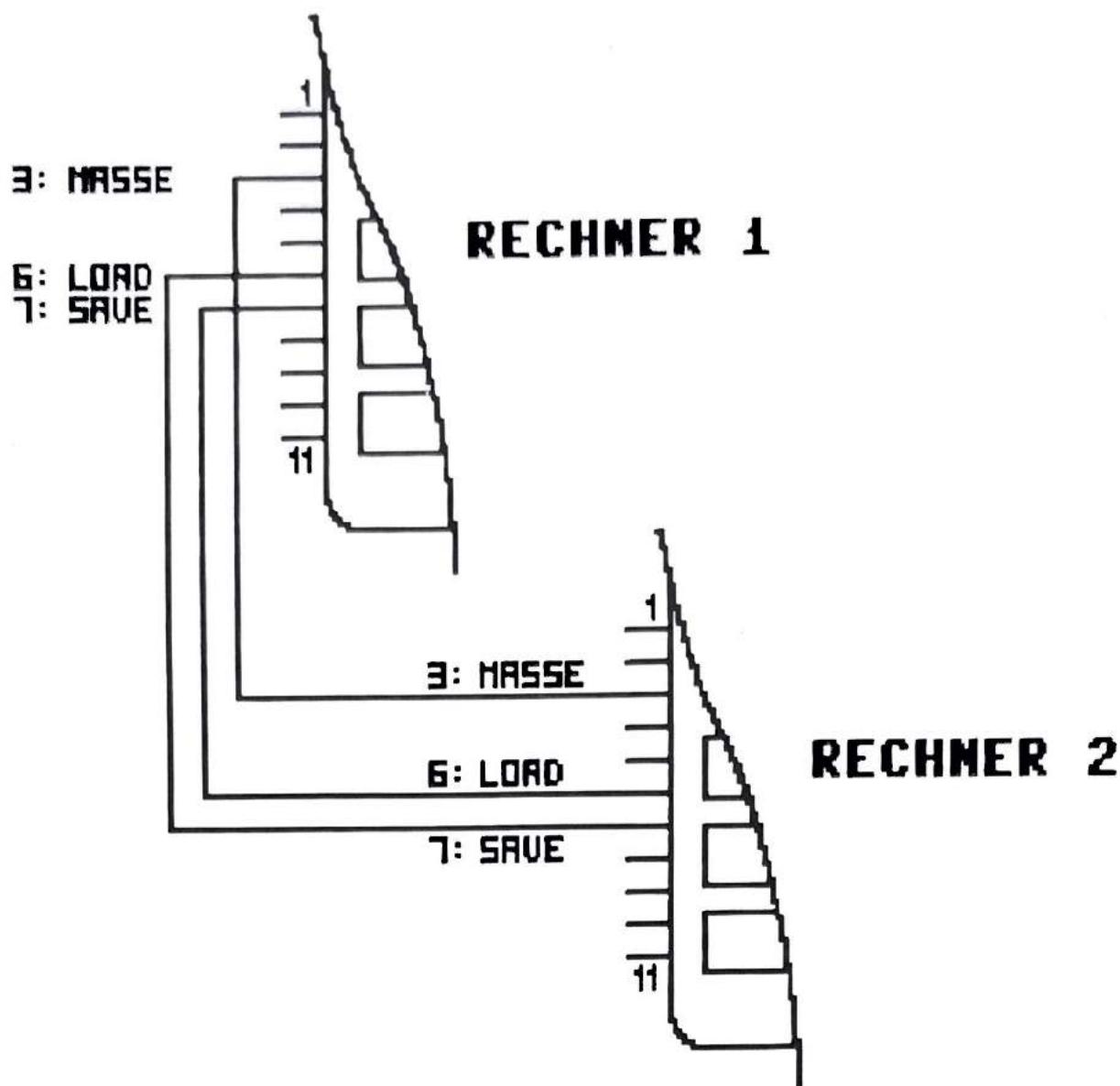
10.3 Datenaustauschkabel für zwei Rechner

Durch das Verbinden des LOAD-Eingangs eines ersten Rechners mit dem SAVE-Ausgang eines zweiten können auf einfache Art Programme direkt von einem Rechner in den andern kopiert werden.

Der Rechner, auf den die Programme kopiert werden sollen, wird dabei genauso angesprochen wie ein Cassettengerät. Daten können in beide Richtungen (sowohl SAVE als auch LOAD) ausgetauscht werden.

Stückliste

2 Stiftleisten 11polig, Rastermaß 2.54 mm
1 Flachbandkabel 3adrig, zirka 30 cm lang



10.4 Elektronisches Relais

Mit dem elektronischen Relais können wir Ihnen eine flexible Hardwaresteuerschaltung für Ihren SHARP PC anbieten. Mit diesem Relais haben Sie die Möglichkeit, zwei (maximal vier) getrennt schaltbare Netzspannungsgeräte anzusteuern. Über dieses Relais können aber auch Messwerte und Steuerspannungen von externen Geräten in den Rechner geladen und ausgewertet werden.

werden; das elektronische Relais besitzt zwei Eingänge für Steuerimpulse (IB 6 INPUT, IB 7 INPUT).

Das elektronische Relais besteht aus drei Teilschaltungen:

1. Netzteil: Die Netzspannung (220 V) wird auf 9 Volt heruntertransformiert. Der Spannungsregler uA 7806 stabilisiert und begrenzt die Spannung auf +6 Volt. Damit wird sowohl das Relais, als auch der Rechner gespeist.
2. Relais mit Treibertransistoren: Die Treibertransistoren verstärken den Strom, der vom F0-Port des Rechners herkommt; somit ziehen die Relaisspulen an, und die angeschlossenen Geräte werden eingeschaltet.
3. I/O-Pufferverstärker: Der Pufferverstärker sorgt dafür, daß der Rechner gegen zu hohe Steuerspannungen externer Geräte geschützt ist (IB-Port).

Vorsicht: Passen Sie auf beim Zusammenbau des elektronischen Relais: Es wird mit 220 Volt Netzspannung betrieben!

Falls Sie die Ein- und Ausgänge direkt an der Schnittstelle des Rechners anzapfen, sollten Sie sich darüber im klaren sein, daß der Rechner nur aus CMOS-ICs besteht. Bei unvorsichtigem Hantieren (z.B. auf synthetischen Teppichen) kann eine statische Ladung mit einer Spannung von einigen 100 bis 1000 Volt entstehen. Da dieser Strom so schwach ist, wird er vom menschlichen Körper nicht wahrgenommen; zur Zerstörung der CMOS-ICs und damit des Rechners reicht er aber völlig aus.

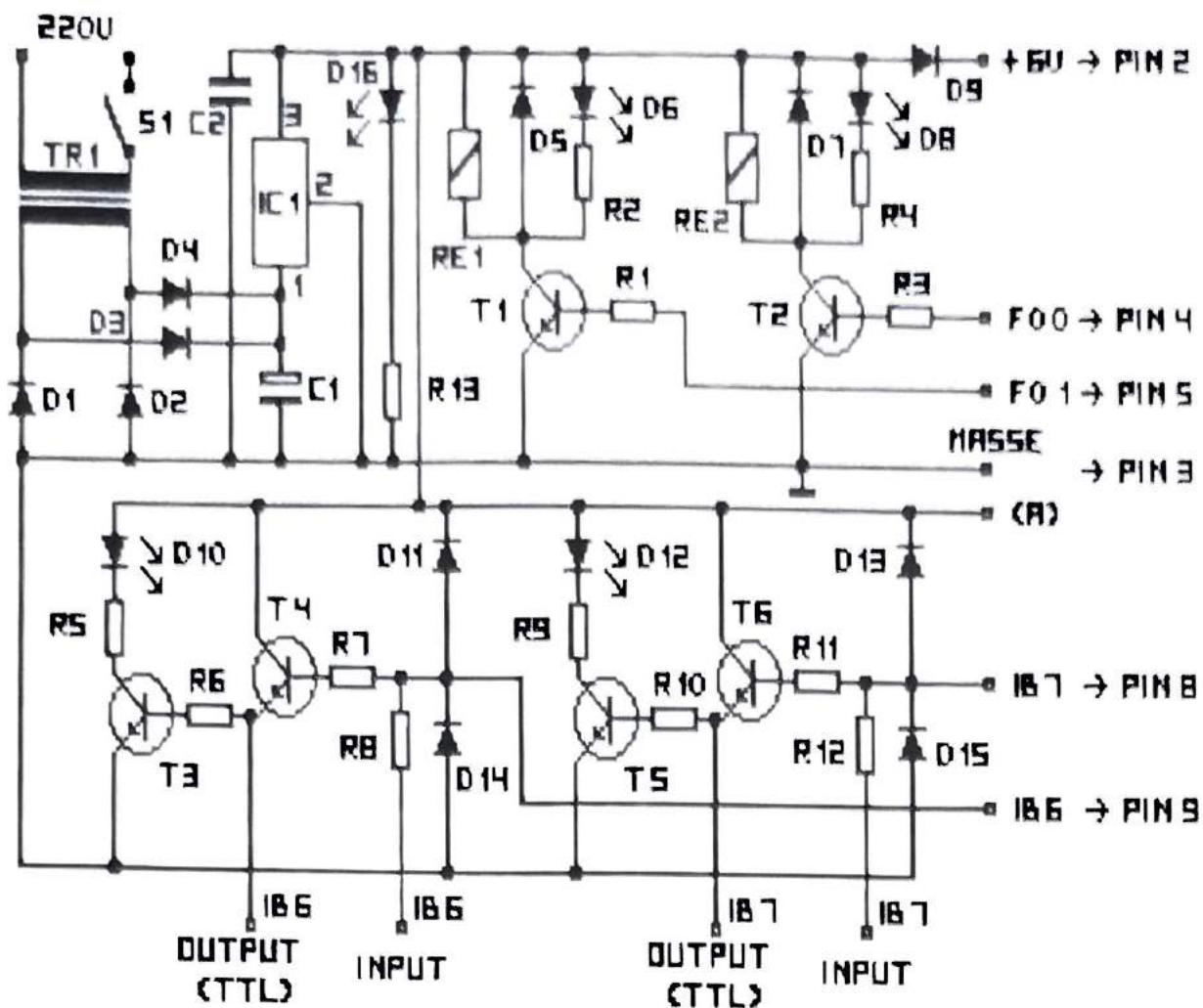
Ein Rechnerausgang darf auch nicht direkt (d.h. ohne das elektronische Relais) zur Ansteuerung eines TTL-ICs benutzt werden, da diese bereits zuviel Schaltstrom benötigen.

Es ist natürlich von Vorteil, das elektronische Relais zusammen mit dem Cassetteninterface und der Quarzzeitbasis (für das

Programm 'Wecker') in dasselbe Gehäuse einzubauen. Die Speisespannung für Cassetteninterface und Zeitbasis wird am Anschluß (A) des Relais abgezapft.

Stückliste zum elektronischen Relais

ICs:	IC1	uA 7806
Dioden:	D1-D4 D5, D7, D9, D11, D13, D14, D15 D6, D7, D10, D12, D16	Gleichrichter 4DV/1A 1N4148 Leuchtdioden
Transistoren:	T1-T6	BC 238
Widerstände:	R1, R3 R2, R4, R5, R9 R13 R6, R10 R7, R11 R8, R12	10 kOhm 150 Ohm 1 kOhm 1 MOhm 47 kOhm/0.5W
Kondensatoren:	C1 C2	4700 uF/15V 100 nF
Sonstiges:	TR1 RE1, RE2 2.54 mm	Transformator 9V/10W Relais, Spule 6V, Schaltleistung 250V/6A 1 Stiftleiste 11polig, Rastermaß 1 Schalter 1xEin 1 Netzstecker 220 V Buchsen für INPUT/OUTPUT- Anschlüsse



Anmerkung: An die Relais können beliebige Geräte, also auch Netzspannungsgeräte, angeschlossen werden.

10.5 Quarzzeitbasis

Die hier beschriebene Quarzzeitbasis wird für das Programm 'Wecker' (Kapitel 9.8) benötigt. Diese Zeitbasis liefert über den IB-Port (IB 7) des Rechners die Zählimpulse für die Uhr.

Die Quarzzeitbasis wird zusammen mit dem elektronischen Relais ins gleiche Gehäuse gebaut. Die Speisespannung für die Zeitbasis wird vom Anschluß (A) des elektronischen Relais genommen.

Die Zeitbasis besitzt drei verschiedene Ausgänge:

A1: Frequenz 2 Hz

A2: Frequenz 1 Hz

A3: Frequenz 0,5 Hz

Vorsicht: Die ICs CD 4040 und CD 4060 sind CMOS-ICs. Ihre Anschlüsse dürfen also nicht mit den Fingern berührt werden, da sie sonst durch elektrostatische Entladungen zerstört werden könnten.

Stückliste zur Quarzzeitbasis

ICs: IC1 CD 4040

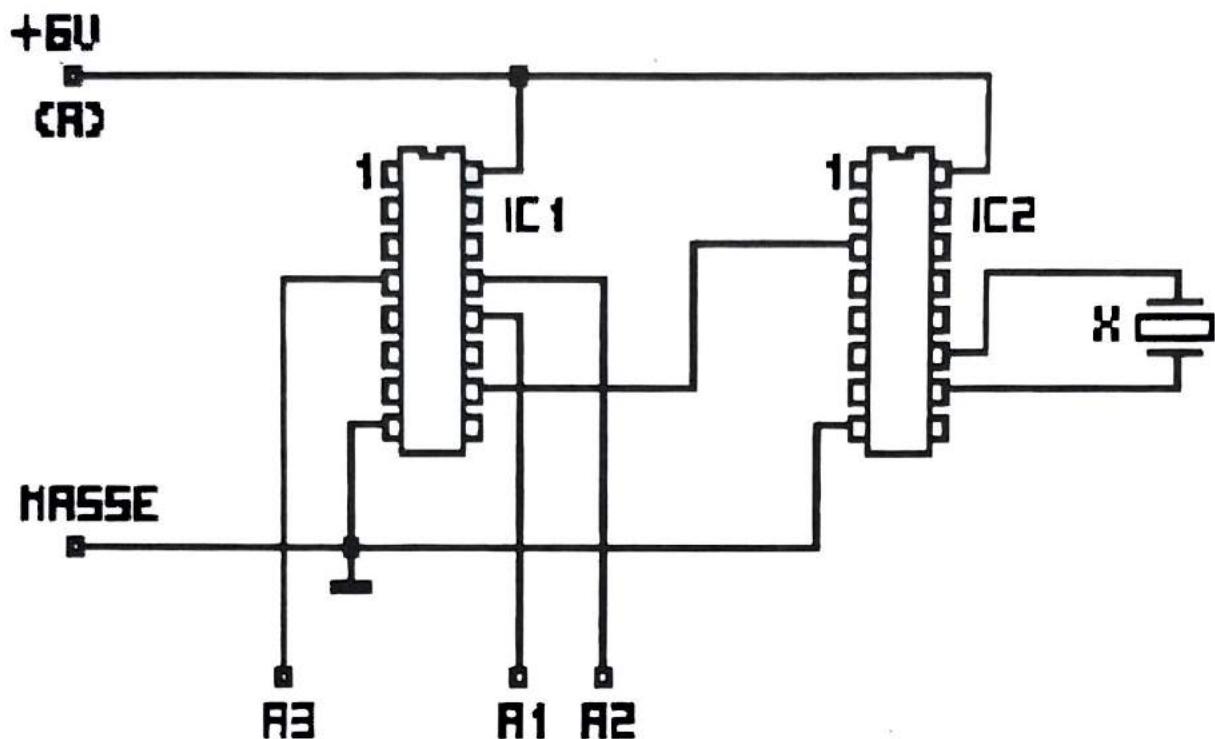
IC2 CD 4060

Quarz: X

Uhrenquarz:

Frequenz: 4.194304 MHz

Sonstiges: 2 IC-Fassungen DIL 16



Anhang - Tabellen, Listen, Übersichten

Anhang A

Maschinenbefehle eine Übersicht

Auf den folgenden Seiten finden Sie eine nach Mnemonics geordnete Übersicht über den Befehlssatz der CPU SC 61860. Zu jedem Befehl ist der Code, die Wirkung, die Beeinflussung der Flags, die Anzahl Bytes (#) und die Anzahl (Takt-) Zyklen angegeben. Besonders wichtige Einzelheiten finden Sie im Feld Bemerkung.

Ein Taktzyklus entspricht zirka 5,2 Mikrosekunden (1 Mikrosekunde = 1 Millionstel Sekunde). Stehen bei einem Befehl im Feld 'Zyklen' zwei Zahlen (z.B. 7/4), so folgt daraus, daß es zwei Möglichkeiten gibt, diesen Befehl auszuführen. Für beide Möglichkeiten gelten verschiedene Taktangaben. Kommt der Faktor I vor (z.B. 4+3*I), so ist die benötigte Anzahl Zyklen vom Inhalt des I-Registers abhängig.

Mnemonic	Code	Wirkung	Flags	#	Zyklen	Bemerkung
			C	Z		
ADB	20	(P)+A->(P) (P+1)+B+C->(P+1)	* *	1	5	P+1->P
ADCM	196	(P)+A+C->(P)	* *	1	3	
ADIA n	116,n	A+n->A	* *	2	4	
ADIM n	112,n	(P)+n->(P)	* *	2	4	
ADM	68	(P)+A->(P)	* *	1	3	
ADN	12	(P)+A->(P)... (P-I)+C->(P-I)	* *	1	7+3*I	P-I-1->P BCD
ADW	14	(P)+(Q)->(P)... (P-I)+(Q-I)+C-> (P-I)	* *	1	7+3*I	P-I-1->P Q-I-2->Q BCD
ANIA n	100,n	A AND n->A	- *	2	4	
ANID n	212,n	(DP) AND n->(DP)	- *	2	6	
ANIM n	96,n	(P) AND n->(P)	- *	2	4	
ANMA	70	(P) AND A->(P)	- *	1	3	

CAL nm	224+n,m PC+2->(R-1,R-2)	- -	2	7	0<=n<=31
	R-2->R, nm->PC				
CALL nm	120,n,m PC+3->(R-1,R-2),	- -	3	8	
	R-2->R, nm->PC				
CPIA n	103,n A-n	* *	2	4	
CPIM n	99,n (P)-n	* *	2	4	
CPMA	199 (P)-A	* *	1	3	
DATA	53 (BA)...(BA+1)->	- -	1	11+4*I	auch für (P)...(P+1)
					int. ROM
DECA	67 I-1->I	* *	1	4	2->Q
DECB	195 B-1->B	* *	1	4	3->Q
DECI	65 I-1->I	* *	1	4	0->Q
DECJ	193 J-1->J	* *	1	4	1->Q
DECK	73 K-1->K	* *	1	4	8->Q
DECL	201 L-1->L	* *	1	4	9->Q
DECP	81 P-1->P	- -	1	2	
DECV	75 V-1->V	* *	1	4	10->Q
DECW	203 W-1->W	* *	1	4	11->Q
DX	5 X-1->X. X->DP	- -	1	6	5->Q
DXL	37 X-1->X. X->DP.	- -	1	7	5->Q
	(DP)->A				
DY	7 Y-1->Y. Y->DP	- -	1	6	7->Q
DYS	39 Y-1->Y. Y->DP.	- -	1	7	7->Q
	A->(DP)				
ETC	105 FOR i=1 TO H	- *	1)	1)	
n,nm	n,nm IF A=n nm->PC				
---	NEXT i				
nm	nm->PC				
EXAB	218 A<->B	- -	1	3	
EXAM	219 A<->(P)	- -	1	3	
EXB	11 (P)...(P+J)<->	- -	1	6+3*I	P+J+1->P
	(Q)...(Q+J)				Q+J+1->Q
EXBD	27 (P)...(P+J)<->	- -	1	7+6*I	P+J+1->P
	(DP)...(DP+J)				DP+J->DP
EXW	9 (P)...(P+I)<->	- -	1	6+3*I	P+I+1->P
	(Q)...(Q+I)				Q+I+1->Q

- 1) Der ganze Befehl ETC nimmt 3+3*H Bytes in Anspruch. Die Ausführungszeit beträgt 3 Zyklen für den ETC-Befehl selbst sowie je 6 Zyklen für jede Tabellenposition und für die Adresse nm (bei nicht gefunden).

EXWD	25	(P)...(P+I)<-> (DP)...(DP+I)	- -	1	7+6*I	P+I+1->P DP+I->DP
FILD	31	A->(DP)...(DP+I)	- -	1	4+3*I	DP+I->DP
FILM	30	A->(P)...(P+I)	- -	1	5+I	P+I+1->P
INA	76	IA-Port->A	- *	1	2	
INB	204	IB-Port->A	- *	1	2	
INCA	66	I+1->I	* *	1	4	2->Q
INCB	194	B+1->B	* *	1	4	3->Q
INCI	64	I+1->I	* *	1	4	0->Q
INCJ	192	J+1->J	* *	1	4	1->Q
INCK	72	K+1->K	* *	1	4	8->Q
INCL	200	L+1->L	* *	1	4	9->Q
INCP	80	P+1->P	- -	1	2	
INCV	74	V+1->V	* *	1	4	10->Q
INCW	202	W+1->W	* *	1	4	11->Q
IX	4	X+1->X. X->DP	- -	1	6	5->Q
IXL	36	X+1->X. X->DP. (DP)->A	- -	1	7	5->Q
IY	6	Y+1->Y. Y->DP	- -	1	6	7->Q
IYS	38	Y+1->Y. Y->DP. A->(DP)	- -	1	7	7->Q
JP nm	121,n,m	nm->PC	- -	3	6	
JPC nm	127,n,m	IF C=1 nm->PC	- -	3	6	
JPNC nm	125,n,m	IF C=0 nm->PC	- -	3	6	
JPNZ nm	124,n,m	IF Z=0 nm->PC	- -	3	6	
JPZ nm	126,n,m	IF Z=1 nm->PC	- -	3	6	
JPZM n	57,n	IF Z=1 PC+1-n->PC	- -	2	7/4	
JRCM n	59,n	IF C=1 PC+1-n->PC	- -	2	7/4	
JRCP n	58,n	IF C=1 PC+1+n->PC	- -	2	7/4	
JRM n	45,n	PC+1-n->PC	- -	2	7	
JRNCM n	43,n	IF C=0 PC+1-n->PC	- -	2	7/4	
JRNCP n	42,n	IF C=0 PC+1+n->PC	- -	2	7/4	
JRNZM n	41,n	IF Z=0 PC+1-N->PC	- -	2	7/4	

JRNZP n	40,n	IF Z=0 PC+1+n->PC	- -	2	7/4	
JRP n	44,n	PC+1+n->PC	- -	2	7	
JRZM n	57,n	F Z=1 PC+1-n->PC	- -	2	7/4	
JRZP n	56,n	IF Z=1 PC+1+n->PC	- -	2	7/4	
LDD	87	(DP)->A	- -	1	3	
LDM	89	(P)->A	- -	1	2	
LDP	32	P->A	- -	1	2	
LDQ	33	Q->A	- -	1	2	
LDR	34	R->A	- -	1	2	
LIA n	2,n	n->A	- -	2	4	
LIB n	3,n	n->B	- -	2	4	
LIDL m	17,m	m->DPL	- -	2	5	
LIDP nm	16,n,m	nm->DP	- -	3	8	
LII n	0,n	n->I	- -	2	4	
LIJ n	1,n	n->J	- -	2	4	
LIP n	18,n	n->P	- -	2	4	
LIQ n	19,n	n->Q	- -	2	4	
LOOP n	47,n	(R)-1->(R), IF C=0 DP+1-n->DP	* *	2	10/7	
LP n	128+n	n->P	- -	1	2	0<=n<=63
MVB	10	(Q)...(Q+J)-> (P)...(P+J)	- -	1	5+2*I	P+J+1->P Q+J+1->Q
MVBD	26	(DP)...(DP+J)-> (P)...(P+J)	- -	1	5+4*I	P+J+1->P DP+J->DP
MVDM	83	(P)->(DP)	- -	1	3	
MVMD	85	(DP)->(P)	- -	1	3	
MVW	8	(Q)...(Q+I)-> (P)...(P+I)	- -	1	5+2*I	P+I+1->P Q+I+1->Q
MVWD	24	(DP)...(DP+I)-> (P)...(P+I)	- -	1	5+4*I	P+I+1->P DP+I->DP
NOPT	206	No Operation	- -	1	3	
NOPW	77	No Operation	- -	1	2	
ORIA n	101,n	A OR n->A	- *	2	4	
ORID n	213,n	(DP) OR n->(DP)	- *	2	6	
ORIM n	97,n	(P) OR n->(P)	- *	2	4	
ORMA n	71,n	(DP) OR n->(DP)	- *	1	3	
OUTA	93	(92)->IA-Port	- -	1	3	92->Q

OUTB	221	(93)->IB-Port	- -	1	2	93->Q
OUTC	223	(95)->C-PORT	- -	1	2	95->Q
OUTF	95	(94)->FO-Port	- -	1	3	94->Q
POP	91	(R)->A, R+1->R	- -	1	2	
PTC	122	n->H, nm->(R-1, n, nm R-2), R-2->R	- *	4	9	
PUSH	52	(R)->A, R-1->R	- -	1	3	
RC	209	0->C, 1->Z	* *	1	2	
RTN	55	(R-1,R-2)->PC R+2->R	- -	1	4	
SBB	21	(P)-1->(P) (P+1)-B-C->(P+1)	* *	1	5	P+1->P
SBCM	197	(P)-A-C->(P)	* *	1	3	
SBIA n	117,n	A-n->A	* *	2	4	
SBIM n	113,n	(P)-n->(P)	* *	2	4	
SBM	69	(P)-A->(P)	* *	1	3	
SBN	13	(P)-A->(P) (P-I)-C->(P-I)	* *	1	7+3*I	P-I-1->P BCD
SBW	15	(P)-(Q)->(P) (P-I)-(Q-I)-C-> (P-I)	* *	1	7+3*I	P-I-1->P Q-I-2->Q BCD
SC	208	1->C, 1->Z	* *	1	2	
SL	90	C->A7...A0->C	* -	1	2	
SLW	29	(P-I)...(P) 4 Bit SL	- -	1	5+I	P-I-1->P
SR	210	C->A0...A7->C	* -	1	2	
SRW	28	(P)...(P+I) 4 Bit SR	- -	1	5+I	P+I+1->P
STD	82	A->(DP)	- -	1	2	
STP	48	A->P	- -	1	2	
STQ	49	A->Q	- -	1	2	
STR	50	A->R	- -	1	2	
SWP	88	A0...A3<-> A4...A7	- -	1	2	
TEST n	107,n	TEST-Byte AND n	- *	2	4	
TSIA n	102,n	A AND n	- *	2	4	
TSID n	214,n	(DP) AND n	- *	2	6	
TSIM n	98,n	(P) AND n	- *	2	4	
WAIT n	78,n	No Operation	- -	2	6+n	

Anhang B

Mnemonics und Gedankenstützen

Die nachfolgende Tabelle gibt in alphabetischer Reihenfolge einen Überblick über sämtliche Mnemonics der CPU SC 61860 und ihre Bedeutung.

Mnemonic	Bedeutung (englisch)	Bedeutung (deutsch)
ADB	Add binary	Addiere binär
ADCM	Add with Carry to Memory	Addiere mit Übertrag zum int. Speicher
ADIA	Add immediate to A	Addiere unmittelbar zu A
ADIM	Add immediate to Memory	Addiere unmittelbar zum int. Speicher
ADM	Add A to Memory Speicher	Addiere A zum int.
ADN	Add A to indexed block	Addiere A zum indizierten Block
ADW	Add two blocks	Addiere zwei Blöcke
ANIA	And immediate with A	Unmittelbare AND-Ver- knüpfung mit A
ANID	And immediate with Data	Unmittelbare AND-Ver- knüpfung mit ext. Speicher
ANIM	And immediate with Memory	Unmittelbare AND-Ver- knüpfung mit int. Speicher
ANMA	And Memory with A	Und-Verknüpfung des int. Speichers mit A
CAL	Call subroutine of internal ROM	Aufruf eines Unterpro- gramms des int. ROM
CALL	Call subroutine	Unterprogrammaufruf
CPIA	Compare immediate with A	Vergleiche unmittelbar mit A
CPIM	Compare immediate with Memory	Vergleiche unmittelbar mit int. Speicher

CPMA	Compare Memory with A	Vergleiche Int. Speicher mit A
DATA	Read Data from internal ROM	Lies Daten aus dem int. ROM
DECA	Decrement A	Dekrementiere A
DECB	Decrement B	Dekrementiere B
DEC1	Decrement I	Dekrementiere I
DECJ	Decrement J	Dekrementiere J
DECK	Decrement K	Dekrementiere K
DECL	Decrement L	Dekrementiere L
DECP	Decrement P	Dekrementiere P
DECV	Decrement V	Dekrementiere V
DECW	Decrement W	Dekrementiere W
DX	Decrement X	Dekrementiere X
DXL	Decrement X and load	Dekrementiere X und lade
DY	Decrement Y	Dekrementiere Y
DYS	Decrement Y and store	Dekrementiere Y und speichere
ETC	Execute table call	Führe Unterprogramm- aufruf nach Tabelle aus
EXAB	Exchange A with B	Vertausche A mit B
EXAM	Exchange A with Memory	Vertausche A mit int. Speicher
EXB	Exchange blocks, counted with J	Vertausche zwei Blocks, Zähler J
EXBD	Exchange block with Data, counted with J	Vertausche Block mit ext. Speicher, Zähler J
EXW	Exchange blocks, counted with I	Vertausche zwei Blocks, Zähler I
EXWD	Exchange block with Data, counted with I	Vertausche Block mit ext. Speicher, Zähler I
FILD	Fill Data	Fülle ext. Speicher
FILM	Fill Memory	Fülle int. Speicher
INA	Input IA-Port	Lese Port IA
INB	Input IB-Port	Lese Port IB
INCA	Increment A	Inkrementiere A
INC8	Increment B	Inkrementiere B

INCI	Increment I	Inkrementiere I
INCJ	Increment J	Inkrementiere J
INCK	Increment K	Inkrementiere K
INCL	Increment L	Inkrementiere L
INCP	Increment P	Inkrementiere P
IX	Increment X	Inkrementiere X
IXL	Increment X and load	Inkrementiere X und lade
IY	Increment Y	Inkrementiere Y
IYS	Increment Y and store	Inkrementiere Y und speichere
JP	Jump	Springe
JPC	Jump if Carry	Springe, wenn C gesetzt
JPNC	Jump if no Carry	Springe, wenn C nicht gesetzt
JPZ	Jump if Zero	Springe, wenn Z gesetzt
JPNZ	Jump if no Zero	Springe, wenn Z nicht gesetzt
JRCM	Jump relative if Carry minus	Springe rückwärts, wenn C gesetzt
JRCP	Jump relative if Carry plus	Springe vorwärts, wenn C gesetzt
JRM	Jump relative minus	Springe rückwärts
JRNCM	Jump relative if no Carry minus	Springe rückwärts, wenn C nicht gesetzt
JRNCP	Jump relative if no Carry plus	Springe vorwärts, wenn C nicht gesetzt ist
JRNZM	Jump relative if no Zero minus	Springe rückwärts, wenn Z nicht gesetzt
JRNZP	Jump relative if no Zero plus	Springe vorwärts, wenn Z nicht gesetzt ist
JRP	Jump relative plus	Springe vorwärts
JRZM	Jump relative if Zero plus	Springe rückwärts, wenn Z gesetzt
JRZP	Jump relative if Zero plus	Springe vorwärts, wenn Z gesetzt
LEAVE	Leave zero at top	Speichere 0 in die ge- rade bearbeitete Spei- cherzelle des Stacks
LDI	Load A with Data	Lade A mit ext. Speicher

LDM	Load A with Memory	Lade A mit int. Speicher
LDP	Load A with P	Lade A mit P
LDQ	Load A with Q	Lade A mit Q
LDR	Load A with R	Lade A mit R
LIA	Load immediate A	Lade unmittelbar A
LIB	Load immediate B	Lade unmittelbar B
LIDL	Load immediate Data	Lade unmittelbar das
	Pointer low byte	Lowbyte des Datenzeigers
LIDP	Load immediate Data Pointer	Lade unmittelbar den Datenzeiger
LII	Load immediate I	Lade unmittelbar I
LTJ	Load immediate J	Lade unmittelbar J
LIP	Load immediate P	Lade unmittelbar P
LIQ	Load immediate Q	Lade unmittelbar Q
LOOP	Loop	Schleife
LP	Load immediate P	Lade unmittelbar P
MVB	Move block, counted with J	Verschiebe Block, Zähler J
MVBD	Move to block from data, counted with J	Verschiebe in int. Speicher aus ext. Speicher, Zähler J
MVDM	Move to Data from Memory	Verschiebe in ext. Spei- cher aus int. Speicher
MVMD	Move to Memory from Data	Verschiebe in int. Spei- cher aus ext. Speicher
MVW	Move block, counted with I	Verschiebe Block, Zähler I
MVWD	Move to block from Data, counted with I	Verschiebe in int. Speicher aus ext. Speicher, Zähler I
NOPT	No operation for three cycles	Keine Operation für drei Zyklen
NOPW	No operation for two cycles	Keine Operation für zwei Zyklen
ORIA	Or immediate with A	Unmittelbare OR- knüpfung mit A
ORID	Or immediate with Data	Unmittelbare OR- verknüpfung mit ext. Speicher

ORIM	Or immediate with Memory	Unmittelbare OR-Ver- knüpfung mit int. Speicher
ORMA	Or Memory with A	OR-Verknüpfung des int. Speicher mit A
OUTA	Output IA-Port	Schreibe Port IA
OUTB	Output IB-Port	Schreibe Port IB
OUTC	Output Control-Port	Schreibe Control-Port
OUTF	Output F0-Port	Schreibe Port F0
POP	Pop	Nehme vom Stapel
PTC	Prepare table call	Bereite Unterprogramm- aufruf nach Tabelle vor
PUSH	Push	Lege auf Stapel
RC	Reset Carry	Setze Carry-Flag zurück
RTN	Return from subroutine	Kehre aus Unterprogramm zurück
SBB	Subtract binary	Subtrahiere binär
SBCM	Subtract with Carry from Memory	Subtrahiere mit Über- trag vom int. Speicher
SBIA	Subtract immediate from A	Subtrahiere unmittelbar von A
SBIM	Subtract immediate from Memory	Subtrahiere unmittelbar vom int. Speicher
SBM	Subtract A from Memory	Subtrahiere A vom int. Speicher
SBN	Subtract A from indexed block	Subtrahiere A von indiziertem Block
SBW	Subtract two blocks	Subtrahiere zwei Blocks
SC	Set Carry	Setze C-Flag
SL	Shift left	Schiebe nach links
SLW	Shift left block, counted with I	Schiebe Block nach links, Zähler I
SR	Shift right	Schiebe nach rechts
SRW	Shift right block, counted with I	Schiebe Block nach rechts, Zähler I
STD	Store A to Data	Speichere A im ext. Speicher
STP	Store A to P	Speichere A in P
STQ	Store A to Q	Speichere A in Q
STR	Store A to R	Speichere A in R

SWP	Swap	Vertausche
TEST	Test	Prüfe
TSIA	Test immediate A	Prüfe unmittelbar A
TSID	Test immediate Data	Prüfe unmittelbar ext.
		Speicher
TSIM	Test immediate Memory	Prüfe unmittelbar int. Speicher
WAIT	Wait	Warte

Meistens setzt sich der Name eines Mnemonics so zusammen:
 Die ersten Zeichen sind eine Abkürzung des Maschinenbefehls;
 die darauf folgenden Buchstaben zeigen an, welche Register
 davon betroffen sind.

Speicherzellen, Register und Flags

- D Data. Speicherzelle des externen RAM (hängt vom Datenzeiger DP ab)
- M Memory. Speicherzelle des internen RAM (hängt vom Zeiger P ab)
- A Akkumulator
- B B-Register
- I I-Register
- J J-Register
- K K-Register
- L L-Register
- V V-Register
- W W-Register
- X X-Register
- Y Y-Register

- C = Carry-Flag
- Z = Zero-Flag

Eine weitere große Gedächtnishilfe bietet die Aufteilung der Mnemonics in Befehlsgruppen. Diese Gruppen beginnen jeweils mit denselben Buchstaben und haben auch sonst vieles

gemeinsam. Die folgende Übersicht enthält nur die wichtigsten Befehlsgruppen.

LI	Direkte Ladebefehle. Bsp.: LIB 10 lädt das B-Register mit 10.
LD	Ladebefehle für den Akkumulator. Bsp.: LDR lädt den Inhalt des R-Registers in den Akkumulator.
ST	Speicherbefehle für den Akkumulator. Bsp.: STR speichert den Inhalt des Akkus ins R-Register.
OR	OR-Verknüpfungen. Bsp.: ORIA 5 bewirkt eine OR-Verknüpfung zwischen dem Inhalt des Akku und dem Wert 5; das Ergebnis wird im Akku gespeichert.
AN	AND-Verknüpfungen. Bsp.: ANIM 5 bewirkt eine AND-Verknüpfung zwischen (P) und 5. Das Ergebnis wird in (P) gespeichert.
INC	Inkrementbefehle. Bsp.: INCB inkrementiert das B-Register.
DEC	Dekrementbefehle. Bsp.: DECL dekrementiert das L-Register.
AD	Additionsbefehle. Bsp.: ADIA 6 addiert den Wert 6 zum Inhalt des Akkumulators.
SB	Subtraktionsbefehle. Bsp.: SBM subtrahiert den Inhalt des Akkus von (P).
CP	Vergleichsbefehle. CPMA vergleicht den Inhalt des Akkus mit (P).
JR	Relative Sprungbefehle. Bsp.: JRM 5 springt 5 Bytes zurück.
JP	Absolute Sprungbefehle. Bsp.: JP 10000 setzt das Programm bei Adresse 10000 fort.
MV	Verschiebebefehle. Bsp.: MVDM verschiebt (P) nach (DP).
EX	Austauschbefehle. Bsp.: EXAB vertauscht den Inhalt des Akkus mit dem des B-Registers.

Anhang C

Wichtige ROM-Unterprogramme

Auf den folgenden Seiten finden Sie eine Zusammenstellung mit den wichtigsten Unterprogrammen des ROM.

Bitte beachten Sie:

In allen diesen Unterprogrammen wird davon ausgegangen, daß das J-Register den Wert 1 hat. Ein Wert kleiner oder größer als 1 kann zu einem Absturz führen!

Viele Register werden durch das Aufrufen eines solchen Unterprogramms verändert. Sollen diese Registerinhalte später noch verwendet werden, so müssen Sie vor dem Aufruf des Unterprogramms gesichert werden.

Die Flags werden durch jeden Unterprogrammaufruf neu gesetzt.

Tastaturabfrage

INKEY

PC 140X: 3387
PC 126X: 2729

PC 1421: 3359
PC 1350: 3003

Funktion:

Wird gerade eine Taste betätigt, so wird der Code dieser Taste in die Speicherzelle Y+1 des externen RAM gespeichert. Laden Sie also zuerst das Y-Register mit einer günstigen Adresse. Wenn Sie dann das Programm INKEY aufrufen und dabei eine Taste gedrückt ist, so wird der Code dieser Taste mit dem Befehl IYS in der Speicherzelle Y+1 gespeichert.

Geänderte Register:

P,Q,DP,A,B,X,Y,K,L,V

Geänderte Speicherzellen im internen RAM:

10..21,23,34,37..39,55..63,92

GETKEY

PC 140X: 5208
PC 126X: 4414

PC 1421: 5153
PC 1350: 4618

Funktion:

Wartet, bis eine Taste gedrückt wird, dann Rückkehr mit dem Tastencode in A und 24634 (PC 126X: 26156, PC 1350: 28503)

Anmerkungen:

Auch die SHIFT-Belegungen werden berücksichtigt. Wird der OFF-Schalter betätigt, so kehrt die Routine mit A=10 zurück. Automatisches Ausschalten des Rechners, falls keine Taste betätigt wird. Nicht berücksichtigt werden die Tasten X->M, RM, M+, +/-, F<->E beim PC 14XX.

Bei einem Aufrufen des Programms im RUN-Modus wird bei der Tastenkombination 'DEF Buchstabentaste' das Programm mit ERROR 4 abgebrochen.

Geänderte Register:

P,Q,DP,I,A,B,X,Y,K,L,V

Geänderter Speicherzellen im internen RAM:

2..23,92

Ausgabe auf die Anzeige

PRINT

PC 14XX: 32789

PC 1421: 32789

PC 126X: 32781

PC 1350: 53958

Funktion:

PC 14XX: Gibt die Zeichen, die im internen RAM von 16..31 gespeichert sind, auf die Anzeige aus.

PC 126X: Gibt die Zeichen, die im Anzeigepuffer von 26192..26240 gespeichert sind, auf die Anzeige aus.

PC 1350: Gibt die Zeichen, die im Anzeigepuffer von 27904..27999 gespeichert sind, auf die Anzeige aus.

Der Inhalt des A-Registers gibt an, auf welche Zeile diese Zeichen ausgegeben werden. Gezählt wird von unten:

- A = 1: Unterste Zeile
- A = 2: Zweitunterste Zeile
- A = 3: Zweite Zeile
- A = 4: Erste Zeile

Die Position des Cursors innerhalb der Zelle wird mit dem K-Register angegeben (vgl. BASIC: CURSOR K,A).

Geänderte Register:

P,Q,DP,A,B,X,Y,K,L,W

Geänderte Speicherzellen im internen RAM:

2..13 (PC 1350: 2..39)

Ausgabe auf den Drucker CE-126P**LPRINT**

PC 140X: 32846
PC 126X: 32801

PC 1421: 32864
PC 1350: 32852

Funktion:

Gibt die Zeichen, die im internen RAM von 16..39 gespeichert sind, auf den Drucker aus. Das Steuerzeichen 13 (Carriage Return) bewirkt, daß die folgenden Zeichen in eine neue Zeile gedruckt werden. Mit einem einzigen Aufruf lassen sich somit auch mehrere Zeilen drucken. Die auf dem Drucker verfügbaren Zeichen finden sich in Anhang D.

Geänderte Register:

P,Q,DP,A,B,K,L

Geänderte Speicherzellen:

2,3,8,9,92..95

Weitere Unterprogramme**OFF**

PC 140X: 1434
PC 126X: 1112

PC 1421: 1479
PC 1350: 1240

Funktion:

Schaltet den Rechner softwaremäßig aus.

Anmerkung:

Einschalten über ON/OFF-Schalter oder BRK/ON-Taste.

Geänderte Register:

alle

Geänderte Speicherzellen im internen RAM:

alle

BEEP (kurz)

PC 140X: 2754
PC 126X: 2004

PC 1421: 2700
PC 1350: 2379

Funktion:

Schaltet den Summer für 2 ms ein. Die Tonhöhe richtet sich beim PC 14XX und beim PC 126X nach dem ersten Bit (Bit 0) des Akkus: Ist dieses Bit gesetzt, hat der Ton eine Frequenz von 2 kHz, ist dieses Bit nicht gesetzt, so beträgt die Frequenz 4 kHz.

Beim PC 1350 wird die Tonfrequenz durch das Carry-Flag bestimmt:

C=1: 4 kHz
C=0: 2 kHz

Anmerkung:

Ist der Summer erst einmal eingeschaltet, bleibt er während des ganzen weiteren Programmverlaufs eingeschaltet. Um das zu verhindern, muß der Summer durch die Befehlsfolge

LIP 95
ANIM 239
OUTC

wieder ausgeschaltet werden. Durch eine Zeitverzögerung vor dem Ausschalten kann jede beliebige Tondauer erzeugt werden.

Geänderte Register:

P,Q,A

Geänderte Speicherzellen im internen RAM:

2,95

BEEP (*lang*)

PC 140X: 49321

PC 1421: 55539

PC 126X: 48908

PC 1350: 49430

Funktion:

Erzeugt einen BEEP von der Frequenz 4KHz (wie BASIC-BEEP). Dieser BEEP muß im Gegensatz zum kurzen BEEP nicht abgestellt werden. Deshalb ist es auch empfehlenswerter, dieses Unterprogramm für einen BEEP zu benutzen. Allerdings sollte der BEEP-Befehl nicht in Schläufen aufgerufen werden, da sonst Störungen auftreten können und die BEEP-Routine nicht mehr ausgeführt wird.

Anmerkung:

Dieser Befehl ist mit Bedacht einzusetzen, da er unter Umständen zu einem Absturz führen kann. Dies vor allem, wenn die internen Speicherzellen 20 bis 90 verändert wurden.

Geänderte Register:

Nicht bekannt.

Anhang D

Druckercodes des CE-126P (14XX und 126X)

Wie der PC hat auch der Drucker ein ROM. In diesem ROM ist ein eigener Zeichensatz des Druckers untergebracht. Dieser Druckerzeichensatz hat viele Sonderzeichen, die man für die Gestaltung von Listings gut gebrauchen kann.

Diese Zeichen des Drucker-ROMs gelten nur für den PC 14XX und den PC 126X. Der PC 1350 kann diese Druckerzeichen leider nicht ausnutzen.

Beim PC 14XX und PC 126X können Sie diese Zeichen mit dem ROM-Unterprogramm LPRINT, das wir im Anhang C vorgestellt haben, auf dem Drucker erzeugen.

Beim PC 126X können die Zeichen mit einem Code größer 10 auch in eine Textvariable gePOKEd und dann mit LPRINT <Textvariable> ausgedruckt werden. Dabei muß bei Codes von 123 bis 255 eine 254 vor dem eigentlichen Zeichencode geschrieben werden, da sonst der Code als BASIC-Befehl gedruckt wird. Die meisten Codes (auch japanische Zeichen) können beim PC 126X auch mit dem Maschinenprogramm PRINT auf der Anzeige in diesem Format dargestellt werden.

Anhang E

Umrechnungstabelle HEX-DEZ

Mit der folgenden Tabelle können hexadezimale Werte in dezimale umgerechnet werden und umgekehrt. Sie ist besonders dann von Nutzen, wenn der Rechner gerade besetzt ist (z.B. bei der Benutzung des ASSEMBLERs) oder wenn der Rechner gewisse Umrechnungsfunktionen gar nicht besitzt, wie z.B. der PC 126X, der den HEX-Befehl gar nicht kennt.

UMRECHNUNGSTABELLE HEX-DEZ

2. Ziffer

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

1. Ziffer

Anhang F

Literaturverzeichnis

"Systemhandbuch für den SHARP PC-1401/-1402 Taschencomputer", Fischel Verlag, 1984

"Maschinensprache für den SHARP PC-1401/-1402", Bernd Saretz, Becker & Partner Verlag+Datentechnik

"PC Learn", Rasso von Schlichtegroll, Holtkötter

SHARP

Brechmann

Pocket Computer BASIC- Programme

für
alle SHARP
Pocket-
Computer

Ein DATA BECKER Buch

Die SHARP-PC-Taschencomputer erfreuen sich breiter Beliebtheit. Untereinander ist das BASIC der SHARP-PC-Rechner aber nicht kompatibel. Deshalb wurde für dieses Buch der BASIC-Standard entwickelt. Neben den Erläuterungen der einzelnen Befehle und deren Besonderheiten enthält es eine komplette Programmsammlung für alle SHARP-PCs. Dieses Buch ist ein Muß für jeden SHARP-PC-Benutzer.

Aus dem Inhalt:

- Telefon-Takt
- Lottozahlen
- Zinseszins
- Temperaturen
- Space Shuttle
- Das Standard-BASIC
- Programme zum Spielen
- und vieles mehr

Brechmann - SHARP-PC BASIC-Programme
ca. 250 Seiten, DM 29,-
Erscheint ca. Juli
ISBN 3-89011-173-4

Bartel · Kraas
Schrüter

Das große

Computer schach buch



EIN DATA BECKER BUCH

Wie funktioniert eigentlich ein Schachprogramm? Die wenigsten wissen, mit welchen Algorithmen man dem Computer das königliche Spiel beibringen kann. Dieses Buch informiert über alle Aspekte der Schachprogrammierung und enthält ein komplettes Schachprogramm in BASIC, das auf anschauliche Weise die Probleme der Programmierung und ihre Lösungen darstellt. Abgerundet wird das Buch durch eine kleine Geschichte des Computerschach und eine Menge Tips zum Thema: Wie spielt man Schach gegen den Computer?

Aus dem Inhalt:

- Programme, Partien und Personen
- Strategiespiele auf dem Computer
- Stack und Rekursion in BASIC
- Brettdarstellung und Zuggenerierung
- Suchalgorithmen in Schachprogrammen
- Bewertungsfunktionen
- Komplettes Schachprogramm in BASIC
- Ausführliche Dokumentation zum Programm
- Testverfahren für Schachprogramme
- 7 goldene Regeln zum Spiel gegen Computer

Kraas/Schrüter/Bartel
Das große Computerschachbuch
458 Seiten, DM 49,-
ISBN 3-89011-117-3

MONADJEMI

**DAS
TRAININGSBUCH
ZU**

FORTH



EIN DATA BECKER BUCH

FORTH ist eine faszinierende Sprache für vielerlei Anwendungen, ob Sie nun Roboter steuern oder schnell hochauflösende Grafiken erstellen wollen. Das Trainingsbuch zu FORTH gibt nicht nur eine leichtverståndliche Einführung, sondern bietet auch viele tiefergehenden Informationen und wichtige Hinweise über den inter-nen Aufbau dieser Programmiersprache.

Aus dem Inhalt:

- Einstieg in eine neue Programmierphilosophie
- Rechnen mit UPN
- Arbeiten mit dem Stack
- Konstanten und Variablen
- Der Editor
- Strukturiertes Programmieren
- Ein-/Ausgabe in FORTH
- FORTH INTERN
- Speicheraufteilung
- Wörterbucheintrag
- Compiler oder Interpreter?
- Erweitern des Compilers
- Stringverarbeitung
- FORTH und die Maschinensprache
- Fehlerbehandlung in FORTH-Systemen
- ... und vieles mehr

Monadjemi · Das Trainingsbuch zu FORTH
300 Seiten, DM 39,—
ISBN 3-89011-055-X