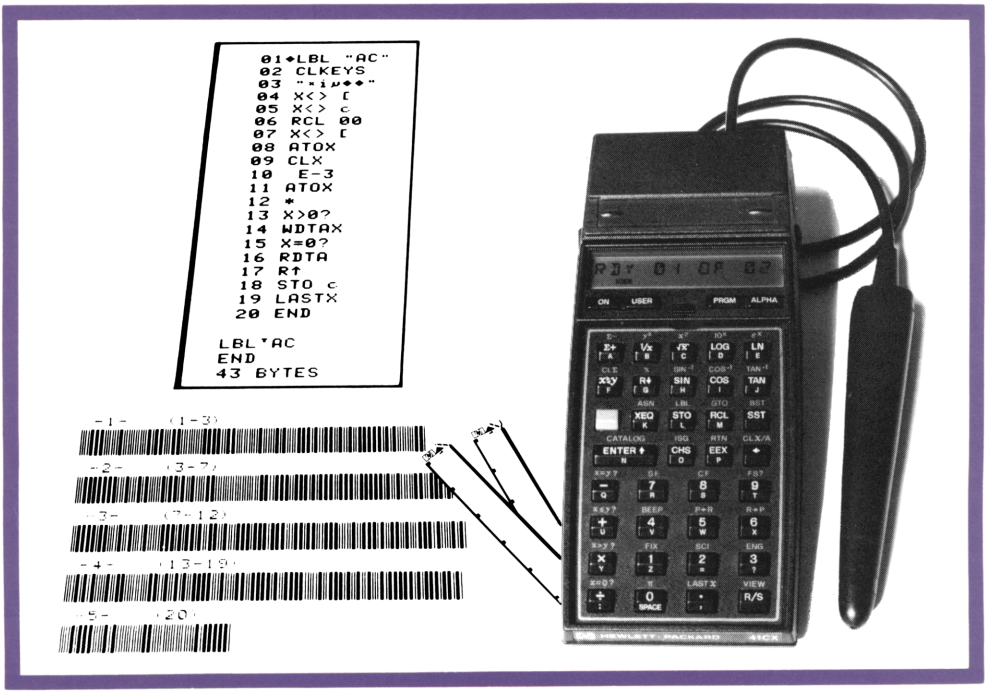
Gerhard Kruse

Optimales Programmieren mit dem HP-41

Vieweg



Gerhard Kruse

Optimales Programmieren mit dem HP-41

Mikrocomputer

Einführung in BASIC
von W. Schneider

Lehr- und Übungsbuch für die Rechnerserien cbm 2001 und cbm 3001
von G. Oetzmann

BASIC für Fortgeschrittene
von W. Schneider

Einführung in Pascal
von W. Schneider

Lehr- und Übungsbuch für die Rechnerserien cbm 4001 und cbm 8001
von G. Oetzmann

BASIC-Programmierbuch zu den grundlegenden Ablaufstrukturen der Datenverarbeitung
von E. Kaier

Lehr- und Übungsbuch für Commodore-Volkscomputer
von G. Oetzmann

Assembler-Programmierung von Mikroprozessoren (8080, 8085, Z 80) mit dem ZX 81
von P. Kahlig

Einführung in die Anwendung des Betriebssystems CP/M
von W. Schneider

Datenstrukturen in Pascal und BASIC
von D. Herrmann

Programmprinzipien in BASIC und Pascal
von D. Herrmann

Softwareentwicklung am Beispiel einer Dateiverwaltung (HP-41)
von M. Gehret

Optimales Programmieren mit dem HP-41
von G. Kruse

Lineares Optimieren – 11 HP-41-Programme
herausgegeben von Harald Schumny

Dienstprogramme (Tool Kit) für den HP-41
herausgegeben von Harald Schumny

Vieweg

Gerhard Kruse

Optimales Programmieren mit dem HP-41

Mit 51 vollständigen Programmen
und Routinen

Friedr. Vieweg & Sohn Braunschweig/Wiesbaden

Das in diesem Buch enthaltene Programm-Material ist mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Der Autor übernimmt infolgedessen keine Verantwortung und wird keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programm-Materials oder Teilen davon entsteht.

1985

Alle Rechte vorbehalten
© Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig 1985

Die Vervielfältigung und Übertragung einzelner Textabschnitte, Zeichnungen oder Bilder, auch für Zwecke der Unterrichtsgestaltung, gestattet das Urheberrecht nur, wenn sie mit dem Verlag vorher vereinbart wurden. Im Einzelfall muß über die Zahlung einer Gebühr für die Nutzung fremden geistigen Eigentums entschieden werden. Das gilt für die Vervielfältigung durch alle Verfahren einschließlich Speicherung und jede Übertragung auf Papier, Transparente, Filme, Bänder, Platten und andere Medien. Dieser Vermerk umfaßt nicht die in den §§ 53 und 54 URG ausdrücklich erwähnten Ausnahmen.

Druck und buchbinderische Verarbeitung: Lengericher Handelsdruckerei, Lengerich
Printed in Germany

ISBN 3-528-04335-0

VORWORT

Die Anwendungsmöglichkeiten des Taschencomputers HP-41 sind in den letzten Jahren entscheidend erweitert worden. Dadurch ist ein komplexes Taschencomputersystem entstanden, das im Laufe der Zeit viele Anhänger gefunden hat.

Die Vorzüge des HP-41 gegenüber Personal-Computern liegen auch heute noch in seiner Kompaktheit und in seinem Permanentspeicher. Er ist wirklich überall sofort einsetzbar. Hinzu kommt die vielseitige Möglichkeit individueller Systemerweiterungen.

Dieses Buch stellt Methoden der fortschrittlichen Programmierung des HP-41 vor. Es werden 51 vollständige Programme und Routinen in Funktion, Aufbau und Anwendung dargestellt. Anhand allgemein verständlicher Programme werden die verschiedenen Techniken zur Programmoptimierung aufgezeigt.

Besonders die intensive Nutzung des X-Functions Moduls und der synthetischen Programmierung werden an vielen Programmbeispielen ausführlich erläutert.

Die Grundlagen der synthetischen Programmierung, "Load Bytes" und "Key Assignment" werden als besonders kurze Programme vorgestellt. Anwendungen und sinnvoller Nutzen synthetischer Programmierung werden an einfachen und komplexen Programmbeispielen schrittweise verständlich gemacht.

Mit den Programmbeschreibungen bietet das Buch dem Benutzer des HP-41 eine Fülle wertvoller Anregungen, um die Möglichkeiten seines Taschencomputers optimal zu nutzen.

Aachen, im August 1984 Gerhard Kruse

INHALTSVERZEICHNIS

1	STACKREGISTER ODER DATENREGISTER ?		1
1.1	"SQRT"	Babylonisches Wurzelziehen	1
1.2	"PYT"	Pythagoräische Zahlentripel	3
1.3	"UZ"	Ulams Zahlenreihen	5
1.4	"Y HOCH X"	Erweiterte Exponentialfunktion	7
2	BASISUMRECHNUNGEN - KOMPAKTE DATENSPEICHERUNG		10
2.1	"DH"	Dezimal - Hex - Wandler	10
2.2	"HD"	HEX - Dezimal - Wandler	11
2.3	"DB"	Dezimal - Binär - Wandler	12
2.4	"BD"	Binär - Dezimal - Wandler	12
2.5	"XB"	Dezimal - Binär - Wandler	13
2.6	"XA"/"AX"	Kompakte Datenspeicherung	13
3	ZUFALLSZAHLEN - LOTTOZAHLEN		15
3.1	"STAT"	Statistische Auswertung	15
3.2	"DTA"	Kompakte Datenspeicherung	16
3.3	"DFL"	Kompakte Datenspeicherung	17
4	EINFACHES CODIEREN VON TEXT		18
4.1	"CA"	Codiere Alpha	18
4.2	"DA"	Decodiere Alpha	18
4.3	"CODE"	Codiere/Decodiere Alpha	19
4.4	"C-A"	Codiere Alpha	19
4.5	"D-A"	Decodiere Alpha	19

5	EINSTIEG IN DIE SYNTHETISCHE PROGRAMMIERUNG		21
	5.1 "LB"	Load Bytes I	23
	5.2 "LB"	Load Bytes II	27
	5.3 "--"	Laufschrift, normal	30
	5.3 "--"	Laufschrift, synthetisch	30
6	BARCODES		32
	6.1 "BA"	Barcode Analyser	32
7	PROGRAMMZEIGERMANIPULATIONEN		35
	7.1 "DP"	Dezimaladresse - Programmzeiger	35
	7.2 "PD"	Programmzeiger - Dezimaladresse	35
	7.3 "XF"	Programm im X-Functions Modul ausführen	38
8	TIME MODUL		41
	8.1 "TD"	Tondauer	41
	8.2 "BAT"	Batterienutzungsdauer	42
	8.3 "O"	Ostertermin	43
	8.4 "A?"	Assignment- und Alarmregister Finder	44
	8.5 "AF"	Alarme - Datenfile	49
	8.6 "AC"	Alarme - Magnetkarte	53
	8.7 "A-"	Alarme unterdrücken	57

9	SORTIEREN VON DATENREGISTERN		58
	9.1 "SØ"	Sortierprogramm	58
	9.2 "SO"	Sortierprogramm	58
	9.3 "SX"	Sortierprogramm	59
	9.4 "SD"	Sortierdauer	60
10	DAS HP - IL DEVELOPMENT MODUL		61
	10.1 "PB"	Print Bytes	62
	10.2 "E?"	.END. Finder	64
	10.3 "C?"	Curtain Finder	64
	10.4 "KA"	Key Assignment I	65
	10.5 "KA"	Key Assignment II	67
	10.6 "NEWM?"	Maximale Einträge je Cassette	69
11	FINANZPROGRAMM		70
	11.1 "F"	Universelles Finanzprogramm	70

1 STACKREGISTER ODER DATENREGISTER ?

Die optimale Nutzung des Speicherplatzes durch den intensiven Gebrauch der Stackregister soll in diesem Kapitel an einigen Beispielen deutlich gemacht werden.

Es ist eine wesentliche Aufgabe des Programmierers, den im Rechner vorhandenen Speicherplatz möglichst optimal auszunutzen. Wie weit man den Rechner auch durch zusätzliche Speichermodule erweitert, der vorhandene Speicherplatz bleibt doch immer begrenzt. Besonders in der neuesten Version des HP-41, im HP-41 CX, ist der Speicherplatz, gemessen an der Rechnergröße, recht bemerkenswert. Trotzdem sollte man nicht verschwenderisch damit umgehen.

1.1 "SQRT" - Babylonisches Wurzelziehen

Nach der Methode des "babylonischen Wurzelziehens" ist es möglich, nur mit Hilfe der Grundrechenarten Division und Addition die Quadratwurzel aus einer positiven Zahl zu ziehen. Dabei wird die folgende Formel verwandt:

$$W = \frac{1}{2} \times \left(W' + \frac{A}{W'} \right)$$

A ist die Zahl, aus der die Wurzel gezogen werden soll. W' der erste Näherungswert für die Wurzel und W der zweite Näherungswert, der sich aus dem ersten ergibt. Diese Rechenformel wird immer wieder ausgeführt, bis W der Quadratwurzel von A soweit angenähert ist, daß die Differenz vernachlässigt werden kann.

Für diese Wurzelberechnung durch Iteration ist der HP-41 bestens geeignet: Das Programm "SQRT" benötigt zwei Datenregister und belegt 29 Bytes. Da jedes Datenregister 7 Bytes belegt, ist der gesamte Platzbedarf des Programms 43 Bytes.

```
01◆LBL "SQR
       T"
02 X<0?
03 CLX
04 STO 00
05 STO 01

06◆LBL 00
07 RCL 00
08 RCL 01
09 /
10 RCL 01
11 +
12 2
13 /
14 ENTER↑
15 X<> 01
16 X≠Y?
17 GTO 00
18 END

LBL'SQRT
END
29 BYTES
```

Das Programm speichert A in Register 00 und W' in Register 01. Als erster Näherungswert wird A angenommen (Zeile 05). Die Annahme des ersten Näherungswertes ist ein entscheidender Punkt bei Iterationsaufgaben. Die erste Annahme kann die Rechenzeit sehr entscheidend beeinflussen oder die Berechnung sogar unmöglich machen. Für das hier behandelte Beispiel ist das jedoch nicht von Bedeutung. In den Zeilen 07 bis 13 wird die eigentliche Formelberechnung durchgeführt. Zeile 15 tauscht dann W gegen W' aus. In Zeile 16 wird verglichen, ob W' (im X-Register) ungleich W (im Y-Register) ist. Ist das der Fall, wird die Formel, beginnend mit Zeile 07, nochmals berechnet. Ist W nicht mehr ungleich W', wird die Berechnung nicht weiter fortgesetzt.

Nach dieser Iterationsmethode könnten W und W' eigentlich niemals wirklich gleich werden. Daß dies dennoch der Fall ist (das Programm läuft ja nicht endlos weiter!), liegt allein daran, daß der Rechner nur mit einer endlichen Stellenzahl rechnet; sobald diese Stellen identisch sind, wird W "gleich" W'. Die Zeilen 02 und 03 sind erforderlich, da die Formel nur für positive A gültig ist; falls ein negativer Wert eingegeben wurde, stoppt das Programm in Zeile 09 mit 'DATA ERROR'. Nun kann man die Funktion des Programms mit beliebigen positiven Werten testen. Natürlich ist das Programm von der Aufgabenstellung her betrachtet nicht erforderlich, da der Rechner eine Quadratwurzelfunktion besitzt. Aber gerade das Programmbeispiel kann sehr viel anschaulich machen. Das gilt besonders bei Vergleich mit der nächsten Programmversion:

```
01◆LBL "SQR
       T"
02 X<0?
03 CLX
04 ENTER↑
05 R↑

06◆LBL 00
07 RDN
08 RCL Y
09 X<>Y
10 /
11 LASTX
12 +
13 2
14 ST/ Y
15 X<> L
16 X≠Y?
17 GTO 00
18 END

LBL'SQRT
END
31 BYTES
```

Das Programm "SQRT" belegt jetzt 31 Bytes, aber es werden keine Datenregister mehr benutzt. Also sind etwa 28 % des ursprünglichen Speicherplatzes eingespart worden.

Betrachtet man zunächst nur die Formelschleife, Zeile 07 bis 14, und läßt dabei die Zeilen 08 und 09 unbeachtet, gilt: Vor der Ausführung der Division ist A im Y-Register und W' im X-Register (Zeile 10). W' bleibt nach der Division und auch nach der Addition (Zeile 10 und Zeile 12) im LastX-Register. Aber warum nicht die Division durch 2 mit 2, / ? Hier wird die Stack-Register-Rechnung 2, ST/ Y verwandt, damit W' für den Vergleich (Zeile 16) und für die späteren Rechnungen im LastX-Register erhalten bleibt. Die Zeilen 08 und 09 haben die Aufgabe, den Wert A zu erhalten, der sonst bei der Division verloren ginge.

Die zweite Programmversion macht Stackbewegungen und Stackrechnungen anschaulich. Man kann das Programm mit SST in Einzelschritten ausführen, um das zu beobachten. Will man die einzelnen Iterationsschritte sichtbar machen, ist vor dem Vergleich (Zeile 16) VIEW X oder PSE einzufügen. Dann ist zu erkennen, daß der erste Näherungswert A ist.

1.2 "PYT" - Pythagoräische Zahlentripel

Das hier aufgelistete Programm "PYT" berechnet sogenannte pythagoräische Zahlentripel. Das sind ganzzahlige Seitenlängen eines rechtwinkligen Dreiecks. Es werden zwei Programmversionen gegenübergestellt: 65 Bytes, SIZE 002 und 72 Bytes, SIZE 000. Für beide Programme sollte FIX 0 eingestellt werden. Dann ist eine positive ganze Zahl (größer als 1) einzugeben, und das Programm kann gestartet werden.

Es läuft solange, bis es mit R/S angehalten wird. Für eine längere Programmlaufzeit ist es ratsam, die Zeilen FS? 49, OFF einzufügen, damit der Rechner bei Erschöpfung der Batterien nicht seine Speicherinhalte verliert. Diese Einfügung sollte sinnvollerweise im dritten Programmabschnitt (LBL 02) erfolgen, damit die Programmlaufzeit nicht übermäßig verlängert wird.

Die einzugebende Startzahl ist eine Kathete des rechtwinkligen Dreiecks. Im ersten Programmabschnitt (LBL 00) wird die maximale Größe der zweiten Kathete berechnet. Im zweiten Programmabschnitt (LBL 01) wird getestet, ob die Hypothenuse ganzzahlig ist. Dazu wird die erste Kathete mit der Größe der eingegebenen Startzahl und die zweite Kathete um 1 größer als die Startzahl angenommen. Die zweite Kathete wird anschließend bis zum vorher errechneten Maximalwert jeweils um 1 erhöht. Die ISG-Befehle dienen dazu, die Kathetengröße jeweils um 1 zu erhöhen (Zeilen 17 und 29). Die anschließende Zeile (CLD) wird immer übersprungen. Dort kann also in diesem Falle jeder beliebige Befehl stehen.

```
01◆LBL "PYT
                "
02 STO 00

03◆LBL 00
04 RCL 00
05 STO 01
06 X↑2
07 DSE X
08 2
09 /
10 FRC
11 LASTX
12 *
13 X=0?
14 LASTX
15 INT

16◆LBL 01
17 ISG 01
18 CLD
19 RCL 00
20 RCL 01
21 R-P
22 FRC
23 X=0?
24 XEQ 02
25 RCL 01
26 R↑
27 X>Y?
28 GTO 01
29 ISG 00
30 CLD
31 GTO 00

32◆LBL 02
33 " "
34 ARCL 00
35 "⊢/"
36 ARCL 01
37 "⊢/"
38 ARCL L
39 TONE 8
40 AVIEW
41 END

LBL'PYT
END
65 BYTES
```

```
01◆LBL "PYT
       "
02◆LBL 00
03 ENTER↑
04 ENTER↑
05 X↑2
06 DSE X
07 2
08 /
09 FRC
10 LASTX
11 *
12 X=0?
13 X<> L
14 INT

15◆LBL 01
16 RDN
17 ISG X
18 CLD
19 R-P
20 X<>Y
21 X<> L
22 X<>Y
23 FRC
24 X=0?
25 XEQ 02
26 CLX
27 RCL Z
28 X<> T
29 X>Y?
30 GTO 01
31 R↑
32 ISG X
33 CLD
34 GTO 00

35◆LBL 02
36 " "
37 ARCL Z
38 "├/"
39 ARCL Y
40 "├/"
41 ARCL L
42 TONE 8
43 AVIEW
44 END

LBL'PYT
END
72 BYTES
```

Um sicherzustellen, daß der Programmbefehl nach einem ISG-Befehl immer ausgeführt wird, fügt man zwischen den ISG-Befehl und den als nächsten vorgesehenen Programmbefehl einen sogenannten NOP ein. Dafür muß ein Befehl gewählt werden, der auch dann keine störende Wirkung hat, wenn er ausgeführt wird.

Ist der maximale Wert der zweiten Kathete erreicht, wird die Startzahl um 1 erhöht, und das Programm beginnt von vorne (Zeilen 29, 30, 31).

Der dritte Programmabschnitt (LBL 02) stellt die Ergebnisausgabe dar. Die Zeile nach dem Label ist ein Leerschritt im Alpharegister (SPACE) und könnte auch durch CLA ersetzt werden. Da dieses Programm je nach eingegebener Startzahl eventuell recht lange bis zu einer Ergebnisanzeige läuft, ist der Ton hier angebracht. (Beispiel: 89/3960/3961!)

Da bei den beiden Programmen "PYT" der gesamte benötigte Speicherplatz fast gleich ist (65 Bytes + 14 Bytes = 79 Bytes bzw. 72 Bytes), kommt hier noch ein weiterer Faktor hinzu, um eine Programmoptimierung zu beurteilen: Die Laufzeit!

Da die Schleife zu Beginn des LBL 01 sehr oft durchlaufen werden muß, ist das Programm, das Datenregister benutzt, in diesem Fall deutlich optimaler, da es wesentlich schneller ist. Bei der Frage, ob man die Benutzung von Datenregistern oder Stackregistern bevorzugt, sollte man also besonders in Programmschleifen den Geschwindigkeitsaspekt mit berücksichtigen.

1.3 "UZ" - Ulams Zahlenreihen

Eine Freude für alle Freunde der Zahlenspielerei ist das folgende Programm "UZ". Es beruht auf einer These von Stanislaw Ulam, die besagt, daß man von jeder positiven ganzen Zahl aus immer 1 erreicht, wenn man nach der folgenden Rechenregel vorgeht:

- Ist die Zahl gerade, ist sie durch 2 zu teilen,
- ist die Zahl ungerade, ist sie mit 3 zu multiplizieren und anschließend 1 zu addieren.

Mit dem Ergebnis ist dann nach der gleichen Rechenregel fortzufahren, bis man schließlich 1 erhält. Das Erstaunliche ist, daß man tatsächlich immer 1 erhält, wenn auch oft auf großen Umwegen.

Das Programm "UZ" belegt 69 Bytes und keine Datenregister. Die ersten Zeilen (Zeile 02 bis 09) dienen dazu, jede falsche Eingabe durch 'DATA ERROR' abzuweisen. Es dürfen nur ganze Zahlen größer als 1 eingegeben werden. Diese Anforderung wird in vielen Programmen gestellt; z.B. bei Primzahlprogrammen oder auch bei dem vorigen Programmbeispiel "PYT". Die meisten dieser Programme würden bei fehlerhaften Eingaben in unendliche Programmschleifen laufen oder falsche Ergebnisse liefern. Deshalb ist eine Fehlerprüfung oft sinnvoll. Will man die Fehlerprüfung aus dem Programm herausnehmen, löscht man die Zeilen 02 bis 09 (einschließlich); statt dessen muß dann zweimal ENTER eingesetzt werden.

Startet man das Programm "UZ" mit einer gültigen Eingabe, werden die einzelnen Rechenergebnisse der Reihe nach angezeigt (VIEW X, Zeile 16), bis 1 erreicht ist. Man kann auch den Drucker anschließen, um die Einzelergebnisse festzuhalten. Zum Schluß werden die Ausgangszahl und die Anzahl der Rechenzyklen angezeigt, die erforderlich waren, um 1 zu erreichen.

```
01◆LBL "UZ"
02 INT
03 LASTX
04 X≠Y?
05 ASIN
06 X<>Y
07 2
08 X>Y?
09 ASIN
10 CLX
11 X<>Y
12 FIX 0

13◆LBL 00
14 X=0?
15 X<> L
16 VIEW X
17 ISG Y
18 CLD
19 2
20 X=Y?
21 GTO 01
22 /
23 FRC
24 X=0?
25 GTO 00
26 X<> L
27 6
28 *
29 1
30 +
31 GTO 00

32◆LBL 01
33 SIGN
34 VIEW X
35 " "
36 ARCL T
37 "⊢ ... "
38 ARCL Z
39 R↑
40 +
41 TONE 8
42 AVIEW
43 END

LBL'UZ
END
69 BYTES
```

Beispiel 13: 13, 40, 20, 10, 5, 16, 8, 4, 2, 1 13 . . . 9

Das Programm ist so aufgebaut, daß man nach der letzten Anzeige mit R/S sofort den nächsten Durchgang für die nächste Zahl starten kann (im Beispielfall also 14).

Schon bei niedrigen Zahlen gibt es recht lange Reihen: 27, 31, 41. Will man nicht jedes einzelne Zwischenergebnis der Reihe, sondern nur die letzte Anzeige sehen, löscht man Zeile 34 und Zeile 16 (VIEW X). Zeile 35 ist wieder ein Leerschritt im Alpharegister, der durch CLA ersetzt werden kann. Setzt man vor das END (Zeile 43) GTO "UZ", so läuft das Programm fortlaufend weiter. Man kann so mit dem Drucker für viele Zahlen feststellen, wie viele Rechenzyklen benötigt werden, um die Zahl 1 zu erreichen.

Erstaunlich dabei ist, wie oft es vorkommt, daß drei aufeinanderfolgende Zahlen genau die gleiche Anzahl von Rechenzyklen hervorbringen!

Beispiele:	124,125,126;
	145,146,147;
	156,157,158;
	164,165,166
oder auch kleinere:	28, 29, 30;
	36, 37, 38
und auch sehr große:	1068,1069,1070;
	1028,1029,1030.

An diesem Programm zeigt sich, daß Zahlenspielereien oft Überraschungen hervorbringen können.

Wenn man die Anzahl der Rechenzyklen fortlaufend ermitteln möchte, sollte man natürlich auch die Zeilen 34 und 16 löschen; außerdem ist es schneller und kürzer, wenn man statt am Ende GTO "UZ" einzugeben, dort GTO 02 eingibt und an den Programmanfang zusätzlich LBL 02 setzt.

Das Programm "UZ" enthält einige Passagen, an denen eine weitere Technik der Programmoptimierung verdeutlicht werden kann: Die Mehrfachbenutzung einzelner Programmbefehle.

Fast alle Programmbefehle des Rechners haben mehrere Auswirkungen, auch wenn man sich dessen nicht immer bewußt ist. Dies gilt besonders für die Stackregister. Gelingt es, in einem Programm mehrere Wirkungen eines Programmbefehls zu nutzen, wird das Programm dadurch wesentlich kürzer und schneller.

Mit dem Befehl FRC in Zeile 23 kann man wegen der vorher erfolgten Division durch 2 zunächst einmal erkennen, ob eine gerade oder ungerade Zahl vorlag. Dies kann in Zeile 24 mit X = 0 ? erkannt werden. Als Nebeneffekt speichert aber FRC die ursprüngliche Zahl in das LastX-Register, so daß später, entweder in Zeile 26 oder in Zeile 15 mit dieser Zahl weitergerechnet werden kann. Hier werden also die Wirkungen des Befehls FRC doppelt genutzt. Der Befehl STO L sollte eigentlich in keinem Programm vorkommen. Er kann wie hier in den meisten Fällen durch einen anderen Befehl (INT, FRC, SIGN oder einen Rechenbefehl) ersetzt werden; dies gilt selbst dann, wenn die eigentliche Funktion des verwendeten Befehls gar nicht benötigt wird, dann spart man immerhin noch ein Byte.

Die in Zeile 19 verwendete Zahl 2 wird sogar in dreifacher Hinsicht genutzt: Zunächst für die Entscheidung, ob die Zahlenreihe beendet werden kann, da die Zahl 2 erreicht worden ist und bei der nächsten Division durch 2 das Ergebnis 1 erreicht würde (X = Y ?, GTO 01); dann um zu erkennen, ob es sich um eine gerade oder ungerade Zahl handelt und schließlich noch für die weitere Berechnung. Eine gerade Zahl ist bereits durch 2 geteilt worden, eine ungerade Zahl müßte mit 3 multipliziert werden, wird dann aber wegen der vorherigen Division durch 2 mit 6 multipliziert.

Eine weitere Möglichkeit, die Nebenwirkungen eines Befehls zu nutzen, kann man in Zeile 05 und Zeile 09 (ASIN) sehen. Die Hauptwirkung des Befehls, die Berechnung des Arcussinus, ist in diesem Fall nicht beabsichtigt. Vielmehr soll hier die Fehlermeldung 'DATA ERROR' erzeugt werden, um falsche Eingaben abzuweisen. Bei der Vielzahl der im Rechner enthaltenen Fehlermeldungen (besonders in Verbindung mit dem X-Functions Modul) kann mit Hilfe dieser Technik fast für jede Situation ein 'passender' Fehlerhinweis programmiert werden. So erhält man wesentlich schnellere und kürzere Programme als mit einer selbst formulierten Fehlermeldung und dem anschließenden PROMPT-Befehl. Man sollte dabei aber nicht vergessen, daß die Verwendung eines Befehls ausschließlich zur Erzeugung eines Fehlerhinweises die Programmlesbarkeit eventuell erheblich erschwert.

Damit ein Programm gut lesbar bleibt, sollte man für die zu überspringende Programmzeile im Anschluß an ISG oder DSE immer den gleichen Programmbefehl verwenden; CLD ruft kaum Mißverständnisse hervor.

1.4 "Y HOCH X" - Erweiterte Exponentialfunktion

Das Programm "Y HOCH X" benutzt keine Datenregister. Es erlaubt (wie die Rechnerfunktion Y hoch X) die Exponentialfunktion für sehr große oder sehr kleine Werte zu berechnen, ohne daß der Rechenbereich überschritten oder unterschritten wird ('OUT OF RANGE'). Die eingegebene Basiszahl muß jedoch positiv sein.

```
01◆LBL "Y↑X
       "
02 X<>Y
03 LOG
04 *
05 INT
06 LASTX
07 FRC
08 10↑X
09 " "
10 FIX 5
11 ARCL X
12 "⊢ "
13 X<>Y
14 X<0?
15 "⊢-"
16 ABS
17 FIX 0
18 100
19 X>Y?
20 "⊢0"
21 RDN
22 10
23 X>Y?
24 "⊢0"
25 RDN
26 ARCL X
27 RDN
28 LASTX
29 X<>Y
30 FIX 9
31 AVIEW
32 END

LBL"Y↑X
END
57 BYTES
```

```
01◆LBL "Y↑X
       "
02 X<>Y
03 LOG
04 *
05 INT
06 LASTX
07 FRC
08 10↑X
09 " "
10 RCLFLAG
11 FIX 5
12 ARCL Y
13 "⊢ "
14 X<> Z
15 X<0?
16 "⊢-"
17 ABS
18 FIX 0
19 100
20 X>Y?
21 "⊢0"
22 RDN
23 10
24 X>Y?
25 "⊢0"
26 RDN
27 ARCL X
28 RDN
29 LASTX
30 X<>Y
31 RCL Z
32 STOFLAG
33 RDN
34 AVIEW
35 END

LBL"Y↑X
END
63 BYTES
```

Ordnet man im USER-Modus den Programmstart der Funktionstaste Y hoch X zu, bleibt die Programmanwendung wie die Benutzung der Rechnerfunktion. Neben der Anzeige als Mantisse und Exponent liefert das Programm im X-Register die Mantisse und im Y-Register und LastX-Register den Exponenten des Ergebnisses.

Bei diesem Programm stellt sich die Aufgabe, den Displaymodus während des Programms zu verändern. Das Ergebnis soll in der Mantisse mit fünf Nachkommastellen angezeigt werden, der Exponent wird ohne Kommastellen mindestens dreistellig angezeigt. Auf diese Weise kann das gesamte Anzeigefeld des Rechners sinnvoll genutzt werden.

Hier werden zwei Programmversionen gezeigt, da (ohne das Ausnutzen der synthetischen Programmierung, auf die später eingegangen wird) nur mit Funktionen des X-Functions Moduls der ursprüngliche Displaymodus erhalten werden kann. Version I setzt am Schluß den Displaymodus auf FIX 9 (57 Bytes). Version II benutzt RCLFLAG und STOFLAG aus dem X-Functions Modul, der ursprüngliche Displaymodus bleibt erhalten (63 Bytes). Zeile 09 ist eine Leerstelle (SPACE) im Alpharegister und könnte durch CLA ersetzt werden.

Die eigentliche Berechnung innerhalb des Programmes "Y HOCH X" geschieht nur in den Zeilen 02 bis 08. Der restliche Programmteil erfüllt den Zweck, die Anzeige zu formatieren.

Die Programmbeispiele in diesem Kapitel haben gezeigt, daß es nützlich sein kann, die Stackfunktionen des HP-41 in möglichst großem Umfange auszunutzen. Will man ganz auf die Benutzung der Datenregister verzichten, ist jedoch ein deutlich höherer Programmieraufwand erforderlich. Dies gilt besonders bei Programmen, die als Unterprogramme oder als Hilfsprogramme 'zwischendurch' benötigt werden. Die Angabe SIZE 000 bedeutet, daß ein Programm kein Datenregister benutzt; es kann also eine beliebige Anzahl von Datenregistern definiert sein, deren Inhalt durch das Programm nicht verändert wird.

Bei Verwendung von Stackfunktionen muß man sich jedoch darüber im klaren sein, daß die Stackinhalte während des Programmablaufs fortwährend verschoben werden. Programme, die den Stack intensiv nutzen, sind auf den ersten Blick nicht so leicht zu lesen und zu verstehen wie vergleichbare Programme, die Datenregister verwenden. Zudem haben Veränderungen in solchen Programmen oft unvorhergesehene oder unbeabsichtigte Folgen. Trotzdem sollte man mit der Benutzung von Datenregistern sehr sparsam umgehen.

Der Reiz des Programmierens liegt auch hier darin, daß viele Wege zum gleichen Ziel führen, man muß nur den kennen, der die Möglichkeiten des Rechners - je nach Anforderung - optimal ausnutzt.

2 BASISUMRECHNUNGEN - KOMPAKTE DATENSPEICHERUNG

In vielen Computern, so auch im HP-41, werden Informationen im Hexadezimalsystem gespeichert, um eine möglichst kompakte Speicherung zu erreichen. Da für die einzelnen Ziffern dieses Systems Werte von 0 bis 15 auftreten können, und die Ziffern mit den Werten 10 bis 15 mit den Buchstaben A bis F dargestellt werden, ist der HP-41 mit seinen alphanumerischen Fähigkeiten für Umrechnungsroutinen des Hexadezimalsystems besonders gut geeignet.

Die in diesem Kapitel vorgestellten Programme benutzen Funktionen des X-Functions Moduls und sind nur für ganze positive Zahlen zu verwenden.

2.1 "DH" - Dezimal-Hex-Wandler

```
01◆LBL "DH"
02 CLA

03◆LBL 00
04 RCL X
05 16
06 MOD
07 9
08 -
09 X>0?
10 64
11 X<=0?
12 57
13 +
14 XTOA
15 CHS
16 SIGN
17 AROT
18 CLX
19 16
20 /
21 INT
22 X>0?
23 GTO 00
24 AVIEW
25 END

LBL'DH
END
40 BYTES
```

Das Programm "DH" rechnet eine im X-Register enthaltene Dezimalzahl in die entsprechende Hexadezimalzahl um. Der Rechenbereich dieses Programms ist durch die zehnstellige Anzeige auf 9999999999 = 2540BE3FF begrenzt.

Die Funktion XTOA hängt das durch den Zeichencode im X-Register bestimmte Zeichen rechts an den Inhalt des Alpharegisters an. Durch wiederholte Benutzung dieser Funktion wird das Alpharegister wie bei einem Schreibvorgang von rechts nach links gefüllt. Die entsprechende Umkehrfunktion ATOX löscht ein Zeichen von links aus dem Alpharegister und bringt den zugehörigen Zeichencode ins X-Register. Wiederholtes Ausführen dieser Funktion kann also mit einem Lesen des Alpharegisters verglichen werden.

Um durch das Programm "DH" die richtige Darstellung einer Hexadezimalzahl im Alpharegister zu erreichen, müssen die errechneten Ziffern jedoch nacheinander links an das Alpharegister angehängt werden, da in aufsteigenden Potenzen gerechnet wird.

XTOA, CHS, SIGN, AROT erfüllen die Aufgabe: Das errechnete Zeichen wird zunächst rechts angehängt und dann durch AROT an die vorderste Stelle des Alpharegisters versetzt.

Die in dem Programm enthaltenen Rechenoperationen mit den Zahlen 9, 64 und 57 sind dadurch zu erklären, daß der Zeichencode für die Ziffern 0 bis 9 (48 bis 57) nicht an den Zeichencode für die Ziffern A bis F (65 bis 70) anschließt und daher eine Abgrenzung erforderlich ist. Sofern nur Berechnungen bis zur Größe der Dezimalzahl 16777215 (FFFFFF) erforderlich sind, kann in Zeile 24 statt AVIEW der Befehl ASTO X zur Anzeige der errechneten Hexadezimalzahl eingesetzt werden.

2.2 "HD" - Hex-Dezimal-Wandler

Das Programm "HD" erfüllt genau die umgekehrte Aufgabe, es errechnet aus der Hexadezimalzahl im Alpharegister die entsprechende Dezimalzahl und schreibt diese in das X-Register. Dabei wird zunächst die Anzahl der im Alpharegister dargestellten Ziffern durch die Funktion ALENG festgestellt und das Ergebnis dann in Zeile 16 (DSE Y) als Schleifenzähler verwandt. "HD" ist wesentlich kürzer als "DH", da die Leserichtung von ATOX direkt verwendet werden kann.

Die Funktionsweise der beiden Programme kann als Grundlage für Umrechnungsroutinen für die Zahlensysteme der Basis 2 bis 36 dienen. Dazu ist lediglich die Zahl 16 in den beiden Programmen durch die gewünschte Basiszahl zu ersetzen. Soll das Umrechnungsprogramm variabel für verschiedene Basen eingesetzt werden können, ist die Zahl 16 jeweils durch einen Registeraufruf (RCL NN) zu ersetzen und die gewünschte Basiszahl im Register NN zu speichern.

```
01◆LBL "HD"
02 ALENG
03 0

04◆LBL 00
05 16
06 *
07 ATOX
08 57
09 -
10 X>0?
11 2
12 X<=0?
13 9
14 +
15 +
16 DSE Y
17 GTO 00
18 X<0?
19 CLX
20 END

LBL'HD
END
33 BYTES
```

2.3 "DB" - Dezimal-Binär-Wandler

2.4 "BD" - Binär-Dezimal-Wandler

Programmoptimierung kann bedeuten, ein Programm für einen Spezialfall (Hexadezimalsystem) in ein für eine Vielzahl von Fällen anwendbares Programm umzuwandeln. Optimierung ist aber auch anders möglich, wie die beiden folgenden Programme "DB" und "BD" zeigen.

Das Programm "DB" wandelt eine Dezimalzahl im X-Register in die entsprechende Binärzahl um und schreibt das Ergebnis in das Alpharegister. Da das Alpharegister maximal 24 Stellen aufnehmen kann, hat der Rechenbereich 16777215 als Obergrenze. Das Programm "BD" erfüllt die umgekehrte Aufgabe, es rechnet eine Binärzahl aus dem Alpharegister in die entsprechende Dezimalzahl um und speichert sie im X-Register. Da hier nur zwei verschiedene Zeichencodes (48 und 49) verwendet werden, können die Rechenschritte gegenüber den Programmen "DH" und "HD" vereinfacht werden. Dadurch ergibt sich die augenfällige Kürze dieser Binärwandler, die natürlich auch der Schnelligkeit zugute kommt.

Die beiden Programme "DB" und "BD" können auch für Umrechnungen zur Basis 3, 4, 6 und 8 verwendet werden, wenn die Basiszahl in den Programmen entsprechend ausgetauscht wird.

```
01◆LBL "DB"
02 CLA

03◆LBL 00
04 RCL X
05 2
06 MOD
07 48
08 +
09 XTOA
10 CHS
11 SIGN
12 AROT
13 CLX
14 2
15 /
16 INT
17 X>0?
18 GTO 00
19 AVIEW
20 END

LBL'DB
END
32 BYTES
```

```
01◆LBL "BD"
02 ALENG
03 0

04◆LBL 00
05 2
06 *
07 ATOX
08 LASTX
09 MOD
10 +
11 DSE Y
12 GTO 00
13 END

LBL'BD
END
24 BYTES
```

2.5 "XB" - Dezimal-Binär-Wandler

Nach einem anderen Prinzip arbeitet das Programm "XB". Es benutzt den Status der Flags 00 bis 07 zur Umwandlung einer Dezimalzahl in die entsprechende Binärzahl.

Der Rechenbereich dieses Programms ist auf 0 bis 255 begrenzt. Der Originalzustand der Flags 00 bis 07 wird durch das Programm wiederhergestellt (Zeile 14). Dieses Programm erfüllt bezüglich der Zahlen 0 bis 255 den gleichen Zweck wie das Programm "DB". Es ist jedoch wesentlich schneller. Zeile 05 enthält wieder einen Leerschritt im Alpharegister (SPACE) und kann durch CLA ersetzt werden.

```
01◆LBL "XB"
02 X<>F
03 7
04 CHS
05 " "

06◆LBL 00
07 FS? IND
          X
08 "⊢1"
09 FC? IND
          X
10 "⊢0"
11 ISG X
12 GTO 00
13 X<>Y
14 X<>F
15 AVIEW
16 END

LBL'XB
END
34 BYTES
```

2.6 "XA"/"AX" - Kompakte Datenspeicherung

Da im Alpharegister des HP-41 Zeichencodes von 0 bis 255 eindeutig verschlüsselt werden können (obwohl diese in der Anzeige nicht immer unterscheidbar sind), müßte sich eine besonders kompakte Datenspeicherungsmethode dadurch ergeben, daß man Zahlen im Zahlensystem zur Basis 256 im Alpharegister verschlüsselt. Da jedoch Null-Bytes im Alpharegister zu Problemen bei der Codierung und Decodierung führen, müssen diese vermieden werden. Das folgende Programm "XA"/"AX" enthält daher Basisumwandlungsroutinen zur Basis 255, der Zeichencode wird zur Vermeidung von Null-Bytes jeweils um 1 erhöht. Bei dem hier verwendeten Codiersystem gibt das erste Byte an, wie viele der folgenden Bytes die nächste codierte Zahl darstellen. Das ist ein Prinzip, wie es in Großcomputern verwandt wird, um Daten mit variabler Feldlänge zu speichern.

Sofern nur eine Zahl codiert wird, ist dieses zusätzliche Kennzeichnungsbyte nicht erforderlich. Diese Routine kann dazu verwandt werden, codierte Zahlen fortlaufend in einem Record eines ASCII-Files abzulegen und so den Speicherplatzbedarf gering zu halten.

Die Routine liefert nur exakte Ergebnisse bis einschließlich 2550000253. Für größere Zahlen sind durch Rechenungenauigkeiten geringe Abweichungen möglich.

```
01◆LBL "XA"
02 CLA

03◆LBL 00
04 RCL X
05 255
06 MOD
07 1
08 +
09 XEQ 01
10 CLX
11 255
12 /
13 INT
14 X>0?
15 GTO 00
16 ALENG

17◆LBL 01
18 XTOA
19 CHS
20 SIGN
21 AROT
22 RTN

23◆LBL "AX"
24 ATOX
25 0

26◆LBL 02
27 255
28 *
29 ATOX
30 1
31 -
32 +
33 DSE Y
34 GTO 02
35 END

LBL'XA
LBL'AX
END
64 BYTES
```

Wie viele Bytes für eine Dezimalzahl verbraucht werden, ist in der folgenden Tabelle festgehalten:

0 ... 254	2 Bytes
255 ... 65024	3 Bytes
65025 ... 16581374	4 Bytes
16581375 ... 4228250623	5 Bytes
bis ca. 1,05 E 12	6 Bytes
bis ca. 2,7 E 14	7 Bytes

Da die Speicherung jeder Zahl in einem Datenregister 7 Bytes beansprucht, sind besonders bei großen Datenmengen, durch die Codierungsroutine "XA"/"AX" erhebliche Einsparungen möglich. Zu beachten bleibt jedoch, daß ein direkter schneller Zugriff auf die Daten nicht mehr möglich ist.

3 ZUFALLSZAHLEN - LOTTOZAHLEN

Ein programmierbarer Rechner ist ein ideales Werkzeug zur Auswertung von Zufallsgeneratoren und Statistiken. Mit dem im folgenden dargestellten Programm "STAT" kann dies modellhaft verdeutlicht werden.

3.1 "STAT" - Statistische Auswertung

Das Programm "STAT" erfordert SIZE 050. Es dient dazu, Zufallsgeneratoren, die in diesem Fall Lottozahlen (1 bis 49) erzeugen, statistisch zu beurteilen.

Der Zufallsgenerator, den man testen möchte, muß nach Zeile 05 in das Programm eingefügt werden. Dazu kann Register 00 benutzt werden, das Programm benutzt nur Register 1 bis 49. Es kann auch ohne Zufallsgenerator in der vorliegenden Form direkt verwendet werden. Die Anzahl der gewünschten Zufallszahlen ist einzugeben, und das Programm zu starten.

Nachdem alle Datenregister gelöscht wurden, (Zeile 02) wird die Wurzel aus 7 als Startwert zur Erzeugung der Lottozahlen vorgegeben. Der Befehl ST+ IND Y in Zeile 12 erhöht den Wert des Registers entsprechend der ermittelten Zahl um 1. Somit steht nach Ablauf dieses Programmabschnitts (LBL 00) in jedem Register, wie oft die Zahl entsprechend der Registernummer ermittelt wurde. Ist z.B. die Zahl 38 sieben mal 'gezogen' worden, findet sich in Register 38 die Zahl 7. Im nächsten Programmabschnitt (LBL 01) wird zu jedem der benutzten Datenregister 1 % der Registernummer addiert. D.h. aus der Beispielzahl in Register 38 wird 7,38.

```
01◆LBL "STA
       T"
02 CLRG
03 7
04 SQRT

05◆LBL 00
06 FRC
07 49
08 *
09 1
10 +
11 SIGN
12 ST+ IND
         L
13 X<> L
14 DSE Y
15 GTO 00
16 49

17◆LBL 01
18 1
19 %
20 ST+ IND
         Y
21 RDN
22 DSE X
23 GTO 01
24 ,049
25 XEQ "SO"
26 END

LBL'STAT
END
50 BYTES
```

Durch dieses Verfahren werden in einem Register zwei Zahlen gespeichert: Vor dem Komma steht die Anzahl, wie oft eine Zahl ermittelt worden ist, und hinter dem Komma steht diese ermittelte Zahl selbst. Die Zeilen 24 und 25 rufen das Sortierprogramm "SO" auf, das in Kapitel 9 vorgestellt wird. Sofern ein Zufallsgenerator in das Programm eingefügt wurde, der Register 00 benutzt, ist Zeile 24 vor der Sortierung in 1,049 zu ändern, damit Register 00 nicht in die Sortierung miteinbezogen wird. Nach Ablauf des Programms (einschließlich Sortierung) ist in den Registern 01 bis 49 in aufsteigender Reihenfolge erkennbar, wie oft eine Zahl 'gezogen' wurde.

Mit Hilfe des Programms "STAT" kann man sehr gut erkennen, daß die Verteilung der Zahlen bei großen Zahlenmengen kaum noch von der Wahl des Zufallsgenerators abhängt.

3.2 "DTA" - Kompakte Datenspeicherung

```
01◆LBL "DTA
       "
02 6
03 FRC
04 CLA
05 ENTER↑

06◆LBL 00
07 X<>Y
08 RDN
09 TONE 9
10 STOP
11 X=0?
12 GTO 01
13 XTOA
14 DSE L
15 GTO 00
16 SIGN
17 ASTO X
18 STO IND
         Z
19 CLA
20 LASTX
21 RTN

22◆LBL 01
23 RCL IND
         Y
24 XTOA

25◆LBL 02
26 ATOX
27 PSE
28 DSE L
29 GTO 02
30 END
```

Für eine statistische Auswertung ist zunächst von großer Bedeutung, sehr viele Daten möglichst effektiv zu speichern, d.h. so wenig wie möglich Speicherplatz dazu zu verwenden. Ergänzend zu dem im vorigen Kapitel gezeigten Beispiel der variablen Datenspeicherung ("XA"/"AX") soll das Programm "DTA" zeigen, wie man sechs Zahlen in einem Register speichern kann.

Das Programm "DTA" dient dazu, sechs positive ganze Zahlen im Bereich von 1 bis 255 in einem Register zu speichern. Möchte man z.B. alle gezogenen Lottozahlen eines Jahres speichern, so wäre dies mit der normalen Speicherungsmethode (je Zahl ein Datenregister) nicht möglich, da der Speicherplatz des Rechners nicht ausreichen würde. Außerdem gäbe es erhebliche organisatorische Probleme (in welchem Register ist welche Zahl?). Das Programm "DTA" löst diese Aufgabe, indem sechs Zahlen als Alpha-Bytes in einem Register gespeichert werden. Das Programm dient sowohl zum Speichern als auch zum Zurückrufen der Zahlen.

Bevor das Programm gestartet wird, ist der Index entsprechend des Registers einzugeben, das angesprochen werden soll.

Der Ton (Zeile 09) kennzeichnet den Programmstopp an dem Eingaben erfolgen können. Man kann nun sechsmal je eine Zahl (Bereich 1 bis 255) eingeben und mit R/S fortfahren. Dann sind die sechs eingegebenen Zahlen in dem vorher bezeichneten Register gespeichert. Startet man nun das Programm erneut mit derselben Registernummer und führt nach dem Ton sofort (ohne Zahleneingabe) mit R/S weiter, so werden die vorher eingegebenen sechs Zahlen zurückgerufen.

Auch das Organisationsproblem läßt sich mit dieser Speichermethode gut lösen: Die Nummer des verwendeten Datenregisters gibt eine gute Ordnungsmöglichkeit. So lassen sich in 52 Datenregistern alle gezogenen Lottozahlen eines Jahres speichern. In Zeile 24 wird ein Nebeneffekt von XTOA benutzt: Wenn das X-Register Alpha-Daten enthält, werden diese an das Alpharegister angehängt. Hier könnte also ebenso der Befehl ARCL X verwendet werden.

3.3 "DFL" - Kompakte Datenspeicherung

Das Programm "DFL" erfüllt einen ähnlichen Zweck wie "DTA", jedoch werden hier die eingegebenen Zahlen in einem Datenfile des erweiterten Speicherbereichs, also nicht in Datenregistern abgelegt. Das Datenfile ist vorher in der erforderlichen Größe anzulegen (CRFLD). Ist der File-Zeiger nicht auf das gewünschte Datenfile positioniert, muß bei der ersten Benutzung des Programms "DFL" der File-Name in das Alpharegister eingegeben werden, sonst ist die Bedienung wie beim Programm "DTA".

Die Programme in diesem Kapitel haben gezeigt, daß man mit recht kurzen Routinen erhebliche Einsparungen des Platzbedarfes bei der Datenspeicherung erreichen kann. Daß alle gezeigten Routinen nur für positive ganze Zahlen verwendbar sind, ist keine für die Praxis wesentliche Einschränkung. Durch einfache Kennzeichnung kann man auch negative Werte in die Routinen mit einbeziehen, und die Verschiebung des Kommas zur Verwendung von Dezimalzahlen ist ebenfalls problemlos realisierbar.

```
01◆LBL "DFL
       "
02 SEEKPTA
03 6
04 FRC
05 CLA
06 ENTER↑

07◆LBL 00
08 X<>Y
09 RDN
10 TONE 9
11 STOP
12 X=0?
13 GTO 01
14 XTOA
15 DSE L
16 GTO 00
17 SIGN
18 ASTO X
19 SAVEX
20 CLA
21 LASTX
22 RTN

23◆LBL 01
24 GETX
25 XTOA

26◆LBL 02
27 ATOX
28 PSE
29 DSE L
30 GTO 02
31 END
```

4 EINFACHES CODIEREN VON TEXT

Dieses Kapitel zeigt mit mehreren Programmbeispielen, daß die Benutzung des X-Functions Moduls (ALENG, XTOA, ATOX) außerordentlich kurze Routinen zur Textcodierung ermöglicht.

4.1 "CA" - Codiere Alpha

4.2 "DA" - Decodiere Alpha

```
01◆LBL "CA"
02 ALENG

03◆LBL 00
04 ATOX
05 RCL Y
06 3
07 *
08 +
09 XTOA
10 RDN
11 DSE X
12 GTO 00
13 END

LBL'CA
END
26 BYTES
```

```
01◆LBL "DA"
02 ALENG

03◆LBL 00
04 ATOX
05 RCL Y
06 3
07 *
08 -
09 XTOA
10 RDN
11 DSE X
12 GTO 00
13 END

LBL'DA
END
26 BYTES
```

Die Routinen "CA" und "DA" verschlüsseln den Text innerhalb des Alpharegisters selbst. Die Routine "CA" verschlüsselt den Text, und durch die Routine "DA" wird wieder entschlüsselt. Es ist auch möglich, die Verschlüsselung häufig zu durchlaufen; dann muß die Entschlüsselung ebenfalls entsprechend mehrfach durchlaufen werden.

Trotz der außerordentlichen Kürze der Routinen werden gleiche Buchstaben nicht gleich verschlüsselt, da die Position des Buchstabens innerhalb des Alpharegisters in die Codierungsberechnung mit einbezogen wird. Außerdem läßt sich auch "DA" als Verschlüsselungsprogramm verwenden; dann wirkt "CA" als Entschlüsselungsprogramm. Datenregister werden von den Programmen nicht benutzt.

Gibt man Text ins Alpharegister ein, läßt den Alpha-Modus eingeschaltet und startet das Programm mit R/S, so ist die Verschlüsselung direkt zu beobachten. Durch Veränderung der Rechenschritte ist nach dem Prinzip dieser Programme eine individuelle Chiffrierung möglich. Dabei muß darauf geachtet werden, daß das Rechenergebnis immer im Bereich zwischen 1 und 255 verbleibt.

4.3 "CODE" - Codiere/Decodiere Alpha

Das Programm "CODE" dient gleichzeitig als Verschlüsselungs- und Entschlüsselungsprogramm. Die Verschlüsselung findet ebenfalls innerhalb des Alpharegisters statt, Datenregister werden nicht benutzt. Zur Verschlüsselung wird eine vor dem Programmstart in das X-Register einzugebende Zahl verwendet.

Für normale Textverschlüsselung im Bereich der Buchstaben A bis Z sollte eine Verschlüsselungszahl im Bereich von 218 bis 320 gewählt werden, um den Text nach der Verschlüsselung unkenntlich zu machen. Dieses Programm verschlüsselt gleiche Buchstaben mit gleichen Werten. Trotzdem ist der Effekt, gemessen an dem geringen Programmaufwand, recht erstaunlich.

```
01◆LBL "COD
       E"
02 SIGN
03 ALENG

04◆LBL 00
05 ATOX
06 LASTX
07 -
08 XTOA
09 RDN
10 DSE X
11 GTO 00
12 SIGN
13 CLST
14 END

LBL'CODE
END
28 BYTES
```

Dieselbe Zahl, mit der verschlüsselt wurde, ist auch zur Entschlüsselung einzugeben. Die Zeilen 12 und 13 dienen dazu, jeden Hinweis auf die Art der Verschlüsselung aus dem Rechner (insbesondere LastX-Register) zu löschen.

4.4 "C-A" - Codiere Alpha

4.5 "D-A" - Decodiere Alpha

Eine weitere Alternative zur Textcodierung stellen die Programme "C-A" und "D-A" dar. Bei diesen Programmen wird der Inhalt des Alpharegisters in Datenregister verschlüsselt. Je nach Länge des zu verschlüsselnden Textes ist also maximal SIZE 025 erforderlich. Diese Programme verarbeiten sämtliche Bytewerte (0 bis 255) einwandfrei; gleiche Zeichen werden nicht gleich verschlüsselt.

Zur Verschlüsselung ist das Programm "C-A" zu starten. Nach dem Ton stoppt das Programm, und es wird der Inhalt des Alpharegisters angezeigt. Soll dieser Inhalt verschlüsselt werden, ist mit R/S zu starten.

Andernfalls kann der zu verschlüsselnde Text eingegeben werden, und anschließend ist das Programm mit R/S zu starten. Nach Programmablauf ist der Inhalt des Alpharegisters in Zahlenwerten in den Datenregistern verschlüsselt; das Alpharegister ist gelöscht.

Zur Entschlüsselung dient das Programm "D-A", es ist ohne jede Eingabe zu starten. Nach der Entschlüsselung bleiben die Inhalte der Datenregister unverändert erhalten, so daß mehrfach darauf zurückgegriffen werden kann. Durch Veränderung der Rechenschritte ist auch bei diesen Programmen eine individuelle Codierung möglich.

```
01◆LBL "C-A
       "
02 CLRG
03 AON
04 TONE 6
05 STOP
06 AOFF
07 ALENG
08 STO 00

09◆LBL 00
10 ATOX
11 LN
12 STO IND
          Y
13 RDN
14 ST/ IND
          X
15 ALENG
16 X≠0?
17 GTO 00
18 CLST
19 END

LBL'C-A
END
34 BYTES
```

```
01◆LBL "D-A
       "
02 CLA
03 RCL 00
04 RCLFLAG
05 FIX 0

06◆LBL 00
07 RCL IND
          Y
08 RCL Z
09 *
10 X≠0?
11 E↑X
12 RND
13 XTOA
14 RDN
15 DSE Y
16 GTO 00
17 STOFLAG
18 TONE 8
19 CLST
20 AVIEW
21 END

LBL'D-A
END
38 BYTES
```

Die in diesem Kapitel gezeigten Routinen erheben nicht den Anspruch auf besonders sichere oder komplizierte Chiffrierung, sie lassen jedoch erkennen, daß mit geringem Aufwand an Programmspeicherplatz bereits ein recht brauchbares Ergebnis erreicht werden kann.

5 EINSTIEG IN DIE SYNTHETISCHE PROGRAMMIERUNG

Unter der synthetischen Programmierung versteht man die Erzeugung von Programmschritten, die durch normale Methoden, wie sie im Benutzerhandbuch genannt sind, nicht hergestellt werden können.

Um dies zu erreichen, sind in der Regel Hilfsprogramme erforderlich, die selbst wiederum synthetische Befehle enthalten. Die entscheidende Frage ist also: Wie ist mit normalen Hilfsmitteln ein Einstieg in die synthetische Programmierung möglich?

Zunächst muß gesagt werden, daß synthetische Befehle, wenn sie einmal im Programmspeicher enthalten sind, ihre ganz normale definierte Funktionswirkung haben und auch durch die üblichen Speichermedien wie Magnetkarten, Barcode oder Digital-Cassette gespeichert und übertragen werden können. Da die Eingabe einiger synthetischer Befehle recht aufwendig ist und nur über Hilfsprogramme erreicht werden kann, sind zur schnelleren Benutzung für den Leser dieses Buches sämtliche Programme im Anhang als Barcode abgedruckt. Die Programme können also 'einfach' eingelesen und benutzt werden, als wären nur normale Befehle enthalten.

Die Grenze zwischen "normaler Programmierung" und synthetischer Programmierung ist fließend: Durch Einführung des X-Functions Moduls sind grundlegende Schritte mit normalen Mitteln programmierbar geworden, die vorher nur sehr aufwendig mit Hilfe der Synthetik erreichbar waren. Hier sind besonders die Befehle XTOA und ATOX von Bedeutung. Das Modul ist daher eine wesentliche Grundlage für die fortschrittliche Programmierung; alle seine Funktionen sind in dem neuen Taschencomputer HP-41CX enthalten. Trotzdem ist es mit den Funktionen dieses Moduls allein noch nicht möglich, ein Hilfsprogramm für synthetische Programmierung herzustellen, das selbst keine synthetischen Befehle enthält.

Dieser entscheidende Schritt ist jedoch mit dem IL-Development Modul möglich. Wegen der grundlegenden Bedeutung des IL-Development Moduls wird darauf in Kapitel 10 besonders eingegangen.

Die synthetische Programmierung hat im wesentlichen vier Aufgabenbereiche:

- Durch einige synthetische Befehle ist es möglich, die Organisation des Rechners kennenzulernen.
- Neue synthetische Befehle ermöglichen Problemlösungen, die mit normalen Programmbefehlen nicht möglich sind.
- Einige synthetische Befehle verringern den Programmspeicherplatzbedarf und/oder erhöhen die Rechengeschwindigkeit.
- Einige synthetische Befehle erlauben die Benutzung von Statusregistern als Zwischenspeicher; der Bedarf an Datenregistern kann dadurch eingeschränkt werden.

Da die Entdeckung und Entwicklung der synthetischen Programmierung zunächst von dem amerikanischen Anwenderclub für Taschencomputer (PPC) ausging, sind Publikationen über die synthetische Programmierung meist in englischer Sprache erschienen. Eine grundlegende und umfassende, nicht nur auf die Theorie beschränkte Darstellung der synthetischen Programmierung, ist in den Büchern von Keith Jarett und im PPC ROM Users Manual enthalten (Siehe Literaturhinweise im Anhang). Das PPC ROM Users Manual ist eigentlich ein Benutzerhandbuch für das von Mitgliedern des PPC programmierte PPC Modul. Alle 122 Programme, die im PPC Modul enthalten sind, werden in diesem 500 Seiten umfassenden Handbuch schrittweise erklärt. Zusätzlich ist eine vorbildliche Einführung in die Theorie der synthetischen Programmierung enthalten. Das PPC ROM Users Manual bietet daher in kompakter und übersichtlicher Form eine Fülle grundlegender Information für die synthetische Programmierung.

Es ist empfehlenswert, daß sich der Leser zunächst mit den theoretischen Grundlagen der synthetischen Programmierung vertraut macht. In diesem und in den folgenden Kapiteln soll weniger die Theorie, als vielmehr die praktische Anwendung und der sinnvolle Nutzen der Synthetik im Vordergrund stehen. Programme mit synthetischen Funktionen und Programmiertechniken werden vorgestellt und in ihrer Funktionsweise analysiert.

Voraussetzung zum Verständnis ist das Wissen um die Organisationsstruktur des HP-41. Wesentlich dazu ist zunächst, daß der Rechner in Registern organisiert ist, jedes Register hat eine absolute Registeradresse, die nicht geändert werden kann. Jedes Register besteht aus 7 Bytes (Byte Nr. 0 bis 6). Alle Register sind identisch strukturiert, unabhängig davon, ob es sich um Programmregister, Datenregister, Tastenzuordnungsregister, Alarmregister (Time Modul) oder Register im erweiterten Speicherbereich handelt. Es gibt also grundsätzlich keine zwingende Zuordnung eines bestimmten Registers zu einer bestimmten Aufgabe. Lediglich 16 Statusregister (absolute Registeradressen 0 bis 15) haben eine feste Aufgabenzuordnung.

Eines dieser Statusregister, Register c, enthält sämtliche Schlüsselinformationen über die Aufgabenzuordnung aller übrigen Register. Hier ist festgelegt, welche Register als Datenregister und welche Register als Programmregister definiert sind. Diese Abgrenzung wird durch die Funktion SIZE oder PSIZE verändert.

Das geschieht dadurch, daß in Register c die absolute Adresse des ersten Datenregisters (Register 00) gespeichert wird. Diese absolute Adresse des relativen Datenregisters 00 bezeichnet man als Curtain. Bei der normalen Programmierung sind die absoluten Adressen ohne Bedeutung. Der Zugriff auf die Datenregister erfolgt (STO, RCL, ...) über relative Adressen. Das Wissen um die absoluten Registeradressen ist jedoch für die synthetische Programmierung von grundlegender Bedeutung.

Ein weiteres Statusregister, Register b, wird in den folgenden Programmen verwandt. In diesem Register ist zunächst die absolute Adresse des Programmzeigers gespeichert (Byte 0 und 1). Bei der Abarbeitung eines Programmes oder der Ausführung von Sprungbefehlen wird diese Adresse fortlaufend angepaßt. Die Befehlsfolge RCL b, ... , STO b verdeutlicht die Funktion des Programmzeigers. Fügt man zwischen RCL b und STO b zum Beispiel TONE 8 ein und startet bei RCL b, so wird der Ton immer wieder ausgeführt, bis das Programm von der Tastatur aus angehalten wird.

Wegen der Vielzahl der Anwendungsmöglichkeiten der Programmzeigermanipulationen werden diese in Kapitel 7 ausführlich behandelt. In den weiteren Bytes des Statusregisters b ist zunächst die erste Rücksprungadresse gespeichert. Tritt in einem Programm ein Befehl XEQ NN auf, so wird die absolute Adresse des nächsten Programmbefehls als Rücksprungadresse gespeichert; sobald im Programm das nächste RTN oder END auftritt, wird diese Rücksprungadresse zur Programmzeigeradresse, und der Programmbefehl wird ausgeführt.

5.1 "LB" - Load Bytes I

Ein klassisches Synthetikprogramm ist "LB". Mit diesem Programm können beliebige Bytewerte in den Programmspeicher eingegeben werden. Es enthält drei synthetische Anweisungen, die zur Funktion des Programms erforderlich sind.

Diese Anweisungen können mit einem Tastenzuordnungsprogramm erzeugt werden und dann problemlos in den Programmspeicher eingegeben werden (z.B. Programm "KA", Seite 65). Das hier vorgestellte "LB" benutzt keine Datenregister, entweder das X-Functions Modul oder der HP-41CX werden benötigt. Bei der Entwicklung dieses Programms wurde besonderer Wert auf Kürze und Schnelligkeit gelegt.

Programmbedienung "LB" (Load Bytes I)

1. RTN ausführen.

2. Man gehe zu der Programmzeile, in der synthetische Bytes eingegeben werden sollen.

3. PRGM einschalten; eingeben: XEQ "LB", STOP, +, +, +, ... als Pufferbytes eingeben (man kann hier jeden Ein-Byte-Befehl eingeben). Man muß mindestens 19 Pufferbytes eingeben, um 1 bis 6 Bytes synthetisch laden zu können. Jedes angefangene zusätzliche Register erfordert die Eingabe von sieben weiteren Pufferbytes.

4. Zu der Programmzeile zurückgehen, die XEQ "LB" enthält, PRGM ausschalten, R/S.

5. Der Ton zeigt den Programmstopp an, um die Dezimalwerte der gewünschten synthetischen Bytes einzugeben. Man sieht jeweils in der Anzeige die letzte Eingabe.

6. Dezimalwert des gewünschen Bytes eingeben (ohne ENTER), dann R/S. Auf diese Weise ist für die weiteren Eingaben fortzufahren.

7. Um die Eingaben zu beenden: nur R/S (ohne Eingabe).

8. Nach dem BEEP: PRGM an und die übriggebliebenen Pufferbytes, sowie XEQ "LB" und STOP löschen.

Man achte darauf, daß genügend Pufferbytes eingegeben werden! Während der Eingabe dürfen die Stackregister, LastX und Alpha, nicht verändert werden. Das Programm sollte nicht angehalten werden; zwischen den Zeilen 40 und 44 erhält man sonst MEMORY LOST. Hält das Programm in Zeile 08 mit NONEXISTENT, wurde entweder "LB" von der Tastatur aus gestartet oder zu Beginn RTN nicht ausgeführt.

Wie funktioniert nun die synthetische Eingabe von Programmbytes mit "LB"? Da das Programm mit XEQ "LB" aufgerufen wird, ist die absolute Adresse des Registers, in das die ersten Bytes synthetisch eingegeben werden sollen, als erste Rücksprungadresse im Register b enthalten. Ein Großteil des Programms "LB" dient dazu, diese erste Rücksprungadresse zu decodieren (Zeilen 03 bis 22). Dazu ist deshalb so großer Aufwand erforderlich, weil Fehlbedienungsmöglichkeiten ausgeschlossen werden sollen. Außerdem müssen sämtliche denkbaren Fälle von Null-Bytes in Register b berücksichtigt werden. Der Befehl GTO IND Z benutzt als Argument die Anzahl der Bytes, die in Register b gestanden haben. Das Programm soll nur einwandfrei arbeiten, wenn genau eine Rücksprungadresse in Register b enthalten war. Das ist nur dann der Fall, wenn Register b entweder vier oder drei Bytes enthielt.

```
01◆LBL "LB"
02 CLA
03 RCL b
04 X<> [
05 ALENG
06 7
07 ATOX
08 GTO IND
          Z

09◆LBL 04
10 2
11 MOD
12 256
13 *
14 ALENG
15 2
16 -
17 ATOX
18 *
19 X<0?
20 CLX
21 +

22◆LBL 03
23 2
24 -
25 FRC
26 SF 22
```

```
27◆LBL 00
28 FS? 22
29 TONE 9
30 FS?C 22
31 STOP
32 FC? 22
33 ,
34 XTOA
35 X<>Y
36 RDN
37 DSE Y
38 GTO 00
39 7
40 X<> c
41 X<> [
42 STO IND
            L
43 X<> [
44 X<> c
45 X<>Y
46 CLA
47 DSE L
48 FS? 22
49 GTO 00
50 BEEP
51 END

LBL'LB
END
83 BYTES
```

Deshalb sind als Sprungziele LBL 04 und LBL 03 vorgesehen. Bis zur Zeile 22 ist aus Register b die absolute Adresse der Stelle errechnet worden, an der der Befehl STOP nach XEQ "LB" steht; der Inhalt von Register b, also auch die gespeicherte Rücksprungadresse, wird dabei nicht verändert. Fortlaufende Programmzeilen haben absteigende absolute Adressen, d.h., die Stelle an der die synthetischen Bytes eingegeben werden sollen, hat eine niedrigere absolute Adresse als die Stelle, an der der Stoppbefehl steht. Es wurde jedoch aus dem Inhalt des Register b nicht die absolute Byteadresse sondern nur die absolute Registeradresse des Stoppbefehls decodiert. Die nach dem Stoppbefehl eingegebenen Pufferbytes sollen später durch die einzugebenden synthetischen Befehle überschrieben werden. Damit ausgeschlossen ist, daß der Stoppbefehl selbst auch überschrieben wird, wird von der errechneten absoluten Adresse 2 abgezogen. In Zeile 25 wird die so errechnete absolute Registeradresse in das LastX-Register gespeichert und gleichzeitig im X-Register eine 0 erzeugt.

Die folgenden Zeilen 27 bis 38 stellen die eigentliche Eingabeschleife dar. Solange noch Eingaben gemacht werden (Flag 22), wird ein Ton erzeugt und in Zeile 31 das Programm angehalten.

In Zeile 34 wird der eingegebene Wert als Zeichen an das Alpharegister angehängt. Zeile 37 (DSE Y) bewirkt, daß jeweils 7 Eingaben zusammengefaßt werden (ein Register enthält 7 Bytes). Sofern keine weitere Eingabe erfolgt, wird ein Null-Byte an das Alpharegister angehängt (Zeilen 32 und 33). Wurde das siebte Byte eingegeben, wird der Sprungbefehl in Zeile 38 nicht ausgeführt. Die 7 in Zeile 39 ist schon der Schleifenzähler für die nächsten 7 Byte-Eingaben und erscheint später wieder im Register Y. Zeile 40 tauscht nun zunächst den Inhalt von Register c mit dieser 7. Das bewirkt, daß der Curtain auf die absolute Registeradresse 000 gesetzt wird. Damit sind für diesen Fall die absoluten und relativen Adressen identisch.

Register c enthält aber neben den Schlüsselinformationen zur Registeradressierung auch noch einen Kontrollwert, der grundsätzlich nicht verändert werden darf. Entspricht dieser Wert nicht mehr der vom Betriebssytem erwarteten Größe, so werden alle Speicherinhalte sofort gelöscht (MEMORY LOST). Dieser Kontrollwert wird während eines laufenden Programms vom Betriebssystem nicht geprüft. Sobald das Programm jedoch angehalten wird oder selbsttätig anhält, muß der Inhalt von Register c unbedingt wieder den richtigen Kontrollwert enthalten. Während Register c die Zahl 7 enthält, darf das Programm daher keinesfalls angehalten werden (Zeilen 40 bis 44).

Zeile 41 überträgt den Inhalt der ersten 7 Bytes des Alpharegisters (Register M) in das X-Register. Dies sind genau die 7 Bytes, die als Programmbefehle eingegeben wurden. Zeile 42 speichert diese in das Datenregister mit der relativen Adresse, die im LastX-Register angegeben ist. Da der Curtain auf 000 gesetzt wurde, die relative Adresse (Datenregisteradresse) also gleich der absolute Adresse ist, ist das durch STO IND L angesprochene Register das Programmregister entsprechend der errechneten Rücksprungadresse aus Register b. Die Zeilen 43 bis 45 stellen im Register c wieder den alten Zustand her, bringen die 7 in Register Y und den letzten Eingabewert in Register X. In Zeile 47 wird die absolute Adresse im LastX-Register um eins verringert (zur Vorbereitung der Eingabe der nächsten Programmbefehle) und falls der letzte Befehl eingegeben wurde, also eventuell noch weitere Eingaben gewünscht werden, wird erneut zur Eingaberoutine (LBL 00) verzweigt.

Die Auffüllung der nicht benötigten Bytes mit Null-Bytes, bis die Anzahl von 7 erreicht ist, ist erforderlich, damit Mehr-Byte-Befehle, die teilweise in einem Register und in dem nächsten Register gespeichert werden, lückenlos aneinander anschließen.

Sofern keine Eingaben mehr gemacht werden, wird der Sprungbefehl in Zeile 49 nicht ausgeführt, es ertönt BEEP und das END bewirkt den Rücksprung ins aufrufende Programm. Dort wird also der Befehl STOP noch ausgeführt, und der Programmzeiger wird auf das erste Pufferbyte positioniert.

Entscheidend zum Verständnis der Funktionsweise des Programms "LB" ist, daß zunächst nacheinander die Bytewerte der gewünschten Programmbefehle im Alpharegister gespeichert werden. Dann wird das Betriebssystem des Rechners durch eine Manipulation des Registers c dazu veranlaßt, ein Programmregister vorübergehend als Datenregister zu behandeln, damit die eingegebenen Werte dort gespeichert werden können. So ist es möglich, mit Hilfe des Programms "LB" jede denkbare Bytefolge als Programmbefehl zu laden. Natürlich können auf diese Weise auch normale Programmbefehle geladen werden; dies empfiehlt sich besonders bei solchen Befehlen, die recht umständlich einzugeben sind. Enthält ein einzugebendes Programm einige synthetische Befehle, so sollte man zunächst nur diese direkt hintereinander in einem Arbeitsgang mit "LB" eingeben und anschließend die normalen Befehle wie üblich einfügen.

5.2 "LB" - Load Bytes II

```
01◆LBL "LB"
02 CLST
03 SIZE?
04 2
05 -
06 X<0?
07 CLX
08 PSIZE
09 CLRG
10 SF 22

11◆LBL 00
12 STO IND
      00
13 RDN
14 DSE 00
15 FC? 22
16 GTO 02
17 CLA
18 7
19 X<>Y

20◆LBL 01
21 FS?C 22
22 STOP
23 FC? 22
24 ,
25 XTOA
26 X<>Y
27 RDN
28 DSE Y
29 GTO 01
30 RCL [
31 GTO 00
32◆LBL 02
33 "*"
34 RCL c
35 X<> [
36 X<> \
37 ASHF
38 POSA
39 AROT
40 X>0?
41 CLX
42 X≠0?
43 ATOX
44 ATOX
45 16
46 MOD
47 LASTX
48 X↑2
49 *
50 +
51 DSE X
52 ENTER↑
53 "◆\
      *i  "
54 256
55 /
56 XTOA
57 X<> L
58 MOD
59 XTOA
60 RCL \
61 STO 00
62 X<> [
63 X<> c
64 CLST
65 BEEP
66 END

LBL'LB
END
113 BYTES
```

Eine andere Methode zum Laden synthetischer Programmbefehle verwendet das zweite "LB" Programm. Zur Benutzung dieses Programmes müssen keine Pufferbytes vorgegeben werden. Dadurch ist in diesem Punkte eine Fehlbedienung ausgeschlossen. Die Inhalte aller Datenregister werden jedoch durch das Programm gelöscht. Mit diesem Programm ist es nicht möglich, synthetische Programmbefehle nachträglich in ein bestehendes Programm einzugeben, diese Programmversion ist vielmehr dahingehend ausgelegt, möglichst schnell eine große Anzahl synthetischer Programmbefehle zusammenhängend eingeben zu können. Normale Programmbefehle können dann ebenfalls nachträglich eingefügt werden.

Das Programm "LB" enthält eine synthetische Textzeile, deren Bytewerte nicht aus dem Ausdruck erkennbar sind (Zeile 53). Die Dezimal-Byte-Werte der Zeile sind:

248, 192, 0, 11, 32, 176, 1, 105, 32.

Mit dieser Textzeile darf das Programm nur mit einem HP-41CV, einem HP-41CX oder einem HP-41C mit 4 Memory Modulen oder mit einem Quad Modul benutzt werden. Sonst muß die Textzeile wie folgt geändert werden:

HP-41C + 3 MM	248, 192, 0, 11, 28, 176, 1, 105, 28
HP-41C + 2 MM	248, 192, 0, 11, 24, 176, 1, 105, 24
HP-41C + 1 MM	248, 192, 0, 11, 20, 176, 1, 105, 20
HP-41C	248, 192, 0, 11, 16, 176, 1, 105, 16

Programmbedienung "LB" (Load Bytes II):

1. Zunächst ist sicherzustellen, daß die Textzeile entsprechend der Rechnerausstattung ist.

2. Zur Speicherung der synthetischen Bytes stehen drei Register weniger zur Verfügung als Datenregister definiert sind. Deshalb muß mindestens SIZE 004 definiert sein. (SIZE 019 erlaubt z.B. 16 x 7 = 112 Bytes zu laden.)

3. XEQ "LB".

4. Wenn das Programm stoppt, sind Dezimalwerte der zu ladenden Bytes einzugeben, danach R/S. Auf diese Weise kann entsprechend der Kapazität der Datenregister fortgefahren werden. Wenn BEEP ertönt, siehe 6; das Programm beendet automatisch die Eingabemöglichkeit, wenn alle Datenregister benutzt sind. In der Anzeige ist jeweils die letzte Eingabe sichtbar.

5. Um die Eingaben zu beenden: R/S (Ohne Eingabe).

6. Nach dem BEEP: GTO .. ! Danach ist SIZE 000 definiert (also keine Datenregister vorhanden).

Anschließend ist CAT 1 auszuführen, aber sofort mit R/S wieder anzuhalten, sobald das erste END in der Anzeige erscheint. Dann können im Programm-Modus mit SST die eingegebenen Programmbefehle angesehen werden.

Nach dem BEEP muß unbedingt GTO .. ausgeführt werden! Deshalb sollte das Flag 26 nicht gelöscht werden. Die Stackregister und das Alpharegister sollten während der Eingabe nicht verändert werden.

Dieses Programm "LB" hat teilweise eine andere Funktionsweise als das vorher vorgestellte. In der Eingabeschleife (LBL 01) wird zwar genau wie im vorigen Programm Byte für Byte entsprechend der Eingabe im Alpharegister zwischengespeichert. Sobald jedoch 7 Bytes eingegeben worden sind, werden diese in einem Datenregister gespeichert (Zeile 30, 31 und 12). Wird das Programm gestartet, wird zunächst die Anzahl der definierten Datenregister um zwei herabgesetzt. Dies ist erforderlich, um sicherzustellen, daß in jedem Fall unterhalb des .END. mindestens zwei freie Register vorhanden sind. Dann werden alle Datenregister gelöscht und die relative Adresse des höchsten Datenregisters in Register 00 gespeichert (Zeile 12 in Verbindung mit Zeile 14). Die jeweils 7 eingegebenen Bytes werden in Datenregister mit absteigender Adresse bis einschließlich Register 01 gespeichert.

Anschließend, oder falls keine Eingaben mehr gemacht werden, wird nach LBL 02 verzweigt. Dort wird bis einschließlich Zeile 50 aus dem Inhalt des Statusregisters c die absolute Adresse des .END. errechnet. Der Wert wird um 1 vermindert (Zeile 51). Mit Hilfe der Textbytes aus Zeile 53 wird nun ein neuer Inhalt für das Statusregister c aufgebaut. Es wird festgelegt, daß es keine Datenregister gibt. Damit wird erreicht, daß die Datenregister, in die die Programmbefehle eingegeben wurden, vom Betriebssytem des Rechners als Programmregister erkannt werden. Da aber in den globalen Programm-Labeln in Verbindung mit den END-Befehlen ausgehend vom unteren Ende des Programmregisterbereichs (.END.) eine wichtige Informationskette gespeichert ist, muß noch erreicht werden, daß diese Kette funktionsfähig bleibt. Das wird in dem Programm "LB" dadurch erreicht, daß in Register c eine um eins zu niedrige Adresse für das .END. gespeichert wird. Bei dem durch GTO .. ausgelösten Packen des Programmspeichers wird diese gesamte Informationskette wieder neu aufgebaut, da das .END. tatsächlich nicht an der angegebenen Adresse gefunden wird. Wird nach Ablauf des Programms nicht sofort GTO .. ausgeführt, ist kein Zugriff mehr auf die gespeicherten Programme möglich; eventuell ist MEMORY LOST die Folge.

Die ersten drei Bytes der Textzeile (192, 0, 11) ergeben als Programmbefehl ein END.

Dieses END wird zuletzt an die eingegebenen Bytes angehängt, um die Eingaben als eigenständiges Programm im CAT 1 erscheinen zu lassen (Zeilen 60 und 61).

Das Programm "LB" belegt 113 Bytes. Es kann aber trotzdem auf einer Magnetkartenspur untergebracht werden, da das END nicht auf der Magnetkarte gespeichert werden muß.

Mit normalen Programmbefehlen ist es meist nicht möglich, dieselben Effekte zu erhalten, die man mit der synthetischen Programmierung erzielen kann. Dies haben die beiden Programme "LB" gezeigt. Es ist aber auch möglich, mit normalen Programmbefehlen ungewöhnliche Effekte zu erzielen. Mit den folgenden Programmbeispielen wird das aufgezeigt.

5.3 "--" - Laufschrift, normal

5.4 "--" - Laufschrift, synthetisch

```
01◆LBL "--"
02 SF 25
03 AVIEW
04 SF 99

05◆LBL 00
06 R-P
07 P-R
08 GTO 00
09 END

LBL'--
END
19 BYTES
```

```
01◆LBL "--"
02 RCL d
03 AVIEW
04 STO d
05 CLX

06◆LBL 00
07 R-P
08 P-R
09 GTO 00
10 END

LBL'--
END
20 BYTES
```

Die beiden Programme "--" haben die gleiche Wirkung: Sie dienen dazu, eine Laufschrift während des Programmablaufs in der Anzeige zu erzeugen. Darüber findet man im Benutzerhandbuch des HP-41 keine Information.

Um die Funktion von "--" zu erkennen, ist ein beliebiger Text in das Alpharegister einzugeben und das Programm zu starten. Dabei muß Flag 21 gelöscht sein, da sonst AVIEW die Programmausführung anhält. Das Phänomen der Laufschrift ist darauf zurückzuführen, daß durch AVIEW eine Meldung im Display angezeigt wird, aber das entsprechende Flag 50 künstlich gelöscht wird. Deshalb wird die aus dem Alpharegister stammende Display-Anzeige für das Programmausführungszeichen gehalten und bei jedem Labelsprung um ein Zeichen nach rechts in der Anzeige verschoben.

Durch den Befehl AVIEW wird Flag 50 zunächst gesetzt; dann aber in dem ersten Programm durch den fehlerhaften Befehl SF 99 gelöscht, um eine Fehlermeldung in der Anzeige vorzubereiten. Da vorher aber Flag 25 zur Unterdrückung der Fehleranzeige gesetzt wurde, kommt es nicht zu der Fehleranzeige, und das Phänomen der Laufschrift tritt auf. R-P und P-R wurden nur eingefügt, um den Labelsprung zu verlangsamen.

In der zweiten Programmversion wird der gleiche Effekt mit synthetischen Mitteln erzeugt. Im Statusregister d sind sämtliche Flagzustände gespeichert. Diese werden durch RCL d in das X-Register übertragen, solange Flag 50 noch gelöscht ist. Dann wird durch den Befehl STO d der ursprüngliche Zustand wieder hergestellt, also Flag 50 wieder gelöscht, obwohl sich eine Anzeige im Anzeige-Display befindet. Somit hält das Betriebssystem wieder den Inhalt des Displays für das Programmausführungszeichen, das bei jedem Labelsprung um ein Zeichen nach rechts verschoben wird.

Dieses kleine Anwendungsbeispiel hat gezeigt, daß zur Erreichung ungewöhnlicher Effekte nicht unbedingt Synthetik erforderlich ist. Es gibt jedoch viele Aufgaben, die nur mit Hilfe der Synthetik erledigt werden können. Zu diesen gehören z.B. die für die "LB"-Programme erforderlichen Curtainverschiebungen. Aber Curtainverschiebungen sind seit der Einführung des IL Development Moduls nicht mehr unabdingbare Voraussetzung der synthetischen Programmierung.

Die synthetische Programmierung ist in letzter Zeit durch das CCD Modul weiter entscheidend vereinfacht worden. Mit dem CCD Modul ist es möglich, synthetische Befehle ganz 'normal' auszuführen oder zu programmieren. Die Bedienung ist dabei so einfach und logisch, daß man sich kaum noch vorstellen kann, wie es vorher war. Das CCD Modul bietet für die Synthetik wirklich alles, was man sich nur wünschen kann. Es wurde von Mitgliedern des deutschen Anwenderclubs CCD (Computerclub Deutschland e.V.) programmiert. Die Bezugsquelle ist im Anhang angegeben.

6 BARCODES

Der Barcode-Leser ist ein vorzügliches Hilfsmittel um umfangreiche Programme schnell, bequem und sicher in den HP-41 zu übertragen. Außerdem kann hierbei das Papier als billiges Speichermedium genutzt werden. Als Nachteil muß man jedoch in Kauf nehmen, daß die Herstellung der Barcodes eigener Programme nur mit erheblichem Aufwand möglich ist.

Der Einstieg in die synthetische Programmierung wird durch den Barcode-Leser wesentlich vereinfacht, da alle grundlegenden Programme zur synthetischen Programmierung als Barcodes veröffentlicht wurden.

6.1 "BA" - Barcode Analyser

```
01◆LBL "BA"
02 WNDSCN
03  E3
04 /
05 ISG X

06◆LBL 00
07 " "
08 RCL d
09 FIX 0
10 ARCL Y
11 "⊢/"
12 ARCL IND
          Y
13 STO d
14 RDN
15 PROMPT
16 ISG X
17 GTO 00
18 CLST
19 END

LBL'BA
END
39 BYTES
```

Das Programm "BA" analysiert die einzelnen Bytes aller Barcode-Typen. Das Programm erfordert SIZE 017. Nach dem Programmstart erscheint bei angeschlossenem Barcode-Leser in der Anzeige 'W: READY'. Anschließend ist der zu analysierende Barcode einzulesen. In der Anzeige des Rechners erscheint jetzt 1/ und der Wert des ersten Bytes. Mit R/S ist fortzufahren, um die Werte der folgenden Bytes in die Anzeige zu bringen. Zu beachten ist dabei jedoch, daß die Funktion WNDSCN keine Prüfung des gelesenen Barcodes vornimmt, so daß Lesefehler nicht zu einer Fehlermeldung führen. Die Funktion WNDSCN bewirkt nicht, daß die gelesenen Informationen ausgeführt werden, statt dessen wird der Wert jedes gelesenen Bytes in ein Datenregister gespeichert. Ein Barcode-Analyseprogramm ist besonders nützlich, wenn man selbstgefertigte Barcodes prüfen will (Fehlerhafte Barcode-Befehle können unter Umständen sofort zu MEMORY LOST führen, ohne daß eine Fehlermeldung erfolgt).

Grundlage des Programms "BA" ist die Funktion WNDSCN, die eingelesene Bytewerte fortlaufend in Datenregister speichert. Außerdem legt WNDSCN die Anzahl der gelesenen Bytes im X-Register ab.

Mit diesen Informationen wird in den folgenden Programmzeilen die Ausgabe formatiert. Die Befehlsfolge RCL d ... STO d bewirkt, daß trotz des Ausgabeformates (FIX 0) der ursprüngliche Display-Modus erhalten bleibt. Hierzu kann ebenfalls RCLFLAG ... STOFLAG verwandt werden.

Der in dem Programm "BA" enthaltene Befehl E3 (= 1.000) hat die gleiche Funktion wie der üblicherweise verwendete Befehl 1 E3. Er gehört zu den sogenannten Kurzformexponenten, die in einem Programm die gleiche Funktion wie die entsprechenden normalen Befehle haben. Sie sind jedoch jeweils ein Byte kürzer und werden schneller ausgeführt. Es sind die einzigen synthetischen Befehle, die mit dem Barcode-Leser direkt als Einzelbefehle in ein Programm eingelesen werden können. Deshalb sind sie auf der folgenden Seite abgedruckt. Alle übrigen synthetischen Befehle sind nur als Bestandteil eines Programmes mit dem Barcode-Leser einzulesen. Dabei können natürlich gerade komplizierte synthetische Programme sicher und schnell eingelesen werden.

Trotzdem können mit dem Barcode-Leser noch einige Besonderheiten erreicht werden; exemplarisch dafür soll hier das Zuordnen einer Funktion oder eines Programmes mit dem Tastencode -31 (Taste SHIFT/SHIFT) gezeigt werden. Eine solche Zuordnung, die normalerweise nur mit synthetischen Hilfsprogrammen vorgenommen werden kann, ist problemlos durch Einlesen des entsprechenden Barcode-Befehls zu erreichen.

Dazu ist zunächst ein beliebiges Programm mit dem Global-Label "SW" zu bezeichnen. Nach Einlesen des Barcode-Befehls, der mit ASN "SW" -31 bezeichnet ist, ist der Programmstart dieses Programmes der Taste SHIFT/SHIFT zugeordnet. Nach zweimaligem Drücken der SHIFT-Taste startet das gewünschte Programm. Man hat mit diesem Barcode-Befehl also eine zusätzliche Tastenzuordnung erreicht, die mit normalen Mitteln nicht möglich ist. Ist kein entsprechendes Global-Label "SW" vorhanden, aber das Time Modul eingesetzt, wird die Stoppuhr-Funktion der Taste SHIFT/SHIFT zugeordnet.

Mit dem HP-41C und dem HP-41CV ist diese Zuordnung nicht ohne weiteres zu löschen. (CLKEYS würde sämtliche Tastenzuordnungen löschen.) Der HP-41CX bietet jedoch innerhalb des CAT 6 die Möglichkeit, jede einzelne Zuordnung zu löschen; dies ist auch für den Tastencode -31 möglich. Zur Löschung der Zuordnung kann der mit CLEAR ASN -31 bezeichnete Barcode-Befehl eingelesen werden.

Wer sich dafür interessiert, eigene Barcodes herzustellen, findet in den Veröffentlichungen der Anwenderclubs (PPC und CCD) entsprechende Hinweise. Die Anschriften dieser Anwenderclubs sind in den Literaturhinweisen im Anhang genannt.

E

E1

E2

E3

E4

E5

E6

E7

E8

E9

E-1

E-2

E-3

E-4

E-5

E-6

E-7

E-8

E-9

ASN "SW" -31

CLEAR ASN -31

7 PROGRAMMZEIGERMANIPULATIONEN

Um die folgenden Programme "DP" und "PD" verwenden zu können, müssen zunächst die Tastenzuordnungen RCL b (Dezimal 144, 124) und STO b (Dezimal 145, 124) vorgenommen werden. Dazu ist ein synthetisches Tastenzuordnungsprogramm, wie z.B. das in Kapitel 10 dargestellte Programm "KA", zu benutzen. Der Befehl STO b kann auch aus einem Programm mit SST ausgeführt werden und braucht dann nicht als Tastenzuordnung gespeichert zu sein.

7.1 "DP" - Dezimaladresse - Programmzeiger

7.2 "PD" - Programmzeiger - Dezimaladresse

```
01◆LBL "DP"
02 ENTER↑
03 CLA
04 7
05 MOD
06 X<>Y
07 LASTX
08 /
09 RCL X
10 16
11 ST* T
12 X↑2
13 MOD
14 X<>Y
15 LASTX
16 /
17 R↑
18 +
19 XTOA
20 X<>Y
21 XTOA
22 RCL [
23 END

LBL'DP
END
36 BYTES
```

```
01◆LBL "PD"
02 CLA
03 X<> [
04 POSA
05 AROT
06 X>0?
07 CLX
08 X≠0?
09 ATOX
10 ATOX
11 RCL X
12 16
13 MOD
14 LASTX
15 ST/ Z
16 X↑2
17 *
18 R↑
19 +
20 7
21 *
22 +
23 INT
24 END

LBL'PD
END
39 BYTES
```

Die Anwendungsmöglichkeiten für die Programme "DP" und "PD" sind sehr vielseitig: Man kann jede absolute Adresse berechnen, zu jeder absoluten Adresse springen, insbesondere auch zwischen Mehr-Byte-Befehle, und man kann Byte-Abstände berechnen. Es ist ferner möglich, einzelne synthetische Befehle herzustellen oder zu verändern. Außerdem kann man Tastenzuordnungen verändern und synthetische Zuordnungen herstellen. Hierbei ist aber eine sehr sorgfältige Ausführung erforderlich, da jedes Fehlverhalten schnell zu MEMORY LOST führen kann.

Das Programm "DP" rechnet eine absolute Byte-Adresse in den zugehörigen Wert für einen STO b-Befehl um. Dabei wird nicht nur eine absolute Registeradresse, sondern für jedes Byte eine eigene Adresse ermittelt. Das Programm "PD" errechnet aus dem Wert, den man aus einem RCL b-Befehl erhält, die absolute Byte-Adresse. Beide Programme sind für den gesamten Rechnerregisterbereich anwendbar, auch für Adressen im X-Functions Modul und im X-Memory Modul.

Im Register b wird in den Bytes 0 und 1 der Programmzeiger gespeichert. Dieser Zeiger gibt das Byte an, das der Rechner im PRGM-Modus zuletzt verarbeitet hat. Wenn man in den PRGM-Modus schaltet, sieht man das nächste Byte immer als Programmbefehl. Dies gilt auch dann, wenn der Programmzeiger so manipuliert wurde, daß er auf ein Datenregister, ein Stackregister oder ein Alpharegister gerichtet ist. Man kann daher durch Manipulation des Programmzeigers jeden Rechnerinhalt als Programm ausführen lassen, sogar im Extended Memory Bereich. Man kann sich auch jedes einzelne Byte nur ansehen und so den gesamten Rechnerinhalt lesen. Ferner ist es möglich, im PRGM-Modus einzelne Bytes zu verändern, auch wenn man sich nicht im Programmspeicherbereich befindet.

Will man an den Anfang eines Registers springen, so muß man die Registeradresse mit 7 multiplizieren, weil je Register 7 Bytes gespeichert werden und dann "DP" ausführen. Man erhält dann einen meist unlesbaren Wert im X-Register. Führt man dann den einer Taste zugeordneten Befehl STO b aus, springt der Programmzeiger sofort zu der berechneten Adresse. Schaltet man dann in den PRGM-Modus, so sieht man (eventuell nach Ausführung von SST) den Wert des berechneten Bytes als Programmbefehl. Dabei wird die Programmzeilennummer nicht berechnet. Sie weicht daher oft von der Zeilennummer im Programmlisting ab.

Durch Eingabe des Dezimalwertes einer absoluten Adresse, danach Ausführung von "DP" und STO b, positioniert man den Programmzeiger an jede gewünschte Stelle. Bei Ausführung des Befehls STO b darf der Programmzeiger nicht auf eine Programmzeile innerhalb eines Software-Moduls gerichtet sein.

Zur Berechnung einer absoluten Byte-Adresse innerhalb des Programmspeichers ist der Programmzeiger an die gewünschte Stelle zu bringen (im PRGM-Modus mit SST). Nach Verlassen des PRGM-Modus ist RCL b auszuführen. Aus dem erhaltenen Wert im X-Register errechnet "PD" die absolute Byte-Adresse.

Um den Byte-Abstand zweier Befehle im Programmspeicherbereich zu berechnen, sind nach dieser Methode beide Adressen zu berechnen und anschließend zu subtrahieren. Die gesamte Länge eines Programms kann mit dem Drucker durch CAT 1 (TRACE-Modus) oder mit dem HP-41CX durch CAT 1 ermittelt werden.

Wem diese Möglichkeiten nicht zur Verfügung stehen, der kann die gesamte Länge eines Programmes wie folgt berechnen: Differenz der Adressen des Programmanfangs und des im CAT 1 unmittelbar folgenden Programmanfangs. An den jeweiligen Programmanfang gelangt man durch Ausführen von CAT 1 und falls kein Global-Label in Zeile 01 des jeweiligen Programms steht, indem man dann noch GTO.000 ausführt.

Mit Hilfe der Tastenzuordnungen RCL b und STO b und den beiden Programmen "DP" und "PD" ist es möglich, einzelne synthetische Programmbefehle zu erzeugen. Als Beispiel soll hier erläutert werden, wie der Befehl RCL d (dezimal 144, 126) erzeugt werden kann.

An die gewünschte Stelle des Programms gibt man RCL 97 als Programmschritt ein. Danach ist der Programmspeicher zu packen. Man berechnet dann die absolute Byte-Adresse dieses Befehls RCL 97, zieht von dem errechneten Ergebnis 1 ab und führt "DP" und STO b aus. Nun (eventuell SST) sieht man im PRGM-Modus den Befehl ABS. Dies ist das zweite Byte von RCL 97 als Ein-Byte-Befehl gelesen. Diesen Befehl kann man nach der ganz normalen Methode in AVIEW ändern. Wenn man sich dann den Programmschritt nochmals ansieht (SST, BST), ist aus RCL 97 der Befehl RCL d geworden! So ist aus einem nichtsynthetischen Zwei-Byte-Befehl ein synthetischer Zwei-Byte-Befehl geworden. In Programmen, die im Extended Memory stehen, dürfen jedoch keine Veränderungen nach dieser Methode vorgenommen werden.

Nach einem ganz ähnlichen Verfahren ist es auch möglich, Tastenzuordnungen zu verändern. Die Tastenzuordnungen von Befehlen aus CAT 2 und CAT 3 werden in den Registern mit absoluter Adresse 192 und größer gespeichert. Alarmdaten des TIME Moduls werden dort ebenfalls gespeichert. In diesem Registerbereich muß man daher sicher sein, daß man die Tastenzuordnungen tatsächlich erkannt hat, bevor man Veränderungen vornimmt. Dann darf man nur genau so viele Bytes einfügen, wie man vorher gelöscht hat. Sonst erhält man sofort MEMORY LOST. Da man normalerweise nicht genau weiß, wie viele Zuordnungsregister belegt sind, und SST Adressen herabspringt (wie es dem Programmablauf entspricht), sollte man zum Sprung in die Zuordnungsregister am besten eine etwas zu hohe Registeradresse verwenden. BST ist sehr langsam und manchmal gar nicht möglich. Obwohl zwei Zuordnungen in einem Register gespeichert werden können, kann man einfach zu 192 die Anzahl der Zuordnungen addieren, das Ergebnis mit 7 multiplizieren und dies als absolute Adresse benutzen. Es müssen allerdings einige freie Register vorhanden sein, damit man nicht in den Bereich des letzten Programms gerät.

Mit der errechneten Adresse ist dann "DP" und anschließend STO b auszuführen. Im PRGM-Modus erreicht man nach dem ersten SST den Anfang des höchsten Zuordnungsregisters, sofern keine Alarmdaten gespeichert sind. Andernfalls muß man mit SST zunächst alle Alarmdaten durchlaufen, die jetzt natürlich als ungeordnete Programmbefehle erscheinen. Fährt man mit SST immer weiter fort, so gelangt man schließlich an den Anfang des ersten Programms des CAT 1.

In den Zuordnungsregistern werden auch die Tastencodes mitverschlüsselt. Die zugeordneten Funktionen sind aber zwischendurch am normalen Aussehen, wie sie in einem Programm erscheinen würden, erkennbar. Ein-Byte-Befehle werden durch LBL 03 und anschließend den Befehl verschlüsselt. Will man eine synthetische Zuordnung erzeugen, so muß man zunächst eine beliebige Zuordnung vornehmen und diese dann verändern.

Beispiel: Die Zuordnung RCL b soll erreicht werden. Man ordnet einer Taste die bisher keiner Taste zugeordnete Funktion BEEP zu. Dann springt man mit der nach der angegebenen Methode errechneten Adresse, XEQ "DP" und anschließendem STO b in die Zuordnungsregister und sucht im PRGM-Modus den Befehl BEEP mit SST. In der 'Programmzeile' davor müßte LBL 03 stehen. Dann wird BEEP gelöscht, es erscheint in der Anzeige LBL 03. Dieser Befehl ist auch noch zu löschen und RCL 97 einzugeben. Dann berechnet man die Adresse dieses Befehls RCL 97. Wie beschrieben ist dann das zweite Byte von RCL 97, das dann als ABS erscheint, in MEAN zu ändern. Dazu ist unbedingt zuerst ABS zu löschen und danach MEAN einzufügen. Die ursprüngliche Zuordnung BEEP ist so in RCL b geändert worden. Anschließend kehrt man mit GTO .. wieder in den Programmspeicher zurück.

7.3 "XF" - Programm im X-Functions Modul ausführen

```
01◆LBL "XF"
02 " "
03 ASTO b
04 END

LBL'XF
END
13 BYTES
```

Das nebenstehende Programm "XF" nutzt eine weitere Möglichkeit einer Programmzeigermanipulation. In Zeile 02 dieses Programms ist ein Zeichen mit dem Zeichencode 190 im Alpharegister enthalten. Dies kann aus dem Ausdruck nicht erkannt werden. (Statt Zeile 02 kann auch die Befehlsfolge 190, XTOA eingesetzt werden.) "XF" bewirkt eine Verzweigung des Programmzeigers auf das im Extended Memory als erstes File gespeicherte Programm.

Mit diesem kurzen Programm ist es möglich, das erste im Extended Memory gespeicherte Programm auszuführen, ohne daß es vorher in den normalen Programmspeicher übernommen werden muß.

Dabei ist jedoch folgendes zu beachten: Ein Unterprogrammaufruf eines Programms im normalen Programmspeicher und anschließender Rücksprung in das Extended Memory ist nicht möglich. In dem im Extended Memory auszuführenden Programm darf nicht der Befehl PSIZE verwendet werden. Wenn das Programm anschließend mit GETP wieder in den Hauptspeicher zurückgeladen werden soll, muß es vorher im Hauptspeicher ablaufen, damit alle Sprungbefehle innerhalb des Programms einmal ausgeführt worden sind. Andernfalls ist ein Zurückrufen des Programms in den Hauptspeicher nicht mehr möglich (es erscheint die Fehlermeldung CHKSUM ERR).

Bei Ausführung der Funktionen SIZE oder PSIZE werden alle Programme im Programmspeicher verschoben, sofern sich tatsächlich eine Änderung der Anzahl der Datenregister ergibt. In diesem Falle erhalten also alle Programmbefehle eine neue absolute Adresse. Wird PSIZE innerhalb eines Programms ausgeführt, so erhält deshalb auch der nachfolgende Programmbefehl eine andere absolute Adresse. Damit das Programm aber mit dem nachfolgenden Befehl fortgesetzt wird, ändert das Betriebssystem aufgrund der Ausführung von PSIZE die in Register b gespeicherte Programmzeigerinformation entsprechend. Tritt der Befehl PSIZE innerhalb eines im erweiterten Speicherbereich ausgeführten Programmes auf, und ändert sich dadurch die Anzahl der Datenregister tatsächlich, wird das Programm wegen der rechnerischen Veränderung des Programmzeigers nicht normal weiter ausgeführt. (Im Extended Memory enthaltene Register behalten immer die gleichen absoluten Adressen.)

Wird im erweiterten Speicherbereich des X-Functions Moduls ein Programm ausgeführt, das bei gelöschtem Alpharegister den Befehl PCLPS enthält, werden dadurch alle Programme im Programmspeicher gelöscht; das ausführende Programm im erweiterten Speicher wird jedoch nicht gelöscht. Der Befehl PCLPS, ausgeführt von einem Programm im erweiterten Speicher, jedoch mit der Angabe eines Programmnamens aus CAT 1 im Alpharegister, löscht Programme, als würde er von der Tastatur ausgeführt. Die Ausführung des Programms im erweiterten Speicher wird jedoch nicht fortgesetzt.

Mit Hilfe des Programms "XF" läßt sich folgendes Verhalten des X-Functions Moduls erkennen: Ist im erweiterten Speicherbereich nur ein Programm gespeichert, (EMDIR oder CAT 4 zeigen also nur dieses Programm an) und wird dieses Programm mit PURFL gelöscht, wird es nicht tatsächlich aus dem erweiterten Speicherbereich entfernt. Trotzdem wird DIR EMPTY angezeigt! Solange im erweiterten Speicher des X-Functions Moduls nichts abgelegt wird, kann man dieses scheinbar gelöschte Programm weiterhin mit "XF" ausführen.

Wie nützlich es sein kann, ein im erweiterten Speicherbereich des X-Functions Moduls abgelegtes Programm direkt auszuführen, kann an dem im Kapitel 11 dargestellten Finanzprogramm erkannt werden: Dieses Finanzprogramm kann z.B. als erstes Programm im erweiterten Speicher mit "XF" ausgeführt werden.

Die direkten Zugriffsmöglichkeiten auf die Label A bis J im USER-Modus bleiben erhalten. Der Programmzeiger bleibt innerhalb des Programms im erweiterten Speicherbereich, solange die Programmausführung nicht mit END angehalten wird.

Ordnet man das Programm "XF" derselben Taste zu, der das im erweiterten Speicherbereich abgelegte Programm zugeordnet war, so ist im USER-Modus kein Unterschied in der Bedienung festzustellen. Erforderliche Zahleneingaben sind in allen Stack-Registern möglich; lediglich das Alpharegister kann bei Aufruf mit "XF" nicht für Eingaben verwandt werden.

Das Prinzip von "XF" ist auch für andere Bereiche im X-Functions Modul oder X-Memory Modul anwendbar; dabei muß man jedoch darauf achten, daß das aufgerufene Programm vollständig in einem Modul enthalten ist.

8 DAS TIME MODUL

Eine besonders vielseitige und deshalb sehr beliebte Ergänzung des HP-41 ist das Time Modul. Es bietet Befehle aus den Bereichen Zeit- und Datumsfunktionen, Stoppuhrfunktionen, Kalenderrechnungen und Alarmfunktionen. Zu allen Anwendungsbereichen werden im folgenden Kapitel Programmbeispiele vorgestellt.

8.1 "TD" - Tondauer

Das Programm "TD" ist ein weiteres Anwendungsbeispiel für die im vorigen Kapitel behandelte Programmzeigermanipulation. Es führt entsprechend der Eingabe bei Programmbeginn einen Ton aus und mißt die Tondauer unter Verwendung der Funktion TIME des Time Moduls. Entscheidend dabei ist, daß mit Hilfe der synthetischen Programmierung insgesamt 128 nominell verschiedene Töne erzeugt werden können, von denen jedoch einige sowohl in Tonhöhe als auch Tondauer identisch sind. Während der Ausführung von "TD" sollte kein Drucker (oder IL Modul) angeschlossen sein, da sich sonst eine falsche Zeitrechnung ergibt. Um mit einem Programm die Aufgabe zu erfüllen, je nach Eingabe verschiedene Töne, also verschiedene Programmbefehle auszuführen, muß das Programm wieder ein Programm erzeugen. Das geschieht in "TD" dadurch, daß die Programmbefehle zunächst im Alpharegister gebildet werden und das Alpharegister später durch Programmzeigermanipulation als Programmteil ausgeführt wird.

```
01♦LBL "TD"
02 "        "

03 XTOA
04 "⊢
 ⍺JÖÆœäA⍺l"
05  E8
06 X<> b
07 " "
08 CLX
09  E4
10 *
11 RCL d
12 FIX 0
13 ARCL Z
14 "⊢___"
15 FIX 2
16 ARCL Y
17 STO d
18 X<> T
19 AVIEW
20 END

LBL'TD
END
59 BYTES
```

Aus diesem Grunde enthält das Programm auch in Zeile 02 und Zeile 04 unlesbare bzw. sinnlos erscheinende Textteile. Diese Programmzeilen, die nur durch synthetische Programmierung eingegeben werden können, enthalten folgende Dezimalwerte:

Zeile 02: 243, 166, 156, 159

Zeile 04: 253, 127, 166, 156, 113, 74, 23, 27, 28, 22, 65, 113, 145, 124

Wird das Programm "TD" mit der Eingabe 120 gestartet, erscheint, nachdem der zugehörige Ton ausgeführt wurde, 120 ___ 0,12 in der Anzeige. Diese Anzeige bedeutet, daß der Ton mit dem Dezimalwert 120 (im Programm als TONE P) 0,12 Sekunden dauert.

Führt man das Programm "TD" mit SST in Einzelschritten aus, so ist zwar das errechnete Zeitergebnis nicht mehr korrekt, aber man kann sämtliche im Alpharegister verschlüsselte Programmbefehle einzeln erkennen. Dabei wird dann auch der zweimal verwendete Befehl TIME erkennbar. Das Programm "TD" benutzt also zur Messung der Tondauer nicht die Stoppuhrfunktion des Time Moduls, sondern die Differenz aus der angegebenen Zeit vor und nach Ausführung des Tons. Dabei ist eine Korrektur von 0,07 Sekunden erforderlich, da die zweimalige Ausführung von TIME innerhalb eines Programmes diese Zeitdifferenz ergibt.

Wegen der Vielzahl der verwendeten synthetischen Befehle empfiehlt es sich, "TD" mittels Barcode einzulesen oder das Programm nach Eingabe durch "LB" auf Magnetkarte dauerhaft zu speichern. Das Programm "TD" sollte nur mit Eingaben im Bereich von 0 bis 127 gestartet werden. Daß die Eingabe des Wertes E8 in Register b den Programmzeiger in das Alpharegister bringt, läßt sich mit dem im vorigen Kapitel vorgestellten Programm "PD" nicht nachvollziehen. Wer jedoch das PPC Modul besitzt, kann mit dem darin enthaltenen gleichnamigen Programm errechnen, daß der Wert E8 ins Register b gebracht, als absolute Byte-Adresse 56 interpretiert wird.

Die aufwendige synthetische Programmierung mit dem Umweg über das Alpharegister ergibt sich im Programm "TD" nur deshalb, weil das zweite Byte des Befehls TONE durch die Eingabe vorgegeben werden soll. Dieses Byte wird dann durch den Befehl XTOA (Zeile 03) in das Programm eingefügt.

8.2 "BAT" - Batterienutzungsdauer

```
01◆LBL "BAT
       "
02 CLX
03 SETSW
04 RUNSW

05◆LBL 00
06 FC? 49
07 GTO 00
08 STOPSW
09 FIX 4
10 RCLSW
11 " "
12 ATIME
13 ASTO X
14 OFF
15 END
```

Das Programm "BAT" zeigt eine einfache Anwendungsmöglichkeit der Stoppuhrfunktionen des Time Moduls. Es dient dazu, die mit einem Akkusatz verfügbare Rechenzeit zu ermitteln. Man setzt dazu einen vollgeladenen Akkusatz in den Rechner ein und läßt das Programm "BAT", am besten über Nacht, laufen. Das Programm erfüllt dabei letztlich nur die Funktion, die Stoppuhr des Time Moduls anzuhalten, sobald durch Flag 49 eine abfallende Batteriespannung erkennbar wird. Die durch die Stoppuhr angezeigte Zeit wird dann über das Alpharegister formatiert, ins X-Register gebracht, und der Rechner wird abgeschaltet.

Nach Wiedereinschalten des Rechners sieht man in der Anzeige die verfügbare Rechenzeit für den entsprechend vollgeladenen Akkusatz in Stunden und Minuten. Durch die Befehle CLX, SETSW, RUNSW, ... , STOPSW ist es also möglich, Programmlaufzeiten zu messen. Die Stoppuhrfunktionen des Time Moduls sind daher ein wichtiges Hilfsmittel, um bei der Programmoptimierung die Schnelligkeit verschiedener Programmversionen objektiv beurteilen zu können.

8.3 "O" - Ostertermin

Ein exzellentes Beispiel für Programmoptimierung durch Benutzung der Kalenderfunktion des Time Moduls bietet das nebenstehende Programm "O". Autor dieses Programms ist Kari Pasanen, Jyväskylä, Finnland.

Das Programm berechnet ohne Benutzung von synthetischen Funktionen und ohne Benutzung von Datenregistern den Termin des Ostersonntags für jedes Jahr im Bereich 1900 bis 2099. Damit die Anzeigeformatierung (Zeile 28 bis 30) wie vorgesehen arbeitet, sollte mindestens FIX 3 (ENG 3, SCI 3) eingestellt sein.

Basierend auf diesem Programm lassen sich alle beweglichen Festtage errechnen. Die Berechnung des Ostertermins erfordert ohne Verwendung der Kalenderfunktionen des Time Moduls etwa den vierfachen Programmspeicherplatz; außerdem ist die Verwendung einiger Datenregister erforderlich.

```
01◆LBL "O"
02 ENTER↑
03 DMY
04  E6
05 /
06 22,03
07 +
08 X<>Y
09 19
10 MOD
11 LASTX
12 *
13 6
14 -
15 30
16 MOD
17 28
18 X<=Y?
19 DSE Y
20 RDN
21 DATE+
22 ENTER↑
23 DOW
24 CHS
25 7
26 MOD
27 DATE+
28 " "
29 ADATE
30 AVIEW
31 END
```

Das Time Modul bietet die Möglichkeit, Alarme zu speichern. Ein Alarm kann zur vorgegebenen Zeit entweder eine bestimmte Meldung auslösen, die Uhrzeit und das Datum anzeigen, eine Rechnerfunktion ausführen oder ein Programm starten. Je nach Wunsch kann dabei ein Wiederholungsintervall vorgegeben werden. Das Time Modul enthält aber nicht wie das X-Functions Modul einen zusätzlichen Speicherbereich. Die zur Speicherung der Alarmdaten benötigten Register werden innerhalb des Hauptspeicherbereichs in dem sogenannten I/O-Buffer belegt.

Der I/O-Buffer ist der Bereich zwischen dem .END. und dem Register mit der absoluten Adresse 192. In diesem Bereich werden auch die Tastenzuordnungen für Funktionen aus CAT 2 und CAT 3 gespeichert. Die insgesamt 319 Register des Hauptspeicherbereichs des HP-41CV stehen als Gesamtheit für Datenregister, Programme oder als I/O-Buffer zur Verfügung. Die jeweils noch freien Register werden im PRGM-Modus z.B. nach Ausführung von GTO .. angezeigt (00 REG NN). Diese freien Register werden sowohl bei Erhöhung der Anzahl der Datenregister, bei Eingabe weiterer Programmbefehle als auch bei der Speicherung neuer Tastenzuordnungen oder Alarme verbraucht. Es ist daher oft wünschenswert zu wissen, wie viele Register mit Tastenzuordnungen oder Alarmen belegt sind.

8.4 "A?" - Assignment- und Alarmregister Finder

Im PPC Modul ist ein Programm "A?" enthalten, das die Anzahl der für Tastenzuordnungen benutzten Register angibt. Dieses Programm darf jedoch nicht benutzt werden, wenn im Bereich des I/O-Buffers andere Informationen als Tastenzuordnungen gespeichert sind. Alarmdaten würden z.B. unbrauchbar aber nicht gelöscht; die dafür gebrauchten Register könnten mit herkömmlichen Methoden nicht mehr nutzbar gemacht werden (Ausnahme: MEMORY LOST).

```
01◆LBL "A?"
02 176
03 XROM "E?
          "
04 17
05 -
06 X<Y?
07 CLST
08 X=0?
09 GTO 02
10  E3
11 /
12 +
13 XROM "OM
          "

14◆LBL 00
15 RCL IND
          Y
16 STO [
17 CLX
18 16
19 ATOX
20 -
21 X≠0?
22 GTO 01
 23 "⊢"
24 CLX
25 RCL [
26 STO \
27 ARCL c
28 X<> \
29 STO IND
          Z
30 RDN
31 CLA
32 ISG Y
33 GTO 00
34 ENTER↑

35◆LBL 01
36 X<>Y
37 STO c
38 ATOX
39 R↑
40 176
41 -

42◆LBL 02
43 INT
44 +
45 END

LBL'A?
END
77 BYTES
```

Das Programm "A?" ermittelt, wie viele Register für Tastenzuordnungen und für Alarme benutzt sind. Natürlich werden dabei Alarmdaten und Tastenzuordnungen nicht verändert. Nach der Programmausführung enthält das X-Register die Anzahl der Register, die für Tastenzuordnungen und Alarme insgesamt belegt sind; die Anzahl der benutzten Tastenzuordnungsregister ist im LastX-Register enthalten.

Dieses Programm benutzt Funktionen aus dem X-Functions Modul und aus dem PPC Modul. Die Textzeile 23 enthält die Bytes: 242, 127, 240. Es werden keine Datenregister benutzt. Das Programm arbeitet einwandfrei, unabhängig davon, ob Alarme und/oder Tastenzuordnungen gespeichert sind oder nicht (also auch ohne das TIME Modul). "A?" darf nicht als erstes Programm im CAT 1 stehen; es muß mindestens ein END vorausgehen.

Um die folgende Programmbeschreibung nachvollziehen zu können, kann das Programm mit SST in Einzelschritten ausgeführt werden; die Ausführung darf jedoch nicht beendet werden, solange das Programm nicht bis zum Ende abgelaufen ist.

Mit Hilfe des Programms "E?" aus dem PPC Modul wird zunächst die absolute Adresse des .END. berechnet. Ist diese 192, so verzweigt das Programm nach LBL 02 und die Programmausführung ist beendet (Ergebnisanzeige = 0). Dieser Fall kann nur auftreten, wenn weder Tastenzuordnungen noch Alarme gespeichert sind und außerdem kein freies Register vorhanden ist.

Für die eigentlich zu untersuchenden Fälle wird bis zur Zeile 12 eine Steuerzahl aufgebaut, die später im Programm mehrfach Verwendung findet (RCL IND Y, STO IND Z und ISG Y). Diese Steuerzahl gibt den maximalen Bereich der im I/O-Buffer zu untersuchenden Register an. Die errechneten relativen Registeradressen sind dabei um 16 geringer als die absoluten Registeradressen, da das in Zeile 13 aufgerufene Programm "OM" den Curtain auf die absolute Registeradresse 16 versetzt. "OM" übergibt dabei den früheren Inhalt des Registers c im X-Register.

RCL IND Y in Zeile 15 bringt nun zuerst den Inhalt des Registers mit der absoluten Adresse 192 (176 + 16) ins X-Register. In den folgenden Programmzeilen wird geprüft, ob das erste Byte dieses Registers 16 ist; trifft dies nicht zu, wird nach LBL 01 verzweigt. Das erste Byte eines Tastenzuordnungsregisters hat immer den Wert 240. Damit entspricht ein Wert aus einem Tastenzuordnungsregister nicht dem Zahlen- oder Alphaformat, wie es durch den Befehl RCL verarbeitet werden kann. Wird ein Wert, der weder dem Zahlen- noch dem Alphaformat entspricht, durch RCL angesprochen, wird dieser zunächst durch das Betriebssytem des Rechners 'normalisiert'. Diese Normalisierung bewirkt, daß das erste Byte eines Tastenzuordnungsregisters von 240 in 16 geändert wird. Die Normalisierung verändert das Quellregister.

Würde man die Programmausführung von "A?" an diesem Punkt beenden, so wären alle Tastenzuordnungen unbrauchbar geworden, außerdem würde die Rechnertastatur nach einem PACK-Befehl blockieren, und der Rechner ließe sich nicht mehr einschalten. Ein solches Blockieren, genannt Crash, beschädigt den Rechner nicht. In der Regel ist es jedoch problematisch, den Rechner weiterzubenutzen, ohne vorher alle Inhalte mit MEMORY LOST zu löschen. Neuere Rechnermodelle lassen sich leicht durch gleichzeitiges Betätigen der CLX-Taste oder der ENTER-Taste zusammen mit der ON-Taste wieder einschalten; bei älteren Modellen ist eventuell die Entfernung der Batterie für einige Zeit erforderlich.

In den Programmzeilen 23 bis 28 wird die durch die Normalisierung hervorgerufene Veränderung wieder rückgängig gemacht. Problematisch dabei ist, daß sämtliche denkbare Konstellationen von Null-Bytes innerhalb des Tastenzuordnungsregisters beachtet werden müssen. ARCL c dient in Zeile 27 lediglich als besonders schnelle und bytesparende Möglichkeit, sechs Zeichen an das Alpharegister anzuhängen.

Zeile 29 bringt den restaurierten Inhalt des untersuchten Tastenzuordnungsregisters an seine ursprüngliche Adresse zurück. Aufgrund des ISG Y-Befehls in Zeile 32 wird die beschriebene Routine ab Zeile 14 für weitere Tastenzuordnungsregister durchgeführt. Die Befehle CLX und RDN dienen dazu, die Steuerzahl jeweils im Y-Register, und den ursprünglichen Inhalt von Register c im X-Register zu erhalten.

Wird nach ISG Y die folgende Zeile übersprungen, so ist das unmittelbar unterhalb des .END. gelegene Register bereits als Tastenzuordnungsregister untersucht worden; für diesen Einzelfall ist durch ENTER in Zeile 34 der ursprüngliche Inhalt von Register c ins Y-Register zu bringen.

Abgesehen von diesem Sonderfall wird Zeile 35 (LBL 01) durch den Sprungbefehl aus Zeile 22 erreicht. Dies ist dann der Fall, wenn das zuletzt aufgerufene Register kein Tastenzuordnungsregister ist. Es kann sich dabei um ein freies Register oder um das erste Alarmregister handeln.

Das erste Byte des untersten Alarmregistes hat den Wert 170, der durch Normalisierung zu 26 geändert wird. Hier bleibt die Normalisierung ohne Folgen und braucht daher nicht rückgängig gemacht zu werden.

Das zweite Byte des untersten Alarmregisters gibt die Anzahl der für Alarme benutzten Register an. Nach Restaurierung des Registers c liefert ATOX in Zeile 38 also die Anzahl der Alarmregister. In den folgenden Zeilen (39 bis 43) wird aus der Steuerzahl die Anzahl der benutzten Tastenzuordnungsregister ermittelt und schließlich die Anzahl der Alarmregister hinzuaddiert.

Die folgende Programmversion von "A?" benötigt keine Funktionen aus dem PPC Modul; diese Programmversion kann jedoch nicht mit SST in Einzelschritten ausgeführt werden.

```
01♦LBL "A?"
02 "*"
03 RCL c
04 X<> [
05 X<> \
06 ASHF
07 POSA
08 AROT
09 X>0?
10 CLX
11 X≠0?
12 ATOX
13 ATOX
14 16
15 MOD
16 LASTX
17 X↑2
18 *
19 +
20 193
21 -
22 X<0?
23 GTO 02
24  E3
25 /
26 "**iμ♦♦"
27 RCL c
28 ASTO c
29♦LBL 00
30 RCL IND
             Y
31 STO [
32 CLX
33 16
34 ATOX
35 -
36 X≠0?
37 GTO 01
 38 "⊢"
39 CLX
40 RCL [
41 STO \
42 ARCL c
43 X<> \
44 STO IND
               Z
45 RDN
46 CLA
47 ISG Y
48 GTO 00
49 ENTER↑

50♦LBL 01
51 X<>Y
52 STO c
53 ATOX
54 R↑

55♦LBL 02
56 X<0?
57 CLST
58 INT
59 +
60 END

LBL'A?
END
104 BYTES
```

In den Zeilen 02 bis 19 wird durch die gleiche Routine, die bereits in einem der Load-Bytes-Programme verwendet wurde, die absolute Adresse des Registers ermittelt, in dem das .END. gespeichert ist.

Zeile 26 enthält folgende synthetische Textzeile:

246, 42, 1, 105, 12, 0, 0.

Durch Verwendung dieser Textzeile in Verbindung mit ASTO c wird der Curtain bei dieser Programmversion auf die absolute Adresse 192 versetzt. Zeile 38 enthält wieder die Bytes 242, 127, 240. Da der Curtain nicht auf Register 16 versetzt wird, ergibt sich für diese Programmversion eine andere Steuerzahl. Im übrigen ist die Programmfunktion jedoch unverändert; auch hier werden keine Datenregister benutzt.

Aus der Information, wie viele Register für Tastenzuordnungen und Alarme belegt sind, läßt sich in vielfacher Weise Nutzen ziehen. Zunächst einmal kann man feststellen, ob in den Tastenzuordnungsregistern ungenutzte Lücken enthalten sind: Es sollten nicht mehr als halbsoviele Tastenzuordnungsregister belegt sein, wie Funktionen aus CAT 2 oder CAT 3 zugeordnet worden sind, da in jedem Tastenzuordnungsregister zwei Zuordnungen gespeichert werden können. Stellt man fest, daß einige ungenutzte Lücken in den Zuordnungsregistern vorhanden sind, kann man zusätzliche freie Register schaffen, indem man alle Tastenzuordnungen löscht, den Rechner kurz aus- und wieder einschaltet und anschließend die Tastenzuordnungen neu eingibt. Ungenutzte Lücken in den Tastenzuordnungsregistern können vermieden werden, indem man Zuordnungen nicht mit der Funktion ASN, sondern mit der Funktion PASN vornimmt. Dadurch läßt sich erreichen, daß die durch Löschen von Zuordnungen entstandenen Lücken zunächst wieder benutzt werden, bevor weitere Register für Tastenzuordnungen belegt werden. PACK oder Aus- und Anschalten des Rechners wandeln nur solche Zuordnungsregister in freie Register um, in denen keine Zuordnung gespeichert ist; Zuordnungsregister mit nur einer Zuordnung bleiben nach wie vor zur Hälfte ungenutzt.

Die durch Tastenzuordnungen belegten Register lassen sich eventuell vorübergehend anderweitig nutzen, indem man die Zuordnungen auf Statuskarten mit dem Magnetkartenleser aufzeichnet. Das Programm "A?" liefert hierzu die Angabe, wie viele Register maximal freigesetzt werden können. Unnötige Lücken in den Tastenzuordnungsregistern bleiben jedoch auch nach dem Aufzeichnen und Wiedereinlesen von Statuskarten unverändert erhalten.

Alarmdaten können normalerweise nur dadurch recht aufwendig in einem externen Speicher abgelegt werden, daß der gesamte Rechnerinhalt gespeichert wird. Dies geschieht entweder durch Speicherauszugskarten mit dem Kartenleser (WALL) oder durch Speichern des gesamten Rechnerinhaltes auf einer Digitalcassette (WRTA).

Die folgenden Programme erlauben mit Hilfe der synthetischen Programmierung, Alarmdaten entweder als Datenfile im erweiterten Speicher des X-Functions Moduls oder auf Magnetkarten zu speichern. Damit erhält man durch Löschen der Alarme zusätzliche freie Register, ohne daß man die Alarme später wieder aufwendig einzeln eingeben muß; darüber hinaus kann man wichtige Alarmdaten auf Magnetkarten unproblematisch speichern. Ein Verlust ist auch bei MEMORY LOST ausgeschlossen.

8.5 "AF" - Alarme - Datenfile

```
01◆LBL "AF"
02 176
03 XROM "OM
        "
04◆LBL 00
05 RCL IND
         Y
06 STO [
07 CLX
08 ATOX
09 16
10 -
11 X≠0?
12 GTO 01
 13 "⊢"
14 CLX
15 RCL [
16 STO \
17 ARCL c
18 X<> \
19 STO IND
         Z
20 RDN
21 CLA
22 ISG Y
23 ""
24 GTO 00
25◆LBL 01
26 X<>Y
27 STO c
28 ATOX
29 R↑
30 LASTX
31 +
32 X<>Y
33 "AF"
34 X≠0?
35 ISG X
36 GTO 02
37 CRFLD
38 DSE X
39  E3
40 /
41 GTO 03
42◆LBL 02
43 FLSIZE
44 SIZE?
45 +
46 PSIZE
47 X<> L
48 PSIZE
49 RDN
50◆LBL 03
51 X<>Y
52 XROM "CX
        "
53 X>0?
54 SAVERX
55 X=0?
56 GETR
57 R↑
58 STO c
59 CLST
60 END

LBL'AF
END
103 BYTES
```

Das Programm "AF" erfüllt gleichzeitig zwei Aufgaben: Es speichert die Alarmregister in ein Datenfile im erweiterten Speicher des X-Functions Moduls und überträgt diese Daten auch wieder als Alarme in den Rechner zurück. Dabei entscheidet das Programm selbständig, ob in ein Datenfile gespeichert oder von dort gelesen wird.

Sind im Rechner Alarme gespeichert, werden diese in ein Datenfile gespeichert (und bleiben unverändert im Rechner erhalten); sind keine Alarme vorhanden, so werden Alarmdaten vom Datenfile in den Rechner gelesen.

Das Programm "AF" darf nicht als erstes im CAT 1 stehen, es muß mindestens ein END vorausgehen. Die Programmausführung sollte nicht unterbrochen werden; das Programm muß bis zum Ende oder bis zu einem Fehlerstopp ablaufen, damit der Inhalt von Register c restauriert wird. Das X-Functions Modul und das PPC Modul werden benötigt.

Die Eingaben mit "LB" für Zeile 13 lauten: 241, 240.

Bevor das Programm "AF" gestartet werden kann, muß festgestellt werden, wie viele freie Register vorhanden sind. Das Programm darf nur dann benutzt werden, wenn mindestens ein freies Register angezeigt wird. Dazu ist RTN (oder GTO.000 oder GTO ..) auszuführen und kurz in den PRGM-Modus zu schalten.

Folgende Fehlermeldungen können auftreten:

DUP FL	(Zeile 37)	Datenfile "AF" und Alarme sind vorhanden
NO ROOM	(Zeile 37)	Im erweiterten Speicher gibt es nicht genug unbenutzte Register zur Aufnahme des Datenfiles
FL NOT FOUND	(Zeile 43)	Weder Datenfile "AF" noch Alarme vorhanden
NO ROOM	(Zeile 46)	Nicht genug freie Register zur Aufnahme der Alarme vorhanden

Hält das Programm wegen einer Fehlermeldung, muß es nach Beseitigung der Fehlerursache von Anfang an neu gestartet werden. Das Programm prüft nicht, ob der Datenfile "AF" tatsächlich als Alarmdaten verwendbare Informationen enthält. Es arbeitet nur dann wie gewünscht, wenn zu Beginn der Programmausführung Alarmdaten entweder im I/O-Buffer oder im Datenfile "AF" gespeichert sind; nach vollständiger Programmausführung sind die Alarmdaten immer zweimal im Rechner gespeichert: Im I/O-Buffer und im Datenfile "AF". Sie können dann also einmal gelöscht werden.

Das Programm "AF" benutzt zunächst bis einschließlich Zeile 32 im wesentlichen die gleiche Routine wie "A?", um die Anzahl der Tastenzuordnungsregister und Alarmregister zu ermitteln. In Zeile 33 enthält das X-Register die Anzahl der Alarmregister, das Y-Register enthält die absolute Adresse des ersten Alarmregisters. Hier wird nun entschieden, ob Alarmdaten in ein Datenfile gespeichert oder von dort gelesen werden sollen. Sind im Rechner Alarmdaten gespeichert, wird in Zeile 37 mit CRFLD ein Datenfile mit der Bezeichnung "AF" angelegt. Um Probleme mit der Normalisierung zu vermeiden, wird die Größe des Datenfiles mit einem Register mehr definiert, als Alarmregister vorhanden sind. Dieses zusätzliche Register darf aber nicht das .END. enthalten, weshalb das Programm "AF" auch nur dann ausgeführt werden darf, wenn mindestens ein freies Register vorhanden ist.

In den Programmzeilen 38 bis 40 wird die Steuerzahl für SAVERX erzeugt. Um diese Steuerzahl benutzen zu können, wird der Curtain durch die Routine "CX" aus dem PPC Modul auf die absolute Adresse des ersten Alarmregisters versetzt. Schließlich erhält Register c wieder seinen ursprünglichen Inhalt, wodurch auch der Curtain wieder zurückgesetzt wird.

Waren im Rechner keine Alarme gespeichert (Zeile 34: X = 0), verzweigt Zeile 36 zu LBL 02 in Zeile 42. Da das Programm in diesem Fall die Aufgabe hat, die Alarmdaten aus dem Datenfile in den Rechner zu übertragen, wird nun in Zeile 43 bis 48 geprüft, ob dazu genügend freie Register vorhanden sind. Dies geschieht, indem die Anzahl der Datenregister um die Anzahl der Fileregister erhöht wird (ergibt sich keine Fehlermeldung, dann ist genügend Platz vorhanden). Anschließend wird wieder die ursprüngliche Anzahl der Datenregister definiert. Der Befehl "CX" (Zeile 52) versetzt auch in diesem Fall den Curtain auf die absolute Adresse des ersten Alarmregisters (hier die absolute Adresse des ersten Registers oberhalb der Tastenzuordnungen). Da die Steuerzahl jetzt 0 ist, wird durch GETR (Zeile 73) der Inhalt aller Register des Datenfiles in die korrespondierenden Datenregister übertragen. Da zuvor der Curtain auf das erste Alarmregister versetzt wurde, wird dieses vom Rechner als Datenregister 00 interpretiert; auf diese Weise können alle Alarmregister durch GETR übertragen werden. Zum Programmende wird Register c restauriert.

Die zweite Programmversion von "AF" benutzt keine Funktionen des PPC Moduls:

```
01◆LBL "AF"
02 CLST
03 "*×iμ◆◆"
04 RCL c
05 ASTO c

06◆LBL 00
07 RCL IND
          Y
08 STO [
09 CLX
10 ATOX
11 16
12 -
13 X≠0?
14 GTO 01
 15 "⊢"
16 CLX
17 RCL [
18 STO \
19 ARCL c
20 X<> \
21 STO IND
          Z
22 RDN
23 CLA
24 ISG Y
25 ""
26 GTO 00

27◆LBL 01
28 X<>Y
29 STO c
30 ATOX
31 R↑
32 X<>Y
33 "AF"
34 X≠0?
35 ISG X
36 GTO 02
37 CRFLD
38 DSE X
39  E3
40 /
41 GTO 03

42◆LBL 02
43 FLSIZE
44 SIZE?
45 +
46 PSIZE
47 X<> L
48 PSIZE
49 RDN

50◆LBL 03
51 X<>Y
52 192
53 +
54 ENTER↑
55 "×i"
56 16
57 /
58 XTOA
59 X<> L
60 MOD
61 LASTX
62 *
63 XTOA
64 X<>Y
65 XTOA
66 RCL c
67 RDN
68 ASTO c
69 CLA
70 X>0?
71 SAVERX
72 X=0?
73 GETR
74 R↑
75 STO c
76 CLST
77 END

LBL'AF
END
135 BYTES
```

Es sind folgende synthetischen Textzeilen vorhanden:

Zeile 03: 246, 42, 1, 105, 12, 0, 0

Zeile 15: 241, 240

Zeile 55: 242, 1, 105.

In den Zeilen 54 bis 68 ist eine Routine enthalten, die den Curtain auf die im X-Register angegebene Adresse versetzt. Sie entspricht damit grundsätzlich dem Programm "CX" aus dem PPC Modul. Um die Routine möglichst kurz zu halten, wird jedoch die Adresse des .END. nicht codiert. Daher können Programme, in denen diese Routine verwandt wird, nicht mit SST in einzelnen Schritten ausgeführt werden.

In beiden Programmversionen von "AF" ist nach dem Befehl ISG Y (Zeile 22 bzw. Zeile 24) ein synthetischer Befehl mit dem Bytewert 240 enthalten. Dieser Programmbefehl kann ideal als NOP eingesetzt werden, d.h. seine Ausführung hat keine Wirkung. Außerdem ist die Ausführung dieses Befehls besonders schnell.

In Zeile 22 bzw. Zeile 24 wird durch den NOP erreicht, daß nach Erhöhung des Schleifenzählers im Y-Register immer nach LBL 00 verzweigt wird. Da "AF" nur ausgeführt werden darf, wenn mindestens ein freies Register vorhanden ist, erübrigt sich in diesem Programm die Begrenzung des Schleifenzählers durch die Adresse des .END. wie in "A?".

Sofern keine Tastenzuordnungen aus CAT 2 oder CAT 3 gespeichert sind, hat das erste Alarmregister die absolute Adresse 192. In diesem Fall ist es besonders einfach, die Alarmdaten zu übertragen.

Die letzte Programmversion von "AF" zeigt besonders deutlich, daß durch die synthetische Programmierung auch in sehr kurzen Programmen Erstaunliches erreicht werden kann. Bevor mit diesem Programm Alarme aus dem Datenfile in den Rechner übertragen werden, ist unbedingt zu prüfen, ob genügend freie Register vorhanden sind. Wird diese Prüfung vergessen, könnte das letzte im Programmspeicher vorhandene Programm einschließlich des .END. durch die Alarmdaten überschrieben werden. Dann ist kein Zugriff mehr auf die Programme möglich (CAT 1). Der Rechnerinhalt läßt sich jedoch größtenteils dadurch retten, daß man alle Alarme wieder löscht (im ALMCAT oder CAT 5) und anschließend durch GTO .. den Programmspeicher wieder zugänglich macht. Zumindest der Schluß des letzten Programms ist jedoch verloren.

Diese Kurzversion von "AF" löscht zunächst sämtliche Tastenzuordnungen und setzt dann den Curtain auf die absolute Adresse 192. Dazu wird in Zeile 03 folgende Textzeile verwandt: 245, 1, 105, 12, 0, 0. Da auch hier die Adresse des .END. nicht codiert wird, kann das Programm nicht mit SST ausgeführt werden. Bis zur Zeile 11 wird aus dem zweiten Byte des im absoluten Register 192 gespeicherten Wertes die Anzahl der Alarmregister ermittelt. Der weitere Programmablauf ist bis auf die entnommene Prüfroutine (LBL 02) prinzipiell wie in den beiden vorigen Programmversionen.

Bei der Programmierung von "AF" wurde Wert darauf gelegt, daß das Register c immer den ursprünglichen Inhalt hat, wenn das Programm wegen einer Fehlermeldung anhalten könnte. Dies gilt für alle Programmversionen von "AF".

Das anhand des Programms "AF" erläuterte Prinzip zur Übertragung von Alarmdaten kann auch angewandt werden, um die Daten der Alarmregister auf Magnetkarten zu speichern. Dadurch hat man die Möglichkeit, diese Daten auch im Falle von MEMORY LOST zu erhalten.

```
01◆LBL "AF"
02 CLKEYS
03 "×iμ◆◆"
04 X<> [
05 X<> c
06 RCL 00
07 X<> [
08 X<>Y
09 X<> c
10 ATOX
11 ATOX
12 "AF"
13 X≠0?
14 ISG X
15 GTO 00
16 CRFLD
17 DSE X
18  E3
19 /

20◆LBL 00
21 SEEKPTA
22 RCL Z
23 X<> c
24 X<>Y
25 X>0?
26 SAVERX
27 X=0?
28 GETR
29 X<>Y
30 STO c
31 CLST
32 END

LBL'AF
END
64 BYTES
```

8.6 "AC" - Alarme - Magnetkarte

Das Programm "AC" überträgt die Daten aus den Alarmregistern auf Magnetkarten und liest entsprechende Datenkarten wieder als Alarmdaten in den Rechner ein. Ähnlich wie in "AF" entscheidet auch das Programm "AC" selbständig, ob Magnetkarten beschrieben oder gelesen werden. Das X-Functions Modul und das PPC Modul sind erforderlich.

Auch dieses Programm darf nicht als erstes im CAT 1 stehen, es muß mindestens ein END vorausgehen. Das Programm muß bis zum Ende ablaufen, damit Register c restauriert wird. Erscheint in der Anzeige die Aufforderung zum Einlesen von Magnetkarten, so ist diese entweder zu beachten oder das Programm muß durch zweimal R/S gestartet werden. Es werden keine Datenregister benutzt. Das Programm enthält eine synthetische Textzeile:

Zeile 21: 242, 127, 240.

```
01◆LBL "AC"
02 176
03 XROM "E?
         "
04 17
05 -
06 X<Y?
07 GTO 02
08  E3
09 /
10 +
11 XROM "OM
         "
12◆LBL 00
13 RCL IND
          Y
14 STO [
15 CLX
16 ATOX
17 16
18 -
19 X≠0?
20 GTO 01
 21 "⊢"
22 CLX
23 RCL [
24 STO \
25 ARCL c
26 X<> \
27 STO IND
          Z
28 RDN
29 CLA
30 ISG Y
31 GTO 00
32 ENTER↑
33◆LBL 01
34 X<>Y
35 STO c
36 ATOX
37 R↑
38 LASTX
39 +
40 INT
41 X<>Y
42 X≠0?
43 GTO 03
44 CLX
45 5
46 XROM "CX
         "
47 CLX
48 TONE 8
49 RDTAX
50 R↑
51 STO c
52 CLX
53 ATOX
54 26
55 X≠Y?
56 ASIN
57 R↑
58 ATOX
59 SIZE?
60◆LBL 02
61 +
62 PSIZE
63 X<> L
64 PSIZE
65 CLX
66◆LBL 03
67 X≠0?
68 DSE X
69  E3
70 /
71 X<>Y
72 XROM "CX
         "
73 TONE 8
74 X>0?
75 WDTAX
76 X=0?
77 RDTA
78 R↑
79 STO c
80 CLST
81 END

LBL'AC
END
130 BYTES
```

Um Alarmdaten auf Magnetkarten zu speichern, ist das Programm "AC" zu starten. Um Alarmdaten von Magnetkarten in den Rechner zurückzuladen, sind zunächst alle Alarme zu löschen und dann ist das Programm "AC" zu starten. Nach einem Ton erscheint in der Anzeige die Aufforderung 'RDY 01 OF NN' oder 'CARD' zum Einlesen von Magnetkarten. Um Alarmdaten in den Rechner zurückzuladen, muß die erste Spur der entsprechenden Datenkarte zweimal eingelesen werden; die entsprechende Aufforderung erscheint daher zweimal. Nach dem Einlesen von Alarmen sollte man den Rechner ausschalten, um eventuell überfällige Alarme erkennen zu können.

Das Programm "AC" kann folgende Fehlermeldungen hervorrufen:

NO ROOM	(Zeile 62)	Es sind nicht genug freie Register vorhanden, um die Alarmdaten aufzunehmen.
DATA ERROR	(Zeile 56)	Die eingelesene Karte enthält keine Alarmdaten - das Programm ist neu zu starten.
NONEXISTENT	(Zeile 49, Zeile 75)	Das Programm wurde ohne Kartenleser gestartet - SF 25, R/S!

Der Aufbau und die Funktion bei der Übertragung der Alarmdaten auf Magnetkarten entspricht dem Prinzip des Programms "AF". Zum Prüfen der eingelesenen Datenkarte wurde eine besondere Routine eingefügt (Zeile 44 bis 58).

In Zeile 46 wird zunächst der Curtain auf das Register mit der absoluten Adresse 5 versetzt; dies ist der erste Teil des Alpharegisters, Register M. Durch RDTAX wird nun das erste Datenregister von der Datenkarte in Register M eingelesen (da X = 0). Handelte es sich dabei um ein erstes Alarmregister, müßte das erste Byte den Wert 26 (nach Normalisierung) haben. Ist dies nicht der Fall, wird in Zeile 56 eine Fehlermeldung ausgelöst.

Das nächste Byte des eingelesenen Registers gibt die Anzahl der erforderlichen Register für die gespeicherten Alarmdaten an; mit diesem Wert wird geprüft, ob genügend freie Register zur Aufnahme der Alarme vorhanden sind (Zeile 59 bis 64).

Das gleiche Programm kann natürlich auch ohne Verwendung des PPC Moduls realisiert werden:

```
01◆LBL "AC"
02  "*"
03 RCL c
04 X<> [
05 X<> \
06 ASHF
07 POSA
08 AROT
09 X>0?
10 CLX
11 X≠0?
12 ATOX
13 ATOX
14 16
15 MOD
16 LASTX
17 X↑2
18 *
19 +
20 193
21 X>Y?
22 GTO 02
23 -
24  E3
25 /
26 "*×iμ◆◆"
27 RCL c
28 ASTO c

29◆LBL 00
30 RCL IND
           Y
31 STO [
32 CLX
33 ATOX
34 16
35 -
36 X≠0?
37 GTO 01
 38 "⊢"
39 CLX
40 RCL [
41 STO \
42 ARCL c
43 X<> \
44 STO IND
           Z
45 RDN
46 CLA
47 ISG Y
48 GTO 00
49 ENTER↑
```

Diese Programmversion enthält folgende synthetische Textzeilen:

Zeile 26: 246, 42, 1, 105, 12, 0, 0

Zeile 38: 242, 127, 240

Zeile 59: 245, 1, 105, 0, 80, 0

Zeile 88: 242, 1, 105

```
50◆LBL 01
51 X<>Y
52 STO c
53 ATOX
54 R↑
55 INT
56 RCL Y
57 X≠0?
58 GTO 03
59 "×i◆P◆"
60 ASTO c
61 TONE 8
62 RDTAX
63 R↑
64 STO c
65 CLX
66 ATOX
67 26
68 X≠Y?
69 ASIN
70 R↑
71 ATOX
72 SIZE?

73◆LBL 02
74 +
75 PSIZE
76 X<> L
77 PSIZE
78 CLX
```

```
79◆LBL 03
80 X≠0?
81 DSE X
82  E3
83 /
84 X<>Y
85 192
86 +
87 ENTER↑
88 "×i"
89 16
90 /
91 XTOA
92 X<> L
93 MOD
94 LASTX
95 *
96 XTOA
97 X<>Y
98 XTOA
99 RCL c
100 RDN
101 ASTO c
102 TONE 8
103 X>0?
104 WDTAX
105 X=0?
106 RDTA
107 R↑
108 STO c
109 CLST
110 END

LBL'AC
END
189 BYTES
```

Programmbedienung und Fehlermeldungen sind wie bei der ersten Version von "AC". Auch dieses Programm muß durch zweimal R/S gestartet werden, wenn die Aufforderung zum Einlesen von Magnetkarten nicht beachtet werden soll. Da die Magnetkarten bei versetztem Curtain eingelesen werden müssen, ist es unbedingt erforderlich, daß das Programm anschließend weiterläuft, damit Register c restauriert wird.

Eine Kurzversion von "AC" mit nur 20 Programmzeilen (entsprechend der Kurzversion von "AF" auf Seite 53) ist auf dem Titelbild dieses Buches abgedruckt. Das PPC Modul wird für die Kurzversion nicht benötigt; die synthetische Textzeile (Zeile 03) entspricht der des Programms "AF" von Seite 53.

8.7 "A-" - Alarme unterdrücken

Im Zusammenhang mit den Alarmfunktionen des Time Moduls ergibt sich bei der synthetischen Programmierung das Problem, daß ein Alarm zu einer ungünstigen Zeit fällig werden kann. Dies kann insbesondere dann zur Löschung des gesamten Rechnerinhalts führen, wenn der Curtain versetzt ist und/oder das Register c vorübergehend einen solchen Inhalt hat, daß die Programmausführung nicht unterbrochen werden darf.

```
01◆LBL "A-"
02 SIGN
03 X<> c
04 ALMNOW
05 X<> c
06 X<> L
07 END

LBL"A-
END
18 BYTES
```

Dieses Problem läßt sich mit dem Programm "A-" beseitigen. Nach Ausführung dieses Programms werden keinerlei Alarme mehr aktiviert; dies gilt auch für eventuell vorher schon überfällige Alarme. Man ist also bei der Programmierung oder Ausführung synthetischer Programme vor überraschenden Unterbrechungen durch Alarme sicher. "A-" verändert nur das LastX-Register. "A-" darf keinesfalls mit SST ausgeführt oder unterbrochen werden!

Sobald der Rechner einmal ausgeschaltet wird, werden Alarme wieder normal aktiviert. Dabei ist zu beachten, daß auch die Ausführung der Funktion CLOCK in Verbindung mit dem Time Modul (nicht im HP41-CX) den Rechner kurzzeitig ausschaltet.

9 SORTIEREN VON DATENREGISTERN

Die Programmoptimierung läßt sich besonders anschaulich an Sortierprogrammen demonstrieren. Zunächst sollte man die Anforderung stellen, daß ein Sortierprogramm selbst keine Datenregister als Zwischenspeicher benutzt, damit durch Vorgabe einer Steuerzahl jeder denkbare Block von Datenregistern sortiert werden kann. Die Steuerzahl sollte in dem für einen ISG-Befehl üblichen Format sein.

Ausgehend von diesen Grundvoraussetzungen kann man nun versuchen, ein möglichst kurzes und schnelles Sortierprogramm zu erstellen.

9.1 "SØ" - Sortierprogramm

9.2 "SO" - Sortierprogramm

```
01◆LBL "SØ"
02 SIGN

03◆LBL 00
04 LASTX
05 STO a
06 RCL IND
          L

07◆LBL 01
08 RCL IND
          a
09 X<>Y
10 X>Y?
11 X<> IND
          a
12 ISG a
13 GTO 01
14 STO IND
          L
15 ISG L
16 GTO 00
17 CLST
18 TONE 8
19 END

LBL'SØ
END
36 BYTES
```

```
01◆LBL "SO"
02 SIGN

03◆LBL 00
04 LASTX
05 RCL IND
          X

06◆LBL 01
07 RCL IND
          Y
08 X<Y?
09 X<>Y
10 STO IND
          Z
11 RDN
12 ISG Y
13 GTO 01
14 STO IND
          L
15 ISG L
16 GTO 00
17 CLST
18 TONE 8
19 END

LBL'SO
END
35 BYTES
```

Die Kürze eines Programmes ist dabei einfach zu beurteilen; ein absoluter Maßstab für die Schnelligkeit ist schon schwieriger zu finden. Vergleicht man verschiedene Sortierprogramme im Hinblick auf ihre Schnelligkeit, so kann sich ergeben, daß ein Programm wenige Datenregister langsamer, aber eine große Anzahl von Datenregistern schneller als ein anderes sortiert.

"SØ" und "SO" sind zwei besonders kurze Sortierprogramme, die das Prinzip des Bubble-Sort benutzen. Beide Programme sortieren den vorgegebenen Block von Datenregistern in aufsteigender Reihenfolge. Wenn man den Vergleichsbefehl umkehrt, ergibt sich eine Sortierung in absteigender Reihenfolge. "SØ" benutzt Register a als Zwischenspeicher. In diesem Register werden Rücksprungadressen ab der dritten Rücksprungadresse gespeichert; daher kann dies Programm maximal als zweites Unterprogramm aufgerufen werden. Von dieser Einschränkung abgesehen kann Register a beliebig als Zwischenspeicher benutzt werden, es wird mit Ende des Programms durch RTN oder END immer gelöscht. Durch Verwendung dieses Zwischenspeichers kann die innere Schleife (Zeile 07 bis 13) kürzer programmiert werden als bei der zweiten Programmversion. Dadurch ergibt sich eine geringfügig schnellere Sortierung. Beide Sortierprogramme sind nur für Datenregister anwendbar, in denen Zahlen gespeichert sind.

9.3 "SX" - Sortierprogramm

Der HP41-CX bietet mit seinen neuen Vergleichsfunktionen nicht nur die Möglichkeit, kürzere und damit schnellere Sortierprogramme zu erstellen, sondern mit diesen neuen Vergleichsfunktionen können auch Alpha-Daten sortiert werden.

Das Programm "SX" sortiert mit erstaunlich geringem Aufwand numerische und Alpha-Daten in aufsteigender Reihenfolge. Auch hier kann für eine absteigende Sortierfolge der entgegengesetzte Vergleichsbefehl benutzt werden.

Um eine wesentlich kürzere Sortierzeit zu erreichen, ist erheblich größerer Programmieraufwand erforderlich. Das wohl schnellste Sortierprogramm in weiten Bereichen, "S2" aus dem PPC Modul, belegt 123 Bytes. Es benutzt sämtliche Teile des Alpharegisters als Zwischenspeicher, zusätzlich wird ein Flag benutzt.

```
01◆LBL "SX"
02◆LBL 00
03 RCL X
04 RCL IND
          X
05◆LBL 01
06 X>NN?
07 X<> IND
          Y
08 ISG Y
09 GTO 01
10 STO IND
          Z
11 X<> Z
12 ISG X
13 GTO 00
14 CLST
15 TONE 8
16 END

LBL'SX
END
34 BYTES
```

9.4 "SD" - Sortierdauer

```
01◆LBL "SD"
02 2
03 X<Y?
04 RDN
05 SIZE?
06 X<>Y
07 X>Y?
08 PSIZE
09  E
10 -
11  E3
12 /
13 STO [
14 VIEW X
15 STO ]
16 PI
17 SQRT
18 STO \

19◆LBL 00
20 RCL \
21 9821
22 *
23 ,211327
24 +
25 FRC
26 STO \
27  E3
28 *
29 INT
30 STO IND
             ]
31 ISG ]
32 GTO 00
33 STOPSW
34 CLX
35 SETSW
36 RCL [
37 CLD
38 RUNSW
39 XEQ "SO"
40 STOPSW
41 TONE 8
42 SW
43 END

LBL'SD
END
82 BYTES
```

Das Programm "SD" dient dazu, die Laufzeit verschiedener Sortierprogramme möglichst objektiv zu beurteilen. Da das Sortierprogramm in Zeile 39 aufgerufen wird, ist diese jeweils entsprechend des Programmnamens zu ändern. Nach Eingabe der Anzahl der zu sortierenden Datenregister ist "SD" zu starten.

Das Programm prüft zunächst, ob genügend Datenregister definiert sind; ggf. wird die Anzahl der Datenregister erhöht. Dann wird die für das spätere Sortierprogramm erforderliche Steuerzahl errechnet und angezeigt.

Während diese Steuerzahl in der Anzeige verbleibt, wird durch einen Standardzufallsgenerator (Zeilen 19 bis 26) für jedes der zu sortierenden Datenregister eine Zufallszahl erzeugt. Wegen der festen Vorgabe im Programm ergeben sich für die gleiche Anzahl von Datenregistern immer identische Zufallszahlen; dadurch sind die Sortiervoraussetzungen für jedes Programm bei gleicher Anzahl von Datenregistern immer identisch. Sobald die Steuerzahl aus der Anzeige gelöscht wird, ruft Zeile 39 das zu testende Sortierprogramm auf. Die Sortierdauer wird durch die Stoppuhrfunktion des Time Moduls gemessen.

Bei der Anwendung dieses Programms zur Beurteilung der Sortierdauer sollte kein Drucker (kein HP-IL Modul) angeschlossen werden, da dies eventuell eine erhöhte Laufzeit ergibt. Außerdem sollte man "SD" im CAT 1 unmittelbar hinter das zu testende Sortierprogramm positionieren, damit die Ausführungsdauer des Sprungbefehls (Zeile 39) das Ergebnis möglichst nicht beeinflußt.

10 DAS HP-IL DEVELOPMENT MODUL

Das HP-IL Development Modul (nicht zu verwechseln mit dem HP-IL Modul!) bietet neben der ausführlichen Analyse und Manipulationsmöglichkeit des IL-Systems phantastische Möglichkeiten für die synthetische Programmierung. So enthält das HP-IL Development Modul wesentlich erweiterte Alphafunktionen sowie Ein- und Ausgabefunktionen in den Zahlensystemen HEX, OCT, BIN. Damit sind Umrechnungsprogramme (fast) nicht mehr erforderlich. Wer zum Beispiel im Hexadezimalsystem (bis zur Größe von FFFF FFFF) rechnen muß, hat mit dem HP-IL Development Modul eine ideale Hilfe. Auch die Boolschen Funktionen AND, NOT, OR, XOR sowie Bitrotation und Bitabfrage sind vorhanden.

Die interessantesten Möglichkeiten für die synthetische Programmierung bietet das HP-IL Development Modul mit dem Buffer. Hierbei handelt es sich um einen Zwischenspeicherbereich, der registerweise und auch byteweise benutzt werden kann. Entscheidend dabei ist, daß die Daten beim Kopieren in den Buffer nicht normalisiert werden.

Jeder beliebige Registerinhalt des Rechners kann in den Buffer kopiert werden, ohne daß dadurch das Quellregister verändert wird. Es gibt also keine Normalisierungsprobleme mehr! Der Inhalt des Buffers kann byteweise ausgedruckt oder angezeigt werden. Er ist vergleichbar einem File im erweiterten Speicherbereich; es gibt auch hier einen Zeiger (Pointer), der das einzelne Byte bezeichnet. Im Gegensatz zum erweiterten Speicherbereich des X-Functions Moduls liegt aber der Speicherbereich des Buffers im Rechner selbst, im Bereich zwischen den Tastenzuordnungen oder Alarmen und den Programmen.

Der Buffer kann durch BSIZEX in der gewünschten Größe (in Bytes) angelegt werden. Die tatsächliche Größe ist jedoch immer durch 7 teilbar, da jedes Register 7 Bytes enthält. Für interne Kennzeichnungen werden, ähnlich wie im erweiterten Speicherbereich, zusätzlich 2 Register gebraucht. Der Buffer wird zwangsweise gelöscht, wenn der Rechner ohne eingesetztes IL Development Modul eingeschaltet wird.

Wie man von jedem beliebigen Register aus (Daten- Stack- und Alpharegister) mit Mitteln der synthetischen Programmierung auch Programm- Tastenzuordnungs- Alarm- und Statusregister (!) in den Buffer speichern kann, so ist auch der umgekehrte Vorgang möglich. Man kann also via Buffer synthetische Programmbefehle oder synthetische Tastenzuordnungen erzeugen. Hierfür erweist sich auch die direkte Eingabemöglichkeit im Hexadezimalsystem als sehr nützlich.

```
01◆LBL "PB"
02 CLST
03 "ABS.RG
        ?"
04 PROMPT
05 "BYTES ?
        "
06 PROMPT
07 BSIZEX
08 CLX
09 BSIZE?
10 7
11 /
12 CHS
13 ISG X
14 STO Z
15 +
16 STOP
17 ENTER↑
18 "×i"
19 16
20 /
21 X-AR
22 X<> L
23 MOD
24 LASTX
25 *
26 X-AR
27 CLX
28 X-AR
29 CLX
30 RCL c
31 RDN
32 ASTO c
33 CLA
34 AIPT

35◆LBL 00
36 RG-BUFX
37 ISG X
38 GTO 00
39 R↑
40 STO c
41 CLST
42 PT=
43 PRBYTES
44 BSIZEX
45 END

LBL'PB
END
88 BYTES
```

Den bedeutendsten Fortschritt kann man darin sehen, daß es mit Funktionen des HP-IL Development Moduls erstmals möglich ist, synthetische Befehle durch ein Programm zu erzeugen, das selbst keinen einzigen synthetischen Befehl enthält. Es ermöglicht also einen problemlosen Einstieg in die synthetische Programmierung.

10.1 "PB" - Print Bytes

Das Programm "PB" dient dazu, Bytes eines jeden beliebigen Registers anzuzeigen/auszudrucken. Das Programm darf nicht als erstes im CAT 1 stehen; es muß mindestens ein END vorausgehen. Zeile 38 darf nicht mit SST ausgeführt werden.

Man gibt auf die erste Aufforderung "ABS.RG ?" die absolute Adresse des gewünschten Registers ein. Dies wäre zum Beispiel für den HP41-CV/CX oder HP41-C mit 4 MM oder QM bei SIZE 100 für den Anfang des ersten Programmes aus dem CAT 1 die Adresse 411 (511 - 100); oder für das erste Tastenzuordnungsregister die Adresse 192. Die Adresse des ersten Files im erweiterten Speicherbereich ist z.B. 190. Man kann jede Adresse innerhalb des Programmspeichers oder im erweiterten Speicherbereich durch einen RCL b-Befehl und anschließende Decodierung (Programm "PD", Seite 35, dividiert durch 7) ermitteln. Auf die zweite Aufforderung "BYTES ?" gibt man die gewünschte Zahl ein, für Statusregister, Tastenzuordnungsregister, Alarm- oder Bufferregister sollte die Zahl 7 eingegeben werden, um jedes Register einzeln zu behandeln.

Werden mehrere Register in den Buffer gelesen, so sind die absoluten Adressen absteigend, wie es der Speicherung von Programmen entspricht. Wird für das obengenannte Programmbeispiel 21 Bytes eingegeben, so werden die absoluten Register 409, 410, 411 kopiert.

Da Tastenzuordnungen in den Registern ab 192 aufsteigend gespeichert werden, muß man, um mehrere Tastenzuordnungsregister zu kopieren, auf die erste Aufforderung "ABS.RG ?" die höchste Registeradresse eingeben. Sollen z.B. 4 Zuordnungsregister angezeigt werden, ist einzugeben: ABS.RG ? = 195, BYTES ? = 28.

Der Programmstopp in Zeile 16 ist zur Sicherheit vorgesehen. Hier hat man die Möglichkeit, die Eingabe zu kontrollieren, bevor mit R/S gestartet wird. Die Zeilen 17 bis 33 des Programms setzen den Curtain auf die absolute Adresse, die bei dem Stopp (Zeile 16) angezeigt wurde. Wird hier eine nicht existente Adresse angesprochen, erhält man MEMORY LOST! Register im erweiterten Speicherbereich können benutzt werden, sofern das entsprechende Modul vorhanden ist.

Aufgrund der Eingabe wird zunächst der Buffer in der gewünschten Größe angelegt (Zeile 07). Dann wird aus der tatsächlichen Größe des Buffers für die spätere Schleifensteuerung in Zeile 37 die Anzahl der zu kopierenden Register bestimmt. Bis zum Programmstopp in Zeile 16 wird die niedrigste absolute Adresse der zu kopierenden Register berechnet. Die Zeilen 17 bis 33 versetzen den Curtain dann auf diese absolute Adresse, so daß das erste zu kopierende Register als relatives Register 00 erscheint.

Zeile 18 enthält die synthetische Textzeile: 242, 1, 105.

Wer ein PPC Modul besitzt, kann die Zeilen 17 bis 33 durch den Befehl XROM "CX" ersetzen und dadurch 24 Bytes sparen. Anschließend werden die Register einzeln in den Buffer kopiert (Zeile 36); dann wird Register c restauriert, und der Inhalt des Buffers wird byteweise angezeigt oder ausgedruckt (Zeile 43). BSIZEX (Zeile 44) löscht den Buffer wieder, da das X-Register 0 enthält.

Mit "PB" läßt sich auf einfache Weise der gesamte Rechnerinhalt erforschen. Dabei können auch Bytes aus den Statusregistern angezeigt oder ausgedruckt werden; man muß jedoch beachten, daß das Programm "PB" selbst einige Statutsregister verändert. Das Alpharegister ist immer gelöscht, der ursprüngliche Inhalt von Register c befindet sich im Register T (ABS.RG ? = 0), wenn in den Buffer kopiert wird. Durch "PB" läßt sich auch nachvollziehen, daß das erste Byte des untersten Alarmregisters den Wert 170 hat, der durch Normalisierung mit dem Programm "A?" zu 26 wird. Auch die internen Kennzeichnungsregister im X-Functions Modul oder X-Memory Modul, ja sogar die internen Kennzeichnungen des Buffers lassen sich decodieren.

So schließt der Buffer nach oben hin immer mit einem Register ab, das die folgenden Werte hat: 16, 77, 79, 78, 73, 84, 82. Dies entspricht der Textzeile MONITR; hier ist der frühere Name des Moduls (HP-IL MONITOR ROM) erkennbar geblieben.

10.2 "E?" - .END. Finder

10.3 "C?" - Curtain Finder

Das HP-IL Development Modul enthält sehr komfortable und vielseitige Alphafunktionen. Damit ist es in bisher nicht gekannter Schnelligkeit möglich, die im Statusregister c enthaltenen Adressen des .END. und des Curtain zu entschlüsseln. Dies wird an den Programmen "E?" und "C?" erkennbar.

```
01◆LBL "E?"
02 RCL c
03 X<> [
04 A-XR
05 A-XR
06 16
07 MOD
08 LASTX
09 X↑2
10 *
11 +
12 END

LBL'E?
END
24 BYTES
```

```
01◆LBL "C?"
02 RCL c
03 X<> [
04 -2
05 A-XX
06 -3
07 A-XX
08 16
09 ST/ Z
10 *
11 +
12 INT
13 END

LBL'C?
END
28 BYTES
```

Die erweiterten Alphafunktionen erlauben, den Zeichencode jeder Stelle im Alpharegister zu ermitteln, ohne daß das entsprechende Zeichen aus dem Alpharegister gelöscht wird (A-XX). Die Position des Zeichens kann von links (positiv) oder auch von rechts (negativ) gezählt werden. Auch der umgekehrte Vorgang, jedes beliebige Zeichen im Alpharegister durch ein anderes zu ersetzten, ohne daß die übrigen Zeichen verändert werden, ist möglich (Y-AX). Dadurch sind alle Probleme der Null-Bytes im Alpharegister gelöst, die bei der Decodierung mit ATOX und AROT auftraten. Zeichen können an das Alpharegister sowohl rechts als auch links angehängt werden. Das Alpharegister kann außerdem von rechts oder von links gelesen werden.

Mit diesen neuen Funktionen können einige in den früheren Kapiteln dieses Buches enthaltene Programme, die die Funktionen ATOX und XTOA aus dem X-Functions Modul benutzen, erheblich verkürzt werden. Das gilt für alle Basisumrechnungsprogramme, insbesondere aber auch für das als Grundlage der Programme zur Übertragung von Alarmdaten dienende Programm "A?". Es gibt wirklich kaum etwas in der synthetischen Programmierung, das nicht durch das HP-IL Development Modul wesentlich kürzer und schneller erledigt werden kann (Diese vielseitigen Alphafunktionen sind auch im Extended I/O Modul enthalten).

10.4 "KA" - Key Assignment I

Die beiden Programme "KA" (Key Assignment I und II) dienen zur synthetischen Erzeugung von Tastenzuordnungen. Sie zeigen, wie dieses klassische Problem der Synthetik erstmals mit Programmen gelöst wird, die nur normale Programmbefehle enthalten.

Mit dem Programm "KA" ist es möglich, jeden Ein-Byte- oder Zwei-Byte-Befehl synthetisch einer beliebigen Taste zuzuordnen. Es können sowohl normale Befehle (z.B. STO IND 00) als auch synthetische Befehle (z.B. STO M) zugeordnet werden. Anschließend kann der Befehl dann entweder auf Tastendruck sofort ausgeführt werden oder bequem in ein Programm eingefügt werden.

Dazu sind die Dezimalwerte der einzelnen Bytes und der Tastencode einzugeben, bevor das Programm "KA" gestartet wird. Soll z.B. der Befehl RCL b der Taste LN zugeordnet werden, ist folgende Bedienung erforderlich: 144, ENTER 124 ENTER 15 XEQ "KA".

Synthetische Tastenzuordnungen standen am Anfang der Entdeckung der synthetischen Programmierung durch Mitglieder des PPC. Auch heute sind diese Tastenzuordnungen noch der wesentliche Schlüssel zur synthetischen Programmierung. So können die für das Load-Bytes-Programm erforderlichen synthetischen Zwei-Byte-Befehle zunächst durch ein Tastenzuordnungsprogramm erzeugt werden; "LB" eröffnet die Möglichkeit, sämtliche synthetischen Programmbefehle zu erzeugen.

Das Programm "KA" ordnet der eingegebenen Taste zunächst die Funktion OR aus dem IL-Development Modul zu. Damit diese Zuordnung wie vorgesehen durch den Befehl PASN vorgenommen werden kann, darf im CAT 1 kein Globallabel mit diesem Namen vorhanden sein. Um eine möglichst schnelle Programmausführung zu erreichen und Konflikte mit anderen Modulen zu vermeiden, die eventuell eine gleichnamige Funktion enthalten, sollte das IL-Development Modul in Port 1 eingesteckt werden.

```
01◆LBL "KA"
02 "OR"
03 PASN
04 CLA
05 SIGN
06 BSIZEX
07 CLX
08 SIZE?
09 703
10 +
11 166
12 X-AR
13 SQRT
14 X-AR
15 ASTO L

16◆LBL 00
17 RDN
18 ISG X
19 CLD
20 RG-BUFX
21 CLA
22 7
23 BUF-AX
24 X<> L
25 POSA
26 X<0?
27 GTO 00
28 R↑
29 X<>Y
30 Y-AX
31 R↑
32 X<>Y
33 ISG X
34 CLD
35 Y-AX
36 A-BUF
37 R↑
38 BUF-RGX
39 CLST
40 BSIZEX
41 END

LBL'KA
END
72 BYTES
```

Diese vorläufige Zuordnung ist aus zwei Gründen erforderlich. Zunächst wird in einem Statusregister gekennzeichnet, daß zu der gewünschten Taste eine Zuordnung besteht. Außerdem wird in den Zuordnungsregistern gespeichert, um welche Zuordnung es sich handelt. Anschließend wird ein Buffer für 7 Bytes angelegt. Aus der Zahl 703 und der Anzahl der definierten Datenregister wird für die spätere Verwendung mit RG-BUFX eine Steuerzahl errechnet.

Bei der Ausführung dieses Befehls prüft das Betriebssystem nicht, ob das angegebene Register tatsächlich als Datenregister existiert. Das gleiche gilt auch für den umgekehrten Befehl BUF-RGX. Die eigentliche Aufgabe dieser Befehle ist es, den Inhalt von Datenregistern in den Buffer zu kopieren und umgekehrt. Durch Vorgabe einer Steuerzahl aus Anzahl der definierten Datenregister + 512 + absolute Adresse können mit Hilfe von RG-BUFX und BUF-RGX nahezu alle Register des Rechners in den Buffer kopiert werden und umgekehrt. Diese Besonderheit wird in dem Programm "KA" für die Tastenzuordnungsregister ausgenutzt.

Der Befehl OR des IL-Development Moduls stellt sich in einem Programm mit den Dezimalbytes 166, 12 dar. Bei Zuordnung dieses Befehls werden diese beiden Bytes neben dem zugehörigen Tastencode auch in den Zuordnungsregistern gespeichert. Da das Programm "KA" die Zuordnungsregister später nach diesen beiden Werten durchsuchen soll, werden sie zunächst über das Alpharegister in das LastX-Register gespeichert (Zeilen 11 bis 15).

Die Programmzeilen 16 bis 27 stellen die Programmschleife dar, die solange durchlaufen wird, bis die Zuordnung der Funktion OR gefunden worden ist. Der Befehl ISG X erhöht dabei die Steuerzahl, also die angesprochene absolute Adresse, jeweils um eins. Bei der ersten Ausführung von RG-BUFX wird das Register mit der absoluten Adresse 192 in den Buffer kopiert. Anschließend werden die 7 Bytes des Buffers in das Alpharegister gebracht und durch POSA wird geprüft, ob die gesuchte Zuordnung in dem kopierten Zuordnungsregister enthalten war. Ist dies nicht der Fall, wird die Suche mit dem nächsten Zuordnungsregister fortgesetzt.

Wurde die Zuordnung der Funktion OR gefunden, so gibt POSA die Stelle an, an der das Byte mit dem Wert 166 im Alpharegister steht. Dies wird nun durch das erste Byte der gewünschten synthetischen Zuordnung ersetzt (Zeilen 28 bis 30). Anschließend wird das zweite Byte entsprechend der eingegebenen synthetischen Zuordnung ersetzt (Zeilen 31 bis 35). Danach wird der Inhalt des Alpharegisters in den Buffer kopiert und dieser in das ursprüngliche Tastenzuordnungsregister zurückgeladen. Anschließend wird der Buffer wieder gelöscht.

Durch diesen gesamten Vorgang wurde die ursprünglich vorgenommene Zuordnung der Funktion OR durch die synthetisch eingegebene Funktion ersetzt.

Da die Bufferfunktionen nicht normalisieren, kann die Ausführung des Programms gefahrlos abgebrochen werden; es wird dann lediglich die gewünschte synthetische Zuordnung nicht erreicht. Außerdem kann das Programm mit SST in Einzelschritten ausgeführt werden. Obwohl keine Datenregister benutzt werden, muß mindestens SIZE 001 definiert sein, damit der Befehl BUF-RGX wie beabsichtigt arbeitet. Die durch dieses Programm vorgenommenen Bytemanipulationen innerhalb der Zuordnungsregister können prinzipiell natürlich auch direkt von der Tastatur aus vorgenommen werden. Durch das am Anfang dieses Kapitels vorgestellte Programm "PB" lassen sich die erzielten Veränderungen kontrollieren.

10.5 "KA" - Key Assignment II

Dieses kürzere und vor allem wesentlich schnellere Tastenzuordnungsprogramm arbeitet nur dann einwandfrei, wenn in den Zuordnungsregistern keine ungenutzten Lücken enthalten sind. Das läßt sich am einfachsten dadurch erreichen, daß man nur PASN (also nicht ASN) benutzt und eine gelöschte Zuordnung sofort mit PASN durch eine neue Zuordnung ersetzt.

Falls dieses Programm "KA" verwendet wird, obwohl Lücken in den Zuordnungsregistern sind, wird die synthetische Zuordnung einer anderen Taste zugewiesen als beabsichtigt war. Auch für dieses Programm gilt, daß mindestens SIZE 001 definiert werden muß, obwohl kein Datenregister benutzt wird.

Auch dieses Programm weist der eingegebenen Taste zunächst die Funktion OR zu. Sofern Konflikte mit Globallabeln oder anderen Modulen entstehen, kann hier aber jeder beliebige Befehl aus CAT 2 oder CAT 3 eingesetzt werden, da die Zuordnungsregister später nicht nach diesem Befehl durchsucht werden. Zur Steigerung der Programmgeschwindigkeit kann ein Befehl aus dem Time Modul (z.B. SW) verwandt werden.

Im weiteren Programmverlauf wird das erste Tastenzuordnungsregister (absolute Adresse 192) in den Buffer kopiert und gelangt von dort in das Alpharegister.

```
01◆LBL "KA"
02  "OR"
03  PASN
04  SIGN
05  BSIZEX
06  CLX
07  SIZE?
08  704
09  +
10  RG-BUFX
11  CLA
12  7
13  BUF-AX
14  CLX
15  3
16  A-XX
17  ISG L
18  CLD
19  X≠0?
20  SIGN
21  X=0?
22  X<> L
23  R↑
24  X<>Y
25  Y-AX
26  ISG X
27  CLD
28  R↑
29  X<>Y
30  Y-AX
31  R↑
32  A-BUF
33  BUF-RGX
34  CLST
35  BSIZEX
36  END

LBL'KA
END
61 BYTES
```

Dann wird das 4. Zeichen (Adresse = 3) des Alpharegisters untersucht. Hierbei liegt folgende Überlegung zugrunde: Wird in einem Zuordnungsregister die erste Zuordnung vorgenommen, so ist das erste Byte 240, die nächsten drei Bytes sind 0 und die nächsten drei Bytes enthalten die Zuordnung und den Tastencode. Das Byte mit der Adresse 3 enthält den Tastencode der Zuordnung, die als zweite in einem Zuordnungsregister vorgenommen wird. Da durch das Programm "KA" eine synthetische Zuordnung nur im ersten Tastenzuordnungsregister vorgenommen wird, muß das Programm nur unterscheiden, ob es sich um die erste oder zweite Zuordnung in diesem Register handelt. Dies wird in den Zeilen 15 bis 22 erreicht.

Handelte es sich um die erste Zuordnung, ergibt Zeile 22 den Wert 4; handelte es sich um die zweite Zuordnung, ergibt Zeile 22 den Wert 1. Dies ist jeweils die Adresse innerhalb des Alpharegisters, an der das erste Byte synthetisch zu ändern ist. In den Zeilen 23 bis 25 wird diese Änderung vorgenommen; in den Zeilen 26 bis 30 wird das zweite Byte entsprechend der Eingabe geändert. Diese Manipulationen sind nur aufgrund der hervorragenden Alphafunktionen des IL-Development Moduls so problemlos möglich.

Anschließend wird das geänderte Alpharegister in den Buffer kopiert; der Inhalt des Buffers wird dann in das erste Tastenzuordnungsregister kopiert, und schließlich wird der Buffer wieder gelöscht.

Die grundlegende Entdeckung, daß ohne Curtain-Versetzung durch die Befehle RG-BUFX und BUF-RGX Register außerhalb der Datenregister angesprochen werden können, wurde von Michael Markov gemacht. Wie viele entscheidende Schritte auf dem Wege zur Verbesserung der synthetischen Programmierung, ist auch diese Entdeckung einem engagierten Mitglied des PPC zu verdanken.

Was dem HP-IL Development Modul eigentlich seinen Namen gegeben hat, ist seine vorzügliche Fähigkeit, als Hilfsmittel zur Analyse und Manipulation des IL-Systems zu dienen. Alle Befehle, die die Kommunikationsschleife des HP-IL-Systems durchlaufen, können protokolliert und - in Einzelschritte aufgeteilt - analysiert werden. Außerdem können in Verbindung mit dem IL-System die wichtigsten Einzelbefehle ausgeführt werden. Damit ist es z.B. möglich, beliebige Bytes von einer Digital-Cassette zu lesen oder beliebige Bytes auf einer Digital-Cassette zu verändern.

10.6 "NEWM?" - Maximale Einträge je Cassette

Das Programm "NEWM?" zeigt einen einfachen praktischen Anwendungsfall aus diesem Bereich. Bevor auf dem Cassettenlaufwerk eine neue Cassette beschrieben werden kann, muß durch NEWM bestimmt werden, wie viele Einträge maximal auf der Cassette vorgenommen werden sollen. Diese Größendefinition wird zwar auf der Cassette aufgezeichnet, läßt sich jedoch mit einfachen Mitteln nicht von der Cassette lesen. Man könnte lediglich so lange neue Einträge auf die Cassette schreiben, bis die Meldung DIR FULL erscheint, um dann die Anzahl der Einträge zu zählen.

Das Programm "NEWM?" liest die Anfangsdefinition von der Cassette, so daß man jederzeit feststellen kann, wie viele Einträge maximal möglich sind. Vor und nach Benutzung des Programms "NEWM?" sollte man die Cassette zurückspulen.

Das Programm bewirkt, daß der zu lesende Teil der Cassette in einen der Buffer des Cassettenlaufwerks gelesen und dann von dort in das Alpharegister des Rechners kopiert wird. Anschließend wird aus einem bestimmten Byte die Anzahl der möglichen Einträge auf der Cassette errechnet.

```
01*LBL "NEW
       M?"
02 AAU
03 2
04 LAD
05 4
06 DDL
07 CLX
08 OUTBINY
09 X<>Y
10 TAD
11 DDT
12 INA
13 19
14 A-XX
15 CLA
16 8
17 *
18 DSE X
19 END

LBL'NEWM?
END
39 BYTES
```

11 FINANZPROGRAMM

11.1 "F" - Universelles Finanzprogramm

Das Programm "F" entstand nach den Anforderungen der Kreditpraxis und ist daher besonders vielseitig verwendbar. Alle denkbaren Größen für Kredite und Annuitätsdarlehen mit festen oder variablen Konditionen können berechnet werden. Ebenso problemlos wird der Effektivzins für Ratenkredite mit Pro-Monat-Zins berechnet. Außerdem ist das Programm auch zur Sparkontenberechnung einsetzbar.

Durch die Zuordnung der Eingabe- und Ausgabefunktionen zu den Tasten A bis J der oberen zwei Reihen des Rechners und einer zusätzlichen Funktionsanzeige wird eine schnelle und sichere Bedienung erreicht. Für eine erneute Berechnung ist nur die Eingabe abweichender Werte erforderlich; dadurch ist eine schnelle Vergleichsrechnung für alternative Konditionsgestaltungen möglich.

Die Zuordnung der Tasten A bis E ist wie bei Finanzprogrammen von HP üblich (Finanzmodul). Die Finanzprogramme von HP sind auf die amerikanischen Verhältnisse zugeschnitten. Wegen der in Deutschland üblichen Unterscheidung zwischen Zinssollstellung, Tilgungsverrechnung und Ratenfälligkeit entsprechen die amerikanischen Programme jedoch nur in seltenen Fällen den Anforderungen der deutschen Kreditpraxis.

Bei der Berechnung ist zu unterscheiden zwischen nominellen und effektiven Größen. Die nominellen Größen bestimmen eindeutig den Kontoablauf über die gesamte Laufzeit. Im nominellen Teil kann für Annuitätsdarlehen und Zwischenkredite eine beliebige Größe von

Laufzeit / Nominalzins / Kapital / Rate / Restschuld

aus den übrigen Größen berechnet werden. Es müssen also vier dieser Größen eingegeben werden, um die fünfte zu errechnen. Dazu ist zusätzlich die Eingabe von

Zinssollstellung / Tilgungsverrechnung / Ratenfälligkeit

erforderlich. Eine Berechnung dieser drei Größen ist nicht möglich.

Die effektiven Größen bestimmen unabhängig von der Art der Kontoführung die tatsächlichen Kosten für ein Darlehen. Der zu berechnende Effektivzinssatz ist dabei nur abhängig von Laufzeit, Auszahlungsbetrag, Rate und Restschuld. Daran ist erkennbar, daß es für den Kreditnehmer ohne Bedeutung ist, wie das Kreditinstitut die Buchungen auf dem Kreditkonto vornimmt. Für ihn sind allein die Zahlungsströme und die Restschuld maßgeblich.

Im effektiven Teil kann für Annuitätsdarlehen und Zwischenkredite eine beliebige Größe von

Laufzeit / Auszahlungsbetrag / Rate / Restschuld / Effektivzinssatz

berechnet werden. Dazu ist zusätzlich die Ratenfälligkeit vorzugeben.

Für Ratenkredite mit Pro-Monat-Zinssatz (sogenannte Laufzeitzinsdarlehen) ist nur die Berechnung des Effektivzinssatzes möglich; dazu sind

Laufzeit / Zins / Gebühr

einzugeben.

Das Programm "F" kann nur im USER-Modus bedient werden. Den Tasten A bis J dürfen keine Funktionen oder Programme zugeordnet sein. Wurde das Programm mit den im Anhang enthaltenen Barcodes im USER-Modus eingelesen, ist der Programmstart durch zweimaliges Drücken der Taste SHIFT aufzurufen. Das Programm enthält zwar einige synthetische Programmbefehle, so auch die erste Zeile (LBL "F"), diese sind jedoch zur Programmfunktion nicht unbedingt erforderlich und können durch entsprechende normale Befehle ersetzt werden. Das Programm "F" benötigt keinerlei Erweiterungen des Rechners; es kann mit einem HP-41C mit einem Memory Modul benutzt werden. Es werden 21 Datenregister benutzt (SIZE 021). Zur Protokollierung der Ein- und Ausgabeergebnisse kann ein Drucker angeschlossen werden.

PROGRAMMBEDIENUNG:

Wird eine Zahl eingegeben und anschließend die zugeordnete Taste (A bis J) gedrückt, so wird die eingegebene Zahl gespeichert und mit der zugehörigen Abkürzung zur Kontrolle angezeigt.

Die Eingabereihenfolge ist grundsätzlich beliebig, jede Zahl kann beliebig oft korrigiert werden.

Soll ein Wert abgefragt werden, so ist die Taste RCL und anschließend die zugeordnete Taste (A bis J) zu drücken. Zinssollstellung, Tilgungsverrechnung und Ratenfälligkeit können durch die zugeordneten Tasten (G bis I) ohne RCL abgefragt werden.

Soll eine Größe berechnet werden, so ist die zugeordnete Taste (A bis F oder J) ohne Eingabe zu drücken. Das Programm beginnt sofort mit der Berechnung, es müssen daher alle erforderlichen Größen vorher eingegeben worden sein.

```
01◆LBL "F"
02 CLRG
03  E2
04 STO 03
05 SIGN
06 STO 07
07 STO 08
08 STO 09
09 CLST

10◆LBL 01
11 ADV
12 STOP
13 GTO 01

14◆LBL 02
15 FC? 19
16 CF 18
17 FC? 20
18 CF 18

19◆LBL 03
20 FIX 2
21 STO 00
22 RDN
23 "⊢ "
24 FC?C 22
25 XEQ 04
26 STO IND
      00
27 RND
28 ARCL X
29 LASTX
30 FIX 6
31 AVIEW
32 STOP
33 GTO 01

34◆LBL A
35  E
36 "LFZ"
37 GTO 02

38◆LBL B
39 2
40 "ZINS"
41 GTO 02

42◆LBL C
43 3
44 "KAP"
45 GTO 02

46◆LBL D
47 4
48 "RATE"
49 GTO 02

50◆LBL E
51 5
52 "REST"
53 GTO 02

54◆LBL F
55 6
56 X<>Y
57 "GEB%"
58 20
59 FS? 22
60 X<Y?
61 "AUSZ"
62 RDN
63 X<>Y
64 SF 18
65 FS? 22
66 SF 19
67 GTO 03

68◆LBL G
69 7
70 "ZSS"
71 GTO 00

72◆LBL H
73 8
74 "TVR"
75 GTO 00

76◆LBL I
77 9
78 "RFL"

79◆LBL 00
80 FS? 22
81 GTO 03
82 RCL IND
      X
83 X<>Y
84 SF 22
85 GTO 03

86◆LBL J
87  E1
88 "EFF"
89 SF 18
90 FS? 22
91 SF 20
92 GTO 03

93◆LBL 04
94 RCL d
95 CF 21
96 AVIEW
97 STO d
98 RCL IND
      00
99 X<=0?
100 17
101 STO 16
102 STO IND
      00
103 RCL 00
104  E1
105 X≠Y?
106 GTO 00
107 2
108 RCL 02
109 X>Y?
110 GTO 00
111 RCL 01
112 *
113 RCL 06
114 +
115  E2
116 STO 06
117 +
118 RCL 01
119 /
120 STO 04
121 12
122 STO 09
123 CLX
124 STO 05

125◆LBL 00
126 RCL 08
127 RCL 09
128 X<Y?
129 STO 08
130 XEQ 06
131 STO 20
132 3
133 ST- IND
      00

134◆LBL 05
135 XEQ 06
136 STO 12
137 RCL 16
138 *
139 RCL 20
140 RCL IND
      00
141 STO 16
142 *
143 -
144 RCL 12
145 ENTER↑
146 X<> 20
147 -
148 /
```

```
149 STO IND
        00
150 RCL 20
151 RND
152 X≠0?
153 GTO 05
154 CF 18
155 RDN
156 "⊢="
157 TONE 8
158 RTN

159◆LBL 06
160 RCL 04
161 FS? 18
162 GTO 07
163 RCL 09
164 *
165 RCL 08
166 STO 12
167 /
168 STO 19
169 RCL 03
170 STO 17
171 RCL 01
172 RCL 08
173 *
174 RCL 09
175 /
176 STO 14
177 RCL 07
178 STO 15
179 RCL 08
180 X>Y?
181 GTO 00
182 STO 15
183  E
184 RCL 02
185 RCL 07
186 /
187 %
188 +
189 RCL 07
190 RCL 08
191 /
192 Y↑X
193  E
194 -
195 RCL 08
196 *
197  E2
198 *
199 GTO 08

200◆LBL 00
201 RCL 02
202 GTO 08

203◆LBL 07
204 CF 19
205 CF 20
206 STO 19
207 RCL 06
208 STO 17
209 RCL 01
210 STO 14
211 RCL 09
212 STO 12
213  E
214 STO 15
215 RCL 10

216◆LBL 08
217 STO 18
218 RCL 14
219 RCL 12
220 RCL 15
221 /
222 STO 11
223 /
224 INT
225 STO 13
226  E
227 RCL 18
228 RCL 15
229 /
230 %
231 STO 12
232 +
233 RCL 13
234 Y↑X
235 STO 13
236 RCL 14
237 RCL 11
238 LASTX
239 *
240 -
241 STO Y
242 RCL 11
243 /
244 RCL 12
245 *
246  E
247 +
248 STO 14
249 X<>Y
250 RCL X
251  E
252 -
253 RCL 12
254 *
255 RCL 11

256◆LBL 00
257 /
258 2
259 /
260  E
261 +
262 *
263 RCL 19
264 *
265 RCL 11
266  E
267 -
268 2
269 /
270 RCL 12
271 *
272 RCL 11
273 +
274 RCL 19
275 *
276 RCL 13
277  E
278 -
279 RCL 12
280 /
281 *
282 RCL 14
283 *
284 RCL 05
285 +
286 +
287 STO 12
288 LASTX
289  E
290 RCL 18
291 %
292 +
293 RCL 17
294 RCL 14
295 *
296 RCL 13
297 *
298 RCL 12
299 -
300 RCL 13
301 /
302 END

LBL'F
END
448 BYTES
```

Taste	Bezeichnung	Erklärung
A	LFZ	Laufzeit in Perioden entsprechend der Ratenfälligkeit (Taste I)
B	ZINS	Nominalzinssatz
C	KAP	Kapitalbetrag
D	RATE	Rate je Fälligkeitstermin
E	REST	Restbetrag
F	AUSZ	Auszahlungsbetrag
	GEB%	Gebührensatz in % für Laufzeitzinsdarlehen
G	ZSS	Zinssollstellungen je Jahr
H	TVR	Tilgungsverrechnungen je Jahr
I	RFL	Ratenfälligkeiten je Jahr
J	EFF	Effektivzins p.a.

Zur sicheren Bedienung wird empfohlen, eine Tastaturschablone anzufertigen.

Berechnungen im Nominalteil werden durch die folgenden Größen beeinflußt:

- LFZ (Taste A)
- ZINS (Taste B)
- KAP (Taste C)
- RATE (Taste D)
- REST (Taste E)
- ZSS (Taste G)
- TVR (Taste H)
- RFL (Taste I)

Berechnungen im Effektivteil werden durch die folgenden Größen beeinflußt:

- LFZ (Taste A)
- RATE (Taste D)
- REST (Taste E)
- AUSZ (Taste F)
- GEB% (Taste F)
- RFL (Taste I)
- EFF (Taste J)

BERECHNUNGSBEISPIEL

Annuitätsdarlehen ohne Restschuld;
Berechnung des Effektivzinssatzes

Konditionen:	Zinssatz	9,75 %,
	Auszahlung	100 %
	Tilgung:	1 % p.a.
	Ratenzahlung:	monatlich
	Zinssollstellung:	vierteljährlich
	Tilgungsverrechnung:	jährlich

EINGABEN:

Zins	(TASTE B)	9,75	
Kapital	(TASTE C)	100	
Rate	(TASTE D)	10,75 ENTER 12 /	
Restschuld	(TASTE E)	0	
Auszahlungsbetrag	(TASTE F)	100	
Zinssollstellung	(TASTE G)	4	
Tilgungsverrechnung	(TASTE H)	1	
Ratenfälligkeit	(TASTE I)	12	
Laufzeit	(TASTE A)	-	Ergebnis = 351,90
Effektivzins	(TASTE J)	-	Ergebnis = 10,71

Zunächst muß die Laufzeit berechnet werden, da diese nicht exakt bekannt ist, aber zur Berechnung des Effektivzinssatzes bekannt sein muß. Das angezeigte Ergebnis ist die Kreditlaufzeit in Monaten, da als Ratenfälligkeit 12 (monatlich) eingegeben wurde. Anschließend ist ohne Eingabe die Taste J zu drücken, damit der Effektivzins berechnet wird. Wird nach Ablauf des Programms mit R/S gestartet, so erscheint das berechnete Ergebnis mit sechs Nachkommastellen. Durch Umschalten in den Alpha-Modus kann jederzeit festgestellt werden, welche Größe zuletzt berechnet oder eingegeben wurde. Durch das Gleichheitszeichen in der Anzeige ist ein Berechnungsergebnis erkennbar.

Während des Berechnungsvorganges wird jeweils die Bezeichnung des Wertes, der berechnet wird, als Laufschrift angezeigt. Dadurch ist eine Fehlbedienung sofort zu erkennen; das Programm kann durch R/S angehalten werden.

Solange das Programm nicht neu gestartet wird, bleiben eingegebene und berechnete Werte gespeichert. Nur beim Programmstart vom Anfang des Programms werden Zinssollstellung, Tilgungsverrechnung und Ratenfälligkeit gleich 1, das Kapital gleich 100 gesetzt.

Das Berechnungsbeispiel kann daher für eine alternative Konditionsgestaltung sofort weitergerechnet werden. Zu berechnen sei der Effektivzinssatz bei vierteljährlicher Tilgungsverrechnung. Dazu ist über die Taste H als Tilgungsverrechnung 4 einzugeben; anschließend ohne Eingabe Taste A (veränderte Laufzeit !) und nach der Laufzeitberechnung ohne Eingabe die Taste J betätigen. Das Ergebnis ist 10,24 (10,235907).

Zu beachten ist, daß über die Taste F der Auszahlungsbetrag (also nicht der Auszahlungskurs) einzugeben ist. Wegen der vereinfachten Berechnung der Rate und des Auszahlungsbetrages wird empfohlen, grundsätzlich mit einem Kreditbetrag von DM 100,-- zu rechnen.

Die Rechenzeit kann durch Eingabe eines Schätzwertes entscheidend verkürzt werden. Grundlage der Berechnung ist zunächst die gespeicherte Zahl der zu berechnenden Größe. Je besser dieser Wert dem Ergebnis bereits angenähert ist, desto kürzer ist die Rechenzeit. Stoppt das Programm mit 'DATA ERROR' oder mit 'OUT OF RANGE', so ist eine Berechnung deshalb nicht möglich, weil die vorher gespeicherte Zahl extrem von dem zu berechnenden Ergebnis abweicht, oder es liegt ein Eingabefehler vor. Kann das Ergebnis nicht geschätzt werden, so ist in jedem Falle eine Berechnung möglich, wenn zunächst 0 für die zu berechnende Größe eingegeben wird. Dies bedingt jedoch immer eine Verlängerung der Rechenzeit.

Analog zu dem Berechnungsbeispiel können beliebige Größen für Darlehen, Zwischenkredite und Kontokorrentkredite berechnet werden. Für Zwischenkredite in Darlehensform gilt, daß im Nominalteil die Laufzeit aber nicht die genaue Restschuld bekannt ist.

Bei Berechnung von Agio statt Disagio ist der Auszahlungsbetrag entsprechend zu erhöhen. Auch zu Angeboten des Versandhandels kann der Effektivzinssatz berechnet werden. Wird z.B. für einen Betrag von DM 1.000,-- Ware gekauft, die erst nach sechs Monaten (einschließlich einer "Gebühr" von DM 80,--) zu bezahlen ist, so ist zu rechnen:

Laufzeit	(TASTE A)	6	
Rate	(TASTE D)	0	
Rest	(TASTE E)	1080	
Auszahlung	(TASTE F)	1000	
Ratenfälligkeit	(TASTE I)	12	
Effektivzins	(TASTE J)	-	Ergebnis = 16,00

An dem Beispiel läßt sich erkennen, daß zur Berechnung sowohl Nominalzins als auch Kapital und Zinssollstellung nicht erforderlich sind. Ist die "Gebühr" sofort zu zahlen, ergibt sich (Ausz = 920, Rest = 1000) ein Effektivzins von 17,39 %. Obwohl für die Berechnung ein (über die Taste B eingegebener) Nominalzinssatz eigentlich ohne Bedeutung ist, muß doch ein Nominalzins größer als 2 gespeichert sein. Bei der Berechnung des Effektivzinssatzes erkennt das Programm aufgrund des Nominalzinssatzes, ob es sich um die Berechnung für ein Laufzeitzinsdarlehen handelt. Dies sind Ratenkredite mit pro-Monat Zinssätzen. Diese Verzinsungsart gilt oft für Konsumentenkredite.

Beispiel:	Laufzeit	40 Monate
	Zinssatz	0,6 % pro Monat
	Gebühr	2 %

Eingaben:

Laufzeit	(TASTE A)	40	
Zins	(TASTE B)	.6	
Gebühr	(TASTE F)	2	
Effektivzins	(TASTE J)	-	Ergebnis = 15,20

An diesem Zahlenbeispiel kann die Rundungsgenauigkeit ähnlicher Programme geprüft werden; das genaue Ergebnis ist 15,204773. Für Laufzeitzinsdarlehen kann nur der Effektivzins berechnet werden. Die Gebühr ist in Prozent vom Kreditbetrag einzugeben. Nach der Berechnung ist die Gebühr nicht mehr gespeichert; für Vergleichsrechnungen muß sie daher neu eingegeben werden.

Programmgesteuert werden bei der Berechnung für Laufzeitzinsdarlehen einige Größen verändert:

- Auszahlungsbetrag wird auf 100 gesetzt
- Rate wird auf Durchschnittsrate für 100 gesetzt
- Ratenfälligkeit wird auf 12 gesetzt
- Restschuld wird auf 0 gesetzt

Nach der Berechnung des Effektivzinssatzes für Laufzeitzinsdarlehen ist ein direkter Konditionsvergleich mit einem normalen Sollstellungdarlehen möglich.

Nach der Ergebnisanzeige (EFF = 15,20) ist z.B. einzugeben:

Zinssollstellung	(TASTE G)	4	
Tilgungsverrechnung	(TASTE H)	4	
Nominalzins	(TASTE B)	-	Ergebnis = 13,58

Ein Sollstellungsdarlehen mit diesen Konditionen entspricht in allen Zahlungsströmen genau dem Laufzeitzinsdarlehen.

Die Größen Laufzeit / Rate / Restschuld beeinflussen sowohl die Berechnung im nominellen als auch im effektiven Teil. Soll eine dieser Größen berechnet werden, muß bestimmt werden, ob zur Berechnung nominelle oder effektive Größen zugrunde gelegt werden sollen. Dies geschieht durch die Art der Eingabe: Wurden vor der Berechnung von Laufzeit / Rate / Restschuld sowohl der Auszahlungsbetrag als auch der Effektivzinssatz eingegeben, wird nach den effektiven Größen gerechnet. Andernfalls erfolgt die Berechnung von Laufzeit / Rate / Restschuld aus den nominellen Größen. Dabei kommt es nicht nur auf den gespeicherten Wert, sondern auf die tatsächliche Eingabe an (wird durch Flag 22 erkannt).

Beispiel: Zu dem ersten Berechnungsbeispiel (Seite 75) soll der Nominalzinssatz berechnet werden, damit sich ein Effektivzinssatz von 10,50 % ergibt. Zunächst ist wieder die Laufzeit zu berechnen, anschließend:

Auszahlung	(TASTE F)	100	
Effektivzins	(TASTE J)	10,5	
Rate	(TASTE D)	-	Ergebnis = 0,88
Nominalzins	(TASTE B)	-	Ergebnis = 9,57

Der Auszahlungsbetrag mußte hier nochmals eingegeben werden, obwohl der gleiche Wert bereits gespeichert war! Da der Nominalzins nicht zu den effektiven Größen gehört, mußte zunächst die letzte unbekannte effektive Größe, die Rate, berechnet werden.

Nach der geschilderten Methode ist mit dem Programm "F" jede Art der Berechnung nomineller Größen aus gegebenen effektiven Größen möglich. Das Programm wird auch für diese Berechnungen nur über die Tasten A bis J bedient.

Einige Beispielrechnungen sollen nun noch die Anwendung des Programms zur Berechnung von Sparkonten zeigen. Alle Berechnungen werden nur im nominellen Teil durchgeführt, da für Sparkonten nominell gleich effektiv gilt.

Da die Bezeichnungen der Eingabegrößen auf Kreditberechnungen abgestellt sind, ist eine gewisse Eingewöhnung erforderlich. Bei der Berechnung des Endkapitals mit Ratenzahlungen sollte man sich an dem folgenden einfachen Beispiel über die Art der Berechnung Klarheit verschaffen. Die Anzahl der Raten entspricht der Laufzeit in Verbindung mit der Ratenfälligkeit. Dabei kann die letzte Rate eventuell genau zum Ende der Laufzeit (also nicht mehr zinswirksam) fällig sein. Die erste Rate wird nach der ersten Ratenfälligkeit verrechnet. Soll die erste Rate sofort bei Beginn der Laufzeit gezahlt werden, so ist das Anfangskapital um die erste Rate zu erhöhen.

Eingaben:

Laufzeit	(TASTE A)	1	
Zinssatz	(TASTE B)	6	
Kapital	(TASTE C)	100	
Rate	(TASTE D)	- 100	Einzahlungen negativ!
Zinskapitalisierung	(TASTE G)	1	
Ratenverrechnung	(TASTE H)	1	
Ratenfälligkeit	(TASTE I)	1	
Rest	(TASTE E)	-	Ergebnis = 206,00

Dieses Ergebnis errechnet sich aus dem Anfangskapital von DM 100,--, das für genau ein Jahr mit 6 % verzinst wird, zuzüglich der ersten Sparrate von DM 100,--, die jedoch nicht mehr zinswirksam ist. Berücksichtigt man diese Berechnungsweise , läßt sich jede Staffelung eines Sparkontos berechnen. Zu beachten ist, daß die Ratenverrechnung (TASTE H) immer gleich der Ratenfälligkeit eingegeben werden muß.

Beispiel für monatliche Sparrate:

Laufzeit	(TASTE A)	48	
Zinssatz	(TASTE B)	5	
Kapital	(TASTE C)	100	
Rate	(TASTE D)	- 100	
Zinskapitalisierung	(TASTE G)	1	
Ratenverrechnung	(TASTE H)	12	
Ratenfälligkeit	(TASTE I)	12	
Rest	(TASTE E)	-	Ergebnis = 5412,23

Sollten 48 Monatsraten mit sofortigem Zahlungsbeginn berechnet werden, ist von dem errechneten Ergebnis die letzte Rate abzuziehen. Diese Modellrechnung weicht jedoch von einer tatsächlichen Sparkontostaffelung geringfügig ab. Zunächst werden Pfennigbeträge nicht verzinst; außerdem müßten alle Zahlungen exakt termingerecht, beginnend vor Anfang der Zinsperiode (also am 31. Dezember des Vorjahres) gezahlt werden. Im übrigen gehen alle Modellrechnungen davon aus, daß der Zinssatz unverändert bleibt.

Auch die Effektivverzinsung von Prämien- oder Bonussparkonten läßt sich berechnen. Wird z.B. nach vier Jahren Verzinsung mit 5 % ein Bonus von 10 % gezahlt, ergibt sich folgende Berechnung:

Laufzeit	(TASTE A)	4	
Zinssatz	(TASTE B)	5	
Kapital	(TASTE C)	1000	
Rate	(TASTE D)	0	
Zinskapitalisierung	(TASTE G)	1	
Ratenverrechnung	(TASTE H)	1	
Ratenfälligkeit	(TASTE I)	1	
Rest	(TASTE E)	-	Ergebnis = 1215,51

Zu diesem Kontostand am Ende der Laufzeit muß der Bonus addiert werden; das Ergebnis ist der neue "Rest", aus dem der neue Zins errechnet wird:

Rest	(TASTE E)	100 +	
Zins	(TASTE B)	-	Ergebnis = 7,10

Der Bonus von 10 % erhöht den Effektivzins also nicht um 2,5 % (10 % verteilt auf vier Jahre), da er erst am Ende der Laufzeit gezahlt wird.

Das Programm "F" enthält einige synthetische Befehle, um ein Optimum an Schnelligkeit zu erreichen und um einige Bytes einzusparen, damit das Programm auf zwei Magnetkarten gespeichert werden kann.

Für LBL "F", einschließlich der Zuordnung zur Taste SHIFT/SHIFT, sind mit dem Load-Bytes-Programm folgende Eingaben erforderlich:

192, 0, 242, 11, 70.

Um den CAT 1 zu berichtigen, muß anschließend mit GTO .. der Programmspeicher gepackt werden. Dann ist aber die Tastenzuordnung noch nicht wirksam!

Um die Tastenzuordnung zu aktivieren, muß das Programm im USER-Modus in einen externen Speicher gebracht werden. Hierzu kann der erweiterte Speicher des X-Functions Moduls, der Kartenleser oder das Cassettenlaufwerk benutzt werden. Dann ist das Programm im Hauptspeicher zu löschen und anschließend, wiederum im USER-Modus, aus dem externen Speicher wieder in den Programmspeicher zurückzuladen. Jetzt ist die synthetisch erzeugte Zuordnung zur Taste SHIFT/SHIFT wirksam.

Die in dem Programm enthaltenen Kurzform-Exponenten E, E1, E2 können durch die entsprechenden normalen Eingaben ersetzt werden.

In den Programmzeilen 81, 85 und 92 sind synthetische Sprungbefehle enthalten, die aus dem Programmausdruck nicht erkannt werden können. Diese sind erforderlich, da die Sprungweite bis zu dem entsprechenden Label in dem normalen Befehl nicht gespeichert werden könnte. Mit dem Load-Bytes-Programm ist hier jeweils einzugeben: 208, 0, 3; sollen keine synthetischen Befehle verwandt werden, so ist LBL 03 durch LBL 15 zu ersetzen, alle zugehörigen Sprungbefehle müssen dann von GTO 03 in GTO 15 geändert werden.

Zur Erzeugung der Laufschrift wurden in Zeile 94 bis 97 die synthetischen Befehle RCL d und STO d verwandt, da der Zustand von Flag 21 nicht verändert werden darf, falls ein Drucker angeschlossen ist. Als Ersatz können folgende Befehle eingegeben werden:

SF 25, CF 21, AVIEW, SF 99, FS? 55, SF 21.

Um das Programm trotz der vielseitigen Anforderungen so kurz wie möglich zu gestalten, wurde auf eine besondere Formatierung der Druckausgabe verzichtet. Wird kein Drucker angeschlossen und ein guter Schätzwert vorgeben, liegt die Rechenzeit bei etwa zwanzig Sekunden.

Die Anwendungsmöglichkeiten des Programms "F" gehen wesentlich über das hinaus, was in den Beispielrechnungen gezeigt wurde. So sind z.B. auch Aufzinsungs- und Abzinsungsrechnungen problemlos möglich. Auch Barwerte zu Ratenforderungen (z.B. Rentenkapitalisierungen) können berechnet werden.

SQRT

Benötigte Programmregister: 5

Zeile 1 (1-6)

Zeile 2 (7-17)

Zeile 3 (18)

Benötigte Programmregister: 5

Zeile 1 (1-6)

Zeile 2 (7-16)

Zeile 3 (17-18)

PYT

Benötigte Programmregister: 10

Zeile 1 (1-7)

Zeile 2 (7-18)

Zeile 3 (19-28)

Zeile 4 (29-35)

Zeile 5 (36-41)

Benötigte Programmregister: 11

Zeile 1 (1-6)

Zeile 2 (7-17)

Zeile 3 (18-27)

Zeile 4 (27-35)

Zeile 5 (36-41)

Zeile 6 (41-44)

UZ

Benötigte Programmregister: 10

Zeile 1 (1-8)

Zeile 2 (9-17)

Zeile 3 (18-27)

Zeile 4 (28-36)

Zeile 5 (37-41)

Zeile 6 (42-43)

Y hoch X

Benötigte Programmregister: 9

Zeile 1 (1-7)

Zeile 2 (8-15)

Zeile 3 (15-21)

Zeile 4 (22-30)

Zeile 5 (30-32)

Benötigte Programmregister: 9

Zeile 1 (1-7)

Zeile 2 (8-14)

Zeile 3 (14-21)

Zeile 4 (21-28)

Zeile 5 (29-35)

DH

Benötigte Programmregister: 6

Zeile 1 (1-6)

Zeile 2 (7-16)

Zeile 3 (17-25)

Zeile 4 (25)

HD

Benötigte Programmregister: 5

Zeile 1 (1-6)

Zeile 2 (7-16)

Zeile 3 (17-20)

DB

Benötigte Programmregister: 5

Zeile 1 (1-7)

Zeile 2 (7-17)

Zeile 3 (18-20)

BD

Benötigte Programmregister: 4

Zeile 1 (1-7)

Zeile 2 (7-13)

XB

Benötigte Programmregister: 5

Zeile 1 (1-6)

Zeile 2 (7-12)

Zeile 3 (12-16)

XA - AX

Benötigte Programmregister: 10

Zeile 1 (1-5)

Zeile 2 (6-14)

Zeile 3 (15-23)

Zeile 4 (23-28)

Zeile 5 (29-35)

STAT

Benötigte Programmregister: 8

Zeile 1 (1-6)

Zeile 2 (7-15)

Zeile 3 (15-23)

Zeile 4 (24-26)

DTA

Benötigte Programmregister: 8

Zeile 1 (1-7)

Zeile 2 (8-15)

Zeile 3 (16-24)

Zeile 4 (25-30)

DFL

Benötigte Programmregister: 8

Zeile 1 (1-6)

Zeile 2 (7-15)

Zeile 3 (16-24)

Zeile 4 (25-31)

CA

Benötigte Programmregister: 4

Zeile 1 (1-5)

Zeile 2 (6-13)

DA

Benötigte Programmregister: 4

Zeile 1 (1-5)

Zeile 2 (6-13)

CODE

Benötigte Programmregister: 4

Zeile 1 (1-5)

Zeile 2 (5-14)

Zeile 3 (14)

C-A

Benötigte Programmregister: 5

Zeile 1 (1-6)

Zeile 2 (7-15)

Zeile 3 (15-19)

D-A

Benötigte Programmregister: 6

Zeile 1 (1-5)

Zeile 2 (6-15)

Zeile 3 (15-21)

LB

Benötigte Programmregister: 12

Zeile 1 (1-5)

Zeile 2 (6-14)

Zeile 3 (14-25)

Zeile 4 (26-33)

Zeile 5 (34-41)

Zeile 6 (42-49)

Zeile 7 (49-51)

LB

Benötigte Programmregister: 17

Zeile 1 (1-7)

Zeile 2 (8-15)

Zeile 3 (16-25)

Zeile 4 (25-33)

Zeile 5 (33-40)

Zeile 6 (41-50)

Zeile 7 (51-54)

Zeile 8 (54-61)

Zeile 9 (62-66)

--

Benötigte Programmregister: 3

Zeile 1 (1-6)

Zeile 2 (7-9)

Benötigte Programmregister: 3

Zeile 1 (1-6)

Zeile 2 (7-10)

BA

Benötigte Programmregister: 6

Zeile 1 (1-5)

Zeile 2 (6-12)

Zeile 3 (12-19)

DP

Benötigte Programmregister: 6

Zeile 1 (1-8)

Zeile 2 (9-18)

Zeile 3 (19-23)

PD

Benötigte Programmregister: 6

Zeile 1 (1-5)

Zeile 2 (6-14)

Zeile 3 (15-24)

XF

Benötigte Programmregister: 2

Zeile 1 (1-4)

TD

Benötigte Programmregister: 9

Zeile 1 (1-4)

Zeile 2 (4)

Zeile 3 (5-12)

Zeile 4 (12-17)

Zeile 5 (17-20)

BAT

Benötigte Programmregister: 5

Zeile 1 (1-5)

Zeile 2 (6-12)

Zeile 3 (12-15)

O

Benötigte Programmregister: 8

Zeile 1 (1-6)

Zeile 2 (6-15)

Zeile 3 (15-23)

Zeile 4 (24-31)

A?

Benötigte Programmregister: 11

Zeile 1 (1-4)

Zeile 2 (5-14)

Zeile 3 (15-22)

Zeile 4 (23-29)

Zeile 5 (29-38)

Zeile 6 (38-45)

Benötigte Programmregister: 15

Zeile 1 (1-5)

Zeile 2 (5-13)

Zeile 3 (14-23)

Zeile 4 (23-27)

Zeile 5 (28-35)

Zeile 6 (36-42)

Zeile 7 (43-51)

Zeile 8 (52-60)

AF

Benötigte Programmregister: 15

Zeile 1 (1-5)

Zeile 2 (5-13)

Zeile 3 (13-19)

Zeile 4 (20-28)

Zeile 5 (29-37)

Zeile 6 (37-44)

Zeile 7 (45-53)

Zeile 8 (54-60)

Benötigte Programmregister: 20

Zeile 1 (1-3)

Zeile 2 (3-10)

Zeile 3 (11-18)

Zeile 4 (18-26)

Zeile 5 (26-34)

Zeile 6 (35-41)

Zeile 7 (42-49)

Zeile 8 (50-57)

Zeile 9 (58-66)

Zeile 10 (66-75)

Zeile 11 (75-77)

AF

Benötigte Programmregister: 10

Zeile 1 (1-3)

Zeile 2 (3-10)

Zeile 3 (11-17)

Zeile 4 (17-25)

Zeile 5 (26-32)

AC

Benötigte Programmregister: 19

Zeile 1 (1-4)

Zeile 2 (5-13)

Zeile 3 (14-21)

Zeile 4 (21-28)

Zeile 5 (29-37)

Zeile 6 (38-48)

Zeile 7 (48-56)

Zeile 8 (57-64)

Zeile 9 (65-73)

Zeile 10 (74-81)

AC

Benötigte Programmregister: 27

Zeile 1 (1-5)

Zeile 2 (5-13)

Zeile 3 (14-22)

Zeile 4 (23-27)

Zeile 5 (28-35)

Zeile 6 (36-42)

Zeile 7 (43-51)

Zeile 8 (52-59)

Zeile 9 (59-64)

Zeile 10 (65-73)

Zeile 11 (74-82)

Zeile 12 (82-89)

Zeile 13 (90-98)

Zeile 14 (99-106)

Zeile 15 (107-110)

A-

Benötigte Programmregister: 3

Zeile 1 (1-5)

Zeile 2 (6-7)

S0

Benötigte Programmregister: 6

Zeile 1 (1-6)

Zeile 2 (7-14)

Zeile 3 (15-19)

SO

Benötigte Programmregister: 5

Zeile 1 (1-7)

Zeile 2 (7-15)

Zeile 3 (15-19)

SX

Benötigte Programmregister: 5

Zeile 1 (1-6)

Zeile 2 (6-12)

Zeile 3 (13-16)

SD

Benötigte Programmregister: 12

Zeile 1 (1-7)

Zeile 2 (8-15)

Zeile 3 (16-23)

Zeile 4 (23-28)

Zeile 5 (29-36)

Zeile 6 (36-42)

Zeile 7 (42-43)

PB

Benötigte Programmregister: 13

Zeile 1 (1-3)

Zeile 2 (3-6)

Zeile 3 (7-15)

Zeile 4 (16-23)

Zeile 5 (24-32)

Zeile 6 (33-40)

Zeile 7 (41-45)

E?

Benötigte Programmregister: 4

Zeile 1 (1-5)

Zeile 2 (5-12)

C?

Benötigte Programmregister: 4

Zeile 1 (1-5)

Zeile 2 (5-13)

Zeile 3 (13)

KA

Benötigte Programmregister: 11

Zeile 1 (1-5)

Zeile 2 (6-12)

Zeile 3 (12-20)

Zeile 4 (21-29)

Zeile 5 (30-38)

Zeile 6 (38-41)

KA

Benötigte Programmregister: 9

Zeile 1 (1-5)

Zeile 2 (5-13)

Zeile 3 (13-22)

Zeile 4 (23-32)

Zeile 5 (32-36)

NEWM

Benötigte Programmregister: 6

Zeile 1 (1-4)

Zeile 2 (4-12)

Zeile 3 (12-19)

F

Benötigte Programmregister: 64

Zeile 1 (1-8)

Zeile 2 (9-17)

Zeile 3 (18-25)

Zeile 4 (25-33)

Zeile 5 (33-39)

Zeile 6 (40-44)

Zeile 7 (44-49)

Zeile 8 (50-55)

Zeile 9 (56-61)

Zeile 10 (61-67)

Zeile 11 (68-74)

Zeile 12 (74-79)

Zeile 13 (80-85)

Zeile 14 (86-91)

Zeile 15 (91-98)

Zeile 16 (98-106)

Zeile 17

Zeile 18 (117-128)

Zeile 19 (129-135)

Zeile 20 (136-144)

Zeile 21 (145-153)

Zeile 22 (154-161)

Zeile 23 (162-171)

Zeile 24 (172-183)

Zeile 25 (184-196)

Zeile 26 (197-205)

Zeile 27 (205-215)

Zeile 28 (216-227)

Zeile 29 (227-239)

Zeile 30 (240-250)

Zeile 31 (251-263)

Zeile 32 (263-274)

Zeile 33 (275-287)

Zeile 34 (288-298)

Zeile 35 (299-302)

Benutzerhandbuch HP-41CX, Band II
Hewlett-Packard, 1983. Best.-Nr. 00041-90494

HP-41 SYNTHETIC PROGRAMMING MADE EASY
Keith Jerett, 1982
SYNTHETIX, 1540 Mathews Ave. Manhattan Beach, CA 90266 U S A

HP-41 EXTENDED FUNCTIONS MADE EASY
Keith Jerett, 1983
SYNTHETIX, 1540 Mathews Ave. Manhattan Beach, CA 90266 U S A

SYNTHETIC PROGRAMMING ON THE HP-41C
William C. Wickes, 1980
Larken Publications, 4517 NW Queens Avenue,
Corvallis, Oregon 97330 U S A

PPC ROM USER'S MANUAL
Handbuch zum PPC Modul, PPC 1981
PPC, P.O. Box 9599, Fountain Valley, California 92728 U S A

CALCULATOR TIPS & ROUTINES
Edited by John Dearing, 1981
Corvallis Software, INC. P.O. Box 1412, Corvallis, Oregon 97339 U S A

Die Bücher von W.C.Wickes, J.Dearing und Keith Jarett
sind auch als deutsche Übersetzung erschienen.
Heldermann Verlag Berlin, Herderstr. 6-7, 1000 Berlin 41

PPC CALCULATOR JOURNAL
Aktuelle Informationsschrift des amerikanischen Anwenderclubs
PPC, P.O. Box 9599, Fountain Valley, California 92728 U S A

PRISMA
Aktuelle Informationsschrift des deutschen Anwenderclubs
CCD, Computerclub Deutschland e.V., Postfach 2119, 6242 Kronberg 2

CCD Modul, 1984
W & W, Dr. W. Baltes und W. Kötz, Postfach 800133,
5060 Bergisch Gladbach 2

HP-41C QUICK REFERENCE CARD FOR SYNTHETIC PROGRAMMING

© 1982, SYNTHETIX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
	CAT	@c (GTO..)	DEL	COPY	CLP	R/S	SIZE	BST	SST	ON	PACK	←(PRGM)	USR/P/A	2 _ _	SHIFT	ASN	
0	NULL 00 ⁻ 0 ◆	LBL 00 01 ⁑ 1 ×	LBL 01 02 ■ 2 ⁜	LBL 02 03 ■ 3 ←	LBL 03 04 ⁑ 4 α	LBL 04 05 ⁑ 5 ß	LBL 05 06 ⁑ 6 Γ	LBL 06 07 ■ 7 ↓	LBL 07 08 ■ 8 Δ	LBL 08 09 ■ 9 σ	LBL 09 10 ■ 10 ◆	LBL 10 11 ■ 11 ↘	LBL 11 12 ⁻ 12 μ	LBL 12 13 ⁻ 13 ∠	LBL 13 14 ■ 14 ⁻	LBL 14 15 ■ 15 ⁜	0
1	0 16 ■ 16 ⊟	1 17 ■ 17 Ω	2 18 ■ 18 δ	3 19 ■ 19 ä	4 20 ■ 20 ä	5 21 ■ 21 ä	6 22 ■ 22 ö	7 23 ■ 23 ü	8 24 ■ 24 ö	9 25 ■ 25 Ü	. 26 ■ 26 ü	EEX 27 ■ 27 ß	NEG 28 ■ 28 œ	GTO ⁻ 29 ⁻ 29 ≠	XEQ ⁻ 30 ■ 30 £	W ⁻ 31 ■ 31 ⁜	1
2	RCL 00 32 32	RCL 01 33 ! 33 !	RCL 02 34 " 34 "	RCL 03 35 ⁒ 35 #	RCL 04 36 $ 36 $	RCL 05 37 % 37 %	RCL 06 38 & 38 &	RCL 07 39 ' 39 '	RCL 08 40 (40 (	RCL 09 41) 41)	RCL 10 42 * 42 *	RCL 11 43 + 43 +	RCL 12 44 , 44 ,	RCL 13 45 - 45 −	RCL 14 46 . 46 .	RCL 15 47 / 47 /	2
3	STO 00 48 0 48 0	STO 01 49 1 49 1	STO 02 50 2 50 2	STO 03 51 3 51 3	STO 04 52 4 52 4	STO 05 53 5 53 5	STO 06 54 6 54 6	STO 07 55 7 55 7	STO 08 56 8 56 8	STO 09 57 9 57 9	STO 10 58 : 58 :	STO 11 59 ; 59 ;	STO 12 60 < 60 <	STO 13 61 = 61 =	STO 14 62 >
62 >	STO 15 63 ? 63 ?	3
4	+ 64 P 64 @	− 65 A 65 A	✱ 66 B 66 B	/ 67 C 67 C	X<Y? 68 D 68 D	X>Y? 69 E 69 E	X≤Y? 70 F 70 F	Σ+ 71 G 71 G	Σ− 72 H 72 H	HMS+ 73 I 73 I	HMS− 74 J 74 J	MOD 75 K 75 K	% 76 L 76 L	%CH 77 M 77 M	P→R 78 N 78 N	R→P 79 O 79 O	4
5	LN 80 P 80 P	X↑2 81 Q 81 Q	SQRT 82 R 82 R	Y↑X 83 S 83 S	CHS 84 T 84 T	E↑X 85 U 85 U	LOG 86 V 86 V	10↑X 87 W 87 W	E↑ X-1 88 X 88 X	SIN 89 Y 89 Y	COS 90 Z 90 Z	TAN 91 [91 [	ASIN 92 \ 92 \	ACOS 93] 93]	ATAN 94 ↑ 94 ↑	→DEC 95 _ 95 _	5
6	1/X 96 ⁻ 96 ⁻	ABS 97 a 97 a	FACT 98 b 98 b	X≠0? 99 c 99 c	X>0? 100 d 100 d	LN1+X 101 e 101 e	X<0? A ■ 102 f	X=0? B ■ 103 g	INT C ■ 104 h	FRC D ■ 105 i	D→R E ■ 106 j	R→D F ■ 107 k	→HMS G ■ 108 l	→HR H ■ 109 m	RND I ■ 110 n	→OCT J ■ 111 o	6
7	CLΣ T ■ 112 p	X<>Y Z ■ 113 q	PI Y ■ 114 r	CLST X ■ 115 s	R↑ L ■ 116 t	RDN M [■ 117 u	LASTX N \ ■ 118 v	CLX O] ■ 119 w	X=Y? P↑ ■ 120 x	X≠Y? Q_ ■ 121 y	SIGN ⊢⁻ ■ 122 z	X≤0? a ■ 123 ⁻	MEAN b ■ 124	SDEV c ■ 125 →	AVIEW d ⁻ 126 Σ	CLD e ⊢ 127 ⊢	7
	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	
	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1	← bit numbers in a 7-byte register
	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31	32 33 34 35	36 37 38 39	40 41 42 43	44 45 46 47	48 49 50 51	52 53 54 55			

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
8	DEG IND 00 128 ◆	RAD IND 01 129 ×	GRAD IND 02 130 ⁜	ENTER↑ IND 03 131 ←	STOP IND 04 132 α	RTN IND 05 133 ß	BEEP IND 06 134 Γ	CLA IND 07 135 ↓	ASHF IND 08 136 Δ	PSE IND 09 137 σ	CLRG IND 10 138 ◆	AOFF IND 11 139 ↘	AON IND 12 140 μ	OFF IND 13 141 ∠	PROMPT IND 14 142 ⁻	ADV IND 15 143 ⁜	8
9	RCL IND 16 144 ⊟	STO IND 17 145 Ω	ST+ IND 18 146 δ	ST− IND 19 147 ä	ST ✱ IND 20 148 ä	ST/ IND 21 149 ä	ISG IND 22 150 ö	DSE IND 23 151 ü	VIEW IND 24 152 ö	Σ REG IND 25 153 Ü	ASTO IND 26 154 ü	ARCL IND 27 155 œ	FIX IND 28 156 œ	SCI IND 29 157 ≠	ENG IND 30 158 £	TONE IND 31 159 ⁜	9
A	XR 0-3 IND 32 160	XR 4-7 IND 33 161 !	XR8-11 IND 34 162 "	X12-15 IND 35 163 #	X16-19 IND 36 164 $	X20-23 IND 37 165 %	X24-27 IND 38 166 &	X28-31 IND 39 167 '	SF IND 40 168 (	CF IND 41 169)	FS?C IND 42 170 *	FC?C IND 43 171 +	FS? IND 44 172 ,	FC? IND 45 173 -	GTO XEQ IND IND 46 174 .	SPARE IND 47 175 /	A
B	SPARE IND 48 176 0	GTO 00 IND 49 177 1	GTO 01 IND 50 178 2	GTO 02 IND 51 179 3	GTO 03 IND 52 180 4	GTO 04 IND 53 181 5	GTO 05 IND 54 182 6	GTO 06 IND 55 183 7	GTO 07 IND 56 184 8	GTO 08 IND 57 185 9	GTO 09 IND 58 186 :	GTO 10 IND 59 187 ;	GTO 11 IND 60 188 <	GTO 12 IND 61 189 =	GTO 13 IND 62 190 >	GTO 14 IND 63 191 ?	B
C	GLOBAL IND 64 192 @	GLOBAL IND 65 193 A	GLOBAL IND 66 194 B	GLOBAL IND 67 195 C	GLOBAL IND 68 196 D	GLOBAL IND 69 197 E	GLOBAL IND 70 198 F	GLOBAL IND 71 199 G	GLOBAL IND 72 200 H	GLOBAL IND 73 201 I	GLOBAL IND 74 202 J	GLOBAL IND 75 203 K	GLOBAL IND 76 204 L	GLOBAL IND 77 205 M	X<>-- IND 78 206 N	LBL -- IND 79 207 O	C
D	GTO -- IND 80 208 P	GTO -- IND 81 209 Q	GTO -- IND 82 210 R	GTO -- IND 83 211 S	GTO -- IND 84 212 T	GTO -- IND 85 213 U	GTO -- IND 86 214 V	GTO -- IND 87 215 W	GTO -- IND 88 216 X	GTO -- IND 89 217 Y	GTO -- IND 90 218 Z	GTO -- IND 91 219 [	GTO -- IND 92 220 \	GTO -- IND 93 221]	GTO -- IND 94 222 ↑	GTO -- IND 95 223 _	D
E	XEQ -- IND 96 224 ⁻	XEQ -- IND 97 225 a	XEQ -- IND 98 226 b	XEQ -- IND 99 227 c	XEQ -- IND100 228 d	XEQ -- IND101 229 e	XEQ -- IND102 230 f	XEQ -- IND103 231 g	XEQ -- IND104 232 h	XEQ -- IND105 233 i	XEQ -- IND106 234 j	XEQ -- IND107 235 k	XEQ -- IND108 236 l	XEQ -- IND109 237 m	XEQ -- IND110 238 n	XEQ -- IND111 239 o	E
F	TEXT 0 IND T 240 p	TEXT 1 IND Z 241 q	TEXT 2 IND Y 242 r	TEXT 3 IND X 243 s	TEXT 4 IND L 244 t	TEXT 5 IND M [245 u	TEXT 6 IND N \ 246 v	TEXT 7 IND O] 247 w	TEXT 8 IND P↑ 248 x	TEXT 9 IND Q_ 249 y	TEXT10 IND ⊢⁻ 250 z	TEXT11 IND a 251 ⁻	TEXT12 IND b 252 l	TEXT13 IND c 253 →	TEXT14 IND d 254 Σ	TEXT15 IND e 255 ⊢	F
	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	

For price information and a list of dealers in your area, send a self-addressed stamped envelope to: SYNTHETIX, 1540 Mathews Ave., Manhattan Beach, CA 90266, USA

Info-Gutschein

Bitte informieren Sie mich (uns) ständig über ihre Neuerscheinungen auf dem Gebiet Mikrocomputer.

Ich (wir) besitze(n) folgendes Gerät:

Hauptanwendungsgebiete:

Diese Karte entnahm(en) ich (wir) dem Buch:
Kruse, Optimales Programmieren mit dem HP-41

Meine (unsere) Buchhandlung

Gleichzeitig bestelle(n) ich (wir) folgende Bücher:

Anzahl	Autor und Titel	Preis
	Schumny, µC-Jahrbuch 1985	38,–

Anschrift:

Beruf/Branche

Datum Unterschrift

Lieber Leser!

Wenn Sie Interesse haben, aktiv an der Weiterentwicklung unseres Literaturprogramms zum Bereich µC mitzuarbeiten, z.B. durch Veröffentlichung ausgetesteter Programme zu bestimmten Anwendungsgebieten, dann schreiben Sie uns.

Wir freuen uns über Ihre Nachricht und werden uns umgehend mit Ihnen in Verbindung setzen.

Mit freundlichem Gruß
Lektorat Infoware

Bitte
mit
60 Pf.
freimachen

Antwort

Friedr. Vieweg & Sohn
Verlagsgesellschaft mbH

Postfach 5829

D-6200 Wiesbaden 1

Gerhard Kruse

Optimales Programmieren mit dem HP-41

Dieses Buch stellt Methoden der fortschrittlichen Programmierung des HP-41 vor. Es werden 51 vollständige Programme und Routinen in Funktion, Aufbau und Anwendung dargestellt. Anhand allgemein verständlicher Programme werden die verschiedenen Techniken zur Programmoptimierung aufgezeigt.

Besonders die intensive Nutzung des X-Functions Moduls und der synthetischen Programmierung werden an vielen Programmbeispielen ausführlich erläutert.

Die Grundlagen der synthetischen Programmierung, "Load Bytes" und "Key Assignment" werden als besonders kurze Programme vorgestellt. Anwendungen und sinnvoller Nutzen synthetischer Programmierung werden an einfachen und komplexen Programmbeispielen schrittweise verständlich gemacht.

Mit den Programmbeschreibungen bietet das Buch dem Benutzer des HP-41 eine Fülle wertvoller Anregungen, um die Möglichkeiten seines Taschencomputers optimal zu nutzen.

Gerhard Kruse

ISBN 3-528-04335-0