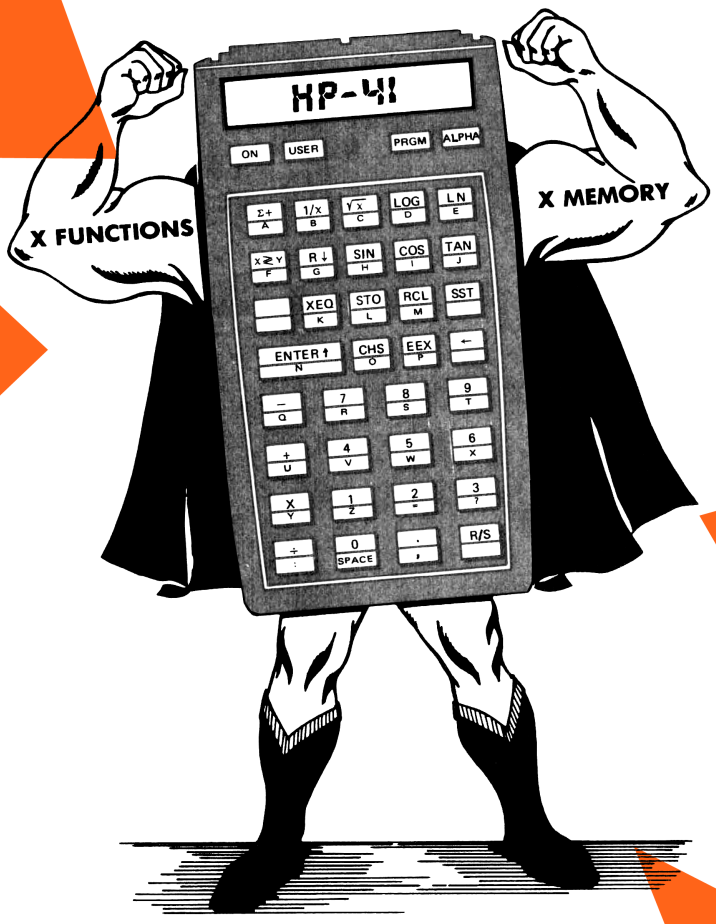
K. Jarett

Erweiterte Funktionen des HP-41 – leicht gemacht

Deutsche Ausgabe von H. Dalkowski



Heldermann Verlag Berlin

Keith Jarett

Erweiterte Funktionen des HP-41 – leicht gemacht

Deutsche Ausgabe von Heinz Dalkowski

Heldermann Verlag Berlin

Keith Jarett
Synthetix
P. O. Box 1080
Berkeley, CA 94701
U.S.A.

Heinz Dalkowski
Seidelbastweg 88a
D – 1000 Berlin 47

CIP – Kurztitelaufnahme der Deutschen Bibliothek

Jarett, Keith: Erweiterte Funktionen des HP-41 – leicht gemacht / Keith Jarett. Dt. Ausg. von Heinz Dalkowski. – Berlin : Heldermann, 1986.
Einheitssacht.: HP-41 extended functions made easy (dt.)
ISBN 3 – 88538 – 803 – 0
NE: Dalkowski, Heinz (Bearb.)

ISBN 3 – 88538 – 803 – 0

INHALTSVERZEICHNIS

Teil II: Anwendung des X-Memorys und der X-Funktionen

Teil III: Ergänzungen

VORWORT DES AUTORS

Dieses Buch ist meiner Frau, Catherine Van De Rostyne, gewidmet, die nicht hilfreicher und der Arbeit förderlicher hätte sein können, als sie es war.

Die wichtigsten Beiträge zu diesem Buch leistete Clifford Stern, der mir während der gesamten Entstehungsphase beratend zur Seite stand. Er ist einer der wenigen 'Großmeister' des HP-41 und wahrscheinlich mehr als jeder andere mit den verborgensten Feinheiten des HP-41 Betriebssystems vertraut. Aus seiner Feder stammen die höchst kunstvollen Hilfsprogramme des Kapitels 10. Er war es, der das Buch einer prüfenden Durchsicht unterzog und wertvolle Verbesserungsvorschläge beisteuerte.

Von den anderen Zulieferern sind zu nennen: Erik Christensen, der der Autor des Text-Editors und der zugehörigen Gebrauchsanweisung in Kapitel 8 ist. Tapani Tarvainen und Gerard Westen, die gemeinsam das erstaunliche Tastenzuweisungsprogramm in Kapitel 10 erstellten, und Alan McCornack, der das Postadressenprogramm des Kapitels 7 schrieb und im Verlaufe der Arbeit ebenfalls viele nützlichen Vorschläge machte.

K.J.

VORWORT DES ÜBERSETZERS

In Leserbriefen zu vorangegangenen HP-41 Übersetzungen wurde wieder und wieder nach der vorliegenden Übertragung des 'Jarett II' gefragt. Verlag und Übersetzer waren sich indessen darin einig, solchem Drängen nach Eile nicht nachzugeben. Eine bis in die Einzelheiten gehende Durchsicht einer Vorlage, und sei sie so ausgezeichnet wie die von Keith Jarett es war, ist allemal vonnöten und kostet ihre Zeit. Ich hoffe, daß das Ergebnis die Verzögerung rechtfertigt.

Dank zuallererst an Christine Masuhr, die hier zum dritten Male ein HP-41 Manuskript höchst zuverlässig in einen lesbaren Text umsetzte und im Verlaufe der Satzarbeiten den ständigen Änderungen, nachträglichen Einschüben und verspätet entdeckten Ungenauigkeiten große Nachsicht entgegenbrachte.

Dank an nächster Stelle an Konrad Albers. Er war es, der mir und somit den Lesern dieses Buches mit seiner Kunst aushalf, indem er die eingestreuten Barcodes (Abschnitte 10C und 10F sowie Kapitel 11) anfertigte. Z.Z. arbeitet er an einem Buch über Barcodes, das — soweit ich nach dem Einblick, den vorzeitig zu nehmen er mir Gelegenheit gab, zu urteilen vermag — alles in den Schatten stellen wird, was bislang in Bezug auf den Einsatz des optischen Lesestiftes einem breiteren HP-Benutzerkreis zur Verfügung steht. Ich selbst bin sehr gespannt auf seine Publikation, um Barcodes endlich für mehr als das bloße Einlesen von Programmen oder den einfachen Aufruf von System-Funktionen benutzen und die im Drucker schlummernden Fähigkeiten wecken zu können.

Dank schließlich an Heinz Kröger, der wieder das Stichwortverzeichnis mit seinem Osborne-1 anfertigte. Es wird sich, so hoffe ich, seines Umfanges wegen als nützlich erweisen und soll insbesondere dabei behilflich sein, auch kleine Hinweise, Randbemerkungen und unscheinbare Tips, die erfahrungsgemäß unmittelbar nach dem Lesen im Dickicht des Textes untergehen, wiederauffindbar zu machen.

Zum Buch selbst soviel: Die Darstellung ist — um dem Wort 'episch' aus dem Wege zu gehen — ausgesprochen textintensiv. Der Grund dafür ist in der verdienstvollen Absicht des Autors, Anfängern die Lektüre zu erleichtern, zu suchen. Mancher wird vielleicht auf mehr Tempo bei der Bewältigung des Stoffes drängen. Doch ich warne vor Hochmut. Wer sich für einen diplomierten HP-41 Benutzer hält, dem sei die eigenständige Analyse der Routine "IN" in Abschnitt 10H empfohlen. Zum Verständnis dieser Routine reicht es keineswegs, ein NNB ('Wickes', S. 125) zu sein. Ich selbst habe durch die Übersetzung sehr viel von Keith Jarett gelernt, obwohl ich zu der Zeit, als ich damit begann, schon lange mit den X-Funktionen arbeitete. Ich möchte dem Autor dafür und für die durch umfangreichen Schriftverkehr unterstützte Begleitung meiner Arbeit an dieser Stelle danken.

Unabweisbar auch diesmal ein Zugeständnis an die offenkundig weitverbreitete Lust am Lösen von Rätseln. Schon einmal, im Dearing nämlich, mußte ein Kryptographieprogramm dafür herhalten. Seinerzeit erprobt und für gut befunden, soll es abermals verwendet werden. Es kommt jetzt aber Lotterie ins Spiel! Aufmerksamkeit oder Scharfsinn sind weniger gefragt. Ein ganzes Kapitel, Kapitel 11, ist der Vorbereitung und Einkleidung der Aufgabe gewidmet.

Synthetischer Schnickschnack dient zur 'Verzierung' und tückischen 'Verriegelung' des die Lösung gestattenden Programms. Ich bin gespannt, wer als erster das richtige Los zieht. Die Aussichten stehen gut, besser als 1:100. Wer durchschaut das System und weiß, wie die Chancen, die Lösung zu finden, genau stehen?

Zum Schluß die Fehlerbörse. Sogar im 'Wickes' hat sich noch ein kleiner Fehler angefunden: S. 29, Zeile 10 v.u.: lokalen statt globalen.

Im 'Dearing' ist folgendes zu verbessern: 1 – 12: ALPHA-Modus statt USER-Modus; 7 – 9, Zeile 33: 'RCL 01' statt 'RCL 02'; 10 – 26: (oder mit Inkrement versehenen) statt (oder zu inkrementieren); 16 – 19, Zeile 25 ist unleserlich, sie lautet ' – ' (Minus); S. 210, Zeile 2: 'CLRALMS'.

Im 'Jarett I' wurden diese Druckfehler entdeckt: S. 15, Zeile 4 v. u.: ..., der dem, den Sie brauchen, ...; S. 33: in "RSHF" sind rechterhand M gegen O zu vertauschen.

H. D., im Februar 1985.

P.S.: Das 'Akrostichon' im 'Jarett I' ist sehr schnell entdeckt worden. Es steht auf S. 129. Ulrich Taubert aus Berlin war der erste, der es fand, und ist somit der Gewinner eines Exemplares des vorliegenden 'Jarett II'. Patrick Seemann aus Wettingen (Schweiz) und Heinz W. Kloos aus Ludwigshafen müssen sich mit der 'Ehre', zweiter und dritter gewesen zu sein, begnügen.

EINFÜHRUNG

Der X-Funktionen-Modul, welcher dem HP-41CX fest eingebaut ist und für den HP-41C und den CV als einsteckbare Erweiterung zur Verfügung steht, versieht Ihren HP-41 mit vielen neuen Funktionen[1]. Er enthält zugleich 127 zusätzliche Register für Speicherzwecke. Der Speicherumfang kann zudem durch Hinzufügung von (höchstens zwei) weiteren X-Memory-Moduln, deren jeder 238 Register enthält, vergrößert werden. Somit kann man sich, in Abhängigkeit von der Anzahl der dem X-Funktionen-Modul angefügten X-Memory-Moduln[2], zwischen 127 und 603 Register in Form eines 'erweiterten Speichers' verschaffen.

In Anlehnung an den Sprachgebrauch bei größeren Rechnern kann man den erweiterten Speicher als 'off-line'-Speicher bezeichnen, denn die in ihm enthaltenen Programme sind einem unmittelbaren Zugriff entzogen. Sie müssen erst in den Hauptspeicher – er entspricht dem 'on-line'-Speicher größerer Maschinen – übertragen werden, bevor sie sich ändern oder ausführen lassen. Der erweiterte Speicher kann überdies Daten aufnehmen.

Bezüglich des Umgangs mit dem erweiterten Speicher liegt durchaus ein Vergleich mit dem scheinbar so anders gearteten Kartenleser HP 82104A nahe. Auch der Kartenleser dient 'hauptamtlich' dazu, Programme und Daten außerhalb des Hauptspeichers abzulegen oder umgekehrt in ihn zu übertragen. Die wesentlichen Unterschiede zwischen dem erweiterten Speicher und dem Kartenleser sind diese:

Kartenleser	erweiterter Speicher
unbegrenzte Speicherfähigkeit	begrenzte Speicherfähigkeit (mindestens 127 höchstens 603 Register)
Karten für gesamten Rechnerinhalt, für Zustand, für Daten und für Programme	Dateien für Programme, für Daten und für Texte
Bedienung von Hand	Bedienung über das Tastenfeld oder durch ein Programm
Langzeitspeicherung	Speicherung anfällig gegenüber Stromausfall und "MEMORY LOST"-Bedingungen

[1] Der X-Funktionen-Modul enthält 47 Funktionen. Diese sowie 14 weitere X-Funktionen, also insgesamt 61 Stück, sind dem HP-41CX fest eingebaut.

[2] Sie dürfen zwei X-Memory-Moduln nicht in übereinanderliegende Steckplätze setzen, wohingegen der X-Funktionen-Modul an beliebiger Stelle sitzen darf. Beachten Sie genau die Anweisungen des zugehörigen Handbuches, damit Sie nur zulässige Belegungen der Steckplätze vornehmen. Wenn Sie lediglich einen X-Memory-Modul verwenden, sollten Sie ihn auf Platz 1 oder Platz 3 setzen.

Einem zusammengehörigen Satz von Magnetkarten entspricht im erweiterten Speicher ein Bereich, der *Datei* genannt wird. Geradeso, wie es verschiedene Kartenarten gibt (Programmkarten, Datenkarten usw.), hat man im erweiterten Speicher wohlunterschiedene Dateiarten, und zwar drei an der Zahl: Programmdateien, Datendateien und Textdateien, letztere auch ASCII[3]-Dateien genannt.

Eine Programmdatei enthält, wie der Name besagt, ein Programm. Eine Datendatei enthält ein oder mehrere Register zur Aufnahme von Zahlen. Eine Text- oder ASCII-Datei enthält eine Anzahl von Zeichenketten. Die genannten Dateiarten werden in den ersten drei Kapiteln untersucht und ihr Gebrauch an Hand von Beispielen erläutert.

'Wanzen' im HP-41

Im gewöhnlichen Sprachgebrauch ist eine Wanze ein widerliches Insekt, dem man mit dem Kammerjäger zu Leibe rückt. In neuerer Zeit werden auch Kleinstabhörgeräte mit diesem Namen belegt. Das ist naheliegend, denn einen Raum, dem verborgene Abhöreinrichtungen eingebaut worden sind, darf man unter Berufung auf all die mit Wanzen verbundenen unangenehmen Vorstellungen mit Fug und Recht 'verwanzt' nennen (und obendrein mit Abscheu auf die Urheber weisen). Ganz anders noch wird der Name dieses Ungeziefers im Zusammenhang mit den Betriebssystemen von Rechnern verwendet. Dort heißt jedes Verhalten von Funktionen des Rechners, das entweder unerwünscht und unerwartet ist oder von der Beschreibung im Handbuch abweicht, 'Wanze'. Bei dieser Namensgebung hat offenbar insbesondere die Vorstellung vom verborgenen Leben der Wanzen und der Schwierigkeit, sie zu beseitigen, Pate gestanden. Bedenkt man die unvorstellbar hohe Komplexität der internen Programme des X-Funktionen-ROMs (*R*ead *O*nly *M*emory), kann es nicht verwundern, daß es einigen solchen 'Wanzen' gelang, die i.a. sehr erfolgreiche Jagd der 'Kammerjäger von HP' lebend zu überstehen. Und so kann es denn unter ungewöhnlichen Umständen geschehen, daß die Ausführung einiger X-Funktionen zu unerfreulichen Nebenerscheinungen, gelegentlich sogar zu "MEMORY LOST" führt.

Wenn Sie entweder einen X-Funktionen-Modul oder einen Kartenleser, die vor September 1983 gefertigt wurden, haben, sollten Sie die Ausführungen dieses Abschnittes durchlesen, bevor Sie mit dem Studium des Buches beginnen. Andernfalls ist Ihr HP-41-System so gut wie wanzenfrei, und Sie können die folgenden Fehlerbeschreibungen überspringen.

Dieses Buch vermittelt Ihnen alle Kenntnisse, die Sie benötigen, um Problemen im Zusammenhang mit dem Fehlverhalten von X-Funktionen vorzubeugen. In einigen Fällen können Sie sogar Schäden, die durch einen 'Wanzenstich' verursacht wurden, nachträglich heilen. Der Zweck der gegenwärtigen kurzen Zusammenfassung ist es, Ihnen jetzt schon ausreichend viele Hinweise auf Wanzen zu geben, so daß Sie solange vor Ärger bewahrt bleiben, bis Sie in diesem Buch die genauen Beschreibungen zur Vermeidung bzw. Beseitigung von diesbezüglichen Mißgeschicken finden.

Sie sollten HP nicht für das Vorhandensein von Wanzen tadeln. Dort wird viel Zeit dafür aufgewendet, die Produkte vor der Beschickung des Marktes zu testen, doch es gibt keine

[3] Die Abkürzung rührt von '*A*merican *S*tandard *C*ode for *I*nformation *I*nterchange' her. Es handelt sich dabei um eine 7 Bits verwendende Kodierung von Zeichen. Im HP-41 werden die ASCII-verschlüsselten Zeichen in je einem Byte (1/7 eines Registers) untergebracht.

zeitlich begrenzte Prüfung, die *alle* Betriebsbedingungen erfaßt. Irgendwann muß die Kontrolle abgeschlossen und die Herstellung begonnen werden. Wer Vollkommenheit wollte, müßte ewig warten. HP entwickelt Rechner und Moduln mit weniger Problemen als jeder andere Hersteller. Die wenigen verbleibenden Wanzen sind gewissermaßen der Preis dafür, daß ein Produkt hinsichtlich Leistung und Wert im Wettbewerb steht.

In den frühen Versionen des X-Funktionen-Moduls gibt es drei Wanzen. Die in diesem Buch dargelegten Verfahren erlauben es, *alle Probleme* mit diesen Wanzen *auszuschalten.* Gewöhnlich werden Schritte zur vorbeugenden Fehlerbekämpfung beschrieben, doch für den am häufigsten auftretenden Fall wird auch eine Reparaturanweisung geliefert.

Der ersten X-Funktionen-Wanze geht man dadurch aus dem Wege, daß man die Funktionen 'SAVEP' (*save p*rogram) und 'PCLPS (*p*rogrammable *cl*ear *p*rogram*s*) *nicht* ausführt, wenn der Adreßzeiger auf eine Adresse in einem Anwendermodul außerhalb des Hauptspeichers weist. Die Einzelheiten dazu finden Sie in Abschnitt 1C genau beschrieben. Falls Sie das Programm "XF" (Abschnitt 10B) benutzen, um X-Funktionen tastensparend auszuführen, umgehen Sie das Problem automatisch.

Die zweite Wanze bringt Sie in eine gefährliche Lage, sobald Sie 'PURFL' (*pur*ge *file*) ausgeführt haben; ein falscher Befehl danach, und Sie haben auf keinen Teil des X-Memorys mehr Zugriff. Glücklicherweise versetzen uns synthetische Techniken (Abschnitt 10E) in die Lage, einen solchen Fehlschritt rückgängig zu machen. Andere einfachen Maßnahmen können helfen, das Problem von vornherein auszuschließen. Mehr über 'PURFL' finden Sie in Abschnitt 1F. Für den Fall, daß Ihnen der Begriff der 'Synthetischen Programmierung' hier das erste Mal begegnet, finden Sie in Abschnitt 10A eine ganz kurze Erläuterung dazu sowie Hinweise auf Quellen, an Hand derer Sie sich umfassend sachkundig machen können.

Die dritte Wanze tritt in Zusammenhang mit den Kartenleserfunktionen 'VER' und '7CLREG' in Erscheinung. Die Ausführung dieser Funktionen vermag den Inhalt des X-Memorys in unerwünschter Weise abzuändern. Solange Sie nicht die diesbezüglichen Einzelheiten im Anhang A nachgelesen haben, sollten Sie daher *unbedingt auf die Ausführung von 'VER' verzichten*, wenn sich ein X-Memory-Modul in den Einschubbuchsen 2 oder 4 befindet. Der X-Funktionen-Modul indessen kann gefahrlos an beliebige Stelle gesteckt werden. Eine kurze synthetische Routine, die in Abschnitt 10D vorgeführt wird, erlaubt die störungsfreie Benutzung von 'VER'.

Die ersten beiden der voranstehend beschriebenen Wanzen erscheinen nur in der Revision 1B des X-Funktionen-Moduls. Um herauszufinden, welche Version sich in Ihrem HP-41 befindet, führen Sie 'CAT 2' aus. Unter der abrollenden Auflistung der Peripherie-Funktionen erscheint dann eine der folgenden Meldungen:

– EXT FCN 1B
– EXT FCN 1C
– EXT FCN 2D (nur HP-41CX)

Falls Ihnen die Auflistung zu schnell abläuft und die entscheidende Meldung entschwindet, bevor Sie sie richtig wahrnehmen, können Sie mit 'R/S' unterbrechen und mit 'BST' zurücksetzen. Die Versionen des X-Funktionen-Moduls von 1C an aufwärts sind hinsichtlich der Funktionen 'SAVEP', 'PCLPS' und 'PURFL' wanzenfrei.

Die dritte Wanze sitzt im Kartenleser und zwar in allen Geräten, bei denen unter 'CAT 2' eine der Meldungen

CARD READER
CARD RDR 1D
CARD RDR 1E
CARD RDR 1F

erscheint. Nur jüngst hergestellte Geräte, deren Version mit 1G oder höher angegeben wird, sind bezüglich 'VER' entwanzt.

Der HP-41CX enthält in Zusammenhang mit den X-Funktionen überhaupt keine Wanze. Doch auch hier gilt es, 'VER' zu vermeiden, sofern Sie einen älteren Kartenleser verwenden und Abschnitt 10D noch nicht durchgearbeitet haben.

Stürzen wir uns nunmehr auf das X-Memory, und finden wir heraus, was es damit auf sich hat.

KAPITEL 1

Programme ins X-Memory übertragen

1A. Programmdateien anlegen

Die am häufigsten verwendeten X-Funktionen sind 'SAVEP' (*save* program) und 'GETP' (*get* program). 'SAVEP' überträgt ein Programm aus dem Hauptspeicher ins X-Memory und 'GETP' holt umgekehrt ein Programm aus dem X-Memory in den Hauptspeicher zurück. Als Übungsmaterial für diese Funktionen wählen wir das folgende Programm "JNX" (Barcode im Anhang D). Führen Sie 'GTO ..' aus, und tasten Sie

```
01♦LBL "JNX"   12 *       22 RCL 03   33 X≠0?     44 STO 01   54 RCL 02
02 STO 03      13 STO 07   23 /        34 /        45 RCL 05   55 ST/ 01
03 ABS         14 1/X      24 RCL 04   35 ST+ 02   46 STO 06   56 ST/ 04
04 5           15 STO 02   25 STO 05   36 RCL 00   47 RCL 07   57 ST/ 05
05 +           16 STO 04   26 *        37 2                    58 ST/ 06
06 X<>Y        17 STO 05   27 +        38 ST- 07   48♦LBL 01   59 RCL 05
07 STO 00                  28 STO 04   39 *        49 X≠0?     60 RCL 04
08 X<Y?        18♦LBL 00   29 RCL 07   40 RCL 07   50 GTO 00   61 RCL 06
09 X<>Y        19 RCL 05   30 4        41 X≠Y?     51 RCL 04   62 RCL 01
10 INT         20 CHS      31 /        42 GTO 01   52 ST- 02   63 END
11 4           21 RCL 07   32 FRC      43 RCL 04   53 RCL 01
                                                              80 Bytes
```

ein. Wenn Sie die Meldung "PACKING", gefolgt von "TRY AGAIN" bekommen, müssen Sie die Anzahl der Datenregister mit 'SIZE' herabsetzen, um mehr Programmregister zur Verfügung gestellt zu bekommen. Sie können stattdessen auch ein oder mehrere Programme, die entbehrlich sind oder bereits auf Karten oder Kassette dauerhaft aufgezeichnet wurden, mit 'CLP' löschen.

Wenn das Programm vollständig eingetastet ist, führen Sie 'GTO ..' aus, um zu 'PACK'en und ein abschließendes 'END' zu setzen. Dies ist insoweit von Bedeutung, als es den späteren Bedarf an Speicherplatz im X-Memory auf das Mindestmaß herabsetzt. Des weiteren ist es sinnvoll, mit 'GTO . _ _ _' auf die Zeilen 42 und 50 zu gehen und beide im RUN-Modus mit 'SST' (mindestens) einmal auszuführen. (Den Grund dafür erfahren Sie in Abschnitt 10F.) Damit Sie sicher sind, daß alles planmäßig verläuft, drücken Sie 'SST' solange, bis die jeweilige Zeile ('42 GTO 01' bzw. '50 GTO 00') erscheint; dann lassen Sie los. Sobald in der Anzeige wieder der X-Inhalt auftaucht, wissen Sie, daß die Zeile ausgeführt wurde. Sie können sich auch davon noch ausdrücklich überzeugen, indem Sie kurz in den PRGM-Modus schalten, um

'48 LBL 01' bzw. '18 LBL 00' vorzufinden. Das Programm ist jetzt zur Übertragung ins X-Memory vollständig vorbereitet. Gedulden Sie sich aber bitte damit noch bis zum Abschnitt 1C, wo dieser Übertrag ordnungsgemäß besprochen wird.

Beschreibung des Programms "JNX"

"JNX" berechnet die Werte der sog. Besselschen Funktionen erster Art, auch Zylinderfunktionen genannt, auf acht Nachkommastellen genau. Das Programm wird für einige Leser nützlich sein, anderen hingegen wenig sagen, abhängig von ihrer Vertrautheit mit höheren mathematischen Funktionen. Auf jeden Fall bildet es ein musterhaftes Beispiel für die Rechenkraft des HP-41, der ja vornehmlich zur Lösung mathematischer Aufgaben entworfen wurde. "JNX" wird noch später in diesem Buch, in Kapitel 6, eine Rolle spielen. Es empfiehlt sich daher, das Programm für die Zwecke fernerer Verwendung auf Magnetkarte oder Kassette aufzuzeichnen. Testen Sie zuvor, ob Sie alles fehlerfrei eingetastet haben: '2, ENTER↑, 1.2, XEQ "JNX"' muß das Ergebnis

$$J_2(1.2) = 1.593490184 \times 10^{-1}$$

liefern.

Wenn Sie an den Besselschen Funktionen nicht sonderlich interessiert sind, können Sie die folgenden Erläuterungen, die eine genaue Beschreibung der Arbeitsweise von "JNX" bilden, überschlagen und gleich zum nächsten Abschnitt, 1B, übergehen.

Zeilen-Analyse von "JNX"

Die Besselschen Funktionen erster Art n-ter Ordnung lauten

$$(1) \qquad J_n(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{2k+n}}{k!(k+n)!},$$

$n = 0,1,2,\ldots$. Sie gehorchen der rekursiven Beziehung

$$(2) \qquad J_{n-1}(x) = \frac{2n}{x} J_n(x) - J_{n+1}(x).$$

Diese liegt dem in "JNX" verwendeten Algorithmus zugrunde. Der Rechenvorgang für $J_n(x)$ wird mit 'n, ENTER↑, x, XEQ "JNX"' gestartet und beginnt mit der (äußerst rohen) Abschätzung

$$(3) \qquad J_m(x) \approx J_{m+1} \approx \frac{1}{2m},$$

worin $m = 2 \cdot \mathrm{INT}(\max(n, x+5))$ ist. Daraus werden der Reihe nach Abschätzungen für die Besselschen Funktionen $J_{i-1}(x)$, beginnend mit $i = m$ und absteigend bis $i = 1$, gemäß der Rekursionsformel (2) berechnet. Darunter befindet sich auch eine Abschätzung für den gesuchten Wert $J_n(x)$. Im Verlauf der Rechnung wird die Summe

$$(4) \qquad J_0(x) + 2J_2(x) + 2J_4(x) + \ldots + 2J_{m-2}(x) + 2J_m(x)$$

gebildet. Aus der Theorie der Besselschen Funktionen ist nun bekannt, daß sich

$$J_0(x) + 2\sum_{k=1}^{\infty} J_{2k}(x) = 1 \qquad (5)$$

ergeben muß. Daher kann die Summe (4), in der der Fehler durch Abbruch wegen der Wahl des relativ großen m gering ist, zur Normierung der Werte, welche sich für die $J_{i-1}(x)$ aus der fehlerbehafteten Anfangsabschätzung (3) ergeben, verwendet werden. [Anmerkung des Übersetzers für Fachkundige: Das Verfahren heißt Rückwärtsrekursion, stammt von J.C.P. Miller und beruht auf der Arbeit 'Computational aspects of three-term recurrence relations', SIAM Review 9, 24 – 82 (1967) von W. Gautschi.]

Wenden wir uns nun den Einzelheiten des Programms zu. Die Datenregister werden von "JNX" wie folgt belegt:

Register	Inhalt	
00	n	
01	$J_n(x)$	
02	Normierungsfaktor	
03	x	
04	$J_i(x)$	(Startwert 1/2m)
05	$J_{i+1}(x)$	(Startwert 1/2m)
06	$J_{n+1}(x)$	
07	2i	(Startwert 2m)

Die Zeilen 01 – 17 von "JNX" beschicken die Datenregister mit den Anfangswerten. Die mit 'LBL 00' beginnende Schleife berechnet die $J_{i-1}(x)$ aus den vorangegangenen Abschätzungen für $J_i(x)$ und $J_{i+1}(x)$. Das neue $J_{i-1}(x)$ ersetzt das alte $J_i(x)$ (Zeile 28), während $J_i(x)$ an den Platz des alten $J_{i+1}(x)$ wechselt (Zeilen 24 – 25). Die Zeilen 30 – 35 fügen der in R_{02} entstehenden Normierungssumme $2J_{i-1}(x)$ zu, und zwar genau dann, wenn i ungerade ist, der gebrochene Anteil von 2i/4 also 0.5 beträgt. Die Zeilen 37 – 38 dekrementieren 2i in R_{07} um 2 für den nächsten Schleifendurchlauf. Sobald i = n erreicht ist, werden die Werte von $J_i(x)$ und $J_{i+1}(x)$ in den Zeilen 43 – 46 nach R_{01} bzw. R_{06} gerettet. Solange $i \neq 0$ ist, wird die Schleife 'LBL 00' wiederholt (Zeilen 49 – 50), wodurch die $J_{i-1}(x)$ bis hinab zu $J_0(x)$ berechnet werden. Ist schließlich i = 0 erreicht, enthalten die Register R_{04} und R_{05} Abschätzungen für $J_0(x)$ bzw. $J_1(x)$. Die Zeilen 51 – 52 berichtigen den Fehler, der sich daraus ergeben hat, daß in der Schleife fälschlicherweise (vgl. (4)) auch der doppelte Wert von $J_0(x)$ addiert wurde (Zeile 35). Zum Schluß wird der in R_{02} entstandene Normierungsfaktor dazu benutzt, die vier zurückbleibenden fehlerbehafteten Abschätzungen für $J_0(x)$, $J_1(x)$, $J_n(x)$ und $J_{n+1}(x)$ zu berichtigen. Wenn das Programm anhält, liegen die richtigen Werte im Stapel:

Register	Inhalt
T	$J_1(x)$
Z	$J_0(x)$
Y	J_{n+1}
X	$J_n(x)$

[Anm. d. Übers: a) Das voranstehend analysierte Programm ist ein hervorragendes Beispiel für die kraftvolle Wirkung, die sich ergibt, wenn das Leistungsvermögen des HP-41 und die Erkenntnisse der Theorie vereint genutzt werden. b) Zeile 53 in "JNX" ist überflüssig.]

1B. Die Funktion 'EMDIR'

Bevor Sie das Programm "JNX" ins X-Memory bringen, ist es ratsam, dessen Zustand zu überprüfen. Führen Sie dazu

'XEQ "EMDIR"'

aus. Der Name dieser Funktion ist eine aus '*ex*tended *m*emory *dir*ectory' entstandene Abkürzung.

Wenn Sie den HP-41CX haben, steht Ihnen ein schnellerer Zugriff auf das Inhaltsverzeichnis des X-Memorys zur Verfügung:

'CAT 4'.

Falls Sie das X-Memory bislang noch nicht benutzt haben, bekommen Sie die Meldung "DIR EMPTY". Liegen jedoch schon Programme oder Daten im X-Memory, enthält das Verzeichnis Einträge, welche die entsprechenden 'Dateien' beschreiben. Jedem Eintrag kann man den Namen der Datei, ihren Typ und ihre Größe entnehmen.

Der Dateiname besteht aus bis zu sieben Zeichen. Die Dateiart wird durch einen einzigen Buchstaben gekennzeichnet: P für Programmdatei, D für Datendatei und A für ASCII-Datei (Textdatei). Die drei Dateiarten werden in den Kapiteln 1, 2 bzw. 3 besprochen. Unter Dateigröße versteht man die Anzahl der dieser Datei im X-Memory zugewiesenen Register. Tatsächlich werden für jede Datei noch zwei weitere Register, die man als Dateikopf bezeichnet, benötigt. Das erste dieser Register enthält den Dateinamen, das zweite den Dateityp, die Dateigröße und gewisse Zeigerwerte (Einzelheiten darüber werden in Abschnitt 10C besprochen). Will man also eine Datei von z.B. 12 Registern Länge anlegen, sei es eine Programm- oder eine beliebige andere Datei, vermindert man damit die Anzahl der freien Register des X-Memorys um 14.

Sie können die Anzahl der im X-Memory verfügbaren Register ermitteln, indem Sie die durch 'EMDIR' ausgelöste Anzeige des Verzeichnisses der Dateien bis zur letzten Datei durchlaufen lassen. Die Anzahl liegt anschließend im X-Register (der Stapel wird dabei angehoben). Sie ist stets um zwei kleiner als die Anzahl der tatsächlich unbenutzten Register, weil der Rechner selbständig jene zwei Register berücksichtigt, die für den Kopf der nächsten Datei benötigt werden. Wenn also 'EMDIR' im X-Register z.B. die Zahl 53 zurückläßt, gibt es insgesamt noch 55 freie Register im X-Memory, doch umfaßt die größte Datei, die Sie gerade noch anlegen können, 53 Register.

Die Funktion 'EMDIR' ähnelt der Funktion 'CAT 1', welche den Inhalt des Programmspeichers auflistet. Wegen ihrer Nützlichkeit ist es zu empfehlen, sie mit 'ASN' einer Taste Ihrer Wahl zuzuweisen.

Auf dem HP-41CX kann das Anzeigen des Inhaltes des X-Memorys unterbrochen und mit 'SST' und 'BST' einzelschrittweise vor- und rückwärts betrieben werden, so wie es auch im Katalog 1 möglich ist. Auf dem HP-41C oder CV läßt sich dies nicht vornehmen. Indessen kann man dort das Anzeigen 'einfrieren', indem man eine beliebige Taste (außer 'R/S' und

'ON') niedergedrückt hält. Nach dem Loslassen der Taste wird die Auflistung fortgesetzt; mit 'R/S' wird sie endgültig unterbrochen.

Auf dem HP-41CX gibt es zwei weitere Funktionen, die sich auf den Inhalt des X-Memorys beziehen. Die Funktion 'EMROOM' (*e*xtended *m*emory *room*) liefert die Anzahl der im X-Memory frei verfügbaren Register im X-Register ab, so wie es durch 'EMDIR' geschieht, wenn dabei die Auflistung nicht abgebrochen wird. Der Unterschied ist der, daß der X-Memory-Inhalt bei 'EMROOM' nicht angezeigt wird, wodurch die gesuchte Zahl wesentlich schneller ins X-Register gelangt. Daher ist 'EMROOM' für Programmzwecke geeigneter als 'EMDIR'. Ein Programm, das beispielsweise Dateien selbständig im X-Memory anlegen soll, prüft erst vermittels 'EMROOM', wie groß der verfügbare Platz ist, und paßt dann, falls erforderlich, die Größe der anzulegenden Dateien der ermittelten Registeranzahl an.

Die zweite nur im HP-41CX vorhandene X-Memory-Funktion heißt 'EMDIRX' (*e*xtendend *m*emory *dir*ectory: file designated by *X*). Sie sucht gemäß der im X-Register stehenden Zahl n die n-te Datei im X-Memory auf und legt ihren Namen ins Alpha-Register sowie ihren Typ in Form einer Zwei-Zeichen-Kette (PR für Programm, DA für Daten, AS für ASCII) ins X-Register (vgl. dazu auch die in Abschnitt 10C als Anm. a) gegebene Routine "DAT?").

1C. Die Verwendung von 'SAVEP' und 'GETP'

Das Programm "JNX" läßt sich dadurch ins X-Memory bringen, daß man einfach "JNX" ins Alpha-Register setzt und

'XEQ "SAVEP"'

ausführt. Für die Dauer der Ausführung verlischt die Anzeige, ausgenommen die Indikatoren der Fußzeile.

Viele anderen X-Funktionen arbeiten nach demselben Grundsatz: erst wird der Dateiname ins Alpha-Register geladen, dann die Funktion ausgeführt.

Das Ergebnis von 'SAVEP' können Sie sich vor Augen führen, indem Sie 'EMDIR' (auf dem HP-41CX auch 'CAT 4') ausführen. Sie sollten

"JNX P012"

erblicken. Wenn diese Meldung zu schnell entschwindet, wiederholen Sie 'EMDIR'. Auf dem HP-41C oder CV halten Sie dann eine Taste niedergedrückt, um die Anzeige verharren zu lassen; auf dem HP-41CX unterbrechen Sie mit 'R/S' und benutzen bei Bedarf 'SST' oder 'BST'.

Eine andere Möglichkeit, das Vorhandensein einer bestimmten Datei im X-Memory und zugleich ihre Größe festzustellen, bietet Ihnen die Funktion 'FLSIZE' (*file size*). Auch für sie wird der Dateiname ins Alpha-Register gesetzt und dann

'XEQ "FLSIZE"'

ausgeführt. Für "JNX" gelangt 12, das ist die Dateigröße des Programms, ins X-Register, wobei die beiden zusätzlichen Register für den Dateikopf unberücksichtigt bleiben.

'FLSIZE' liefert die Größe der benannten Datei natürlich nur dann, wenn die Datei existiert, andernfalls erscheint die Fehlermeldung "FL NOT FOUND". Die Funktion läßt sich auf alle drei Dateiarten (Programm, Daten, Text) in gleicher Weise ansetzen. Ist das Alpha-Register bei Aufruf von 'FLSIZE' leer, wird diese Funktion bezüglich der sog. 'Arbeitsdatei' ausgeführt. Arbeits- oder 'gegenwärtige' Datei ist die Datei, auf deren Namen Sie sich das letzte Mal mit einer X-Funktion, die zur Ausführung einen Dateinamen benötigt (z.B. 'SAVEP'), bezogen haben, oder diejenige, bei deren Anzeige die durch 'EMDIR' ausgelöste Auflistung abgebrochen wurde. Weitere Einzelheiten zum Begriff der Arbeitsdatei finden Sie im Abschnitt 1F.

WARNUNG (gilt nur für Version 1B): Stellen Sie vor Aufruf von 'SAVEP' sicher, daß der Rechner auf ein Programm im Hauptspeicher positioniert ist, nicht etwa auf ein in einem Anwender-Modul ansässiges ROM-Programm. Seien Sie also besonders achtsam, wenn ein solcher Modul (etwa Math Pac, Standard Pac o.ä.) auf einem Steckplatz sitzt. Beachten Sie, daß auch bei angeschlossenem Drucker oder eingestecktem HP-IL-Modul Zugang zu drei ROM-Programmen ("PRAXIS", "PRPLOT" und "PRPLOTP"), also Gefahr besteht. Das in Abschnitt 10B vorgeführte Programm "XF" gewährleistet, daß Sie sich stets im Hauptspeicher befinden, wenn Sie 'SAVEP' ausführen. Auf dem HP-41CX und bei Versionen des X-Funktionen-Moduls von 1C an aufwärts sind keine Vorsichtsmaßregeln vonnöten.

Um festzustellen, ob Sie sich in einem ROM-Programm befinden, brauchen Sie nur (im RUN-Modus) 'RTN' zu tasten. Wenn Sie anschließend in den PRGM-Modus übergehen und

"00"

erblicken, sind Sie tatsächlich in einem solchen. Mit 'CAT 1', 'GTO ..' oder 'GTO "ABC"' ("ABC" der Name eines im Hauptspeicher liegenden Programms) gelangen Sie zurück in den Hauptspeicher. Davon, daß Sie dort richtig angelangt sind, können Sie sich überzeugen, indem Sie abermals 'RTN' im RUN-Modus tasten. Wenn Sie jetzt in den PRGM-Modus übergehen, erblicken Sie

"00 REG lmn",

worin 'lmn' die Anzahl der freien Register des Hauptspeichers kundgibt. Sollten Sie dennoch versehentlich 'SAVEP' außerhalb des Hauptspeichers ausgeführt haben, jagen Sie sofort ein 'PURFL' (Abschnitt 1F) hinterher. Die Programmdatei, die 'SAVEP' unter diesen Bedingungen anlegt, ist nämlich unbenutzbar, und darüber hinaus kann sogar das Tastenfeld des Rechners lahmgelegt werden, wenn man sie in den Hauptspeicher zu rufen versucht. Wollen Sie all dessen ungeachtet ein ROM-Programm ins X-Memory bringen, müssen Sie es erst mit 'COPY' in den Hauptspeicher holen. Von dort aus ist dann die Übertragung in das X-Memory möglich.

Die Warnung, sich davor zu hüten, außerhalb des Hauptspeichers zu hantieren, gilt in noch stärkerem Maße hinsichtlich 'PCLPS' (Abschnitt 1E). Bei dieser X-Funktion wird ein Fehlschritt gnadenlos mit dem gefürchteten "MEMORY LOST" bestraft. Doch auch hier kann man die Warnung in den Wind schlagen, wenn man mit einer X-Funktionen-Version von 1C an aufwärts oder dem HP-41CX arbeitet.

Wir wollen jetzt das Programm "JNX" aus dem X-Memory zurückholen. Setzen Sie dazu "JNX" ins Alpha-Register, führen Sie ein 'GTO ..' aus und dann 'XEQ "GETP"'. Unter der Voraussetzung, daß genügend Programmspeicherplatz vorhanden war, haben Sie jetzt "JNX" zweimal im Hauptspeicher liegen. Auf 'CAT 1' hin sehen Sie 'LBL "JNX", END, LBL "JNX", .END. REG lmn' als letzte Einträge des Katalogs 1. Dies zeigt, daß 'GETP' das angesprochene Programm holt und zwischen das letzte 'END' und das ständige '.END.' legt, und zwar auch dann, wenn dort bereits ein Programm läge, welch letzteres in einem solchen Fall einfach überschrieben würde. Das heißt insbesondere, Sie hätten "JNX" beim Ablauf von 'CAT 1' nur einmal vorgefunden, wenn es versäumt worden wäre, das ursprüngliche "JNX" durch 'GTO ..' mit einem schützenden 'END' zu versehen: das mit 'GETP' geholte "JNX" hätte das vorhandene "JNX" überschrieben; nach wie vor wäre dann "JNX" nur einmal im Hauptspeicher zu finden gewesen.

Weist man 'SAVEP' und 'GETP' Tasten zu, hat man eine äußerst bequeme Möglichkeit zur Hand, Programme auszulagern oder einzulesen, wie sie zuvor nur der Kartenleser mit dem Beschreiben und Einlesen von Magnetkarten bot. Man macht davon günstig Gebrauch, wenn man beispielsweise ein Programm im Katalog 1 nach unten bringen will, um es schneller edieren und 'PACK'en zu können. (Das Einfügen von Befehlen in ein Programm, welches weit oben im Katalog 1 liegt, und das 'PACK'en eines solchen Programms nehmen beträchtlich Zeit in Anspruch, weil alle nachfolgenden Programme bei diesen Vorgängen mit verschoben werden müssen.) Dazu führt man 'SAVEP' aus, löscht das Programm im Hauptspeicher und bringt es anschließend mit 'GETP' an den Schluß des Katalogs 1. Techniken wie diese sind durchaus nützlich, doch die volle Kraft entfalten die Funktionen 'SAVEP' und 'GETP' erst dann, wenn sie ihre Arbeit als Befehle eines Programms verrichten. Bevor wir uns aber solch höherer Kunst widmen, müssen Sie noch ein wenig mehr über 'SAVEP' und 'GETP' wissen. (Anm. d. Übers.: Beachten Sie in diesem Zusammenhang die Routine "DATROT" am Ende von Abschnitt 10F, wo 'GETP' und 'SAVEP' dazu dienen, die Reihenfolge der Programmdateien im X-Memory völlig programmgesteuert zu verändern.)

1D. Höherwertige Eigenschaften von 'SAVEP'

Die vorangegangenen Beispiele könnten Voreilige zu der Annahme verführen, daß Programme nur unter ihrem eigenen Namen ins X-Memory geschrieben werden können. Mit ein wenig mehr Aufwand ist es aber möglich, ein Programm unter einem beliebigen anderen Dateinamen abzulegen (vgl. Kapitel 5, wo diese Möglichkeit sinnvoll ausgenutzt wird). Statt einfach nur den Programmnamen ins Alpha-Register zu setzen (tatsächlich kann man übrigens dafür auch jede im Katalog 1 enthaltene globale Marke, die sich innerhalb des zu übertragenden Programms befindet, nehmen), vervollständigt man dazu den Alpha-Register-Eintrag durch einen Dateinamen freier Wahl, abgesetzt gegen den voranstehenden Programmnamen durch ein trennendes Komma. Will man das gerade gegenwärtige Programm übertragen, reicht es, dessen Namen zu unterdrücken und das Alpha-Register lediglich mit einem Komma, gefolgt vom Dateinamen, zu laden. (Unter '*gegenwärtigem Programm*' soll stets dasjenige verstanden werden, das sich Ihnen zeigt, wenn Sie in den PRGM-Modus schalten. Es ist gewöhnlich das Programm, welches Sie als letztes haben laufen lassen, es sei denn, Sie haben inzwischen 'GTO ..' oder 'CAT 1' ausgeführt, beides Eingriffe, die Sie (i.a.) in einen anderen Teil des Programmspeichers bringen.)

Die für die Ausführung von 'SAVEP' zulässigen Alpha-Register-Inhalte haben diese Gestalt:

Alpha-Register-Inhalt	Wirkung
"PROGRAMMNAME"	das Programm, welches eine globale Marke dieses Namens enthält, wird unter demselben Namen ins X-Memory gelegt
"PROGRAMMNAME,DATEINAME"	das benannte Programm wird unter dem Namen der Datei im X-Memory abgelegt (Vorsicht: hinter dem Komma darf sich keine Leerstelle befinden, es sei denn, Sie haben 'SPACE' bewußt als erstes Zeichen des Dateinamens vorgesehen)
",DATEINAME"	das gegenwärtige Programm wird unter dem Namen der Datei im X-Memory abgelegt.

Bemerkung: Kommata sind nicht als Bestandteile eines Dateinamens zulässig, damit dessen Rolle als Trennzeichen eindeutig bleibt. Fernerhin dürfen Dateinamen nicht die Länge 7 überschreiten (zusätzlich benutzte Zeichen werden abgeschnitten).

1E. Höherwertige Eigenschaften von 'GETP', einschließlich 'GETSUB' und 'PCLPS'

Im Gegensatz zu 'SAVEP' arbeitet die Funktion 'GETP' anders, wenn sie als Programmbefehl und nicht über das Tastenfeld ausgeführt wird. In beiden Fällen allerdings bringt 'GETP' das durch den Inhalt des Alpha-Registers bestimmte Programm in den Hauptspeicher, und zwar, wie schon erwähnt, hinter das letzte 'END' des Katalogs 1, doch vor das ständige '.END.'. So wie in den anderen Fällen des Einlesens ganzer Programme (Kartenleser, Kassette, Lesestift) werden auch hier Tastenzuweisungen von globalen Marken des einzulesenden Programms nur dann berücksichtigt, wenn während des Einlesevorgangs der USER-Modus eingeschaltet ist (zuvor vorhandene Tastenzuweisungen werden dabei überschrieben, wenn sie zu Konflikten Anlaß geben würden).

Wenn man 'GETP' über das Tastenfeld ausgeführt hat, steht der Rechner auf der ersten Zeile des eingelesenen Programms. Das ist eine vernünftige Einrichtung, denn man kann das Programm sogleich mit 'R/S' starten oder auch in den PRGM-Modus überwechseln, um es zu betrachten oder abzuändern.

Wenn 'GETP' dagegen durch ein Programm A befohlen wird, gelangt das Programm B, welches einzulesen bestimmt ist, zwar in den Hauptspeicher; unmittelbar nach dem Einlesen aber fährt der Rechner mit der Ausführung des Programms A fort, ausgenommen A war das letzte Programm im Speicher (als *'letztes Programm'* bezeichnet man das Programm, welches das ständige '.END.' als letzte Zeile enthält – sind alle Programme durch ein 'END' abgeschlossen, ist das letzte Programm ein *'leeres Programm'*). In diesem Fall wird A folgerichtig durch B überschrieben. Damit bricht natürlich die Programmausführung von A ab; stattdessen fährt der Rechner ohne innezuhalten mit der Ausführung von B, beginnend auf

dessen erster Zeile, fort (diese Tatsache wird bei der sog. Segmentierung von Programmen, s.u., ausgenutzt).

Wegen der beschriebenen Eigenschaften von 'GETP' muß man die Ausführung dieser Funktion durch ein Programm sorgfältig planen, damit wichtige Programme nicht unbeabsichtigt überschrieben werden. Versäumen Sie daher nicht, in Programmdokumentationen, die Sie hier und da werden anlegen müssen, auf ein u.U. notwendig auszuführendes 'GTO ..', welche das schützende 'END' erzeugt, hinzuweisen. Achten Sie andererseits darauf, 'GTO ..' nicht leichtfertig zu benutzen. Wenn Sie sich nämlich nicht im leeren Programm, welches das bloße '.END.' enthält, befinden und Ihren Programmschritten nicht wenigstens schon eine globale Marke beigefügt ist, erzeugen verschwenderisch ausgeführte 'GTO ..'s einen lästig wuchernden Katalog 1 voll 'kopfloser' 'END's, denen nur umständlich beizukommen ist.

Gerade die gefahrbringende Eigenschaft von 'GETP', letzte Programme im Speicher zu überschreiben, kann äußerst gewinnbringend verwendet werden, wenn sehr große Programme, insbesondere solche, die insgesamt nicht mehr in den Arbeitsspeicher passen, abzuarbeiten sind. Entweder legt man alle zugehörigen Unterprogramme in das X-Memory und ruft sie gemäß der Reihenfolge ihres Bedarfs mit 'GETP' in den Hauptspeicher, wobei jedes Unterprogramm das vorangehend eingelesene überschreibt. Diese mit *'Überlappung'* bezeichnete Technik befreit auf handliche Weise von Engpässen beim Speicherplatzbedarf. Oder man zerlegt ein Programm, soweit es sein Aufbau zuläßt, in Teilstücke, deren jedes als letzten Befehl ein 'GETP' enthält, welches auf das nächste im X-Memory wartende Teilstück zielt, eine unter der Bezeichnung *'Segmentierung'* bekannte Methode.

Die bezüglich 'GETP' zu beachtende Vorsicht mag Sie dazu verführen, stattdessen auf das günstiger scheinende 'GETSUB' (*get sub*routine) zurückzugreifen. Diese Funktion arbeitet nämlich geradeso, als ob man 'GTO ..', gefolgt von 'GETP' ausführt. Der Unterschied ist der, daß ein neues 'END' 'ohne Ansehen des Programms' erzeugt wird, also auch dann, wenn das letzte Programm leer ist. Das kann sich sehr nachteilig auswirken. Wenn Sie es sich nämlich zur Gewohnheit machen, die Bequemlichkeit der Umsicht vorzuziehen, werden Sie Ihren Katalog 1 bald mit höchst überflüssigen 'nackten' 'END's vollgestopft finden. Man kann sich ihrer auf folgende etwas umständliche Weise entledigen: Der Rechner wird auf ein bloßes 'END', dem keine globale Marke vorangeht, gesetzt, indem man 'CAT 1' ablaufen läßt und unterbricht, sobald dieses 'END' in der Anzeige vorgewiesen wird. Dann kann man es löschen, indem man in den PRGM-Modus überwechselt und die Korrekturtaste betätigt. Man kann stattdessen auch 'XEQ "CLP"' ausführen und die dadurch ausgelöste Aufforderung zur Eingabe des Programmnamens mit 'ALPHA, ALPHA' beantworten; dann wird das gegenwärtige Programm – hier nur aus dem unnützen 'END' bestehend – gelöscht. Liegen mehrere wertlose 'END's unmittelbar hintereinander im Katalog 1, unterbricht man denselben beim ersten dieser 'END's, geht in den PRGM-Modus und drückt dann abwechselnd die Korrekturtaste und 'SST', bis kein 'END' mehr auftaucht. Danach ist die gesamte Gruppe der überzähligen 'END's getilgt. Liegt eine solche den Programmspeicher belastende Gruppe ganz am Ende des Katalogs 1, kann man ihr schneller beikommen: man setzt den Rechner wie beschrieben auf das erste 'END' und führt dann im RUN-Modus bei leerem Alpha-Register 'XEQ "PCLPS"' aus.

Ob 'GETSUB' vom Tastenfeld aus oder durch ein Programm ausgeführt wird, die Wirkung ist in beiden Fällen dieselbe: das letzte Programm, ob leer oder nicht, erhält ein abschließendes 'END', und das durch den Inhalt des Alpha-Registers bestimmte Programm wird in den Hauptspeicher gelesen, wobei das '.END.' zu seiner letzten Zeile gerät. Die Programmausfüh-

rung wird in keinem Fall an das neu eingelesene Programm weitergereicht.

Aus den voranstehenden Erläuterungen geht hervor, daß 'GETSUB' einzig dann zu Recht benutzt wird, wenn das letzte Programm 'offen' ist, also '.END.' als letzte Zeile besitzt, und nicht überschrieben werden darf. 'GETSUB' ist mithin regelmäßig dort angezeigt, wo mehrere Programmdateien nacheinander und dauerhaft in den Hauptspeicher gebeten werden sollen und das jeweils letzte Programm mit '.END.' schließt. Man geht mit 'GETSUB' der Erzeugung von 'END's über das Tastenfeld aus dem Wege. Umgekehrt ist zu beachten, daß man mit 'GETP' die richtige Wahl trifft, wenn das letzte Programm abgeschlossen oder offen aber nicht der Bewahrung wert ist.

HP-41 Freunden, die mit dem Kartenleser vertraut sind, ist ohne Zweifel aufgefallen, daß 'GETP' genau dem Einlesen eines Satzes von Programmkarten entspricht, 'GETSUB' hingegen der nahe verwandten Kartenleserfunktion 'RSUB' gegenüber einen kleinen Unterschied aufweist: 'RSUB' schließt ein offenes letztes Programm L (auch ein leeres) nur dann von sich aus mit einem 'END' ab und setzt das von 'RSUB' angeforderte Programm dahinter, wenn Sie sich beim Aufruf von 'RSUB' in L selbst befinden oder das gerade wirksame 'RSUB' ein Befehl innerhalb von L ist, andernfalls wird wie bei 'GETP' das letzte Programm überschrieben.

Wenn Sie alle im Programmspeicher befindlichen Programme zugleich löschen wollen, können Sie sich dafür der X-Funktion 'PCLPS' (*p*rogrammable *cl*ear *p*rogram*s*) bedienen. Sie brauchen dazu nur den Namen des ersten Programms ins Alpha-Register zu legen, bevor Sie 'PCLPS' aufrufen. Die Funktion löscht grundsätzlich das Programm, welches durch den Inhalt des Alpha-Registers bestimmt ist, sowie alle darauf im Katalog 1 folgenden Programme. Ist das Alpha-Register leer, beginnt die Löschung mit dem gegenwärtigen Programm. Diese Gesetzmäßigkeiten gelten sowohl für den Fall, daß man 'PCLPS' über das Tastenfeld ausführt, als auch für den, daß die Funktion als Programmbefehl wirksam wird. Liegt 'PCLPS' in einem Programm P, das sich oberhalb des zu löschenden Blockes von Programmen befindet, wird die Programmtätigkeit in P wiederaufgenommen, sobald 'PCLPS' sich seiner Aufgabe entledigt hat.

WARNUNG (gilt nur für Version 1B): Ist der Rechner auf ein ROM-Programm außerhalb des Hauptspeichers positioniert und das Alpha-Register *nicht* leer, bedient Sie 'PCLPS' erbarmungslos mit "MEMORY LOST", ein Drama, dem eine Unheil verkündende Verzögerung von mehreren Sekunden vorangeht. Beachten Sie in diesem Zusammenhang noch einmal die in Abschnitt 1C ausgesprochene Warnung bezüglich 'SAVEP'. Das Programm "XF" (Abschnitt 10B) enthebt Sie späterhin wie bei 'SAVEP' aller Sorgen.

Scharfsinnige Leser, die an ein 'GETP' denken, welches das Programm überschreibt, in dem es selbst enthalten ist, vermuten gewiß längst, welch einen Sonderfall es bei 'PCLPS' zu beachten gibt. 'PCLPS' löscht, wie beschrieben, das benannte und alle folgenden Programme. Arbeitet 'PCLPS' als Befehl eines Programms, welches selbst dem zur Löschung vorgesehenen Block von Programmen angehört, schließt die Programmtätigkeit mit der Ausführung des '.END.' des leeren Programms, das im Anschluß an die Löschung entsteht und auf jene Stelle im Programmspeicher zu liegen kommt, wo vorher der gelöschte Block begann. Arbeitet 'PCLPS' jedoch *als Befehl eines Unterprogramms*, wird das genannte '.END.' folgerichtig mit der Wirkung eines 'RTN' ausgeführt. Damit gelangt der Logik des Betriebssystems gemäß die Programmkontrolle zurück an das rufende Programm, welches, wenn Sie nicht sorgfältig genug planen, dem durch 'PCLPS' betroffenen Block angehört. Ist das rufende Programm nun tatsächlich ebenfalls der Löschung zum Opfer gefallen, gerät der die Programmtätigkeit

steuernde Adreßzeiger in den hinter dem '.END.' liegenden Bereich des Hauptspeichers. Diese Unfallmöglichkeit besteht, wie man leicht einsieht, auch bei völlig einwandfrei arbeitenden Versionen des X-Funktionen-Moduls. Auf Abenteuer versessene Synthetiker werden fraglos Geschmack daran finden, so mühelos verbotenes Gelände betreten zu können, doch sollte man unbeabsichtigte Verirrungen des Adreßzeigers vermeiden. Wenn Sie einmal auf die beschriebene Weise verunglückt sind, müssen Sie den Programmlauf mit 'R/S' unterbrechen (möglicherweise hält der Rechner auch von selbst mit irgendeiner Fehlermeldung an) und mit 'CAT 1' oder 'GTO ..' in den Programmspeicherbereich zurückgehen.

Anm. d. Übers.: Man kann die Wirkungsweisen von 'GETP', 'GETSUB' und 'PCLPS' mit dem folgenden (u.a. von Clifford Stern vorgeschlagenen) kleinen Dienstprogramm "SPR" deutlich vor Augen führen. Nehmen Sie an, Sie hätten im oberen Teil Ihres Programmspeichers einige Programme liegen, deren ständige Anwesenheit erwünscht ist, während weiter unten Wechselgäste einquartiert sind, die gelegentlich allesamt den Speicher verlassen sollen. Dann läßt sich dies völlig programmgesteuert erzwingen. Legen Sie zunächst zwischen die beiden Programmgruppen die Routine '*Sp*eicher *r*äumen'

```
01♦LBL "SPR"
02 "$"
03 PCLPS
04 GETP
05 GETSUB
06 PCLPS
07 END
```

20 Bytes

und unmittelbar dahinter das Scheinprogramm

```
01♦LBL "$"
02 END
```

8 Bytes

"$" muß außerdem noch im X-Memory untergebracht werden. Durch den doppelten Aufruf von 'PCLPS' und die Verwendung sowohl von 'GETP' als auch von 'GETSUB' in genau dieser Reihenfolge wird allen Umständen beim Gebrauch von "SPR" Rechnung getragen. Und das so: Das erste 'PCLPS' fegt den hinter "SPR" liegenden Speicherbereich leer. Anschließend wird "$" wieder geladen, denn man braucht es ja für den nächsten Aufruf von "SPR". Da es durch ein folgendes anderweitiges Programm-Laden jedoch überschrieben würde, muß ein 'GETSUB' dafür sorgen, daß es ein schützendes 'END' erhält. Ein 'GETSUB' gleich an erster Stelle wäre dazu ungeeignet, weil dieses bei jedem Aufruf von "SPR" ein überflüssiges alleinstehendes 'END' in den Speicher brächte und dennoch kein solches 'END' hinter "$" setzte, wo es gebraucht wird. Das durch 'GETSUB' erzeugte überzählige zweite Exemplar von "$" wird zuletzt durch das zweite 'PCLPS' wieder entfernt.

1F. Das Löschen von Dateien im X-Memory

So wie man mit dem Befehl 'CLP' ein einzelnes Programm im Hauptspeicher tilgen kann, läßt sich mit der X-Funktion 'PURFL' (*purge file*) eine einzelne Datei im X-Memory löschen. Die dazu im Alpha-Register zu benennende Datei kann eine Programm-, eine Daten- oder eine Textdatei sein. Im Anschluß an die Löschung der Datei werden die restlichen Inhalte des X-Memorys vom Rechner selbständig zusammengeschoben, so daß keine Lücken durch das Löschen von Dateien zurückbleiben. Dieser Vorgang entspricht in etwa dem im Anschluß an 'CLP' automatisch vorgenommenen 'PACK'en.

WARNUNG (gilt nur für Version 1B): Die Funktion 'PURFL' hat eine sehr unangenehme Eigenschaft: Sobald sie ihre Aufgabe erfüllt hat, gibt es *keine Arbeitsdatei mehr*. Wird eine Arbeitsdatei nicht sofort wieder bestimmt, kann es geschehen, daß der Zugriff auf den gesamten Inhalt des X-Memorys verloren geht. Jede X-Funktion, die zur Ausführung einer Arbeitsdatei bedarf, eine solche aber zu dem Zeitpunkt, zu welchem sie aufgerufen wird, nicht vorfindet, *zerstört* nämlich *das Inhaltsverzeichnis des X-Memorys*. In Abschnitt 10E werden besondere Verfahren, die die Reparatur dieses Mißgeschicks zulassen, beschrieben.

Bemerkung: Die zerstörerische Tätigkeit der 'PURFFL'-Wanze in Version 1B kann in einem ganz besonderen Fall von Nutzen sein: Die Befehlsfolge 'PURFL, RCLPT' bietet eine schnelle und bequeme Möglichkeit, das gesamte X-Memory zu löschen, ohne den Hauptspeicher zu behelligen [vgl. dazu auch die in Abschnitt 10C als Anm. b) gegebene Routine "XML"]. Gerade deswegen darf aber diese Befehlsfolge nicht in Programmen auftauchen.

Die Arbeitsdatei, auch gegenwärtige Datei genannt, ist die zuletzt benutzte oder erzeugte Datei, es sei denn, die Funktion 'EMDIR' (auf dem HP-41CX auch 'CAT 4') ist inzwischen aufgerufen worden. Wenn man nämlich auf dem HP-41C oder CV mit dieser Funktion das Inhaltsverzeichnis des X-Memorys 'durchblättert', wird stets die zuletzt angezeigte Datei zur Arbeitsdatei, und zwar unabhängig davon, ob man das Verzeichnis bis zum Ende abrollen läßt oder den Ablauf unterbricht. Auf dem HP-41CX hingegen wechselt die Arbeitsdatei nur dann, wenn man das Verzeichnis 'unterwegs' anhält. Läßt man es jedoch frei durchlaufen, ohne mit 'R/S' einzugreifen, bleibt die Datei Arbeitsdatei, die diese Sonderstellung bereits vor dem Aufruf von 'EMDIR' innehatte.

Der Begriff der gegenwärtigen Datei (X-Memory) ist in naheliegender Weise dem Begriff des *gegenwärtigen Programms* (Hauptspeicher) nachgebildet. Das gegenwärtige Programm ist dasjenige, dessen Zeilen in der Anzeige erscheinen, sobald man in den PRGM-Modus überwechselt. Alle programmbezogenen Befehle, die nicht der gesonderten Angabe einer globalen Marke bedürfen, arbeiten bezüglich des gegenwärtigen Programms. Beispielsweise beziehen sich Befehle wie 'GTO .001', 'LIST 999', 'DEL 005' oder 'CLP' ohne Angabe eines Programmnamens ('ALPHA, ALPHA') auf das gegenwärtige Programm. Es kann im Hauptspeicher auf zweierlei Weise aufgesucht werden. Gewöhnlich ist es das Programm, auf das man zuletzt mit 'GTO "NAME"' oder 'XEQ "NAME"' zugegriffen hat. Wird jedoch – das ist die zweite Möglichkeit – 'CAT 1' ausgeführt, so gerät dasjenige Programm zum gegenwärtigen Programm, bei dem die Katalogdurchsicht unterbrochen wird. Man kann also durch 'gefühlvolles Bremsen' den Katalog 1 so anhalten, daß man in ein bestimmtes Programm gelangt, ohne dessen Namen buchstabierend hinter ein 'GTO' setzen zu müssen, vorausgesetzt natürlich, daß das gesuchte Programm eine globale Marke enthält. Ist das nicht der Fall, bleibt überhaupt nur die Möglichkeit, mit 'CAT 1' zum 'END' des markenlosen Programms vorzustoßen.

So wie man also im Hauptspeicher mit Hilfe von 'CAT 1' umherwandern kann, läßt sich im X-Memory 'EMDIR' dazu verwenden, durch rechtzeitiges Betätigen von 'R/S' eine bestimmte Datei als Arbeitsdatei auszuwählen. Ansonsten wird die Datei zur Arbeitsdatei, die man neu anlegt, oder diejenige, die man mit einem Befehl anspricht, welcher den Benutzer zwingt, ihren Namen im Alpha-Register bereitzustellen.

Nehmen Sie jetzt an, Sie hätten gerade 'PURFL' ausgeführt, doch anschließend keine neue Arbeitsdatei festgelegt. Wenn Sie anschließend bei leerem Alpha-Register eine X-Funktion aufrufen, die eine Arbeitsdatei verlangt, z.B. 'FLSIZE', erhalten Sie die Meldung "FL NOT FOUND". Das scheint soweit in Ordnung zu sein, denn es gibt ja in diesem Moment keine Arbeitsdatei, deren Größe feststellbar wäre. Aber wenn Sie nun – Version 1B und nichtleeres X-Memory vorausgesetzt – 'EMDIR' aufrufen, trifft Sie der Schlag: "DIR EMPTY"! Der gesamte Inhalt des X-Memorys ist dahin.

Es gibt mehrere Wege, die '1B-Geschädigte' beschreiten können, um das Problem mit 'PURFL' zu umgehen. Zunächst einmal sollten Sie es sich zur Gewohnheit machen, stets unmittelbar nach der Benutzung von 'PURFL' eine neue Arbeitsdatei durch Aufruf von 'EMDIR' oder anderweitig festzulegen. Diese Vorsichtsmaßregel muß insbesondere auch dann beachtet werden, wenn 'PURFL' als Programmbefehl verwendet wird. Fernerhin ist es empfehlenswert, sich grundsätzlich bei jedem Aufruf einer X-Funktion, die eine Arbeitsdatei verlangt, zu vergewissern, ob das Alpha-Register auch wirklich entsprechend beschickt ist. Sofern eine im Programm benutzte X-Funktion sowohl eine benannte Datei als auch die gegenwärtige Datei als Arbeitsdatei ansehen kann, läßt sich mit den Befehlen 'ALENG, 1/X' (vgl. Abschnitt 4B) sicherstellen, daß ein leeres Alpha-Register rechtzeitig entdeckt wird. Schließlich erlaubt das in Abschnitt 10E vorgestellte Programm "PFF" eine Wiederherstellung des beschädigten X-Memory-Verzeichnisses. Jenes Programm enthält einige synthetischen Befehle, die nicht auf herkömmliche Weise eingetastet werden können. Daher wird Ihnen im Anhang D der Barcode für dieses Programm zur Verfügung gestellt. – Und hier noch einmal: In Modul-Revisionen von 1C an aufwärts und im HP-41CX gibt es keine Schwierigkeiten mit 'PURFL'.

Eine Kleinigkeit bezüglich der Wirkung von 'SAVEP' bleibt anzumerken übrig: Sobald man die neue Fassung eines Programms, das bereits im X-Memory liegt, dorthin bringt, wird die alte Datei automatisch gelöscht und die neue *ans Ende des X-Memorys* gesetzt. Beachten Sie dies, und verfallen Sie nicht in Panik, wenn Sie Ihr Programm am Anfang des Verzeichnisses erwarten und es nicht gleich dort erscheint. Natürlich können Sie auch umstandslos zwei Fassungen eines Programms oder gar noch mehr ins X-Memory legen, indem Sie einfach verschiedene Dateinamen (Abschnitt 1D) wählen. (Anm. d. Übers.: Mit der am Ende von Abschnitt 10F vorgeschlagenen Routine "DATROT" können Sie die Reihenfolge der Programmdateien im X-Memory programmgesteuert verändern.)

KAPITEL 2

Daten ins X-Memory übertragen

2A. Der Aufbau von Dateien

Das Übertragen von Daten ins X-Memory verlangt einige Handgriffe mehr als das Übertragen von Programmen. Während 'SAVEP' die erforderlichen Dateien im X-Memory grundsätzlich selbständig anlegt, muß der Benutzer, der Daten dorthin bringen will, vorher eine (leere) Datendatei, die die Daten aufnehmen kann, mit einem getrennten Befehl erzeugen. Abbildung 2.1 zeigt die für alle drei Dateiarten (Programme, Daten, Texte) gültige Gliederung der Register im X-Memory. Sie unterscheidet sich bei den drei Dateiarten lediglich in der grenzbe- bzw. grenzmißachtenden Belegung von Registern, die wie im Hauptspeicher je sieben Bytes umfassen.

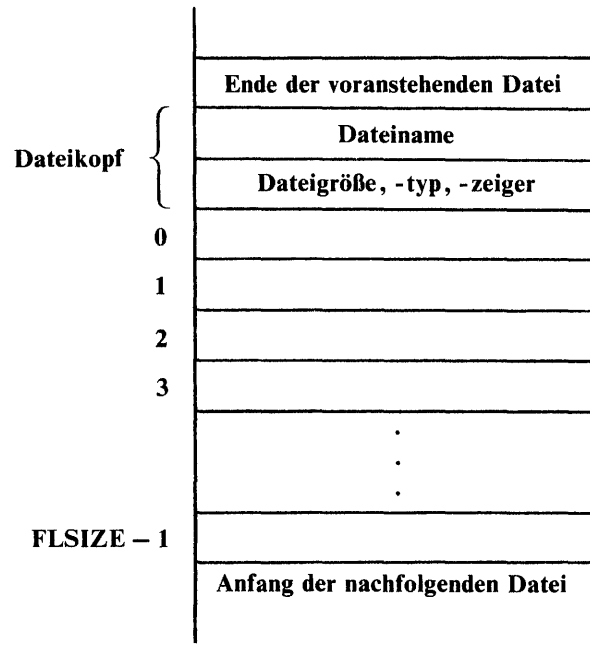


Abbildung 2.1: Registergliederung einer Datei im X-Memory. Zugriff auf die Inhalte der Dateiköpfe erfolgt mit 'FLSIZE', 'EMDIR' usw.

Wir wollen uns mit der folgenden kurzen Routine, die die Datenregister mit ihren eigenen Adressen lädt, Übungsmaterial für dieses Kapitel beschaffen. R_{00} wird von ihr mit 0 beschickt, R_{01} mit 1 usw., bis die Meldung "NONEXISTENT" davon kündet, daß das nächste Register nicht mehr vorhanden ist.

```
01 LBL "LADEREG"
02 1                 ⎫
03 ENTER↑            ⎪
04 ENTER↑            ⎬  lädt den Stapel mit Einsen
05 ENTER↑            ⎪
06 CLX               ⎫
07 LBL 01            ⎭  Schleifenbeginn mit 0 in X
08 STO IND X            legt n nach Rn (n = 0,1,2,...)
09 +                    inkrementiert X um 1
10 GTO 01               zurück an den Schleifenanfang
11 END                                          25 Bytes
```

2B. Die Funktion 'SAVERX' und der Begriff der 'Arbeitsdatei'

Nehmen Sie an, Sie wollen die Inhalte der Register R_{02} bis R_{09} dauerhaft aufbewahren. Bei Verwendung eines Kartenlesers würden Sie das mit

'2.009, XEQ "WDTAX"'

tun. Ist es Ihr Wunsch, die Inhalte ins X-Memory zu bringen, müssen Sie zuvor eine Datendatei von ausreichender Größe, d.h. im vorliegenden Fall von mindestens acht Registern, erzeugen, denn es gibt bezüglich der Datenübertragung ins X-Memory keinen Befehl, der wie die Funktion 'SAVEP' zugleich eine Datei anlegt und beschreibt.

Die Erzeugung der benötigten Datei wird so bewerkstelligt: wir entscheiden uns für einen Dateinamen, hier "ABC", und legen ihn ins Alpha-Register; alsdann bestimmen wir die Dateigröße durch den Inhalt von X, hier 8; danach wird

'XEQ "CRFLD"'

ausgeführt, woduch eine leere Datendatei von acht Registern im X-Memory entsteht. Der Befehl 'CRFLD' (*cr*eate *fil*e for *d*ata) erwartet stets den Dateinamen im Alpha-Register und die Dateigröße, das ist die Anzahl der Register der geplanten Datei, in X. Die Register der von 'CRFLD' angelegten Datei sind solange leer, bis sie mittels dafür vorgesehener Befehle gefüllt werden.

Wenn Sie jetzt 'EMDIR' ausführen, müssen Sie als letzten Eintrag des X-Memory-Verzeichnisses

"ABC D008"

erblicken. Mit

'2.009, XEQ "SAVERX"'

können Sie nun, wie beabsichtigt, die Inhalte der Register R_{02} bis R_{09} ins X-Memory, nämlich in die acht Register der Datendatei "ABC", schreiben.

Die Funktion 'SAVERX' (*save registers designated by X*) erwartet eine Kontrollzahl der Form lmn.pqr im X-Register. Sie überträgt die Inhalte eines Blockes von Registern, beginnend mit Register R_{lmn} und endend mit Register R_{pqr}, in die gegenwärtige Datendatei. Gegenwärtig ist dabei die Datei, die als letzte benutzt oder angelegt wurde, es sei denn, zwischenzeitlich ist 'EMDIR' oder 'PURFL' ausgeführt worden. 'EMDIR' macht stets diejenige Datei zur Arbeitsdatei, bei der die Anzeige des Verzeichnisses unterbrochen wird. Abweichend davon gilt: auf dem HP-41CX bleibt die Arbeitsdatei erhalten, wenn 'EMDIR' ohne Unterbrechung zu Ende läuft; auf dem HP-41C und CV wird dagegen in diesem Fall die letzte Datei zur Arbeitsdatei.

2C. Die Funktion 'GETRX' und der Dateizeiger

Es ist nicht verwunderlich, vielmehr folgerichtig mitgedacht, wenn Sie, eifrig vorausschauend, jetzt erwarten, daß etwa die Befehlsfolge '12.019, XEQ "GETRX"' die acht in "ABC" liegenden Zahlen in die Register R_{12} bis R_{19} holt. Doch so naheliegend und selbstverständlich dies auch scheint, dem ist nicht so. Ein Versuch wird Sie sofort belehren: die genannte Befehlsfolge ruft überraschend eine Fehlermeldung, nämlich "END OF FL", hervor. Um zu verstehen, daß dies in der Tat sinnvoll ist, sind einige Erklärungen nötig.

Eine Datendatei im X-Memory kann in Einzelfällen sehr groß sein. Sie kann somit dazu verwendet werden, viele verschiedenen Blöcke von Registerinhalten aufzunehmen. Fernerhin wird es sich bei großen Dateien i.a. ergeben, daß nur Teile der Datei in den Hauptspeicher zu lesen sind. Ja selbst Einzelregister, also Blöcke der Länge 1, zu beschicken oder zu lesen wird sich häufig als notwendig erweisen. Der Preis für solche Handlichkeit beim Umgang mit Datendateien ist die sich zwingend ergebende zusätzliche Bedienung eines *Zeigers*, der, gesteuert vom Benutzer, auf die Stelle der Datei weist, an der Daten abgelegt bzw. von der welche geholt werden sollen. Ohne einen solchen Zeiger wäre es offenkundig nicht sichergestellt, mit 'GETRX' stets den richtigen Datenblock einzulesen.

Doch warum, so die richtige Frage, ist das Setzen des Zeigers vor 'SAVERX' nicht nötig? Die Antwort ist einfach: Auch 'SAVERX' erfordert gewöhnlich die vorangehende Zeigerbedienung, nur war das in unserem Beispiel deshalb nicht nötig, weil "ABC" unmittelbar zuvor erzeugt worden war. Beim Anlegen einer neuen Datei wird nämlich der Dateizeiger automatisch auf den Wert 0 gesetzt, so daß die Befehlstypen 'SAVE' oder 'GET' ohne weiteren Eingriff beim Register 0, das ist das erste Dateiregister, beginnen. (Die Register einer Datei im X-Memory werden, ganz wie die Datenregister im Hauptspeicher, mit 0 beginnend numeriert.) Folglich legt '2.009, SAVERX' die Inhalte der Datenregister R_{02} bis R_{09} in den ersten (in unserem Fall einzigen) acht Dateiregistern von "ABC" ab. Die Aufklärung für die vorhin entstandene Fehlermeldung lautet nun so: 'SAVERX' hatte die unerwartete Wirkung, den Zeiger vom Wert 0 (erstes Dateiregister) auf den Wert 8 (erstes nicht mehr vorhandenes Dateiregister) vorzusetzen (was sich für den Normalfall insofern als äußerst sinnvoll erweist, als man, ohne den Zeiger zwischendurch 'anfassen' zu müssen, die nächsten Register, vorausgesetzt sie existieren, mit dem nächsten 'SAVERX' erreicht). Jeder weitere den Zeigerstand beachtende Befehl wie z.B. 'GETRX' führt daher solange zu der Fehlermeldung "END OF FL", als der Zeiger nicht zurückgesetzt wird.

Der Zeigerstand läßt sich vom Benutzer mit der X-Funktion 'SEEKPTA' (*seek pointer for*

the file named in *A*lpha) beeinflussen. Sie setzt den Zeiger der Datei, deren Name im Alpha-Register steht, auf den im X-Register liegenden Wert. Ist das Alpha-Register leer, arbeitet 'SEEKPTA' bezüglich der Arbeitsdatei. Jede Datei hat ihren eigenen Zeiger; sein Wert liegt im zweiten Register des jeweiligen Dateikopfes. 'SEEKPTA' erreicht immer nur den Zeiger der gegenwärtigen Datei; die Zeiger der anderen Dateien bleiben unberührt.

Nehmen Sie an, Sie wollen jetzt die vormaligen Inhalte der Datenregister R_{04} bis R_{07} aus der Datei "ABC" zurückholen und in den Datenregistern R_{11} bis R_{14} ablegen. Der Abbildung 2.2 können Sie entnehmen, daß jene Zahlen aufgrund von 'SAVERX' in die Dateiregister 2 bis 5 von "ABC" gelangt sind. Mit

'ALPHA, A, B, C, ALPHA, 2, XEQ "SEEKPTA"'

setzen Sie den Zeiger von "ABC" auf 2, der somit auf das dritte Dateiregister weist. Dies getan, führt

'11.014, XEQ "GETRX"'

zum Ziel: Mit 'RCL' können Sie sich davon überzeugen, daß die Register R_{11} bis R_{14} nunmehr die Zahlen 4, 5, 6 und 7 enthalten.

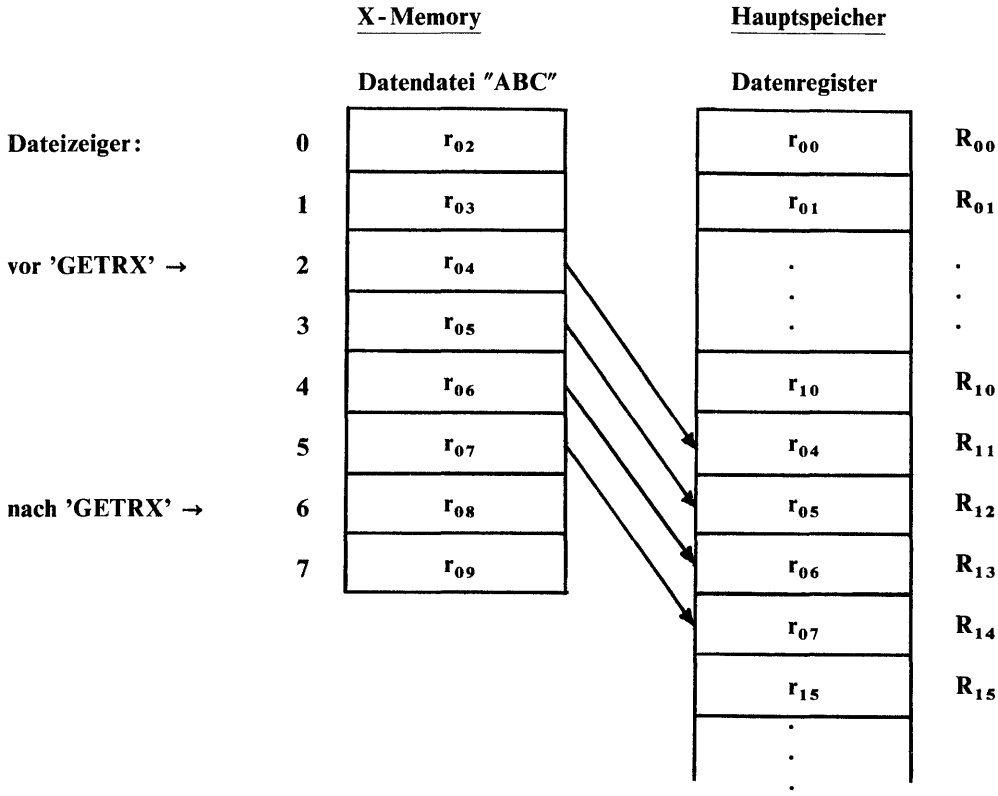


Abbildung 2.2: Wirkung der Befehlsfolge '"ABC", 2, SEEKPTA, 11.014, GETRX'

Die Funktion 'GETRX' (*get* registers designated by *X*) erwartet eine Kontrollzahl der Form lmn.pqr im Register X. Sie holt Daten aus der Arbeitsdatei und legt sie in dem durch die Kontrollzahl bestimmten Block von Datenregistern, beginnend mit Register R_{lmn} und endend mit Register R_{pqr}, ab. Die Daten werden aus dem Block von Dateiregistern, dessen erstes Register durch den gültigen Zeigerstand festgelegt ist, geholt. 'GETRX' addiert außerdem die Blocklänge (das ist pqr − lmn + 1) auf den Zeigerstand, so daß ein folgendes 'GETRX' die nächsten Daten erreicht, ohne daß der Zeigerstand gesondert verstellt zu werden braucht.

Alle 'SAVE'- und 'GET'-Befehle, die sich auf Datendateien beziehen, setzen den Dateizeiger nach der gleichen Logik höher, nämlich auf das erste Dateiregister, das dem Block, der durch die jeweilige Funktion gerade beschrieben oder gelesen wurde, folgt.

Die Funktion 'RCLPTA' (*recall pointer* for the file named in *A*lpha) bietet eine Möglichkeit, den gültigen Stand des Zeigers einer beliebigen Datei schnell und einfach festzustellen. Man braucht nur den Namen der Datei ins Alpha-Register zu legen und

'XEQ "RCLPTA"'

auszuführen. Anschließend liegt der Zeigerwert in X. Ist das Alpha-Register leer, arbeitet 'RCLPTA' bezüglich der gegenwärtigen Datei. Die Funktion hebt den Stapel an, genau wie es z.B. bei 'RCL' geschieht, es sei denn, unmittelbar zuvor ist ein die Stapelbewegung sperrender Befehl ('ENTER↑' oder 'CLX') durchgeführt worden. Als Beispiel wollen wir den Zeigerstand, der durch unser letztes 'GETRX' entstanden ist, überprüfen. Wenn Sie in der Zwischenzeit nichts verändert haben, fördert 'RCLPTA', wie zu erwarten war, 6 zutage, denn als letztes hatten Sie die Inhalte der Register 2 bis 5 von "ABC" in den Hauptspeicher geholt. *Tip:* 'RCLPTA' ist besonders gut dazu geeignet, eine Datei zur Arbeitsdatei zu machen, weil der Inhalt der Datei dabei völlig unbehelligt bleibt.

2D. Die Funktionen 'SEEKPT' und 'RCLPT'

Die Funktion 'SEEKPT' (*seek pointer*) arbeitet genau wie 'SEEKPTA' mit dem Unterschied, daß grundsätzlich der Zeiger der Arbeitsdatei gemäß dem Inhalt von X gesetzt wird, unabhängig davon, was im Alpha-Register steht. Das hat den beachtlichen Vorteil, über das Alpha-Register frei und unbelastet verfügen zu können, sobald durch einen anderen Befehl sichergestellt wird, daß die richtige Datei gegenwärtig ist. Sie werden diesen Vorteil schätzen lernen, weil Sie 'SEEKPT' günstig in Programmen verwenden können, um der Fehlermeldung "FL NOT FOUND", die bei 'SEEKPTA' auftritt, wenn das Alpha-Register nicht leer ist und keinen gültigen Dateinamen enthält, aus dem Wege zu gehen. Natürlich muß anfangs die richtige Arbeitsdatei mit irgendeinem Dateibefehl ausgewählt werden.

Die Funktion 'RCLPT' (*recall pointer*) arbeitet genau wie 'RCLPTA' mit dem Unterschied, daß grundsätzlich der Zeigerwert der Arbeitsdatei in X abgeliefert wird. Wie bei 'SEEKPT' bleibt der Inhalt des Alpha-Registers unbeachtet, so daß auch hierbei über dieses wichtige Register ohne Einschränkungen verfügt werden kann, sobald die Arbeitsdatei anderweitig festgelegt worden ist.

2E. Weitere Dateifunktionen: 'SAVEX', 'GETX', 'SAVER', 'GETR' und 'CLFL'

Die Funktion 'SAVEX' (*save* register *X*) überträgt den Inhalt des X-Registers in die

Arbeitsdatei. Der gültige Zeigerstand bestimmt dabei, welches Dateiregister diesen Inhalt aufnimmt. Nach der Übertragung wird der Zeigerstand um 1 heraufgesetzt, so daß ein folgendes 'SAVEX' das nächste Dateiregister füllt. Will man beispielsweise den Wert 15 ins dritte Dateiregister (Register 2) bringen, führt man

'2, XEQ "SEEKPT", 15, XEQ "SAVEX"'

aus. Anschließend lautet der Zeigerwert 3, so daß ein zweites 'SAVEX' den Inhalt von X ins Register 3 der Arbeitsdatei brächte. Wie nützlich das automatische Inkrementieren des Dateizeigers ist, zeigt das folgende Programm-Modell, das in einer Schleife Werte berechnet, die in einer Datendatei abgelegt werden:

```
"DATEINAME"
CRFLD     (oder 'CLFL', weiter unten besprochen)
LBL 01
  Schritte zur
  Berechnung
  irgendwelcher
  Werte
SAVEX
GTO 01
```

Es gibt also keinen Grund, Zeit und Platz mit der Prüfung von Dateizeigern oder der Verwaltung von Zählern ('ISG') zu verschwenden. Sie können sich u.U. sogar umgekehrt des automatisch bereitgestellten Dateizeigerstandes zur Berechnung von Werten in der Schleife bedienen, weil Ihnen ja 'RCLPT' zur Verfügung steht. Auch den Schleifenabbruch können Sie gelegentlich durch Verwendung der Fehlermeldung "END OF FL" (in Verbindung mit Flag 25) vereinfachen. Dies erlaubt hier und da eine recht einfache und wirtschaftliche Programmgestaltung.

Die Funktion 'GETX' (*get* current register and transfer to *X*) ist die Umkehrung von 'SAVEX'. Der Inhalt des Dateiregisters, auf das der Dateizeiger weist, wird nach X geholt und der Dateizeiger um 1 heraufgesetzt. Der Stapel wird dabei angehoben, um Platz für die Zahl zu schaffen, so wie es bei 'RCL' geschieht. Will man beispielsweise die Zahl 15 aus Register 2 der Datei "ABC" (die immer noch Arbeitsdatei ist, wenn Sie den vorangegangenen Erläuterungen ohne Unterbrechung gefolgt sind) nach X holen, leistet

'2, XEQ "SEEKPT", XEQ "GETX"'

das Gewünschte. Anschließend steht der Dateizeiger auf 3, wie 'RCLPT' enthüllt. Auch hier wird also der Zeiger, wie bei 'SAVEX', automatisch inkrementiert, so daß 'GETX' bequem und bezüglich des Byte-Verbrauchs haushälterisch in Schleifen verwendet werden kann.

Die Funktion 'SAVER' (*save r*egisters) überträgt die Inhalte *aller* Datenregister in die Arbeitsdatei, wenn das Alpha-Register leer ist, oder in diejenige Datei, deren Name im Alpha-Register steht. Anders jedoch als 'SAVEX' und 'SAVERX' läßt 'SAVER' den Zeigerstand völlig unberücksichtigt. Der Inhalt von Datenregister R_{00} gelangt ins Dateiregister 0, der von R_{01} ins Dateiregister 1, usw. Bedauerlicherweise ändert 'SAVER' aber trotz Mißachtung des Zeigerstandes für die eigene Arbeit diesen ab: der Zeiger weist anschließend auf das Dateiregister, das dem zuletzt beschriebenen unmittelbar folgt.

Ist die durch 'SAVER' angesprochene Datei nicht groß genug, um die Inhalte aller existierenden Datenregister zu übernehmen, wird die Fehlermeldung "END OF FL" ausgegeben und darüber hinaus die Übertragung von Inhalten *gänzlich abgewiesen.* Es gelangen also in diesem Fall auch die Zahlen, die Platz in der Datendatei gefunden hätten, nicht dorthin. Diese mißliche Eigenschaft, die selbst durch den Einsatz von Flag 25 nicht umgangen werden kann, mindert den Gebrauchswert der Funktion 'SAVER'. Außer wenn die gegenwärtige Anzahl der verfügbaren Datenregister der Anzahl zu übertragender Daten angepaßt ist und überdies die Größe der angesprochenen Datei nicht übertrifft, sollten Sie sich daher auf die Benutzung von 'SAVERX' beschränken, statt mit 'SAVER' zu arbeiten und dafür Platz im X-Memory durch das Anlegen hinreichend großer Dateien zu vergeuden [ausgenommen sind natürlich solche Sonderfälle, wie sie sich beispielsweise für die Routine "HS + XFM" in Anm. c) zu Abschnitt 10C anbieten, wo 'SAVER' 447 (!) Datenregisterinhalte auf einen Schlag überträgt]. Sie können die Anpassung der Datenregisteranzahl zwar jederzeit nachträglich vornehmen. Will man beispielsweise die Inhalte der Datenregister R_{00} bis R_{23} ins X-Memory übertragen, führt

```
24
PSIZE
"DATEINAME"
SAVER
```

fehlerfrei zum Ziel. Die Funktion 'PSIZE' (vgl. Abschnitt 4D) setzt die verfügbare Datenregisteranzahl auf 24 fest, so daß 'SAVER' die Übertragung der Inhalte von R_{00} bis R_{23} nicht verweigern kann, wenn die Dateigröße mindestens 24 beträgt, doch gehen Ihnen dabei u.U. – nämlich dann, wenn die Datenregisteranzahl vermindert wird – Zahlen in den Registern von R_{24} an aufwärts, die Sie vielleicht noch brauchen, verloren. Daher ist die folgende Methode, die nur unbedeutend mehr Aufwand verlangt, i.a. besser:

```
"DATEINAME"
0
SEEKPTA
.023
SAVERX
```

Überlegen Sie sich stets genau, welchem Verfahren Sie den Vorzug geben.

Im Gegensatz zu der beschränkt verwendungsfähigen Funktion 'SAVER' ist die Funktion 'GETR' (*get r*egisters) wesentlich nützlicher. 'GETR' holt Daten, beginnend mit Dateiregister 0, aus der Arbeitsdatei, wenn das Alpha-Register leer ist, oder aus derjenigen Datei, deren Name im Alpha-Register steht, und legt sie in den Datenregistern des Hauptspeichers, beginnend bei R_{00}, ab. Anders aber als 'SAVER' versagt 'GETR' den Dienst nie, gleichgültig, ob die angesprochene Datei größer oder kleiner als die Datenregisteranzahl ist, und ebensowenig erscheint jemals die Fehlermeldung "END OF FL". Im ersten Fall (Datei größer) werden nur soviele Daten geholt, wie Platz im Hauptspeicher finden; im zweiten Fall (genügend Datenregister vorhanden) gelangt der gesamte Dateiinhalt dorthin. Daher ist das Übertragen einer vollständigen Datei denkbar einfach:

```
"DATEINAME"
GETR
```

bringt bereits die ganze Datei in die Datenregister von R_{00} an aufwärts, so daß man nur noch 'REGMOVE' oder 'REGSWAP' (vgl. Abschnitt 4E) nachzuschicken braucht, falls die erlangten Daten in einen nicht mit R_{00} beginnenden Block von Datenregistern gehören.

'GETR' kümmert sich ebensowenig wie 'SAVER' um den aktuellen Zeigerstand, setzt ihn jedoch ebenso wie 'SAVER' hoch, und zwar stets auf den Wert, der sich ergibt, wenn man die Nummer des letzten übertragenen Dateiregisters um 1 erhöht. Das ist bei vollständig gelungener Übertragung genau die Dateigröße, so daß im Anschluß an 'GETR' die Meldung "END OF FL" ausgelöst werden kann, wenn man die Datei ohne die Berücksichtigung des Zeigerstandes mit anderen Dateibefehlen anspricht.

Die Funktion 'CLFL' (*cl*ear *file*) löscht den gesamten Inhalt einer Datendatei, d.h., alle ihre Register werden auf Null gesetzt. Desgleichen wird der Dateizeiger auf Null gesetzt, so daß man gleich danach damit beginnen kann, die Datei neu mit 'SAVE'-Befehlen zu füllen. 'CLFL' verlangt in jedem Fall die Benennung der zu löschenden Datei im Alpha-Register. Gewöhnlich wird 'CLFL' der Neuverwendung einer bereits vorhandenen Datei vorangeschickt, um sicherzugehen, daß keine Reste aus früherer Verwendung die neue Dateibenutzung stören. Bei Neuanlage von Datendateien mit 'CRFLD' ist 'CLFL' aber überflüssig, weil 'CRFLD' die geschaffene Datei dem Benutzer leer übergibt.

'CLFL' löst die Fehlermeldung "NAME ERR" aus, wenn das Alpha-Register leer ist. 'CLFL' arbeitet also *nicht* automatisch auf der gegenwärtigen Datei, sondern schafft sich deren 'Gegenwart' grundsätzlich erst. Andererseits macht 'CLFL' bedingungsloser als andere Dateifunktionen die im Alpha-Register benannte Datei zur gegenwärtigen Datei (das soll heißen ungeachtet ihres Typs), weist die Ausführung aber – sozusagen 'ehrlich bleibend' – mit der Meldung "FL TYPE ERR" zurück, sobald es sich dabei um eine Programmdatei handelt (eine solche Datei kann nur mit 'SAVEP' ersetzt oder mit 'PURFL' gänzlich getilgt werden). Von der gewissermaßen nutzlosen, nichtsdestoweniger tatsächlich erbrachten Leistung der Funktion 'CLFL', eine nicht zu beeinflussende Datei dennoch zur Arbeitsdatei zu machen [die Routine "DAT?" aus Anm. a) zu Abschnitt 10C läßt sich zur direkten Nachprüfung verwenden], können Sie sich leicht an Hand einer anderen, ebenfalls 'wesensfremden', in dem Fall allerdings nutzbringenden Leistung überzeugen: Die Funktionen 'RCLPTA' und 'RCLPT' können ihrer Bestimmung gemäß eigentlich nicht auf Programmdateien angesetzt werden, liefern aber überraschenderweise auch für solche Dateien ein sinnvolles und nützliches Ergebnis; sie legen nämlich die Byte-Anzahl der gegenwärtigen Programmdatei nach X. 'RCLPTA' tut dies, wenn im Alpha-Register eine Programmdatei benannt ist, und 'RCLPT', wenn das Alpha-Register leer und die gegenwärtige Datei eine Programmdatei ist. Wenn Sie also 'RCLPT' auf 'CLFL' folgen lassen, sobald der Name einer Programmdatei im Alpha-Register liegt, erhalten Sie deren Byte-Anzahl. In der Routine "CBX" des Kapitels 5 wird die genannte Eigenschaft von 'RCLPT(A)' ausgenutzt.

Die Funktion 'PURFL' (*purge file*) löscht die im Alpha-Register benannte Datei vollständig im X-Memory, das Ihnen anschließend mit einer entsprechend vergrößerten Anzahl freier Register wieder für andere Zwecke zur Verfügung steht. Wie 'CLFL' muß auch 'PURFL' grundsätzlich einen gültigen Dateinamen im Alpha-Register vorfinden. Beachten Sie unter allen Umständen die in Abschnitt 1F in Bezug auf 'PURFL' ausgesprochene WARNUNG.

Auf dem HP-41CX gibt es zusätzlich die Funktion 'RESZFL' (*resize file*), welche die Größe einer bereits existierenden Datei zu verändern vermag. 'RESZFL' arbeitet einzig und

allein auf der gegenwärtigen Datei, die zuvor mit irgendeiner anderen Datei-bestimmenden Funktion, etwa 'RCLPTA', ausgewählt werden muß. Anschließend legt man die neue Dateigröße ins X-Register und ruft 'RESZFL' auf. Setzt man die Dateigröße herab, werden die höchstnumerierten Dateiregister getilgt. Im Falle, daß diese Register Daten enthalten, erscheint die Fehlermeldung "FL SIZE ERR", und der Befehl wird nicht ausgeführt, so daß der Benutzer ausreichend davor geschützt ist, Daten im X-Memory unbeabsichtigt durch 'RESZFL' zu verlieren. Man hat aber die Möglichkeit, diese Sicherung zu umgehen, weil 'RESZFL' arbeitet, ohne zu prüfen, ob Dateiregisterinhalte verloren gehen, wenn die neue Dateigröße in X *negativ* angegeben wird.

Sie werden zweifellos erkannt haben, daß das X-Memory in Bezug auf die Ablage und den Rückruf von Daten wesentlich vielseitiger als der Kartenleser ist. Es bereitet keine Probleme, einzelne Dateiregister oder Unterblöcke von Blöcken von Dateiregistern anzusprechen. Daher lassen sich auch größere Datendateien ganz bequem handhaben. Man kann je nach Bedarf einzelne Zahlen oder kleine Gruppen davon hernehmen, ohne gleich die ganze Datei dem Hauptspeicher einverleiben zu müssen, wie es beim Kartenleser zwingend geschieht. Unstreitig eine erhebliche Verbesserung.

Wie sehr Datendateien zur Leistungskraft des HP-41 und zur Datensicherheit im Rechner beitragen, das wird in Kapitel 6 vorgeführt, wo sich die Programme "SOLVE", "DERIV" und "INTEG" des X-Memorys bedienen, um Daten zerstörungssicher abzulegen, so daß vom Benutzer programmierte Routinen zur Berechnung der Werte einer Funktion f(x) einschränkungslos mit den Datenregistern des Hauptspeichers arbeiten können.

KAPITEL 3

Textdateien im X-Memory

3A. Was ist eine Textdatei?

12 der 47 Funktionen des X-Funktionen-Moduls und 14 der 61 X-Funktionen des HP-41CX sind ausschließlich für die Handhabung von Textdateien vorgesehen. Dieses Kapitel erläutert den Aufbau von und den Umgang mit ASCII-Dateien, die gegenüber der Grundausstattung des HP-41 erheblich bessere Möglichkeiten der Verarbeitung von Zeichenketten erlauben. Sollten für Ihre eigenen Anwendungen nur wenige kurze Text-Einsprengsel genügen, können Sie dieses Kapitel vorerst überspringen.

Vor der Einführung des X-Memorys war die Verarbeitung von Zeichenketten auf dem HP-41 zwar möglich, aber zugegebenermaßen doch recht schwerfällig, wenn mehr geschehen sollte, als Ein- und Ausgabe von Zahlen mit erläuternden Texten zu versehen. Die Ketten mußten beispielsweise in Teilstücke von höchstens sechs Zeichen zerlegt werden, damit sie mit 'ASTO' überhaupt in Datenregistern untergebracht werden konnten. Das X-Memory erlaubt nun eine völlig neuartige Speicherung von Alpha-Ketten, bei der eine registergerechte Zerlegung der genannten Art nicht mehr nötig ist. Stattdessen werden die Ketten unzerstückelt in einer ASCII-Datei im X-Memory untergebracht. (Textdatei und ASCII-Datei werden in diesem Buch, wie auch in den Handbüchern von HP, als gleichwertige Begriffe benutzt.) Eine Zeichenkette, also eine zusammengehörige Gruppe von Zeichen, heißt *Satz*, wenn sie als Teil einer Textdatei aufgefaßt wird. Ein Satz kann bis zu 254 Zeichen lang sein. Die Anzahl von Sätzen, die in einer einzelnen Textdatei Platz haben, ist lediglich durch die Aufnahmefähigkeit des X-Memorys begrenzt. Wie bei Datendateien (Abschnitt 2B) müssen Sie auch bei Textdateien die gewünschte Dateigröße als Registeranzahl im X-Register angeben, wenn Sie die Datei anlegen. Diese Anzahl läßt sich so bestimmen: Bezeichnen wir mit N_S die Höchstanzahl von Sätzen, die eine Textdatei aufnehmen soll, und mit N_Z die Höchstanzahl von Zeichen, die in allen Sätzen zusammen enthalten sein wird, so gilt für die Mindestanzahl von Registern N_R, die bei der Anlage der Datei in X anzugeben ist,

$$N_R = \text{INT}[(N_S + N_Z + 7)/7].$$

Durch die Addition von 7 wird das sog. *Textabschlußbyte* berücksichtigt und zugleich für die richtige Abrundung gesorgt. [Anm. d. Übers. für Synthetiker: Das Textabschlußbyte lautet hexadezimal FF. Es signalisiert das Ende des gültigen Textes einer ASCII-Datei. Seine Gestalt ist es, welche die Beschränkung der Satzlängen auf höchstens 254 (hexadezimal FE) Bytes erzwingt, denn jedem Satz steht das sog. *Satzlängenbyte* voran, und dieses darf, wie man leicht

einsieht, natürlich nie mit dem Textabschlußbyte übereinstimmen. Beachten Sie in diesem Zusammenhang die in Abschnitt 10C als Anm. c) gegebene Routine "HS + XFM".] Wenn Sie z.B. 20 Namen von jeweils höchstens 25 Zeichen Länge in eine Datei bekommen wollen, brauchen Sie dafür

$$N_R = INT[(20 + 20 \cdot 25 + 7)/7] = INT(75{,}29) = 75$$

Register. Natürlich ist es i.a. zu empfehlen, eine etwas größere Anzahl vorzusehen, weil die Speicherbedürfnisse häufig wachsen. Das Handbuch des X-Funktionen-Moduls schlägt vor, Ihrer Anfangsschätzung für die Zeichenanzahl 20% zuzuschlagen und dann durch 7 zu teilen. Auf dem HP-41CX bekommen Sie keine Probleme, wenn die Dateigröße nicht mehr ausreicht, weil Sie die anfangs festgelegte Dateigröße jederzeit dem tatsächlichen Bedarf mit 'RESZFL' (Abschnitt 2E) anpassen können, auch nach unten. Allerdings lassen Textdateien im Gegensatz zu Datendateien auch mit negativen Einträgen in X keine Verminderung der Dateigröße mehr zu, wenn dabei Text verloren ginge. In Abschnitt 3G finden Sie zwei Programme, die es Ihnen ermöglichen, auf dem HP-41C und CV ebenfalls die Dateigröße zu verändern.

So wie der Dateizeiger einer Datendatei auf das nächste zu lesende oder zu beschreibende Dateiregister weist, markiert der Dateizeiger einer Textdatei deren nächsten Satz. Doch nicht nur den. Weil es nämlich auch gelingen soll, auf einzelne Zeichen eines Satzes zuzugreifen, ist der Dateizeiger einer Textdatei aus zwei Zeigern zusammengesetzt: aus dem *Satzzeiger*, dargestellt als ganzzahliger Vorkomma-Anteil v des Dateizeigers, und dem *Zeichenzeiger*, dargestellt als dessen Nachkomma-Anteil pqr ($0 \leq pqr \leq 253$). Der Dateizeiger einer Textdatei hat somit die kombinierte Gestalt v.pqr und ist mit den schon in Kapitel 2 besprochenen Zeigerbefehlen 'SEEKPT(A)' und 'RCLPT(A)' in sinngemäßer Weise ansprechbar.

Wir wollen uns die eben besprochenen Punkte an einem Beispiel veranschaulichen. Dazu schaffen wir uns im X-Memory eine Textdatei von 25 Registern mit den Namen "NAMEN". Vergewissern Sie sich vorher mit 'EMDIR' oder – auf dem HP-41CX – mit 'CAT 4' oder 'EMROOM', ob genügend freie Register im X-Memory dafür bereitstehen. Löschen Sie, falls das Ergebnis Ihrer Prüfung Sie zwingt, für mehr Platz zu sorgen, ein oder mehrere Dateien mit 'PURFL'. Legen Sie dann die Zahl 25 ins X-Register und den Text "NAMEN" ins Alpha-Register, und führen Sie anschließend

'XEQ "CRFLAS"'

aus. Die Funktion 'CRFLAS' (*cr*eate *file* for *AS*CII-text) arbeitet also genauso wie die Funktion 'CRFLD': die Dateigröße wird in X erwartet und der Dateiname im Alpha-Register.

Sobald Sie die Textdatei "NAMEN" angelegt haben, können Sie Text hineinschreiben. Dazu dient in zunächst einfacher Form die Funktion 'APPREC' (*app*end *rec*ord). Nehmen Sie an, Sie wollten drei Namen in die Datei bringen:

Satz-Nr.	Name
0	RICHARD NELSON
1	ROGER HILL
2	JOHN MCGECHIE

Der Vorgang ist denkbar einfach. Erst schreiben Sie den Namen ins Alpha-Register, dann führen Sie

'XEQ "APPREC"'

aus. 'APPREC' fügt der Arbeitsdatei einen neuen Satz an, dessen Inhalt der Text wird, welcher gerade im Alpha-Register liegt. Der Dateizeiger wird auf das Zeichen gestellt, welches dem letzten Zeichen des letzten Satzes unmittelbar folgt. Mit der nachstehenden Befehlsfolge bringen Sie die drei Namen in die Datei:

```
'"RICHARD NELSON", XEQ "APPREC",
 "ROGER HILL", XEQ "APPREC",
 "JOHN MCGECHIE", XEQ "APPREC"'
```

Die doppelten Anführungszeichen (") bedeuten, wie überall in diesem Buch, die Ein- (oder Aus)gabe von Alpha-Zeichen.

Es ist viel leichter, Alpha-Daten in eine Textdatei zu bringen, als in Datenregistern abzulegen. Die Funktion 'APPREC' überträgt den gesamten Alpha-Inhalt, also bis zu 24 Zeichen auf einen Schlag, wohingegen 'ASTO' nur höchstens sechs Zeichen umzusetzen vermag. Allein den Namen "RICHARD NELSON" in Datenregistern unterzubringen, erfordert fünf Befehle: 'ASTO 01, ASHF, ASTO 02, ASHF, ASTO 03'. Unstreitig schwerfällig, verglichen mit dem einen Befehl 'APPREC'; und noch mühevoller, falls der Alpha-Inhalt unversehrt erhalten bleiben soll ('CLA, ARCL 01, ARCL 02, ARCL 03' wäre anzuschließen).

Doch ist das umstandslose Einbringen von Text keineswegs der einzige Vorteil, den man gewinnt, wenn man sich einer Textdatei bedient, um Texte aufzubewahren. Der entscheidende Nutzen dieser Dateien tritt erst dort zutage, wo man all die X-Funktionen zu Dienstleistungen verpflichtet, die für den Zugriff auf Texte im X-Memory bereitstehen, darunter solche, mit denen man ganze Sätze oder einzelne Zeichen an beliebiger Stelle einfügen oder löschen kann.

3B. Zugriff auf Textdateien

Es gibt zwei Funktionen, mit denen man die Inhalte von Textdateien abrufen kann: 'ARCLREC' (*a*lpha *recall rec*ord) und 'GETREC' (*get rec*ord). 'ARCLREC' holt, wie der Name schon vermuten läßt, Zeichen aus der Arbeitsdatei und hängt sie dem Inhalt des Alpha-Registers an, wobei der gültige Dateizeigerstand bestimmt, ab welchem Zeichen der Arbeitsdatei die Übertragung erfolgen soll. Es werden die Zeichen bis zum Satzende abgeholt, allerdings nur, soweit das Alpha-Register (noch) Fassungsvermögen dafür hat. Sobald das Alpha-Register gefüllt oder das Satzende erreicht ist, wird die Übertragung abgebrochen. 'ARCLREC' setzt Flag 17, wenn die Übertragung vor Erreichen des Satzendes abgebrochen wird, um anzuzeigen, daß noch weitere Zeichen aus dem zuletzt angesprochenen Satz zu übertragen sind; andernfalls wird Flag 17 gelöscht. Außerdem rückt der Dateizeiger auf das Zeichen, welches dem als letztes übertragenen Zeichen unmittelbar folgt, vor. Die Funktion 'GETREC' arbeitet genau wie die Befehlsfolge 'CLA, ARCLREC'.

Angenommen, Sie wollten sich den Inhalt der Datei "NAMEN", die noch immer gegenwärtig sein sollte, ansehen. Die Folge

'0, SEEKPT, GETREC, AVIEW'

bringt den Inhalt des ersten Satzes, "RICHARD NELSON", in die Anzeige. Wenn Sie die

Folge

'ARCLREC, AVIEW'

anschließen, erscheint

"RICHARD NELSONROGER HILL".

Hoppla! Wir haben das 'CLA' vor 'ARCLREC' vergessen. Sie werden die Funktion 'GETREC' meistens handlicher finden, weil sie das Alpha-Register selbständig löscht, bevor der Satz oder Teile davon (wenn dieser länger als 24 Zeichen ist) übertragen werden. Die Funktion 'ARCLREC' ist in jenen selteneren Fällen, in denen der Satzinhalt einer schon vorhandenen Meldung angeschlossen werden soll, vorteilhafter.

In dem voranstehenden Beispiel konnte 'ARCLREC' den ganzen zweiten Satz noch im Alpha-Register unterbringen. Folgerichtig wurde Flag 17 gelöscht. Hätten Sie jedoch vor dem Aufruf von 'ARCLREC' dem schon vorhandenen Alpha-Inhalt "RICHARD NELSON" eine Leerstelle angehängt, wäre das letzte "L" von "HILL" nicht mehr berücksichtigt worden; erst ein anschließendes 'GETREC' hätte dieses zurückgebliebene "L" (und nur dieses) ins Alpha-Register geholt. Außerdem wäre Flag 17 gesetzt worden. Wenn Sie also ein Programm schreiben, das Texte einer Datei entnimmt und druckt, werden Sie i.a. Flag 17 abfragen müssen. Beispiel:

```
n         (Satznummer)
SEEKPT    setzt den Zeiger auf den Anfang des n-ten Satzes
LBL 01
GETREC    holt (bis zu) 24 Zeichen des Satzes
ACA       schickt die Zeichen in den Druckpuffer
FS? 17    bei unvollständigem Satz weitere (24) Zeichen holen
GTO 01
PRBUF     andernfalls Druckpuffer ausdrucken
```

Einige HP-IL Peripheriegeräte prüfen von sich aus den Zustand von Flag 17 im Anschluß an 'GETREC' oder 'ARCLREC'. Wenn Flag 17 gesetzt ist, wird der normalerweise ausgeführte 'Wagenrücklauf' unterdrückt, so daß der Rest des Satzes, falls dieser nicht gerade überlang ist, in dieselbe Zeile gelangt.

3C. Einfügungen in bestehende Textdateien

Angenommen, Sie wollten den ersten Satz unserer Übungsdatei von "RICHARD NELSON" in "RICHARD NELSON, PPC-GRUENDER" abändern. Zunächst müssen Sie dafür den Dateizeiger auf 0 setzen, so daß er auf das erste Zeichen des ersten Satzes weist. Sie erreichen dies einfach mit '0, SEEKPT', wenn "NAMEN" immer noch gegenwärtig ist.

Die Funktion 'APPCHR' (*app*end *cha*racters) hängt den Inhalt des Alpha-Registers dem Satz an, auf den der Satzzeiger weist. Der Stand des Zeichenzeigers wird nicht berücksichtigt. Der (zusammengesetzte) Dateizeiger weist anschließend auf das Zeichen, welches dem zuletzt angefügten unmittelbar folgt. Anders als die Funktion 'APPREC' erzeugt 'APPCHR' keinen neuen Satz. Führen Sie jetzt die Änderung mit

'", PPC-GRUENDER", XEQ "APPCHR"'

durch, und betrachten Sie das Ergebnis mit

'0, SEEKPT, GETREC, AVIEW, GETREC, AVIEW'.

Unstreitig lassen sich Texte im X-Memory mit 'APPCHR' wesentlich leichter erweitern als es zuvor mit 'ARCL', 'APPEND' und 'ASTO' bei in Datenregistern liegenden Zeichenketten geschehen konnte.

Mit der Funktion 'INSCHR' (*ins*ert *cha*racter) hat man eine weitere, doch andersgeartete Möglichkeit, Zeichen einzufügen. Als Beispiel für ihre Anwendung wollen wir zwischen "RICHARD " (Leerstelle beachten) und "NELSON" die Zeichen "J. " (Leerstelle beachten) einsetzen. Zuvor muß natürlich durch den Dateizeiger genau festgelegt werden, wo die Zeichen eingefügt werden sollen. In unserem Beispiel hat der Zeigerstand 0.008 zu lauten, weil die Zeichen "J. " genau vor das "N", das ist das 9. Zeichen des 1. Satzes, gehören. 'INSCHR' setzt nämlich den Inhalt des Alpha-Registers *vor* das Zeichen, auf welches der Dateizeiger weist, und stellt ihn dann um soviele Zeichen höher, als eingefügt wurden, so daß er anschließend – genau wie bei den anderen einfügenden Funktionen – auf das Zeichen weist, welches dem als letztes eingefügten unmittelbar folgt. Die Befehle

'0.008, SEEKPT, "J. ", XEQ "INSCHR"'

bewirken die gewünschte Einfügung, was Sie mit anschließendem

'0, SEEKPT, GETREC, AVIEW, GETREC, AVIEW'

sofort überprüfen können. Jene Befehlsfolge hingegen, die – aus 'ARCL', 'APPEND' und 'ASTO' bestehend – dasselbe in Datenregistern bewirken würde, trotzt (besser spottet) jeder Beschreibung.

Damit nicht genug: Es gibt noch eine (letzte) Möglichkeit, bestehende Textdateien zu ergänzen, und zwar durch Einfügung ganzer Sätze. Die Funktion 'INSREC' (*ins*ert *rec*ord) dient diesem Zweck. In folgerichtiger Anlehnung an 'INSCHR' wird ein neuer Satz durch 'INSREC' *vor* den Satz gestellt, auf den der Satzzeiger weist (der Zeichenzeiger bleibt unberücksichtigt), wobei der Inhalt des Alpha-Registers den neuen Satz liefert. 'INSREC' setzt zugleich den Dateizeiger auf das Zeichen, welches dem letzten Zeichen des neuen Satzes unmittelbar folgt.

Als Beispiel für die Anwendung von 'INSREC' wollen wir den Namen "CLIFFORD STERN" zwischen "ROGER HILL" und "JOHN MCGECHIE" einfügen. Weil der neue Satz vor den 3. Satz (Satznummer 2) gelangen soll, erreichen wir unser Ziel mit

'2, SEEKPT, "CLIFFORD STERN", XEQ "INSREC"'.

Die weiter unten in Abschnitt 3E beschriebene Funktion 'POSFL' erleichtert die Suche nach jener Zeichenfolge, vor der Sie Zeichen mit 'INSCHR' oder Sätze mit 'INSREC' einfügen wollen. Sie benutzen erst 'POSFL', um die Zeichenkette zu finden, denen der neue Text vorangehen soll, und rufen dann, je nach Bedarf, 'INSCHR' oder 'INSREC' auf.

3D. Löschen von Teilen bestehender Textdateien

Das Ergebnis unserer Übungen im letzten Abschnitt sieht so aus:

Satz-Nr.	Name
0	RICHARD J. NELSON, PPC-GRUENDER
1	ROGER HILL
2	CLIFFORD STERN
3	JOHN MCGECHIE

Nehmen Sie an, Sie wollten nunmehr den letzten Satz der Datei, Satz Nr. 3, löschen. Das kann mit der Funktion 'DELREC' (*del*ete *rec*ord) bewerkstelligt werden. Sie löscht in der Arbeitsdatei den Satz, auf welchen der Satzzeiger weist. 'DELREC' läßt den Satzzeiger unangetastet, setzt jedoch den Zeichenzeiger auf 0. Mit der Folge

'3, SEEKPT, DELREC'

löschen Sie den 4. Satz. Wenn Sie gleich danach 'GETREC' ausführen (der Satzzeiger steht noch auf 3), erhalten Sie ganz richtig die Meldung "END OF FL", denn den Satz 3 gibt es ja nicht mehr. Hätten Sie Satz 1 gelöscht, wären die Sätze 2 und 3 aufgerückt: sie wären zu Satz 1 bzw. Satz 2 geworden. 'DELREC' arbeitet genau wie 'INSREC' nur bezüglich eines einzelnen Satzes. Haben Sie an derselben Stelle einer Datei mehrere Sätze hintereinander einzufügen oder zu löschen, muß das mit einer Schleife, die 'DELREC' bzw. 'INSREC' enthält, geschehen.

Nehmen Sie nun ferner an, Sie wollten im Satz 0 den Teil ", PPC-GRUENDER" löschen. Wieder gibt es einen geeigneten Befehl: Die Funktion 'DELCHR' (*del*ete *char*acter) löscht Teile des Satzes, auf den der Satzzeiger weist, nach folgender Regel: die Löschung beginnt mit dem Zeichen, auf das der Zeichenzeiger weist, und erstreckt sich auf soviele Zeichen, als durch die Zahl im X-Register, und zwar durch deren ganzen Anteil, bestimmt wird. Ist diese Zahl größer als die Anzahl der restlichen Zeichen des Satzes, vom gültigen Zeigerstand aus gerechnet, werden nur die Zeichen bis zum Satzende gelöscht; der nachfolgende Satz wird nicht angegriffen. 'DELREC' läßt Satz- und Zeichenzeiger sowie das X-Register unverändert. Die beabsichtigte Löschung von ", PPC-GRUENDER" wird demgemäß mit der Befehlsfolge

'0.017, SEEKPT, 14, DELCHR'

bewirkt. Das erste zu löschende Zeichen, das Komma hinter "NELSON", war das 18. Zeichen (Zeichen Nr. 17) von Satz 0. Die Anzahl der zu löschenden Zeichen betrug 14. Da im vorstehenden Beispiel allerdings alle Zeichen des Satzes 0 vom 18. ab zu löschen waren, hätten Sie die Anzahl der zu löschenden Zeichen nicht auszuzählen brauchen. Mit 99 im X-Register wäre dieselbe Wirkung wie mit 14 erzielt worden; es wäre einzig erforderlich gewesen, die Zahl im X-Register mindestens so groß wie die Anzahl der restlichen Zeichen des Satzes 0, vom Komma aus gerechnet, zu wählen.

Den gesamten Inhalt einer Textdatei – nicht die Datei selber! – löscht man mit der Funktion 'CLFL' (*cl*ear *file*), die den Namen der Datei im Alpha-Register erwartet. Die Angabe des Dateinamens ist zwingend erforderlich, denn 'CLFL' arbeitet nicht einfach bezüglich der gegenwärtigen Datei. Nach Ausführung von 'CLFL' ist aber die Datei, deren Inhalt gelöscht wurde, gegenwärtig, und ihr Dateizeiger steht auf 0. Das ist zweckmäßig, denn auf diese Weise wird die Datei dem Benutzer durch 'CLFL' gerade so zur Verfügung gestellt, als

wäre sie neu angelegt worden. 'CLFL' hat auf Textdateien also dieselbe Wirkung wie auf Datendateien (vgl. Abschnitt 2E).

Wollen Sie die Textdatei selbst tilgen, um das X-Memory für andere Zwecke freizumachen, müssen Sie den Dateinamen ins Alpha-Register legen (auch hier wie bei 'CLFL' zwingend vorgeschrieben) und 'PURFL' (*purge file*) ausführen. Beachten Sie die bezüglich 'PURFL' in Abschnitt 1F beschriebenen Einzelheiten, insbesondere die dort ausgesprochene WARNUNG.

3E. Die Dateifunktionen 'POSFL', 'SAVEAS' und 'GETAS'

Die Funktion 'POSFL' (*position in file*) sucht in der gegenwärtigen Textdatei, beginnend beim gültigen Zeigerstand, nach der Zeichenkette, die mit dem Inhalt des Alpha-Registers genau übereinstimmt. Dabei darf die Kette, nach der gesucht wird, nicht satzübergreifend sein, sondern muß vollständig in einem einzelnen Satz liegen. Wenn die Suche erfolgreich ist, wird der Dateizeiger auf das erste Zeichen der gefundenen Kette gesetzt und sein Wert ins X-Register gelegt. Ist die Suche dagegen erfolglos, gelangt die Zahl -1 ins X-Register. Eine Fehlermeldung gibt es in diesem Fall nicht. Wenn Sie also 'POSFL' in einem Program verwenden, müssen Sie die Abfrage 'X<0?' einbauen, um feststellen zu können, ob die gesuchte Kette gefunden wurde oder nicht. 'POSFL' kann sehr gut in Verbindung mit 'DELCHR' oder 'DELREC' eingesetzt werden, um Satzteile oder ganze Sätze zu löschen, oder in Verbindung mit 'INSCHR' oder 'INSREC', um Zeichenketten oder Sätze einzufügen.

Die Behandlung des Stapels durch 'POSFL' ist ganz ungewöhnlich. Auf dem HP-41CX wird der Stapel angehoben, und Register L bleibt unversehrt. Das ist ebenso, als ob 'RCLPT' an der Stelle ausgeführt worden wäre, an welcher die gesuchte Kette gefunden wurde. Auf dem HP-41C und dem CV hingegen, gilt diese Regel für 'POSFL' nur, wenn die Kette auch tatsächlich vorhanden war; andernfalls wird der Inhalt von X nach L gebracht und dann mit -1 überschrieben.

Wir wollen uns ein Beispiel vor Augen führen. Nehmen Sie an, der Name "HILL" soll in der Datei "NAMEN" aufgefunden werden. Dies ist mit den vorhandenen Mitteln leicht zu bewerkstelligen. Man tastet einfach

'0, SEEKPT, " HILL", POSFL'

und erhält als Ergebnis 1.005, wodurch angezeigt wird, daß die Leerstelle vor "HILL" das 6. Zeichen des zweiten Satzes bildet. Für die Suche wurde die Kette " HILL", also "HILL" einschließlich des davorstehenden Leerzeichens verwendet, um sicherzustellen, daß "HILL" nicht als Vorname oder Teil eines anderen Namens gefunden wird.

Die Funktionen 'SAVEAS' (*save ASCII file*) und 'GETAS' (*get ASCII file*) sind nur dann verwendbar, wenn Sie eine HP-IL-Massenspeichereinheit wie das Kassettenlaufwerk HP 82161A angeschlossen haben. Sie sollen daher hier nicht beschrieben werden. Wenn Sie ihrer bedürfen, schlagen Sie im Handbuch des X-Funktionen-Moduls nach. Sollten Sie den ständigen Gebrauch von ASCII-Dateien beabsichtigten, ist die Anschaffung einer HP-IL-Massenspeichereinheit allerdings sehr zu empfehlen. Mit 'SAVEAS' können Sie umfangreiche Textdateien sehr bequem daueraufzeichnen und mit 'GETAS' jederzeit umstandslos ins X-Memory lesen. Falls Sie die Inhalte zweier Textdateien ineinander mischen wollen, wofür

'SAVEAS' und 'GETAS' nicht vorgesehen sind, können Sie sich auf die in Abschnitt 3G vorgestellten Programme stützen.

3F. Inhalt einer Textdatei betrachten

Das nachstehend vorgeführte Programm "VAS" (*v*iew *AS*CII file) weist den gesamten Inhalt einer Textdatei vor und zwar Satz für Satz. Es benutzt einige X-Funktionen, die erst in Kapitel 4 besprochen werden, so daß Sie jenes Kapitel erst lesen müssen, bevor Sie die Arbeitsweise von "VAS" genau verstehen können.

Wenn Sie eine Textdatei betrachten wollen, legen Sie deren Namen ins Alpha-Register und führen 'XEQ "VAS"' aus. Bei angeschlossenem Drucker (und gesetztem Flag 21) wird der Inhalt der Datei Satz für Satz ausgedruckt, andernfalls nur angezeigt. Die unter 'LBL 10' stehende Unterroutine ist für die Entscheidung 'drucken oder anzeigen' zuständig. Sie bildet ein mustergültiges Beispiel für den Einsatz der X-Funktionen 'RCLFLAG' und 'STOFLAG' (Abschnitt 4C).

Nachstehend das Beispiel einer durch "VAS" bewirkten Druckausgabe:

```
RECORD 0:
DIESES BEISPIEL ZEIGT DR
UCK BZW. ANZEIGE, WIE SI
E DURCH "VAS" ERZEUGT WE
RDEN
RECORD 1:
KURZE SAETZE EINE ZEILE
RECORD 2:
LAENGERE SAETZE UEBER ME
HRERE ZEILEN VERTEILT
RECORD 3:
```

Wenn Sie eine Datei nur teilweise auflisten wollen, müssen Sie einen Satzzähler im 'ISG'-Format lmn.pqr (lmn ist die Nummer des ersten und pqr die des letzten zu betrachtenden Satzes) nach X legen. Dann füllen Sie das Alpha-Register mit dem Dateinamen und rufen "PVAS" (*p*artial *v*iew *AS*CII file) auf.

Mit Rücksicht auf die Benutzer der Version 1B des X-Funktionen-Moduls ist eine Fehlerprüfung in "VAS" eingebaut: Auf Zeile 05 erhalten Sie die Meldung "DATA ERROR", falls Sie das Programm bei leerem Alpha-Register aufrufen. Sie können dann die Angabe des Dateinamens nachholen und mit 'BST, R/S' fortfahren. Schlagen Sie noch einmal im Abschnitt 1F nach, um sich die Gefahr zu vergegenwärtigen, die bei der Version 1B besteht, wenn Sie unmittelbar zuvor 'PURFL' ausgeführt haben und danach der Befehl 'SEEKPTA' ein leeres Alpha-Register bei fehlender Arbeitsdatei vorfindet.

Programmausdruck von "VAS"/"PVAS"
(Barcode im Anhang D)

```
01♦LBL "VAS"    08 SEEKPTA      15 FIX 0        22 XEQ 10       29 SF 25        36 RDN
02 ,9                           16 ARCL Y       23 FS? 17       30 PRA          37 FS?C 25
                10 "RECORD "    17 STOFLAG      24 GTO 02       31 RCLFLAG      38 FC? 21
03♦LBL "PVAS"   11 LASTX        18 "┣:"         25 ISG L        32 FS?C 21      39 PSE
04 ALENG        12 INT          19 XEQ 10       26 GTO 01       33 FC?C 25      40 END
05 1/X          13 RCLFLAG                      27 RTN          34 AVIEW
06 RDN          14 CF 29        20♦LBL 02                       35 STOFLAG
07 INT                          21 GETREC       28♦LBL 10                       89 Bytes
```

Zeilen-Analyse von "VAS"/"PVAS"

Zeile 02 sorgt für einen Satzzähler der Gestalt 0.900, welcher – außer bei ungewöhnlich großen Dateien – das Vorzeigen aller Sätze sicherstellt. Er wird außer Kraft gesetzt, wenn Sie mit einem eigenen Zähler und "PVAS" arbeiten. Die Zeilen 04 und 05 bilden die Falltür zum Abfangen des durch ein leeres Alpha-Register entstehenden Fehlers (Alpha-Kette der Länge 0). Wenn Sie einen HP-41CX oder einen X-Funktionen-Modul von der Version 1C an aufwärts benutzen, können Sie diese beiden Zeilen löschen. Die Zeilen 07 und 08 setzen den Dateizeiger auf den Anfang des ersten zu betrachtenden Satzes. Die hinter 'LBL 01' stehende Befehlsfolge erzeugt im Alpha-Register die Meldung "RECORD n:", welche dann in Zeile 19 mit 'XEQ 10' angezeigt oder gedruckt wird. Die hinter 'LBL 02' stehende Befehlsfolge beginnt mit 'GETREC', wodurch (bis zu) 24 Zeichen des durch den Dateizeiger bestimmten Satzes ins Alpha-Register geholt werden. Anschließend zeigt oder druckt 'XEQ 10' diese Zeichen. Flag 17 wird gesetzt, sofern dabei (noch) nicht alle Zeichen des Satzes erfaßt wurden. Die Abfrage von Flag 17 in Zeile 23 bewirkt dann durch 'GTO 02' ein weiteres 'GETREC' auf denselben Satz. Ist Flag 17 gelöscht, der Satz also vollständig übertragen worden, wird der Satzzähler, welcher in Zeile 07 nach Register L gelangt ist, inkrementiert (Zeile 25). 'GTO 01' in Zeile 26 setzt das gesamte Verfahren für den nächsten Satz in Gang. Sobald ein Satzzähler, der von Ihrer Hand stammt, das Erreichen des letzten vorzuweisenden Satzes meldet, hält das Programm, 'GTO 01' überspringend, durch das 'RTN' in Zeile 27 an. Erreicht das Programm, weil Sie "VAS" aufgerufen haben, den letzten Dateisatz, wird die Ausführung notwendigerweise mit der Meldung "END OF FL" auf Zeile 21 abgebrochen. Das ist in Ordnung so und nicht etwa ein Fehlverhalten.

3G. Beschreiben von Magnetkarten mit Textdateien

Wenn Sie einen Kartenleser Ihr eigen nennen, können Sie das nachstehend angebotene Programm "WAS" (*w*rite *AS*CII file) dazu benutzen, eine Textdatei in Datenregister, deren Inhalte der Befehl 'WDTAX' (*w*rite *data* registers designated by *X*) dann auf Magnetkarten schreibt, zu übertragen. Das Programm "RAS" (*r*ead *AS*CII file) kehrt den gesamten Vorgang um. Beide Programme unterliegen nur einer unbedeutenden Einschränkung: die Textdatei darf keine 'NULL'-Bytes (Dezimalwert 0, vgl. dazu Abschnitt 4B), nicht zu verwechseln mit dem Leerzeichen 'SPACE' – dieses ist natürlich erlaubt, enthalten. Doch hat diese Einschränkung wenig zu sagen, denn 'NULL'en finden gewöhnlich keinen Eingang in ASCII-Dateien.

Um das Programm zu benutzen, legen Sie einfach den Dateinamen ins Alpha-Register und rufen "WAS" auf. Der Dateiname muß dem Programm vom Benutzer zur Verfügung gestellt werden, damit eine Fehlermeldung auf Zeile 03 vermieden wird. Die dort eingerichtete Fehler-

falle dient dazu, die durch den Befehl 'FLSIZE' drohende Gefahr, den Zugriff auf das X-Memory zu verlieren, falls unmittelbar zuvor 'PURFL' ausgeführt wurde, unter allen Umständen auszuschließen (vgl. Abschnitt 1F). Wenn Sie also die Fehlermeldung "DATA ERROR" erhalten sollten, müssen Sie den Dateinamen nachträglich angeben und dann mit 'BST, R/S' fortfahren. "WAS" sorgt selbständig dafür, daß es genügend Datenregister zur Aufnahme des Inhaltes der Textdatei gibt, indem erforderlichenfalls die Datenregisteranzahl mit der X-Funktion 'PSIZE' (Abschnitt 4D) heraufgesetzt wird. Falls Ihnen das Programm auf Zeile 20 "NO ROOM" entgegenhält, müssen Sie entweder ein oder mehrere Programme im Hauptspeicher löschen, um mehr Platz für Datenregister zu erlangen, oder auf "PWAS" (*p*artial *w*rite *AS*CII file) zurückgreifen. Der bei der Fehlermeldung "NO ROOM" in X liegende Wert gibt Ihnen an, welche Datenregisteranzahl im vorliegenden Fall von "WAS" benötigt wird.

Sobald der Kartenleser mit "RDY 01 OF lm" zur Karteneingabe auffordert, können Sie dem nachkommen, um die Datenregisterinhalte aufzuzeichnen, oder auch *zweimal* 'R/S' tasten, um eine solche Aufzeichnung zu vermeiden. Wenn "WAS" anhält, liegt im X-Register eine Zahl der Form 0.pqr, die besagt, daß der Inhalt der Textdatei in den Datenregistern R_{00} bis R_{pqr} untergebracht wurde. Die Gesamtanzahl der benötigten Datenregister beträgt also pqr + 1, während die Anzahl der zur Aufzeichnung benötigten Kartenspuren durch 1 + INT(pqr/16) gegeben wird.

Die Umkehrung "RAS" arbeitet ganz ähnlich: Sie legen (wieder zwingend vorgeschrieben) den Dateinamen ins Alpha-Register und führen 'XEQ "RAS"' aus. Die Aufforderung "CARD" wird entweder mit der Karteneingabe beantwortet oder mit 'R/S, R/S' quittiert, falls sich der ins X-Memory zu übertragende Text schon gestückelt in den Datenregistern befindet. Das Programm "RAS" 'weiß von sich aus', bis zu welchem Datenregister Inhalte zur Übertragung bereitstehen; Sie brauchen dies dem Programm nicht gesondert mitzuteilen.

"WAS" und "RAS" sind recht dienlich bei der Behandlung eines Problems, das einem beim Umgang mit ASCII-Dateien des öfteren sehr ungelegen querkommt. Nehmen Sie an, Sie haben eine ASCII-Datei von 50 Registern angelegt und damit begonnen, diese zu füllen. Plötzlich und unerwartet erhalten Sie die unerfreuliche Mitteilung "END OF FL". Die Datei ist voll! Vor Ihnen liegt ein dornenreicher Weg: Sie müssen die vollgelaufene Datei ganz löschen, eine neue größere anlegen und mit der lästigen Texteingabe von vorn beginnen. Doch halt! Es gibt ja doch noch eine Möglichkeit, die erneute Texteingabe zu umgehen: "WAS" schreibt Ihnen den Inhalt der Datei in Datenregister, die ihn getreulich aufbewahren, so daß sie sorglos 'PURFL' aufrufen und anschließend eine neue größere Textdatei anlegen können. Diese kann dann sogleich durch "RAS" mit dem geretteten Text gefüllt werden. So läßt sich eine Menge Tastenarbeit sparen.

Auf dem HP-41CX steht die Funktion 'RESZFL' zur Verfügung. Sie macht die Verwendung von "WAS" und "RAS" für das vorstehend beschriebene Problem überflüssig und gestattet eine elegantere Lösung. Zunächst müssen Sie die Datei, welche überzulaufen droht oder schon voll ist, zur Arbeitsdatei machen, falls sie das nicht schon sein sollte, was mit 'FLSIZE' oder 'RCLPTA', vorab den Dateinamen im Alpha-Register benennend, geschehen kann oder auch durch passende Unterbrechung des X-Memory-Verzeichnisses. Dann legen Sie die gewünschte neue Dateigröße nach X und führen 'XEQ "RESZFL"' aus. 'RESZFL' gestattet sowohl die Vergrößerung als auch die Verkleinerung (vgl. die Routine "DGM" in Abschnitt 3H) der Arbeitsdatei, letzteres allerdings nur, solange dabei keine Dateiinhalte verloren gehen (vgl. Abschnitt 3A). *Achtung:* Die Funktion 'RESZFL' arbeitet nur bezüglich der Arbeitsdatei; sie

läßt sich durch den Inhalt des Alpha-Registers nicht beeinflussen, sondern ändert unbeirrt die Größe der gegenwärtigen Datei ab. Prüfen Sie daher im Zweifelsfall das Ergebnis von 'RESZFL' mit 'EMDIR' (bzw. 'CAT 4').

Wenn Sie eine Datei nur teilweise auf Magnetkarten aufzeichnen wollen, legen Sie einen Satzzähler der Form abc.rst ins X-Register sowie den Dateinamen ins Alpha-Register und rufen danach "PWAS" auf. Dann werden die Dateisätze Nr. abc bis Nr. rst in Datenregister übertragen; sie werden zudem auf Magnetkarten aufgezeichnet, wenn Sie sich bei der diesbezüglichen Aufforderung des Magnetkartenlesers entsprechend entscheiden, nämlich Karten einführen. Die partielle Übertragung von Text ist insbesondere immer dann angezeigt, wenn "WAS" zuviele Datenregister beanspruchen würde oder wenn Sie Teile verschiedener Textdateien ineinander mischen möchten.

Wenn Sie in einer Textdatei einen bestimmten Bereich durch Text ersetzen wollen, der sich auf Magnetkarten befindet, können Sie dazu den Programmteil "PRAS" (*p*artial *r*ead *AS*CII file) verwenden. Ins X-Register gehört zuvor eine Kontrollzahl der Form abc.rst und ins Alpha-Register der Dateiname; dann wird "PRAS" aufgerufen. Die Inhalte der Dateisätze Nr. abc bis Nr. rst werden gelöscht und durch die von den Karten eingelesenen oder bereits in den Datenregistern liegenden Textteile, beginnend bei Register R_{00}, ersetzt. Zusammengehörige Textteile, die in benachbarten Datenregistern liegen und gemeinsam einen Satz bilden, werden dabei erkannt (vgl. die nachfolgende Zeilen-Analyse der Programmgruppe). Sollen die in den Datenregistern zerstückelt liegenden Sätze der vorhandenen Textdatei angefügt werden, muß die Kontrollzahl die Form lpq.9 haben, worin lpq größer als die Anzahl der vorhandenen Sätze der Datei ist und 900 – lpq größer als die Anzahl der anzufügenden Sätze. Die Anzahl der Sätze einer Datei läßt sich leicht ermitteln; vgl. Sie dazu die Übungsaufgabe 3.1 am Schluß dieses Kapitels.

Zeilen-Analyse von "WAS"/"PWAS"/"RAS"/"PRAS"

Die Zeilen 02 und 03 stellen sicher, daß der Programmlauf nicht mit leerem Alpha-Register beginnt, so daß die Funktion 'FLSIZE' in Zeile 05 nicht bezüglich einer Arbeitsdatei, die nicht gemeint ist oder gar überhaupt nicht existiert, arbeitet bzw. fehlschlägt. Die Funktion 'ALENG' wird in Abschnitt 4B besprochen. Sofern Sie "WAS" und "RAS" nur auf die jeweilige Arbeitsdatei anwenden wollen, müssen Sie die Zeilen 02, 03, 95 und 96 löschen. Doch berücksichtigen Sie das Risiko, welches Sie eingehen, wenn Sie einen X-Funktionen-Modul der Version 1B benutzen (Abschnitt 1F) und diese Fehlerfallen aus dem Programm herausnehmen.

Programmausdruck von "WAS"/"PWAS"/"RAS"/"PRAS"
(Barcode im Anhang D)

```
01♦LBL "WAS"
02 ALENG
03 1/X
04 SIZE?
05 FLSIZE
06 7
07 *
08 APPREC
09 DELREC
10 RCLPT
11 5
12 *
13 4
14 +
15 +
16 6
17 /
18 INT
19 X>Y?
20 PSIZE
21 0,9
22 ENTER↑
23 GTO 00

24♦LBL "PWAS"
25 ALENG
26 1/X
27 X<>Y
28 SIZE?
29 2
30 -
31 1 E3
32 /
33 X<>Y

34♦LBL 00
35 ENTER↑
36 INT
37 SEEKPTA
38 SF 25
39 DSE L

40♦LBL 01
41 GETREC
42 FC? 17
43 ISG L
44 FC? 25
45 GTO 03

46♦LBL 02
47 R↑
48 R↑
49 ASTO IND X
50 ISG X
51 X<0?
52 GTO 04
53 ALENG
54 11
55 ASHF
56 X<=Y?
57 GTO 02
58 RDN
59 5
60 X<Y?
61 FS? 17
62 GTO 01
63 GTO 02

64♦LBL 03
65 LASTX
66 R↑
67 CLA
68 ASTO IND X
69 INT
70 1 E3
71 /
72 SF 25
73 WDTAX
74 CF 25
75 RTN

76♦LBL 04
77 6
78 R↑

79♦LBL 05
80 DSE Z

81♦LBL 05
82 CLA
83 ARCL IND Z
84 RDN
85 ALENG
86 X=Y?
87 GTO 05
88 DSE L

89♦LBL 04
90 ISG Z
91 ACOS
92 GTO 03

93♦LBL "RAS"
94 CF 05
95 ALENG
96 1/X
97 CLFL
98 ,9
99 SIGN
100 GTO 08

101♦LBL "PRAS"
102 CF 05
103 ALENG
104 1/X
105 RDN
106 ENTER↑
107 INT
108 SF 25
109 SEEKPTA
110 FC? 25
111 GTO 07

112♦LBL 06
113 DELREC
114 FC? 25
115 GTO 07
116 ISG Y
117 GTO 06
118 CF 25

119♦LBL 07
120 APPREC
121 DELREC
122 RCLPT
123 X<>Y
124 SF 25
125 SEEKPT
126 CF 25
127 X<Y?
128 SF 05

129♦LBL 08
130 SF 25
131 RDTA
132 CF 25
133 CLX
134 SF 06
135 LASTX
136 6

137♦LBL 09
138 CLA
139 ARCL IND Z
140 ALENG
141 X=0?
142 FC? 06
143 X<0?
144 RTN
145 FC? 06
146 APPCHR
147 FC?C 06
148 CLA
149 FS? 05
150 INSREC
151 FC? 05
152 APPREC
153 X≠Y?
154 SF 06
155 X≠Y?
156 FC? 05
157 GTO 10
158 RDN
159 RCLPT
160 INT
161 ISG X

162♦LBL 09
163 SEEKPT

164♦LBL 10
165 RDN
166 ISG Z
167 ACOS
168 FS? 06
169 ISG Y
170 GTO 09
171 .END.
```

291 Bytes

Die Zeilen 05 bis 18 berechnen die Datenregisteranzahl, welche zur Speicherung des gesamten Inhaltes der Textdatei benötigt wird. Wir wollen uns diese Speicherung durch ein Beispiel veranschaulichen. Gehen Sie dazu vom derzeitigen Zustand unserer Musterdatei aus:

Satz-Nr.	Name
0	RICHARD NELSON
1	ROGER HILL
2	CLIFFORD STERN

Nach Aufruf von "WAS" wird dieser Inhalt folgendermaßen in den Datenregistern R_{00} bis R_{08} abgelegt:

Datenregister	Inhalt
00	"RICHAR"
01	"D J. N"
02	"ELSON"
03	"ROGER "
04	"HILL"
05	"CLIFFO"
06	"RD STE"
07	"RN"
08	"" (leere Zeichenkette)

Das Ende eines Satzes wird dadurch erkannt, daß im Datenregister eine Kette von weniger als sechs Zeichen abgelegt ist. Falls die Länge des Satzes ein Vielfaches von 6 war, ist die das Satzende kennzeichnende Zeichenkette eine leere. Befindet sich eine leere Zeichenkette in einem Datenregister, welches eigentlich den Anfang eines neuen Satzes enthalten müßte (im vorliegenden Beispiel Register R_{08}), wird dadurch das Ende der gesamten Datei signalisiert. Somit können in einem besonderen Fall, nämlich dann, wenn die Länge des *letzten* Satzes einer Datei ein Vielfaches von 6 beträgt, die *beiden* letzten Register der Registergruppe, in der die Datei Aufnahme gefunden hat, leere Zeichenketten enthalten: eine zeigt das Satzende, eine das Dateiende an.

Die vorstehend beschriebene Ablage einer Textdatei in Datenregistern bildet einen vernünftigen Kompromiß zwischen Verarbeitungsgeschwindigkeit und Registerverbrauch. Weiteres Verdichten von Text ist nur dann sinnvoll, wenn man sich der synthetischen Programmierung bedient (vgl. Abschnitt 10I).

Die zur Speicherung einer Textdatei benötigte Datenregisteranzahl hängt von der Dateigröße N (Anzahl der von der Datei besetzten Register im X-Memory) und der Anzahl S der Sätze der Datei ab. Das Programm muß also für die Datenregisteranzahl eine obere Schranke berechnen, die so beschaffen ist, daß auch im ungünstigsten Fall der gesamte Text einschließlich der u.U. zur Trennung erforderlich werdenden leeren Zeichenketten Platz im Hauptspeicher findet. Der ungünstigste, d.h. die meisten Trennregister erzeugende Fall liegt offenbar dann vor, wenn die ersten S − 1 Sätze sämtlich die Länge 6 haben, so daß jeder von ihnen durch eine leere Zeichenkette beendet werden muß, und der letzte Satz den verbleibenden Rest der Datei einnimmt, wobei seine Zeichenanzahl überdies ein Vielfaches von 6 ist. Berücksichtigt man das Textabschlußbyte (vgl. Abschnitt 3A) und die S Satzlängenbytes, zeigt eine leichte Überlegung, daß die Gesamtanzahl D der erforderlichen Datenregister durch

$$D = 2\cdot(S-1) + 1 + \mathrm{INT}\{(7N - 1 - S - 6\cdot(S-1) + 6)/6\}$$

$$= \frac{12}{6}(S-1) + \frac{6}{6} + \mathrm{INT}\{(7N - 7S + 11)/6\}$$

$$= \mathrm{INT}\{(7N + 5S + 5)/6\}$$

nach oben abgeschätzt wird, und zwar scharf, d.h. so, daß in Einzelfällen (z.B. N = 11, S = 4, 3 Sätze zu 6 Zeichen, 1 Satz mit 54 Zeichen, D = 17) kein Register bei dieser Abschätzung verschenkt wird. "WAS" berechnet in Zeile 05 die Dateigröße N und in den Zeilen 08 – 10 die Anzahl S der Sätze der Datei. Damit diese letztere Berechnung auch wirklich durchgeführt werden kann und das Programm nicht schon auf Zeile 08 mit der Meldung "END OF FL" unterbrochen wird, muß die Datei noch 8 Bytes (Satzlängenbyte + maximal 7 Zeichen) zur Aufnahme des Dateinamens, der vorübergehend den letzten Satz der Datei bildet, frei haben. Diese freien Bytes sind der Grund dafür, daß die vorstehend ermittelte Formel für D noch weiter verschärft werden kann: 4 statt 5 in Zeile 13. Die Zeilen 19 und 20 vergleichen D mit der gegenwärtig gültigen Datenregisteranzahl und setzen sie bei Bedarf herauf. Möglicherweise werden dann nicht alle Register zur Aufnahme des Textes benötigt, doch für die Bequemlichkeit, mit Sicherheit und ohne eigene Rechenleistung genügend Datenregister zur Verfügung zu haben, muß der Benutzer an dieser Stelle seinen Preis zahlen. In Zeile 21 wird der Sätzzähler auf 0.9 gesetzt, so daß alle Sätze übertragen werden.

Die hinter 'LBL 00' stehende Befehlsfolge stellt den Dateizeiger auf den Anfang des ersten zu übertragenden Satzes. Außerdem wird Flag 25 gesetzt, so daß 'GETREC' in Zeile 41 das Programm nicht unterbricht, wenn das Ende der Datei erreicht wird.

Die mit 'LBL 01' beginnende Schleife bringt bei jedem Durchlauf bis zu 24 Zeichen eines Satzes in die Datenregister. Sobald das Dateiende erreicht wird, löst ein vergebliches 'GETREC' die Löschung von Flag 25 aus, so daß das Programm nach 'LBL 03' verzweigt (Zeilen 44 und 45). Andernfalls mündet es in die mit 'LBL 02' beginnende Schleife, innerhalb welcher der Inhalt des Alpha-Registers zu Stücken der Länge 6 in die Datenregister befördert wird (Zeile 49).

Die sechs äußersten linken Zeichen des ersten Satzes gelangen nach R_{00}. Die Zeilen 42 und 43 inkrementieren den im Register L liegenden Satzzähler um 1 für jeden neuen ins Alpha-Register geholten Satz. Die Zeilen 53 – 57 veranlassen den Rücksprung auf 'LBL 02', um weitere sechs Zeichen in das nächste Datenregister zu übertragen, wenn im Alpha-Register 12 oder mehr Zeichen vorhanden waren, bevor 'ASHF' in Zeile 55 die sechs Zeichen aus dem Alpha-Register schob, die gerade (Zeile 49) gespeichert wurden. Waren höchstens 11 Zeichen vorhanden, dann wird vor dem nächstfälligen 'GETREC' noch höchstens ein weiteres 'ASTO' benötigt. Es gibt dann zwei Fälle: 1) waren höchstens fünf Zeichen vorhanden, ist der Satz bereits vollständig übertragen worden, so daß die Antwort 'nein' auf 'X<Y?' in Zeile 60 den notwendigen Sprung auf 'LBL 01' einleitet; 2) waren mehr als fünf Zeichen vorhanden, muß noch ein 'ASTO' stattfinden, um entweder den Satz vollständig zu übertragen (7 bis 11 Zeichen) oder die trennende Leerkette in ein Datenregister zu setzen (sechs Zeichen). Doch gibt es eine Ausnahme. Sie tritt ein, wenn 'GETREC' 24 Zeichen holt, aber damit nicht das Satzende erreicht, und sie wird am gesetzten Flag 17 erkannt. Die letzten sechs von 24 Zeichen sind dann zwar sehr richtig durch 'ASTO' in ein Datenregister gelangt, doch darf nicht noch einmal nach 'LBL 02' verzweigt werden, weil sonst zu Unrecht eine Leerkette als

Satzende-Marke entstünde; stattdessen muß ein weiteres 'GETREC' die nächsten Zeichen desselben Satzes holen. Dies geschieht durch die Abfrage in Zeile 61, die genau dann erreicht wird, wenn die Antwort auf 'X<Y?' in Zeile 60 'ja' lautet und das Programm deshalb wie gewünscht nach 'LBL 01' verzweigt. Ist Flag 17 dagegen gelöscht, der Satz also schon vollständig geholt, muß jedoch das Satzende, zwischen 0 und 5 Zeichen enthaltend, noch übertragen werden, sorgt Zeile 63 für den notwendigen letzten Sprung auf 'LBL 02'.

Die Marke 'LBL 03' leitet den Abschluß der Textübertragung ein. Sie wird von Zeile 45 aus angesprungen, sobald beim 'ISG L' in Zeile 43 die zugehörige Sprungbedingung entsteht oder Flag 25 gelöscht wird, weil am Dateiende ein vergebliches 'GETREC' stattfindet. Die Leerkette, welche im letzten Datenregister das Dateiende kennzeichnet, entsteht in den Zeilen 67 und 68, so daß später "RAS" feststellen kann, wo der von "WAS" abgespeicherte Text endet. Außerdem wird in den Zeilen 69 – 71 noch die für 'WDTAX' (*w*rite *data* registers designated by *X*) erforderliche Kontrollzahl 0.lmn hergestellt.

Das Programm "RAS" prüft am Anfang, ob das Alpha-Register nicht leer ist, damit 'CLFL' richtig arbeiten kann (vgl. Abschnitt 3D). Der Inhalt der benannten Textdatei wird durch 'CLFL' gelöscht und der Dateizeiger dabei gleichzeitig auf 0 gesetzt. Der Satzzähler erhält die Voreinstellung 0.9 und gelangt durch 'SIGN' (Zeile 99) nach Register L. "PRAS" beginnt ähnlich, doch wird der Inhalt der Datei hier nicht gelöscht. Sofern der Befehl 'SEEKPTA' (Zeile 109) nicht ausgeführt werden kann, nimmt das Programm an, daß die neuen Sätze anzufügen sind und keine gelöscht werden müssen. Es springt dann, diesen Fall am gelöschten Flag 25 erkennend, nach 'LBL 07'. Andernfalls mündet es in die mit 'LBL 06' beginnende Schleife, welche die Sätze löscht, die Sie durch die anfangs nach X gelegte Kontrollzahl (sie befindet sich inzwischen in Y) bestimmt haben. Für den Fall, daß versehentlich zuviele Sätze zur Löschung bestimmt wurden und daher ein "END OF FL" das Programm zur Unzeit unterbrechen würde, wird noch Flag 25 getestet (Zeile 114). In Zeile 128 wird Flag 5 gesetzt, falls der erste zur Löschung bestimmte Satz innerhalb der Datei liegt und nicht 'dahinter' (also eine Nummer trägt, die nicht größer als die Nummer des letzten Satzes ist). Das Flag wird dann später dazu benutzt, 'INSREC' bzw. 'APPREC' fallgerecht auszuführen (Zeilen 149 – 152).

Hinter 'LBL 08' werden (wenn gewünscht) Karten eingelesen, und Flag 6 wird gesetzt, um anzuzeigen, daß aus dem nächsten Datenregister der Anfang eines neuen Satzes ins Alpha-Register geholt wird (Zeile 139). Die Zahl 0 (Zeile 133) bildet den Anfangswert des später (Zeile 166) in Z liegenden Registerzählers; der in Zeile 135 aus L geholte Wert bildet später (Zeile 169) in Y einen Satzzähler für den Befehl 'ISG'; die 6 in X (Zeile 136) wird zum Vergleich mit dem Ergebnis von 'ALENG' (Zeilen 153 und 155) benötigt. Sobald die Länge der in einem Datenregister enthaltenen Kette kleiner als 6 ist, liegt das Ende eines Satzes vor.

Die mit 'LBL 09' beginnende Schleife holt aus dem Datenregister, dessen Adresse in Z liegt, eine Kette von bis zu sechs Zeichen. Ist deren Länge zwar 0, Flag 6 aber gesetzt, wodurch eigentlich der Anfang eines neuen Satzes angekündigt wird, muß genau das Datenregister vorliegen, dessen leerer Inhalt das Dateiende anzeigt. In diesem Fall wird die Zeile 144 erreicht, deren 'RTN' das Programm "RAS" ordnungsgemäß beendet. Ist Flag 6 dagegen gelöscht, wird der Alpha-Inhalt dem gegenwärtig im X-Memory entstehenden Satz durch 'APPCHR' (Zeile 146) angefügt. Bei gesetztem Flag 6 wird ansonsten – d.h. bei von 0 verschiedener Kettenlänge – der Alpha-Inhalt dazu benutzt, in Abhängigkeit vom Zustand des Flags 5 mit 'APPREC' oder 'INSREC' (Zeilen 149 – 152) einen neuen Satz zu beginnen. Ist die Kettenlänge dabei kleiner als 6, wird das in Zeile 147 in jedem Fall gelöschte Flag 6 wieder gesetzt (Zeile 154),

damit der als nächstes zu holende Datenregisterinhalt folgerichtig als Satzanfang behandelt wird. Die Zeilen 155 – 163 inkrementieren den Dateizeiger (nicht zu verwechseln mit dem Registerzähler in T oder dem Satzzähler in Z) genau dann, wenn sowohl Flag 5 als auch Flag 6 gesetzt sind, so daß das folgende 'INSREC' den nächsten Satz an der richtigen Stelle einfügt. Außerdem wird in Zeile 166 der Registerzähler inkrementiert, damit die nächste Zeichenkette aus dem richtigen Register kommt.

Anm. d. Übers.: a) In Zeile 54 kann man, wie eine leichte Überlegung zeigt, 11 durch 7 ersetzen. – b) "PWAS" arbeitet nicht einwandfrei, sobald der Satz, welcher dem letzten in Datenregistern abzulegenden Satz folgt, mehr als 24 Zeichen enthält. In diesem Fall wird wegen gesetzten Flags 17 die Zeile 43 ('ISG L') nicht erreicht, woraus sich eine falsche Adresse für das Dateiende-Register ergibt. Der Fehler ist jedoch durch die folgende Ausbesserung leicht zu beheben:

```
40♦LBL 01
41 GETREC
42 ISG L
43 FC? 25
44 GTO 03
45 FS? 17
46 DSE L
47♦LBL 02
```

c) Der von 'LBL 04' angeführte Programmteil wird immer dann erreicht, wenn die Anzahl der Datenregister, die für die Ausführung von "PWAS" benötigt werden, nicht ausreicht. In diesem Falle wird durch die 'LBL 05'-Schleife der Registerzähler soweit zurückgesetzt, daß das in Zeile 68 mit der leeren Zeichenkette beschickte Dateiende-Register hinter den letzten vollständig übertragenen Satz zu liegen kommt. Eine Warnung, die gegebenenfalls darauf hinweist, daß nicht alle durch "PWAS" zur Übertragung vorgesehenen Sätze in die Datenregister gelangt sind, enthält das Programm nicht. – d) Auch der pathologische Fall, daß bereits der erste von "PWAS" zu übertragende Satz keinen Platz in den zur Verfügung stehenden Datenregistern findet, führt zu einem Fehler. Hiergegen schafft das Einfügen von 'X<0?, CLX' hinter Zeile 66 Abhilfe.

3H. Zusätzliche Textdateifunktionen auf dem HP-41CX

Der HP-41CX hat zwei weitere X-Funktionen, die für den Umgang mit Textdateien vorgesehen sind. Die erste dieser Funktionen heißt 'ASROOM' (*AS*CII file *room*). Sie legt die Anzahl der Bytes, die in der im Alpha-Register benannten Textdatei noch verfügbar sind, nach X. Bei leerem Alpha-Register arbeitet 'ASROOM' bezüglich der Arbeitsdatei. Wenn Sie eine Textdatei im X-Memory liegen haben, der Sie aller Voraussicht nach selten etwas oder gar nichts mehr zufügen müssen, können Sie die folgende kleine Routine (*D*ateigröße *m*inimieren) dazu benutzen, den Register-Verbrauch der Datei auf die Mindestanzahl zu beschränken. Die Routine erwartet den Dateinamen im Alpha-Register:

```
01♦LBL "DGM"
02 FLSIZE        ermittelt die Anzahl der belegten Register
03 ASROOM        ermittelt die Anzahl der freien Bytes
04 7
05 /
06 INT           liefert die Anzahl der freien Register
07 -             berechnet die Anzahl der benutzten Register
08 RESZFL        paßt die Dateigröße der Anzahl der benutzten Register an
09 END
```

20 Bytes

Wenn Sie einen HP-41C oder CV haben, können Sie mit "WAS" und "RAS" arbeiten, um Ihren Text in einer Datei kleinster Größe unterzubringen. Doch ist es dazu notwendig, die Funktion 'ASROOM' zu simulieren, was mit der folgenden Routine "ASROOM", die den Dateinamen ebenfalls im Alpha-Register erwartet, geschehen kann:

```
01♦LBL "ASROOM"
02 ALENG         }
03 1/X           }   Fehlerfalle für die 'PURFL'-Wanze
04 RDN
05 FLSIZE            Registeranzahl der benannten Datei
06 7
07 *
08 0
09 SEEKPTA           setzt den Dateizeiger an den Dateianfang
10 +                 liefert die Gesamtanzahl der Bytes der Datei
11 SF 25             verhindert Fehlerstop auf Zeile 14
12♦LBL 01
13 CLA
14 GETREC
15 ALENG         }
16 -             }   subtrahiert die Anzahl der Zeichen des geholten Satzes
17 FC? 17        }
18 DSE X         }   subtrahiert 1 Byte für jeden Satz und 1 Byte am Dateiende
19 FS? 25        }
20 GTO 01        }   Rücksprung, sofern das Dateiende noch nicht erreicht ist
21 END
```

42 Bytes

Diese Routine liefert solange die richtige Anzahl freier Bytes, als sich in der Textdatei keine 'NULL'-Bytes befinden, die nach einem 'GETREC' zu führenden Bytes im Alpha-Register werden, weil 'ALENG' solche Bytes nicht berücksichtigen kann.

Die zweite der Textdateifunktionen des HP-41CX heißt 'ED' (*ed*it). Sie soll hier nicht näher beschrieben werden. Schlagen Sie im Bedienungshandbuch, das diese Funktion ausführlich beschreibt, nach. Sie werden dann feststellen, daß 'ED' das Tastenfeld des HP-41CX zu einem äußerst handlichen Werkzeug zur Behandlung von Textdateien macht, mit dem in einfachster Weise Zeichen oder Sätze eingefügt oder gelöscht werden können, und das ein freies, von einem blinkenen Positionszeiger optisch unterstütztes Umherwandern in den Dateien ermöglicht.

Für Besitzer des HP-41C oder CV wird in Kapitel 8 ein Programm namens "TE" (*t*ext *e*ditor) vorgestellt, das zwar langsamer als das durch 'ED' bereitgestellte Tastenfeld arbeitet, doch all die Möglichkeiten von 'ED' bietet, ja darüber hinaus weitere Eigenschaften hat. Sie werden "TE" vielleicht als sehr hilfreich für das Einrichten und Ändern von Textdateien schätzen lernen, selbst dann, wenn Ihnen das schnellere 'ED' zur Verfügung steht.

Aufgaben (Lösungen im Anschluß an Kapitel 11)

3.1 Schreiben Sie eine kurze Routine, die feststellt, wieviel Sätze die gegenwärtige Textdatei hat.

3.2 Schreiben Sie eine Routine, die eine Textdatei vollständig ausdruckt, und zwar je Zeile ein Satz (dies natürlich nur insoweit, als die Satzlänge keinen Überlauf der Druckzeile verursacht). Setzen Sie dabei voraus, daß das Alpha-Register beim Start den Dateinamen enthält.

KAPITEL 4

Weitere X-Funktionen

Nicht alle der im X-Modul sitzenden oder in den HP-41CX eingebauten X-Funktionen betreffen das X-Memory. 16 der 47 Funktionen des X-Moduls (25 der 61 Funktionen des HP-41CX) beuten weidlich das zugrunde liegende Betriebssystem aus. Sie erlauben z.B. den Zugriff auf einige der ansonsten nur dem Betriebssystem selbst zugänglichen in den Zustandsregistern befindlichen Kenngrößen, wie dies zuvor nur durch die Kunstgriffe der synthetischen Programmierung (Kapitel 10) möglich war, und helfen ganz erheblich beim Umgang mit Alpha-Ketten, Flags, Datenregistergruppen und Tastenzuweisungen. Eine dieser Funktionen, 'GETKEY', bietet gar die Möglichkeit, das Tastenfeld ganz nach den Bedürfnissen des Benutzers unter Programmkontrolle zu bringen; dies wird in den Anwendungsbeispielen (Kapitel 7, 8 und 9) vorgeführt.

4A. Stapelbeeinflussung und Verarbeitung von Eingaben

Es gibt einen bedeutsamen Unterschied zwischen X-Funktionen und Standardfunktionen (Katalog 3), auf den im Handbuch nicht ausdrücklich hingewiesen wird: Die meisten X-Funktionen, die einen in X liegenden Wert verarbeiten, lassen diesen Wert in X unversehrt zurück (einzig die Funktionen 'X<>F', 'POSA', 'POSFL' und 'GETKEY(X)' sind von dieser Regel ausgenommen). Darüber hinaus gelangt der verarbeitete Wert außer bei 'POSA', 'POSFL' und 'GETKEY(X)' nicht ins Register L. X-Funktionen sind in ihrem Verhalten also indirekten Funktionen wie z.B. 'ARCL IND X' wesentlich ähnlicher als direkten Funktionen wie z.B. '1/X'. Dieser Unterschied bei der Wirkung auf den Stapel sollte von Ihnen bei der Gestaltung Ihrer Programme bewußt wahrgenommen und ausgenutzt werden. Ungünstigstenfalls müssen Sie hier und da ein zusätzliches 'RDN' befehlen, um einen zurückbleibenden Eingabewert zu beseitigen. Besonders vorteilhaft ist die Möglichkeit, das Register L zur Aufnahme eines Schleifenzählers o.ä. verwenden zu können, weil es von den X-Funktionen, außer in den genannten Fällen, nicht verändert wird.

Jene X-Funktionen, die einen Wert nach X bringen, arbeiten bezüglich des Stapels so wie 'RCL'. Der Stapel wird also angehoben, es sei denn, 'CLX', 'ENTER↑' oder eine andere die Stapelbewegung außer Kraft setzende Funktion ist unmittelbar zuvor ausgeführt worden. Auch hier gibt es allerdings wieder Ausnahmen von der Regel, und zwar bei zwei Funktionen: 'POSA' überschreibt X und schiebt dessen Inhalt nach L. Die zweite Ausnahme begegnet uns bei der Funktion 'POSFL', jedoch nur auf dem HP-41C und CV. Sie überschreibt dort X und bringt dessen Inhalt nach L, wenn die gesuchte Zeichenkette nicht gefunden wird. War die

Suche dagegen erfolgreich oder benutzt man den HP-41CX, wird der Stapel angehoben, und L bleibt unberührt (vgl. auch Abschnitt 3E).

Eine weitere, allen betreffenden X-Funktionen gemeinsame Eigenschaft ist die, daß eine X-Funktion all jene Ziffern der Zahl in X unbeachtet läßt, die für sie ohne Belang sind. Gewöhnlich bedeutet dies, daß der gebrochene Anteil der Zahl in X ignoriert wird. In anderen Fällen haben nur die informationsbehafteten Nachkommastellen Bedeutung für die Funktion, wohingegen alle Nachkommastellen, die überflüssig sind, wirkungslos in X verweilen. Z.B. können Sie mit 'STOFLAG' den Zustand der Flags 36 – 39 wiederherstellen, wenn die in X liegende Zahl 36.39xxxxxx heißt, wobei die Werte der 6 letzten Ziffern, wie sie auch immer lauten mögen, keinerlei Einfluß auf das Ergebnis der Funktion nehmen. Diese Eigenschaft kann Bytes sparend ausgenutzt werden. In Abschnitt 2E beispielsweise hatten wir die Befehlsfolge

```
"DATEINAME"
0
SEEKPTA
.023
SAVERX
```

verwendet, um die Inhalte der Datenregister R_{00} bis R_{23} gemeinsam in eine Datendatei zu übertragen. Weil der Dateizeiger nur ganzzahlige Werte annehmen kann, interessiert sich die Funktion 'SEEKPTA' nun überhaupt nicht für die Nachkommastellen der Zahl in X. Darum hätte die Folge

```
"DATEINAME"
.023
SEEKPTA
SAVERX,
```

die um ein Byte kürzer ist, ebenso gereicht, um den beabsichtigten Zweck zu erzielen. Ein weiterer nicht unmittelbar zu erkennender Vorteil (vgl. Anhang B) ist der, daß 'SEEKPTA' auch noch schneller arbeitet, wenn .023 statt 0 in X liegt. Nicht immer gibt es so günstige Gelegenheiten, die verborgeneren Eigenschaften der X-Funktionen auszunutzen, doch berücksichtigen Sie stets, daß die Freiheit, die Sie haben, wenn Sie den X-Funktionen Eingaben zur Verarbeitung anbieten, i.a. dazu führt, Programme bezüglich des Speicherplatzbedarfes und der Geschwindigkeit günstiger zu gestalten.

Ein anderes beachtenswertes Merkmal hinsichtlich der Großzügigkeit, mit der die X-Funktionen abartig gestaltete Eingabewerte entgegennehmen, betrifft deren Vorzeichen. X-Funktionen machen keine Klassenunterschiede; sie betreiben keine Rassendiskriminierung: Negative Zahlen werden auf die gleiche Weise wie positive Zahlen behandelt. Nur 'AROT' natürlich, sowie – auf dem HP-41CX – 'RESZFL' (Abschnitt 2E) und 'GETKEYX' (Abschnitt 4G) ziehen zusätzliche Information aus einem negativen Vorzeichen. Die letzten beiden Funktionen benutzen es wie ein Flag, das bei 'RESZFL' eine wichtige Sicherungsschranke aufhebt und bei 'GETKEYX' ein vom Normalfall abweichendes Antwortverhalten bei der Tastenbedienung zuläßt. Gewinn kann man aus der Vorzeichen mißachtenden Verarbeitung der X-Inhalte insbesondere dort ziehen, wo negative Zählerwerte auftreten. Beispielsweise läßt sich umstandslos eine Funktion 'DSL' (*d*ecrement and *s*kip if *l*ess than zero)

simulieren, indem man 'ISG' auf negative ganze Zahlen anwendet. Dabei wird der absolute Wert der inkrementierten Zahl, also genau der Wert, welcher für die Arbeit von X-Funktionen maßgebend ist, dekrementiert, und der Sprung erfolgt, sobald der Wert 0 überschritten (vom Standpunkt der X-Funktionen aus unterschritten) wird.

4B. Behandlung von Texten im Alpha-Register

Ein 'nackter' HP-41C oder CV bietet nur sehr begrenzte Möglichkeiten, mit Texten umzugehen. Ein Alpha-Register von 24 Zeichen Länge und ein aus heutiger Sicht bescheidener Satz von Alpha-Befehlen ('ASTO', 'ARCL', 'ASHF' usw.), das ist schon alles. Ausreichend zwar, um ein Programm dazu zu veranlassen, Meldungen herauszuschreiben, Botschaften vorzulegen oder Ausgaben zu kommentieren, doch kaum geeignet, höhere Bedürfnisse zu befriedigen.

Das X-Memory zusammen mit den X-Funktionen fügt dem Beachtliches hinzu. Sie können Textdateien, die ganze Gruppen von Alpha-Ketten aufnehmen, einrichten, und Sie können diese Textdateien auf mannigfache Weise gezielt bearbeiten: ändern, ergänzen, verkleinern, aufrufen, durchsuchen. Darüber hinaus gibt es 6 weitere X-Funktionen, die sich auf Zeichenketten beziehen, und zwar auf solche, die Inhalt des Alpha-Registers, nicht Teil einer Textdatei sind. Diese 6 Funktionen, sie heißen 'ALENG', 'ANUM', 'AROT', 'ATOX', 'POSA' und 'XTOA', haben bemerkenswerte Fähigkeiten, wenn sie auch keine allzu ausufernde Zeichenkettenverarbeitung zulassen. Doch bedenken Sie: Der HP-41 ist 'von Natur aus' keine Textverarbeitungsmaschine, sondern ein Rechenknecht. Solange Sie dies nicht vergessen, werden Sie die Möglichkeiten, Alpha-Inhalte zu verarbeiten, sehr hoch einschätzen und zu dem Urteil gelangen, daß sie für ein Textfenster von 12 Zeichen Aufnahmevermögen wahrlich mehr als ausreichend sind.

Die Funktion 'AROT' (*A*lpha *rot*ate) vertauscht die Zeichen des Alpha-Inhaltes zyklisch, und zwar nach links um soviele Zeichen, als durch die ganze Zahl in X vorgegeben wird. Eine negative Zahl in X bewirkt eine nach rechts gerichtete Rotation des Alpha-Inhaltes. Der absolute Wert in X muß kleiner als 256 sein, andernfalls erscheint die Fehlermeldung "DATA ERROR".

Der Hauptzweck von 'AROT' besteht darin, ein bestimmtes Zeichen an eines der beiden Enden einer Alpha-Kette zu verbringen. Schiebt man beispielsweise ein ausgewähltes Zeichen an das linke Ende, kann es anschließend mit der Funktion 'ATOX' (siehe weiter unten) in seinen Dezimalwert gemäß ASCII-Verschlüsselung umgesetzt werden. Oder: ein einzelnes Zeichen, das gerade angefügt wurde, kann von seiner Position am rechten Ende mit '1, CHS, AROT' an den Anfang der Zeichenkette geholt werden.

Die Funktion 'AROT' senkt den Stapel nicht und läßt auch Register L ungeschoren. Die Steuerzahl für 'AROT' verbleibt in X, so daß Sie gewöhnlich ein 'RDN' werden anschließen müssen.

Die Funktion 'XTOA' (*X to A*lpha) hängt ein einzelnes Zeichen an das rechte Ende des Alpha-Inhaltes. Das Zeichen wird durch eine Dezimalzahl zwischen 0 und 255 in X festgelegt. Bei dieser Dezimalzahl handelt es sich um die sogenannte ASCII-Kodierung (*A*merican *s*tandard *c*ode for *i*nformation *i*nterchange) des Zeichens. Die Wechselbeziehung zwischen Anzeigesymbol, Druckerdarstellung und ASCII-Kode können Sie der auf den Seiten 49–50 abgedruckten Tabelle entnehmen.

Wenn das Alpha-Register 23 oder 24 Zeichen enthält, erklingt warnend 'TONE 7', sobald man ein weiteres Zeichen anfügt. Der Warnton läßt sich auch dann vernehmen, wenn 'XTOA'

für das Anfügen verwendet wird, und zwar — im Gegensatz zum Anfügen mit 'APPEND' — unabhängig davon, ob dies von Hand oder durch ein Programm geschieht, vorausgesetzt natürlich, daß das Tonflag 26 gesetzt ist. Enthält X jedoch eine Alpha-Kette, dann arbeitet 'XTOA' genau wie ein 'ARCL X' (einschließlich der für diesen Fall geltenden Mißachtung des Alpha-Überlaufes). Es ist aber i.a. besser, bei solchen Gelegenheiten 'ARCL X' statt 'XTOA' zu verwenden, weil durch 'ARCL X' in einem Programm-Ausdruck klarer bekundet wird, was geschehen soll. 'XTOA' arbeitet bezüglich des Stapels genau wie 'AROT': es senkt ihn nicht und läßt ebenso Register L unberührt.

Die Funktion 'XTOA' kann insbesondere vorteilhaft dazu benutzt werden, Alpha-Ketten aus gewöhnlichen und nicht-tastbaren Zeichen (z.B. runde Klammern oder Anführungszeichen) zusammenzusetzen, eine Kunst, die vor dem Erscheinen der X-Funktionen nur durch synthetisches Programmieren möglich war. Beispielsweise erzeugt die nachstehende Befehlsfolge im Alpha-Register den Text "X(1) = ":

```
"X"
40
XTOA
"⊢1"
1          } diese beiden Schritte machen Gebrauch von der Tatsache,
+          } daß der Stapel durch 'XTOA' nicht gesenkt wird
XTOA
"⊢ = "
```

Hieran können 'ARCL' und 'AVIEW' angeschlossen werden, um eine Zahl anzufügen und den zusammengesetzten Alpha-Inhalt dann in der Anzeige vorzuweisen. 'XTOA' kann ebensogut dazu verwendet werden, Zeichenketten zu bilden, die Kleinbuchstaben oder Sonderzeichen für Druckzwecke enthalten, wenn auch in der Druckerfunktion 'ACCHR' bereits eine Funktion zur Verfügung steht, mit der sich dasselbe erreichen läßt. Außer für "a" bis "e" entarten diese Zeichen in der Anzeige allerdings zu Vollzeichen ▩, denn LCD-Elemente mit nur 14 Segmenten bieten sehr begrenzte Darstellungsmöglichkeiten.

Entstehen Zeichenketten nicht erst im Verlaufe eines Programms, sondern ist ihre Zusammensetzung schon vorher bekannt, sollte man Textzeilen, sofern sie nicht-tastbare Zeichen enthalten, mit den Methoden der synthetischen Programmierung herstellen, weil dies in Hinsicht auf Speicherplatz und Programmgeschwindigkeit noch wesentlich günstiger ist als der Einsatz von 'XTOA' (beachten Sie die diesbezüglichen Literaturhinweise im Anhang C). Jedoch ist 'XTOA' genau der richtige Befehl, sobald ein oder mehrere Zeichen dem Alpha-Inhalt in Abhängigkeit von Ergebnissen im Programm angefügt werden sollen. Das in Kapitel 9 vorgestellte Programm "HP-16" benutzt diese Technik musterhaft in dem hinter 'LBL 08' liegenden Teil, in welchem die durch einen Basiswechsel erzwungene Umgestaltung einer Zahl vorgenommen wird.

Die Funktion 'ATOX' (*A*lpha *to X*) bildet fast genau die Umkehrung der Funktion 'XTOA'. Diese wandelt eine im X-Register liegende Dezimalzahl in das zugehörige Zeichen um und fügt es dem Alpha-Inhalt an, und zwar *rechts* — im Gegensatz zur Funktion 'ATOX', welche das äußerste *linke* Zeichen des Alpha-Inhaltes hernimmt und in die zugehörige Dezimalzahl umwandelt. Der Stapel wird von 'ATOX' angehoben; das Register L wird von 'ATOX' nicht berührt. Die Funktion 'ATOX' erfüllt neben ihrem Hauptzweck, Alpha-Zeichen zu entschlüsseln, auch noch häufig gute Dienste, wenn nichts weiter als ein Zeichen aus dem Alpha-Register zu entfernen ist. Mit 'AROT' läßt sich das zu entfernende Zeichen an den Anfang des Alpha-Inhaltes bringen und mit 'ATOX' dann beseitigen.

Die ASCII-Tabelle

Dezimal-kode	Anzeige-symbol	Druck-zeichen	Dezimal-kode	Anzeige-symbol	Druck-zeichen
0			32	(Leerz.)	(Leerz.)
1			33	!	!
2			34	"	"
3			35	#	#
4			36	$	$
5			37	%	%
6			38	&	&
7			39	'	'
8			40	(	(
9			41	)	)
10			42	*	*
11			43	+	+
12			44	,	,
13			45	-	-
14			46	.	.
15			47	/	/
16			48	0	0
17			49	1	1
18			50	2	2
19			51	3	3
20			52	4	4
21			53	5	5
22			54	6	6
23			55	7	7
24			56	8	8
25			57	9	9
26			58	:	:
27			59	;	;
28			60	<	<
29			61	=	=
30			62	>	>
31			63	?	?

Die ASCII-Tabelle

Dezimal-kode	Anzeige-symbol	Druck-zeichen	Dezimal-kode	Anzeige-symbol	Druck-zeichen
64	@	@	96	`	`
65	A	A	97	a	a
66	B	B	98	b	b
67	C	C	99	c	c
68	D	D	100	d	d
69	E	E	101	e	e
70	F	F	102	f	f
71	G	G	103	g	g
72	H	H	104	h	h
73	I	I	105	i	i
74	J	J	106	j	j
75	K	K	107	k	k
76	L	L	108	l	l
77	M	M	109	m	m
78	N	N	110	n	n
79	O	O	111	o	o
80	P	P	112	p	p
81	Q	Q	113	q	q
82	R	R	114	r	r
83	S	S	115	s	s
84	T	T	116	t	t
85	U	U	117	u	u
86	V	V	118	v	v
87	W	W	119	w	w
88	X	X	120	x	x
89	Y	Y	121	y	y
90	Z	Z	122	z	z
91	[	[	123	{	{
92	\	\	124	|	|
93	]	]	125	}	}
94	↑	↑	126	~	~
95	_	_	127	←	←

Bemerkungen zur ASCII-Tabelle

Wenn Sie einen HP-IL-Drucker benutzen, müssen Sie beachten, daß die Dezimalkodes 9, 10 und 27 eine andere Bedeutung haben, als sie die vorstehende Tabelle ausweist. Kode 9 erzeugt einen 'Zeilenabschluß', Kode 10 erzeugt einen 'Wagenrücklauf', und Kode 27 erzeugt ein 'Nachrichtensignal', welches ankündigt, daß die nachfolgenden Zeichen eine Meldung für den Drucker darstellen. Eine solche Druckerkontrollmeldung wird nicht gedruckt. [Anm. d. Übers.: Erschöpfende Auskunft über den Umgang mit dem HP-IL-Drucker und die Verwendung seiner Druck-Modi gibt ein gerade entstehendes Buch über Barcodes von Konrad Albers.]

Die Dezimalkodes 128 — 255 entarten in der Anzeige sämtlich zu Vollzeichen (alle Segmente einer Anzeigezelle eingeschaltet). Die Druckerzeichen für die Kodes 128 — 255 sind dieselben wie für die Kodes 0 — 127, ausgenommen die drei genannten Sonderkodes für den HP-IL-Drucker.

Der Dezimalkode 0 bildet das 'NULL'-Zeichen, welches man nicht mit der Meldung "NULL", die erscheint, wenn eine Taste überlang gedrückt wird, verwechseln darf. 'NULL' und "NULL" haben nichts miteinander zu tun. Das 'NULL'-Zeichen ist i.a. nur für die synthetische Programmierung von Bedeutung, bei normaler Benutzung des HP-41C begegnet es Ihnen so gut wie nie. Im Anhang C des Benutzerhandbuches für den X-Funktionen-Modul können Sie eine vollständige Zusammenfassung darüber finden, welche z.T. sehr merkwürdigen Erscheinungen die 'NULL' auslöst und was bezüglich ihrer Benutzung zu beachten ist.

Der Kode 255 hat ebenfalls einige Eigenschaften, die von denen der übrigen Kodes abweichen. Angezeigt als Teil des Alpha-Registers erscheint er als Vollzeichen — soweit in Ordnung. Doch eine mit 'ASTO' in ein Datenregister beförderte Zeichenkette, die den Kode 255 enthält, zeigt unter 'VIEW' ein verstümmeltes Erscheinungsbild: Vom Bytes 255 an, dieses selbst eingeschlossen, verschwinden die Zeichen aus der Anzeige. Allerdings nur aus der Anzeige, was Sie mit einem einfachen 'ARCL' sofort überprüfen können. Ferner ist eine Vorsichtsmaßregel vonnöten, sofern Sie einen X-Funktionen-Modul der Version 1B besitzen: Legen Sie in diesem Fall niemals mehr als 6 *aufeinanderfolgende* Kodes 255 in einer Textdatei ab. Andernfalls laufen Sie Gefahr, sowohl die damit beschickte als auch die nachfolgenden Dateien zu verlieren, sobald Sie eine weiter oben im X-Memory angesiedelte Datei mit 'PURFL' löschen. Und zwar deswegen, weil der HP-41 ein mit 7 Bytes FF angefülltes Register benutzt, um das Ende des im X-Memory belegten Bereiches zu kennzeichnen (vgl. Abschnitt 10C). Beim HP-41CX und bei X-Funktion von der Version 1C an aufwärts, brauchen Sie solche Vorsicht nicht mehr walten zu lassen.

Die Funktion 'ALENG' (*A*lpha *leng*th) berechnet die Anzahl der im Alpha-Register liegenden Zeichen. Ihr Ergebnis liegt mithin zwischen 0 und 24. Es wird, unter Anhebung des Stapels, ins X-Register gebracht. L bleibt unberührt. Wollen Sie beispielsweise genau dann auf 'LBL 99' springen, wenn das Alpha-Register leer ist, gelingt das ganz einfach mit

```
ALENG
X = 0?
GTO 99
```

Soll ein leeres Alpha-Register eine warnende Fehlermeldung auslösen, geschieht dies am schnellsten mit

```
ALENG
1/X  ("DATA ERROR" wenn x = 0)
```

Eine nicht ganz so am Wege liegende Anwendung von 'ALENG' besteht darin, Alpha-Ketten, die 'NULL'en enthalten, vor und nach einer zyklischen Vertauschung ihrer Zeichen mit 'AROT' auf ihre Länge hin zu überprüfen. Wenn bei der Vertauschung nämlich eine 'NULL' an das äußerste linke Ende der Kette gelangt, verschwindet sie, weil sie dort zur 'führenden', mithin zur unsichtbaren 'NULL' wird. Die einzige Möglichkeit das Wegtauchen einer oder mehrerer 'NULL'en bei der Rotation zu erkennen, besteht darin, die Verkürzung der Kette festzustellen. Sie werden 'ALENG' allerdings wohl erst dann zu diesem Zweck

einsetzen, wenn Sie nennenswert in die synthetische Programmierung (vgl. Abschnitt 10A) eingedrungen sind. Doch wenn Sie von jetzt ab in einem synthetischen Programm 'AROT' erblicken, dem ein 'ALENG' voransteht und ein anderes folgt, dann wissen Sie bereits, wofür das geschieht.

Die Funktion 'POSA' (*pos*ition in *A*lpha) nimmt einen Dezimalkode (ASCII-Tabelle) in X entgegen und durchsucht dann den Inhalt des Alpha-Registers nach dem entsprechenden Zeichen, und zwar von links nach rechts. Die Position, auf der das Zeichen das erste Mal angetroffen wird, gelangt in Form einer ganzen Zahl ins X-Register. Dabei wird der ursprüngliche Dezimalkode in X überschrieben – der Stapel bewegt sich also nicht – und kommt seinerseits ins Register L. Beachten Sie: 'POSA', 'POSFL' (Abschnitt 3E) und 'GETKEYX' (Abschnitt 4G) sind die einzigen X-Funktionen, welche das Register L verändern.

Der Positionskode, den 'POSA' ins X-Register legt, ist eine ganze Zahl zwischen 0 und 23. Hierbei zeigt die 0 an, daß das gesuchte Zeichen als äußerstes linkes Zeichen im Alpha-Register aufgefunden wurde, und 23 bedeutet, daß es einen 'Treffer' auf Position 24 gab. Wird das gesuchte Zeichen überhaupt nicht gefunden, gelangt der Wert -1 ins X-Register. Diese Regeln für den Positionskode mögen auf den ersten Blick etwas unverständlich anmuten, doch sind sie bewußt so gewählt worden; und zwar deswegen, weil man eine ganz bestimmte Anwendung dabei im Hinterkopf hatte: Will man nämlich das durch seinen ASCII-Kode festgelegte Zeichen, vorausgesetzt es ist im Alpha-Register vorhanden, an den Anfang des Alpha-Inhaltes bringen, wird die Aufgabe durch eine einfache Befehlsfolge so gelöst:

```
Zeichenkode ins X-Register setzen
POSA
X<0?      } Meldung "NONEXISTENT", wenn das
SF 99     } Zeichen nicht gefunden wird
AROT
```

Das ermittelte und nach vorn gebrachte Zeichen kann dann noch mit 'ATOX' beseitigt werden. Hat man beispielsweise Teilketten im Alpha-Register durch Trennzeichen voneinander abgesondert, kann man 'POSA', 'AROT' und 'ATOX' vereint einsetzen, um die Trennzeichen zu beseitigen.

Als Anwendungsbeispiel für 'POSA' wollen wir annehmen, Sie haben jemandes Namen in der üblichen Form 'Vorname, Leerstelle, Nachname' im Alpha-Register liegen und wollen ihn in die Form 'Nachname, Komma, Leerstelle, Vorname' umsetzen. Mit der folgenden einfachen Befehlsfolge gelingt das Kunststück:

```
"ULRICH GÜNTZER"  (als Beispiel)
"⊢, "     legt ein Komma und eine Leerstelle hinter den Nachnamen
32        ASCII-Kode der Leerstelle
POSA      ermittelt die Lage der (ersten) Leerstelle
AROT      bringt die (erste) Leerstelle nach vorn
ATOX      beseitigt die vorn stehende Leerstelle
```

Die Funktion 'POSA' sucht das Alpha-Register nicht nur nach einem durch seinen ASCII-Kode bestimmten Zeichen ab, sondern erlaubt auch die Suche nach einer Zeichenkette bis zur Länge 6. Für diesen Fall wird das X-Register statt mit einem Dezimalkode durch 'ASTO' mit einer Zeichenkette beschickt. Vollziehen Sie das folgende Beispiel nach:

```
"EI"
ASTO X
"JOSEF EIBNER"
POSA
```

Das Ergebnis lautet 6 und zeigt an, daß die Kette "EI" mit dem 7. Zeichen des Alpha-Inhaltes beginnt. "EI" gelangt, der Arbeitsweise von 'POSA' gemäß, ins Register L und steht somit auch nach dem Suchvorgang dem Benutzer zur Verfügung.

Die Funktion 'ANUM' (*a*lpha *num*ber) ist im wesentlichen eine Umkehrung der Funktion 'ARCL'. Der Hauptzweck von 'ARCL' besteht darin, dem Alpha-Inhalt eine Zahl anzufügen. Im Gegensatz dazu *entnimmt* 'ANUM' dem Alpha-Register eine Zahl. Liegt dort z.B. die Zeichenkette "A = 452", so bringt 'ANUM' die Zahl 452 ins X-Register. Der Stapel wird dazu angehoben; L bleibt unberührt.

'ANUM' durchsucht das Alpha-Register von links nach rechts und liefert die erste dabei auftauchende Teilkette, die entsprechend der Darstellungslogik des HP-41 als Zahl aufgefaßt werden muß, in X ab. Hierbei werden Punkte und Kommata gemäß der gerade gültigen Zahlendarstellung, bestimmt duch den Zustand der Flags 28 und 29, berücksichtigt. Sobald das Alpha-Register ein oder mehrere Punkte oder Kommata enthält, die sich *zwischen oder unmittelbar vor* Ziffern befinden, führt 'ANUM' daher zu Ergebnissen, deren Folgerichtigkeit man erst bei genauer Analyse erkennt. Auf den ersten Blick scheint die Wirkung von 'ANUM' leicht überschaubar: Wenn die Flags 28 und 29 beide gesetzt sind, wird ein Punkt als Dezimalpunkt und ein Komma als Zifferngruppierungszeichen aufgefaßt. Ist Flag 29 weiterhin gesetzt, Flag 28 hingegen gelöscht, wird – gemäß europäischer Gepflogenheit – der Punkt als Zifferngruppierungszeichen, das Komma jedoch als Dezimalkomma aufgefaßt. Bis hierhin ist alles wie zu erwarten war. Doch es wird verwickelter: Ist Flag 29 nämlich gelöscht, werden die Zifferngruppierungszeichen (ihre Gestalt hängt von Flag 28 ab) als Alpha-Zeichen behandelt. Sie erhalten also dieselbe Trennwirkung, welche die gewöhnlichen Alpha-Zeichen haben. Daher liefert ein auf den Alpha-Inhalt "12,003.05" angesetzter Befehl 'ANUM' das schlichte Ergebnis '12', wenn Flag 28 gesetzt und Flag 29 gelöscht ist, denn das gelöschte Flag 29 läßt das Komma zum reinen Alpha-Zeichen werden, welches die Zahl '12' von der Zahl '3.05' (der Punkt ist wegen des gesetzten Flags 28 Dezimalpunkt) trennt. Wird Flag 28 jetzt auch noch gelöscht, holt 'ANUM' die Zahl '12,003' ins X-Register ('FIX 3' einstellen, um dies zu erkennen!), denn nunmehr ist der Punkt trennendes Alpha-Zeichen, das Komma hingegen Dezimalkomma. Wird schließlich Flag 29 wieder gesetzt, erhält man merkwürdigerweise '12,00305' ('FIX 5' nicht vergessen). Dieses überraschende Ergebnis erklärt sich aus einer besonders abartig anmutenden Regel: Zifferngruppierungszeichen (das sind Kommata bei gesetztem und Punkte bei gelöschtem Flag 28) werden ignoriert, sofern sie nicht die gewöhnliche Dreiergruppierung im Vorkommabereich vornehmen, und zwar unabhängig davon, ob sie die Zahl im Alpha-Register im Vor- oder im Nachkommabereich 'verunreinigen', ja sogar gleichgültig, mit welcher Häufigkeit sie dabei auftreten. Das führt z.B. dazu, daß 'ANUM' aus einem so verkorksten Alpha-Inhalt wie "12...34,..5.6" das ansehnliche Ergebnis '1.234,56' filtriert, hier und in den folgenden Beispielen gelöschtes Flag 28 und gesetztes Flag 29 vorausgesetzt. Nicht nur das. Darüber hinaus wird allein das erste Dezimalkomma als solches anerkannt; folgende Kommata werden wie Zifferngruppierungszeichen betrachtet und gemäß der vorstehenden Regel wie Punkte ignoriert: aus dem Gebilde "1,.,,39" macht 'ANUM' die manierliche Zahl '1,39'. Selbst damit nicht genug der verwirrenden Arbeitsweise von 'ANUM': Wegen der Notwendigkeit nämlich, mit 'ANUM' eine Funktion bereitzustellen, die auch negative sowie sehr große und sehr kleine Zahlen (also mit Exponenten versehene Zahlen) dem

Alpha-Register zu entnehmen vermag, ergeben sich weitere Fallgruben für den unvorsichtigen Benutzer. So haben z.B. Vorzeichen keine Trennwirkung ("1 + + 3" führt zu '13'), mehrere Minuszeichen — nachgestellte eingerechnet! — heben sich wie in der Arithmetik gegeneinander auf (" – 1 + 13 – " ergibt '113'), und von mehreren Exponenten-E's wird nur das erste berücksichtigt, was bei verzwickter Verschränkung mit Minuszeichen, Punkten und Kommata zu zwar folgerichtigen doch schwer vorhersagbaren Resultaten führt: die chaotische Zeichenkette "2 – ,.7EE – E0E, .3 – " wird von 'ANUM' 'ganz einfach' als ' – 2.700' gedeutet.

Aus all dem folgt, daß man sehr sorgfältig planen muß, falls 'ANUM' Zahlen aus solchen Zeichenketten holen soll, deren Zusammensetzung nicht im voraus bekannt ist, sondern die erst durch ein Programm im Alpha-Register zusammengestellt werden und somit 'ANUM' die Gelegenheit bieten, den Benutzer intrigant mit unverwertbaren Ergebnissen zu bedienen. Die häufigste Fehlerquelle bildet ein gelöschtes Flag 29, weswegen man darauf achten sollte, 'ANUM' nur bei gesetztem Flag 29 aufzurufen.

Beachten Sie schließlich noch, daß 'ANUM' Flag 22 setzt, wenn die Suche nach einer Zahl erfolgreich war.

Aufgaben

4.1. Stellen Sie dem Alpha-Inhalt ein Zeichen, dessen Dezimalkode in X liegt, voran.

4.2. Bilden Sie eine kurze Befehlsfolge, die den Alpha-Inhalt mit 'ASTO' in Datenregister bringt, ohne dabei Register für leere Ketten zu verschwenden.

4.3. Bilden Sie eine kurze Befehlsfolge, mit der Sie n Zeichen des Alpha-Inhaltes
a) gerechnet von links aus
b) gerechnet von rechts aus
löschen können.

4.4. Verändern Sie die bei der Besprechung von 'POSA' vorgeführte Methode, Vor- und Nachnamen zu vertauschen und ein Komma einzuschieben, derart, daß auch ein *möglicherweise* vorhandener zweiter Vorname berücksichtigt wird.

4C. Handhabung von Flags

Unter den X-Funktionen befinden sich drei neue Befehle, die sehr hilfreich beim Umgang mit Flags sind. Die beiden wichtigsten lauten 'RCLFLAG' und 'STOFLAG'.

Betrachten wir folgende Lage: Sie schreiben ein Programm, welches ein Ergebnis in bestimmter Weise abgerundet vorzeigen soll, etwa im 'FIX 2'-Format. Nun hätten Sie es gern, daß das Programm den Rechner grundsätzlich in das ursprüngliche Format zurücksetzt, bevor es anhält. Vor Einführung der X-Funktionen war diese sehr einfach erscheinende Aufgabe nur umständlich zu lösen. Zunächst mußte der Anfangszustand der Flags, der ja unbekannt ist, durch aufwendige Tests am Beginn des Programms ermittelt werden (versuchen Sie einmal übungshalber, ein 'FIX n'-Format festzustellen, wenn Sie 'n' nicht kennen), und vor dem Ende waren entsprechend unwirtschaftliche Befehlsfolgen erforderlich, um den alten Zustand wiederherzustellen. Außerdem gab es noch das Problem, den ursprünglichen Zustand der Flags in irgendeiner Form während des Programmlaufs aufzubewahren.

Seit die X-Funktionen 'RCLFLAG' und 'STOFLAG' verfügbar sind, sind alle diese Schwierigkeiten aus der Welt. Man befiehlt einfach 'RCLFLAG', um den Zustand der Flags

in Form eines Registerinhaltes geliefert zu bekommen, ändert das Format nach Belieben und setzt den Rechner dann mit 'STOFLAG' in den ursprünglichen Zustand zurück. Eine typische dies bewirkende Befehlsfolge sieht z.B. so aus:

```
RCLFLAG
"ZINS: DM "
FIX 2
ARCL 01
AVIEW
STOFLAG
```

Die unter 'LBL 92' im Programm "NAP" des Kapitels 7 stehende Befehlsfolge ist ein sinnfälliges Beispiel für diese Technik. Wenden wir uns nun einigen Einzelheiten der Funktionen 'RCLFLAG' und 'STOFLAG' zu.

Die Funktion 'RCLFLAG' (*recall flags*) ruft eine – i.a. unverständliche – Alpha-Kette, die den gültigen Zustand der Flags 00 bis 43 darstellt, ins X-Register. 'RCLFLAG' arbeitet genauso, wie die gewöhnlichen 'RCL'-Befehle: der Stapel wird außer nach Befehlen, welche die Bewegung sperren ('ENTER↑', 'CLX' usw.), angehoben; L bleibt unberührt.

Die von 'RCLFLAG' hergestellte Alpha-Kette kann im Stapel verbleiben oder in beliebigen Datenregistern abgelegt werden. Sie dient lediglich dazu, später das 'Futter' für den Befehl 'STOFLAG' zu bilden, um den ursprünglichen Zustand der Flags wiederherzustellen.

Die Funktion 'STOFLAG' (re*st*ore *flags*) kann auf zweierlei Weise benutzt werden. Das voranstehende Beispiel führt die einfachere Möglichkeit vor. Man holt die Alpha-Kette mit 'RCLFLAG' ins X-Register, beläßt sie dort und führt zum richtigen Zeitpunkt einfach 'STOFLAG' aus. Die Flags 00 bis 43 werden dann wieder in den Zustand versetzt, in welchem sie sich bei der Ausführung von 'RCLFLAG' befanden.

'STOFLAG' erlaubt aber auch eine selektive Wiederherstellung des ursprünglichen Flagzustandes. Um die Funktion auswählend tätig werden zu lassen, wird die bewußte Alpha-Kette ins Y-Register gebracht und X mit einer Kontrollzahl der Form 'st,vw' (nicht 'st,uvw') beschickt. Wenn man bei so vorbereitetem Stapel 'STOFLAG' aufruft, werden genau die Flags F_{st} bis F_{vw}, diese selbst eingeschlossen, in den ursprünglichen Zustand zurückgesetzt. Sogar ein einzelnes Flag kann auf diese Weise behandelt werden; für solchen Zweck reicht es, die Zahl 'st' nach X zu legen.

Die Fähigkeit von 'STOFLAG', die Flags blockweise zu bearbeiten, ermöglicht es dem Benutzer, einzelne Flaggruppen, z.B. die allgemeinen Benutzerflags F_{00} bis F_{10}, oder jene, die den Anzeige-Modus oder den trigonometrischen Modus festlegen, gesondert zurückzusetzen. Um beispielsweise allein den Anzeige-Modus wiederherzustellen, könnte man diese Befehlsfolge verwenden:

```
RCLFLAG  }
STO 05   }  Flagzustand in R05 ablegen
.        )
.        }  Programmschritte, die den Anzeige-Modus verändern
.        )
RCL 05      Flagzustand nach X zurückholen
36,41       die Flags 36 – 41 bestimmen den Anzeige-Modus
STOFLAG     Flags 36 – 41 in den alten Zustand versetzen
```

Wie man leicht sieht, kann man diese Technik dahingehend verallgemeinern, in einem Programm mit verschiedenen durch 'RCLFLAG' erzeugten Alpha-Ketten, die verschiedenen Flag-Kombinationen entsprechen, zu arbeiten. Die Alpha-Ketten können in verschiedenen Datenregistern untergebracht werden. Je nach Bedarf ruft man sie dann herbei und läßt 'STOFLAG' die gewünschten Flagzustände schlagartig herstellen. Es ist klar, daß dadurch die Programmausführung u.U. erheblich vereinfacht und beschleunigt werden kann.

Eine weitere Anwendung des Befehlspaares 'RCLFLAG'/'STOFLAG' führt die folgende kurze Routine vor. Sie druckt den Alpha-Inhalt, wenn der Drucker eingeschaltet und druckbefugt ist (Flag 21 gesetzt), andernfalls wird ein 'AVIEW' mit 'PSE' ausgeführt. Dies ist gegenüber einem einfachen 'AVIEW' vorteilhafter, weil

1. keine Programmunterbrechung stattfindet, wenn Flag 21 gesetzt, der Drucker jedoch ausgeschaltet ist, und Sie
2. nicht gezwungen werden, eine langsam durch die Anzeige rollende Laufschrift geduldig entgegenzunehmen, falls der Drucker druckbefähigt ist, mithin seinem schnellen Wirken der Vorzug gegeben werden muß.

Dieselbe Befehlsfolge wurde bereits im Programm "VAS" (Abschnitt 3F) verwendet. Hier sei sie als eigenständige Routine "ADA" (*A*lpha-Inhalt *d*rucken oder *a*nzeigen) wiedergegeben:

```
01♦LBL "ADA"
02 SF 25
03 PRA                  Alpha-Register (bei gesetztem F21) ausdrucken
04 RCLFLAG
05 FS?C 21              F21 löschen, damit 'AVIEW' nicht unterbricht
06 FC? 25
07 AVIEW                'AVIEW' bei erfolglosem Druckversuch oder gelöschtem F21
08 STOFLAG
09 RDN
10 FS?C 25
11 FC? 21               'PSE' bei erfolglosem Druckversuch oder gelöschtem F21
12 PSE
13 END                                                    29 Bytes
```

Mit 'RCLFLAG'/'STOFLAG' kann man übrigens den HP-41 überlisten und einen sonst unzugänglichen Anzeige-Modus, nämlich das Format 'FIX/ENG' (Flags 40 *und* 41 gesetzt), herstellen. Dieses Format verhält sich wie das gewöhnliche 'FIX', solange die Zahl in X zu ihrer Darstellung keines Exponenten bedarf. Im Gegensatz zu 'FIX' jedoch, welches zur 'Verlegenheitslösung' eines scheinbaren 'SCI' greift, wenn die Darstellung nicht mehr regelgemäß möglich ist, schwenkt 'FIX/ENG' in ein scheinbares 'ENG' um (scheinbar, weil Flag 40 gesetzt bleibt). Tasten Sie die folgende Routine "FEX" ('*F*IX/*E*NG'-Format gemäß *X*) ein, legen Sie eine Zahl zwischen 0 und 9 zur Bestimmung der Nachkommastellen ins X-Register, führen Sie "FEX" aus, und stellen Sie dann Versuche mit dem so erzeugten 'uneigentlichen' Anzeige-Modus an.

```
01♦LBL "FEX"
02 ENG 0                    Flag 41 setzen
03 RCLFLAG
04 FIX IND Y                Flag 40 setzen und Nachkommastellen festlegen
05 X<>Y
06 RDN
07 41                       } Flag 41 setzen, ohne die anderen Flags zu verändern
08 STOFLAG
09 R↑
10 R↑
11 END                                                    24 Bytes
```

Die dritte X-Funktion, welche Zugriff auf mehrere Flags zugleich verschafft, heißt 'X<>F' (*X* exchange *f*lags). Sie behandelt die Benutzerflags F_0 bis F_7 zusammen wie den Inhalt eines 'Kleinstregisters' F und erlaubt einen Datenaustausch zwischen diesem F und dem Register X. F kann nur ganze Zahlen zwischen 0 und 255 aufnehmen. Gemäß den in Abschnitt 4A beschriebenen Regeln verhält sich 'X<>F' als friedliche und willige X-Funktion, indem Sie Vorzeichen und Nachkommastellen der in X liegenden Zahl außer acht läßt, so als ob ihr die Schritte 'ABS' und 'INT' vorgeschaltet wären, mit der (u.U. sehr willkommenen) Einschränkung jedoch, daß Register L nicht berührt wird, im Gegensatz zu den 'echten' Befehlen 'ABS' und 'INT'. Darum kann ein doppeltes 'X<>F, X<>F' gelegentlich 'wesensfremde Dienste' übernehmen, nämlich dazu verwendet werden, eine Zahl zwischen $-255{,}9999999$ und $+255{,}9999999$ mit 'ABS' und 'INT' zu bearbeiten, ohne den Inhalt von L zu verändern. Für Zahlen außerhalb dieses Bereiches erhalten Sie die Fehlermeldung "DATA ERROR".

'X<>F' erweist sich insbesondere deswegen als kraftvolle Funktion, weil man vermöge ihrer die Information über den Zustand mehrerer Benutzerflags in einem Datenregister aufbewahren kann.

Nun zu den Einzelheiten. Sobald 'X<>F' ausgeführt worden ist, befinden sich die Flags F_0 bis F_7 in einem Zustand, welcher der Binärdarstellung der zuvor in X gelegenen Zahl x entspricht. Mathematisch ausgedrückt

$$x = \sum_{i=0}^{7} 2^i \cdot f_i,$$

worin $f_i = 1$ für gesetztes und $f_i = 0$ für gelöschtes Flag F_i ist. Aus dieser Binärdarstellung ergibt sich für gesetztes Flag F_0 der Summand 1, für gesetztes Flag F_1 der Summand 2, für gesetztes Flag F_2 der Summand 4, usw. Nachstehend ein tabellarisch ausgeführtes Beispiel, aus welchem die Äquivalenz der Dezimalzahl 133 mit dem Flagzustand 'F_0, F_2, F_7 gesetzt' ersichtlich ist:

Flag	Wert des gesetzten Flags	gesetzt	Summand
00	1	ja	1
01	2	nein	0
02	4	ja	4
03	8	nein	0
04	16	nein	0
05	32	nein	0
06	64	nein	0
07	128	ja	128
		Summe:	133

Ganz offenkundig läßt sich also mit 'X<>F' sehr Nützliches vollbringen. Sollen etwa am Anfang eines Programms die Flags F_0 bis F_3 gelöscht und die Flags F_4 und F_5 gesetzt werden, so kann man statt der schweißtreibenden Folge

```
CF 00
CF 01
CF 02
CF 03
SF 04
SF 05
```

(12 Bytes)

'kurzerhand'

```
48          (F4 ≙ 16, F5 ≙ 32)
X<>F
```

(4 Bytes)

befehlen. Sollte Ihnen die eigenhändige Berechnung der benötigten Dezimalzahl Pein bereiten oder verständlicherweise unter Ihrer Würde sein, lassen Sie den HP-41 selbst tätig werden; dazu ist er schließlich da:

```
0
X<>F      } löscht 'im Handumdrehen' F0 bis F7
SF 04
SF 05
X<>F
```

Als Ergebnis schreibt Ihnen der HP-41 48 ins X-Register, also gerade die Zahl, die gebraucht wird, um mit 'X<>F' genau die Flags F_4 und F_5 zu setzen und die restlichen Flags der 8-er Gruppe zu löschen.

Nehmen wir an, daß aus irgendeinem Grunde in dem voranstehenden Beispiel der Zustand von F_6 und F_7 eine Rolle spielt und daher bewahrt bleiben soll, dann läßt sich mit der Befehlsfolge

```
0
X<>F
64
/
LASTX     } addiert den Summenbeitrag von F4 und F5
X<>Y
INT
*
48
+         } ermittelt den Summenbeitrag von F6 und F7
X<>F
```

diesem Umstand Rechnung tragen. Sie löscht F_0 bis F_3, setzt F_4 und F_5 und beläßt F_6 und F_7 in ihrem Zustand. Analysieren Sie die Befehlsfolge als Übung. Sie erreichen damit volle Einsicht in das Wirken von 'X<>F'. Doch lassen Sie sich nun nicht dazu verleiten, 'X<>F' begeistert überall einzusetzen! Die Durchsicht der voranstehenden Folge vermittelt nämlich auch die folgende lehrreiche Erkenntnis: Das direkte Löschen und Setzen der 6 Flags F_0 bis F_5 ist um drei Bytes 'billiger' als die übereifrige Verwendung von 'X<>F'.

Aufgaben

4.5. Schreiben Sie eine Befehlsfolge, welche die Funktion $f(x) = \sin(\pi \cdot x)/\pi \cdot x$ berechnet. Die Berechnung muß im Bogenmaß ('RAD'-Modus) durchgeführt werden, aber anschließend soll der – *unbekannte* – ursprüngliche trigonometrische Modus wieder gelten.

4.6. Ersinnen Sie eine Befehlsfolge, die das ungebräuchliche Format 'FIX/ENG' einstellt, wobei die Anzahl der Nachkommastellen, festgelegt durch F_{36} bis F_{39}, unverändert bleiben soll.

Anwendungen von 'RCLFLAG'/'STOFLAG' in der synthetischen Programmierung

Bei angeschlossenem Drucker wird die Programmausführung verlangsamt. Dem kann dadurch abgeholfen werden, daß man Flag 55 synthetisch löscht. In diesem Zustand verbleibt es solange, als das Programm läuft. Hält es an, werden beim Drucker HP 82143A die Flags 55 und 21 gesetzt. Beim Drucker HP 82162A wird Flag 55 wieder gesetzt, wenn das Programm einer Druckerfunktion begegnet *und* Flag 21 gesetzt war. Die folgende kurze von Steve Wandzura stammende Routine löscht Flag 55, ohne die Zustände anderer wichtigen Flags zu verändern:

RCLFLAG	
SIGN	bringt die Alpha-Kette nach L und löscht X
STO d	löscht *alle* Flags, insbesondere F_{55}
X<> L	holt die Alpha-Kette nach X
STOFLAG	stellt den Zustand von F_0 bis F_{43} wieder her
RDN	stellt die Stapelordnung (bis auf T) wieder her

Die genaue Bauart der durch 'RCLFLAG' erzeugten Alpha-Kette lautet hexadezimal

$$1F\ Ff_0\ f_1f_2\ f_3f_4\ f_5f_6\ f_7f_8\ f_9f_{10}.$$

Hierin stellen die 11 halben Bytes f_0 bis f_{10} Hexadezimalziffern dar, deren Wert durch den Zustand der Flags F_0 bis F_{43} bestimmt wird. Das einzelne f_i enthält die Information über F_{4i} bis F_{4i+3}. Beispielsweise hat f_0 den Wert A, wenn F_0 und F_2 gesetzt sowie F_1 und F_3 gelöscht sind. Der Inhalt von Register d wird also um 1 1/2 Bytes nach rechts verschoben. Diese Tatsache kann für synthetische Programme, in denen Verschiebungen *um 1/2 Byte* gefragt sind, Bedeutung erlangen. Beispiel: Aus dem rechtsbündig in M liegenden Byte-Paar N_1N_2 N_3N_4 (N_i ein Nybble) soll das Byte N_2N_3 zusammengezimmert und in X abgelegt werden. Lösung:

```
01♦LBL "N2N3"          09 ARCT
02 "┣AA"               10 "┣A"
03 RCL [               11 R↑
04 X<> d               12 R↑
05 RCLFLAG             13 X<> d
06 CLA                 14 RCL \
07 STO [               15 END
08 5
```

In der in Anm. a) am Ende von Abschnitt 10C stehenden Routine "DAT?" wird von dieser Möglichkeit, 'RCLFLAG' 'außerdienstlich' einzusetzen, nutzbringend Gebrauch gemacht.

4D. Erweiterte 'SIZE'-Funktionen

Zwei der neuen X-Funktionen gestatten dem Benutzer, die Datenregisteranzahl *programmgesteuert* festzustellen und zu verändern. Sie bilden eine äußerst zweckmäßige Hilfe, denn vor Einführung der X-Funktionen war Vergleichbares nur innerhalb der synthetischen Programmierung möglich.

Die Funktion 'SIZE?' (*size* of data registers?) ermittelt die Anzahl der gegenwärtig verfügbaren Datenregister und legt diese Zahl ins X-Register. Der Stapel wird wie bei 'RCL'-Funktionen behandelt: er wird angehoben; L bleibt unberührt.

'SIZE?' ist ein klassisches Beispiel dafür, daß auch als ausgereift zu beurteilende Betriebssysteme, die unter der Aufsicht erfahrener Konstrukteure entstanden sind, wesentlicher und eigentlich selbstverständlicher Funktionen ermangeln können. Wenn Sie je mit einem HP-41 ohne X-Funktionen umgegangen sind, wissen Sie das. Wie oft wohl war damals ein Benutzer gehalten, die Datenregisteranzahl umständlich vor einem Programmstart zu prüfen, um ja ärgerliche Programmunterbrechungen, gemeldet mit "NONEXISTENT", auszuschalten. Das gewöhnliche Verfahren bestand darin, zahlreiche Probe-'RCL's auszuführen, bis man schließlich auf das erste "NONEXISTENT"-Register stieß. Eine ermüdende Beschäftigung. Um sich dabei nicht die Finger wund zu tasten, konnte man zwar ein Programm wie z.B.

```
01♦LBL "SZFIND"        07 RTN
02 CLX                 08 RDN
03 SF 25               09 1
04♦LBL 01              10 +
05 RCL IND X           11 GTO 01
06 FC? 25              12 END
```

27 Bytes

mit der Suche beauftragen, doch selbst dessen Tätigkeit konnte einen — insbesondere bei größerer Datenregisteranzahl — unmutig werden lassen. Sogar die pfiffigsten der nicht-synthetischen Verfahren, die auf *I*ntervall*s*chachtelung beruhen und gegenüber der vorstehenden einfachen Methode bei größerer Datenregisteranzahl einen erheblichen Geschwindigkeitsvorteil für sich verbuchen können, sind unbefriedigend, denn sie verbrauchen viel zu viel Platz. Aus historischen Gründen sei hier noch ein Beispiel aus der Trickkiste vergangener Tage vorgeführt:

```
01♦LBL "IVS"           14 X≠0?
02 128                 15 GTO 01
03 ENTER↑              16 RDN
04♦LBL 01              17 SF 25
05 SF 25               18 STO IND X
06 STO IND Y           19 X<0?
07 FC? 25              20 CLX
08 CHS                 21 FC?C 25
09 ST+ Y               22 RTN
10 ABS                 23 1
11 2                   24 +
12 /                   25 END
13 INT
```

Laufzeit: 2,8 Sek., unabhängig von der Datenregisteranzahl

43 Bytes

Die Funktion 'SIZE?' dagegen ist unvergleichlich viel schneller als solche tapfer erkämpften Umwege und obendrein natürlich wesentlich praktischer. Sie verdient es, mit 'ASN' zugewiesen zu werden, um ihr Dauerwohnrecht auf dem Tastenfeld zu verschaffen.

Die Funktion 'PSIZE' (*p*rogrammable *size*) leistet dasselbe wie 'SIZE' mit dem Unterschied, daß sie nicht mit drei Unterstrichen zur Eingabe auffordert, sondern stattdessen den in X liegenden Wert x – genauer INT(ABS(x)) – verarbeitet. 'PSIZE' kann darüber hinaus, und das ist entscheidend, als Programmbefehl eingesetzt werden. Bezüglich der Verwendung in Unterprogrammen gibt es für 'PSIZE' keine Einschränkung: selbst in einer auf sechster Stufe laufenden Unterroutine arbeitet diese hilfreiche Funktion einwandfrei, ohne irgendwelche schädliche Wirkung auf die darüberliegenden aufrufenden Programme auszuüben. Das ist nicht ganz selbstverständlich, denn die sogenannten Rücksprungadressen verändern sich ja in einem solchen Falle während des Programmlaufs.

Die folgende kurze Befehlsgruppe prüft, ob die gegenwärtige Datenregisteranzahl für einen bestimmten Zweck ausreicht, und vergrößert sie erforderlichenfalls gemäß der vorab nach X zu legenden Zahl:

```
(erforderliche Anzahl in X)
SIZE?
X<>Y
X>Y?
PSIZE
```

Eine andere Spielart, die ebenfalls die Mindestanzahl der benötigten Datenregister in X erwartet, gibt ein akustisches Zeichen, das den Benutzer auf die unmittelbar bevorstehende Datenregisteranpassung aufmerksam machen soll, damit er diese, falls er sich kurzfristig noch eines anderen besinnt, im letzten Moment mit 'R/S' unterbinden kann:

```
(erforderliche Anzahl in X)
SIZE?
X>Y?
GTO 01
TONE 9
X<>Y
PSE
PSIZE
LBL 01
```

4E. Umgang mit Datenregister-Blöcken

Die X-Funktionen 'REGMOVE' und 'REGSWAP' gestatten es dem Benutzer, die Inhalte ganzer Blöcke von Datenregistern auf einen Schlag zu kopieren oder gegeneinander auszutauschen, ja sogar die Inhalte eines Blockes zyklisch zu vertauschen. Auf dem HP-41CX lassen sich außerdem Datenregister-Blöcke mit 'CLRGX' löschen, eine Aufgabe, die auf dem HP-41C oder CV leicht mit einer kurzen Routine zu lösen ist, wenn auch nicht so blitzschnell, wie die Funktion 'CLRGX' dies vollbringt.

Die Funktion 'REGMOVE' (*reg*ister *move*) erwartet in X eine Kontrollzahl der Form 'abc,qrslmn'. Sie kopiert die Inhalte eines Blockes der Länge 'lmn', wobei das erste Register des Quellblockes die Nummer 'abc' trägt, und das erste Register des Zielblockes 'qrs' lautet. Ist 'lmn' Null, wird dennoch ein Register kopiert, so als ob 'lmn' = 1 gewesen wäre. Die Funktion verändert weder den Stapel noch Register L. Beispiel: 'REGMOVE' mit '6,001003'

in X bringt die Inhalte der Register R_{06}, R_{07} und R_{08} (Quellblock) nach R_{01}, R_{02} und R_{03} (Zielblock).

Greifen Sie jetzt auf die Routine "LADEREG" aus Abschnitt 2A zurück, und setzen Sie 'SIZE 020' fest, damit sich die folgenden Beispiele bequemer nachvollziehen lassen. 'XEQ "LADEREG"' beschickt die Register mit ihren eigenen 'Etiketten', so daß Sie schnell 'frisches Spielmaterial' bekannter Gestalt zur Hand haben und die Wirkung von 'REGMOVE' und 'REGSWAP' mühelos verfolgen können.

Als einfaches Beispiel für die Wirkung von 'REGMOVE' wählen wir

```
XEQ "LADEREG"
3,007006
REGMOVE
```

Dabei werden die Inhalte des Quellblockes $R_{03} - R_{08}$ in den Zielblock $R_{07} - R_{12}$ befördert, und zwar in einer Reihenfolge, die durch die folgende Tabelle (sie enthält nur dort Einträge, wo Veränderungen stattfinden) veranschaulicht wird:

Register:	03	04	05	06	07	08	09	10	11	12
Startbelegung	3	4	5	6	7	8	9	10	11	12
Schritt 1:										8
Schritt 2:									7	
Schritt 3:								6		
Schritt 4:							5			
Schritt 5:						4				
Schritt 6:					3					
Ergebnis:	3	4	5	6	3	4	5	6	7	8

Aus dieser Tabelle lassen sich die einzelnen Zwischenschritte ablesen, und es wird verständlich, warum der Vorgang trotz der Überlappung von Quell- und Zielblock wunschgemäß verläuft: er beginnt beim *letzten* Register des Zielblockes, so daß das entsprechende Register des Quellblockes, R_{08}, noch seinen ursprünglichen Inhalt besitzt; erst zu einem späteren gefahrlosen Zeitpunkt, Schritt 5, wird dieser Inhalt überschrieben. Der ganze Vorgang läuft sinngemäß mit '7,003006' als Kontrollzahl ab: er beginnt in diesem Fall beim *ersten* Register des Zielblockes, bei R_{03}.

Die Funktion 'REGSWAP' (*reg*ister *swap*) tauscht die Inhalte zweier Datenregister-Blöcke gegeneinander aus. Sie erwartet genau wie 'REGMOVE' eine Kontrollzahl der Form 'abc,qrslmn' in X, wobei die drei Zifferngruppen dieselbe Bedeutung wie dort haben, nur daß jetzt beide Blöcke zugleich Quell- und Zielblock sind. Ist 'lmn' = 0 nimmt der HP-41 auch hier an, daß 'lmn' = 1 gemeint war, und tauscht die Inhalte der beiden Register R_{abc} und R_{qrs} gegeneinander aus. Stapel und Register L bleiben unverändert.

Verfolgt man die interne Arbeitsweise von 'REGSWAP', stellt man fest, daß genau wie bei 'REGMOVE' Registerpaare Schritt für Schritt verarbeitet werden. Es gilt wieder dieselbe Regel: ist abc < qrs, beginnt der Austausch bei den höchstnumerierten Datenregistern; ist dagegen abc > qrs, werden zuerst die Register mit den niedrigsten und zuletzt die mit den höchsten Adressen verarbeitet. Gewöhnlich ist es nicht nötig, sich der Einzelheiten dieses Ablaufs bewußt zu sein, doch kann ihre Kenntnis in besonderen Fällen der Überlappung von Quell- und Zielblock nutzbringend verwendet werden, weil die Reihenfolge der Schritte das Ergebnis beeinflußt: Führen Sie zunächst

```
XEQ "LADEREG"
3,007006
REGSWAP
```

aus. Die folgende Tabelle zeigt die Durchführung der einzelnen Schritte:

Register:	03	04	05	06	07	08	09	10	11	12
Startbelegung:	3	4	5	6	7	8	9	10	11	12
Schritt 1:						12				8
Schritt 2:					11				7	
Schritt 3:				10				6		
Schritt 4:			9				5			
Schritt 5:		12				4				
Schritt 6:	11				3					
Ergebnis:	11	12	9	10	3	4	5	6	7	8

Man erkennt, daß 'REGSWAP' die Inhalte der betroffenen Blöcke regelrecht miteinander 'verrührt'. Daraus läßt sich unverhoffter Vorteil ziehen. Wir machen uns die Arbeitsweise von 'REGSWAP' zunutze, um eine zyklische Vertauschung der Registerinhalte eines Blockes vorzunehmen. Und das so:

```
XEQ "LADEREG"
4,003009
REGSWAP
```

Betrachten wir wieder die einzelnen Schritte. Wegen $4 > 3$ beginnt der Austausch mit dem Registerpaar R_{03}/R_{04} und endet bei R_{11}/R_{12}. Folglich ergibt sich:

Register:	03	04	05	06	07	08	09	10	11	12
Startbelegung:	3	4	5	6	7	8	9	10	11	12
Schritt 1:	4	3								
Schritt 2:		5	3							
Schritt 3:			6	3						
Schritt 4:				7	3					
Schritt 5:					8	3				
Schritt 6:						9	3			
Schritt 7:							10	3		
Schritt 8:								11	3	
Schritt 9:									12	3
Ergebnis:	4	5	6	7	8	9	10	11	12	3

Mithin sind die Inhalte des aus 10 Registern bestehenden Blocks $R_{03} - R_{12}$ um ein Register in abwärts steigender Richtung zyklisch vertauscht worden. Mit '3,004009' in X hätten Sie eine zyklische Vertauschung in aufwärts steigender Richtung erreicht.

Die gewonnene Erkenntnis läßt sich leicht in eine allgemeine Regel fassen: Um die Inhalte eines mit R_{abc} beginnenden Blockes von lmn Registern um ein Register zyklisch zu vertauschen, muß der Funktion 'REGSWAP' eine Kontrollzahl in X zur Verfügung gestellt werden, die bei aufwärts gerichteter Vertauschung

$$abc,(abc + 1)(lmn - 1)$$

und bei abwärts gerichteter Vertauschung

$$(abc + 1),abc(lmn - 1)$$

lautet.

Diese Regel läßt sich noch weiter verallgemeinern: Besitzt der in Rede stehende Block lmn = k Register und soll eine zyklische Vertauschung um t Register stattfinden, so kann man dies mit einem einzigen Befehl 'REGSWAP' bewirken, sofern sich die Registeranzahl k ohne Rest durch die Vertauschungslänge t teilen läßt. Die erforderlichen Kontrollzahlen lauten in diesen Fällen

$$abc,(abc + t)(lmn - t) \quad \text{(aufwärts gerichtete Vertauschung)}$$

$$(abc + t),abc(lmn - t) \quad \text{(abwärts gerichtete Vertauschung).}$$

Beispiel für k = 12 und t = 4:

```
XEQ "LADEREG"
5,001008
REGSWAP
```

füllt die Register $R_{01} - R_{12}$ mit den Zahlen 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4.

Die im HP-41CX zusätzlich enthaltene Funktion 'CLRGX' (*cl*ear *r*egisters designated by *X*) nimmt eine in X liegende Kontrollzahl der Form 'abc,pqrij' entgegen und löscht gemäß dieser den Datenregisterblock $R_{abc} - R_{pqr}$, wobei 'ij' als Inkrement benutzt wird, so daß R_{abc} das erste von der Löschung betroffene Register ist, das Endregister R_{pqr} aber möglicherweise ungelöscht bleibt. Gibt man keinen Wert 'ij' an ('ij' = 0), wird die Voreinstellung 'ij' = 1 wirksam, dergemäß alle Register von R_{abc} bis R_{pqr}, dieses selbst eingeschlossen, gelöscht werden. 'CLRGX' verarbeitet also die in X liegende Kontrollzahl nach genau den Regeln, die hinsichtlich der Arbeitsweise der Standardfunktion 'ISG' gelten. Will man beispielsweise die Register R_{04} bis R_{08} löschen, wird dies mit

```
4,008
CLRGX
```

bewerkstelligt. Sollen dagegen die Register R_{01}, R_{03}, R_{05}, R_{07} und R_{09} gelöscht werden, muß man

```
1,00902
CLRGX
```

befehlen. Zwar wird ein Inkrement 'ij' $\neq 0$ (also > 1) selten erforderlich sein. Nichtsdestoweniger gibt es Probleme, bei denen die Möglichkeit, ein solches vorschreiben zu können, nützlich ist. Etwa beim Umgang mit Matrizen, deren Elemente in fortlaufend numerierten Datenregistern liegen. Unter Verwendung von 'ij' (> 1) lassen sich ausgewählte Spalten oder Diagonalelemente gezielt löschen. In einer 5×5-Matrix z.B., die in den Registern $R_{01} - R_{25}$ liegt, setzt man die Diagonale $a_{1,1}, \ldots, a_{5,5}$ mit der Kontrollzahl 1,02506 auf Null.

'CLRGX' läßt die Kontrollzahl unbeschädigt in X zurück und berührt Register L nicht.

Auf dem HP-41C oder CV muß man zu einem kleinen Hilfsprogramm Zuflucht nehmen, um zu erreichen, was die Funktion 'CLRGX' leistet. Das folgende einfache Programm erwartet, genau wie 'CLRGX', eine Kontrollzahl der Form 'abc,pqrij' in X und löscht damit, unter Verwendung der Funktion 'ISG', die Register R_{abc} bis R_{pqr}, wobei ein u.U. angegebenes Inkrement 'ij' mitberücksichtigt wird.

```
01♦LBL "BC"
02 SIGN          bringt die Kontrollzahl nach Register L
03 CLX           erzeugt den zu speichernden Wert 0
04♦LBL 03
05 STO IND L     löscht das durch den Stand der Kontrollzahl bestimmte Register
06 ISG L         inkrementiert die Kontrollzahl
07 GTO 03
08 END
```

Wenn es in Ihren Anwendungen erforderlich sein sollte, sehr große Blöcke von Datenregistern zu löschen, können Sie statt "BC" das folgende Programm "BCΣ" (*b*lock *c*lear using 'CLΣ'), welches sich der Funktion 'CLΣ' bedient und darum schneller zum Ziel führt, nehmen. Um z.B. 100 Register zu löschen, benötigt "BC" eine Laufzeit von knapp 13 Sekunden, "BCΣ" hingegen weniger als 4 Sekunden. Beachten Sie jedoch, daß das Programm selbst die von Ihnen nach X gelegte Kontrollzahl mit dem Wert '06' inkrementiert, so daß nur einfache Kontrollzahlen der Form 'abc,pqr' zu brauchbaren Löschvorgängen führen.

Programmausdruck von "BCΣ"
(Barcode im Anhang D)

```
01♦LBL "BCΣ"      07 GTO 02      13♦LBL 02      19♦LBL 03
02 6 E-5          08♦LBL 01      14 DSE X       20 STO IND Y
03 +              09 CLΣ         15♦LBL 02      21 ISG Y
04 ΣREG IND X     10 ΣREG IND X  16 LASTX       22 GTO 03
05 ISG X          11 ISG X       17 -           23 END
06 X<0?           12 GTO 01      18 0                          44 Bytes
```

Aufgabe

4.7. Schreiben Sie ein kurzes Programm, das die Inhalte eines Blockes von lmn Datenregistern, beginnend mit R_{abc}, zyklisch vertauscht, und zwar um r Register aufsteigend oder absteigend, je nachdem ob r positiv oder negativ ist. Es soll vorausgesetzt werden, daß zu Programmbeginn die Adresse 'abc' des ersten Registers in X, die Blocklänge lmn in Y und

die Vertauschungslänge r in Z liegen. Benutzen Sie keine Datenregister, sondern nur den Stapel und Register L. (Lassen Sie sich nicht dazu verführen, die Lösung für so einfach zu halten wie sie scheint.)

4F. Die Kontrolle von Tastenzuweisungen

Zwei X-Funktionen, 'CLKEYS' und 'PASN', erweitern Ihre Möglichkeiten im Umgang mit Tastenzuweisungen, die im USER-Modus wirksam sind. Eine andere auf das Tastenfeld bezogene X-Funktion, 'GETKEY' (auf dem HP-41CX außerdem noch 'GETKEYX'), erlaubt Ihnen sogar, die Tasten völlig programmabhängig zu 'lesen', weil man mit ihr eine nur auf das jeweilig laufende Programm zugeschnittene Reaktion auf die Tastenbedienung erzielen kann.

Zunächst ein paar Bemerkungen über Tastenzuweisungen. Die Möglichkeit, eine beliebige Funktion nach eigenem Gutdünken einer bestimmten frei wählbaren Taste zuzuordnen, gehört zu den Merkmalen, die bessere programmierbare Taschenrechner von schlichteren Modellen abhebt. Der HP-65 und der HP-67, beides programmierbare Vorgänger des HP-41, besaßen eine Tastenreihe – es war die oberste –, die mit den Marken 'A' bis 'E' (bei Umschaltung 'a' bis 'e') versehen war. Ein Druck auf eine dieser Tasten veranlaßte den Rechner, den Teil des Programmspeichers abzuarbeiten, dem die auf der Taste enthaltene Marke voranstand.

Der HP-41 übertrifft hinsichtlich der Tastenzuweisungsmöglichkeiten alle seine Vorgänger bei weitem. Der Benutzer kann die Bedeutung fast jeder Taste nach Belieben neu festsetzen, solle sie nun zur Ausführung einer Funktion oder zum Aufruf eines Programms, angeführt durch eine globale Marke, dienen (globale Marken sind diejenigen, die durch den Katalog 1 vorgewiesen werden; die Namen von Funktionen erscheinen in den Katalogen 2 und 3). Um der früheren Möglichkeiten nicht verlustig zu gehen und insbesondere die Übertragung von Programmen der vorangegangenen Modelle auf den HP-41 zu gewährleisten, hat die oberste Tastenreihe des HP-41 dieselbe Aufgabe wie bei dessen Vorgängern, und zwar – und das ist bereits eine beachtliche Erweiterung – in dem Programm, in welchem sich der Benutzer gerade befindet. Man hat also die Marken 'A' bis 'E' und 'a' bis 'e' und zusätzlich noch 'F' bis 'J' in der zweiten Tastenreihe zu lokalen Marken gemacht, die nunmehr in *jedem* als geschlossene Einheit auftretenden Programm als USER-Ansprungstellen dienen können. Der Zugriff auf die lokalen Marken 'A' – 'J'/'a' – 'e' ist allerdings nur insoweit möglich, als den entsprechenden Tasten keine Funktion oder globale Marke zugewiesen sein darf. Die Gesichtspunkte, nach denen der HP-41 seine Suche nach Marken durchführt, werden im Benutzerhandbuch unter dem Stichwort 'Lokale Marken' beschrieben. Dies sind die wesentlichen Merkmale: Wenn eine Taste der obersten beiden Reihen, bei der obersten Reihe auch nach Umschaltung, im USER-Modus gedrückt wird, und wenn einer solchen Taste keine Funktion oder globale Marke zugewiesen ist, begibt sich der HP-41 im gegenwärtigen Programm auf die Suche nach der entsprechenden lokalen Marke. Findet er sie, wird das Programm an der durch die Marke angeführten Stelle gestartet. Sie können das leicht durch Verweilen auf der Taste überprüfen: es erscheint 'XEQ', gefolgt von der jeweiligen Marke.

Beiläufig bemerkt kann die Suche nach lokalen Marken u.U. recht lange dauern, wenn keine lokalen Marken im gegenwärtigen Programm vorhanden sind. Es ist daher sinnvoll, einige häufig benutzten Funktionen wie z.B. 'X<>Y' und 'RDN' ihrer eigenen Taste zuzuweisen, wenn man im USER-Modus verbleiben will und dennoch deren schnelle Ausführung wünscht. Zugewiesene Funktionen oder globale Marken werden nämlich gegenüber lokalen Marken bevorzugt behandelt, so daß die Antwortzeit, weil überhaupt keine Suche nach lokalen Marken mehr stattfindet, erheblich verkürzt wird. All dies – das muß Ihnen

klar sein — gilt natürlich nur für den USER-Modus.

Die Funktion 'PASN' (*p*rogrammable *ASN*) arbeitet ganz ähnlich wie die Funktion 'ASN', welche allerdings allein von Hand über das Tastenfeld ausgeführt werden kann. Bei 'ASN' müssen Sie in den ALPHA-Modus schalten, um den Namen der Funktion, die zugewiesen werden soll, zu 'buchstabieren': Genauso bei 'PASN', nur daß hierbei der Funktions- oder Programmname *vor* dem Aufruf der X-Funktion ins Alpha-Register gelegt werden muß. Bei 'ASN' bestimmen Sie die Taste, welcher die Funktion zugewiesen werden soll, dadurch, daß Sie sie im Anschluß an das Buchstabieren des Funktionsnamens direkt drücken. Dabei erscheint in der Anzeige kurz der zugehörige Tastenkode. Dessen erste Ziffer benennt die Zeile (zwischen 1 und 8), in der die Taste liegt, die zweite Ziffer die Reihe (zwischen 1 und 5). Genau diesen Tastenkode müssen Sie ins Register X legen, bevor Sie 'PASN' aufrufen.

Mit 'ASN' löscht man von Hand eine Zuweisung, indem man 'leer buchstabiert', also einfach 'ALPHA, ALPHA' drückt, ohne dazwischen Zeichen einzutasten. Die leere ALPHA-Eingabe veranlaßt den HP-41, Ihrem Wunsch gemäß die Taste von einer darauf liegenden Zuweisung zu befreien. 'PASN' arbeitet wieder sinngemäß: Das Alpha-Register muß *leer* sein und der Tastenkode in X liegen, bevor die Funktion aufgerufen wird.

Die voranstehenden Regeln — Funktionsname ins Alpha-Register bzw. Alpha-Register leer *und* Tastenkode nach X *vor* Aufruf von 'PASN' — gelten gleichermaßen für die Ausführung von Hand und für die Ausführung in einem Programm. Letztere Möglichkeit — das *programmierte* Zuweisen und Löschen von Zuweisungen — macht, wie wir gleich sehen werden, den HP-41 einmal mehr zu einem ganz persönlichen Werkzeug, dessen Handhabung völlig den augenblicklichen eigenen Bedürfnissen, und seien sie noch so ausgefallen, angepaßt werden kann.

Mit 'PASN' können Sie also Programme schreiben, die 'in eigener Verantwortung' Tastenzuweisungen tätigen. Angenommen, Sie haben ein Programm vorliegen, welches Textdateien auf den neuesten Stand bringen soll. Dann kann es sinnvoll sein, etwa die Funktionen 'INSREC' und 'DELREC' des bequemen Zugriffs halber auf Tasten liegen zu haben, sobald Sie mit dem Programm zu arbeiten beginnen. Weil diese Funktionen andererseits nicht wichtig genug sind, um Tasten ständig mit ihnen zu besetzen, werden Sie den Streit um den Anspruch darauf (schließlich bedeutet es Ruhm und Ehre für eine Funktion, 'Sitz und Stimme' auf dem Tastenfeld verliehen zu bekommen) ganz einfach mit 'PASN' schlichten: Sie beginnen Ihr Programm mit der Zuweisung der benötigten Funktionen und schließen es mit deren Löschung (leeres Alpha-Register) ab. Und schon herrscht Friede zwischen den miteinander um die Tasten wetteifernden Funktionen.

Die folgende kurze Routine stammt von Alan McCornack. Sie löscht die fünf Zuweisungen — natürlich nur, sofern solche überhaupt vorhanden sind —, welche auf den nicht-umgeschalteten Tasten der obersten Tastenreihe liegen. Die verwendete Technik kann umstandslos anderen Verhältnissen angepaßt werden.

```
01♦LBL "KZL"        ('Kopfzeile löschen')
02 CLA
03 11,015           'ISG'-Zähler für die Tastenkodes 11 bis 15
04♦LBL 01
05 PASN             Taste, deren Kode in X liegt, freistellen
06 ISG X
07 GTO 01
08 END                                          24 Bytes
```

Die Funktion 'CLKEYS' (*cl*ear *keys*) löscht sämtliche im USER-Modus wirksamen Zuweisungen von Funktionen oder globalen Marken. Beachten Sie, daß im Anschluß an 'CLKEYS' die Suche nach lokalen Marken ('A' bis 'J' und 'a' bis 'e') nicht mehr durch bevorzugt behandelte Zuweisungen unterdrückt wird. Ein 'Kahlschlag' mit 'CLKEYS' ist allerdings ein viel zu rauhes Verfahren zur Lösung alltäglicher Zuweisungskonflikte. Für den Wechsel von Zuweisungen oder die Freistellung von Tasten liegen Sie mit der gezielten Verwendung von 'PASN' fast immer besser. In Abschnitt 10G werden Sie zusätzlich eine synthetische Methode, die eigentlich keine Wünsche mehr offen läßt, kennenlernen.

Nachfolgend ein sehr einfaches Programm zur Anwendung von 'PASN' und 'CLKEYS'. Angenommen, Sie arbeiten i.a. mit zwei verschiedenen Gruppen von Tastenzuweisungen, abhängig davon, wofür Sie Ihren HP-41 gerade einsetzen. Dann können Sie sich zwei kleine Programme schreiben, welche die jeweils benötigten Zuweisungen vornehmen und deren jedes die folgende allgemeine Gestalt hat:

```
LBL "TF1"              ('Tastenfeld 1')
CLKEYS
"FUNKTION 1"
(Tastenkode 1)
PASN
"FUNKTION 2"
(Tastenkode 2)
PASN
.
.
.
END
```

Weil 'PASN', genau wie 'ASN', jede bereits bestehende Zuweisung, die im Wege sein könnte, sowieso überschreibt, ist 'CLKEYS' natürlich nicht unbedingt notwendig, zumal dadurch auch solche Zuweisungen getilgt werden könnten, die weiterhin benötigt werden und somit in einem Programm "TF2" neu getätigt werden müßten. Wenn Sie also die gezielte Löschung einzelner Tastenzuweisungen vorziehen, müssen Sie statt 'CLKEYS' Befehlsgruppen der nachstehenden Form verwenden:

```
CLA
(Tastenkode 1)
PASN
(Tastenkode 2)
PASN
.
.
.
```

Die dritte X-Funktion, die sich auf das Tastenfeld, aufgefaßt als Ansammlung von Tastenkodes, bezieht, ist 'GETKEY'. Bei ihr handelt es sich um eine sehr ungewöhnliche Funktion, die sich auf den ersten Blick als abseits normaler Bedürfnisse liegend ausnimmt, vielleicht aber auch von Ihnen für leistungsfähiger als andere X-Funktionen gehalten wird, wenn Sie erst einmal die Kapitel 8 und 9 gelesen haben. Warten Sie mit Ihrem Urteil bis dahin ab.

Die Funktion 'GETKEY' (*get key*code) führt einen völlig neuen Typ von Funktion in das 'Wörterbuch' des HP-41 ein. Wenn 'GETKEY' als Befehl eines Programms ausgeführt wird, unterbricht der Rechner für etwa 10 Sekunden die Programmausführung und wartet in dieser Zeit darauf, daß Sie eine Taste drücken. Sobald Sie das tun, gelangt der Kode (Zeilen-/Spaltennummer) der gedrückten Taste ins Register X. Versäumen Sie es, innerhalb der Wartezeit eine Taste zu drücken, wird stattdessen 0 ins Register X gelegt. In beiden Fällen wird der Stapel angehoben, während L ungestört bleibt. Möchten Sie sich durch die vorgegebenen 10 Sekunden nicht in Zeitnot bringen lassen, ist das Problem mit einer kleinen Schleife leicht aus der Welt zu schaffen:

```
LBL 00
GETKEY
X = 0?
GTO 00
```

Solange keine Taste gedrückt wird, 'tritt das Programm jetzt brav auf der Stelle'. Damit der Stapel bei diesem Verfahren nicht nach mehrfachem Schleifendurchlauf 'leergeschrieben' zurückbleibt, können Sie zusätzlich ein 'RDN' zwischen 'LBL 00' und 'GETKEY' einsetzen.

Anders als 'PASN' erkennt 'GETKEY' auch die oberhalb des eigentlichen Tastenfeldes liegenden vier 'Schaukeltasten' zur Modus-Wahl als 'vollwertig' an: Während der durch 'GETKEY' ausgelösten Wartezeit führt ein Druck auf die Tasten 'ON' bis 'ALPHA' zur Ablieferung von Tastenkodes 1 bis 4. Der diese Tasten bergenden Zeile wird also gleichsam die Zeilennummer 0 zuerkannt. Jedoch nur von 'GETKEY', nicht von 'PASN'.

Wenn Sie 'GETKEY' verwenden, sollten Sie tunlichst Befehlsfolgen wie

```
LBL 00
GETKEY
GTO 00
```

vermeiden, weil das Fehlen der Abfrage 'X = 0?' Ihr Programm in eine unendliche Schleife führt, die auch nicht mehr mit der sonst zuverlässig arbeitenden Bremse 'R/S' unterbrochen werden kann, denn: 'R/S' bedeutet für 'GETKEY' nur die Anweisung 'Tastenkode 84 nach X legen'. Selbst die Notbremse 'ON' versagt aus ebendemselben Grunde. Was bleibt zu tun? Sie könnten die Batterien entfernen. Doch das wird Ihnen nicht zusagen. Daher hier ein Tip: Halten Sie 'R/S' nieder, drücken Sie dann 'ON', lassen Sie nun erst 'R/S' und danach 'ON' wieder los. Das hilft. Doch besser ist es natürlich, Schleifen gar nicht erst so anzulegen, daß sie auf programmiertechnisch einwandfreiem Wege nicht mehr verlassen werden können.

Mit 'GETKEY' können Sie Programme schreiben, welche die Eigenschaft der beiden oberen Tastenreihen, unmittelbaren Zugang zu lokalen Marken zu verschaffen, nachahmen, und dies sogar ausgedehnt auf das gesamte Tastenfeld, auf alle 35 nicht-umgeschalteten Tasten; wenn Sie wollen, auch die 4 Schaukeltasten eingeschlossen. Entscheidender noch ist, daß ein solches Programm keinen Anlaß mehr dazu gibt, sich über vorhandene Zuweisungen von Funktionen zu grämen, ja daß überdies der normalerweise notwendige Wechsel in den USER-Modus entfällt. Und die Bedeutung jeder Taste darf sogar *innerhalb eines Programms* wechseln.

Nachstehend eine Programmskizze, welche die typische Verwendung von 'GETKEY' vorführt:

```
LBL "MUSTER"
LBL 00
RDN
GETKEY
X = 0?          } zurück an den Schleifenanfang, wenn
GTO 00          } keine Taste gedrückt wurde
XEQ IND X         Ausführung eines Unterprogramms
                  entsprechend der gedrückten Taste

.               } Anzeige von Ergebnissen oder Aufruf von Befehlen,
.               } die unabhängig von der gedrückten Taste
.               } in jedem Fall ausgeführt werden sollen

RTN oder GTO 00
LBL 11          }
.               } Unterprogramm, welches bei
.               } Druck auf 'Σ + ' ausgeführt wird
.               }
RTN
LBL 12          }
.               } Unterprogramm, welches bei
.               } Druck auf '1/x' ausgeführt wird
.               }
RTN
.               } weitere Unterprogramme, die als
.               } Antwort auf die Tastenbedienung
.               } ausgeführt werden sollen
END             }
```

Zu Beginn sorgt eine Schleife dafür, daß das Programm geduldig auf das Drücken einer Taste wartet. Sobald dies geschieht, führt es die Schritte aus, welche vom Kode der gedrückten Taste angeführt werden. Drücken Sie z.B. die Korrekturtaste, hält der HP-41 Ausschau nach der lokalen numerischen Marke 'LBL 44' und arbeitet dann die dahinter stehenden Befehle als Unterprogramm ab. Für jeden der insgesamt 39 Tastenkodes können Sie ein Unterprogramm, das Befehle Ihrer Wahl enthält, einrichten.

Gewöhnliche Tastenzuweisungen, die im USER-Modus wirksam sind, können mit 'GETKEY' nicht zusammenstoßen, weil diese X-Funktion gegenüber dem USER-Modus – genauso wie gegenüber der Betätigung der Schaukeltasten – 'vorfahrtberechtigt' ist. 'GETKEY' kann gewissermaßen als *Benutzer-Modus höherer Stufe* betrachtet werden.

Hier noch ein Beispiel für eine einfache Verwendung von 'GETKEY'. Die nachstehende Befehlsfolge fordert zu einer 'Ja/Nein'-Antwort auf und kann als Teil eines Dialogs verwendet werden. 'R/S' oder 'J' (die 'TAN'-Taste) werden als 'Ja' aufgefaßt, jede andere Taste wirkt als 'Nein'.

```
"HINRICHTEN?"          Frage
SF 25
*LBL 00
AVIEW
GETKEY
RDN                    Tastenkode ins Register T schieben
GTO IND T
  :                    } Programmteil für 'Nein'
  :
RTN
*LBL 25                Ansprungstelle bei 'Ja'
*LBL 84                Ansprungstelle bei 'R/S'
CF 25
  :                    } Programmteil für 'Ja'
  :
RTN
```

Die Anweisung 'GTO IND T' verursacht alle 10 Sekunden einen Sprung zur Marke 'LBL 00', wenn keine Taste gedrückt wird, oder auf 'LBL 25' oder 'LBL 84', je nachdem die Frage (Sie haben hoffentlich freundlicher klingende in Ihren Anwendungen) mit 'J' oder 'R/S' bejaht wird. Für andere Tastenkodes ist keine Marke vorhanden. Man erhielte für alle anderen, 'Nein' bedeutenden Tasten also eine das Programm unterbrechende "NONEXISTENT"-Meldung, wenn nicht Flag 25 gesetzt wäre. Das Fehlerignorierflag aber bewirkt, daß das Programm wunschgemäß in den hinter 'GTO IND T' liegenden 'Nein'-Bereich mündet. Flag 25 wird dabei gelöscht.

Die voranstehend vorgeführte 'GETKEY'-Technik setzt – jedenfalls fast immer – stillschweigend voraus, daß sich nirgendwo im Programm solche Marken herumtreiben, die einen unbeabsichtigt durch 'GETKEY' ausgelösten Ansprung ermöglichen. Dies bedeutet, daß innerhalb des Programms keine lokalen Marken aus der folgenden Tabelle zugelassen sind, außer als Ansprungstelle für einen 'GTO IND'-Befehl, der Tastenkodes vom 'GETKEY'-Befehl nach Plan und Absicht entgegennimmt.

01	02		03	04	(Schaukeltasten-Zeile)
11	12	13	14	15	(Zeile 1)
21	22	23	24	25	(Zeile 2)
31	32	33	34	35	(Zeile 3)
41		42	43	44	(Zeile 4)
51	52		53	54	(Zeile 5)
61	62		63	64	(Zeile 6)
71	72		73	74	(Zeile 7)
81	82		83	84	(Zeile 8)

Natürlich können Sie auf solche Vorsichtsmaßnahmen verzichten, wenn Sie entweder die Tastatur des HP-41 so sicher beherrschen, daß Fehlgriffe auf falsche Tasten ausgeschlossen sind, oder wenn Sie die Regeln berücksichtigen, nach denen die mehrfache Verwendung lokaler numerischen Marken zulässig ist, oder wenn Sie gar fehlerhafte Ergebnisse, die durch das Drücken falscher Tasten bei 'GETKEY'-Antworten entstehen, großzügig in Kauf nehmen.

Das Adressen-Programm in Kapitel 7 beinhaltet eine Abfrage, deren 'Ja/Nein'-Beantwortung mit dem gerade besprochenen Verfahren abgewickelt wird. Zwar werden dort noch zwei weitere Antworten zugelassen, doch grundsätzlich Neues geschieht bei deren zusätzlicher Bearbeitung nicht. Ein sehr viel ausgefeilteres Beispiel für den Einsatz von 'GETKEY' lernen Sie hingegen in Kapitel 9 kennen, wo ein Programm vorgeführt wird, das die Funktionen des HP-16, der vorzugsweise zur wechselseitigen Umrechnung von Zahlendarstellungen dient, nachahmt: ein *einzelner* Tastendruck bewirkt hier wie dort den Wechsel von einer Basis in die andere.

4G. Zusätzliche X-Funktionen auf dem HP-41CX

Im HP-41CX sind weitere 14 X-Funktionen, die der für den Ausbau des HP-41C oder CV dienende X-Funktionen-Modul nicht enthält, ansässig. Sechs davon wurden schon beschrieben: 'EMROOM' und 'EMDIRX' in Abschnitt 1B, 'RESZFL' in den Abschnitten 2E und 3G, 'ASROOM' und 'ED' in Abschnitt 3H sowie 'CLRGX' im Abschnitt 4E. Die verbleibenden acht X-Funktionen, 'GETKEYX', 'ΣREG?', 'X<NN?', 'X<=NN?', 'X=NN?', 'X≠NN?', 'X>=NN?' und 'X>NN?', sind Gegenstand dieses Abschnitts, der die Besprechung der X-Funktionen abschließt. Die restlichen Kapitel dieses Buches sind dann den Anwendungen gewidmet.

Die Funktion 'GETKEYX' (*get key*code within x seconds designated by *X*) ist eine erweiterte Spielart von 'GETKEY'. Durch die Zahl x in X ($-99.9 \leq x \leq 99.9$) wird festgelegt, wieviele Sekunden der Rechner auf die Antwort, das Drücken einer Taste, warten soll. Ist x kleiner als 0.1, gibt sich der Rechner redlich Mühe, die von Ihnen bestimmte Wartezeit einzuhalten, doch wird diese Zeit bei so kurzen Angaben i.a. leicht überschritten. 'GETKEYX' liefert den Tastenkode im Register Y ab (nicht in X wie 'GETKEY') und *zusätzlich einen Zeichenkode* in X nach den weiter unten beschriebenen Regeln. Die ursprünglich in X enthaltene Wartezeit wird ins Register L übertragen, während die Inhalte von Y und Z nach Z bzw. T gelangen.

Drückt man während der Wartezeit eine nicht-umgeschaltete Taste, gelangt — wie gesagt — der Tastenkode ins Register Y. Wird jedoch die 'goldene' Taste ('SHIFT') betätigt, erlangt man *nicht* wie bei 'GETKEY' den Kode 31. Stattdessen beginnt der Rechner gemäß der anfangs festgelegten Zeit erneut mit dem Warten. Drückt man anschließend eine andere Taste, wird deren Kode mit negativem Vorzeichen nach Y gelegt. Wegen dieses Verhaltens hat man die Möglichkeit, von 'GETKEYX' die Kodes von nicht-umgeschalteten *und umgeschalteten* Tasten zu empfangen. Außerdem kann man, insbesondere bei Unentschlossenheit bezüglich der Tastenwahl, die Wartezeit beliebig verlängern, indem man immer wieder rechtzeitig die goldene Taste drückt und so die Funktion zwingt, ständig mit dem Warten von neuem zu beginnen. Läuft die Wartezeit ab, bevor eine Taste gedrückt wurde, wird 0 nach Y gebracht, um diesen Umstand anzuzeigen.

Der von 'GETKEYX' nach X gelegte Wert hängt vom Zustand des Flags 48 ab. Ist der ALPHA-Modus während der durch 'GETKEYX' ausgelösten Wartezeit aktiv, also Flag 48 gesetzt (dies ist, wie man leicht einsieht, nur dann möglich, wenn 'GETKEYX' als Programmbefehl ausgeführt wird), kommt der ASCII-Kode, welcher der umgeschalteten bzw. nicht-umgeschalteten Taste entspricht, nach X, ganz so, wie Sie es von den Regeln für die Beschickung des Alpha-Registers mit Zeichen her kennen. Insoweit wird also nur dann ein Kode nach X gebracht, als die gedrückte Taste auch wirklich mit einem Alpha-Zeichen belegt

ist. Sollte das nicht der Fall sein wie z.B. bei 'R/S' oder 'SHIFT, STO', gelangt 0 nach X. Aber die Funktion 'GETKEYX' ist hierbei sogar päpstlicher als der Papst: feinfühlig, ja übergenau reicht Sie Ihnen bei 'SHIFT, XEQ' den Kode 127 für das Anhangssymbol "⊢" nach X. Wegen der vorstehend beschriebenen Gesetze läßt sich das der gedrückten Taste entsprechende Zeichen ganz einfach ins Alpha-Register bringen: ein einfaches anschließendes 'XTOA' reicht dazu aus.

Ist der Alpha-Modus während der Wartezeit ausgeschaltet (Flag 48 gelöscht), werden die ASCII-Kodes der Ziffern (Kodes 48 bis 57), einschließlich Dezimaltrennung (Kodes 44 oder 46, abhängig vom Zustand von Flag 28) und Vorzeichen 'CHS' (Kode 45), nach X gebracht. Alle anderen Tasten liefern 0 in X ab. Auch hier kann man wieder mit 'XTOA' arbeiten, um das Alpha-Register den gedrückten Tasten entsprechend mit Zeichen zu beschicken.

Legt man eine *negative* Zahl als Wartezeit für 'GETKEYX' nach X, wartet der Rechner mit der weiteren Ausführung des Programms nicht – wie er dies bei positiven Wartezeiten tut – darauf, daß Sie die gedrückte Taste wieder loslassen. Vielmehr nimmt er in diesem Fall die Programmausführung bereits in dem Moment wieder auf, in dem Sie die ausgewählte Taste drücken. Daher wird eine Schleife wie z.B.

```
LBL 01
.
.
.
– 0.1
GETKEYX
RDN
X ≠ 0?
GTO 01
```

solange durchlaufen, als Sie irgendeine Taste niedergedrückt halten. Zwar ist es recht unwahrscheinlich, daß Sie diese sehr abseits liegende Eigenschaft von 'GETKEYX' für Ihre eigenen Anwendungen werden nutzen können. Sollte dies aber ausnahmsweise doch einmal der Fall sein, dann müssen Sie sich über ein kleines Problem, das in diesem Zusammenhang auftritt, klar sein. Dies nämlich ist zu beachten: 'GETKEYX' liest Tasten- und ASCII-Kode und bringt sie nach Y bzw. X. Doch sofern Sie den Tastendruck bei negativer Wartezeit nicht blitzartig aufheben, sondern auch nur einen kleinen Augenblick auf der Taste verweilen, wird die auf der Taste liegende Normalfunktion (im USER-Modus die zugewiesene Funktion) ausgeführt (übrigens auch nach einem 'SHIFT': Umschaltungen bleiben bis auf den Vorzeichenwechsel des Tastenkodes in Y unwirksam!). All dies verwirrende Verhalten wiederum stört in der obigen Schleife keineswegs, weil dabei ja ein Programm abläuft. Ein laufendes Programm aber schert sich bekanntlich den Teufel um die zwischenzeitlichen Fingerübungen des Benutzers, es sei denn, er berührt dabei zwei bestimmte Tasten: 'ON' schaltet den Rechner auch bei laufendem Programm aus, und die 'Betriebsbremse' 'R/S' unterbricht das Programm pflichtschuldigst. Das Tastenfeld verhält sich also ganz normal, sobald 'GETKEYX' negative Wartezeiten vorfindet. Mithin dürfen die beiden Tasten 'ON' und 'R/S' in Programmentwürfen, die 'GETKEYX' negative Wartezeiten anbieten, nicht in die Gruppe der 'Antworttasten' aufgenommen werden, außer natürlich mit ihrer ursprünglichen Bedeutung: Rechner ausschalten bzw. Programm unterbrechen.

Alles miteinander berücksichtigt, werden Sie zugeben: 'GETKEYX' ist eine außergewöhnliche, eine anspruchsvolle Funktion.

Hier zwei Musteranwendungen von 'GETKEYX'. Die erste, "VREG" (*view reg*ister), weist den Inhalt eines ausgewählten Registers solange vor, als eine dem Register entsprechende Taste niedergedrückt gehalten wird.

Programmausdruck von "VREG"
(Barcode im Anhang D)

```
01♦LBL "VREG"      09 CHS          19 -            28 GETKEYX
02 CF 21           10 GETKEYX      20 X<0?         29 RDN
                   11 SIGN         21 CLX          30 X≠0?
03♦LBL 01          12 X<>Y         22 VIEW IND X   31 GTO 03
04 "REG?"          13 X=0?         23 .1           32 GTO 01
05 AON             14 GTO 02       24 CHS          33 END
06 AVIEW           15 X=Y?         25 SIGN
                   16 RTN
07♦LBL 02          17 LASTX        26♦LBL 03
08 10              18 64           27 X<> L                  57 Bytes
```

Sobald Sie 'XEQ "VREG"' ausführen, fordert Sie der Rechner mit "REG?" zur Auswahl einer Registeradresse auf. Mit den Tasten 'A' bis 'Z' erreichen Sie nun die Register R_{01} bis R_{26}. Die 'Buchstaben-freien' Tasten fördern den Inhalt von R_{00} zutage. Wollen Sie den Programmlauf abbrechen, müssen Sie 'R/S' oder 'ON' drücken.

Das Programm arbeitet folgendermaßen: Hinter der von 'LBL 02' angeführten Warteschleife wird der von 'GETKEYX' ermittelte ASCII-Kode durch 'SIGN' (Zeile 11) ins Register L gebracht, mit 'LASTX' (Zeile 17) von dort zurückgeholt und durch Subtrahieren von 64 in die benötigte Registeradresse umgerechnet: 'A','B',...,'Z' bringen 1,2,...,26 nach X. Der Inhalt des jeweiligen Registers wird dann in Zeile 22 angezeigt. Das 'GETKEYX' in der zweiten, von 'LBL 03' angeführten Schleife sorgt dafür, daß das Vorweisen des ausgewählten Registerinhaltes solange anhält, als die die Auswahl treffende Taste niedergedrückt bleibt. Hebt man den Tastendruck auf, gelangt 0 nach X, so daß eine Verzweigung auf 'LBL 01', wo das Programm wieder mit der Frage "REG?" beginnt, stattfindet. (Anm. d. Übers.: Die Zeilen 15 und 16 sind überflüssig; sie können gelöscht werden.)

Eine zweite, mehr sinnfällige Anwendung von 'GETKEYX' besteht darin, Tasten auszuwählen, damit sie mit Funktionen, die für die Arbeit mit irgendeinem Programm benötigt werden, belegt werden können. Denken Sie sich beispielsweise, ein Programm, welches den Wechsel von Zahlendarstellungen vornehmen soll, benötige die Zuweisungen der Standardfunktionen 'MOD', 'INT' und 'FRC' zum flinken Zugriff im USER-Modus. Im Abschnitt 4F haben wir vorgeführt, wie die X-Funktion 'PASN' dazu verwendet werden kann, Tastenzuweisungen programmgesteuert vorzunehmen. Die X-Funktion 'GETKEYX' kann für zusätzliche Unabhängigkeit bei diesen Zuweisungen sorgen, weil sie dem Benutzer gestattet, über die zu belegenden Tasten beim Programmstart sozusagen 'nach Tageslaune' zu entscheiden, wobei er sich nicht einmal mehr der geringen Mühe zu unterziehen braucht, die für 'PASN' benötigten Tastenkodes selbst zu ermitteln. Ein Programm solcher Art kann etwa die folgende Gestalt haben:

```
"MOD"
XEQ 99  →        } das Unterprogramm '99' fordert den
"INT"            } Benutzer auf, eine Taste zu drücken,
XEQ 99  →        } welche dann von 'PASN' belegt wird,
"FRC"            } sobald 'GETKEYX' ihren Kode
XEQ 99  →        } ermittelt hat
.
.
.
LBL 99
"⊦ TASTE"        beachten Sie, daß dieser Anhang aus
                 6 Zeichen besteht
AVIEW
10
LBL 00
GETKEYX          bringt den Tastenkode nach Y
X<> L
X>Y?             } zurück nach 'LBL 00', falls der'
GTO 00           } Tastenkode < 10 ist
RDN
– 6              } diese drei Schritte beseitigen die
AROT             } 6 Zeichen " TASTE" aus dem
ASHF             } Alpha-Register
RDN              der Tastenkode gelangt nach X
PASN             die Zuweisung wird vorgenommen
RTN
```

Die im HP-41CX ansässige Funktion 'ΣREG?' (address of 1st *Σ-reg*ister?) liefert die Adresse des ersten Registers des Statistik-Blockes, welcher bekanntlich aus 6 Registern besteht und vom Rechner für die Arbeit mit den Statistik-Befehlen 'Σ+', 'Σ–', 'MEAN' und 'SDEV' benutzt wird, in X ab. Mit der Standardfunktion 'ΣREG' können Sie dieses erste Register selbst bestimmen. Wenn Sie 'ΣREG?' ausführen, erhalten Sie genau die Zahl zurück, die Sie zuvor als Parameter-Eingabe für 'ΣREG' benutzt haben. Sollten Sie seit der letzten Totallöschung 'ΣREG' kein einziges Mal aufgerufen haben, ist das Ergebnis von 'ΣREG?' die Zahl 11. 'ΣREG?' hebt den Stapel an und läßt L unberührt. Die nachstehende Übersicht gibt die Σ-Register und ihre Inhalte wieder:

$R_{\Sigma REG?}$	Σx
$R_{\Sigma REG?+1}$	Σx^2
$R_{\Sigma REG?+2}$	Σy
$R_{\Sigma REG?+3}$	Σy^2
$R_{\Sigma REG?+4}$	Σxy
$R_{\Sigma REG?+5}$	n

Der Hauptzweck von 'ΣREG?' besteht darin, dem Benutzer zu ermöglichen, statistische Summen aus dem Statistik-Block zu holen, ohne daß er dazu im voraus wissen muß, wo dieser Block gerade liegt. Beispielsweise ahmt die Befehlsfolge

```
2
ΣREG?
+
RDN
RCL IND T
RCL IND L
```

die Funktion 'RCLΣ' des früheren HP-67/97 nach, indem sie Σy nach Y und Σx nach X bringt. Ganz offensichtlich können Sie diese Befehlsfolge ohne Umstände so abändern, daß Ihnen jede gewünschte Summe aus dem Statistik-Block für Ihre Berechnungen zur Verfügung gestellt wird.

Die verbleibenden 6 Funktionen des HP-41CX, 'X<NN?', 'X< = NN?', 'X = NN?', 'X ≠ NN?', 'X> = NN?' und 'X>NN?', erlauben Vergleiche des Inhaltes von X mit dem jedes anderen numerischen Datenregisters. Die Adresse des Vergleichsregisters muß dazu in Y liegen. Wendet man die bei indirekten Funktionen übliche Ausdrucksweise an, kann man sagen: X wird indirekt mit Y verglichen. Die Benutzung von z.B. 'X> = NN?' ist denkbar einfach: man legt die Adresse des Vergleichsregisters nach Y und führt 'XEQ "X> = NN?"' aus.

Statt der Adresse eines Datenregisters kann man auch den Namen eines Stapelregisters, also "X", "Y", "Z", "T" oder "L", z.B. mit 'ASTO', nach Y legen. Ist der Inhalt von X größer oder gleich dem Inhalt des durch Y bestimmten Daten- oder Stapelregisters erscheint die Antwort "YES"; andernfalls erblickt man "NO". Auch mit "X" oder "Y" in Y erhält man 'richtige' Antworten, doch kann man aus naheliegenden Gründen kaum etwas damit anfangen.

Wenn Sie die Vergleiche als Programmbefehle durchführen lassen, erscheinen die Antworten "YES" oder "NO" natürlich nicht in der Anzeige. Vielmehr wird, wie Sie das schon von den gewöhnlichen Abfragen ('X = 0?', 'FS? 10' usw.) her kennen, der Befehl, welcher unmittelbar auf den Vergleich folgt, ausgeführt, sobald die Antwort 'ja' lautet; andernfalls wird er übersprungen. Die neuen Vergleichsfunktionen sind also der Programmlogik früherer Modelle des HP-41 vollständig angeglichen.

In einem wichtigen Punkte jedoch unterscheiden sich die 6 Vergleichsfunktionen ganz wesentlich von ihren Vettern im Katalog 3. Sie unterliegen nämlich bezüglich der Vergleichsfähigkeit der Registerinhalte keinerlei Einschränkungen: Zeichenketten werden ebenso vollständig verglichen wie Zahlen, und zwar auf der Grundlage der ASCII-Kodierung, so als ob Sie die Kette von links nach rechts fortschreitend unter Verwendung von 'ATOX' verglichen und die Entscheidung über den Vergleich beim ersten Zeichen, welches einen Unterschied der beiden Ketten enthüllte, träfen. Die ASCII-Ordnung von Zeichenketten stimmt fast genau mit der sogenannten lexikographischen Ordnung überein, ausgenommen, daß

a) Ziffern und Satzzeichen den Buchstaben in der ASCII-Ordnung vorangehen und
b) Kleinbuchstaben den Großbuchstaben insgesamt nachfolgen.

Die vollständige ASCII-Ordnung können Sie unmittelbar und mühelos der in Abschnitt 4B abgedruckten ASCII-Tabelle entnehmen.

Hinweis für Synthetiker: Es lassen sich sogar nicht-normalisierte Zahlen miteinander vergleichen, oder noch allgemeiner ausgedrückt: Es lassen sich je zwei beliebige Bitmuster der Länge 56 miteinander vergleichen.

Das nachstehend abgedruckte Programm "ALSORT" (*al*phabetische *Sort*ierung) sortiert die Inhalte eines Datenregister-Blockes, beginnend mt Register R_{abc} und endend mit Register R_{klm}, in aufsteigender Reihenfolge, Anfangs- und Endregister eingeschlossen. Es handelt sich

dabei um ein einfaches Sortier-Verfahren, das unter der Bezeichnung 'bubble-sort Algorithmus' bekannt ist. Man legt eine Kontrollzahl der Form 'abc.klm' nach X und führt 'XEQ "ALSORT"' aus.

Programmausdruck von "ALSORT"
(Barcode im Anhang D)

```
01♦LBL "ALSORT"     11 RCL X          21 GTO 03         32 GTO 02
02 ENTER↑           12 RCL Z          22 X<> IND Y
03 ISG Y            13 INT            23 STO IND L      33♦LBL 03
04 X<0?             14 +              24 FS?C 06        34 R↑
05 RTN              15 DSE X          25 GTO 03         35 R↑
06 INT              16 X<0?           26 RDN            36 ISG Y
07 1 E3             17 SF 06          27 ABS            37 GTO 01
08 /                18 RCL IND L      28 RCL IND X      38 END
                                      29 DSE Y
09♦LBL 01           19♦LBL 02         30 FS? 53
10 CF 06            20 X>=NN?         31 SF 06                    71 Bytes
```

Weil der Vergleich in "ALSORT" mit der Funktion 'X> = NN?' (Zeile 20) durchgeführt wird, sortiert das Programm sowohl Zahlen als auch Zeichenketten, die überdies noch miteinander vermischt sein dürfen.

Der bubble-sort Algorithmus ist ein elementares Verfahren, das in BASIC etwa so geschrieben würde:

```
for i = abc + 1 to klm
      for j = i – 1 to abc by – 1
            if R(j+1) > R(j) go to new i
            else interchange (R(j+1), R(j))
            next j
new i: next i
```

In "ALSORT" stellt die von 'LBL 01' angeführte Schleife die i-Schleife dar, in der mit 'ISG' der Zähler 'i.klm' bearbeitet wird. Die von 'LBL 02' angeführte Schleife ist die j-Schleife mit dem 'DSE'-Zähler 'j.abc – 1'. Um es Ihnen zu erleichtern, die Benutzung des Stapels während des Programmlaufs zu verfolgen, seien nachstehend die entscheidenden Inhalte zum Zeitpunkt des Einlaufs in die Schleifen ausgeführt:

	LBL 01	LBL 02
T		i.klm
Z		0.abc – 1
Y	i.klm	j.abc – 1
X	0.abc – 1	R_{j+1}
L		j + 1

KAPITEL 5

Ein Programm zum Zählen von Bytes

Aus dem Benutzerhandbuch des HP-41 wissen Sie, daß das *Byte* die kleinste Einheit des Programmspeichers ist und daß jeder Befehl innerhalb eines Programms ein oder mehrere Bytes dieses Speichers belegt. Darüber hinaus finden Sie im Handbuch eine Tabelle, der Sie die für jeden Befehlstyp benötigte Byte-Anzahl entnehmen können.

Wenn Sie im Besitz eines HP-41CX sind und wissen wollen, wieviele Bytes ein bestimmtes Ihrer Programme besetzt hält, brauchen Sie einfach nur 'CAT 1' auszuführen und die Katalogdurchsicht mit 'R/S' anzuhalten, sobald das 'END' des ins Auge gefaßten Programms in der Anzeige erscheint. Rechterhand in der Anzeige erblicken Sie dann die gesuchte Anzahl. Ist das ständige '.END.' die letzte Programmzeile des Programms, dessen Byte-Anzahl gesucht wird, ist das Programm, dessen Länge Sie kennen wollen, also das letzte im Speicher, müssen Sie zuvor 'GTO..' ausführen, um auch dieses letzte Programm mit einem gewöhnlichen 'END' zu versehen, denn das '.END.' wird unter 'CAT 1' nicht mit der Byte-Anzahl des Programms, zu dem es gehört, versehen.

Sofern Sie die Byte-Anzahl von Programmen im HP-41C oder CV zu ermitteln wünschen, müssen Sie sich die genannte Tabelle aus dem Handbuch zurechtlegen und die Bytes Befehl für Befehl auszählen. Wenn Sie das Ergebnis noch durch 7 teilen und auf die nächste ganze Zahl aufrunden, erhalten Sie überdies die vom Programm belegte Register-Anzahl. Das solchermaßen durchgeführte Auszählen von Bytes ist natürlich zeitraubend. Was liegt also näher als der Gedanke, den HP-41 selbst mit dieser Arbeit zu beauftragen, zumal er den Benutzer fast nie enttäuscht und sich immer wieder als kraftvolles Werkzeug bei allen Arten von Rechenproblemen erweist.

Mit dem X-Funktionen-Modul können Sie in der Tat mühelos die Bytes eines Programms programmgesteuert auszählen. Die hier vorgestellte Routine "CBX" (*c*ount *B*ytes using *X*-memory) leistet das Erwünschte. Zunächst bringt "CBX" Ihr Programm unter dem Dateinamen "**CBX" ins X-Memory, es sei denn, es lag dort schon unter seinem eigenen Namen. Dann nutzt "CBX" die Tatsache aus, daß 'RCLPT' (wie auch 'RCLPTA'), angesetzt auf Programmdateien, deren Byte-Anzahl ins X-Register bringt (vgl. dazu auch die Ausführungen in Abschnitt 2E). Schließlich wird die vorübergehend erzeugte Programmdatei "**CBX" wieder gelöscht. Lag Ihr Programm allerdings schon unter seinem Namen im X-Memory, findet natürlich keine Löschung statt.

Sobald "CBX" die Programm-Bytes ausgezählt hat, berechnet es noch die Anzahl der belegten Register. Diese Zahl stimmt i.a. mit der vom Befehl 'FLSIZE' ermittelten Größe überein, ausgenommen in den Fällen, in welchen das sog. Prüfsummenbyte (vgl. Abschnitt 10C) erst Platz in einem neuen Register findet, weil die Summe der Programm-Bytes ohne Rest

durch 7 teilbar ist, das Programm also die belegten Register 'randvoll' ausfüllt. In einigen Fällen ist daher die Anzahl der benötigten Programm-Register um Eins kleiner als die von 'FLSIZE' ausgewiesene Zahl.

Programmausdruck von "CBX"
(Barcode im Anhang D)

```
01↑LBL "CBX"     07 SAVEP      13 CLD        18 +
02 SF 25         08 RCLPT      14 RDN        19 7
03 RCLPTA        09 "**CBX"                  20 /
04 FS?C 25       10 PURFL      15↑LBL 01     21 INT
05 GTO 01        11 ENTER↑     16 RCL X      22 X<>Y
06 "┣,**CBX"     12 EMDIR      17 6          23 END
```

52 Bytes

Gebrauchsanweisung für "CBX"

1. Stellen Sie sicher, daß das auszuzählende Programm ge'PACK't ist und seine letzte Zeile aus einem echten 'END' (nicht dem ständigen '.END.') besteht. Es handelt sich um dieselben Vorbereitungen, die Sie treffen müssen, wenn Sie ein Programm ins X-Memory bringen und dafür den Speicherplatz auf das Mindestmaß beschränken wollen (vgl. Abschnitt 1A).
2. Setzen Sie den Namen des auszuzählenden Programms ins Alpha-Register. Dieser Name darf nicht mit einer im X-Memory vorhandenen Daten- oder Textdatei übereinstimmen.
3. Führen Sie 'XEQ "CBX"' aus.
4. Wenn das Programm schon vorher im X-Memory lag, erscheint das Ergebnis sehr schnell. Die Byte-Anzahl gelangt nach X und die Register-Anzahl nach Y. Mit 'X<>Y' oder 'RDN' holen Sie sich also die letztere Größe in die Anzeige.
5. War das Programm noch nicht im X-Memory, müssen Sie ein paar Sekunden länger auf das Ergebnis warten. Zunächst erscheint dann nämlich das Inhaltsverzeichnis des X-Memorys. Erst eine halbe Sekunde, nachdem dies erledigt ist, gelangen Byte- und Register-Anzahl nach X bzw. Y. Sie können den Ablauf natürlich beschleunigen, indem Sie das Verzeichnis mit 'R/S' anhalten und anschließend die Korrekturtaste betätigen, wodurch das Programm dazu veranlaßt wird, sofort mit der Ausführung der hinter 'EMDIR' stehenden Befehle fortzufahren.

"CBX"-Beispiel 1

Das Programm soll 'sich selbst auszählen': Sie bringen "CBX" ins Alpha-Register und führen 'XEQ "CBX"' aus. Als Ergebnis muß 52 in X und 8 in Y stehen. Falls Ihr "CBX" unge'PACK't ist oder '.END.' als letzte Zeile enthält, kann Ihr Ergebnis leicht davon abweichen.

Zeilen-Analyse von "CBX"

Wie schon erwähnt, muß das Alpha-Register beim Programm-Start den Namen des Programms, für welches die Byte-Anzahl ermittelt werden soll, enthalten. Für Programme, die

schon im X-Memory liegen, erlangt man bereits auf Zeile 03 die gesuchte Byte-Anzahl, weil 'RCLPTA' auch 'dateifremd' arbeitet (vgl. Abschnitt 2E). Bei Programmen, die nicht im X-Memory vorhanden sind, greift 'RCLPTA' ins Leere und verursacht so die Löschung von Flag 25. Diese Löschung wird in jedem Fall spätestens auf Zeile 04 vorgenommen, von wo aus das Programm nach 'LBL 01' verzweigt, falls 'RCLPTA' erfolgreich war. Andernfalls wird im X-Memory vorübergehend eine Programmdatei mit dem Namen "**CBX" angelegt (vgl. Abschnitt 1D). Sie enthält das auszuzählende Programm. Ihre Byte-Anzahl wird in Zeile 08 ermittelt. Gleich darauf verschwindet sie wieder aus dem X-Memory (Zeile 10).

Die Funktion 'EMDIR' ist nur deswegen Bestandteil des Programms, weil bei 1B-Versionen des X-Funktionen-Moduls im Anschluß an 'PURFL' der Zugriff auf den Inhalt des X-Memorys verloren gehen kann (vgl. die Warnung in Abschnitt 1F). Bei Versionen von 1C an aufwärts können Sie die Zeilen 11 bis 14 bedenkenlos löschen.

Die Befehle 'ENTER↑' und 'RDN' (Zeilen 11 und 14) stellen sicher, daß das X-Register die festgestellte Byte-Anzahl enthält, unabhängig davon, ob der Ablauf des Katalogs 4 unterbrochen wird oder nicht. Ein vollständig durchgeführter Befehl 'EMDIR' hebt nämlich den Stapel und füllt das X-Register mit der Anzahl freien Register im X-Memory. Bei einer Unterbrechung von 'EMDIR' bleibt der Stapel hingegen unangetastet. 'RDN' schiebt also das Ergebnis von 'RCLPT' in jedem Fall nach X, weil dieses wegen 'ENTER↑' in X und Y liegt, bevor 'EMDIR' ausgeführt wird. Der Befehl 'CLD' dient dazu, die Anzeige vom letzten Eintrag des Katalogs 4 zu befreien, sobald "CBX" in die hinter 'LBL 01' durchgeführte Berechnung einläuft.

Bei 'LBL 01' beginnt die Berechnung der Register-Anzahl n_R nach der Formel

$$n_R = \text{INT}[(n_B + 6)/7] \quad (n_B = \text{Byte-Anzahl}),$$

welche die Division von n_B durch 7 und die anschließende Aufrundung auf die nächste ganze Zahl darstellt. Mit n_B in X und n_R in Y hält "CBX" an.

"CBX"-Beispiel 2

Zählen Sie die Bytes im Programm "JNX" aus Abschnitt 1A. Voraussetzung ist, daß "JNX" im Hauptspeicher oder im X-Memory liegt. Wenn Sie das Alpha-Register mit "JNX" füllen und "CBX" aufrufen, muß 80 nach X und 12 nach Y gelangen.

KAPITEL 6

Die Verwendung von Datendateien

6A. Ein allgemeines Wurzel-Suchprogramm

Programmierbare Taschenrechner werden besonders häufig dort eingesetzt, wo es gilt, Gleichungen der Form $f(x) = 0$ zu lösen, d.h. solche Werte x zu finden, für die diese Gleichung gültig ist. Beispielsweise mögen sich die Herstellungskosten für n Stücke eines gewissen Erzeugnisses auf $\sqrt{n}$ belaufen, solange man mit einer bestimmten Maschine 1 arbeitet, hingegen auf $10 + \ln(n + 1)$, sobald man auf einer Maschine 2 produziert. Weil die Funktion $\ln x$ flacher ansteigt als die Funktion $\sqrt{x}$, wird der Einsatz der Maschine 2 bei größeren Stückzahlen irgendwann wirtschaftlicher werden. Doch von welcher *genauen* Stückzahl an gilt dies? Denn für kleinere Stückzahlen ist der Einsatz von Maschine 1 sichtlich günstiger. Offenbar muß der Schnittpunkt der beiden Kurven $\sqrt{x}$ und $10 + \ln(x + 1)$ bestimmt, mithin die Gleichung

$$f(x) = 10 + \ln(x + 1) - \sqrt{x} = 0$$

gelöst werden. Weiter unten in diesem Abschnitt wird dies geschehen.

Minimum- und Maximum-Probleme lassen sich häufig ebenfalls in Gleichungen der Form $g(x) = 0$ kleiden, wenn man für g erste Ableitungen f' von Funktionen f einsetzt. Bei einfachen Funktionen f greift man für die Ermittlung von f' auf die Ableitungsregeln der Analysis zurück. Doch gibt es in der Praxis viele Fälle, bei denen die analytische Herleitung von f sehr schnell unbequem wird. Das Programm "DERIV", welches im nächsten Abschnitt vorgestellt wird, kann hier Abhilfe schaffen, weil es Ableitungswerte $f'(x)$ rein numerisch aus Funktionswerten $f(x)$ zu berechnen vermag.

Jedes Programm W, das die Wurzeln einer Gleichung $f(x) = 0$ sucht, muß mehrfach ein Programm P, welches $f(x)$ berechnet, aufrufen. Es muß deswegen Datenregister zur Verfügung gestellt bekommen, die zu benutzen nur ihm allein gestattet werden darf. Denn sollte sich auch P dieser Register zur Berechnung von $f(x)$ bedienen, werden deren Inhalte mit Sicherheit für die weitere Verwendung im aufrufenden Suchprogramm W unbrauchbar. Falsche Ergebnisse oder Programmabbrüche sind die Folge. Es wird zwar einige wenigen einfachen Fälle von Funktionen f geben, in denen P keine oder andere Datenregister als W benötigt. Im allgemeinen aber werden sich Überschneidungen bei der Datenregisterbenutzung nicht ausschließen lassen, welche Register man W auch immer zuweist. Man kann dem ganzen Problem natürlich dadurch aus dem Wege gehen, daß man in jedem einzelnen Fall von Wurzelsuche prüft, welche Register

von W und P benutzt werden, und eines der beiden Programme entsprechend umschreibt. Doch das bringt Mühsal mit sich und kostet Zeit. Überdies können sich Programmfehler dabei einschleichen.

Der X-Funktionen-Modul löst das Datenregister-Überlappungsproblem ein für allemal. Er erlaubt es Ihnen, einen *unbeschränkt verwendbaren* 'Wurzel-Sucher' zu schreiben, der mit *jedem* Programm P, welches Funktionswerte f(x) ausrechnet, zusammenarbeiten kann. (Selbstverständlich muß das Programm P von einer globalen Marke aus höchstens 6 Zeichen angeführt werden, damit es durch 'XEQ IND'-Befehle erreichbar ist.)

Statt nämlich die Daten von W in numerierten Datenregistern zurückzulassen, wenn P angesprungen wird und darin Unheil anrichten kann, bringt man sie vor jedem Aufruf von P ins X-Memory. Dort können sie solange sicher ruhen, als sich P im Hauptspeicher zu schaffen macht. Hat P seine Arbeit beendet und pflichtgemäß einen Wert f(x) ausgerechnet, werden die Daten für W unversehrt aus dem X-Memory zurückgeholt. Ein Musterbeispiel für die durch das X-Memory erreichte Leistungssteigerung des HP-41.

Das nachstehend abgedruckte Programm enthält nicht nur den Wurzel-Sucher "SOLVE", sondern gleich noch zwei weitere Routinen, die erst in den nächsten zwei Abschnitten abgehandelt werden. Alle drei Routinen sind deswegen zu einem einzigen Programm zusammengefügt worden, weil sie sich erstens gemeinsam einiger Befehlsfolgen, in denen das X-Memory angesprochen wird, bedienen und weil sie zweitens häufig zusammen benutzt werden. Tasten Sie das Programm, sofern Sie es nicht mit einem optischen Lesestift einlesen, sorgfältig in den Rechner, damit Sie die nachfolgenden Beispiele fehlerfrei auf Ihrem Gerät mitrechnen können.

Programmausdruck von "SOLVE"/"DERIV"/"INTEG"
(Barcode im Anhang D)

```
01↑LBL "SOLVE"
02 ASTO 00
03 STO 01
04 1
05 %
06 +
07 STO 02
08 "**SOLVE"
09 4
10 XEQ 05
11 2 E-3
12 SAVERX
13 RCL 01
14 XEQ IND 00
15 "**SOLVE"
16 SAVEX
17 GETR

18↑LBL 01
19 CLX
20 SEEKPTA
21 3 E-3
22 SAVERX
23 RCL 02
24 FS? 10
25 VIEW X
26 XEQ IND 00
27 "**SOLVE"
28 GETR
29 ENTER↑
30 ENTER↑
31 X<> 03
32 -
33 X≠0?
34 /
35 RCL 02
36 ENTER↑
37 X<> 01
38 -
39 *
40 ST- 02
41 RCL 01
42 RND
43 RCL 02
44 RND
45 X≠Y?
46 GTO 01
47 LASTX
48 GTO 21

49↑LBL "DERIV"
50 ASTO 03
51 STO 04
52 RDN
53 STO 05
54 3 E-3
55 STO 06
56 "**DERIV"
57 7
58 XEQ 05

59↑LBL 02
60 CLX
61 SEEKPTA
62 6 E-3
63 SAVERX
64 RCL 06
65 INT
66 RCL 05
67 *
68 RCL 04
69 +
70 XEQ IND 03
71 "**DERIV"
72 GETR
73 STO IND 06
74 ISG 06
75 GTO 02
76 RCL 03
77 RCL 02
78 RCL 01
79 ENTER↑
80 +
81 -
82 9
83 *
84 -
85 +
86 RCL 00
87 11
88 *
89 -
```

```
 90 6
 91 /
 92 RCL 05
 93 /

 94♦LBL 21
 95 ENTER↑
 96 PURFL
 97 EMDIR
 98 CLD
 99 RDN
100 RTN

101♦LBL "INTEG"
102 ASTO 00
103 STO 01
104 X<>Y
105 -
106 4
107 /
108 STO 02
109 ST+ X
110 ST- 01
111 CLX
112 STO 03
113 STO 06
114 STO 07
115 SF 20
116 20
117 "**INTEG"
118 XEQ 05

119♦LBL 22
120 2
121 RCL 03
122 CHS
123 Y↑X
124 STO 05
125 ST+ 05
126 1
127 -

128♦LBL 03
129 STO 04
130 "**INTEG"
131 CLX
132 SEEKPTA
133 ,019
134 SAVERX
135 FS? 49
136 OFF
137 3
138 RCL 04
139 X↑2
140 -
141 RCL 04
142 *
143 RCL 02
144 *
145 RCL 01
146 +
147 XEQ IND 00
148 "**INTEG"
149 GETR
150 1
151 RCL 04
152 X↑2
153 -
154 *
155 ST+ 06
156 1
157 RCL 04
158 RCL 05
159 +
160 X<Y?
161 GTO 03
162 RCL 03
163 STO 04
164 RDN
165 7
166 STO 05
167 SIGN
168 ST+ 03
169 -
170 RCL 02
171 *
172 RCL 06
173 *
174 3
175 *

176♦LBL 04
177 ENTER↑
178 X<> IND 05
179 ST- Y
180 RND
181 X<> Z
182 4
183 *
184 STO Z
185 DSE X
186 /
187 RCL IND 05
188 +
189 ISG 05
190 LN
191 DSE 04
192 GTO 04
193 STO IND 05
194 FS? 10
195 VIEW X
196 RND
197 R↑
198 FC?C 20
199 X≠Y?
200 GTO 22
201 LASTX
202 GTO 21

203♦LBL 05
204 SF 25
205 CRFLD
206 FS?C 25
207 RTN
208 SF 25
209 PURFL
210 FC?C 25
211 GTO 06
212 SF 25
213 CRFLD
214 FS?C 25
215 RTN
216♦LBL 06
217 "NO ROOM - EM"
218 PROMPT
219 END
```

405 Bytes

Für das Programm	müssen im X-Memory
"SOLVE"	4
"DERIV"	7
"INTEG"	20
	freie Register zur Verfügung stehen.

"SOLVE"-Beispiel 1

Suchen wir das eingangs dieses Kapitels aufgeworfene Problem zu lösen. Wir müssen ein x_0 mit $\sqrt{x_0} = 10 + \ln(x_0 + 1)$ finden. Unterhalb von x_0 ist der Einsatz von Maschine 1 wirtschaftlicher, oberhalb dieses Wertes ist es günstiger, mit Maschine 2 zu arbeiten. Der erste Schritt besteht darin, eine Routine zu schreiben, die

$$f(x) = 10 + \ln(x + 1) - \sqrt{x}$$

berechnet. x ist die Stückzahl des Produktes, f(x) die *K*osten-*Diff*erenz zwischen Maschine 2 und Maschine 1. Das folgende Programm berechnet diese Differenz:

```
01♦LBL "KDIFF"          (beim Programm-Start liegt x in X)
02 SQRT
03 LASTX
04 1
05 +
06 LN
07 X<>Y
08 -
09 10
10 +
11 END                                          22 Bytes
```

Wenn Sie nun "KDIFF" und "SOLVE" laufbereit im Rechner liegen haben, ist die Lösung einfach:

1. Stellen Sie sicher, daß mindestens 4 Datenregister zur Verfügung stehen.
2. Legen Sie den Funktionsnamen (hier "KDIFF") ins Alpha-Register.
3. Tasten Sie eine Anfangsvermutung für die Lösung ein. Der Wurzel-Sucher benutzt sie als Startwert. (In unserem Beispiel kann man mit der recht rohen Abschätzung 100 beginnen; weil nur eine Wurzel vorhanden ist, führt hier nämlich jeder leidlich vernünftige Startwert zum Ziel.)
4. Stellen Sie den Anzeige-Modus gemäß der erstrebten Lösungsgenauigkeit fest. Wenn die Wurzel z.B. auf 4 signifikante Ziffern genau bestimmt werden soll, liefert 'SCI 3' die gewünschte Genauigkeit. Benötigen Sie eine auf 4 Nachkommastellen genaue Lösung (die Anzahl der signifikanten Ziffern lautet in diesem Fall 7, weil hier vom Vorkomma-Anteil der Wurzel 3 Stellen 'verbraucht' werden), setzen Sie 'FIX 4' fest. 'SCI 6' brächte natürlich ebenfalls 7 signifikante Ziffern hervor. Der Wurzel-Sucher beendet seine Arbeit, wenn zwei aufeinanderfolgende Approximationen innerhalb der festgelegten Genauigkeit übereinstimmen. Vermeiden Sie 'FIX 9', 'ENG 9' und 'SCI 9', weil dabei wegen der verwendeten Approximationsformel Rundungsfehler auftreten können. Für unser Beispiel reicht bereits 'FIX 2', denn wir wollen ja nur eine ganzzahlige Lösung ermitteln.
5. Mit 'SF 10' können Sie die fliegende Graugans vertreiben, um die aufeinanderfolgenden Approximationen der Wurzel in der Anzeige zu verfolgen. Mit 'CF 10' holen Sie die Graugans zurück.
6. Der Wurzel-Sucher wird mit 'XEQ "SOLVE"' gestartet.
7. Das Programm schließt mit einer Durchsicht des Katalogs 4 ab. Sie können diese Durchsicht unterbrechen, um sofort die Lösung zu sehen, doch nötig ist dies nicht. Der Grund für das Einfügen des Befehls 'EMDIR' ist die 'PURFL'-Wanze (vgl. Sie Kapitel 5 und Abschnitt 1F). Haben Sie eine Version des X-Funktionen-Moduls von 1C an aufwärts oder den HP-41CX, können Sie die Zeilen 95, 97, 98 und 99 gefahrlos löschen. In diesem Fall wird Ihnen die Wurzel unmittelbar nach Beendigung der Suche vorgelegt.

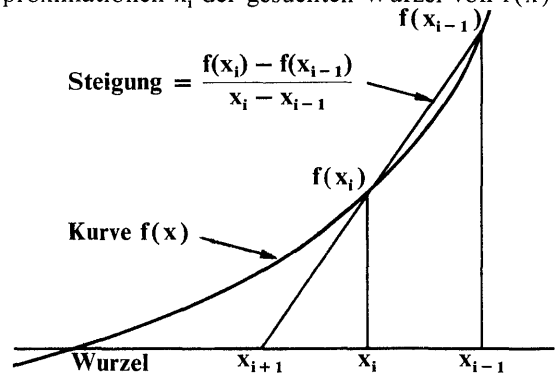
Wenn Sie den Wurzel-Sucher mit unserem Beispiel "KDIFF" betreiben, 'FIX 2' festlegen und Flag 10 setzen, erblicken Sie diese Folge von Approximationen:

101,00
215,31
236,07
239,66
239,75

Unsere Frage nach der Wirtschaftlichkeit der beiden Maschinen wird also so beantwortet: Für Stückzahlen bis 239 ist Maschine 1 billiger, für Stückzahlen von 240 an aufwärts sind die Herstellungskosten auf Maschine 2 niedriger. Zur Veranschaulichung der Wurzel-Suche können Sie "SOLVE" mit verschiedenen Anfangswerten starten. Bei gesetztem Flag 10 läßt sich dann die Wirkung verschiedener Startwerte gut beobachten. Sie werden stets bemerken, daß der Wurzel-Sucher, sobald er erst einmal in die Nähe des Lösungswertes geraten ist, *sehr schnell* konvergiert.

Der Such-Algorithmus

Das Programm "SOLVE" beruht auf einem einfachen Sekanten-Algorithmus, dessen aufeinanderfolgende Approximationen x_i der gesuchten Wurzel von $f(x) = 0$ entgegenstreben.



Zur Berechnung der Approximation x_{i+1} werden die Funktionswerte an den beiden vorangegangenen Approximationsstellen x_i und x_{i-1} benötigt.

$$x_{i+1} = x_i - \frac{f(x_i)\cdot(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \quad .$$

Zwar gibt es pathologische Beispiele von Funktionen und eigenwillig verlaufende Kurven, die sich dem Sekanten-Algorithmus nicht unterwerfen, doch in Fällen von praktischer Bedeutung ist es unwahrscheinlich, auf Funktionen zu stoßen, die sich der Wurzel-Suche durch das vorliegende Verfahren widersetzen. Daher enthält "SOLVE" nicht die komplexen Absicherungsmaßnahmen, die erforderlich wären, auch alle abseits liegenden Fälle erfolgreich zu behandeln oder wenigstens programmgesteuert auszusondern.

Sie sollten sich dessen bewußt sein, daß der Rechner die Differenz zweier Approximationswerte x_i und x_{i-1} nicht mehr hinreichend genau – bezüglich des Verfahrens, nicht bezüglich Ihrer eigenen Bedürfnisse! – berechnen kann, sobald diese Werte sehr dicht nebeneinander liegen, wenn sich also x_i und x_{i-1} nur noch um einen Betrag unterscheiden, der die Größenordnung von $10^{-8}x_i$ unterschreitet. Die Multiplikation mit $x_i - x_{i-1}$ kann dann einen die Konvergenz zerstörenden Fehler in die Rechnung bringen, der x_{i+1} unbrauchbar macht. Daher dürfen Sie beim Start von "SOLVE" i.a. nicht die volle 10-ziffrige Genauigkeit fordern.

Zeilen-Analyse von "SOLVE"

Die ersten 7 Zeilen von "SOLVE" legen den Funktionsnamen und die Anfangsvermutung x_0 in R_{00} bzw. R_{01} ab und berechnen eine zweite Vermutung x_1 zu $1{,}01 \cdot x_0$, welche nach R_{02} gelangt. Es bleibt Ihnen unbenommen, diesen Teil so abzuändern, daß Sie die zweite Vermutung selbst eingeben können. "SOLVE" benutzt insgesamt 4 Register:

Register	Inhalt
R_{00}	Funktionsname
R_{01}	vorangegangene Approximation x_{i-1}
R_{02}	gegenwärtige Approximation x_i
R_{03}	Funktionswert $f(x_{i-1})$

Die hinter 'LBL 05' (Zeile 203) beginnende Befehlsfolge legt eine aus 4 Registern bestehende Datendatei mit dem Dateinamen "**SOLVE" an. Hierbei ist mehr als ein schlichter Befehl 'CRFLD' erforderlich, um das Programm unterbrechungsfrei arbeiten zu lassen, falls eine Datei dieses Namens bereits im X-Memory liegt. Das kann z.B. geschehen, wenn "SOLVE" regelwidrig abgebrochen wurde, bevor der Befehl 'PURFL' (Zeile 209) tätig werden konnte. Flag 25 erweist sich hier als segensreich.

Die Zeilen 11 und 12 verbringen den Inhalt der Register R_{00} bis R_{02} in die neu angelegte Datei, damit diese Register bedenkenlos zur Berechnung von $f(x)$ benutzt werden können. Dann wird $f(x_0)$ ausgerechnet. Danach bringt 'SAVEX' diesen Funktionswert in das vierte Register der Datei "**SOLVE", und 'GETX' holt schließlich die Inhalte aller 4 Register in den Hauptspeicher zurück (Zeilen 16 – 17). Zu diesem Zeitpunkt sind die Register R_{00} bis R_{03} so beschickt, daß das eigentliche Iterationsverfahren beginnen kann.

Die Zeilen 19 – 22 retten die Inhalte von $R_{00} - R_{03}$ ins X-Memory zur Vorbereitung von 'XEQ IND 00' (Berechnung von $f(x)$). Dies wäre natürlich beim ersten Durchlauf der von 'LBL 01' angeführten Schleife nicht nötig, doch bei späteren Iterationen kann darauf nicht mehr verzichtet werden. Ist Flag 10 gesetzt, wird x_i mit 'VIEW' vorgezeigt, bevor die Funktion, welche $f(x)$ berechnet, aufgerufen wird. Im Anschluß an die Berechnung von $f(x)$ holt 'GETR' die von "SOLVE" benötigten Werte aus dem X-Memory nach $R_{00} - R_{03}$ zurück.

Die Zeilen 29 – 40 bringen die Inhalte der Register $R_{01} - R_{03}$ gemäß der folgenden Aufstellung auf den neuen Stand:

Register	alter Inhalt	neuer Inhalt
R_{00}	Funktionsname	Funktionsname
R_{01}	x_{i-1}	x_i
R_{02}	x_i	$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$
R_{03}	$f(x_{i-1})$	$f(x_i)$

Danach werden die Inhalte von R_{01} und R_{02} nach X geholt, dort entsprechend der anfangs bestimmten Genauigkeit gerundet und dann verglichen. Bei Übereinstimmung der gerundeten Werte verzweigt das Programm zu der bei 'LBL 21' beginnenden Schlußroutine. Andernfalls erfolgt ein Rücksprung auf 'LBL 01'.

Die Schlußroutine (Zeilen 94 – 100) löscht zunächst die Datei "**SOLVE" im X-Memory. Dies beschwört bei Versionen 1B des X-Funktionen-Moduls jene gefährliche Lage herauf, die

wir hier als 'PURFL'-Wanze bezeichnen (vgl. Abschnitt 1F): Im Anschluß an 'PURFL' gibt es keine Arbeitsdatei mehr. Dies scheint harmlos. Doch der Schein trügt. Denn wenn Sie jetzt zufällig einen Befehl, der die Gegenwart einer Datei erfordert, wie z.B. 'SEEKPT' oder 'SAVERX', ausführen und keine Datei gegenwärtig ist, trifft Sie das Verhängnis: Der Zugriff auf das Verzeichnis des X-Memorys ist plötzlich nicht mehr möglich – auf 'EMDIR' hin erhalten Sie ein unberechtigtes "DIR EMPTY". In Abschnitt 10E werden wir in die Einzelheiten eingehend beschreiben, wie eine solche 'unehrliche' Meldung mit synthetischen Methoden rückgängig gemacht werden kann. Für Versionen von 1C an aufwärts und den HP-41CX stellt sich das Problem nicht.

Um die gefährliche Klippe sicher zu umschiffen, wird hinter 'LBL 21' wieder eine Arbeitsdatei herbeigeschafft, und zwar auf dem einzig möglichen Wege, der ohne unterbrechenden Eingriff von Hand gangbar ist: mit dem Befehl 'EMDIR'. Leider hat 'EMDIR' indessen, als Programmbefehl ausgeführt, zwei unerwünschte Nebenwirkungen. Erstens nimmt das Durchblättern des Katalogs 4 kostbare Zeit in Anspruch. Zweitens bleibt, den Stapel hebend, die Anzahl der freien Register des X-Memorys in X zurück, wenn das Verzeichnis 'bis zum bitteren Ende' durchgeblättert wird. Da wir doch aber das Ergebnis von "SOLVE" zur Hand haben wollen, wenn der Programmlauf beendet ist, muß 'EMDIR' von den beiden Befehlen 'ENTER↑' und 'RDN' eingerahmt werden. Nur so gelangt das "SOLVE"-Ergebnis unabhängig davon, ob der Katalog 4 unterbrochen wird oder nicht, ins X-Register.

Es ist zu empfehlen, die Zeilen 95, 97, 98 und 99 zu löschen, wenn Sie, im Besitz eines entwanzten Moduls oder des CX, auf die 'PURFL'-Wanze keine Rücksicht zu nehmen brauchen. Sie sparen dann beträchtlich Programmlaufzeit ein.

Für die Einzelheiten der Schlußroutine 'LBL 21' mögen Sie sich vielleicht im Augenblick noch nicht sehr interessiert haben, doch soll Ihnen die Möglichkeit geboten werden, bei Bedarf hier nachschlagen zu können, um sich eingehend sachkundig zu machen. Immerhin könnte Ihnen, im Besitz der Modul-Version 1B, die obige ausführliche Beschreibung helfen, bei einem ähnlich gelagerten Fall in Ihren eigenen Programmen die durch 'PURFL' ausgelöste Gefahr zu bannen.

"SOLVE"-Beispiel 2

Aufgabe: Suchen Sie die zweite Nullstelle der Besselfunktion erster Art dritter Ordnung, d.i. $J_3(x)$. Es handelt sich dabei um den zweiten von Null verschiedenen Wert x, für den $J_3(x) = 0$ gilt. Es ist klar, daß Sie dafür das Programm "JNX" aus Abschnitt 1A im Hauptspeicher liegen haben müssen. Der Bequemlichkeit halber können Sie sich zudem eines übergeordneten Programms, eines 'Betreibers' von "JNX", bedienen, um $J_3(x)$ zu berechnen:

```
01*LBL "J3X"
02 3
03 X<>Y
04 XEQ "JNX"
05 END                                    17 Bytes
```

"J3X" nimmt einfach den gegebenen Wert x her und ruft dann "JNX" mit 3 in Y und x in X auf. Diese äußerst einfache, fast triviale Technik lohnt einen Moment der Betrachtung. Sie erweist sich nämlich häufig als äußerst nützlich im Zusammenhang mit Programmen wie "SOLVE". Denn wenn Sie eine Funktion vorliegen haben, die mehrere Eingabe-Parameter erfordert, müssen Sie sie für den Aufruf durch "SOLVE" oder ähnlich geartete Programme

abändern, was im Einzelfall lästig sein kann, oder Sie stellen sich, das ist meistens einfacher, rasch einen Betreiber (hier "J3X") des Programms, das eigentlich von "SOLVE" aufgerufen werden soll (hier "JNX"), her, um so die für "SOLVE" erforderliche Ein-Parametrigkeit 'vorzutäuschen'. Auch sollte der Namen des Betreibers nicht mehr als 6 Zeichen umfassen, damit der Befehl 'XEQ IND' in "SOLVE" arbeitsfähig bleibt. Darüber hinaus gehört es ganz allgemein zu einem guten HP-41-Programmierstil, nur Alpha-Marken von höchstens 6 Zeichen zu verwenden, damit man später (man weiß nie, was im Schoß der Programm-Zukunft verborgen liegt) nicht in den Zwang gerät, Marken für die Zwecke indirekter Ansteuerung kürzen zu müssen.

Wir wollen jetzt mit unserem Beispiel fortfahren. Zunächst einmal müssen wir uns einen ungefähren Überblick über den Verlauf der Funktion $J_3(x)$ verschaffen. Um Tastenarbeit zu sparen, weisen wir "J3X" einer passenden Taste zu und machen dann einige Versuche, damit wir eine Vorstellung von der Kurvengestalt von $J_3(x)$ gewinnen. Doch vermeiden Sie den Wert $x = 0$, denn dafür meldet "JNX" zu Recht "DATA ERROR".

x	$J_3(x)$
0,01	2,10E – 8
1,00	0,02
2,00	0,13
3,00	0,31
4,00	0,43
5,00	0,36
6,00	0,11
7,00	–0,17
8,00	–0,29
9,00	–0,18
10,00	0,06

Aus diesen Werten geht ganz offenbar hervor, daß die zweite Nullstelle von $J_3(x)$ irgendwo zwischen $x = 9$ und $x = 10$ liegt.

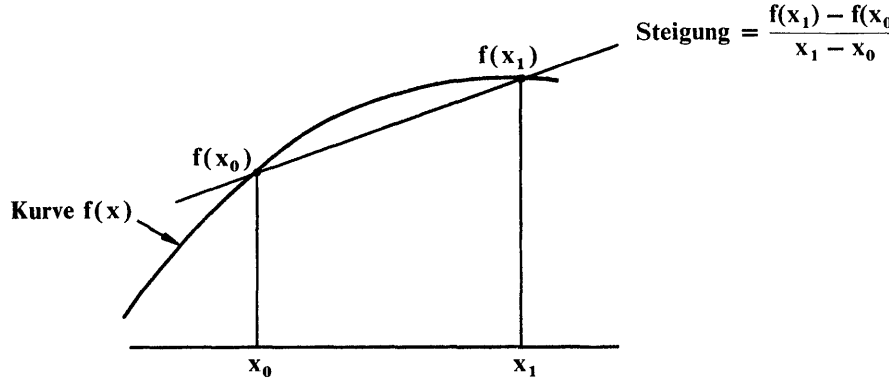
Um diese Nullstelle nunmehr genauer zu bestimmen, legen Sie 'SIZE 008' (Speicherplatzbedarf von "JNX") fest, wählen 'FIX 8' aus, füllen das Alpha-Register mit "J3X", tasten die Anfangsvermutung 9.5 ein, setzen noch Flag 10 und führen schließlich 'XEQ "SOLVE"' aus. Daraufhin sehen Sie diese Folge von Approximationen:

9,59500000
9,76155455
9,76102548
9,76102313

Wenn Sie sich eingehender mit der Technik der Nullstellenbestimmung von Gleichungen beschäftigen und andere scharfsinnigere Algorithmen dafür kennenlernen wollen, müssen Sie ein gutes Buch über Numerische Mathematik zu Rate ziehen. Besitzer des PPC ROMs (vgl. Anhang C) finden in dessen Benutzerhandbuch in der Beschreibung des Programms "SV" (*solve*) einschlägige Erläuterungen und Hinweise, die über das hier Gesagte hinausgehen.

6B. Numerische Differentiation

Es gibt viele Rechenprobleme, für deren Lösung die numerische Berechnung der Ableitung einer Funktion erforderlich wird, insbesondere dann, wenn nach den Minima oder Maxima gewisser Funktionen gefragt ist. Sofern die zur Diskussion stehenden Funktionen von verhältnismäßig einfacher Bauart sind, gelangt man am schnellsten und mathematisch saubersten zum Ziel, indem man die Funktionen den Regeln der Differentialrechnung gemäß ableitet und die Ableitungen dann mit Hilfe eines Programms auswertet und ihre Nullstellen ermittelt. Sobald die Funktionen aber verwickelter werden, ist es günstiger und manchmal fast der einzig vernünftige Weg, die analytische Lösung völlig zu vergessen und stattdessen rein numerische Methoden, für die der HP-41 wie geschaffen ist, zu verwenden. Die einfachste solcher Möglichkeiten ist die, die Funktion an lediglich zwei eng benachbarten Punkten x_0 und x_1 zu berechnen und die Steigung dann ganz 'billig' mit Hilfe der vier Werte x_0, x_1, $f(x_0)$ und $f(x_1)$ zu bestimmen:



Es leuchtet ein, daß ein so schlichter Ansatz nicht all den merkwürdigen Funktionen, denen man begegnen kann, mit hinreichender Genauigkeit gerecht zu werden vermag. Deswegen sind zahlreiche andere Methoden ersonnen worden. Eine gute Abschätzung der ersten Ableitung einer Funktion wird z.B. durch die folgende 4-Punkte-Formel gegeben:

$$f'(x) \approx [2f(x + 3h) - 9f(x + 2h) + 18f(x + h) - 11f(x)]/6h.^{*)}$$

Dieser Ausdruck stimmt für Polynome vom Grade ≤ 3 mit der analytischen Lösung *genau* überein. Für andere Funktionen ist der Fehler von der Größenordnung h^3, mithin in vielen praktischen Fällen vernachlässigbar klein. Doch Vorsicht. Was auf den ersten Blick naheliegend und nutzbar scheint, kann eine üble Falle darstellen. Bei abnehmendem h wird der Fehler nicht automatisch und einschränkungslos kleiner! Vielmehr zerstören die bei zu klein gewähltem h entstehenden Rundungsfehler erbarmungslos die erzielte Genauigkeit. Einen diese Erscheinung veranschaulichenden Nachweis werden wir in Beispiel 2 dieses Abschnittes liefern.

Wegen der durch Subtraktionen verursachten Rundungsfehler kann bei der numerischen Differentiation auf dem HP-41 nur eine Genauigkeit von 6 signifikanten Ziffern als sicher

*) Anm. d. Übers.: Man leitet diese Formel leicht her, wenn man die Taylorentwicklung an den Stellen x + h, x + 2h und x + 3h niederschreibt und nach dem 3. Glied abbricht (vgl. weiter unten: 'Zur Theorie von "DERIV"').

angesehen werden. Jede darüber hinausgehende Genauigkeit ist als zufällig zu betrachten. Fälle aus der Praxis leiden aber unter dieser Einschränkung kaum.

Das Programm "DERIV" ist Teil der im vorangegangenen Abschnitt 6A abgedruckten Programmgruppe "SOLVE"/"DERIV"/"INTEG". Es approximiert f'(x) nach der obigen 4-Punkte-Formel, indem es eine vom Benutzer zu schreibende Routine zur Berechnung von f(x) viermal aufruft. Ebenso wie "SOLVE" schützt auch "DERIV" seine Daten durch Auslagerung ins X-Memory mit Hilfe einer selbständig erzeugten, nur während des Programmlaufs lebenden Datei. "DERIV" ist daher mit jeder vom Benutzer gelieferten Routine verträglich.

Gebrauchsanweisung für "DERIV"

Die Benutzung von "DERIV" ist der von "SOLVE" ganz ähnlich. Die Datenregisteranzahl muß mindestens 7 betragen – mehr nur dann, wenn Ihre eigene Routine mehr benötigt. Der Name Ihrer Routine gehört ins Alpha-Register. Im Register Y muß die Schrittweite h, nach Belieben positiv oder negativ, liegen. Auf diese Weise können Sie auch Ableitungen von solchen Funktionen berechnen, die auf einer Seite unstetig werden. Mit $h > 0$ berechnet man die Ableitung von rechts kommend, mit $h < 0$ von links kommend. Im Register X muß der Wert x, für den f'(x) abgeschätzt werden soll, liegen.

Mit 'XEQ "DERIV"' startet man den Rechenvorgang, der f'(x) abschätzt. Die Genauigkeit der Abschätzung hängt von der gewählten Schrittweite h ab (vgl. Beispiel 2), doch kann in keinem Fall ein Ergebnis erwartet werden, das auf mehr als 6 Ziffern genau ist. Der Anzeige-Modus hat – im Gegensatz zu "SOLVE" – keinen Einfluß auf die Genauigkeit, weil hier keine Rundungen mit 'RND' durchgeführt werden.

"DERIV"-Beispiel 1

Bestätigen Sie numerisch die folgenden Beziehungen zwischen den Besselfunktionen erster und zweiter Ordnung und ihren Ableitungen:

$$J_0'(x) = -J_1(x) \quad \text{und}$$

$$J_1'(x) = J_0(x) - J_1(x)/x.$$

Zunächst beschaffen wir uns wieder – wie im "SOLVE"-Beispiel 2 – zwei Betreiber für "JNX", um ein-parametrige Funktionen $J_0(x)$ und $J_1(x)$ zu gewinnen:

```
01♦LBL "J0X"
02 0
03 X<>Y
04 XEQ "JNX"
05 END
```

```
01♦LBL "J1X"
02 1
03 X<>Y
04 XEQ "JNX"
05 END
```

Mit '0.01, ENTER↑, 1, XEQ "DERIV"' und "J0X" im Alpha-Register schätzen wir nun die Ableitung von $J_0(x)$ an der Stelle $x = 1$ ab. Die Schrittweite 0.01 gibt eine etwa 6-ziffrige Genauigkeit. Bei Schrittweiten von 0.1 oder 0.001 erhält man nur ein 4-ziffrige Genauigkeit, die jedoch ebenfalls annehmbar ist.

Vergleichen Sie die Ergebnisse Ihrer Bemühungen mit der folgenden Tabelle:

	Näherungswerte		genaue Werte	
x	$J'_0(x)$	$J'_1(x)$	$-J_1(x)$	$J_0(x) - J_1(x)/x$
1	– 0,440050367	0,325147283	– 0,440050586	0,325147101
2	– 0,576724900	– 0,064471683	– 0,576724808	– 0,064471625
3	– 0,339059067	– 0,373071550	– 0,339058958	– 0,373071608
4	0,066043233	– 0,380638955	0,066043328	– 0,380638978
5	0,327579133	– 0,112081133	0,327579138	– 0,112080944

Die obigen Zahlen erhält man mit dem nachfolgend abgedruckten Programm "VERGLCH", welches sämtliche in der Tabelle aufgeführten Ergebnisse ohne weiteren Eingriff von außen ermittelt. Sofern Sie auch noch mit einem Drucker arbeiten, kann ein Programm wie "VERGLCH" nicht nur Ihre Fingerkuppen erheblich schonen, sondern zudem das fehleranfällige Niederschreiben von Ergebnissen ersparen. Sie stellen einfach Ihren Drucker an, starten das Programm "VERGLCH" und ziehen sich zu willkommener Ruhe zurück, um eine Viertelstunde später Ihre Erwartung, daß die Rechenvorgänge in einem HP-41 stets mit äußerster Zuverlässigkeit ablaufen, auf das angenehmste bestätigt zu finden.

Programmausdruck von "VERGLCH"

```
01♦LBL "VERGLCH"   10 AVIEW       19 RCL 10       28 "J0T = "     37 "J1T = "
02 1,005           11 "J0X"       20 "J1X"        29 ARCL X       38 ARCL X
03 STO 09          12 ,01         21 XEQ "DERIV"  30 AVIEW       39 AVIEW
04♦LBL 01          13 X<>Y        22 "J1X = "     31 RCL 10       40 ADV
05 RCL 09          14 XEQ "DERIV" 23 ARCL X       32 /            41 ISG 09
06 INT             15 "J0X = "    24 AVIEW        33 X<> 10       42 GTO 01
07 STO 10          16 ARCL X      25 RCL 10       34 XEQ "J0X"    43 END
08 "X = "          17 AVIEW       26 XEQ "J1X"    35 RCL 10
09 ARCL X          18 ,01         27 CHS          36 +                  125 Bytes
```

Zeilen-Analyse von "DERIV"

"DERIV" benutzt die ersten 7 Datenregister wie folgt:

Register	Inhalt
R_{00}	f(x)
R_{01}	f(x + h)
R_{02}	f(x + 2h)
R_{03}	Funktionsname, später f(x + 3h)
R_{04}	x
R_{05}	h
R_{06}	Schleifenzähler (anfangs 0.003)

Beim Start von "DERIV" sind Stapel und Alpha-Register so beschickt:

Register	Inhalt
Y	Schrittweite h
X	Wert x, für den f'(x) bestimmt werden soll
ALPHA	Funktionsname

Die Zeilen 49 – 55 laden die Starteingaben in ihre Register und stellen einen Schleifenzähler bereit. Die Zeilen 56 – 58 legen eine aus 7 Registern bestehende Datendatei mit dem Namen "**DERIV" im X-Memory an. Die Zeilen 59 – 75 bilden eine Schleife, die viermal durchlaufen wird, um der Reihe nach die Funktionswerte $f(x + ih)$ $(i = 0,1,2,3)$ zu berechnen. Der zugehörige Schleifenzähler liegt in R_{06}. Er wird zusätzlich dazu benutzt, die Register R_{00} bis R_{03} auszusteuern (Zeile 73), um die Schleifenergebnisse abzuspeichern. Innerhalb der Schleife werden die für "DERIV" schutzbedürftigen Inhalte von $R_{00} - R_{06}$ in den Zeilen 60 – 63 ins X-Memory gerettet. Die Zeilen 64 – 70 berechnen $f(x + ih)$. Zu diesem Zweck muß der Nachkomma-Anteil .003 des Zählers mit 'INT' weggeschnitten werden (Zeile 65). Die Zeilen 71 – 73 holen die für "DERIV" benötigten Werte aus dem X-Memory wieder in den Hauptspeicher und legen das gerade erzielte Schleifenergebnis $f(x + ih)$ ins Register R_i. Die Zeilen 74 – 75 bewirken solange den Rücksprung auf 'LBL 02', bis f für alle 4 Werte $x + ih$ berechnet worden ist.

Der anschließende Teil von "DERIV" (Zeilen 76 – 93) bedient sich der in $R_{00} - R_{03}$ bereitgestellten Funktionswerte, um die Ableitung gemäß der Formel

$$f'(x) \approx [2f(x + 3h) - 9f(x + 2h) + 18f(x + h) - 11f(x)]/6h$$

näherungsweise zu berechnen, die wir vorher noch, um die rechenfreundlichen Stapeleigenschaften des HP-41 ausnutzen zu können, so umformen:

$$f'(x) \approx \{f(x + 3h) + f(x + 3h) - 9[f(x + 2h) - 2f(x + h)] - 11f(x)\}/6h.$$

"DERIV" endet wie "SOLVE" mit der 'EMDIR'-Befehlsfolge, die – wie in Abschnitt 6A ausgeführt wurde – unterbrochen werden kann oder eine Kürzung verträgt, sofern Ihr X-Funktionen-Modul von der Version 1C oder höher ist.

"DERIV"-Beispiel 2

Benutzen Sie "DERIV", um die Ableitung der Funktion

$$f(x) = x^4 + 10x^3 + 100x^2 + 1000x + 10000$$

im Punkte $x = 1$ zu berechnen. Führen Sie vor, wie sich die Genauigkeit der Abschätzung mit wechselnder Schrittweite verändert.

Den Regeln der Differentialrechnung gemäß lautet die Ableitung von $f(x)$

$$f'(x) = 4x^3 + 30x^2 + 200x + 1000,$$

woraus sich f'(1) = 1234 ergibt (ausnahmsweise einmal ohne HP-41! – mäßig scharfes Hinsehen genügt). "DERIV" ermittelt diese Werte:

Schrittweite	Näherungswert für die Ableitung
1,0	1240,000000
0,1	1234,006000
0,01	1234,000000 (0.01 ist hier die optimale Schrittweite)
0,001	1234,016667
0,0001	1233,833333
0,00001	1233,333333

Für diejenigen, die möglicherweise das obige Polynom programmtechnisch allzu umständlich berechnen wollen, sei hier auf das unter dem Namen 'Horner-Schema' bekannte Verfahren hingewiesen. Die Eigenschaft des 'Nachrutschens' von T-Inhalten beim Senken des Stapels ist wie geschaffen für das Horner-Schema. Das Verfahren stützt sich auf die einfache Tatsache, daß jedes Polynom durch ständig wiederholtes Ausklammern von Potenzen von x solange faktorisiert werden kann, bis nur noch potenzfreie x auftreten. Ein solchermaßen umgeformtes Polynom kann dann im HP-41 bezüglich Speicherplatzbedarf und Rechenzeit unübertroffen elegant programmiert werden. In unserem Beispiel sieht das so aus:

$$f(x) = (((x + 10)x + 100)x + 1000)x + 10000.$$

Daraus ergibt sich das folgende von Laufzeit verschlingenden Befehlen 'Y↑X' freie und auf Datenregister völlig verzichtende Programm

```
01♦LBL "FX"
02 ENTER↑
03 ENTER↑
04 ENTER↑
05 10
06 +
07 *
08 1 E2
09 +
10 *
11 1 E3
12 +
13 *
14 1 E4
15 +
16 END
```

30 Bytes

Jetzt findet man die Ableitung an der Stelle x = 1 ganz einfach: Schrittweite eintasten, 'ENTER↑, 1, "FX", XEQ "DERIV"'. Ihre Ergebnisse müssen mit den Zahlen der obigen Tabelle übereinstimmen. Falls Sie auch hier wieder überflüssiger Tastenarbeit aus dem Wege gehen wollen, benutzen Sie folgende kleine Routine:

```
01♦LBL "SW"
02 "FX"
03 1
04 XEQ "DERIV"
05 END
```

20 Bytes

Beachten Sie, daß die manuelle Eingabe der Schrittweite nicht durch ein 'ENTER↑' von der Eingabe der '1' durch "SW" getrennt zu werden braucht, weil sowohl Zeile 01 als auch Zeile 02 eine Stapelsperre wieder aufheben. Es lohnt sich, die im Handbuch des HP-41 gegebenen Erläuterungen zum Verhalten des Stapels aufmerksam durchzulesen und genau zu verstehen.

Zur Genauigkeit von "DERIV"

Das voranstehende Beispiel 2 zeigt, daß die Genauigkeit der Abschätzung zunächst zunimmt, wenn die Schrittweite vermindert wird. Doch tritt von einer zu kleinen Schrittweite an ein auf den technischen Grenzen des Rechners beruhendes Fehlverhalten zutage. Man kann nun ein Programm schreiben, welches die Schrittweiten in vorbestimmter Weise vermindert und jedesmal von neuem "DERIV" aufruft. Man erhält auf diese Weise eine Folge von Abschätzungen D_i, welche sich zweier Eigenschaften erfreuen muß:

1. sie muß monoton sein,
2. die Folge der absoluten Differenzen $|D_i - D_{i-1}|$ muß monoton fallen.

Sobald die Schrittweite so klein ist, daß auch nur eine dieser beiden Bedingungen verletzt wird, hat man sich mit dem unmittelbar zuvor erlangten Wert D_{i-1} die beste der durch "DERIV" erreichbaren Abschätzungen beschafft. Genau diese Überlegung liegt der im PPC ROM ansässigen Routine "FD", die mit einer Schrittweiten-Abnahme von 0.7 arbeitet, zugrunde. Auf S. 146 des PPC ROM-Handbuches finden Sie die diesbezüglichen Einzelheiten. Der Hauptnachteil bei diesem Verfahren liegt natürlich darin, daß die Laufzeit durch die mehrfach wiederholte Berechnung der Ableitung erheblich verlängert wird. In den meisten Fällen, Maxima- und Minima-Probleme eingeschlossen, ist eine so teuer erkaufte Genauigkeitsverbesserung aber gar nicht nötig. Überdies sollte "DERIV" insbesondere immer dann so schnell als irgend möglich durchlaufen werden, wenn es seinerseits von "SOLVE" aufgerufen wird. Prüfen Sie daher, ob es Ihre Anwendungen u.U. gestatten, sogar mit der einfachst möglichen aller Ableitungsabschätzungen, mit

$$f'(x) \approx [f(x + h) - f(x)]/h$$

zu arbeiten. Sie verbessert die Rechenzeit gegenüber der in "DERIV" verwendeten 4-Punkte-Formel um einen Faktor > 2.

Zur Theorie von "DERIV"

Der Ausdruck

$$f'(x) \approx [2f(x + 3h) - 9f(x + 2h) + 18f(x + h) - 11f(x)]/6h$$

gehört zu einer unübersehbaren großen Klasse von Ableitungsabschätzungen, die alle aus der sogenannten Taylor-Entwicklung der Funktion $f(x)$ gewonnen werden und die man sich am besten von Fall zu Fall, ganz nach Bedürfnis und Genauigkeitsanspruch, 'persönlich' beschafft. Beispielsweise läßt sich aus der Taylor-Entwicklung ohne Schwierigkeiten eine approximative 4-Punkte-Formel für die zweite Ableitung herleiten:

$$
\begin{aligned}
f''(x) &= a_0 f(x) + a_1 f(x+h) + a_2 f(x+2h) + a_3 f(x+3h) \\
&\approx a_0 f(x) + a_1 f(x) + a_2 f(x) + a_3 f(x) \\
&\quad + a_1 h f'(x) + 2a_2 h f'(x) + 3a_3 h f'(x) \\
&\quad + a_1 \frac{h^2}{2} f''(x) + 4a_2 \frac{h^2}{2} f''(x) + 9a_3 \frac{h^2}{2} f''(x) \\
&\quad + a_1 \frac{h^3}{6} f^{(3)}(x) + 8a_2 \frac{h^3}{6} f^{(3)}(x) + 27a_3 \frac{h^3}{6} f^{(3)}(x)
\end{aligned}
$$

Weil eine *Vier*-Punkte-Formel entstehen soll, entfallen alle Ableitungen vierter und höherer Ordnung. Da die obige Gleichung für *alle* h gelten soll, müssen die Koeffizienten a_0, a_1, a_2 und a_3 notwendigerweise dem folgenden linearen Gleichungssystem genügen:

$$
\begin{aligned}
a_0 + a_1 + a_2 + a_3 &= 0 \\
a_1 + 2a_2 + 3a_3 &= 0 \\
a_1 + 4a_2 + 9a_3 &= 2/h^2 \\
a_1 + 8a_2 + 27a_3 &= 0
\end{aligned}
$$

Aus diesen vier Gleichungen lassen sich die Koeffizienten eindeutig bestimmen. (Diese Stelle erhellt übrigens, daß vier Koeffizienten, also vier Punkte, nicht ausreichen können, um Ableitungen vierter und höherer Ordnung auf diese Weise abzuschätzen. Das System wäre sonst überbestimmt.) Die Lösung des Gleichungssystems ergibt die gesuchte 4-Punkte-Abschätzung:

$$f''(x) \approx [2f(x) - 5f(x+h) + 4f(x+2h) - f(x+3h)]/h^2.$$

Es bleibt Ihnen überlassen, ihre numerische Auswertung in ein Programm "DERIV2", das wie "DERIV" arbeitet, einzubauen.

Eine nützliche Übung ist es auch, eine 4-Punkte-Formel für $f'(x)$ auf dem hier beschrittenen Wege herzuleiten und festzustellen, daß sie mit der in "DERIV" verwendeten Formel übereinstimmt. (Hinweis: Man braucht nur das die Koeffizienten bestimmende lineare Gleichungssystem rechterhand leicht zu verändern.)

"DERIV"-Beispiel 3

Ermitteln Sie das Maximum der Funktion $J(x) = J_1(x) - J_0(x)$, welches gleich hinter dem ersten Gipfel von $J_1(x)$ liegen muß. Diese Funktion ist natürlich ohne Umstände analytisch differenzierbar:

$$J'(x) = J'_1(x) - J'_0(x) = J_1(x) + (1 - 1/x)J_1(x).$$

Sie soll hier nur dazu dienen, den vereinten Einsatz von "SOLVE" und "DERIV" vorzuführen. Der Grundgedanke ist der, mit "SOLVE" den Wert x zu bestimmen, für den die Ableitung $J'(x)$ verschwindet.

Wir beschaffen uns zunächst wieder zwei Betreiber, einen für "JNX" und einen für "DERIV":

```
01♦LBL "J1-J0"
02 0
03 X<>Y
04 XEQ "JNX"
05 -
06 END
```

Diese Routine berücksichtigt die Tatsache, daß die Berechnung von $J_0(x)$ durch "JNX" den Wert $J_1(x)$ in Y zurückläßt (vgl. Abschnitt 1A).

```
01♦LBL "DJ1-J0"
02 "J1-J0"
03 ,01
04 X<>Y
05 XEQ "DERIV"
06 END
```

Diese Routine ruft "DERIV" auf, um die erste Ableitung von J(x) approximativ zu berechnen.

Wenn Sie die beiden Betreiber eingetastet haben, führen Sie '"DJ1 – J0", 2, FIX 4, XEQ "SOLVE"' aus*); dann läuft der HP-41 an, um das Maximum von $J_1(x) - J_0(x)$ in der Nähe von x = 2 zu suchen (Suchzeit etwa 6½ Minuten!). Wollen Sie auch noch den dazugehörigen Funktionswert bestimmen, reicht es, mit 'XEQ "J1 – J0"' fortzufahren. Ihre Ergebnisse müssen lauten:

Maximum bei	Funktionswert im Maximum
$x_0 = 2{,}9386$	$J(x_0) = 0{,}6002$

Für die Berechnung des Funktionswertes im Maximum ist es nicht nötig, dieses Maximum mit hoher Genauigkeit zu bestimmen, denn die flache Gestalt von J(x) in der Umgebung des Gipfels verzeiht dort Fehler in x. Der Vollständigkeit halber sei hier aber noch der genaue Wert angegeben: $x_0 = 2{,}938642096$.

6C. Ein allgemeines Integrationsprogramm

Die Auswertung bestimmter Integrale ist, ebenso wie das Suchen von Wurzeln, eine der häufigsten auf programmierbaren Taschenrechnern durchgeführten Berechnungen. Das hier vorgestellte Programm "INTEG" erfordert die Bereitstellung einer Benutzerroutine, die g(x) berechnet, sowie die Angabe der Integrationsgrenzen a und b. Dann wertet "INTEG" das Integral

$$\int_a^b g(y)dy$$

aus. Zusammen mit dem Wurzel-Sucher "SOLVE" läßt "INTEG" außerdem die Auflösung von Gleichungen der Form

$$f(x) = \int_a^b g(x,y)dy = c$$

nach x zu. Eine numerische Integration erfordert gewöhnlich die Berechnung der Benutzerfunktion g(x) an mehr Stützstellen x als bei der numerischen Differentiation oder der Nullstellensuche. Daraus ergibt sich für "INTEG" naturgemäß eine wesentlich längere Laufzeit bis zur Antwort als bei "DERIV" und "SOLVE". Der hier für "INTEG" verwendete Algorithmus ist derselbe, der für die im PPC ROM enthaltene Routine "IG" benutzt wurde, und er ähnelt auch jenem Algorithmus, der in den Rechnern des Typs HP-34C bei Aufruf der Integrationsfunktion tätig wird.

*) Anm. d. Übers.: Wenn Sie mit einem HP-41CX arbeiten, müssen Sie in "SOLVE" hinter Zeile 15 noch die Befehle 'RCLPTA' und 'RDN' einschieben, weil Sie sonst wegen fehlender Arbeitsdatei (das in Zeile 14 aufgerufene "DERIV" schließt mit der hinter 'LBL 21' stehenden Befehlsfolge!) auf 'SAVEX' hin die Fehlermeldung "FL NOT FOUND" bekommen.

Gebrauchsanweisung für "INTEG"

Um das Integral der Funktion g(y) von y = a bis y = b zu berechnen, legt man den Namen der Routine, die g(y) ausrechnet, ins Alpha-Register und führt 'a, ENTER↑, b, XEQ "INTEG"' aus. Der Anzeige-Modus bestimmt – wie bei "SOLVE" – die Genauigkeit. Weil höhere Genauigkeit hier mit größerer Stützstellenanzahl einhergeht, bestimmt der Anzeige-Modus zugleich die Laufzeit. Die Berechnung erfolgt iterativ. Sie wird abgebrochen, sobald zwei aufeinanderfolgende Abschätzungen infolge der durch den Anzeige-Modus bestimmten Abrundung übereinstimmen. Bei gesetztem Flag 10 werden die nacheinander entstehenden Approximationen mit 'VIEW' angezeigt (und gedruckt, falls der Drucker angeschlossen ist). Zur Berechnung einer neuen Abschätzung wird etwa ebensoviel Zeit verbraucht, wie zuvor bereits verflossen ist. Mit anderen Worten: Jeder Approximationsschritt *verdoppelt* die bereits aufgewendete Laufzeit. Darum ist es dringend zu empfehlen, keine höhere Genauigkeit zu fordern, als jeweils unbedingt benötigt wird.

"INTEG"-Beispiel 1

Benutzen Sie "INTEG", um das Integral

$$\int_{-1/2}^{1/2} 3(1 - y^2)^{-1/2} dy$$

auf 6 Ziffern genau zu berechnen. – Stellen Sie sich dafür zunächst eine Routine her, die den Integranden berechnet:

```
01♦LBL "I1"
02 X↑2
03 1
04 X<>Y
05 -
06 SQRT
07 1/X
08 3
09 *
10 END                                    17 Bytes
```

Legen Sie nun den Anzeige-Modus 'SCI 5' fest. Weil "INTEG" die Zerlegung des Integrationsintervalles solange verfeinert, bis zwei aufeinanderfolgende Abschätzungen nach ihrer Abrundung übereinstimmen, liefert 'SCI 5' eine 6-ziffrige Genauigkeit. Bringen Sie dann den Funktionsnamen "I1" ins Alpha-Register. Setzen Sie Flag 10, um die Folge der Approximationen beobachten zu können. Tasten Sie schließlich die Integrationsgrenzen ein: '.5, ENTER↑, CHS, X<>Y', und starten Sie mit 'XEQ "INTEG"'. Dann erhalten Sie folgende Abschätzungen:

3,00000 + 00
3,14601 + 00
3,14214 + 00
3,14158 + 00
3,14159 + 00
3,14159 + 00

Sollten Sie die Tätigkeit Ihres HP-41 mit Ungeduld verfolgen, können Sie mit 'R/S' abbrechen, den Anzeige-Modus 'heruntersetzen' (gemeint ist: geringere Genauigkeit fordern) und danach mit 'R/S' fortfahren. Der genaue Wert dieses Integrals ist übrigens π.

"INTEG" arbeitet auch mit Funktionen, die an den Integrationsgrenzen singulär werden, dort also undefiniert sind, weil der verwendete Algorithmus den Integranden nie an den Integrationsgrenzen selbst zu berechnen versucht. Das nächste Beispiel veranschaulicht dies.

"INTEG"-Beispiel 2

Werten Sie das Integral

$$\int_0^1 y^{-1/2} dy$$

auf 4 Ziffern genau aus. – Lösung:

```
01♦LBL "I2"
02 SQRT
03 1/X
04 END                                    11 Bytes
```

und dann 'SCI 3, "I2", 0, ENTER↑, 1, SF 10, XEQ "INTEG"'. – Falls Sie Geduld genug haben sollten (knapp 2 Stunden), bekommen Sie diese Approximationen angezeigt:

1,414 + 00
1,710 + 00
1,865 + 00
1,934 + 00
1,967 + 00
1,984 + 00
1,992 + 00
1,996 + 00
1,998 + 00
1,999 + 00
1,999 + 00

Der richtige Wert des Integrals ist 2. Das vorstehende Beispiel führt entnervend deutlich vor Augen, wie langsam die Konvergenz sein kann, wenn der Integrand an einer oder gar beiden Integrationsgrenzen über alle Schranken hinaus wächst. "INTEG" läßt, so wie es hier eingerichtet ist, höchstens 13 Iterationen garantiert fehlerfrei ablaufen. n Iterationen erfordern $2^n - 1$ Berechnungen des Integranden. Dies bedeutet, daß die Integration selbst bei einfachsten Integranden erst nach über acht Stunden abgeschlossen ist, falls alle 13 Iterationen der Genauigkeit wegen durchgerechnet werden müssen. Wenn Sie dennoch weitere Iterationen zulassen wollen, weil es Ihnen nichts ausmacht, den HP-41 tagelang mit einer einzigen Aufgabe zu beschäftigen, brauchen Sie nur die Zeilen 116 und 133, in denen sich der wachsende 'Verbrauch' von Datenregistern widerspiegelt, bedarfsgemäß abzuändern. Allerdings ist nicht einmal dieser Eingriff in "INTEG" erforderlich, wenn die Routine, welche den Integranden g(y) berechnet, die Inhalte der Datenregister von R_{20} an aufwärts unangetastet läßt.

Zur Theorie von "INTEG"

Der in "INTEG" eingebaute Algorithmus ist derselbe, der für das PPC ROM-Programm "IG" verwendet wurde, und er ist jenem im HP-34C fest verdrahteten Algorithmus sehr ähnlich. Der Kern des Algorithmus besteht in einer wiederholten Intervall-Halbierung. Zunächst wird der Integrand g(y) im Mittelpunkt des Intervalls ausgerechnet und damit eine erste grobe Abschätzung vorgenommen. Dann wird g(y) in zwei weiteren Punkten, die zwischen dem Mittelpunkt und den Intervallgrenzen liegen, berechnet, um eine 3-Punkte-Abschätzung nach der Mittelpunktsregel, also eine Abschätzung von 'Histogramm-Charakter', zu erlangen. Für die dritte Iteration werden vier weitere Punkte, welche die drei zuvor gewählten Punkte eingabeln, hinzugenommen. Mit jedem Iterationsschritt wird auf diese Weise die Anzahl der Punkte, an denen eine Berechnung des Integranden erfolgt, etwa verdoppelt.

Im Verlaufe des Integrationsverfahrens werden an den Schätzwerten zwei Verbesserungen vorgenommen. Die erste Verbesserung besteht darin, daß man aus zwei aufeinanderfolgenden Mittelpunktsabschätzungen unter Verzicht auf weitere Stützstellen eine Abschätzung nach der Simpsonschen Regel vornimmt. Zwei aufeinanderfolgende Simpson-Schätzwerte werden dann ihrerseits dazu verwendet, eine Newton-Cotes-Abschätzung höherer Ordnung durchzuführen. Diese Schätzwertverfeinerung wird bei der k-ten Iteration bis zur insgesamt $(k-1)$-ten Verbesserung vorangetrieben.

Die zweite Verbesserung des Integrationsverfahrens ergibt sich aus der Verwendung nicht-gleichmäßig verteilter Stützstellen, die das zugrunde liegende Intervall schneller ausschöpfen als Stützstellen gleichen Abstandes. Die nicht-gleichmäßig verteilte Streuung der Integrationspunkte erhält man durch eine Substitution der Variablen y:

$$y = h(u) = \frac{b-a}{4}(3u - u^3) + \frac{a+b}{2}$$

$$I = \int_a^b g(y)dy = \int_{h^{-1}(a)}^{h^{-1}(b)} g\{h(u)\}h'(u)du$$

$$= \frac{3}{4}(b-a)\int_{-1}^{1} g\{\frac{b-a}{4}u(3-u^2) + \frac{a+b}{2}\}(1-u^2)du$$

und durch eine abstandsgleiche Verteilung der Stützstellen für den neuen Integranden $g\{h(u)\}h'(u)$. Prüfen Sie nach, wie eine durch Intervall-Halbierung entstehende gleichmäßige Stützstellenverteilung von Punkten u_i im Intervall $(-1,1)$ zu einer nicht-gleichmäßigen Streuung der Stützstellen $y_i = h(u_i)$ im Intervall (a,b) führt. Um die Wirkung der Substitution besonders deutlich sichtbar zu machen, wählt man als y-Intervall (a,b) am besten $(-1,1)$. Daraus ergibt sich das u-Intervall $(h^{-1}(-1), h^{-1}(1))$, also ebenfalls $(-1,1)$:

u_i	y_i
0	0
$\pm .5$	$\pm .6875$
$\pm .25, \pm .75$	$\pm .3672, \pm .9141$
$\pm .125, \pm .375, \pm .625, \pm .875$	$\pm .1865, \pm .5361, \pm .8154, \pm .9775$

Man erkennt, daß sich die y_i den Integrationsgrenzen sehr viel schneller nähern als die u_i. Aber leider kommt man nicht ganz ungestraft davon, denn die Mittelpunktsabschätzungen in der Nähe der Integrationsgrenzen zu berechnen wird immer schwieriger, weil jeder Wert $g(y_i)$ mit der Breite des Teilungsintervalles, dessen Mittelpunkt y_i ist, bewichtet werden muß.

Die vollständige Erklärung des in "INTEG" verwendeten Integrationsverfahrens verschlänge mehrere Seiten und gehört nicht mehr in dieses Buch. Wenn Sie sich also über das hier Gesagte hinaus sachkundig machen wollen, müssen Sie ein einschlägiges Buch über numerische Verfahren der Mathematik zur Hand nehmen. Eng mit unserem HP-41 verbundene diesbezügliche Quellen sind die folgenden:

1. Handbuch des PPC ROMs, Beschreibung des Programms "IG" auf den Seiten 222 – 224.
2. W. M. Kahan, "Handheld Calculator Evaluates Integrals" [HP-34C], Hewlett-Packard Journal, August 1980.

In "INTEG" benutzte Formeln

Beim k-ten Iterationsschritt ($k = 0,1,2,...$) werden folgende Werte berechnet:

erste Stützstelle: $u_0 = -1 + 2^{-k}$
i-te Stützstelle: $u_i = u_{i-1} + 2^{1-k}$

k-te Mittelpunktsabschätzung $M(k,0)$:

$$S_k = S_{k-1} + \sum_{i=0}^{2^k-1} (1 - u_i^2) g\{u_i(3 - u_i^2)\frac{b-a}{4} + \frac{a+b}{2}\}$$

$$M(k,0) = \frac{3}{4}(b - a)2^{-k}S_k.$$

Unterwirft man die Folge der $M(k,0)$, die gegen das Integral I konvergiert, dem sogenannten Romberg-Algorithmus

$$M(k,j) = M(k,j-1) + \frac{M(k,j-1) - M(k-1,j-1)}{4^j - 1},$$

erhält man eine Folge von Abschätzungen $M(k,k)$, die – verglichen mit der ursprünglichen Folge der $M(k,0)$ – konvergenzbeschleunigt gegen I strebt. Der Aufwand zur zusätzlichen Berechnung der $M(k,k)$ lohnt sich im Hinblick auf die Rechenzeit, die nötig wäre, dem angestrebten Grenzwert I mit der Folge der $M(k,0)$ entgegenzugehen. Für Einzelheiten, insbesondere die zugehörigen Fehlerabschätzungen betreffend, sei noch einmal auf die reichhaltig vorhandene Literatur verwiesen.

Zeilen-Analyse von "INTEG"

"INTEG" ist eine veränderte und verbesserte Fassung des im PPC ROM ansässigen Programms "IG", welches von Read Predmore geschrieben und von John Kennedy durchgesehen

wurde, beides PPC-Mitglieder. An einigen Stellen wurden Bytes eingespart, ein Datenregister weniger wird benötigt, und — dies ist die wesentlichste Verbesserung — die vom X-Memory gebotenen Möglichkeiten machen das Programm jetzt mit jeder vom Benutzer gelieferten Integranden-Routine verträglich.

"INTEG" benutzt die Datenregister wie folgt:

Register	Inhalt	Inhalt für die 'LBL 04'-Schleife
R_{00}	Funktionsname	
R_{01}	$(a + b)/2$	
R_{02}	$(b - a)/4$	
R_{03}	k	
R_{04}	u_i	Wert $k - j$ des 'DSE'-Zählers
R_{05}	$u_{i+1} - u_i = 2^{1-k}$	Registeradresse für M(k,j)
R_{06}	S_{k-1}	
R_{07}	M(0,0)	
R_{08}	$M(1,0) \rightarrow M(1,1)$	
R_{09}	$M(2,0) \rightarrow M(2,1) \rightarrow M(2,2)$	
R_{10}	$M(3,0) \rightarrow M(3,1) \rightarrow M(3,2) \rightarrow M(3,3)$	
usw.		

Beim Start von "INTEG" sind Stapel und Alpha-Register so beschickt:

Register	Inhalt
Y	untere Integrationsgrenze a
X	obere Integrationsgrenze b
ALPHA	Funktionsname

Die Zeilen 101 — 114 rechnen die voranstehenden Startwerte in $(a + b)/2$ und $(b - a)/4$ um, beschicken damit die vorgesehenen Register und löschen R_{03}, R_{06} und R_{07}. Dann wird Flag 20 gesetzt, damit (von Zeile 198 aus) der Test, welcher zur rechten Zeit für den Einlauf in die Schlußroutine sorgt, für $k = 0$ übersprungen wird. Danach wird im X-Memory eine 20 Register umfassende Datendatei namens "**INTEG" erzeugt (Zeilen 116 — 118). Die zu diesem Zweck angesprungene Unterroutine 'LBL 05' berücksichtigt den Fall, in welchem eine so benannte Datei bereits im X-Memory liegt.

Die hinter 'LBL 22' stehende Befehlsfolge berechnet u_0 und das Inkrement $u_{i+1} - u_i = 2^{1-k}$. Die von 'LBL 03' angeführte Schleife bildet den zeitaufwendigsten Programmteil. Zunächst wird dort der Inhalt der von "INTEG" verwendeten Datenregister ins X-Memory gebracht, damit $g(y_i)$ gefahrlos berechnet werden kann. Die Zeilen 137 — 146 stellen dafür erst einmal den notwendig aus u_i zu ermittelnden Wert

$$y_i = u_i(3 - u^2_i)(b - a)/4 + (a + b)/2$$

bereit. Dann wird $g(y_i)$ durch indirekten Aufruf der Benutzerroutine berechnet und sofort darauf der Datenregisterzustand, wie er für den fehlerfreien Fortgang von "INTEG" erforderlich ist, wiederhergestellt (Zeilen 147 — 149). Bevor der Wert $g(y_i)$ der in R_{06} liegenden Summe S_k zugeschlagen wird, erhält er noch die Bewichtung $(1 - u^2_i)$. Die Zeilen 156 — 160 ermitteln $u_{i+1} = u_i + 2^{1-k}$ und verursachen das Verlassen der Schleife 'LBL 03', sobald dieses Ergebnis größer als 1 ist und damit anzeigt, daß die Integrationsgrenzen überschritten würden.

Die Zeilen 135 – 136 bilden eine Vorsichtsmaßnahme gegen einen durch Akku- oder Batterie-Entleerung drohenden Speicherverlust. Einige Integralberechnungen können nämlich derart lange dauern, daß selbst voll aufgeladene Akkus der Beanspruchung nicht mehr gewachsen sind. Falls schwache Akkus oder Batterien ihres Zustandes wegen den HP-41 dazu veranlassen, "INTEG" auf Zeile 136 zu unterbrechen, brauchen Sie nur frische Batterien einzulegen, wieder einzuschalten und mit 'R/S' fortzufahren. Sollten Ihnen keine Ersatzbatterien oder -Akkus zur Verfügung stehen, ist es anzuraten, einen erschöpften Akku mehrere Stunden lang aufzuladen, damit hinreichend viel Energie gespeichert wird, um die beabsichtigte Berechnung zu Ende führen zu können. Die meisten HP-41 neueren Datums bewahren ihren Speicherinhalt mehr als 24 Stunden lang auf, wenn die Batterien entfernt werden. Selbst bei den ältesten Modellen können Sie sicher sein, daß die Speicherinhalte mindestens acht Stunden lang unbeschädigt überstehen. Doch wagen Sie es nicht, den Rechner bei entfernten Batterien einzuschalten! Ein "MEMORY LOST" ist dann so gut wie sicher.

Am Ende der 'LBL 03'-Schleife liegen $1 + 2^{-k}$ in X und 1 in Y. Die Zeilen 162 – 175 bereiten die Datenregister $R_{03} - R_{05}$ für die von 'LBL 04' angeführte Schleife vor, in der die weiter oben angegebene Formel des Romberg-Algorithmus auf die zuvor berechneten Werte $M(k-1,0), M(k-1,1), \ldots, M(k-1,k-1)$, die in den Registern von R_{07} an aufwärts liegen, angewendet wird. Als erstes gelangt k nach R_{04}, um dort den Wert $k-j$ des 'DSE'-Zählers zu bilden; beim ersten Schleifendurchlauf ist $j = 0$. Ins Register R_{05} wird 7 gesetzt, um das Register, welches $M(k-1,0)$ enthält, indirekt ansteuern zu können. Bei jedem 'LBL 04'-Durchlauf werden die Inhalte von R_{04} und R_{05} angepaßt: der von R_{05} wird inkrementiert, der von R_{04} dekrementiert. Die Zeilen 167 – 168 inkrementieren k in R_{03} für den nächsten 'LBL 22'-Durchlauf. In Zeile 169 entsteht 2^{-k} durch Substraktion von 1, denn die 'LBL 03'-Schleife wurde mit $1 + 2^{-k}$ in X verlassen. Die Zeilen 170 – 175 berechnen $M(k,0) = 3 \cdot 2^{-k} \cdot S_k \cdot (b - a)/4$. Beachten Sie, daß Y zu diesem Zeitpunkt immer noch 1 enthält. Y enthält in der 'LBL 04'-Schleife den Wert 4^j, der beim Einlauf in diese Schleife, also bei $j = 0, 1$ lauten muß.

Am Anfang der 'LBL 04'-Schleife liegt $M(k,j)$ in X und 4^j in Y. Nach Ausführung der Zeilen 177 – 186 enthält X den Wert $[M(k,j) - M(k-1,j)]/(4^{j+1} - 1)$, Y die Potenz $4 \cdot 4^j = 4^{j+1}$ und Z den – gemäß der verlangten Genauigkeit gerundeten – Wert $M(k-1,j)$. Dann wird $M(k,j)$ auf den Inhalt von X addiert, wodurch – der Romberg-Formel entsprechend – der Wert $M(k,j+1)$ entsteht. Die Zeilen 189 – 192 inkrementieren die Registeradresse in R_{05}, dekrementieren den Schleifenzähler in R_{04} und verzweigen nach 'LBL 04', sofern der Wert $M(k,k)$ noch nicht erreicht ist. Dies ist erst der Fall, wenn die 'LBL 04'-Schleife k-mal durchlaufen wurde. Danach liegt $M(k,k)$ in X und der gerundete Wert von $M(k-1,k-1)$ in Z und T. $M(k,k)$ wird indirekt im zugehörigen Datenregister abgelegt, dann gerundet und in der gerundeten Fassung mit $M(k-1,k-1)$ verglichen. Stimmen die gerundeten Werte überein, wird $M(k,k)$ aus L zurückgeholt und als Ergebnis vorgewiesen. Andernfalls erfolgt der Rücksprung auf 'LBL 22', wo die Berechnung von $M(k+1,0)$ beginnt, die bis $M(k+1,k+1)$ vorangetrieben wird, sofern müde Batterien der Sache nicht ein Ende setzen.

"INTEG" mündet genau wie "SOLVE" und "DERIV" in die den Befehl 'EMDIR' enthaltende 'LBL 21'-Routine, um dem Befehl 'PURFL' nicht Gelegenheit zu geben, die Tretmine 'keine Arbeitsdatei vorhanden' zu legen, über die man, wie Sie inzwischen wissen, ahnungslos stolpern kann, wenn man mit der 1B-Version der X-Funktionen arbeitet.

"INTEG"-Beispiel 3

Bestätigen Sie numerisch die Beziehung

$$\int_2^3 J_1(x)dx = J_0(2) - J_0(3).$$

Lösung: Setzen Sie 'FIX 4' fest, tasten Sie den "JNX"-Betreiber

```
01♦LBL "J1X"
02 1
03 X<>Y
04 XEQ "JNX"
05 END                                    17 Bytes
```

ein, und führen Sie '2, ENTER↑, 3, "J1X", SF 10, XEQ "INTEG" aus. Sie erhalten die Approximationen:

0,4971
0,4831
0,4839
0,4839

Um sie mit dem genauen Integralwert zu vergleichen, müssen Sie '0, ENTER↑, 2, XEQ "JNX" und '0, ENTER↑, 3, XEQ "JNX" ausführen. Das ergibt

$$\begin{aligned} J_0(2) &= \quad 0{,}2239 \\ -J_0(3) &= \underline{-(-0{,}2601)} \\ &\quad\ 0{,}4840 \end{aligned}$$

also eine Übereinstimmung innerhalb einer annehmbaren Genauigkeit.

Hinweis d. Übers.: Beachten Sie die Fußnote zu "DERIV"-Beispiel 3, falls Sie "INTEG" von "SOLVE" aufrufen lassen wollen, um eine Gleichung der Form

$$f(x) = \int_a^b g(x,y)dy = c$$

auf einem HP-41CX zu lösen.

KAPITEL 7

Ein Postadressen-Programm

Das in diesem Kapitel behandelte Programm "NAP" (*N*ame/*A*dresse/Tele*p*hon) führt vor, wie Textdateien dazu benutzt werden können, Blöcke von Alpha-Daten geordnet im X-Memory abzulegen. Das Programm wurde von Alan McCornack geschrieben. Es hat hauptsächlich deswegen Aufnahme in dieses Buch gefunden, weil ein Musterprogramm zur Hand liegen sollte, dessen sorgfältige Durchsicht sich für den Leser lohnt. Möglich auch, daß Sie es unmittelbar für eigene Zwecke einsetzen. Der verhältnismäßig klare Aufbau vermag jedenfalls gewiß, Ihre Vorstellungen von der Einrichtung einer eigenen 'Datenbasis' vorteilhaft zu beeinflussen. So ist z.B. Alan's Gebrauch von 'GETKEY' im Korrektur-Teil von "NAP" recht lehrreich, weil Sie sich dort ansehen können, wie man mit Hilfe dieses Befehles bequem in einem bestimmten Programm-Abschnitt verweilen und nach ganz Wunsch darin umherwandern kann.

Das Programm "NAP" erwartet die zu einem vollständigen Listen-Eintrag gehörigen Daten

Name, Adresse, Telephon-Nr., Verschiedenes

im folgenden Format:

Eingabe	Höchstlänge
Name	24 Zeichen
Adresse	
Zeile 1	24 Zeichen
Zeile 2	24 Zeichen
Zeile 3	24 Zeichen
Tel.-Nr.	22 Zeichen
Verschiedenes	22 Zeichen

Die den Daten auferlegte Beschränkung auf 24/22 Zeichen Länge hat den Zweck, die einzelnen Angaben beim Ausdruck in jeweils einer eigenen ungebrochenen Zeile unterzubringen. Die zusätzliche Beschränkung der beiden letzten Angaben auf 22 Zeichen erlaubt es, diese Daten durch Voranstellen zweier Leerzeichen optisch abgesetzt auszudrucken. In den Texten sind sämtliche Zeichen zugelassen, einschließlich der Sonderzeichen, die dem Alpha-Register nur mit 'XTOA' eingefügt werden können.

Das Programm "NAP" formt aus jedem Listen-Eintrag einen Block von sechs Sätzen (vgl.

zu diesem Begriff Abschnitt 3A). Ist n die (Ordnungs)nummer des gesamten Eintrages, lauten Satz-Nummer und Satz-Inhalt wie folgt:

Satz-Nr.	Satz-Inhalt
6n	Name
6n + 1	Adressen-Zeile 1
6n + 2	Adressen-Zeile 2
6n + 3	Adressen-Zeile 3
6n + 4	Telephon-Nr.
6n + 5	Verschiedenes

Die Einträge werden aufsteigend, beginnend mit 0, numeriert. Bei 10 Einträgen enthält die Textdatei mithin 60 Sätze. Die einzelnen Einträge haben die Nummern 0 bis 9.

Gebrauchsanweisung für das Programm "NAP"

1. Vor dem ersten Gebrauch von "NAP" müssen Sie von Hand eine Textdatei mit dem Namen "ML" (*m*ailing *l*ist = Postliste) erzeugen. Entscheiden Sie sich also für eine geeignete Dateigröße, legen Sie die gewählte Zahl ins X-Register, füllen Sie das Alpha-Register mit "ML", und führen Sie 'CRFLAS' aus. Die Dateigröße sollte natürlich so bestimmt werden, daß die ins Auge gefaßte Postliste Platz in der Datei "ML" findet. Jeder Eintrag benötigt ein Register für je sieben Zeichen Text sowie ein weiteres Register für die 6 Satzlängenbytes (vgl. Abschnitt 3A). Im allgemeinen wird ein Eintrag mit 8 bis 12 Registern auskommen. Steht das gesamte X-Memory von 603 Registern Umfang für die Postliste "ML" zur Verfügung, lassen sich also etwa 50-75 Einträge darin unterbringen. Sollte dies Ihren Bedürfnissen nicht genügen, müssen Sie den Zeichenbedarf durch Abkürzung einiger Daten herunterdrücken. Es wird in den meisten Fällen nicht schwer sein, einzelne Angaben so zu stutzen, daß ihr Informationsgehalt nicht darunter leidet.
2. Liegt die ASCII-Datei "ML" erst einmal an Ort und Stelle, bietet Ihnen das Programm "NAP" viele Möglichkeiten, mit der Datei zu arbeiten. Und zwar sind die fünf Tasten der obersten Reihe mit den verschiedenen Programmmteilen belegt, so daß Sie Ihre Auswahl durch einfachen Tastendruck im USER-Modus treffen können. Sollten diesen Tasten schon andere Funktionen oder globale Marken zugewiesen sein, ist es zu empfehlen, die Zuweisungen unter Verwendung des Programms "SK" (*s*uspend *k*ey assignments) aus Abschnitt 10G vorübergehend aufzuheben. Oder Sie löschen die Zuweisungen einzeln von Hand oder bequemer mit der Routine "KZL" aus Abschnitt 4F; zusätzlich müssen Sie auch noch die umgeschaltete Taste 'X↑2' freistellen. Es sei in diesem Zusammenhang an die in Abschnitt 4F kurz angesprochene im USER-Modus geltende Ausführungshierarchie erinnert. Diese bevorrechtigt bekanntlich im Konfliktfall globale Marken und System-Funktionen gegenüber lokalen Marken und kann daher sehr hinderlich sein, wenn man sie unberücksichtigt läßt.
3. Schalten Sie in den USER-Modus. Stellen Sie sicher, daß wenigstens ein Datenregister zur Verfügung steht, denn "NAP" bedient sich des Registers R_{00}. Sorgen Sie dafür, daß das Ton-Flag 26 gesetzt ist; am einfachsten durch Aus- und Einschalten. Löschen Sie Flag 21, es sei denn, Ihr Drucker ist angeschlossen und soll die Ausgaben drucken.
4. Springen Sie mit 'GTO "NAP"' in das Programm. Wenn Sie die Tasten der Kopfzeile gemäß Punkt 2 freigestellt haben, finden Sie dort jetzt die folgenden Programmteile:

'A': Eintrag vornehmen
'B': Zeichenkette suchen
'C': Eintrag anzeigen/ausdrucken 'c': gesamte Datei anzeigen/ausdrucken
'D': Eintrag löschen
'E': Eintrag ändern/berichtigen

Jedes dieser Programmteile soll nun in allen Einzelheiten beschrieben werden. Wenn Sie mit einer leeren Datei "ML" beginnen, werden Sie wahrscheinlich erst einmal ein paar Einträge vornehmen wollen, um Übungsmaterial zur Hand zu haben. Drücken Sie also einfach die Taste 'A' oder – falls Sie noch nicht 'GTO "NAP"' ausgeführt haben – 'XEQ "NAP"', um mit dem Sprung ins Programm "NAP" sogleich den Programmteil 'A' zu starten.

'A': Eintrag vornehmen

Das Programm fordert Sie mit "NAME 0?" dazu auf, die erste Zeile eines 6-zeiligen Eintrages einzugeben, und zwar im ALPHA-Modus, so daß Sie sofort damit beginnen können, bis zu 24 Zeichen einzutasten. Mitzuzählen brauchen Sie dabei natürlich nicht, weil sich der Rechner mit einem Warnton meldet, wenn der Überlauf des Alpha-Registers droht. Schließen Sie das Eintasten des Namens mit 'R/S' ab. Die nächste Eingabe-Aufforderung lautet "ADD. L1?", also 'Zeile 1 der Adresse eingeben'. Auch hier schließen Sie wieder mit 'R/S'. Beantworten Sie in derselben Weise die Aufforderungen zur Eingabe der Zeilen 2 und 3 der Adresse. Soll eine Zeile leer bleiben, wird die entsprechende Aufforderung mit einem bloßen 'R/S' beantwortet. Auf "PHONE?" hin geben Sie die Telephon-Nummer ein. Denken Sie dabei daran, sich auf 22 Zeichen zu beschränken, um den späteren Ausdruck optisch nicht zu verunstalten. Wenn Sie im Zweifel sind, ob Sie die Schranke eingehalten haben, können Sie Leerzeichen hinzufügen, bis sich der Warnton hören läßt, und dann zweimal die Korrekturtaste betätigen. Danach wird die Eingabe mit 'R/S' abgeschickt. Die letzte Aufforderung lautet "MISC.?" und gibt Ihnen Gelegenheit, in einem sechsten Satz zusätzliche Information unterzubringen. Hierfür sollten Sie ebenfalls höchstens 22 Zeichen benutzen, wenn das Druckbild einwandfrei sein soll.

Sobald sämtliche zu einem Eintrag gehörenden Angaben eingetastet worden sind, kann man – die für den Eintrag verbrauchten Register vor Augen ("RGS." = n) – mit 'R/S' oder 'A' zur Eingabe des nächsten Eintrages übergehen.

WARNUNG: Verlassen Sie den Programmteil 'A' nie, bevor Sie alle 6 Eingabe-Aufforderungen programmgerecht beantwortet haben. Wenn Ihnen ein Fehler unterlaufen ist, versuchen Sie nicht, diesen vorzeitig auf eigene Faust zu berichtigen, sondern fahren Sie zunächst damit fort, alle 6 Sätze eines Eintrages planmäßig in die Datei zu bringen, andernfalls zerstören Sie den Dateiaufbau. Berichtigen Sie den Fehler anschließend unter Verwendung des Programmteils 'E' (Eintrag ändern), der weiter unten beschrieben wird. Falls Sie einen *unvollständigen* Eintrag ganz löschen wollen, bedienen Sie sich des Programmteils 'D' (Eintrag löschen), der ebenfalls gleich erläutert wird. Doch tun Sie dies *nur, wenn* der Datei inzwischen keine weiteren Einträge hinzugefügt wurden. Ein unvollständiger Eintrag darf nämlich mit 'D' nur dann gelöscht werden, wenn er der letzte Eintrag ist, weil mit 'D' stets 6 Sätze getilgt werden, Folge-Einträge also beschädigt würden.

'B': Zeichenkette suchen

Mit der Taste 'B' springen Sie in den Programmteil, der es Ihnen ermöglicht, die Datei nach einer bestimmten Zeichenkette zu durchsuchen. Sie werden mit "FIND?" zur Eingabe der gesuch-

ten Kette, die bis zu 24 Zeichen umfassen kann, aufgefordert. Das Programm hält zu diesem Zweck im Alpha-Modus an. Da die gesamte Datei von vorn an durchsucht wird, kann die Kette nicht übersehen werden, wenn sie irgendwo in "ML" vorhanden ist, allerdings nur insoweit sie nicht satzübergreifend liegt, sondern vollständig in einem Satz enthalten ist (vgl. Abschnitt 3E).

Bei erfolgloser Suche gelangt der Wert — 1 ins X-Register. Wird die Kette dagegen gefunden, erscheint die Meldung "ENTRY NO. n", um mitzuteilen, welcher Eintrag die gesuchte Kette enthält. Anschließend werden alle 6 Sätze des Eintrages vorgewiesen.

Halten Sie das Programm mit 'R/S' an, wenn Sie den Eintrag vor sich haben, der gesucht wurde. Andernfalls werden auch noch die nachfolgenden Einträge der Suche unterworfen, allerdings mit der Einschränkung, daß nur mehr die ersten 6 Zeichen der ursprünglichen Kette für die weitere Nachforschung maßgebend sind. Solange Sie den Suchvorgang nicht mit 'R/S' unterbrechen, werden alle Einträge, welche die gesuchte Kette bzw. deren erste 6 Zeichen enthalten, Satz für Satz angezeigt. Sobald das Dateiende erreicht ist und somit keine weiteren Übereinstimmungen zwischen Kette und Dateiinhalten mehr festgestellt werden können, gelangt die dies bezeugende — 1 nach X, und das Programm hält an.

Die gesuchte Kette braucht, um gefunden zu werden, natürlich nicht in dem Satz, der den Namen enthält, zu liegen. Sie kann vielmehr Bestandteil jeden Satzes der Datei "ML" sein. Wenn Sie z.B. unter 'Verschiedenes' Geburtstage aufzuzeichnen wünschen, sagen wir im Format "bTT/MM/JJ" mit "b" als 'Etikett', das die nachfolgende Zahlengruppe als Geburtsdatum ausweist, können Sie mit der Suche nach "/6/" die Einträge aller der Personen, die im Juni Geburtstag haben, herausfiltern. Die Kleinbuchstaben a bis e sind übrigens besonders gut als Etiketten geeignet, weil sie sich einerseits umstandslos eintasten lassen und andererseits selten benötigt werden, allenfalls einmal für eine Hausnummer.

'C': Eintrag anzeigen/ausdrucken

Drücken Sie die Taste 'C', wenn Sie sich einen bestimmten Eintrag ansehen wollen. Das Programm fragt Sie mit "ENTRY NO. ?" nach der gewünschten Adresse. Es reicht, deren Nummer einzutasten (beachten Sie hierbei, daß die Nummer des ersten Eintrages 0 lautet und nicht etwa 1, die des zweiten 1, usw.) und 'R/S' zu drücken. Alle 6 Sätze, welche zu dem angesteuerten Eintrag gehören, werden der Reihe nach angezeigt oder, wenn der Drucker angeschlossen und Flag 21 gesetzt sind, auch ausgedruckt.

'c': gesamte Datei anzeigen/ausdrucken

Drücken Sie die umgeschaltete Taste 'c', wenn Sie sich die gesamte Postliste ansehen wollen. Eine Eingabe ist in diesem Fall nicht erforderlich. Satz für Satz wird die gesamte Datei "ML" angezeigt oder bei erfüllten Druckbedingungen (Drucker angeschlossen, Flag 21 gesetzt) ausgedruckt. Die Inhalte der beiden letzten Sätze jeden Eintrages, die Telephon-Nr. und die Zeile 'Verschiedenes', werden dabei um zwei Druckstellen nach rechts eingerückt. Je zwei Einträge werden durch zwei Leerzeilen voneinander getrennt.

Die Auflistung der Adressen wird mit einer 'Datei-Bilanz' abgeschlossen: In der Anzeige erscheint die Anzahl der von der Postliste inzwischen belegten Register sowie die Dateigröße von "ML". Bei einem Ausdruck der Liste bildet diese Bilanz die letzte Druckzeile.

'D': Eintrag löschen

Drücken Sie die Taste 'D', wenn Sie einen Eintrag löschen wollen. Das Programm fragt Sie mit "ENTRY NO. ?" nach der Nummer der zu tilgenden Adresse. Tasten Sie diese Nummer ein, und drücken Sie 'R/S', um die Löschung zu veranlassen. Es versteht sich von selbst, hierbei

besondere Vorsicht walten zu lassen und nicht blindlings die Axt anzulegen. Im Zweifelsfall können Sie sich mit Programmteil 'C' jederzeit vergewissern, ob Sie auch wirklich die richtige Nummer im Sinn hatten. Vergessen Sie hier weniger denn je, daß der n-te Eintrag die Nummer n – 1 hat.

'E': Eintrag ändern/berichtigen

Drücken Sie die Taste 'E', wenn Sie eine Adresse ändern oder einen Fehler darin berichtigen wollen. Das Programm meldet sich zunächst wieder mit der üblichen Frage "ENTRY NO. ?". Tasten Sie als Antwort die Nummer des änderungsbedürftigen Eintrages ein, und drücken Sie 'R/S'. Daraufhin wird Zeile für Zeile der angeforderten Adresse angezeigt bzw. gedruckt, jedoch nicht 'atemlos' hintereinander. Vielmehr folgt jeder Einzelzeile nach einer 'PSE' die Frage "OK?": Bedarf eine Zeile keiner Änderung, weil sie in Ordnung ist, können Sie dies dem Programm mit der Taste 'Y' oder mit 'R/S' mitteilen. Wollen Sie die Zeile noch einmal betrachten, weil Sie ohne Drucker arbeiten und die Pause zu kurz für eine sichere Überprüfung war, läßt sie sich mit 'ALPHA' (beliebig oft) in die Anzeige zurückbringen. Wollen Sie den Programmteil 'E' ganz verlassen, müssen Sie entweder die Korrekturtaste oder die 'ON'-Taste drücken. Soll dagegen eine Berichtigung vorgenommen werden, ist die Taste 'N' zuständig. Versäumen Sie es, innerhalb von 10 Sekunden überhaupt irgendeine Taste zu betätigen, wird dieselbe Zeile von neuem vorgezeigt, jedesmal gefolgt von dem fragenden "OK?", so als ob die Taste 'ALPHA' gedrückt worden wäre. Jede andere Taste als die, deren Bedeutung eben erläutert wurden, hat dieselbe Wirkung wie die Taste 'N'.

Auf die Tasten 'Y' oder 'R/S' hin, die beide bekunden, daß an der vorgezeigten Zeile nichts auszusetzen ist, findet ein Übergang zum nächsten Satz der Datei "ML" statt; am Ende eines Eintrages gelangt man dabei über dessen Grenzen hinweg zum Anfang des nächsten Eintrages.

Wenn Sie die Taste 'N' oder eine ihr in der Wirkung gleichgestellte Taste (das sind alle außer 'ON', 'ALPHA', '←', 'Y' und 'R/S') drücken, erscheint die Aufforderung "LINE?". Sie wird mit dem Eintasten einer Ersatzzeile beantwortet. Dann drückt man 'R/S'. Soll die beanstandete Zeile ersatzlos gelöscht werden, reicht ein bloßes 'R/S' als Antwort auf "LINE?". Haben Sie sich anders besonnen und wollen Sie eine mit "LINE?" zur Änderung anstehende Zeile so bestehen lassen, wie sie ist, müssen Sie den ALPHA-Modus mit 'ALPHA' verlassen. Danach steht es Ihnen frei, den Korrekturgang mit 'E' ganz von vorn zu beginnen oder mit 'XEQ 84' zum nächsten Satz der Datei vorzustoßen.

Zeilen-Analyse von "NAP"

Das Programm "NAP" enthält mehrere Unterroutinen, die von verschiedenen Programmteilen gleichermaßen benötigt werden. Hinter der dem Programm durch diese Routinen aufgeprägten Struktur verbirgt sich aber nicht nur schlichte Unterprogrammtechnik, sondern eine Methode, die unter dem Namen 'modulare Programmierung' bekannt ist: Voneinander unabhängige Programmteile mit klar festgelegten Ein- und Ausgangsbedingungen, sogenannte 'Module', werden zu einem Programmpaket zusammengeschnürt. Dabei erlangt man nicht nur den von der Unterprogrammtechnik her wohlbekannten Vorteil der Verminderung des Speicherplatzbedarfes und die Möglichkeit, dieselben Programmteile von verschiedenen Stellen aus anspringen zu können. Vor allem wird ein Programm wartungsfreundlich und läßt sich, insbesondere dann, wenn sehr verwickelte Aufgaben zu lösen sind, wesentlich leichter zu einem lauffähigen Werkstück entwickeln, weil sich die einzelnen übersichtlichen und 'pflegeleichten' Teilstücke viel einfacher als das ganze Programm fehlerfrei gestalten lassen.

Programmausdruck von "NAP"
(Barcode im Anhang D)

```
01♦LBL "NAP"
02♦LBL A
03 XEQ 90
04 SF 25
05 5
06 CHS

07♦LBL 05
08 6
09 +
10 SEEKPT
11 FS? 25
12 GTO 05
13 LASTX
14 /
15 "NAME "
16 XEQ 92
17 SIGN
18 X<>Y
19 AON
20 XEQ 94
21 XEQ 93
22 XEQ 93
23 XEQ 93
24 "PHONE"
25 XEQ 94
26 "MISC. "
27 XEQ 94
28 AOFF
29 "RGS. = "
30 7
31 /
32 XEQ 92

33♦LBL 98
34 RCLFLAG
35 CF 21
36 AVIEW
37 STOFLAG
38 RDN
39 RTN
40 GTO A

41♦LBL 90
42 0
43 GTO 06

44♦LBL 91
45 "ENTRY NO. ?"
46 PROMPT
47 6
48 *

49♦LBL 06
50 "ML"
51 SEEKPTA
52 7
53 RTN

54♦LBL 92
55 INT
56 RCLFLAG
57 FIX 0
58 CF 29
59 ARCL Y
60 STOFLAG
61 RDN
62 RTN

63♦LBL 93
64 "ADD. L"
65 X<>Y
66 XEQ 92
67 X<>Y
68 ISG Y

69♦LBL 94
70 "⊢?"
71 XEQ 98
72 " "
73 STOP
74 APPREC

75♦LBL 07
76 1
77 +
78 RCLPT
79 FRC
80 1 E3
81 *
82 +
83 RTN

84♦LBL B
85 XEQ 90
86 "FIND?"
87 AON
88 STOP
89 AOFF
90 ASTO 00

91♦LBL 08
92 POSFL
93 X<0?
94 RTN
95 6
96 /
97 LASTX
98 X<>Y
99 "ENTRY NO. "
100 XEQ 92
101 XEQ 99
102 *
103 SEEKPT
104 RDN
105 XEQ 95
106 ARCL 00
107 GTO 08

108♦LBL 96
109 SF 25
110 ARCLREC
111 XEQ 99
112 CLA
113 FS? 25
114 GTO 07
115 RTN

116♦LBL C
117 XEQ 91

118♦LBL 95
119 ADV
120 CLA
121 XEQ 96
122 XEQ 96
123 XEQ 96
124 XEQ 96
125 " "
126 XEQ 96
127 " "

128 XEQ 96
129 ADV
130 CLD
131 RTN

132♦LBL D
133 XEQ 91

134♦LBL 09
135 DELREC
136 DSE L
137 GTO 09
138 RTN

139♦LBL c
140 XEQ 90
141 CLA

142♦LBL 10
143 SF 25
144 X<>Y
145 SEEKPT
146 6
147 +
148 X<>Y
149 FS? 25
150 XEQ 95
151 FS? 25
152 GTO 10
153 7
154 /
155 FLSIZE
156 X<>Y
157 XEQ 92
158 "⊢ : "
159 X<>Y
160 XEQ 92
161 "⊢ RGS."

162♦LBL 99
163 RCLFLAG
164 SF 25
165 PRA
166 RCLFLAG
167 FS?C 21
168 FC? 25
169 AVIEW

170 STOFLAG
171 RDN
172 FS? 21
173 FC? 25
174 PSE
175 STOFLAG
176 RDN
177 RTN

178♦LBL E
179 XEQ 91

180♦LBL 71
181♦LBL 84
182 SF 25
183 GETREC
184 FC? 25
185 GTO 01
186 XEQ 99
187 "OK?"
188 XEQ 98
189 GETKEY
190 CLD
191 RDN
192 GTO IND T
193 "LINE?"
194 XEQ 98
195 " "
196 AON
197 STOP
198 AOFF
199 DELREC
200 INSREC
201 GTO 84

202♦LBL 00
203♦LBL 04
204 RCLPT
205 INT
206 SEEKPT
207 GTO 04

208♦LBL 44
209♦LBL 01
210 CLST
211 END
```

447 Bytes

Hinter 'LBL 90' wird eine 0 ins X-Register gebracht, um den Dateizeiger für "ML" auf den Dateianfang zu setzen (Zeile 51). Hinter 'LBL 91' wird ein Wert berechnet, der den Dateizeiger auf den ersten Satz eines ausgewählten Eintrages setzt. Zeile 52 legt den Anfangswert 7 eines Zeichenzählers, dessen ganzzahliger Anteil später als 'Anzahl belegter Register' vorgewiesen wird, fest (Anm. d. Übers.: richtig wäre 6, damit bei voll ausgeschöpften Registern nicht eines zuviel vorgewiesen wird).

Der von 'LBL 92' angeführte Modul hängt den ganzzahligen Anteil der in X liegenden Zahl dem Inhalt des Alpha-Registers im 'FIX 0'-Format ohne Dezimaltrennzeichen an. Die Befehle 'RCLFLAG' und 'STOFLAG' sorgen dafür, daß das ursprüngliche Anzeigeformat wiederhergestellt wird. Der von 'LBL 98' angeführte Modul weist den Inhalt des Alpha-Registers vor, ohne ihn zu drucken. Der von 'LBL 99' angeführte Modul druckt entweder den Alpha-Inhalt, oder er zeigt ihn mit einer Pause an, falls kein Drucker angeschlossen oder ein angeschlossener nicht druckbereit ist. Er weicht insoweit von der in Abschnitt 4C vorgestellten Routine "ADA" ab, als er den Zustand von Flag 25, der hier dazu benötigt wird, das Dateiende ("END OF FL") zu entdecken, bewahrt.

Der von 'LBL 96' angeführte Modul holt einen Satz aus der Datei "ML" ins Alpha-Register. Statt des auf den ersten Blick näherliegenden Befehles 'GETREC' wird der Befehl 'ARCLREC' verwendet, weil auf diese Weise das Einrücken der Zeilen 'Tel.-Nr.' und 'Verschiedenes' bequemer bewerkstelligt werden kann. Der von 'LBL 99' angeführte Modul ist für die eigentliche Ausgabe (Druck/Anzeige) zuständig. Solange man auf Zeile 110 keiner "END OF FL"-Bedingung begegnet, wird die von 'LBL 07' angeführte Befehlsfolge angesprungen. Diese rechnet die Zeichenanzahl des gerade abgeholten Satzes aus (Zeilen 78 – 81), erhöht sie, das Satzlängenbyte berücksichtigend, um 1 und schlägt das Ergebnis dem in X liegenden Zeichenzähler zu. Einzelheiten über die Inhalte der Register einer ASCII-Datei finden Sie in Abschnitt 10C.*)

Der Modul '95' wird von den Programmteilen 'B', 'C' und 'c' dazu benutzt, alle 6 Zeilen einer Adresse auszugeben. Nach einem Papiervorschub und der Löschung des Alpha-Registers werden die ersten 4 Zeilen der Adresse unter Verwendung des Moduls '96' ausgegeben. Den letzten beiden Zeilen werden vor der Ausgabe je zwei Leerzeichen vorangestellt, um sie einzurücken. Danach folgen noch ein Papiervorschub und die abschließende 'Reinigung' der Anzeige.

Unter 'LBL A' findet die Neuaufnahme vollständiger Einträge in die Datei "ML" statt. In den Zeilen 04 – 12 wird nach der zuletzt eingetragenen Adresse gesucht. Flag 25 läßt sich dabei zweifach benutzen: erstens, um die Meldung "END OF FL" zu unterdrücken und zweitens, um festzustellen, ob das Dateiende schon erreicht ist. Die Zeilen 15 – 28 fordern zur Eingabe der 6 Zeilen, aus denen eine vollständige Adresse zusammengesetzt ist, auf und fügen sie als 6 neue Sätze der Datei "ML" an. In dem hinter 'LBL 93' stehenden Teilstück finden Sie einen kleinen Kunstgriff, der Bytes einspart. Es werden dort die drei Aufforderungen "ADD. L1?", "ADD. L2?" und "ADD. L3?" erzeugt. Die Zeilen 17 – 18 setzen den Anfangswert 1 eines Zählers ins Y-Register. In Zeile 68 wird dieser Zähler bei jedem Aufruf von 'LBL 93' inkrementiert. Die Tatsache, daß unmittelbar hinter dem 'ISG'-Befehl eine Marke steht ('LBL 94') enthebt uns der Mühe des Nachdenkens über die Sprungbedingung, denn eine Marke ist der Wirkung nach ein 'no operation'-Befehl. Hinter 'LBL 94' wird dem Alpha-Inhalt ein Fragezeichen angehängt. Alsdann erscheint vermittels des Moduls '98' der im Alpha-Register zusammengesetzte Text als Eingabe-Aufforderung ohne Ausdruck. Danach gelangt ein einzelnes Leerzeichen ins

*) Anm. d. Übers.: Beachten Sie an dieser Stelle einen nachahmenswerten Trick: Die Benutzung von 'RCLFLAG' und 'STOFLAG' im Modul '99' erlaubt es, die Wirkungsweise von Flag 25 sowohl bezüglich des Befehles 'PRA' (Zeile 165) als auch des Befehles 'ARCLREC' (Zeile 110) *wechselseitig unabhängig voneinander* auszunutzen.

Alpha-Register, und das Programm hält an, damit die Eingabe vorgenommen werden kann. Drückt man jetzt einfach 'R/S', besteht der Inhalt des entstehenden Satzes nur aus einem einsamen 'gesichtslosen' Leerzeichen. Andernfalls wird das, was Sie auch immer ins Alpha-Register verbringen, Inhalt des Satzes, der in Zeile 74 der Datei "ML" angefügt wird. Die 'LBL 07'-Routine rechnet anschließend noch den Zeichenzähler in X auf den neuen Stand hoch. Der Programmteil 'A' schließt mit der Bekanntgabe der von dem neuen Eintrag belegten Register, wobei 'angebrochene' Register als volle Register gezählt werden.

Vom Programmteil 'B' werden Sie dazu aufgefordert, eine Zeichenkette, nach der die Datei "ML" durchsucht werden soll, einzutasten. Gleich nach der Eingabe werden die ersten 6 Zeichen der Kette nach R_{00} gebracht, damit später weitere Suchvorgänge nach ihnen ausgelöst werden können. In der von 'LBL 08' angeführten Befehlsfolge wird zunächst die Suche durchgeführt (Zeile 92). Ist der Eintrag gefunden, wird seine Nummer angezeigt/ausgedruckt (Zeilen 95 – 101). Danach muß der Dateizeiger auf den Anfang des Fundstückes gestellt werden (Zeile 102 – 103), damit der Modul '95' bemüht werden kann, die aufgestöberte Adresse vorzulegen. Anschließend wird der nachfolgende Dateibereich nach den nur mehr ersten 6 Zeichen der Kette durchforstet. Sobald sich das Dateiende meldet, gelangt – 1 nach X, und das Ergebnis des Tests auf Zeile 93 hält das Programm an.

Der Programmteil 'C' bedient sich des schon weiter oben beschriebenen Moduls '91', um den Dateizeiger auf den Anfang des ausgewählten Eintrages zu setzen und läuft dann in den Modul '95', der für das Vorweisen/Ausdrucken eines Eintrages zuständig ist, ein.

Der Programmteil 'c' verzweigt zunächst zur Marke '90', um eine Null für die Dateizeiger-Einstellung zu gewinnen und den Zeichenzähler auf den Anfangswert 7 zu setzen. Alsdann wird der Dateizeiger auf den Dateianfang eingestellt (Zeile 145). Die Zeilen 146 – 148 rechnen den im nächsten Schleifendurchlauf benötigten Dateizeiger-Wert aus. Der Modul '95' dient dazu, die 6 Zeilen einer Adresse anzuzeigen bzw. auszudrucken (Zeile 150). Solange sich das Dateiende nicht blicken läßt, wird zum Schleifenanfang ('LBL 10') rückverzweigt. Stößt der Befehl 'SEEKPT' (Zeile 145) jedoch ins Leere, 'verlischt' Flag 25, so daß ein Sprung auf Zeile 153 stattfindet: Der Wert des Zeichenzählers in X wird durch 7 dividiert, um die Anzahl der belegten Register zu ermitteln, und in den Zeilen 155 – 161 entsteht jene weiter oben als 'Bilanz' bezeichnete Meldung, welche diese Anzahl mit der Dateigröße vergleicht und dem Benutzer den Vergleich mitteilt.

Der Programmteil 'D' bedient sich zur Dateizeiger-Einstellung genauso des Moduls '91' wie 'C' und mündet dann in eine von 'LBL 09' angeführte Schleife, um die 6 Sätze des zum Tode verurteilten Eintrages umzubringen (6 mal 'DELREC', Zeile 135).

Der letzte Programmteil 'E' greift auch wieder zum Modul '91'. Der erste Satz des angesteuerten Eintrages wird angezeigt oder gedruckt (Zeile 186). Dann erscheint die Frage "OK?", und der Befehl 'GETKEY' nimmt seine 'Wartestellung' ein. Sobald eine Taste gedrückt wird oder die Wartezeit ohne Tastenbetätigung verstreicht, verursachen die Zeilen 190 – 192 einen indirekten Sprung auf die Marke, welche mit dem durch 'RDN' nach Register T gelangten Tastenkode übereinstimmt. Die Routinen '71' (Taste 'Y') und '84' (Taste 'R/S') bringen die nächste Zeile hervor. Die Routinen '00' (keine Taste gedrückt) und '04' (Taste 'ALPHA') setzen den Dateizeiger auf den Anfang des gegenwärtigen Satzes zurück und veranlassen, daß dessen Inhalt noch einmal angezeigt/gedruckt wird. Die Routinen '44' (Taste '←') und '01' (Taste 'ON') schließlich brechen den Korrekturlauf endgültig ab, so daß er nur mit 'XEQ E' von neuem aufgenommen werden kann. Jede andere als die genannten Tasten bringt durch

'GETKEY' einen Kode nach X, für den keine ansprungfähige Marke existiert. Weil aber Flag 25 zuvor gesetzt wurde (Zeile 182), wird 'GTO IND T' einfach mißachtet und so Zeile 193 erreicht. Dadurch kommt die Aufforderung "LINE?" zur Eingabe eines Ersatztextes, den das Programm in den Zeilen 199 – 200 gegen den alten Satzinhalt austauscht, zustande. Antwortet man auf "LINE?" nur mit 'R/S', besteht der Ersatztext aus dem in Zeile 195 ins Alpha-Register gebrachten Leerzeichen. Der durch 'GTO 84' veranlaßte Rücksprung bewirkt, daß der nächste Dateisatz zur Berichtigung angeboten wird.

Hinweis des Übers.: Beachten Sie das im Anhang C aufgeführte Buch 'Softwareentwicklung am Beispiel einer Dateiverwaltung (HP-41)' von M. Gehret. Das darin zur Verfügung gestellte Programmpaket geht weit über das, was "NAP" bietet, hinaus. Im Gegensatz allerdings zu K. Jarett, der seine Programm-Angebote stets in vorbildlich leserfreundlicher Weise mit ausführlichen Gebrauchsanweisungen, Analysen und Hinweisen auf Besonderheiten versieht, begnügt sich M. Gehret mit äußerst knappen und allgemein gehaltenen Modul-Beschreibungen, so daß der Leser harte Arbeit leisten muß, wenn er sich die vorgelegten Programme zunutze machen und ihre erstaunliche Leistungsfähigkeit erkennen will.

KAPITEL 8

Textverarbeitung auf dem HP-41

In diesem Kapitel werden Sie einen sogenannten Text-Editor, das ist ein Programm, mit dem man Texte aufnehmen, verändern und löschen kann, kennenlernen. Das Programm heißt "TE" und gestattet es, den Inhalt beliebiger Textdateien mit dem geringst möglichen Maß an 'Tastenarbeit' in die Anzeige zu holen und zu betrachten. Ebenso wie das im nächsten Kapitel 9 behandelte Programm "HP-16" verwendet es ausgiebig die Funktion 'GETKEY', wodurch ein bemerkenswert hoher Grad an Bequemlichkeit und Benutzerfreundlichkeit erreicht wird. Wenn Sie einen HP-41CX haben, steht Ihnen mit dessen X-Funktion 'ED' ein fest eingebauter Text-Editor zur Verfügung, der im wesentlichen dasselbe leistet wie "TE". Er ist "TE" bezüglich der Geschwindigkeit, mit der er auf Editionsbefehle antwortet, sogar überlegen. Andererseits hat aber auch "TE" einige Eigenschaften, die Sie bei 'ED' vergebens suchen; beispielsweise bessere Suchmöglichkeiten und die Fähigkeit, nichttastbare Sonderzeichen 'ohne Klimmzüge' in die Texte einzubringen. HP-41CX Besitzer sollten also nicht voreilig das Leistungsangebot von "TE" verschmähen. Das Programm "TE" als auch die zugehörige Bedienungsanleitung wurden von Erik Christensen geschrieben und werden hier mit dessen Erlaubnis wiedergegeben.

"TE" (*T*ext-*E*ditor) ist ein Textbearbeitungsprogramm für einen HP-41C oder CV, der mit einen X-Funktionen-Modul ausgerüstet ist. Wahlfrei können ein oder zwei X-Memory-Module dazugesteckt werden. Das Programm hat einen Speicherplatzbedarf von 115 Registern und benötigt ein Datenregister. Es versieht Sie mit höchst bequemen Möglichkeiten, Texte in ASCII-Dateien des X-Memorys aufzunehmen und dann zu betrachten, zu ergänzen, zu verkürzen, zu ändern oder zu löschen. Es gibt dafür insbesondere einen 'Edit-Modus', in dem das Tastenfeld ganz im Dienste der Dateibearbeitung steht. In diesem Edit-Modus löst jeder Tastendruck eine ganz bestimmte die Textbearbeitung betreffende Funktion aus. Sie werden dabei zu Antworten, die das Programm verarbeitet, aufgefordert. Die Dateiinhalte können Sie sich durch ein 12 Zeichen enthaltendes 'Fenster', welches sich vermittels verschiedener Tasten über die gesamte Datei hinweg verschieben läßt, ansehen. Auf Fehlbedienungen werden Sie aufmerksam gemacht, ohne daß das Programm deswegen anhält. Auf den folgenden Seiten wird die in "TE" gültige Bedeutung der lokalen Marken 'A' bis 'H' beschrieben. Der Marke 'E' widmen wir uns dabei ganz zum Schluß, weil Sie den Zugang zum Edit-Modus, der den umfangreichsten Teil des Programms bildet, verschafft.

Genau wie bei "NAP" im vorangegangenen Kapitel 7 müssen Sie die Tasten, welche für "TE" im USER-Modus ansprechbar sein sollen, also die Tasten 'A' bis 'H', von möglicherweise vorhandenen Belegungen befreien; am besten mit dem im Abschnitt 10G angebotenen

und dort ausführlich beschriebenen Programm "SK". Haben Sie dies getan, können Sie den Text-Editor "TE" aufrufen. Sie finden dann auf den obersten beiden Tastenreihen die folgenden Programmfunktionen vor:

Zeile 1:

'A'	'B'	'C'	'D'	'E'
ADD = Datei anlegen	FRE = Anzahl freier Bytes?	CLR = Datei-inhalt löschen	DEL = Datei tilgen	ED = Edit-Modus

Zeile 2:

'F'	'G'	'H'
DIR = Datei-Verzeichnis	GTO = Datei auswählen	Tasten-bedeutung erklären

Die Leistung dieser Funktionen soll nun im einzelnen erklärt werden:

'A': ADD = neue Datei im X-Memory anlegen

Diese Routine versetzt Sie in die Lage, eine Datei programmgesteuert im X-Memory anzulegen. Sie werden zunächst mit der Frage "LINES?" dazu aufgefordert, die Anzahl der zu erwartenden Zeilen der Textdatei einzugeben. Haben Sie diese eingetastet, müssen Sie mit 'R/S' fortfahren. Alsdann befragt Sie die Routine mit "CHAR?" nach der Gesamtanzahl der in die Datei aufzunehmenden Zeichen. Tasten Sie auch diese Zahl ein, und drücken Sie 'R/S'. Schließlich erkundigt sich die Routine noch mit "NAME?" danach, wie denn die neue Datei heißen soll. Darauf antwortet man mit der Eingabe eines Namens aus höchstens 7 Zeichen und dem üblichen Druck auf 'R/S'. Sollte der eingegebene Name schon für eine andere Datei vergeben worden sein, werden Sie erneut mit "NAME?" zur 'Taufe' aufgefordert – übrigens immer wieder, falls Sie beharrlich auf schon besetzte Namen zurückgreifen; die Routine 'gibt in keinem Fall nach'. *) Läßt sich Ihre Forderung nach Platz für die neue Datei vom X-Memory gegenwärtig nicht erfüllen, werden Sie – wieder mit "LINES?" beginnend – aufgefordert, sich neu zu entscheiden, wobei Sie jetzt natürlich genügsamer sein müssen. Ist an Ihren Angaben nichts mehr auszusetzen, tut die Routine die ihr zugewiesene Pflicht: sie erzeugt die gewünschte Datei und hält mit einem "OK" in der Anzeige an, um zu bekunden, daß Sie nunmehr alle lokalen Alpha-Marken von "TE" vereinbarungsgemäß ansprechen können.

'B': FRE = Anzahl der freien Bytes ermitteln

Mit dieser Routine läßt sich feststellen, wieviele Zeichen einer Textdatei noch zugefügt werden können. Mit "FILE NAME?" fragt die Routine nach der Datei, deren restliches Fassungsvermögen ermittelt werden soll. Tasten Sie also deren Namen ein, und drücken Sie 'R/S'. Existiert keine Textdatei mit dem angegebenen Namen, oder ist dieser Name der einer

*) Anm. d. Übers.: Damit der Stapel durch die irrtümliche Angabe existierender Dateien nicht mit falschen Zahlen verunreinigt wird (Zeile 243), müssen Sie hinter Zeile 236 noch ein 'ENTER↑' und hinter Zeile 243 noch ein 'RDN' einschieben.

Programm- oder Datendatei, wiederholt die Routine geduldig ihre Anfrage. Sollte sich Ihre Wißbegier auf die gegenwärtige Datei beziehen, brauchen Sie überhaupt keinen Dateinamen einzutasten. In diesem Fall reicht ein bloßes 'R/S'. Die Routine rechnet dann die Anzahl der in der Textdatei noch freien Bytes aus und teilt das Ergebnis dem Benutzer in der Form "CHAR LEFT = n", in welcher n eben diese Anzahl bezeichnet, mit. Auf ein erneutes 'R/S' hin sehen Sie wieder das schon bekannte "OK", welches stets bedeutet: Wählen Sie eine der Routinen 'A' bis 'H'.**)

'C': CLR = Inhalt einer Datei löschen

Diese Routine löscht den gesamten in einer ASCII-Datei abgelegten Text, doch läßt sie die Datei selbst unangetastet. Diese belegt also auch nach dem Wirken von 'C' den ihr zugewiesenen Platz im X-Memory, und ihr Name ist nach wie vor im Datei-Verzeichnis des X-Memorys auffindbar. 'C' fragt mit "FILE NAME?" nach dem Namen der zu löschenden Datei. Wie üblich müssen Sie diesen eintasten und 'R/S' drücken. Im Gegensatz zu 'B' aber ist 'C' nie mit einem bloßen 'R/S' zufrieden, und zwar deswegen nicht, weil der Befehl 'CLFL' (Zeile 286) nicht automatisch bezüglich der gegenwärtigen Datei arbeitet, wenn das Alpha-Register leer ist (vgl. Abschnitte 2E und 3D). Durch diese Eigenschaft ist ein zusätzlicher Schutz gegen das zufällige Löschen der Arbeitsdatei gegeben. Bei falschem Namen (Datei nicht vorhanden, oder Name zu einer Programm- oder Datendatei gehörig) werden Sie erneut zur Benennung der Datei eingeladen. Nach Vollzug des Löschungsauftrages erscheint das vertraute "OK".

'D': DEL = Datei im X-Memory auflösen

Diese Routine beseitigt die Textdatei, welche Sie auf die Aufforderung "FILE NAME?" hin benennen, vollständig im X-Memory, sobald Sie 'R/S' drücken. Man bedient sich ihrer, wenn Platz für neue Dateien durch Tilgung überflüssiger oder veralteter Dateien geschaffen werden soll. Der eingegebene Name wird mißachtet, wenn unter ihm keine Textdatei abgelegt ist oder wenn er zu einer Programm- oder Datendatei gehört. Die Routine schließt mit dem Ihnen geläufigen "OK" ab.

'F': DIR = Datei-Verzeichnis des X-Memorys durchsehen

Diese Routine ruft die X-Funktion 'EMDIR' auf. Infolgedessen werden sämtliche im Verzeichnis des X-Memorys enthaltenen Einträge in der Ihnen bekannten Gestalt (Dateiname, Dateiart, belegte Register) der Reihe nach in die Anzeige geholt, Programm- und Datendateien eingeschlossen. Nach Abschluß der Durchsicht hält die Routine mit der Mitteilung "FREE = n" an und gibt damit die Anzahl n der Bytes im X-Memory, die noch für weitere Dateien zur Verfügung stehen, bekannt. Wenn Sie zum "OK" zurückkehren wollen, müssen Sie noch einmal 'R/S' drücken.

'G': GTO = Datei zur Arbeitsdatei machen

Mit dieser Routine können Sie eine beliebige Datei zur Arbeitsdatei machen. Die Auswahl treffen Sie wie üblich mit dem Eintasten des Dateinamens als Antwort auf "FILE NAME?" und

*) Anm. d. Übers.: Auf dem HP-41CX lassen sich die Programmzeilen 260-272 insgesamt durch den Einzelbefehl 'ASROOM' ersetzen. Das beschleunigt bei vollen Dateien die Routine 'B' erheblich.

dem anschließenden 'R/S'. Eine Datei, die mit 'G' zur Arbeitsdatei bestimmt worden ist, braucht in den Routinen 'B' und 'E' nicht erneut benannt zu werden. Dort reicht dann ein einfaches 'R/S' als Antwort auf "FILE NAME?". Auch 'G' arbeitet genau wie die anderen "TE"-Routinen nur dann widerspruchslos, wenn der eingetastete Name zu einer vorhandenen Datei gehört und diese überdies eine Textdatei ist. Andernfalls wird die Frage "FILE NAME?" erneut vorgelegt. Hat die Routine Ihre Auswahl gebilligt und vorgenommen, kehrt das "OK" in die Anzeige zurück.

'H': Bedeutung der lokalen Alpha-Marken bloßlegen

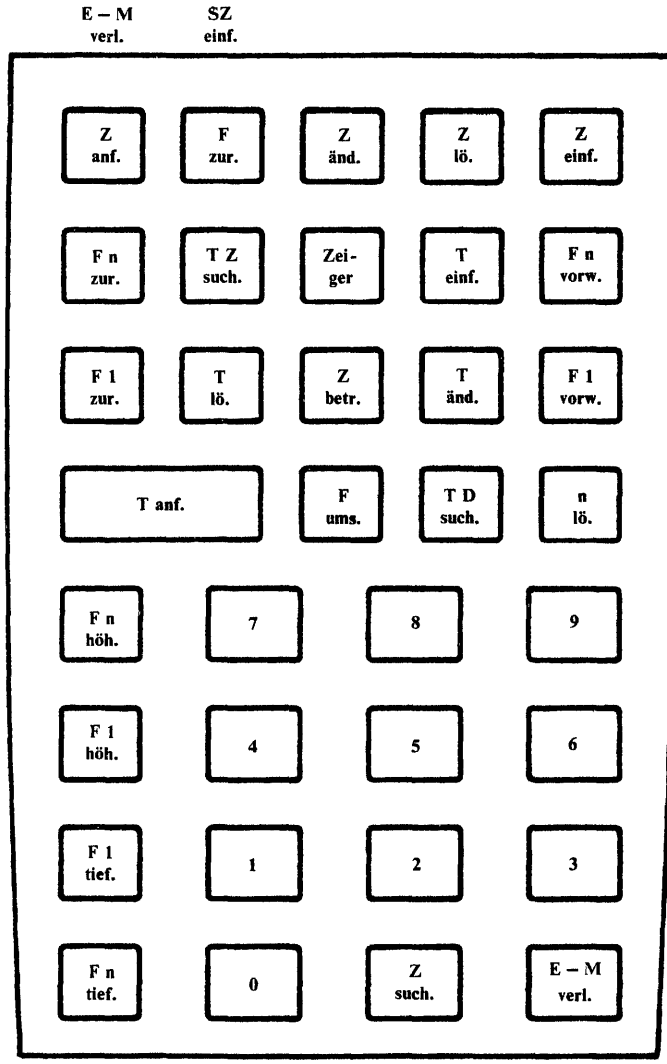
Diese Routine dient allein dazu, das Erinnerungsvermögen des "TE"-Benutzers aufzufrischen. Sie erlaubt es nämlich, die Bedeutung der lokalen Marken 'A' bis 'G' mit Hilfe eines mnemotechnischen Kürzels ins Gedächtnis zurückzurufen. Zu diesem Zweck werden Sie nach Betätigung der Taste 'H' mit der Frage "PROBLEM KEY?" dazu ermuntert, genau die Taste aus der Gruppe 'A' bis 'G', über deren Funktion innerhalb von "TE" Sie sich im unklaren sind, zu drücken. Als Antwort wird für einen flüchtigen Augenblick jenes zur gedrückten Taste gehörende Kürzel, das Sie schon aus der weiter oben gegebenen USER-Tasten-Übersicht kennen, zutage gefördert: 'A' ≙ ADD, 'B' ≙ FRE, 'C' ≙ CLR, 'D' ≙ DEL, 'E' ≙ ED, 'F' ≙ DIR und 'G' ≙ GTO. Unmittelbar nach der kurzzeitigen Enthüllung der Tasten-Bedeutung wird wieder die Frage "PROBLEM KEY?" vorgelegt, und zwar stets von neuem, solange Sie lediglich eine der Tasten 'A' bis 'G' drücken oder gar den HP-41 vollständig vernachlässigen und unbeachtet sich selbst überlassen. Betätigen Sie eine andere als die befragungsfähigen Tasten, hält die Routine mit dem wohlbekannten "OK" in der Anzeige an.

'E': ED = Textdatei bearbeiten

Wenn Sie die Taste 'E' drücken, begibt sich der HP-41 in den vom Programm "TE" erzeugten Edit-Modus. In diesem Modus wird der eigentliche Editor wirksam, indem sich das gesamte Tastenfeld in ein für die vorgesehene Textverarbeitung geeignetes 'Bedienungspult' verwandelt. Sämtliche zum Editor gehörigen Befehle werden 'fliegend', also bei laufendem Programm, entgegengenommen. Eine Ausnahme bilden die Alpha-Eingaben, die der Rechner 'stehend' annimmt und die man mit 'R/S' zur Verarbeitung abschicken muß.

Auf 'E' hin wird Ihnen zunächst einmal die bekannte Frage "FILE NAME?" vorgelegt. Sie müssen also den Namen der Datei, die bearbeitet werden soll, eintasten und 'R/S' drücken. Geben Sie den Namen einer Programm- oder Datendatei ein, wird Ihnen die Anfrage erneut vorgelegt. Haben Sie die Datei schon vorher mit 'G' ausgewählt, reicht ein bloßes 'R/S', um sie anzusteuern. Nach dem 'R/S' wird die Anzeige zu einem Fenster, durch das Sie in die Datei 'hineinblicken'. Das Fenster, welches den Blick auf bis zu 12 Zeichen freigibt, läßt sich mit verschiedenen Befehlen über die gesamte Datei hinweg verschieben. Anfangs liegt es genau über dem ersten Satz, dessen erstes Zeichen linksbündig enthaltend. Zugleich erkennen Sie, daß Flag 0 gesetzt ist. Dadurch wird signalisiert, daß das Tastenfeld nunmehr Ihren Edit-Befehlen 'entgegenfiebert', die Tasten sich also, wenn sie jetzt gedrückt werden, gemäß ihrer Bedeutung im Edit-Modus verhalten. Sofern für die Ausführung einer Edit-Funktion Eingaben nötig sind, werden Sie dazu zuverlässig aufgefordert. Anschließend wird die Anzeige wieder zum Datei-Fenster, und das Tastenfeld harrt erneut Ihrer Edit-Befehle. Möchten Sie aus dem Edit-Modus aussteigen, müssen Sie 'ON' oder 'R/S', stets bei gesetztem Flag 0, drücken. Am "OK" erkennen Sie dann, daß Sie auf die eingangs beschriebene Programmebene, die USER-Tasten-Steuerung von "TE", zurückgekehrt sind.

Die nachstehende Abbildung zeigt das Tastenfeld des HP-41 als Steuerpult des Editors von "TE":



Beachten Sie, daß Tastenzuweisungen im Edit-Modus zwar mißachtet aber nicht etwa 'ums Leben gebracht' werden. Sobald Sie den Edit-Modus verlassen, finden Sie Ihre Zuweisungen wieder vor.

Im folgenden sollen die einzelnen Edit-Funktionen von "TE" unter Bezug auf die im obigen Tastenfeld verwendeten Abkürzungen der Reihe nach ausführlich beschrieben werden, so daß Sie sich Ihrer sicher bedienen können.

'Z anf.': neue *Z*eile *anf*ügen

Die Taste 'Σ +' fordert Sie mit "NEW TEXT?" dazu auf, eine Zeile von bis zu 24 Zeichen einzutasten und mit 'R/S' zur Verarbeitung abzuschicken. Aus Ihrer Eingabe entsteht ein neuer Satz, welcher der Datei als letzter Satz angefügt wird. Anschließend zeigt sich der neue Satz im Fenster; falls er mehr als 12 Zeichen lang ist natürlich nur soweit, als er hineinpaßt. Sollte für den neuen Satz nicht mehr genug Platz in der Datei sein, erscheint die Meldung "ERROR", und der Edit-Befehl wird nicht ausgeführt.

Das Wort 'Zeile' ist von jetzt an stets im Sinne von 'Satz' (vgl. Abschnitt 3A) zu verstehen. Eine Zeile kann also bis zu 254 Zeichen enthalten.

'F zur.': *F*enster auf Zeilenanfang *zur*ücksetzen

Mit der Taste '1/x' können Sie das Fenster auf die ersten 12 Zeichen der gegenwärtigen Zeile zurücksetzen. Befindet sich das Fenster schon dort, geschieht nichts. Die Routine erfordert keine Eingaben, sondern führt den Auftrag sofort aus.

'Z änd.': Inhalt einer *Z*eile *änd*ern

Mit der Taste '$\sqrt{x}$' können Sie den Inhalt einer Zeile ändern. Der alte Text wird vollständig durch den neuen Text ersetzt. Zur Eingabe des Textes, der den gegenwärtigen Text überschreiben soll, werden Sie mit "NEW TEXT?" aufgefordert. Die Eingabe kann wie üblich aus bis zu 24 Zeichen bestehen und wird mit 'R/S' abgeschickt. Falls für den neuen Text nicht mehr genügend Platz vorhanden ist, erscheint die Meldung "ERROR".*) Der alte Text ist zu diesem Zeitpunkt aber schon gelöscht, so daß der neue Text jetzt nur noch mit dem Edit-Befehl 'Z einf.' (Taste 'LN', siehe weiter unten) in die Datei geschrieben werden kann. Sie erkennen diese Tatsache daran, daß nach der "ERROR"-Meldung nicht der Austauschtext im Fenster erscheint, was der Fall sein muß, wenn er Ihrer Absicht entsprechend den alten ersetzt hat, sondern die Folgezeile (bzw. die vorangehende Zeile, falls die Zeile, welche weichen mußte, die letzte war) dort auftaucht.

'Z lö.': *Z*eile vollständig *lö*schen

Wenn Sie die Taste 'LOG' drücken, wird die Zeile, welche gerade im Fenster steht, voll-

*) Anm. d. Übers.: Damit Sie diese Meldung wirklich erhalten, muß das Programm noch um vier Zeilen ergänzt werden: hinter Zeile 41 'FC? 25, SF 09' einfügen, hinter Zeile 363 'CF 09' einfügen, hinter Zeile 393 'FC?C 09' einfügen.

ständig gelöscht. Anschließend kommt die Folgezeile ins Fenster. War die zu löschende Zeile die letzte, gibt es keine Folgezeile, so daß stattdessen die vorangehende Zeile erscheint. *) Die Routine erfordert keine Eingabe, sondern führt den Löschauftrag sofort aus.

'Z. einf.': *Z*eile *ein*fügen

Wenn Sie die Taste 'LN' drücken, werden Sie mit "NEW TEXT?" zur Eingabe einer neuen Zeile aufgefordert. Sobald Sie 'R/S' drücken, wird die Zeile unmittelbar vor der Zeile, die zuvor im Fenster zu sehen war, eingefügt. Anschließend wird das Fenster über den Anfang der neuen Zeile geschoben. Sollte nicht mehr genügend Platz in der Datei vorhanden sein, um die neue Zeile aufzunehmen, bekommen Sie die "ERROR"-Meldung. Die Routine beläßt in diesem Fall das Fenster über der ursprünglichen Zeile.

'F n zur.': *F*enster um *n* Zeichen *zur*ücksetzen

Mit der Taste 'x ≶ y' können Sie das Fenster um n Zeichen in der gegenwärtigen Zeile zurücksetzen (nach links setzen), wobei Sie dafür zur Eingabe der Zahl n mit der Frage "NUMBER?" aufgefordert werden. Die Eingabe muß 3-ziffrig erfolgen, erfordert also im allgemeinen die Verwendung führender Nullen: z.B. 007 für 7. Die Flags 1, 2 und 3 dienen dabei als Eingabe-Hilfe: jedes eingeschaltete Flag bedeutet, daß die entsprechende Ziffer (1. Ziffer, 2. Ziffer, 3. Ziffer) eingegeben werden soll. Ist dies geschehen, läuft das Programm selbständig weiter; ein 'R/S' ist nach dem Eintasten der dritten Ziffer nicht mehr nötig. Bevor die Verschiebung des Fensters stattfindet, erscheint noch kurz die eingegebene 3-ziffrige Zahl in der Anzeige. Jeder Versuch, das Fenster über den Anfang der gegenwärtigen Zeile hinaus zu verschieben, wird mit der Meldung "ERROR" zurückgewiesen (außer bei der ersten Zeile der Datei – d. Übers.).

'T Z such.': *T*ext in gegenwärtiger *Z*eile *such*en

Mit dieser Routine können Sie in der gegenwärtigen Zeile nach einem Textstück suchen. Ist das Textstück vorhanden, wird das Fenster auf dessen erstes Zeichen geschoben. Auf 'R↓' hin werden Sie mit "TARGET TEXT?" zur Eingabe des Zieltextes aufgefordert. Nach dem üblichen 'R/S' beginnt die Suche. Es wird lediglich in der gegenwärtigen Zeile gesucht; falls das Fenster über dieser Zeile nach rechts verschoben steht, sogar nur in dem dadurch bestimmten Reststück der Zeile. Bei erfolgreicher Suche wird das Fenster an den Anfang des Zieltextes geschoben, bei erfolgloser Bemühung dagegen auf den Zeilenanfang, auch wenn dieser zu Beginn der Suche links außerhalb des Fensters lag.

'Zeiger': gegenwärtigen *Zeiger*wert ermitteln

Nach einem Druck auf die Taste 'SIN' wird der gegenwärtige Dateizeigerwert kurzfristig in der Form "LINEnCHARm" angezeigt. Der Befehl wird ohne irgendeine Eingabe-Aufforderung sofort ausgeführt. Beachten Sie bei der Entgegennahme der Meldung, daß die n-te Zeile einer Datei die Nummer n – 1 trägt und das m-te Zeichen einer Zeile die Nummer m – 1.

*) Anm. d. Übers.: Auch hier ist eine kleine Reparatur des Programms nötig: hinter Zeile 219 muß noch 'SF 25' eingefügt werden, damit keine "ERROR"-Meldung erscheint, wenn die zu löschende Zeile die letzte ist.

'T einf'.: *T*ext in gegenwärtige Zeile *einf*ügen

Wenn Sie die Taste 'COS' drücken, werden Sie mit "NEW TEXT?" dazu aufgefordert einen Text einzugeben. Sobald Sie 'R/S' drücken, gelangt dieser Text in die gegenwärtige Zeile, und zwar unmittelbar vor das Zeichen, welches gerade linksbündig im Fenster steht. Das Fenster wird dabei so verschoben, daß es anschließend über dem Anfang des neu eingefügten Textes steht. Sollte für das einzuschiebende Textstück nicht mehr genügend Platz in der Datei zur Verfügung stehen, wird "ERROR" gemeldet, und das Fenster verharrt in seiner Ausgangslage.

'F n vorw.': *F*enster um *n* Zeichen *vor*wärts setzen

So wie Sie mit 'x≦y' das Fenster um n Zeichen zurücksetzen können, läßt sich das mit 'TAN' in umgekehrter Richtung (nach rechts) bewerkstelligen. Die unter 'F n zur.' gegebenen Erläuterungen gelten hier in gleicher Weise. Der Versuch, das Fenster über das letzte Zeichen einer Zeile hinweg zu verschieben, führt zur Meldung "ERROR", und der Fensterinhalt bleibt der alte.

'F 1 zur.': *F*enster um *1* Zeichen *zur*ücksetzen

Mit der 'goldenen' Taste wird das Fenster um 1 Zeichen zurückgesetzt (nach links gesetzt). Eine Eingabe-Aufforderung erfolgt hierfür nicht. Der Befehl wird sofort ausgeführt. Der Versuch, das Fenster über das erste Zeichen einer Zeile hinweg zurückzusetzen, führt – außer bei der ersten Zeile – zur "ERROR"-Meldung.

'T lö.': *T*extstück in gegenwärtiger Zeile *lö*schen

Wenn Sie 'XEQ' drücken, werden Sie mit "OLD TEXT?" dazu aufgefordert, ein Textstück, das in der gegenwärtigen Zeile gelöscht werden soll, einzugeben. Der Befehl zur Löschung wird mit 'R/S' ausgelöst. Die Suche nach dem Zieltext beginnt bei dem Zeichen, das gerade linksbündig im Fenster steht. Das Fenster wird anschließend auf das dem gelöschten Textstück unmittelbar folgende Zeichen verschoben. Bildet das zu löschende Textstück den Schluß der Zeile, wird das Fenster auf den Zeilenanfang zurückgesetzt. Wird die eingegebene Zeichenkette überhaupt nicht gefunden, geschieht gar nichts.

'Z betr.': *Z*eile vollständig *bet*rachten

Wenn Sie sich eine Zeile ganz ansehen wollen, müssen Sie 'STO' drücken. In Teilstücken zu je 12 Zeichen erscheint die gegenwärtige Zeile dann in der Anzeige, und zwar unabhängig von der Lage, die das Fenster gerade einnimmt. Zuvor wird noch kurz die Nummer der gegenwärtigen Zeile mit "LINE n" gemeldet. Anschließend ist die Fensterdurchsicht wieder dieselbe, die vor dem 'STO' gültig war. Bei angeschlossenem Drucker wird die Zeile außerdem gedruckt, und zwar ebenfalls in Teilstücken zu je 12 Zeichen.

'T änd.': *T*ext in gegenwärtiger Zeile *änd*ern

Mit dieser Routine haben Sie ein besonders wirksames Werkzeug zur Textänderung in der Hand. Auf 'RCL' hin werden Sie mit "OLD TEXT?" dazu aufgefordert, ein Textstück einzugeben, das in der gegenwärtigen Zeile gelöscht werden soll. Sobald dies, ausgelöst durch 'R/S', geschehen ist, bekommen Sie die Anschlußfrage "NEW TEXT?" vorgelegt. Daraufhin tasten Sie ein neues Textstück ein und schicken auch dieses mit 'R/S' ab. Er gelangt an die Stelle des alten Textstückes, und das Fenster wird so verschoben, daß darin linksbündig das neue Textstück zu sehen ist. Altes und neues Textstück dürfen verschiedene Länge haben. Wenn das neue Textstück länger als das alte ist und der Platz dafür nicht mehr ausreicht, bekommen Sie die "ERROR"-Meldung. Die Löschung des alten Textstückes wird nichtsdestoweniger vorgenommen, wovon Sie sich durch Verschieben des Fensters leicht überzeugen können.*)

'F 1 vorw.': *F*enster um *1* Zeichen *vorw*ärts setzen

Mit der Taste 'SST' wird das Fenster um 1 Zeichen vorgesetzt (nach rechts gesetzt). Eine Eingabe-Aufforderung erfolgt nicht. Vgl. 'F 1 zur.'. Der Versuch, das Fenster über das letzte Zeichen einer Zeile hinweg zu verschieben, führt zur Meldung "ERROR".

'T anf.': *T*ext der gegenwärtigen Zeile *anf*ügen

Mit dieser Routine können Sie der gegenwärtig im Fenster stehenden Zeile Text anfügen, und zwar unabhängig von der gerade eingestellten Fensterlage. Auf 'ENTER↑' hin werden Sie mit "NEW TEXT?" zum Eintasten des anzufügenden Textes aufgefordert. Danach müssen Sie 'R/S' drücken. Wenn der neue Text keine Aufnahme mehr finden kann, werden Sie mit "ERROR" darauf aufmerksam gemacht. Die Lage des Fensters bleibt in jedem Fall die alte.

'F ums.': *F*enster in gegenwärtiger Zeile *ums*etzen

Wenn Sie 'CHS' drücken, werden Sie mit "NUMBER?" zur Eingabe einer 3-ziffrigen Zahl (vgl. 'F n zur.') aufgefordert. Sobald Sie diese eingetastet haben – ein abschickendes 'R/S' ist nicht nötig –, wird das Fenster auf das durch diese Zahl bestimmte Zeichen der gegenwärtigen Zeile umgesetzt, wobei Sie berücksichtigen müssen, daß das n-te Zeichen mit der Zahl n – 1 erreicht wird. Dem Eintasten unzulässiger Zahlen wird mit der "ERROR"-Meldung widersprochen.

'T D such.': *T*ext in *D*atei *such*en

Im Gegensatz zur Routine 'T Z such.' (Taste 'R↓'), die nur bezüglich der gegenwärtigen Zeile arbeitet, können Sie mit dieser Routine die gesamte Datei nach einem bestimmten Text durchsuchen. Auf 'EEX' hin werden Sie mit "TARGET TEXT?" zur Eingabe des gesuchten

*) Anm. d. Übers.: Die Routine ist nicht völlig gegen Fehlbedienung abgesichert. Wenn man z.B. als 'altes Textstück' *die ganze gegenwärtige Zeile* eingibt, verschmilzt das anschließend eingetastete 'neue Textstück' mit der folgenden Zeile. Für die Fälle vollständiger Zeilenänderung sollte man also mit 'Z änd.' (s.o.) arbeiten. Desgleichen gerät die Dateizeigersteuerung in Unordnung, wenn man als 'altes Textstück' *das Ende der gegenwärtigen Zeile* eingibt und dann kein 'neues Textstück' mehr eintastet, sondern die Aufforderung "NEW TEXT?" mit einem bloßen 'R/S' beantwortet, oder wenn das 'neue Textstück' so lang ist, daß es keinen Platz mehr findet.

Textes aufgefordert. Die Suche wird mit 'R/S' ausgelöst und beginnt grundsätzlich am Dateianfang. Wiederholt auftretende Zieltexte werden also nicht ausfindig gemacht. Bei erfolgreicher Suche wird das Fenster auf die Fundstelle gesetzt, andernfalls bleibt seine Lage unverändert.

'n lö.': *n* Zeichen *lö*schen

Mit dieser Routine können Sie n Zeichen in der gegenwärtigen Zeile, beginnend bei dem linksbündig im Fenster stehenden Zeichen, löschen. Auf '←' hin werden Sie in der schon bekannten Weise (vgl. 'F n zur.') zur Eingabe einer 3-ziffrigen Zahl eingeladen. Sobald die Eingabe beendet ist – denken Sie dabei an die gewöhnlich zu berücksichtigenden führenden Nullen –, beginnt die Löschung. Diese erfaßt den gesamten Zeilenrest, wenn die eingetastete Zahl größer als die Anzahl der Zeichen, deren erstes linksbündig im Fenster steht, ist. Das Fenster wird auf das erste dem gelöschten Teilstück folgende Zeichen verschoben. Ist der ganze Zeilenrest von der Löschung betroffen, gelangt der Zeilenanfang ins Fenster.

'F n höh.': *F*enster um *n* Zeilen *höh*er setzen

Wenn Sie die Minustaste ' – ' drücken, erhalten Sie die Ihnen schon aus anderen Routinen bekannte Aufforderung "NUMBER?". Ihre aus 3 Ziffern bestehende Antwort wird hier dazu verwendet, das Fenster um entsprechend viele Zeilen in der Datei nach oben zu versetzen. Würden Sie mit der eingetasteten Zahl über den Dateianfang hinausgelangen, wird das Fenster auf die erste Zeile versetzt.

'F 1 höh.': *F*enster um *1* Zeile *höh*er setzen

Mit der Plustaste ' + ' setzen Sie das Fenster ohne zusätzliche Eingabe um genau eine Zeile höher. Befinden Sie sich dabei auf der obersten Zeile, geschieht gar nichts.

'F 1 tief.': *F*enster um *1* Zeile *tief*er setzen

Mit der Taste ' × ' setzen Sie das Fenster um genau eine Zeile tiefer. Befinden Sie sich bereits auf der letzten Zeile der Datei, bekommen Sie 'ERROR' gemeldet, und das Fenster verbleibt in seiner Lage.

'F n tief.': *F*enster um *n* Zeilen *tief*er setzen

Wenn Sie die Taste ' ÷ ' drücken, werden Sie mit "NUMBER?" zur Eingabe einer 3-ziffrigen Zahl aufgefordert. Das Fenster wird um entsprechend viele Zeilen tiefer gesetzt, es sei denn, Sie gelängen dabei über die letzte Zeile hinaus. In diesem Fall gibt es die "ERROR"-Meldung, und das Fenster bleibt unverrückt.

'Z such.': *Z*eile *such*en

Mit dieser Routine können Sie eine beliebige Zeile, deren Nummer Ihnen bekannt ist,

aufsuchen. Sobald Sie die Dezimalpunkttaste '.' drücken, bekommen Sie die Frage "NUMBER?" vorgelegt. Ihre 3-ziffrige Eingabe bewirkt hier, daß das Fenster auf die durch diese Zahl bestimmte Zeile gesetzt wird, wobei man die n-te Zeile mit der Zahl n – 1 erreicht. Der Versuch, eine nicht vorhandene Zeile auszusteuern, scheitert an der "ERROR"-Meldung; das Fenster bleibt dabei an Ort und Stelle.

'E-M verl.': *E*dit-*M*odus *verl*assen

Mit einem Druck auf eine der Tasten 'R/S' oder 'ON' – bei eingeschaltetem Flag 0 – können Sie den Edit-Modus verlassen. Sie erlangen dann wieder Zugriff auf jene weiter oben beschriebenen lokalen Marken 'A' bis 'H', was Sie an dem in der Anzeige auftauchenden "OK" erkennen können. Zugleich wird der Flagzustand, welcher beim Einlauf in den Edit-Modus gültig war, wiederhergestellt.

'SZ einf.': *S*onderzeichen *einf*ügen

Mit dieser Routine können Sie Ihrer Datei auch solche Zeichen umstandslos einfügen, die nicht auf herkömmliche Weise eintastbar sind. Auf 'USER' hin werden Sie mit "NUMBER?" zum Eintasten eines Zeichenkodes – das ist eine Dezimalzahl zwischen 0 und 255 aus der ASCII-Tabelle (vgl. Abschnitt 4B) – aufgefordert. Entsprechend der durchweg im Edit-Modus geltenden Bedienungslogik müssen Sie die Eingabe auch hier wieder 3-ziffrig, also u.U. unter Verwendung führender Nullen, vornehmen. Das Sonderzeichen gelangt unmittelbar vor das im Fenster linksbündig stehende Zeichen, und das Fenster wird um eine Stelle zurückgesetzt, so daß das eingefügte Zeichen sichtbar ist. Ist die Datei völlig ausgebucht, erscheint die übliche dies verkündende "ERROR"-Meldung, und Datei und Fenster bleiben unverändert.

Falls Sie die vorangegangenen Beschreibungen der "TE"-Funktionen im 'Trockenverfahren' durchgelesen haben und Ihnen deswegen einige Vorgänge nicht ganz klar geworden sind, sollten Sie sich eine kleine ASCII-Datei anlegen und darin mit den verschiedenen Funktionen des Editors probeweise arbeiten. Sie werden finden, daß die gegebenen Erläuterungen tatsächlich erschöpfend sind und Ihnen in "TE" ein Programm zur Verfügung steht, das so gut wie alles, was man von einem guten Editor fordert, leistet.

Persönliche Umgestaltung von "TE"

Obwohl "TE" kaum einen Wunsch in Bezug auf das Edieren von Textdateien offen läßt, mag es sein, daß der eine oder andere Benutzer dennoch einige Routinen abändern oder hinzufügen will. Leser, die diese Absicht nicht haben, können zum nächsten Kapitel übergehen. Die anderen müssen folgende Hinweise beachten:

1. Wählen Sie die Taste, welcher Ihre neue Routine zugewiesen werden soll, aus. Der zugehörige Kode läßt sich ermitteln, indem man 'GETKEY' von Hand ausführt und anschließend die ausgewählte Taste drückt. Der gesuchte Kode steht danach im X-Register.

2. Die lokale numerische Marke, die Ihrer neuen Routine als Ansprungstelle für den Befehl 'XEQ IND T' (Zeile 391) dient, muß mit dem in Schritt 1 ermittelten

Zeilen-/Spaltenkode übereinstimmen. Soll Ihre Routine beispielsweise bei Betätigung der Taste 'ENTER↑' ausgeführt werden, muß man ihr die Marke 'LBL 41' voranstellen.

3. Hinter die Marke, welche Ihrer Routine vorangeht, können Sie alles setzen, was Ihren Absichten entspricht. Die von Ihnen gewählte Befehlsfolge muß mit einem 'RTN' abgeschlossen werden, damit der Rücksprung auf Zeile 392 gesichert ist.

4. Ihre Routine unterliegt bezüglich einiger Flags und Register den folgenden Beschränkungen:

Flags/Register/ Anzeige-Modus	vor dem Einlauf in die Routine	beim Verlassen der Routine
$F_{00} - F_{03}$, F_{09}, F_{10}	gelöscht	gelöscht
F_{25}	gesetzt	"ERROR" wenn gelöscht
X	v.pqr	v.pqr
T	Tastenkode	frei benutzbar
ALPHA	Fensterinhalt	frei benutzbar
R_{00}	Flagzustand	gesperrt
F_{29}	gelöscht	gelöscht
'FIX'	0	0

Sobald Ihre Routine auf den Schlußbefehl 'RTN' stößt, muß X den für die Lage des Fensters verantwortlichen Dateizeigerwert 'v.pqr' enthalten. Hierbei ist 'v' die Satznummer und 'pqr' die Nummer des im Fenster linksbündig erscheinenden Zeichens (vgl. Abschnitt 3A). Ein in der Routine gelöschtes Flag 25 führt zur "ERROR"-Meldung und muß wieder gesetzt werden, wenn diese vermieden werden soll. Die Stapelregister Y, Z und L sind völlig frei verwendbar und bedürfen keines besonderen Inhaltes beim Verlassen der Routine. Auch T und ALPHA stehen nach dem Einlauf in die Routine zur freien Verfügung. Sofern Ihre Routine irgendwelchen Text in der Datei löscht, sollte sie mit 'GTO 26' statt mit 'RTN' enden. Bei 'LBL 26' beginnt nämlich eine Routine zur Fehlerbehandlung.

5. Wenn Sie die durch 'PROMPT'-Befehle ausgelösten Eingabe-Aufforderungen, die in "TE" enthalten sind, für Ihre Routine nutzen wollen, müssen Sie dies folgendermaßen tun:

Progr.-Befehl	Aufforderung	Eingabe ins Reg.	benutzte Reg.
'XEQ 06'	" TEXT?"	ALPHA	
'XEQ 07'	"NEW TEXT?"	ALPHA	
'XEQ 08'	"TARGET TEXT?"	ALPHA	
'XEQ 09'*)	"NUMBER?"	X	T,Z,Y,ALPHA

*) Anm. d. Übers.: Es lohnt sich für den Leser, die bemerkenswerte Routine '09' einmal genau zu analysieren. Synthetiker verschmelzen die Zeilen 132 – 136 selbstverständlich zu einer einzigen Textzeile.

Beispiel für eine zusätzliche Routine

Fügen Sie dem Programm "TE" eine Routine hinzu, mit der man in einer Textdatei *alle* Textstücke löschen kann, die mit einer Zeichenkette, welche der Benutzer einzugeben aufgefordert wird, übereinstimmen. Sehen Sie dafür die Taste 'ALPHA' als Bedienungstaste im Edit-Modus vor.

Lösung:

```
01♦LBL 04     zum Tastenkode von 'ALPHA' gehörige Marke
02 ENTER↑     'v.pqr' nach Y
03 XEQ 08     Aufruf der Aufforderung "TARGET TEXT?"
04♦LBL 88     Schleifenanfang
05 RDN        'v.pqr' nach X
06 POSFL      Suche bei Fensterausschnitt beginnen
07 X<0?       Textstück gefunden?
08 CLA        'CLA' falls nein ( ≙ – 1 in X)
09 RDN        'v.pqr' nach X
10 ALENG      Textstück-Länge ermitteln
11 DELCHR     entsprechend viele Zeichen löschen
12 X≠0?       ALPHA nicht leer?
13 GTO 88     Rücksprung auf Schleifenanfang, falls ja
14 RDN        'v.pqr' nach X
15 SF 25      Fehlerflag setzen
16 SEEKPT     Herstellung des alten Fensterausschnittes vorsehen
17 FS? 25   }
18 RTN      } Rücksprung auf Zeile 392, falls Wiederherstellung möglich
19 GTO 26     andernfalls Sprung zur Fehlerbehandlung
```

Sie können die vorstehende Befehlsfolge ganz an den Anfang von "TE" setzen: 'GTO "TE"' und 'RTN' von Hand im RUN-Modus ausführen, dann Übergang in den PRGM-Modus und Routine eintasten.

Jedesmal, wenn Sie jetzt die Taste 'ALPHA' im Edit-Modus drücken, bekommen Sie die Frage "TARGET TEXT?" vorgelegt, um eine Zeichenkette einzutasten. Auf das anschließende 'R/S' hin wird diese Kette überall in dem Dateiabschnitt, welcher beim gegenwärtigen Fensterausschnitt beginnt, gelöscht, so oft sie dort auch auftritt. Danach gelangt das Fenster in die bei Suchbeginn gültige Lage, es sei denn, linksbündig darin stand die zu löschende Kette. In diesem Fall wird der Ausschnitt auf den Zeilenanfang verschoben.

Liste der lokalen numerischen Marken

Marke	Zeile	Beschreibung/Funktion
'LBL 00'	386	Schleife für Tastenbedienung im Edit-Modus
'LBL 01'	221	Edit-Modus verlassen
'LBL 02'	120	Sonderzeichen einfügen
'LBL 06'	79	Aufforderung " TEXT?"
'LBL 07'	77	Aufforderung "NEW TEXT?"
'LBL 08'	74	Aufforderung "TARGET TEXT?"
'LBL 09'	129	Aufforderung "NUMBER?"
'LBL 10'	177	Schleife für 'LBL 33'
'LBL 11'	32	Zeile anfügen
'LBL 12'	205	Fenster auf Zeilenanfang zurücksetzen
'LBL 13'	38	Zeile ändern
'LBL 14'	161	Zeile löschen
'LBL 15'	165	Zeile einfügen
'LBL 16'	311	Aufforderung "FILE NAME?"
'LBL 17'	236	Aufforderung "NAME?" für 'LBL A'
'LBL 18'	368	Meldung "OK"
'LBL 19'	260	Schleife für 'LBL B'
'LBL 20'	272	Schleifenausgang für 'LBL B'
'LBL 21'	107	Fenster n Zeichen zurücksetzen
'LBL 22'	65	Text in Zeile suchen
'LBL 23'	190	Zeigerwert ermitteln
'LBL 24'	52	Text in Zeile einfügen
'LBL 25'	101	Fenster n Zeichen vorwärts setzen
'LBL 26'	208	Fehlerbehandlung bei Löschungen
'LBL 31'	01	Fenster 1 Zeichen zurücksetzen
'LBL 32'	43	Text in Zeile löschen
'LBL 33'	169	Zeile betrachten
'LBL 34'	27	Text in Zeile ändern
'LBL 35'	07	Fenster 1 Zeichen vorwärts setzen
'LBL 41'	23	Text der Zeile anfügen
'LBL 42'	87	Fenster umsetzen
'LBL 43'	56	Text in Datei suchen
'LBL 44'	90	n Zeichen löschen
'LBL 51'	94	Fenster n Zeilen höher setzen
'LBL 61'	16	Fenster 1 Zeile höher setzen
'LBL 71'	11	Fenster 1 Zeile tiefer setzen
'LBL 81'	115	Fenster n Zeilen tiefer setzen
'LBL 83'	128	Zeile suchen
'LBL 84'	222	Edit-Modus verlassen
'LBL 99'	374	Schleife für den 'Blick durchs Fenster'

Liste der Fehlerbedingungen

Funktion	Bedeutung der "ERROR"-Meldung
Z anf.	Datei läuft über
F zur.	keine Fehlermeldungen
Z änd.	Datei läuft über
Z lö.	keine Fehlermeldungen
Z einf.	Datei läuft über
F n zur.	angestrebte Fensterlage unzulässig *)
T Z such.	keine Fehlermeldungen
Zeiger	keine Fehlermeldungen
T einf.	Datei läuft über
F n vorw.	angestrebte Fensterlage unzulässig
F 1 zur.	erstes Zeichen steht bereits im Fenster *)
T lö.	keine Fehlermeldungen
Z betr.	keine Fehlermeldungen
T änd.	Datei läuft über
F 1 vorw.	letztes Zeichen steht bereits im Fenster
T anf.	Datei läuft über
F ums.	angestrebte Fensterlage unzulässig
T D such.	keine Fehlermeldungen
n lö.	keine Fehlermeldungen
F n höh.	keine Fehlermeldungen
F 1 höh.	keine Fehlermeldungen
F 1 tief.	letzte Zeile steht bereits im Fenster
F n tief.	angesteuerte Zeile nicht vorhanden
Z such.	angesteuerte Zeile nicht vorhanden
E-M verl.	keine Fehlermeldungen
SZ einf.	Datei voll *oder* unzulässiger ASCII-Kode

*) Anm. d. Übers.: Die Fehlermeldung wird bei der ersten Zeile unterdrückt.

Programmausdruck von "TE"
(Barcode im Anhang D)

```
01♦LBL 31
02 1 E-3
03 -
04 X<0?
05 CLX
06 RTN

07♦LBL 35
08 1 E-3
09 +
10 RTN

11♦LBL 71
12 INT
13 1
14 +
15 RTN

16♦LBL 61
17 INT
18 1
19 -
20 X<0?
21 CLX
22 RTN

23♦LBL 41
24 XEQ 07
25 APPCHR
26 RTN

27♦LBL 34
28 SF 10
29 XEQ 32
30 SF 25
31 GTO 24

32♦LBL 11
33 XEQ 07
34 APPREC
35 RCLPT
36 INT
37 RTN

38♦LBL 13
39 XEQ 07
40 DELREC
41 INSREC
42 GTO 26

43♦LBL 32
44 "OLD"
45 XEQ 06
46 POSFL
47 X<0?
48 CLA
49 ALENG
50 DELCHR
51 GTO 26

52♦LBL 24
53 XEQ 07
54 INSCHR
55 RTN

56♦LBL 43
57 0
58 SEEKPT
59 RDN
60 XEQ 08
61 POSFL
62 X<0?
63 RDN
64 RTN

65♦LBL 22
66 INT
67 XEQ 08
68 POSFL
69 INT
70 X<>Y
71 X=Y?
72 LASTX
73 RTN

74♦LBL 08
75 "TARGET"
76 GTO 06

77♦LBL 07
78 "NEW"

79♦LBL 06
80 "⊢ TEXT?"
81 AVIEW
82 CLA
83 AON
84 STOP
85 AOFF
86 RTN

87♦LBL 42
88 INT
89 GTO 25

90♦LBL 44
91 XEQ 09
92 DELCHR
93 GTO 26

94♦LBL 51
95 INT
96 XEQ 09
97 -
98 X<0?
99 CLX
100 RTN

101♦LBL 25
102 XEQ 09
103 1 E3
104 /
105 +
106 RTN

107♦LBL 21
108 XEQ 09
109 1 E3
110 /
111 -
112 X<0?
113 INT
114 RTN

115♦LBL 81
116 INT
117 XEQ 09
118 +
119 RTN

120♦LBL 02
121 XEQ 09
122 CLA
123 XTOA
124 FS? 25
125 INSCHR
126 RDN
127 RTN

128♦LBL 83
129♦LBL 09
130 "NUMBER?"
131 AVIEW
132 "RHIJ>?"
133 64
134 XTOA
135 RDN
136 "⊢456"
137 SF 01
138 GETKEY
139 SF 02
140 GETKEY
141 SF 03
142 GETKEY
143 CF 03
144 CF 02
145 CF 01
146 POSA
147 RDN
148 POSA
149 RDN
150 POSA
151 RDN
152 CLA
153 ARCL T
154 ARCL Z
155 ARCL Y
156 ANUM
157 X<0?
158 GTO 09
159 AVIEW
160 RTN

161♦LBL 14
162 DELREC
163 XEQ 26
164 RTN

165♦LBL 15
166 XEQ 07
167 INSREC
168 RTN

169♦LBL 33
170 FS? 55
171 SF 21
172 INT
173 "LINE "
174 ARCL X
175 AVIEW
176 SEEKPT

177♦LBL 10
178 CLA
179 ARCL 00
180 ARCL 00
181 ARCLREC
182 ASHF
183 ASHF
184 AVIEW
185 FS? 17
186 GTO 10
187 LASTX
188 CF 21
189 RTN

190♦LBL 23
191 ENTER↑
192 INT
193 "LINE"
194 ARCL X
195 "⊢CHAR"
196 LASTX
197 FRC
198 1 E3

199 *
200 ARCL X
201 AVIEW
202 RDN
203 RDN
204 RTN

205♦LBL 12
206 INT
207 RTN

208♦LBL 26
209 RCLPT
210 SF 25
211 SEEKPT
212 FC?C 10
213 FS? 25
214 RTN
215 INT
216 SF 25
217 SEEKPT
```

```
218 FC? 25
219 XEQ 61
220 RTN

221♦LBL 01
222♦LBL 84
223 RCL 00
224 STOFLAG
225 GTO 18

226♦LBL A
227 "LINES?"
228 PROMPT
229 "CHAR?"
230 PROMPT
231 +
232 7
233 /
234 1
235 +

236♦LBL 17
237 "NAME?"
238 AON
239 CLD
240 STOP
241 AOFF
242 SF 25
243 RCLPTA
244 FS? 25
245 GTO 17
246 SF 25
247 CRFLAS
248 FC? 25
249 GTO A
250 GTO 18
251♦LBL B
252 XEQ 16
253 ,
254 SEEKPT
255 FLSIZE
256 7
257 *
258 1
259 -

260♦LBL 19
261 SF 25
262 GETREC
263 FC?C 25
264 GTO 20
265 ALENG
266 -
267 FS? 17
268 GTO 19
269 1
270 -
271 GTO 19

272♦LBL 20
273 "CHAR LEFT="
274 RCLFLAG
275 FIX 0
276 CF 29
277 ARCL Y
278 STOFLAG
279 AVIEW
280 RDN
281 STOP
282 GTO 18

283♦LBL C
284 XEQ 16
285 SF 25
286 CLFL
287 FC? 25
288 GTO C
289 GTO 18
290♦LBL D
291 XEQ 16
292 SF 25
293 PURFL
294 FC? 25
295 GTO D
296 GTO 18

297♦LBL E
298 XEQ 16
299 RCLFLAG
300 STO 00

301 RDN
302 ,
303 SEEKPT
304 FIX 0
305 CF 29
306 CF 27
307 GTO 99

308♦LBL G
309 XEQ 16
310 GTO 18

311♦LBL 16
312 "FILE NAME?"
313 AON
314 AVIEW
315 CLA
316 STOP
317 AOFF
318 SF 25
319 RCLPTA
320 FC?C 25
321 GTO 16
322 SF 25
323 POSFL
324 FC?C 25
325 GTO 16
326 RTN

327♦LBL F
328 EMDIR
329 7
330 *
331 "FREE="
332 ARCL X
333 PROMPT
334 GTO 18

335♦LBL H
336 5
337 "PROBLEM KEY?"
338 AVIEW
339 "DIRED DELCLRFRE"
340 "+ADDGTO"
341 GETKEY
342 23
343 X<=Y?
344 GTO 18
345 RDN
346 10
347 -
348 X>Y?
349 +
350 10

351 MOD
352 -3
353 *
354 AROT
355 ASHF
356 ASHF
357 ASHF
358 AVIEW
359 GTO H
360♦LBL "TE"
361 SF 27
362 CF 21
363 CF 10
364 SIZE?
365 1
366 X>Y?
367 PSIZE
368♦LBL 18
369 CF 25
370 CLST
371 "OK"
372 PROMPT
373 GTO 18

374♦LBL 99
375 RCLPT
376 SF 25
377 CLA
378 ARCL 00
379 ARCL 00
380 ARCLREC
381 ASHF
382 ASHF
383 SEEKPT
384 AVIEW
385 SF 25

386♦LBL 00
387 SF 00
388 GETKEY
389 CF 00
390 RDN
391 XEQ IND T
392 SEEKPT
393 "ERROR"
394 FC? 25
395 AVIEW
396 GTO 99
397 END
```

803 Bytes

KAPITEL 9

Ein HP-16 Simulator

In diesem Kapitel stellen wir ein Programm vor, das einige der Funktionen des HP-16C simuliert. Weil mit diesem Programm u.a. die Basis von Zahlendarstellungen gewechselt werden kann, soll dazu einiges vorausgeschickt – statt vorausgesetzt – werden.

Für eine Dezimal*zahl* wxyz gilt

$$wxyz_{10} = w \cdot 10^3 + x \cdot 10^2 + y \cdot 10 + z,$$

wobei w, x, y und z irgendwelche *Ziffern* zwischen Null und Neun bedeuten. Der Index 10 weist darauf hin, daß der Wert der Zahl wxyz gemäß der obigen Gleichung zu verstehen ist. Man sagt dazu, die Zahl sei *zur Basis 10* dargestellt. Eine Zahl kann ebensogut hexadezimal, also zur Basis 16, dargestellt werden. Soll beispielsweise qrst eine Hexadezimal*zahl* sein, so ist ihr Wert in genauer Entsprechung zur obigen Formel gemäß der folgenden Gleichung zu bestimmen:

$$qrst_{16} = q \cdot 16^3 + r \cdot 16^2 + s \cdot 16 + t.$$

Auch hier bedeuten q, r, s und t wieder irgendwelche *Ziffern*, diesmal zwischen Null und Fünfzehn, und der Index 16 weist auf die Basis 16 hin. [Abschweifung des Übers.: Beachten Sie in diesem Zusammenhang ganz genau den Unterschied zwischen Zahl und Ziffer. Eine Ziffer ist lediglich ein *Zahlzeichen*, welches genau einer Zahl zugeordnet ist und diese eine Zahl und keine andere versinnbildlicht. Eine Zahl hingegen ist ein Begriff, von dem zwar jeder, der noch nie streng darüber nachgedacht hat, lächelnd meint, er wüßte genau, was darunter zu verstehen sei. Jedes Kind könne ja darüber Auskunft geben. Indessen ist es gerade die lange, schon in der Kindheit begonnene Übung im Umgang mit diesem alltäglichen Begriff, dem auszuweichen nicht einmal Naturvölkern gelingt, welche eine höchst trügerische Sicherheit verschafft und übel darüber hinweg täuscht, daß eine genaue sprachliche Festlegung dieses intuitiv so klar erfaßten Urbegriffes alles andere als einfach ist. Erst wer sauber erklären will, was eine Zahl ist, wird der Schwierigkeiten gewahr und muß gewöhnlich scheitern, wenn er ungeschult in Grundlagenfragen ist. Wir überlassen an dieser Stelle verunsicherte Leser, deren es hoffentlich einige gibt, ihrem plötzlich nagenden Zweifel und begnügen uns hier mit der unanfechtbaren Feststellung: Zahlen werden mit Hilfe von i.a. mehreren in bestimmter Ordnung hintereinander gereihten Ziffern *dargestellt*.] Da man im bürgerlichen Alltag nur im Dezimalsystem rechnet, also stets mit Zahlen, deren Zifferngestalt grundsätzlich bezüglich der Basis 10 gewählt wird, haben sich verständlicherweise für die Zahlen von 10 an aufwärts nie

eigene Zahlzeichen, wohlverstanden Ziffern also, entwickelt [Abschweifung II: für Zahl*bezeichnungen* – dies ist wieder ein anderer Begriff! – hingegen sehr wohl: Zwanzig, Dreißig, Hundert usw. sind zahleigene Bezeichnungen, entsprechende zahleigene Symbole gibt es aber nicht]. Daher bedient man sich hilfsweise der Anfangsbuchstaben des Alphabetes, sobald man über das Dezimalsystem hinausgeht: $A_{16} = 10_{10}$, $B_{16} = 11_{10}$, $C_{16} = 12_{10}$, $D_{16} = 13_{10}$, $E_{16} = 14_{10}$ und $F_{16} = 15_{10}$. Folglich gilt z.B.

$$C5_{16} = 12 \cdot 16 + 5 = 197_{10},$$

$$FF_{16} = 15 \cdot 16 + 15 = 255_{10}.$$

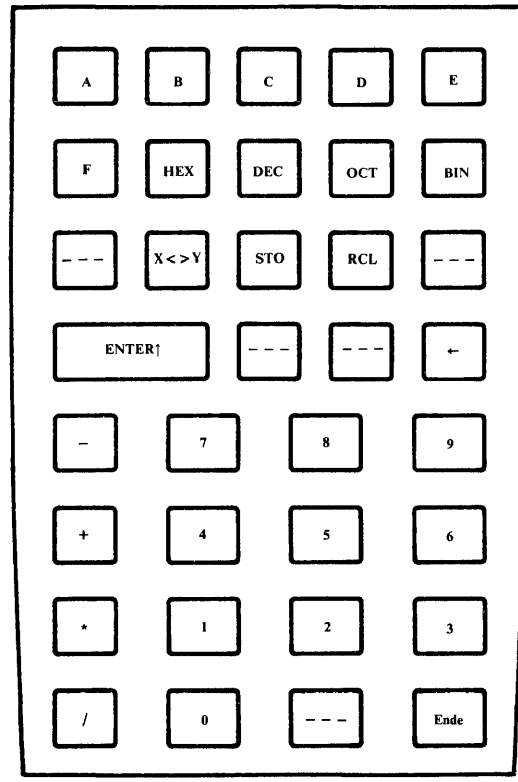
[Abschweifung III: Machen Sie sich an Hand der vorstehenden Gleichungen einmal ganz bewußt, wie zwanghaft unser rechnerisches Denken auf das Dezimalsystem abgerichtet ist. Genaugenommen müßten die Indizes 16 und 10 ja ihrerseits ausdrücklich als Darstellungen zur Basis 10 gekennzeichnet sein. Gäbe es nämlich jemanden, der völlig unbelastet vom Dezimalsystem wäre, würde er beim Lesen der Gleichungen notwendigerweise sofort stolpern und nach dem System, in welchem die Indizes dargestellt sind, fragen.]

Die vorstehend dargelegten Regeln der Zahlendarstellung zur Basis 10 und zur Basis 16 gelten natürlich für jedes System. Auch in anderen Systemen bedeutet jede Ziffer den Koeffizienten einer Potenz der jeweiligen Basis, und die Summe der Produkte ergibt die dargestellte Zahl.

Der Basiswechsel ist eine häufig vorkommende Aufgabe, für deren Lösung ein programmierbarer Taschenrechner offenkundig das richtige Werkzeug ist. Um den Bedürfnissen der Benutzer dieses Werkzeuges entgegenzukommen, wurde von HP sogar ein eigener Rechner, der HP-16C, welcher vor allem für das Rechnen in verschiedenen Zahlensystemen geeignet ist, entworfen. Er erlaubt es, beliebig zwischen den Basen 2 (Binärsystem), 8 (Oktalsystem), 10 (Dezimalsystem) und 16 (Hexadezimalsystem) zu wechseln und die vier Grundrechnungsarten mit Zahlen, die in irgendeinem dieser vier Systeme dargestellt sind, auszuführen. Es gibt Tasten, die mit den Ziffern A bis F versehen sind, um Hexadezimalzahlen problemlos eintasten zu können. Dies geschieht in dem entsprechenden Modus, dem 'HEX-Modus'. Anschließend läßt sich durch einfachen Tastendruck ein Wechsel in den 'DEC-Modus' vornehmen, wodurch die Hexadezimalzahl automatisch in eine Dezimalzahl umgewandelt und jede fernere Eingabe ebenfalls als eine solche gedeutet wird. Der Rechner verbleibt solange in einem bestimmten System-Modus, als man nichts anderes befiehlt.

Das weiter unten abgedruckte Programm "HP-16" simuliert die Basiswechsel-Funktionen des HP-16C. Führen Sie 'XEQ "HP-16"' aus, um das Programm zu starten. Das Tastenfeld wird daraufhin zu einem Bedienungspult für einen HP-16 Simulator. Der nachstehenden Abbildung können Sie die neue Bedeutung der einzelnen Tasten entnehmen. Tasten, deren Kästchen ohne Eintrag sind, bleiben ohne Wirkung, wenn sie gedrückt werden, denn es gibt keine ihren Tastenkodes entsprechenden lokalen numerischen Marken im Programm.

"HP-16"-'GETKEY'-Tastenfeld



Das Programm macht den Eindruck, etwas arbeitsscheu zu sein, weil die Simulation vollständig über die Programmierung erfolgen muß und sich nirgends auf schnelle, festverdrahtete Funktionen, auch nicht auf 'OCT' und 'DEC', stützt. Es simuliert den HP-16C zudem lediglich mit einem 2-Register-Stapel (Register X und Y). Der System-Modus wird durch gesetzte Flags angezeigt: Flag 1 signalisiert den HEX-Modus, Flag 2 den DEC-Modus, Flag 3 den OCT-Modus und Flag 4 den BIN-Modus. Ein gesetztes Flag 0 zeigt — wie beim Edit-Modus von "TE" aus Kapitel 7 — an, daß der HP-41 bereit ist, über 'GETKEY' auf die Betätigung einer Taste zu antworten. Verlischt das Flag 0, was nur kurzzeitig während eines Schleifenrücksprunges geschieht, wird jeder Tastendruck (außer der auf 'R/S') mißachtet. Warten Sie also stets auf die Rückkehr der kleinen Null, bevor Sie die nächste Taste drücken.

Eine Reihe von Beispielen soll jetzt den Umgang mit dem Programm erläutern. Zunächst müssen Sie es mit 'XEQ "HP-16"' starten. Daraufhin erscheint in der Fußleiste der Anzeige die kleine Eins, um zu verkünden, daß Sie sich mit dem Programm im HEX-Modus befinden, und die kleine Null signalisiert die Bereitschaft zur Entgegennahme Ihrer Befehle.

Drücken Sie nun die Taste 'C' (Zeile 1, Spalte 3). Flag 0 verlischt dann kurz und kehrt, zusammen mit einem "C", in die Anzeige zurück. Wenn Sie anschließend eine Dezimalzifferntaste, beispielsweise '2', drücken, tritt "C2" in die Anzeige. Eingabefehler können Sie wie üblich verbessern, indem Sie mit '←' die äußerste rechte Ziffer löschen und

anschließend die richtige Ziffer eintasten.

Wechseln Sie nun in eine Darstellung zur Basis 8, indem Sie die zugehörige Taste (Zeile 2, Spalte 4) drücken. Nach kurzer 'Besinnung' teilt Ihnen das Programm "302" mit. Gleichzeitig zeigt Flag 3 die Gültigkeit des OCT-Modus an.

Wir wollen jetzt 7 auf die Oktalzahl 302 addieren. Dies geschieht in der nächstliegenden Weise: man drückt die Taste '7'; kurz darauf erscheint "7" in der Anzeige; danach drückt man ' + '. Die beiden Oktalzahlen 302_8 und 7_8 werden daraufhin zu dem Ergebnis 311_8, welches als "311" in der Anzeige erscheint, zusammengezogen.

Sie können diese Oktalzahl nun umstandslos im Binärsystem darstellen, indem Sie einfach die entsprechende Taste (Zeile 2, Spalte 5) drücken. Die Zeit, welche der HP-41 benötigt, um die zur Darstellung erforderlichen 8 Binärziffern zu ermitteln, ist spürbar lang, doch schließlich erscheint "11001001", also 11001001_2. Flag 4 gibt über den gültigen BIN-Modus Auskunft.

Die Umrechnung in die Dezimaldarstellung der Zahl 11001001_2 wird durch einen einfachen Druck auf die Taste in Zeile 2, Spalte 3 vorgenommen. Sie erfahren "201" und sehen Flag 2, das DEC-Signal, gesetzt.

Wenn Sie an dieser Stelle 'X< >Y' befehlen (Achtung: Zeile 3, Spalte 2, nicht etwa Zeile 2, Spalte 1!), gelangt $194_{10} = C2_{16}$ in die Anzeige, also jene Zahl, die während der Addition von 7 im OCT-Modus — sie hatte dort die Gestalt 302_8 — ins Register Y gelangt war. Dies geschieht durch einen 'Schiebevorgang', welcher dem wohlbekannten 'Herabtropfen' des Inhaltes von T nach Z bei Stapelsenkung auf Grund einer normalen Addition ähnlich ist.

Die Taste 'ENTER↑' arbeitet erwartungsgemäß: sie schließt die Zifferneingabe ab und bringt den Inhalt von X nach Y.

Mit dem Programm "HP-16" können Sie auch Zahlen in den Registern R_{01} bis R_{15} ablegen und von dort zurückrufen. Allerdings nicht ganz in der gewohnten Weise. Sie drücken 'STO' oder 'RCL', warten auf die Empfangsbereitschaft signalisierende Rückkehr der kleinen Null, und steuern dann das ins Auge gefaßte Register mit einer der Einzeltasten '1' (für R_{01}) bis 'F' (für R_{15}) an. Register R_{00} müssen Sie unbedingt unbehelligt lassen, weil es die gerade gültige Basis (16, 10, 8 oder 2) beherbergt. (Hinweis: Wenn Sie sich nicht darüber wundern, daß 'RCL, 0' in *jedem* System-Modus "10" in die Anzeige befördert, ist Ihnen die Arbeitsweise des Programms völlig klar geworden.) Die Möglichkeit, mit 'STO' und 'RCL' arbeiten zu können, tröstet darüber hinweg, in "HP-16" nur einen 2-Register-Stapel zur Verfügung zu haben. 'RCL' hebt den Stapel wie beim gewöhnlichen Gebrauch des HP-41, so daß diesem Befehl kein 'ENTER↑' vorausgeschickt zu werden braucht. Umgekehrt muß dagegen dem 'STO', anders als beim normalen Verhalten, ein 'ENTER↑' nachgeschickt werden, wenn die Eingabe unterbrochen werden soll, sonst werden die folgenden Ziffern der angezeigten Zahl angefügt.*)

Wenn Sie das Programm verlassen wollen, können Sie das mit 'R/S' oder 'ON' tun. Alle von "HP-16" benutzten Flags werden dabei gelöscht. In den Registern X und Y bleiben die Zahlen zurück, welche beim Verlassen des Programms darin enthalten waren, und zwar in wohlvertrauter Dezimalgestalt, unabhängig davon, in welchem System-Modus der Programmlauf beendet wurde.

*) Anm. d. Übers.: Dieser Schönheitsfehler läßt sich dadurch beseitigen, daß man den Befehl 'CF 07' hinter Zeile 187 einschiebt.

Programmausdruck von "HP-16"
(Barcode im Anhang D)

```
01♦LBL "HP-16"
02 2
03 X<>F
04 16
05 STO 00
06 CLST
07 CLA
08 CF 21

09♦LBL 05
10 RDN
11 AVIEW
12 SF 00
13 GETKEY
14 CF 00
15 X=0?
16 GTO 05
17 RDN
18 SF 25
19 XEQ IND T
20 R↑
21 GTO 05

22♦LBL 82
23 0
24 GTO 10

25♦LBL 72
26♦LBL 73
27♦LBL 74
28 R↑
29 71
30 -
31 GTO 10

32♦LBL 62
33♦LBL 63
34♦LBL 64
35 R↑
36 58
37 -
38 GTO 10

39♦LBL 52
40♦LBL 53
41♦LBL 54
42 R↑
43 45
44 -

45♦LBL 10
46 48
47 GTO 06

48♦LBL 11
49♦LBL 12
50♦LBL 13
51♦LBL 14
52♦LBL 15
53 R↑
54 1
55 -
56 GTO 10

57♦LBL 21
58 15

59♦LBL 10
60 55

61♦LBL 06
62 FS?C 05
63 GTO 10
64 X<>Y
65 +
66 FS? 07
67 GTO 09
68 0
69 X<>Y
70 CLA
71 SF 07

72♦LBL 09
73 XTOA
74 X<> L
75 X<> 00
76 ST* Y
77 X<> 00
78 +
79 RTN

80♦LBL 10
81 RDN
82 SIGN
83 RDN
84 FC? 06
85 RCL IND L
86 STO IND L
87 FS?C 06
88 RTN
89 GTO 07

90♦LBL 44
91 RCL 00
92 /
93 INT
94 1
95 CHS
96 AROT
97 RDN
98 ATOX
99 RDN
100 RTN

101♦LBL 01
102♦LBL 84
103 0
104 X<>F
105 RDN
106 CF 25
107 CLD
108 STOP
109 RTN

110♦LBL 22
111 2
112 16
113 GTO 10

114♦LBL 23
115 4
116 10
117 GTO 10

118♦LBL 24
119 8
120 ENTER↑
121 GTO 10

122♦LBL 25
123 16
124 2

125♦LBL 10
126 STO 00
127 RDN
128 X<>F
129 RDN

130♦LBL 07
131 CF 07
132 CLA
133 ENTER↑

134♦LBL 08
135 ENTER↑
136 X<> 00
137 ST/ 00
138 MOD
139 9
140 ST- Y
141 RDN
142 7
143 X<>Y
144 X>0?
145 ST+ Y
146 X<=0?
147 X<>Y
148 RDN
149 57
150 ST+ Y
151 RDN
152 XTOA
153 X<> L
154 X<> 00
155 1
156 CHS
157 AROT
158 RDN
159 INT
160 X>0?
161 GTO 08
162 RDN
163 RTN

164♦LBL 41
165 ENTER↑
166 CF 07
167 RTN

168♦LBL 32
169 X<>Y
170 GTO 07

171♦LBL 51
172 CHS

173♦LBL 61
174 X<>Y
175 ST+ Y
176 X<>Y
177 ABS
178 GTO 07

179♦LBL 81
180 1/X
181♦LBL 71
182 X<>Y
183 ST* Y
184 X<>Y
185 INT
186 GTO 07

187♦LBL 33
188 SF 06

189♦LBL 34
190 SF 05
191 END
```

297 Bytes

Das Programm "HP-16" ist ein anschauliches Beispiel dafür, wie weitgehend Sie den HP-41 zu einem ganz auf Ihre eigenen Bedürfnisse abgerichteten Rechenwerkzeug machen können, wenn Sie die Eigenschaften der Funktion 'GETKEY' ausgiebig nutzen. An Geschwindigkeit und Umfang des Programms müssen im Zweifelsfalle natürlich einige Zugeständnisse gemacht

werden, aber die durch 'GETKEY' erreichbare Benutzerfreundlichkeit ist kaum noch zu überbieten.

Zeilen-Analyse von "HP-16"

Das Programm "HP-16" ist aus einer Reihe von kleinen Teilstücken, die alle ein paar einfachen Grundregeln gehorchen, zusammengesetzt. Im Register R_{00} wird die Basis des Systems, das gerade gelten soll, festgehalten. Beim Start ist dies die Basis 16 des Hexadezimalsystems (Zeilen 04 – 05). Die beiden Stapelregister des Programms, welche eigentlich nur Scheinregister sind, sollen während der gegenwärtigen Analyse zur Unterscheidung von den gewöhnlichen Stapelregistern X und Y mit $\bar{X}$ und $\bar{Y}$ bezeichnet werden. Vor und nach jedem 'XEQ IND T' (Zeile 19) enthalten X und Y die Inhalte von $\bar{X}$ und $\bar{Y}$. Die im echten Stapel liegenden Zahlen haben grundsätzlich Dezimalgestalt. Die in der Anzeige als $\bar{X}$-Inhalt vorgewiesene Zahl ist hingegen nur das Bild einer im Alpha-Register liegenden Zeichenkette. Sobald diese Zahl sich durch einen anderen Vorgang als durch den der Hinzufügung oder Entfernung einer Ziffer während noch andauernder Eingabe ändert, wird die von 'LBL 07' angeführte Unterroutine (Zeile 130) angesprungen. Dort entsteht das vollständige Alpha-Bild und somit der $\bar{X}$-Inhalt aus der in X enthaltenen Dezimalzahl nach Maßgabe des geltenden Systems.

'LBL 10' wird mehrfach benutzt, um kurze Sprünge *vorwärts* im Programm auszuführen. Beachten Sie in diesem Zusammenhang die im Handbuch des HP-41 erläuterte diesbezügliche Regel, welche besagt, daß die Mehrfachbenutzung einer numerischen Marke im selben Programm nur dann zulässig ist, wenn mit den zugehörigen Sprungbefehlen keine Rücksprünge ausgelöst werden sollen.

Die von 'LBL 05' (Zeile 09) angeführte Schleife ist die Hauptschleife des Systems. In ihr arbeitet 'GETKEY', um Ihre Befehle entgegenzunehmen. Sobald Sie eine Taste betätigt haben, verlischt Flag 0 und die zur gedrückten Taste gehörige Unterroutine wird ausgeführt. Anschließend erscheint der neue $\bar{X}$-Inhalt in der Anzeige, Flag 0 läßt sich wieder sehen, und der nächste 'GETKEY'-Befehl erwartet den Fortgang des Geschehens. Flag 25 wird dazu benutzt, Programmunterbrechungen, die durch das unbeabsichtigte Drücken funktionsfreier Tasten (z.B. 'CHS') verursacht werden könnten, auszuschließen. Beiläufig bemerkt verhindert das in Zeile 08 gelöschte Flag 21, solche Programmunterbrechungen, die ein angeschlossener aber ausgeschalteter Drucker beim 'AVIEW' herbeiführen würde. Wollen Sie jedoch einen laufenden Ausdruck des Alpha-Inhaltes erzielen, müssen Sie das genannte 'CF 21' tilgen.

Bei Zifferneinträgen gelangt deren Dezimalwert nach Y und die Differenz zu dem für die Alpha-Anzeige maßgebenden ASCII-Kode nach X. Beispielsweise bringt ein Druck auf die Taste 'C' (Tastenkode 13) den Wert 12 (Zeilen 53 – 55) nach Y und die Zahl 55 nach X (Zeile 60). Für die Hexadezimalziffern A bis F entsteht der ASCII-Kode nämlich gerade dadurch, daß man 55 auf den jeweiligen Dezimalwert (10 bis 15) addiert (Zeile 65). Entsprechendes findet für alle Ziffern statt. Die von 'LBL 06' angeführte Befehlsfolge hängt das ermittelte ASCII-Zeichen dem Alpha-Inhalt an. Flag 7 zeichnet verantwortlich für den Fortbestand (gesetzt) bzw. den Neubeginn (gelöscht) einer Zifferneingabe. Sobald eine Zifferntaste gedrückt wird, nachdem eine zusammenhängende Zifferneingabe durch z.B. ' + ' oder 'ENTER↑' abgeschlossen worden ist (beide Befehle lassen Flag 7 gelöscht zurück), hebt sich der gedachte "HP-16"-Stapel. Dies geschieht vornehmlich durch die Befehle '0, X<>Y, CLA' (Zeilen 68 – 70), eine Folge, welche bei gesetztem Flag 7, also bei fortbestehendem Eingabevorgang, durch den Sprungbefehl 'GTO 09' umgangen wird. Die von 'LBL 09' angeführte Befehlsfolge bringt den durch Ziffernanfügung in X entstehenden Dezimalwert auf den neuesten Stand. Die Zeilen 75 – 76 multiplizieren zu diesem Zweck den zu diesem Zeitpunkt in Y liegenden bislang entstandenen Dezimalwert mit der Basis aus R_{00}, und die Zeilen 77 – 78 schlagen diesem Produkt dann den Dezimalwert der zuletzt eingetasteten Ziffer zu.

Drückt man eine der Zifferntasten '1' bis 'F' als Teil eines 'STO'- oder 'RCL'-Befehles, welche beide Flag 5 setzen (Zeile 190) und damit eine Verzweigung von Zeile 63 aus auf Zeile 80 veranlassen, wird der gewünschte Registerbefehl ('STO' bei gesetztem, 'RCL' bei gelöschtem Flag 6) in den Zeilen 86 bzw. 85 ausgeführt. Bei 'RCL'-Befehlen findet folgerichtig ein Sprung zur Marke 'LBL 07' statt, weil ja der neu nach X geholte Wert für die Anzeige in ein gültiges Alpha-Bild verwandelt werden muß (s. weiter unten), also in das Scheinregister $\bar{X}$ gelangen soll.

In der mit 'LBL 44' beginnenden Routine wird der X-Inhalt ganz einfach durch die gegenwärtige Basis dividiert. Der verbleibende ganzzahlige Anteil des Quotienten ist genau der X-Rest, der zurückbleiben muß, wenn man die letzte Ziffer des $\bar{X}$-Inhaltes fortnimmt (Zeilen 94 – 98).

Die sowohl über 'LBL 01' ('ON') als auch über 'LBL 84' ('R/S') erreichbare Schlußfolge benutzt 'X<>F', um alle 8 Flags F_{00} bis F_{07} zu löschen, und befreit die Anzeige vom $\bar{X}$-Inhalt (Zeile 107). Man kann das Programm danach wieder mit 'R/S' starten, doch läßt sich nach einem so durchgeführten Neustart der Anzeigenfußleiste nicht mehr entnehmen, in welchem System man rechnet, es sei denn, man drückt eine der vier Basistasten ('R↓' bis 'TAN').

Die Auswahl des Systems erfolgt über die Marken 'LBL 22' bis 'LBL 25'. Dabei wird die zur Erzeugung des Flagsignals erforderliche Zahl nach Y gelegt und die zugehörige Basis nach X. Der anschließende Sprung zur Zeile 125 bringt die Basis nach R_{00} und setzt das richtige – und nur dieses eine – Flag.

Nach einem Basiswechsel ist das Alpha-Register neu herzurichten, m.a.W. der neue $\bar{X}$-Inhalt vorzuweisen. Die bei 'LBL 07' beginnende Routine führt dies aus. Das Alpha-Register wird zunächst gereinigt (Zeile 132), und dann werden die neuen Ziffern Stück für Stück von rechts nach links fortschreitend ermittelt und ins Alpha-Register gesetzt. Der Befehl 'ENTER↑' unmittelbar vor 'LBL 08' dient dazu, eine Kopie der abzubildenden Zahl nach Y zu retten, weil der X-Inhalt durch den Verarbeitungsvorgang verloren geht.

Gleich hinter 'LBL 08' liefert die Funktion 'MOD' den Dezimalwert der hintersten Ziffer. Die nächsten 13 Zeilen (139 – 151) rechnen diesen Dezimalwert in den zugehörigen ASCII-Kode um. Dann hängt ein 'XTOA' das entsprechende Zeichen dem Alpha-Inhalt an. Weil es wegen der Verarbeitungsrichtung (von rechts nach links) auf diese Weise zu Unrecht ans Ende gelangt, muß es noch mit 'AROT' an den Kettenanfang geschoben werden (Zeile 157). Der jeweils am Schleifenanfang in X liegende Dezimalwert ist durch die Basis zu dividieren (dies geschieht in Zeile 137) und der verbleibende ganzzahlige Anteil des Quotienten weiter zu verarbeiten (Zeilen 159 – 161). Auf diese Weise entsteht bei jedem Durchlauf durch die 'LBL 08'-Schleife eine Ziffer für die Darstellung des Ausgangswertes zur gültigen Basis. Solange nicht Null auf Zeile 159 zurückbleibt, muß eine weitere Ziffer ermittelt, mithin die Schleife 'LBL 08' abermals durchlaufen werden.

Bei 'LBL 41' beginnt eine Befehlsfolge, die Flag 7 löscht und somit die schon oben erwähnte Befehlsfolge '0, X<>Y, CLA' (Zeilen 68 – 70) zugänglich macht, wodurch das Alpha-Register zur Neuaufnahme einer Folge von Ziffern vorbereitet wird. – 'LBL 32' steht dem Befehl 'X<>Y' voran, hinter dem eine Verzweigung zur Routine 'LBL 07' stattfindet, um den nach $\bar{X}$ gelangten $\bar{Y}$-Inhalt anzuzeigen. – Die von zwei Befehlen 'X<>Y' eingerahmten Stapeloperationen 'ST + Y' und 'ST* Y' (Zeilen 175 und 183) erlauben es, den ursprünglichen Y-Inhalt trotz Addition bzw. Multiplikation unversehrt zurückzulassen (ein beherzigenswerter Stapeltrick!, d. Übers.). – Weil die 'LBL 07'-Routine weder negative noch nicht-ganze Zahlen zu verarbeiten vermag, sorgen die Befehle 'ABS' (Zeile 177) nach einer Subtraktion und 'INT' (Zeile 185) nach einer Division für eine rechtzeitige Anpassung der errechneten Werte. – 'STO' ('LBL 33') und 'RCL' ('LBL 34') setzen beide Flag 5, um anzuzeigen, daß die nächste Zifferntaste zur Adressierung eines Registers betätigt wird (Zeilen 62, 63 und 80). Flag 6 entscheidet zwischen 'STO'- und 'RCL'-Befehlen (Zeilen 84 – 86).

Der Autor hofft, daß der HP-16 Simulator einen überzeugenden Eindruck von den nahezu unbeschreiblich vielfältigen Einsatzmöglichkeiten der Funktion 'GETKEY' zu vermitteln vermag. Ihre eigenen Anwendungen mögen einfacher oder verwickelter sein, die Grundgedanken der Verwendung von 'GETKEY' jedenfalls sind in dem vorstehenden Musterbeispiel vollständig enthalten und sinnfällig verwirklicht.

KAPITEL 10

Synthetische Programmierung

10A. Was ist synthetische Programmierung?

Synthetische Befehle sind solche HP-41-Befehle, die es zwar gibt, die aber ein Schattendasein führen, weil man sie nicht durch gewöhnliche, vom Hersteller beschriebene Tastenbedienung in den Programmspeicher bringen kann. Die Erzeugung und Verwendung synthetischer Befehle heißt synthetische Programmierung. Man kann tausende synthetischer Befehle herstellen, angefangen mit Ton-Befehlen, bei denen Ton-Höhe und -Dauer z.T. erheblich von den bekannten 'Klängen' abweichen, über leistungsfähige Befehle, die Zugriff auf die Register des Betriebssystems gestatten, bis hin zu schnellen, Bytes sparenden Textzeilen. Synthetische Programmierung kann dem HP-41 nichts anhaben, wenn man von kleinen Unglücksfällen – bekannt unter der Bezeichnung GAU (*g*rößter *a*nzunehmender *U*nfall) – absieht, die bei leichtsinnigen Versuchen in unbekannten Gefilden des Betriebssystems auftreten können und z.B. in Gestalt eines "MEMORY LOST" oder einer zeitweiligen Blockade des Tastenfeldes auftreten. In Abschnitt 10K finden Sie Hinweise zur Erkennung und Behebung von GAUs.

Synthetische Programmierung ist auf allen Rechnern der Familie HP-41 möglich, unabhängig vom Typ (C, CV, CX) und vom Herstellungsdatum. Sie beruht allein auf dem allen Rechnern der Serie HP-41 gemeinsamen Betriebssystem und dessen grundlegenden Eigenschaften.

Die in diesem Kapitel vorgestellten Programme stützen sich auf synthetische Techniken, um Zugang zu jenen Bereichen des Speichers zu erlangen, die gewöhnlich unerreichbar sind. Dazu gehören u.a. die Register, in welchen die getätigten Tastenzuweisungen vermerkt werden, und die Kopfregister im X-Memory.

Weil sich synthetische Befehle nicht ohne entsprechende Herrichtung des HP-41 eintasten lassen, einem Anfänger also das Edieren der Programme verwehrt wäre, sind im Anhang D alle Programme dieses Kapitels im Barcode abgedruckt. Leser, die keinen optischen Lesestift zur Verfügung haben, können die Programme auf Magnetkarten gelesen bekommen, wenn Sie die Karten selbst liefern und einen Freiumschlag beilegen. Berücksichtigen Sie in einem solchen Fall den Platzbedarf laut Bedienungsanleitung des Kartenlesers. Die Post ist an den

Heldermann Verlag
Nassauische Str. 26
D-1000 Berlin 31
Fed. Rep. Germany

oder an den Übers. zu richten. Mühevoller aber wesentlich spannender ist es allerdings, das synthetische Programmieren selbst zu erlernen, um die vorgestellten Programme und eigene Entwürfe selbständig in den Rechner bringen zu können. Im Anhang C finden Sie entsprechende Literaturhinweise.

Jedem, der seinem HP-41 wirklich gewachsen sein möchte, sozusagen das 'volle Hoheitsrecht' über ihn ausüben will, ist zu empfehlen, die synthetische Programmierung zu erlernen. Es ist, als entdeckte man in dem vertrauten Gefährten einen völlig neuen Rechner, und es gibt wohl unter den Benutzern niemanden, dessen Puls sich nicht beschleunigt beim Anblick der ersten selbst zusammengebastelten Befehle und Töne, die aus geheimnisvoller Tiefe zu kommen scheinen.

Ich hoffe, daß die Programme dieses Kapitels Sie anregen werden, in die Synthetik einzudringen. Sollten Sie ein schon erfahrener Synthetiker sein — und deren wird es nicht wenige geben — werden Sie die Leistungsfähigkeit und Vielseitigkeit der Programme sicher zu würdigen wissen. Dem Neuling verschaffen Sie eine lebendige Vorstellung von der Kraft und den Möglichkeiten synthetischer Programmierung.

10B. Alle X-Funktionen auf einer Taste

In diesem Abschnitt werden Sie ein Programm kennenlernen, mit dessen Hilfe *jede* X-Funktion dadurch aufgerufen werden kann, daß man statt ihres Namens einfach einen kurzen Zahlenkode eintastet. Das Programm stammt von Clifford Stern, einem 'Großmeister' auf dem Gebiet der synthetischen Programmierung. Clifford hat sich mit äußerst schwierigen synthetischen Programmen beschäftigt, insbesondere mit nützlichen Routinen zur Verminderung von 'Tastenarbeit'.

Es ist eine einfache Angelegenheit, mehrere X-Funktionen in ihrer Eigenschaft als HP-41-Befehle gemäß der vorgesehenen Regel mit 'ASN' einer Taste zuzuweisen. Sie haben das vielleicht schon mit so häufig benutzten Funktionen wie 'EMDIR' oder 'SAVEP' und 'GETP' getan. Sofern Sie viel mit Datendateien arbeiten, werden Sie auch die Funktionen 'RCLPT', 'SEEKPT', 'SAVERX' und 'GETRX' auf Tasten liegen haben. Solche Zuweisungen, das lernt man schnell, sind handlich, doch kann im Handumdrehen ein nennenswerter Teil des USER-Tastenfeldes ausgebucht sein, so daß Sie leicht in Verlegenheit kommen, wenn viele Zuweisungen zur Hand liegen sollen.

Was halten Sie davon *alle* X-Funktionen *einer einzigen* Taste zuzuweisen? Unmöglich sagen Sie? Keineswegs bei synthetischer Programmierung! Alles was Sie benötigen ist das Programm "XF" (*X-F*unktionen).

Wenn Sie eine beliebige X-Funktion ausführen wollen, brauchen Sie nur den zugehörigen Zahlenkode ins X-Register zu legen und "XF" auszuführen. Die zu den einzelnen X-Funktionen gehörigen Zahlenkodes bilden die jeweils erste Spalte in der nachstehenden Tabelle. Eine kurze Abschweifung soll dazu dienen, Ihnen die Beziehungen zwischen den X-Funktionen und den ihnen zugeordneten Kodes verständlich zu machen:

Zahlenkodes für "XF"
(in der jeweils dritten Spalte sind die XROM-Zahlen aufgeführt)

	-EXT FCN 2D			-CX EXT FCN			-CX TIME	
1	ALENG	25,01	49	ASROOM	25,49	95	CLALMA	26,31
2	ANUM	25,02	50	CLRGX	25,50	96	CLALMX	26,32
3	APPCHR	25,03	51	ED	25,51	97	CLRALMS	26,33
4	APPREC	25,04	52	EMDIRX	25,52	98	RCLALM	26,34
5	ARCLREC	25,05	53	EMROOM	25,53	99	SWPT	26,35
6	AROT	25,06	54	GETKEYX	25,54			
7	ATOX	25,07	55	RESZFL	25,55			
8	CLFL	25,08	56	Σ REG?	25,56			
9	CLKEYS	25,09	57	X=NN?	25,57			
10	CRFLAS	25,10	58	X≠NN?	25,58			
11	CRFLD	25,11	59	X<NN?	25,59			
12	DELCHR	25,12	60	X<=NN?	25,60			
13	DELREC	25,13	61	X>NN?	25,61		- WAND 1F -	
14	EMDIR	25,14	62	X>=NN?	25,62	129	WNDDTA	27,01
15	FLSIZE	25,15				130	WNDDTX	27,02
16	GETAS	25,16				131	WNDLNK	27,03
17	GETKEY	25,17				132	WNDSUB	27,04
18	GETP	25,18		-TIME 2C		133	WNDSCN	27,05
19	GETR	25,19	65	ADATE	26,01	134	"WNDTST	27,06
20	GETREC	25,20	66	ALMCAT	26,02			
21	GETRX	25,21	67	ALMNOW	26,03			
22	GETSUB	25,22	68	ATIME	26,04			
23	GETX	25,23	69	ATIME24	26,05			
24	INSCHR	25,24	70	CLK12	26,06			
25	INSREC	25,25	71	CLK24	26,07			
26	PASN	25,26	72	CLKT	26,08			
27	PCLPS	25,27	73	CLKTD	26,09			
28	POSA	25,28	74	CLOCK	26,10			
29	POSFL	25,29	75	CORRECT	26,11			
30	PSIZE	25,30	76	DATE	26,12			
31	PURFL	25,31	77	DATE+	26,13			
32	RCLFLAG	25,32	78	DDAYS	26,14			
33	RCLPT	25,33	79	DMY	26,15			
34	RCLPTA	25,34	80	DOW	26,16			
35	REGMOVE	25,35	81	MDY	26,17			
36	REGSWAP	25,36	82	RCLAF	26,18			
37	SAVEAS	25,37	83	RCLSW	26,19			
38	SAVEP	25,38	84	RUNSW	26,20			
39	SAVER	25,39	85	SETAF	26,21			
40	SAVERX	25,40	86	SETDATE	26,22			
41	SAVEX	25,41	87	SETIME	26,23			
42	SEEKPT	25,42	88	SETSW	26,24			
43	SEEKPTA	25,43	89	STOPSW	26,25			
44	SIZE?	25,44	90	SW	26,26			
45	STOFLAG	25,45	91	T+X	26,27			
46	X<>F	25,46	92	TIME	26,28			
47	XTOA	25,47	93	XYZALM	26,29			

Wenn Sie X-Funktionen als Bestandteil eines Programms eintasten, erscheinen – bei eingestecktem X-Modul – in der Anzeige Programmzeilen der Form

XEQ⌜FUNKTIONSNAME,

die sich unmittelbar nach Verlassen des ALPHA-Modus in

FUNKTIONSNAME

ändern. Entfernen Sie später den X-Modul, zeigen sich diese Zeilen im PRGM-Modus in abermals neuer Gestalt als

XROM 25,xy

und werden auf 'PRP' hin auch so ausgedruckt. Die Abkürzung 'XROM' weist darauf hin, daß die Funktion in einem '*ex*ternen *r*ead-*o*nly *m*emory' ansässig ist. An der Nummer 25 erkennt man, daß es sich um den X-Funktionen-Modul handelt. Die zweiziffrige Zahl 'xy' kennzeichnet die Funktion innerhalb des Moduls. Genau diese zweiziffrige Zahl ist es, derer sich das Programm "XF" bedient. Der oben stehenden Tabelle können Sie übrigens entnehmen, daß "XF" nicht nur für die X-Funktionen verwendbar ist, sondern auch für die Funktionen des Time-Moduls (XROM-Zahl 26) und des Lesestiftes (XROM-Zahl 27). Die Kodes 49 – 62 und 95 – 99 haben nur für den HP-41CX Gültigkeit.

Programmausdruck von "XF"
(Barcode im Anhang D)

01♦LBL "XF"	10 RDN	19 X<> a	28 STO c
02 X<>Y	11 XTOA	20 RCL [	29 X<> L
03 SIGN	12 RDN	21 STO IND \	30 ENDIR
04 X<> \	13 501	22 X<>]	31 CLD
05 STO a	14 SIZE?	23 CLA	32 END
06 RDN	15 ST- Y	24 X<> [	
07 64	16 RDN	25 RDN	
08 ST+ Y	17 X<> \	26 X<> \	
09 "⊦♦♦×i♦δ♦▼→t	18 X<> c	27 X<> a	**74 Bytes**

Für Leser, die die synthetischen Zeilen des Programms "XF" selbständig mit dem Byte-Schnapper, mit einem Byte-Lader oder mit irgendeiner anderen synthetischen Methode eintasten wollen, hier die Dezimalwerte der Bytes:

Zeile 04: 206, 118
Zeile 05: 145, 123
Zeile 09: 254, 127, 0, 0, 1, 105, 0, 18, 0, 123, 145, 125, 206, 116, 166
Zeile 17: 206, 118
Zeile 18: 206, 125
Zeile 19: 206, 123
Zeile 20: 144, 117
Zeile 21: 145, 246
Zeile 22: 206, 119
Zeile 24: 206, 117
Zeile 26: 206, 118
Zeile 27: 206, 123
Zeile 28: 145, 125

Beschreibung von "XF"

"XF" *muß das erste Programm im Katalog 1 sein.* Sie sind also gezwungen, alle Programme im Programmspeicher zu löschen, bevor Sie "XF" laden. Um dabei keiner Programme verlustig zu gehen, sollten Sie vorher all das, was Sie noch brauchen, mit 'SAVEP' in das X-Memory bringen oder auf Karten oder Kassette schreiben. Anschließend fegen Sie den Programmspeicher am schnellsten dadurch leer, daß Sie den Namen des ersten Programms ins Alpha-Register setzen und 'PCLPS' ausführen. Danach laden Sie "XF".

Wenn Sie mit einem HP-41C arbeiten, kann es sein, daß Sie die Zeile 13 abändern müssen. Die dort auftretende Zahl, gewöhnlich 501, muß 245 + 64·n lauten, wobei n die Anzahl der eingesteckten Speichererweiterungsmodule ist. Gemeint sind die Speichererweiterungen des Hauptspeichers, nicht etwa die Erweiterungen durch X-Memory-Module. Sollten Sie also beispielsweise zwei Speichererweiterungsmodule stecken haben, muß die Zahl in Zeile 13 folgerichtig 373 heißen. Mit einem Quad-Modul lautet sie 501, wie im Barcode ausgewiesen.

WARNUNG: Eine falsche Zahl in Zeile 13 ist nicht auf die leichte Schulter zu nehmen, denn "MEMORY LOST" kann die Folge sein. Wenn Sie also mit wechselnder Hauptspeichergröße arbeiten, vergessen Sie nie, die Zeile 13 dem jeweils gültigen Speicherumfang anzupassen. Sie sind sonst einen Tastenschritt breit vom Abgrund entfernt. X-Memory-Module betrifft diese Warnung nicht.

Um das größtmögliche Maß an Bequemlichkeit zu erreichen, weist man "XF" einer Taste zu. Erfahrene Benutzer verwenden für solche Zuweisungen keine Zifferntasten, weil diese ständig benötigt werden und daher allzu leicht einer unbeabsichtigten Berührung ausgesetzt sind. Statt einen Ziffereintrag vorzunehmen, ruft man dann voreilig Programme oder Funktionen auf. Die meisten Benutzer haben die fortwährend benötigte Funktion 'SIZE' auf einer Taste liegen. Weisen Sie "XF" am besten dieser Taste zu, dann können Sie im Anschluß an die Eingabe der gewünschten Datenregister-Anzahl mit 'ENTER↑', 30 (Zahlenkode für 'PSIZE'), XEQ "XF"' dasselbe wie mit 'SIZE' bewirken und zusätzlich auf alle anderen X-Funktionen über dieselbe Taste zugreifen.

"XF" ist ein Programm, das sich, während es abläuft, selbst abändert, indem es eine kurze Befehlsfolge, welche die gewünschte X-Funktion enthält, erzeugt und in jenen Programmspeicher-Bereich überträgt, in dem das erste Programm liegt. Das erste Programm wird also bedingungslos abgeändert, gleichgültig, um welches Programm es sich dabei handelt. Ist, wie vorgesehen, "XF" das erste Programm, gelangt die erzeugte Befehlsfolge genau an die richtige Stelle, auf Zeile 30 nämlich, auf der Sie stets die zuletzt über "XF" aufgerufene X-Funktion vorfinden. Zeile 31, 'CLD', kann bei Verwendung eines HP-41CX gelöscht werden, weil sie nur dazu dient, die Anzeige nach einem 'EMDIR' wieder freizumachen.

WARNUNG: Ändern Sie das Programm "XF" nicht ab ohne sicherzustellen, daß die Byte-Anzahl zwischen dem Programmanfang und der Zeile 30 unverändert bleibt, sonst wird die erzeugte X-Funktion an falscher Stelle abgelegt.*)

*) Anm. d. Übers.: Aus dem genannten Grunde dürfen Sie auch ein u.U. erwünschtes 'CF 27' auf keinen Fall an eine andere Stelle als unmittelbar vor das 'END' setzen. Überdies ist es zu empfehlen, unmittelbar nach Edition oder Einlesen des Programms ein alle störenden Null-Bytes mit Sicherheit beseitigendes 'PACK' auszuführen.

"XF" ist ein hervorragendes Beispiel für die Leistungskraft der synthetischen Programmierung. Programme, die gewissermaßen 'Hand an sich selbst legen', sind stets sehr trickreich und schwierig herzustellen, doch zeigen sie, auf wie einfache Weise Aufgaben gelöst werden können, die sich sonst kaum oder gar nicht in Angriff nehmen ließen, mindestens aber einen unverhältnismäßig hohen Aufwand an gewöhnlichen Hilfsmitteln erfordern würden.

Anwendungsbeispiel für "XF"

Eine der vermutlich besonders häufig benutzten X-Funktionen ist 'EMDIR', denn immer wieder will man wissen, was im X-Memory liegt. Der "XF"-Tabelle können Sie entnehmen, daß 14 der zu 'EMDIR' gehörende Zahlenkode ist. Folglich wird Ihnen auf

'14, XEQ "XF"'

hin das Inhaltsverzeichnis des X-Memorys vorgewiesen.

Gebrauchsanweisung für "XF"

1. Stellen Sie sicher, daß "XF" das erste Programm im Katalog 1 ist. Sie brauchen sich dessen natürlich nicht bei jedem Gebrauch von "XF" von neuem zu vergewissern, doch dürfen nachträgliche Verschiebungen des Programms nicht vorkommen.
2. Laden Sie die Stapelregister X, Y und Z sowie das Alpha-Register mit den Größen bzw. Texten, die zur ordnungsgemäßen Ausführung der ins Auge gefaßten X-Funktion dort liegen müssen. Die für das Alpha-Register vorgesehene Kette darf dabei nicht mehr als 14 Zeichen umfassen; darüber nach links hinausreichende Zeichen gehen durch "XF" verloren.
3. Drücken Sie 'ENTER↑', und legen Sie den Zahlenkode gemäß der "XF"-Tabelle ins X-Register. Die zuvor in X, Y und Z abgelegten Größen sind jetzt in Y, Z und T. *Benutzen Sie* — außer auf dem HP-41CX — *nie den Zahlenkode 0*, sonst gehen Ihnen die Zuweisungen sämtlicher globalen Marken verloren, und es bleiben nur 'Geisterzuweisungen', die sich meistens wie 'ABS' verhalten, zurück.
4. Drücken Sie die Taste, der Sie "XF" zugewiesen haben, im USER-Modus. Die angestrebte X-Funktion wird wunschgemäß ausgeführt. "XF" stellt zu diesem Zweck im Alpha-Register eine Folge von Bytes her, überträgt diese in den Programmspeicher an die Stelle, wo die Zeilen 27 — 30 liegen, und führt die entstandene Befehlsfolge dann aus. Wenn Sie "XF" unterbrechen und nicht sofort wieder starten, laufen Sie Gefahr, ein "MEMORY LOST" zu erleiden. Zwar enthält das Programm ein paar dagegen gerichtete Sicherheitsmaßnahmen, doch wenn Sie zwischen den Zeilen 18 und 28 unterbrechen und dem Programm dann nicht mehr den vorgesehenen Ablauf gestatten, werden Sie mit großer Wahrscheinlichkeit Opfer des gefürchteten GAUs.
5. Sobald die X-Funktion ausgeführt ist, verschwindet die 'Graugans'. Bei Fehlern wird die entsprechende Meldung gegeben. Wenn Sie z.B. "XF" mit 23 im X-Register, das entspricht 'GETX', ausführen, obwohl die Arbeitsdatei keine Datendatei ist, hält Ihnen der Rechner beleidigt die Meldung "FL TYPE ERR" entgegen.
6. In Zeile 30 können Sie sich darüber unterrichten, welche X-Funktion zuletzt mit "XF" ausgeführt wurde ('GTO "XF", GTO.030'). Dies kann insbesondere beim Erproben von "XF" hilfreich sein.

Das Programm "XF" beseitigt gründlich und dauerhaft den Verdruß, der aus einem mit

X-Funktionen verstopften USER-Tastenfeld erwächst, solange Sie sich auf den RUN-Modus beschränken. Im PRGM-Modus müssen Sie X-Funktionen nach wie vor auf herkömmliche Weise 'buchstabierend' eintasten. Wenn Sie also gewisse X-Funktionen einem bestimmten Programm mehrfach einfügen wollen und zur großen Gruppe der tastenfaulen Benutzer gehören, bleibt Ihnen nur der alte Weg, nämlich die jeweiligen Funktionen zuvor mit 'ASN' auf dem Tastenfeld festzunageln und dann wie gewohnt im USER-Modus ins Programm zu rufen.

Zwei Gefahrenquellen beim Umgang mit "XF" verdienen noch Beachtung. Erstens: Benutzen Sie "XF" nie bei leerem Alpha-Register mit 27 in X, sonst fegt Ihnen das zugehörige 'PCLPS' den gesamten Programmspeicher leer, das unschuldige "XF" eingeschlossen; ausgenommen natürlich, Sie hätten gerade diesen Kahlschlag im Sinn. In allen normalen Fällen müssen Sie das Alpha-Register mit dem Namen des zu löschenden Programms füllen. Wenn Sie beispielsweise alle auf "XF" folgenden Programme löschen wollen, ermitteln Sie mit 'CAT 1' den Namen des zweiten Programms, legen denselben ins Alpha-Register und führen "XF" aus. — Zweitens: Rufen Sie "XF" nicht von einem Unterprogramm aus, das selbst bereits auf zweiter oder tieferer Stufe legt, auf, m.a.W., lassen Sie "XF" nicht bei schon zwei oder mehr wartenden Rücksprungadressen ablaufen. "XF" bedient sich nämlich des Systemregisters a, welches bei entsprechendem Abstieg in Unterprogrammebenen die Rückkehradressen 3 bis 6 enthält, läßt dieses leer zurück und vereitelt daher in den genannten Fällen einen geordneten Rücklauf innerhalb des Programmstapels.

Eine Spielart von "XF" ist das folgende ebenfalls von Clifford Stern stammende Programm "EFTW" (*e*rweiterte *F*unktionen/*t*ime-Funktionen/*w*and), welches ein ganzes Register kürzer als "XF" ist. Es verweilt einen Augenblick im Alpha-Modus, um dem Benutzer Gelegenheit zu geben, eine Kette von bis zu 7 Zeichen einzutasten. Die Gebrauchsanweisung ist ansonsten dieselbe wie die für "XF" (beachten Sie, daß wegen Punkt 1 dieser Anweisung "XF" und "EFTW" nicht beide nebeneinander benutzt werden können!). Zeile 14 enthält hier die Zahl 246 + 64·n, n wieder die Zahl der eingesteckten Speichererweiterungsmodule. Außerdem kann "EFTW" die Funktion 'XYZALM' (Zahlenkode 93) nur unter Verzicht auf ein Wiederholungsintervall ausführen, weil das Register Z zum entscheidenden Zeitpunkt Null enthält.

Programmausdruck von "EFTW"
(Barcode im Anhang D)

```
01♦LBL "EFTW"      09 64              17 X<> \           25 STO c
02 RCL [           10 +               18 X<> c           26 RDN
03 CLA             11 "┣♦♦▫i♦δ♦uu→u   19 RCL [           27 ENDIR
04 STO [           12 XTOA            20 STO IND \       28 CLD
05 AON             13 CLX             21 X<> ]           29 END
06 PSE             14 502             22 CLA
07 AOFF            15 SIZE?           23 X<> [
08 CLX             16 -               24 RDN                  67 Bytes
```

Dezimalwerte der synthetischen Befehle:

Zeile 02: 144, 117
Zeile 04: 145, 117
Zeile 11: 254, 127, 0, 0, 1, 105, 0, 18, 0, 117, 117, 145, 125, 117, 166
Zeile 17: 206, 118
Zeile 18: 206, 125
Zeile 19: 144, 117
Zeile 20: 145, 246
Zeile 21: 206, 119
Zeile 23: 206, 117
Zeile 25: 145, 125

10C. Der Aufbau des X-Memorys

In diesem Abschnitt werden wir beschreiben, wie die Dateien im X-Memory angeordnet sind, und zeigen, welche Bereiche dort durch die Kartenleserfunktionen 'VER' und '7CLREG' angesprochen werden. Dann erläutern wir für Synthetiker die Einzelheiten des Aufbaus von Dateiköpfen und erklären, welche Wege beschritten werden müssen, um die in der synthetischen Programmierung auftretende so lästige Erscheinung der Normalisierung zu vermeiden.

Das X-Memory besteht aus einem, zwei oder drei zusammenhängenden Registerblöcken, deren Anzahl abhängig davon ist, wieviele X-Memorys in den Einschubbuchsen stecken. Der erste Block besteht aus den 128 Registern des X-Funktionen-Moduls, die Blöcke 2 und 3 aus je 239 Registern eines nur Speicherplatz enthaltenden X-Memory-Moduls. Dem Benutzer sind davon jeweils nur 127 bzw. 238 Register zugänglich, weil das letzte Register jedes X-Moduls dem Betriebssystem vorbehalten bleibt. Diese letzten Register, 3 Stück also an der Zahl, enthalten zwei Zeiger, welche dem Betriebssystem die Verbindung zu dem logisch (nicht physikalisch!) vorangehenden und dem logisch folgenden Modul beschreiben. Sie dienen somit dazu, die Teilstücke von solchen Dateien, die über die Modulgrenzen hinweg von einem Modul in den nächsten reichen, richtig zu verketten, weil die Reihenfolge, in der die Module mit Dateien belegt werden, davon abhängig ist, ob alle Module von Anfang an vorhanden sind oder ob sie wegen wachsenden Platzbedarfes Stück für Stück nachgesteckt werden.

Die nachstehende Abbildung zeigt die Einzelheiten des X-Memory Aufbaus. Für jene synthetisch vorgebildeten Benutzer, die sich das Abenteuer leisten wollen, wie früher durch den Hauptspeicher jetzt 'mit Flinte und Kamera durchs X-Memory' zu stöbern, sind die absoluten Registeradressen vermerkt.

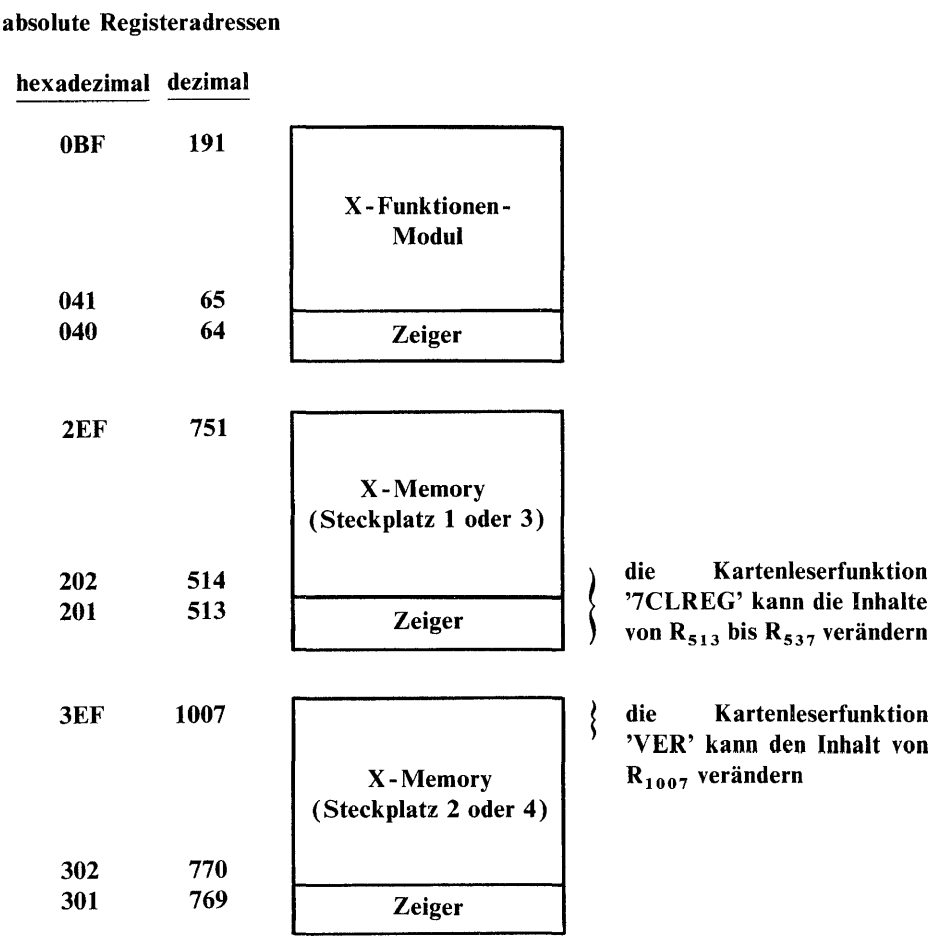


Abbildung 10.1: Gesamtaufbau des X-Memorys

Innerhalb der dem Benutzer zugänglichen X-Memory Bereiche liegen die Dateien in derselben Ordnung, in welcher sie auf 'EMDIR' hin im X-Memory Verzeichnis vorgewiesen werden. Jede Datei besitzt zwei Kopfregister, wie schon in Abbildung 2.1 (Abschnitt 2A) veranschaulicht wurde. Unmittelbar hinter dem letzten Register der letzten Datei liegt ein besonderer den endgültigen Schluß der Dateien signalisierender *Trennkode*, hexadezimal FF FF FF FF FF FF FF. Er scheidet den belegten Teil des X-Memorys vom freien Teil und teilt durch seine Lage dem Betriebssystem mit, daß der nachfolgende Rest des X-Memorys für weitere Dateien zur Verfügung steht.

Bei völlig leerem X-Memory ("DIR EMPTY") liegt der Trennkode am Kopf des X-Funktionen-Moduls. Die erste Datei, welche Sie erzeugen, besetzt die obersten Register des X-Funktionen-Moduls und schiebt den Trennkode abwärts. Nachfolgend erzeugte Dateien gelangen stets unmittelbar hinter die letzte Datei, während der Trennkode immer weiter abwärts wandert.

Unter Umständen — oder eigentlich sehr schnell, denn 'der Appetit kommt beim Essen' — reichen Sie mit den 127 Registern des ersten Blockes nicht mehr aus. Sobald dies der Fall ist, schwappt die Datei, welche keine Aufnahme mehr im ersten Modul findet, in den nächsten Modul über. Gewöhnlich ist davon zunächst der in Buchse 1 oder 3 steckende Modul betroffen und erst beim zweiten Überlauf der in Buchse 2 oder 4 steckende Modul. Ausnahmen von dieser Reihenfolge gibt es, wenn

1. überhaupt kein Modul in Buchse 1 oder 3 steckt oder
2. der in Buchse 2 oder 4 steckende Modul schon Dateien aufgenommen hat, noch bevor der letzte Modul nachgeschoben wurde.

Nach einem "MEMORY LOST" wird die 'natürliche' Ordnung (Buchse 1 oder 3 zuerst) wiederaufgenommen.

Gliederung der Kopf- und Zeigerregister

Jeder Dateikopf besteht aus zwei am Dateianfang liegenden Kopfregistern. Das erste dieser Register enthält den aus bis zu 7 Zeichen bestehenden Dateinamen. Kürzeren Dateinamen werden rechterhand soviele Leerzeichen (hexadezimal 20) angefügt, wie nötig sind, um ein ganzes Register, also 7 Bytes, mit dem Namen zu besetzen.

Das zweite Kopfregister enthält mehrere Teilstücke, die verschiedene Auskünfte über die Datei geben. Wir werden ihre Struktur mit Hilfe von Nybbles, das sind Hexadezimalziffern, beschreiben. Zwei Nybbles bilden ein Byte; 7 Bytes bilden ein Register. Das äußerste linke Nybble des zweiten Kopfregisters bezeichnet die Dateiart: 1 für Programmdateien, 2 für Datendateien und 3 für Textdateien.

Für Programmdateien lauten die 14 Nybbles

$$10\ 00\ 00\ 00\ B_1B_2\ B_3D_1\ D_2D_3,$$

worin $B_1B_2B_3$ die Anzahl der vom Programm belegten *B*ytes, das 'END' eingerechnet, und $D_1D_2D_3$ die *D*ateigröße in Registern bezeichnet. Beide Zahlen haben hexadezimale, nicht dezimale Darstellung. Ein im X-Memory liegendes Programm hat dort dieselbe Gestalt wie im Hauptspeicher, das 'END' eingeschlossen. Dem 'END' folgt dann noch ein sog. *Prüfsummenbyte*, dessen Wert sich aus der Summe aller Programmbytes modulo 256 ergibt. Dieses zusätzliche Prüfsummenbyte berücksichtigt, versteht man, daß ein Programm im X-Memory einen Platz beansprucht, der um 3 Register höher sein kann als im Hauptspeicher. Wenn ein Programm nämlich beispielsweise genau 49 Bytes im Hauptspeicher belegt, wird von 'SAVEP' dafür eine 10 Register umfassende Datei im X-Memory angelegt: 2 Kopfregister, 7 eigentliche Programmregister und 1 'angebrochenes' Register zur Unterbringung des 'überhängenden' Prüfsummenbytes.

Für Datendateien lauten die 14 Nybbles

$$2A_1\ A_2A_3\ 00\ 00\ R_1R_2\ R_3D_1\ D_2D_3,$$

worin $A_1A_2A_3$ die absolute *A*dresse des gerade vorliegenden zweiten Kopfregisters, also die 'eigene Adresse' ist, $R_1R_2R_3$ den Dateizeiger (vgl. Abschnitt 2C) bildet und $D_1D_2D_3$ wieder die *D*ateigröße darstellt. Der Dateizeiger weist auf die mit 0, 1, 2 usw. numerierten *R*egister, wobei die Zählung unmittelbar hinter dem zweiten Kopfregister beginnt.

Bei ASCII-Dateien hat das zweite Kopfregister die Gliederung

$$3A_1\ A_2A_3\ 00\ Z_1Z_2\ S_1S_2\ S_3D_1\ D_2D_3,$$

worin $A_1A_2A_3$ wieder die *A*dresse des Kopfregisters selbst ist, $Z_1Z_2S_1S_2S_3$ den aus dem *Z*eichenzeiger Z_1Z_2 und dem *S*atzzeiger $S_1S_2S_3$ zusammengesetzten Dateizeiger für Textdateien bildet (vgl. Abschnitt 3A) und $D_1D_2D_3$ abermals die *D*ateigröße darstellt.

Die Zeigerregister am Fuß jedes zusammenhängenden Blockes aus X-Memory Registern haben den folgenden Inhalt:

$$00\ 0G_1\ G_2V_1\ V_2V_3\ F_1F_2\ F_3E_1\ E_2E_3.$$

Hierin bedeuten G_1G_2 die Nummer der *g*egenwärtigen Datei (die Zählung beginnt bei 1), $V_1V_2V_3$ die absolute Adresse des letzten Registers des logisch *v*orangehenden Blockes, $F_1F_2F_3$ die absolute Adresse des ersten Registers des logisch *f*olgenden Blockes und $E_1E_2E_3$ die absolute Adresse des *e*rsten Registers im 'Eigenblock' des Zeigerregisters. Die Größe G_1G_2 wird allerdings nur im Fußregister des X-Funktionen-Moduls aufbewahrt, und eine Adresse $V_1V_2V_3$ gibt es natürlich für diesen Modul nicht (der freie Platz wird im HP-41CX zur Ablage der Nummer der Datei, die vor der gegenwärtigen Datei Arbeitsdatei war, benutzt). Ebensowenig gibt es eine Adresse $F_1F_2F_3$ für den letzten, den 3. X-Modul. Die Informationen in den Fußregistern der Module werden immer dann auf den neuesten Stand gebracht, wenn man eine Datei erzeugt, die Platz in den betreffenden Moduln beansprucht. Sofern man eine auf das X-Memory bezogene X-Funktion ausführt, obwohl noch gar keine Dateien vorhanden sind, gelangen die Adressen der Fußregister selbst in die jeweiligen $E_1E_2E_3$-Felder.

Die Nybbles der Kopf- und Zeigerregister, welche vom Betriebssystem nicht benutzt werden, brauchen nicht unbedingt auf Null zu stehen. Diese Tatsache kommt häufig synthetischer Arbeitsweise geradezu entgegen, weil man z.B. das erste Nybble eines Zeigerregisters gefahrlos auf 1 setzen kann, um den Registerinhalt als Zeichenkette abzurufen.

Dies ist die Stelle, ein Wort über den Begriff der *Normalisierung* zu verlieren. Sobald ein Datenregister ein Bitmuster enthält, das keine Zahl darstellt und das der HP-41 ebensowenig als Alpha-Kette erkennt, kann sich der Registerinhalt beim Abruf verändern. Diese meist unerwünschte Abänderung heißt Normalisierung. Sie ist ein arger Störenfried der synthetischen Programmierung und behindert diese überall dort, wo die Befehle 'RCL', 'ARCL', 'X< >' und 'VIEW', direkt oder 'IND'irekt, im Spiel sind. In der Literatur über synthetische Programmierung (vgl. Anhang C) können Sie sich erschöpfend sachkundig machen und nachlesen, wann und mit welchen Mitteln man sich dagegen wehren kann.

Unter den X-Funktionen gibt es nun glücklicherweise einige vom Typ 'SAVE' und 'GET', welche die Registerinhalte ohne Normalisierung übertragen. Auf diesem Umstand beruhen viele Anwendungen höherer synthetischen Programmierung, u.a. auch einige der in diesem Kapitel enthaltenen Programme. 'GETX', 'SAVEX', 'GETR', 'SAVER' und 'GETRX' normalisieren überhaupt nicht. Die Funktionen 'SAVERX' und 'REGSWAP' normalisieren beide lediglich die Inhalte des ersten und letzten Registers im angesprochenen Block, 'REGMOVE' sogar nur den Inhalt des höchsten der betroffenen Register.

Sobald eine Datei im X-Memory mit 'PURFL' getilgt wird, bewegen sich alle der getilgten Datei folgenden Dateien im X-Memory nach oben, um die durch die Tilgung entstandene Lücke zu schließen. War die getilgte Datei die letzte im X-Memory, findet folgerichtig keine Verschiebung von zurückgebliebenen Dateien statt. Lediglich der oben erwähnte Trennkode

wird in diesem Fall hochgerückt, und zwar so, daß er unmittelbar hinter die Datei, welche nun das Schlußlicht bildet, zu liegen kommt. Bemerkenswert ist aber, daß die Register hinter dem Trennkode *nicht* mehr gelöscht werden. Daraus folgt, daß ein einmal benutzter X-Memory Bereich ständig verschmutzt bleibt, auch wenn man noch so heftig mit 'PURFL' darin herumputzt. Nur der Zugriff auf die alten Dateitrümmer bleibt dem Benutzer versagt, es sei denn, er bedient sich synthetischer Methoden, um den hinter dem Trennkode liegenden Schutthaufen zu durchstöbern.

All diese Einzelheiten sind hauptsächlich für maturierte synthetische Programmierer von Belang, doch zeigen Ihnen die beschriebenen Verhältnisse deutlich, welche Fülle und Vielfalt von Speicherstellen und Zeigerwerten der HP-41 berücksichtigen muß – mithin auch der synthetische Wegelagerer –, wenn das X-Memory dem Benutzer die beabsichtigten Dienste erweisen soll.

Anm. d. Übers.: a) Als Musterbeispiel für die Möglichkeit, sich der Zeiger des X-Memorys durch synthetische Techniken zu bedienen, mag die folgende kleine – nur auf dem HP-41CX programmierbare – Routine, welche dem Benutzer die Frage 'Welche Datei ist Arbeitsdatei?' beantwortet, gelten:

```
01♦LBL "DAT?"
02 "0♦. "        F5 10 00 2E F0 BF
03 0
04 X<> c
05 RCL 64
06 ASTO 64
07 X<>Y
08 X<> c
09 X<>Y
10 X<> d
11 RCLFLAG
12 STO [
13 X<>Y
14 X<> d
15 2
16 AROT
17 "┣↑↑"
18 CLX
19 X<> \
20 X<> [
21 ATOX
22 ENDIRX
23 ASTO X
24 END
```

Die Routine darf nicht unterbrochen werden. (Barcode im Anhang D)

55 Bytes

Mit "DAT?" läßt sich beispielsweise die in Abschnitt 2E behauptete Leistung von 'CLFL', auch Programmdateien zur Arbeitsdatei zu machen, bequem überprüfen.

b) U.a. von Harald Schumann wurde die folgende wohl kürzeste Routine zur Totallöschung des X-Memorys entdeckt:

```
01♦LBL "XML"
02 0
03 X<> c
04 STO 64
05 X<> c
06 END
```

Die Routine darf nicht unterbrochen werden. (Barcode im Anhang D)

17 Bytes

c) Für 'X-Memory-Forscher' hier noch eine kurze Routine, mit der man – den vollen Ausbau des X-Memorys vorausgesetzt – die genaue Bauart von Dateien, die sich im Speicherbereich des X-Funktionen-Moduls befinden, sichtbar machen kann:

```
01♦LBL "HS+XFM"
02 "AUSZUG"
03 447
04 CRFLD
05 "α0♦"
06 RCL [
07 CLA
08 X<> c
09 SAVER
10 X<> c
11 END
```

F3 04 10 00

Die Routine darf nicht unterbrochen werden. (Barcode im Anhang D)

38 Bytes

Die Routine schreibt den Inhalt des gesamten *H*auptspeichers sowie den aller 127 nutzbaren Register des *X-F*unktionen-*M*oduls in eine 447 Register umfassende Datendatei des Namens "AUSZUG". Mit Hilfe des im 'Wickes' (vgl. Anhang C, Punkt 1), 2. Aufl., S. 137, vorgeschlagenen Verfahrens kann man sich anschließend, völlig unbehindert von Normalisierungsschwierigkeiten, hexadezimale Speicherauszüge aus allen diesen Registern herstellen und auf diese Weise die Kopfregister der verschiedenen Dateiarten selbständig untersuchen. Insbesondere lassen sich die zum Prüfsummenbyte (s.o.) sowie die zum Textabschluß- und zum Satzlängenbyte (Abschnitt 3A) gemachten Bemerkungen, den bewährten Lehrsatz 'Vertrauen ist gut – Kontrolle ist besser' befolgend, nachprüfen.

10D. Eine Medizin gegen die 'VER'-Wanze

Das folgende Programm "VER" ist dazu vorgesehen, an Stelle der im Kartenleser ansässigen Funktion 'VER' benutzt zu werden, um sicherzustellen, daß dem X-Memory kein Leids durch die 'VER'-Wanze angetan wird. Auch dieses Meisterstück der synthetischen Programmierung stammt aus der Hexenküche von Clifford Stern. Das Programm wird den meisten von Ihnen gute Dienste leisten, nur diejenigen, die schon einen Kartenleser der Version von 1G an aufwärts besitzen oder keinen X-Modul in den Buchsen 2 oder 4 stecken haben, können darauf verzichten. Die Version der X-Funktionen, mit der Sie arbeiten, spielt hier keine Rolle.

Im Anhang D (Barcodes) finden Sie zwei Spielarten des Programms "VER". Gewöhnlich werden Sie sich der ersten Spielart bedienen. Die zweite Version ist nur für die folgenden zwei Sonderfälle vorgesehen:

1. Sie haben einen X-Memory-Modul in der Buchse 2 oder 4 stecken, aber keinen in der Buchse 1 oder 3.
2. Sie haben einen X-Memory-Modul auf den Steckplatz 1 oder 3 gesetzt, nachdem ein auf Steckplatz 2 oder 4 befindlicher X-Memory-Modul bereits Dateien oder ein Teilstück davon aufgenommen hat und der HP-41-Frieden seitdem durch kein "MEMORY LOST" gestört worden ist. Sind beide X-Memory-Module dem Rechner zur gleichen Zeit eingesetzt worden, ist die Version 1 zuständig.

Falls Sie allein mit dem X-Funktionen-Modul (oder einem HP-41CX) arbeiten und keine

weiteren X-Memory-Module angeschlossen haben, oder wenn, dann nur einen auf Steckplatz 1 oder 3, können Sie die Kartenleserfunktion 'VER' beruhigt benutzen. Sobald Sie aber das X-Memory erweitern, wird es wohl dazu kommen, daß Sie "VER" Ihren Standard-Routinen erleichtert eingliedern, um zukünftig sorgenfrei leben zu können.

WARNUNG: Bevor Sie "VER" benutzen, gleichgültig ob Spielart 1 oder Spielart 2, stellen Sie unbedingt sicher, daß

1. wenigstens eine Tastenzuweisung, die sich nicht auf eine globale Marke bezieht, vorhanden ist, oder
2. keine Weckaufträge in den Alarmregistern lagern *und* mindestens ein freies Programmregister zur Verfügung steht (überprüfen Sie dies sicherheitshalber an Hand der bekannten PRGM-Modus Meldung "00 REG mn").

Keine der beiden Bedingungen ist schwer zu erfüllen, und jede der beiden bürgt einzeln dafür, daß "VER" seinen Auftrag fehlerfrei erfüllt.

Gebrauchsanweisung für "VER"

Nach einem 'XEQ "VER"' erscheint die Kartenleser-Aufforderung "CARD". Zugleich sehen Sie den ALPHA-Indikator eingeschaltet. Sollte das nicht der Fall sein, lag "VER" nicht im Speicher, so daß Sie mit Ihrem 'XEQ "VER"' unbeabsichtigt die Funktion 'VER' aufgerufen haben. Schieben Sie unter diesen Umständen keine Karte ein!

Drücken Sie 'R/S' oder '←', um die Anzeige von der Aufforderung "CARD" zu befreien, wenn Sie die letzte Karte mit "VER" überprüft haben. Sie erhalten danach die unbedingt zu beherzigende Mahnung "PRESS R/S".

WARNUNG: Leisten Sie der Aufforderung, "VER" noch einmal zu starten, widerstandslos Folge! Wenn Sie das nämlich versäumen, wird nicht nur der von 'VER' im X-Memory angerichtete Schaden nicht beseitigt. Es kommt noch viel schlimmer. Wichtige Systeminformationen gehen zum Teufel, so daß die gefürchteten Tastatur-Blockaden auftreten können, die meist noch schmerzlicher als das Schreckgespenst "MEMORY LOST" empfunden werden. Mit solchen Gefahren muß man aber leben, wenn man sich der hier verwendeten synthetischen Techniken bedient; kraftvolle Kämpfer im System-Dickicht des HP-41 sind diese Programme, doch gnadenlos, wenn man die von ihnen diktierten Spielregeln nicht beachtet.

Beachten Sie beim Einsatz von "VER" auch noch folgende zwei Punkte:

1. Wenn Sie den auf 'R/S' ausgeübten Tastendruck nicht ganz schnell wieder aufheben, zögert der Rechner einen Moment, weil der glaubt, Sie wollten das Programm schrittweise abarbeiten, und darum die Programmzeilennummer auszurechnen beginnt. Während dieser Verzögerung verbleibt die Mahnung "PRESS R/S" in der Anzeige, und auch der PRGM-Indikator springt derweil nicht an; es scheint so, als ob im Rechner überhaupt nichts geschähe. Das ist aber ein Irrtum, der sich u.U. als folgenschwer erweisen kann, wenn Sie 'R/S' – in guter Absicht freilich – noch rasch ein zweites Mal drücken. Warten Sie also ab. Der Rechner kommt auch ohne Ihre Nachhilfe in Fahrt. Nach endgültig beendetem Programmlauf finden Sie im X-Register i.a. einige Voll- oder anderen Fremdzeichen vor.
2. Sie dürfen "VER" als Unterprogramm aufrufen, aber nicht von einer Routine aus, die selbst bereits auf zweiter oder tieferer Unterprogrammebene liegt. Im Rückkehrstapel darf also *höchstens eine* Rückkehradresse warten, wenn Sie 'XEQ "VER"' als Programmbefehl verwenden.

Programmausdruck von "VER"
(Barcode im Anhang D)

```
01♦LBL "VER"     11 REGMOVE       21 STO IND Z     31 AOFF          41 DSE 10
02 CF 25         12 "├*"          22 DSE 10        32 X<>Y          42 STO IND 10
03 RCLFLAG       13 191           23 STO IND 10    33 SEEKPT        43 R↑
04 "×j♦á×♦0←♦x̄"  14 STO 10        24 R↑            34 RDN           44 STO 12
05 X<> \         15 R↑            25 RCL \         35 SAVEX         45 END
06 ENTER↑        16 STO 06        26 RCLPT         36 X<>Y
07 X<> c         17 "├♦♦♦ä "      27 GETX          37 STO IND 10
08 RCL 04        18 X<> ]         28 "PRESS R/S"   38 LASTX
09 "0♦. "        19 STO IND Y     29 AON           39 X=Y?
10 ASTO 63       20 X<> [         30 VER           40 STO 63                  112 Bytes
```

Dezimalwerte der synthetischen Befehle in Version 1:

Zeile 04: 252, 1, 105, 0, 19, 240, 1, 137, 0, 48, 3, 0, 2
Zeile 05: 206, 118
Zeile 07: 206, 125
Zeile 09: 245, 16, 0, 46, 240, 191
Zeile 17: 247, 127, 0, 0, 0, 22, 191, 255
Zeile 18: 206, 119
Zeile 20: 206, 117
Zeile 25: 144, 118
Zeile 30: 167, 133 (Dezimalwerte von 'VER')

Abweichungen in Version 2:

Zeile 09: 245, 16, 0, 62, 240, 191
Zeile 17: 247, 127, 0, 0, 0, 7, 223, 255

Die hinter "VER" stehende Idee ist recht einfach (was man übrigens nicht von ihrer Durchführung sagen kann, d. Übers.). Das Ziel des hinterlistigen Programms besteht darin, den Inhalt der Speicherstelle 1007 (vgl. Sie Abbildung 10.1 in Abschnitt 10C) abzurufen, alsdann die verwanzte Funktion 'VER' ganz 'scheinheilig' damit zu beauftragen, ihre Arbeit abzuleisten, so als wüßte man gar nicht, welchen Schaden sie stiftet, und hinterher völlig unbeeindruckt, sozusagen 'Hände reibend', den sicher aufbewahrten Inhalt zurückzuspeichern. Die Zeilen 02 und 03 dienen nur dazu, eine "NONEXISTENT"-Meldung zu liefern, falls der X-Funktionen-Modul nicht eingesteckt ist; auf dem HP-41CX können sie entfallen.

"VER" ändert vorübergehend das zweite Kopfregister der ersten Datei so ab, daß eine Datendatei von 4095 Registern Umfang vorgetäuscht wird. Auf diese Weise kann 'GETX' dazu benutzt werden, den Inhalt der Stelle 1007 normalisierungsfrei als gewöhnlichen Registerinhalt abzurufen. Dieser kunstvolle Trick, der von Clifford Stern ersonnen wurde, erlaubt es, auf jedes Register des X-Memorys ohne Normalisierungsgefahr zuzugreifen, und zwar sowohl mit 'SAVEX' als auch mit 'GETX'. Die Funktion 'REGMOVE' (Zeile 11) kümmert sich in "VER" um die Sicherstellung der beiden Kopfregisterinhalte der ersten Datei, so daß auch diese vor Programmende dem X-Memory unversehrt zurückgegeben werden können.

Lassen Sie sich nicht durch das Auftreten der Registeradressen 04, 06, 10, 12 und 63 täuschen. Die Register, welche Sie beim herkömmlichen Programmieren darunter verstehen,

sind nämlich gar nicht angesprochen. Wegen der durch "VER" bewirkten Umsetzung eines internen Zeigers auf die Speicherstelle 01 des HP-41 (das ist in Wirklichkeit Stapelregister Z) sprechen Befehle wie 'DSE 10' oder 'STO 06' vielmehr die sog. Systemregister an. Der Befehl 'ASTO 63' (Zeile 10) bringt aus demselben Grunde auch gar nichts in das Hauptspeicherregister R_{63}, sondern in das Fußregister des X-Funktionen-Moduls, um den dort befindlichen Zeiger für die gegenwärtige Datei (in Abschnitt 10C mit G_1G_2 bezeichnet) auf 01 zu setzen.

10E. Ein Heilmittel gegen die 'PURFL'-Wanze

Das hier vorgestellte Programm "PFF" (*p*urge *f*ile *f*ix) ist insbesondere für Besitzer von X-Funktionen der Version 1B gedacht. Es erlaubt ihnen, Fehler gut zu machen, die dadurch entstehen, daß man eine auf eine Arbeitsdatei Bezug nehmende X-Funktion aufruft, obwohl gar keine Datei gegenwärtig ist. Diese unheilschwangere Lage kann nach einem 'PURFL'-Befehl entstehen (vgl. Abschnitt 1F). Der genaue Grund für den Verlust des Zugriffes auf das X-Memory ist die Tatsache, daß ohne gegenwärtige Datei der Kopfinhalt des X-Memorys durch die ins Leere greifende X-Funktion mit dem in Abschnitt 10C erwähnten Trennkode überschrieben wird. Der fälschlich an den Anfang des X-Memorys gesetzte Kode macht den HP-41 Glauben, das X-Memory sei leer.

Die Berichtigung ist einfach. Das Programm "PFF" legt lediglich den Dateinamen, der ins Kopfregister des X-Memorys, also in die Speicherstelle 191_{10} (vgl. Abbildung 10.1 in Abschnitt 10C) gehört, zurück. Weil andere X-Memory-Register von der 'PURFL'-Wanze nicht in Mitleidenschaft gezogen werden, gibt es auch keinen Anlaß, an irgendeiner anderen Stelle im X-Memory etwas zu bereinigen.

Das Programm "PFF", welches die vorstehend beschriebene Reparatur ausführt, stammt von Clifford Stern. Es bedient sich synthetischer Techniken, weil man auf gewöhnlichem Wege nichts in das reparaturbedürftige Register schreiben kann.

Da die 'PURFL'-Wanze nur in der Version 1B des X-Funktionen-Moduls ihr Unwesen treibt, können Besitzer der Version 1C oder des HP-41CX diesen Abschnitt überschlagen.

Gebrauchsanweisung für "PFF"

1. Prüfen Sie zunächst, ob Ihr X-Memory-Verzeichnis tatsächlich "DIR EMPTY" behauptet. Wenn es dies tut und Sie weder vorsätzlich alles gelöscht haben noch Opfer eines "MEMORY LOST" waren, wissen Sie, daß das Kopfregister von der 'PURFL'-Wanze gestochen wurde, also einen falschen Inhalt hat. Mit "PFF" können Sie dann den Schaden aus der Welt schaffen, vorausgesetzt allerdings, daß Sie inzwischen nicht weitere Dateien im X-Memory erzeugt haben. Sollte das der Fall sein, sind alte Dateien überschrieben worden, und es läßt sich beim besten Willen nichts mehr in Ordnung bringen.
2. Legen Sie den Namen der ersten Datei des X-Memorys ins Alpha-Register. Bemerkung: Dabei brauchen Sie übrigens nicht einmal ein lückenhaftes eigenes Gedächtnis als Hindernis anzusehen, denn jede beliebige Alpha-Kette aus höchstens sieben Zeichen (ausgenommen selbstredend FF FF FF FF FF FF FF), z.B. "WANZMED", erfüllt den beabsichtigten Zweck. Sobald "PFF" das Verzeichnis wieder instandgesetzt hat, können Sie "WANZMED" mit 'GETP' in den Hauptspeicher holen, um dort nachzusehen, wie der eigentliche Programmname lautete – er muß ja nicht unbedingt mit dem Dateinamen übereingestimmt haben (vgl. Abschnitt 1D) – und

denselben dann, natürlich auch mit "PFF", ins Kopfregister des X-Memorys schreiben. Im ungünstigsten Fall war die erste Datei eine Daten- oder Textdatei, deren Name Ihnen endgültig entfallen ist und der sich auch nicht mehr anderweitig auftreiben läßt. In diesem Fall müssen Sie den für die Reparatur gewählten Namen beibehalten. Der Vollständigkeit halber sei noch erwähnt, daß Sie mit "PFF" nicht die "DUP FL"-Warnung erhalten können und somit u.U. unerwünschte Doppelbenennungen ins X-Memory bekommen.

3. Rufen Sie "PFF" auf. Das Programm endet mit dem Befehl 'EMDIR', um erstens eine Arbeisdatei dingfest zu machen und zweitens zu zeigen, daß sich das X-Memory tatsächlich wieder fügsam zeigt und dem Benutzer den Zugriff auf die vorhandenen Dateien gestattet. Sie können das Vorweisen der Einträge selbstverständlich unterbrechen, denn schon das Erscheinen der ersten Datei zeigt ja, daß alles wieder seine Ordnung hat.

WARNUNG: "PFF" darf einzelschrittweise abgearbeitet werden, doch müssen Sie, sobald Zeile 08 ausgeführt worden ist, auch unbedingt bis zur Zeile 11 weiterschreiten. Wird das Programm nämlich zwischen den Zeilen 08 und 11 schnöde verlassen, steht das Schreckgespenst "MEMORY LOST" vor der Tür. Bei vorschriftsmäßiger Verwendung von "PFF" jedoch arbeitet das Programm völlig einwandfrei.

Programmausdruck von "PFF"
(Barcode im Anhang D)

```
01♦LBL "PFF"
02 "⊢      "      Zeile 02 hängt 6 Leerzeichen ('SPACE') an.
03 7
04 AROT
05 RCL c
06 RCL [
07 "▪i↓♦"
08 ASTO c
09 STO 00
10 X<>Y
11 STO c
12 CLST
13 EMDIR
14 CLD
15 END                                          41 Bytes
```

Dezimalwerte der synthetischen Zeilen:

Zeile 05: 144, 125
Zeile 06: 144, 117
Zeile 07: 245, 1, 105, 11, 242, 0
Zeile 08: 154, 125
Zeile 11: 145, 125

10F. Die Ausführung von Programmen *im* X-Memory

Wenn eine Programmdatei ganz innerhalb des Blockes der 127 Speicherregister des X-Funktionen-Moduls liegt, ist es möglich, das Programm laufen zu lassen, ohne es erst mit 'GETP' in den Hauptspeicher zu rufen. Natürlich muß man synthetische Methoden anwenden, um dies zu bewirken, doch nicht einmal besonders phantasiereiche. Es ist lediglich nötig, den aus 2 Bytes bestehenden Adreßzeiger, der ganz rechts im Systemregister b liegt und stets auf den gegenwärtigen Programmbefehl weist (sichtbar im PRGM-Modus), so zu verändern, daß er auf die Befehle des im X-Memory liegenden Programms zeigt.

WARNUNG: Bevor Sie ein Programm im X-Memory laufen lassen, müssen Sie unbedingt sicherstellen, daß alle lokalen 'GTO's und 'XEQ's, also die auf lokale Marken 00 – 99, A – J oder a – e zielenden Sprunganweisungen, *kompiliert* sind. Sie müssen also dafür sorgen, daß nach dem letzten Edieren oder 'PACK'en alle 'GTO'- und 'XEQ'-Befehle des genannten Typs wenigstens einmal ausgeführt wurden. Andernfalls kann es geschehen, daß die Ausführung des im X-Memory liegenden Programms dessen Prüfsumme (vgl. Abschnitt 10C) verletzt, was das nächste auf dieses Programm bezügliche 'GETP' an der Fehlermeldung "CHKSUM ERR" scheitern lassen würde.

BEMERKUNG: Sollten Sie die Absicht haben, ein bestimmtes Programm sowieso nur im X-Memory laufen zu lassen, ist die aus voreiliger Programmauslagerung resultierende Gehorsamsverweigerung von 'GETP' ohne größere Bedeutung für Sie. Sie dürfen großzügig darüber hinwegsehen. Außerdem gibt Ihnen das in Abschnitt 10I vorgestellte Programm "RPF" (*r*etrieve *p*rogram *f*ile) immer noch die Möglichkeit, auch bei "CHKSUM ERR" ein Programm unter Verzicht auf 'GETP' oder 'GETSUB' wieder in der Hauptspeicher zu holen.

Wenn Sie ein bestimmtes Programm häufig mit 'GETP' aus dem X-Memory holen müssen, insbesondere, wenn dies ständig programmgesteuert zu geschehen hat, ist es wesentlich günstiger, das in Rede stehende Programm nur mit kompilierten 'GTO's und 'XEQ's im X-Memory abzulegen. Um Ihnen den Sinn dieser Empfehlung klar zu machen, ist hier eine Abschweifung auf den Begriff der *Kompilation von Sprunganweisungen* angebracht.

Bei der ersten Ausführung eines lokalen 'GTO'- oder 'XEQ'-Befehles wird die Sprungrichtung und die Sprungdistanz vom HP-41 ermittelt und die Information darüber *in den Sprunganweisungen selbst* abgelegt. Diese Ablage heißt Kompilation. Auf diese Weise bleibt dem Rechner bei der nächsten Begegnung mit einem solchen mit Information angereicherten 'GTO' oder 'XEQ' die erneute, u.U. zeitraubende Suche nach der angesteuerten lokalen Marke erspart. Die Ablage von Sprunginformationen in den Sprungbefehlen ist es nun, die das bestehende Prüfsummenbyte einer Programmdatei ungültig macht. Wenn die 'GTO's und 'XEQ's jedoch schon vor Überstellung des Programms ins X-Memory kompiliert worden sind, kann sich aus der Ausführung des Programms im X-Memory keine abweichende Prüfsumme mehr ergeben, denn die Byte-Kombinationen der Sprungbefehle erfahren in diesem Fall ja keine Änderung mehr. Indirekte und globale Sprungbefehle (letztere sind die Sprünge auf globale Marken des Katalogs 1) – 'GTO's und 'XEQ's – werden nicht kompiliert und sind daher in dem hier besprochenen Zusammenhang keiner besonderen Fürsorge bedürftig. Bleibt noch zu vermerken, daß selbst bei Verzicht auf Programmläufe im X-Memory das Hereinholen kompilierter Programme wegen des oben erwähnten Fortfalls erneuter Suche nach lokalen Marken ersichtlich vorteilhafter ist als das Arbeiten mit Programmen, die unkompiliert im X-Memory liegen.

Alle einmal ermittelten Sprunginformationen werden in dem Moment, in welchem Sie Ihr Programm ändern, also Einfügungen oder Löschungen vornehmen, ungültig und darum

getilgt, auch dann, wenn sich die Sprungdistanzen dabei gar nicht ändern würden. 'PACK'-Befehle lassen die Informationen ebenfalls verloren gehen, es sei denn, das Programm ist schon ge'PACK't. Darum hier noch einmal eine genaue Anweisung, die zu beachten empfohlen wird, wenn Sie sichergehen wollen, daß ein Programm nur mit kompilierten 'GTO's und 'XEQ's in Ihr X-Memory gelangt, weil Sie es dort sorgenfrei laufen lassen möchten.

1. Das Programm muß im Hauptspeicher liegen. Sollte es sich schon im X-Memory befinden, ist es zunächst mit 'GETP' von dort zu holen.
2. Bevor Sie beginnen, die 'GTO's und 'XEQ's zu kompilieren, müssen Sie noch 'PACK'en, und zwar mit 'GTO..', wenn das zu behandelnde Programm das letzte im Hauptspeicher ist. Besitzt es bereits ein eigenes 'END', reicht 'PACK'. Jede Zeile, die ein lokales 'GTO' oder 'XEQ' enthält, muß nun in folgender Weise behandelt werden: 'GTO.mnl' auf die betreffende Zeile; dann 'SST' im RUN-Modus (nicht im PRGM-Modus!), um den Sprungbefehl auszuführen. (Sie drücken die Taste 'SST' zu diesem Zweck solange, bis der Sprungbefehl in der Anzeige erscheint und lassen dann los, bevor "NULL" kundtut, daß Sie zu lange gezögert haben. Danach ist der Sprungbefehl kompiliert. Sie können sicherheitshalber mit einem unmittelbar anschließenden 'SST' derselben Art überprüfen, ob Sie auch wirklich auf der angesteuerten Marke gelandet sind, mithin die Kompilation tatsächlich erfolgt ist.) Die beiden Befehle 'GTO.mnl' und 'SST' müssen für jedes lokale 'GTO' und 'XEQ' ausgeführt werden. Hinweis: Ein Programm, in dem sämtliche Sprungbefehle mindestens einmal erreicht werden, wenn es abläuft, kann natürlich einfach aufgerufen werden, um die vollständige Kompilation zu bewirken. In diesem Fall bietet ein Programmlauf offenkundig mehr 'Kompilationsgewißheit' als die 'SST'-Verarbeitung von Hand. Doch befleißigen Sie sich eines gesunden Mißtrauens: ein Programm erreicht nämlich sehr schnell jenen Grad von Verwicklung, bei dem man nicht mehr ohne weiteres übersehen kann, ob ein einfacher Lauf garantiert alle Sprungbefehle berührt, insbesondere dann, wenn Testbedingungen im Spiel sind.
3. Nach erfolgter Kompilation bringen Sie das mit den Sprunginformationen 'gesättigte' Programm unter Verwendung von 'SAVEP' ins X-Memory.

Das Programm "EXM" (*e*xecute *X*-*m*emory) bedient sich nur eines einzigen synthetischen Befehls: 'ASTO b'. Er wird dazu benutzt, ein im Alpha-Register liegendes Zeichen in den im Systemregister b angesiedelten Adreßzeiger zu übertragen.

"EXM"-Beispiel

Nehmen Sie an, die im X-Memory an erster Stelle liegende Datei sei unser Programm "JNX" aus Abschnitt 1A. Mit Hilfe von "EXM" können Sie "JNX" ausführen, ohne es in den Hauptspeicher zu holen. Sie brauchen nur die beiden Stapelregister Y und X mit den benötigten Startwerten n und x zu beschicken und dann 'XEQ "EXM"' aufzurufen. Alsbald erscheint das Ergebnis in X. Mit 'GTO..', mit 'CAT 1' oder mit einem auf eine Marke des Katalogs 1 zielenden 'GTO' oder 'XEQ' gelangen Sie anschließend in den Hauptspeicher zurück.

Die nachstehende Form von "EXM" erlaubt es nur, die erste Datei im X-Memory anzuspringen. Wenn Sie jedoch auf Grund genauen Nachrechnens die absolute Adresse des zweiten Kopfregisters einer beliebigen anderen Programmdatei kennen, können Sie auch das dort lagernde Programm im X-Memory laufen lassen. Sie müssen dafür lediglich die die Ansprung-

stelle bestimmende Zeile 02 von "EXM" entsprechend ändern. Doch, um es noch einmal deutlich auszusprechen: die Programmdatei muß in jedem Fall vollständig innerhalb des X-Funktionen-Moduls liegen; sie darf keinesfalls bis in den nächsten X-Modul hineinragen.

Programmausdruck von "EXM"
(Barcode im Anhang D)

```
01♦LBL "EXM"
02 CLA
03 190
04 XTOA
05 RDN
06 ASTO b      (dezimal: 154, 124)
07 END                              19 Bytes
```

[Anm. d. Übers.: a) Die Zeilen 02 – 05 können insgesamt zu der synthetischen Textzeile F1 $\overline{\text{BE}}$ zusammengefaßt werden. – b) Unter gewissen einschränkenden Voraussetzungen läßt sich eine beliebige Programmdatei völlig programmgesteuert zur ersten Datei des X-Memorys machen. Die folgende kleine nur auf dem HP-41CX programmierbare Routine "DATROT" erwartet den Namen einer Programmdatei im Alpha-Register und bringt diese dann an die erste Stelle im X-Memory.

```
01♦LBL "DATROT"          14♦LBL 14
02 ASTO 00               15 32
03♦LBL 00                16♦LBL 13
04 1                     17 POSA
05 ENDIRX                18 X<0?
06 XEQ 14                19 RTN
07 CLST                  20 AROT
08 ASTO X                21 ATOX
09 X=NN?                 22 GTO 13
10 STOP                  23 END
11 GETP
12 SAVEP
13 GTO 00                           48 Bytes
```

Folgende Voraussetzungen müssen dazu allerdings erfüllt sein: 1. Im X-Memory dürfen nur Programmdateien liegen. 2. Programmnamen und Dateinamen müssen für alle Dateien übereinstimmen. 3. Die Programmnamen dürfen keine Leerstellen enthalten. 4. Jedes der Programme muß einzeln Platz im Hauptspeicher finden, damit auf Zeile 12 kein Fehlerstop ausgelöst wird.]

Gebrauchsanweisung für "EXM":

1. Stellen Sie zunächst sicher, daß die Datei, welche das im X-Memory auszuführende Programm enthält, auch wirklich die erste Datei im X-Memory-Verzeichnis ist. Sollte das nicht der Fall sein, müssen Sie die absolute Adresse ihres zweiten Kopfregisters

bestimmen. Dazu ziehen Sie von 190 die Summe der von den voranstehenden Dateien besetzten Register ab. Beachten Sie dabei, daß jede Datei 2 Register mehr benötigt, als im Verzeichnis vorgewiesen wird. Setzen Sie dann in Zeile 02 von "EXM" die von Ihnen ermittelte Zahl ein. (Oder bedienen Sie sich der vorstehend vorgeschlagenen Routine "DATROT".)

2. Vergewissern Sie sich davon, daß das Programm, welches im X-Memory laufen soll, auch ganz innerhalb des X-Funktionen-Moduls liegt. Um dessen sicher zu sein, brauchen Sie nur die Summe aller Register festzustellen, die von dem auszuführenden Programm und den ihm voranstehenden Dateien belegt wird. Denken Sie auch hier wieder daran, je Datei 2 Kopfregister dazuzurechnen. Insgesamt dürfen Sie dabei nicht auf mehr als 127 Register kommen.
3. Lassen Sie gemäß den oben gegebenen Ratschlägen Programme, die im X-Memory laufen sollen, erst dann dorthin entweichen, wenn alle lokalen 'GTO's und 'XEQ's kompiliert sind, damit Sie später nicht vergeblich 'GETP' befehlen. (Im Notfall müssen Sie auf "RPF", Abschnitt 10I, zurückgreifen.)
4. Laden Sie die Stapelregister, sofern das aufzurufende Programm darin Eingabewerte erwartet. Es stehen für diesen Zweck allerdings nur die 3 Register X, Y und Z zur Verfügung. Das Alpha-Register kann für Starteingaben überhaupt nicht verwendet werden, weil es von "EXM" gelöscht wird.
5. Führen Sie 'XEQ "EXM"' aus. Von Zeile 06 aus erfolgt ein verzögerungsfreier Sprung auf die erste Zeile des im X-Memory wartenden Programms, das unmittelbar hinter der in Zeile 02 von "EXM" niedergelegten Adresse beginnt. (Hinweis: Wenn Sie "EXM" das erste Mal ausführen, können Sie sich mit 'SST' zur Ansprungstelle im X-Memory vortasten, um danach erst mit 'R/S' den Programmlauf zu starten. Sie wissen dann, ob Ihre vorangegangenen Adreßberechnungen gestimmt haben und "EXM" tatsächlich zur richtigen Stelle im X-Memory führt.)

Anm. d. Übers.: Man kann ohne Schwierigkeiten von einem im X-Memory liegenden Programm "P" programmgesteuert zu "EXM" in den Hauptspeicher *zur*ückspringen, indem man "P" mit der Befehlsfolge 'SF 01, GTO "ZUR"' abschließt und "EXM" folgendermaßen umgestaltet:

```
01♦LBL "E-M"        (Barcode im Anhang D)
02 CF 01
03♦LBL "ZUR"
04 FS?C 01
05 STOP
06 " "              F1 BE (gemäß obigem Vorschlag)
07 ASTO b
08 END                                  26 Bytes
```

Dann können Sie in gewohnter Weise "P" mit einem einfachen 'R/S' in die nächste Runde schicken.

WARNUNG: Benutzen Sie "EXM" nicht für ein Programm, das die Funktion 'PSIZE' enthält. Jedesmal nämlich, nachdem 'PSIZE' die zur Verfügung stehende Datenregisteranzahl geändert hat, wird der Adreßzeigerwert den neuen Speicherverhältnissen angepaßt, um der Tatsache Rechnung zu tragen, daß sich dadurch auch alle Programme im Speicher verschoben haben. Weil nun Ihr Programm im X-Memory von einer solchen Verschiebung gar nicht betroffen ist, der Adreßzeiger hingegen trotzdem so behandelt wird, als gelte die Verschiebung

auch für dieses Programm, gibt es unerwünschte Sprünge, die zu schlimmen Fehlern führen können. Einzig dann, wenn 'PSIZE' keine Änderung der Datenregisteranzahl bewirkt, weil die Eingabe für diesen Befehl mit der vorhandenen Anzahl übereinstimmt, kann nichts Unangenehmes geschehen.

10G. Aufheben und Wiederbeleben von Tastenzuweisungen des USER-Modus

Zu den Eigenschaften, die der HP-41 und der HP-67 notwendigerweise gemeinsam haben, damit man vorhandene Programme vom HP-67 auf den HP-41 übertragen kann, gehört die Möglichkeit, mit den beiden obersten Tastenreihen insgesamt 15 lokale Marken ('A' bis 'J' und 'a' bis 'e') ansprechen zu können, so daß die hinter den jeweiligen Marken stehenden Programmteile bequem im USER-Modus zu erreichen sind. Doch tritt diese nützliche Eigenschaft leider dann nicht zutage, wenn den genannten Tasten vorrangig zu bedienende globale Marken oder System-Funktionen zugewiesen sind. So mancher Benutzer hat schon zuversichtlich mit der automatischen Zuweisung der lokalen Marken gerechnet und stieß dann unerwartet auf globale Marken oder System-Funktionen. Man drückt voller Vertrauen 'LOG', um 'LBL D' aufzurufen, und findet sich unversehens in ganz andere Speicherbereiche geworfen. Es wäre zweifellos vorteilhaft, vorrangige Zuweisungen vorübergehend aufheben und nach beendeter Tätigkeit wieder zurückholen zu können.

Einmal mehr ist es die synthetische Programmierung, welche uns aus der Not zu befreien vermag. Ein sehr kurzes, aus der Feder von Tapani Tarvainen stammendes synthetisches Programm, "SK" (*s*uspend *k*ey assignments) genannt, hebt vorübergehend alle vorrangigen Tastenzuweisungen – System-Funktionen und Benutzerprogramme – auf.

Ein zweites Programm, "RK" (*r*eactivate *k*ey assignments), ebenfalls von Tapani Tarvainen geschrieben, erweckt die schlafenden Zuweisungen wieder. In "RK" wird ein 'GETP' tätig, das sich auf eine besondere synthetische Programmdatei bezieht. Diese synthetische Datei muß zuvor im X-Memory mit einem im nächsten Abschnitt besprochenen Programm "IN" (*in*itialisieren) angelegt werden. Tatsächlich ist es so, daß Sie die Wiederbelebung der Tastenzuweisungen durch jeden auf eine *beliebige* Programmdatei angesetzten 'GETP'-Befehl herbeiführen können. Beim Hereinholen eines Programms in den Hauptspeicher nämlich erweckt der HP-41 grundsätzlich alle schlafenden Zuweisungen zu neuem Leben, gleichgültig von wo und auf welche Weise ein Programm geholt wird: von Magnetkarten, mittels Lesestift, von einem Massenspeicher oder aus dem X-Memory. Der Vorteil des Gebrauches von "RK" liegt darin, daß das letzte Programm des Katalogs 1 dabei nicht überschrieben wird, auch wenn es nur das ständige '.END.' und nicht ein eigenes 'END' als letzte Zeile besitzt. Jede Art von 'GETP'-Befehl überschreibt ja bekanntlich das letzte Programm im Katalog 1 (vgl. Abschnitt 1E). Ausgenommen, Sie wollen eine solche Überschreibung ausdrücklich erwirken, sollten Sie daher "RK" statt 'GETP' wählen, wenn die schlummernden Zuweisungen wieder munter gemacht werden sollen.

Anm. d. Übers.: Vom HP-Club Austria stammt folgender besonders elegante Tip, die Zuweisungen aufzuheben und wiederzubeleben: Aufheben mit 'CLX, STO e, STO ⊢'; wiederbeleben mit 'GTO . . , CLA, PCLPS'.

10H. Aufbewahren von Tastenzuweisungen im X-Memory

Der Kartenleser HP 82104A erlaubt es Ihnen, mit der Funktion 'WSTS' die Information über die getätigten Tastenzuweisungen auf Magnetkarten aufzuzeichnen. Dadurch ist es ein leichtes, verschiedene Gruppen von Tastenzuweisungen auf Karten bereitzuhalten. Sie können gewissermaßen eine 'Bibliothek' von Tastenzuweisungen einrichten. Wird eine bestimmte Gruppe von Tastenzuweisungen benötigt, braucht man nur die entsprechende Magnetkarte einzulesen. (Es sei an dieser Stelle daran erinnert, daß dies alles nur die Zuweisungen von System-Funktionen und Funktionen aus Moduln und Peripheriegeräten betrifft; Zuweisungen von Benutzerprogrammen werden in den jeweiligen globalen Marken selbst notiert.)

Mit Hilfe der synthetischen Programmierung läßt sich auch das X-Memory als Bibliothek für Tastenzuweisungen benutzen. Die nachstehend besprochenen Programme "SAVEK" (*save* key assignments) und "GETK" (*get* key assignments) stammen von Tapani Tarvainen und sind von Clifford Stern, der u.a. den hinter 'LBL 04' stehenden Programmteil hinzugefügt hat, verbessert worden. "SAVEK" legt die Information über die Tastenzuweisungen ins X-Memory, und "GETK" holt sie von dort zurück. Anders als vorangegangene Fassungen arbeiten diese Lesarten hier auch dann völlig einwandfrei, wenn der Time-Modul oder Moduln, die I/O-Puffer anlegen, eingesteckt sind. Die Dienstprogramme "SK" und "RK", über deren Sinn und Wirkungsweise im vorangegangenen Abschnitt 10G gesprochen wurde, sind Teil der von "SAVEK" angeführten Programmgruppe.

HINWEIS: Bevor Sie "RK" oder "GETK" das erste Mal aufrufen, müssen Sie das Programm "IN" (*in*itialisieren), welches weiter unten ebenfalls abgedruckt ist, laufen lassen. Sowohl "RK" als auch "GETK" enden mit einem 'GETP', wodurch die ruhenden Tastenzuweisungen wieder betriebsfähig werden (vgl. Abschnitt 10G). Eine wundersame, von Tapani Tarvainen ersonnene Methode benutzt eine synthetische Programmdatei als Spielball für den 'GETP'-Befehl. Dabei wird die gewöhnlich von 'GETP' veranlaßte Überschreibung des letzten Programms im Hauptspeicher vermieden. Die synthetische Datei hat den entlegenen Namen " " (eine Leerstelle), der höchst unwahrscheinlich anderweitig benutzt wird, und eine Dateigröße von *Null* Bytes. Das Programm "IN", eine esoterische Schöpfung von Clifford Stern, erzeugt die synthetische Datei völlig programmgesteuert im X-Memory.

WARNUNG: Beachten Sie die in Abschnitt 10D für die Benutzung von "VER" geforderten Voraussetzungen. Nur wenn wenigstens eine der beiden dort ausgesprochenen Bedingungen erfüllt ist, dürfen Sie 'XEQ "IN"' ausführen, um die von "RK" und "GETK" benötigte Programmdatei anzulegen. Ist dies erst einmal geschehen, brauchen Sie "IN" solange nicht mehr aufzurufen, als die Kunstdatei " " im X-Memory verbleibt. Um sicher zu sein, daß alles glatt verlaufen ist, sollten Sie nach dem Tätigwerden von "IN" Ihr X-Memory-Verzeichnis durchblättern. Wenn Sie unter den Einträgen " P001" entdecken, ist alles in bester Ordnung.

Gebrauchsanweisung für "SAVEK" und "GETK"

1. Oberhalb von "SAVEK" muß es wenigstens ein 'END' im Katalog 1 geben. Wenn Sie diese Voraussetzung mißachten, steht "MEMORY LOST" drohend vor der Tür.
2. Rufen Sie, falls das noch nicht geschehen ist, "IN" auf, um die vorstehend beschriebene Programmdatei " " der Größe Null ins X-Memory zu setzen.
3. Um die vorhandenen Tastenzuweisungen ins X-Memory zu bringen, müssen Sie das Alpha-Register mit einem von Ihnen zu wählenden Dateinamen füllen und 'XEQ

"SAVEK"' aufrufen. Sollten Sie auf Zeile 43 die Meldung "NO ROOM" erhalten, sind die üblichen Maßnahmen zu treffen: überflüssige oder derzeit entbehrliche Dateien tilgen. Danach sind Sie gehalten, wieder ganz von vorn anzufangen, also abermals den zukünftigen Dateinamen ins Alpha-Register zu setzen und erneut "SAVEK" aufzurufen. Wenn "SAVEK" den Lauf beendet hat, liegt in X die Dateigröße der Datendatei, welche Ihre Tastenzuweisungen aufgenommen hat.

4. Für den Gebrauch von "GETK" müssen Sie, wie nicht anders zu erwarten, das Alpha-Register mit dem Namen der von "SAVEK" angelegten Datei laden. Dann führen Sie 'XEQ "GETK"' aus. Dabei kann es geschehen, daß Sie — hier gewiß überraschend — auch wieder die Meldung "NO ROOM" entgegengehalten bekommen. Diesmal kann sie erstens durch Zeile 72 hervorgerufen werden und besagt an dieser Stelle, daß es keine freien Register mehr im Hauptspeicher gibt. "GETK" erfordert aber das Vorhandensein mindestens eines freien Registers unterhalb des '.END.', und die auf Zeile 72 eingebaute Fehlerfalle zwingt den Benutzer, dieses Register zur Verfügung zu stellen, wenn es fehlt. Sie müssen also im Meldungsfalle entweder die Datenregisteranzahl um 1 heruntersetzen oder ein Programm löschen. "NO ROOM" kann aber zweitens auch auf Zeile 82 gemeldet werden, und zwar aus demselben Grunde wie auf Zeile 72 ('PSIZE' nicht ausführbar), jetzt jedoch mit der Absicht, den Benutzer darauf aufmerksam zu machen, daß *für die Übernahme der Tastenzuweisungsdaten aus dem X-Memory* nicht mehr genügend freie Register unterhalb des '.END.' zur Verfügung stehen (nach dem Übertragen der Information ins X-Memory könnten die Tastenzuweisungen ja gelöscht und neue Programme in den Hauptspeicher geschrieben worden sein, so daß die zurückkehrende Zuweisungsinformation im Hauptspeicher plötzlich 'vor verschlossener Tür steht'). Wenn Sie eine Meldungsunterbrechung auf Zeile 82 bekommen, tasten Sie 'X<>Y, —'. Das Ergebnis ist die Zahl, der Sie die Meldung "00 REG mn" über die freien Speicherregister anpassen müssen, und zwar durch entsprechende Verminderung der Datenregisteranzahl oder durch das Löschen von Programmen. (Sie können übrigens bereits an dieser Stelle die Zuweisungen globaler Marken, die durch das 'CLKEYS' in Zeile 78 im Gegensatz zu den dadurch total gelöschten Zuweisungen von Funktionen der Kataloge 2 und 3 nur *stillgelegt* worden sind — ihre Tastenzuweisungsbytes 'leben' nämlich nach wie vor —, wiedergewinnen, indem Sie einfach 'XEQ "RK"' ausführen.) Gleichgültig ob eine Unterbrechung durch Zeile 72 oder 82 ausgelöst wurde, in jedem Fall muß das Alpha-Register wieder mit dem Dateinamen geladen und 'XEQ "GETK"' neu ausgeführt werden.

Beachten Sie, daß "SAVEK" nur die Information über Zuweisungen von Funktionen der Kataloge 2 und 3, also die Inhalte der sog. Tastenzuweisungsregister ins X-Memory bringt, nicht hingegen die Information über die Zuweisungen globaler Marken. "GETK" holt die Zuweisungsdaten aus dem X-Memory in die Tastenzuweisungsregister zurück. Wegen der z.T. recht verwickelten Zusammenhänge kann es durch zwischenzeitlich getätigte Zuweisungen globaler Marken nach "GETK" zu Zusammenstößen zwischen verschiedenen Zuweisungen kommen, die dazu führen, daß trotz richtigen Rückrufes der Information aus dem X-Memory durch "GETK" globale Marken an unerwarteter Stelle bevorzugt werden.

Beachten Sie ferner, daß "SAVEK"/"GETK" und "SK"/"RK" zusammengehörige Paare sind. "SAVEK" verändert die Inhalte der Systemregister ⊢ und e nicht, "SK" dagegen sehr wohl. "SAVEK" dient zur längerfristigen Ablage einer geschlossenen Gruppe von Tastenzuweisungen im X-Memory, wobei keine der Zuweisungen aufgehoben wird, "SK" dagegen zur kurzfristigen Aufhebung aller Zuweisungen allein für die Zwecke des Zugriffs auf lokale Marken im

USER-Modus, wobei keine Zuweisungsinformation irgendwohin übertragen wird.*)

Besitzer eines PPC ROMs (vgl. Anhang C) können die Zeilen 46 – 62 löschen und die Aufrufe 'XEQ 04' in den Zeilen 04 und 86 durch 'XROM "E?"' ersetzen.

Für den Fall, daß Sie das 'RTN' in Zeile 01 stutzig macht, hier die Erklärung: Wenn die von "SAVEK" angeführte Programmgruppe die letzte im Hauptspeicher ist, also '.END.' als letzte Zeile besitzt, überträgt das 'GETP' in Zeile 105 die Programmausführung auf Zeile 01, so daß ohne 'RTN' der Programmlauf von vorn begänne.

Programmausdruck von "SAVEK"/"GETK"/"RK"/"SK"
(Barcode im Anhang D)

```
01 RTN

02♦LBL "SAVEK"
03 RCL [
04 XEQ 04
05 193
06 X>Y?
07 RTN
08 SF 10

09♦LBL 01
10 -
11  E3
12 /
13 "8"
14 RCL [
15 XEQ 10
16 SIGN

17♦LBL 02
18 RDN
19 RCL IND Y
20 ""
21 X<> [
22 "⊢*"
23 X<> \
24 X≠Y?
25 GTO 03
26 ARCL c
27 X<> \
28 STO IND Z
29 FC? 10
30 SAVEX
31 ISG Z
32 GTO 02

33♦LBL 03
34 X<> L
35 X<> c
36 R↑
37 CLA
38 STO [
39 R↑
40 INT
41 FC?C 10
42 RTN
43 CRFLD
44  E
45 GTO 01

46♦LBL 04
47 RCL c
48 "*"
49 X<> [
50 STO \
51 ASHF
52 RDN
53 ALENG
54 8
55 Y↑X
56 ATOX
57 *
58 512
59 MOD
60 ATOX
61 +
62 RTN

63♦LBL 10
64 RCL c
65 "ê×iµx̄♦"
66 ASTO c
67 RTN

68♦LBL "GETK"
69  E
70 SIZE?
71 +
72 PSIZE
73 LASTX
74 PSIZE
75 X<>Y
76 FLSIZE
77 XEQ 10
78 CLKEYS
79 X<> c
80 RDN
81 +
82 PSIZE
83 RDN
84 PSIZE
85 X<> L
86 XEQ 04
87 RCL Y
88 -
89 192
90 -
91  E3
92 ST/ Z
93 X↑2
94 /
95 +
96 X<>Y
97 X<> c
98 X<>Y
99 REGMOVE
100 GETR
101 X<>Y
102 STO c

103♦LBL "RK"
104 " "
105 GETP
106 RTN

107♦LBL "SK"
108 ,
109 STO '
110 STO e
111 END
```

212 Bytes

*) Anm. d. Übers.: Zum Verständnis der Vorgänge ist es hilfreich, sich klarzumachen, daß die Funktion 'CLKEYS' viel strenger durchgreift. Sie löscht nämlich unerbittlich *alles*, was irgend mit den Zuweisungen zu tun hat: die Tastenzuweisungs*register*, die Tastenzuweisungs*bits* in den Registern ⊢ und e und die Tastenzuweisungs*bytes* in den globalen Marken. Letzteres gelingt ihr in "GETK" nur deswegen nicht, weil der Befehl 'ASTO c' in Zeile 66 die Kette der globalen Marken unterbricht, so daß ihr die Tastenzuweisungsbytes nicht zum Opfer fallen können und somit ein 'XEQ "RK"' beim Fehlerstop auf Zeile 82 in Bezug auf die globalen Marken erfolgreich ausgeführt werden kann.

Dezimalwerte der synthetischen Zeilen:

Zeile 03: 144, 117
Zeile 11: 27, 19
Zeile 13: 241, 16
Zeile 14: 144, 117
Zeile 20: 241, 240
Zeile 21: 206, 117
Zeile 23: 206, 118
Zeile 26: 155, 125
Zeile 27: 206, 118
Zeile 35: 206, 125
Zeile 38: 145, 117
Zeile 44: 27
Zeile 47: 144, 125
Zeile 49: 206, 117
Zeile 50: 145, 118
Zeile 64: 144, 125
Zeile 65: 246, 64, 1, 105, 12, 2, 0
Zeile 66: 154, 125
Zeile 69: 27
Zeile 79: 206, 125
Zeile 91: 27, 19
Zeile 97: 206, 125
Zeile 102: 145, 125
Zeile 109: 145, 122
Zeile 110: 145, 127

Nachstehend das Programm "IN", welches einmal laufen muß, bevor "RK" oder "GETK" das erste Mal benutzt werden dürfen:

Programmausdruck von "IN"
(Barcode im Anhang D)

```
01♦LBL "IN"      10 REGMOVE       19 STO 01        28 X<> \         37 X>0?
02 EMDIR         11 "⊢*"          20 X<> J         29 STO 02        38 ASTO L
03  E            12 RDN           21 STO 03        30 CLA           39 ASTO X
04 " ,×i♦A×♦0♦♦X" 13 RCL 12        22 RDN           31 STO [         40·SAVEX
05 CRFLD         14 STO 06        23 EMDIR         32 R↑            41 X<> L
06 +      15 "⊢♦×iλ                24 CLD           33 X>0?          42 STO 01
07 RCL \         16 RCL [         25 ST- T         34 E↑X           43 R↑
08 X<> c         17 STO 12        26 X<>Y          35 SEEKPTA       44 X<> c
09 RCL 04        18 X<>Y          27 STO 01        36 "×"           45 END
```

93 Bytes

Dezimalwerte der synthetischen Zeilen:

Zeile 03: 27
Zeile 04: 254, 32, 44, 1, 105, 0, 19, 240, 1, 137, 0, 48, 3, 0, 2
Zeile 07: 144, 118
Zeile 08: 206, 125
Zeile 15: 247, 127, 0, 1, 105, 11, 223, 255
Zeile 16: 144, 117
Zeile 20: 206, 119
Zeile 28: 206, 118
Zeile 31: 145, 117
Zeile 36: 241, 1
Zeile 44: 206, 125

Auf dem HP-41CX können die Zeilen 02 und 23 durch 'EMROOM' ersetzt werden. Ebenso wie in "VER" (vgl. Abschnitt 10D) werden auch hier keine numerischen Datenregister angetastet, obwohl Datenregisteradressen auftauchen (01 – 04 und 12), die das vermuten lassen.

Ungeachtet seiner Kürze ist "IN" ein hochgradig trickreiches synthetisches Programm. Wenn Sie sich für einen Fachmann in der synthetischen Programmierung halten, sollten Sie versuchen zu verstehen, was genau während des Programmlaufes geschieht. Ich halte dafür, daß einzig Clifford Stern alle in dem Programm verborgenen Kniffe kennt.

10I. Das Übertragen von X-Memory-Dateien auf Magnetkarten

In Abschnitt 3G wurden die Programme "WAS"/"RAS", mit denen ASCII-Dateien auf Magnetkarten geschrieben bzw. von Magnetkarten gelesen werden konnten, vorgestellt. Die in diesem Abschnitt zu besprechenden Programme "WFL" (*w*rite *file*) und "RFL" (*r*ead *file*) hat Clifford Stern geschrieben. Sie erlauben es, Dateien *jeden Typs* auf Magnetkarten aufzuzeichnen (beachten Sie in diesem Zusammenhang den am Schluß von Kapitel 11 gegebenen Hinweis!) oder auch nur für vorübergehende Zwecke in numerischen Datenregistern abzulegen. Darüber hinaus wird dabei die denkbar geringste Anzahl von Datenregistern überhaupt verwendet: je Register werden 7 Bytes untergebracht, nicht nur 6 oder weniger wie bei der Speichertechnik in "WAS". Ein drittes Programm, "RPF" (*r*etrieve *p*rogram *f*ile), erlaubt außerdem die Rückgewinnung von Programmdateien, welche Prüfsummenfehler aufweisen und deswegen auf 'GETP' nicht mehr ansprechen. Prüfsummenfehler können sich dadurch ergeben, daß man ein Programm, dessen 'GTO's und 'XEQ's vor Übertragung ins X-Memory nicht alle kompiliert worden sind, im X-Memory laufen läßt (vgl. Abschnitt 10F).

Einschränkungen beim Gebrauch von "WFL", "RFL" und "RPF"

1. Oberhalb der Programmgruppe "WFL"/"RFL"/"RPF" muß es wenigstens ein 'END' im Katalog 1 geben. Ist diese Voraussetzung verletzt, steht "MEMORY LOST" ins Haus.
2. Ebenso wie für den Betrieb von "VER" (Abschnitt 10D) muß mindestens eine der Bedingungen 'keine Weckaufträge anwesend und mindestens ein freies Register vorhanden' oder 'mindestens eine nicht-globale Zuweisung gegenwärtig' erfüllt sein.
3. Jedes der drei Programme erwartet bei Aufruf einen Dateinamen im Alpha-Register. Aus diesem Grunde dürfen die Zeilen 14 und 15 ('ALENG, 1/X') nicht gelöscht werden, denn sie stellen sicher, daß der Programmlauf nicht mit leerem Alpha-Register beginnt. Der Befehl 'POSA' verursacht eine Fehlerunterbrechung auf Zeile 19, falls das Alpha-Register ein Komma enthält, denn Kommata sind als Bestandteil eines Dateinamens verboten, weil sie vom Rechner als Trennzeichen, die verschiedene Namen gegeneinander absondern, angesehen werden (vgl. Abschnitt 1D). Ein Komma und alle nachfolgenden Zeichen werden daher von jeder X-Funktion, die Dateien anspricht – 'SAVEP' ausgenommen –, ignoriert. Läge also ein Komma im Alpha-Register, nützte die Fehlerfalle 'ALENG, 1/X' auch nicht viel.
4. Alle drei Programme können von einem Oberprogramm aufgerufen werden, nicht jedoch von einem Programm, das selbst bereits als Unterprogramm läuft. Mit anderen Worten: es dürfen keine Rückkehradressen im Rücksprungstapel liegen, wenn eines der drei Programme als Unterprogramm aufgerufen wird.

Gebrauchsanweisung für "WFL"

1. Setzen Sie Flag 14, falls die von Ihnen benutzen Karten schreibgeschützt sind.
2. Es gibt zwei Modi, in denen Sie "WFL" betreiben können. Im ersten Modus, bei dem die gesamte Datei in den Hauptspeicher geholt wird, befinden Sie sich, wenn Flag 01

gelöscht ist. Der zweite Modus arbeitet etwas langsamer, ist dafür aber sparsamer im Verbrauch von Datenregistern, weil er von ASCII-Dateien nur die Register berücksichtigt, die tatsächlich gefüllt sind. Er ermittelt zunächst die Gesamtanzahl der von der Datei verbrauchten Bytes und überträgt dann auf Grund der festgestellten Anzahl nur die Inhalte der benutzten X-Memory-Register in Datenregister. (Anm. d. Übers.: Auf dem HP-41CX lassen sich die Zeilen 31 – 42 durch die bei großen ASCII-Dateien wesentlich schnellere Befehlsfolge 'FLSIZE, 7, *, ASROOM, –, 6' ersetzen.) Sie erreichen diesen zweiten Modus durch das Setzen von Flag 01. Sobald Sie sich über den zu wählenden Modus im klaren sind und Flag 01 entsprechend eingestellt haben, legen Sie den Namen der ins Auge gefaßten Datei ins Alpha-Register und führen 'XEQ "WFL"' aus.

3. Wenn der auf Zeile 47 stehende Befehl 'PSIZE' sich genötigt sieht, "NO ROOM" zu melden, müssen Sie Programme oder Tastenzuweisungen (von System-Funktionen) löschen. Im Register X finden Sie die 'beantragte' Registeranzahl, nach der Sie sich beim Platz Schaffen richten müssen, vor.
4. Sobald die Aufforderung "RDY 01 OF mn" erscheint, können Sie entweder eine Magnetkarte einführen, um die Daten dauerhaft aufzuzeichnen, oder *zweimal 'R/S'* drücken, um das Beschreiben der Magnetkarten zu umgehen. *Drücken Sie nicht einfach die Korrekturtaste*, mit der man normalerweise eine überflüssige Kartenleseraufforderung zurückweist, weil sich eine Programm- oder Textdatei, die für die Zwecke der unbehinderten Übertragung in den Hauptspeicher vorübergehend in eine Datendatei umgeformt wurde, sonst nicht mehr in ihren ursprünglichen Dateityp zurückverwandelt. *Ändern Sie auch nicht den Stapelinhalt*, bevor Sie "WFL" wieder starten, es sei denn, ein "MEMORY LOST" rührt Sie nicht. Wenn der Kartenleser nicht eingesteckt ist, erscheint "RDY 01 OF mn" überhaupt nicht; der Dateiinhalt wird lediglich in Datenregister übertragen.
5. Falls Sie Flag 14 zu setzen vergessen haben sollten und geschützte Karten deshalb dem Beschreiben Widerstand entgegensetzen, weisen Sie die Aufforderung "RDY 01 OF mn" mit dem unter Punkt 4 erwähnten zweimaligen Drücken der Taste 'R/S' zurück. Sobald das Programm nach dem zweiten 'R/S' anhält, können Sie 'SF 14' nachholen und dann 'WDTA' von Hand über das Tastenfeld ausführen, um so Ihren 'Willen durchzusetzen' und die Registerinhalte doch noch auf die Karten zu bekommen.
6. Nachdem die letzte Karte ihren Weg durch den Kartenleser genommen hat oder zweimal 'R/S' gedrückt wurde, um das Beschreiben von Karten zu umgehen, schließt das Programm mit einer Befehlsfolge, die die neu geltende Datenregisteranzahl in X zurückläßt. Diese Zahl beurkundet die Mindestanzahl der von der herausgeschriebenen Datei benötigten Register und sollte auf den zugehörigen Karten vermerkt werden, damit sie später beim Gebrauch von "RFL" (s.u.) sogleich zur Hand ist.

WARNUNG: Wenn Sie die Absicht haben, die in den Hauptspeicher übertragenen Dateiinhalte alsbald wieder ins X-Memory zurückzuschreiben statt auf Magnetkarten aufzuzeichnen, müssen Sie sehr sorgsam mit ihnen umgehen, damit sie nicht vor dem Gebrauch von "RFL" zerstört werden. Die Daten haben nämlich ein höchst verletzliches Format: sie sind nicht-normalisiert (vgl. dazu die Ausführungen in Abschnitt 10C). Jedes 'RCL', jedes 'VIEW' und alle verwandten Funktionen verändern die Daten. Sie dürfen diese flüchtigen Gebilde daher solange nicht bewegen oder zu betrachten suchen, als sie nicht zurück ins Schutz bietende X-Memory gelangt sind. Diese Einschränkung ist der Preis für die Platz sparende Ausnutzung der Register durch das Programm "WFL", welches die Daten in dem Format beläßt, das sie im X-Memory, wo eine nicht-normalisierte Speicherung üblich ist, haben. Wenn Sie also

versehentlich Daten aus numerischen Registern, die von "WFL" beschickt worden sind, abgerufen oder betrachtet haben, müssen Sie "WFL" erst noch einmal aufrufen, bevor es möglich ist, einwandfreie Daten mit "RFL" zurückzuspeichern.

Gebrauchsanweisung für "RFL"

1. Ebenso wie für "WFL" gibt es auch für "RFL" zwei Betriebsmodi, und ebenso wie dort wird auch hier der Modus durch das Einstellen von Flag 01 gewählt. Weil jetzt aber in erster Linie der im Hauptspeicher zur Verfügung stehende Platz maßgebend ist, arbeitet der erste Modus (Flag 01 gelöscht) unabhängig davon, welcher Modus beim Betrieb von "WFL" gültig war. Um die Verhältnisse zu klären, müssen Sie zunächst Flag 01 löschen, den Namen der Datei ins Alpha-Register setzen und dann 'XEQ "RFL"' ausführen.
2. Wenn die Funktion 'PSIZE' (Zeile 47) "NO ROOM" meldet und Flag 01 beim Beschreiben von Karten mit "WFL" gelöscht war, werden Sie durch das Tilgen von Programmen oder Tastenzuweisungen Platz für die einzulesenden Daten schaffen müssen. In X finden Sie die Zahl, nach der Sie sich dabei zu richten haben, vor. Falls Sie jedoch "NO ROOM" gemeldet bekommen, obwohl Flag 01 beim Gebrauch von "WFL" gesetzt war (seinerzeit also nur die im X-Memory belegten Register herausgeschrieben wurden), haben Sie noch eine andere Möglichkeit: Sie können Flag 01 setzen, um dem Programm mitzuteilen, daß es die Datenregisteranzahl nicht der Größe der angesteuerten Datei anpassen soll. Jedesmal, wenn Sie "RFL" im zweiten Modus (Flag 01 gesetzt) aufrufen, müssen Sie dann aber selbst prüfen, ob wirklich ausreichend viele Datenregister zugeteilt sind, und gegebenenfalls 'SIZE' von Hand ausführen, um die Anzahl der im Hauptspeicher zur Verfügung stehenden Datenregister der Anzahl der auf Karten geschriebenen Register (an dieser Stelle ist die unter Punkt 6 von "WFL" empfohlene Karten-Notiz von Nutzen) anzupassen. Das Ihnen durch die Einstellung von Flag 01 zugestandene Wahlrecht können Sie insbesondere dazu verwenden, den Inhalt von Datenkarten in eine Datei zu lesen, deren Größe die Anzahl der beim Einlesen der Karten benötigten Register mit Sicherheit übersteigt. Ungeachtet allerdings, mit welcher Maßnahme Sie auf "NO ROOM" antworten, in jedem Fall müssen Sie das Alpha-Register neu laden und abermals 'XEQ "RFL"' ausführen.
3. Wenn der Kartenleser eingesteckt ist, erscheint dessen Aufforderung "CARD". An dieser Stelle schieben Sie nun die Datenkarten ein, welche mit "WFL" beschrieben worden sind. Sollten sich die Daten schon in den Registern des Hauptspeichers befinden, können Sie die Aufforderung des Kartenlesers mit dem *zweimaligen* Drücken der Taste 'R/S' übergehen. Ebenso wie bei "WFL" dürfen Sie auch hier *keinesfalls mit der Korrekturtaste* auf "CARD" antworten, weil sonst Ihre nach Plan nur vorübergehend zum Typ 'Daten' übergewechselte Datei keine Anstalten mehr macht, ihren ursprünglichen Typ wieder anzunehmen. Und drittens ist, wie bei "WFL", auch die *Veränderung des Stapels* zu *vermeiden*, bevor Sie das Programm mit 'R/S' planmäßig zu Ende laufen lassen; andernfalls platzt ein "MEMORY LOST" in Ihre friedliche Arbeit. Bei Abwesenheit des Kartenlesers wird die Aufforderung "CARD" ganz beiseite gelassen.
4. "RFL" überträgt die von den Karten in die Datenregister gelangende Information automatisch in die Register der im X-Memory liegenden Datei. Dabei ist es völlig gleichgültig, ob man mit einer wie auch immer gearteten Programmdatei (hier nochmals ein dezenter Hinweis auf den Wink am Schluß von Kapitel 11), einer Datendatei oder

einer Textdatei arbeitet. Da allerdings der Kartenleser für das Beschreiben und Lesen von Programm- und Datenkarten schon 'von Natur aus' eingerichtet ist, liegt der wesentliche Vorteil des Programmpaares "WFL"/"RFL" in der Möglichkeit, nun auch Textdateien platzsparend auf Magnetkarten unterbringen und von dort einlesen zu können.

Eine sinnvolle Anwendung von "WFL" und "RFL" besteht darin, die Anzahl der Register, welche für eine sich nur selten verändernde Textdatei benötigt werden, auf das Mindestmaß herabzuschrauben: Man erzeugt zunächst eine großzügig bemessene ASCII-Datei, füllt diese, ohne mit Platznot rechnen zu müssen, mit dem gewünschten Text und benutzt dann "WFL" bei gesetztem Flag 01, um die entstandenen Sätze der Datei – und eben nur diese – in Datenregister und möglicherweise auch auf Magnetkarten zu bringen. Die bei Beendigung von "WFL" in X liegende Zahl muß man sich merken. Danach wird die ASCII-Datei gelöscht, eine neue Datei gleichen Typs und gleichen Namens erzeugt, diesmal von der Größe der von "WFL" zurückgelassenen Zahl. Anschließend führt man 'XEQ "RFL"' aus, beantwortet die Aufforderung "CARD" mit zweimaligem 'R/S' und liest so die im Hauptspeicher abgelegten Sätze der alten Datei in die neue Datei, welche jetzt *genau* die benötigte Größe für den erzeugten Text besitzt, zurück.*)

Eine andere Möglichkeit, "WFL"/"RFL" nutzbringend anzuwenden, ist die, eine ASCII-Datei, deren Größe nicht mehr ausreicht, aufzublähen, ohne der Inhalte verlustig zu gehen: Bei gesetztem Flag 01 wird "WFL" ausgeführt, um den gesamten Dateiinhalt in den Hauptspeicher (und u.U. auch auf Karten) zu bringen. Dann löscht man die aus den Nähten platzende Datei, erzeugt eine neue größere gleichen Namens und ruft zum Schluß "RFL" auf, und zwar bei gesetztem Flag 01, damit die neue Dateigröße keinen Fehlerstop auslöst (vgl. Punkt 2 der Gebrauchsanweisung zu "RFL"). Denken Sie auch daran, daß Sie wegen drohender Normalisierung die Daten zwischenzeitlich weder ansehen noch umschaufeln dürfen und daß "CARD" mit 'R/S, R/S' ignoriert werden kann.

Drittens lassen sich "WFL" und "RFL" dazu verwenden, die Inhalte einer Datendatei unter Verzicht auf 'GETR' und 'SAVER' (abgesehen davon, daß diese Befehle natürlich innerhalb der Programme "WFL"/"RFL" verwendet werden) zwischen Hauptspeicher und X-Memory hin- und herzuschieben. Zwar sind die Programme ein großer Aufwand gegenüber dem Gebrauch einfacher X-Funktionen, doch erspart man sich u.U. das Heraufsetzen der Datenregisteranzahl, weil die Programme dies selbständig erledigen. Überdies braucht man keinen Gedanken an den Dateizeiger zu verschwenden, weil er auf seinen ursprünglichen Wert zurückgestellt wird (vgl. dagegen Abschnitt 2E). Damit hat man in "WFL"/"RFL" einen geringen doch erwähnenswerten Vorteil gegenüber 'GETR'/'SAVER'. Schließlich kann man mit "WFL" eine Programmdatei ohne Umweg über den Hauptspeicher (abgesehen davon, daß "WFL" diesen Umweg nimmt) aus dem X-Memory auf Karten lesen, vorausgesetzt man nimmt keinen Anstoß daran, daß das ausgelesene Programm in einem Format aufgezeichnet wird, welches nur durch "RFL" wieder 'aufgelöst' werden kann. Dies mag insbesondere dann annehmbar sein, wenn das aufzuzeichnende Programm sowieso nur für Ausführungen im X-Memory vorgesehen ist.

Die Routine "RPF" (*r*etrieve *p*rogram *f*ile) bringt ein Programm, das wegen eines Prüfsummenfehlers der Aufforderung, sich in den Hauptspeicher zu begeben, nicht nachkommt, 'mit

*) Anm. d. Übers.: Auf dem HP-41CX läßt sich die Aufgabe, die Größe einer Textdatei auf das Mindestmaß zu senken, wesentlich bequemer mit der schon in Abschnitt 3H vorgeschlagenen kleinen Routine "DGM" lösen.

Gewalt' dorthin. Auf Prüfsummenfehler werden Sie durch die Meldung "CHKSUM ERR" stets dann aufmerksam gemacht, wenn sich ein 'GETP' oder ein 'GETSUB' wegen eben dieses Fehlers vergeblich durchzusetzen versuchen. Bei unverletzten Programmen sind 'GETP' und 'GETSUB' selbstverständlich jederzeit vorzuziehen, weil "RPF" ein 'Medikament mit Nebenwirkung' ist: es verwandelt die im X-Memory liegende Programmdatei in eine Datendatei. "RPF" sollte daher 'letzte Zuflucht' bleiben – zumal das im X-Memory liegende Programm ernsthafte Beschädigungen erlitten haben könnte – und erst eingesetzt werden, nachdem alle anderen Quellen (Einlesen aus einem Massenspeicher, von Magnetkarten oder durch Barcode) versiegt sind. Sollte der Prüfsummenfehler jedoch nur von einem Programmlauf im X-Memory herrühren (vgl. Abschnitt 10F), können Sie "RPF" unbesorgt benutzen, denn Sie erhalten dann eine saubere – sogar kompilierte! – Kopie in den Hauptspeicher gelegt.

Die für den Gebrauch von "RPF" erforderlichen Handgriffe schließen – insbesondere wegen der über das Tastenfeld zu gebenden Befehle – die Verwendung der Routine als Unterprogramm aus. Folgen Sie den nachstehenden Anweisungen sorgfältig und genau.

Gebrauchsanweisung für "RPF"

1. 'GTO..'
2. Übergang in den PRGM-Modus. Prüfen Sie, ob wenigstens ein freies Register zur Verfügung steht. Wenn ja, dann 'SST' auf das ständige '.END.'. Mit diesem '.END.' in der Anzeige einen beliebigen Befehl (etwa 'ENTER↑') einfügen und sofort wieder löschen.
3. Zurück in den RUN-Modus. Namen der beschädigten Datei ins Alpha-Register setzen. 'XEQ "RPF"' ausführen. Wenn Sie – auf Zeile 47 – die Meldung "NO ROOM" erhalten, müssen Sie Platz im Hauptspeicher schaffen und danach noch einmal mit Schritt 1 beginnen.
4. 'PACK'en mit 'XEQ "PACK"' (nicht mit 'GTO..'), ein 'lebensnotwendiger' Schritt!
5. Ein 'BEEP' zeigt die Beendigung des Programmlaufs an. Sie finden danach 'SIZE 000' vor und können sich mit 'EMDIR' davon überzeugen, daß aus Ihrer Programm- tatsächlich eine Datendatei geworden ist.
6. Prüfen Sie das wiedergewonnene Programm auf seine Richtigkeit hin, insbesondere dann, wenn Sie berechtigten Anlaß zu der Vermutung haben, daß der gemeldete Prüfsummenfehler nicht bloß auf ein Kompilieren von Sprungbefehlen zurückzuführen ist.
7. Falls das wiedergewonnene Programm weniger als 14 Bytes lang war, kann es vorkommen, daß sich "RPF" nicht ein zweites Mal mit demselben Erfolg wie beim ersten Mal auf den jetzt als Datendatei im X-Memory liegenden Deserteur hetzen läßt. Diese Einschränkung gilt aber nur dann, wenn zugleich die Bytesumme modulo 7 die Dateigröße ('FLSIZE') übertrifft. Es ist allerdings unwahrscheinlich diesem Problem außer beim Experimentieren gegenüberzustehen, weil jedes Programm, das der Ausführung im X-Memory für wert befunden wird, sehr viel mehr als 13 Bytes umfassen dürfte.

BEMERKUNG: Wenn Sie "RPF" aus der Programmgruppe "WFL"/"RFL"/"RPF" entfernen wollen, müssen Sie die Zeilen 149 – 232, 122 – 123, 11 – 12 und 01 – 04 (am besten in dieser Reihenfolge, um beim Löschen bequem 'Spur halten' zu können) herausnehmen. Dann schrumpft die Gruppe zu einem 265 Bytes zählenden reinen "WFL"/"RFL"-Programm zusammen. Besitzer eines PPC ROMs (vgl. Anhang C) können das Ganze auch noch auf andere Weise abmagern: Die Zeilen 153 – 166 lassen sich insgesamt durch 'XROM "E?"' ersetzen.

Programmausdruck von "WFL"/"RFL"/"RPF"
(Barcode im Anhang D)

```
01♦LBL "RPF"
02 CF 01
03 SF 05
04 GTO 02

05♦LBL "WFL"
06 CF 06
07 GTO 01

08♦LBL "RFL"
09 SF 06
      57 "⊢♦×i
10♦LBL 01
11 CF 05

12♦LBL 02
13 CF 25
14 ALENG
15 1/X
16 44
17 POSA
18 CHS
19 LN
20 FLSIZE
21 "⊢
22 7
23 AROT
24 X<> [
25 X<>Y
26 FC? 01
27 GTO 04
28 SF 25
29 FS? 06
30 GTO 04
31 CLX
32 SEEKPT

33♦LBL 03
34 CLA
35 GETREC
36 ALENG

37 +
38 FS? 25
39 GTO 03
40 RCLPT
41 +
42 8
43 +
44 7
45 /
46♦LBL 04
47 PSIZE
48 "×i♦A×♦0←♦X̄"
49 X<> \
50 X<> c
51 STO 10
52 X<> [
53 REGMOVE
54 RCL c
55 "⊢*"
56 STO 06
58 CLX
59 STO 07
60 RCL \
61 R↑
62 CF 07
63 X=Y?
64 SF 07
65 X<> [
66 STO 12
67 STO 01
68 X<> ]
69 STO 03
70 2
71 CHS
72 LASTX
73 FS? 07
74 GTO 06
75 ,

76♦LBL 05
77 STO ]
78 LASTX
79 STO 01
80 R↑
81 FLSIZE
82 ST+ Y
83 2
84 ST+ Z
85 CLX
86 STO ]
87 RCL c
88 STO 01
89 R↑
90 SEEKPTA
91 RCL [
92 GETX
93 X≠Y?
94 GTO 05
95 X<> \
96 STO [
97 RCLPT
98 GETX

99♦LBL 06
100 STO ]
101 "⊢♦"
102  E20
103 STO ↑
104  E
105 CHS
106 AROT
107 R↑
108 SEEKPT
109 R↑
110 ,
111 X<> ]
112 FC? 07
113 SAVEX
114 FS? 07
115 SIGN

116 RDN
117 LASTX
118 STO 01
119 RCL a
120 X<> c
121 SF 25
122 FS? 05
123 GTO 07
124 TONE 8
125 FS? 06
126 RDTA
127 FS? 06
128 SAVER
129 FC? 06
130 GETR
131 FC? 06
132 WDTA
133 CF 25
134 STO c
135 STO 01
136 CLX
137 STO \
138 R↑
139 SEEKPTA
140 R↑
141 FC? 07
142 SAVEX
143 FC? 07
144 LASTX
145 STO 01
146 X<> a
147 STO c
148 SIZE?
149 RTN

150♦LBL 07
151 RCLPTA
152 STO 00
153 RCL c
154 "*"
155 X<> [

156 STO \
157 ASHF
158 ALENG
159 8
160 Y↑X
161 ATOX
162 *
163 512
164 MOD
165 ATOX
166 +
167 ENTER↑
168 "×i"
169 16
170 *
171 2
172 +
173 ENTER↑
174 XEQ 10
175 RCL 00

176 SIGN
177 CLX
178 XTOA
179 SEEKPT
180 PSIZE
181 X<> [
182 X<> c
183 FLSIZE
184 "♦-"
185 STO ]
186 X<>Y
187 STO \
188 R↑
189 X<> [
190 STO 00
191♦LBL 08
192 GETX
193 STO IND ]
194 DSE ]
195 GTO 08

196 X<> [
197 RCL \
198 LASTX
199 7
200 MOD
201 SEEKPT
202 CF 25
203 LASTX
204 -
205 AROT
206 CHS

207♦LBL 09
208 "⊢♦"
209 DSE X
210 GTO 09
211 X<>Y
212 X<> [
213 STO 01
214 2
215 CHS
216 AROT
217 R↑
218 ENTER↑
219 XEQ 10
220 X<> [
221 X<> c
222 BEEP
223 STOP

224♦LBL 10
225 256
226 MOD
227 X<>Y
228 LASTX
229 /
230 XTOA
231 RDN
232 XTOA
233 END
```

419 Bytes

Zeile 21 hängt 6 Leerzeichen ('SPACE') an.

Dezimalwerte der synthetischen Zeilen:

Zeile 24: 206, 117
Zeile 48: 252, 1, 105, 0, 19, 240, 1, 137, 0, 48, 3, 0, 2
Zeile 49: 206, 118
Zeile 50: 206, 125
Zeile 52: 206, 117
Zeile 54: 144, 125
Zeile 57: 247, 127, 0, 1, 105, 11, 223, 255
Zeile 60: 144, 118
Zeile 65: 206, 117
Zeile 68: 206, 119
Zeile 77: 145, 119
Zeile 86: 145, 119
Zeile 87: 144, 125
Zeile 91: 144, 117
Zeile 95: 206, 118
Zeile 96: 145, 117
Zeile 100: 145, 119
Zeile 101: 242, 127, 0
Zeile 102: 27, 18, 16
Zeile 103: 145, 120
Zeile 104: 27
Zeile 111: 206, 119
Zeile 119: 144, 123
Zeile 120: 206, 125
Zeile 134: 145, 125
Zeile 137: 145, 118
Zeile 146: 206, 123
Zeile 147: 145, 125
Zeile 153: 144, 125
Zeile 155: 206, 117
Zeile 156: 145, 118
Zeile 168: 242, 1, 105
Zeile 181: 206, 117
Zeile 182: 206, 125
Zeile 184: 243, 192, 0, 45
Zeile 185: 145, 119
Zeile 187: 145, 118
Zeile 189: 206, 117
Zeile 193: 145, 247
Zeile 194: 151, 119
Zeile 196: 206, 117
Zeile 197: 144, 118
Zeile 208: 242, 127, 0
Zeile 212: 206, 117
Zeile 220: 206, 117
Zeile 221: 206, 125

10J. Tastenzuweisungen synthetischer Funktionen

Wenn man synthetisch programmiert, ist es hilfreich, die häufiger benutzten synthetischen Zwei-Byte-Funktionen auf dem Tastenfeld im USER-Modus zur Verfügung zu haben. In den Büchern über synthetische Programmierung (vgl. Anhang C) werden einige sogenannte Tastenzuweisungsprogramme, mit denen man solche Zuweisungen vornehmen kann, geboten. Das hier vorgestellte Programm "ASG" (*assign*) ist von Tapani Tarvainen entwickelt und später von ihm zusammen mit Gerard Westen verbessert worden. Es stellt einen ganz beachtlichen Fortschritt in der synthetischen Programmierung dar, denn frühere Programme dieser Art verlangten vom Benutzer die Bereitstellung der beiden Dezimalwerte, aus denen die angestrebte Funktion zusammenzusetzen war. "ASG" begnügt sich großzügig mit der Entgegennahme des *Namens* der zuzuweisenden Funktion, also mit deren 'buchstabierter' Eingabe.

WARNUNG: Der Marke 'LBL "ASG"' muß wenigstens ein 'END' im Katalog 1 vorangehen. Wenn Sie, wie in Abschnitt 10B empfohlen, "XF" als erstes Programm im Hauptspeicher liegen haben, ist schon alles Notwendige getan. Sollte dagegen versehentlich "ASG" Ihren Katalog 1 anführen und in dieser Lage aufgerufen werden, ergibt sich sehr wahrscheinlich eine Tastenfeldblockade und möglicherweise auch ein "MEMORY LOST".

"ASG" - Beispiel 1

Nehmen Sie an, Sie wollen die synthetische Funktion 'RCL b' einer Taste zuweisen, dann beginnen Sie mit 'XEQ "ASG"'. Daraufhin fordert Sie das Programm mit "ASN" – so, wie es sonst der Rechner mit der Funktion 'ASN' tut – dazu auf, Ihren Wunsch auszusprechen, wofür bereits der ALPHA-Modus eingeschaltet ist. Die Aufforderung beantworten Sie ganz schlicht mit buchstabierender Eingabe:

R, C, L, 'SPACE', 'SHIFT' B,

was wegen des ALPHA-Modus ohne Umstände sogleich geschehen kann. Dann startet man das Programm wieder mit 'R/S'. Etwa ½ Sekunde Wartezeit muß man nun verstreichen lassen, bevor man die Taste, auf der die gewünschte Funktion liegen soll, drücken kann. Auch dies wieder ganz wie von 'ASN' her gewohnt, also unter Vorabbetätigung der Taste 'SHIFT', falls eine umgeschaltete Taste belegt werden soll (in diesem Fall müssen Sie allerdings noch das zögernde Erscheinen des Minuszeichens abwarten, bevor die Taste selbst gedrückt werden darf).

Haben Sie alles – den Namen der Funktion und die ausgewählte Trägertaste – eingetastet, müssen Sie dem Programm ein wenig Zeit lassen, Ihren Auftrag durchzuführen: der Preis für die Bequemlichkeit ist Laufzeit. Die ganze Prozedur ist derjenigen, die für die eingebaute Funktion 'ASN' vorgeschrieben ist, verblüffend ähnlich.

Die Marke "PASG" (*p*rogrammable *ass*ign) bietet eine dem Wirken der X-Funktion 'PASN' nachgeahmte Einstiegsmöglichkeit in "ASG" zur programmierten Zuweisung synthetischer Funktionen. Legen Sie dazu den Namen der Funktion ins Alpha-Register sowie den Tastenkode (denselben, der für 'PASN' in Frage käme) ins X-Register, und führen Sie 'XEQ "PASG"' aus.

Der bei "PASG" beginnende Programmteil führt den wundersamen Geniestreich der Umwandlung des Funktionsnamens in ihre Dezimalwerte durch. Dies geschieht in zwei Schritten. In unserem obigen Beispiel 'RCL b' weist "PASG" die Vorsilbe 'RCL' getrennt von der Nachsilbe 'b' zu (Zeile 125) und entzieht dann einem der Systemregister den zugehörigen Dezimalkode (Zeile 132). Außerdem wird die Nachsilbe 'b' in ihr Dezimaläquivalent umgerechnet. Beide Dezimalwerte dienen dann als Eingabe für ein mehr 'bürgerliches' Tastenzuweisungsprogramm, "MKX" genannt, ebenfalls eine Schöpfung von T. Tarvainen, wiederum verbessert von G. Westen.

Hinweise für experimentierfreudige Neulinge:

Wenn Sie mit den Tücken der synthetischen Programmierung noch nicht vertraut sind, den Programmen der vorangegangenen Abschnitte aber bereits soviel Geschmack abgewonnen haben, daß Sie sich selbst ein wenig in dieser abenteuerlichen Kunst erproben wollen, werden Sie vermutlich damit beginnen, "ASG" zur Erzeugung Ihrer ersten eigenen synthetischen Befehle zu verwenden. Ein paar grundlegende Einblicke sollen Sie davor bewahren, schon bei den ersten tastenden Versuchen Opfer der bekanntesten Fallgruben zu werden. Darum soviel: Zuerst und vor allem nie den Inhalt von Register c abändern, bevor Sie nicht genau wissen, was Sie dabei tun. Das Schreckgespenst "MEMORY LOST" liegt ewig wachsam auf der Lauer und stürzt sich sofort auf jeden Eindringling, der sich unachtsam oder kenntnisarm an dieses heilige Register wagt. Zweitens müssen Sie sich wenigstens einigermaßen über die verwirrenden Umstände bei der Anzeige und dem Ausdruck synthetischer Programmzeilen im klaren sein. Viele

solcher Zeilen geben ihren Inhalt gar nicht erst preis, viele weisen in der Anzeige andere Zeichen als beim Ausdruck vor. Am unanständigsten benehmen sich dabei Textzeilen. In ihnen enthaltene Zeichen der Dezimalwerte 128 – 255 verschwinden beim Ausdruck völlig, erscheinen in der Anzeige sämtlich als Vollzeichen und stiften u.U. auch noch den Drucker beim Auflisten des Programms zu scheinbar planlosen Untaten an. Wichtige Funktionen, die auf Systemregister zugreifen, geben sich in der Anzeige und beim Ausdruck in verschiedener Gestalt zu erkennen:

Anzeige	Programmausdruck
STO M	STO [
STO N	STO \
STO O	STO]
STO P	STO ↑
STO Q	STO _
STO ⊢	STO ⊤

Was hier mit dem Beispiel 'STO' aufgelistet wurde, gilt gleichermaßen für die Gestalt sämtlicher anderen verwandten Zwei-Byte-Funktionen, also z.B. für die Funktionen mit den Vorsilben 'RCL' oder 'X<>'. Die Register M, N, O und P bilden zusammenhängend das Alpha-Register. Diese sowie die Register Q und a sind die zuverlässigsten Partner für den experimentierenden Anfänger.

Vollständige Aufklärung über die Bedeutung und die Nutzungsmöglichkeiten der Systemregister finden Sie in der einschlägigen HP-41 Literatur (s. Anhang C). Dort werden selbstverständlich auch Verfahren beschrieben, mit denen sich synthetische Textzeilen, die ihrer Natur nach nicht mit "ASG" erzeugt werden können, herstellen lassen.

Gebrauchsanweisung für "ASG"

1. Stellen Sie sicher, daß es wenigstens ein 'END' oberhalb von 'LBL "ASG"' und keine Benutzermarke 'LBL "ANUM"' im Katalog 1 gibt. Wenn Sie diese Voraussetzungen nicht beachten, ist "MEMORY LOST" gewiß.
2. Führen Sie 'XEQ "ASG"' aus. Das Programm hält mit der Aufforderung "ASN" im ALPHA-Modus an.
3. Buchstabieren Sie den Namen der zuzuweisenden Funktion unter Verwendung des ALPHA-Tastenfeldes. Gehört die Nachsilbe zu einer synthetischen Funktion, können Sie stattdessen auch den entsprechenden Dezimalwert (zwischen 0 und 255) eintasten, und zwar ebenfalls im ALPHA-Modus. Sind Ihnen aus irgendeinem Grunde die Dezimalwerte überhaupt lieber, können Sie sogar für die Vorsilbe die zugehörige Dezimalzahl wählen. Bei indirekten Funktionen wie z.B. 'GTO IND X' sind Sie nicht gezwungen, den Befehlsteil 'IND' ausdrücklich zu buchstabieren. Sie brauchen vielmehr bloß genau zwei Leerstellen zwischen die Vorsilbe, hier 'GTO', und den zweiten Teil der Nachsilbe, hier 'X', zu setzen.
4. Drücken Sie 'R/S', um das Programm wieder zu starten.
5. Warten Sie ½ Sekunde, bis Sie die Taste, welche die ausgewählte Funktion tragen soll, drücken. Für umgeschaltete Tasten muß zuvor 'SHIFT' betätigt und das Erscheinen von " – " in der Anzeige abgewartet werden.
6. Das Programm wird, wenn alles normal verläuft, treulich die exotischsten Zuweisungen tätigen. Sollte dennoch einmal ein Fehlerstop auftreten, versuchen Sie *nicht*, mit 'R/S' fortzufahren. Beginnen Sie in einem solchen Fall unter allen Umständen mit Schritt 2

von vorn. Lautete die Meldung "NO ROOM" (Zeilen 51 oder 157), müssen Sie wie üblich Platz schaffen durch Löschen eines Programms oder überflüssiger Tastenzuweisungen.

7. Für jede weitere Zuweisung ist erneut 'XEQ "ASG"' auszuführen.

Gebrauchsanweisung für "PASG"

1. Stellen Sie sicher, daß es wenigstens ein 'END' oberhalb von 'LBL "PASG"' und keine Benutzermarke 'LBL "ANUM"' im Katalog 1 gibt. Wie bei "ASG" wird die Mißachtung dieser Bedingungen mit der Geißel "MEMORY LOST" bestraft.
2. Laden Sie das Alpha-Register mit der Zeichenkette, die sich ergibt, wenn man die zuzuweisende Funktion buchstabiert. Die unter Schritt 3 der Erläuterungen zu "ASG" gemachten Bemerkungen über die abweichenden Formen der zugelassenen Zeichenketten gelten auch hier.
3. Legen Sie den Zeilen-/Spaltenkode der ausgewählten Taste ins Register X. "PASG" arbeitet insoweit genau wie 'PASN'.
4. Führen Sie 'XEQ "PASG"' aus. Das Programm tätigt dann die angestrebte synthetische Zuweisung. Lassen Sie sich wie bei "ASG" durch einen Fehlerstop nicht dazu verleiten, 'R/S' zu drücken. Beginnen Sie beileibe wieder mit Schritt 2.
5. Für weitere Zuweisungen müssen das Alpha- und das X-Register neu geladen und "PASG" abermals aufgerufen werden.

Gebrauchsanweisung für "MKX"

1. Legen Sie den Dezimalwert der Vorsilbe ins Register Z, den der Nachsilbe ins Register Y und den Tastenkode ins Register X.
2. Rufen Sie "MKX" auf, um die Zuweisung geliefert zu bekommen.

Vorsichtsmaßregeln für den Gebrauch von "ASG", "PASG" und "MKX"

1. Unterbrechen Sie keines der Programme und verfolgen Sie deren Einzelschritte auch nicht mit 'SST'. Wenn Sie versehentlich doch einmal zwischen den Zeilen 158 und 161 unterbrechen oder mit 'SST' voranschreiten, müssen Sie wieder ganz von vorn beginnen. Sollten Sie hinter Zeile 176 unterbrechen, hilft nur ein sofortiges 'R/S', das drohende "MEMORY LOST" abzuwenden, und auch das nicht immer.
2. Es darf keine globale Marke vorhanden sein, die denselben Namen wie die zuzuweisende Funktion trägt. Es darf also z.B. kein 'LBL "STO"' im Katalog 1 geben, wenn "ASG" oder "PASG" einen synthetischen 'STO'-Befehl zuweisen sollen.
3. **WARNUNG:** Vergewissern Sie sich unter allen Umständen davon, daß es keine globale Marke des Namens "ANUM" im Katalog 1 gibt. Andernfalls entartet Ihr Programmspeicher zu einer Datenmüllhalde, der nichts Brauchbares mehr entnommen werden kann.

Die Programme "ASG", "PASG" und "MKX" sind uneingeschränkt einsatzfähig, auch dann, wenn Weckaufträge für den Time-Modul oder Ein-/Ausgabepuffer anderer Module im Hauptspeicher liegen. Mit "ASG" und "PASG" können Sie selbstverständlich auch nicht-synthetische Funktionen und sogar Marken aus dem Katalog 1 auf Tasten legen. "ASG"

und "PASG" bilden einen vollständigen Ersatz für die Funktionen 'ASN' und 'PASN'. Sie nehmen jede Eingabe, die für 'ASN' oder 'PASN' zulässig ist, entgegen und darüber hinaus auch noch alle diejenigen Zeichen und Werte, welche synthetischen Funktionen entsprechen.

Weitere "ASG"- und "PASG"-Beispiele

Die nachstehende Liste führt eine Reihe von Zuweisungen, synthetisch als auch nicht-synthetisch, auf, um darzulegen, wie die für "ASG"/"PASG" erforderlichen Eingaben in den verschiedenen Fällen lauten müssen und dürfen. Im allgemeinen sind, wie die Liste zeigt, mehrere voneinander abweichende Alpha-Eingaben möglich. Funktionen, die Sie allein durch die Angabe der zugehörigen Dezimalwerte beschreiben wollen, lassen sich, wie leicht einzusehen ist, am bequemsten durch Laden des Stapels und Aufruf von "MKX" zuweisen.

Marke/Funktion	Alpha-Eingabe
"ASG"	"ASG" jede globale Marke läßt sich mit
"VER"	"VER" "ASG" oder "PASG" zuweisen
'BST'	"BST"
'SIGN'	"SIGN"
'SF 14'	"SF 14" oder "168 14"
'STO N'	"STO N", "STO 118", "145 N" oder "145 118"
'X<> M'	"X<> M", "X<> 117", "206 M" oder "206 117"
'GTO IND X'	"GTO IND X", "GTO X" (2 Leerzeichen!), "GTO I 115", "GTO 243", "208 243", usw.
'RCL IND X'	"RCL IND X", "RCL X" (2 Leerzeichen!), "RCL I X", "RCL 243", "144 243", usw.
'XROM 29,08'	"XROM 29,08" oder "X 29,08" (≙ 'PRA')
'TONE 10'	"TONE 10" oder "159 10"
'TONE 89'	"TONE 89" oder "159 89"
'FIX 10'	"FIX 10" oder "156 10"
'XROM 28,35'	"XROM 28,35" oder "X 28,35" (≙ 'OUTA')

synthetische Funktionen für den Kenner:

'GTO.000'	"199 133" (arbeitet im PRGM-Modus nur dann, wenn der Kartenleser eingesteckt ist)
'eGØBEEP'	"4 167" oder "0 167" (gestattet Zugriff auf die Drucker-, Massenspeicher und HP-IL-Funktionen)
Q-Bote	"27 0" (nur für den PRGM-Modus)
Byte-Schnapper	"247 63" (für Anfänger ungeeignet: "MEMORY LOST" oder Zerstörung des Katalogs 1 bei unsachgemäßem Gebrauch in der Nähe eines 'END')

Anfänger sollten sich unbedingt in der Literatur (s. Anhang C) sachkundig machen, bevor Sie mit den voranstehenden Funktionen ihre Versuche anstellen.

Programmausdruck von "ASG"/"PASG"/"MKX"
(Barcode im Anhang D)

```
01♦LBL "ASG"
02 RCLFLAG
03 SIGN
04 ""
05 ASTO d
06 "ASN "
07 STOP
08 CF 21
09 "⊢ "

10♦LBL 01
11 31
12 AVIEW
13 GETKEY
14 X=0?
15 GTO 01
16 X≠Y?
17 GTO 02
18 "⊢-"
19 FS?C 03
20 GTO 01
21 2
22 CHS
23 AROT
24 ATOX
25 ATOX
26 SF 03
27 GTO 01

28♦LBL 02
29 ARCL X
30 AVIEW
31 FC? 03
32 CHS
33 X>0?
34 XTOA
35 LASTX
36 STOFLAG
37 ATOX
38 ATOX
39 LN
40 CHS
41 AROT
42 ASHF
43 R↑

44♦LBL "PASG"
45 AOFF
46 32
47 POSA
48 X>0?
49 GTO 03
50 RDN
51 PASN
52 CLD
53 RTN

54♦LBL 03
55 "⊢♦"
56 AROT
57 "⊢@0"
58 ATOX
59 POSA
60 ISG X
61 AON
62 AROT
63 44
64 POSA
65 ISG X
66 GTO 04
67 AROT
68 R↑
69 ANUM
70 ASHF
71 ANUM
72 640
73 +
74 64
75 *
76 +
77 RCL X
78 256
79 ST/ Z
80 MOD
81 GTO 06

82♦LBL 04
83 R↑
84 ANUM
85 ATOX
86 84
87 -
88 X<=0?
89 GTO 04
90 CHS
91 7
92 +
93 X>0?
94 GTO 05
95 CHS
96 31
97 MOD
98 2

99♦LBL 04
100 X<0?
101 9
102 X>0?
103 +
104 X>0?
105 3
106 X>0?
107 +

108♦LBL 05
109 17
110 +
111 X>0?
112 95
113 X>0?
114 +
115 X>0?
116 X<>Y
117 CLX
118 POSA
119 X>0?
120 AROT
121 CLX
122 X<> d
123 R↑
124 SF 25
125 PASN
126 RCL _
127 STO ↑
128 RDN
129 X<>Y
130 X<> d
131 X≠0?
132 ATOX
133 ASHF
134 X=0?
135 ANUM
136 R↑
137 FS? 48
138 GTO 06
139 X<>F
140 R↑
141 208
142 R↑
143 X≠Y?
144 SF 07
145 X<=Y?
146 X=Y?
147 RDN
148 X>Y?
149 174
150 R↑
151 X<>F

152♦LBL 06
153 AOFF
154 R↑

155♦LBL "MKX"
156 "ANUM"
157 PASN
158 RCL '
159 CLA
160 STO ↑
161 "⊢      B"
162 ASTO \
163 ARCL \
164 R↑
165 XTOA
166 R↑
167 XTOA
168 RCL \
169 ATOX
170 SIGN
171 AROT
172 RCL [
173 RCL c
174 "@÷iλX♦"
175 ASTO c
176 R↑

177♦LBL 07
178 RCL IND L
179 "**"
180 STO \
181 X<> [
182 STO ]
183 ASTO [
184 ASHF
185 X<> \
186 ASHF
187 X≠Y?
188 X<> [
189 X=Y?
190 R↑
191 "⊢****"
192 STO \
193 "⊢"
194 X<> [
195 STO ]
196 ARCL c
197 X<> ]
198 STO IND L
199 RDN
200 X≠Y?
201 ISG L
202 X≠Y?
203 GTO 07
204 X<> Z
205 STO c
206 CLST
207 CLA
208 CLD
209 END
```

372 Bytes

Zeile 156 enthält den *Text* "ANUM", nicht die Funktion 'ANUM'!

Dezimalwerte der synthetischen Zeilen:

Zeile 04: 242, 132, 128
Zeile 05: 154, 126
Zeile 55: 242, 127, 0
Zeile 57: 243, 127, 64, 48
Zeile 122: 206, 126
Zeile 126: 144, 121
Zeile 127: 145, 120
Zeile 130: 206, 126
Zeile 158: 144, 122
Zeile 160: 145, 120
Zeile 161: 243, 127, 166, 66
Zeile 162: 154, 118
Zeile 163: 155, 118
Zeile 168: 144, 118
Zeile 172: 144, 117
Zeile 173: 144, 125
Zeile 174: 246, 64, 1, 105, 11, 2, 0
Zeile 175: 154, 125
Zeile 180: 145, 118
Zeile 181: 206, 117
Zeile 182: 145, 119
Zeile 183: 154, 117
Zeile 185: 206, 118
Zeile 188: 206, 117
Zeile 192: 145, 118
Zeile 193: 245, 127, 132, 132, 132, 240
Zeile 194: 206, 117
Zeile 195: 145, 119
Zeile 196: 155, 125
Zeile 197: 206, 119
Zeile 205: 145, 125

Bei Anwesenheit eines Time-Moduls oder auf dem HP-41CX kann man das Programm eine Spur schneller machen, wenn man die folgenden Änderungen vornimmt:

Zeile 156: "SW" (Text!); Zeile 161: 243, 127, 166, 154.

Das Programm "ASG" ist das vielleicht fortschrittlichste synthetische Programm, das bislang bekannt geworden ist, weil es eine breite Vielfalt synthetischer Techniken anwendet, um ein Höchstmaß an Benutzerfreundlichkeit zu bieten. Möge es Ihnen gefallen, dem Fachmann, der diese Meisterleistung zu würdigen weiß, ebenso wie dem Anfänger, den es dazu anregen kann, selbst Synthetiker zu werden.

10K. Der HP-41 bockt. Was tun?

Beim normalen Gebrauch des HP-41 braucht man nichts zu fürchten. Ein Ausfall des Rechners ist praktisch ausgeschlossen. Beim synthetischen Programmieren jedoch kann es lästige Erscheinungen geben. Die hinderlichsten unter ihnen nennt man 'GAU's (*g*rößte *a*nzunehmende *U*nfälle). Die Schwere des Unfalles betreffend gehören in aufsteigender Reihenfolge eine Zerstörung des Katalogs 1, ein "MEMORY LOST" und eine Tastenfeldblockade zu den unangenehmsten Ereignissen der in Rede stehenden Art. Es ist nicht schwer, den Eintritt eines GAUs festzustellen, doch die Genesung des HP-41 zu betreiben ist eine ganz andere Sache. "MEMORY LOST" ist bei den meisten GAUs eine Vorbedingung für den Versuch, den HP-41 wieder zur Mitarbeit zu bewegen. Sollte dieser 'flächendeckende Kahlschlag' nicht schon vom Rechner selbst herbeigeführt worden sein, muß man das, vorausgesetzt der Rechner reagiert überhaupt, von Hand nachholen (Einschalten bei gedrückter Korrekturtaste). Hat der HP-41 die Gesprächsbereitschaft mit der zu recht so gefürchteten Tastenblockade völlig aufgekündigt, setzt er also beliebiger Tastenbetätigung – 'R/S' und 'ON' eingeschlossen – eselsgleiche Sturheit entgegen, kann man folgende Überredungsversuche unternehmen:

0. *Aus*schalten bei gedrückter Korrekturtaste.

1. 'R/S' bei gedrückter Korrekturtaste betätigen; anschließend '←' loslassen.

2. Neuere Modelle (etwa ab 1982) erlauben es manchmal, die sogenannte Totallöschung, das ist das *Ein*schalten bei gedrückter Korrekturtaste, zur bloßen Aufhebung einer Blockade zu verwenden. Seien Sie aber vorsichtig: ältere Modelle reagieren ganz gewiß mit "MEMORY LOST" darauf, und auch die neueren Rechner wissen nur mit der Totallöschung zu antworten, wenn gar keine Blockade vorliegt.

3. Bei den meisten GAUs hilft es, die Batterien kurzzeitig zu entfernen.

4. Wer einen Kartenleser besitzt, kann versuchen, mit dessen Hilfe weiterzukommen. Man führt einfach eine Karte beliebigen Typs ein. Wird sie nicht ganz durchgezogen, entfernt man die Batterien ein paar Sekunden, wobei die auf halbem Wege steckengebliebene Karte in ihrer Lage verbleiben muß. Im Normalfall vollendet die Karte ihren Lauf beim Wiedereinsetzen der Batterien, und in der Anzeige erscheint dann die der Karte entsprechende Kartenleser-Meldung statt des befürchteten "MEMORY LOST". Diese Technik wurde von Clifford Stern entdeckt.

5. Entfernen Sie die Batterien, drehen Sie jede Zelle um, und setzen Sie sie dann wieder ein (bei einem aufladbaren Akku ist dies natürlich nicht möglich). Halten Sie die Taste 'ON' 10 Sekunden lang gedrückt. Stellen Sie danach die richtige Batterie-Polung wieder her, und drücken Sie alsbald abermals 'ON'. "MEMORY LOST" ist bei diesem Verfahren unvermeidbar, weil es hier das angestrebte Ergebnis bildet.

6. In hartnäckigen Fällen kann es helfen, die Batterien eine ganze Nacht lang zu entfernen. Ältere Modelle geben ihren Widerstand dann auf. Neuere Modelle sind leider unfügsamer und geben sich nicht so schnell geschlagen. Bei ihnen kann es vorkommen, daß sie die Gefolgschaft trotz gesperrter Energiezufuhr eine ganze Woche lang oder gar noch länger verweigern.

Nachtrag des Übers. für die Verzweifelten:

7. Wer sich an die Eingeweide seines Lieblings wagt, öffnet die Rückseite (Gummifüßchen abheben und die darunter sitzenden Kreuzschrauben lösen) und schließt den von der Rückseite aus gesehen links oben sitzenden *großen Kondensator* mit einer gewöhnlichen Büroklammer kurz. Dies muß zwar auf eigene Verantwortung geschehen und bringt natürlich die Gefahr eines Garantieausschlusses durch den Kundendienst mit sich. Aber es ist andererseits eine sichere Methode, den bockigsten HP-41 wiederzubeleben. Wer erst einige Male die Hemmschwelle, sich chirurgisch zu betätigen, überschritten hat, den verlassen alsbald alle Skrupel.

8. Für die Zaghaften unter den Besitzern höchst widerspenstiger Modelle des HP-41 gibt es seit einiger Zeit ein Erzeugnis, das Hardware-orientierte Benutzer entwickelt haben und das unter dem treffenden Namen 'Weckmodul' hier und da erworben werden kann. Ein solcher Weckmodul bietet die zuverlässigste, sauberste und bequemste Möglichkeit, jeden sich durch Blockade äußernden Trotz des Rechners sofort zu brechen. Ein kurzzeitiges Einstecken eines Weckmoduls bei gleichzeitig entfernten Batterien bringt jedem

HP-41 im Handumdrehen den schuldigen Gehorsam bei. Im Anhang C finden Sie eine Adresse, unter der Sie einen Weckmodul bestellen können.

Anm. d. Übers.: Brandneu ist folgende vom HP-Club Austria gemachte 'Entdeckung': Die gleichzeitige Betätigung der Tasten 'ENTER↑' und 'ON' *unterbricht* in Rechnern mit einer Seriennummer größer als 2200 *jede Funktion*, insbesondere also auch die gefürchteten Tiefschlaf-Schleifen, die das Tastenfeld lahmlegen. Die 'ENTER↑ & ON'-Funktion könnte sich – auf neueren Modellen des HP-41 – als *der* Rettungsanker erweisen. Man sollte sie *Blockadebrecher* taufen. Um festzustellen, ob Ihr Rechner den Blockadebrecher enthält, brauchen Sie nur zu versuchen, ob sich die 'LOG'-Funktion bei nicht-normalisiertem Argument (vgl. 'Wickes', S. 109, vorletzte Zeile) unterbrechen läßt.

Wenn der Katalog 1 sinnlose Einträge vorweist, das Tastenfeld aber wenigstens arbeitsfähig geblieben ist, werden Sie in den sauren Apfel der Totallöschung beißen müssen. Nur Besitzern eines PPC ROMs (s. Anhang C) bleibt noch eine Hoffnung, den Speicherinhalt zu erhalten und wieder in einen zugriffsfähigen Zustand gesetzt zu bekommen. Altmeister Clifford Stern weiß – wie so häufig – auch hier zu raten: Setzen Sie die Zeichenkette "C0002D" ins Alpha-Register (statt der 3 Nullen können Sie auch 3 Leerstellen nehmen), und führen Sie dann nacheinander 'XEQ "HN"' und 'XEQ "E?"' aus. Liegt das Ergebnis von "E?" unterhalb von 192 oder oberhalb von 511, müssen Sie aufgeben und endgültig die Flucht nach vorn ins nicht mehr zu vermeidende "MEMORY LOST" antreten. Andernfalls können Sie mit 'XEQ "SX"' fortfahren, wofür der Stapel in dem Zustand, in dem er von "HN" und "E?" zurückgelassen wurde, unbedingt verbleiben muß. Abschließend ist noch ein 'PACK' erforderlich. Prüfen Sie danach, ob der Katalog 1 Ihre Programme wieder anzeigt.*) Für sicheren Erfolg bürgt dieses Verfahren aber leider auch nicht. Es schlägt beispielsweise fehl, wenn der Zeiger für das '.END.' oder die Lage des Registers R_{00} abgeändert wurden. In einem solchen Fall können sich allein noch diejenigen helfen, die erstens einen PPC ROM ihr eigen nennen, zweitens über eine genaue Kenntnis des Registers c verfügen, drittens durch die Charaktereigenschaft der Beharrlichkeit ausgezeichnet sind und viertens 'fortune' – wie Friedrich der Große gesagt haben würde – besitzen.

*) Anm. d. Übers.: In ungünstigen Fällen können bis zu vier Bytes des letzten Programms verloren gegangen sein.

KAPITEL 11

Geheimes auf dem HP-41

(Nachtrag des Übersetzers)

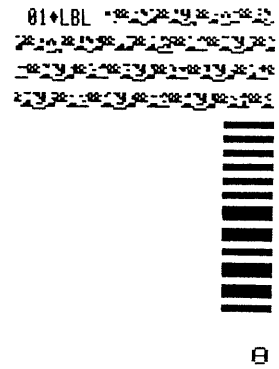
Gedanken zu verbergen ist jedem ein leichtes. Sehr viel anders steht es dagegen mit Nachrichten, die in irgendeiner Form, etwa schwarz auf weiß, greifbare Gestalt angenommen haben. Denn die Bedeutung einer schriftlichen Botschaft auf Dauer geheim zu halten, ist so gut wie ausgeschlossen. Zwar sind die Versuche, Texten durch Verschlüsselung ihren offenkundigen Sinn zu nehmen, uralt – vom großen Julius Cäsar sogar stammt eine der einfachsten Kodierungen, die sogenannte Verschiebechiffrierung –, doch ebenso alt und im Grunde erfolgreicher waren die unermüdlichen Versuche Außenstehender, den Sinn verschlüsselter Texte aufzudecken. Einfache Kodierungen erliegen bereits dem ersten Angriff, dem durch Häufigkeitsanalyse, das ist die Auszählung der Häufigkeit des Auftretens von Zeichen oder Zeichengruppen im Chiffretext und die Auswertung der sich daraus ergebenden Schlußfolgerungen. Wegen der entnervenden Erfolge der Verfolger wurden im Laufe der Zeit immer sinnreichere Verfahren zur Verschlüsselung ersonnen. Pseudo-Zufallsgeneratoren fanden Einlaß in die Kodierungsverfahren, um wenigstens die Gleichverteilung der Zeichen im Chiffretext zu erreichen, so daß mindestens die gefährlichen Häufigkeitsanalysen versagen mußten. Doch es war wie in der Evolution, wo Tarnfarben der Beutetiere unweigerlich zu schärferen Augen der Räuber führten. Sobald einem Angreifer längerer zusammengehöriger Klar- und Chiffretext in die Hände fiel, konnte er Koinzidenztabellen aufstellen, und alsbald war alle Mühe, das zu verschlüsseln, was verborgen gehalten werden sollte, umsonst.

Das Aufkommen moderner Rechenanlagen brachte eine erhebliche Ausweitung der Verfahren der Chiffrierer und einen entsprechenden Zuwachs der Analysemöglichkeiten für die Entzifferer mit sich. Texte, die dem menschlichen Auge unentwirrbar erscheinen müssen, weil es durch ständiges Lesen 'vernünftiger' Wörter und Sätze eine praktisch unüberwindbare Prägung auf sinnvolle Buchstabenkombinationen erfährt, lassen sich von einem Rechner, der ja ohne jedes persönliche Vorurteil arbeitet, ohne Schwierigkeiten auf komplizierte Gesetzmäßigkeiten durchsehen. Folglich hat es wenig Sinn, sich der Illusion hinzugeben, man könnte auf dem HP-41 Chiffretexte erzeugen, die einem mit hoher Rechenkapazität ausgestatteten Experten ohne Aussicht auf Entzifferungserfolg überlassen werden dürften (einmal ganz abgesehen davon, daß sich ein solcher Experte, der übrigens meist sehr suspekten Institutionen angehört, sowieso nicht dazu herbeiläßt, Geheimschrift aus einem HP-41 in Augenschein zu nehmen, obwohl er gar nicht weiß, was ihm da entgeht!). Nichtsdestoweniger kann man das Problem als intellektuelle Herausforderung ansehen und ungeachtet aller eigentlich entmutigenden Erkenntnis unbefangen angehen. Nachfolgend finden Sie das Ergebnis einer solchen unverdrossenen Bemühung.

HP-41-Benutzer, die der Sache mit ein wenig Interesse gegenüberstehen, werden vielleicht ihre Freude daran haben, weil sie bei der Beurteilung der Güte der Verschleierungsfähigkeit des angewendeten Verfahrens allein auf ihre unvollkommenen Augen, die ihnen nur das nackte Chaos melden können, angewiesen sind.

Dem Charakter des Themas angemessen wird das folgende Ver- und Entschlüsselungsprogramm — aus rein 'sportlichen' Gründen versteht sich — in undurchsichtiger Verpackung und mit 'Selbstschußanlage' versehen geliefert, jedoch natürlich so, daß seine volle Einsatzfähigkeit gewährleistet ist:

1. Hier der 'Programmausdruck' auf einem HP-IL-Drucker:

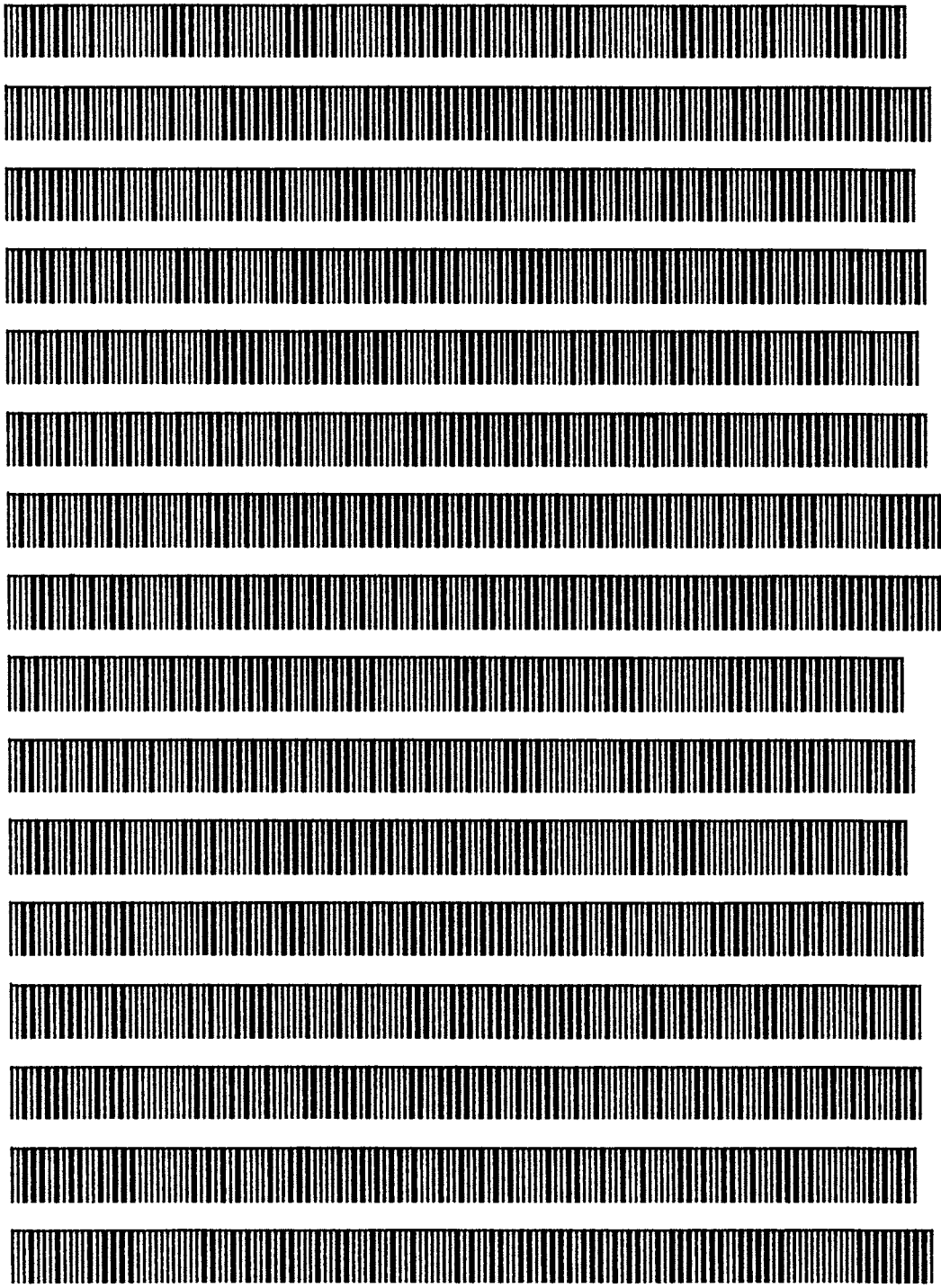


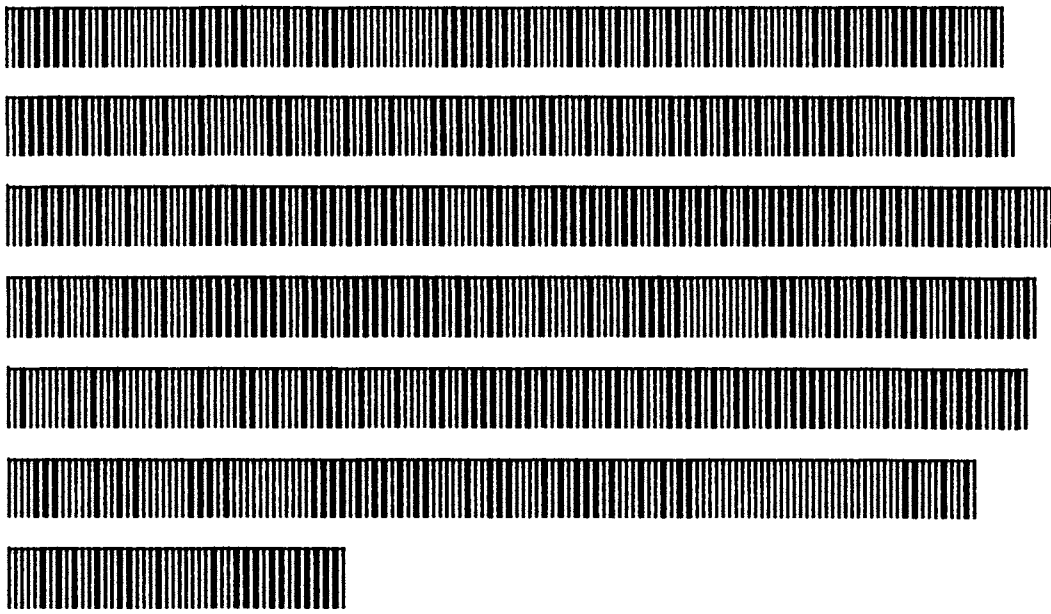
Seinen Namen gibt das Programm überhaupt nur jenen preis, die sich auf die Herstellung hexadezimaler Speicherauszüge verstehen.

2. Der untenstehende Barcode, den freundlicherweise Konrad Albers lieferte, ist selbstverständlich privatisiert. Darauf hat, das wird jeder einsehen, ein solches Programm einen gewissermaßen 'einklagbaren Anspruch'. Synthetikern, denen diese Hürde nur ein Lächeln abzwingt, bleibt immerhin noch — Anteilnahme am Problem vorausgesetzt — die Zeilen-Analyse auf eigene Faust.

3. Schließlich ist das Programm noch mit einigem nutzlosen aber stilgerechten Zierat versehen: a) Wer an Hand der geschützten Form zu ergründen sucht, welcher lokalen Marken sich das Programm bedient, wird sich Mühe geben müssen, fündig zu werden. Sie sind ebensowenig dingfest zu machen, wie es möglich ist, Reste von Rechenspuren aufzutreiben. b) Die Aufforderung zur Texteingabe verharrt *während der Texteingabe* in der Anzeige, was bekanntlich eigentlich gar nicht geschehen kann. c) Das Programm arbeitet bei 'verstummter' Anzeige: nur der einsame PRGM-Indikator hält während des Programmlaufes tapfer Wache. d) Jeden Versuch, Zwischenergebnisse zu betrachten, weist das Programm empört mit "MEMORY LOST" zurück.

Barcode des Kryptographieprogramms
Speicherplatzbedarf: 41 Register, 284 Bytes





Gebrauchsanweisung für das Kryptographieprogramm

0. Lesen Sie den Barcode *bei eingeschaltetem USER-Modus* ein: Da sich nämlich weder die lokalen noch die globalen Marken 'in die Karten schauen' lassen, hat man *nur im USER-Modus* Zugriff auf das Programm, und zwar mit 'USER, Σ +' auf den Verschlüsselungsteil und mit 'USER, 1/x' auf den Entschlüsselungsteil.
1. Im Register R_{10} muß eine – nicht notwendig ganze – Zahl *zwischen – 10 und + 21259* liegen, der sogenannte *Schlüssel*. Eine Fehlerfalle gegen außerhalb der Grenzen liegende Schlüssel ist nicht eingebaut, weil man sich über den verwendeten Schlüssel sowieso stets im klaren sein muß, denn Entzifferungsversuche haben selbstredend nur dann Erfolg, wenn sie mit demselben Schlüssel wie die ursprüngliche Chiffrierung vorgenommen werden.
2. Seine volle Wirkung, soll heißen: sein totales Verwirrspiel, entfaltet das Programm nur bei Anwesenheit eines Time-Moduls. Um aber Leser ohne solchen Modul nicht 'vor der Tür stehen zu lassen', ist das Programm so abgefaßt, daß es ohne Time-Modul ebenfalls verwendbar ist, wenn auch in minderer Güte: ohne Time-Modul taucht zu einem Klartext nämlich immer nur *ein* Chiffretext auf, wohingegen ein Time-Modul dafür sorgt, daß abhängig vom Zeitpunkt der Verschlüsselung aber unbeeinflußbar vom Benutzer *verschiedene* Chiffretexte, die *alle* auf den ursprünglichen Klartext zurückführen, entstehen.
3. Der Verschlüsselungsteil chiffriert Texte bis zur Länge 23 (Achtung: *nicht bis zur Länge 24*). Die Texte dürfen folgende Zeichen enthalten:

þ	!	″	#	$	%	&	'	(	)	*	+	,	–	.
/	Ø	1	2	3	4	5	6	7	8	9	:	;	<	=
>	?	@	A	B	C	D	E	F	G	H	I	J	K	L
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	

Die darunter befindlichen 9 'nichttastbaren' Zeichen

! " # & ' ( ) ; @

liegen nach dem ersten Aufruf des Programms zum Abruf durch 'ARCL' in den Registern R_{01} bis R_{09} in der aufgeführten Reihenfolge bereit. Das Zeichen "þ" steht in der voranstehenden Tabelle für die Leerstelle 'SPACE'. Auf 'USER, Σ + ' hin wird man mit "VERSCHL:" zur Eingabe von Klartext aufgefordert. Der Entschlüsselungsteil dechiffriert Chiffretexte, die durch den Verschlüsselungsteil entstanden sind. Er wird mit 'USER, 1/x' aufgerufen, fordert mit "ENTSCHL:" zur Eingabe von Chiffretexten auf und legt ebenfalls die nichttastbaren Zeichen zu Beginn in die Register R_{01} bis R_{09}. Weil Chiffretext grundsätzlich ein Zeichen mehr als der zugehörige Klartext enthält, können Chiffretexte – im Gegensatz zu Klartexten – gelegentlich die volle Alpha-Länge 24 haben. Bei angeschlossenem Drucker *und* gesetztem Flag 21 werden die Ergebnisse der Ver- bzw. Entschlüsselung ausgedruckt. Hat man mehrere Klar- oder Chiffretexte vorliegen, die nacheinander bearbeitet werden sollen, kann man – im gleichen Programmteil verbleibend – mit 'R/S' fortfahren. Die Aufforderungen "VERSCHL:" bzw. "ENTSCHL:" zeigen zuverlässig an, ob alles wunschgemäß verläuft.

4. Sonderfall 'SPACE': Weil das Leerzeichen gelegentlich *als letztes Zeichen* eines Chiffretextes entstehen kann, dort aber nicht ohne weiteres zu erkennen wäre, wird ein solches Leerzeichen in der Anzeige durch die Überstreichung "¯", beim Druck durch die Raute "◆" ersetzt (der 7. der weiter unten tabellierten 39 Chiffretexte bildet ein Beispiel dafür). Beim Entschlüsseln ist jedoch ordnungsgemäß 'SPACE' einzutasten. *Führende* Leerzeichen und Leerzeichen *innerhalb* des Textes sind von dieser Regelung nicht betroffen.

WARNUNGEN: a) Setzen Sie die erforderliche Schlüsselzahl *vor* Aufruf des Programms ins Register R_{10}. Sie können dann sicher sein, daß das Programm nicht zu ungelegener Zeit anhält, um mit "NO ROOM" Speicherplatz nachzufordern oder gar "ALPHA DATA" zu bemängeln, um danach alles zu zerstören (s. Punkt d). – b) Die Datenregisteranzahl wird vom Programm grundsätzlich auf 11 festgesetzt. Beachten Sie das, damit Ihnen keine wichtigen Zahlen in höher numerierten Registern überraschend verloren gehen. – c) Sobald Sie eine der Aufforderungen "VERSCHL:" oder "ENTSCHL:" vorgelegt bekommen haben, dürfen Sie auf keinen Fall mehr den Stapelinhalt verändern. Insbesondere dürfen Sie zu diesem Zeitpunkt keine Schlüsselzahl mehr eingeben oder das Alpha-Register mit 'XTOA' beschicken. Verbleiben Sie im ALPHA-Modus, und greifen Sie auf 'ARCL 01' bis 'ARCL 09' zurück, wenn Sie die nichttastbaren Sonderzeichen benötigen. – d) Und schließlich die wichtigste Regel: *Unterbrechen Sie den Programmlauf unter gar keinen Umständen*, nachdem Sie die Ver- bzw. Entschlüsselung mit 'R/S' gestartet haben. Das Programm ahndet nämlich jede Absicht, auch nur einen verstohlenen Blick auf Zwischenergebnisse zu werfen, *augenblicklich* mit einem unnachsichtigen "MEMORY LOST".

Und nun eine Aufgabe, deren *ersteingesandte* Lösung mit einem kostenlosen Exemplar der nächsten HP-41 Publikation des Heldermann Verlages, einem Buch über Barcodes von Konrad Albers, ausgepreist ist. Fassen Sie den Text

4NGU8F, #A$%ØF', PPX?C

als 'Klartext' auf, auch wenn sich alles in Ihnen dagegen sträubt, solch ein Gebräu als Klartext

anzuerkennen, und *ver*schlüsseln Sie ihn mit dem Schlüssel 71.71 in R_{10}. Wenn Sie keinen Time-Modul haben, erzeugen Sie beim Verschlüsseln leider nur immer einen Chiffretext – und zwar nicht den gesuchten (s.u.) – und müssen aufgeben, es sei denn, Sie entprivatisieren das Programm, analysieren seine Wirkungsweise und lösen dann die Aufgabe. Time-Modul-Besitzer stoßen beim Verschlüsseln auf die verschiedensten Chiffretexte. Nachstehend als Hilfestellung eine Auswahl aus den Chiffretexten, die Ihnen dabei begegnen *können* (nicht müssen!):

```
UI!)+Y50B!U1*0!,K@$W%
+$4"&:)UBV/&'3E@ZPN+&
+7K)N,J5:SB!MP?.:Q8%6
TDS0TCULZ4);-Q=D$P0$(
6J5FXLJT(!8,'=4.)6"$5
FZCT)XU$1)?2,A7/)J56F
,Y!0X(0R76K4H42V?FX9+
)P7ETEAGTI"NFYM1@J42@
2VGR@2GK,$*!+FNT?327%
.XQU$R)SIYK3.6XT%!0P!
;4V3*5Q"A5Q219M,/PNT9
F''=!Y95UJZC=&$1*$;"0
KS*BD)&*II6CZOY9C&T3N
```

```
$L7QT6-4+6QX$J)=B0)'S
8I3-R0"XR/QC;XX4SR28B
(5E!S,9VEM0"@5(/!2L7(
/Y,D,U,"/C8H:1X$=-H;?
V2':;?,4PB$SXI;J)M=PH
E,$E?JA7L0Z0-*)ZU2(E8
KO5=NJ%PP@CWYV9&YW)D&
2TP!Y-MY*TMQ O&OEUG)*
DE?1H)I*5SG'B@I!:("Z$
8@7$,I;WLRD6E$I9MDS/%
J6AUW8.4)2LRXA315".QA
IU9.J+*3G')LJP57BM*!-
-"2V085@;5L2$V65L/,GC
```

```
45@QL*ZX4.D8G59LO+R+)
L@R3ZL''3LDYE,W&E9XPX
FES?ART-R):/-7F?VJE R
JEW70FG)/: P5.7,!5R?3
7&;PIKY1.I@@YUH8I:?TZ
C/.=9508,%CE$DD%IRJ G
+$3Q$F=!?@L'DI23;$'JL
D-Y)QKY!)5(3=X$+PDCI7
X,E.W/&4I)PL(PV7(D7(-
M(N?F!'A0F"VY-A) ST3/
+XZLP+-/O)87 (G39!NPN
5!2&1D7FOM8(UC!NP),90
8)WG"925DXU7/0JFPR*V!
```

Es gibt nun noch weitere Chiffretexte, die beim *Ver*schlüsseln von "4NGU8F,#A$%ØF',PPX?C" entstehen können, und darunter befindet sich einer, der in dem Moment, in dem man ihn erblickt, sofort als Lösung der Aufgabe erkannt wird. Wer Glück hat, der findet den gesuchten 'Chiffretext', welcher seinem Erscheinungsbild nach nur als 'klarer Text' aufgefaßt werden kann, beim ersten Anlauf. Wer Pech hat, wird lange lange suchen müssen. Sie sind also auf Gedeih und Verderb der 'Metze Fortuna' ausgeliefert, 'ganz wie im richtigen Leben'. Wie dem auch sei, ich halte es jedenfalls für möglich, daß Sie nach einigem Umgang mit dem Programm doch den Eindruck gewinnen werden, daß der HP-41 der eingangs angesprochenen Herausforderung immerhin leidlich gut gewachsen ist.

Wenn Sie das Programm auf Magnetkarten bringen wollen, aber keine Entprivatisierung vornehmen können, brauchen Sie sich nur des Programms "WFL" aus Abschnitt 10I zu bedienen. Wer keinen optischen Lesestift zur Verfügung hat, kann das Programm gegen Einsendung *zweier* Magnetkarten, denen ein *beschrifteter und freigemachter* Umschlag beigefügt sein muß, beim Übersetzer anfordern. Auf ausdrücklichen Wunsch wird dabei die 'offene Form' verschickt, dies allerdings erst, nachdem die Lösung der Aufgabe beim Übersetzer oder beim Verlag eingegangen ist.

LÖSUNGEN DER AUFGABEN

3.1.

```
01 "*"
02 APPREC
03 DELREC
04 RCLPT
```

3.2. Hier eine Lösung:

```
01♦LBL "ASDR"                    (ASCII-Datei drucken)
02 CLX
03 SEEKPTA
04 SF 25
05♦LBL 01
06 GETREC
07 FC? 25
08 RTN
09 ACA
10 FS? 17
11 GTO 01
12 PRBUF
13 ADV
14 GTO 01
15 END
```

4.1. Unter Berücksichtigung besonders schneller Befehle läßt sich die Aufgabe so lösen:

```
01 XTOA
02 SIGN   } vgl. S. 191
03 CHS    }
04 AROT
```

4.2. Bei der folgenden Lösung muß die Adresse des ersten Datenregisters beim Start in X liegen:

```
01 ,1                07 RDN
02 +                 08 ISG X
03 ENTER↑            09 ALENG
04♦LBL 01            10 X>0?
05 ASTO IND Y        11 GTO 01
06 ASHF
```

4.3a. 3 Zeichen von links aus löschen:

```
01 3
02*LBL 01
03 ATOX
04 RDN
05 DSE X
06 GTO 01
```

4.3b. 4 Zeichen von rechts aus löschen:

```
01 4
02 CHS
03 AROT
04 CHS
05*LBL 01
06 ATOX
07 RDN
08 DSE X
09 GTO 01
```

4.4. Die folgende Routine erwartet einen mit einem oder zwei Vornamen versehenen Nachnamen im Alpha-Register und stellt die Namen dann so um, daß anschließend der Nachname die Gruppe anführt und außerdem durch ein Komma von dem oder den Vornamen abgetrennt ist.

```
01*LBL "VTN"      (Vertauschen von Namen)
02 32             } erste Leerstelle aufsuchen
03 POSA           }
04 "⊢, "          Komma und Leerstelle anhängen
05 AROT           ersten Vornamen nach hinten setzen
06 ATOX           Leerstelle, die hinter dem ersten Vornamen stand, beseitigen
07 POSA           nächste Leerstelle aufsuchen
08 44             } Komma aufsuchen
09 POSA           }
10 X<>Y           }
11 X<Y?           } falls noch eine (ursprünglich zweite) Leerstelle
12 X<0?           } vor dem Komma steht, 'RTN' überspringen
13 RTN            }
14 "⊢ "           Leerstelle hinter den ersten Vornamen setzen
15 AROT           zweiten Vornamen nach hinten bringen
16 ATOX           Leerstelle, die hinter dem zweiten Vornamen stand, beseitigen.
17 END
```

4.5.

```
01 PI
02 *
03 RCLFLAG
04 X<>Y
05 RAD
06 SIN
07 X<>Y
08 STOFLAG
09 X<> L
10 /
```

4.6. Die folgende Routine "FE" ('*F*IX/*E*NG') setzt *zugleich* die Flags 40 ('FIX') und 41 ('ENG'), während die Ziffernflags 36 – 39 unangetastet bleiben. Sie ist der Routine "FEX" aus Abschnitt 4C ähnlich mit dem Unterschied, daß jene Routine die Nachkommastellen der Vorgabe des Benutzers gehorchend festlegt, während hier ein zusätzliches 'RCLFLAG' gleich am Anfang erforderlich ist, um den Lauf tatsächlich mit dem jeweils bei Programmbeginn geltenden Zustand der Flags 36 – 39 beenden zu können.

```
01♦LBL "FE"
02 RCLFLAG          Zustand der Flags 0 – 39 im Stapel aufbewahren
03 ENG 0            Flag 41 setzen
04 RCLFLAG          Zustand von Flag 41 im Stapel aufbewahren
05 FIX 0            Flag 40 setzen
06 X<>Y
07 ,39          }
08 STOFLAG      }   Zustand der Flags 0 – 39 wiederherstellen
09 R↑           }
10 R↑           }   schneller als 'RDN, RDN'
11 41           }
12 STOFLAG      }   Zustand von Flag 41 wiederherstellen
13 R↑
14 R↑
15 END
```

Beachten Sie, daß die Lösung eine verhältnismäßig umständliche Form deswegen haben muß, weil kein bestimmtes Format bei Programmbeginn vorausgesetzt werden soll. Könnte man beispielsweise sicher sein, daß stets das 'FIX'-Format eingestellt ist, wenn "FE" seinen Lauf beginnt, leistete eine wesentlich kürzere Befehlsfolge dasselbe:

```
01 RCLFLAG
02 ENG 0
03 ,4
04 STOFLAG
05 R↑
06 R↑
```

4.7. Die nachfolgende Routine "BR" (*B*lock *r*otieren) bildet eine von mehreren Lösungen. Die ersten 10 Zeilen des Programms stellen die Zahl

$$1{,}001 \cdot abc + 0{,}000001 \cdot (lmn - 1)$$

her. Die Zeilen 11 – 21 addieren darauf 1 im Falle $r < 0$, dagegen 0,001 im Falle $r > 0$. Anschließend enthält X

$$abc{,}(abc + 1)(lmn - 1) \text{ für } r > 0$$

bzw.

$$(abc + 1){,}abc(lmn - 1) \text{ für } r < 0.$$

Der absolute Wert von r (Zeile 14) gelangt nach Y, wo er als 'DSE'-Zähler für die von 'LBL 01' angeführte Schleife dient.

Programmausdruck von "BR"
(Barcode im Anhang D)

```
01♦LBL "BR"      08 1 E-6         15 X<> T         22♦LBL 01
02 ,1            09 *             16 SIGN          23 REGSWAP
03 %             10 ST+ Y         17 CHS           24 DSE Y
04 +             11 X<> L         18 X>0?          25 GTO 01
05 X<>Y          12 SQRT          19 X<>Y          26 END
06 1             13 R↑            20 RDN
07 -             14 ABS           21 +                  43 Bytes
```

ANHANG A

Die 'VER'-Wanze und die '7CLREG'-Wanze

Wenn Sie einen Kartenleser der Version 1G oder höher haben, können Sie gleich zu den weiter unten stehenden Ausführungen zur '7CLREG'-Wanze übergehen. Die Version des Kartenlesers läßt sich ganz einfach dadurch ermitteln, daß man 'CAT 2' aufruft. Sofern Sie daraufhin einen der Kopfeinträge

CARD READER
CARD RDR 1D
CARD RDR 1E
CARD RDR 1F

vorgewiesen bekommen, sitzt in Ihrem Kartenleser die 'VER'-Wanze. Sehen Sie hingegen

CARD RDR 1G,

haben Sie einen Kartenleser ohne 'VER'-Wanze. Hier die genaue Beschreibung der 'VER'-Wanze (für Kartenleser bis zur Version 1F): Sobald die Kartenleserfunktion 'VER' mit einem in der Einschubbuchse 2 (die Nummern der Buchsen finden Sie auf der Rückseite des Rechners) steckenden X-Memory-Modul aufgerufen wird, erfährt der Inhalt des ersten Registers dieses Moduls eine Änderung. Dasselbe Unglück geschieht, wenn der Modul im Steckplatz 4 einer Buchsenerweiterung sitzt oder Teil eines Doppelmoduls ist (vgl. Anhang C, Punkt 7).

Die beschriebenen Bedingungen vorausgesetzt, wird der Inhalt des auf der Speicherstelle 1007 liegenden Registers beim Gebrauch von 'VER' abgeändert, es sei denn, der Modul auf Platz 2 enthält überhaupt keine Daten. Es ist übrigens im ungünstigsten Falle sogar möglich, daß das gefährdete Register gar das zweite Kopfregister einer Datei bildet, woraus sich eine als besonders schmerzhaft zu beurteilende Beschädigung des X-Memory Verzeichnisses ergibt. Schlagen Sie im Zweifelsfall noch einmal in Abschnitt 10C nach, um zu verstehen, was geschieht, wenn die 'VER'-Wanze zusticht.

Der 'VER'-Wanze können Sie sich leicht dadurch erwehren, daß Sie überhaupt keinen X-Memory-Modul auf Platz 2 stecken. Platz 4 ist sowieso durch den Kartenleser besetzt, und ein X-Memory-Modul auf Platz 1. oder 3 ist nicht bedroht. Haben Sie dagegen zwei X-Memory-Module nötig, müssen Sie entweder einen auf Platz 1 oder 3 sitzenden Modul erst mit Dateien vollaufen lassen, bevor Sie den zweiten auf Platz 2 setzen, oder Sie stecken beide

– bei ausgeschaltetem Rechner versteht sich – gleichzeitig ein. Wenn Sie diese Empfehlung beherzigen, können Sie wenigstens das durch die 'VER'-Wanze gefährdete Register an wohlbestimmter Stelle vorfinden: es ist, wenn Sie wie beschrieben verfahren, genau das 366. Register des X-Memorys. Um festzustellen, zu welcher Ihrer Dateien dieses Register gehört, brauchen Sie nur das X-Memory-Verzeichnis durchzusehen und den vorgewiesenen Dateigrößen zwei Register je Datei (für die beiden Kopfregister) zuzurechnen. Sie können dadurch genau feststellen, in welcher Datei und an welcher Stelle in der ermittelten Datei Zerstörungen zu befürchten sind. Somit läßt sich im Anschluß an die Ausführung von 'VER' die betreffende Datei überprüfen und, falls sich eine Reparatur als unmöglich erweist, gezielt tilgen. Ist das 366. Register unglücklicherweise das zweite Kopfregister einer Datei, erlischt der Zugriff auf diese und alle folgenden Dateien fast immer ganz (vgl. Abschnitt 10C).

Gute Nachricht nun für diejenigen, die Abschnitt 10D noch nicht gelesen haben sollten: Es ist möglich, der Zerstörung des 366. Registers völlig aus dem Wege zu gehen, indem man das in Abschnitt 10D beschriebene synthetische Programm "VER" benutzt. Es benötigt zur Ausführung weniger Zeit als die, die Sie aufwenden müssen, um 'XEQ "VER"' zu tasten.

Falls Sie eine Buchsenerweiterung (vgl. Anhang C, Punkt 7) angeschlossen haben, gibt es noch eine andere ebenso einfache Möglichkeit, der 'VER'-Wanze zu entgehen: Schalten Sie die X-Funktionen und die X-Memory-Module ab, bevor Sie 'VER' ausführen.

In *allen* Kartenlesern sitzt die '7CLREG'-Wanze. Die Kartenleserfunktion '7CLREG' dient dazu, die Übertragungsfähigkeit von HP-67/97 Programmen, welche die Funktion 'CLREG' enthalten, auf den HP-41 sicherzustellen; sie simuliert die HP-67/97 Funktion 'CLREG'. Die '7CLREG'-Wanze bringt es fertig, den Inhalt eines ganzen X-Memory-Moduls zu verderben. Wenn Sie '7CLREG' nämlich bei weniger als 25 zur Verfügung stehenden Datenregistern aufrufen, gehen Daten in der Umgebung 360. Registers des X-Memorys verloren, vorausgesetzt die zur 'VER'-Wanze gegebenen Empfehlungen zur Nutzungsreihenfolge der X-Memory-Module sind befolgt worden, so daß der auf Platz 1 oder 3 sitzende X-Memory-Modul das 365. Register als letztes Register enthält. Überdies entziehen sich nach der Benutzung von '7CLREG' unter der genannten Datenregisterbedingung fast immer alle Daten vom 366. Register an aufwärts dem Zugriff. Als Gegenmaßnahme bleibt nur, entweder '7CLREG' völlig zu vermeiden oder diesem Befehl die Folge 'SIZE?, 25, X>Y?, PSIZE' (vgl. Abschnitt 4D) voranzuschicken, damit sichergestellt ist, daß die Datenregisteranzahl mindestens 25 beträgt.

ANHANG B

Die Ausführungsdauer von X-Funktionen

Jedesmal, wenn Sie ein Programm schreiben, stellen Sie fest, daß Sie das erstrebte Ergebnis auf verschiedenen Wegen erhalten können. Als nützliche Entscheidungshilfe bei der Auswahl von Lösungswegen soll die auf den nächsten Seiten abgedruckte Liste, der Sie die Ausführungszeiten der verschiedenen X-Funktionen entnehmen können, dienen. Wollen Sie beispielsweise – um zunächst noch bei den Standardfunktionen zu verbleiben – eine Eins ins Register X bringen, so bietet es sich gelegentlich an, dafür die scheinbar umständliche Folge 'CLX, SIGN' zu wählen, weil sie um ewa 37 Millisenkunden, mithin 62% schneller ist als der gewöhnlich verwendete schlichte Zahleneintrag 1. Dies kann trotz des zusätzlich benötigten Bytes sehr günstig sein, dann nämlich, wenn diese Eins in einer vielfach durchlaufenen Schleife einzutragen ist.

Eine Liste über die Ausführungszeiten der wichtigsten Funktionen des Katalogs 3, das sind die eingebauten Standardfunktionen, finden Sie in dem Buch 'Synthetisches Programmieren auf dem HP-41 – leicht gemacht' (s. Anhang C, Punkt 2). Ihr ist das eben erwähnte Zahlenbeispiel entnommen. In der nachstehenden Liste sind die Ausführungszeiten der meisten X-Funktionen enthalten, insbesondere derjenigen Funktionen, bei denen Sie i.a. Wahlmöglichkeiten haben, so daß Sie in Bezug auf die Laufzeit Ihrer Programme durch einfache Befragung der Liste stets beste Entscheidungen zu treffen in der Lage sind.

Die Ausführungszeiten sind von Clifford Stern ermittelt worden, der dafür ausgiebig sein – ebenfalls in 'Synthetisches Programmieren auf dem HP-41 – leicht gemacht' enthaltenes – den Time-Modul verwendendes Meßprogramm bemüht hat. Obwohl die eigentlichen Messungen damit vollständig programmgesteuert erfolgen, hat die Herstellung der Liste viel Arbeit mit sich gebracht, weil die Anzahl der bei den Messungen der X-Funktionen zu berücksichtigenden Variablen beträchtlich ist.

Liste der Ausführungszeiten der X-Funktionen

(alle Zeiten in Millisekunden; Drucker nicht angeschlossen)

ALENG	$168 - 3.5 \cdot C - 5.8 \cdot ((C - 1)/7)$ C die Anzahl der Zeichen im Alpha-Register
ANUM	zwischen 95 und 390 (nicht näher vorausbestimmbar)
APPCHR	$237 + 10.2 \cdot C + 12.1 \cdot S + I$ C die Anzahl der anzufügenden Zeichen, S die Satznummer,

I das Datei-Inkrement
[ein Datei-Inkrement lautet $12.9 \cdot (N - 1) + 9 \cdot E$ für eine gegenwärtige und $136 + 12.3 \cdot (N - 1) + 9 \cdot E$ für eine im Alpha-Register benannte Datei wobei N die Nummer der Datei im X-Memory Verzeichnis ist (von 1 an aufwärts) und E die Nummer des Blockes des X-Memorys, in dem die Datei liegt, bezeichnet ($0 \leq E \leq 2$, vgl. Abb. 10.1 in Abschnitt 10C)]; die Formel setzt voraus, daß S der letzte Satz der Datei ist, andernfalls arbeitet 'APPCHR' langsamer aber nicht näher vorausbestimmbar

APPREC $211 + 6.9 \cdot C + 12.1 \cdot S. + I$

ARCLREC $458 + 12 \cdot S + I$

AROT $x > 0$: $286 - 7.77 \cdot C - 6.6 \cdot INT((C - 1)/7) - 10.9 \cdot x$
$x < 0$: $287 - 7.77 \cdot C - 6.6 \cdot INT((C - 1)/7) - 10.9 \cdot (C - |x|)$

ATOX $x = 0$: 145
$x > 0$: $167 - 4.28 \cdot C$

CLFL $199 + 3.24 \cdot D + I$
D die Dateigröße

CLKEYS $326 + 12.2 \cdot G$
G die Anzahl der globalen Marken im Katalog 1

CRFLAS $246 + 3.6 \cdot D$

CRFLD $246 + 3.6 \cdot D$

DELCHR nicht vorauszubestimmen aber langsam

DELREC nicht vorauszubestimmen aber langsam

FLSIZE $72.9 + I$

GETP $1099 + 115 \cdot D$

GETR $58.5 + 12.5 \cdot D + 2.5 \cdot (A - D) + I$
A die Datenregisteranzahl

GETREC $350 + 12.1 \cdot S + I$

GETRX $110 + 9.9 \cdot R + I$
R die Anzahl der betroffenen Datenregister

GETX $63.5 + I$

INSCHR nicht vorauszubestimmen aber langsam

INSREC nicht vorauszubestimmen aber langsam

PASN hängt von der Suchzeit nach der benannten Marke oder Funktion in den Katalogen 1, 2 bzw. 3 ab

PCLPS $700 + P + 16 \cdot G$
P die Anzahl der gelöschten Programmregister

POSA zwischen 83.5 und 281

POSFL $190 + 17 \cdot C + 24.5 \cdot S + I$
C die Länge der gesuchten Zeichenkette,
S die Anzahl der durchsuchten Sätze

PSIZE $746 + 4.1 \cdot x$

PURFL $270 + I$
die Formel setzt voraus, daß die letzte Datei im X-Memory gelöscht wird, andernfalls ist 'PURFL' langsamer aber nicht näher vorausbestimmbar

RCLFLAG 36.8

RCLPT(A) $102 + I$

REGMOVE $77 + 6.5 \cdot B$
B die Blocklänge

REGSWAP $75.6 + 7.4 \cdot B$

SAVEP $350 + 90 \cdot D + I$

SAVER $56 + 12.5 \cdot A + I$

SAVERX $INT(x) = 0: 102.6 + 9.9 \cdot R$
$INT(x) \neq 0: 109.3 + 9.9 \cdot R$

SAVEX $59.6 + I$

SEEKPT(A)

Datendateien	$x = 0: 69.6 + I$
	$0.001 \leq x < 1: 65.2 + I$
Textdateien	$x = 0: 102.3 + I$
	$0.001 \leq x < 1: 96.2 + I$

$x \geq 1$: langsamer aber nicht näher vorausbestimmbar

SIZE? $56 + 2.6 \cdot x$

STOFLAG 35.1 für Alpha-Ketten in X
$\approx 58 + 20 \cdot F$ für Kontrollzahlen 'st,uv' in X
F die Anzahl der betroffenen Flags ($uv - st + 1$)

X<>F	100
XTOA	≈ 48 für numerische Werte in X 47 + 3.7·C für Alpha-Ketten in X C die Anzahl der anzuhängenden Zeichen, plus 1 oder 2 ms, falls das Alpha-Register leer ist
ASROOM	nicht vorauszubestimmen
CLRGX	67.6 + 1.19·R
EMROOM	91 + I
RESZFL	nicht vorauszubestimmen
ΣREG?	53
X vgl. **NN?**	zwischen 45 und 50, minus 4 ms, falls y = 0

Liste der in den Formeln verwendeten Parameternamen:

C	Zeichenanzahl
S	Satznummer
I	Datei-Inkrement (Erklärung unter 'APPCHR')
D	Dateigröße ('FLSIZE')
G	Anzahl der globalen Marken im Katalog 1
A	Datenregisteranzahl ('SIZE')
R	Registeranzahl
P	Programmregisteranzahl
B	Blocklänge
F	Flaganzahl
x	Inhalt von X
y	Inhalt von Y

ANHANG C

Publikationen zum HP-41 und Hardware

Nachstehend finden Sie eine Reihe von Publikationen zum HP-41 mit Inhaltsbeschreibungen sowie Bezugsquellen für Module.

1. Synthetische Programmierung auf dem HP-41C/CV, von W. C. Wickes, Heldermann Verlag Berlin, 2. erw. Aufl. 1983

Dieses Buch war die erste zusammenfassende Darstellung der synthetischen Programmierung auf dem HP-41. Es ist ein hervorragendes Buch, das inzwischen in 2. Auflage vorliegt. Aus heutiger Sicht muß gesagt werden, daß es sich am besten als Folgelektüre zum nachfolgend genannten 'Jarett I' eignet. Und dies aus zwei Gründen: a) Die Darstellung des Stoffes ist im 'Wickes' anspruchsvoll und gründlich. Er dringt an jeder Stelle tief in die Dinge ein. Eine vorausgehende Lektüre des 'Jarett I' kann also sehr hilfreich sein. b) Entsprechend dem Stand der synthetischen Programmierung zur Zeit der Entstehung des 'Wickes' werden darin umständlichere Verfahren zur Erzeugung synthetischer Befehle verwendet. Diese Verfahren ersetzt man besser durch die im 'Jarett I' vorgestellten Methoden und Dienstprogramme. Jedoch handelt es sich – wohlgemerkt – nur um Verfahrensweisen, für die die Bezeichnung 'überholt' zutreffen mag und nicht um den Stoff selbst. Bereiche, die im 'Jarett I' nur angedeutet werden, sind im 'Wickes' ausführlich erläutert. Auch sollte der Titel des 'Wickes' nicht zu der Annahme verführen, der HP-41CX sei ausgeschlossen. Er taucht nur deswegen nicht im Titel auf, weil es ihn zur Zeit der Entstehung des Wickes noch gar nicht gab. Synthetische Programmierung ist völlig unabhängig vom Modell des HP-41.

2. Synthetisches Programmieren auf dem HP-41 – leicht gemacht, von K. Jarett, Heldermann Verlag Berlin, 1985

Das Buch ist eine didaktisch glänzend geschriebene Einführung in die synthetische Programmierung. Wer es gelesen hat, dem bietet der unter Punkt 1 genannte 'Wickes' gewiß keine Schwierigkeiten mehr. Außerdem stellt das Buch bequeme Verfahren und Dienstprogramme, mit deren Hilfe der 'Wickes' leichter zu bewältigen ist, zur Verfügung. Insbesondere werden die X-Funktionen, die Zeit-Funktionen und der PPC ROM (vgl. Punkt 7) in die synthetischen Programme mit einbezogen. Die deutsche Ausgabe enthält zusätzlich Ausdrucke der im Buch angesprochenen PPC ROM Routinen. Zu den Perlen des 'Jarett I' (so genannt, um ihm vom vorliegenden 'Jarett II' zu unterscheiden) gehören ein Meßprogramm zur programmgesteuerten Messung der Ausführungsdauer von Befehlen und ein unüberbietbares Morseprogramm.

3. Tips, Tricks und Routinen für Taschenrechner der Serie HP-41, von J. S. Dearing, Heldermann Verlag Berlin, 1984

Das Buch ist die deutsche Ausgabe einer von John Dearing erstellten Programmsammlung, die u. a. viele der PPC ROM Routinen (vgl. Punkt 7) enthält. Allen in die Sammlung aufgenommenen Programmen ist eine Benutzeranleitung vorangestellt. Ein umfangreiches Stichwortverzeichnis erlaubt ein zielsicheres und schnelles Auffinden der Routinen und Tips, immerhin fast 400 an der Zahl.

4. Plotten und Drucken auf dem HP-41 Thermodrucker, von W. Meschede, Heldermann Verlag Berlin, 1985

Der Autor bietet eine sehr nützliche Sammlung von Programmen zur graphischen Darstellung von einer oder mehreren Funktionen auf dem HP-41 Thermodrucker. Die Liste der Sonderzeichen ist außerordentlich hilfreich. Ein empfehlenswertes Buch.

5. HP-41 Barcodes mit dem HP-IL-System, von K. Albers, Heldermann Verlag Berlin, 1986

Zum Zeitpunkt der Abfassung dieses Textes liegt mir das Buchmanuskript nur in Auszügen vor. Der Grund, warum ich es doch in diese Liste aufnehmen will, liegt darin, daß es zweifellos eine Bereicherung der Literatur über den HP-41 darstellen wird. Eine genauere inhaltliche Darstellung findet der Leser im anschließend abgedruckten Programm des Heldermann Verlages.

6. Softwareentwicklung am Beispiel einer Dateiverwaltung (HP-41), von M. Gehret, Verlag Vieweg, 1984

Diese Publikation ist eine bemerkenswerte, fast rätselhafte Erscheinung auf dem HP-41 Buch-Markt. Es bietet ein vollkommener wohl kaum zu denkendes, modular (vgl. die Zeilen-Analyse von "NAP" in Kapitel 7) aufgebautes Programmpaket zur Verwaltung von Dateien auf dem HP-41 unter Einsatz eines Massenspeichers (Kassette oder Diskette). Das im Kapitel 7 des vorliegenden Buches vorgestellte Programm "NAP" wird davon weit übertroffen. Unverständlicherweise führt der Autor die Schätze seines Buches aber ganz offenbar bewußt in einem so asketischen und nahezu leserfeindlichen Stil vor, daß nur diejenigen mit dem Buch etwas werden anfangen können, die Zähigkeit genug besitzen, sich all das, was der Autor – man ist versucht zu sagen in 'hochmütiger Darstellung' – unter den Tisch fallen läßt, zähneknirschend selbst zu erarbeiten. Es sei dem Autor ins Stammbuch geschrieben, daß auch eine außerordentliche Leistung – und die liegt hier zweifellos vor – nie und nimmer dazu berechtigt, den größten Teil der Leser achselzuckend ihrem Schicksal zu überlassen.

7. Der PPC ROM ist ein Kunden-Modul für den HP-41, der von den PPC-Mitgliedern (vgl. Punkt 9) entworfen wurde und von Hewlett-Packard hergestellt wird. Er enthält 122 Routinen und Programme, davon über 60 synthetisch. Sie werden alle Zeile für Zeile in dem 492 Seiten umfassenden englischsprachigen Benutzerhandbuch analysiert. PPC ROM einschließlich Handbuch können bei

Microtec GmbH
Lepsiusstr. 81
1000 Berlin 41

bestellt werden. Dieselbe Firma baut auch kundenseitig gelieferte Module sachkundig zusammen, so daß Steckplätze freigehalten werden können; z.B. Mathematik und Navigation oder zwei X-Memory-Module in einem Modul-Gehäuse integriert. Ferner kann man dort *Buchsenerweiterungen* erwerben. Das sind kleine Geräte, die einen Einzelsteckplatz des HP-41 zur Aufnahme von 4 bis 7 Moduln befähigen, so daß insgesamt 7 bis 10 Steckplätze für Module zur Verfügung stehen (allerdings sind bei so hoher Modul-Anzahl gewisse Einschränkungen für die ROM- und RAM-Kombinationen zu beachten). Außerdem liefert die genannte Firma den in Abschnitt 10K beschriebenen *Weckmodul.*

8. PRISMA ist die Clubzeitschrift des

CCD-Computerclub Deutschland e.V.
Limburger Str. 15
Postfach 2129
6242 Kronberg 2

Der Club ist ein eingetragener als gemeinnützig anerkannter Verein mit etwa 2500 Mitgliedern. Er unterstützt tragbare Computer-Systeme, vorzugsweise HP-41 und HP-71; es existiert auch eine CP/M 80-Gruppe. PRISMA erscheint jährlich 10 mal mit aktuellen Beiträgen zur Hard- und Software. Für alle in PRISMA veröffentlichten Programme werden Barcodes abgedruckt. Der Mitgliedspreis beträgt 60,– DM pro Jahr und schließt den Bezug der Zeitschrift ein.

9. Das PPC Calculator Journal wird vom '*P*ersonal *P*rogramming *C*enter' herausgegeben. PPC und PPC CJ sind im wesentlichen auf den HP-41 fixiert. Wer Mitglied werden will oder das PPC CJ bestellen möchte, wende sich an

PPC
P.O. Box 9599
Fountain Valley, CA 92728
U.S.A.

10. Kryptologie (HP-41C/CV), von Frank Altensen, Verlag Vieweg, 1983

Das Buch enthält 8 Programme, mit denen auf dem HP-41 elementare Kryptologie betrieben werden kann. Wer sich als HP-41 Benutzer mit diesem Gebiet beschäftigen will und einen Einstieg 'von ganz unten' sucht, kann mit dieser Publikation beginnen.

11. Das HP-IL-System, Einführung in die Hewlett-Packard Interface-Schleife, von Gerry Kane, Steve Harper und David Ushijima, Verlag McGraw-Hill, 1984

Dieses Buch, eine Übersetzung aus dem Englischen, enthält eine sehr ausführliche, stark

Hardware-orientierte Beschreibung der HP-IL-Schleife. Sämtliche HP-IL Steuermeldungen und Meldungsfolgen werden erläutert. Die Publikation wendet sich vornehmlich an Bastler und Gerätehersteller. Befehle des HP-41 selbst spielen keine Rolle in der abgehandelten Nomenklatur.

12. Der CCD ROM ist ein Kunden-Modul für den HP-41, der von der Firma

W&W
Postfach 80 01 33
Im Aehlemaar 20
5060 Bergisch Gladbach 2

entworfen wurde und unter deren Regie hergestellt wird. Die in ihm enthaltenen Funktionen sind im Gegensatz zum PPC ROM (vgl. Punkt 7) sämtlich in Maschinensprache geschrieben, mithin äußerst schnell. Synthetisches Programmieren jeder Art wird mit dem neuen Modul zum Kinderspiel, weil er die völlig problemlose Erzeugung beliebiger synthetischen Befehle, Zeilen und Tastenzuweisungen, sowohl im RUN- als auch im PRGM-Modus, gestattet. Hilfsmittel wie der Byte-Schnapper, Programme zum Laden von Bytes oder Tastenzuweisungsprogramme haben ausgedient, sobald der CCD ROM im HP-41 steckt. Im X-Memory läßt sich eine neue Dateiart, Typ Matrix (Kennung 4, vgl. Abschnitt 10C), erzeugen. Zur Bearbeitung der neuen Dateiart stehen alle gängigen Matrizenoperationen, die als Maschinenbefehle blitzschnell ausgeführt werden, zur Verfügung. Ferner gibt es ausgeklügelte Funktionen zur Bitmanipulation, einen Generator für Zufallszahlen, nützliche neue Ein-/Ausgabe-Befehle, spezielle Funktionen für den Umgang mit dem Adreßzeiger und dem Rücksprungstapel, die das Herz des Synthetikers höher schlagen lassen, 'PEEK'- und 'POKE'-Befehle, sowie einige nur in Verbindung mit dem X/F-Modul oder auf dem HP-41CX arbeitenden Funktionen, darunter als Höhepunkt ein unglaublich schneller Sortierbefehl für Datendateien. Der Modul macht seinen Herstellern und den am Entwurf beteiligten CCD-Mitgliedern (vgl. Punkt 8) alle Ehre. — Zwei kleine Wanzen sitzen in der Version A des ROMs: 1. Zahlen, die der Zufallsgenerator erzeugt, sind nicht-normalisiert, sobald sie kleiner als 0,1 sind. 2. Leerstellen sollten nur im USER-Modus in das Alpha-Register getastet werden, weil andernfalls überraschende Veränderungen im Arbeitsspeicher eintreten können.

13. In Österreich finden HP-Freunde Unterstützung durch den

CCA
HP-Club Austria
Postfach 50
A-1111 Wien

ANHANG D

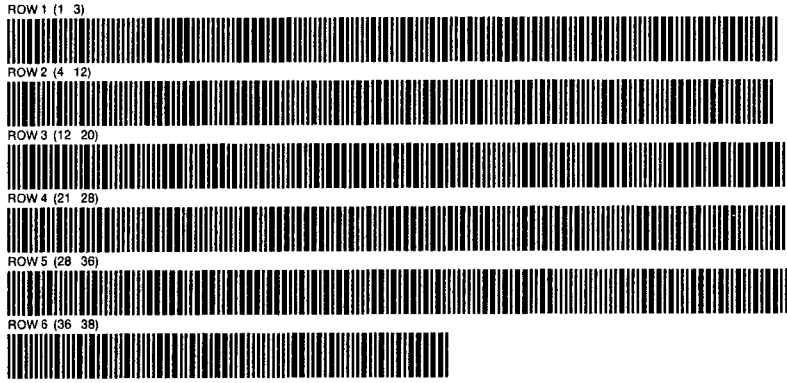
Barcodes für die in diesem Buch enthaltenen Programme

Nachstehend finden Sie die Barcodes der in den Kapiteln 1 bis 9 dieses Buches besprochenen Programme, ausgenommen nur diejenigen, deren Kürze und Einfachheit eine Herstellung solcher Kodes nicht rechtfertigt, und ausgenommen das Kryptographieprogramm des Kapitels 11, so daß Sie sie bequem mit Hilfe eines Lesestiftes HP 82153A in Ihren Rechner bringen können. Sie sollten versuchen, sich einen Lesestift auszuleihen, wenn Sie keinen besitzen, weil Sie dadurch viel Zeit sparen können.

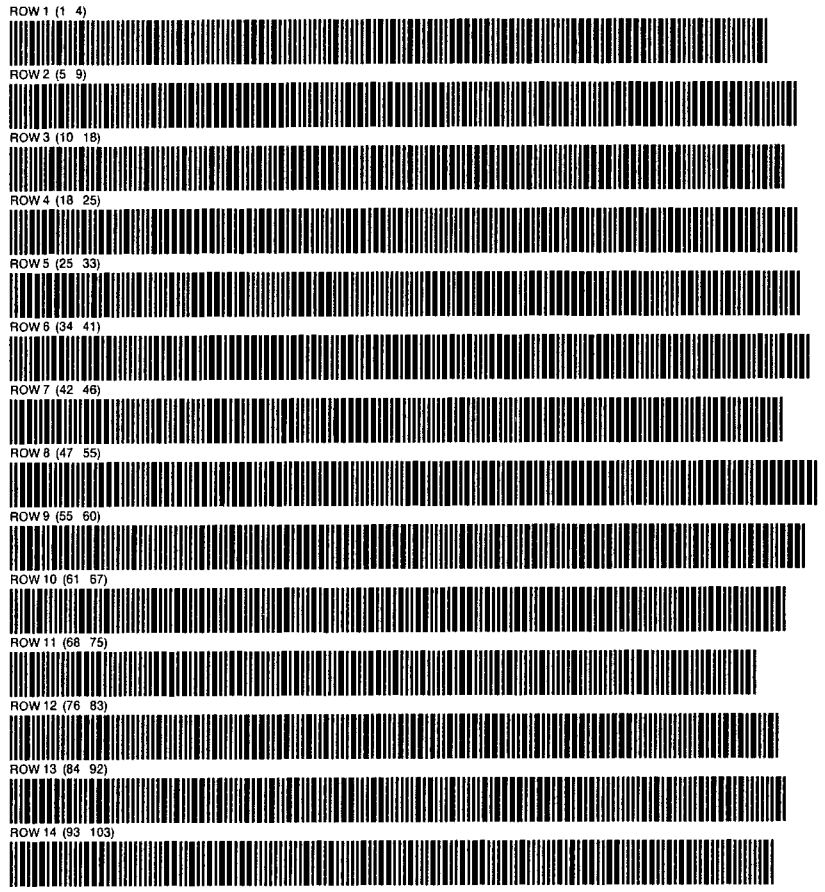
Schützen Sie die Barcodes beim Abtasten stets durch eine saubere durchsichtige Plastik-Folie. Sollten Sie ausnahmsweise einmal Leseschwierigkeiten haben, können Sie diese i.a. dadurch überwinden, daß Sie ein gleichmäßig dunkles Blatt *unter* die abzutastende Seite legen. Das verbessert den Kontrast erheblich. Auch sollte die darübergelegte Folie keine dem Lesestift zugewandte glänzende Oberfläche besitzen, vielmehr möglichst stumpf sein. Falls Sie Fehlermeldungen bekommen, prüfen Sie, ob sich in der vom Lesestift beanstandeten Zeile vielleicht unvollständige Striche befinden. Diese sollten Sie im Zweifelsfall vorsichtig nachschwärzen. Es hilft häufig auch, die Abtastgeschwindigkeit mit Hilfe eines Lineals zu erhöhen oder den Lesestift in einem anderen Winkel anzusetzen.

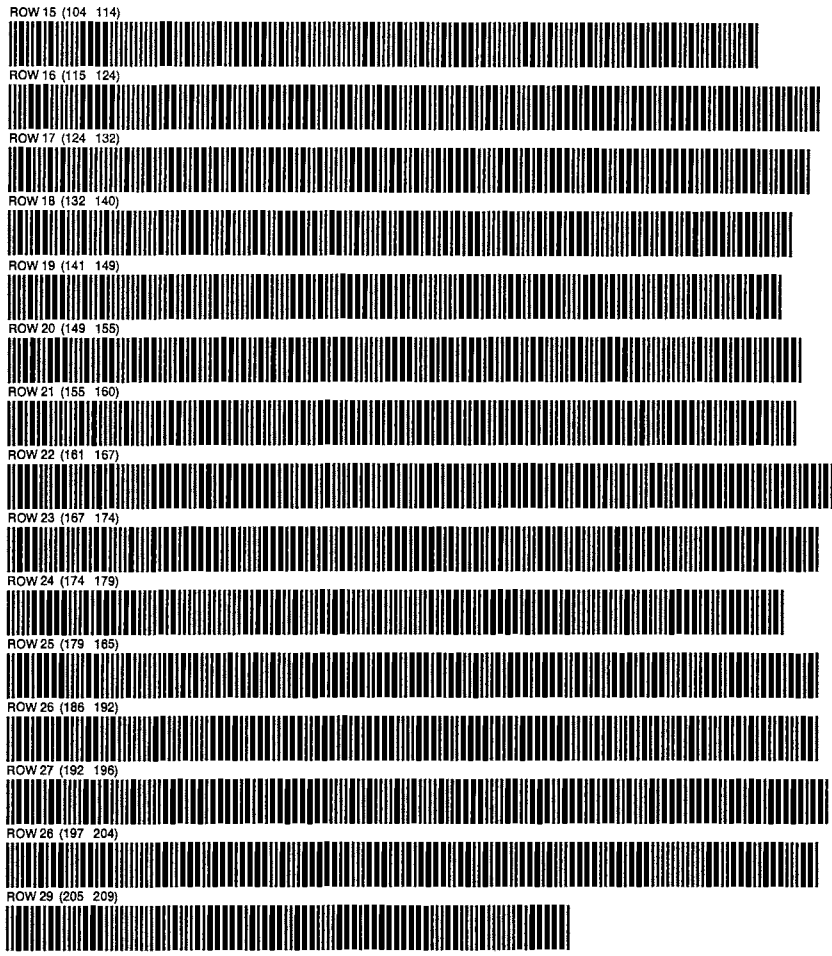
Besitzer eines Kartenlesers oder eines Massenspeichers (Kassette, Diskette) sollten die eingelesenen Programme zusätzlich auf dem ihnen zur Verfügung stehenden Medium dauerhaft aufzeichnen, um sich gegen Barcode-feindliche Umstände (Nagetiere, Wassereinbruch, kunstbeflissene Kleinkinder usw.) zu schützen. Dem X-Memory aber sollten Sie nicht vertrauen, wenn es um das langfristige Aufbewahren von Programmen geht. Bei einem "MEMORY LOST" nämlich verhält es sich mit dem Hauptspeicher solidarisch und läßt sich, treulos wie dieser, alles entreißen, was doch behütet werden sollte.

Alphabetische Sortierung: "ALSORT"
Speicherplatzbedarf: 11 Register

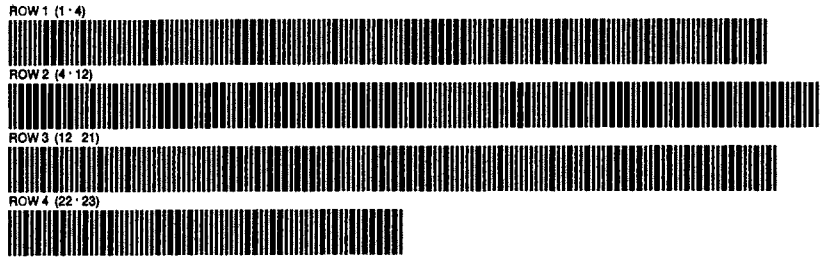


Tastenzuweisungen tätigen: "ASG"/"PASG"/"MKX"
Speicherplatzbedarf: 54 Register

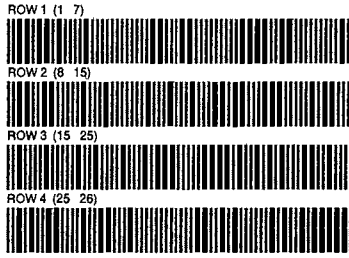




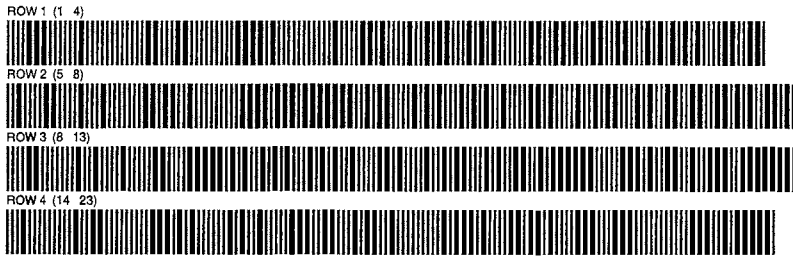
Blocklöschung mit 'ΣREG': "BCΣ"
Speicherplatzbedarf: 7 Register



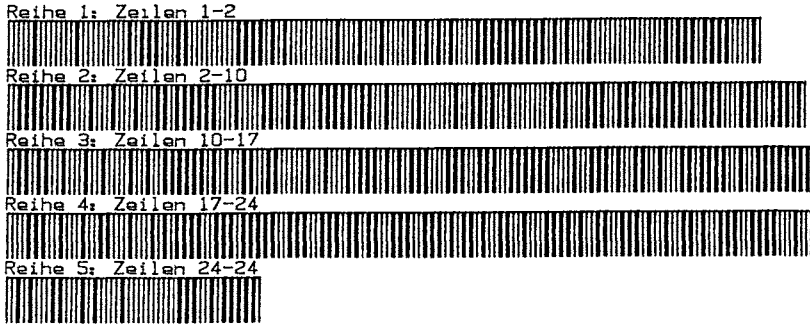
Blockrotation: "BR"
Speicherplatzbedarf: 7 Register



Bytes mit Hilfe des X-Memorys zählen: "CBX"
Speicherplatzbedarf: 8 Register



Arbeitsdatei ermitteln: "DAT?"
Speicherplatzbedarf: 8 Register



Nullstellensuche/Ableitung/Integration: "SOLVE"/"DERIV"/"INTEG"
Speicherplatzbedarf: 58 Register

ROW 3 (10 15)

ROW 4 (15 20)

ROW 5 (20 26)

ROW 6 (26 30)

ROW 7 (31 40)

ROW 8 (41 49)

ROW 9 (49 54)

ROW 10 (54 57)

ROW 11 (58 63)

ROW 12 (64 71)

ROW 13 (71 77)

ROW 14 (78 89)

ROW 15 (90 99)

ROW 16 (100 103)

ROW 17 (104 114)

ROW 18 (115 118)

ROW 19 (118 127)

ROW 20 (128 132)

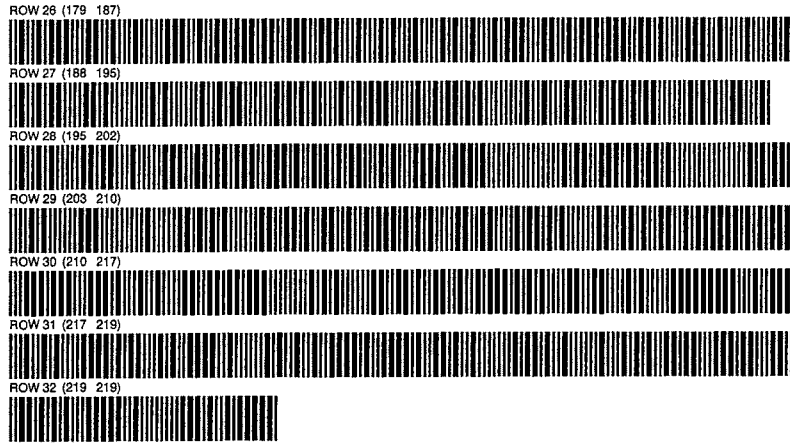
ROW 21 (133 140)

ROW 22 (141 148)

ROW 23 (148 156)

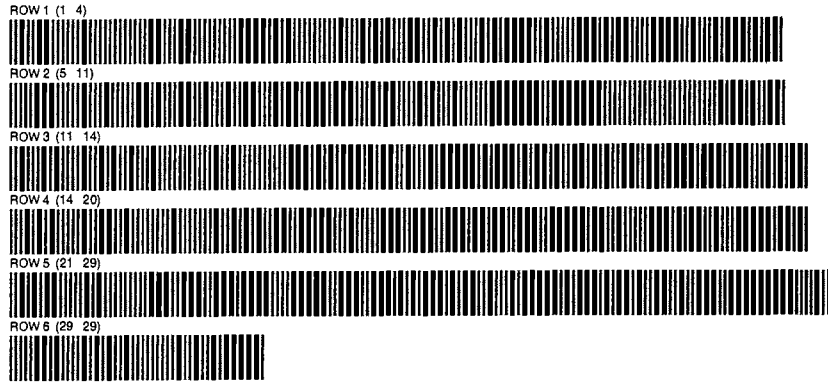
ROW 24 (157 168)

ROW 25 (168 179)



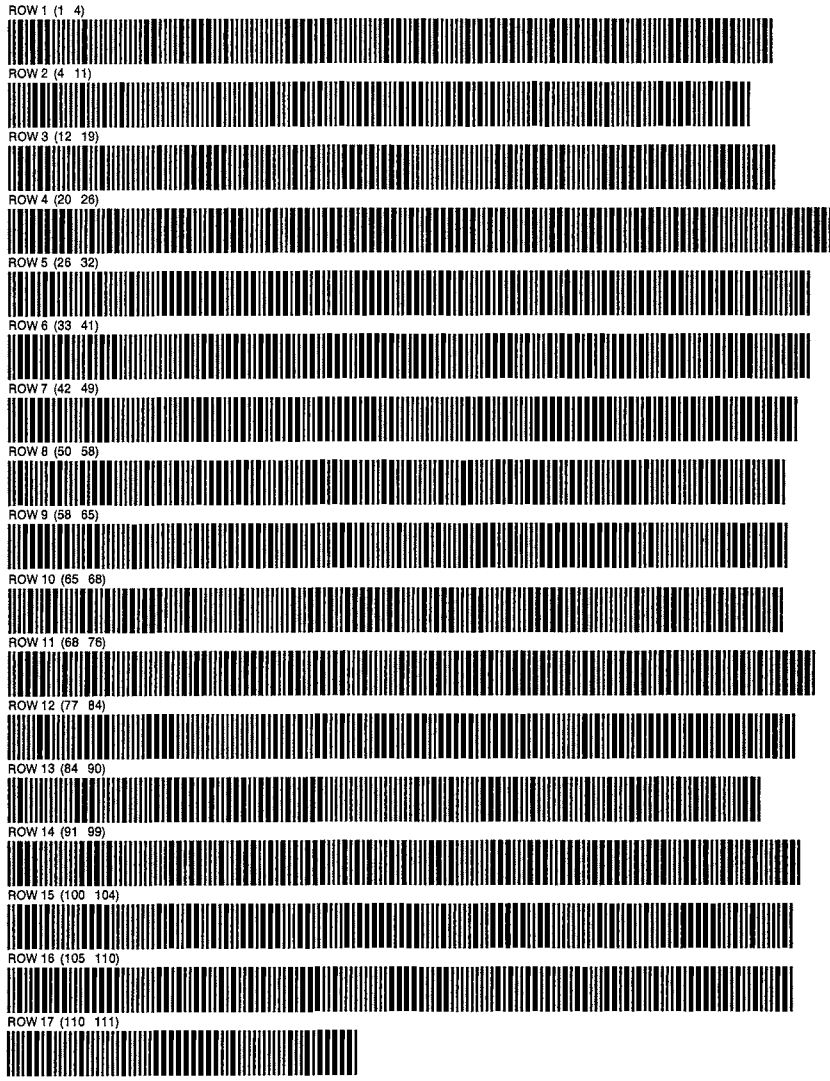
Programmausführung an den Hauptspeicher zurückgeben: "E–M"
Speicherplatzbedarf: 4 Register

X-Funktionen, Zeit-Funktionen und Funktionen des optischen Lesestiftes aufrufen: "EFTW"
Speicherplatzbedarf: 10 Register



Programme im X-Memory ausführen: "EXM"
Speicherplatzbedarf: 3 Register

Aufheben und Wiederbeleben von Tastenzuweisungen: "SAVEK"/"GETK"/"RK"/"SK"
Speicherplatzbedarf: 31 Register



HP-16 Simulator: "HP-16"
Speicherplatzbedarf: 43 Register

ROW 1 (1 4)

ROW 2 (4 13)

ROW 3 (14 21)

ROW 4 (22 29)

ROW 5 (29 36)

ROW 6 (37 44)

ROW 7 (45 54)

ROW 8 (55 62)

ROW 9 (63 71)

ROW 10 (72 79)

ROW 11 (80 88)

ROW 12 (89 98)

ROW 13 (98 107)

ROW 14 (108 115)

ROW 15 (116 122)

ROW 16 (123 131)

ROW 17 (132 141)

ROW 18 (142 151)

ROW 19 (152 160)

ROW 20 (161 169)

ROW 21 (170 178)

ROW 22 (178 186)

ROW 23 (187 191)

Hauptspeicher und X-Funktionen-Modul in Datendatei bringen: "HS + XFM"
Speicherplatzbedarf: 6 Register

Reihe 1: Zeilen 1-2

Reihe 2: Zeilen 2-5

Reihe 3: Zeilen 6-11

Initialisierung für "GETK": "IN"
Speicherplatzbedarf: 14 Register

ROW 1 (1 4)

ROW 2 (4 5)

ROW 3 (6 13)

ROW 4 (14 18)

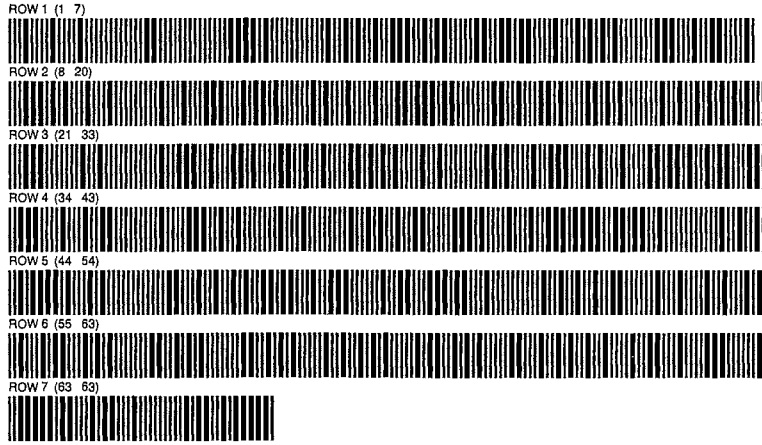
ROW 5 (19 28)

ROW 6 (28 37)

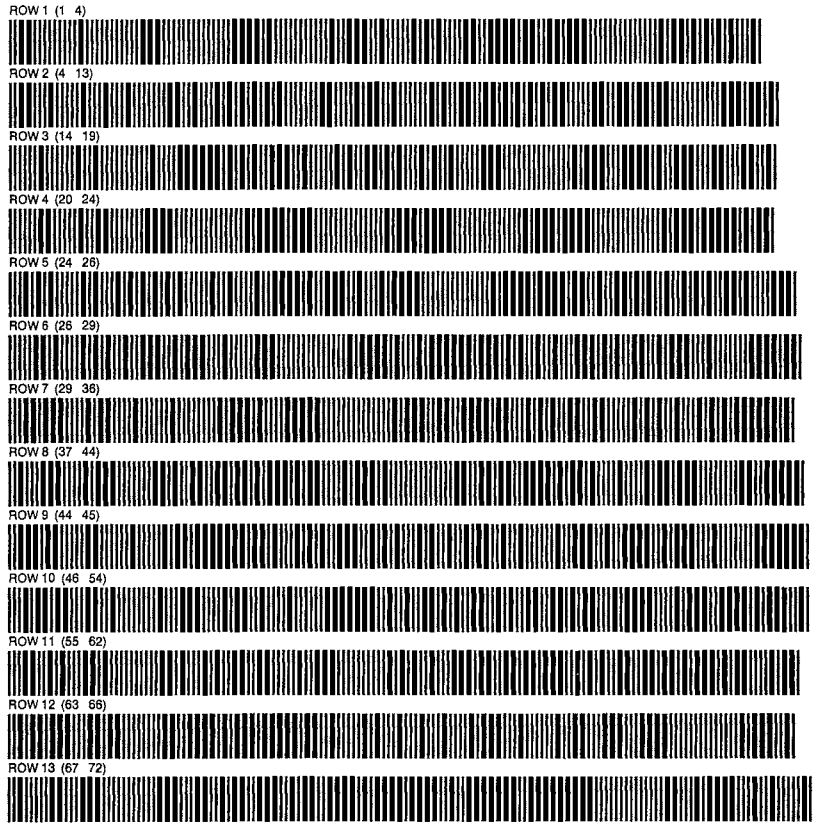
ROW 7 (38 45)

ROW 8 (45 45)

Besselfunktionen: "JNX"
Speicherplatzbedarf: 12 Register



Postadressen: "NAP"
Speicherplatzbedarf: 64 Register



ROW 14 (73 81)

ROW 15 (82 86)

ROW 16 (87 97)

ROW 17 (98 100)

ROW 18 (100 106)

ROW 19 (106 112)

ROW 20 (113 119)

ROW 21 (120 124)

ROW 22 (125 129)

ROW 23 (130 137)

ROW 24 (137 145)

ROW 25 (145 152)

ROW 26 (153 158)

ROW 27 (159 162)

ROW 28 (163 169)

ROW 29 (170 178)

ROW 30 (178 184)

ROW 31 (184 188)

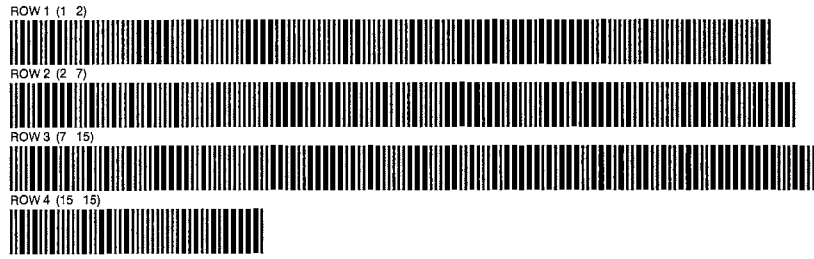
ROW 32 (189 194)

ROW 33 (194 201)

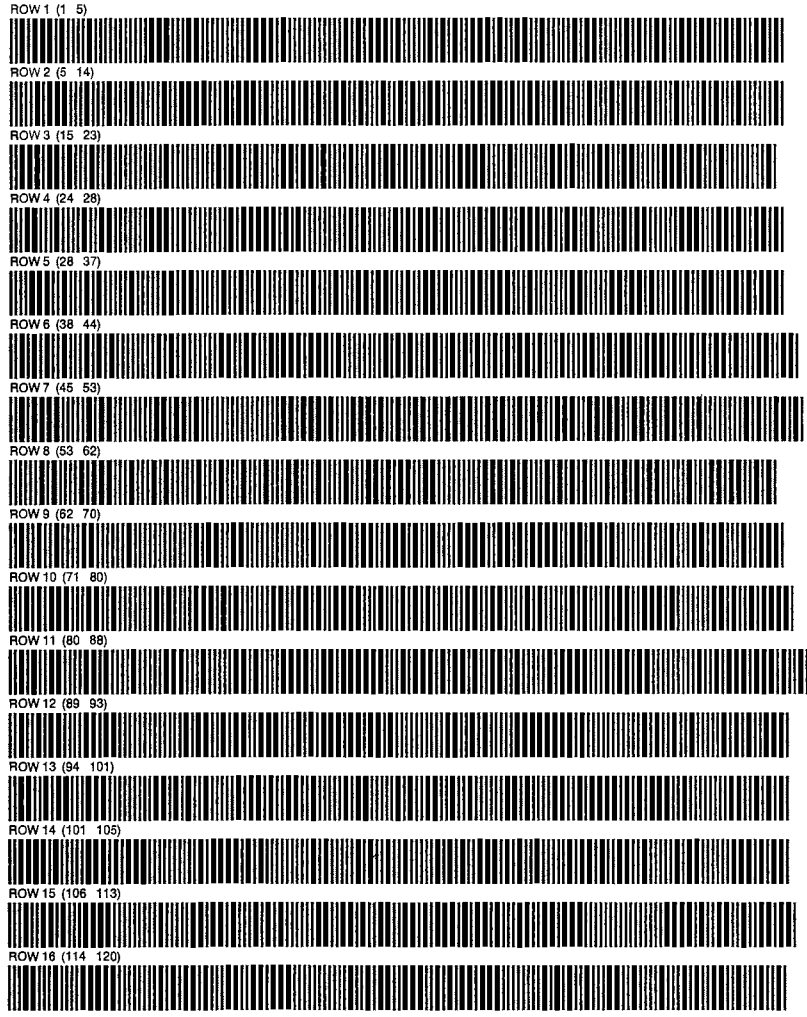
ROW 34 (201 208)

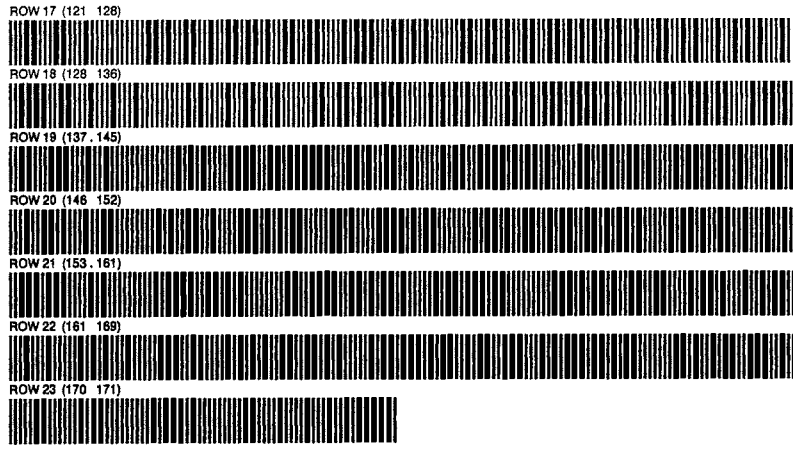
ROW 35 (209 211)

Zugriff auf das X-Memory wiederherstellen: "PFF"
Speicherplatzbedarf: 6 Register

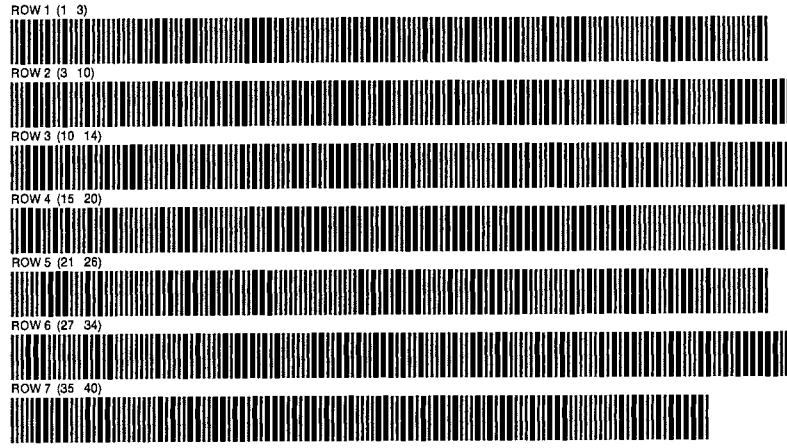


Magnetkarten mit Textdateien beschreiben: "WAS"/"PWAS"/"RAS"/"PRAS"
Speicherplatzbedarf: 42 Register

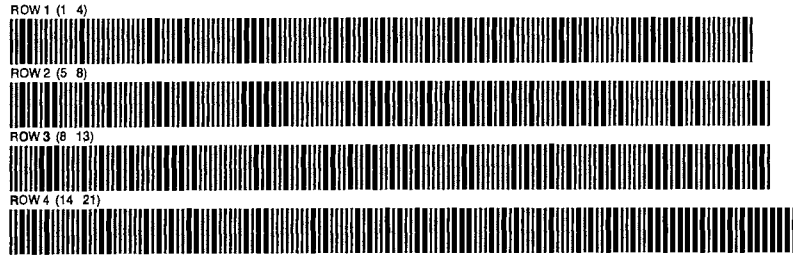




Inhalt einer Textdatei betrachten: "VAS"/"PVAS"
Speicherplatzbedarf: 13 Register



Beschädigte Programme wiedergewinnen und X-Memory-Dateien auf Magnetkarten bringen: "RPF"/"WFL"/"RFL"
Speicherplatzbedarf: 60 Register



ROW 5 (21 26)

ROW 6 (26 33)

ROW 7 (34 41)

ROW 8 (42 48)

ROW 9 (48 52)

ROW 10 (52 57)

ROW 11 (57 64)

ROW 12 (64 73)

ROW 13 (74 82)

ROW 14 (83 91)

ROW 15 (91 98)

ROW 16 (98 104)

ROW 17 (105 113)

ROW 18 (113 121)

ROW 19 (122 128)

ROW 20 (128 134)

ROW 21 (135 143)

ROW 22 (143 151)

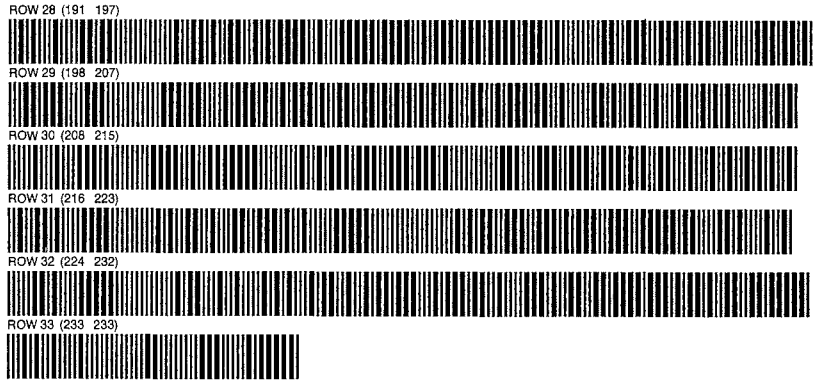
ROW 23 (152 159)

ROW 24 (160 168)

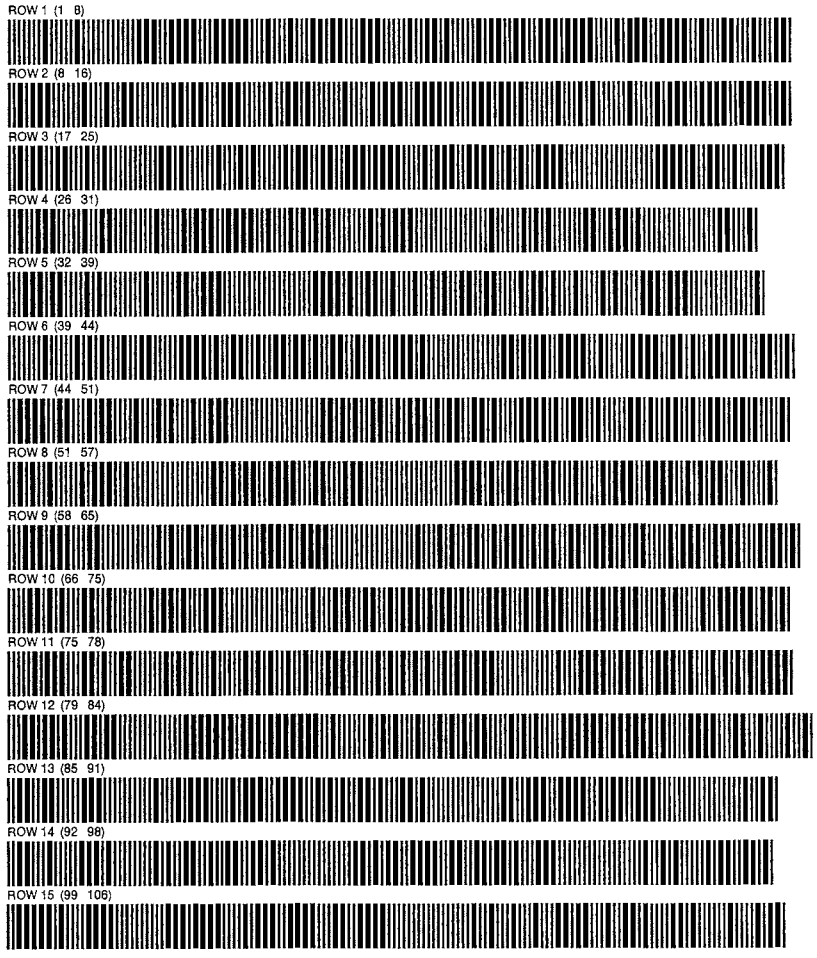
ROW 25 (168 176)

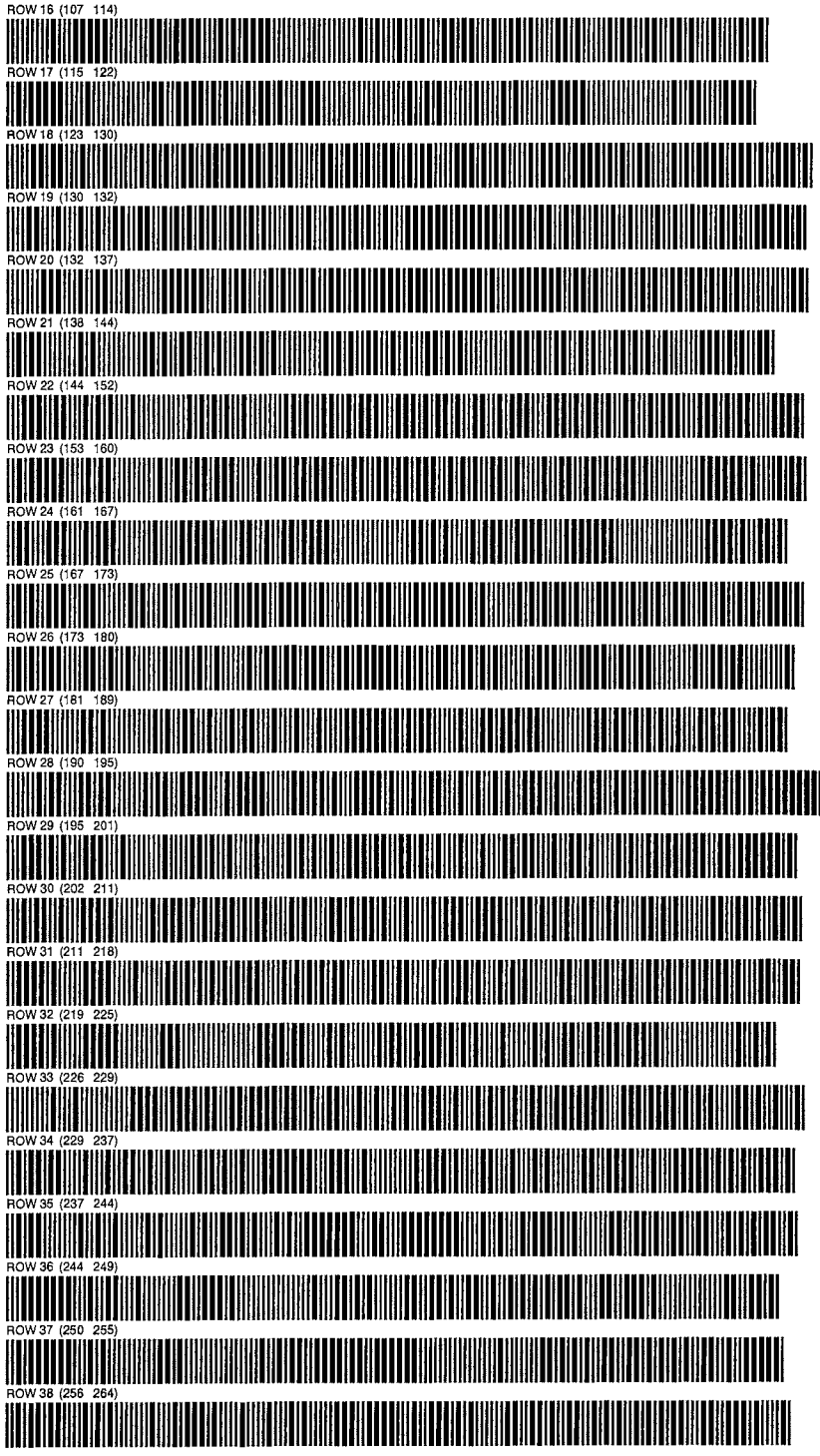
ROW 26 (177 183)

ROW 27 (184 190)



Text-Editor: "TE"
Speicherplatzbedarf: 115 Register




ROW 16 (107 114)
ROW 17 (115 122)
ROW 18 (123 130)
ROW 19 (130 132)
ROW 20 (132 137)
ROW 21 (138 144)
ROW 22 (144 152)
ROW 23 (153 160)
ROW 24 (161 167)
ROW 25 (167 173)
ROW 26 (173 180)
ROW 27 (181 189)
ROW 28 (190 195)
ROW 29 (195 201)
ROW 30 (202 211)
ROW 31 (211 218)
ROW 32 (219 225)
ROW 33 (226 229)
ROW 34 (229 237)
ROW 35 (237 244)
ROW 36 (244 249)
ROW 37 (250 255)
ROW 38 (256 264)

ROW 39 (264 271)

ROW 40 (271 273)

ROW 41 (273 279)

ROW 42 (280 286)

ROW 43 (286 291)

ROW 44 (291 296)

ROW 45 (297 304)

ROW 46 (304 309)

ROW 47 (310 312)

ROW 48 (312 320)

ROW 49 (320 325)

ROW 50 (326 331)

ROW 51 (332 337)

ROW 52 (337 339)

ROW 53 (339 339)

ROW 54 (340 343)

ROW 55 (344 352)

ROW 56 (352 360)

ROW 57 (360 365)

ROW 58 (366 373)

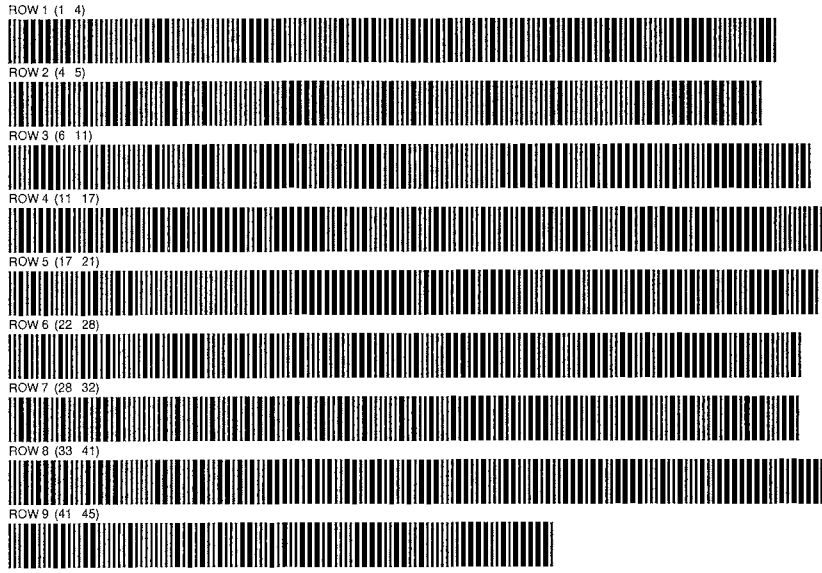
ROW 59 (373 379)

ROW 60 (380 388)

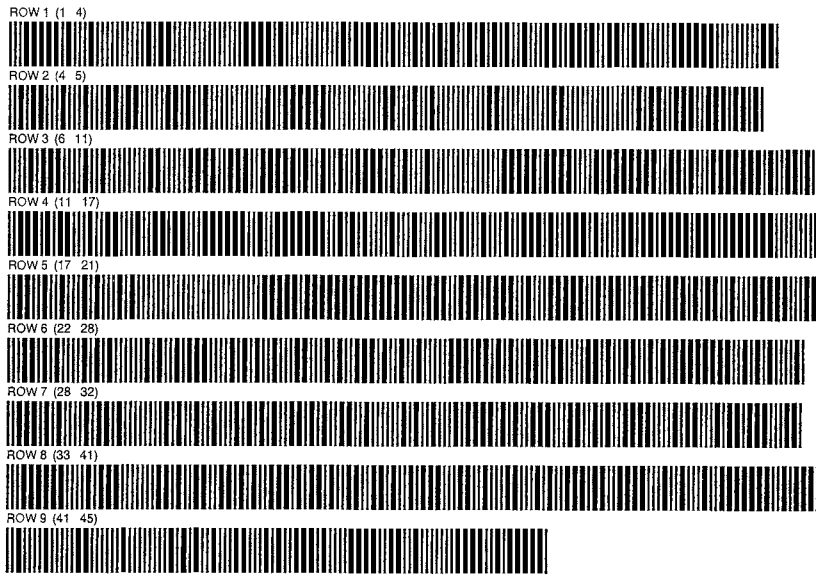
ROW 61 (388 393)

ROW 62 (393 397)

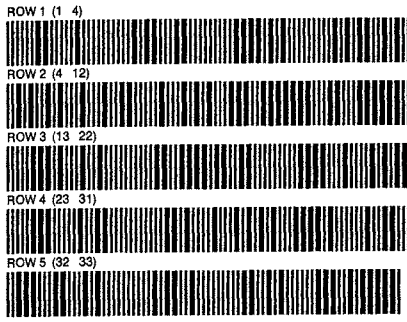
Magnetkarten überprüfen (Spielart 1): "VER"
Speicherplatzbedarf: 16 Register



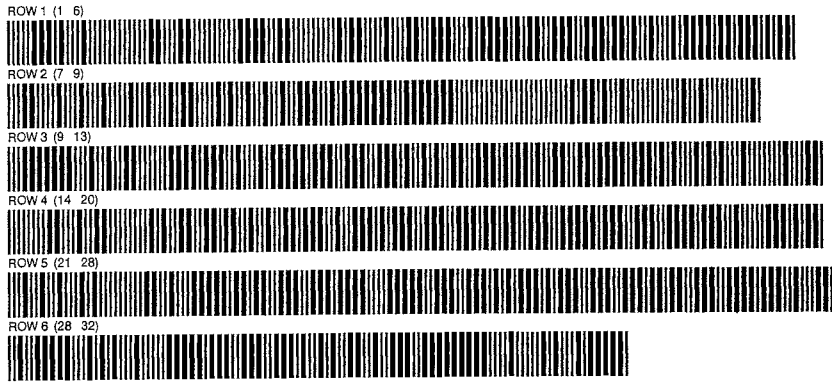
Magnetkarten überprüfen (Spielart 2): "VER"
Speicherplatzbedarf: 16 Register



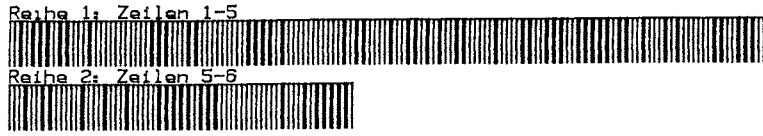
Registerinhalte betrachten: "VREG"
Speicherplatzbedarf: 9 Register



X-Funktionen aufrufen: "XF"
Speicherplatzbedarf: 11 Register



X-Memory löschen: "XML"
Speicherplatzbedarf: 3 Register



STICHWORTVERZEICHNIS

VERZEICHNISSE DER X-FUNKTIONEN, DER PROGRAMMAUSDRUCKE UND DER BARCODES

X-Funktionen

Programmausdrucke

Barcodes

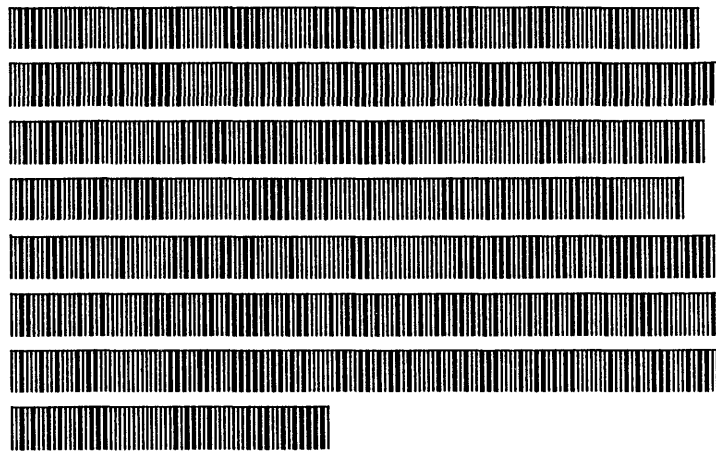
Dessert

'XEQ "WXM"' schreibt den Inhalt des *gesamten* X-Memorys in eine 606 Register umfassende Massenspeicher-Datendatei namens "XMEM". 'XEQ "RXM"' liest den Inhalt von "XMEM" ins X-Memory. – Die Routine darf nicht unterbrochen werden.

```
01♦LBL "WXM"     10 RCL c         19 X<> Z         28 ASTO c
02 SF 04         11 "α♦♦"         20 STO c         29 SF 25
03 GTO 00        12 ,127          21 CLX           30 FC? 04
04♦LBL "RXM"     13 XEQ 02        22 CF 25         31 READRX
05 CF 04         14 " 0♦"         23 "XMEM"        32 FS? 04
06♦LBL 00        15 ,238          24 FS?C 04       33 WRTRX
07 "XMEM"        16 XEQ 02        25 VERIFY        34♦LBL 01
08 0             17 "00♦"         26 GTO 01        35 END
09 SEEKR         18 XEQ 02        27♦LBL 02
```

```
Zeile 11: F3 04 00 00
Zeile 14: F3 20 10 00
Zeile 17: F3 30 10 00
```

X-Memory-Inhalt auf/vom Massenspeicher bringen/einlesen: "WXM"/"RXM"
Speicherplatzbedarf: 15 Register



COMPUTERBÜCHER IM HELDERMANN VERLAG

Albers, K.: HP-41 Barcodes mit dem HP-IL-System.
Der Barcodelesestift ist ein preiswertes, dabei jedoch sehr effizientes Zubehörteil für das Einlesen von Daten oder zum Programmieren. Barcodes (BCs) sind die kostengünstigste Methode der externen Datenspeicherung und deren Massenverbreitung. Über HP-41 BCs, ihre oft verblüffend ergiebige Anwendung, Herstellung und ihren Aufbau ist wenig dokumentiert. Mit diesem Buch schließt der Autor die Lücke.
Das Buch behandelt auf rund 300 Seiten ausführlich Vor- und Nachteile von BCs und Lesestift, gibt eine Übersicht über alle BC-Typen des 41-er Systems und erläutert den grundsätzlichen Aufbau und das Kodierungsschema. Die Herstellung von BCs auf dem IL-Thermodrucker, dem ThinkJet-Drucker und dem Plotter 7470A mit vielen nützlichen Hinweisen für einwandfreie Ergebnisse, sowie die dafür notwendigen oder wünschenswerten Geräte werden eingehend beschrieben. 2-Byte-BCs der ASCII-Zeichen 0 – 127, 3-Byte-Ausführung 0 – 255, Alpha- und synthetische Textzeilen-BCs zum Programmieren, überraschend einfache und schnelle Verfahren für 'load bytes' ohne Hilfsprogramm und BCs von einfachen oder längeren 'BLDSPEC'-Sonderzeichen geben rationelle Arbeitshilfen. Die Drucker-Modi 8-Bit/ESCAPE werden genau erklärt. BCs von Zahlen, trickreiches Programmieren großer Zahlen, Kurzformexponenten, numerische und Alphafolgedaten für sequentielle Registereingabe werden ausführlich beschrieben. BCs von 1- und 2-Byte-Tastenfeldfunktionen und von nicht programmierbaren Rechnerfunktionen, INDirekt, besondere Funktionen und deren BCs (↑__, μ__, eGØBEEP__, ___, $T + N IA__, sowie der Q-Bote) werden erschöpfend behandelt. Aus einer speziellen Art BCs bewirkt 'SNAP 2', der BC-Byteschnapper die verblüffend problemlose Lesestifteingabe jedes synthetischen Befehls an beliebiger Programmstelle ohne Tastenbelegungen. XROM-Funktionen-BCs, Mehrbyte-BCs von Tastenfeldfunktionen mit und ohne Argument, alle nicht programmierbaren Rechner- und XROM-Funktionen mit Argument (SIZE, DEL, ASN, PRP usw.) sowie BCs von LBL/GTO/XEQ-"Alpha" und für ein simuliertes synthetisches Tastenfeld für Sofortausführung jeder Art synthetischer Befehle ohne Tastenzuweisung erweitern die Arbeitsmöglichkeiten ganz erheblich. Der Aufbau von Programm-BCs und deren Herstellung auf dem Thermodrucker ohne das Plottermodul, auf dem ThinkJet-Drucker und dem Plotter wird in allen Einzelheiten erklärt. Eine große Zahl von Tabellen der Funktions- und Alphazeichen 0 – 127, 0 – 255 (replace und append), für das synthetische Programmieren, für ein simuliertes, synthetisches Tastenfeld sowie der XROM-Befehl aller zur Zeit verfügbaren Systemmodule sind willkommene Arbeitserleichterungen auch für den Anwender, der keinen Drucker besitzt. Es werden eine Menge praktischer Anwendungen gezeigt für das synthetische Programmieren, zur Datenmasseneinlesung für Schriftfahnen oder Querdruck, zur Herstellung beliebiger Textzeilen und zum Umgang mit speziellen LBL, GTO bzw. XEQ-Befehlen.
Jedes Kapitel enthält komfortable Programme zur BC-Herstellung oder für Umrechnungen in Form von Listing und BCs, insgesamt über 90 Programme! Nach Möglichkeit sind alle Programme dreifach vorhanden: 1. zur Herstellung nur mit Rechner und Thermodrucker ohne Systemmodule; 2. zusätzlich mit dem XF-Modul und 3. außerdem mit dem Plottermodul. Durch 1. sind die Möglichkeiten der BC-Herstellung weitestgehend auch dem Benutzer erschlossen, der nicht über Zusatzmodule vefügt. In einem Anhang werden nicht HP-BCs wie

der Europäische Artikelnummern-BC und einige anderen Typen vorgestellt. Weiterhin wird ein Ausblick auf die zukünftige Entwicklung gegeben.
Wer die Möglichkeiten seines Lesestiftes mit dem HP-41 System optimal nutzen und selbst BCs anfertigen will, braucht dieses leicht verständlich geschriebene Buch. Es wird ihm bald zur unentbehrlichen Arbeitshilfe werden.
(1986, ISBN 3 – 88538 – 804 – 9)

Dearing, J.S.: Tricks, Tips und Routinen für Taschenrechner der Serie HP-41
Dieses Buch enthält über 350 Routinen und Tips für den HP-41. Von elementaren Tricks und pfiffigen Abkürzungen bis hin zu komplizierten synthetischen Programmen aus dem PPC-Modul und umfangreichen Druck-Routinen ist alles vertreten, was die große Gemeinde der HP-41-Benutzer im Laufe von Jahren herausgefunden und zusammengetragen hat. Dem Autor ist die herkulische Leistung zu verdanken, die Fülle dieser Ergebnisse gesammelt, gesichtet und geordnet zu haben. Der Übersetzer, Heinz Dalkowski, hat in Zusammenarbeit mit dem Autor das Original erweitert und in einem Nachwort die Funktionen des neuesten Rechners aus der Serie HP-41, des HP-41CX, beschrieben. Ein umfangreiches Stichwortverzeichnis von über 1000 Einträgen erschließt die Sammlung lückenlos und läßt zielsicher auffinden, worüber man sich zu informieren wünscht. Man spart so manche Programmierstunde und kommt jedem merkwürdigen Verhalten des HP-41 auf die Spur.
(1984, ISBN 3 – 88538 – 801 – 4)

Horn, J.: HP-71 Basic – leicht gemacht
Der HP-71 bietet bei erstaunlich geringen Abmessungen Anwendungmöglichkeiten in einer Vielfalt, wie sie bisher bei Rechnern dieser Größe unbekannt war. Es ist klar, daß zu einem so komfortablen Rechner auch entsprechend umfangreiche Handbücher gehören. Deren Lektüre stellt aber, nicht nur für Computer-Neulinge, eine teils harte, teils zu gehaltvolle Kost dar, deren Genuß durchaus zu Schwierigkeiten führen kann. In diesem Vergleich spielt nun das Buch "HP-71 Basic – leicht gemacht" die Rolle des Kräuter-Likörs, der Völlegefühl und Verstimmungen beseitigt und damit weiteren Appetit auf das ganze Menue macht.
Der betont lockere Stil des Originaltextes wurde auch in der deutschen Bearbeitung voll beibehalten, weil der Umgang mit einem Rechner vorwiegend Freude und möglichst wenig Streß erzeugen soll.
(1986, ISBN 3-88538-807-3)

Jarett, K.: Erweiterte Funktionen des HP-41 – leicht gemacht
Das Buch beschreibt die Eigenschaften des erweiterten Speichers und der X-Funktionen, mit denen der HP-41C/CV aufgerüstet werden kann und die im HP-41CX fest eingebaut sind. Da das Bedienungshandbuch zum Umgang mit X-Modulen nur knappe Hinweise gibt, war ein Buch nötig, das die Fähigkeiten der X-Funktionen und des HP-41CX vollständig beschreibt. Der Autor, ein führender Experte des HP-41 Systems und Pionier der synthetischen Programmierung, hat dieses Buch in seinem unnachahmlichen Stil – einfach, klar und doch präzise – geschrieben. Die vorliegende deutsche Ausgabe von Heinz Dalkowski wurde darüber hinaus in vielerlei Hinsicht inhaltlich ergänzt. Nach der Lektüre kann der Leser die X-Funktionen wirkungsvoll einsetzen und, wenn er Kenntnisse in synthetischer Programmierung besitzt, die Kraft dieser Kunst mit der der X-Funktionen verbinden. Insbesondere bekommt er über 30 ausgereifte Programme, die von den führenden Experten auf dem Gebiet

stammen, an die Hand, darunter einen umfangreichen Text-Editor für den HP-41C/CV, ein Adressenverzeichnis-Programm, eine Simulation des HP-16, mathematische Programme, Programme zum Übertragen von Textdateien auf Magnetkarten, sowie – in Verbindung mit synthetischer Programmierung – Programme zum Reparieren fehlerhaften Verhaltens einiger speziellen Vorgänge beim Einsatz von X-Funktionen (Betriebssystemfehler). Sämtliche Programme sind als BCs abgedruckt, insgesamt 4181 Bytes! Wer aus seinem Kraftpaket herausholen will, was drin steckt, benötigt dieses Buch.
(1986, ISBN 3 – 88538 – 803 – 0)

Jarett, K.: Synthetisches Programmieren auf dem HP-41 – leicht gemacht
Der Autor wendet sich an HP-41 Benutzer, denen die gründliche aber anspruchsvolle Darstellung der synthetischen Programmierung durch W. C. Wickes ("Synthetische Programmierung auf dem HP-41C/CV", Heldermann Verlag) Schwierigkeiten bereitet. Es gelingt ihm, in ausführlicher Weise einen Zugang zu diesem Gebiet zu bereiten, der zugleich abwechslungsreich und spannend ist. Andererseits berücksichtigt er neueste Erweiterungen des HP-41 durch den Hersteller und bringt Programme, welche die Funktionen aus dem X-Modul und dem Time-Modul beinhalten, wodurch viele synthetische Programme – z. Bsp. das Tastenzuweisungsprogramm – wesentlich kürzer und schneller werden. Für Kundige ist dieses Buch somit eine spielend lesbare Ergänzung des Buches von Wickes, für Anfänger ist es die ideale Einführung.
Die Übersetzung besorgte in gewohnt fachmännischer Weise Heinz Dalkowski.
Diesem Buch liegt eine HP-41 Quick Reference Card bei.
(1985, ISBN 3 – 88538 – 802 – 2)

Meschede, W.: Plotten und Drucken auf dem HP-41 Thermodrucker
Dieses Buch enthält 18 Programme zum Plotten auf den HP-41 Thermodruckern und zusätzlich 224 Zeichen in drei Darstellungsformen und alle benötigten Zahlencodes für BLDSPEC und die synthetische Programmierung der Zeichen. Jedem Programm ist ein Programm-Ablaufplan und eine ausführliche Bedienungsanleitung beigegeben, damit sowohl reine Programm-Anwender, als auch Selbst-Programmierer voll auf ihre Kosten kommen. Durch diese Programme stellen selbst logarithmische Skalierung, die Darstellung mehrerer Funktionen in einem Schaubild, hochauflösendes Plotten, sowie Histogramme (Balkendiagramme) keine Probleme mehr dar und durch kleine Änderungen ist die Anpassung an ganz spezielle Anforderungen leicht möglich. Hat man ein Programm längere Zeit nicht mehr benutzt, ermöglichen die Kurzanleitungen im Anhang ein schnelles Rekapitulieren der sehr einfachen und komfortablen Bedienung. Zum Schluß werden dann in Kapitel 7 alle Wünsche nach ganz speziellen Zeichen für die selbst programmierte Ausgabe – einschließlich Querschrift – erfüllt.
Alle Programme sind als BCs abgedruckt, insgesamt 6755 Bytes! Wer graphische Ausgaben oder mehr als nur die einfachen 127 Zeichen benötigt, kann an diesem Buch nicht vorübergehen.
(1985, ISBN 3 – 88538 – 805 – 7)

Stroinski, W.: Zusammenfassung der Bedienungshandbücher und Programmieranleitungen für das I/O-ROM, IB- und IL-Interface der Rechner HP-83/85 und HP-86/87
Handbücher in deutscher Sprache sind meist nur für den Computer und die wichtigsten

Peripherie-Geräte (Drucker, Monitor, Massenspeicher) erhältlich, für die "seltenere" Peripherie, durch deren Anschluß der Computer erst seine volle Wirksamkeit erlangt, sind die Beschreibungen häufig nur in englischer Sprache lieferbar. Unabhängig von der Qualität der vorhandenen Sprachkenntnisse ist das Verstehen dieser neuen Materie sicherlich einfacher, wenn die Beschreibungen in deutscher Sprache vorliegen.
Für die Hewlett-Packard-Rechner der 80er Serie (HP-83/85 bzw. HP-86/87) wird mit diesem Buch über das I/O-ROM und die Interfaces für HP-IB (IEEE 488 bzw. IEC 625) und HP-IL (Interface-Loop) dieser Mangel behoben. Es enthält die Übersetzungen der nachfolgend genannten HP-Druckschriften in korrigierter Form:

I/O-ROM, Owner's Manual, 00087 – 90121, Jan. 83
HP-IB Interface, Owner's Manual, 82937 – 90017, Jan. 82
HP-IL Interface, Owner's Manual, 82938 – 90001, Jan. 82.

Der Vorläufer dieser Handbücher (I/O Programming Guide, 00085 – 90142) wurde ebenfalls berücksichtigt, wenn die dort gegebenen Erläuterungen umfangreicher waren, als in den neueren Beschreibungen. Auch für die GPIO-, BCD- und Serial Interfaces sind im Syntax-Anhang vollständige Angaben über die Auswirkungen der einzelnen Anweisungen zu finden.
(1986, ISBN 3 – 88538 – 806 – 5)

Wickes, W. C.: Synthetische Programmierung auf dem HP-41C/CV, 2. erweiterte Auflage
Die englische Originalausgabe dieses Buches ist in den U.S.A. ein Bestseller unter der Literatur über Kleinrechner geworden und auch in Deutschland wurden über 8000 Exemplare verkauft. Die deutsche Ausgabe von Heinz Dalkowski enthält gegenüber dem Original zahlreiche Verbesserungen, Verfeinerungen und Ergänzungen und wird zu Recht die "Bibel der synthetischen Programmierung" genannt.
Der Autor führt den Leser in leicht verständlicher Weise "durch" den HP-41C/CV, entdeckt ihm alle seine verborgenen Fähigkeiten und geht so inhaltlich weit über das hinaus, was das "Handbuch" bietet. Der Leser lernt die Synthetische Programmierung kennen, durch die der Rechner zu unglaublichen Taten veranlaßt werden kann: Erzeugung neuer Zeichen in der Anzeige; Verwendung des Alpha-Registers als arithmetisches Daten-Register; vollständige Benutzerkontrolle über alle Flags (einschließlich der normalerweise unzugänglichen Systemflags); Zugriff auf sämtliche Informationen über den Zustand des Rechners; schnelle Alphabetisierung von Alpha-Daten; Erzeugung neuer Töne; Verwandlung von Programmzeilen in Daten und umgekehrt; programmierter Zugriff auf beliebige Zeilen in ROM's; Herstellung programmierender Programme.
Synthetische Programmierung ist nicht nur für Hobby-Anwender, sondern in gleicher Weise für professionelle Benutzer von Interesse, wenn es darum geht, Speicherplatz zu sparen oder die Bearbeitungsgeschwindigkeit zu erhöhen.
Die zweite Auflage des Buches ist um inzwischen bekannt gewordene Fortschritte der synthetischen Programmierung erweitert worden: Der Byte-Schnapper, Programme zur automatischen Erzeugung synthetischer Programmzeilen, die Funktion eGØBEEP, Programme zur Herstellung vollständiger hexadezimaler Speicherauszüge, synthetischer Vorstoß ins X-Memory.
Obwohl dieses Buch manche Schwierigkeit enthält, was für den einen oder anderen Leser die vorausgehende Lektüre des Buches von K. Jarett, "Synthetisches Programmieren auf dem HP-41 – leicht gemacht" (ebenfalls im Heldermann Verlag) empfehlenswert macht, ist dieses Buch unerläßlich für jeden HP-41C/CV/CX Benutzer, der die Fähigkeiten und Möglichkeiten des Rechners voll ausschöpfen will.
(1983, ISBN 3 – 88538 – 800 – 6)

DIE HP-PALETTE DES HELDERMANN VERLAGES

Albers, K.: HP-41 Barcodes mit dem HP-IL-System. 320 Seiten, 44.00 DM, ISBN 3-88538-804-9 (1986)

Dalkowski, H., Fegert, S.: Eine Programmsammlung für den HP-41. ISBN 3-88538-809-X (1987)

Dearing, J. S.: Tricks, Tips und Routinen für Taschenrechner der Serie HP-41. Deutsche Ausgabe von H. Dalkowski. 220 Seiten, 36.00 DM, ISBN 3-88538-801-4 (1984).

Emery, K.: HP-41 Mikrocode-Programmierung für Anfänger. ISBN 3-88538-808-1 (1987)

Horn, J.: HP-71 Basic — leicht gemacht. Deutsche Ausgabe von W. Stroinski. Ca. 200 Seiten, ca. 40.00 DM, ISBN 3-88538-806-5 (1986)

Jarett, K.: Erweiterte Funktionen des HP-41 — leicht gemacht. Deutsche Ausgabe von H. Dalkowski. 240 Seiten, 44.00 DM, ISBN 3-88538-803-0 (1986).

Jarett, K.: Synthetisches Programmieren auf dem HP-41 — leicht gemacht. Deutsche Ausgabe von H. Dalkowski. 170 Seiten, 40.00 DM, ISBN 3-88538-802-2 (1985). Dem Buch liegt eine Quick Reference Card bei.

Meschede, W.: Plotten und Drucken auf dem HP-41 Thermodrucker. 176 Seiten, 36.00 DM, ISBN 3-88538-805-7 (1985).

Stroinski, W. (Herausgeber): Zusammenfassung der Bedienungshandbücher und Programmieranleitungen für das I/O-ROM, IB - und IL - Interface der Rechner HP-83/85 und HP-86/87. Ca. 200 Seiten, ca. 58.00 DM, ISBN 3-88538-806-5 (1986).

Wickes, W. C.: Synthetische Programmierung auf dem HP-41C/CV. Deutsche Ausgabe von H. Dalkowski. 165 Seiten, 36.00 DM, ISBN 3-88538-800-6 (1983).

HP-41 Kombinierte Hex/Dezimale Byte Tabelle, 7 × 11.5 cm Plastikkarte, 6.00 DM.

HP-41 Quick Reference Card, 7 × 15 cm Plastikkarte, 8.00 DM. (Diese Karte liegt dem Buch "Synthetisches Programmieren auf dem HP-41 — leicht gemacht" von K. Jarett kostenlos bei.)

Alle Produkte sind zu den oben angegebenen Preisen ohne Versandkosten direkt vom Verlag erhältlich, die Plastikkarten nur auf diese Weise. Bitte richten Sie Ihre Bestellung an

Heldermann Verlag Berlin
Nassauische Str. 26
D-1000 Berlin 31

Jetzt gemeinsam HP-41 und CCD-Modul

Das CCD-Modul – ein Kraftpaket für den HP-41. In dieser intelligenten Erweiterung stecken 8 kByte (über 100 neue Funktionen!) pure Energie, mit der das Arbeiten auf dem Rechner noch mehr Spaß macht. Bisher schwer durchführbare Operationen lassen sich durch die vielen interessanten Funktionen jetzt spielend programmieren. Besondere Stärken sind z. B. die Betriebssystemerweiterungen, die nun eine direkte Eingabe von „synthetischen" Funktionen und Kleinbuchstaben ermöglichen. Tastenzuordnungen, Zeitalarme und Matrizen (max. 24 x 25) können auch im Extended-Memory abgespeichert und sekundenschnell ausgetauscht werden. Mit den Input-/Outputfunktionen ist endlich auch die Druckerformatierung kein Problem mehr, ja es lassen sich sogar Barcodes drucken (auch mit dem ThinkJet!). Das Modul und Informationsmaterial ist direkt bei uns oder im guten HP-Fachhandel erhältlich.

- „synthetische" Programmierung, neue Kataloge und Kleinbuchstaben direkt über die Tastatur
- Matrixfunktionen, Binärfunktionen und Funktionen für formatierte Ein- und Ausgaben
- PEEK- und POKE-Funktionen für fortgeschrittene Programmierer

W&W SOFTWARE PRODUCTS GMBH

Im Aehlemaar 20
Postfach 800133
5060 Bergisch Gladbach 2
☎ (02202) 85068

Das Buch beschreibt die Eigenschaften des erweiterten Speichers und der X-Funktionen, mit denen der HP-41C/CV aufgerüstet werden kann und die im HP-41CX fest eingebaut sind. Da das Bedienungshandbuch zum Umgang mit X-Modulen nur knappe Hinweise gibt, war ein Buch nötig, das die Fähigkeiten der X-Funktionen und des HP-41CX vollständig beschreibt. Der Autor, ein führender Experte des HP-41 Systems und Pionier der synthetischen Programmierung, hat dieses Buch in seinem unnachahmlichen Stil – einfach, klar und doch präzise – geschrieben. Die vorliegende deutsche Ausgabe von Heinz Dalkowski wurde darüber hinaus in vielerlei Hinsicht inhaltlich ergänzt. Nach der Lektüre kann der Leser die X-Funktionen wirkungsvoll einsetzen und, wenn er Kenntnisse in synthetischer Programmierung besitzt, die Kraft dieser Kunst mit der der X-Funktionen verbinden. Insbesondere bekommt er über 30 ausgereifte Programme, die von den führenden Experten auf dem Gebiet stammen, an die Hand, darunter einen umfangreichen Text-Editor für den HP-41C/CV, ein Adressenverzeichnis-Programm, eine Simulation des HP-16, mathematische Programme, Programme zum Übertragen von Textdateien auf Magnetkarten, sowie – in Verbindung mit synthetischer Programmierung – Programme zum Reparieren fehlerhaften Verhaltens einiger speziellen Vorgänge beim Einsatz von X-Funktionen (Betriebssystemfehler). Sämtliche Programme sind als Barcodes abgedruckt, insgesamt 4181 Bytes! Wer aus seinem Kraftpaket herausholen will, was drin steckt, benötigt dieses Buch.

ISBN 3-88538-803-0