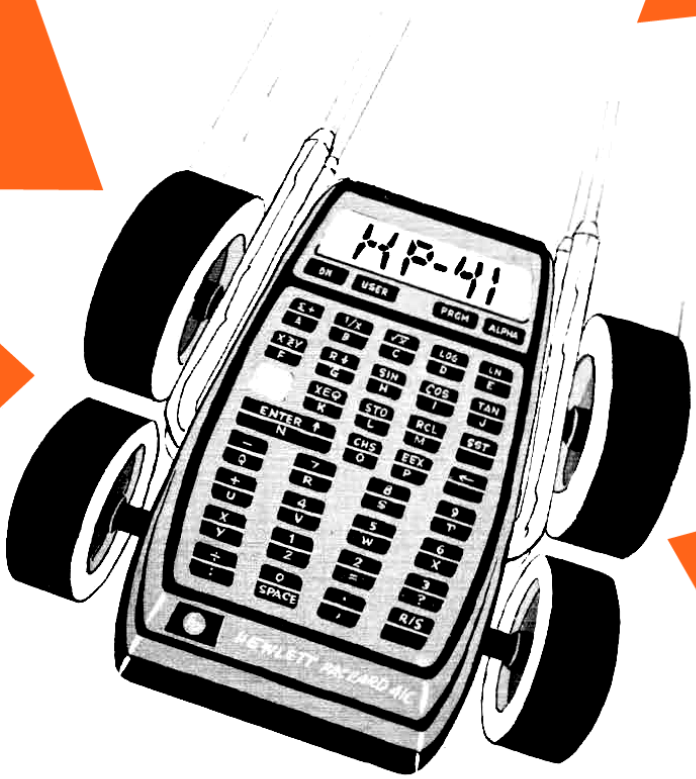
K. Jarett

Synthetisches Programmieren auf dem HP-41 – leicht gemacht

Deutsche Ausgabe von H. Dalkowski



Heldermann Verlag Berlin

Keith Jarett

Synthetisches Programmieren auf dem HP-41 – leicht gemacht

Deutsche Ausgabe von Heinz Dalkowski

Heldermann Verlag Berlin

Keith Jarett
Synthetix
2939 Winchester Drive
Hayward, CA 94541
U.S.A.

Heinz Dalkowski
Seidelbastweg 88a
D-1000 Berlin 47

CIP–Kurztitelaufnahme der Deutschen Bibliothek

Jarett, Keith:
Synthetisches Programmieren auf dem HP 41 - leicht gemacht / Keith Jarett.
Dt. Ausg. von Heinz Dalkowski. - Berlin : Heldermann, 1985.
Einheitssacht.: HP 41 synthetic programming made easy (dt.)
ISBN 3-88538-802-2
NE: Dalkowski, Heinz (Bearb.)

ISBN 3–88538–802–2

INHALTSVERZEICHNIS

VORWORT DES AUTORS

Dieses Buch wäre ohne den PPC nie entstanden. Mitglieder des PPC waren es, unter deren Fürsorge sich die synthetische Programmierung seit dem Erscheinen des HP-41C im Jahre 1979 so kräftig entwickeln konnte. Viele von ihnen haben unmittelbar zu den in diesem Buch vorgestellten Techniken beigetragen.

Die meisten Beiträge stammen von Clifford Stern, der zu der kleinen Spitzengruppe der 'Großmeister' der synthetischen Programmierung gehört. Er begleitete dieses Buch während seiner Entstehung, entwickelte mehrere Programme für die besonderen Zwecke hier und bestätigte sich erfolgreich beim Aufspüren von Fehlern während der verschiedenen Stufen der Herstellung.

Viele anderen PPC-Mitglieder sind auf Umwegen ebenfalls an diesem Buch beteiligt, durch ihre Entdeckungen und Entwicklungen nämlich, die die synthetische Programmierung während der letzten drei Jahre vorantrieben. Richard Nelson, dem Gründer des PPC, gebührt namentlich Dank und Anerkennung für seine Leistung, den PPC über die Dauer von 8 Jahren am Leben erhalten zu haben, und das ganz allein mit unermüdlicher Anstrengung.

Ich widme das Buch meiner Frau, Catherine Van de Rostyne, die es geduldig ertrug, wenn ich süchtig am HP-41 saß, und mir unschätzbare Hilfe während der Anfertigung dieses Buches zuteil werden ließ.

VORWORT DES ÜBERSETZERS

Mit der vorliegenden Übersetzung des 'Jarett' übergebe ich den deutschsprachigen Lesern nunmehr schon einen dritten Bestseller aus der amerikanischen HP-41 Literatur. Der Widerhall aus dem Leserkreis auf das seinerzeitige Erscheinen der deutschsprachigen Ausgabe des 'Wickes' war Auslöser für die nachfolgende deutschsprachige Ausgabe des 'Dearing', und das Echo auf beide Veröffentlichungen wiederum führte unausweichbar zur Arbeit an diesem dritten Buch, welchem, so hoffe ich, in absehbarer Zeit das vierte, die 'X-Funktionen' von Keith Jarett, folgen soll. Den zahlreich erhaltenen Leserbriefen kann ich entnehmen, daß man inzwischen überall in HP-41 Gefilden gleichmäßig gut unterrichtet ist, so daß ich auf Verständnis treffend einfach vom 'Wickes', vom 'Dearing' oder – so will ich sie hier taufen – vom 'Jarett I' und 'Jarett II' reden darf. Ich halte es daher auch für zulässig, die 'Büchergrenzen zu überschreiten' und auf diesem Forum einige Bemerkungen zu machen, die sich nicht nur auf die vorliegende Übersetzung beziehen. Es ist ja ohnehin so, daß die drei amerikanischen Autoren bestens miteinander bekannte PPC-Mitglieder sind, die wechselseitig billigend davon wissen, daß ihre Bücher eine einheitliche deutsche Terminologie erhalten, die also gewissermaßen (Moltke:) 'getrennt marschieren, aber vereint schlagen'.

Zunächst einmal Dank für die viele Leserpost, auf welche Publikation sie sich auch immer bezieht, und zudem die Zusicherung, daß jede Zuschrift beachtet und, sofern angezeigt, einzeln beantwortet wird. Eine Ausnahme in dieser Hinsicht mußte ich allerdings beim 'Dearing' machen; nur noch mit einem Formbrief war es möglich, der Flut falscher Vermutungen über das 'Hoch' zu wehren. Darum auch kurz ein Blick in die Rätselküche: a) Im 'Wickes' steht das Hoch auf den Autor auf S. 11. Nur wenige Leser haben es seinerzeit gefunden. b) Im 'Dearing' steht das Hoch auf den Autor gleich vorn auf der Umschlagseite. Ein geübtes Auge muß stutzen, wenn es den Deckel aufmerksam betrachtet. Frank Schneemann aus Hamburg war erfolgreich. Er wußte mit dem merkwürdigen Fund etwas anzufangen. Ein bißchen Kombinationsgabe, unterstützt durch die überschlägige Kenntnis des Buchinhaltes, führte ihn zum Ziel und der Gewißheit, das 'Hoch' wirklich entdeckt zu haben. c) Die Resonanz auf die Rätsel zwang mich, im vorliegenden 'Jarett I' das Spiel fortzusetzen. Wieder ist ein Lob auf den Autor verborgen. Diesmal unverschlüsselt, vielmehr – ich hoffe hinreichend listig – 'kaleidoskopiert'. Als Anreiz zur Suche ist erneut ein Preis ausgesetzt: ein kostenloses Exemplar des gerade entstehenden 'Jarett II'.

Wenn ich jetzt – endlich – ausdrücklich bekunde: 'die Arbeit am 'Jarett I' hat mir, wie vordem die am 'Wickes', großen Spaß gemacht; dem Autor, den man nicht erst vorzustellen braucht, gebührt hohe Anerkennung für seine sorgfältige und didaktisch ausgefeilte Einführung in die Synthetische Programmierung des HP-41', so ist dies – notabene – nicht das ausgepreiste Lob, sondern meine offene mit Überzeugung ausgesprochene Danksagung an KJ.

In ein Vorwort gehört auch die Anerkennung anderer wichtigen Beteiligten. Was im 'Dearing' sträflich vernachlässigt wurde, wird hiermit schuldbewußt nachgeholt: Dank an Christine Masuhr, die den 'Dearing' und den 'Jarett I' mit viel Sinn für äußere Gestaltung und vor allem mit unendlicher Geduld gegenüber meinen nicht enden wollenden Korrekturwünschen in die schließlich entstandene Form goß. Dank ebenso an Heinz Kröger, der in beiden Fällen die Stichwortverzeichnisse herstellte, wenn auch – das soll nicht unerwähnt bleiben – HP-fremd: auf einem Osborne. Sei's drum. Und Dank schließlich an den Verlag, genauer: an den Verleger Dr. Norbert Heldermann, der sich davon überzeugen ließ, daß er seinen bis dahin rein wissenschaftlichen Publikationen durchaus Literatur aus den Randbereichen der Mathematik und Informatik an die Seite stellen könne, und mit dem die Zusammenarbeit jederzeit angenehm, fruchtbar und unbürokratisch war.

Zum Schluß soll das Vorwort rasch noch als Fehlerbörse herhalten: a) 'Wickes', S. 154, Nullen anhängen statt abhängen. b) 'Dearing', S. 9, Zeile 3 in "BS" 'X<>d' statt 'X<>c' (eine wichtige Berichtigung!) – S. 38, 6–5, "VF" statt "VA" – S. 81, 14–2, 355/113 statt 355/133 – S. 171, die 'Anm. des Übers.' muß gestrichen werden – S. 176, 3. Zeile 255 statt 225.

Nach wie vor die schon im 'Dearing' ausgesprochene Bitte: Teilen Sie mir alle Fehler, auch harmlose, derer Sie habhaft werden, mit.

Und nun viel Vergnügen beim Durcharbeiten.

HD, im Juni 1984.

EINFÜHRUNG

Was heißt synthetisch programmieren?

Haben Sie sich schon einmal gefragt, warum Ihr HP-41 nur 10 verschiedene Töne zuläßt? Oder ist Ihnen schon einmal bewußt geworden, daß Sie Zahlen nicht im Alpha-Register ablegen bzw. von dort zurückrufen können und daß z.B. Klammern nicht als Zeichen für die Anzeige zur Verfügung stehen? 'Synthetisches Programmieren des HP-41 – leicht gemacht' zeigt Ihnen, wie Sie diese Beschränkungen überwinden können, und gibt Ihnen dazu eine ganze Reihe neuer Funktionen an die Hand, die Sie dem 'HP-41-Vokabular' eingliedern können. Hier einige Beispiele der verborgenen Fähigkeiten des HP-41:

- Techniken, mit denen Programme schneller, kürzer und weniger Datenregister verwendend gemacht werden können.
- Existenz mehrerer Notizregister von 'Stapel-Charakter' zum allgemeinen Gebrauch.
- Existenz 21 zusätzlicher Anzeige-Zeichen, darunter Klammern, Anführungszeichen, kaufmännisches Und "&".
- Existenz über 100 zusätzlicher Töne.
- Erweiterte Möglichkeiten der Alpha-Ketten-Behandlung.
- Aufhebung und Wiederbelebung der Tastenzuweisungen des Benutzers.
- Gleichzeitiges Setzen aller 56 Benutzer- und Systemflags in einen vorbestimmten Zustand.
- Programmiertes Umnumerieren der Datenregister zur Vermeidung von Doppelzugriffen durch verschiedene Unterprogramme.

Die Erzeugung und Verwendung synthetischer Befehle heißt 'Synthetisches Programmieren'. Synthetische Befehle können nicht durch normale Bedienung des Tastenfeldes erlangt werden. Dennoch sind unübersehbar viele synthetische Programmzeilen herstellbar, darunter synthetische Töne und leistungsfähige Befehle, die Zugriff auf Notizregister des Systems gestatten. Synthetische Programmierung beeinträchtigt den HP-41 selbst in keiner Weise; allerdings hat man – jedenfalls solange man noch lernt – gelegentliche kleine 'Katastrophen' (Einfrieren der Anzeige, Unbedienbarkeit des Tastenfeldes, "MEMORY LOST") zu erwarten, bekannt unter der Bezeichnung *GAU* (*g*rößter *a*nzunehmender *U*nfall).*) Synthetische Programmierung ist auf allen Rechnern der Familie HP-41 (C, CV und CX) möglich, unabhängig vom Herstellungsdatum. Sie beruht nur auf Eigenschaften, die das interne Betriebssystem, das allen HP-41 eigen ist, aufweist.

*) Beachten Sie in diesem Zusammenhang den Punkt 9 in Anhang C, wo von einem 'Weckmodul' die Rede ist, der Sie mancher diesbezüglichen Sorgen entheben kann.

In dem folgenden einfachen Beispiel, das Sie von der Nützlichkeit synthetischer Programmierung sogleich überzeugen sollte, steht links die – nicht-synthetische – Standard-Methode, den Text "Hewlett-Packard" auszudrucken. Dafür werden 40 Bytes des Programmspeichers benötigt (mehr über Bytes in Kapitel 1). Das Programm rechterhand benutzt eine synthetische Textzeile, um dieselbe Aufgabe mit nur 20 Bytes, exakt die Hälfte, zu lösen. In diesem Beispiel, dessen genaue Erklärung Ihnen in Abschnitt 2E begegnen wird, erlaubt die synthetische Programmierung den direkten Zugriff auf die Kleinbuchstaben des HP-41.

Programme zum Ausdruck des Textes "Hewlett-Packard"

nicht-synthetisch	synthetisch
01 "H"	01 "Hewlett-Packard"
02 ACA	02 AVIEW
03 SF 13	03 END
04 "EWLETT-"	
05 ACA	
06 CF 13	
07 "P"	
08 ACA	
09 SF 13	
10 "ACKARD"	
11 ACA	
12 PRBUF	
13 CF 13	
14 END	

Sie müssen ein Fachmann werden, um aus all den Vorteilen der synthetischen Programmierung Nutzen zu ziehen. Versehen mit den Kenntnissen und dem Selbstvertrauen, die dieses Buch Ihnen vermitteln wird, können Sie schnell und bequem *jedes* synthetische Programm schreiben und laufen lassen, stamme es aus der HP-Benutzer-Bibliothek, aus dem 'PPC Calculator Journal' [1] oder aus einer beliebigen anderen Quelle. Zugleich werden die häufigsten 'Dienstprogramme' synthetischer Programmierung bereitgestellt, damit Sie Ihren eigenen Programmen umstandslos synthetische Zeilen beimischen können.

[1] Das 'PPC Calculator Journal' (PPC CJ) ist eine vom 'Personal Programming Center' (PPC) herausgegebene Zeitschrift. Das PPC seinerseits ist eine nicht gewinnorientierte gemeinnützige Vereinigung in Kalifornien, die sich dem Betrieb sogenannter Personal-Computer widmet. Sie hat mehrere tausend Mitglieder, die meisten von ihnen HP-41 Enthusiasten. PPC-Mitglieder sind es, denen eigentlich jede Entdeckung auf dem Gebiet der synthetischen Programmierung zu verdanken ist, beginnend mit ihrer ersten Beschreibung durch William C. Wickes im PPC CJ im Jahre 1979. Das PPC CJ ist nach wie vor die erste Quelle für alle neuen Informationen über die synthetische Programmierung. Im Anhang C können Sie nachlesen, wie man das PPC CJ bestellen kann.

Dieses Buch ist so geschrieben, daß es eine ganz einfache und praktische Einführung in das synthetische Programmieren des HP-41 bildet. Es verwendet die jüngsten und einfachsten Techniken synthetischer Programmierung und verschafft damit einen unmittelbaren Zugang, der das Probieren der Beispiele während des Lesens zu einer leichten und vergnüglichen Übung macht. Die Stoffauswahl ist absichtlich begrenzt, um eine leicht lesbare Einführung in die synthetische Programmierung zu bieten. Einzelheiten werden häufig übersprungen, jedoch sind stets Hinweise gegeben, die den interessierten Leser weiterführen. Derjenige, der nur gelegentlich synthetisch programmieren will, wird alles, was er braucht, aus diesem Buch lernen. Für andere ist dieses Buch eine 'Eintrittskarte' in den ständig wachsenden 'Garten' der synthetischen Programmierung und gibt ihnen genau den Rahmen an die Hand, den sie benötigen, um ihre Kenntnisse voll auszubauen.

Wenn Sie einen PPC ROM [2] besitzen, können Sie sich beschleunigt durch das Buch arbeiten, indem Sie sich einfach der darin enthaltenen Programme höherer synthetischen Programmierkunst, nämlich des Tastenzuweisungsprogramms und des Programms zum Byte-Laden, bedienen. Wenn Sie nur den Rechner selbst haben, werden Sie manchmal den ausführlichen Anweisungen folgen müssen, um Ihr System Schritt für Schritt auf volle synthetische Leistung 'hochzufahren'. Jedenfalls sind beide Wege recht einfach zu beschreiten.

Hewlett-Packard unterstützt – aus naheliegenden Gründen – die synthetische Programmierung nicht. Obwohl viele Mitarbeiter der 'HP Corvallis Division' natürlich mit der synthetischen Programmierung vertraut sind, stellt HP keine Arbeitszeit zur Verfügung, um Anfragen der Kunden zur synthetischen Programmierung zu beantworten. Stellen Sie also bezüglich der synthetischen Programmierung *keine Fragen an HP*! Lesen Sie einfach dieses Buch, und suchen Sie in den im Anhang C angegebenen Quellen Antwort auf Ihre Fragen, sobald sie tiefer eindringen wollen.

Den größten Nutzen ziehen Sie mit 'Synthetisches Programmieren des HP-41 – leicht gemacht' aus der Tatsache, daß Ihnen alle veröffentlichten synthetischen Programme zugänglich werden. Viele synthetischen Programme im PPC ROM enthalten Funktionen, die von nicht-synthetischen Programmen nicht nachgeahmt werden können. Sobald Sie dieses Buch aber gelesen haben, werden Ihnen synthetische Programme nicht länger mysteriös oder unzugänglich erscheinen. Es gibt hunderte leistungsfähiger Programme im 'PPC Calculator Journal' und anderweitig, die Ihrem HP-41 Fähigkeiten verleihen, von denen Sie wahrscheinlich nie zu träumen gewagt hätten.

[2] Der PPC ROM ist ein Kunden-Modul für den HP-41, der von PPC-Mitgliedern entworfen wurde und von Hewlett-Packard hergestellt wird. Er umfaßt 122 Programme, von denen die meisten als Unterprogramme von Ihnen eingesetzt werden können. Sie enthalten fast alle synthetische Befehle. Das zugehörige Handbuch hat den erstaunlichen Umfang von 492 Seiten und ist vermutlich von noch niemandem ganz gelesen worden. Im Anhang C können Sie nachlesen, wie Sie den PPC ROM erwerben können.

KAPITEL 1

Sie erzeugen Ihre erste synthetische Funktion

Eine Dezimal*zahl* (Basis 10) xyz hat den Wert $x \cdot 10^2 + y \cdot 10 + z$, wobei x, y und z *Ziffern* zwischen 0 und 9 sind. Entsprechend versteht man unter einer Binärzahl (Basis 2) $qrst_2$ (der Index 2 weist auf die Basis 2 hin) den Wert $q \cdot 2^3 + r \cdot 2^2 + s \cdot 2 + t$, wobei q, r, s und t eine der beiden Ziffern 0 oder 1 sind. q beziffert die 'Achter' (so wie x bei der obigen Dezimalzahl die 'Hunderter'), r die 'Vierer' usw. Z.B. gilt $1011_2 = 8+2+1 = 11$ und $11111111_2 = 128+64+32+16+8+4+2+1 = 255$.

Eine Hexadezimalzahl (Basis 16) uv_{16} hat den Wert $u \cdot 16 + v$, wobei u und v Hexadezimal*ziffern* zwischen 0 und 15 sind. Weil es keine arabischen *Ziffern*, die den (Dezimal-)*Zahlen* 10 bis 15 entsprechen, gibt, entlehnt man sie gewöhnlich dem Alphabet: $A_{16}=10$, $B_{16}=11$, $C_{16}=12$, $D_{16}=13$, $E_{16}=14$ und $F_{16}=15$. Z.B. ist $C5_{16} = 12 \cdot 16 + 5 = 197$ und $FF_{16} = 15 \cdot 16 + 15 = 255$.

Sollten Sie mit Rechnungen zur Basis 2 oder 16 nicht vertraut sein, lesen Sie die beiden voranstehenden Absätze noch einmal, um sie ganz zu verstehen. Wie auch im verbleibenden Teil dieses Kapitels wird es Ihnen spätestens nach wiederholtem Lesen wie Schuppen von den Augen fallen. Knien Sie sich ruhig ein wenig hinein; als Lohn werden wir am Schluß des Kapitels einigen Spaß haben.

Die Grundeinheit des Programmspeichers im HP-41 heißt Byte. Ein Byte ist die Zusammenfassung von 8 Bits (Bit leitet sich aus der Abkürzung der englischen Bezeichnung *bi*nary digi*t* = Binärziffer her; es stellt die elementarste Informationseinheit dar) und kann daher Werte (zur Basis 2) von 00000000 bis 11111111, das entspricht dezimal 0 bis 255, annehmen. Obwohl ein Byte also nur 256 verschiedene Werte darzustellen vermag, gibt es dennoch tausende verschiedener HP-41-Befehle. Allein die 'STO'- und 'RCL'-Befehle lassen über 400 Variationen zu. Die Vielfalt entsteht dadurch, daß viele Befehle aus mehreren Bytes zusammengesetzt werden. Einfache Funktionen wie '+', 'LOG' oder 'MOD' belegen nur ein Byte im Programmspeicher; andere wie 'VIEW 14', 'RCL 99' oder 'ΣREG IND X' erfordern zwei Bytes: eines für den Namen der Funktion, genannt Vorsilbe, das zweite, die Nachsilbe, um den Bezug zum angesprochenen Register herzustellen. Einige wenige Befehlsarten besetzen drei Bytes, während Textzeilen sogar bis zu 16 Bytes (für 15 Zeichen) benötigen.

Synthetische Befehle kann man dadurch erzeugen, daß man die Vorsilbe von Zwei-Byte-Befehlen entfernt, indem man sich einer einfachen in diesem und im nächsten Kapitel beschriebenen Methode bedient. Sie werden an Hand der Beispiele dieses und des nächsten Kapitels sehen, wie das Entfernen einer Vorsilbe die Nachsilbe 'befreit' und sie ermächtigt, ihrerseits zu einer Vorsilbe zu werden, die nunmehr das Folge-Byte als Nachsilbe 'einfängt'. Durch sorgfältige Auswahl der Befehle, mit denen wir beginnen, können wir eine große Vielfalt synthetischer Befehle hervorbringen, sobald wir die ursprüngliche Vorsilbe entfernen. Zur Entfernung der

Vorsilbe bedienen wir uns eines einer Taste zugewiesenen 'Arbeitspferdes', des sogenannten 'Byte-Schnappers', der von Erwin Gosteli entdeckt wurde, nachdem Jack Baldrige einige Pionierarbeit geleistet hatte. Nebenbei bemerkt sind Erwin und Jack beide PPC-Mitglieder, und ihre Entdeckungen wurden im PPC CJ veröffentlicht. In der Tat sind alle Leute, die in Zusammenhang mit Entdeckungen oder Programmen in diesem Buch erwähnt werden, PPC-Mitglieder.

Weil der Byte-Schnapper keine gewöhnliche Tastenzuweisung ist, muß er mit einer besonderen Methode erzeugt werden. Es wird nun nicht von Ihnen erwartet, daß Sie an dieser Stelle das Verfahren zu seiner Erzeugung verstehen; folgen Sie lediglich sorgfältig den vorgeschriebenen Schritten. Denken Sie erst wieder mit, wenn der Byte-Schnapper auf der Taste liegt.

Sollten Sie Ihren HP-41 nicht schon vor sich liegen haben, nehmen Sie ihn jetzt zur Hand. Kommen Sie bitte nicht auf den Gedanken, erst zu lesen und dann die Beispiele nachzuvollziehen. Die Beispiele sind nämlich wesentlicher Teil des Lernprozesses, den Sie vor sich haben. Außerdem erleichtert Ihnen das Abarbeiten der Beispiele sehr das Lesen des Textes. Beispielsweise wird eine Anweisung wie 'gehen Sie auf Zeile 05, und löschen Sie diese' nur dann ganz klar, wenn Sie dies auch wirklich tun. Zwar scheint das Nachvollziehen der Beispiele das Lesen zu verzögern, aber im ganzen sparen Sie Zeit, weil ein späteres abermaliges Lesen entfällt.

Besitzer eines PPC ROMs können gleich mit Schritt 12 beginnen. Andernfalls weisen Sie den Byte-Schnapper gemäß der nachstehenden von Keith Kendall ersonnenen Prozedur zu. Folgen Sie dabei den Anweisungen äußerst präzise; sofern sich ein Fehler einschleichen sollte, beginnen Sie von vorn. Möglicherweise benötigen Sie ein paar Versuche, bis alles stimmt; seien Sie aber geduldig.

1. Totallöschung, um "MEMORY LOST" zu erhalten. Dies erlangt man dadurch, daß man den Rechner bei niedergedrückter Korrekturtaste einschaltet und diese erst dann losläßt. Es gibt zwar eine kompliziertere Prozedur, den Byte-Schnapper, ohne eine Totallöschung vorauszuschicken, zuzuweisen. Betrachten Sie aber diesen ersten Schritt gewissermaßen als einen obligaten Initialritus zur Einweihung in die Mysterien der synthetischen Programmierung. Außerdem wird es wohl nicht das letzte Mal sein, daß Ihnen ein "MEMORY LOST" unterläuft.

2. Weisen Sie '+' mit 'ASN' der Taste 'LN' zu. Diese Zuweisung wird später durch den Byte-Schnapper ersetzt.

3. Weisen Sie 'DEL' mit 'ASN' der Taste 'LOG' zu.

4. Schalten Sie in den PRGM-Modus. Wenn alles richtig ist, sehen Sie "00 REG 45" (beim HP-41CX "00 REG 218").

5. Starten Sie – im PRGM-Modus – 'CAT 1', und stoppen Sie mit 'R/S', noch bevor die Anzeige blinkt. Wiederholen Sie diesen Schritt, falls Sie 'R/S' nicht schnell genug gedrückt haben sollten.

6. Schalten Sie in den ALPHA-Modus, und drücken Sie – mit ".END. REG 45" in der Anzeige – die Korrekturtaste.

7. Sie müssen die Programmzeile '4094 RCL 01' erblicken. Die Ursache dieser mysteriösen Zeilennummer wird in Abschnitt 6A erklärt. Ein Fehlverhalten im Ablauf interner Vorgänge im HP-41 erlaubt nun hier die normalerweise unüberschreitbaren Grenzen im Programmspeicher zu überspringen. Wir befinden uns jetzt im Bereich der Notizregister des Systems. Auch hierüber mehr in Kapitel 6. Schalten Sie nun den ALPHA-Modus wieder aus.

8. 'GTO .005' (Sie können 'LN' für den numerischen Eintrag 005 benutzen, um 'Tastenarbeit' zu sparen). Als Ergebnis sollten Sie '05 LBL 03' sehen. Sie befinden sich in dem Speicherbereich, in welchem die Tastenzuweisungen festgehalten werden und der ebenfalls in Abschnitt 6A besprochen wird. Der nächste Schritt besteht darin, die 'Platzhalter'-Zuweisung der Funktion '+' durch die Zuweisung des synthetisch zu erzeugenden Byte-Schnappers zu ersetzen. Da der Rechner annimmt, daß er sich noch im Programmspeicherbereich befindet, läßt sich dieses Ersetzen dadurch vornehmen, daß man Programmbefehle eintastet, die gerade den Byte-Kombinationen entsprechen, welche für die Zuweisung des Byte-Schnappers erforderlich sind. Die Beziehung zwischen Programmbytes und Tastenzuweisungsbytes ist nicht ohne weiteres einzusehen; erwarten Sie also nicht, den Zusammenhang hier schon zu verstehen.

9. 'DEL 003'. Auch hier können Sie Tastenarbeit sparen: im USER-Modus wird die Zuweisung von 'DEL' (aus Schritt 3) aktiviert und somit der verlangte Löschvorgang durch 'LOG, SQRT' ausgeführt. Damit ist die Zuweisung von '+' gelöscht. Wir brauchen sie nur noch durch den Byte-Schnapper zu ersetzen.

10. Tasten Sie die Textzeile "?AAAAAA" ein. Wenn Sie keinen X-Funktionen-Modul im Rechner stecken haben, erblicken Sie "?A‾‾‾‾‾". Die letzten fünf "A" haben das Speicherende der Grundausstattung des HP-41 überschritten und sind in den Teil gelangt, der gegebenenfalls den Anfang des erweiterten Speichers bildet; daher ihre Anzeige als 'Geisterzeichen' "‾".

11. Schalten Sie den PRGM-Modus aus, und führen Sie 'GTO ..' oder 'CAT 1' aus, um die Tastenzuweisungsregister zu verlassen. Schritt 12 können Sie überspringen und mit dem daran anschließenden Text fortfahren.

12. Sind Sie im Besitz des PPC ROMs, oder steht Ihnen anderweitig bereits das in Kapitel 4 beschriebene "MK" (*m*ake *k*ey assignments) zur Verfügung, dann erlangen Sie den Byte-Schnapper vermittels der folgenden kurzen Prozedur:
    a) Löschen Sie alle u.U. vorhandenen Weckaufträge für den Time-Modul.
    b) Stellen Sie – mit 'ASN, ALPHA, ALPHA, LN' – die Taste 'LN' frei, falls diese eine Zuweisung trägt.
    c) 'XEQ MK' *) oder 'XEQ "MK"'.
    d) Wenn die Aufforderung "PRE↑POST↑KEY" erscheint, antworten Sie mit der Eingabe '247, ENTER↑, 63, ENTER↑, 15' und drücken anschließend 'R/S'. Sobald das Programm anhält, ist alles erledigt. Die abermalige Aufforderung "PRE↑POST↑KEY" können Sie mit der Korrekturtaste löschen, notwendig ist dies jedoch nicht.

*) Die Namen der Programme aus dem PPC ROM werden 'negativ' gesetzt.

Wenn Sie dem voranstehenden Verfahren mit der erforderlichen Sorgfalt gefolgt sind, liegt der Byte-Schnapper jetzt auf der Taste 'LN'. Rufen Sie ihn aber noch nicht auf; er kann sich als gefährlich erweisen, sofern Sie nicht vorsichtig genug mit ihm umgehen. Wenn Sie 'LN' im USER-Modus drücken und *die Taste gedrückt halten*, sollten Sie 'XROM 28,63', gefolgt von "NULL", sehen, womit angezeigt wird, daß die Zeitgrenze zur Ausführung der Funktion überschritten wurde. Beim Erscheinen von "NULL" wird die Byte-Schnapper-Funktion annulliert, und 'LN' kann ohne Gefahr wieder losgelassen werden. Ein paar Seiten weiter können Sie den Byte-Schnapper benutzen; bitte, seien Sie nicht ungeduldig. Ein wenig weitere Kenntnis kann Ihnen so manches "MEMORY LOST" ersparen.

Sollten Sie einen Kartenleser besitzen, ist es sinnvoll eine Statuskarte zu beschreiben, um Ihre neue synthetische Zuweisung dauerhaft aufzuzeichnen. Nach einem "MEMORY LOST" können Sie dann stets die Byte-Schnapper-Zuweisung wiedergewinnen, indem Sie die Spur 2 der Statuskarte einlesen; die Aufforderung zum Einlesen von Spur 1 darf mit der Korrekturtaste zurückgewiesen werden.

Bemerkung: In den nun folgenden Ausführungen soll für 'Byte-Schnapper betätigen' die Abkürzung 'BS' gelten und, falls nicht ausdrücklich anderes gesagt ist, so verstanden werden, daß die Taste 'LN' zu drücken ist, während der USER-Modus *und zugleich der PRGM-Modus* aktiv sind.

WARNUNG: Drücken Sie den Byte-Schnapper im PRGM-Modus nicht aufs Geratewohl! Wenn Sie sich dabei nämlich auf oder unmittelbar vor einem 'END' befinden, kann es sein, daß Sie nur noch nach einer Totallöschung wieder Zugang zu 'CAT 1' finden. (Zunächst sollten Sie in einem solchen Fall versuchen, mit 'BST' wieder auf die Zeile zu gelangen, die vor dem Aufruf des Byte-Schnappers angezeigt wurde, und dann müssen Sie ihn erneut bedienen.) Sollte Ihr Tastenfeld einmal blockiert sein, entfernen Sie die Batterien und, falls angeschlossen, den Drucker für einige Sekunden, und setzen Sie die Batterien dann wieder ein. Schlägt auch das fehl, versuchen Sie, den HP-41 ohne Batterien ein- und auszuschalten. Im Zweifelsfall hilft es, eingesteckte Module (insbesondere den Quad- und die X-Module) ebenfalls herauszunehmen. Nur in *sehr* seltenen Fällen wird es nötig sein, die Batterien eine ganze Nacht lang außerhalb des Rechners zu belassen. Es sei hier auch noch einmal auf den 'Weckmodul' (Punkt 9 in Anhang C) hingewiesen.

Wechseln Sie jetzt in den PRGM-Modus, führen Sie 'GTO ..' aus, und tasten Sie dann die folgenden Befehle, die wir gleich verwenden wollen, ein:

```
01 ENTER↑
02 X<> 00
03 STO IND 31
04 PI
```

Dem Handbuch können Sie entnehmen, wie man die HP-41-Funktionen, die nicht mehr aufs Tastenfeld passen, erreichen kann: mit 'XEQ, ALPHA, Funktionsname, ALPHA'. Die

Funktion 'X< >', die – außer nach Zuweisung – nicht auf dem Tastenfeld liegt, gehört dazu. Die Lage der für diese Funktionen benötigten Alpha-Zeichen " < " und " > " finden Sie auf dem auf der Rückseite des Rechners befindlichen Aufkleber. Für Anfänger sei noch darauf hingewiesen, daß indirekte Befehle mit der SHIFT-Taste erzeugt werden: Zeile 03 entsteht aus 'STO, SHIFT, 3, 1'.

Bevor Sie den Byte-Schnapper verwenden, müssen Sie noch ein wenig mehr über Bytes erfahren. Legen Sie darum Ihren Rechner noch für ein paar Minuten beiseite, währenddessen Sie sich die nächsten zwei Seiten zu Gemüte führen.

Für das Verständnis der Vorgänge beim synthetischen Programmieren ist die Darstellung der 256 möglichen Werte eines Bytes als Hexadezimalzahl (Basis 16) besonders geeignet. Zerlegt man die acht Bits eines Bytes in zwei Gruppen zu je vier Bits und schreibt man für jede dieser beiden Vierergruppen die Hexadezimal*ziffer*, die sie darstellt, auf, erhält man für den Wert eines Bytes eine *zweiziffrige* Hexadezimal*zahl*. In hexadezimaler Schreibweise bezeichnen die Buchstaben A bis F die (Dezimal-)Zahlen 10 bis 15. Die folgende Tabelle gibt die Äquivalenz der verschiedenen Zahlendarstellungen unserer Vierergruppen wieder:

binär	hexadezimal	dezimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15
10000	10	16

Mithin hat z.B. das Byte 0100 1101 den Hexadezimalwert 4D, und das Byte 1111 0001 heißt in Hexadezimaldarstellung F1.

Nehmen Sie jetzt die dem Buch als Plastik-Karte beigefügte Kurzanleitung zur synthetischen Programmierung zur Hand, oder schlagen Sie die im Anhang D abgedruckte Byte-

Tabelle auf. Die Byte-Tabelle ist der 'Stein von Rosette'*) der synthetischen Programmierung. Sie offenbart die verschiedenen Bedeutungen der Bytes im Programmspeicher und bildet daher den Schlüssel zur Erzeugung synthetischer Befehle. Auf die Kurzanleitung beziehen wir uns unter der Abkürzung 'QRC' (= *Q*uick *R*eference *C*ard).

Für jedes Byte enthält die Byte-Tabelle ein Kästchen, dessen Lage durch die Hexadezimaldarstellung zs_{16} des Bytes bestimmt ist: z und s (beide zwischen 0 und F) sind die Zeilen- bzw. Spaltennummer, unter denen man das Kästchen des Bytes zs_{16} finden kann. Die Zeilen 0 bis 7 bilden den ersten Teil der Byte-Tabelle, die Zeilen 8 bis F den zweiten Teil. Im oberen Teil jedes Kästchens steht die Bedeutung des jeweiligen Bytes als Einzelfunktion bzw. Vorsilbe; unmittelbar darunter steht seine Bedeutung als Nachsilbe. Im unteren Teil des Kästchens ist der Dezimalwert des Bytes notiert. Rechterhand untereinander findet man die Zeichen, mit denen das entsprechende Byte in der Anzeige bzw. auf dem Drucker dargestellt wird; wir benötigen sie im Abschnitt 2E. (Beachten Sie auch die Erläuterungen zur QRC und zur Byte-Tabelle in Anhang D.)

Betrachten Sie als Beispiel die Funktion 'ENTER↑', die Sie eben als Zeile 01 eingetastet haben. Sie finden diese Funktion im oberen Teil des in Zeile 8, Spalte 3 befindlichen Kästchens. Mithin wird 'ENTER↑' intern als 83_{16} dargestellt. Sie können dem Kästchen zugleich entnehmen, daß 83_{16} der Dezimalzahl 131 äquivalent ist. Für diese Erkenntnis haben Sie zwar im Augenblick noch keine unmittelbare Verwendung, aber im Kapitel 3 werden Sie es sehr angenehm finden, sich so bequem sachkundig machen zu können.

Nehmen wir uns als nächstes den Befehl 'X< >88' aus Zeile 02 vor. Die Funktion 'X< >' finden Sie in Zeile C, Spalte E, und '88' ist die im Kästchen von Zeile 5, Spalte 8 eingetragene Nachsilbe. 'X< >88' ist also ein Zwei-Byte-Befehl, der intern mit CE_{16} 58_{16} verschlüsselt ist. Zeile 03 lautet 'STO IND 31'. 'STO' steht in Zeile 9, Spalte 1, während 'IND 31' die Nachsilbe aus Zeile 9, Spalte F ist. Daher besteht 'STO IND 31' aus den zwei aufeinanderfolgenden Bytes 91 9F (wir lassen von jetzt ab, sofern keine Mißverständnisse auftreten können, den Index 16 als Hinweis auf Hexadezimaldarstellung weg). Zeile 04, 'PI', ist Byte 72. Beachten Sie, daß die Zeilennummern nicht im Programmspeicher mit enthalten sind; der HP-41 berechnet sie jedesmal, wenn sie benötigt werden, neu, indem er die Befehle vom Programmanfang aus durchzählt.

Nehmen Sie nun an, wir könnten uns auf irgendeine Weise des Bytes CE, also der Vorsilbe 'X< >' des Befehls 'X< >88', entledigen. Dann bliebe die Nachsilbe '88' (Byte 58) zurück und müßte plötzlich 'für sich selbst sorgen': sie würde gezwungenermaßen zu der Ein-Byte-Funktion 'E↑X−1' aus dem Kästchen in Zeile 5, Spalte 8.

Der Byte-Schnapper ist es nun, der es uns tatsächlich auf ganz bequeme Weise erlaubt, die führenden Bytes in beliebigen Programmzeilen zu beseitigen. Aus diesem Grunde wird er auch manchmal als 'Vorsilben-Erzeuger' bezeichnet. Der Byte-Schnapper fängt stets das erste Byte derjenigen Programmzeile weg, die auf die gerade angezeigte Zeile *folgt*.

Nehmen Sie jetzt wieder Ihren HP-41 zur Hand, schalten Sie ihn ein, und prüfen Sie, ob Ihr Programm noch unversehrt besteht, indem Sie sich seine Zeilen mit 'SST' ansehen.

Wir werden uns gleich die Wirkung des Byte-Schnappers auf den Befehl 'X< >88' ansehen. Vorher müssen Sie jedoch noch ein 'PACK' ausführen, und zwar mit 'XEQ, ALPHA, P, A, C,

*) Anm. des Übers.: In Rosette (Ägypten) wurde 1799 eine für die Entzifferung der Hieroglyphen wichtige Steintafel, die jetzt im Britischen Museum aufbewahrt wird, gefunden.

K, ALPHA', um im Programmspeicher an der Stelle zu verbleiben, an der Sie gerade sind. Ein 'GTO ..' hat nämlich die hier unerwünschte Nebenwirkung, dem Programm, in welchem Sie sich gerade befinden, erstens ein 'END' anzufügen und Sie zweitens aus diesem Programm 'herauszuwerfen'. Schalten Sie in den USER-Modus. Führen Sie dann 'GTO .001' aus, um auf die Zeile *vor* 'X< >88' zu gelangen. Gehen Sie in den PRGM-Modus, falls Sie bislang noch im RUN-Modus waren, *und jetzt endlich zum ersten Mal BS* (einfacher Druck auf die Taste 'LN'). Als Ergebnis erblicken Sie eine fremdartig anmutende Textzeile:

02 ᵀ⁻?⁻⁻⁻⁻▓.

Das Vollzeichen ▓ (sämtliche 14 Segmente eines Anzeigenelementes eingeschaltet) am Zeilenende ist – genauer *war* – die Vorsilbe 'X< >' des Befehls 'X< >88'. Dieses Byte CE ist vom Byte-Schnapper ergriffen worden mit dem Erfolg, daß die zurückbleibende Nachsilbe, Byte 58, zur selbständigen Funktion zu werden gezwungen war. Führen Sie 'SST' aus, um diese Behauptung bestätigt zu finden:

03 E↑X−1.

Sollten Ihnen die Vorgänge im Programmspeicher noch nicht ganz klar sein, gehen Sie das Beispiel nochmals durch. Wenn Sie die Byte-Struktur des Speichers und die Wirkungsweise des Byte-Schnappers einmal durchschaut haben, sind Sie über den Berg und auf dem besten Wege zur richtigen synthetischen Programmierung.

Was wird sich wohl ereignen, wenn wir die Vorsilbe 'STO' des Befehls 'STO IND 31' fortnehmen? Die Nachsilbe 9F ist laut Byte-Tabelle Vorsilbe eines 'TONE'-Befehls. Ein 'TONE'-Befehl aber bedarf seinerseits einer Nachsilbe, denn er ist ein *Zwei*-Byte-Befehl. Woher aber nimmt ein freigestelltes (also nicht mehr selbst als Nachsilbe dienendes) Byte 9F die erforderliche Nachsilbe? Wir werden das sogleich erforschen: BS auf Zeile 03 ('GTO .003', falls Sie nicht schon dort sind, und 'LN' im USER- und PRGM-Modus), um das 'STO'-Byte wegzuschaffen. 'SST', tatsächlich

05 TONE Y, ein synthetischer Befehl!

Ein Blick in die Byte-Tabelle Zeile 7, Spalte 2 enthüllt, daß sich die Vorsilbe des 'TONE'-Befehls skrupellos das nächste Byte, nämlich die Ein-Byte-Funktion 'PI', als Nachsilbe 'unterworfen' und als Y angezeigt hat (vgl. Sie Abbildung 1.1). Es ist leicht einzusehen, daß sich der 'TONE'-Befehl seine Nachsilbe aus dem folgenden Programmbefehl holte – er bedurfte ihrer schließlich, um vollständig zu sein; und woher hätte er sie sonst bekommen können?

Sie können sich Ihren neuen synthetischen Ton im RUN-Modus mit 'SST' anhören; 'BST' und 'SST', wenn Sie ihn abermals hören wollen. Es gibt mehr als 100 synthetische Töne, die ihrer Entdeckung harren!

hexadezimale Byte-Werte	Programm-Befehle	Programm-Befehle nach Byte-Schnappen
83	ENTER↑	ENTER↑
CE	X<>	┬─?────
58	88	E↑X-1
91	STO	┬─?────
9F	IND 31	TONE
72	PI	Y

Abbildung 1.1: Umwandlung von Programm-Befehlen durch Byte-Schnappen.

KAPITEL 2

Häufig benutzte synthetische Befehle

Dieses Kapitel behandelt die acht am häufigsten benutzten Typen synthetischer Befehle. Unabhängig davon, ob Sie sich später einmal damit befassen, synthetische Programme exotischen Charakters zu schreiben, werden Sie bald die hier besprochenen und leicht zu verstehenden Befehle für Ihre Programmiertätigkeit schätzen lernen und sie einbeziehen. Wir werden die folgenden Befehlstypen in diesem Kapitel behandeln:

A. synthetische Töne, die Ihren Programmen eine persönliche Note verleihen;
B. synthetische Eingabe von Exponenten (sogenannte 'Kurzform-Exponenten'), die Bytes einsparen;
C. Flag-Register Kontrolle, geeignet zur Bewahrung des Anzeige-Modus bei 'PROMPT'-Befehlen;
D. Adreßzeiger Kontrolle, mit der Sie die fliegende 'Graugans' 'festnageln' können;
E. synthetische Textzeilen, in denen Sie Kleinbuchstaben oder synthetische Zeichen, z.B. Klammern, direkt unterbringen können;
F. den 'TEXT 0'-Befehl, ein 'No Operation'-Befehl, äquivalent zum Befehl 'NOP' des HP-25;
G. Kontrolle über Datenregister, die durch 'Tranchieren' des Alpha-Registers entstehen und als Hilfsregister für Zwischenergebnisse dienen können, ohne daß Inhalte numerischer Datenregister zerstört werden; und schließlich
H. Benutzung weiterer Notizregister des Betriebssystems für die vorübergehende Ablage von Daten.

Insoweit Beispiele für synthetische Befehle in diesem Kapitel dargelegt werden, sind auch die Methoden, sie zu erzeugen, Schritt für Schritt beschrieben. Dabei bedienen wir uns ständig des in Kapitel 1 erzeugten Byte-Schnappers, den wir der Taste 'LN' zugewiesen haben. Besitzer des PPC ROMs haben die Möglichkeit, diese Prozeduren zu überspringen und die synthetischen Befehle unmittelbar mit der PPC ROM-Routine LB (*l*oad *b*ytes) in den Programmspeicher zu bringen. Die dafür notwendigen LB-Eingaben werden für jedes Beispiel mitgeliefert. Wenn ein synthetischer Befehl aus zwei Bytes besteht und kein Zahleneintrag ist, kann statt LB auch die PPC ROM-Routine MK benutzt werden, falls man gleichzeitig eine Tastenzuweisung dieses Befehls wünscht. Trotzdem ist auch den PPC ROM Besitzern zu empfehlen, dieses oder jenes Beispiel der Übung halber 'zu Fuß' durchzuführen und den Byte-Schnapper statt der Routinen MK bzw. LB einzusetzen.

Für diejenigen unter den Lesern, die keinen PPC ROM haben, wird eine Kurzfassung von "LB" – zusammen mit Anweisungen zur Benutzung des Byte-Schnappers für das Eintasten dieses Programms – im Kapitel 3 angeboten. Sie könnten also an dieser Stelle schon "LB"

herstellen, aber Sie lernen mehr über die Verwendung des Byte-Schnappers, wenn Sie abwarten, bis Sie auf dem vorgesehenen Wege zum Kapitel 3 gelangen und dann erst "LB" edieren und benutzen.

2A. Synthetische Töne

Am Schluß von Kapitel 1 wurde erwähnt, daß es über 100 verschiedene synthetische Töne gibt; sie variieren in Tonhöhe und Tondauer. Es gibt insgesamt 16 verschiedene Tonhöhen, von denen die ersten 10 in den bekannten Tönen 'TONE 0' bis 'TONE 9' auftreten. Die Dauer synthetischer Töne liegt zwischen mehreren Millisekunden (Töne, die nur noch als 'Klick' zu hören sind) und mehreren Sekunden. Für viele Gelegenheiten, z.B. bei 'PROMPT's, ist ein verhältnismäßig kurzer hoher Ton besonders geeignet. 'TONE 89' ist ein solcher Ton. Er kann folgendermaßen erzeugt werden. Löschen Sie, was aus den Beispielen in Kapitel 1 zurückgeblieben ist, und tasten Sie diese Programmzeilen ein:

```
01 ENTER↑                    LB / MK -Eingaben
02 STO IND 31                TONE 89 = 159, 89
03 SIN
```

Gehen Sie nun, im PRGM-Modus verbleibend, mit 'GTO .001' auf Zeile 01 zurück und dann BS ('LN' im USER-Modus). Wie schon gewohnt erblicken Sie diese Textzeile: 02ᵀ⁻?⁻⁻⁻⁻. Mit 'SST' gelangen Sie auf Ihren neuen synthetischen Befehl, der so angezeigt wird: 03 TONE 9. Er sieht zwar nicht synthetisch aus, aber durch Anhören können Sie sich davon überzeugen, daß er es ist.

Das Byte 9F ('IND 31') wurde zur 'TONE'-Vorsilbe, als das BS das Byte 91 ('STO') verschlang. Das Byte 59 ('SIN') geriet dadurch zur Ton-Zahl 89 ('TONE'-Nachsilbe). Von synthetischen Ton-Zahlen zwischen dezimal 10 und 101 erscheint in der Anzeige nur die äußerste rechte Ziffer. Daher wird 'TONE 89' als 'TONE 9' angezeigt. Andere Töne, deren zweite Bytes zwischen hexadezimal 66 und 7F liegen, erhalten in der Anzeige als Nachsilbe einen Buchstaben wie z.B. 'TONE Y' im Beispiel von Kapitel 1.

Schalten Sie um in den RUN-Modus, um 'TONE 89' mit 'SST' anzuhören. Vielleicht werden auch Sie ihn für 'PROMPT's bevorzugen.

Die Tabelle 2.1 führt alle synthetischen Töne, die Ihnen zur Verfügung stehen, auf. Die Tonhöhe wird durch die Spaltennummer (hexadezimal 0 bis F) bestimmt. Dabei entsprechen den Spalten A bis F wachsende Tonhöhen, deren höchste (Spalte F) jedoch noch unter derjenigen des normalen Tones 'TONE 0' liegt.

Die Ton-Dauer in Sekunden wird in der Tabelle für jeden Ton angegeben. Dabei handelt es sich um die Gesamtzeit, die der HP-41 zur Ausführung des jeweiligen 'TONE'-Befehls benötigt. Aus diesem Grunde ist die Zeit, während der der Ton tatsächlich erklingt, hörbar kürzer, insbesondere bei den 'flüchtigen' Tönen. Außerdem kann die Ton-Dauer in Abhängig-

Tabelle 2.1

HP-41 Ton-Tabelle: Ton-Dauer und XROM-Zahlen

In jedem Kästchen ist die (dezimale) Ton-Zahl, die Ausführungsdauer (sie kann, in Abhängigkeit vom Herstellungsdatum von der hier angegebenen abweichen) und die XROM-Zahl vermerkt.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0 0.28 60,00	1 0.28 60,01	2 0.28 60,02	3 0.28 60,03	4 0.28 60,04	5 0.28 60,05	6 0.28 60,06	7 0.28 60,07	8 0.27 60,08	9 0.27 60,09	10 2.08 60,10	11 2.42 60,11	12 3.37 60,12	13 0.67 0.80 60,13	14 2.30 60,14	15 0.35 60,15
1	16 1.82 60,16	17 0.32 60,17	18 1.43 60,18	19 0.29 60,19	20 0.48 60,20	21 0.94 60,21	22 0.45 60,22	23 0.82 60,23	24 0.29 60,24	25 0.49 60,25	26 4.70 60,26	27 3.23 60,27	28 1.75 60,28	29 3.85 60,29	30 3.46 60,30	31 2.37 60,31
	32 .022 60,32	33 1.10 60,33	34 2.25 60,34	35 1.90 60,35	36 1.17 60,36	37 .020 60,37	38 .020 60,38	39 0.35 60,39	40 0.65 60,40	41 0.49 60,41	42 0.83 60,42	43 0.43 60,43	44 3.80 60,44	45 1.71 60,45	46 1.29 60,46	47 0.12 60,47
	48 0.50 60,48	49 0.26 60,49	50 2.04 60,50	51 1,85 60,51	52 0.29 60,52	53 0.14 60,53	54 0.75 60,54	55 0.77 60,55	56 0.62 60,56	57 .046 60,57	58 4.07 60,58	59 3.99 60,59	60 3.19 0.39 60,60	61 3.77 0.39 60,61	62 0.93 60,62	63 0.27 60,63
4	64 1.79 61,00	65 2.29 61,01	66 0.16 0.40 61,02	67 0.19 61,03	68 1.01 61,04	69 0.25 61,05	70 .072 .032 61,06	71 0.21 61,07	72 0.13 61,08	73 0.15 61,09	74 3.58 61,10	75 0.28 61,11	76 3,60 61,12	77 3.30 61,13	78 0.85 61,14	79 0.87 61,15
5	80 .075 61,16	81 0.22 61,17	82 1,68 61,18	83 0.72 61,19	84 0.30 61,20	85 1.16 61,21	86 0.46 61,22	87 .093 61,23	88 0.56 61,24	89 .038 61,25	90 2.61 61,26	91 0.39 61,27	92 3.12 61,28	93 3.78 61,29	94 0.30 61,30	95 2.45 61,31
6	96 0.62 61,32	97 2.21 61,33	98 0.41 61,34	99 1.21 61,35	100 0.11 61,36	101 1.27 61,37	102 A 0.96 61,38	103 B 0.80 61,39	104 C 0.64 61,40	105 D 0.45 61,41	106 E 2.26 0.23 61,42	107 F 0.43 61,43	108 G 3.54 61,44	109 H 0.31 61,45	110 I 2.00 61,46	111 J 2.33 0.33 61,47
7	112 T 0.25 *1.66* 61,48	113 Z .061 *0.64* 61,49	114 Y 0.55 *1.40* 61,50	115 X 1.19 *0.48* 61,51	116 L 0.40 *1.20* 61,52	117 M [1.07 61,53	118 N \ 0.22 61,54	119 O] 0.78 61,55	120 P ↑ 0.13 61,56	121 Q _ 0.32 61,57	122 ← 0.29 61,58	123 a 4.38 61,59	124 b 0.73 61,60	125 c 3.77 61,61	126 d 3.45 61,62	127 e 2.84 61,63

keit vom Herstellungsdatum des Rechners schwanken, z.T. erheblich: z.B. ist 'TONE Z' auf neueren Modellen 0,64 Sekunden, auf den ältesten dagegen nur 0,061 Sekunden lang.

Wenn Sie sich die Ton-Tabelle genauer ansehen, stellen Sie fest, daß 'TONE 37' und 'TONE 38' mit jeweils 0,02 Sekunden die kürzesten sind. Das folgende Beispiel zeigt, wie sie passend verwendet werden können. Löschen Sie zunächst das vorangegangene Beispiel, und tasten Sie dann diese Programmzeilen ein:

```
01 DEG                    LB / MK -Eingaben:
02 CLX
03*LBL 01
04 STO IND 31             TONE 37 = 159, 37
05 RCL 05
06 SIN
07 SQRT
08 STO IND 31             TONE 38 = 159, 38
09 RCL 06
10 SIN
11 SQRT
12 GTO 01
```

Anschließend 'GTO .007', BS, Textzeile löschen, 'SST', um 'TONE 38' (angezeigt als 'TONE 8') zu betrachten, 'GTO .003', BS, Textzeile löschen, 'SST', um 'TONE 37' (angezeigt als 'TONE 7') zu erblicken. Schalten Sie nun in den RUN-Modus, und tasten Sie 'RTN' und 'R/S'. Der HP-41 ahmt jetzt das charakteristische Ticktack einer Pendeluhr nach.

Synthetische Töne lassen sich vielseitig einsetzen. Im Anhang B z.B. wird ein in der Praxis verwendbares Morse-Programm hoher Signaldichte, das synthetische Töne benutzt, vorgeführt. Die folgende Abbildung 2.1 können Sie dazu benutzen, die für Ihre eigenen Anwendungen passenden Töne herauszusuchen. Sie können Tonhöhe und -Dauer im voraus wählen und dann den synthetischen Ton ermitteln, der dem, den s/e brauchen, am nächsten kommt. Tabelle 2.1 und Abbildung 2.1 stammen von Robert E. Swanson und werden mit seiner Erlaubnis hier abgedruckt. Er stellte diese Daten für das von Cary E. Reinstein herausgegebene 'HP-41/HP-IL System Dictionary' zusammen.

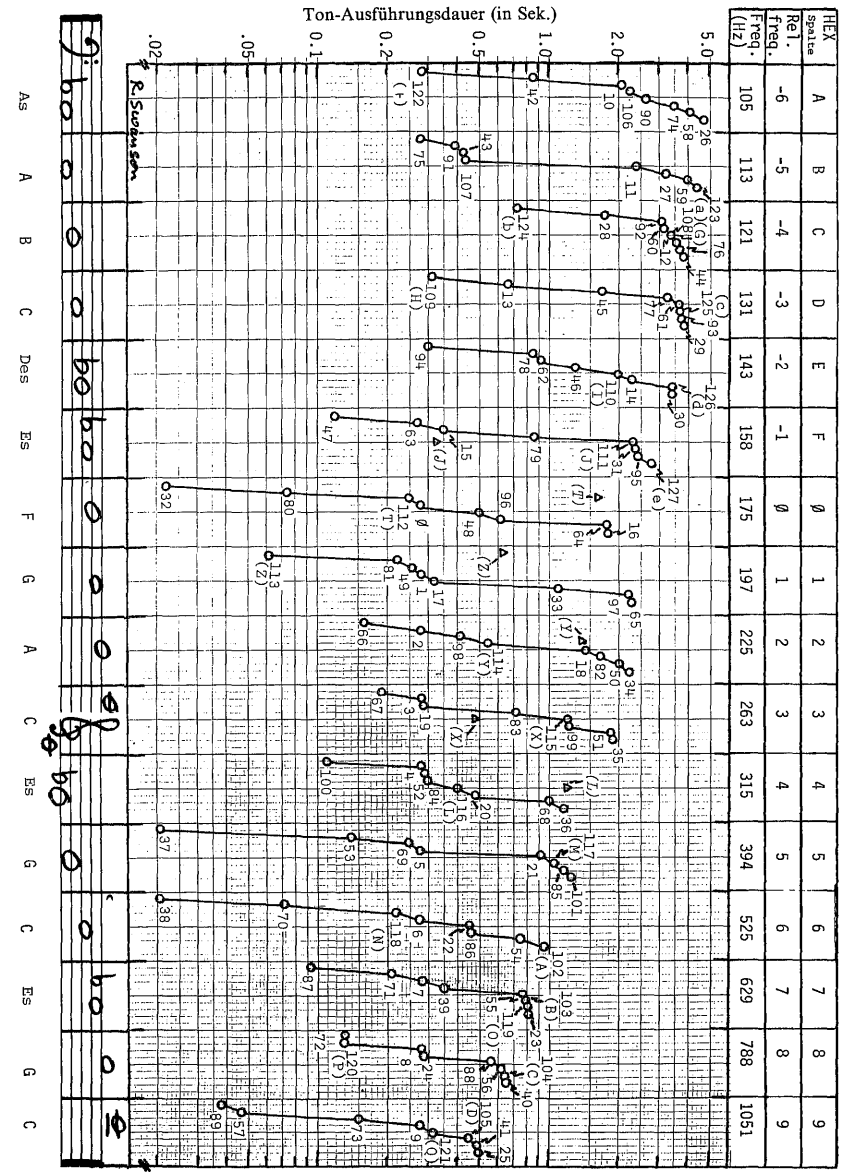


Abbildung 2.1

2B. Synthetische Eingabe von Exponenten

Drückt man im RUN-Modus 'EEX, CHS, 3', erhält das X-Register den Eintrag 1×10^{-3}. Tun Sie dasselbe im PRGM-Modus, entsteht ein Befehl der Form '1 E−3', obwohl Sie eigentlich nur E−3 getastet haben. Der Rechner beharrt offensichtlich darauf, eine überflüssige 1 dazuzusetzen und auf diese Weise ein Byte zu verschwenden. Ich wette, daß Sie – inzwischen im Besitz des Byte-Schnappers – selbst vermuten können, wie man sich dieser lästigen 1 entledigt. Löschen Sie das vorangegangene Beispiel, und tasten Sie

```
01 ENTER↑                    LB -Eingaben:
02 1 E-3                     E-3 = 27, 28, 19
```

ein. Diesmal müssen Sie dem BS ein 'PACK' voranschicken, und zwar so wie in Kapitel 1 mit 'XEQ, ALPHA, P, A, C, K, ALPHA', *nicht* mit 'GTO ..', was einfacher wäre. Denn 'GTO ..' setzt Sie 'aufs Trockene' und läßt Ihnen nur noch die Möglichkeit, mit 'CAT 1' zeitraubend in Ihr Programm zurückzukehren. Sie können übrigens ein wenig Suchzeit und Tastenarbeit sparen, wenn Sie 'PACK' mit 'ASN' einer freien Taste zuweisen und dann im USER-Modus durch einfachen Tastendruck aufrufen.

Jetzt 'GTO .001', BS und Textzeile löschen (das Vollzeichen am Ende der Textzeile war die überflüssige 1); dann 'SST', und Sie sehen 02 E−3, eine synthetische Exponenten-Eingabe, meist als 'Kurzform-Exponent' bezeichnet.

Prüfen Sie diesen synthetischen Befehl mit 'SST' auf seine Wirkung, und Sie werden finden, daß er ebenso wie '1 E−3' arbeitet. Offenbar haben Sie ein Byte im Programmspeicher eingespart, doch sollten Sie auch wissen, daß dieser synthetische Eintrag obendrein schneller ausgeführt wird.

Ausführungszeit – allerdings nicht Programmspeicherplatz – kann auch eingespart werden, wenn man statt des Eintrags einer 0 das Dezimalkomma benutzt und statt des Eintrags einer 1 ein Exponenten-E ('EEX'). Das einzeln stehende Dezimalkomma ist kein synthetischer Eintrag, aber das einzeln stehende E ist ein solcher. Man erzeugt dieses einsame E, indem man den Byte-Schnapper z.B. auf den Befehl 'STO 27' ansetzt und das Vorsilben-'STO' beseitigt. Der Byte-Tabelle können Sie entnehmen, daß die Nachsilbe 27 (Byte 1B) alleinstehend dem Befehl 'EEX' entspricht.

Weiter oben ist darauf hingewiesen worden, daß dem BS ein 'PACK' voranzugehen hat, wenn die führende 1 eines Exponenten-Eintrages weggeschnappt werden soll. Der Grund dafür ist die Tatsache, daß sämtlichen numerischen Einträgen ein unsichtbares 'NULL'-Byte (Byte 00) vorangestellt wird, welches allein dazu dient, neue Zahleneinträge von vorangegangenen zu trennen. Verwechseln Sie diese 'NULL' nicht mit der Meldung "NULL", die im Anschluß an einen in der Anzeige auftauchenden Funktions- oder Programmnamen erscheint, sobald man die entsprechende Taste länger als 2 Sekunden gedrückt hält. Wie man sich denken kann, ist das 'NULL'-Byte ein 'Platzhalter', ein 'leerer Befehl', der nichts bewirkt, wenn er 'ausgeführt' wird (es sei denn, Byte 00 stellt eine Nachsilbe dar wie z.B. in 'X<>00' oder 'ΣREG 00'). 'NULL'-Bytes sind – außer in Textzeilen – stets unsichtbar. Sie entstehen beim Löschen von Befehlen und werden (sofern sie nicht *zwischen* unmittelbar aufeinanderfolgenden Zahlenein-

trägen stehen) durch 'PACK' beseitigt. Wir gehen genauer darauf in Kapitel 5 ein.

Im ersten Beispiel dieses Abschnitts benutzten wir 'PACK', um die 'NULL', welche der HP-41 zwischen 01 ENTER↑ und 02 1 E−3 setzte, zu beseitigen. Wäre Zeile 01 ein Zahleneintrag gewesen, hätte 'PACK' die 'NULL' nicht behelligt. Sie wäre dazu nötig gewesen, die Trennung zwischen den Zahleneinträgen aufrechtzuerhalten. Außer in diesem Sonderfall beseitigt 'PACK' die 'NULL'en, welche zwischen Befehlen stehen, immer.

Es gibt aber noch eine andere Möglichkeit, ein 'NULL'-Byte aus dem Weg zu schaffen, nämlich es einfach zu 'überschreiben', indem man den von der 'NULL' freigehaltenen Platz mit einer Ein-Byte-Funktion füllt. Wollen wir dies anhand unseres Beispiels prüfen. Löschen Sie die Zeile 02, und tasten Sie

```
01 ENTER↑
02 1 E-3
```

ein. Jetzt befindet sich eine unsichtbare 'NULL' zwischen den beiden Zeilen 01 und 02. Weil wir die 1 aus '1 E−3' wegschnappen wollen, nicht die vor '1 E−3' stehende 'NULL', überschreiben wir diese zuerst: 'GTO .001' oder einfach 'BST', dann 'RDN'. 'RDN' ist ein Ein-Byte-Befehl, der die vom 'NULL'-Byte freigehaltene Stelle besetzt. Anschließend BS, um die führende 1 wegzuschnappen, dann zweimal Korrekturtaste, und wir haben

```
01 ENTER↑
02  E-3
```

Fügen wir also der am Anfang dieses Abschnitts beschriebenen Prozedur zwei Tastendrucke hinzu, so entfällt das andernfalls notwendige 'PACK'. Dies ist insbesondere dann von Vorteil, wenn Sie einen synthetischen Exponenten-Eintrag in einem langen Programm vornehmen wollen, bei welchem das 'PACK' mehrere Sekunden in Anspruch nimmt.

In Kapitel 5 wird das etwas schwer faßbare Verhalten der 'NULL'-Bytes ausführlich erklärt und veranschaulicht. Man bedarf einer synthetischen Technik, um es sichtbar zu machen. Ehrgeizige Programmierer, die sich das luxuriöse Vergnügen eines synthetischen Befehls '− E' (Eintrag einer −1) leisten wollen, müssen beachten, daß negative Vorzeichen in Zahleneinträgen durch das Byte 1C ('NEG'), *nicht* etwa durch das Byte 54 ('CHS') dargestellt werden. 'CHS' ist allein für den Vorzeichenwechsel einer bereits im X-Register befindlichen Zahl zuständig.

2C. Kontrolle des Flag-Registers

Normalerweise muß man den Anzeige-Modus ändern, wenn in einem Programm eine Alpha-Meldung, die Zahlen beinhaltet, ausgegeben werden soll. Z.B. fordert die Befehlsfolge

```
01 1,01
02 STO 00
03 FIX 0          } diese beiden Schritte werden dazu benötigt,
04 CF 29          } die Registernummern ohne Dezimalkomma anzugeben.
05*LBL 01
06 "EINGABE "       (beachten Sie die Leerstelle hinter dem
07 ARCL 00          letzten Buchstaben)
08 "+?"
09 TONE 9
10 PROMPT
11 STO IND 00
12 ISG 00
13 GTO 01
```

zu (von 1 bis 10) numerierten Eingaben auf und legt diese dann in den Registern R_{01} bis R_{10} ab. Leider ändern die Zeilen 03 und 04 den Anzeige-Modus, um die Aufforderung zur Eingabe optisch befriedigend zu gestalten. Synthetisches Programmieren eröffnet einen leicht zu beschreitenden Weg, die Veränderung in Fällen wie dem voranstehenden zu vermeiden.

Es ist jetzt an der Zeit, einen kurzen Ausflug in die Welt der Flags zu unternehmen. Weil ein Flag nur zwei Zustände annehmen kann – gesetzt oder gelöscht – ist es sinnvoll, im Rechner ein Bit dafür zu benutzen, um den Zustand eines Flags darzustellen. Und zwar wird der Zustand 'gesetzt' durch den Wert 1 des Bits und der Zustand 'gelöscht' durch Wert 0 dargestellt. Wie wir im Kapitel 1 gesehen haben, besteht ein Byte aus acht Bits. Aus dem Handbuch des HP-41 wissen Sie, daß ein Register aus sieben Bytes besteht. Mithin gibt es $8 \times 7 = 56$ Bits in einem Register. Diese Zahl kommt Ihnen gewiß bekannt vor: der HP-41 besitzt genau 56 Benutzer- und Systemflags, die Flags F_{00} bis F_{55}. Daher kann es Sie nicht überraschen, wenn Sie hiermit erfahren, daß die 56 Flags genau ein Register des HP-41 mit Beschlag belegen.

Das Flag-Register ist eines der 16 Notizregister des Betriebssystems. Sie kennen bereits die ersten fünf davon: die Stapelregister T, Z, Y, X und L. Die Namen aller Systemregister finden Sie in Zeile 7 der Byte-Tabelle. Der Name des Flag-Registers lautet d (Zeile 7, Spalte E).

Nun zurück zu unserem Fall. Wir wollten den Anzeige-Modus bewahren, derweil eine Alpha-Meldung, die Zahlen ohne Dezimalkomma beinhaltet, zusammengestellt wird. Zu diesem Zweck können wir ein 'RCL d' vor den Aufbau der Meldung setzen, um den Inhalt des Flag-Registers d ins Register X zu bringen.

Nach dem Aufbau der Meldung können wir dann mit dem Befehl 'STO d' diesen Flag-Register-Inhalt wieder von X nach d zurückbringen. Auf diese Weise wird der ursprüngliche Zustand aller 56 Flags – mithin auch der ursprüngliche Anzeige-Modus – wiederhergestellt.

In unserem Beispiel vom Beginn dieses Abschnitts wird dies so durchgeführt. Sie tasten

```
01 1,01                    LB /MK -Eingaben:
02 STO 00
03*LBL 01
04 "EINGABE "
05 STO IND 16              RCL d = 144,126
06 AVIEW
07 FIX 0
08 CF 29
09 ARCL 00
10 STO IND 17              STO d = 145,126
11 AVIEW
12 "⊢?"
13 TONE 9
14 PROMPT
15 STO IND 00
16 ISG 00
17 GTO 01
```

ein; dann 'GTO .009', BS, Textzeile löschen; 'SST', um 'STO d' zu sehen; 'GTO .004', BS, Korrekturtaste und 'SST', um 'RCL d' zu sehen. Die Nachsilbe 'IND 17' (Byte 91) wurde zu 'STO' und die Nachsilbe 'IND 16' (Byte 90) zu 'RCL'. Der Befehl 'AVIEW' (Byte 7E) wurde in beiden Fällen zur Nachsilbe der neugeschaffenen Vorsilben 'STO' und 'RCL'. Auch diese Programmversion fordert zur Eingabe in die Datenregister R_{01} bis R_{10} auf, doch anschließend ist der Anzeige-Modus unverändert im Gegensatz zu der entschieden unfreundlichen 'FIX 0'-Darstellung, die bei der ersten Programmversion zurückbleibt.

```
01 1,01
02 STO 00
03*LBL 01
04 "EINGABE "
05 RCL d
06 FIX 0
07 CF 29
08 ARCL 00
09 STO d
10 "⊢?"
11 TONE 9
12 PROMPT
13 STO IND 00
14 ISG 00
15 GTO 01
```

Sie können das Befehlspaar 'RCL d'/'STO d' an jeder Stelle eines Programms, an der Sie die durch Flags bestimmten Zustände des Rechners (Anzeige-Modus, trigonometrischer Modus usw.) aufbewahren wollen, benutzen. Der Inhalt von Register d darf jedoch *nur in Stapelregistern*, nicht in numerierten Datenregistern abgelegt werden. Daten, die aus numerierten Registern abgerufen werden, unterliegen nämlich einer Prüfung und u.U. einer daraus resultierenden Veränderung, die *Normalisierung* genannt wird: sobald die 56 Bits des Flag-Registers eine Konfiguration bilden, die der HP-41 (bei Abruf aus einem Datenregister) nicht als eine Alpha-Kette oder eine Zahl erkennt, ändert er sie eigenmächtig so ab, daß eine Alpha-Kette oder eine Zahl entsteht. Damit würde das nach d zurückzuspeichernde Bit-Muster nicht mehr mit dem ursprünglichen Muster übereinstimmen, und unser Zweck wäre verfehlt.

Die ins Einzelne gehende Beschreibung der Bit-Muster, die als Alpha-Ketten oder Zahlen erkannt werden, liegt außerhalb des Rahmens dieses Buches (in dem Buch 'Synthetische Programmierung auf dem HP-41C/CV' von W.C. Wickes, das im Anhang C, Punkt 8 aufgeführt ist, finden Sie ausreichend Auskunft darüber). Für unsere Zwecke reicht eine 'Faustregel zur Normalisierung': Jedes aus 56 Bits bestehende Muster, dessen erste vier Bits 0001 lauten, kann ohne Gefahr, Veränderungen zu erleiden, in Datenregistern abgelegt und von dort zurückgerufen werden. Haben die ersten vier Bits jedoch einen davon abweichenden Wert, werden die Daten einer Normalisierung und damit einer möglichen Abänderung unterworfen, sobald man sie (z.B. mit 'RCL' oder 'X< >') abruft. Dies ist im Normalfall, also im Fall von Alpha-Ketten oder Zahlen problemlos. Die Normalisierung wird erst dann zum Problem, wenn man mit nicht-standardisierten Bit-Mustern, wie sie u.a. im Flag-Register entstehen, umgeht.

Wollen Sie also den Flagzustand in einem Datenregister ablegen, müssen Sie vorab dafür sorgen, daß die ersten vier Bits 0001 lauten. Dies läßt sich in der Tat leicht bewirken, wie das gleich folgende Beispiel lehren wird. Löschen Sie das vorangegangene Beispiel bis auf die Befehle 'RCL d' und 'STO d'; dann 'GTO .000' und anschließend

```
01 CF 00   ┐   die ersten vier Zeilen versetzen
02 CF 01   │   die ersten vier Bits des Registers
03 CF 02   │   d in den erforderlichen Zustand
04 SF 03   ┘   0001
05 RCL d
06 STO 01
07 GRAD
08 SF 01
09 CF 03
10 STOP
11 RCL 01
12 STO d
```

eintasten. Verlassen Sie den PRGM-Modus, 'RTN', dann 'R/S'. Jetzt ist Flag 1 gesetzt, und der Rechner befindet sich im GRAD-Modus. Nach einem zweiten 'R/S' kehrt der Rechner in seinen ursprünglichen Zustand (F_{00}, F_{01}, F_{02} gelöscht, F_{03} gesetzt) zurück. Wenn Sie nichts

gegen ein Beispiel einzuwenden haben, das nach Ausführung ein bißchen Aufräumarbeit in Ihrem Flag-Register erfordert, ändern Sie die Zeile 01 in 'SF 00' ab, und beobachten Sie, daß der Zustand mehrerer Flags diesmal verändert ist, wenn das Programm nach dem zweiten 'R/S' anhält. Sie können schneller zum alten Zustand zurückkehren, wenn Sie dazu die im Stapelregister Z zurückbleibende Kopie des ursprünglichen Zustandes benutzen. Weil diese Kopie nicht aus einem Datenregister abgerufen wurde, ist sie unverändert: 'RCL Z', 'GTO .012' und 'SST' stellen den alten Zustand wieder her.

Eine besonders nützliche Anwendung der durch synthetische Befehle ermöglichten vollen Kontrolle über das Flag-Register ist die gezielte Löschung von Flag 55 bei Anwesenheit des Druckers. Ist ein solcher nämlich angeschlossen, wird die Ausführung von Programmen verlangsamt, unabhängig davon, ob er an- oder ausgeschaltet ist, und auch unabhängig vom Zustand von Flag 21. Sogar Befehle, die mit dem Drucker überhaupt nichts zu tun haben, werden langsamer ausgeführt. Dieses Geschwindigkeitsopfer kann vermieden werden, wenn man Flag 55, welches bei Anwesenheit des Druckers automatisch vom Betriebssystem gesetzt wird, synthetisch löscht. Jede der drei nachstehenden Befehlsfolgen vollbringt das Gewünschte.

'nackter' HP-41	X-Funktionen vorhanden	PPC ROM vorhanden
SF 07 *)	RCLFLAG **)	55
RCL d	SIGN	FC? 55
CLA	STO d	RDN
STO M } ***)	X<>L	FS? 55
ASTO M }	STOFLAG	XROM IF
⊢⁻	RDN	
X<>M		
STO d		
RDN		

*) statt Flag 07 kann auch jedes der Flags F_{00} bis F_{06} benutzt werden
**) die Routine stammt von Steve Wandzura
***) Register M wird in Abschnitt 2G besprochen.

Solange ein laufendes Programm keiner Druckerfunktion begegnet, bleibt Flag 55 gelöscht, so daß die Ausführungsgeschwindigkeit erhöht wird. Ist zugleich Flag 21 gelöscht, setzt selbst das Auftreten einer Druckerfunktion Flag 55 nicht mehr: die Funktion wird einfach ignoriert, so wie man es auch sonst schon von gelöschtem Flag 21 her kennt. Ist Flag 21 dagegen gesetzt, hängt das Programmverhalten vom Typ des Druckers ab. Ein Drucker 82143A stellt alle seine Funktionen solange 'dienstunfähig', bis das Programm anhält. Zu diesem Zeitpunkt wird Flag 55 wieder gesetzt (übrigens auch Flag 21, falls es gelöscht war). Beim HP-IL-Drucker hingegen legt der Zustand von Flag 21 fest, ob Flag 55 wieder gesetzt und die Funktion ausgeführt wird; ein einfacher Programmstop wie beim Drucker 82143A reicht dazu nicht, doch ein beliebiger Test, ein 'VIEW' oder eine verwandte Funktion bewirken dies, sofern sie über das Tastenfeld ausgeführt werden.

2D. Kontrolle des Adreßzeigers

In einem der Notizregister des Betriebssystems hält sich der HP-41 einen Adreßzeiger, dessen Inhalt bestimmt, welcher Teil des Programmspeichers im PRGM-Modus in der Anzeige erscheint. Das Systemregister, welches den Adreßzeiger (zusammen mit einem Teil der sogenannten Rücksprungadressen – diese werden in Abschnitt 6A dieses Buches und im Benutzerhandbuch des PPC ROMs unter 'Line by Line Analysis of LR' besprochen) enthält, heißt Register b.

Löschen Sie das vorangegangene Beispiel, und tasten Sie die folgenden Zeilen ein, um sich vor Augen zu führen, wie leicht der Adreßzeiger zu kontrollieren ist:

```
01 ENTER↑                LB / MK-Eingaben:
02 STO IND 16            RCL b = 144, 124
03 MEAN
04 STO IND 31            TONE 89 = 159, 89
05 SIN
06 STO IND 17            STO b = 145, 124
07 MEAN
```

Anschließend 'GTO .005', BS, Korrekturtaste, 'GTO .003', BS, Korrekturtaste, 'GTO .001', BS, zweimal Korrekturtaste, 'PACK' (nicht 'GTO ..'). Übergang in den RUN-Modus, 'RTN' und 'R/S'. Sie hören ein schnelles Stakkato auf 'TONE 89'. Die 'fliegende Graugans' ist eingefroren.

Wie kommt dieses Verhalten des Rechners zustande? Der Befehl 'RCL b' legt den Adreßzeiger ins X-Register, 'TONE 89' wird ausgeführt, und anschließend gibt 'STO b' den gerade aus Register b abgerufenen Wert wieder dorthin zurück. Zu dem Zeitpunkt, zu dem der Adreßzeiger mit 'RCL b' aus Register b abgerufen wird, lautet der als nächstes auszuführende Programmbefehl 'TONE 89'. Aus diesem Grunde verursacht der Befehl 'STO b' einen Rücksprung auf 'TONE 89'. Sie können das direkt beobachten, wenn Sie (im RUN-Modus) den Adreßzeiger mit 'RTN' auf den Programmanfang zurücksetzen und dann mehrfach 'SST' tasten: Sie erblicken 'RCL b, TONE 89, STO b, TONE 89, STO b, ...'.

Der Grund dafür, daß die Graugans während des Programmlaufs stillsteht, ist ganz einfach der: die Graugans bewegt sich immer dann in die nächste Position der Anzeige, wenn eine Marke ('LBL') erreicht wird. Aber es gibt keine Marken in diesem Programm, trotz der Schleife. Daher hat auch die Graugans keinen Anlaß, sich zu bewegen.

Das nächste Beispiel gibt eine Antwort auf die einfache Frage: Wie stellt man die 'kürzeste Dauerschleife' auf dem HP-41 her? Die Antwort ist eine Programmzeile lang – 2 bescheidene Bytes: Löschen Sie 'TONE 89', und 'PACK'en Sie. Es bleibt

```
01 RCL b
02 STO b
```

zurück. Wenn Sie jetzt (im RUN-Modus) 'RTN' tasten und mit 'SST' das Programm 'abarbeiten', sehen Sie, daß diese Befehlsfolge ausgeführt wird: 'RCL b, STO b, STO b, STO b, ...' – ad infinitum; gleichzeitig werden aber die Zeilennummern hochgezählt. Denn die Programmausführung mit 'SST', also im 'Einzelschrittverfahren', veranlaßt den HP-41, die Programmzeilen hochzuzählen, außer nach Befehlen des Typs 'GTO', 'XEQ' oder 'END', bei deren Ausführung die Zeilennummern neu von vorn berechnet werden. Den Befehl 'STO b' aber erkennt der Rechner nicht als 'Sprungbefehl'; folglich macht er sich auch keine Sorgen über die Zeilennumerierung. Wenn Sie einen geduldigen Finger haben und unermüdlich 'SST' tasten, werden Sie herausfinden, daß die Zeilennummern bis 4094 durchgezählt werden und es dann wieder mit 02 von vorn beginnt. In Abschnitt 6A wird Ihnen klar werden, daß es intern eine besondere Bewandtnis mit der Nummer 4095 hat: sie ist es, die dem HP-41 signalisiert, daß mit der Zeilenzählung neu begonnen werden muß.

Läuft ein Programm frei ab, also nicht im Einzelschrittverfahren, werden die Zeilennummern überhaupt nicht ausgerechnet. Dies würde die Ausführung überflüssigerweise verlangsamen.

Weitere Kenntnis der Techniken synthetischer Programmierung wird allerdings noch benötigt, bevor Sie die ganze 'Kraft' des Befehls 'STO b' richtig nutzen können. Das ultraschnelle Morsezeichen-Programm in Anhang B ist ein Beispiel dafür: es führt Ihnen vor, wie mit Hilfe einer mittelbaren Kontrolle des Adreßzeigers ein indirektes Verzweigen möglich wird, bei dem die Ansprungadressen 'vorzeitig', also ohne Zeitverzögerung durch irgendwelche Suchvorgänge, zur Verfügung stehen. Ferner bietet die Befehlsfolge '0, STO b, GTO .002' eine einfache Möglichkeit, den Adreßzeiger in die Tastenzuweisungsregister zu schicken. Einzelheiten darüber, wie die Tastenzuweisungen dort verschlüsselt liegen, findet man im Benutzerhandbuch des PPC ROMs unter 'Background for MK' und in 'Synthetische Programmierung auf dem HP-41C/CV' von W.C. Wickes.

2E. Synthetische Textzeilen

Der HP-41 unterscheidet sich von seinen Vorgängern ganz erheblich dadurch, daß man die Möglichkeit hat, mit ihm alphanumerisch tätig zu werden, ihn also 'textverarbeitend' einzusetzen. Diese Fähigkeit wird in erster Linie dazu benutzt, Ein- und Ausgaben mit Text zu versehen. Die Anzahl der zur Verfügung stehenden Zeichen ist aber doch etwas begrenzt; z.B. fehlen Klammern und Anführungszeichen.

Synthetisches Programmieren erlaubt es, 21 zusätzliche Anzeigesymbole in Text-Befehlen zu verwenden, darunter runde Klammern, Anführungszeichen, Apostroph, kaufmännisches Und. Diese synthetisch erzeugbaren Symbole können nach einer Methode, die gleich beschrieben wird und unseren Byte-Schnapper benutzt, in Text-Befehle gebracht werden. Mit PPC ROM-Programmen hat man zwei weitere Methoden zur Verfügung. Die einfachste davon ist die, mit LB synthetische Textbefehle direkt zu erzeugen. Die zweite bedient sich der sogenannten 'Q-Übertragung' und erfordert als Unterstützung das Programm DC aus dem PPC ROM. Die Anwendung von LB wird in Kapitel 3, die der Q-Übertragung im Verein mit DC in Abschnitt 4B vorgeführt.

Die in diesem Abschnitt zu besprechende Methode zum Erzeugen von Textzeilen ist recht einfach und erfordert so gut wie keine 'Infrastruktur', lediglich den tastenzugewiesenen Byte-

Schnapper. Sie sollten daher unabhängig von der Verfügbarkeit anderer Methoden den Beispielen dieses Abschnitts folgen. Vielleicht werden Sie es für die bequemste Methode halten, den Byte-Schnapper einzusetzen, wenn nur ein oder zwei synthetische Textzeilen herzustellen sind.

Besitzer eines Druckers oder eines X-Funktionen-Moduls sind – wegen der Funktionen 'BLDSPEC' bzw. 'XTOA' – mit weiteren, allerdings programmtechnisch schwerfälligeren Verfahren, synthetische Anzeigesymbole zu erzeugen, vertraut. Hier werden wir zeigen, daß synthetisch erzeugte Textzeilen gegenüber normal erzeugten, wie sie bei der Verwendung von 'BLDSPEC' oder 'XTOA' entstehen, außerordentlich Byte-sparend und überdies Laufzeit verkürzend sind.

Der Aufbau einer Textzeile aus n Zeichen ist denkbar einfach. Ein Fn-Byte (Zeile F, Spalte n, $0 \leq n \leq 15$) geht n Bytes voran, deren jedes ein Zeichen darstellt. Daher belegt eine Textzeile von n Zeichen im Programmspeicher n+1 Bytes. Der Byte-Tabelle können Sie entnehmen, welches Byte zu welchem Zeichen gehört. Z.B. wird Byte 5F sowohl in der Anzeige als auch auf dem Drucker durch "_" dargestellt. Manche synthetisch erzeugten Zeichen erscheinen in der Anzeige in ganz anderer Form als auf dem Drucker. Z.B. zeigt sich Byte 04 in der Anzeige als "⌅", auf dem Drucker jedoch als "α". Ein Byte wird nur dann als Zeichen interpretiert, wenn es im 'Machtbereich' oder 'Einzugsbereich' eines Fn-Bytes steht, also eines der auf ein Fn-Byte folgenden n Bytes ist. Steht kein Fn-Byte voran, werden Programmbytes als Befehle oder Nachsilben voranstehender Befehle interpretiert. Die Bytes der Zeile F kann man als 'TEXT'-Befehle auffassen, die im allgemeinen mehrerer Nachsilben (zwischen 0 und 15) zu ihrer Ergänzung bedürfen. Der wesentliche Unterschied zwischen 'TEXT'-Befehlen und den meisten anderen Funktionen ist der, daß die Anzahl der Nachsilben hinter einem 'TEXT'-Byte variabel ist, abhängig nämlich von der zweiten Hälfte des 'TEXT'-Bytes, und daß ein 'TEXT'-Befehl eine vom Normalfall abweichende Interpretation der Nachsilben auslöst, nämlich eine Interpretation als Zeichen.

Synthetische Textzeilen können mit dem Byte-Schnapper in einer vier Schritte umfassenden Prozedur erzeugt werden. Zuerst tastet man eine Textzeile der gewünschten Länge ein, wobei man an den Stellen, an welchen synthetische Zeichen stehen sollen, beliebige Zeichen (wir bevorzugen hier "X") einsetzt. Dann schnappt man das voranstehende 'TEXT'-Byte weg. Dies 'befreit' die nachfolgenden Bytes von ihrer 'Pflicht', als Zeichen bildende Nachsilben zu dienen und macht sie zu 'freien Bürgern' im Programmspeicher: zu selbständigen Befehlen. In dieser Form können die "X" ohne Umstände durch solche Befehle ersetzt werden, die den gewünschten synthetischen Zeichen entsprechen. Zum Schluß entläßt man das weggeschnappte Fn-Byte aus seiner Funktion als letztes Zeichen der beim BS entstandenen Textzeile "¯?¯¯¯¯▒", wodurch es seinerseits wieder seine ursprüngliche Aufgabe als 'Beherrscher' der n folgenden Bytes übernimmt. Unter diesen Folge-Bytes befinden sich jetzt die neu edierten Bytes, welche gegen die "X" (Byte 88, dem Befehl 'E↑X−1' entsprechend) ausgetauscht wurden, und das 'TEXT'-Byte Fn zwingt sie nun dazu, zusammen mit den unveränderten Bytes als Zeichen zu erscheinen.

Ein Beispiel wird Ihnen dies alles völlig klarmachen. Nehmen wir an, Sie wollen die Textzeile "HP'S #1" erzeugen. Löschen Sie wie üblich das letzte Beispiel, und tasten Sie

```
01 ENTER↑          LB -Eingaben:
02 "HPXS X1"       247, 72, 80, 39, 83, 32, 35, 49
```

ein. Anschließend 'GTO .001', BS, aber *auf keinen Fall* die entstandene Textzeile "⁻?⁻⁻⁻⁻■" mit der Korrekturtaste löschen. Sie enthält nämlich das weggeschnappte Byte F7, das Führungsbyte unserer herzustellenden Textzeile, das wir gleich wieder benötigen. Wenn Sie mehrere 'SST's ausführen, sehen Sie jetzt diese Programmzeilen:

```
01 ENTER↑
02 "⁻?⁻⁻⁻⁻■"
03 Σ-
04 LN
05 E↑X-1
06 Y↑X
07 RCL 00
08 E↑X-1
09 STO 01
```

Die Zeilen 03 bis 09 entsprechen jede einem der Zeichen der ursprünglich edierten Textzeile. Z.B. gehört 'RCL 00' zum Zeichen 'SPACE' (Byte 20). In Zeile 2, Spalte 0 der Byte-Tabelle können Sie diese Zusammengehörigkeit feststellen. Wir werden nunmehr die Befehle 'E↑X−1', die den "X" entsprechen, passend ersetzen. In Zeile 08 unseres Programms müßte nach Plan ein Befehl stehen, der dem Symbol "#" entspricht. Wir finden dieses Zeichen in Zeile 2, Spalte 3 der Byte-Tabelle und entnehmen ihr, daß ein 'RCL 03' das 'E↑X−1' ersetzen muß: 'GTO .008', Korrekturtaste, 'RCL 03'. Auf dieselbe Weise ergibt sich: 'GTO .005', Korrekturtaste, 'RCL 07', denn 'RCL 07' entspricht dem Apostroph "'" (Byte 27).

Wenn Sie die voranstehenden Anweisungen sorgfältig befolgt haben, brauchen Sie kein 'PACK' mehr auszuführen; es kann aber an dieser Stelle auch nicht mehr schaden. Sie sollten nun

```
01 ENTER↑
02 "⁻?⁻⁻⁻⁻■"
03 Σ-
04 LN
05 RCL 07
06 Y↑X
07 RCL 00
08 RCL 03
09 STO 01
```

im Programmspeicher haben. Mit 'GTO .001' und BS schnappen Sie jetzt das Führungsbyte der vom ersten Byte-Schnapper zurückgebliebenen Textzeile "⁻?⁻⁻⁻⁻■" weg. Dadurch werden das Fragezeichen "?" (Byte 3F) und das Vollzeichen "X" (Byte F7) eigenständige Befehle: mit 'SST' stoßen Sie daher folgerichtig zuerst auf den Befehl 'STO 15' und dann auf die in

gewünschter Gestalt erscheinende Textzeile "HP'S #1', denn Byte F7 hat seine frühere Rolle als 'TEXT 7'-Byte wieder aufgenommen und zwingt die 7 ihm folgenden Bytes, Zeichen darzustellen. Die außerdem zurückbleibenden 5 'NULL'-Bytes, welche als "." erschienen waren, bleiben unsichtbar zurück (vgl. Abschnitt 2B). Folglich lautet Ihr Programm

```
01 ENTER↑
02 "⁻?----■"
03 STO 15
04 "HP'S #1"
```

Wenn Sie einen Drucker haben, werden Sie vergleichen wollen, wie die neuen synthetischen Zeichen dort ausgedruckt werden. (Das können Sie natürlich auch ohne Drucker als 'Trockenübung' tun: in der Byte-Tabelle steht in jedem Kästchen unter dem Anzeigesymbol rechts unten das Drucksymbol.) Sie werden feststellen, daß Anzeige- und Drucksymbol für die Zeichen "#" und "?" nicht voneinander abweichen. Aber das Vollzeichen "■" aus Zeile 02 verschwindet bei der Programm-Auflistung spurlos. Diese 'Feigheit vor dem Feind', das Verschwinden aus Programmausdrucken (nicht aus Alpha-Register-Ausdrucken) ist jedem Byte aus der zweiten Tabellenhälfte, Zeilen 8 bis F, eigen. Wir kommen am Schluß dieses Abschnitts noch einmal darauf zurück.

Der zu den HP-41-Befehlen gehörige 'Anhangsbefehl' bedarf zu seiner Durchführung zweier 'Kontrollbytes'. Er besteht aus einem 'TEXT'-Befehl aus Zeile F, einem darauffolgenden Anhangssymbol "⊢" (Byte 7F) und den anzuschließenden eigentlichen Zeichen. Weil also das Anhangssymbol eine Stelle einer Textzeile besetzt und eine Textzeile nicht mehr als 15 Zeichen aufnehmen kann, können höchstens 14 Zeichen an einen bereits (im Alpha-Register) bestehenden Text angefügt werden. Wird das Anhangssymbol allerdings synthetisch in eine Textzeile eingefügt, kann es auch eine andere Stelle als die erste hinter dem Fn-Befehl einnehmen, verliert dort dann aber seine bevorrechtigte Rolle als 'Kontrollbyte' und wird zu einem 'ordinären' Zeichen degradiert.

Wir wollen uns jetzt einige synthetischen Zeichen in einen Anhangsbefehl holen. Löschen Sie das vorangegangene Beispiel und tasten Sie

```
01 ENTER↑
02 "⊢ABCDEFGHIJKL"
```

ein, dann 'GTO .001' und BS, aber *nicht* die Korrekturtaste drücken. Die durch BS entstandene Zeile enthält nämlich das aus der ursprünglichen Textzeile, Zeile 02, stammende 'TEXT 13'-Byte und bewahrt es auf, bis wir es wieder benötigen. Wenn Sie sich mit 'SST' durch das Programm tasten, finden Sie diese Zeilen vor:

```
01 ENTER↑          09 X≤Y?
02 "⁻?----■"       10 Σ+
03 CLD             11 Σ-
04 -               12 HMS+
05 *               13 HMS-
06 /               14 MOD
07 X<Y?            15 %
08 X>Y?
```

Zeile 03 ist aus dem Anhangssymbol (Byte 7F) hervorgegangen, die Zeilen 04 bis 15 stammen von den Zeichen A bis L her. Sie finden diese Buchstaben alle in der Zeile 4 der Byte-Tabelle. Springen Sie nun mit 'GTO .004' auf Zeile 04, und löschen Sie mit 'DEL 012' alle Zeilen von 04 bis 15. Wir werden alle 12 Zeichen durch synthetische Zeichen ersetzen. Wir brauchen dazu nur die entsprechenden Befehle einzutasten. Wählen Sie die folgenden Befehle aus:

Befehl:	Zeichen:
04 -	A
05∗LBL 00	⌐
06∗LBL 11	⌡
07 RCL 02	"
08 RCL 08	(
09 RCL 09	)
10 STO 11	, (Semikolon)
11 ASIN	/
12 DEC	
13 CLD	¯
14 1/X	⊢
15 +	@

'PACK'en Sie sicherheitshalber noch, damit Sie auch gewiß sein können, daß sich keine 'NULL'-Bytes mehr an unpassender Stelle herumtreiben. Löschen Sie anschließend Zeile 04, um genau ein 'NULL'-Byte zwischen 'CLD' und 'LBL 00' zu erzeugen, dann 'GTO .001', BS und Korrekturtaste drücken. Sie sollten als Ergebnis dies erblicken:

```
01 ENTER↑
02 STO 15
03 "⊢¯⌐⌡"();\_⊢'@"
```

Die LBL-Eingaben für dieses Beispiel lauten 253, 127, 0, 1, 12, 34, 40, 41, 59, 92, 95, 127, 96 und 64.

Legen Sie "ABC" ins Alpha-Register, und führen Sie Zeile 03 aus. Anschließend enthält das Alpha-Register "ABC¯⌐⌡"();_⊢'@". Wenn Sie das Alpha-Register löschen ('CLA') und dann noch einmal Zeile 03 ausführen, enthält das Alpha-Register überraschenderweise nur "⌐⌡"();_⊢'@". Die 'NULL', angezeigt als Überstreichungszeichen "¯", ist verschwunden. Die Regel für die Anzeige von 'NULL'en sind im Alpha-Register nur dann sichtbar, wenn sie nicht führende, sondern innere oder nachfolgende Zeichen sind.

Wenn Sie 'ASTO X' ausführen, werden im X-Register auch noch die inneren und nachfolgenden 'NULL'en unsichtbar. Es wird aber nur ihre Anzeige unterdrückt, sie sind nichtsdestoweniger nach wie vor vorhanden, was sich leicht mit 'X = Y?' überprüfen läßt: die Anwort lautet "NO", wenn z.B. Register X eine unsichtbare 'NULL' enthält, Register Y aber nicht, beide Register jedoch dieselbe Anzeige auslösen, also *scheinbar* denselben Inhalt haben. Dieses merkwürdige Verhalten ist nicht wichtig genug, um ihm ein ganzes Beispiel zu widmen. Sie sollten sich dennoch dessen bewußt sein, daß eine mit 'ASTO' gespeicherte Zeichen-Kette, die 'NULL'en enthält, diese Tatsache nicht durch Anzeige 'zugibt'. Falls Sie Zweifel haben, überprüfen Sie den Sachverhalt mit 'ARCL' und 'AVIEW'.

Besitzer eines Druckers wissen vielleicht, daß die Druckerfunktion 'BLDSPEC' dazu benutzt werden kann, jedes beliebige synthetische Anzeigesymbol zu erzeugen. Z.B. liefert die Befehlsfolge

```
01 ,            (Dezimalkomma)
02 X<>Y
03 BLDSPEC
04 PRX
```

das dem im X-Register vorhandenen Dezimalwert (zwischen 0 und 127) entsprechende Zeichen, wobei Anzeige und Druckbild bei einem Teil der Zeichen voneinander abweichen. Mit '38' in X erhält man auf 'GTO .001, R/S' hin das kaufmännische Und "&", ein synthetisches Zeichen, Byte-Tabelle Zeile 2, Spalte 6. '5, R/S' erzeugt in der Anzeige den 'Einarmigen' "ℑ" und auf dem Drucker den griechischen Buchstaben "β"; vgl. Zeile 0, Spalte 5. Ein großer Teil der 128 Standard-Drucksymbole entartet in der Anzeige zum Vollzeichen. Etwas dieser Art ist selbstverständlich zu erwarten, denn 14 Anzeigesegmente besitzen natürlich nicht die Darstellungsflexibilität, die eine 7 × 7 Matrix, wie sie der Drucker verwendet, hat.

Besitzern eines X-Funktionen-Moduls steht die leistungsfähige Funktion 'XTOA' zur Verfügung, die ebenfalls dazu verwendet werden kann, synthetische Anzeigesymbole zu erzeugen. 'XTOA' ist sozusagen eine äußerst schnelle Version der PPC ROM-Routine DC .Weisen Sie 'XTOA' (oder DC) einer Taste zu, und führen Sie 'CLA, 38, XTOA' aus. Wenn Sie anschließend in den ALPHA-Modus wechseln, sehen Sie das synthetische Zeichen "&". Verlassen Sie den ALPHA-Modus wieder, um '5, XTOA' auszuführen, und gehen Sie danach erneut zum ALPHA-Modus über: Sie sehen jetzt "&ℑ". Der 'Einarmige' ist dem "&" angefügt worden. Mit 'PRA' können Sie sich zum Vergleich das Druckbild dieses Zeichenpaares anschauen.

Besitzer eines Druckers werden es besonders zu schätzen wissen, daß die Erzeugung synthetischer Textzeilen, in denen zugleich Groß- und Kleinbuchstaben enthalten sind, möglich ist, weil sich dadurch viele Bytes sparen lassen. Betrachten Sie das normale Verfahren, den Text "Hewlett-Packard" auf dem Drucker auszugeben:

```
01 "H"
02 ACA
03 SF 13              (Übergang zu Kleinbuchstaben)
04 "EWLETT-"
05 ACA
06 CF 13              (Übergang zu Großbuchstaben)
07 "P"
08 ACA
09 SF 13              (Übergang zu Kleinbuchstaben)
10 "ACKARD"
11 ACA
12 PRBUF
13 CF 13              (zurück zu Großbuchstaben)
```

Die Byte-Bilanz dieser Mißgeburt von Programm lautet 37, verglichen mit 18 Bytes, die für die elegante synthetische Textzeile "Hewlett-Packard", gefolgt von einem 'PRA'-Befehl, benötigt werden. Überdies besetzt jede Modus-Änderung zwischen Groß- und Kleinbuchstaben auch noch eine kostbare Druckpufferzelle (ein Byte im Druckpuffer), die besser anderweitig verwertet werden kann. Eine ins Einzelne gehende Beschreibung dieser Dinge finden Sie auf Seite 19 des PPC CJ vom Juli 1980. Synthetische Textzeilen ersparen also sowohl Programmspeicher- als auch Druckpufferplatz. Natürlich erscheinen die meisten der in synthetischen Textzeilen enthaltenen Kleinbuchstaben (alle bis auf a, b, c, d und e) nur als Vollzeichen in der Anzeige, aber sie werden in Programm-Auflistungen richtig ausgegeben. Wenn Sie also die etwas liederliche 'SST'-Anzeige ertragen können, werden Sie erheblichen Byte-Gewinn daraus ziehen, synthetische Textzeilen zu verwenden, sobald Kleinbuchstaben, allein oder gemischt mit Großbuchstaben, gedruckt werden sollen. Außerdem ist es gut zu wissen, daß fertige Text-Befehle schneller ausgeführt werden als der umständliche Aufbau einer Textzeile.

Synthetische Text-Befehle lassen sich weit allgemeiner als zur bloßen Erzeugung synthetischer Anzeigesymbole verwenden. Sie geben uns ein Mittel an die Hand, schnell und einfach (fast völlig) beliebige Byte-Kombinationen unter Programm-Kontrolle in Register zu befördern. Das Programm zum Byte-Laden (Kapitel 3), das Tastenzuweisungsprogramm (Kapitel 4) und andere synthetische Programme benutzen synthetische Textzeilen, deren Zweck *nicht* die Erzeugung von Anzeigesymbolen oder Druckbildern ist, an vielen Stellen.

Wir wenden uns noch einmal dem ersten Beispiel dieses Abschnitts zu, um zu veranschaulichen, wie überlegen synthetische Textzeilen selbst der besten Alternative, der Verwendung des Befehls 'XTOA' aus dem X-Funktionen-Modul, sind.

Ziel: Erzeugung der Textzeile "HP'S #1"

Beste Methode: synthetischer Text-Befehl 01 "HP'S #1"
Byte-Verbrauch: 8, Ausführung: schnell

Nächstbeste Methode: Verwendung des Befehls 'XTOA' oder der PPC ROM-Routine DC

```
01 "HP"
02 39
03 XTOA    (oder XROM DC )
04 "+S "   (Leerstelle beachten)
05 35
06 XTOA    (oder XROM DC )
07 "+1"
```

Byte-Verbrauch: 18, Ausführung: langsamer.

Besitzer des Druckers, die 'BLDSPEC' dazu verwenden, Sonderzeichen nach eigenem Bedarf herzustellen, können Bytes sparen und die Programmausführung beschleunigen, wenn sie stattdessen synthetische Textbefehle dazu nehmen. Das normale Verfahren: 'Zahl, BLDSPEC, Zahl, BLDSPEC, ... , Zahl, BLDSPEC, ACSPEC' wird ersetzt durch eine sieben Zeichen lange Textzeile, gefolgt von 'RCL M, ACSPEC'. Der Befehl 'RCL M' wird in Abschnitt 2G besprochen. Einzelheiten über den Zusammenhang zwischen den 'BLDSPEC'-Zahlen und dem synthetischen 'TEXT 7'-Befehl findet man im Benutzerhandbuch des PPC ROMs unter BL, im PPC CJ vom Juni 1980 und in dem Buch 'Synthetische Programmierung auf dem HP-41C/CV' von W.C. Wickes.

In vielen Fällen mehr exotischer synthetischen Programmierung werden Text-Befehle, welche Bytes aus den Zeilen 9 bis F der Byte-Tabelle enthalten, benötigt. Diese Bytes gehören zu Mehrbyte-Befehlen. Die Byte-Schnapper-Technik, die Sie am Anfang dieses Kapitels kennengelernt haben, vermag im allgemeinen nicht, solche Bytes in Textzeilen zu bringen. Der bequemste Weg, dennoch Textzeilen zu erzeugen, die diese widerspenstigen Bytes beinhalten, ist die Anwendung des Programms LB, wie Sie im Kapitel 3 sehen werden. Doch *Vorsicht ist geboten!* Für Bytes aus den Zeilen 8 bis F der Byte-Tabelle gelten, sobald sie in synthetischen Zeilen enthalten sind, ganz besondere Regeln: in der Anzeige entarten sie alle zum Vollzeichen – das mag noch hingehen; beim Ausdruck mit 'PRA' werden sie gemäß der Byte-Tabelle gedruckt (z.B. "M" für Byte CD) – das ist befriedigend; aber beim Programm-*Auflisten* machen sie sich plötzlich aus dem Staube, ohne auch nur wenigstens eine Leerstelle als Hinweis auf ihre Existenz zu hinterlassen – ohne Zweifel eine Gemeinheit. Damit nicht genug: Einige aus dem Bereich, der auf der QRC mit Farben unterlegt ist, verursachen scheinbar planlose Reaktionen des Druckers (Einfügen von Leerstellen, Übergang zu Kleinbuchstaben u.ä. oder gar völliges Chaos). Falls solch unziemliches Verhalten Ihre Programmausdrucke in Unordnung bringt, gehen Sie mit 'GTO' auf die nächste Programmzeile, und 'LIST'en Sie den Rest auf. Nebenbei bemerkt gibt Ihnen eine Auflistung im 'NORM'al-Modus im allgemeinen einen bescheidenen Hinweis auf die Gegenwart unsichtbarer synthetisch erzeugten Zeichen, insoweit nämlich, als die entsprechende Textzeile, die solche Zeichen enthält, beim Ausdruck gewöhnlich eingerückt oder herausgeschoben wird. Wenn Sie mehr darüber wissen wollen, ziehen Sie das PPC CJ vom Juli 1980 zu Rate; dort werden die Drucker-Kontrollbytes ausführlich und verständlich beschrieben.

2F. Der 'TEXT 0'-Befehl

Der HP-41 läßt Text-Befehle von bis zu 15 Zeichen bzw. Textanhangsbefehle von bis zu 14 Zeichen zu. Das erste Byte eines Text-Befehls stammt aus der Zeile F der Byte-Tabelle, wobei die Spaltennummer mit der Anzahl der im Text-Befehl enthaltenen Zeichen übereinstimmt.

Wie aber steht es mit dem Byte in Spalte 0? Logischerweise müßte ein F0-Befehl einer Textzeile der Länge 0 gleichkommen. Man könnte erwarten, daß ein 'TEXT 0'-Befehl die Wirkung eines 'CLA' hat. Das läßt sich herausfinden. Tasten Sie

01 "ABC"	LB -Eingabe:	MK -Eingaben:
02 STO IND T	240	240, 240

ein; 'GTO .001', BS, Korrekturtaste. Das 'STO' ist nun entfernt worden; folglich übernimmt 'IND T' die Rolle eines 'TEXT 0'-Befehls. In der Anzeige erscheint dieser Befehl als einsames Textsymbol "ᵀ" ohne nachfolgende Zeichen. Auf dem Drucker ergibt sich die Programmzeile 02 "" (Anführungszeichen ohne 'Angeführtes'). Wenn Sie das Programm ablaufen lassen und anschließend in den ALPHA-Modus schalten, gibt es eine kleine Überraschung: "ABC", durch Zeile 01 ins Alpha-Register gesetzt, bleibt unversehrt darin zurück. Der 'TEXT 0'-Befehl hat also nicht die Wirkung eines 'CLA'. Wenn Sie weiterexperimentieren, werden Sie zu der Erkenntnis gelangen, daß 'TEXT 0' ein Befehl *ohne jede Wirkung* ist. Kein Register wird behelligt, das Flag-Register eingeschlossen. Lediglich eine u.U. erfolgte Sperre der Stapelbewegung wird vermöge 'TEXT 0' – wie bei fast allen anderen Befehlen des HP-41 – aufgehoben. Bekanntlich wird die Stapelbewegung nur durch 'ENTER↑', 'CLX', 'Σ+' und 'Σ−' blockiert, eine Regel, die somit auch durch 'TEXT 0' nicht außer Kraft gesetzt wird. Tröstlich.

Wofür ist ein so ohnmächtiger Befehl wie 'TEXT 0' überhaupt zu gebrauchen? Was kann man mit einem wirkungslosen Befehl anfangen? Nehmen Sie an, Sie möchten eine unbekannte ganze Zahl im Y-Register inkrementieren, ohne die anderen Stapelinhalte anzutasten. 'ISG Y' bewirkt das Erwünschte, doch anschließend wird eine Programmzeile übersprungen, wenn der Inhalt von Y nicht-negativ war. Daher brauchen wir einen Befehl, der dem 'ISG Y' folgt und den Zustand des Rechners nicht beeinflußt, gleichgültig ob er ausgeführt wird oder nicht. 'TEXT 0' ist genau solch ein Befehl. Überdies ist er der einzige *Ein*-Byte-Befehl des HP-41 mit dieser Eigenschaft ('STO X' z.B. leistet zwar dasselbe, erfordert aber zwei Bytes). Befehle, die nichts bewirken, heißen 'NOP'-Befehle (für *No Op*eration). 'NOP'-Befehle gibt es im HP-25, HP-33, HP-55 und anderen Rechnern. Synthetische Techniken haben uns diesen Befehl nunmehr auch auf dem HP-41 beschert. Befehlspaare wie

```
01 ISG X
02 TEXT 0
```

tauchen in vielen synthetischen Programmen auf. Man kann sie überall dort einsetzen, wo ein 'Inkrementieren ohne Sprungbedingung' gewünscht wird. Natürlich kann man ein 'TEXT

0' ebenso einem 'DSE'-Befehl folgen lassen, um zu dekrementieren, ohne anschließend von Sprungbedingungen abhängig zu sein.

2G. Verwendung des Alpha-Registers zur Datenspeicherung

Wir haben gesehen, daß je Zeichen eines Text-Befehls ein Byte des Programmspeichers benötigt wird. Wir dürfen daraus schließen, daß das 24 Zeichen umfassende Alpha-Register 24 Bytes im Speicher besetzt, und zwar außerhalb jenes Bereiches, der für den Benutzer den Programmspeicher bildet. 24 Bytes, das entspricht 3 Registern + 3 Bytes (24 = 3·7+3). Diese Register bilden zusammen mit den Stapelregistern, dem Flag-Register sowie weiteren Registern einen Speicherbereich, den man die Notizregister, Systemregister oder Zustandsregister nennt. Es sind insgesamt 16 Stück. Der Kartenleser-Befehl 'WSTS' zeichnet den Inhalt dieser 16 Register auf der Spur 1 einer Karte auf.

Auf das Flag-Register und den Adreßzeiger können wir mit synthetischen Befehlen direkt zugreifen. Möglicherweise besteht solch eine Zugriffsmöglichkeit auch auf die drei Register, welche Teil des Alpha-Registers sind. Die Nachsilben für das Flag-Register und das Register mit dem Adreßzeiger findet man in Zeile 7, Spalte E bzw. C der Byte-Tabelle. Sie haben wahrscheinlich schon vermutet, daß die anderen Nachsilben der Zeile 7 zu den anderen Systemregistern gehören. Doch bevor Sie mit Ihren Untersuchungen beginnen, ist eine Warnung angebracht. Sie können zwar gefahrlos den Inhalt jedes Zustandsregisters mit 'RCL' abrufen (die in Abschnitt 2C erwähnte 'Normalisierung' von Daten findet bei Systemregistern nicht statt); doch ändern Sie den Inhalt dieser Register solange nicht mit 'STO' ab, wie Sie nicht genau wissen, was Sie tun, es sei denn, Sie sehen dem Schlimmsten gefaßt entgegen: *wenn Sie* z.B. *Register c* – risikofreudig – *löschen, gibt es "MEMORY LOST"*!

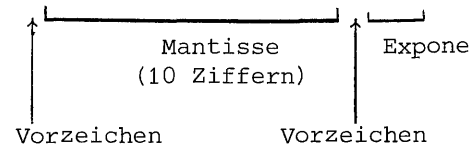
Das Alpha-Register besetzt die Systemregister M, N, O und zusätzlich einen Teil (3 Bytes) von P. Solange oder sobald sich im Alpha-Register nichts für Sie Erhaltenswertes befindet, dürfen Sie die Register M, N und O so frei benutzen wie numerierte Datenregister. Nach Ihrem gegenwärtigen Wissensstand müßten Sie in der Lage sein, das folgende Programm zu edieren:

```
01*LBL "RSHF"
02 CLX
03 X<> ]        }
04 X<> \        } Druckbilder von Register { M O
05 X<> [        }                          { N N
                                           { O M
```

Sollten Sie Hilfe benötigen, bedienen Sie sich der Anweisungen am Schluß dieses Abschnitts.

Wir wollen uns zunächst dem Befehl 'X<>M' zuwenden. Geben Sie (im RUN-Modus) '1.274065002 E−40' ein, dann 'CLA', 'GTO .005' und 'SST'. Wenn Sie nun in den Alpha-Modus schalten, sehen Sie "Ж'@e ¯)ᵀ". Was ist passiert? Wir befragen die QRC, um die 7 Bytes, die für die Anzeige dieser Zeichenkette verantwortlich sind, zu ermitteln. Nachstehend das Ergebnis unserer Bemühung in Tabellenform.

Byte in Hexadezimaldarstellung	01	27	40	65	00	29	60
Byte in Zeichendarstellung	↑	'	@	e	-	)	⊤
Zahlendarstellung in numerischem Register	+1.	27	40	65	00	2E-	40



Die vierzehn Hexadezimalziffern, welche in den sieben Bytes enthalten sind, lauten '01274065002960'. In der zweiten bis elften dieser 14 Ziffern erkennt man sofort die zehn signifikanten Ziffern des ursprünglichen Inhalts von X wieder. Die erste der 14 Ziffern bestimmt das Vorzeichen; sie ist 0 für positive, 9 für negative Zahlen und 1 für Alpha-Ketten. Die letzten drei Ziffern stellen den Exponenten und sein Vorzeichen dar. Die zwölfte ist 0 für positive und 9 für negative Exponenten. Die letzten zwei Ziffern bilden den Exponenten selbst. Sie lauten wie der Exponent, wenn er positiv ist. Sobald er negativ ist, dient das Komplement zu 100 zu seiner Darstellung. In unserem Beispiel lautet der Exponent −40, folglich heißen die letzten beiden Ziffern 60 = 100−40. Sie können sich folgende einfache Regel merken: Man addiere 1000 auf den vorzeichenbehafteten Exponenten und beschränke sich auf die letzten drei Ziffern des Ergebnisses, dann hat man genau die im HP-41 verwendete Zifferndarstellung für den vollständigen, also mit Vorzeichen versehenen Exponenten; in unserem Fall 1000−40 = 960.

Tasten wir erneut 'GTO .005' und 'SST', um 'X< >M' auszuführen, kehrt die Zahl '1.274065002 E−40' ins X-Register zurück, und das Alpha-Register ist wieder leer. Versuchen wir ein weiteres Beispiel. Führen Sie, die gewählte Zahl noch in X liegend, abermals 'X< >M' aus, gehen Sie in den Alpha-Modus, und ersetzen Sie das Zeichen "⊤" durch "A" ('APPEND', Korrekturtaste und 'A'). Jetzt steht " ↑'@e-)A " im Alpha-Register. Nun wieder Alpha-Modus ausschalten und erneut 'X< >M' ausführen. Ergebnis: '1.274065002 E−59'. Weil das Zeichen "A" den Hexadezimalwert 41 hat, entstehen die Exponentenziffern 941 = 1000−59, das entspricht 'E−59'.

Untersuchen Sie nach eigenem Belieben die Äquivalenz von Zahlen und sieben Zeichen langen Alpha-Ketten mit dem Befehl 'X< >M'. Die meisten Zahlen bringen dabei Vollzeichen ins Alpha-Register. Beachten Sie auch, daß Alpha-Ketten, die durch 'X< >M' ins X-Register gelangen, ein sehr merkwürdiges Erscheinungsbild zeigen können, sobald die beiden Vorzeichen-Ziffern nicht 0 oder 9 lauten oder Hexadezimalziffern A bis F an den Zeichen im Alpha-Register beteiligt sind.

Wenn Sie Register M als Notizregister für die Ablage numerischer Daten benutzen, werden Sie sich im allgemeinen nicht darum kümmern, daß Zahlen Zeichen-Ketten ähnlich sehen können, doch läßt sich die Äquivalenz zwischen Zeichen und Zahlen in einigen Fällen fortgeschrittener synthetischen Programmierung durchaus vorteilhaft ausnutzen. Wenn Sie z.B. in einem Programm den Zahleneintrag '1.274065002 E−40' tätigen wollen, können Sie 5 Bytes des Programmspeichers sparen, sofern Sie stattdessen die Textzeile " ↑'@e-)⊤ ", gefolgt von 'RCL M', verwenden.

Die Befehle 'X< >N' und 'X< >O', arbeiten genauso wie 'X< >M' mit dem Unterschied, daß 'X< >M' auf die 7 äußersten rechten Bytes des Alpha-Registers zugreift bzw. sie belegt, 'X< >N' und 'X< >O' dagegen auf die jeweils nächsten 7 Bytes, gerechnet von rechts nach links. Abbildung 2.2 veranschaulicht die Lage der Register:

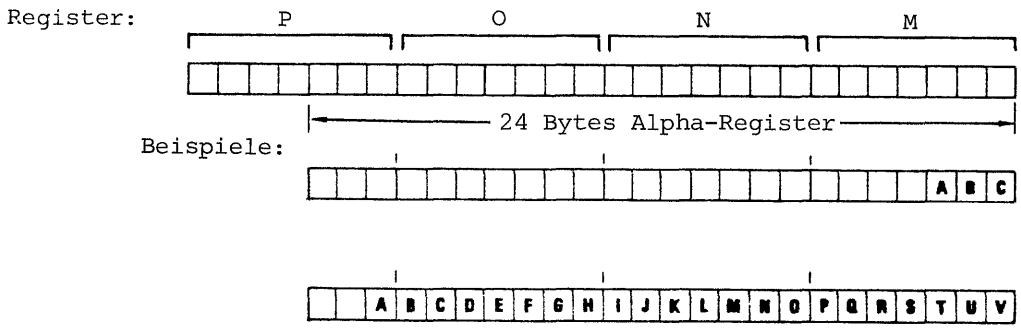


Abbildung 2.2. Das Alpha-Register

Zeichen-Ketten der Länge 1 bis 24 werden stets *rechtsbündig* abgelegt. Führende nicht belegte Bytes lauten 'NULL' (Byte 00) und sind unsichtbar.

Vielleicht sollten Sie das folgende kurze Beispiel nachvollziehen. Laden Sie "ABCDEFGHIJKLMNOPQRSTUV" ins Alpha-Register. Führen Sie 'CLX' und 'X< >O' ('GTO .002, SST, SST') aus. Das Alpha-Register enthält anschließend "A⁻⁻⁻⁻⁻⁻⁻IJKLMNOPQRSTUV". Die sieben Bytes in Register O (vgl. Abb. 2.2) sind durch Überstreichungszeichen ('NULL'-Bytes: Zeile 0, Spalte 0) ersetzt worden. Das Register O enthält jetzt die Zahl 0. Nach einem 'X< >N' geht der Alpha-Inhalt in "A⁻⁻⁻⁻⁻⁻⁻BCDEFGHPQRSTUV" über. Wenn Sie schließlich noch einmal 'X< >O' ausführen, erhalten Sie 'AIJKLMNOBCDEFGHPQRSTUV". Die Leistung der Befehle 'STO', 'RCL' und 'X< >' beschränkt sich also nicht allein auf die Speicherung und den Austausch von numerischen Daten, vielmehr können diese Befehle in Verbindung mit den Registern M, N und O zusätzlich dazu verwendet werden, Zeichen-Ketten im Alpha-Register aufzuspalten und neu zusammenzusetzen. Diese Möglichkeit, Zeichen-Ketten zu behandeln, wird in der synthetischen Programmierung weidlich dazu genutzt, Bytes in Entzifferungsprogrammen zu isolieren oder bestimmte Zeichen in Ketten durch andere Zeichen zu ersetzen.

Eine leicht zu verfolgende Anwendung der neuen Befehle auf Alpha-Ketten ist die Verschiebung des Alpha-Inhalts um 7 Zeichen nach rechts. Das Programm "RSHF" vollbringt solch eine Verschiebung für Ketten bis zur Länge 21: die 7 äußersten rechten Zeichen verschwinden aus dem Alpha-Register. Z.B. ergibt 'XEQ "RSHF"', angewendet auf "ABCDEFGHIJKLMNOP", das Ergebnis "ABCDEFGHI". Mit 'SST' im Alpha-Modus kann man verfolgen, wie "RSHF" arbeitet.

Wir wollen uns nun näher ansehen, wie nützlich der Zugriff auf die Systemregister M, N und O beim Rechnen mit Zahlen sein kann. Drei Register 'in der Rückhand' haben, kann häufig von Vorteil sein, wenn man Datenregisterinhalte unversehrt lassen muß. Beispielsweise können Sie viele Unterprogramme nunmehr so abfassen, daß numerierte Datenregister nicht mehr betroffen sind. Derart gestaltete Unterprogramme sind dann mit jedem Programm, das nur auf numerierte Datenregister zugreift, verträglich. Z.B. benutzen viele der Routinen des PPC ROMs keine numerierten Datenregister, so daß Ihre Programme numerierte Datenregister ohne Einschränkung verwenden und dennoch die Routinen gefahrlos aufrufen können. Natürlich sollten Sie bei einem solchen Programmierstil grundsätzlich berücksichtigen, daß nach einem Unterprogrammaufruf die Inhalte von M, N und O im allgemeinen verändert sind.

Sehr kurze Unterprogramme läßt man häufig Teile des Alpha-Registers benutzen, um den Gebrauch sowohl des Stapels als auch numerischer Datenregister zu umgehen. Im Idealfall arbeitet eine solche Routine so wie die festverdrahteten Funktionen des HP-41: L enthält nach Ausführung den Eingabewert aus X, T bleibt ungeschoren, und das Ergebnis steht in X. Als Beispiel dafür nehmen wir eine Routine namens "CNK", die eine Formel aus der Kombinatorik auswertet:

$$C(n,k) = \frac{n!}{k!\,(n-k)!} = \frac{(n-k+1)\,(n-k+2)\cdots n}{k\,(k-1)\cdots 1}$$

C(n,k) ist die Anzahl der Kombinationen von n Elementen zur k-ten Klasse. Diese Routine nimmt die Werte n und k aus den Registern Y bzw. X und liefert das Ergebnis in X ab. Die ursprünglichen Inhalte von Z und T liegen am Schluß in Y und Z, k verbleibt in L, und n wird sogar nach T gerettet. C(n,k) arbeitet mit Bezug auf die Stapelinhalte also so ähnlich wie beispielsweise 'Y↑X', nur daß der ursprüngliche Inhalt von Y überdies noch nach T gelangt.

Weil die Berechnung von C(n,k) verhältnismäßig verwickelt ist, kann "CNK" die Inhalte von Z und T nicht bewahren, ohne ein Notizregister hinzuzuziehen. Wir werden dafür Register M nehmen. Auf diese Weise wird "CNK" mit jedem aufrufenden Programm, das nur numerierte Datenregister benutzt, verträglich. Nachfolgend die Auflistung von "CNK" zum Eintasten und Ausprobieren:

```
01*LBL "CNK"    LB / MK-Eingaben:
02 -
03  E           27/27, 0
04 STO [        145, 117
05 RDN
06 LASTX
07 X>Y?
08 X<>Y
09*LBL 01
10 X<>Y
11 ISG X
12 "" *)        240/240, 240
13 ST* [        148, 117
14 X<>Y
15 ST/ [        149, 117
16 DSE X
17 GTO 01
18 X<>Y
19 RDN
20 X<> [        206, 117
21 END
```

*) Ausdruck von 'TEXT 0'

Zur Herstellung der synthetischen Zeilen 03, 04, 12, 13, 15 und 20 bedienen Sie sich der Befehle 'STO 27', 'STO IND 17, RDN', 'STO IND T', 'STO IND 20, RDN', 'STO IND 21, RDN', 'STO IND 78, RDN' und schnappen jedem der 6 'STO'-Befehle die Vorsilbe mit dem Byte-Schnapper weg.

Testen Sie "CNK" mit '49, ENTER↑, 6, XEQ "CNK"'. Sie erhalten '13.983.816', das ist die Anzahl der Möglichkeiten im Deutschen Zahlen-Lotto, '6 aus 49'. Natürlich liefert '49, ENTER↑, 43, XEQ "CNK"', also '43 aus 49', dasselbe Ergebnis.

Wir wollen nun die Arbeitsweise von "CNK" genau analysieren. Beim Start enthält X den Wert 'k' und Y den Wert 'n'. "CNK" bringt in Zeile 04 den Anfangswert '1' ins Register M, so daß die Befehle 'ST* M' und 'ST/ M' in der mit 'LBL 01' angeführten Schleife auch beim ersten Durchgang fehlerfrei arbeiten. Nach der Ausführung von Zeile 06 enthält M den Wert '1', X den Wert 'k' und Y die Differenz 'n−k'. Die Zeilen 07 und 08 vertauschen 'k' und 'n−k', falls 'n−k' kleiner als 'k' ist, so daß der kleinere der beiden Werte auf jeden Fall nach X gelangt. Auf diese Weise macht die Routine Gebrauch von der Beziehung $C(n,k) = C(n,n-k)$ mit dem Ziel, die Ausführungszeit so weit wie möglich zu drücken. Die Schleife inkrementiert 'n−k' und multipliziert den Inhalt von M in M mit diesem Ergebnis. In Zeile 14 wird 'k' nach X zurückgebracht, dann der Inhalt von M in M durch 'k' dividiert und schließlich 'k' dekrementiert. An dieser Stelle enthält X den Wert 'k−1', Y den Wert 'n−k+1' und M den Wert '(n−k+1)/k', also den ersten Faktor aus der Formel für C(n,k). Die Voraussetzungen zum zweiten Durchlauf der Schleife sind somit gegeben. Die Schleife wird insgesamt k-mal durchlaufen. Danach liegt in X der Wert '0' und in Y der Wert 'n'. Die Zeilen 18−20 legen 'n' nach T und das Ergebnis nach X, wobei gleichzeitig M gelöscht wird.

Wenn Sie in den Zeilen 04, 13, 15 und 20 statt des Registers M das Register O benutzen, bleiben sogar Alpha-Ketten von bis zu 14 Zeichen unbehelligt, weil die Inhalte von M und N nicht von Rechenbefehlen angesprochen werden und die führenden 'NULL'-Bytes, die durch 'X<>O' (Zeile 20) zum Schluß ins Register O kommen, unsichtbar sind.

Hier noch die oben versprochenen Einzelschritt-Anweisungen zur Erzeugung der im Programm "RSHF" enthaltenen synthetischen Zugriffe auf die Teile des Alpha-Registers:

```
01♦LBL "RSHF"          LB / MK -Eingaben:
02 CLX
03 STO IND 72          X<>0 = 206, 119
04 CLX
05 STO IND 78          X<>N = 206, 118
06 LASTX
07 STO IND 76          X<>M = 206, 117
08 RDN
```

'GTO .006', BS, Korrekturtaste, 'GTO .004', BS, Korrekturtaste, 'GTO .002', BS und Korrekturtaste.

2H. Die Verwendung anderer Zustandsregister zur Datenspeicherung

Die Zustandsregister P, Q und a können unter einschränkenden Bedingungen ebenfalls zur Datenspeicherung benutzt werden. Einzelheiten darüber, wie das Betriebssystem des HP-41 diese Register verwendet, finden Sie im Abschnitt 6A dieses Buches, auf S. 19 des PPC CJ vom September 79 und in dem Buch 'Synthetische Programmierung auf dem HP-41 C/CV' von W.C. Wickes. Hier geben wir nur eine kurze Zusammenfassung.

Das Register P (in Programm-Auflistungen erscheint es mit dem Symbol ↑) kann zur Datenspeicherung verwendet werden, solange keine Zahleneinträge durch das Programm ausgeführt werden und kein 'VIEW'-Befehl, der numerische Daten anzeigen soll, auftritt.

Das Register Q (in Programmauflistungen erscheint es mit dem Symbol _) kann ebenfalls zur Datenablage benutzt werden, doch kann man sich der Bewahrung des Inhalts nicht sehr sicher sein, weil es eine Fülle interner Prozesse gibt, die ihn zerstören können. Wenn z.B. ein 'GTO'- oder ein 'XEQ'-Befehl, der eine globale Marke (aus den Katalogen 1 oder 2) anspricht, ausgeführt wird, geht der Inhalt von Q verloren. Dies gilt nicht für entsprechende 'LBL'-Befehle. 'XROM'-Befehle, die, wie wir im nächsten Kapitel noch sehen werden, eine andere Struktur als globale 'XEQ'-Befehle haben, können ebenfalls unbedenklich verwendet werden. Doch gilt wiederum: *jedes 'Buchstabieren'* von 'GTO'-, 'XEQ'- oder 'LBL'-Befehlen über das Tastenfeld unter Ein- und Ausschalten des ALPHA-Modus verändert den Inhalt von Q. Andere Befehle, die Q nicht ungeschoren lassen, sind: Zifferneinträge, 'SIN', 'COS', 'P–R', 'R–P', 'Y↑X', 'SDEV' und jeder Befehl, der eine Alpha-Register-Anzeige veranlaßt ('AVIEW', 'PROMPT' oder 'PSE' mit 'AON'). Außerdem wird das Register Q ausgiebig vom Drucker 82143A zum Informationsaustausch mit dem HP-41 benutzt. Sie sollten daher in Q keine Daten ablegen, wenn Ihre Programme mit angeschlossenem Drucker laufen müssen.

Das Zustandsregister a kann solange als Datenregister verwendet werden, als die Unterprogrammstufe 3 nicht erreicht wird. Das bedeutet, daß ein Programm, welches a anspricht und selbst keine 'XEQ'-Befehle enthält, auf höchstens erster Unterprogrammstufe aufgerufen werden darf. Sobald es auf zweiter oder gar höherer Stufe aufgerufen wird, kann das 'END' (oder ein 'RTN') im Hauptprogramm seine Wirkung verlieren: das Programm hält u.U. überhaupt nicht mehr an. Wenn a nämlich nicht leer (d.h. mit Inhalt 0) zurückgelassen wird, bevor die Rücksprünge beginnen, startet der Rechner eine durch 'END' (oder 'RTN') ausgelöste Suche nach einer Adresse, die teilweise durch den bei Datenablage entstandenen Inhalt von a bestimmt wird. Beachten Sie in diesem Zusammenhang, daß 'XEQ' oder 'RTN' den Inhalt von a durch Verschiebung um zwei Bytes – das ist der Platzbedarf für eine Rücksprungadresse – verändert. Führen Sie auch nicht die X-Funktion 'PSIZE' aus, wenn der Inhalt von a unversehrt bleiben soll, denn in diesem Fall behandelt der Rechner den Inhalt von a als Satz von Rücksprungadressen, die dem neuen 'SIZE' angeglichen werden müssen. All dies wird in Kapitel 6 genauer besprochen.

Aufgaben

2.1 Stellen Sie unter Verwendung des normalen 'TONE 8' und des synthetischen 'TONE P' eine Befehlsfolge her, die die Morsezeichen für 'CQ' (dah–di–dah–dit, dah–dah–di–dah) ertönen läßt. (Lösung: S.109)

2.2 Stellen Sie mit dem Byte-Schnapper den synthetischen Befehl '−E1' her. Hinweis: erst 'E1' erzeugen. (Lösung: S.109)

2.3 Schreiben Sie unter Verwendung von 'RCL d', 'STO d' eine kurze Routine, mit der Sie alle 10 Ziffern des X-Register-Inhaltes betrachten können, ohne den Anzeige-Modus zu verändern. Hinweis: Ergänzen Sie die nachstehende Routine so, daß der ursprüngliche Anzeige-Modus wiedergewonnen wird.

```
01*LBL "VX"
02 "  "          (2 Leerstellen)
03 SCI 9
04 ARCL X
05 AVIEW
06 END
```

(Lösung: S.110)

2.4 Berechnen Sie den 'Goldenen Schnitt' $x = 1+1/x$ in einer Schleife, die die aufeinanderfolgenden Approximationen anzeigt und mit 'RCL b', 'STO b' arbeitet, um auf den Schleifenanfang zurückzuspringen. (Lösung: S.110)

2.5 a) Stellen Sie eine Befehlsfolge her, die synthetische Text-Befehle enthält, mit deren Hilfe 'PROMPT' zur Aufforderung "X(n)=?" führt. Hierin soll 'n' eine ganze Zahl aus R_{00} sein.

b) Ändern Sie die Befehlsfolge so, daß der Anzeige-Modus erhalten bleibt. (Lösung: S.110/111)

2.6 Stellen Sie eine Befehlsfolge her, die zur Ausgabe "OUT=xμV" führt, ohne den Anzeige-Modus zu verändern. Hierbei soll x durch 'ARCL' im 'FIX 2'-Format eingefügt werden. (Lösung: S.112)

2.7 Stellen Sie eine vollständige 'MOD'-Funktion her, die ebenso wie eine fest verdrahtete Funktion arbeitet. Die Registerinhalte von Z und T sollen erhalten bleiben, in L soll x, in Y soll y mod x, und in X soll (y − y mod x)/x zurückbleiben. Als Notizregister werden Sie einen Teil des Alpha-Registers, etwa M, verwenden müssen. (Lösung: S.112)

2.8 Benutzen Sie den Byte-Schnapper, um einen Zwei-Byte-Befehl mit dem Hexadezimalwert F1 F0 (eine Textzeile der Länge 1, deren Textzeichen den Hexadezimalwert F0 hat) herzustellen. (Lösung: S.112)

KAPITEL 3

Bytes laden

Wenn Sie – unter Verwendung des Byte-Schnappers – die Beispiele von Kapitel 2 nachvollzogen haben, wird Ihnen nicht entgangen sein, daß der Byte-Schnapper ein sehr leistungsfähiges Werkzeug zum Erzeugen der verschiedensten Arten synthetischer Befehle ist. Sollte man jedoch sehr viele solcher Befehle in einem Zug erzeugen müssen, könnte man gehalten sein, nach einer schnelleren Methode Ausschau zu halten. In der Tat gibt es die, nämlich Dienstprograme, genannt Byte-Lader, mit deren Hilfe beliebige Befehle erzeugt und umittelbar in den Programmspeicher geladen werden können. Man braucht dazu nur die der gewünschten Byte-Folge entsprechenden Dezimalwerte (zwischen 0 und 255) auszuwählen und als Eingaben für einen solchen Byte-Lader zu verwenden.

Die in Byte-Ladern verwirklichten Ideen sind im Benutzerhandbuch des PPC ROMs unter LB und im PPC CJ in der Ausgabe vom Dezember 1980 beschrieben. Die Pionierleistungen für die Byte-Lader wurden von vielen PPC-Mitgliedern, darunter William Cheeseman, Roger Hill, John McGechie, William Wickes und der Autor selbst, erbracht. In diesem Buch beschränken wir uns darauf, die *Anwendung* von Byte-Ladern zu besprechen.

Wir beziehen uns im folgenden auf drei verschiedene Programme der genannten Art. Das erste heißt "LB" (*l*oad *b*ytes); zu seiner Anwendung genügt der bloße HP-41. Es wurde von Clifford Stern geschrieben, belegt 214 Bytes und paßt auf eine Magnetkarte.

Beim zweiten handelt es sich um das Program LB aus dem PPC ROM, einen erstklassigen Byte-Lader von Roger Hill. Wenn Sie einen PPC ROM besitzen, machen Sie sich mit der Gebrauchsanweisung für LB vertraut; sie ist der für "LB" ähnlich, aber nicht mit ihr identisch.

Der dritte Byte-Lader heißt "LBX" und erfordert den X-Funktionen-Modul. Dieses Programm stammt ebenfalls von Clifford Stern. Es ist eine Spielart von "LB", die sich ausgiebig der Funktionen des X-Moduls, wie z.B. 'XTOA', bedient und daher kürzer und schneller ist. Wenn Sie sich für den Gebrauch von "LBX" entscheiden, schlagen Sie unter Aufgabe 3.5 nach; dort finden Sie die Programm-Auflistung.

Trotz seines geringen Umfanges leistet "LB" fast alles, was die ROM-Version LB bietet; nur entbehrliche Bequemlichkeiten wie Lösch-Meldungen und die Möglichkeit zu unterbrechen fehlen. Es konnten nicht alle Vorteile der ROM-Version übernommen werden, ohne das Programm unangemessen aufzublähen. In ROM Programmen dagegen braucht man nicht unbedingt mit Bytes zu geizen, weil der Arbeitsspeicher des HP-41 davon nicht betroffen ist. Wie auch immer, was "LB" an Annehmlichkeiten verweigert, bietet es an Geschwindigkeit. Wenn Sie den X-Funktionen-Modul zur Verfügung haben, sollten Sie unbedingt "LBX" (Aufgabe 3.5) verwenden, denn dieser Byte-Lader ist auch noch kürzer und schneller als "LB".

Sofern Sie Zugang zum optischen Lesestift des HP-41 haben, können Sie sich der Barcodes bedienen, die im Anhang E abgedruckt sind. Sie finden dort die Barcodes aller wichtigen

Routinen, die dieses Buch enthält – insbesondere die für "LB" und "LBX" –, so daß Sie die synthetischen Routinen schnell und fehlerfrei in den HP-41 bringen können. Vergessen Sie dabei nicht, vorher eine Schutzfolie aufzulegen, damit die Barcodes nicht beschädigt werden. Sollten Sie sich allerdings mehr Übung im Umgang mit dem Byte-Schnapper erwerben wollen, vergessen Sie die Barcodes zunächst.

Leser ohne PPC ROM und ohne X-Funktionen-Modul beginnen mit den folgenden Befehlen, um die für "LB" erforderlichen synthetischen Zeilen zu erzeugen:

```
01 ENTER↑              22 AVIEW
02 STO IND 16          23 STO IND 78
03 MEAN                24 RDN
04 STO IND 17          25 STO IND 17
05 RDN                 26 LASTX
06 STO IND L           27 STO IND 78
07 CLD                 28 LASTX
08 ENTER↑              29 STO IND 78
09 ENTER↑              30 SDEV
10♦LBL 01              31 STO IND 17
11 STO IND 78          32 SDEV
12 RDN                 33 STO IND Y
13 STO IND 78          34 CLD
14 AVIEW               35 ENTER↑
15 STO IND 78          36 STO IND 78
16 AVIEW               37 SDEV
17 STO IND 17          38 STO IND 16
18 RDN                 39 RDN
19 STO IND 78          40 STO IND 17
20 AVIEW               41 SDEV
21 STO IND 78
```

Schnappen Sie nun der Reihe nach die 'STO'-Bytes der Zeile 40, 38 und 36 weg (für Zeile 40 geht das so: 'GTO .039', BS, Korrekturtaste). Löschen Sie anschließend Zeile 35, *aber 'PACK'en Sie nicht*, bevor Sie damit beginnen, den nächsten Zeilen 33, 31, 29, 27, 25, 23, 21, 19, 17, 15, 13 und 11 der Reihe nach die 'STO'-Bytes zu entreißen. Danach werden die Zeilen 08 und 09 gelöscht und (wieder ohne zu 'PACK'en) die letzten 'STO'-Bytes, die der Zeilen 06, 04 und 02, beseitigt. Schließlich müssen Sie auch noch die Hilfszeile 01 tilgen und mit dem Dazwischentasten der fehlenden nicht-synthetischen Befehle die Edition von "LB" vervollständigen. Zeile 61 ist eine Textzeile, die ein einzelnes Leerzeichen enthält. Zeile 71 können Sie ebenfalls mit dem Byte-Schnapper herstellen, indem Sie einem '1 E4' die führende '1' wegnehmen (Abschnitt 2B). Wenn Sie darüber hinaus 'im rechten Geiste' synthetisch programmieren wollen, ersetzen Sie außerdem die Zahleneinträge der Zeilen 09 und 51 durch schnellere 'E'.

Falls Sie sich wegen vorhandener X-Funktionen sinnvollerweise für "LBX" entschieden haben, liefert die voranstehende Prozedur (bis auf einen) ebenfalls die notwendigen syntheti-

schen Befehle. Einige davon sind allerdings, wie Sie schnell feststellen werden, überflüssig. Der zusätzlich benötigte Befehl steht in Zeile 34 von "LBX". Er lautet 'STO N' und wird aus 'STO IND 17, LASTX' Byte-schnappend hergestellt.

Der Byte-Lader "LB" von Clifford Stern:

```
01♦LBL 01
02 CLST
03 BEEP
04 STOP
05 GTO "++"
06♦LBL "LB"
07 FS? 50
08 GTO 02
09 1
10 ENTER↑
11 ENTER↑
12 CLA
13 CF 21
14 AVIEW
15 -10
16 GTO "++"
17♦LBL 02
18 7
19 /
20 INT
21 FIX 0
22 CF 29
23 ARCL X
24 "├ REGS."
25 TONE 8
26 AVIEW
27 PSE
28 RCL b
29 STO [
30 "├♦♦%"
31 X<> [
32 X<> d
33 CF 04
34 CF 05
35 CF 06
36 FS?C 07
37 SF 05
38 FS?C 08
39 SF 06
40 FS?C 09
41 SF 07
42 FS?C 10
43 SF 09
44 FS?C 11
45 SF 10
46 FS?C 12
47 SF 11
48 X<> d
49 INT
50 DEC
51 1
52 +
53 ,1
54 %
55 +
56 +
57♦LBL 03
58 1,007
59 ENTER↑
60♦LBL 04
61 " "
62 ARCL Y
63 "├?"
64 AVIEW
65 STO [
66 RDN
67 STOP
68 FC?C 22
69 GTO 05
70 OCT
71  E4
72 +
73 X<> d
74 FS?C 19
75 SF 20
76 FS?C 18
77 SF 19
78 FS?C 17
79 SF 18
80 FS? 15
81 SF 17
82 FS? 14
83 SF 16
84 X<> d
85 X<> [
86 "├**"
87 STO \
88 ARCL Y
89 X<> \
90 ISG Y
91 GTO 04
92 SIGN
93 X<> c
94 LASTX
95 STO IND T
96 X<>Y
97 STO c
98 R↑
99 DSE X
100 GTO 03
101 GTO 01
102♦LBL 05
103 "├♦"
104 ISG X
105 GTO 05
106 X<> c
107 RCL [
108 STO IND Z
109 X<>Y
110 STO c
111 GTO 01
112 END
```

214 BYTES

Hinweise: Die Nachsilben M und N werden auf dem Drucker als '[' bzw. '\' ausgegeben. Zeile 30 lautet hexadezimal F4 7F 00 00 02, Zeile 103 lautet F2 7F 00.

Es ist zu empfehlen, das neu edierte Programm *sehr* sorgfältig an Hand der Programm-Auflistung zu überprüfen, bevor Sie es das erste Mal benutzen. Denn jedes synthetische Programm, welches das Zustandsregister c berührt, kann sehr leicht ein "MEMORY LOST" verursachen, wenn es fehlerbehaftet abläuft. Aus diesem Grunde ist es auch sinnvoll, das fertige Programm – selbst wenn es noch nicht ausgetestet ist – auf eine Magnetkarte zu schreiben, um nicht wegen eines kleinen Fehlers ganz von vorn beginnen zu müssen. Beachten Sie, daß einige synthetischen Zeilen anders angezeigt als ausgedruckt werden. Beispielsweise erscheinen die Zeilen 30 und 103 in der Anzeige als "⊢⁻⁻x" bzw. "⊢⁻". Befehle, die auf M und N zugreifen, werden mit '[' bzw. '\' aufgelistet. Die einander wechselseitig entsprechenden Symbole können der Zeile 7 der QRC entnommen werden. So findet man dort z.B., daß die Nachsilbe O auf dem Drucker ']' lautet.

Gebrauchsanweisung:
Die folgende Anleitung gilt für Clifford Stern's "LB". Die Anleitung für LB, die im Benutzerhandbuch des PPC ROMs enthalten ist, lautet in allen wesentlichen Punkten ganz ähnlich.

An der Stelle des Programmspeichers, an der Sie eine Anzahl synthetischer Befehle edieren wollen, müssen Sie zunächst diese Folge von Befehlen eintasten:

```
LBL "++"
+
+
+
.
.
.
+
+
+
XEQ "LB"
```

(Bei Verwendung des PPC ROMs verwandelt sich der letzte Befehl in 'XROM "LB"'.) Die Anzahl der '+'-Befehle muß die Anzahl der Bytes, die Sie erzeugen wollen, um 16 übertreffen. Wenn Sie die voranstehende Befehlsfolge nicht hintereinander eintasten, wenn Sie also z.B. zurücksetzen und dann noch einige '+' einfügen, müssen Sie 'PACK'en, bevor Sie fortfahren; es sei denn, Sie fügen nachträglich eine Anzahl von '+'-Zeilen ein, die ein Vielfaches von 7 ist. Warum ein solches 'PACK' notwendig werden kann, wird Ihnen klar sein, sobald Sie Kapitel 5 gelesen haben.

Wenn Sie "LB" häufig benutzen wollen, ist es zu empfehlen, die von 'LBL "++"' angeführte Befehlsfolge auf eine Magnetkarte aufzuzeichnen. Dafür wählt man am besten 99 '+'-Befehle

('XEQ "LB"' bildet dann die Zeile 101) und führt anschließend 'GTO ..' aus. Als Ergebnis hat man ein 'Programm' des Namens "+ +", welches genau auf eine Spur einer Magnetkarte paßt. Hat man einen X-Modul zur Verfügung, kann man "+ +" auch mit 'SAVEP' ins X-Memory bringen und mit 'GETP' oder 'GETSUB' bei Bedarf abrufen. Allerdings ist ein so verwahrtes "+ +", im Gegensatz zur Aufzeichnung auf Magnetkarte, nicht gegen "MEMORY LOST" immun.

Sie können jetzt den PRGM-Modus verlassen und 'XEQ "LB"' über das Tastenfeld ausführen. Befinden Sie sich noch auf der letzten Zeile der eingetasteten Befehlsfolge, genügt natürlich ein bloßes 'R/S'. "LB" teilt Ihnen als erstes mit, wieviele Register zur Aufnahme von Bytes zur Verfügung stehen; dann fordert es Sie zur Eingabe der einzelnen Bytes, deren je 7 ein Register füllen, auf. Die Anzahl der bereitgestellten Register lautet INT((p−10)/7), worin p die Anzahl der eingetasteten '+' bedeutet. In der folgenden Tabelle 3.1 haben Sie einen kurzen Auskunftgeber zur Hand, um die Anzahl benötigter '+'-Zeilen zu bestimmen.

Anzahl der '+'-Zeilen	Anzahl verfügbarer Register	Anzahl verfügbarer Bytes
0-16	0	0
17-23	1	7
24-30	2	14
31-37	3	21
(10+7n)-(16+7n)	n	7n

Tabelle 3.1. Anzahl der zum Einsatz von "LB" benötigten '+'-Befehle

Als Antwort auf die Aufforderung, ein Byte einzugeben, tasten Sie die dem Byte entsprechende Dezimalzahl (zwischen 0 und 255) ein und drücken 'R/S'. **WARNUNG:** Wenn Sie eine solche numerische Eingabe vor dem 'R/S' korrigieren müssen, ist zunächst ein 'RDN' nötig. Erst dann dürfen Sie den richtigen Wert eingeben. Andernfalls zerstören Sie die für den fehlerfreien Ablauf von "LB" erforderliche Anordnung der Stapelinhalte. Diese Warnung betrifft nicht den Gebrauch der ROM-Version LB.

Sind alle Bytes nach dem voranstehend beschriebenen Verfahren eingegeben, drücken Sie 'R/S' ohne numerische Eingabe. Dadurch wird der Vorgang des Ladens der Bytes beendet. Wenn die Gefahr des 'Überladens', also des Verlassens der bereitgestellten Register besteht, beendet "LB" den Ladevorgang automatisch.

Wir wollen uns nun ein Beispiel vornehmen. Angenommen, Sie wollen die Aufgabe 2.7 mit Hilfe von "LB" lösen. Die als Lösung auf S.112 angegebene Programm-Auflistung enthält die "LB"-Eingaben:

```
01*LBL "CMOD"          LB / MK -Eingaben:
02 X<>Y
03 STO [               145, 117
04 X<>Y
05 MOD
06 ST- [               147, 117
07 LASTX
08 ST/ [               149, 117
09 CLX
10 X<> [               206, 117
```

Die Dezimalwerte rechterhand werden dazu verwendet, die vier ausgewählten synthetischen Zwei-Byte-Befehle zu erzeugen.

Verfahren Sie wie oben beschrieben mit 'LBL "+ +"' – 24 '+'-Zeilen sind erforderlich –, und führen Sie 'XEQ "LB"' aus, indem Sie den PRGM-Modus verlassen und 'R/S' drücken. Sie erblicken zunächst die Meldung "2 REGS." und dann die Aufforderung "1?". Die Meldung "2 REGS." besagt, daß Sie für bis zu 14 Bytes (2 Register enthalten 14 Bytes) die Dezimalwerte eingeben können.

Die Antwort auf "1?" lautet '145, R/S'. Die nachfolgende Liste zeigt alle Aufforderungen und die zugehörigen Antworten:

Aufforderung	Antwort
1?	145, R/S
2?	117, R/S
3?	147, R/S
4?	117, R/S
5?	149, R/S
6?	117, R/S
7?	206, R/S
1?	117, R/S
2?	R/S

Die ersten sieben Eingaben bilden zusammen den Inhalt eines Registers, der in den Bereich der '+'-Zeilen von 'LBL "+ +"' gelegt wird. Die nächste Runde, also die Füllung des nächsten Registers beginnt wieder mit der Aufforderung "1?", nunmehr zur Eingabe des ersten Bytes des zweiten Registers einladend. Wenn Sie auf "2?" mit 'R/S' ohne Zahleneintrag antworten, wird der Ladevorgang abgeschlossen, indem das zweite Register mit 'NULL'en aufgefüllt und der so vervollständigte Registerinhalt ebenfalls im Bereich der '+'-Zeilen, und zwar im Anschluß an den Inhalt des ersten Registers, abgelegt wird. Sobald "LB" anhält, drücken Sie einmal 'SST' um nach 'LBL "+ +"' zu gelangen. Danach können Sie in den PRGM-Modus übergehen, um sich von der Existenz der soeben erzeugten synthetischen Befehle zu überzeugen. Was Ihnen

noch zu tun verbleibt ist einfach: die zurückgebliebenen '+'-Zeilen, 'LBL "++"' und 'XEQ "LB"' müssen getilgt und die nicht-synthetischen Zeilen dazwischengetastet werden.

Es ist unschwer zu erkennen, daß man mit "LB" umgehen kann, ohne tiefer in die synthetische Programmierung einzudringen. Es genügt zu wissen, welche Dezimalwerte herzunehmen sind, um die gewünschten synthetischen Befehle zu erlangen. In Kapitel 2 haben Sie durch die Vorführung des Gebrauchs der QRC einen großen Teil dieser Kenntnisse erworben. So sollten Sie jetzt in der Lage sein, mit Hilfe der QRC ohne Zögern die "LB"-Eingaben für z.B. den Befehl '– E1' zu ermitteln: 28, 27, 17.

Es gibt aber größere Bereiche der QRC, insbesondere die Zeilen A–E, deren Bedeutung bislang noch nicht erläutert wurde. Diese Bereiche werden teilweise sehr genau in den 'Corvallis Division Columns' des PPC CJ, Ausgaben Juli, August und September 1979, besprochen. Wir werden diese Bereiche jetzt kurz abhandeln und hier und da Hinweise auf Quellen und Stellen, an denen Sie weitere Auskünfte finden, einstreuen.

Es folgt eine Zusammenstellung der verschiedenen Befehlsarten und der Überlegungen, nach denen die zugehörigen Dezimalwerte bestimmt werden. In den meisten Fällen werden Sie die QRC befragen müssen. Zur Erinnerung: die Dezimalwerte stehen in der linken unteren Ecke des jeweiligen Byte-Kästchens; beispielsweise legt der Dezimalwert 126 entweder den Befehl 'AVIEW' oder die Nachsilbe d oder das Zeichen "Σ" fest, je nach Art des vorliegenden Befehls.

I. Ein-Byte-Befehle

Mit Ausnahme von 'TEXT 0', Dezimalwert 240, sind alle Befehle dieses Typs nicht-synthetisch. Jeder Dezimalwert der Zeilen 0 und 2 bis 8 ergibt eine nicht-synthetische Ein-Byte-Funktion, solange kein anderes Byte, das eine Nachsilbe erfordert, unmittelbar voransteht.

Aufeinanderfolgende Bytes der Zeile 1, Spalte 0 bis C, verschmelzen zu einem einzigen mehrziffrigen Zahleneintrag, solange sie nicht durch ein 'NULL'-Byte oder einen beliebigen anderen Befehl getrennt sind. Beispielsweise ergibt die Dezimalwertfolge 28, 27, 28, 19 den Zahleneintrag '– E–3'.

II. Zwei-Byte-Befehle

Zwei-Byte-Befehle haben eine Vorsilbe oder ein erstes Byte, die auf der QRC in dem gelb unterlegten Bereich zu finden sind. Es gibt 4 Klassen von Zwei-Byte-Befehlen.

Die erste Klasse besteht aus Befehlen mit Vorsilbe. Die als Vorsilben wirkenden Bytes stehen in Zeile 9, in Zeile A, Spalte 8 bis D, und in Zeile C, Spalte E und F, der QRC. Die zulässigen Nachsilben stammen aus sämtlichen Bereichen der QRC. Die Bildung der Zwei-Byte-Funktionen mit Vor- und Nachsilbe erfolgt nach Gesetzen, die man anhand der folgenden Beispiele erkennen kann: 'STO M' ≙ 145, 117; 'TONE C' ≙ 159, 104; 'RCL IND N' ≙ 144, 246; 'LBL X' (lokal!) ≙ 207, 115.

Die zweite Klasse besteht aus den Kurzform-'GTO'-Befehlen. Deren erstes Byte stammt aus Zeile B, Spalte 1 bis F; das zweite Byte ist zunächst ein automatisch nachgestelltes 'NULL'-Byte. Wenn der HP-41 das 'GTO' das erste Mal ausführt, ersetzt er selbständig das 'NULL'-Byte durch ein Byte, das dem Prozessor die Sprungrichtung und den Sprungabstand zum zugehörigen 'LBL' mitteilt.

Die dritte Klasse besteht aus den beiden Befehlen 'GTO IND' und 'XEQ IND'. In beiden Fällen lautet der Dezimalwert der Vorsilbe 174. Folgt als Nachsilbe ein Wert aus der oberen Hälfte der Byte-Tabelle (0 bis 127), hat die Vorsilbe 174 die Bedeutung 'GTO IND'; folgt ein Wert aus der unteren Hälfte (128 bis 255) wird 174 als 'XEQ IND' interpretiert. Beispiele: 174, 117≙'GTO IND M'; 174, 245≙'XEQ IND M'.

Die vierte und letzte Klasse der Zwei-Byte-Funktionen enthält die 'XROM's. Dies sind die in Peripherie-Geräten und Moduln (*externe ROM*s) ansässigen Funktionen. Solange ein Peripherie-Gerät nicht angeschlossen ist, erscheint eine darin ansässige Funktion als 'XROM i,j'. In dieser Darstellung ist i eine zweiziffrige Zahl zwischen 0 und 31 und j eine ebensolche zwischen 0 und 63. Die Zahl i identifiziert das Peripherie-Gerät und wird daher ROM-ID-Zahl genannt. Manche Peripherie-Geräte enthalten zwei ROMs zu je 4 Kilobyte, von denen dann jedes seine eigene ROM-ID-Zahl hat. Die Zahl j bildet die der jeweiligen Funktion zugewiesene Ordnungszahl ('CAT 2'-Ordnung) in Bezug auf das 4 Kilobyte ROM, in dem sie enthalten ist.

'XROM'-Funktionen bestehen aus 4 halben Bytes, das sind 16 Bits, die folgendermaßen gruppiert sind. Das erste Halbbyte lautet stets hexadezimal A (binär 1010). Darauf folgen zwei Gruppen zu je 6 Bits. Die erste dieser beiden 6-er Gruppen enthält in binärer Darstellung die ROM-ID-Zahl (dezimal 0 bis 31) des externen ROMs. Beispielsweise 29 (binär 011101) für den Drucker, 30 für den Kartenleser. Die zweite 6-er Gruppe enthält in derselben Darstellung die Ordnungszahl (dezimal 0 bis 63) der Funktion im externen ROM. So ist z.B. 'WSTS' die 10. Funktion (binär 001010) im ROM des Kartenlesers. Dies können Sie dadurch nachprüfen, daß Sie – mit angeschlossenem Kartenleser – 'CAT 2' aufrufen. 'WSTS' erscheint dann als 10. Funktion hinter der Überschrift "CARD READER". Mithin ist 'XROM 30,10' die Darstellung von 'WSTS' bei abgekoppeltem Kartenleser. In Dezimalwerten entspricht das dem Paar 167,138 (vgl. Abb. 3.1). Die allgemein gültigen Formeln für die Dezimalwerte von 'XROM i,j' lauten (vgl. auch Aufgabe 3.2):

Dezimalwert von Byte 1: $160+\mathrm{INT}(i/4)$
Dezimalwert von Byte 2: $64\cdot(i \bmod 4)+j$

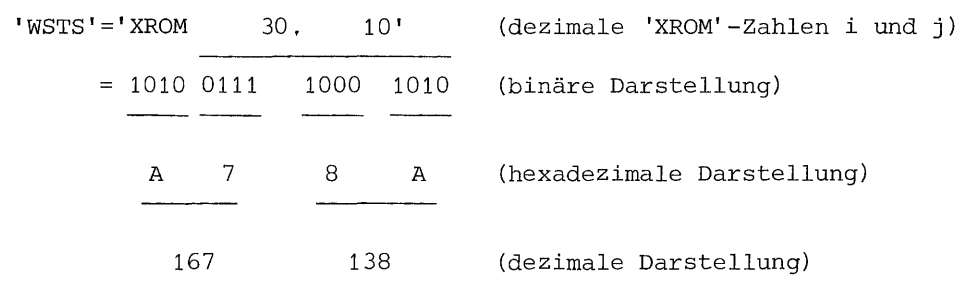


Abbildung 3.1. Ein 'XROM'-Befehl mit seinen verschiedenen Bit-Gruppierungen und Darstellungen

III. Drei-Byte-Befehle

Das erste Byte von Drei-Byte-Befehlen entstammt dem grün unterlegten Bereich der QRC. Man unterscheidet drei Klassen.

Die erste Klasse besteht aus Langform-'GTO'-Befehlen. Das sind 'GTO's, die auf lokale Marken von 'LBL 15' an aufwärts weisen. Allerdings kann man mit "LB" auch Drei-Byte-'GTO's, die zu den Marken 'LBL 00' bis 'LBL 14' verzweigen, erzeugen. So läßt sich mit den Techniken synthetischer Programmierung die mit den numerischen Marken 00 bis 14 verbundene Beschränkung auf Sprunglängen von höchstens 112 Bytes umgehen. Nicht, daß Sie nicht auch mit Zwei-Byte-'GTO's auf 'LBL 00' bis 'LBL 14' über beliebige Distanzen springen könnten; doch dauert die Verzweigung wesentlich länger, wenn mit den Kurzformen mehr als 112 Bytes zu überspringen sind, weil wegen der Raum verschlingenden Binärdarstellung der Sprunglängen größere Distanzen nicht in den Kurzform-'GTO's notiert werden können und daher die Speicherplatzstellen der angesprungenen Marken jedesmal von neuem gesucht werden müssen. Drei-Byte-'GTO's hingegen stellen genügend Platz zur Ablage beliebiger Sprunglängeninformation zur Verfügung, so daß jede Sprunglängenbegrenzung entfällt. Es ist daher sinnvoll, wenn irgend möglich, Kurzform-Marken ('LBL 00' bis 'LBL 14') zu verwenden, die ja nur ein Byte im Speicher belegen, und sie – in Abhängigkeit von der Sprungdistanz – entweder mit Zwei-Byte-'GTO's oder mit synthetischen Drei-Byte-'GTO's anzuspringen. Man kann folglich bei Verwendung synthetischer Drei-Byte-'GTO's Bytes einsparen, und zwar nicht mit den 'GTO's selbst, sondern wegen der Möglichkeit, ungeachtet der Sprungdistanzen die Kurzform-Marken einzusetzen.

Drei-Byte-'GTO's erfordern die folgenden "LB"-Eingaben:

```
Byte 1 = 208
Byte 2 = 0
Byte 3 = 0 bis 127
```

Byte 3 bestimmt hierbei die Marke: z.B. ergibt 208, 0, 1 ein Drei-Byte-'GTO' auf 'LBL 01', während man mit 208, 0, 115 ein lokales 'GTO X' (nicht zu verwechseln mit einem globalen 'GTO "X"') erzeugt; letzteres erfordert natürlich ein – ebenfalls nur synthetisch erzeugbares – lokales 'LBL X', dezimal 207, 115.

Die zweite Klasse von Drei-Byte-Befehlen besteht aus 'XEQ'-Befehlen, die sich auf lokale Marken beziehen. Sie ähneln den Drei-Byte-'GTO's 'aufs Haar': der einzige Unterschied besteht darin, daß Byte 1 dezimal 224 statt 208 lautet. Also: 'XEQ 98' ≙ 224, 0, 98; 'XEQ L' ≙ 224, 0, 116 (hierfür ist ein wiederum nur synthetisch erzeugbares lokales 'LBL L' – dezimal 207, 116 – erforderlich).

Wer *kompilierte* 'GTO's und 'XEQ's (das sind Befehle, die die Sprunglängeninformation bereits enthalten) herstellen will, findet Einzelheiten dazu auf Seite 21 der Ausgabe August 1979 des PPC CJ.

Die 'END'-Befehle bilden die dritte Klasse der Drei-Byte-Befehle. Ein 'END', das mit "LB" erzeugt werden soll, verlangt für Byte 1 den Eingabewert 192, für Byte 2 den Wert 0 und für Byte 3 einen Wert, der den Typ des 'END's festlegt und der der nachstehenden Tabelle entnommen werden kann.

Typ des 'END'	"LB"-Eingaben für Byte 3
gepacktes 'END'	9
ungepacktes 'END'	13
gepacktes '.END.'	41
ungepacktes '.END.'	45

Tabelle 3.1. "LB"-Eingaben für Byte 3 eines 'END'.

Vergessen Sie nicht, im Anschluß an synthetisch erzeugte 'END's oder globale Alpha-Marken ein 'PACK' auszuführen, um sie in den Katalog 1 einzufügen. Die globalen Marken (Alpha-Marken und 'END's) sind nämlich in diesem Katalog zu einer Art Kette, beginnend mit dem ständigen '.END.', zusammengefügt. Dabei enthält jede Marke (jedes 'Kettenglied') in ihrem ersten und zweiten Byte die Information über den Abstand zur jeweils nächsten Marke. Die Verschlüsselung des Abstandes ist dieselbe wie bei Drei-Byte-'GTO's bzw. -'XEQ's mit dem Unterschied, daß kein Richtungsbit benötigt wird, weil es für die Kette des Katalogs 1 nur eine Richtung gibt, nämlich 'aufwärts im Programmspeicher'. Die hier notierten "LB"-Eingaben zur Erzeugung der 'END's sind vereinfachte Kodes, weil der Prozessor bei jedem 'PACK' die fehlenden Abstandsangaben selbständig einsetzt und die Verkettung im Katalog 1 auf den neuesten Stand bringt.

Beachten Sie in diesem Zusammenhang noch folgenden einfachen empfehlenswerten synthetischen Trick, der von Clifford Stern stammt und Ihnen erlaubt, *zu 'PACK'en, ohne zu 'dekompilieren'*. Angenommen, Sie haben ein Programm, dessen 'GTO's und 'XEQ's nach der ersten Ausführung kompiliert sind, auf Magnetkarten aufgezeichnet. Wenn Sie dieses Programm wieder einlesen, sind die 'GTO's und 'XEQ's nach wie vor kompiliert, so daß keine Suche nach den angesprungenen Marken erforderlich ist. Ein 'PACK' oder 'GTO ..' jedoch vernichtet die Sprunglängenkodes. Diesem Mißstand kann man so abhelfen: Programm einlesen, Übergang in den PRGM-Modus, 'BST' (damit gelangen Sie auf die letzte Zeile, das '.END.', welches für einen einwandfreien Ablauf dieses Verfahrens mindestens zwei freie Register vorweisen muß: ".END. REG mn", mn ≥ 02), 'ENTER', 'STO IND 64', 'BST', BS, zweimal Korrekturtaste, 'PACK' (nicht 'GTO ..'), und zwar immer noch im PRGM-Modus. Die Nachsilbe aus 'STO IND 64' wird so zum ersten Byte eines gepackten 'END', das den Prozessor davon abhält, die Sprunglängeninformationen zu löschen. Bytes werden dabei nicht vergeudet, weil das 'PACK' alle überflüssigen 'NULL'en beseitigt, während das neue synthetisch erzeugte 'END' die normalerweise dabei erfolgende 'Dekompilation' verhindert. Natürlich können Sie dieses Verfahren auch auf Programme, die aus dem X-Memory oder von der Kassette eingelesen werden, anwenden.

IV. Befehle, die Alpha-Ketten enthalten

Textzeilen werden von einem Byte aus Zeile F der QRC angeführt (dezimal 240, erhöht um die Anzahl der Zeichen der Kette), wie schon in Abschnitt 2E ausgeführt wurde. Jedes Zeichen erfordert eine gesonderte Dezimalwert-Eingabe, die gewöhnlich zwischen 0 und 127 liegt. Z.B. lautet "X(5)=?" dezimal 246, gefolgt von 88, 40, 53, 41, 61 und 63.

Anhangsbefehle sind Textzeilen, deren erstes Zeichen das Anhangssymbol "⊢" (Byte 7F, dezimal 127) ist. Weil das Anhangssymbol als zur Kette gehörig gilt, muß das Textführungsbyte entsprechend gewählt werden. Z.B. lautet "⊢@" dezimal 242, 127, 64.

'GTO⌜'-Befehle sind Textzeilen, denen Byte 1D (dezimal 29) vorangeht. Beispielsweise ergibt 29, 243, 65, 66, 67 ein 'GTO "ABC"'. 'XEQ⌜'-Befehle sind ebenso aufgebaut, nur steht ihnen Byte 1E (dezimal 30) voran, z.B. 30, 242, 70, 88 für 'XEQ "FX"'. Das geheimnisvolle 'W⌜', Byte 1F, weist abermals dieselbe Bauart auf, doch erwarten Sie nicht zuviel von dieser Funktion: ihre Leistung beschränkt sich nämlich i.a. darauf, einen GAU auszulösen, der nur durch Entfernen und Wiedereinsetzen der Batterien beseitigt werden kann.

Globale Alpha-Marken bestehen aus 4+n Bytes, wobei n die Anzahl der Zeichen der Marke angibt. Die für "LB" nötigen Eingaben lauten: 192, 0, 241+n, 0, gefolgt von den n Dezimalwerten der Zeichen. 'LBL "A"' (eine globale Marke, die in den Katalog 1 gelangt, nicht zu verwechseln mit einem lokalen 'LBL A', dezimal 207, 102) lautet also dezimal 192, 0, 242, 0, 65. Wenn Sie eine synthetisch erzeugte Alpha-Marke einer Taste zuweisen wollen, benötigen Sie für die vierte "LB"-Eingabe einen von 0 verschiedenen Wert. Gleichzeitig müssen Sie in einem der Zustandsregister ⊢ oder e (s. Abschnitt 6A) ein Bit setzen. Die zur synthetischen Herstellung von Tastenbelegungen notwendige Kenntnis über den Zusammenhang zwischen Byte-Werten und Bit-Zahlen vermittelt das Benutzerhandbuch des PPC ROMs unter 'Background for MK' sowie das Buch 'Synthetische Programmierung auf dem HP-41C/CV' von W.C. Wickes.

Ein wesentlich einfacherer Weg, eine synthetisch erzeugte Marke einer Taste zuzuweisen, ist natürlich der, die eingebaute Funktion 'ASN' zu benutzen. Nur dann, wenn die Marke nicht tastbare Zeichen (z.B. "Ẋ") oder solche, die von 'ASN' nicht angenommen werden (z.B. ":"), enthält, bleibt dieser Weg versperrt. In solchen Fällen hilft aber die X-Funktion 'PASN'.

Bemerkung: Denken Sie stets daran, synthetisch erzeugte Alpha-Marken oder 'END's sofort durch 'PACK' dem Katalog 1 einzufügen. – Üben Sie mit "LB" solange, bis Sie die in Kapitel 2 besprochenen Typen synthetischer Befehle einwandfrei erzeugen können.

Aufgaben

3.1 Verwenden Sie "LB", um die folgenden synthetischen Befehle zu erzeugen:

```
E
STO O
ST+ O
X<> O
STO M
ISG M
TEXT O
ΣREG IND M
VIEW O
FS? IND M
TONE E
"ЖЛЖЈ"
"├-Т"
ASTO N
VIEW N
```

Die voranstehenden Befehle stehen in keinem sinnvollen Zusammenhang, doch veranschaulichen sie die breit gefächerten Möglichkeiten, Funktionen synthetisch zu erzeugen, und im Einzelfall und bei passender Verknüpfung können sie außerordentlich gute Dienste leisten.

3.2 Schreiben Sie ein kurzes nicht-synthetisches Programm, um 'XROM'-Zahlen in die zugehörigen "LB"-Eingaben umzurechnen. Die Eingabe für dieses Programm soll in der Form 'i, ENTER↑, j' erfolgen, die Ergebnisse sollen 160+INT(i/4) bzw. 64·(i mod 4)+j lauten, wie bereits im Abschnitt über die Zwei-Byte-Befehle ausgeführt wurde. Die Ergebnisse sind die für die Erzeugung von 'XROM i,j' gesuchten "LB"-Eingaben. – Schreiben Sie auch eine synthetische Version, die i und j durch die beiden Ergebnisse ersetzt und die Register Z und T unversehrt läßt. (Lösung S.113)

3.3 Veranschaulichen Sie die Verwendung lokaler synthetischen Marken durch Erzeugung der Befehlsfolge

```
LBL P       (nicht 'LBL "P"'!)
TONE 37     (angezeigt als 'TONE 7')
GTO P       (nicht 'GTO "P"'!)
```

3.4 Erzeugen Sie eine globale synthetische Alpha-Marke einer Länge größer als 7, etwa 'LBL "RPN CALCULATOR"', und fügen Sie sie dem Katalog 1 ein.

3.5 Wenn Sie keinen PPC ROM, hingegen einen X-Funktionen-Modul besitzen, können Sie die folgende kürzere und schnellere Version von "LB", die ebenfalls von Clifford Stern geschrieben wurde, mit dem Byte-Schnapper edieren. Die synthetischen Befehle für "LBX" sind mit denen für "LB" gleichlautend, so daß Sie zur Edition von "LBX" den Anweisungen zur Herstellung von "LB" folgen können. Haben Sie "LB" schon fertig, können Sie natürlich auch "LBX" mit "LB" edieren. Die für die Erzeugung der synthetischen Zeilen von "LBX" nötigen "LB"-Eingaben finden Sie als Lösung von Aufgabe 3.5 auf S.115. Dort können Sie also nachschlagen, falls Sie es eilig haben. Sofern Sie sich dafür entscheiden, ständig "LBX" statt "LB" zu benutzen, sollten Sie nach Edition von "LBX" das Programm "LB" löschen und "LBX" in "LB" umbenennen.

```
01♦LBL 01       23 ARCL X       45 +        67 GTO 04
02 CLST         24 "├ REGS."    46 +        68 SIGN
03 BEEP         25 TONE 8       47 ,1       69 X<> c
04 STOP         26 AVIEW        48 %        70 LASTX
05 GTO "++"     27 PSE          49 +        71 STO IND T
06♦LBL "LBX"    28 RCL b        50 +        72 X<>Y
07 FS? 50       29 "*"          51♦LBL 03   73 STO c
08 GTO 02       30 X<> [        52 1,007    74 R↑
09 1            31 -2           53 ENTER↑   75 DSE X
10 ENTER↑       32 AROT         54♦LBL 04   76 GTO 03
11 ENTER↑       33 RDN          55 " "      77 GTO 01
12 CLA          34 STO \        56 ARCL Y   78♦LBL 05
13 CF 21        35 ASHF         57 "├?"     79 "├♦"
14 AVIEW        36 SIGN         58 AVIEW    80 ISG X
15 -10          37 ALENG        59 STO [    81 GTO 05
16 GTO "++"     38 8            60 RDN      82 X<> c
17♦LBL 02       39 Y↑X          61 STOP     83 RCL [
18 7            40 ATOX         62 FC?C 22  84 STO IND Z
19 /            41 *            63 GTO 05   85 X<>Y
20 INT          42 512          64 XTOA     86 STO c
21 FIX 0        43 MOD          65 X<> [    87 GTO 01
22 CF 29        44 ATOX         66 ISG Y    88 END
```

160 BYTES

KAPITEL 4

Synthetische Tastenzuweisungen

4A. Programme für synthetische Tastenzuweisungen

Gegenüber dem einfachen Byte-Schnapper stellen Byte-Lader ohne Zweifel einen großen Fortschritt dar. Synthetische Tastenzuweisungsprogramme erweitern unsere Möglichkeiten, synthetische Befehle zu erzeugen, abermals ganz erheblich: ein solches Programm vermag jede Ein- oder Zwei-Byte-Funktion, sei sie synthetisch oder nicht-synthetisch, jeder beliebigen Taste zuzuweisen. Wenn Sie Ihre synthetischen Befehle mit dem äußerst möglichen Maß an Bequemlichkeit in Ihre Programme einfügen wollen, dann können Sie häufig benutzte synthetische Funktionen Tasten zuweisen und die verbleibenden selteneren synthetischen Zeilen mit "LB" erzeugen.

Tastenzuweisungsprogramme arbeiten ganz ähnlich wie Byte-Lader, insoweit sie nämlich Dezimalwerte als Eingaben erwarten, diese in die Hexadezimalwerte der entsprechenden Bytes umwandeln und dann den für Tastenzuweisungen vorgesehenen Teil des Arbeitsspeichers damit belegen. Anders jedoch als bei "LB", das Wert für Wert einzeln geliefert bekommt, werden hier alle erforderlichen Angaben auf einen Schlag verarbeitet: Der Stapel wird mit den Dezimalwerten zweier Bytes und dem Zeilen/Spalten-Kode der Taste, welche die Zuweisung tragen soll, geladen.

Die ersten Tastenzuweisungsprogramme wurden Anfang 1980 von John McGechie geschrieben. Sie waren aus heutiger Sicht eine wahrlich furchteinflößende Verwirklichung der Vorstellungen, die man dazu hatte, entsprechend dem Stand der synthetischen Programmierung zu jener Zeit.

In diesem Kapitel werden Ihnen – ebenso wie bei den Byte-Ladern – insgesamt drei verschiedene Tastenzuweisungsprogramme zur Verfügung gestellt. Das erste heißt "MK" (*m*ake *k*ey assignments) und bedarf nur des einfachen HP-41. Dieses Programm belegt drei Spuren zweier Karten. Es stammt von Clifford Stern.

Das zweite Tastenzuweisungsprogramm ist das im PPC ROM enthaltene MK, geschrieben von Roger Hill. MK ist ein Meisterstück synthetischer Programmierung, im wesentlichen unempfindlich gegen Bedienungsfehler. Wenn Sie den PPC ROM Ihr eigen nennen, lesen Sie die zu MK gehörige Gebrauchsanweisung im Benutzerhandbuch durch.

Das dritte Programm heißt "MKX" und erfordert den X-Funktionen-Modul. Geschrieben hat es Tapani Tarvainen. Es paßt auf eine Magnetkarte, ist kürzer und schneller als "MK" und MK und verzeiht Fehler des Benutzers noch großzügiger als die beiden anderen Programme. Die Auflistung von "MKX" finden Sie am Ende dieses Kapitels unter Aufgabe 4.4.

Obwohl das von Clifford Stern stammende "MK" ein recht kurzes Programm ist, enthält es viele der angenehmen Eigenschaften der PPC ROM Version MK. Wie im Falle LB/"LB" war es auch hier nicht möglich, alle Bequemlichkeiten und fehlerabweisenden Teilstücke aus MK in "MK" einzufügen, ohne das Programm unangemessen aufzublähen. Die wichtigste Fehlerfalle, die Meldung "KEY TAKEN" bei belegter Taste, ist aber eingebaut. Ein wenig Fehlerprüfung durch den Benutzer statt durch das Programm erspart eben viele Bytes.

Wenn Sie einen optischen Lesestift haben, können Sie "MK" oder "MKX" unmittelbar Ihrem HP-41 einverleiben, indem Sie sich der im Anhang E abgedruckten Barcodes bedienen. Allerdings mag es besser sein, wenn Sie zunächst den Umgang mit "LB" üben und mit dessen Hilfe eines der beiden Tastenzuweisungsprogramme edieren.

"MK", das nur den 'nackten' HP-41 erfordert, wird – zusammen mit den "LB"-Einträgen zur Erzeugung der synthetischen Befehle – nachstehend aufgelistet. Sobald Sie die synthetischen Befehle mit "LB" hergestellt haben, tasten Sie die nicht-synthetischen Zeilen in gewohnter Weise dazwischen, um das Programm zu vervollständigen. Denken Sie dabei daran, daß die Nachsilben M, N, O, P, Q und ⊢ auf dem Drucker als [, /,], ↑, _ bzw. ⁻ erscheinen (P und Q, also ↑ und _, werden in "MK" nicht verwendet).

Beachten Sie auch, daß die Zeilen 11, 20 und 38 von mehr Bytes bevölkert sind, als ihr Ausdruck verrät. Dem im Anschluß an die Programm-Auflistung abgedruckten Verzeichnis der "LB"-Eingaben können Sie die vollständige Zusammensetzung dieser und der anderen synthetischen Programmzeilen entnehmen.

```
01*LBL "MK"
02 CLST
03 CF 02
04 CF 05
05 CF 06
06 CF 21
07 192
08 SIGN
09 X<> c
10 X<> Z
11 ""
12 RCL b
13 RDN
14 X<> IND L
15 X=Y?
16 GTO 02
17 X<> [
18 "⊢*"
19 STO \
20 "⊢*****"
21 X<> \
22 X<> IND L
23 R↑
24 ISG L
25 -
26 STO b
27*LBL 01
28 "⊢****"
29 X<> ]
30 "KEY TAKEN"
31 TONE 0
32 AVIEW
33 PSE
34*LBL 16
35 "PRE↑POST↑KEY"
36 TONE 8
37 AVIEW
38 ""
39 FS? 02
40 STO [
41 CLST
42 STOP
43 LASTX
44 XEQ 03
45 XEQ 03
46 R↑
47 X<0?
48 SF 05
49 ABS
50 STO \
51 R↑
52 X<> \
53  E1
54 MOD
55 X<>Y
56 LASTX
57 /
58 INT
59 4
60 DSE Z
61 X≠Y?
62 X=0?
63 ISG Z
64 ""
65 ST+ X
66 ENTER↑
67 R↑
68 *
69 ENTER↑
70 R↑
71 +
72 ST+ Y
73 RDN
74 FS? 05
75 +
76 R↑
77 RCL \
78 R↑
79 XEQ 03
80 X<> T
81 X<Y?
82 SF 06
83 36
84 -
85 FS? 06
86 +
87 R↑
88 SIGN
89 FS? 05
90 RCL e
91 FC? 05
92 RCL '
93 STO \
94 FS? 06
95 "⊢*"
96 X<> \
97 X<> d
98 FS? IND Z
99 DSE Y
100 SF IND Z
101 X<> d
102 STO \
103 "⊢*******"
104 FC?C 06
105 "⊢*"
106 X<> ]
107 FS? 05
108 STO e
109 FC?C 05
110 STO '
111 X<>Y
112 X=0?
113 GTO 01
114 X<> c
115 RCL \
116 FC? 02
117 "⊢***"
118 RCL \
119 STO IND L
120 FS?C 02
121 ISG L
122 SF 02
123*LBL 02
124 X<> Z
125 STO c
126 X<>Y
127 GTO 16
128*LBL 03
129 R↑
130 OCT
131 STO \
132 CLX
133  E4
134 ST+ \
135 X<> \
136 X<> d
137 FS?C 19
138 SF 20
139 FS?C 18
140 SF 19
141 FS?C 17
142 SF 18
143 FS? 15
144 SF 17
145 FS? 14
146 SF 16
147 X<> d
148 X<> [
149 "⊢**"
150 STO \
151 "⊢*"
152 X<> \
153 STO [
154 END
```

313 BYTES

"LB"-Eingaben:

```
Zeile 09:  206, 125     Zeile 11:  241, 240*)    Zeile 12:  144, 124
  "   17:  206, 117       "   19:  145, 118
  "   20:  247, 127,  42, 42, 42,  42, 42, 240*)
  "   21:  206, 118     Zeile 26:  145, 124        "   29:  206, 119
  "   38:  241, 240*)     "   40:  145, 117        "   50:  145, 118
  "   52:  206, 118       "   53:  27, 17          "   64:  240
  "   77:  144, 118       "   90:  144, 127        "   92:  144, 122
  "   93:  145, 118       "   96:  206, 118        "   97:  206, 126
  "  101:  206, 126       "  102:  145, 118
  "  103:  247, 127,   0,  0,  0,  42, 42, 42
  "  106:  206, 119       "  108:  145, 127        "  110:  145, 122
  "  114:  206, 125       "  115:  144, 118        "  118:  144, 118
  "  125:  145, 125       "  131:  145, 118        "  133:  27, 20
  "  134:  146, 118       "  135:  206, 118        "  136:  206, 126
  "  147:  206, 126       "  148:  206, 117        "  150:  145, 118
  "  152:  206, 118       "  153:  145, 117
```

*) Textzeile enthält unsichtbare Zeichen aus der zweiten Hälfte der Byte-Tabelle.

Überzeugen Sie sich sorgfältig davon, daß Sie "MK" fehlerfrei eingetastet haben, bevor Sie das Programm zum ersten Mal aufrufen. Ebenso wie bei "LB" können Sie nämlich auch hier ein "MEMORY LOST" erleiden, falls das Programm fehlerbehaftet ist oder nicht weisungsgemäß benutzt wird. Die hinter "MK" stehende Theorie ist zu verwickelt, um hier besprochen zu werden. Das Schreiben eines Tastenzuweisungsprogramms mit der Bedingung 'SIZE 000', also unter Verzicht auf numerierte Datenregister, ist übrigens in der synthetischen Programmierung eine Herausforderung ersten Ranges. In diesem Buch beschränken wir uns auf eine Besprechung der *Benutzung* von "MK".

Gebrauchsanweisung für Clifford Stern's "MK":

1.) Falls Sie den Time-Modul angeschlossen haben, löschen Sie alle Weckaufträge, weil diese, wenn "MK" (bzw. MK) ausgeführt wird, durch den Vorgang der Normalisierung (vgl. Abschnitt 2C) als unverwertbarer Schutt zurückgelassen werden. Sie können die Weckaufträge neu setzen, wenn Sie die Tastenzuweisungen mit "MK" getätigt haben. Im Abschnitt 4E werden Sie zwei handliche Routinen kennenlernen, die alle Weckaufträge automatisch in den erweiterten Speicher bringen und von dort zurückholen. Mit "SA" (*s*ave *a*larms) werden sie vor dem Aufruf von "MK" gerettet und im Arbeitsspeicher gelöscht und hinterher mit "RA" (*r*ecall *a*larms) zurückgeholt. Benutzer des PPC ROMs müssen zusätzlich beachten, daß die Weckauf-

träge auch vor Aufruf von PK oder einer Routine die ihrerseits LF (1K, +K, A? oder F?) aufruft, gelöscht werden müssen. Bei Anwendung von "MKX" (Aufgabe 4.4) gibt es keine Beschränkung bezüglich der Weckaufträge.

Außerdem gibt es noch ein auf verzwickten Zusammenhängen zwischen den Registern e, ⊢ und d und dem Betriebssystem des HP-41 beruhendes Fehlverhalten, sobald man Tastenzuweisungsprogramme ("MKX", welches nicht direkt auf die Tastenzuweisungsbits zugreift, ausgenommen) bei eingestecktem Time-Modul verwendet. Es besteht in einer unerwünschten Nebenwirkung, die von Bill Childers aufgedeckt und von Clifford Stern untersucht worden ist: Wenn man nämlich Funktionen anderen Tasten als denen der Zeilen 1–7/Spalte 1 zuweist, gilt folgende 'Verlustregel': nimmt man eine Zuweisung auf eine nicht-umgeschaltete Taste vor, *kann* eine bereits auf der Taste 61 ('+') liegende Zuweisung verloren gehen; nimmt man eine Zuweisung auf eine umgeschaltete Taste vor, geht eine bereits auf der Taste –61 liegende Zuweisung *mit Sicherheit* verloren. Falls Ihnen Tastenzuweisungen auf diese Weise abhanden gekommen sind, brauchen Sie nur eine Programmkarte einzulesen, um die Tastenzuweisungsbits in den alten Zustand zu versetzen (vgl. Abschnitt 6B) und die verlorenen Zuweisungen zurückzugewinnen. Man kann die gefährdeten Zuweisungen auch vor Aufruf von "MK" 'freiwillig' löschen und sie *am Schluß* (wenn sie synthetisch sind mit "MK") neu herstellen.

2.) Vergewissern Sie sich vor dem Aufruf von "MK", daß genügend Tastenzuweisungsregister zur Verfügung stehen, indem Sie die Anzahl der freien Register mit 'GTO .000' im PRGM-Modus feststellen. Sie können doppelt soviele Zuweisungen mit "MK" tätigen, wie freie Register durch 'GTO .000' angezeigt werden, weil jedes Register zwei Zuweisungen aufzunehmen vermag. MK aus dem PPC ROM ist bedienungsfreundlicher und teilt dem Benutzer selbständig "NO ROOM" mit, wenn keine freien Datenregister mehr vorhanden sind.

3.) Der Vorgang der Zuweisung wird mit 'XEQ "MK"' gestartet. Das Programm sucht sich das erste freie Tastenzuweisungsregister, so daß bereits vorhandene Zuweisungen unbehelligt bleiben. Unterbrechen Sie weder "MK" noch "MKX", denn bei "MK" kann ein "MEMORY LOST" die Folge sein, und bei "MKX" ist es möglich, des Zugriffs auf den Katalog 1 verlustig zu gehen. Sollten Sie dennoch "MKX" gestoppt haben, starten Sie sofort wieder, ohne zwischendurch in den PRGM-Modus zu wechseln. Ein Übergang in den PRGM-Modus wirft Sie nämlich aus dem Programm und zwingt Sie, mit einer Totallöschung die Kontrolle über den Rechner zurückzugewinnen, es sei denn, Sie finden im Stapel noch den Inhalt von Register c auf und speichern ihn mit 'STO c' ordnungsgemäß zurück. Die Bedeutung eines 'STO c' wird Ihnen klar sein, wenn Sie Kapitel 6 durchgearbeitet haben.

4.) Sobald die Aufforderung "PRE↑POST↑KEY" erscheint, tasten Sie die drei Bausteine einer Tastenzuweisung – Dezimalwert 1, 'ENTER↑', Dezimalwert 2, 'ENTER↑', Tastenkode (Zeile/Spalte) – ein und drücken 'R/S'. Z.B. ergibt '144, ENTER↑, 124, ENTER↑, 12, R/S' die Zuweisung von 'RCL b' auf die Taste '1/x': die Vorsilbe 'RCL' lautet dezimal 144, der Dezimalwert der Nachsilbe b ist 124, und der Zeilen-/Spaltenkode der (nicht-umgeschalteten) Taste '1/x' heißt 12. Die ersten beiden Dezimalwerte dürfen beliebige ganze Zahlen zwischen 0 und 255 sein, während die dritte Eingabe einen gültigen Tastenkode darstellen muß, also eine Zahl der Form '±zs', in der z die Zeilen- und s die Spaltennummer einer Taste bilden und ein negatives Vorzeichen anzeigt, daß die mit 'SHIFT' umgeschaltete Taste gemeint ist. Es handelt sich somit um genau den Tastenkode, der kurz in der Anzeige erscheint, wenn Sie 'ASN'

ausgeführt haben, und der auch für 'PASN' (X-Funktion zur programmierten Tastenzuweisung) erforderlich ist. MK und "MK" erlauben beide, auch der umgeschalteten Umschalttaste 'SHIFT' (Tastenkode −31) eine Zuweisung aufzuerlegen (mit "MKX" gelingt dies nicht). Falls Sie dies tun, ist es günstig, dafür eine Funktion zu wählen, die zur Parametereingabe auffordert, z.B. 'X< >__', weil durch eine solche Aufforderung eine Unterbrechung stattfindet, die dem Benutzer Zeit zur 'Besinnung' gibt und so eine unbeabsichtigte Ausführung der zugewiesenen Funktion vermeiden hilft. Sollten Sie allerdings so tollkühn sein, der Umschalttaste selbst (Kode 31) eine Funktion zuzuweisen, legen Sie sie damit für den Gebrauch im USER-Modus gründlich lahm: es wird nur noch die zugewiesene Funktion ausgeführt. Beachten Sie auch, daß in beiden Fällen (Tastenkodes −31 und 31) das Löschen mit 'ASN, ALPHA, ALPHA' fehl schlägt: die Zuweisungen sind zementiert! Es gibt dann nur noch folgende unbefriedigenden Möglichkeiten, die Taste zu befreien: Einlesen von Statuskarten oder 'Kahlschlag' mit 'CLKEYS' oder CK aus dem PPC ROM. Lediglich der HP-41CX erlaubt eine saubere Lösung: 'SHIFT C' im Katalog 6 beim Vorweisen der Belegung von Taste 31.

WARNUNG: Sobald "MK" mit der Aufforderung zur Eingabe anhält, dürfen Sie nicht 'PACK'en, 'SIZE' neu festsetzen, den Rechner ausschalten, 'ASN' benutzen oder den Inhalt von ALPHA oder L verändern. Erst nach Beendigung der Zuweisung durch "MK" haben Sie dafür wieder grünes Licht.

5.) Sobald "PRE↑POST↑KEY" zum zweiten Mal erscheint (nunmehr mit gesetztem Flag 2, wenn Sie "MK" benutzen), können Sie die drei Eingaben für eine zweite Zuweisung tätigen. Damit vervollständigen Sie ein Tastenzuweisungsregister.

6.) Die Meldung "PRE↑POST↑KEY" erscheint abermals (jetzt wieder ohne Flag 2), um zur Eingabe für die erste Zuweisung, die in das nächste freie Register eingetragen werden soll, aufzufordern. Wiederholen Sie die Schritte 4 und 5, bis alle beabsichtigten Zuweisungen erfolgt sind. Denken Sie dabei daran, daß Sie u.U. wegen der anfangs ermittelten Zahl der freien Register nur eine gewisse Höchstanzahl von Zuweisungen vornehmen dürfen.

7.) Sobald alle Zuweisungen erfolgt sind, dürfen Sie die Aufforderung zur nächsten Eingabe mißachten. Dies gilt auch für den Fall, daß mit der letzten Zuweisung ein neues Register 'angebrochen' wurde. Sie verschwenden jedoch ein halbes Register, wenn Sie die Eingabe-Aufforderung bei gesetztem Flag 2 (nur "MK") ignorieren, es sei denn, sie beabsichtigen, das angebrochene Register mit einer normal zu tätigenden Zuweisung ('ASN' oder 'PASN') zu vervollständigen. Im Gegensatz zu "MK" halten 'ASN' und 'PASN' nämlich Ausschau nach Lücken in den Tastenzuweisungsregistern, bevor sie ein neues Register zu füllen beginnen.

8.) Der Versuch, eine Taste zu belegen, die schon eine Zuweisung trägt, führt zur Meldung "KEY TAKEN". Sie haben dann zwei Möglichkeiten (vergessen Sie jetzt nicht die unter Schritt 4 ausgesprochene Warnung!): Sie können entweder die in Aussicht genommene Taste mit 'ASN, ALPHA, ALPHA, Tastenkode' von ihrer Belegung befreien und anschließend die drei notwendigen Eingaben erneut tätigen, gefolgt von 'R/S'. Sie können aber auch eine andere freie Taste wählen, indem Sie die beiden Dezimalwerte abermals eintasten und daraufhin den Kode der freien Taste eingeben; dann 'R/S'.

Um uns die Leistungsfähigkeit des Programms vor Augen zu führen, werden wir die folgenden Zuweisungen synthetischer Funktionen vornehmen:

```
STO b -11   STO d -12   STO M -13   STO N -14   STO O -15
RCL b  11   RCL d  12   RCL M  13   RCL N  14   RCL O  15
BS    -21   X<> d -22   X<> M -23   X<> N -24   X<> O -25
```

Folgende Schritte sind erforderlich:

1. Sämtliche Zuweisungen, die auf der obersten Tastenreihe liegen, und sämtliche Zuweisungen, die auf den umgeschalteten Tasten der zweiten Reihe liegen, von Hand löschen.
2. Anzahl der freien Register mit 'GTO .000' ermitteln. Für die beabsichtigten 15 Zuweisungen sind 8 Register erforderlich.
3. 'XEQ "MK"'. Eingabe-Aufforderungen entsprechend der nachstehenden Liste beantworten.

Flag 2 (nur "MK")	Eingabe: ("MK", MK oder "MKX")
gelöscht	145, 124, -11, R/S
gesetzt	144, 124, 11, R/S
gelöscht	145, 126, -12, R/S
gesetzt	144, 126, 12, R/S
gelöscht	145, 117, -13, R/S
gesetzt	144, 117, 13, R/S
gelöscht	145, 118, -14, R/S
gesetzt	144, 118, 14, R/S
gelöscht	145, 119, -15, R/S
gesetzt	144, 119, 15, R/S
gelöscht	247, 63, -21, R/S
gesetzt	206, 126, -22, R/S
gelöscht	206, 117, -23, R/S
gesetzt	206, 118, -24, R/S
gelöscht	206, 119, -25, R/S
gesetzt	Korrekturtaste oder ignorieren

Mit diesen synthetischen Funktionen haben Sie etwa zwei Drittel dessen zur Hand, was durchschnittlich an synthetischen Programmzeilen auftaucht. Beispielsweise liegen nur ein Drittel der in "LB" und "MK" vorkommenden synthetischen Zeilen außerhalb dieser Gruppe.

Einige nicht-synthetischen Funktionen werden vorteilhafterweise ebenfalls auf Tasten gelegt. Zu empfehlen sind diese Zuweisungen:

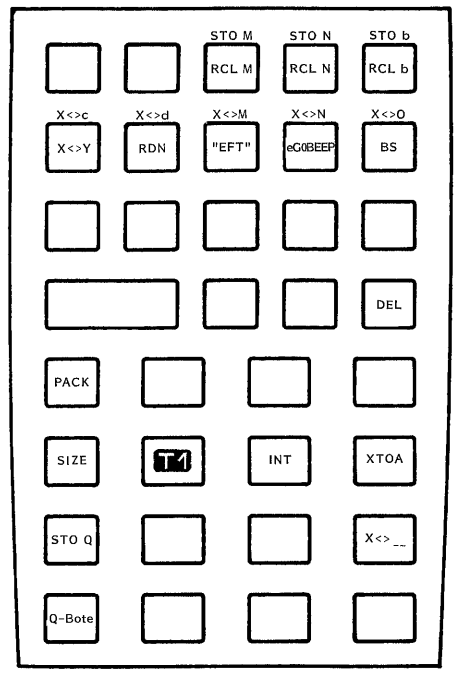
```
ASN "X<>Y"    21
ASN "RDN"     22
ASN "SIZE"    23
ASN "PACK"    24
ASN "DEL"     25
```

Die ersten beiden dieser Zuweisungen beseitigen die zeitraubende Suche nach den lokalen Marken 'LBL F' und 'LBL G', wenn man 'X< >Y' bzw. 'RDN' im USER-Modus drückt; die Antwort wird in vielen Fällen erheblich beschleunigt. Die anderen drei Funktionen werden so häufig benötigt, daß es handlich ist, gerade sie auf einer Taste liegend schnell erreichbar zur Verfügung zu haben, obwohl es natürlich eine Frage des persönlichen Geschmacks ist, welche der gewöhnlichen Funktionen man vorzugsweise Tasten zuweist. 'PACK' und 'DEL' sind insbesondere im Zusammenwirken mit dem Byte-Schnapper nützlich. Dieser und "LB" können dazu verwendet werden, jede der synthetischen Funktionen, die keiner Taste zugewiesen sind, zu erzeugen.

Obwohl Sie nicht-synthetische Funktionen im Normalfall mit 'ASN' auf Tasten legen werden, erlaubt Ihnen "MK" die Zuweisung dieser Funktionen genauso wie die synthetischer Funktionen. In Erwiderung auf "PRE↑POST↑KEY" tastet man einfach einen einzelnen Dezimalwert zwischen 0 und 255, gefolgt vom Tastenkode, ein. Für 'X< >Y' lautet der Dezimalwert 113; für 'RDN' müssen Sie 117 wählen. Prüfen Sie zur Bestätigung die QRC. Für Mehrbyte-Funktionen gilt dasselbe: 'DSE' erfordert 151, 'FC?C' verlangt 171, für 'END' benötigt man 192, 207 für 'LBL', 208 für 'GTO' und 224 für 'XEQ'. Für nicht-prorammierbare Funktionen müssen Sie die Dezimalwerte aus der ersten Zeile (Zeile 0) der QRC nehmen. So werden beispielsweise 'SIZE', 'PACK' und 'DEL' mit den Dezimalwerten 6, 10 bzw. 2 zugewiesen. Falls Sie die voranstehenden Beispiele mit "MKX" statt mit "MK" durchführen, müssen Sie u.U. noch 0 als ersten Dezimalwert eintasten (bei "MK" ist automatisch gewährleistet, daß der Dezimalwert 0 im Register Z liegt, falls nur ein Dezimalwert eingegeben wird).

Falls Sie jemals 'STO c' oder 'X< >c' einer Taste zuweisen, um diese Funktionen bequem einem Programm einfügen zu können, sollten Sie die Zuweisungen gleich darauf wieder löschen. Oder Sie müssen *äußerste Sorgfalt* walten lassen. Ein zufälliges 'STO c' oder 'X< >c' über das Tastenfeld führt nämlich so gut wie sicher zu "MEMORY LOST". Da es andererseits günstig sein kann, 'X< >c' ständig auf dem Tastenfeld zur Verfügung zu haben, sollte eine solche Zuweisung wenigstens auf einer gewissermaßen unauffälligen Taste, die kaum zu einer versehentlichen Benutzung verführt, liegen, z.B. auf der Taste mit dem Kode −21 (im NORMAL-Modus 'CLΣ'). Mein eigenes *) vollständiges synthetisches Tastenfeld hat diese Belegung:

*) des Autors, nicht des Übersetzers



Die voranstehende Zusammenstellung von Tastenzuweisungen läßt sich leicht merken, und der Wechsel in den USER-Modus und zurück beim Schreiben synthetischer Programme bleibt auf das Notwendigste beschränkt.

Vier freie Tasten in Zeile 1 lassen Platz für zeitweilige Zuweisungen von Programmen oder anderen HP-41-Befehlen.

In Zeile 2 liegt ein Programm mit dem Namen "EFT", das Ihnen in der Aufgabe 4.5 begegnen wird. Dieses Programm gestattet es, die Funktionen des X-Funktionen-Moduls und des Time-Moduls auf einfachste Weise über das Tastenfeld auszuführen, nämlich durch Aufruf über eine Schlüsselnummer.

Die Funktion 'eGØBEEP' ist eine synthetische Ein-Byte-Tastenzuweisung, die von Robert Edelen entdeckt wurde. Man erlangt sie mit '0, ENTER↑, 167, ENTER↑, Tastenkode, R/S'. Wenn Sie die so belegte Taste drücken, erscheint 'eGØBEEP__' in der Anzeige. Beantwortet man die durch die Unterstreichung nahegelegte Aufforderung zur Parametereingabe mit dem Eintasten zweier Ziffern kl, erhält man bequemen Zugang zu den Funktionen des Druckers und des HP-IL-Moduls. Und zwar ergibt kl = 00 bis 63 die Funktionen 'XROM 28,kl', welche die Funktionen des Kassettenlaufwerkes und diejenigen zur Kontrolle der HP-IL-Schleife einschließen, während kl = 64 bis 99 die Funktionen 'XROM 29,kl−64', unter denen sich sämtliche Befehle des Druckers befinden, liefert. Beispielsweise ist 'XROM 29,12' = 'PRKEYS', also ergibt 'eGØBEEP 76' genau diese Funktion. Die Drucker-Funktion 'PRP' –

sie wird durch 'eGØBEEP 77' erreicht – erfordert eine Alpha-Eingabe. Wenn Sie aber 'eGØBEEP 77' eintasten, werden Sie dazu nicht aufgefordert. Stattdessen wird als Eintrag der Inhalt des Zustandsregisters Q gewählt, und zwar in Umkehrung der darin enthaltenen Byte-Folge, so wie es bei der Tätigkeit des sogenannten Q-Boten, den wir gleich kennenlernen werden, geschieht. Besitzer des Druckers 82143A seien vor 'eGØBEEP 89' (auf dem HP-IL-Drucker erhält man damit 'FMT') gewarnt: ein GAU ist die Folge!

Mit den Tastenzuweisungen "EFT" und 'eGØBEEP' läßt sich bemerkenswert viel Zeit und Arbeit beim Eintasten sparen, wenn man nur erst die Schlüsselzahlen für die am häufigsten benutzten Funktionen auswendig kann. Eine vollständige Liste der Schlüsselzahlen für die über "EFT" und 'eGØBEEP' erreichbaren Funktionen finden Sie am Ende dieses Kapitels im Anschluß an Aufgabe 4.5.

In Zeile 2 liegt außerdem unser Byte-Schnapper, dessen Zuweisung über "MK" mit den Dezimalwerten 247 und 63 erreicht wird. In Zeile 6 ist das Programm TI aus dem PPC ROM greifbar. Es besteht aus einer kurzen Folge synthetischer Töne und bietet eine vergnügliche Alternative zum gewöhnlichen 'BEEP' auf Kosten eines zusätzlichen Programm-Bytes. 'XTOA' ist eine Zuweisung aus dem X-Funktionen-Modul, deren Nützlichkeit sich im nächsten Kapitel zeigen wird.

4B. 'Des kleinen Mannes Byte-Lader'

Die beiden letzten Tastenzuweisungen auf dem eben vorgestellten Tastenfeld, die Zuweisungen 'STO Q' und 'Q-Bote', bedürfen noch einiger Erläuterung. Zusammen mit einem der verschiedenen 'Byte-Sammler' bilden sie einen 'Byte-Lader für arme Leute'. Weisen Sie sie mit "MK" Tasten zu, indem Sie 145, 121 für 'STO Q' und 27, 0 für den Q-Boten verwenden. Außerdem müssen Sie den Byte-Schnapper und 'RCL M' auf Tasten zur Verfügung haben.

Wenn Sie glücklicher Besitzer eines X-Funktionen-Moduls sind, können Sie sich der Funktion 'XTOA' als Byte-Sammler bedienen. Haben Sie einen PPC ROM, verwenden Sie DC dafür: 'XTOA' und DC nehmen beide die im X-Register liegende Zahl (zwischen 0 und 255), verwandeln sie in das entspechende Byte und fügen dieses dann dem im Alpha-Register vorhandenen Text rechts an, so daß es zum letzten Byte des Registers M wird. Können Sie weder auf 'XTOA' noch auf den PPC ROM zugreifen, müssen Sie sich folgende kleine Routine, die ebenfalls als Byte-Sammler arbeitet, schreiben:

```
01♦LBL "DC"     08 FS?C 18      15 SF 16        22 X<> \
02 OCT          09 SF 19        16 X<> d        23 STO [
03  E4          10 FS?C 17      17 X<> [        24 RDN
04 +            11 SF 18        18 "┣**"        25 END
05 X<> d        12 FS? 15       19 STO \
06 FS?C 19      13 SF 17        20 "┣*"
07 SF 20        14 FS? 14       21 CLX
```

54 BYTES

"LB"-Eingaben:

Zeile 03:	27, 20	Zeile 05:	206, 126	Zeile 16:	206, 126
Zeile 17:	206, 117	Zeile 19:	145, 118	Zeile 22:	206, 118
Zeile 23:	145, 117				

Nebenbei bemerkt ist dies genau der Byte-Sammler, den Clifford Stern für seine Programme "MK" und "LB" schrieb.

Weisen Sie nun 'XTOA' oder DC oder "DC" mit 'ASN' einer passenden Taste zu. Danach sind wir startbereit. Der Q-Bote überbringt Text in folgender Weise: Er holt eine Folge von bis zu 7 Zeichen aus dem Zustandsregister Q ab und legt diese Zeichen in einem Programm, welches ediert wird, als Textzeile nieder, und zwar in umgekehrter Reihenfolge zu der, in der er sie in Q vorgefunden hat. Will man z.B. den Text "HP'S #1" erzeugen, muß man zunächst die Zeichenfolge "1# S'PH" ins Alpha-Register setzen, dann 'RCL M', danach 'STO Q' ausführen und schließlich den Q-Boten mit der Überbringung beauftragen. Hier das Beispiel:

```
CLA
49 XTOA
35 XTOA      (oder DC oder "DC"; die
32 XTOA     nicht-synthetischen Zeichen
83 XTOA     können natürlich in dersel-
39 XTOA     ben Arbeitsweise wie die syn-
80 XTOA     thetischen angehängt werden)
72 XTOA
```

Jetzt haben Sie den Text "1# S'PH" im Alpha-Register liegen. Suchen Sie nun einen geeigneten Platz im Programmspeicher, um dort die Textzeile einzufügen. Wenn Sie keines der vorhandenen Programme damit behelligen wollen, reicht 'GTO ..', womit Sie ans Ende des belegten Programmspeichers gelangen. Sobald Sie sich an der wie auch immer ausgewählten Stelle befinden, kehren Sie zurück in den RUN-Modus und benutzen die tastenzugewiesenen Funktionen 'RCL M' und 'STO Q'. Danach schalten Sie zurück in den PRGM-Modus, um den Q-Boten aufzurufen. Sie erblicken als Ergebnis zunächst den synthetischen Zifferneintrag 'E', der vom Dezimalwert 27, mit dem die Zuweisung des Q-Boten getätigt wurde, herrührt. Mit 'SST' gelangen Sie dann auf die beabsichtigte Textzeile "HP'S #1". Wenn Sie jetzt den Q-Boten abermals aufrufen, erzeugen Sie im Programm die synthetischen Zeilen 'E' und 'TEXT 0'. Der Q-Bote 'überbringt' also im wörtlichen Sinne den Text aus Q: Q wurde nach dem ersten 'Botengang' leer zurückgelassen. Bei leerem Q kann der Bote nichts mehr überbringen; es entsteht somit nur der 'TEXT 0'-Befehl. Der Q-Bote kann folglich als 'Bote mit leeren Händen' dazu benutzt werden, auf schnelle und bequeme Weise synthetische Zifferneinträge 'E' und synthetische 'NOP'-Befehle zu erzeugen.

Doch die wahre Kraft des Q-Boten erweist sich erst dann, wenn man ihn vereint mit dem Byte-Schnapper einsetzt: Zunächst benutzen Sie den Q-Boten, um eine Textzeile von 7 Bytes

zu erzeugen, alsdann lassen Sie den Byte-Schnapper deren Führungsbyte ergreifen. Im Ergebnis bleiben die Zeichen der Textzeile als eigenständige Befehle im Programmspeicher zurück. Das folgende verhältnismäßig lange Beispiel soll Ihnen die nicht zu verachtende Kraft dieses 'Byte-Laders für die Armen' vor Augen führen. Folgen Sie dem Beispiel mehrfach mit Sorgfalt, bis Ihnen die Vorgänge im Programmspeicher völlig klar sind.

In unserem Beispiel wollen wir die zur Lösung der Aufgabe 2.7 erforderlichen synthetischen Zeilen der Routine "CMOD" erzeugen. Die vier Befehle lauten 'STO M', 'ST – M', 'ST/ M' und 'X< >M'. Die zugehörigen Dezimalwerte sind 145, 117, 147, 117, 149, 117, 206 und 117. Wegen der später durch den Q-Boten vorgenommenen Umkehrung der Bytes beginnen wir notwendigerweise von hinten:

```
CLA
117 XTOA
206 XTOA
117 XTOA
149 XTOA
117 XTOA
147 XTOA
117 XTOA
```

Die erste Gruppe von 7 Bytes liegt nun bereit und kann in den Programmspeicher geladen werden: 'GTO ..', dann 'LBL "CMOD"', um gleich die passende globale Marke zu setzen; Übergang in den RUN-Modus, 'RCL M', 'STO Q'; Übergang in den PRGM-Modus, Q-Boten aufrufen. Sie sehen daraufhin das schon bekannte 'E'. Drücken Sie jetzt nicht etwa voreilig 'SST', sondern stattdessen gleich BS. Damit schnappen Sie nämlich sofort das 'TEXT 7'-Byte des vom Q-Boten überbrachten Textes weg. Dann zweimal Korrekturtaste, was die durch BS entstandene Textzeile und das überflüssige 'E' löscht. Als Ergebnis haben Sie:

```
01♦LBL "CMOD"
02 RDN
03 ST- [
04 ST/ [
05 X<> [
06 .END.
```

Es verbleibt noch, das fehlende 'STO'-Byte 145 zu laden. Tasten Sie dazu

```
CLA
145 XTOA
```

ein; dann 'GTO "CMOD"', 'RCL M', 'STO Q'; Übergang in den PRGM-Modus und Q-Bote. Jetzt wird ein 'PACK' nötig, um die unsichtbaren 'NULL'-Bytes, welche zwischen den vom Q-Boten überbrachten Text der Länge 1 und die zuvor erzeugten Programmzeilen gelangten, zu beseitigen. Abschließend BS auf der Zeile 'E' und zweimal Korrekturtaste. Wenn Sie nun einzelschrittweise durch das Programm wandern, müssen Sie

```
01♦LBL "CMOD"
02 STO [
03 ST- [
04 ST/ [
05 X<> [
06 .END.
```

erblicken. Das 'STO'-Byte war in der Textzeile enthalten. Nachdem es diese durch die Wirkung des Byte-Schnappers verlassen mußte, ergriff es sich das 'RDN'-Byte, das somit zur Nachsilbe M wurde.

Sobald Sie ein wenig Übung erlangt haben, werden Sie bestimmt feststellen, daß dieser 'Byte-Lader der Armen' sehr gut dazu dienen kann, mit geringstem Aufwand schnell synthetische Befehle jeder Art zu erzeugen. Die ganze Ausrüstung besteht in den Zuweisungen von 'RCL M', von 'STO Q', vom Q-Boten und von BS, unterstützt von einem Byte-Sammler, der durch den X-Funktionen-Modul, das PPC ROM oder das Programm "DC" bereitgestellt wird, sowie der unentbehrlichen QRC.

Es erweist sich übrigens i.a. als sinnvoll, nur zusammenhängende synthetische Befehle zu erzeugen und nicht Teile davon, wie wir es in dem eben gezeigten Beispiel vorgeführt haben. Es wäre besser gewesen, beim sechsten Byte zu unterbrechen, um zunächst drei vollständige Befehle zu laden, und alsdann die verbleibenden zwei Bytes zusammen in einem zweiten Arbeitsgang zu erzeugen. Auf diese Weise umgeht man nämlich das u.U. zeitraubende zwischenzeitliche 'PACK'en. Wir haben auf den 'PACK'-Vorgang hier nur deswegen nicht verzichtet, um klärend vorführen zu können, was im Programmspeicher vorgeht, wenn synthetische Befehle von mehr als 7 Bytes Länge erzeugt werden müssen.

Die einzige Beschränkung, der man beim Byte-Laden mit dem Q-Boten begegnet, resultiert aus der Tatsache, daß führende 'NULL'-Bytes (gemeint ist führend im Register Q) unterdrückt werden. Wenn Sie z.B. den Befehl F2 7F 00 (eine 'NULL' anhängen) erzeugen wollen, müssen Sie sich eines vorübergehenden zusätzlichen Schlußbytes, etwa 'ENTER↑', bedienen. Dann läuft das Verfahren so: 'CLA', 131 (= 'ENTER↑'), 'XTOA', 0 (= 'NULL'), 'XTOA', 127 (Anhangsbyte), 'XTOA', 242 ('TEXT 2'-Byte), 'XTOA', Programmstelle aufsuchen, RUN-Modus, 'RCL M', 'STO Q', PRGM-Modus, Q-Bote, BS, zweimal Korrekturtaste; schließlich noch das jetzt überflüssig gewordene 'ENTER↑' hinter dem neu erzeugten "⊢⁻" löschen. (Bis zu 3 'NULL'en können Sie übrigens auch ohne das den Schluß sichernde 'ENTER↑' anhängen, weil beim Aufruf des Q-Boten 'NULL'-Bytes eingeschoben werden – vgl. dazu Kapitel 6 – so daß genügend davon zur Verfügung stehen, um vom freigestellten 'TEXT'-Byte eingefangen zu werden, wenn man den Byte-Schnapper betätigt. Sicherer arbeitet man aber mit dem beschriebenen Verfahren.)

Weitere Beschreibungen des Einsatzes von Q-Boten dieser und anderer Art finden Sie auf Seite 27 der Ausgabe Oktober '80 des PPC CJ.

4C. Das Vorweisen von Pseudo-XROM-Zahlen

Die einzigen Zwei-Byte-Funktionen die man nicht-synthetisch auf Tasten legen kann, sind Peripherie-Funktionen. Wenn das jeweilige Peripherie-Gerät nicht eingesteckt ist, erscheint die Funktion bei niedergedrückter Taste als 'XROM i,j', wobei i und j zweiziffrige Zahlen zwischen 00 und 63 sind. Die Bezeichnung XROM rührt von '*e*xternal *ROM*' (*r*ead-*o*nly *m*emory) her. Die Zahl i *id*entifiziert das Peripherie-Gerät; i heißt deshalb auch ROM-ID-Zahl. Manche Peripherie-Geräte enthalten zwei ROMs zu je 4 Kilobyte; in diesem Fall hat jedes ROM seine eigene ROM-ID-Zahl. Die Zahl j ist die Nummer der jeweiligen Funktion innerhalb eines 4K ROMs in 'CAT 2'-Ordnung.

Sobald eine Taste, der eine synthetische Zwei-Byte-Funktion zugewiesen ist, gedrückt wird, nimmt der HP-41 für den Zweck der Funktionsanzeige grundsätzlich an, es handele sich um eine normale (aus einem Peripherie-Gerät stammende) XROM-Funktion. Sind dabei x und y die Dezimalwerte, mit denen die Zuweisung einer synthetischen Funktion vorgenommen wurde, und sind i und j die XROM-Zahlen, mit denen diese Funktion dann in der Anzeige vorgewiesen wird, so gelten zwischen ihnen folgende Beziehungen:

```
i = 4(x mod 16) + INT(y/64)
j = y mod 64
```

Hierin bedeutet mod die HP-41 Funktion 'MOD'. Z.B. erscheint 'ST+ IND M' ≙ 146, 245 als 'XROM 11, 53', während 'TONE Y' ≙ 159, 114 als 'XROM 61, 50' erscheint. Diese Wechselbeziehung ist auch auf der QRC zu erkennen: Die Spaltennummer des ersten Bytes x ist x mod 16 und liegt somit zwischen 0 und 15. Weil INT(y/64) nur die vier Werte 0, 1, 2 und 3 annehmen kann, fällt also i in eine der 15 Vierer-Gruppen 0 – 3, 4 – 7, ... , 60 – 63, von denen die ersten 8 auf der QRC in Zeile A/Spalten 0 – 7 notiert sind. XR 8 – 11 in Spalte 2 der Zeile A zeigt an, daß die erste der beiden XROM-Zahlen, die aus einem Dezimalwert x hervorgeht, welcher aus Spalte 2 der QRC stammt, 8, 9, 10 oder 11 lautet. Der genaue Wert von i wird durch y bestimmt, und zwar durch den 4er-Block von Zeilen, in welchem y liegt. Als optische Hilfe beim Blick in die QRC sind dort diese 4er-Blöcke durch fettgesetzte Horizontallinien unterteilt. Die Zeilen 0 bis 3 liefern den ersten der jeweils 4 möglichen i-Werte, die Zeilen 4 bis 7 den zweiten, 8 bis B den dritten und C bis F den vierten. Damit sind Sie in der Lage, der QRC unmittelbar den Wert INT(y/64) zu entnehmen. Genauso einfach läßt sich j = y mod 64 ablesen: Sie brauchen nur das dem Wert y entsprechende Kästchen im ersten 4er-Block aufzusuchen, z.B. j = 55 für y = 183.

Wir wollen rasch noch mit den oben gewonnenen Erkenntnissen das Beispiel 'ST+ IND M' untersuchen: Weil die Vorsilbe 'ST+' in Spalte 2 der QRC und die Nachsilbe 'IND M' im vierten 4er-Block der Zeilen der QRC liegt, hat i den Wert 11. Ein weiterer Blick weist uns von 'IND M', also Zeile F/Spalte 5, hinauf in das entsprechende Kästchen des ersten 4er-Blockes

nach Zeile 3/Spalte 5. Dort finden wir 53. Mithin wird 'ST+ IND M' in der Anzeige als (Pseudo-)'XROM 11,53' vorgewiesen.

Die vorgewiesene XROM-Zahl enthüllt einiges über die zugewiesene synthetische Funktion, bestimmt sie jedoch nicht eindeutig. Z.B. wird die Zuweisung 'DSE IND 10' als 'XROM 30,10' angezeigt; diese XROM-Zahl gehört aber gleichzeitig zur Kartenleser-Funktion 'WSTS'. Die Zuweisung von 'WSTS' ist also von der Zuweisung 'DSE IND 10' nicht zu unterscheiden, solange die Taste, welche die Zuweisung trägt, nicht (innerhalb der Wartezeit) losgelassen wird. Wenn Sie einmal über die Identität einer Zuweisung im Zweifel sind, sollten Sie die Taste mit der zweifelhaften Zuweisung *im PRGM-Modus* betätigen. Doch darf es sich bei diesem 'Tasten auf Probe' nicht um den Byte-Schnapper handeln, wenn Sie sich in der Nähe eines 'END' oder des '.END.' befinden. Beachten Sie die in Kapitel 1 ausgesprochene Warnung.

Weitere Einzelheiten über XROM-Zahlen finden Sie auf Seite 47 der Ausgabe März '81 des PPC CJ. Auf Seite 45 der Ausgabe August '81 beginnt ein von Roger Hill beigesteuerter hochinteressanter Artikel, der beschreibt, wie sich die verschiedensten synthetisch hergestellten Tastenzuweisungen, die in Pseudo-XROM-Gestalt erscheinen, im PRGM-Modus verhalten.

4D. Die Zuweisung von 'RCL b'

Unter den zuweisungsfähigen synthetischen Funktionen spielt 'RCL b' eine besondere Rolle. Im Gegensatz zu allen anderen synthetischen Funktionen nämlich, bei denen kein wesentlicher Unterschied zwischen dem Aufruf über eine Taste oder der Ausführung durch ein Programm, dem die Funktionen eingefügt wurden, besteht, ist die Tastenfeldausführung von 'RCL b' wesentlich nützlicher als die Ausführung durch ein Programm. Denn über das Tastenfeld ausgeführt setzt 'RCL b' den *gegenwärtigen* Wert des Adreßzeigers ins X-Register, auf welche Stelle im Programmspeicher der Adreßzeiger auch immer weisen mag, wohingegen ein Programmbefehl 'RCL b' stets denselben Adreßzeigerwert liefert, nämlich den zum Programmbefehl 'RCL b' selbst gehörigen Wert.

Das Ergebnis eines 'RCL b' ist der Wert des Adreßzeigers, verschlüsselt in den letzten beiden Bytes von X, intern dargestellt durch vier Hexadezimalziffern. In verschlüsselter Form ist der Wert des Adreßzeigers allerdings recht unhandlich. Wir besprechen daher hier zwei Routinen, die den Adreßzeigerwert in eine unmittelbar verständliche Dezimalzahl umrechnen. Zwei weitere Routinen eröffnen dann einen bequemen Weg, die Anzahl der Bytes zwischen zwei beliebigen Stellen im Programmspeicher zu bestimmen.

Die Routine "RAMBYT" leistet genau dasselbe wie die PPC ROM Routine PD. Um "RAMBYT" anzuwenden, begebe man sich an einen beliebigen Punkt des Programmspeichers und rufe 'RCL b' im RUN-Modus über die Taste, der Sie diese Funktion zugewiesen haben, auf. Das Ergebnis ist der Adreßzeigerwert für die aufgesuchte Prorammspeicherstelle. Mit 'XEQ "RAMBYT"' (oder 'XEQ PD') rechnet man nun diesen in einen Dezimalwert um.

Die Routine "ROMBYT" arbeitet ganz ähnlich wie "RAMBYT" mit dem Unterschied, daß sie in X den Adreßzeigerwert einer Stelle aus einem ROM erwartet. Wenn Sie einen PPC ROM oder einen beliebigen anderen Anwender-Modul haben, können Sie "ROMBYT" ausprobieren. Sie brauchen dazu nur an eine Marke oder eine beliebige andere Stelle des ROMs zu gehen, 'RCL b' vom Tastenfeld aus im RUN-Modus auszuführen und dann mit 'XEQ "ROMBYT"' das Ergebnis von 'RCL b' in die Dezimalzahl, die der ausgewählten Stelle im ROM entspricht, umzurechnen.

Die gebräuchlichste Anwendung von "RAMBYT" besteht darin, die zwischen zwei Programmspeicherstellen liegende Anzahl von Bytes zu zählen. Beispielsweise läßt sich so die Gesamtanzahl der Bytes eines Programms ermitteln. Das Programm "RAMBC" führt diese Aufgabe selbständig durch. Es wendet "RAMBYT" auf die beiden Adreßzeigerwerte, die zu zwei Programmstellen gehören, an und bildet dann die Differenz zwischen den beiden durch Entschlüsselung entstandenen Dezimalwerten. "RAMBC" entspricht in seiner Arbeitsweise der Routine CB (*c*ount *b*ytes) aus dem PPC ROM.

Um die Leistung von "RAMBC" vorzuführen, wollen wir ermitteln, wieviel Bytes in der Gruppe der vier Routinen "RAMBC", "RAMBYT", "ROMBC" und "ROMBYT" enthalten sind. Dazu ist zunächst ein 'PACK' nötig, falls dies noch nicht geschehen sein sollte. Dann 'GTO "RAMBC"', 'RCL b' im RUN-Modus, 'BST' (auf das 'END'), abermals 'RCL b' im RUN-Modus und schließlich 'XEQ "RAMBC"'. Das Ergebnis muß 156 lauten, wodurch angezeigt wird, daß zwischen dem Programmanfang und dem Anfang des 'END' genau 156 Bytes liegen. Wollen Sie das 'END' in die Zählung einbringen, müssen Sie noch 3 Bytes dazuschlagen, so daß Sie auf eine Gesamtanzahl von 159 Bytes kommen. Lautet die letzte Zeile der Programmgruppe '.END.', kann die Zählung bis zu 6 Bytes mehr ergeben. In diesem Fall erreicht man die wahre Byte-Anzahl dadurch, daß man dem Verfahren ein 'GTO ..' vorausschickt.

Wenn man die Anzahl der durch ein Programm belegten Bytes durch 112 dividiert, findet man die Anzahl der Spuren, die man benötigt, um das Programm auf Magnetkarten aufzuzeichnen. Das 'END' wird zwar auf Karten mit aufgezeichnet, doch wenn man ein Programm vorliegen hat, das ohne das 'END' genau 112 Bytes umfaßt, kann man die zweite Spur einsparen; sie enthielte nämlich nur das 'informationslose' 'END'. In diesem Fall beantwortet man sowohl beim Aufzeichnen als auch beim Einlesen des Programms die Anforderung der zweiten Spur mit dem Drücken der Korrekturtaste.

Eine Anwendung von "RAMBC" für Fortgeschrittene ist die, festzustellen, ob Langform-'GTO's (Drei-Byte-'GTO's) erforderlich sind oder Kurzform-'GTO's (Zwei-Byte-'GTO's) ausreichen (s. Kapitel 3). Kurzform-'GTO's ('GTO 00' bis 'GTO 14') sollten nur dann benutzt werden, wenn der Sprungabstand weniger als 112 Bytes beträgt. In diesen Fällen wird nämlich der Abstand berechnet und das Ergebnis der Berechnung im Sprungbefehl notiert, wenn der Sprungbefehl das erste Mal ausgeführt wird (der ganze Vorgang heißt Kompilation). Nachfolgende Sprungausführungen sind dann wesentlich schneller, weil die Suche nach der angesprungenen Marke entfällt. Nur in Langform-'GTO's können Sprungabstände von mehr als 112 Bytes gespeichert werden; Kurzform-'GTO's hingegen müssen bei zu großem Sprungabstand stets von neuem nach der Marke suchen, was den Programmlauf bei mehrfach auszuführenden Sprüngen ganz erheblich verlängert.

Um festzustellen, ob ein Zwei-Byte-'GTO' und sein Partner, das zugehörige Ein-Byte-'LBL', ohne Verzicht auf den Vorteil der kompilierten Verzweigung verwendet werden können, setzt man zunächst 'GTO mn' und 'LBL mn' ($00 \leq mn \leq 14$) an die vorgesehenen Stellen im Programm. Danach 'PACK't man zur Beseitigung überflüssiger 'NULL'en. Dann geht man auf die dem 'GTO mn' folgende Zeile (sollte diese ausnahmsweise das '.END.' sein, wird vorübergehend noch ein beliebiger Befehl dazwischengesetzt und abermals ge'PACK't) und ruft 'RCL b' im RUN-Modus auf. Anschließend geht man auf das zugehörige 'LBL mn' (dazu lassen sich bequem 'BST' und 'SST' verwenden) und ruft ein zweites Mal 'RCL b' auf. Nach einem 'XEQ "RAMBC"' hat man dann den gesuchten Sprungabstand in Bytes im Register X stehen. Liegt diese Zahl zwischen -111 und $+111$, beide Werte eingeschlossen, reicht das Zwei-Byte-'GTO' aus. Andernfalls muß ein Drei-Byte-'GTO' her.

Eine zweite einfachere Möglichkeit ist die, 'RCL b' direkt auf dem 'GTO mn' aufzurufen, dann unmittelbar mit 'SST' nach 'LBL mn' zu springen, dort das zweite 'RCL b' auszuführen und dann mit 'XEQ "RAMBC"' den Sprungabstand zu ermitteln. Bei diesem Verfahren muß das Ergebnis zwischen −109 und +113, beide Werte eingeschlossen, liegen, um noch mit den Kurzformen arbeiten zu können.

Wenn man der großen Sprunglänge wegen Drei-Byte-'GTO's benötigt, hat man noch die Möglichkeit, sich synthetisch erzeugter 'GTO's zu bedienen, indem man für "LB" die Eingaben 208, 0, mn (00 ≤ mn ≤ 14) verwendet. Oder man tastet einfach die Folge 'STO IND 80', 'ISG mn', 'BST', 'BST', BS, '←' ein, die ebenfalls ein Drei-Byte-'GTO' auf ein Kurzform-'LBL' liefert (statt 'ISG' kann auch eine beliebige andere Zwei-Byte-Funktion, die eine zweistellige Parametereingabe 00 bis 14 zuläßt, gewählt werden; etwa 'SF'). Auf diese Weise spart man ein Byte ein und braucht auch bei längeren Sprüngen nicht auf die Langformen 'LBL 15' bis 'LBL 99' zurückzugreifen. Ein einmal erzeugtes synthetisches Drei-Byte-'GTO' behält seine Bauart und verwandelt sich nicht etwa nachträglich in ein Zwei-Byte-'GTO'; es bewahrt vielmehr treu wie die normalen Drei-Byte-'GTO's die Information über einmal ausgeführte Sprünge beliebiger Länge auf. Von Zwei-Byte-'GTO's kann man es nur unterscheiden, wenn man seine Bytes mit "RAMBC" auszählt.

Hier nun die Programm-Auflistung von "RAMBC", "RAMBYT", "ROMBC" und "ROMBYT". "ROMBC" arbeitet genau wie "RAMBC", allerdings nur mit Adreßzeigerwerten aus ROMs.

```
01*LBL "RAMBC"
02 X<>Y
03 XEQ 01
04 X<>Y
05 XEQ 01
06 -
07 RTN
08*LBL "RAMBYT"
09*LBL 01
10 XEQ 03
11  E41
12 /
13 INT
14 LASTX
15 FRC
16  E4
17 *
18 DEC
19 7
20 *
21 +
22 RTN
23*LBL "ROMBC"
24 XEQ 02
25 X<>Y
26 XEQ 02
27 -
28 RTN
29*LBL "ROMBYT"
30*LBL 02
31 XEQ 03
32  E37
33 /
34 DEC
35 RTN
36*LBL 03
37 "*"
38 X<> [
39 STO \
40 ASHF
41 "⊢+++A"
42 X<> [
43 X<> d
44 CF 08
45 FS?C 09
46 SF 05
47 FS?C 10
48 SF 06
49 FS?C 11
50 SF 07
51 FS?C 12
52 SF 09
53 FS?C 13
54 SF 10
55 FS?C 14
56 SF 11
57 FS?C 15
58 SF 13
59 FS?C 16
60 SF 14
61 FS?C 17
62 SF 15
63 FS?C 18
64 SF 17
65 FS?C 19
66 SF 18
67 FS?C 20
68 SF 19
69 X<> d
70 END
```

159 BYTES

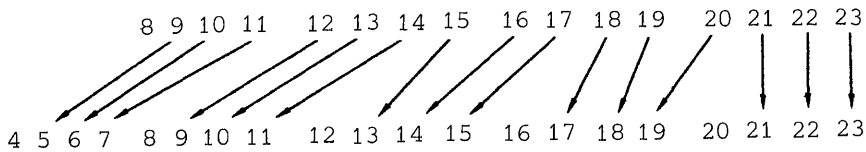
"LB"-Eingaben:

```
Zeile 11:  27, 20, 17, 0*) Zeile 16: 27, 20, 0*)
Zeile 32:  27, 19, 23  Zeile 38: 206, 117     Zeile 39: 145, 118
Zeile 41: 245, 127, 0, 0, 0, 65
Zeile 42: 206, 117,    Zeile 43:  206, 126    Zeile 69:  206, 126
```

*) Trenn-'NULL'en (vgl. Abschnitt 2B)

Der entscheidende Teil in dieser Gruppe von Routinen ist das unter 'LBL 03' stehende Unterprogramm, welches zwei Tricks aus der 'höheren' synthetischen Programmierung verwendet. In den ersten vier Schritten werden dort die letzten beiden Bytes des X-Registers abgetrennt und ins Alpha-Register gebracht. Diese Bytes werden dann in Zeile 41 nach links verschoben und anschließend ins Flag-Register übertragen. Als Ergebnis von Zeile 43 spiegeln die Flags 8 bis 23 den Zustand der 16 Bits des Adreßzeigers (das äußerste linke Bit, dem Flag 8 entsprechend, wird hier übrigens nicht benötigt) wieder. Die dem Benutzer zur Verfügung stehenden Befehle zur Umschaltung von Flags werden nun dazu verwendet, nach einem listigen Verfahren die Bits so einzustellen, daß eine Oktaldarstellung (Basis 8) mit drei Bits je Oktalziffer entsteht (vgl. untenstehende Abbildung). Als Ergebnis dieses Tricks ergeben sich in den Flags 4 bis 23 fünf Oktalziffern, deren jede eine Gruppe von 4 Flags umfaßt. Die ersten Bits dieser fünf Gruppen, die Bits 4, 8, 12, 16 und 20, stehen dabei auf 0.

In Zeile 69 wird die auf diese Weise erzeugte 5-ziffrige Oktalzahl in der Form $a.bcde \times 10^{41}$ ins X-Register gebracht *). Mit normalen arithmetischen Befehlen kann man dann die Ziffern nach Bedarf bearbeiten und schließlich mit der Funktion 'DEC' in gewöhnliche Dezimalziffern umrechnen. Der Trick, die Bits einfach umzuschalten, um eine Oktaldarstellung zu erhalten, stammt von Roger Hill, dem Autor vieler Routinen des PPC ROMs.



Um die Arbeit der vorstehenden Routinen "RAMBYT" und "ROMBYT" an den Oktalziffern vollständig zu verstehen, müssen Sie noch die Erläuterungen zum Aufbau des Adreßzeigers in Kapitel 6 lesen.

*) Anm. des Übers.: Man erzielt eine leichte Verbesserung, wenn Zeile 11 in 'E51', Zeile 32 in 'E47' und Zeile 41 in "⊦⁻⁻⁻Q" abgeändert wird, dann läßt sich die Routine nämlich ohne zerstörerische Wirkung bezüglich Flag 51 ('SST') auch einzelschrittweise abarbeiten.

4E. Abspeichern und Rückrufen von Weckaufträgen

Die meisten Tastenzuweisungsprogramme (ausgenommen "MKX" aus Aufgabe 4.4) haben eine mißliche Eigenschaft gemeinsam: sie arbeiten nicht mehr einwandfrei, sobald Weckaufträge vorhanden sind, und überdies zerstören sie diese. Als Lösung bietet sich zunächst einmal an, die Weckaufträge zuvor von Hand oder über 'ALMCAT' zu löschen. Das ist natürlich lästig und erzwingt das ausführliche Aufzeichnen und anschließende Rückschreiben der Weckaufträge.

Wenn Sie sowohl den X-Funktionen-Modul als auch den PPC ROM besitzen, können Sie stattdessen Clifford Stern's "SA" (*s*ave *a*larms) und "RA" (*r*ecall *a*larms) einsetzen, um die Weckaufträge programmgesteuert vorübergehend im X-Memory abzulegen und nach getätigter Zuweisung zurück in den Hauptspeicher zu holen. "SA" verwendet die X-Funktion 'SAVERX', welche im Gegensatz zu 'RCL' den Abruf von Daten aus dem Hauptspeicher ohne Normalisierung (vgl. Abschnitt 2C) vornimmt. Tatsächlich werden zwar das erste und das letzte Register des Blockes der Alarmaufträge normalisiert, doch macht "RA" diesen Schaden dann wieder gut.

Hier die Gebrauchsanweisung für "SA" und "RA":

1. Stellen Sie zunächst sicher, daß sich wenigstens ein 'END' oberhalb von 'LBL "SA"' im Katalog 1 befindet. Dies ist nötig, weil sonst das rückwärts gerichtete 'GTO' in Zeile 66 bei 'herabgelassenem Vorhang' (dieser Begriff wird in Abschnitt 6C erklärt) fehlgeht.

2. Nachdem dies geschehen ist, führen Sie 'XEQ "SA"' aus, um die Weckaufträge in eine von "SA" selbst angelegte Datei namens "ALM" im X-Memory zu bringen und im Hauptspeicher zu löschen. Auf Zeile 86 können dabei mehrere Fehlermeldungen auftreten: "DATA ERROR" bedeutet, daß keine Weckaufträge zum Abspeichern vorliegen. Die Meldung "DUP FL" zeigt an, daß bereits eine Datei des Namens "ALM" im X-Memory vorhanden ist. Man kann dann 'PURFL' von Hand ausführen und anschließend das Programm mit 'R/S' zu Ende laufen lassen. "NO ROOM" weist darauf hin, daß nicht mehr genügend Platz im X-Memory zur Verfügung steht, um "ALM" anzulegen. Sie müssen dann vom Zustand Ihres Rechners abhängige Entscheidungen treffen (Dateien im X-Memory löschen) und dürfen mit 'R/S' erst fortfahren, nachdem das Alpha-Register wieder mit dem Dateinamen "ALM" gefüllt worden ist.

3. Jetzt ist es möglich, ein beliebiges Tastenzuweisungsprogramm zu benutzen. Nachdem die synthetischen Zuweisungen vorgenommen worden sind, führen Sie 'XEQ "RA"' aus, um die Weckaufträge in den Hauptspeicher zurückzuholen und "ALM" im X-Memory zu löschen. "RA" bedient sich der X-Funktion 'PSIZE', sofern es nötig wird, unterhalb des '.END.' Platz zu schaffen, um die zurückzuholenden Weckaufträge unterzubringen. Falls dies fehlschlägt, weil nicht mehr genügend Register zur Verfügung stehen, bekommen Sie auf Zeile 15 die Fehlermeldung "DATA ERROR". In diesem Fall hilft 'PACK' und/oder Löschen eines entbehrlichen Programms und anschließender Neustart von "RA". Das Programm "RA" schließt mit dem Befehl 'OFF', wodurch Sie zwar gezwungen werden, den HP-41 wiedereinzuschalten. Doch hat dies den Vorteil, daß Sie auf inzwischen u.U. überfällig gewordene Weckaufträge aufmerksam gemacht werden. Wenn Sie nämlich "SA" ausführen

und dann den Rechner ausschalten, bevor Sie mit "RA" die Weckaufträge zurückgeholt haben, können Ihnen solche überfällig gewordenen Weckaufträge entgehen, weil die im Time-Modul vorhandene diesbezügliche Überwachungseinrichtung erfolglos bleiben muß, wenn sie beim Ausschalten des Rechners überhaupt keine Weckaufträge vorfindet. Der Befehl 'OFF' hingegen aktiviert sie sofort im Anschluß an "RA" und meldet Ihnen so, ob zwischenzeitlich Weckaufträge überfällig geworden sind. Ein 'ALMNOW' kann denselben Zweck erfüllen wie 'OFF'; wollen Sie also "RA" als Unterprogramm einsetzen, müssen Sie 'OFF' durch 'ALMNOW, RTN' ersetzen.

Hinweis: Leider dürfen Sie "SA" und "RA" nicht benutzen, wenn Ein-/Ausgabe-Puffer, wie sie der HP-IL-Development-Modul erzeugt, vorhanden sind.

Hier die Programm-Auflistung von Clifford Stern's "SA" und "RA" (den Barcode finden Sie in Anhang E, die angesprochenen PPC ROM-Routinen in Anhang F):

```
01*LBL "RA"
02 XROM "F?"
03 INT
04 XROM "E?"
05 X<>Y
06 -
07 SIZE?
08 ENTER↑
09 "ALM"
10 LASTX
11 +
12 FLSIZE
13 -
14 X<0?
15 SQRT
16 X<Y?
17 PSIZE
18 R↑
19 XROM "CX"
20 GETR
21 R↑
22 FLSIZE
23 ""
24 RCL [
25 "          "
26 STO \
27 ARCL 00
28 RCL [
29 STO 00
30 X<> \
31 DSE Z
32 STO IND Z
33 R↑
34 STO c
35 "ALM"
36 PURFL
37 BEEP
38 OFF
39*LBL "SA"
40 XROM "OM"
41 176
42 XROM "E?"
43 17
44 -
45 X<Y?
46 GTO 03
47  E3
48 /
49 +
50 SIGN
51 "0"
52 X<> [
53*LBL 01
54 ""
55 RCL IND L
56 X<> [
57 "├          "
58 X<> \
59 X≠Y?
60 GTO 02
61 ARCL c
62 X<> \
63 STO IND L
64 RDN
65 ISG L
66 GTO 01
67 CLA
68 GTO 03
69*LBL 02
70 ARCL IND L
71 X=0?
72 CLA
73 X<> [
74 STO IND L
75*LBL 03
76 ATOX
77 R↑
78 X<> c
79 LASTX
80 INT
81 ENTER↑
82 DSE X
83 ATOX
84 "ALM"
85 CF 25
86 CRFLD
87 +
88  E3
89 /
90 +
91 X<>Y
92 X<> c
93 X<>Y
94 SAVERX
95 XROM "BC"
96 X<>Y
97 STO c
98 BEEP
99 END
```

175 BYTES

"LB"-Eingaben:

Zeile 23: 241, 240*)	Zeile 24: 144, 117	Zeile 25: 241, 170*)
Zeile 26: 145, 118	Zeile 28: 144, 117	Zeile 30: 206, 118
Zeile 34: 145, 125	Zeile 47: 27, 19	Zeile 51: 241, 16
Zeile 52: 206, 117	Zeile 54: 241, 240*)	Zeile 56: 206, 117
Zeile 57: 242, 127, 170*)		
Zeile 58: 206, 118	Zeile 61: 155, 125	Zeile 62: 206, 118
Zeile 73: 206, 117	Zeile 78: 206, 125	Zeile 88: 27, 19
Zeile 92: 206, 125	Zeile 97: 145, 125	

*) Diese Zeilen enthalten ein unsichtbares Zeichen aus der zweiten Hälfte der QRC (Dezimalwerte 128 bis 255).

Beachten Sie auch, daß die Zeilen 25 und 57 das Byte AA (dezimal 170) enthalten. AA ist ein Kontrollbyte für den Drucker und veranlaßt ihn, bei der Programm-Auflistung 10 Zeichen zu überspringen. Einige Bemerkungen über Drucker-Kontrollbytes, die zu sehr merkwürdigem Verhalten bei Programm-Auflistungen führen können, wurden schon am Ende von Abschnitt 2E gemacht. Sämtliche auf der QRC farbunterlegten Bytes arbeiten als Drucker-Kontrollbytes.

Aufgaben:

4.1 Prüfen Sie die Lösungen zu den Aufgaben des Kapitels 2 auf die Möglichkeit hin, das Eintasten jener Programme mit Hilfe synthetischer Tastenzuweisungen zu beschleunigen.

4.2 Versuchen Sie, Clifford Stern's Programm "LB" einzutasten, indem Sie zunächst mit Hilfe des 'Byte-Laders für Habenichtse' die folgenden synthetischen Befehle erzeugen:

```
hexadezimal F4 7F 00 00 02
E4
X<> c
STO c
hexadezimal F2 7F 00
X<> c
STO c
```

Dann ergänzen Sie diese vermöge Ihres synthetischen Tastenfeldes; und schließlich vervollständigen Sie das Ganze zu "LB" durch Einfügen der normalen Befehle.

4.3 'Berechnen' Sie die XROM-Zahlen, die für die folgenden synthetischen Tastenzuweisungen in der Anzeige erscheinen müssen:

```
a) TONE 89
b) X<> P
c) ISG IND N
```

Bestätigen Sie Ihre 'Voraussage' durch tatsächliche Zuweisungen dieser Funktionen.

4.4 Hier ein weiteres Tastenzuweisungsprogramm. Es heißt "MKX", verwendet X-Funktionen und stammt von Tapani Tarvainen. Clifford Stern hat es durchgesehen und verkürzt. Der Grundgedanke ist ein anderer als bei "MK"; er macht Gebrauch von der X-Funktion 'PASN', mit der Tastenzuweisungen programmgesteuert vorgenommen werden können. "MKX" verwendet 'PASN', um der ausgewählten Taste zunächst eine 'Strohmann-Funktion' zuzuweisen. Dann wird diese Zuweisung in den Tastenzuweisungsregistern aufgesucht und durch die eigentlich beabsichtigte Zuweisung ersetzt. "MKX" unterscheidet sich so stark von "MK" und MK , daß eine gesonderte Gebrauchsanweisung vonnöten ist:

1) Stellen Sie sicher, daß der Katalog 1 keine Marke des Namens "ANUM" enthält und daß oberhalb von "MKX" ein 'END' vorhanden ist (letzteres läßt sich, falls sich "MKX" am Anfang des Programmspeichers befindet, einfach so bewerkstelligen: 'GTO .MKX, GTO .000, XEQ "END"'). Das Versäumnis, eine dieser Einschränkungen zu beachten, bevor Sie "MKX" ausführen, zwänge Sie zu einer Totallöschung. Die zum Programm "CU" (besprochen in Abschnitt 6C) gehörige Einschränkung 1 wird Ihnen später die Notwendigkeit des 'END' erklären. Die zweite Einschränkung stellt sicher, daß die 'Strohmann-Zuweisung' in Zeile 04 keine Programmzuweisung "ANUM", sondern die Zuweisung der *Funktion* 'ANUM' ist. Vgl. Abschnitt 6A.
2) Füllen Sie nun Z, Y und X mit den drei für Tastenzuweisungen erforderlichen Eingaben, und führen Sie dann "MKX" aus. Die Eingaben sind dieselben, die Sie für "MK" oder MK verwendet hätten, mit dem Unterschied, daß die beiden Dezimalwerte und der Tastenkode hier *vor* Aufruf des Programms in den Stapel geladen werden.
3) Weckaufträge brauchen nicht ausgelagert oder gelöscht zu werden. Sie werden auch nicht zerstört.
4) Falls nicht hinreichend viele freie Register zur Verfügung stehen, erhalten Sie auf Zeile 04 die Meldungen "PACKING" und "TRY AGAIN". "MKX" ist gewissermaßen 'nachsichtiger' als "MK".
5) "MKX" darf ebensowenig unterbrochen werden wie "MK"!
6) Wenn Sie sich anschicken, eine Taste zu belegen, die schon eine Zuweisung trägt, wird die alte Zuweisung durch die neue ersetzt, ohne daß eine diesbezügliche Warnung oder Ankündigung erfolgt. Wollen Sie solch ein Mißgeschick vermeiden, müssen Sie vor Aufruf von "MKX" die ins Auge gefaßte Taste überprüfen.
7) Für weitere Zuweisungen wird der Stapel mit den erforderlichen Eingaben geladen und "MKX" erneut aufgerufen. Es reicht aber auch 'R/S', weil "MKX" auf seinem 'END' ankommt und daher der Adreßzeiger nach Beendigung des Programmlaufs wieder auf den Programmanfang weist.
8) "MKX" vergeudet keine Registerhälften, weil jede neue Zuweisung folgerichtig behandelt und ein neues Zuweisungsregister nur dann eröffnet wird, wenn sich herausstellt, daß keine 'Löcher' mehr in den bestehenden Zuweisungsregistern 'klaffen'.

```
01*LBL "MKX"    14 "├***      B"    27 X<> [      40 STO ]
02 "ANUM"       15 X<> ]            28 "├*"       41 "├**"
03 CF 25        16 X<> [            29 STO \      42 X<> ]
04 PASN         17 STO \            30 "├***"     43 STO IND L
05 "▫i▮*"       18 "├"              31 X<> \      44 FC?C 25
06 RCL [        19 X<> ]            32 "├***"     45 ISG L
07 R↑           20 R↑               33 X≠Y?       46 X=Y?
08 XTOA         21 X<> c            34 X<> \      47 GTO 01
09 R↑           22 RCL \            35 X=Y?       48 R↑
10 XTOA         23 ,                36 SF 25      49 STO c
11 RCL '        24 SIGN             37 X=Y?       50 CLST
12 STO ]        25*LBL 01           38 R↑         51 END
13 X<> [        26 X<> IND L        39 "├****"                  123 BYTES
```

"LB"-Eingaben:

```
Zeile 05: 245, 1, 105, 12, 0, 240*)
Zeile 06: 144, 117      Zeile 11: 144, 122      Zeile 12: 145, 119
Zeile 13: 206, 117
Zeile 14: 247, 127, 0, 0, 0, 240*), 166*), 66
Zeile 15: 206, 119      Zeile 16: 206, 117      Zeile 17: 145, 118
Zeile 18: 242, 127, 240*)
Zeile 19: 206, 119      Zeile 21: 206, 125      Zeile 22: 144, 118
Zeile 27: 206, 117      Zeile 29: 145, 118      Zeile 31: 206, 118
Zeile 34: 206, 118
Zeile 39: 245, 127, 42, 42, 42, 0
Zeile 40: 145, 119
Zeile 41: 244, 127, 0, 0, 240*)
Zeile 42: 206, 119      Zeile 49: 145, 125
```

*) Zeichen aus der zweiten Hälfte der QRC, unsichtbar in Programm-Auflistungen, doch in der Anzeige als Vollzeichen vorgewiesen.

4.5 Falls Sie den schnellen Zugriff auf die HP-IL-Funktionen vermittels 'eGØBEEP' schätzen, werden Sie vielleicht auch Gefallen an der folgenden kurzen Routine, die bezüglich der X-Funktionen und der Befehle des Time-Moduls und des optischen Lesestiftes ähnlich wie 'eGØBEEP' arbeitet, finden. Man lädt, falls erforderlich, den Stapel, tastet 'ENTER↑', gibt einfach die Nummer der gewünschten Funktion ein und führt dann 'XEQ "EFT"' aus. "EFT" hält einen kurzen Moment an, um dem Benutzer Gelegenheit zu geben, einen (auf 7 Zeichen begrenzten!) Alpha-Text, z.B. einen Dateinamen, einzutasten. Sofern der Alpha-Text schon vorhanden oder ein solcher nicht notwendig ist, läßt man die Eingabe-Möglichkeit ungenutzt verstreichen. "EFT" stellt eine kurze Byte-Folge, die u.a. die ge-

wünschte XROM-Funktion enthält, her und führt diese dann aus. Die Byte-Folge liegt, während sie ausgeführt wird, in den Zustandsregistern b und a.
Zwei Einschränkungen sind bei der Benutzung von "EFT" zu beachten. Erstens: "EFT" arbeitet im Gegensatz zu 'eGØBEEP' leider nur im RUN-Modus, nicht im PRGM-Modus. Daher kann es nicht dazu benutzt werden, die X-Funktionen oder die des Time-Moduls in ein Programm einzufügen. Zweitens kann man einige X-Funktionen nicht einwandfrei mit Hilfe von "EFT" ausführen: 'PSIZE' verändert die Byte-Folge in den Registern a und b; "EFT" darf also nicht mit 30 im X-Register aufgerufen werden. 'XYZALM' (Funktionswert 93) liefert grundsätzlich einen Weckauftrag mit Wiederholungsintervall 0, weil Register Z zu dem Zeitpunkt, zu dem 'XYZALM' als in den Zustandsregistern enthaltener Programmbefehl ausgeführt wird, der Wirkung nach 0 enthält (tatsächlich handelt es sich um den Inhalt von Register b zum Zeitpunkt der Ausführung von Zeile 21). 'PCLPS' (Funktionsnummer 27) sollte vermieden werden, sobald "EFT" selbst mitgelöscht würde, weil Sie dann mit dem Adreßzeiger in die Tastenzuweisungsregister gelangten und deren Inhalte als Programmbefehle bearbeitet würden. Ansonsten bietet 'PCLPS' allerdings die schnellste Möglichkeit, Programme im Arbeitsspeicher zu löschen.
Beiläufig bemerkt haben die Zeilen 15 und 23 den Sinn, jeden Fehler-Stop, der auftreten könnte, während der Adreßzeiger noch auf die in den Zustandsregistern befindlichen Befehle weist, zu vermeiden und auf die Rückkehr in den Arbeitsspeicher zu verschieben. Wenn Sie nämlich in den Zustandsregistern anhalten, werden Sie beim 'SST' bemerken, daß der Prozessor ungewöhnlich lange an der Berechnung von Zeilennummern arbeitet.

```
01♦LBL "EFT"    08 CLX          15 SF 25        22 RDN
02 RCL [        09 64           16 "┼┼t¿"       23 FS?C 25
03 CLA          10 +            17 RDN          24 STOP
04 STO [        11 RCL [        18 X<> [        25 SF 30
05 AON          12 "pTwu        19 X<> a        26 END
06 PSE          13 X<>Y         20 X<> \
07 AOFF         14 XTOA         21 X<> b                  58 BYTES
```

Den Barcode für "EFT" finden Sie im Anhang E.

"LB"-Eingaben:

Zeile 02: 144, 117 Zeile 04: 145, 117 Zeile 11: 144, 117

Zeile 12: 247, 145*), 112, 207*), 84, 12, 117, 166*)

Zeile 16: 245, 127, 127, 116, 145*), 124

Zeile 18: 206, 117 Zeile 19: 206, 123 Zeile 20: 206, 118

Zeile 21: 206, 124

*) Unsichtbar in Programm-Auflistungen. 166 (hexadezimal A6) in Zeile 12 verursacht ein Überspringen von 6 Zeichen.

Anm. des Übers. für Fortgeschrittene: In der Routine "EFT" steckt ein meisterliches Stück Synthetik, bei dem ein glücklicher Umstand dem Scharfsinn zur Hilfe kommt: In Zeile 21 gelangt der Inhalt von X ins Register b. Dabei wird der Adreßzeiger, der den Schluß von Register b bildet, auf einen Wert gesetzt, der auf Byte 5 *von Register b selbst* weist. Damit dies gelingt, muß das letzte Byte von Register b durch den Befehl 'X< >b' auf den Hexadezimalwert 0C gesetzt werden (Dezimalwert 12 in Zeile 12). In diesem Moment nun beginnt der Prozessor den Inhalt von Register b als Befehlsfolge abzuarbeiten. Dabei kommt es zu einem dramatischen Vorgang: Beim Voranschreiten stößt der Prozessor unaufhaltsam auf den Adreßzeiger vor, dessen Inhalt einerseits dadurch, daß das Programm läuft, ständig verändert wird, andererseits darauf wartet, als Befehlsfolge interpretiert und ausgeführt zu werden. Da Zeile 12 aber scharfsinnigerweise so zusammengestellt wurde, daß das erste (linke) Byte des Adreßzeigers die Nachsilbe einer Marke ist, mithin Änderungen ohne bösen Folgen übersteht, und das zweite (rechte) Byte, welches notwendigerweise 0C lauten *mußte*, als Befehl betrachtet ebenfalls eine Marke ist, nämlich 'LBL 11' (hierin liegt der erwähnte glückliche Umstand), kann man getrost den Adreßzeiger gewissermaßen *'sich selbst verarbeiten lassen'*. Ein wahrhaft abenteuerliches Ereignis. Die Tatsache, daß der Adreßzeiger während des gesamten Vorgangs überdies eine Adresse außerhalb des Hauptspeichers enthält, enthalten darf, ja sogar muß, hat noch andere tiefer liegende Gründe, die hier nicht mehr erläutert werden sollen.

Funktionskodes und XROM-Zahlen für "EFT" (X-Funktionen, Time-Modul, Lesestift):

	-EXT FCN 2D	
1	ALENG	25,01
2	ANUM	25,02
3	APPCHR	25,03
4	APPREC	25,04
5	ARCLREC	25,05
6	AROT	25,06
7	ATOX	25,07
8	CLFL	25,08
9	CLKEYS	25,09
10	CRFLAS	25,10
11	CRFLD	25,11
12	DELCHR	25,12
13	DELREC	25,13
14	EMDIR	25,14
15	FLSIZE	25,15
16	GETAS	25,16
17	GETKEY	25,17
18	GETP	25,18
19	GETR	25,19
20	GETREC	25,20
21	GETRX	25,21
22	GETSUB	25,22
23	GETX	25,23
24	INSCHR	25,24
25	INSREC	25,25
26	PASN	25,26
27	PCLPS	25,27
28	POSA	25,28
29	POSFL	25,29
30	PSIZE	25,30
31	PURFL	25,31
32	RCLFLAG	25,32
33	RCLPT	25,33
34	RCLPTA	25,34
35	REGMOVE	25,35
36	REGSWAP	25,36
37	SAVEAS	25,37
38	SAVEP	25,38
39	SAVER	25,39
40	SAVERX	25,40
41	SAVEX	25,41
42	SEEKPT	25,42
43	SEEKPTA	25,43
44	SIZE?	25,44
45	STOFLAG	25,45
46	X<>F	25,46
47	XTOA	25,47

	-CX EXT FCN	
49	ASROOM	25,49
50	CLRGX	25,50
51	ED	25,51
52	EMDIRX	25,52
53	EMROOM	25,53
54	GETKEYX	25,54
55	RESZFL	25,55
56	ΣREG?	25,56
57	X=NN?	25,57
58	X≠NN?	25,58
59	X<NN?	25,59
60	X<=NN?	25,60
61	X>NN?	25,61
62	X>=NN?	25,62

	- WAND 1F -	
129	WNDDTA	27,01
130	WNDDTX	27,02
131	WNDLNK	27,03
132	WNDSUB	27,04
133	WNDSCN	27,05
134	"WNDTST	27,06

	-TIME 2C	
65	ADATE	26,01
66	ALMCAT	26,02
67	ALMNOW	26,03
68	ATIME	26,04
69	ATIME24	26,05
70	CLK12	26,06
71	CLK24	26,07
72	CLKT	26,08
73	CLKTD	26,09
74	CLOCK	26,10
75	CORRECT	26,11
76	DATE	26,12
77	DATE+	26,13
78	DDAYS	26,14
79	DMY	26,15
80	DOW	26,16
81	MDY	26,17
82	RCLAF	26,18
83	RCLSW	26,19
84	RUNSW	26,20
85	SETAF	26,21
86	SETDATE	26,22
87	SETIME	26,23
88	SETSW	26,24
89	STOPSW	26,25
90	SW	26,26
91	T+X	26,27
92	TIME	26,28
93	XYZALM	26,29

	-CX TIME	
95	CLALMA	26,31
96	CLALMX	26,32
97	CLRALMS	26,33
98	RCLALM	26,34
99	SWPT	26,35

Funktionskodes und XROM-Zahlen für 'eGØBEEP' (HP-IL-Funktionen, Drucker):

	-MASS ST 1H	
1	CREATE	28,01
2	DIR	28,02
3	NEWM	28,03
4	PURGE	28,04
5	READA	28,05
6	READK	28,06
7	READP	28,07
8	READR	28,08
9	READRX	28,09
10	READS	28,10
11	READSUB	28,11
12	RENAME	28,12
13	SEC	28,13
14	SEEKR	28,14
15	UNSEC	28,15
16	VERIFY	28,16
17	WRTA	28,17
18	WRTK	28,18
19	WRTP	28,19
20	WRTPV	28,20
21	WRTR	28,21
22	WRTRX	28,22
23	WRTS	28,23
24	ZERO	28,24

	-CTL FNS	
27	AUTOIO	28,27
28	FINDID	28,28
29	INA	28,29
30	IND	28,30
31	INSTAT	28,31
32	LISTEN	28,32
33	LOCAL	28,33
34	MANIO	28,34
35	OUTA	28,35
36	PWRDN	28,36
37	PWRUP	28,37
38	REMOTE	28,38
39	SELECT	28,39
40	STOPIO	28,40
41	TRIGGER	28,41

	-PRINTER 2E	
65	ACA	29,01
66	ACCHR	29,02
67	ACCOL	29,03
68	ACSPEC	29,04
69	ACX	29,05
70	BLDSPEC	29,06
71	LIST	29,07
72	PRA	29,08
73	'PRAXIS	29,09
74	PRBUF	29,10
75	PRFLAGS	29,11
76	PRKEYS	29,12
77	PRP	29,13
78	'PRPLOT	29,14
79	'PRPLOTP	29,15
80	PRREG	29,16
81	PRREGX	29,17
82	PRΣ	29,18
83	PRSTK	29,19
84	PRX	29,20
85	REGPLOT	29,21
86	SKPCHR	29,22
87	SKPCOL	29,23
88	STKPLOT	29,24
89	FMT	29,25

Auf die Funktionen des Kartenlesers läßt sich nicht vermittels 'eGØBEEP' zugreifen.

4.6 Wenn Sie einen Mathematik-Modul (XROM 01), einen Statistik-Modul (XROM 02) oder einen Modul für Vermessung (XROM 03) besitzen, tätigen Sie mit den Dezimalwerten 0 und 160 eine synthetische Tastenzuweisung, und 'erforschen' Sie deren Wirkung.
Wenn Sie einen Finanz-Modul (XROM 04), einen Modul für Netzwerkanalyse (XROM 06) oder einen Modul für Baustatik (XROM 07) haben, 'erforschen' Sie die Zuweisung, die sich aus den Dezimalwerten 0 und 161 ergibt.

KAPITEL 5

Die Vorgänge im HP-41 beim Edieren eines Programms

Im Abschnitt 2B wurde versprochen, die Vorgänge im Programmspeicher bezüglich der 'NULL'-Bytes genauer vorzuführen. Dies geschieht am besten mit Hilfe eines ungewöhnlichen synthetischen Befehls, genannt 'FØ-Marke'. Diese FØ-Marke versetzt uns in die Lage, mehrere der auf sie folgenden Programmbefehle als Textzeichen in der Anzeige sichtbar zu machen, ohne sie mit der Byte-Schnapper-Technik einer Textzeile 'rechtskräftig' einzuverleiben.

Wir wollen uns diesen merkwürdigen synthetischen Befehl zunächst einmal herstellen. Entweder mit "LB", wobei man die Dezimalwerte 192, 0 und 240 verwenden muß, oder mit dem Byte-Schnapper: 'ENTER↑, STO IND 64, STO IND T, BST, BST', BS, zweimal Korrekturtaste. Ein 'PACK' ist noch nötig, damit die neue Marke dem Katalog 1 eingefügt wird. Sie haben nun eine synthetisch hergestellte globale Marke, deren drittes Byte dezimal 240 (hexadezimal F0) lautet, woher die Bezeichnung FØ-Marke rührt. Normalerweise lautet das dritte Byte einer im Katalog 1 enthaltenen globalen Marke 241 + n, wenn n die Anzahl der Zeichen im Namen der Marke ist. Dieser Regel entsprechend hätte der Name der FØ-Marke die Länge −1. Es erweist sich nun, daß der Prozessor durchaus nicht in Verlegenheit gerät, wenn er diesem abartigen Parameter begegnet, sondern ihn einfach modulo 16 (−1 mod 16 = 15) verarbeitet. Bei der Anzeige der Marke im PRGM-Modus erblickt man daher 'LBL ᵀ', gefolgt von 15 Zeichen. Der Prozessor überspringt ein Byte (es ist dasjenige, welches in normalen globalen Marken die Information über die Tastenzuweisung der Marke aufnimmt) und zeigt die 15 darauf folgenden Bytes gemeinsam als 'Namen' der Marke an. Wenn Sie jedoch mit 'SST' im Programmspeicher voranschreiten, werden Sie feststellen, daß der angezeigte Marken-Name nur ein Trugbild ist; die Bytes sind nicht wirklich in die FØ-Marke gelangt.

Wir wollen uns nun das Ganze an Hand eines Beispiels veranschaulichen. Doch vorab eine Warnung: Laufen Sie nicht im RUN-Modus mit 'SST' über eine FØ-Marke hinweg, und lassen Sie auch kein Programm, das eine solche Marke enthält, laufen, sonst erhalten Sie einen GAU, bei dem das Tastenfeld solange bedienungsunfähig bleibt, bis die Batterien kurzzeitig entfernt werden. Diese 'Notbremse' unterbricht eine 'unendliche Schleife'. Wenn die Batterien gleich darauf wiedereingesetzt werden, bleiben die Speicherinhalte unversehrt. Ein harmloser GAU, verglichen etwa mit einem Byte-Schnappen, welches das '.END.' zerstört. Letzteres führt fast unvermeidlich zu "MEMORY LOST".

Tasten Sie jetzt, mit der FØ-Marke in der Anzeige, diese Folge von Befehlen ein: '−, *, /, X<Y?, X>Y?, X≤Y?, Σ+, Σ−, HMS+, HMS−, MOD, %, %CH, P→R, R→P, LN, X↑2, SQRT, Y↑X, CHS, E↑X, LOG, 10↑X, E↑X−1, SIN, COS'. Gehen Sie danach auf die FØ-Marke zurück, und Sie erblicken

'LBL "BCDEFGHIJKLMNOP"'.

(Sollte das nicht der Fall sein, müßten Sie noch 'PACK'en.) Die Zeichen "B" bis "P" entsprechen den auf die FØ-Marke folgenden Befehlen '*, /, ... , LN'. Sie können dies unmittelbar aus der QRC ablesen (Zeilen 4 und 5). Um die Wechselbeziehung noch weiter zu verdeutlichen, tasten Sie sich jetzt zum Befehl '/' durch, löschen Sie ihn, und gehen Sie dann wieder zur FØ-Marke zurück. Ergebnis:

```
'LBL "B¯DEFGHIJKLMNOP"'.
```

Dies offenbart, daß Befehle, welche gelöscht werden, durch 'NULL'en, die normalerweise unsichtbar bleiben, ersetzt werden. Ein 'NULL'-Byte (hexadezimal 00, dezimal 0) wird in der Anzeige durch das Zeichen "¯" (Überstrich) kenntlich gemacht. Ein 'PACK' an dieser Stelle hat folgende Wirkung:

```
'LBL "BDEFGHIJKLMNOPQ"'.
```

Das 'NULL'-Byte ist also durch den 'PACK'-Vorgang beseitigt worden.

Die FØ-Marke kann uns ferner eine genaue Vorstellung davon vermitteln, wie der Prozessor arbeitet, wenn neue Befehle in ein Programm eingefügt werden. Tasten Sie sich zu der dem "D" entsprechenden Programmzeile 'X<Y?' durch, und fügen Sie hinter ihr ein '+' ein. Die anschließende Inspektion mit der FØ-Marke enthüllt

```
'LBL "BD@------EFGHIJ"'.
```

Das Zeichen "@" entspricht dem '+'. Sie haben aber wahrscheinlich nicht die sechs durch Überstriche angezeigten 'NULL'en erwartet. Wir dürfen daraus schließen, daß jedesmal, wenn kein Platz mehr für einzuschiebende Befehle zur Verfügung steht, wenn also zu wenige 'NULL'-Bytes vorhanden sind, sieben neue eingesetzt werden, und zwar auch dann, wenn nur eines zum Überschreiben benötigt wird. Die im nachfolgenden Teil des Programmspeichers vorhandenen Inhalte bis hinab zum '.END.', dieses selbst eingeschlossen, werden bei diesem Vorgang um ein Register (= 7 Bytes) nach unten verschoben, wodurch die Anzahl der freien Register um eines vermindert wird. (In Kapitel 6 können Sie nachlesen, wie der Arbeitsspeicher aufgebaut ist und wo genau die freien Register liegen.) Der Grund für diese Art des Platzschaffens liegt darin, daß der Prozessor die Verschiebung um ein ganzes Register sehr viel schneller bewältigen kann als die um ein einzelnes Byte.

Einfügungen an solchen Stellen, an denen noch genügend 'NULL'en zum Überschreiben bereitliegen, verändern den nachfolgenden Teil des Programmspeichers nicht. Beispiel: Setzen Sie hinter das eben eingefügte '+' noch die Befehle 'STO 01, STO 02, STO 03, STO 04, STO 05, STO 06'. Die FØ-Marke zeigt daraufhin

```
'LBL "BD@123456EFGHIJ"'.
```

Die sechs neuen Befehle haben genau den verfügbaren Platz ausgefüllt.

Nachdem Sie jetzt eine Vorstellung davon erlangt haben, wie das Einfügen von Befehlen durch den Prozessor bewerkstelligt wird, können Sie auch verstehen, wie der Byte-Schnapper arbeitet. Sobald die ihn tragende Taste im PRGM-Modus gedrückt wird, erzeugt er drei Bytes: ein 'TEXT 7'-*Byte*, dem ein 'NULL'-Byte und daran anschließend ein drittes Byte, für welches wir in diesem Buch dezimal 63 gewählt haben (mit Hilfe von "MK" können Sie dafür auch einen beliebigen anderen Wert nehmen), folgen. Ein 'TEXT 7'-*Befehl* besetzt im Programmspeicher aber 8 Bytes: das 'TEXT 7'-Byte, gefolgt von 7 als Zeichen darzustellenden Bytes. Doch der Prozessor 'weiß' lediglich von drei Bytes, für die Platz zu schaffen ist. Wenn aber keine überschreibungsfähigen 'NULL'en vorhanden sind, müssen neue her, und zwar sieben an der Zahl, wie wir gesehen haben. So erklärt es sich denn also, daß eine Textzeile entsteht, deren erstes und zweites Zeichen aus der unmittelbaren Wirkung des Byte-Schnappers herrühren, deren folgende vier zusammenhängende 'NULL'-Bytes aus dem unbenutzten Rest des bereitgestellten Registers stammen und deren siebentes Zeichen (also achtes Byte) räuberisch der nächstfolgenden Programmzeile entzogen wird. Abbildung 5.1 veranschaulicht das Ergreifen dieses Bytes an Hand des Beispiels aus Kapitel 1.

vorher

Befehl:	ENTER↑	STO	IND 31	PI
Hexadezimalwerte:	83	91	9F	72
Dezimalwerte:	131	145	159	114

nachher

Befehl:	ENTER↑				"-?----▒"					TONE	Y
Hexadezimalwerte:	83	F7	0	3F	00	00	00	00	91	9F	72
Dezimalwerte:	131	247	0	63	0	0	0	0	145	159	114

Abbildung 5.1. Erzeugung von 'TONE Y' mit dem Byte-Schnapper

Der Byte-Schnapper kann, wenn entsprechende Vorbereitungen getroffen werden, dazu benutzt werden, bis zu 5 Bytes zu ergreifen. Sie müssen zu diesem Zweck zunächst 'PACK'en oder auf andere Weise dafür sorgen, daß den Bytes, welche Sie ergreifen wollen, keine 'NULL'en vorangehen. Dann müssen Sie *unmittelbar vor Aufruf* des Byte-Schnappers 1 bis 4 'Füll'-Bytes einfügen. Um beispielsweise zwei Bytes zu ergreifen, können Sie 'X< >Y' als Füll-Byte eingeben, bevor Sie BS ausführen. Wir haben diese Technik bereits in Abschnitt 2B verwendet, um die 1 vor einem Exponenten-Eintrag ohne 'PACK'-Vorgang zu beseitigen. Um 3 Bytes zu ergreifen, kann man unmittelbar vor BS eine einziffrige Zahl eintasten; sie besetzt 2 Bytes. Um vier Bytes zu ergreifen, wählt man am besten 'EEX', weil dieser Befehl gleich 3 der entstehenden 7 'NULL'en überschreibt, und sollen 5 Bytes in den Einflußbereich des zukünftigen 'TEXT 7'-Bytes gelangen, tut es 'EEX', gefolgt von einer Ziffer, am schnellsten. Der Vorgang ist in all diesen Fällen derselbe: Normalerweise ist es der Byte-Schnapper selber, welcher den Prozessor veranlaßt, ein volles Register 'NULL'en zur Verfügung zu stellen, von

denen dann drei 'für ursprünglich eigene Zwecke des Byte-Schnappers' verwendet werden. Wird der Prozessor aber durch gewöhnliches Einfügen gewöhnlicher Befehle dazu angehalten, ein neues Register freizumachen, und stehen in diesem Register anschließend noch genügend 'NULL'en für den Byte-Bedarf des Byte-Schnappers zur Verfügung, nämlich 3 Stück, so macht das erste Byte, welches durch den Aufruf des Byte-Schnappers entsteht, also das 'TEXT 7'-Byte, die im Programm folgenden Bytes zu einer Textzeile, unter deren Zeichen sich – gegenüber 'normalem' Byte-Schnappen – gerade soviel weniger Überstriche ('NULL'-Bytes) befinden, als Bytes beim anfänglichen Einfügen von Befehlen in das neu bereitgestellte Register gelangt sind. Die letzten Zeichen der durch BS erzeugten Textzeile entstehen aus den Bytes, die Sie ergreifen wollten.

Seien Sie aber vorsichtig, wenn Sie mehr als ein Byte schnappen. Falls Sie dabei versehentlich einmal das '.END.' ganz oder teilweise ergreifen, betätigen Sie nicht etwa voller Panik die Korrekturtaste, sondern gehen Sie vielmehr besonnen mit 'BST' vor die durch den Byte-Schnapper entstandene Textzeile, um ihr das führende 'TEXT 7'-Byte zu entziehen und so das '.END.' wieder freizusetzen bzw. zu vervollständigen.

Möglicherweise setzt sich bei Ihnen der Gedanke fest, daß der 'PACK'-Vorgang sämtliche 'NULL'-Bytes in allen Programmen beseitigt. Dem ist aber nicht so. Gelegentlich ist eine 'NULL' nämlich der Träger wichtiger Information und darf nicht fortfallen.

Der erste Fall dieser Art liegt vor, wenn sich in einem Programm eine 'NULL' zwischen zwei unmittelbar aufeinanderfolgenden numerischen Einträgen befindet (vgl. Abschnitt 2B). Um das zu zeigen, wollen wir mit unseren Untersuchungen dort fortfahren, wo wir die FØ-Marke in der Gestalt

```
'LBL "BD@123456EFGHIJ"'
```

verließen. Gehen Sie mit 'SST' auf das '—', welches vor dem '*', angezeigt als "B", steht. Tasten Sie aufeinanderfolgend die beiden numerischen Einträge '1 E3' und '56' ein. Um Sie getrennt voneinander eingeben zu können, schließen Sie den Eintrag von '1 E3' dadurch ab, daß Sie einfach einmal den Alpha-Modus an- und ausschalten. Wenn Sie sich jetzt die FØ-Marke ansehen, erblicken Sie

```
'LBL "-■■■-■■-------B"'.
```

Hinter den ersten drei Vollzeichen verbirgt sich der Befehl '1 E3', während die anderen beiden von '56' herrühren. Beiden Gruppen ist automatisch ein 'NULL'-Byte vorangesetzt worden. Wenn Sie nun 'PACK'en, ergibt sich

```
'LBL "■■■-■■BD@123456"'.
```

Alle 'NULL'en außer der, welche die beiden numerischen Einträge trennt, sind verschwunden. Die zurückgebliebene 'NULL' wird dazu benötigt zu verhindern, daß die beiden Einträge zu einer einzigen Programmzeile verschmelzen. Sie widersetzt sich daher jedem 'PACK'. Und dies ist auch die Erklärung für die anfangs entstandenen 'NULL'en *vor* den Einträgen. Der HP-41 besteht zunächst darauf, solche 'NULL'en den Zahleneinträgen zu dem Zeitpunkt, zu

dem sie eingetastet werden, voranzustellen: es könnten ja davor auch Zahleneinträge stehen. Beim 'PACK'en werden die 'NULL'en dann beseitigt, sofern sie sich nicht zwischen zwei Einträgen befinden. Der HP-41 besteht aber noch überdies darauf, einen Zahleneintrag auch vom *nach*folgenden Programmbefehl durch eine Trenn-'NULL' abzusondern, und zwar aus demselben Grunde. In unserem Beispiel erkennt man dies daran, daß beim Eintasten der 6 von '56' offenkundig sieben weitere 'NULL'en zum Überschreiben zur Verfügung gestellt doch sichtlich 'unverbraucht' zurückgelassen wurden. Ohne den 'Eigensinn' des HP-41 wäre für die 6 ja eigentlich noch Platz gewesen. Aber: hätte der nachfolgende Befehl selbst aus einzutragenden Zahlen bestanden, so wäre er unzulässigerweise mit der '56' verschmolzen worden. Die zwangsweise vor- und nachgestellten 'NULL'-Bytes verhindern derartige Fehler.

Ein anderes Beispiel für 'PACK'-resistente 'NULL'en begegnet uns in jenen, die Teil eines Mehrbyte-Befehls sind. Beispielsweise taucht der Befehl 'ST+ 00' in der FØ-Marke als ″▓¯″ auf. Das zweite Byte ist eine 'NULL'. Sie muß einem 'PACK' widerstehen, weil sie Teil des Befehls ist und somit eine wesentliche Information, hier die Registernummer, enthält. Bedenkt man die vielen Regeln, nach denen überflüssige von wesentlichen 'NULL'en zu unterscheiden sind, nimmt es nicht wunder, daß ein 'PACK' so zeitraubend sein kann.

Eine weitere bemerkenswerte Erscheinung bezüglich der 'NULL'en muß noch erörtert werden. Gewöhnlich überschreibt ein Befehl, der eingetastet wird, die 'NULL'en, welche hinter dem gerade angezeigten Befehl stehen, wobei ein neues Register bereitgestellt wird, falls der Platz nicht mehr ausreicht. Wird jedoch beim Eintasten ein 'END' (oder das '.END.') angezeigt, gelangt der neue Befehl nicht etwa dahinter, sondern genau dorthin, wo eben noch das 'END' stand, welches seinerseits um 7 Bytes abwärts im Programmspeicher verschoben wird. Diese Regel gilt auch dann, wenn noch hinreichend viele 'NULL'en vor dem 'END' vorhanden sind, um den neuen Befehl aufzunehmen.

Führen wir uns die Verhältnisse beim 'END' mit der FØ-Marke vor Augen: 'FØ-Marke, –, *, END'. 'GTO' auf die FØ-Marke, 'PACK'. Es erscheint

'LBL⌜B▓¯▓',

gefolgt von weiteren Zeichen. Das zweite, dritte und vierte Zeichen in der FØ-Marke sind die Bytes des 'END'. Löschen Sie nun den Befehl '*'. Wenn Sie sofort wieder ein neues '*' einfügen, nimmt es genau die Stelle des alten '*' ein. Nichts hat sich verändert. Wenn Sie dagegen mit 'SST' zum 'END' vorschreiten und dann erst das neue '*' einfügen, haben Sie folgendes Ergebnis

'LBL⌜¯B¯¯¯¯¯¯▓▓▓',

gefolgt von weiteren vier Zeichen. Der Befehl '*' ist dort eingefügt worden, wo vorher das 'END' war, während dieses um 7 Bytes nach unten verschoben wurde. Sechs zusätzliche 'NULL'en sind entstanden, obwohl dazu eigentlich keine Notwendigkeit bestand. Es ist daher zu empfehlen, Einfügungen nicht gerade mit dem 'END' in der Anzeige vorzunehmen. Stattdessen sollte man besser mit 'BST' zurücksetzen, um Gebrauch von im allgemeinen vor dem 'END' vorhandenen 'NULL'en zu machen. Natürlich beseitigt ein 'PACK' überflüssigerweise eingefügte 'NULL'en, doch bei knapp in den Speicher passenden Programmen kann man so gelegentlich vermeiden, die Speicheraufteilung neu festzusetzen zu müssen.

Es wird Ihnen gewiß aufgefallen sein, daß das 'END' nach seiner Verschiebung nicht nur seine Lage, sondern auch sein Erscheinungsbild verändert hat. Das liegt daran, daß in den ersten beiden Bytes eines 'END' oder einer globalen Marke die sogenannte relative Adresse zu dem im Katalog 1 vorangehenden Element abgelegt wird. Wenn der Katalog 1 beispielsweise genau die Elemente 'LBL "ABC"', 'END' und '.END.' enthält, dann liegt im '.END.' die Information über die Entfernung zum 'END', in diesem die Information über die Entfernung zum 'LBL "ABC"', und 'LBL "ABC"' selbst enthält die relative Adresse 0, wodurch der Anfang des Katalogs 1 angezeigt wird. Der Rechner verwendet diese aus Entfernungsangaben bestehende 'Liste', um die Kette der Marken und 'END's vom '.END.' an aufwärts 'abzuklappern', sobald eine globale Marke gesucht wird. Die Liste wird auch beim Rückschreiten im Programm verwendet: sobald man 'BST' drückt, sucht der Rechner das erste voranstehende Element der globalen Kette auf und zählt von dort aus abwärts, um die mit 'BST' angeforderte Programmzeile zu finden. Dieser Umweg ist deshalb nötig, weil die Zeilennummern nicht als Bestandteil des Programms im Speicher aufbewahrt werden. Der Rechner kann nur dann feststellen, ob ein Byte ein Befehl oder bloß die Nachsilbe eines solchen ist (und somit keinen 'Anspruch auf eine eigene Zeilennummer' hat) wenn er von einer bekannten Position aus, wie sie durch ein Element des Katalogs 1 gegeben ist, abwärts mit der Zeilenzählung startet. Die Anweisung 'BST' kann nur in der Weise ausgeführt werden, daß von einer festen Stelle aus *vor*wärts gezählt wird. Daher auch die bekannte Langatmigkeit von 'BST' am Ende eines umfangreichen Programms, das nur eine globale Marke am Anfang stehen hat.

Wie schon in Kapitel 3 dargelegt wurde, werden auch in lokalen 'GTO's und 'XEQ's relative Adressen, also Entfernungsangaben, abgelegt. Die erste Ausführung eines 'GTO' oder 'XEQ' verschlingt Zeit für die Suche nach der angesprungenen Marke. Doch wenn die Suche erfolgreich war, wird das 'GTO' bzw. 'XEQ' durch Ablage der relativen Adresse vervollständigt, so daß alle späteren Sprünge bzw. Verzweigungen wesentlich schneller durchgeführt werden. Mit der FØ-Marke ist es möglich, auch diese Vorgänge genau zu verfolgen. Man braucht dazu nur ein 'GTO' oder 'XEQ' vor und nach dem ersten Sprung bzw. der ersten Verzweigung als Teil des Textes der FØ-Marke zu betrachten. Der Aufbau relativer Adressen ist ausführlich in W.C. Wickes, 'Synthetische Programmierung auf dem HP-41C/CV', S. 24–28, beschrieben.

Aufgaben:

5.1 Sagen Sie das Ergebnis der folgenden Schritte, Anzahl und Lage der unsichtbaren 'NULL'en eingeschlossen, voraus, und bestätigen Sie anschließend Ihre Voraussagen mit der FØ-Marke:

a) Tasten Sie '+, 3, −, 4, 5, *' ein. Fügen Sie 'Σ+' und 'Σ−' hinter '+' ein. Fügen Sie 'RCL 05' hinter '4' ein.

b) Tasten Sie '+, −, XEQ 00, GTO 99, *, /' ein. Löschen Sie 'GTO 99', und tasten Sie 'ST+ 75' ein.

KAPITEL 6

Der Aufbau des HP-41-Speichers und der Zustandsregister

Dieses Kapitel wird Ihre Kenntnisse über die Grundlagen der Arbeitsweise des HP-41 vervollständigen. Einige der hier beschriebenen Einzelheiten sind nicht für den sofortigen Gebrauch gedacht, sondern nur dafür, daß Sie nachschlagen können, wenn Sie etwas genauer wissen wollen. Gleichzeitig werden Sie mit den notwendigen Voraussetzungen zum Schreiben eigener Programme, die sich der für die synthetische Programmierung so charakteristischen Bit-Manipulationen bedienen, versehen. Auch dann, wenn Sie nur den Einsatz einfacher synthetischen Techniken planen und ansonsten die 'konservierten' synthetischen Programme des PPC ROMs oder der HP-Benutzerbibliothek verwenden wollen, sobald luxuriöse Synthetik ins Spiel kommt, können Ihnen die nachfolgenden Beschreibungen helfen, eine allgemeine Vorstellung vom Geist synthetischer Programme, die mit den Bits nur so jonglieren, zu erlangen.

6A. Der Aufbau des Speichers

Abbildung 6.1 veranschaulicht die Gliederung der eingebauten und zusätzlichen Speicher des HP-41 zur Aufnahme der Programme, Daten und System-Notizen. Der erweiterte Speicher, im folgenden kurz X-Memory genannt, einschließlich des Teiles, der im X-Funktionen-Modul enthalten ist, heißt in Anlehnung an den Sprachgebrauch bei großen Rechenanlagen *off-line Speicher*, weil in ihm enthaltene Programme nicht unmittelbar abgearbeitet werden können (Ausnahmen gestattet die Synthetik), sondern dazu erst in den Haupt- oder Arbeitsspeicher (*on-line Speicher*) gebracht werden müssen.

Einzelheiten über Inhalt und Aufbau des X-Memorys findet man auf S. 18 der Ausgabe März '82 des PPC CJ. Ein weiterer Artikel auf S. 26 der Ausgabe April '82 beschreibt, wie man mit Hilfe synthetischer Techniken Programme direkt im X-Memory laufen lassen kann.

Wir wenden uns zunächst dem Hauptspeicher, dessen Gliederung in Abbildung 6.2 gezeigt wird, zu.

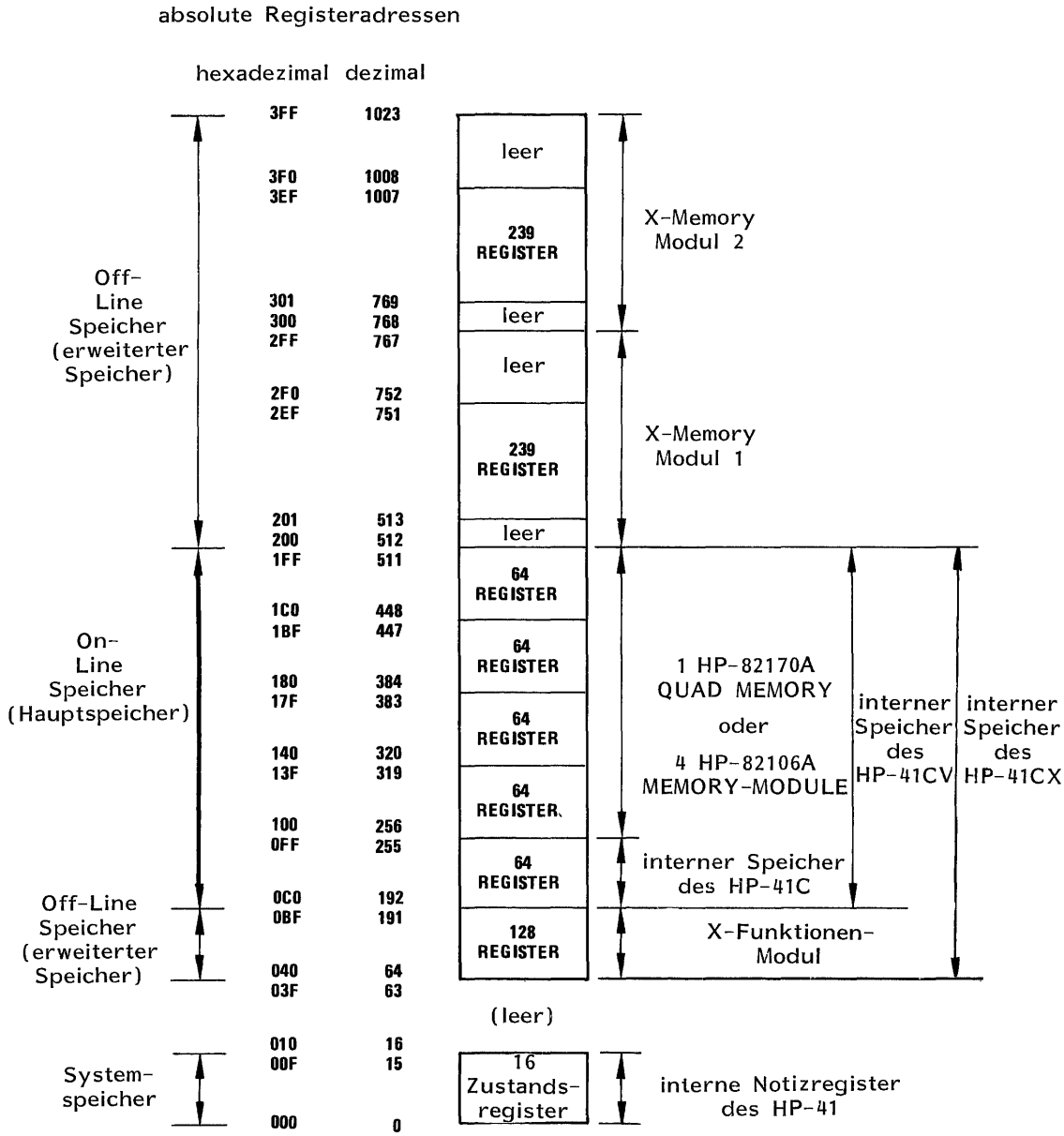


Abbildung 6.1. Speicheraufbau des HP-41

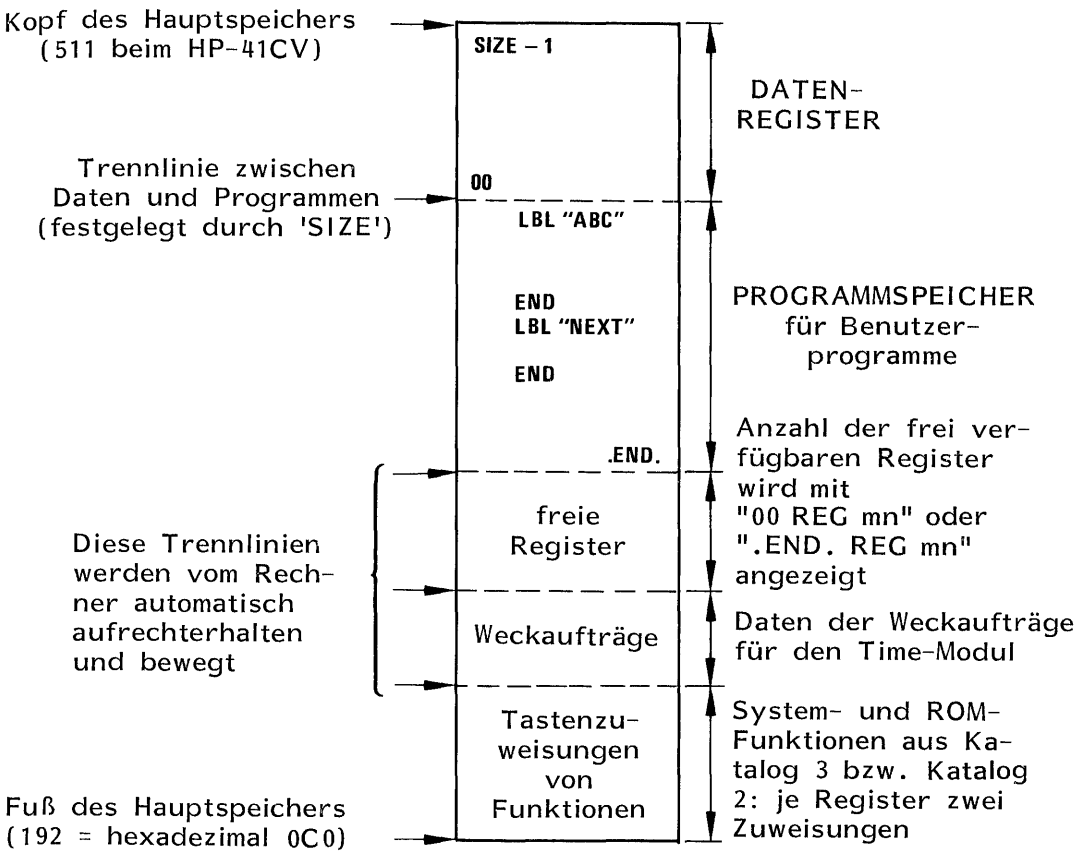


Abbildung 6.2. Aufbau des Hauptspeichers

Die Datenregister beginnen mit R_{00} oberhalb einer Trennlinie, über die beim Register c noch zu reden sein wird, und enden mit R_{mn-1} (wenn 'mn' die durch 'SIZE' festgelegte Datenregister-Anzahl ist) am Kopf des Hauptspeichers. Die Benutzerprogramme erstrecken sich von derselben Trennlinie aus nach unten bis zum '.END.', welches – dem benötigten Programmspeicherplatz entsprechend – vom Rechner automatisch verschoben wird. Unterhalb des '.END.' liegen 'freie' Register, die für weitere Programme (durch die Verschiebung des '.END.'), Weckaufträge und Tastenzuweisungen zur Verfügung stehen. Sie können durch Verwendung von 'SIZE' auch indirekt den Datenregistern zugute kommen: Erhöhung der Datenregister-Anzahl mit 'SIZE' verschiebt alle vorhandenen Datenregister-Inhalte und Programme nach unten in den Block der freien Register. Verminderung der Datenregister-Anzahl

mit 'SIZE' schiebt alle Daten und Programme im Speicher aufwärts, wobei die Anzahl der freien Register folgerichtig größer wird und die Inhalte der höher numerierten Datenregister am Kopf des Speichers verloren gehen. Die jeweils frei verfügbare Anzahl von Datenregistern kann jederzeit durch 'GTO.000' im PRGM-Modus oder 'RTN' im RUN-Modus und anschließendes Umschalten in den PRGM-Modus sichtbar gemacht werden: in beiden Fällen teilt die Anzeige in der Form "00 REG mn" mit, daß noch 'mn' Register frei sind.

Unterhalb der freien Register liegen die Weckaufträge und die Tastenzuweisungen. Tastenzuweisungen von ROM-Funktionen (Katalog 2) und System-Funktionen (Katalog 3) besetzen die mit der Dezimaladresse 192 beginnenden Register in aufwärts laufender Reihenfolge. Jedes Tastenzuweisungsregister ist durch ein führendes FØ-Byte gekennzeichnet. Die restlichen 6 Bytes enthalten zwei Zuweisungen, jede 3 Bytes lang. Die jeweils ersten beiden Bytes einer solchen Dreier-Gruppe legen die zugewiesene Funktion fest. Es handelt sich dabei um die Bytes, deren Dezimalwerte man verwendet, wenn man "MK" benutzt. Das dritte Byte jeder Dreier-Gruppe bestimmt die Taste, der die Funktion zugewiesen ist. Den genauen Zusammenhang zwischen Tastenkodes und Tastenzuweisungsbytes finden Sie in W.C. Wickes, 'Synthetische Programmierung auf dem HP-41C/CV', S. 31/32 beschrieben. Außerdem enthält die Seite 280 des Benutzerhandbuchs für den PPC ROM eine klare Zusammenfassung der Verhältnisse.

Die Weckaufträge für den Time-Modul liegen in Alarmregistern. Diese schließen an die Zuweisungsregister an. Jeder Weckauftrag benötigt ein Register für die Weckzeit sowie zusätzlichen Speicherplatz, wenn eine Meldung und/oder ein Wiederholungsintervall zum Weckauftrag gehören. Überdies wird ein Zählregister benötigt, in welchem die Gesamtanzahl der in Gebrauch befindlichen Alarmregister notiert wird. Es liegt als erstes Alarmregister unmittelbar oberhalb des obersten Tastenzuweisungsregisters. Als letztes grenzt ein Schlußregister die Alarmregister gegen die freien Register ab.

Damit ist die Beschreibung des Arbeitsspeichers des HP-41 abgeschlossen. Es gibt jedoch noch einen weiteren sehr wichtigen internen Speicherbereich, nämlich die Zustandsregister, welche das Betriebssystem für Notizen verwendet. Es handelt sich um 16 Register, die ganz am Anfang des Adreßraumes des HP-41 liegen. Sie haben die Dezimaladressen 0 bis 15 und heißen T, Z, Y, X, L, M, N, O, P, Q, ⊢, a, b, c, d und e. Die meisten davon sind Ihnen schon vertraut: die ersten 5 werden bereits im Benutzerhandbuch beschrieben, während einige der anderen im Kapitel 2 dieses Buches eingeführt wurden. Abbildung 6.3 ist eine kurze Darstellung, aus der hervorgeht, wie der Prozessor diese Register nutzt.

Auf die *Stapelregister T, Z, Y, X und L* hat der Benutzer mit normalen Befehlen Zugriff. Zusätzlich zu den schon von Vorgängern des HP-41 bekannten Befehlen 'ENTER↑', 'RDN', 'R↑' und 'LASTX' erlaubt der HP-41 direkten Zugriff auf alle Stapelregister mit Befehlen wie 'RCL Z' oder 'X<>L'. Mittels der synthetischen Programmierung kann der Gebrauch der Befehle 'STO', 'RCL' und 'X<>' aber sogar auf die restlichen 11 Zustandsregister ausgedehnt werden.

Die *Register M, N, O und P* enthalten das 24 Zeichen umfassende Alpha-Register. Der Inhalt des Alpha-Registers liegt stets rechtsbündig in den Zustandsregistern, soweit sie betroffen sind. So enthält das Byte 0 von Register M das äußerste rechte Zeichen des Alpha-Registers,

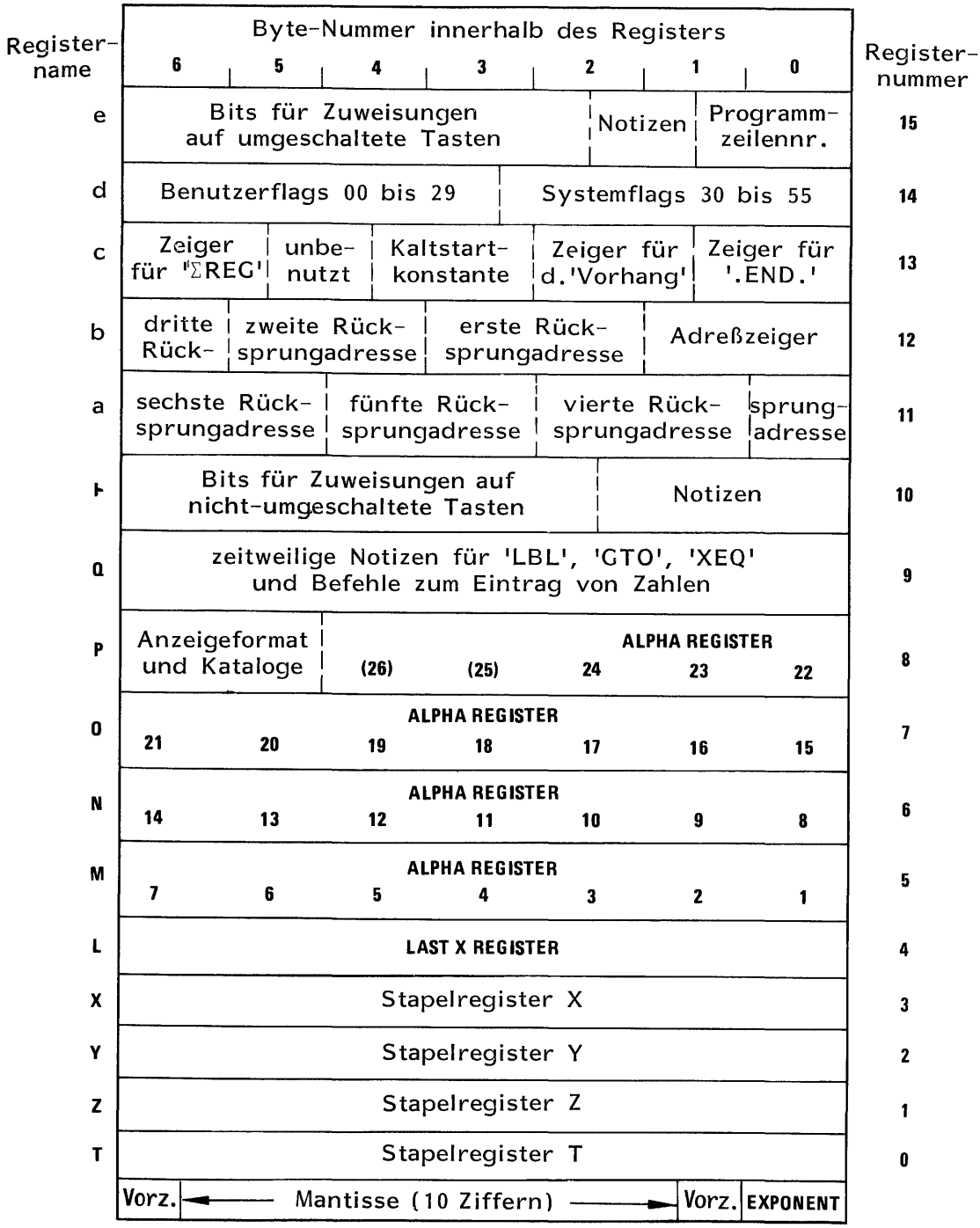


Abbildung 6.3. Die Zustandsregister

das Byte 1 von M enthält das zweite Zeichen des Alpha-Registers von rechts aus gerechnet, usw. Die durch die linksbündig arbeitende Anzeige nahegelegte Vermutung, Byte 2 von Register P wäre stets das äußerste linke Zeichen des Alpha-Registers, entspricht also nicht den tatsächlichen Verhältnissen. Dies gilt nur in dem Fall, in welchem das Alpha-Register 24 Zeichen enthält. Wenn das Alpha-Register nur 7 oder weniger Zeichen enthält, ist allein Register M betroffen. Sobald mehr Zeichen angefügt werden, gelangen die führenden, also die äußersten linken Zeichen, von rechts nach links aufwärts in die Register N, O und P. Sobald das 24. Zeichen eingetastet wird (jetzt erst gelangt das äußerste linke Zeichen nach Byte 2 von Register P), erklingt ein Warnton. Das Anfügen weiterer Zeichen schiebt nun die äußersten linken Zeichen in den Bereich von Register P, der auch für Notizen benutzt wird, womit sie im allgemeinen verloren gehen.Wenn man jedoch im Alpha-Modus verbleibt oder zwischenzeitlich wenigstens keine Zahlenanzeige zuläßt, bleiben die in den Bytes 6 bis 3 enthaltenen äußersten 4 linken Zeichen unbehelligt und lassen sich mit synthetischen Befehlen wie 'RCL P' zurückgewinnen. Das Morsezeichen-Programm in Anhang B nutzt diese Möglichkeit, über ein auf 28 Zeichen verlängertes Alpha-Register zu verfügen, aus.

Die Bytes 6 und 5 von Register P werden vom Prozessor unter bestimmten Bedingungen benutzt. Das erste Byte enthält eine verschlüsselte Darstellung des Anzeige-Modus ('FIX', 'SCI', 'ENG', Flags 28, 29, Anzahl der Nachkommastellen). Es wird vom Prozessor in den jeweils entsprechenden Zustand versetzt, sobald eine *numerische* Anzeige erfolgt oder ein Zahleneintrag durch ein Programm ausgeführt wird. Das zweite Byte von P wird ebenfalls bei Zahleneinträgen, sowohl von Hand als auch durch ein Programm, benutzt.

Eine weitere Verwendung finden die ersten beiden Bytes von P, wenn ein Katalog 'durchblättert' wird. Byte 6 enthält dann die Nummer des Katalogs, während in Byte 5 die 'Seitennummern' innerhalb eines Kataloges mitgezählt werden.

Einzelheiten über die Benutzung der Bits in den ersten beiden Bytes von Register P findet man auf Seite 13 der Ausgabe Juli '81 des PPC CJ.

Register Q wird immer dann benutzt, wenn der Name einer globalen Alpha-Marke 'buchstabiert' wird. Dies geschieht sowohl, wenn die Marke eingetastet wird, als auch dann, wenn die zugehörigen Befehle 'GTO' oder 'XEQ' eingegeben oder über das Tastenfeld ausgeführt werden. Dabei gelangen die einzelnen Buchstaben und Zeichen des Namens in umgekehrter Reihenfolge ins Register Q. Bei Zahleneinträgen, von Hand oder durch ein Programm, wird Q ebenfalls verwendet. Die Zahl wird in Q zusammengesetzt, bevor der Prozessor sie ins Register X überträgt. Einzelheiten über den Gebrauch von Register Q findet man auf Seite 78 der Ausgabe August '81 des PPC CJ. Falls Sie Q selbst untersuchen wollen, denken Sie daran, daß auch der Drucker von diesem Register fleißig Gebrauch macht.

In einigen Fällen erlauben die mit Register Q verbundenen Eigenheiten des Prozessors gewisse Abkürzungen beim Umgang mit dem Rechner. So läßt sich z.B. 'eGØBEEP 77', das entspricht 'PRP' (vgl. Abschnitt 4A), benutzen, wenn unmittelbar zuvor ein 'GTO' auf eine globale Marke des auszudruckenden Programms buchstabiert wird. Statt 'GTO' kann man auch 'LBL' oder, wie Robert Edelen entdeckt hat, 'eGØBEEP' selbst verwenden: ein Alpha-Name wird von dieser Funktion angenommen ('eGØBEEP "NAME"') und füllt Register Q genauso, wie es 'GTO' oder 'XEQ' täte.

Eine weitere nützliche Abkürzung, die von Register Q Gebrauch macht, stammt von Clifford Stern und wurde schon in Abschnitt 4B angesprochen. Man löscht Q durch 'XEQ, ALPHA, ←' (dies kann gleich im PRGM-Modus geschehen) und ruft dann den Q-Boten auf.

Dadurch gelangt ein 'E' (schnelle Eins) und ein 'TEXT 0'-Byte ('NOP') ins Programm. Ruft man im RUN-Modus 'eGØBEEP 77' bei gelöschtem Q auf, wird das gegenwärtige Programm ausgedruckt.

Register ⊢ (in Programm-Auflistungen erscheint es mit dem Symbol ⊤) enthält in seinen ersten 4½ Bytes eine 36 Stellen umfassende Bitgruppe – sie soll im folgenden *Bitbild* heißen – für die Tastenzuweisungen auf nicht-umgeschaltete Tasten. Dieses Bitbild ist Bestandteil eines pfiffigen Verfahrens, dessen sich das Betriebssystem des HP-41 bedient, um den Zugriff auf vom Tastenfeld aus aufgerufene Funktionen und Programme zu beschleunigen. Wenn eine nicht-umgeschaltete Taste *im USER-Modus* gedrückt wird, prüft der Prozessor das zu dieser Taste gehörige Bit im Bitbild. Ist es gelöscht, weiß er, daß die Taste keine Zuweisung trägt. Dann wird folgende Fallunterscheidung vorgenommen: Liegt die Taste nicht in den beiden oberen Tastenreihen (Tasten 'A' bis 'J'), wird die *Säumnisfunktion* (darunter versteht man die der Taste aufgeprägte Funktion, die im USER-Modus als 'Voreinstellung' wirksam wird) ausgeführt. Handelt es sich aber um eine Taste aus den beiden obersten Reihen, wird zunächst das gegenwärtige Programm nach der zugehörigen lokalen Marke ('A' bis 'J') durchsucht. Findet sich eine solche Marke, beginnt dort die Ausführung des Programms. Bleibt die Suche jedoch erfolglos, sorgt der Prozessor schließlich (!, d.h. nach u.U. entnervend langer Suchzeit) für die Ausführung der Säumnisfunktion.

Ist das Bit im Bitbild dagegen gesetzt, weiß der Prozessor, daß die Taste eine Zuweisung erhalten hat. Er sucht dann als erstes in den Tastenzuweisungsregistern nach einer solchen. Findet er dort keine Zuweisung, prüft er die Tastenzuweisungsbytes (das sind die jeweils an 4. Stelle liegenden Bytes) der globalen Marken im Katalog 1 vom '.END.' an aufwärts bis zum Vorhang, der die Datenregister von den Programmregistern trennt. Findet er auch im Katalog 1 keine Marke, die der gedrückten Taste zugewiesen wurde (dies ist jedenfalls nicht der Normalfall), greift er zu einer Art 'Verlegenheitslösung' und führt 'ABS', manchmal auch 'CAT' oder '1/X', vorgewiesen mit abwegigen Pseudo-XROM-Zahlen wie '7,10', '7,53' o.ä. (vgl. Abschnitt 4C), aus.

Dank des Bitbildes der Tastenzuweisungen ist der erste Schritt des voranstehend beschriebenen Verfahrens sehr schnell. Doch kann die Suche nach lokalen Marken sehr viel Zeit in Anspruch nehmen, insbesondere dann, wenn das gegenwärtige Programm sehr lang ist. Daher ist es zu empfehlen, gelegentlich Säumnisfunktionen wie 'X<>Y' oder 'RDN' ihren eigenen Tasten zuzuweisen. Solche scheinbar unnützen Zuweisungen bewirken, daß man, ohne den USER-Modus verlassen zu müssen, diese Funktionen vom Rechner unter Vermeidung der zeitraubenden, fruchtlosen Suche nach lokalen Marken ausgeführt bekommt.

Die verbleibenden 2 1/2 Bytes rechts im Register ⊢ enthalten die Hexadezimalverschlüsselung der zuletzt über das Tastenfeld ausgeführten Funktion. Auch der Drucker bedient sich zeitweilig dieser Speicherstellen.

Die *Register a und b* enthalten den Adreßzeiger und den Stapel der Rücksprungadressen. Jede Adresse beansprucht zwei Bytes für vier Hexadezimalziffern. In den Bytes 1 und 0 von b liegt der jeweils gültige Wert des Adreßzeigers, das ist die Adresse des Befehls, der (im PRGM-Modus) gerade abgearbeitet wird oder (im RUN-Modus) zur Ausführung ansteht. Sobald das Programm einem 'XEQ'-Befehl begegnet, wird dessen Adreßwert nach links in den Rückkehrstapel, also in die Bytes 2 und 3 von b, geschoben. Der Adreßzeiger nimmt die Adreßwerte der Befehle des angesprungenen Unterprogramms auf. Wird ein neues 'XEQ' vor dem 'RTN' des

ersten angetroffen, werden der gerade gültige Wert des Adreßzeigers und die vordem notierte Rücksprungadresse, also der Adreßwert des ersten 'XEQ', beide um zwei Bytes nach links verschoben. Auf diese Weise können im Rücksprungstapel in a und b bis zu sechs wartende Rückkehradressen, darunter ganz vorn in a der ursprüngliche für das erste 'XEQ' gültige Wert des Adreßzeigers, untergebracht werden.

Trifft der Prozessor auf einen 'RTN'-Befehl, prüft er den Inhalt der Bytes 2 und 3 von b. Ist dieser Wert 0, wird der Zustand des Adreßzeigers 'eingefroren' und die Kontrolle an das Tastenfeld und damit an den Benutzer zurückgereicht. Andernfalls wird der Rückkehrstapel um zwei Bytes nach rechts verschoben, so daß die zuletzt aufgezeichnete Rücksprungadresse in den Adreßzeiger gelangt. Folglich wird die Programmausführung mit dem Befehl fortgesetzt, der sich hinter dem zuletzt angetroffenen 'XEQ', welches die Einspeisung seines eigenen Adreßwertes in den Rückkehrstapel verursachte, befindet.

Wir wollen uns nun die Einzelheiten der Adreßverschlüsselung ansehen. Die vier Hexadezimalziffern, aus denen eine Adresse besteht, werden für RAM-Adressen (der Hauptspeicher des HP-41 ist ein Speicher mit wahlfreiem Zugriff und wird im Englischen mit *r*andom *a*ccess *m*emory bezeichnet) und ROM-Adressen (die einsteckbaren Module enthalten Speicher, aus denen nur gelesen werden kann: *r*ead *o*nly *m*emory) verschieden interpretiert. In RAM-Adressen stellen die ersten 4 Bits die Byte-Nummer innerhalb eines Registers dar, während die restlichen 12 Bits die absolute Adresse eines Registers, gerechnet vom Fuß des Gesamtspeichers aus, bezeichnen. Das Format lautet demnach:

$$0B_1B_2B_3 \quad 000R_1 \quad R_2R_3R_4R_5 \quad R_6R_7R_8R_9.$$

$B_1B_2B_3$ ist die Byte-Nummer, für die drei Bits (B_1, B_2, B_3) zur Darstellung ausreichen, weil der höchste Wert einer Byte-Nummer bei 6 ($\triangleq$0110) erreicht ist. $R_1R_2R_3R_4R_5R_6R_7R_8R_9$ ist die Register-Nummer, für die 9 Bits (R_1,...,R_9) zur Darstellung ausreichen, weil der höchste Wert einer Register-Adresse im Hauptspeicher bei 511 ($\triangleq$0001 1111 1111) erreicht ist. Beispielsweise lautet die Adresse des 6. Bytes vom 431. Register in Hexadezimalzifferndarstellung 51AE ($1AE_{16} \triangleq 430_{10}$). Ihre Binärgestalt ist 0101 0001 1010 1110. Sobald ein Programm läuft oder einzelschrittweise abgearbeitet wird, ändert sich der Wert des Adreßzeigers, und zwar 'abwärts' in Richtung absteigender Byte-Nummern 6 bis 0 und absteigender Register-Nummern. So ist etwa 61AE die Adresse eines Befehls oberhalb von 41AE und 41AE die eines Befehls oberhalb von 41AD.

RAM-Adressen nehmen außerdem im Rückkehrstapel ein anderes Format als im Adreßzeiger an. Sie werden 'verdichtet', indem die drei Bits, welche zur Byte-Nummer gehören, nach rechts verschoben werden, wo, wie wir eben festgestellt haben, stets 3 Bits unbenutzt bleiben, weil 9 Bits für die Darstellung einer Register-Nummer ausreichen. Das Format einer RAM-Adresse, die Teil des Rückkehrstapels ist, lautet demgemäß

$$0000 \quad B_1B_2B_3R_1 \quad R_2R_3R_4R_5 \quad R_6R_7R_8R_9.$$

ROM-Adressen sind anders aufgebaut. Sie bestehen aus einer Steckplatz-Adresse, die die ersten 4 Bits besetzt, und einer 12 Bit langen Adresse innerhalb des Steckplatzes:

$$S_1S_2S_3S_4 \quad A_1A_2A_3A_4 \quad A_5A_6A_7A_8 \quad A_9A_{10}A_{11}A_{12}.$$

Die Steckplatz-Adresse als Teil der ROM-Adresse stimmt nicht mit der Steckplatz-Nummer (deren gibt es vier für die vier Einschubbuchsen) überein.

Die Zusammengehörigkeit von Steckplatz-Adresse einerseits und Steckplatz-Nummer sowie den überdies vorhandenen internen ROMs und anschließbaren Peripheriegeräten andererseits zeigt die folgende Liste:

Steckplatz-Adresse	physikalischer Steckplatz
0	internes ROM 0
1	internes ROM 1
2	internes ROM 2
3	unbenutzt 41CX
4	Diagnose-Modul (Kundendienst)
5	Time-Modul
6	Drucker (82143A und 82162A)
7	Massenspeicher- und HP-IL-Funktionen
8	Steckplatz Nr. 1, untere 4K
9	Steckplatz Nr. 1, obere 4K
A	Steckplatz Nr. 2, untere 4K
B	Steckplatz Nr. 2, obere 4K
C	Steckplatz Nr. 3, untere 4K
D	Steckplatz Nr. 3, obere 4K
E	Steckplatz Nr. 4, untere 4K
F	Steckplatz Nr. 4, obere 4K

Auf jede Steckplatz-Adresse paßt ein ROM von höchstens 4K = 4096 Bytes. Sobald ein Programm in einem ROM abläuft, ändert sich der Wert des Adreßzeigers in Richtung aufwärts steigender Adressen, wobei die letzten 12 Bits einen Wert zwischen hexadezimal 0 und hexadezimal FFF annehmen.

Eine wichtige beim Umgang mit dem Adreßzeiger zu beachtende Eigenheit des Betriebssystems ist folgende: Wenn man im RUN-Modus ein 'RCL b' über das Tastenfeld ausführt, erhält man den Adreßwert des Befehls, der dem Befehl, welcher im PRGM-Modus angezeigt würde, unmittelbar vorangeht! Wenn beispielsweise ein 'RCL M' in den Bytes 6 und 5 des Registers 1AE legt, und 'RCL b' vom Tastenfeld aus auf dieser Programmzeile ausgeführt wird, erhält man den Adreßwert 01AF, also einen Wert, der genau oberhalb der Stelle, wo 'RCL M' beginnt, liegt. Gibt es überdies 'NULL'en zwischen 'RCL M' und dem vorangehenden Befehl, ist der Adreßzeigerwert, den 'RCL b' nach X befördert, noch weiter von 'RCL M' entfernt: man

erhält den Adreßwert des ersten Nicht-'NULL'-Bytes oberhalb des angezeigten 'RCL M'. Anders hingegen – auch dies muß genau beachtet werden (vgl. Anhang B) –, wenn 'RCL b' durch ein laufendes Programm oder im RUN-Modus mit 'SST' ausgeführt wird: in diesem Fall bringt 'RCL b' seinen eigenen Adreßwert, das ist der Wert des zweiten Bytes dieses Befehls, ins X-Register.

Schließlich ist noch auf einen anderen Umstand beim Umgang mit RAM- und ROM-Adressen hinzuweisen. Da der HP-41 manche RAM-Adressen nicht von ROM-Adressen unterscheiden könnte, wenn er sich nur an den Adreßzeiger hielte, muß er intern Gebrauch von einer Art Flag, das dem Benutzer unzugänglich ist, machen, damit er weiß, ob der Adreßzeiger auf RAM- oder ROM-Adressen weist. Infolgedessen ist das nicht möglich, was man vielleicht bei oberflächlicher Betrachtung glaubt vorteilhafterweise tun zu können, nämlich sich beliebige Adressen aus ROMs mit 'RCL b' zu 'besorgen' und bei Bedarf vom RAM aus mit 'STO b' anzuspringen. Führt man 'STO b' vom Tastenfeld her oder über ein Programm aus, kann man damit nur von RAM- zu RAM- oder von ROM- zu ROM-Adressen springen, je nachdem, in welchem Speichertyp man sich gerade befindet – das interne Flag verbleibt beharrlich in seinem Zustand. Einen völlig anders gearteten Weg, beliebige ROM-Stellen vom RAM aus anzuspringen, beschreitet W.C. Wickes in 'Synthetische Programmierung auf dem HP-41C/CV', S. 113–116.

Register c enthält die entscheidenden Informationen über die Aufteilung des Hauptspeichers. Wir beziehen uns bei den folgenden Erläuterungen auf die Abbildung 6.3, wobei wir die Inhalte von c von rechts nach links gehend beschreiben werden.

Die äußersten rechten 1 1/2 Bytes von c bilden den Zeiger für das ständige '.END.', welches das Ende des Programmspeichers markiert. Dieser Zeiger enthält die Hauptspeicheradresse des Registers, in welchem das '.END.' gerade liegt, in Form dreier Hexadezimalziffern. Dabei gilt folgende Regel: Das '.END.' liegt stets in den letzten drei Bytes eines Registers, so daß zwischen dem letzten Befehl des letzten Programms und dem '.END.' bis zu 6 'NULL'en liegen können, die nicht durch 'PACK' zu beseitigen sind.

Die nächsten 1 1/2 Bytes bilden den Zeiger für den Vorhang, also die Trennlinie zwischen Daten und Programmen im Hauptspeicher. Er enthält in Form dreier Hexadezimalziffern die Hauptspeicheradresse des Registers R_{00}. Jedesmal, wenn man die Datenregisteranzahl mit 'SIZE' verändert, wird der Zeigerinhalt der neuen Speicheraufteilung angepaßt, und die Speicherinhalte werden entsprechend verschoben. Im Laufe der Zeit sind viele Programme ersonnen worden, um den Vorhang unter Umgehung von 'SIZE' zu bewegen und auf diese Weise Programmschritte in Daten und umgekehrt zu verwandeln. In Abschnitt 6C wird Ihnen ein solches Programm zusammen mit einer genauen Beschreibung der beim Umgang mit dem Vorhang zu beachtenden Vorsichtsmaßregeln begegnen. In LB und MK gibt es Befehlsfolgen, die den Zeiger für den Vorhang kurzfristig auf $010_{16} = 16_{10}$ setzen. Dadurch erlangt man Zugriff auf die Programmspeicher- und Tastenzuweisungsregister mit Befehlen wie 'STO IND' und 'RCL IND'. 'RCL' normalisiert natürlich die Registerinhalte. Der ursprüngliche Inhalt von c wird daher zweckmäßig im Stapel oder in anderen Zustandsregistern aufbewahrt, um ihn vor Normalisierung zu schützen und vor einem Programmstop unversehrt nach c zurückbringen zu können. "LB" und "MK" führen vor, welch ungewöhnliche Möglichkeiten sich für die Programmgestaltung anbieten, wenn man den Vorhang im Griff hat.

Die nächsten drei in c enthaltenen Hexadezimalziffern bilden die unter der Bezeichnung *Kaltstart-Konstante* bekannte Zahl 169_{16}. Sie ist nach bisherigen Erkenntnissen jedem bislang hergestellten HP-41 (den neuen HP-41CX eingeschlossen) eigen. Stellt der Prozessor jemals

fest, daß diese Ziffern verändert sind, führt er eine Totallöschung durch und zeigt "MEMORY LOST" an. Die Überlegung, die hinter diesem erbarmungslos anmutenden Vorgehen steckt, beruht auf der Tatsache, daß die Kaltstart-Konstante niemals vom Rechner selbst verändert wird, jede Abweichung also auf Fehler, insbesondere Stromfehler, hinweist. (Zugeständnisse an die im Verlaufe der Evolution erst spät aufgetauchte Spezies der Synthetischen Programmierer wurden und werden nicht gemacht. Ihren Irrtümern läßt man keine Fürsorge angedeihen.) Es könnten, so sagt sich der Prozessor beim Anblick einer zerstörten Kaltstart-Konstante, auch Fehler in anderen Teilen des Speichers aufgetreten sein. So zieht er also die Notbremse und bewahrt damit den von jedem Mißtrauen freien Benutzer davor, ahnungslos irrige Ergebnisse zu erhalten. Beim Umgang mit Register c lebt man also stets unter dem Damoklesschwert des "MEMORY LOST". Es gilt daher der eherne Grundsatz: Lade nichts ins Register c, wenn sich nicht die drei Hexadezimalziffern 1, 6 und 9 an Ort und Stelle befinden. Beiläufig bemerkt führt auch das Setzen des Zeigers für den Vorhang zu "MEMORY LOST", wenn das Register unterhalb der neu festgesetzten Trennlinie gar nicht existiert. Seien Sie also äußerst vorsichtig, wenn Sie etwas nach c bringen.

Die äußersten linken 1 1/2 Bytes von c bilden den Zeiger für den Statistikblock. Steht beispielsweise der Zeiger für den Vorhang auf 1EB (das entspricht bei voll ausgebautem Hauptspeicher der Festsetzung 'SIZE 021') und wird 'ΣREG 01' ausgeführt, so steht der Zeiger für 'ΣREG' auf 1EC, also 1EB+1.

Die aus zwei Halbbytes bestehende Speicherplatzstelle, die sich in c zwischen dem Zeiger für 'ΣREG' und der Kaltstart-Konstante befindet, wird anscheinend weder vom Betriebssystem noch vom Drucker benutzt.

Register d ist die Heimat der 56 Flags. Das äußerste linke Byte, Byte 6, enthält die Flags 0 bis 7. Diese Ordung setzt sich folgerichtig nach rechts fort, so daß in Byte 0 die Flags 48 bis 55 liegen.

Das Flagregister ist das Hauptlabor der synthetischen Programmierung. Solange es den X-Funktionen-Modul noch nicht gab, konnten die meisten Bit-Manipulationen nur dadurch vorgenommen werden, daß man ein oder mehrere Bytes, die behandelt werden sollten, ins Flag-Register brachte, in welchem man mit den Flag-Befehlen die ersten 30 Flags, Flag 00 bis Flag 29, beherrscht. Ein Musterbeispiel dieser Technik ist das Programm "RAMBYT" in Kapitel 4. Dort und auch in vielen synthetischen Routinen des PPC ROMs finden Sie häufig den Befehl 'X<>d' paarig auftreten, stets getrennt durch mehrere Zeilen, in denen Flags geschickt den jeweils beabsichtigten Zielen entsprechend umgeschaltet werden.

Register e enthält in seinen ersten 4 1/2 Bytes das Bitbild für die Tastenzuweisungen auf umgeschaltete Tasten. Dieses Bitbild entspricht genau dem, welches wir bei Register ⊦ für die nicht-umgeschalteten Tasten kennengelernt haben. Die dort für die Ausführungshierarchie bezüglich der Tasten 'A' bis 'J' getroffenen Feststellungen gelten hier in entsprechender Weise für die Tasten 'a' bis 'e'.

Die nächsten beiden Halbbytes, die Hälfte von Byte 2 und die Hälfte von Byte 1, werden vom Prozessor für Notizen verwendet.

Die letzten 1 1/2 Bytes von Register e enthalten die Programmzeilennummer. Weil diese Zeilennummer nicht Bestandteil der im Programmspeicher abgelegten Befehle ist, muß sie der Prozessor bei Bedarf stets neu berechnen. Und weil die Befehlslängen überdies nicht konstant sind, vielmehr zwischen 1 und 16 variieren können, alle Zwischenwerte eingeschlossen, ist diese Berechnung umständlich, mithin zeitraubend.

Dies nimmt man immer dann schmerzlich wahr, wenn man bei einem auf höherer Zeilennummer anhaltenden Programm in den PRGM-Modus schaltet oder wenn man von dort aus mit 'GTO .mnl' oder 'BST' auf eine niedrigere Zeilennummer springt (vgl. auch die Ausführungen zu 'BST' am Ende von Kapitel 5). Der Zeitaufwand für die Berechnung ist der Grund dafür, daß sie nicht in einem laufenden Programm durchgeführt wird. Die Geschwindigkeit der Programmausführung würde dadurch erheblich ~~erhöht~~ verringert. Als Preis zahlt man die Wartezeit beim Umschalten in den PRGM-Modus nach beendetem Lauf. Der Prozessor weist die Nummer der Zeile, auf der das Programm angehalten hat (genaugenommen die Nummer der Folgezeile), erst dann vor, wenn er sie ausgerechnet hat. (Ist der Programmlauf beim 'END' beendet worden, kann man eine Zeitverzögerung natürlich nicht wahrnehmen, weil die 'berechnete' Zeilennummer in diesem Fall 00 lautet.) Woher weiß der Prozessor, daß die Neuberechnung nötig ist? Ganz einfach: Sobald ein Programmlauf beginnt (eine 'SST'-Ausführung ist kein Programmlauf im eigentlichen Sinne), wird die Zeilennummer im Register e auf $FFF_{16} = 4095_{10}$ gesetzt. Sie bleibt solange unverändert in diesem Zustand, wie kein 'SST' ausgeführt oder keine Umschaltung in den PRGM-Modus vorgenommen wird. Geschieht dies aber, erkennt der Prozessor am Wert FFF, daß er die zum gültigen Adreßzeigerwert gehörige Zeilennummer neu berechnen muß, und zwar vom letzten 'CAT 1'-Element aus (vgl. die schon erwähnten Ausführungen in Kapitel 5).

Die bei der Erzeugung des Byte-Schnappers in Kapitel 1 überraschend aufgetretene Zeilennummer 4094 erklärt sich aus den vorstehend beschriebenen Regeln: In dem Moment, in dem Sie bei der in Rede stehenden Prozedur die Korrekturtaste im ALPHA-Modus drücken (Punkt 6), vermindert der Prozessor die Zeilennummer um 1, ohne zu bemerken, daß FFF keinen gültigen Wert bildet, sondern nur Signalwert hat. Der Befehl 'RCL 01', welcher bei dem Verfahren auftaucht, ist ein Phantom-Befehl, der sich ergibt, wenn Register b den Wert 0 enthält.

6B. Zustandsregister – Anwendung 1: Aufhebung der Tastenzuweisungen

Zu den zwischen dem HP-41 und dem HP-67 übereinstimmenden Eigenschaften – notwendig für die Übertragungsfähigkeit bestehender Programme vom HP-67 auf den HP-41 – 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 angenehme Eigenschaft leider dann nicht zutage, wenn diesen Tasten globale Marken oder System-Funktionen zugewiesen sind, weil der HP-41 der Ausführung der letzteren den Vorrang gibt (vgl. die Ausführungen zu Register ⊦). So mancher Benutzer hat schon vertrauensvoll mit der automatischen Zuweisung der lokalen Marken gerechnet, doch waren plötzlich globale Marken oder System-Funktionen im Wege. Man drückt hoffnungsfroh 'LOG', um 'LBL D' auszuführen, und findet sich unversehens in einem anderen Programm wieder o.ä. Es wäre wohl vorteilhaft, solch vorrangige Zuweisungen vorübergehend aufzuheben und nach beendeter Tätigkeit zurückholen zu können. Die synthetische Programmierung bringt es zustande: Der PPC ROM enthält zwei Routinen, die das Erwünschte leisten. Mit SK kann man die vorrangigen Zuweisungen von Funktionen und globalen Marken aufheben und mit RK zurückholen.

Vor dem Gebrauch von SK legt man eine Registeradresse k nach X. Dann führt man 'XEQ SK' aus. Die in den Registern ⊦ und e liegenden Bitbilder der Tastenzuweisungen werden in den

Registern R_k und R_{k+1} untergebracht und die entsprechenden Speicherbereiche in ⊢ und e gelöscht. Weil der Prozessor jetzt keine gesetzten Bits mehr dort antrifft, nimmt er an, daß keine Zuweisungen vorhanden sind. Folglich führt jeder Druck auf eine Taste der beiden oberen Reihen – bei der ersten Reihe natürlich auch ein Druck nach Umschaltung – zur Suche nach der zugehörigen lokalen Marke. Die ursprünglichen Zuweisungen sind aber nicht 'tot'. Man kann sie ganz einfach wiederbeleben, indem man 'XEQ RK' mit k in X ausführt. Dann werden die Inhalte von R_k und R_{k+1} in die Zustandsregister ⊢ und e zurückgebracht: die alten Bitbilder sind wieder an Ort und Stelle, und die Tastenzuweisungen können wie zuvor benutzt werden.

Es gibt noch einen zweiten ganz anderen Weg, die Zuweisungen zurückzugewinnen: Man liest ganz einfach über den Kartenleser eine beliebige Spur einer beliebigen Programmkarte ein. Es spielt keine Rolle, ob der USER-Modus dabei aktiv ist oder nicht. Nur eine Programmkarte muß es sein. Diese Möglichkeit ist eine brauchbare Hilfe, falls die Inhalte von R_k und R_{k+1} zufällig einmal verloren gegangen sein sollten, nachdem Sie SK ausgeführt haben. RK hülfe dann nämlich auch nicht mehr.

Wir wollen uns jetzt die Arbeitsweise der PPC ROM Routinen SK und RK (*s*uspend resp. *r*eactivate *k*ey assignments) genau ansehen und die einzelnen Schritte verfolgen. Wenn Sie keinen PPC ROM, benutzen Sie "LB", um "SK" und "RK" einzutasten:

```
01♦LBL "SK"                 "LB"-Eingaben:
02 SIGN
03 CLX
04 X<> '                    206, 122
05 XEQ 14
06 ISG L
07 ""                       240
08 ,
09 X<> e                    206, 127
10♦LBL 14
11 "*"
12 X<> [                    206, 117
13 STO \                    145, 118
14 ASTO IND L
15 RDN
16 RTN
17♦LBL "RK"
18 SIGN
19 ARCL IND L
20 "⊢♦"                     242, 127, 0
21 ISG L
22 ""                       240
23 ARCL IND L
24 "⊢‡"                     243, 127, 15, 255
25 X<> \                    206, 118
26 STO '                    145, 122
27 X<> [                    206, 117
28 STO e                    145, 127
29 RDN
30 CLA
31 END
                                          64 BYTES
```

Formblatt zur Stapel- und Alpha-Analyse

Zeile	Befehl	L	X	Y	Z	T	P	Q	N	M
53	LBL"SK"		x	y	z	t				
54	SIGN	x	1							
55	CLX		0							
56	X<>⊢		⊢							
57	XEQ 14									
62	LBL 14									
63	"*"						gelöscht	gelöscht	gelöscht	*
64	X<> M		------*							1 2 3 5 6 7
65	STO N								*	
66	ASTO IND L									
67	RDN		y	z	t	------*				
68	RTN									
58	ISG L	x+1								
59	"" (NOP)									
60	,		0	y	z	t				
61	X<> e		e							
62	LBL 14									
63	"*"						gelöscht	gelöscht	gelöscht	*
64	X<> M		------*							8 9 10 11 12 13 14
65	STO N								*	
66	ASTO IND L									
67	RDN		y	z	t	------*				
68	RTN									
84	LBL"RK"		x	y	z	t				
85	SIGN	x	1							
86	ARCL IND L									* 1 2 3 4 5
87	"⊢-"									* 1 2 3 4 5 -
88	ISG L	x+1								
89	"" (NOP)									
90	ARCL IND L								* 1 2 3 4 5 -	* 8 9 10 11 12
91	"⊢Φ"							* 1 2 3 4 5 - *	8 9 10 11 12 0F FF	
92	X<> N		⊢'					* X - - - - - -	8 9 10 11 12 0F FF	
93	STO ⊢									
94	X<> M		e'					* X - - - - - -	1 2 3 4 5 - *	
95	STO e									
96	RDN		y	z	t	e'				
97	CLA						gelöscht	gelöscht	gelöscht	gelöscht
98	RTN									

Formblatt zur Stapel- und Alpha-Analyse

Zeile	Befehl	L	X	Y	Z	T	P	O	N	M

Die nachstehend abgedruckten 'Formblätter zur Stapel- und Alpha-Analyse' bilden ein unerläßliches Hilfsmittel, wenn man die i.a. sehr unübersichtlichen und verwirrenden Bewegungen der Speicherinhalte in synthetischen Programmen Schritt für Schritt verfolgen will. Sie werden dies erkennen, sobald Sie mir durch SK und RK gefolgt sind:

Nach dem Aufruf von SK wird als erstes die Registeradresse k durch 'SIGN' ins Register L gebracht. Dann holt 'X< >⊢' das Bitbild aus ⊢ ins X-Register, wobei gleichzeitig ⊢ gelöscht wird. Das unter 'LBL 14' stehende Unterprogramm benutzt 'ASTO', um eine 6 Zeichen lange Kette ins Register R_k zu bringen. Sie besteht aus einem "*", dem die ersten 5 Bytes aus ⊢, also das vollständige Bitbild, folgen. "*" wird als 'Platzhalter' verwendet, weil das äußerste linke Byte aus ⊢ 'NULL' sein könnte. Die drei Befehle '"*", X< >M, STO N' bereiten den Inhalt des Alpha-Registers für das Wirken des Befehls 'ASTO' vor. Im Formblatt läßt sich das deutlich verfolgen. Nehmen Sie sich im Zweifelsfall Zeit, diese Befehlsfolge genau zu verstehen, wenn Sie selbst synthetische Programme schreiben wollen.

Der Rest von SK arbeitet ganz ähnlich, um das Bitbild aus e als Teil einer Zeichenkette nach R_{k+1} zu bringen und e leer zurückzulassen.

Wenn Sie RK aufrufen, wird als erstes wieder k durch 'SIGN' nach L gebracht. Die Zeichenkette aus R_k gelangt mit 'ARCL' ins Alpha-Register und wird durch Anhängen einer 'NULL' um ein Byte nach links verschoben; ein "*" täte es ebensogut. Zwei zusätzliche Bytes, hexadezimal 0F FF, werden angehängt, wodurch ein weiteres Verschieben nach links bewirkt wird. Im Formblatt können Sie all diese Bewegungen genau erkennen. Jetzt enthält Register N das ins Register ⊢ gehörige Bitbild und Register M dasjenige für Register e. Die letzten Zeilen von RK holen die Inhalte von N und M nach X und bringen sie von dort aus nach ⊢ und e. Zum Schluß werden der Stapel zurückgesetzt und das Alpha-Register gelöscht. Beachten Sie, daß die letzten beiden Bytes von e auf hexadezimal 0F FF gesetzt werden, damit der Prozessor richtige Zeilennummern berechnen kann. Frühere Fassungen von RK verwendeten 00 00 statt 0F FF, wodurch falsche Zeilennummern vorgewiesen wurden, wenn man mit 'SST' durch das Programm schritt oder es im TRACE-Modus laufen ließ.

6C. Zustandsregister – Anwendung 2: Umbenennung der Registeradressen

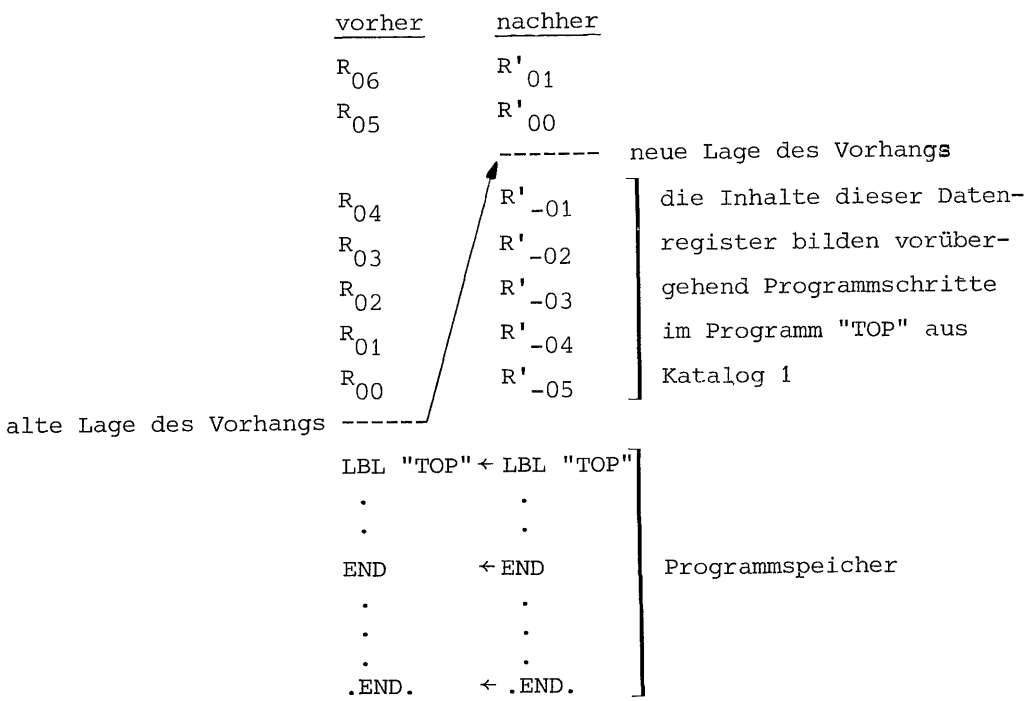
Nehmen Sie an, Sie hätten ein Programm, das auf eine von Ihnen erstellte Unterroutine zugreift. Ein typisches Beispiel für einen solchen Fall wäre ein Programm N zur Nullstellensuche: gesucht x mit f(x) = 0. Hierfür wird f(x) in einer vom Benutzer zu schreibenden Routine berechnet. Der Benutzer benennt die Routine und stellt ihren Namen dem Programm in einem Register R_{mn} zur Verfügung. N verzweigt auf die Benutzerroutine mit 'XEQ IND mn'.

Schreibt man ein Programm zur Nullstellensuche – es heiße N –, muß man eine nicht leicht zu treffende Entscheidung fällen. Und das so: Das Programm N braucht numerische Datenregister, in denen Zwischenergebnisse gespeichert werden, die von der Routine, welche f(x) berechnet – sie heiße R –, nicht zerstört werden dürfen. Welche Register man nun für N auch immer wählt, stets besteht die Gefahr, daß zwischen der Registerbenutzung von N und der von R Überschneidungen auftreten. Man kann versuchen, dem aus dem Wege zu gehen, indem man

für N die Register von sagen wir R_{50} an aufwärts benutzt, in der berechtigten Erwartung, daß vernünftige Programme R zur Berechnung von f(x) diese Register nicht benötigen. Selbst wenn man mit dieser Überlegung richtig liegt, geht man mindestens höchst verschwenderisch mit Speicherplatz um, denn andererseits wird es kaum einmal eine Routine R geben, die auch nur annähernd 50 Datenregister braucht.

Synthetische Programmierung befreit uns aus dieser Zwangslage. Eine kurze synthetische Routine vermag den Vorhang, der die Daten- von den Programmregistern trennt, zu versetzen, wodurch eine Umbenennung der Registeradressen, das ist eine echte Umnumerierung der Datenregister, erfolgt.

Nehmen Sie z.B. an, daß für N die fünf Register R_{00} bis R_{04} benötigt werden. Unmittelbar vor Aufruf von R ruft N die synthetische Routine "CU" (*c*urtain *u*p) auf, um den Vorhang fünf Register höher 'aufzuhängen'. Die untenstehende Abbildung veranschaulicht die Wirkung dieses Eingriffs in Register c. Obwohl die Inhalte der Datenregister keine Änderung erfahren haben, fördert ein 'RCL 00' nach Versetzen des Vorhangs den Inhalt des Registers zutage, das vor dem Versetzen R_{05} hieß.



Genauso liefert 'RCL 01' den Inhalt des vormaligen R_{06} usw. Die für die Zwecke von N schutzbedürftigen Inhalte der vormaligen Register R_{00} bis R_{04} sind für 'STO'- und 'RCL'-Befehle unerreichbar geworden, so daß R ihnen nichts mehr anhaben kann. Sie liegen vorübergehend in dem Teil des Hauptspeichers, der die Programme enthält. Würden Sie jetzt in das erste Programm aus Katalog 1 schalten, fänden Sie diese Daten oberhalb der Kopfmarke des ersten Programms. Natürlich würden Sie als Programmbefehle (i.a. in sinnloser Zusammenstellung), nicht etwa als Zahlen, in der Anzeige erscheinen.

Der entscheidende Punkt bei diesem Verfahren ist der, daß das Programm N nach dem Versetzen des Vorhangs die Routine R sorgenfrei aufrufen kann: R ist überhaupt nicht mehr in der Lage, die Daten von N zu zerstören, und darum völlig unbehindert in der Auswahl von Registeradressen für eigene Zwecke.

Sobald R die Kontrolle an N zurückgibt, muß dieses seine Daten wieder 'befreien', indem es den Vorhang in die ursprüngliche Lage versetzt. Dadurch wird die alte Registerbenennung wiederhergestellt, so daß N auf die unversehrten Inhalte von R_{00} bis R_{04} zurückgreifen kann.

Die nachstehende Programm-Auflistung der Routine "CU" zusammen mit einem typischen Programm zur Nullstellensuche, "SOLVE" genannt, veranschaulicht das oben beschriebene Verfahren. Die hier gegebene Fassung von "CU" stammt von Tapani Tarvainen und stellt gegenüber früheren Varianten einen Durchbruch im Hinblick auf Eleganz und Byte-Verbrauch dar.

```
01*LBL "SOLVE"      14 RCL 00      27 RCL 02      40 BEEP
02 "FNAME?"         15 LASTX       28 *           41 RTN
03 AON              16 XEQ 14      29 CHS         42*LBL 14
04 STOP             17 STO 01      30 STO 02      43 4
05 ASTO 00          18*LBL 10      31 RCL 03      44 XEQ "CU"
06 AOFF             19 RCL 00      32 +           45 XEQ IND Y
07 "XGUESS1?"       20 RCL 03      33 STO 03      46 4
08 PROMPT           21 XEQ 14      34 RCL 01      47 CHS
09 STO 03           22 ENTER↑      35 ABS         48 XEQ "CU"
10 "XGUESS2?"       23 ENTER↑      36  E-6       49 END
11 PROMPT           24 X<> 01      37 X<=Y?
12 -                25 -           38 GTO 10
13 STO 02           26 /           39 RCL 03                  97 BYTES
```

```
01*LBL "CU"         10 STO ]       19 FRC         28 X<> d
02 INT              11 RDN         20 X≠0?        29 STO [
03 RCL c            12*LBL 03      21 SF IND [    30 "├***"
04 STO [            13 FS?C IND [  22 DSE [       31 X<> \
05 "├***!"          14 ISG X       23 ABS         32 X<> c
06 RDN              15 ""          24 -           33 RDN
07 11               16 2           25 X≠0?        34 CLA
08 X<> [            17 /           26 GTO 03      35 END
09 X<> d            18 ENTER↑      27 X<> ]                  67 BYTES
```

"LB"-Eingaben für "CU":

Zeile 03: 144, 125 Zeile 04: 145, 117

Zeile 05: 245, 127, 0, 0, 0, 33

Zeile 08: 206, 117 Zeile 09: 206, 126 Zeile 10: 145, 119

Zeile 13: 170, 245 Zeile 15: 240 Zeile 21: 168, 245

Zeile 22: 151, 117 Zeile 27: 206, 119 Zeile 28: 206, 126

Zeile 29: 145, 117

Zeile 30: 244, 127, 0, 0, 0

Zeile 31: 206, 118 Zeile 32: 206, 125

Die Barcodes für "SOLVE" und "CU" finden Sie im Anhang E.

"SOLVE" fordert Sie zunächst zur Eingabe des Namens der Routine, die f(x) berechnet, auf und verlangt außerdem die Eingabe zweier Werte, von denen Sie vermuten, daß zwischen ihnen die gesuchte Nullstelle x der Funktion f(x), auch Wurzel der Gleichung f(x) = 0 genannt, liegt. Das Programm bedient sich dann der sogenannten Newtonschen Methode, um die Wurzel zu suchen. Im Verlauf der Rechnung muß f(x) für mehrere Werte von x berechnet werden. Jede Berechnung von f(x) wird durch die Unterroutine 'LBL 14' durchgeführt, die den Vorhang erst 4 Register höher setzt, dann die Benutzerroutine aufruft und danach den Vorhang wieder an die alte Stelle bringt.

"CU" hebt den Vorhang um die durch den Wert in X bestimmte Anzahl von Registern. Ist dieser Wert negativ, wird der Vorhang entsprechend gesenkt. Der Inhalt zweier Stapelregister bleibt erhalten: die vor der Ausführung von "CU" in Y und Z liegenden Inhalte findet man am Schluß in X und Y wieder. Diese Eigenschaft wird im Programm "SOLVE" dazu ausgenutzt, den Namen der Benutzerroutine und den aktuellen Wert von x im Stapel zurückzulassen. 'XEQ IND Y' reicht dann aus, um die Benutzerroutine mit dem richtigen Eingabewert zu bedienen.

Wir wollen "SOLVE"/"CU" mit dem folgenden Beispiel erproben: Zunächst 'GTO ..' und dann

```
01▸LBL "TEST"
02 1/X
03 LASTX
04 -
05 1
06 +
07 END
```

Diese kurze Routine berechnet f(x) = (1/x)−x+1. Durch Vergleich mit Aufgabe 2.4 können Sie sich davon überzeugen, daß die Lösung von f(x) = 0 durch den sogenannten

Goldenen Schnitt gegeben wird, weil f(x) = 0 gleichbedeutend mit x = 1+1/x ist. Führen Sie nun 'XEQ "SOLVE"' aus, und beantworten Sie die Eingabe-Aufforderungen

Aufforderung	Antwort
FNAME?	TEST (R/S)
XGUESS1?	1 (R/S)
XGUESS2?	2 (R/S)

Nach etwa 40 Sekunden ertönt der 'BEEP', und Sie erblicken 1,618033989. Dieses einfache Beispiel nutzt das Zusammenspiel von "SOLVE" und "CU" natürlich nicht richtig aus, aber Sie können sicher sein, daß beide Programme ebensogut arbeiten, wenn Sie eine Nullstelle einer beliebigen anderen Funktion g(x) suchen, unabhängig von scheinbaren Registerüberschneidungen. Natürlich unterliegt die Nullstellensuche den Einschränkungen, die durch das Newtonsche Verfahren selbst gegeben sind. Gewisse Funktionen mit bösartigem Verhalten als auch ungeschickt gewählte Anfangswerte können Probleme verursachen. Die aber sind dann mehr mathematischer als programmtechnischer Natur. In den meisten 'aus dem Leben gegriffenen' Fällen arbeitet "SOLVE" jedenfalls einwandfrei und schnell.

Beschränkungen bei der Benutzung von "CU"

1.) Während sich der Vorhang in nach oben versetzter Lage befindet, sind Datenregister zeitweilig Programmregister am Kopf des Programmspeichers. Daher werden ihre Inhalte nicht als Zahlen, sondern als Programmbefehle interpretiert. Weil sich unter diesen Befehlen Marken befinden können, darf man bei gehobenem Vorhang im ersten Programm nicht rückwärts auf lokale Marken springen, es sei denn, die 'GTO's und 'XEQ's sind kompiliert; aber dessen kann man i.a. nicht sicher sein.

2.) Bei gehobenem Vorhang darf auf keinen Fall der Programmspeicher ge'PACK't werden, denn es ist mehr als sehr wahrscheinlich, daß sich zwischen den zu bewahrenden Daten 'NULL'-Bytes, die beim 'PACK'en verloren gehen würden, befinden. Man kann sich gegen einen solchen Unfall leidlich zuverlässig schützen, indem man vor dem ersten Aufruf von "CU" 'PACK't. Das mit entsprechender Information versehene 'END' des ersten Programms hält den Prozessor dann davon ab, die 'NULL'en zu beseitigen. Sorgen Sie auch immer dafür, daß sich unterhalb vom '.END.' noch einige freie Register befinden, bevor Sie "CU" benutzen. Andernfalls könnte das Einfügen eines Programmschrittes, das Tätigen einer Tastenzuweisung oder das Einschreiben eines Weckauftrages einen unbeabsichtigten 'PACK'-Vorgang auslösen. Die Folgen wären gewiß unerfreulich.

3.) Wenn Sie den Vorhang wieder senken, dann stets in seine Ausgangslage. Sollten Sie zufällig einmal nicht mehr wissen, in welche Lage der Vorhang zurückgesetzt werden muß, können Sie in das erste Programm gehen und dort die oberhalb der Kopfmarke liegenden, aus Zahlen entstandenen Programm-fremden Befehle löschen (das zerstört natürlich die eigentlich zu schützenden Daten), um dann mit 'PACK' die Programme an die neue Position des Vorhangs zu schieben.

4.) Versetzen Sie den Vorhang nie so, daß er unmittelbar oberhalb eines nicht existierenden Registers zu liegen kommt. Eine Verbringung auf dezimal 16 ist in Ordnung, weil das Register mit der absoluten Adresse 15_{10} existiert – es ist Register e. Doch wenn Sie den Vorhang z.B. auf die absolute Adresse 17_{10} bringen, erhalten Sie "MEMORY LOST", weil es ein Register mit der absoluten Adresse 16_{10} nicht gibt. Zwar kann auch eine solche Klippe erfolgreich umschifft werden, nämlich dadurch, daß der Vorhang auf eine zulässige Stelle zurückgesetzt wird, bevor das Programm anhält ("MK" und "LB" tun dies), doch besser ist es, wenn Sie sich zunächst an die empfohlene Sicherheitsregel halten und erst später, wenn Sie genau wissen, was Sie tun, kühnen Programmierstil pflegen.

Die Umbenennung der Registeradressen mit "CU" kann auf mehreren Stufen geschehen: Ein Programm rufe "CU" auf, bevor es in ein zweites Programm (mit Rückkehrabsicht) verzweigt. Dieses zweite Programm vermag ebenfalls "CU" aufzurufen, bevor es in ein drittes Programm verzweigt. Das Verfahren ist lediglich durch die zulässige Anzahl von Unterprogrammebenen beschränkt. Sie können auf diese Weise mit einem mehrstufigen 'Datenregister-Stapel' arbeiten. Die entscheidende Befehlsfolge, die in den verschiedenen Programmen auftreten muß, um die Inhalte von k Registern vor der Zerstörung durch das nächste Unterprogramm zu schützen, lautet stets:

```
k
XEQ "CU"
XEQ Unterprogramm
-k
XEQ "CU"
```

Registerumbenennung durch programmgesteuerte Überwachung des Vorhangs führt zu ganz erheblichen Freiheiten bei der Programmgestaltung und -Benutzung. Wenn ein Programm z.B. die Datenregister R_{10} bis R_{19} beschickt, ist es dennoch ohne weiteres möglich, beim Start mit 'SIZE 10' zu arbeiten. Man braucht nämlich nur im Programm selbst den Vorhang rechtzeitig um 10 Register zu senken, weil dadurch die Register R_{00} bis R_{09} in R_{10} bis R_{19} umgewidmet werden. Allerdings sollte man nicht vergessen, den Vorhang anschließend wieder zu heben – ein unachtsames 'RCL 00' etwa tilgt sonst einen Teil Ihrer Programme.

"CU" von Tapani Tarvainen leistet genau dasselbe wie das im PPC ROM enthaltene CU von Bill Wickes. Die Routinen können darum wechselweise benutzt werden. Im PPC ROM liegen auch die wesentlich schnelleren Routinen HD, UD und Σ. Sie dienen ebenfalls zur Umsetzung des Vorhangs, unterliegen jedoch gewissen Beschränkungen bei der Benutzung. Sie sollten daher ihre Gebrauchsanweisung genau durchlesen, bevor Sie sie verwenden. Auskunft über die hinter den Programmen stehenden allgemeinen Überlegungen und die hier genannten Routinen im besonderen finden Sie im PPC CJ: Mai '80, S. 23 – Juni '80, S. 45 – Juli '80, S. 2 – März '81, S. 2. Die in den Artikeln des PPC CJ besprochenen Programme "MS" und "RS" sind frühere Fassungen von HD und UD. Das Handbuch zum PPC ROM enthält ebenfalls hilfreiche Informationen in den Gebrauchsanweisungen zu CU, HD, UD und Σ. Schließlich gibt der Anhang M dieses Handbuches noch eine ausführliche und alles zusammenfassende Beschreibung der Vorgänge beim Versetzen des Vorhangs.

Analyse von "CU"

Zunächst wird der Inhalt von Register c nach M, also in den äußersten rechten Teil des Alpha-Registers gebracht. Dann hängt Zeile 05 vier Bytes an. Danach enthält M die letzten drei Bytes von c, gefolgt von drei 'NULL'-Bytes und Byte 21_{16}. Folglich liegt der Zeiger für den Vorhang in den ersten 1 1/2 Bytes von M.

Als nächstes werden die Inhalte von M und d auf dem (notwendigen) Umweg über X gegeneinander ausgetauscht, wodurch der Zeiger in den Bereich von d, der die Flags 00 bis 11 enthält, zu liegen kommt. Daher wird man im Anschluß an Zeile 09 mit Sicherheit gelöschte Flags 00 und 01 beobachten, denn der Zeiger kann nie auf einen Wert größer als 512_{10} = 0010 0000 0000_2. weisen. Der ursprüngliche Flagszustand wird nach Register O gerettet, um ihn später nach d zurückbringen zu können, während gleichzeitig 11 als Kontrollzahl für eine weiter unten zu durchlaufende Schleife nach M gelangt.

Das rätselhaft anmutende Byte 21_{16} = 0010 0001_2 dient dazu, die Flags 50 und 55 zu setzen. Flag 50 unterbindet jede Bewegung von Meldungen in der Anzeige (man vgl. dazu die Beispiele auf S. 120/121 in 'Synthetische Programmierung auf dem HP-41C/CV' von W.C. Wickes sowie Beispiel 6 von IF im Handbuch des PPC ROMs). Flag 55 muß gesetzt werden, damit "CU" bei angeschlossenem Drucker gefahrlos unterbrochen oder einzelschrittweise abgearbeitet werden kann. Wäre Flag 55 gelöscht, würden die Flags 21 und 55 beide bei einer Unterbrechung gesetzt werden (vgl. Abschnitt 2C), was einer möglichen Änderung des Teiles von d gleichkäme, der während der 'Auslagerung' von c dem '.END.' entspricht.

Die bei 'LBL 03' beginnende Schleife führt im Flagregister eine binäre Addition, welche auf Tapanis einzigartig elegantem Algorithmus beruht, durch. Die in den Flags 00 bis 11 liegende Binärzahl wird erst in eine Dezimalzahl umgewandelt und dann dem dezimalen Inkrement (das ist die Anzahl der Register, um die der Vorhang versetzt werden soll) zugeschlagen. Danach wird die Summe in eine Binärzahl zurückgerechnet und schließlich wieder nach d in die Flags 00 bis 11 gebracht.

Die Eigenschaft, durch welche Tapanis Lösung des Problems der Addition einer binären auf eine dezimale Zahl sich gegenüber anderen Lösungen so auszeichnet, ist die scharfsinnige Umwandlung binär → dezimal → binär, bei der an jeder Bitstelle das Ergebnis bereits 'vollendet' wird, noch bevor das nächste Bit Eingang in den Umrechnungsprozeß findet. Bei jedem Durchgang durch die Schleife 'LBL 03' wird ein Bit des alten Zeigerwertes verarbeitet und zugleich in den für den neuen Zeigerwert gültigen Zustand versetzt.

Betrachten wir den Vorgang für das letzte betroffene Bit, dargestellt in Flag 11, welches im ersten Schleifendurchgang bearbeitet wird. Sobald das Programm das erste Mal in die Schleife läuft, enthält X das Inkrement, demgemäß der Vorhang versetzt werden soll. Die Zeilen 13 und 14 bewirken zweierlei: sie löschen Flag 11, also das 'Einser'-Bit des anfänglichen Zeigerwertes, und sie addieren genau dann 1 auf den Inhalt von X, wenn Flag 11 gesetzt war. Dies kommt einer dezimalen Addition von 1 auf das Inkrement in X gleich, wenn der anfängliche Zeigerwert ungerade war, was sich in gesetztem Flag 11 ausdrückt. In Zeile 21 wird Flag 11 genau dann wieder eingeschaltet, wenn die eben beschriebene Addition in X eine ungerade Zahl hinterlassen hat (man beachte, daß dieser erste Durchlauf bereits endgültig darüber entscheidet, ob der schließlich entstehende neue Zeigerwert gerade oder ungerade ist).

Die in den Zeilen 15 bis 24 stehenden Befehle haben insgesamt die Aufgabe, Flag 11 wieder zu setzen und 1 vom Inhalt von X abzuziehen, wenn die Addition in Zeile 14 eine ungerade Zahl ergeben hat, andernfalls Flag 11 im gelöschten Zustand zu belassen, und in jedem Fall den Inhalt von X durch 2 zu teilen. Wegen der Zeilen 19 und 24 ist sichergestellt, daß dabei stets

eine ganze Zahl in X zurückbleibt. Außerdem wird in Zeile 22 der Flag-Index, das ist die in M liegende Schleifenkontrollzahl, von 11 auf 10 heruntergesetzt. Flag 11 hat am Ende der Schleife den endgültigen Zustand für die binäre Darstellung des angestrebten Zeigerwertes gewonnen: gesetzt bei ungeradem, gelöscht bei geradem Zeigerwert. Die Zeilen 25 und 26 verzweigen zum Schleifenanfang, sofern in X noch nicht der Wert 0 erreicht ist.

Beim zweiten Schleifendurchlauf ist die zu verarbeitende Binärzahl in d nur noch 11 Bits lang (Flag 00 bis 10). Weil wir aber im Durchlauf zuvor den Inhalt von X durch 2 dividiert haben, liegt dort jetzt ein reduziertes Inkrement, welches der verkürzten Binärzahl (Flag 10 ist ihr 'Einser'-Bit) angepaßt, sozusagen mit ihr 'verträglich' ist. Man kann sich leicht klarmachen, was das heißen soll, wenn man beide ursprünglichen Zahlen, die 12 Bit lange Binärzahl und das Inkrement aus X, in Binärdarstellung untereinander schreibt und sich die letzte Stelle beider weggeschnitten denkt. Außerdem enthält das in X liegende reduzierte Inkrement u.U. noch einen 'Übertrag', der sich aus der vorangehenden Addition (Zeile 14) ergeben kann. Das ist z.B. der Fall, wenn Flag 11 gesetzt war und das ursprüngliche Inkrement ungerade ist.

Wie zuvor Flag 11 wird jetzt Flag 10 behandelt: gelöscht, nach X übertragen, wenn es gesetzt war, und wieder gesetzt genau dann, wenn das Additionsergebnis in X ungerade ist. Abermals entsteht in X eine ganze Zahl. Das Verfahren wird solange fortgesetzt, wie der Inhalt von X noch nicht auf 0 gesetzt ist, was schließlich geschehen muß, weil ständig durch 2 dividiert wird.

Beachten Sie, daß man an keiner Stelle des Programms wissen muß, ob der Inhalt von X positiv oder negativ ist. "CU" arbeitet in beiden Fällen auf dieselbe Weise: wenn ein gesetztes Flag gelöscht wird, wird der Inhalt von X um 1 inkrementiert (Übertrag), wenn es gesetzt werden muß, um 0.5 dekrementiert und bei jedem Schleifendurchlauf durch 2 dividiert bis schließlich 0 in X steht.

Die Zeilen 27 bis 29 bringen den Inhalt von Register d mit dem dort 'erarbeiteten' neuen Zeigerwert nach M, wobei gleichzeitig der in Register O aufbewahrte ursprüngliche Flagzustand nach d zurückgelangt. Bei dieser Umordnung geraten die ersten 4 Bytes aus dem ursprünglichen Inhalt von c, welche immer noch rechtsbündig in Register N warten, und die linken 3 Bytes aus dem 'Bit-Laboratorium' d, welche zusammen den neuen Zeiger für den Vorhang und den alten Zeiger für das '.END.' enthalten und jetzt linksbündig in M liegen, zu Nachbarn im Alpha-Register. Sie bilden gemeinsam die erstrebte Byte-Kombination, die nun nach c muß. Durch Anhängen dreier Bytes (Zeile 30) werden diese 7 Bytes zum Inhalt von Register N. In den Zeilen 31 und 32 werden die Inhalte von N und c gegeneinander ausgetauscht, wodurch das Ziel, den neuen Zeiger in c platzgerecht unterzubringen, erreicht wird. Weil "CU" dabei den Befehl 'X< >c' verwendet, bekommt man den alten Inhalt von c nach X, so daß man die ursprüngliche Lage des Vorhangs später mit einem einfachen 'STO c' wiederherstellen kann, vorausgesetzt natürlich, daß man diesen Wert unversehrt, insbesondere ohne Normalisierung, zur Hand behält, beispielsweise durch Aufbewahrung im Stapel.

Die letzten Zeilen 'räumen auf': das Alpha-Register wird gelöscht und der Stapel in Ordnung gebracht. Die Werte, welche vor dem Programmlauf in Y und Z lagen, gelangen nach X und Y; Z behält seinen Inhalt, und in T bleibt der durch 'X< >c' erlangte vormalige Inhalt von c zurück.

Machen Sie sich die Mühe, und folgen Sie den voranstehenden Erläuterungen mehrmals, bis Ihnen alles klar ist (denken Sie an G.A.Bürgers 'Schatzgräber'!). Laden Sie den Stapel mit '4, ENTER↑, 3, ENTER↑, 2, ENTER↑, 1', führen Sie 'SIZE 001' und 'GTO "CU"' aus, und gehen Sie die ganze Routine mit 'SST' durch. 'Unterwegs' können Sie beliebig oft in den Alpha-

Modus und zurück schalten, um zu beobachten, was im Alpha- und im X-Register vor sich geht, wenn der Vorhang um 1 Register gehoben wird.

Seien Sie nicht besorgt, wenn ein Teil oder sogar das meiste dieses Kapitels schwierig erscheint, um beim ersten Lesen ganz erfaßt zu werden. Schließlich ist dieser Stoff nicht zufällig bis zum Schluß aufgespart worden. Bedenken Sie, daß der Byte-Schnapper und das abenteuerliche Verfahren, ihn ohne vornehme Hilfsmittel einer Taste zuzuweisen, beide erst zwei Jahre nach den Anfängen der synthetischen Programmierung entdeckt worden sind. Ohne Zweifel gibt es noch viel mehr im HP-41 aufzuspüren. Vielleicht gelingt es gerade Ihnen, fündig zu werden.

Lösungen der Aufgaben

Zu Kapitel 2

2.1

```
01*LBL "CQ"
02 RAD
03 CLX
04 TONE 8
05 TONE ↑
06 TONE 8
07 TONE ↑
08 SIN
09 TONE 8
10 TONE 8
11 TONE ↑
12 TONE 8
13 END
```

LB / MK -Eingaben:

159, 120

159, 120

159, 120

Für die Zeitverzögerung zwischen den beiden Buchstaben C und Q kann man natürlich auch eine andere Lösung wählen.

2.2

```
01 ENTER↑
02 1 E1
```

eintasten, 'GTO. 001', 'RDN' eintasten, BS, ←, ←. Jetzt steht 'E1' in Zeile 02. Zurück auf Zeile 01, 'STO 28' eintasten, 'PACK', 'BST', BS, ←. Das 'PACK' setzt die Nachsilbe 28 unmittelbar vor den Befehl 'E1', indem es die durch das Eintasten von 'STO 28' entstandenen 'NULL'en alle beseitigt. Sobald die Vorsilbe 'STO' vom Byte-Schnapper erfaßt wird, gerät die Nachsilbe 28 zu einem 'NEG', das dem Zifferneintrag 'E1' zugeschlagen wird. Die LB-Eingaben für '– E1' lauten 28, 27, 17.

2.3

```
01♦LBL "Vx"                LB / MK -Eingaben:
02 "  "      (2 Leerstellen)
03 RCL d                   144, 126
04 SCI 9
05 ARCL Y    (nicht etwa X, weil 'RCL d' den Stapel anhebt)
06 STO d                   145, 126
07 RDN
08 AVIEW
09 END
```

In Fällen wie diesem sollten Sie sich angewöhnen, das 'AVIEW' hinter das 'STO d' zu setzen, nicht davor, um das Abändern der System-Flags zu verhüten. Hier nämlich würde bei vorangestelltem 'AVIEW' die Anzeige in den Ausgangsmodus zurückfallen (die mit 'AVIEW' angezeigte Zahl verschwände), weil das 'STO d' das Meldungsflag 50 löschte, es sei denn, vor 'RCL d' stünde ein anderer '(A)VIEW'-Befehl, der Flag 50 setzt.

2.4

```
01♦LBL "GS"                LB / MK -Eingaben:
02 FIX 9
03  E                      27/27, 0 (vgl. Abschnitt 4B)
04 RCL b                   144, 124
05 X<>Y
06 1/X
07  E                      27/27, 0
08 +
09 X<>Y
10 VIEW Y
11 STO b                   145, 124
12 END
```

Das Verfahren konvergiert in etwa 8 Sekunden gegen einen auf 10 Ziffern genauen Wert.

2.5 a)

```
01♦LBL "PX"                LB -Eingaben:
02 FIX 0
03 CF 29
04 "X("                    242, 88, 40
05 ARCL 00
06 "├)=?"                  244, 127, 41, 61, 63
07 PROMPT
```

Die synthetischen Zeilen können mit dem Byte-Schnapper so erzeugt werden:

```
01 ENTER↑
02 "XX"
03 "├X=?"
```

'GTO .002', BS, 'GTO .005', ←, 'RCL 09', 'GTO .002', BS, 'DEL 002', 'GTO .001', BS, 'GTO .004', ←, 'RCL 08', 'GTO .001', BS, 'DEL 002', ←. Anschließend werden die nicht-synthetischen Zeilen dazwischengetastet.

2.5 b)
Um den Anzeigemodus zu bewahren, müssen noch die Befehle 'RCL d' und 'STO d' passend eingefügt werden:

```
01♦LBL "PX"                LB /MK-Eingaben:
02 RCL d                   144, 126
03 CF 29
04 FIX 0
05 "X("
06 ARCL 00
07 "├)=?"
08 STO d                   145, 126
09 RDN
10 PROMPT
```

In diesem Programm läßt sich ein Byte einsparen, indem man die Zeilen 02 und 03 durch

```
02 ,          (Dezimalkomma)
03 X<> d                   206, 126
```

ersetzt. Damit wird 0 ins Flag-Register gebracht, was alle 56 Flags löscht. Dann genügt 'FIX 0', um den gewünschten Zustand der Flags 29 und 36 – 39 zu erlangen. Der alte Inhalt des Registers d liegt wie zuvor in X bereit, so daß 'STO d' den vorherigen Gesamtzustand der Flags wiederherstellen kann. 'X< >d' erlangt man mit dem Byte-Schnapper, indem man 'STO IND 78', gefolgt von 'AVIEW', eintastet und das 'STO'-Byte mit ihm ergreift. Die Nachsilbe 'IND 78' wird zu 'X< >' und 'AVIEW' bildet die Nachsilbe d.

2.6

```
01♦LBL "OX"          LB -Eingaben
02 RCL d             144, 126
03 FIX 2
04 "OUT="
05 ARCL Y
06 STO d             145, 126
07 RDN
08 "⊢ρV"             243, 127, 12, 86
```

Zeile 08 läßt sich mit dem Byte-Schnapper so erzeugen:

```
01 ENTER↑
02 "⊢XV"
```

'GTO .001', BS, 'GTO .004', ←, 'LBL 11', 'GTO .001', BS, 'DEL 002', ←.

2.7

```
01♦LBL "CMOD"        LB /MK -Eingaben
02 X<>Y
03 STO [             145, 117
04 X<>Y
05 MOD
06 ST- [             147, 117
07 LASTX
08 ST/ [             149, 117
09 CLX
10 X<> [             206, 117
11 END
```

Die Zeilen 02 – 05 retten y nach M und x nach L. y mod x wird dann von y (in M) abgezogen. Die Zeilen 07 – 10 dividieren den Inhalt von M durch x, bringen dieses Ergebnis nach X und löschen M.

2.8
Erzeugen Sie erst das 'F0'-Byte:

```
01 ENTER↑
02 STO IND T
```

'BST', BS, ←. Dann 'STO IND Z' eintasten, 'PACK', 'BST', BS, ←, ←. Damit wird die Nachsilbe 'IND Z' zu einem 'TEXT 1'-Byte, welches sich das 'TEXT 0'-Byte als Text 'unterwirft'.

Zu Kapitel 3

3.1
'GTO ..', 'LBL "++"', mindestens 45 '+'-Befehle (vgl. Tabelle 3.1 auf S.44) und 'XEQ "LB"' eintasten. Übergang in den RUN-Modus, 'R/S', die Eingabe-Aufforderungen wie folgt beantworten:

Eingabe-Aufforderung	Antwort	Eingabe-Aufforderung	Antwort
1?	27 R/S	4?	245 R/S
2?	145 R/S	5?	159 R/S
3?	119 R/S	6?	106 R/S
4?	146 R/S	7?	244 R/S
5?	119 R/S	1?	1 R/S
6?	206 R/S	2?	4 R/S
7?	119 R/S	3?	5 R/S
1?	145 R/S	4?	6 R/S
2?	117 R/S	5?	242 R/S
3?	150 R/S	6?	127 R/S
4?	117 R/S	7?	96 R/S
5?	240 R/S	1?	154 R/S
6?	153 R/S	2?	118 R/S
7?	245 R/S	3?	152 R/S
1?	152 R/S	4?	118 R/S
2?	119 R/S	5?	R/S
3?	172 R/S		

Wenn das Programm anhält, können Sie mit 'SST' auf 'LBL "++"' springen und sich Ihre neuen synthetischen Befehle ansehen.

3.2
Hier ein einfaches nicht-synthetisches Programm, welches die "LB"-Eingaben für XROM-Zahlen berechnet. Es nutzt die Tatsache aus, daß 64·(i mod 4) den gleichen Wert ergibt wie 256·FRC(i/4). Im rechten Teil der nachstehenden Auflistung ist festgehalten, wie sich die Stapelinhalte während des Programmlaufs verändern. Fehlende Einträge zeigen an, daß keine Veränderung stattgefunden hat.

01*LBL "XRLB"	L	X	Y	Z	T
02 X<>Y		i	j	z	t
03 4		4	i	j	z
04 /	4	i/4	j	z	z
05 INT	i/4	INT(i/4)			
06 X<>Y		j	INT(i/4)		
07 LASTX		i/4	j	INT(i/4)	z
08 FRC	i/4	FRC(i/4)			
09 256		256	FRC(i/4)	j	INT(i/4)
10 *	256	64(i mod 4)	j	INT(i/4)	INT(i/4)
11 +	64(i mod 4)	Byte 2	INT(i/4)		
12 X<>Y		INT(i/4)	Byte 2		
13 160		160	INT(i/4)	Byte 2	
14 +	160	Byte 1	Byte 2	INT(i/4)	INT(i/4)
15 END					

'i, ENTER↑, j, XEQ "XRLB"' liefert zu den Eingaben i und j die Ergebnisse in X und Y ab, und zwar Byte 1 dezimal in X und Byte 2 dezimal in Y.

Nachstehend eine synthetische Fassung "XRLBS" von "XRLB", die die Inhalte von Z und T unversehrt läßt. Rechterhand wieder die Stapel- und diesmal auch die Zustandsregisterinhalte, soweit sie das Programm betreffen und sich während seines Laufs verändern.

01*LBL "XRLBS"	N	M	L	X	Y	Z	T
02 STO [		j		j	i	z	t
03 RDN				i	z	t	j
04 4				4	i	z	t
05 /			4	i/4	z	t	t
06 STO \	i/4						
07 FRC			i/4	FRC(i/4)			
08 256				256	FRC(i/4)	z	t
09 *			256	64(i mod 4)	z	t	t
10 RCL [				j	64(i mod 4)	z	t
11 +			j	Byte 2	z	t	t
12 160				160	Byte 2	z	t
13 ST+ \	160 + i/4						
14 X<> \	160			160 + i/4			
15 INT			160 + i/4	Byte 1	Byte 2	z	t
16 CLA	0	0					
17 END				Byte 1	Byte 2	z	t

3.3

Die 7 Eingaben für "LB" lauten 207, 120, 159, 37, 208, 0, 120. Sie müssen dafür mindestens 17 '+'-Zeilen eintasten (vgl. Tabelle 3.1 auf S.44).

3.4
Die 18 Eingaben für "LB" lauten 192, 0, 255, 0, 82, 80, 78, 32, 67, 65, 76, 67, 85, 76, 65, 84, 79, 82. Sie müssen dafür mindestens 31 '+'-Zeilen eintasten (vgl. Tabelle 3.1 auf S.44). Die neue Marke wird durch 'PACK' dem Katalog 1 einverleibt. Da die Marke mehr als 7 Zeichen lang ist, kann sie weder mit direkten noch indirekten Befehlen erreicht werden; eine reine l'art pour l'art-Marke also.

3.5
Die für "LB" erforderlichen Dezimalwerte lauten 144, 124, 206, 117, 145, 118, 145, 117, 206, 117, 206, 125, 145, 125, 242, 127, 0, 206, 125, 144, 117, 145, 125.

Zu Kapitel 4

4.2
Die benötigten Dezimalwerte lauten: 244, 127, 0, 0, 2, 27, 20, 206, 125, 145, 125, 242, 127, 0, 206, 125, 145, 125. Sie beginnen mit 'GTO ..', tasten 'LBL "LB"' ein und führen im RUN-Modus dies aus: 'CLA, 125, XTOA, 145, XTOA, 125, XTOA, 206, XTOA, 0, XTOA, 127, XTOA, 242, XTOA'. 'GTO "LB"', 'RCL M', 'STO Q', Übergang in den PRGM-Modus, Q-Bote, BS, ←, ←. Zurück in den RUN-Modus. Dann: 'CLA, 125, XTOA, 145, XTOA, 125, XTOA, 206, XTOA, 20, XTOA, 27, XTOA'. 'GTO "LB"', 'RCL M', 'STO Q', Übergang in den PRGM-Modus (ein 'PACK' ist hier nicht nötig, weil das Byte 242_{10} nicht Teil eines vorangehenden Befehls ist und somit keine Anbindung an die neuen Bytes erforderlich wird), Q-Bote, BS, ←, ←. Abermals zurück in den RUN-Modus und fortfahren mit: 'CLA, 2, XTOA, 0, XTOA, 0, XTOA, 127, XTOA, 244, XTOA'. 'GTO "LB"', 'RCL M', 'STO Q', PRGM-Modus, Q-Bote, BS, ←, ←. Die Tatsache, daß wir das Byte 2_{10} nicht zusammen mit der zweiten Gruppe von Bytes geladen haben, befreit uns von der Notwendigkeit eines 'PACK' vor dem Laden der dritten Gruppe. Überdies ist das Abschließen mit einem nicht-'NULL'-Byte sowieso erforderlich, um der Unfähigkeit des Q-Boten bezüglich der Übertragung nachgeschleppter (im Register Q führender) 'NULL'en (vgl. Abschnitt 4B) zuvorzukommen. Wir hätten die Folge F4 7F 00 00 als eigenständige Gruppe nicht mit ihm in den Programmspeicher bringen können.

4.3

```
a) XROM 61,25
b) XROM 57,56
c) XROM 27,54
```

4.6
Mit den Dezimalwerten 0 und 160 erlangen Sie eine Zuweisung, die ganz ähnlich wie 'eG∅BEEP' arbeitet. Sie erscheint in der Anzeige in der Gestalt '↑' und fordert Sie mit den Unterstrichen zur Eingabe zweier Ziffern auf. Für 00 bis 99 erhalten Sie 'XROM 00,00'

bis 'XROM 01,35'. Dabei entsprechen die Einträge 65 bis 99 den Zahlen 'XROM 01,01' bis 'XROM 01,35'. Dies sind die für die 35 globalen Marken "MATRIX" bis "TANH" des Mathematik-Moduls (die letzten 10 Marken werden nicht erfaßt) vorgewiesenen 'XROM'-Zahlen bei Abwesenheit des Moduls. Im RUN-Modus haben Sie wie bei 'eG∅BEEP' schnellen Zugriff auf die Marken durch einfache Zahleneinträge. Im PRGM-Modus können Sie, ähnlich wie bei 'eG∅BEEP', diese Marken als Unterprogrammaufrufe Ihren Programmen einfügen, und zwar, wieder wie bei 'eG∅BEEP', auch in Abwesenheit des Moduls.

Im Gegensatz zu 'eG∅BEEP' nimmt '↑' sogar pseudo-indirekte Eingaben entgegen und hat damit einen doppelt so großen Wirkungsbereich: drücken Sie vor der Zifferneingabe 'SHIFT', erhalten Sie zu '↑ IND 00' bis '↑ IND 99' die Zahlen 'XROM 02,00' bis 'XROM 03,35'. Darunter befinden sich für '↑ IND 01' bis '↑ IND 29' die zu den globalen Marken des Statistik-Moduls gehörigen Zahlen 'XROM 02,01' bis 'XROM 02,29' und für '↑ IND 65' bis '↑ IND 96' die zu den globalen Marken des Moduls für Vermessung gehörigen Zahlen 'XROM 03,01' bis 'XROM 03,32'. Für RUN- und PRGM-Modus gilt das oben Gesagte.

Mit den Dezimalwerten 0 und 161 erlangt man eine synthetische Tastenzuweisung, die in der Anzeige als 'μ' erscheint. Sie verschafft mit 'μ 00' bis 'μ 99' bzw. 'μ IND 00' bis 'μ IND 99' Zugang zu

```
'XROM 04,00 - 04,63/05,00 - 05,35/06,00 - 06,63/07,00 - 07,35'
```

und verhält sich in allen Punkten wie '↑'. Die Module für Finanzen, Netzwerkanalyse und Baustatik können mit 'μ' so bequem genutzt werden wie die Module für Mathematik, Statistik und Vermessung mit '↑'.

Zu Kapitel 5

5.1

Die Byte-Folgen lauten hexadezimal: a) 40, 47, 48, 00, 00, 00, 00, 00, 00, 13, 41, 00, 14, 25, 15, 42. Für 'Σ+' (47_{16}) war noch Platz, doch das Einfügen von 'Σ−' (48_{16}) eröffnete ein neues Register. Das 'RCL 05' fand Unterkunft in der 'NULL' zwischen den Zahleneinträgen '4' und '5'.

b) 40, 41, E0, 00, 00, 92, 4B, 00, 42, 43. 'ST+ 75' nimmt nur zwei der drei Bytes ein, die vorher von 'GTO 99' besetzt wurden.

ANHANG A

Das Messen der Ausführungsdauer von Befehlen

Beim Lesen von Kapitel 2 haben Sie sich vielleicht gefragt, wie man überhaupt feststellen kann, daß der Eintrag eines synthetisch erzeugten 'E' schneller ausgeführt wird als der Eintrag einer gewöhnlichen '1' oder das Dezimalkomma ',' schneller als '0' ist. In den Tagen des HP-67 konnte man solche Messungen nur durchführen, indem man 100 mal oder öfter den gleichen Befehl eintastete, dann die Zeit zur Ausführung der gesamten Befehlsfolge stoppte und das Ergebnis durch die Anzahl der Befehle in der Folge teilte. Überflüssig zu sagen, daß dieses Verfahren mühselig, zeitraubend und obendrein ungenau war.

Die synthetische Programmierung erlaubt einen vollständig programmgesteuerten Ablauf des Meßverfahrens, wobei hunderte von Kopien eines bestimmten Befehls (sogar Kopien einer kurzen Folge von Befehlen) ohne Tastenleistung des Benutzers entstehen. Die zur Messung anstehende Byte-Folge wird in Blöcken zu je 7 Bytes erzeugt und in benachbarten Registern abgelegt. Die Bytes können dann als Programmbefehle abgearbeitet werden, indem der Adreßzeiger geeignet umgesetzt wird.

Es ist ein Beweis für die Leistungsfähigkeit des HP-41 Systems, daß der Time-Modul 82182A sogar die automatische Messung einer Folge von synthetisch gespeicherten Befehlen gestattet. Clifford Stern hat ein synthetisches Programm geschrieben, das X-Funktionen, den Time-Modul und den PPC ROM benutzt, um die Ausführungsdauer für eine beliebige Gruppe von 1 bis 7 Bytes zu messen. Das Programm erzeugt soviele Kopien der Byte-Gruppe, als innerhalb des unbenutzten Arbeitsspeichers abgelegt werden können. Es läßt alsdann die gesamte Folge der Byte-Gruppen abarbeiten, mißt die dafür verbrauchte Zeit, dividiert sie durch die Anzahl der Gruppen und zeigt als Ergebnis die Ausführungsdauer für eine Gruppe an.

Tabelle A.1 zeigt die Ergebnisse der Messungen für eine Reihe von Befehlen. Dabei wurde Wert darauf gelegt, solche Befehle aufzunehmen, für die Alternativen verfügbar sind. Wenn man 'LOG' verwenden will, ist es unwesentlich zu wissen, wieviel Zeit die Ausführung dieser Funktion kostet, da man ohnehin keine Ausweichmöglichkeit hat, einen Logarithmus zu berechnen. Hingegen ist es durchaus wissenswert, daß das Inkrementieren des Registers X mit 'ISG X' und 'TEXT 0' (als 'NOP') etwas schneller ist (74 msec) als die Folge 'E, +' (78,7 msec). Hat die Geschwindigkeit Vorrang für Sie, werden Sie bereit sein, ein paar zusätzliche Bytes im Programmspeicher dafür zu opfern. Nur müssen Sie eben die Ausführungsdauer verschiedener Alternativen kennen, bevor Sie eine vernünftige Entscheidung zu treffen in der Lage sind.

Hier die wichtigsten Ergebnisse aus der Meßtabelle:

- 'R↑, R↑' ist schneller als 'RDN, RDN';
- 'X< >' ist schneller als 'RCL', doch langsamer als 'STO';

- Zugriffe auf Zustandsregister sind stets schneller als die entsprechenden Zugriffe auf numerische Datenregister;
- kompilierte 'GTO's sind sehr schnell, 'XEQ's ein wenig langsamer;
- Zahleneinträge sind grundsätzlich sehr langsam, was daher rührt, daß erst P und Q geladen werden müssen, bevor der Eintrag das Register X erreicht (vgl. Abschnitt 6A);
- ',' ist schneller als '0', 'E' ist schneller als '1' und 'CLX, SIGN' ist noch wesentlich schneller als 'E';
- negative Zahlen werden schneller eingetragen, wenn man das negative Vorzeichen nicht der Zeile, die die Zahl enthält, zuschlägt (auch wenn dies übersichtlicher ist), sondern stattdessen eine positive Zahl einträgt und ein getrenntes 'CHS' als gesonderte Zeile folgen läßt (man beendet den Zahleneintrag mit 'ALPHA, ALPHA', so daß das folgende 'CHS' in eine eigene Zeile gelangt), denn 'CHS' ist wesentlich schneller als das 'NEG', welches das negative Vorzeichen *innerhalb* eines Zahleneintrages bildet.

Die voranstehend aufgeführten Erkenntnisse sind ein besonderes zunächst nicht ohne weiteres zu erwartendes Beispiel dafür, wie die synthetische Programmierung zur Verbesserung Ihrer allgemeinen Programmiertechniken beitragen kann.

Tabelle A.1. Ausführungszeiten in Millisekunden (gemessen ohne Drucker)

Stapel-Operationen		
ENTER↑		11.7
X<>Y		10.3
RDN		16.9
R↑		12.0
CLX		9.8
LASTX		13.0
CLST		10.5
SIGN		13.3
CHS		12.5
CLA		9.5
RCL	Zustandsregister	20.3
STO		16.8
X<>		19.7
verschiedene Befehle		
LBL 00-14		10.6
Zwei-Byte-LBL		13.1
CLD		20.6
TEXT 0		12.3
AON, AOFF		19.0
ADV (ohne Drucker)		9.2
BEEP (Flag 26 gesetzt)		1042.4
(Flag 26 gelöscht)		14.9
DEG		19.8
RAD		19.9
GRAD		20.5

PSE	1333.2	
NULL	5.7	

Zugriffe auf Register

STO 00-15	19.3	
STO 16-99	20.6	
STO Zustandsregister	16.8	
STO IND 00-99	32.3	
STO IND Zustandsregister	32.1	
RCL 00-15	22.8	
RCL 16-99	24.1	
RCL Zustandsregister	20.3	
RCL IND 00-99	35.7	
RCL IND Zustandsregister	35.6	
X<> 00-99	23.4	
X<> Zustandsregister	19.7	
X<> IND 00-99	35.1	
X<> IND Zustandsregister	35.0	
ST+ 00-99	38.9	
ST+ Zustandsregister	35.3	
ST- 00-99	40.8	
ST- Zustandsregister	37.3	
ST* 00-99	46.8	
ST* Zustandsregister	43.0	
ST/ 00-99	49.5	
ST/ Zustandsregister	45.8	
ISG X, TEXT 0 (Sprung)	73.2	(x = 1)
(kein Sprung)	74.4	(x = -1)
DSE X, TEXT 0 (Sprung)	72.9	(x = 1)
(kein Sprung)	74.0	(x = 2)

Zahlen-Einträge

0	69.7
1 bis 9	59.8
.	61.8
E	53.6
- (NEG; setzt negatives Vorzeichen vor Mantisse oder Exponenten; setzt alleinstehend 0 ins X-Register)	60.9

verschiedene Mehr-Byte-Befehle

GTO 00-14 kompiliert	17.3
Drei-Byte-GTO kompiliert	24.5
XEQ kompiliert	35.2
globales LBL 1 Zeichen	45.4
2 Zeichen	49.3
3 Zeichen	51.9

Wenn Sie den PPC ROM (beachten Sie Anhang F), den X-Funktionen-Modul und den Time-Modul besitzen, können Sie Clifford Stern's Programm benutzen, um Befehlsmessungen auf eigene Faust und für eigene Zwecke durchzuführen. Hier die Gebrauchsanweisung:

1) Stellen Sie zunächst sicher, daß sich oberhalb des Meßprogramms ein 'END' im Katalog 1 befindet. Das ist notwendig, damit die 'GTO'-Befehle fehlerlos arbeiten, wenn sich der Vorhang auf hexadezimal 010 befindet (schlagen Sie zur Erklärung dieser Einschränkung in Abschnitt 6C nach).

2) Löschen Sie Flag 2, und setzen Sie eine Datenregisteranzahl von mindestens 4 fest. Löschen Sie alle Weckaufträge, u.U. mit dem in Abschnitt 4E beschriebenen Programm "SA". Wenn Sie wollen, können Sie jetzt noch Tastenzuweisungen tätigen, doch tun Sie dies (außer für globale Marken) *nicht*, sobald Sie mit Schritt 3 begonnen und noch nicht mit Schritt 9 geendigt haben.

3) Tasten Sie die Anzahl von Registern ein, die Sie für die Ablage der zu messenden Byte-Gruppe verwenden wollen. Diese Anzahl sollte so gewählt werden, daß ein Vielfaches der Anzahl der Bytes der Gruppe die Anzahl der Meßregister genau ausfüllt (die Gesamtanzahl benutzter Bytes muß ein gemeinsames Vielfaches von 7 und der Anzahl der Bytes der Gruppe sein). Nur bei Gruppen von genau 1 oder 7 Bytes braucht nicht mitgedacht zu werden. Beispiel: Für eine Gruppe von 3 Bytes muß eine Anzahl von Registern gewählt werden, die ein Vielfaches von 3 ist. Wenn die Anzahl der Meßregister kein Vielfaches der Anzahl der Bytes je Gruppe ist, können Sie u.U. auf Zeile 114 die Meldung "DATA ERROR" bekommen. Sofern Sie sich für ein Vielfaches von 60 entscheiden, kann gewiß nichts mißlingen, denn 60 ist durch 2, 3, 4, 5 und 6 teilbar. Mit 'XEQ "IN"' stellen Sie das Meßprogramm auf die gewählte Registeranzahl ein. Sollten für die angeforderte Anzahl von Meßregistern nicht mehr hinreichend viele freie Register unterhalb von '.END.' zur Verfügung stehen, setzt das Programm die Datenregisteranzahl mit 'PSIZE' selbständig herunter, es sei denn, daß auch diese Anpassung fehlschlagen muß, weil in jedem Fall zu wenig Register für Meßzwecke frei sind. Sie erhalten dann auf Zeile 49 die Meldung "DATA ERROR". Wenn dies geschieht, löscht man Programme oder setzt die Anzahl der angeforderten Meßregister herunter und beginnt wieder mit 'XEQ "IN"'.

4) 'XEQ "IN"' führt automatisch in den mit "S" beginnenden Programmteil, welcher die Routine für das Kopieren der Befehlsfolge und das Ablegen der Kopien in den Meßregistern bildet. "S" fordert Sie auf, Dezimalzahlen für die zu messenden Befehle einzugeben. Tasten Sie nacheinander bis zu 7 Zahlen zwischen 0 und 255 ein (z.B. 118 für 'LASTX'), und beenden Sie jeden Eintrag mit 'R/S'. Drücken Sie 'R/S' ohne Eintrag, wenn die Eingabe der Byte-Gruppe abgeschlossen ist. Die Gruppe wird dann vervielfacht und Kopie für Kopie in die festgelegten Meßregister, die sich zwischen dem '.END.' und den Tastenzuweisungsregistern befinden, gebracht.

5) Bei gelöschtem Flag 1 hält die Routine "S" auf der Marke "T", die die eigentliche Meßroutine anführt, an. Zu diesem Zeitpunkt ist der Stapel leer. Sie können ihn jetzt Ihren Absichten entsprechend laden. Anschließend starten Sie das Meßverfahren mit 'R/S' oder 'XEQ "T"'. Sobald die Routine anhält, liegt das Ergebnis, ausgedrückt in Millisekunden je

Befehl(sgruppe), im X-Register. Wenn es während der Ausführung der zu messenden Befehle eine fehlerbedingte Unterbrechung (angezeigt durch z.B. "OUT OF RANGE") gibt, müssen Sie 'GTO "S"' und 'XEQ 10' ausführen. Anschließend können Sie wieder mit Schritt 4 beginnen oder auch einfach den Stapel richtig füllen und mit 'XEQ "T"' fortfahren.

6) Wenn Sie den Meßvorgang mit denselben Befehlen doch für andere Zahlenwerte wiederholen wollen, müssen Sie den Stapel neu laden und abermals 'XEQ "T"' ausführen (*nicht* einfach 'R/S' drücken – beachten Sie Schritt 9). Falls Sie sich für Ihre Messungen sowohl der Inhalte des Stapels als auch der des Alpha-Registers bedienen wollen, brauchen Sie nur die Flags 1 und 2 zu setzen, bevor Sie "T" aufrufen. Die Meßroutine hält dann am Anfang an, damit Sie das Alpha-Register (und auch den Stapel, falls Sie das wünschen), füllen können. Es ist überdies möglich, "T" als Unterprogramm aufzurufen, so daß man die Ausführungszeit einer Funktion programmgesteuert für verschiedene Stapelinhalte messen kann.

7) Will man dazu übergehen, eine andere Befehlsgruppe zu messen, beginnt man wieder mit 'XEQ "S"'. Wenn Sie zuvor Flag 1 setzen, geht die Routine "S" nahtlos in "T" über, wobei mit leerem Stapel gearbeitet wird. Wenn Sie Flag 1 und Flag 2 setzen, hält die Routine "T" an, so daß Sie das Alpha-Register laden können.

8) Will man mit einer anderen Anzahl von Meßregistern arbeiten, gibt man diese Anzahl ein und führt 'XEQ "IN"' aus.

9) Soll die Arbeit mit dem Meßprogramm ganz beendet werden, drückt man 'RTN, R/S', oder nur 'R/S', wenn gerade ein Lauf von "T" beendet worden ist. Dabei werden die hinter dem '.END.' liegenden Meßregister sämtlichst wieder gelöscht.

10) Zur bequemen Nutzung ist das Programm mit drei zusätzlichen Einstiegsmöglichkeiten versehen. Es handelt sich dabei um Spielarten, bei denen man ohne unterbrechende Eingabe-Aufforderung durch "S" Befehle kopieren und speichern kann:
    a) 'XEQ "1"' (nicht etwa 'XEQ 01'!) verarbeitet *eine* Dezimalwertangabe (zwischen 0 und 255), um Ein-Byte-Befehle zu messen.
    b) 'XEQ "2"' verarbeitet eine Dezimalwerteingabe, um eine Befehlsfolge, die aus dem zum Eingabewert gehörigen Ein-Byte-Befehl und dem Befehl 'LASTX' besteht, herzustellen und zu kopieren. Diese Möglichkeit ist sehr nützlich, wenn man einargumentige Funktionen wie 'SIN' oder 'LN' messen will.
    c) 'XEQ "3"' verarbeitet eine Dezimalwerteingabe, um eine Befehlsfolge aus einer Ein-Byte-Funktion und dem Befehl 'X<>L' herzustellen. Dies dient dazu, zweiargumentige Funktionen wie '+' oder 'MOD' zu messen. Man lädt den ganzen Stapel mit dem y-Argument, legt dann das x-Argument nach X und führt 'XEQ "T"' aus.

Zur Erzielung gültiger Ergebnisse aus "2" und "3" muß man selbstverständlich noch die Befehle 'LASTX' bzw. 'X<>L' gesondert messen und von den zuerst erhaltenen Werten abziehen. Dann erst erlangt man die richtigen Ausführungszeiten für die jeweiligen ein- und zweiargumentigen Funktionen.

Will man die Ausführungszeit numerischer Einträge messen, muß man sie natürlich voneinander trennen, etwa durch 'NULL' oder 'LASTX' (vgl. Kapitel 5), damit sie nicht zu einem einzigen riesigen Eintrag zusammenfließen. Die Ausführungszeit für den Trennbefehl muß wie bei "2" und "3" vom 'Bruttoergebnis' abgezogen werden.

Programm zum Messen der Ausführungsgeschwindigkeit von HP-41-Befehlen (Barcode im Anhang E, PPC ROM-Routinen im Anhang F):

```
01 XROM "RF"
02 AVIEW
03 XROM "LF"
04 XROM "OM"
05 X<>Y
06 ISG X
07 XROM "BC"
08 GTO 13
09*LBL "3"
10 "t"
11 3
12 GTO 01
13*LBL "2"
14 "v"
15 2
16 GTO 01
17*LBL "1"
18 CLA
19  E
20*LBL 01
21 STO a
22 ASTO X
23 CLA
24 AVIEW
25 CF 29
26 FIX 0
27 X<>Y
28 XTOA
29 ARCL Y
30 GTO 16
31*LBL "IN"
32 STO 03
33 XROM "F?"
34 INT
35 ENTER↑
36 XROM "E?"
37 X<>Y
38 -
39 STO 01
40 SIZE?
41 ENTER↑
42 R↑
43 +
44 RCL 03
45 -
46 7
47 -
48 X<0?
49 SQRT
50 4
51 +
52 X<Y?
53 PSIZE
54 XROM "OM"
55 R↑
56  E
57 +
58 XROM "CX"
59 X<> c
60 RCL 03
61  E
62 +
63 X<>Y
64 X<> c
65 "                    ▪▪"
66 RCL [
67 STO 00
68 "                    x̄
"
69 ASTO IND Z
70 RDN
71 X<> c
72 X<> 01
73 +
74 2561
75 +
76 7
77 *
78 XROM "DP"
79 ASTO 02
80 BEEP
81*LBL 10
82 STOPSW
83 CLX
84 SETSW
85*LBL "S"
86 CF 29
87 FIX 0
88 CLA
89 CLX
90*LBL 11
91 XTOA
92 ISG a
93 -
94 X<> [
95 "DEC. "
96 ARCL a
97 "├?"
98 AVIEW
99 STO [
100 STOP
101 FS?C 22
102 GTO 11
103 CLA
104 AVIEW
105 STO [
106 DSE a
107*LBL 16
108 RCL 03
109 7
110 *
111 RCL a
112 /
113 STO 00
114 OCT
115 GTO IND a
116*LBL 07
117 X<> [
118 X<> ]
119 STO a
120 GTO 12
121*LBL 04
122 FIX 1
123*LBL 05
124 SF 29
125*LBL 06
126*LBL 03
127*LBL 02
128*LBL 01
129 ASTO X
130 17
131 RCL a
132 /
133 INT
134 RCL b
135 ARCL Z
136 DSE Y
137 STO b
138 "├*"
139 FC? 29
140 "├**"
141 RCL a
142  E5
143 /
144*LBL 12
145 RCL 03
146 +
147 ABS
148 RCL 01
149 X<> c
150 RCL ]
151 GTO 09
152*LBL 00
153  E
154 ST- L
155 ARCL X
156 LASTX
157 R↑
158 RCL ]
159*LBL 09
160 STO IND Z
161 DSE Z
162 GTO 09
163 DSE a
164 GTO 00
165*LBL 13
166 CLD
167 X<>Y
168 STO c
169 CLST
170 FC? 02
171 FC? 01
172 TONE 8
173 FC? 01
174 RTN
175*LBL "T"
176 ARCL 02
177 XROM "XE"
178 SETSW
179 X<>Y
180 36 E5
181 *
182 RCL 00
183 /
184 FIX 9
185 TONE 8
186 END
```

329 BYTES

Einige synthetischen Zeilen haben einen Ausdruck, dem man den Inhalt nicht ansehen kann. Sie werden nachstehend aufgelistet:

Zeile	hexadezimal	dezimal ("LB"-Eingaben)
10	F2 CE 74	242 206 116
14	F1 76	241 118
65	F7 A6 99 A6 93 6D 1C 85	247 166 153 166 147 109 28 133
68	F5 AC 02 84 A6 94	245 172 2 132 166 148

Die Zeilen 65 und 68 enthalten Kontrollbytes für den Drucker (vgl. Abschnitt 2E). A6 veranlaßt den Drucker, 6 Stellen zu überspringen, AC veranlaßt ihn, 12 Leerstellen in die Auflistung zu bringen.

Liste der möglichen Fehlermeldungen des Meßprogramms:

Zeile 49 "DATA ERROR": Vorhandener Speicherplatz zu gering, um die angeforderten Meßregister zur Verfügung zu stellen.
Zeile 114 "DATA ERROR": Anzahl der Bytes je Gruppe teilt nicht die Anzahl der angeforderten Meßregister.
Zeile 115 "NONEXISTENT": Es wurde versucht, eine Gruppe von 8 Bytes zu messen, obwohl das Programm nur Gruppen, die aus höchstens 7 Bytes bestehen, behandelt.

Datenregister-Benutzung des Meßprogramms:
R_{00}: Notizen (Anzahl der Befehlsgruppen)
R_{01}: Zeiger für den nach unten versetzten Vorhang (vorübergehend in Register c untergebracht)
R_{02}: Ansprungadresse für die Meßregister
R_{03}: Anzahl der Meßregister.

Falls der Inhalt eines der Register R_{01} bis R_{03} zerstört wird, müssen Sie von vorn anfangen: Meßregister-Anzahl eingeben, 'XEQ "IN"'.

ANHANG B

Morsezeichen und 'STO b'

Den Gedanken, mit dem HP-41 makellose Morsezeichen zu erzeugen, hatte zuerst Richard Nelson, der Gründer des PPC und Herausgeber des PPC CJ. Er veröffentlichte ihn auf S.50 der Ausgabe vom Februar '80 des PPC CJ. Sein Programm benutzte bereits den synthetischen 'TONE P', doch steckte die synthetische Programmierung noch in den Kinderschuhen, so daß die Programmlogik auf herkömmliche Techniken beschränkt blieb. Deswegen wurde auch nur eine Übertragungsgeschwindigkeit von etwa 28 Zeichen je Minute erreicht. Für das Erlangen einer Amateurfunklizenz werden jedoch 60 Zeichen je Minute verlangt. Übliche Methoden sind offenkundig ganz ungeeignet dazu, Zeichen mit dieser Geschwindigkeit zu erzeugen.

Dann schrieb Clifford Stern ein Morsezeichen-Programm, das die ganze Kraft synthetischer Programmierung zur Geltung brachte. Um die verwendete Technik zu verstehen, betrachten Sie die folgende Schleife, die in einer früheren Fassung dieses Programms auftauchte:

```
LBL 01
RCL IND L
XEQ IND X
ISG L
GTO 01
```

Die einzelnen Zeichen einer zusammenhängenden Funkmeldung liegen in einer Folge von Datenregistern, und Register L enthält einen Zähler für diese Datenregister. Der Befehl 'RCL IND L' legt ein einzelnes Zeichen der Meldung ins X-Register. 'XEQ IND X' ruft eine kurze dem in X liegenden Zeichen entsprechende Routine von Tönen auf. Enthielte X beispielsweise den Buchstaben "C", würde die nachstehende Befehlsfolge ausgeführt:

```
LBL "C"
TONE 8
TONE ↑
TONE 8
TONE ↑
RTN
```

Die Einfachheit dieses Verfahrens beruht auf der Benutzung synthetisch erzeugter globalen Marken der Länge 1. Sie werden für die – andernfalls unzugänglichen – Satzzeichen (, . :) und die Buchstaben A bis J hergestellt; die nicht-synthetischen Marken A bis J sind lokal und daher dem Zugriff durch indirekte Adressierung entzogen.

Dennoch bleibt die Geschwindigkeitkeit nach wie vor ein Problem. Denn 'XEQ IND X' muß den Katalog 1 nach der globalen Marke durchsuchen, um die richtige Tonfolge aufzuspüren, und das dauert immer noch verhältnismäßig lange. 16 Millisekunden nämlich werden benötigt, um in der globalen Kette von Marke zu Marke, vom '.END.' an aufwärts, zu gelangen. Dies bedeutet eine beträchtliche Verzögerung für Marken, die weit oben im Katalog angesiedelt sind.

Der entscheidende Durchbruch in diesem Morsezeichen-Programm gelang, als man auf den Gedanken kam, 'XEQ IND X' durch ein 'STO b' zu ersetzen, um verzögerungsfrei auf die jeweilige Tonfolge zu springen. Dies bringt nicht nur eine dramatische Beschleunigung in den Programmablauf, es ist auch ein überzeugendes Beispiel dafür, wie die synthetische Programmierung Dinge ermöglicht, die auf normale Weise nie zuwege gebracht würden, welche Klimmzüge man auch immer vollführte. Synthetische Methoden aber erlauben es, Adressen zum indirekten Ansprung zu sammeln und dem Befehl 'STO b' zur Verfügung zu stellen.

Wir wollen uns einige Einzelheiten des angedeuteten Verfahrens ansehen. Zunächst einmal ist es nötig, die genaue Ansprungadresse zu bestimmen. Dies wird mit einem 'RCL b', welches jeder Tonfolge voransteht, durchgeführt. Beispiel:

```
LBL "C"
RCL b
TONE 8          ← dies ist die Stelle, an der später
TONE ↑                die durch 'STO b' ausgelöste
TONE 8                Ausführung der Routine "C"
TONE ↑                aufgenommen wird
RTN
```

Diese Folgen werden in der 'Aufbauphase', während der die Adressen gesammelt werden, bei gelöschtem Flag 26 aufgerufen. Die Ergebnisse der Befehle 'RCL b' werden in aufeinanderfolgenden Datenregistern abgelegt. Dabei werden die vom 'XEQ' in Zeile 58 herrührenden im Register b liegenden Rückkehradressen mitberücksichtigt, so daß das 'RTN' am Ende jeder Tonfolge eine einwandfreie Rückkehr auf den Befehl 'ISG L' (Zeile 59) gewährleistet.

Die schließlich erlangte Höchstgeschwindigkeit wird dadurch erzielt, daß man den ursprünglich verwendeten hinter 'ISG L' liegenden Befehl 'GTO 01' durch 'RTN' ersetzt. Im Register b wird durch das 'XEQ' in Zeile 45 zusammen mit der eben besprochenen Rückkehradresse eine zweite abgelegt, die den in früheren Versionen durch 'GTO 01' bewirkten Rücksprung auslöst. Sie gibt die Ausführung direkt an den Befehl 'RCL IND L' (Zeile 46) weiter, wodurch sich die zuvor verwendete Marke 'LBL 01' erübrigt. 'RTN' ist um etwa 15% schneller als ein kompiliertes Zwei-Byte-'GTO'.

Der für die jeweilige Tonfolge-Routine gültige Adreßzeiger und die beiden oben besprochenen Rückkehradressen belegen zusammen genau 6 Bytes, die von 'STO b' nach b zu bringen sind. Das führende siebente Byte wird (in Programmzeile 14) der Zeile 1 der QRC entnommen, um Normalisierungsprobleme, die beim Abruf der Kodes aus den Datenregistern (Zeile 46)

entstünden, zu vermeiden (stammt das erste Byte aus Zeile 1 der QRC, ist sichergestellt, daß der Kode als Alpha-Kette behandelt wird). Das 'STOP' auf Zeile 85 darf nicht durch ein 'RTN' ersetzt werden, weil das Programm sonst nicht mehr anhält. Wenn die Morsezeichen nämlich abgearbeitet werden, springt das 'ISG L' beim letzten Zeichen über das 'RTN' auf Zeile 60, so daß die Adresse des 'RCL IND L', weiterhin auf ein 'RTN' wartend, im Rückkehrstapel verbleibt. Folglich würde das Programm von Zeile 85 aus zurück auf Zeile 46 springen. Unverdrossen. Solange, bis leere Batterien das muntere Spiel abbrechen.

Bei dem vorstehend beschriebenen Verfahren werden beide Rückkehradressen durch gewöhnliche Unterprogrammaufrufe ('XEQ IND T' und 'XEQ 05') erzeugt. Das ist wesentlich einfacher, als sie mit synthetischen Methoden herzustellen, was aufwendige Berechnungen der Adreßzeigerwerte mit sich brächte und kniffliges Zusammensetzen der Rückkehradressen erforderte. Der jeder Tonfolge voranstehende Befehl 'RCL b' liefert daher gleich vollständige Kodes für die spätere Rückspeicherung nach b: die richtigen Rückkehradressen liegen, auf 'RTN's wartend, an richtiger Stelle, sowohl beim 'RCL b' als nach dem 'STO b'.

Das Ergebnis der scharfsinnigen Programmkonstruktion ist ein Morse-Programm, welches mit einer Geschwindigkeit von 85 Zeichen je Minute Morsezeichen ertönen läßt – eine ganz wesentliche Verbesserung gegenüber den ersten Versuchen. Auch das volle Fassungsvermögen des Alpha-Registers wird glanzvoll ausgenutzt: bis zu 28 Zeichen können während des Aufbaus der Botschaft auf einmal eingegeben werden. Dies ist möglich, weil der Rechner während der gesamten Texteingabe im Alpha-Modus verbleibt (vgl. Sie die in Abschnitt 6A gegebenen Erläuterungen zu Register P). Ehrgeizige synthetische Programmierer sollten sich nicht die auf Seite 13 der Ausgabe Juli '81 des PPC CJ zusammengetragenen Erkenntnisse über Register P entgehen lassen, wenn sie genau wissen wollen, wie die Zahleneinträge der Zeilen 42 und 52 dazu verwendet werden, den Inhalt von P zielbewußt abzuändern.

Hier die Gebrauchsanweisung für Clifford's Morse-Programm "MC":

1) Setzen Sie mit 'SIZE' eine Anzahl von Datenregistern fest, die um 1 größer ist als die Anzahl der in der Botschaft zu übermittelnden Zeichen.

2) 'XEQ "MC"'. Tasten Sie die Botschaft in Teilstücken, die zwischen 1 und 28 Zeichen enthalten können, ein. Der Warnton, welcher im Normalfall signalisiert, daß das Alpha-Register voll ist, zeigt hier an, daß noch genau 4 Zeichen eingetastet werden dürfen. Schließen Sie die Eingabe jeder Gruppe mit 'R/S' ab. Wenn Ihnen "NONEXISTENT" gemeldet wird, setzen Sie die Anzahl der Datenregister mit 'SIZE' herauf und fahren anschließend mit 'R/S' fort.

3) Ist die Botschaft vollständig eingegeben, wird sie, ohne einen weiteren Eintrag zu tätigen, mit 'R/S' übermittelt. Soll die Übermittlung wiederholt werden, kann dies mit 'R/S' oder 'XEQ 10' erreicht werden.

4) Will man zwischen den einzelnen Zeichen bei der Übermittlung eine größere Zeitlücke haben, kann dies dadurch bewirkt werden, daß man zwischen die Zeilen 45 und 46 beliebige Befehle, die nicht das Register L beeinträchtigen, einschiebt. Allerdings muß dann mit 'XEQ "MC"' von vorn begonnen werden, weil sich die für Register b benötigten Adressen durch das Einfügen verändern.

Wenn Sie einen optischen Lesestift besitzen, können Sie den im Anhang E abgedruckten Barcode einlesen, um das Morse-Programm in Ihren Rechner zu bekommen. Andernfalls hier ein paar Hinweise, deren Befolgung Ihnen Tastenarbeit beim Edieren des Programms erspart.

Die folgenden synthetischen Tastenzuweisungen erleichtern das Eintasten von "MC" an Hand der Programm-Auflistung: 159, 120 ('TONE P'), 159, 8 ('TONE 8') und 205, 0 (das ist der für die Übertragung von Marken zuständige 'Kollege' des Q-Boten 27, 0). Diese letzte Zuweisung wurde von Tom Cadwallader entdeckt. Sie kann dazu benutzt werden, umstandslos synthetische Marken herzustellen. Um etwa 'LBL "A"'zu erzeugen, tastet man 'XEQ A' oder 'LBL A'. Dies bringt das Zeichen "A" ins Register Q. (Löschen Sie den Befehl wieder, falls Sie ihn im PRGM-Modus eingetastet haben sollten.) Drücken Sie dann die Taste, der der neue 'LBL'-Q-Bote zugewiesen wurde, im PRGM-Modus: Sie erhalten 'LBL "A"'. Das Verfahren wurde von Valentin Albillo, einem anderen Wegbereiter der synthetischen Programmierung, entdeckt. Es kann für alle Marken "A" bis "J" verwendet werden.

Zur Herstellung der Marken ".", "," und ":" muß ein anderer Weg beschritten werden. Eine Möglichkeit besteht darin, das jeweilige Satzzeichen ins Alpha-Register zu bringen, anschließend 'ASTO X' und dann 'GTO IND X' auszuführen, und zwar alles im RUN-Modus. Dies bringt das Satzzeichen ins Register Q. Nach Erscheinen der Meldung "NONEXISTENT" schaltet man in den PRGM-Modus und ruft den 'LBL'-Q-Boten auf, um die Marke zu erlangen.

Eine andere Möglichkeit bietet unser treues Arbeitspferd, der Byte-Schnapper:

```
01 ENTER↑          LBL -Eingaben:
02 STO IND 64      192
03 SIN             0 (jeder Wert ist zulässig)
04 "::"            242, 0, Dezimalwert des Zeichens,
                           das die Marke bilden soll
```

BS auf Zeile 01 schnappt das 'STO'-Byte weg und erzeugt 'LBL ":"'. Vergessen Sie nicht, am Schluß zu 'PACK'en, denn im Gegensatz zum 'LBL'-Q-Boten werden die mit dem Byte-Schnapper erzeugten synthetischen Marken nicht automatisch der globalen Kette des Katalogs 1 eingefügt.

Drei der Textzeilen des nachfolgend aufgelisteten Morse-Programms haben keinen eindeutigen Ausdruck. Sie lauten:

Zeile	hexadezimal	dezimal
03	F4 2C 01 80 81	244 44 1 128 129
36	F2 7F 00	242 127 0
53	F2 7F 00	242 127 0

```
01♦LBL "MC"
02 SF 26
03 ",≠"
04 X<> [
05 X<> d
06 RCL b
07 FC?C 26
08 GTO 01
09 CLA
10 ASTO Z
11 X<> [
12 SIGN
13 ASTO X
14 "≠"
15 ARCL X
16 ASTO b
17♦LBL 01
18 SF 26
19 "CHARACTERS?"
20 PROMPT
21 FC?C 23
22 GTO 06
23 VIEW Z
24 CF 26
25 CLX
26 ENTER↑
27 X<> ↑
28 X=Y?
29 GTO 02
30 SF 05
31 X<> ]
32 X<> \
33 X<> [
34 X<>Y
35♦LBL 02
36 "⊢♦"
37 X<> ↑
38 X=0?
39 GTO 02
40 STO ↑
41 RDN
42 0
43 FC?C 29
44 GTO 05
45 XEQ 05
46 RCL IND L
47 STO b
48♦LBL 03
49 STO IND L
50 RDN
51♦LBL 04
52 ,
53 "⊢♦"
54♦LBL 05
55 X<> ↑
56 RDN
57 SF 25
58 XEQ IND T
59 ISG L
60 RTN
61 FS?C 25
62 GTO 03
63 FS? 26
64 GTO 07
65 DSE L
66 FC?C 05
67 GTO 01
68 STO ]
69 GTO 04
70♦LBL 06
71 LASTX
72  E3
73 +
74 LASTX
75 /
76 STO 00
77 SIGN
78 R↑
79 STO d
80 RCL 01
81 STO b
82♦LBL 07
83 RCL 00
84 SIGN
85 STOP
86♦LBL 10
87 RCL 01
88 STO b
89♦LBL ":"
90 RCL b
91 TONE 8
92 TONE 8
93 TONE 8
94 TONE ↑
95 TONE ↑
96 TONE ↑
97 RTN
98♦LBL "-"
 99 RCL b
100 TONE 8
101 TONE ↑
102 TONE ↑
103 TONE ↑
104 TONE 8
105 RTN
106♦LBL "/"
107 RCL b
108 TONE 8
109 TONE ↑
110 TONE ↑
111 TONE 8
112 TONE ↑
113 RTN
114♦LBL "?"
115 RCL b
116 TONE ↑
117 TONE ↑
118 TONE 8
119 TONE 8
120 TONE ↑
121 TONE ↑
122 RTN
123♦LBL "."
124 RCL b
125 TONE ↑
126 TONE 8
127 TONE ↑
128 TONE 8
129 TONE ↑
130 TONE 8
131 RTN
132♦LBL ","
133 RCL b
134 TONE 8
135 TONE 8
136 TONE ↑
137 TONE ↑
138 TONE 8
139 TONE 8
140 RTN
141♦LBL "0"
142 RCL b
143 TONE 8
144 TONE 8
145 TONE 8
146 TONE 8
147 TONE 8
148 RTN
149♦LBL "9"
150 RCL b
151 TONE 8
152 TONE 8
153 TONE 8
154 TONE 8
155 TONE ↑
156 RTN
157♦LBL "8"
158 RCL b
159 TONE 8
160 TONE 8
161 TONE 8
162 TONE ↑
163 TONE ↑
164 RTN
165♦LBL "7"
166 RCL b
167 TONE 8
168 TONE 8
169 TONE ↑
170 TONE ↑
171 TONE ↑
172 RTN
173♦LBL "6"
174 RCL b
175 TONE 8
176 TONE ↑
177 TONE ↑
178 TONE ↑
179 TONE ↑
180 RTN
181♦LBL "5"
182 RCL b
183 TONE ↑
184 TONE ↑
185 TONE ↑
186 TONE ↑
187 TONE ↑
188 RTN
189♦LBL "4"
190 RCL b
191 TONE ↑
192 TONE ↑
193 TONE ↑
194 TONE ↑
195 TONE 8
196 RTN
```

```
197♦LBL "3"
198 RCL b
199 TONE †
200 TONE †
201 TONE †
202 TONE 8
203 TONE 8
204 RTN
205♦LBL "2"
206 RCL b
207 TONE †
208 TONE †
209 TONE 8
210 TONE 8
211 TONE 8
212 RTN
213♦LBL "1"
214 RCL b
215 TONE †
216 TONE 8
217 TONE 8
218 TONE 8
219 TONE 8
220 RTN
221♦LBL "Z"
222 RCL b
223 TONE 8
224 TONE 8
225 TONE †
226 TONE †
227 RTN
228♦LBL "Q"
229 RCL b
230 TONE 8
231 TONE 8
232 TONE †
233 TONE 8
234 RTN
235♦LBL "J"
236 RCL b
237 TONE †
238 TONE 8
239 TONE 8
240 TONE 8
241 RTN
242♦LBL "K"
243 RCL b
244 TONE 8
245 TONE †
246 TONE 8
247 RTN
248♦LBL "F"
249 RCL b
250 TONE †
251 TONE †
252 TONE 8
253 TONE †
254 RTN
255♦LBL "U"
256 RCL b
257 TONE †
258 TONE †
259 TONE 8
260 RTN
261♦LBL "A"
262 RCL b
263 TONE †
264 TONE 8
265 RTN
266♦LBL "B"
267 RCL b
268 TONE 8
269 TONE †
270 TONE †
271 TONE †
272 RTN
273♦LBL "O"
274 RCL b
275 TONE 8
276 TONE 8
277 TONE 8
278 RTN
279♦LBL "L"
280 RCL b
281 TONE †
282 TONE 8
283 TONE †
284 TONE †
285 RTN
286♦LBL "N"
287 RCL b
288 TONE 8
289 TONE †
290 RTN
291♦LBL "I"
292 RCL b
293 TONE †
294 TONE †
295 RTN
296♦LBL "E"
297 RCL b
298 TONE †
299 RTN
300♦LBL "M"
301 RCL b
302 TONE 8
303 TONE 8
304 RTN
305♦LBL "D"
306 RCL b
307 TONE 8
308 TONE †
309 TONE †
310 RTN
311♦LBL "H"
312 RCL b
313 TONE †
314 TONE †
315 TONE †
316 TONE †
317 RTN
318♦LBL "S"
319 RCL b
320 TONE †
321 TONE †
322 TONE †
323 RTN
324♦LBL "R"
325 RCL b
326 TONE †
327 TONE 8
328 TONE †
329 RTN
330♦LBL "X"
331 RCL b
332 TONE 8
333 TONE †
334 TONE †
335 TONE 8
336 RTN
337♦LBL "V"
338 RCL b
339 TONE †
340 TONE †
341 TONE †
342 TONE 8
343 RTN
344♦LBL "G"
345 RCL b
346 TONE 8
347 TONE 8
348 TONE †
349 RTN
350♦LBL "W"
351 RCL b
352 TONE †
353 TONE 8
354 TONE 8
355 RTN
356♦LBL "Y"
357 RCL b
358 TONE 8
359 TONE †
360 TONE 8
361 TONE 8
362 RTN
363♦LBL "P"
364 RCL b
365 TONE †
366 TONE 8
367 TONE 8
368 TONE †
369 RTN
370♦LBL " "
371 RCL b
372 FC? 26
373 RTN
374 LASTX
375 LN
376 RTN
377♦LBL "C"
378 RCL b
379 TONE 8
380 TONE †
381 TONE 8
382 TONE †
383 RTN
384♦LBL "T"
385 RCL b
386 TONE 8
387 END
```

845 BYTES

ANHANG C

Quellen zur synthetischen Programmierung

1. *PPC Calculator Journal*, herausgegeben vom 'Personal Programming Center', einer nicht gewinnorientierten gemeinnützigen Vereinigung in Kalifornien, die sich dem Betrieb sogenannter Personal-Computer widmet. Die Ausgaben vom Juli '79 an (Volume 6, Number 4) bis zum gegenwärtigen Zeitpunkt enthalten eine fast unübersehbare Fülle von Informationen zum HP-41 im allgemeinen und zur synthetischen Programmierung im besonderen. Das PPC CJ ist bislang die aktuellste und umfassendste Quelle für synthetische Programme, Verfahren und Entdeckungen. Wer Mitglied des PPC werden will, wende sich an

PPC
2545 W. Camden Place
Sante Ana, CA, 92704
USA

2. *PPC Technical Notes*, herausgegeben vom 'Melbourne, Australia chapter of PPC'. Die PPC TN sind eine Veröffentlichung geringeren Umfangs als das PPC CJ, doch in besonderem Maße der synthetischen Programmierung zugewandt. Die Ausgabe 'Number 9' enthält die beste gegenwärtig verfügbare Zusammenfassung über den Mikrokode (Maschinen-Kode) des HP-41. Wer die PPC TN bestellen möchte, wende sich an

R.M. Eades
P.O. Box 15
Hampton, Victoria, 3188
AUSTRALIA

3. *PPC-UK Journal*, herausgegeben vom 'United Kingdom chapter of PPC'. Das PPC-UKJ ist eine verhältnismäßig neue Erscheinung, die ihr Augenmerk auf Arbeitsgemeinschaften und andere nützliche Hilfen für Anfänger richtet. Auskunft erhält man von

David M. Burch
Astage
Rectory Lane
Windlesham, Surrey
GU20 6BW
ENGLAND

4. *PRISMA*, herausgegeben vom

CCD-Computerclub Deutschland e.V.
Limburger Str. 15
Postfach 2129
6242 Kronberg 2

Der CCD ist ein unabhängiger, gemeinnütziger Anwenderclub, der im Oktober 1981 gegründet wurde. Er hat sich zum Ziel gesetzt, Besitzern von Taschen- und Kleinrechnern beim Einsatz ihrer Geräte zu helfen. Das meistbehandelte System ist der HP-41. PRISMA ist die Clubzeitschrift des CCD. Behandelt werden u.a. die Themen Anwendungsprogramme (stets mit Barcode abgedruckt), Hardware-Umbau und Synthetische Programmierung. Es existiert eine Programmbibliothek, deren Programme gegen Unkostenerstattung auf Magnetkarten oder Kassetten geschrieben werden.

5. Der Katalog der *Hewlett-Packard Users' Library* enthält nur wenige synthetischen Programme, weil sie von der Bibliothek bis Januar '82 nicht angenommen wurden. Daher spiegelt der verfügbare Katalog nicht unbedingt den Umfang wieder, in dem sich synthetische Programme inzwischen durchgesetzt haben und in der Bibliothek enthalten sind. Die Anschrift der Bibliothek lautet

HP Users' Library
1000 N.E. Circle Boulevard
Corvallis, Oregon 97330
USA

6. Die in Genf ansässige europäische HP-Benutzerbibliothek UPLE (*Users' Program Library in Europe*) hat ihre Tätigkeit Ende Oktober 1983 eingestellt und sich entschlossen, ihre Leistungen auf nationaler Ebene fortzuführen. *HP-Deutschland* strebt an, Software-Dienste in enger Zusammenarbeit mit dem CCD (Punkt 4) anzubieten. Bislang ist jedoch noch keine Vereinbarung erzielt worden.

7. *HP Key Notes*, früher von Hewlett-Packard als 'Rundschreiben' veröffentlicht, doch nunmehr eingestellt. Eine begrenzte Anzahl synthetischer Programme ist in den Key Notes erschienen, und zwar seit Januar '82, als die Benutzerbibliotheken begannen, synthetische Programme aufzunehmen. Seit August '83 erscheinen die Key Notes als Abschnitt des von Hewlett-Packard vierteljährlich herausgegebenen *Portable-Computation Guide*, der Mitgliedern der 'HP Users' Library' (Punkt 5) kostenlos zugesandt wird. Wer nachträglich frühere Ausgaben der Key Notes erlangen möchte, wende sich an

HP Key Notes
1000 N.E. Circle Boulevard
Corvallis, Oregon 97330
USA

8. *Synthetische Programmierung auf dem HP-41C/CV* von Bill Wickes. Dieses Buch war die erste Publikation über synthetische Programmierung. Die zweite deutsche Auflage berücksich-

tigt bereits den Byte-Schnapper, der zu der Zeit, als das amerikanische Original geschrieben wurde, noch nicht bekannt war. Wickes entwickelt die synthetische Programmierung anders, als es in diesem Buch geschieht. Jedes Thema wird dort vollständig und in die Tiefe gehend abgehandelt, bevor man sich dem nächsten Punkt zuwendet. Das macht die Lektüre etwas schwieriger. Wer jedoch zuerst 'Synthetisches Programmieren auf dem HP-41 – leicht gemacht' liest, hat gewiß keine Schwierigkeit mehr mit dem Buch von Wickes und wird erheblichen Nutzen daraus ziehen. Bemerkenswerte Beispiele und genaue Analysen verschiedener Programme helfen dem Leser, die synthetische Programmierung einschränkungslos zu meistern. Es läßt sich bestellen beim

Heldermann Verlag
Herderstraße 6-7
D-1000 Berlin 41

9. Das *PPC ROM User's Manual* ist ein Handbuch, welches zum PPC ROM gehört. Der PPC ROM ist ein Kunden-Modul für den HP-41, der von den PPC-Mitgliedern entworfen wurde und von Hewlett-Packard hergestellt wird. Er enthält 122 Programe, davon über 60 synthetisch. Sie werden alle Zeile für Zeile im Handbuch analysiert. PPC ROM einschließlich Benutzerhandbuch können bei

Microtec GmbH
Lepsiusstraße 81
D-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 ein ganz ungewöhnliches Erzeugnis erwerben: einen sogenannten *'Weck-Modul'*. So mancher genervte HP-41-Synthetiker wird einen solchen Modul zu schätzen wissen. Wenn sich der HP-41 nämlich aufgrund unsachgemäßer Behandlung beim synthetischen Programmieren in einen hartnäckigen Tiefschlaf zurückgezogen hat, kann man ihn durch kurzzeitiges Einstecken des Weck-Moduls, bei gleichzeitigem Entfernen der Batterie, garantiert *sofort* wieder beleben.

10. *Tricks, Tips und Routinen für Taschenrechner der Serie HP-41* ist die deutsche Ausgabe einer von John Dearing bearbeiteten Programmsammlung, die viele der PPC ROM Routinen (Punkt 9), synthetische und nicht-synthetische, enthält. Allen in die Sammlung aufgenommenen Programmen ist eine Benutzeranleitung vorangestellt. Überdies sind eine große Anzahl nicht-synthetischer Programm-Tips enthalten. Das Buch ist erhältlich beim

Heldermann Verlag
Herderstraße 6-7
D-1000 Berlin 41

11. Der ***HP-41 SYNTHETIC Quick Reference Guide*** ist eine in die Brieftasche passende Kurzanleitung (8,9 × 15,2 cm) zur synthetischen Programmierung. Sie ist nur unerheblich

größer als die diesem Buch beigefügte Plastik-Karte (Punkt 12) und enthält XROM-Listen, eine Speicher-Abbildung, eine Byte-Tabelle, eine Ton-Tabelle, Ausführungszeiten von Funktionen und einiges andere exotische Zuckerwerk aus der Synthetik-Tüte. Es handelt sich jedoch nur um einen Kleinstführer zum Nachschlagen, nicht zum Lernen. Daher werden dort, wo es erforderlich ist, lediglich Verweise auf das PPC CJ oder sonstige Quellen gegeben. Bestellungen sind zu richten an

J.J. Smith
Dept. SPME
19451 Mesa Drive
Villa Park, CA 92667
USA

12. Die diesem Buch beigefügte *HP-41C Quick Reference Card for Synthetic Programming* (QRC) (7,3 × 15,2 cm) kann vom

Heldermann Verlag
Herderstraße 6-7
D-1000 Berlin 41

auch als Einzelstück erworben werden. Zugleich gibt es dort – solange der Vorrat reicht – eine noch weiter gestutzte Fassung, die frühere *HP-41C Combined Hex/Decimal Byte Table*. Sie ist nur 7,3 × 11,6 cm groß und paßt daher in jede HP-41C-Tragetasche. Diese Karte, ebenfalls aus Plastik, hat keine farbig abgesetzten Bereiche, enthält aber im wesentlichen dieselben Informationen wie die QRC; sie entbehrt lediglich der Flag-Tabelle und der Zusammenfassung über den Aufbau der Mehr-Byte-Befehle.

ANHANG D

Die Kurzanleitung zur synthetischen Programmierung ('QRC')

Die QRC ist eine Plastik-Karte von 7,3 × 15,2 cm, die eine Fülle von Informationen zur synthetischen Programmierung enthält und daher unentbehrlich ist, wenn man in diesem Gebiet auf dem HP-41 tätig werden will. Sie liegt jedem Exemplar von 'Synthetisches Programmieren auf dem HP-41 – leicht gemacht' bei.

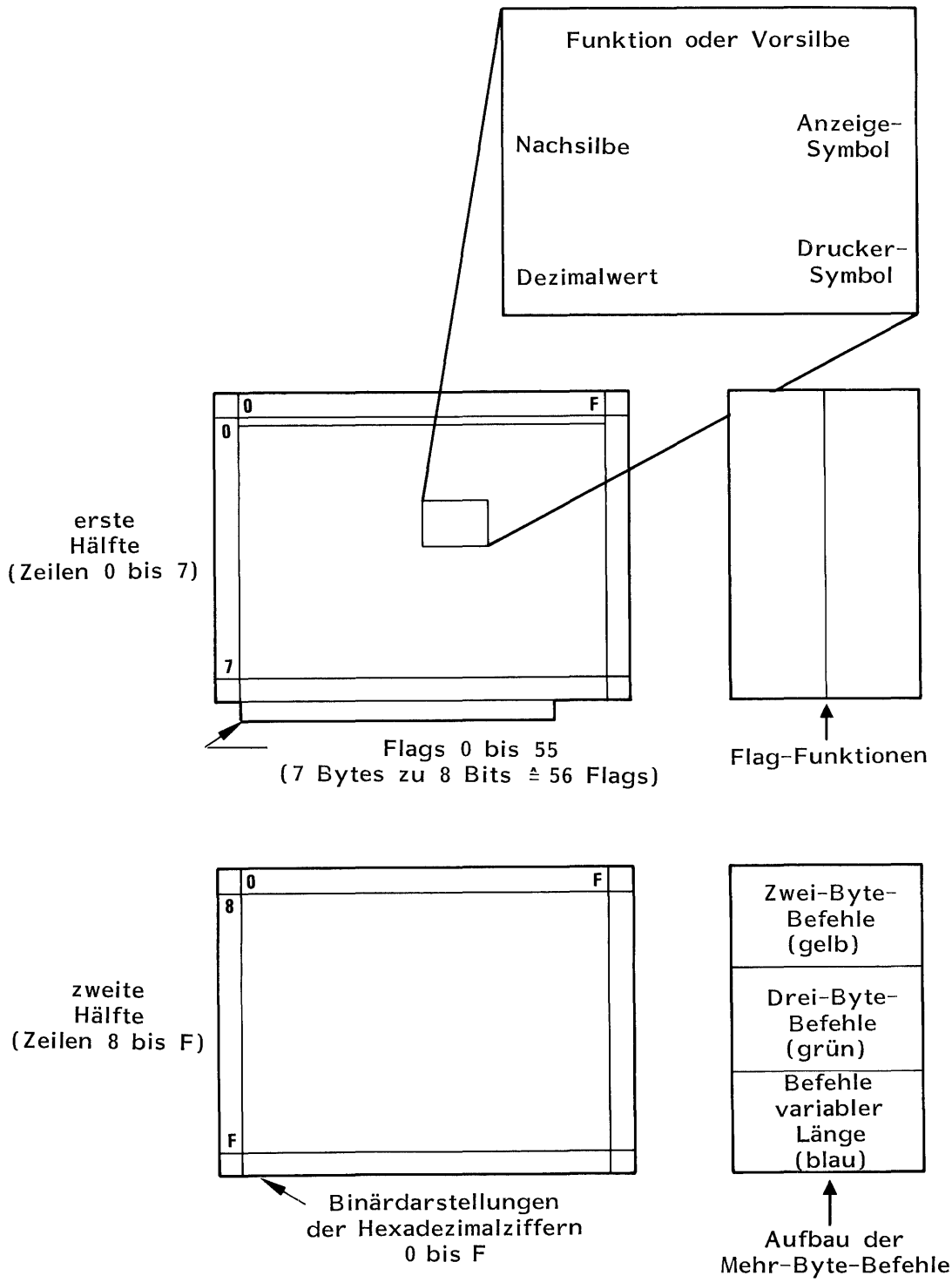
Der linke größere Teil der Karte wird von der Byte-Tabelle belegt. Jedes Kästchen der Tabelle enthält alle die Eintragungen, die zum Verständnis der verschiedenen Bedeutungen oder Darstellungen eines Bytes (erläutert in den Kapiteln 1 und 2) erforderlich sind. Befragen Sie dazu die nachfolgend abgedruckte 'Legende der QRC'.

Die in der Anzeige vorgewiesenen Zeichen werden für die zweite Tabellenhälfte (Zeilen 8 bis F) nicht angeführt, weil für diese Bytes stets das Vollzeichen (alle 14 Anzeigesegmente eingeschaltet) erscheint. Auf diese Weise bleibt in der Mitte jedes Kästchens der Zeilen 8 bis F Raum für die volle Angabe der Nachsilben indirekter Befehle. Die eintragenen Druck-Zeichen sind jene, die sich ergeben, wenn das jeweilige Byte im Alpha-Register liegt und 'PRA' ausgeführt wird. Am Fuß jeder Tabellenhälfte ist eine 'Bit-Leiste' abgedruckt, der man die Binärdarstellung der Hexadezimalziffern 0 bis F entnehmen kann.

Rechterhand zur ersten Tabellenhälfte finden Sie eine Übersicht über die Bedeutung der 56 Flags des HP-41. Rechterhand zur zweiten Tabellenhälfte steht eine Zusammenfassung über den Aufbau der Mehr-Byte-Befehle des HP-41 (ausführlich besprochen in Kapitel 3), die besonders nützlich ist, wenn man mit "LB" arbeitet und dafür die Dezimalwerteingaben braucht.

Einige rätselhaften Erscheinungen in Zusammenhang mit der QRC müssen noch erwähnt werden: Die Bytes aus der zweiten Tabellenhälfte sind in Textzeilen unsichtbar, solange diese Textzeilen Bestandteil einer Programm-Auflistung sind; ist der Textbefehl aber ausgeführt, liegen diese Bytes im Alpha-Register; dann fördert ein 'PRA' sie zutage. Die schattierten Zeichen (Zeilen A, B, D und E) geben Anlaß zu fremdartigem Verhalten des Druckers (vgl. Abschnitt 2E). Die Zeile 0 zeigt in einer in Kleindruck abgesetzten Kopfleiste nichtprogrammierbare Funktionen, die unter Verwendung der zugehörigen Zahlen 0 bis 15 mit "MK" auf Tasten gelegt werden können (in Abschnitt 4A finden Sie dazu Einzelheiten). In Zeile 1 liegt die nichtswürdige Funktion 'W$^{\tau}$', die keine andere Wirkung hat als die, unter ungünstigen Umständen, die hier nicht weiter verfolgt werden sollen, das Tastenfeld solange zu blockieren, bis die Batterien entfernt werden. Die Bytes 'SPARE' in den Zeilen A und B führen zu Zwei-Byte-'NOP's.

Sofern Ihnen die vorstehende zusammenfassende Beschreibung der QRC zu wenig sagt, müssen Sie Kapitel 1 und 2 noch einmal lesen.



LEGENDE DER QRC

Aufbau der Mehr-Byte-Befehle des HP-41

Structure of multi-byte instructions
Two-byte instructions
STO 16=145,16 DSE IND 55 =151,183
LBL e =207,127 FS?C IND Y =170,242
RCL b =144,124 TONE 89 =159,89
X<>M=206,117 ST+ IND N =146,246
LBL Q =207,121 VIEW H(109)=152,109
Two-byte special cases
GTO IND=174,reg. XEQ IND=174,128+r
GTO IND 09=174,9 XEQ IND X=174,243
XROM i,j =160+i/4,64(i mod 4)+j
WSTS =XROM 30,10 =167,138
short form GTO =177+label,0
GTO 12 =189,0
Three-byte instructions
long form GTO =208,0,label
GTO 32 =208,0,32
XEQ =224,0,label
XEQ D =224,0,105
END =192,0,9+sum of status indicators
32(.END.), 4(rePACK), 2(decompile)
Variable length instructions
TEXT =240+n, n character bytes
Append symbol counts as first char.
"& =241,38 "⊢)? =243,127,41,63
GTO " =29,240+n, n character bytes
GTO "XYZ =29,243,88,89,90
XEQ " =30,240+n, n character bytes
XEQ "A =30,241,65 (synthetic)
LBL " =192,0,241+n, (key), n chars.
LBL ": =192,0,242,0,58 (synthetic)

Liste der 56 Flags des HP-41

FLAGS (Register d)

00-10 general purpose
11 auto execute
12 doublewide
13 lower case
14 overwrite
15-16 IL printer
0 0 MAN
0 1 NORM
1 0 TRACE
1 1 TR/STACK
17 record incomplete
18 ⎫ general use
19 ⎬ cleared at
20 ⎭ turn-on
21 prtr enable
22 num. entry
23 alpha entry
24 range ignore
25 error ignore
26 audio enable
27 USER mode
28 dec./comma
29 digit grouping
30 CAT
31 timer DMY/MDY
32 manual IL I/O
33 IL absolute manual
34 not used
35 not used
36-39 number of digits
40-41 display
0 0 SCI
0 1 ENG
1 0 FIX
1 1 FIX/ENG
42-43 trig mode
0 0 DEG
0 1 RAD
1 0 GRAD
1 1 RAD
44 cont. ON
45 system data entry
46 partial key sequence
47 SHIFT
48 ALPHA
49 low BAT
50 message
51 SST
52 PGRM
53 I/O
54 PSE
55 printer existence

HP-41C QUICK REFERENCE CARD FOR SYNTHETIC PROGRAMMING © 1982, SYNTHETIX

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

Byte-Tabelle des HP-41 (erste Hälfte)

HP-41C QUICK REFERENCE CARD FOR SYNTHETIC PROGRAMMING © 1982, SYNTHETIX

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

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

ANHANG E

Barcodes für die in diesem Buch enthaltenen Programme

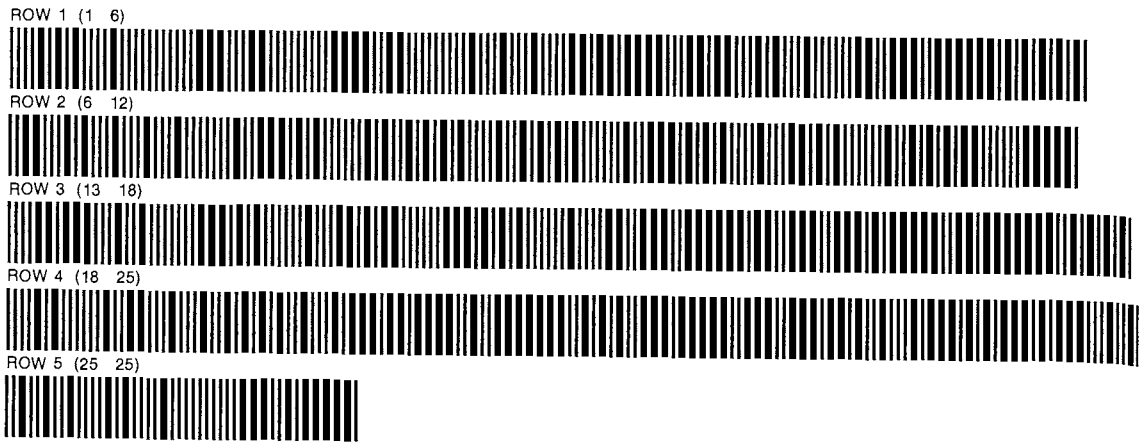
Nachstehend finden Sie die Barcodes aller in diesem Buch besprochenen Programme, so daß Sie sie bequem mit einem optischen Lesestift 82153A in Ihren HP-41 bringen können. Sie sollten versuchen, sich einen Lesestift auszuleihen, wenn Sie keinen besitzen, weil Sie dadurch viel Zeit sparen können.

Schützen Sie die Barcodes beim Abtasten stets durch eine saubere durchsichtige Plastik-Folie. Sollten Sie ausnahmsweise 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.

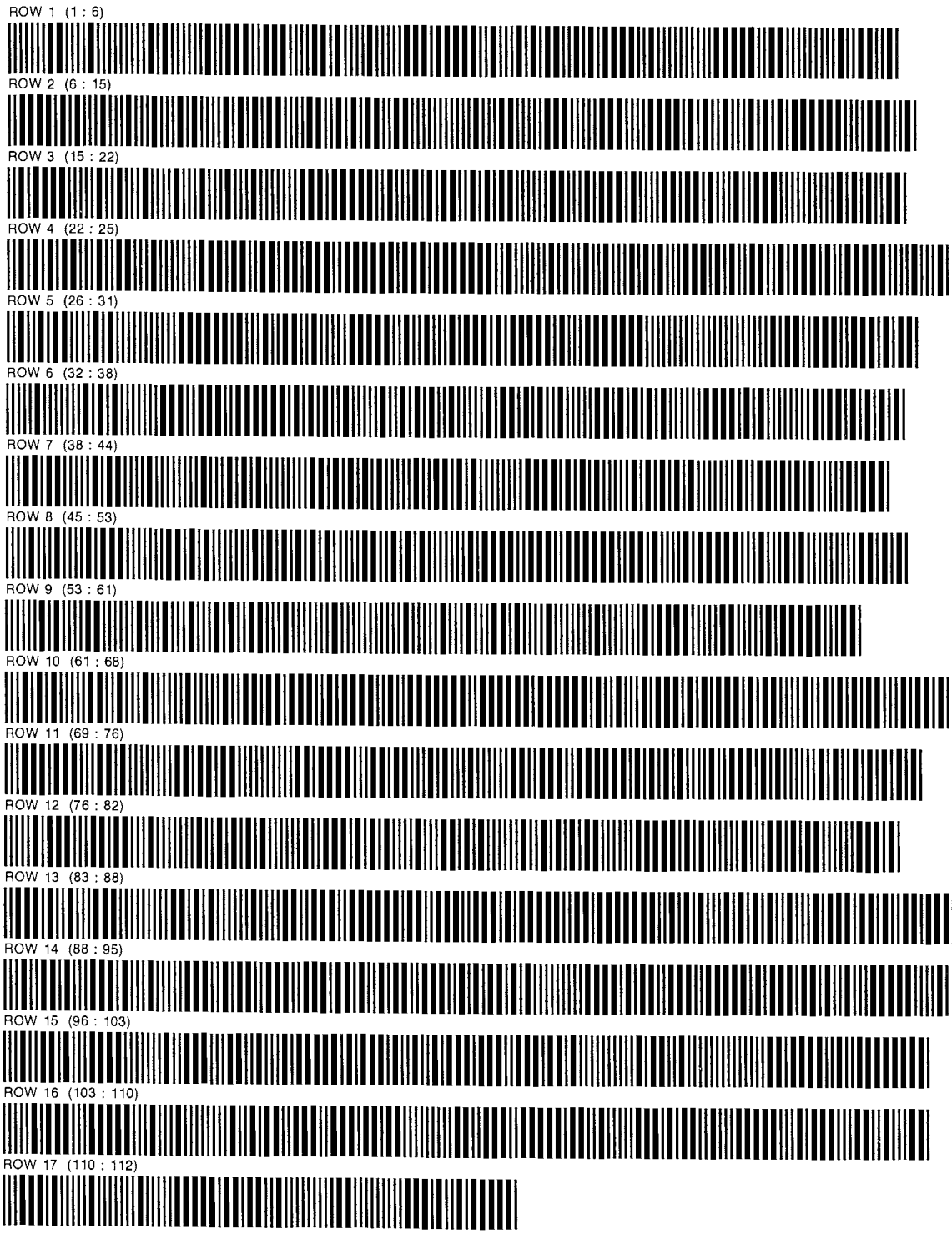
Die abgedruckten Barcodes wurden an Hand von Musterdrucken getestet und für lesbar befunden. Falls Sie dennoch Fehlermeldungen bekommen, prüfen Sie, ob sich in der betreffenden Zeile vielleicht unvollständige Striche befinden. Diese sollten Sie im Zweifelsfall vorsichtig nachschwärzen. Es hilft häufig auch, die Abtastgeschwindigkeit mit Hilfe eines Lineals zu erhöhen oder den Lesestift in einem anderen Winkel anzusetzen. Wenn alles fehlschlägt, müssen Sie den Lesestift wechseln.

Besitzer eines Kartenlesers sollten die eingelesenen Programme zusätzlich auf Karten speichern; dann können Sie dieses Buch auch ruhig einmal Ihrem Haustier überlassen, ohne gleich befürchten zu müssen, wegen unleserlich gewordener Barcodes ein weiteres Exemplar erwerben zu müssen. Auch Kassetten sind geeignet, die Programme dauerhaft aufzuzeichnen, das X-Memory jedoch weniger, denn es ist leider einem "MEMORY LOST", das Anfängern der synthetischen Programmierung immer wieder droht, hilflos ausgeliefert.

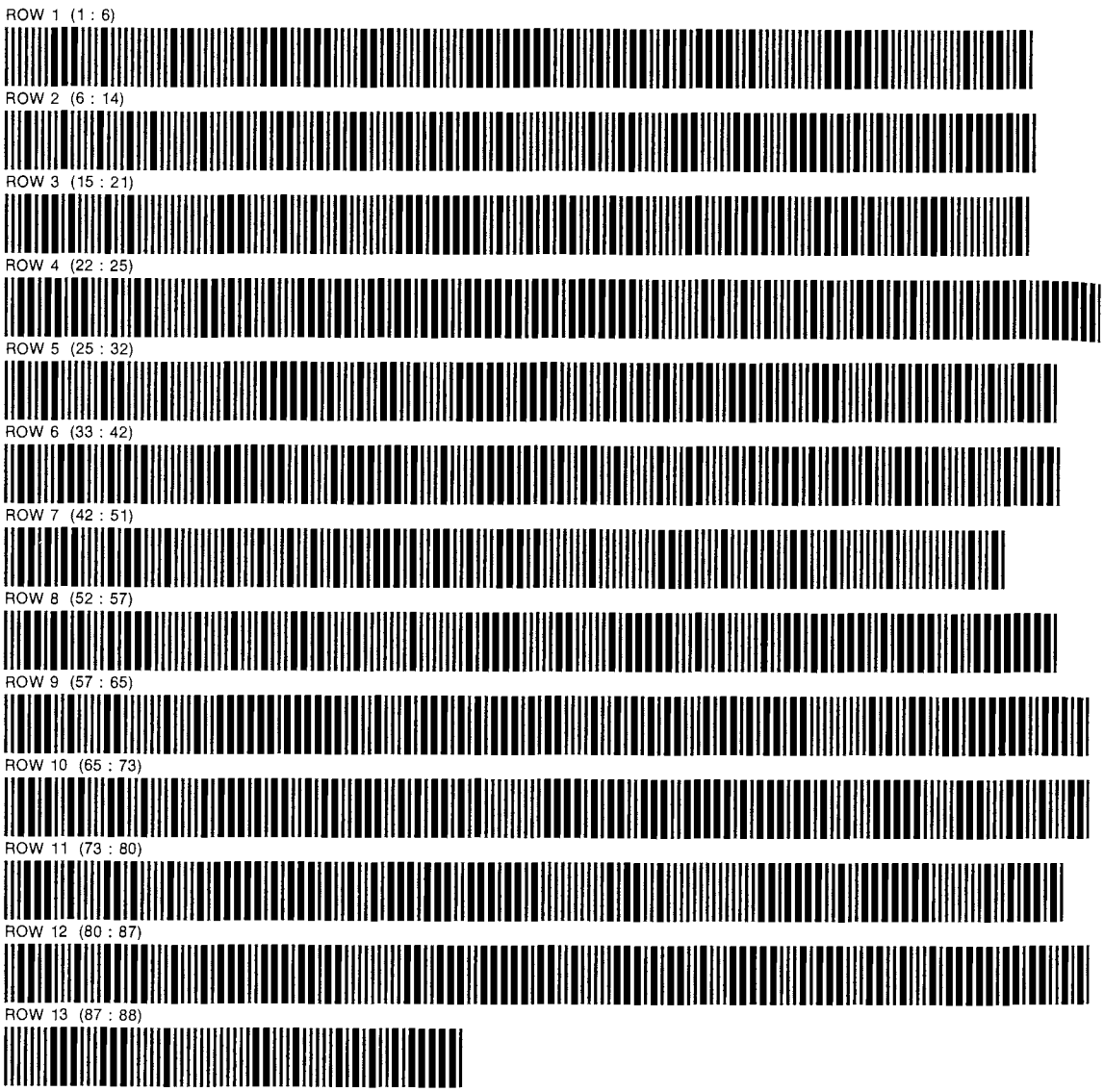
"DC" (Dezimalwert in Zeichen wandeln; entspricht 'XTOA')
Speicherplatzbedarf: 8 Register



"LB" (Bytes laden)
Speicherplatzbedarf: 31 Register



"LBX" (Bytes mit X-Funktionen laden)
Speicherplatzbedarf: 23 Register



"MK" (Tastenzuweisungsprogramm)
Speicherplatz: 45 Register

ROW 1 (1 : 5)

ROW 2 (6 : 12)

ROW 3 (12 : 19)

ROW 4 (19 : 22)

ROW 5 (23 : 28)

ROW 6 (29 : 31)

ROW 7 (31 : 35)

ROW 8 (35 : 40)

ROW 9 (40 : 48)

ROW 10 (48 : 57)

ROW 11 (58 : 67)

ROW 12 (68 : 77)

ROW 13 (78 : 85)

ROW 14 (85 : 93)

ROW 15 (93 : 99)

ROW 16 (99 : 103)

ROW 17 (103 : 108)

ROW 18 (109 : 116)

ROW 19 (116 : 121)

ROW 20 (121 : 128)

ROW 21 (129 : 136)

ROW 22 (137 : 143)

ROW 23 (143 : 149)

ROW 24 (149 : 154)

ROW 25 (154 : 154)

"MKX" (Tastenzuweisungsprogramm mit X-Funktionen)
Speicherplatzbedarf: 18 Register

ROW 1 (1 : 3)

ROW 2 (3 : 8)

ROW 3 (8 : 14)

ROW 4 (14 : 18)

ROW 5 (18 : 26)

ROW 6 (27 : 31)

ROW 7 (31 : 37)

ROW 8 (38 : 41)

ROW 9 (41 : 48)

ROW 10 (49 : 51)

"RAMBC" (Bytes zählen)
Speicherplatzbedarf: 23 Register

ROW 1 (1 : 3)

ROW 2 (4 : 8)

ROW 3 (8 : 14)

ROW 4 (15 : 23)

ROW 5 (23 : 27)

ROW 6 (28 : 31)

ROW 7 (31 : 38)

ROW 8 (39 : 43)

ROW 9 (44 : 50)

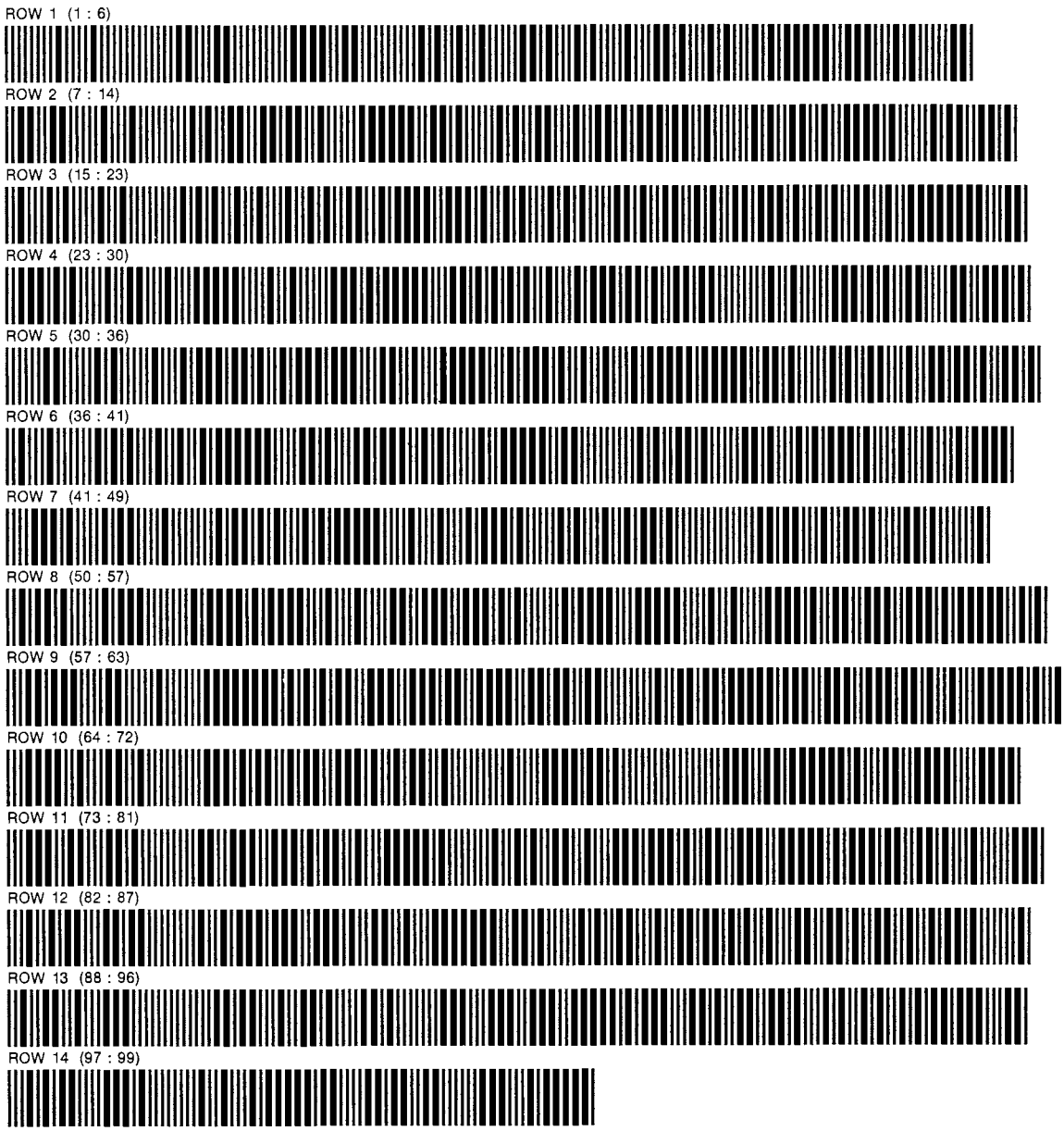
ROW 10 (50 : 56)

ROW 11 (57 : 63)

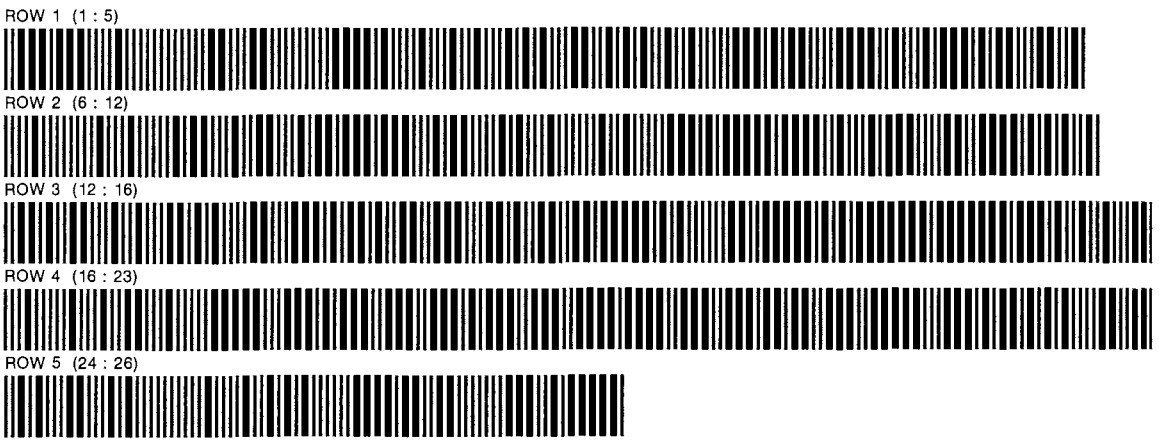
ROW 12 (63 : 69)

ROW 13 (70 : 70)

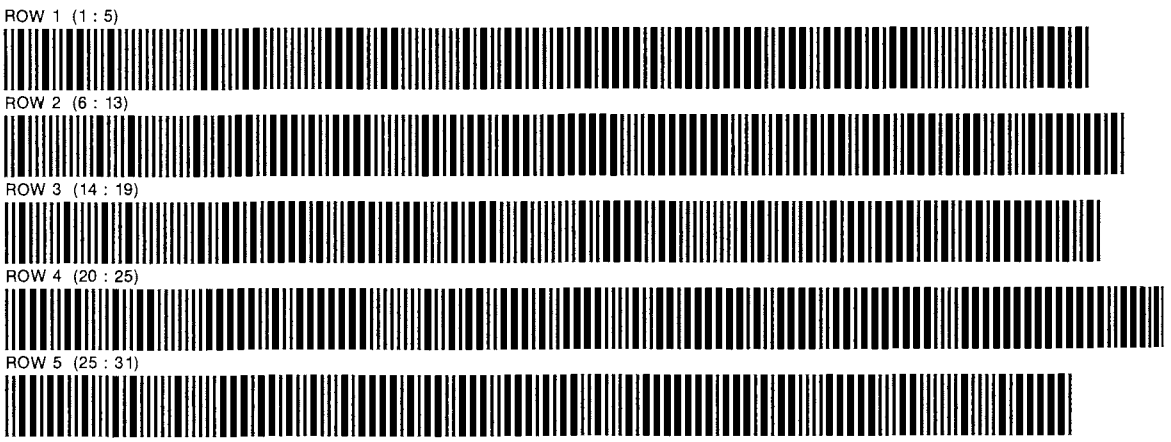
"SA" und "RA" (Weckaufträge speichern und rückrufen)
Speicherplatzbedarf: 25 Register



"EFT" (X- und Zeit-Funktionen aufrufen)
Speicherplatzbedarf: 9 Register



"SK" und "RK" (Tastenzuweisungen aufheben und wiedergewinnen)
Speicherplatzbedarf: 10 Register



"SOLVE" (f(x) = 0 lösen)
Speicherplatzbedarf: 14 Register

ROW 1 (1 : 2)

ROW 2 (2 : 7)

ROW 3 (7 : 10)

ROW 4 (10 : 19)

ROW 5 (20 : 29)

ROW 6 (30 : 39)

ROW 7 (40 : 48)

ROW 8 (48 : 49)

"CU" (Vorhang versetzen)
Speicherplatzbedarf: 10 Register

ROW 1 (1 : 5)

ROW 2 (5 : 10)

ROW 3 (11 : 21)

ROW 4 (21 : 29)

ROW 5 (29 : 35)

ROW 6 (35 : 35)

"IN" (Ausführungszeiten messen)
Speicherplatzbedarf: 47 Register

ROW 1 (1 : 8)

ROW 2 (8 : 13)

ROW 3 (13 : 17)

ROW 4 (17 : 26)

ROW 5 (26 : 31)

ROW 6 (31 : 40)

ROW 7 (40 : 52)

ROW 8 (53 : 61)

ROW 9 (62 : 66)

ROW 10 (66 : 71)

ROW 11 (72 : 79)

ROW 12 (79 : 85)

ROW 13 (86 : 94)

ROW 14 (94 : 98)

ROW 15 (99 : 106)

ROW 16 (107 : 116)

ROW 17 (117 : 124)

ROW 18 (124 : 133)

ROW 19 (134 : 139)

ROW 20 (140 : 147)

ROW 21 (148 : 155)

ROW 22 (156 : 163)

ROW 23 (164 : 172)

ROW 24 (172 : 177)

ROW 25 (178 : 185)

ROW 26 (185 : 186)

"MC" (Morsezeichen senden)
Speicherplatzbedarf: 121 Register

ROW 1 (1 : 3)

ROW 2 (4 : 10)

ROW 3 (11 : 18)

ROW 4 (18 : 19)

ROW 5 (20 : 27)

ROW 6 (28 : 35)

ROW 7 (36 : 43)

ROW 8 (43 : 49)

ROW 9 (50 : 58)

ROW 10 (58 : 65)

ROW 11 (65 : 72)

ROW 12 (73 : 83)

ROW 13 (84 : 90)

ROW 14 (91 : 97)

ROW 15 (98 : 102)

ROW 16 (103 : 108)

ROW 17 (108 : 114)

ROW 18 (114 : 120)

ROW 19 (120 : 125)

ROW 20 (126 : 132)

ROW 21 (132 : 137)

ROW 22 (138 : 143)

ROW 23 (143 : 149)

ROW 24 (149 : 155)

ROW 25 (155 : 160)

ROW 26 (161 : 166)

ROW 27 (166 : 173)

ROW 28 (173 : 178)

ROW 29 (178 : 183)

ROW 30 (184 : 189)

ROW 31 (189 : 195)

ROW 32 (196 : 201)

ROW 33 (201 : 206)

ROW 34 (207 : 213)

ROW 35 (213 : 218)

ROW 36 (219 : 224)

ROW 37 (224 : 229)

ROW 38 (230 : 235)

ROW 39 (235 : 242)

ROW 40 (242 : 247)

ROW 41 (247 : 252)

ROW 42 (253 : 258)

ROW 43 (258 : 263)

ROW 44 (264 : 269)

ROW 45 (269 : 275)

ROW 46 (275 : 281)

ROW 47 (281 : 286)

ROW 48 (287 : 292)

ROW 49 (292 : 297)

ROW 50 (298 : 304)

ROW 51 (305 : 310)

ROW 52 (310 : 315)

ROW 53 (316 : 321)

ROW 54 (321 : 327)

ROW 55 (327 : 333)

ROW 56 (334 : 338)

ROW 57 (339 : 344)

ROW 58 (344 : 349)

ROW 59 (350 : 355)

ROW 60 (355 : 360)

ROW 61 (361 : 366)

ROW 62 (366 : 371)

ROW 63 (372 : 377)

ROW 64 (377 : 382)

ROW 65 (383 : 387)

ANHANG F (Nachtrag des Übers.)

Programm-Auflistungen der in den Programmen dieses Buches angesprochenen PPC ROM-Routinen

Da wohl kaum allen Lesern der deutschen Übersetzung der PPC ROM zur Verfügung steht, werden nachstehend jene PPC ROM-Routinen, auf welche die Programme in Abschnitt 4E und Anhang A zugreifen, ausgedruckt. Somit braucht keiner auf den Gebrauch der nützlichen Programme "RA" und "SA" und vor allem auf eigenständige Untersuchungen mit dem leistungsfähigen Meßprogramm aus Anhang A mangels PPC ROM zu verzichten. Die Routinen sind so, wie sie im PPC ROM liegen, also mit den dort gültigen Zeilennummern, aufgelistet. Ihre Leistung wird nur ganz kurz erläutert. Wer mehr wissen will, muß auf den PPC ROM und das umfangreiche Benutzerhandbuch zurückgreifen.

"F?" ermittelt die Anzahl der freien Register zwischen dem '.END.' und den belegten Tastenzuweisungsregistern. Ein halbes erstes Tastenzuweisungsregister wird dabei berücksichtigt.

```
195♦LBL "F?"
196 XROM "LF"
197♦LBL 11
198 INT
199 LASTX
200 FRC
201  E3
202 *
203 X<>Y
204 .5
205 FC? 10
206 SIGN
207 -
208 -
209 END
```

"LF" ermittelt die absoluten Adressen des ersten und letzten freien Registers, vermindert um 16, in der Form 'abc,lmn'.

```
01♦LBL "LF"
02 XROM "E?"
03 17
04 -
05  E3
06 /
07 177
08 +
09 XROM "OM"
10 "*0*****"
11 ,
12 ENTER↑
13 DSE T
14 GTO 14
15♦LBL 00
16 X<> IND T
17 X=Y?
18 GTO 14
19 X<> [
20 "├0"
21 STO \
22 ARCL X
23 RDN
24 RCL \
25 X<> IND T
26 ISG T
27 GTO 00
28♦LBL 14
29 X<> [
30 ARCL X
31 X<> \
32 SF .0
33 X=Y?
34 DSE T
35 CF 10
36 X<> Z
37 X<> c
38 R↑
39 RTN
```

Zeile 10: F8 2A 10 2A 00 00 2A 2A F0

"E?" ermittelt die absolute Adresse des '.END.'

```
195♦LBL "E?"
196 RCL c
197 XROM "2D"
198 16
199 MOD
200 LASTX
201 X↑2
202 *
203 RCL [
204 +
205 CLA
206 END
```

"2D" setzt die beiden letzten Bytes des X-Registers in zwei Dezimalzahlen, die nach X und M gelangen, um.

```
 94♦LBL "2D"
 95 "*"
 96 X<> [
 97 X<> \
 98 ASHF
 99 "├♦┴♦'♦♦"
100 X<> [
101 X<> \
102 X<> [
103 "├♦♦β"
104 RCL [
105 INT
106 +
107 RCL \
108 *
109 ST+ [
110 X<> \
111 RCL ]
112 INT
113 HMS
114 *
115 RCL ]
116 +
117  E1
118 ST* [
119 *
120 X<> [
121 RTN
```

″OM″ versetzt den Vorhang auf die absolute Adresse 16. Vorsicht beim selbständigen Gebrauch.

```
142♦LBL "OM"
143 XE0 14
144 "%×i×"
145 X<> [
146 STO \
147 "├**"
148 X<> \
149 CLA
150 X<> c
151 RTN


183♦LBL 14
184 RCL c
185 STO [
186 "├*****"
187 X<> [
188 X<> d
189 CF 00
190 CF 01
191 CF 02
192 CF 03
193 X<> d
194 RTN
```

Zeile 144: F5 1F F0 01 69 01

″CX″ versetzt den Vorhang auf die durch den Inhalt von X festgelegte absolute Adresse. Vorsicht beim selbständigen Gebrauch.

```
128♦LBL "CX"
129 XROM "C?"
130 -
131♦LBL "CU"
132 ABS
133 RDN
134 RCL c
135 STO [
136 "├****"
137 11
138 X<> [
139 X<> d
140 STO ]
141♦LBL 00
142 RDN
143 X<> L
144 INT
145 X=0?
146 GTO 14
147 2
148 /
149 RCL [
150 X<>Y
151 FRC
152 X=0?
153 GTO 13
154♦LBL 01
155 FC?C IND Y
156 SF IND Y
157 FC? IND Y
158 CHS
159 X>0?
160 GTO 13
161 FC? IND Y
162 CHS
163 DSE Y
164 GTO 01
165♦LBL 13
166 DSE [
167 GTO 00
168♦LBL 14
169 X<> ]
170 X<> d
171 STO [
172 "├ABC"
173 X<> \
174 X<> c
175 RDN
176 CLA
177 RTN
```

"C?" ermittelt die absolute Adresse von R_{00} (vgl. 'Dearing', 1-17).

```
46*LBL "C?"          62 SF 10
47 RCL c             63 FS?C 13
48*LBL 14            64 SF 11
49 STO [             65 FS?C 14
50 "⊢**A"            66 SF 13
51 X<> [             67 FS?C 15
52 X<> d             68 SF 14
53 CF 01             69 FS?C 16
54 CF 02             70 SF 15
55 CF 04             71 X<> d
56 CF 07             72  E38
57 FS?C 10           73 /
58 SF 07             74 INT
59 FS?C 11           75 DEC
60 SF 09             76 RTN
61 FS?C 12
```

"BC" löscht einen Block von Datenregistern gemäß der in X liegenden Kontrollzahl. Das Programm ist äquivalent zum Befehl 'CLRGX' aus dem HP-41CX (vgl. 'Dearing', 10-23).

```
208*LBL "BC"
209 SIGN
210 CLX
211*LBL 13
212 STO IND L
213 ISG L
214 GTO 13
215 RTN
```

"RF" versetzt die Flags in den Zustand, den sie nach einem "MEMORY LOST" haben, mit ler Ausnahme 'FIX 2' statt 'FIX 4' (vgl. 'Dearing', 6-7).

```
17*LBL "RF"
18 ",X*"
19 ASTO d
20 CF 03
21 CLA
22 RTN
```

Zeile 18: F4 2C 02 80 00

″DP″ setzt eine in X liegende Dezimalzahl in das Adreßzeiger-Format um, welches benötigt wird, um mit 'STO b' eine RAM-Stelle anspringen zu können. Die Dezimalzahl wird in Bytes (nicht in Registern) ausgedrückt, gerechnet vom Fuß des Adreßraumes (Register T) aus.

```
67♦LBL "DP"
68 7
69 XROM "QR"
70 X<>Y
71 16
72 ST* Z
73 X↑2
74 XROM "QR"
75 X<> Z
76 +
77 CLA
78 XROM "DC"
79 XROM "DC"
80 RCL [
81 RTN
```

″QR″ berechnet das Vielfache des Nenners und den Rest des in Y und X liegenden Bruches (vgl. 'Dearing', 15-12).

```
82♦LBL "QR"
83 X<>Y
84 STO ]
85 X<>Y
86 MOD
87 ST- ]
88 LASTX
89 ST/ ]
90 CLX
91 X<> ]
92 X<>Y
93 RTN
```

″DC″ ist äquivalent zur Routine ″DC″ in Abschnitt 4B und zur X-Funktion 'XTOA' (vgl. 'Dearing', 5-6).

```
176♦LBL "DC"
177 INT
178 256
179 MOD
180 LASTX
181 +
182 OCT
183 X<> d
184 FS?C 11
185 SF 12
186 FS?C 10
187 SF 11
188 FS?C 09
189 SF 10
190 FS? 07
191 SF 09
192 FS? 06
193 SF 08
194 X<> d
195 X<> [
196 RCL \
197 "⊢*"
198 X<> ]
199 X<>Y
200 STO \
201 X<> †
202 "⊢*"
203 STO †
204 RDN
205 X<> ]
206 X<> \
207 STO [
208 RDN
209 END
```

"XE" verschafft Zugang zu beliebigen ROM-Stellen. Die Einstiegsstelle muß als ROM-Adresse in M liegen. Der Zugriff hat Unterprogramm-Charakter: ein 'RTN' im ROM schickt den Adreßzeiger ins RAM zurück.

```
119*LBL "XE"
120 XEQ 14
121*LBL 14
122 "⊢**"
123 STO \
124 RDN
125 SF 14
126 RCL b
127 X<> \
128 FC? 14
129 CLA
130 FC?C 14
131 RTN
132 "⊢***"
133 X<> [
134 X<> \
135 X<> [
136 "⊢**"
137 STO \
138 X<> ]
139 X<> \
140 "⊢**"
141 X<> \
142 STO b
```

STICHWORTVERZEICHNIS

K.Albers: HP-41- und andere Barcodes mit dem HP-IL-System.
Der Barcodelesestift ist ein preiswertes, dabei jedoch sehr effizientes Zubehörteil für das Einlesen von Daten in Register oder zum Programmieren von Verarbeitungssystemen. Barcodes sind die kostengünstigste Methode der externen Datenspeicherung und deren Massenverbreitung. Über HP-41- aber auch andere Barcodes, ihre teils verblüffend ergiebige Anwendung, Herstellung und ihren Aufbau ist wenig dokumentiert. Mit seinem Buch schließt der Autor diese Lücke.
Das Buch behandelt auf rund 200 Seiten ausführlich Vor- und Nachteile von Barcodes und Lesestift, gibt eine Übersicht über alle Barcodetypen, erläutert den grundsätzlichen Aufbau und das Kodierungsschema. Die Herstellung von Barcodes auf Druckern und dem Plotter mit vielen nützlichen Hinweisen für einwandfreie Ergebnisse und die dafür notwendigen oder wünschenswerten Geräte werden eingehend beschrieben. Barcodes von Alpha- und synthetischen Textzeilen, Druckersonderzeichen, überraschend einfache und schnelle Verfahren für 'load bytes' und noch leichtere Eingabe von synthetischen Befehlen an beliebiger Programmstelle – alles ohne Programmspeicher- oder Tastenbelegungen – geben rationelle Arbeitshilfen. Barcodes von Zahlen und Folgedaten für sequentielle Registereingabe werden ausführlich beschrieben. Der Aufbau von Programmbarcodes und deren Herstellung ohne das Plotter-Modul werden erschöpfend behandelt. Die Sofortausführung von beliebigen synthetischen Befehlen ohne Tastenbelegung wird gezeigt. Barcodes von Anweisungen, Befehlen, X-ROM- und anderen Funktionen sowie synthetische Zeilen zum Programmieren mit dem Lesestift (eGØBEEP und andere als Barcodes) und eine große Zahl von Tabellen der Funktions- und Alphazeichen 0 – 127 , aller Alphazeichen 0 – 255 (replace and append), für das synthetische Programmieren, für ein synthetisch simuliertes Tastenfeld, zur Daten-Masseneinlesung für Schriftfahnen und Querdruck, Kurzformexponenten und spezielle GTO- bzw. XEQ-Formen, sowie für alle Befehle der z.Zt. vefügbaren Module sind praktische Arbeitserleichterungen auch für den Anwender, der keinen Drucker besitzt.
Jedes Kapitel gibt teils längere, komfortable Programme als Listings und Barcodes an die Hand und es werden eine Vielzahl praktischer Anwendungen gezeigt. Nach Möglichkeit sind alle Programme zur Barcodeherstellung 3-fach vorhanden: 1. zur Herstellung nur mit Rechner und Drucker ohne Module; 2. zusätzlich mit dem XF-Modul und 3. außerdem mit dem Plotter-Modul. Hierdurch sind die Möglichkeiten der Barcodeherstellung weitestgehend auch dem Anwender erschlossen, der nicht über Zusatzmodule vefügt. In zwei weiteren Kapiteln werden andere, nicht HP-Barcodes, ihr Aufbau und deren Herstellung mit dem HP-IL-System beschrieben. Dem Aufbau und der Herstellung des Europäischen Artikel Nummern-Barcodes ist ein eigenes Kapitel gewidmet.
Wer die Möglichkeiten seines Lesestiftes optimal nutzen und selbst Barcodes anfertigen möchte, braucht dieses leicht verständlich geschriebene Buch. Es wird ihm bald zur unentbehrlichen Arbeitshilfe werden.
(1985, 38.00 DM, ISBN 3 – 88538 – 804 – 9)

J.S.Dearing: 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 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 informie-

ren wünscht. Man spart so manche Programmierstunde und kommt jedem merkwürdigen Verhalten des HP-41 auf die Spur.
(1984, 34.00 DM, ISBN 3-88538-801-4)

K.Jarett: 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. Dem deutschen Leserkreis wird dieser amerikanische Bestseller durch Heinz Dalkowski zugänglich gemacht, der sich schon als Übersetzer des 'Wickes' auszeichnete. 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 spezieller 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.
(1985, 38.00 DM, ISBN 3-88538-803-0)

K.Jarett: 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. Als Hilfsmittel benutzt er den "Byte-Schnapper", der leichter als der "Byte-Hüpfer" handzuhaben 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 den Anfänger ist es die ideale Einführung. Die Übersetzung besorgte in gewohnt fachmännischer Weise Heinz Dalkowski.
(1985, 40.00 DM, ISBN 3-88538-802-2).

W. Meschede: 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 synthetische Programmierung der Zeichen. Jedem Programm ist ein Programm-Ablauf-Plan 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, Mehrfunktionen- und hochauflösendes Plotten sowie Histogramme (Balkendiagramme) kein Problem mehr dar und durch kleine Änderungen ist die Anpassung an ganz spezielle Anforderungen leicht möglich. Hat man ein Programm länger nicht mehr benutzt, ermöglichen die Kurzanleitungen im Anhang ein schnel-

les 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 Barcodes abgedruckt, insgesamt 6755 Byte! Wer graphische Ausgaben oder mehr als nur die einfachen 127 Zeichen benötigt, kann an diesem Buch nicht vorübergehen.
(1985, 36.00 DM, ISBN 3-88538-805-7)

W. Stroinski (Hrsg.): 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 diese 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.
(1985, 36.00 DM, ISBN 3-88538-806-5)

W.C.Wickes: Synthetische Programmierung auf dem HP-41C/CV.
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 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 ROMs; 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 letzte Ausgabe 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.
Die Synthetische Programmierung wird – laut Autor – nun auch von Hewlett-Packard unterstützt und Programme, die synthetische Zeilen enthalten, werden von den Programm-Bibliotheken akzeptiert.
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" empfehlenswert macht, ist dieses Buch unerläßlich für jeden HP-41C/CV/CX Benutzer, der die Fähigkeiten und Möglichkeiten des Rechners voll ausschöpfen will.
(1983, 34.00 DM, ISBN 3–88538–800–6)

DIE HP-PALETTE DES HELDERMANN VERLAGES

Albers, K.: HP-41- und andere Barcodes mit dem HP-IL-System. Ca. 200 Seiten, ca. 38.00 DM, ISBN 3-88538-804-9 (1985).

Dearing, J.S.: Tricks, Tips und Routinen für Taschenrechner der Serie HP-41. Deutsche Ausgabe von Heinz Dalkowski. 220 Seiten, 34.00 DM, ISBN 3-88538-801-4 (1984).

Jarett, K.: Synthetisches Programmieren auf dem HP-41 – leicht gemacht. Deutsche Ausgabe von Heinz Dalkowski. 40.00 DM, ISBN 3-88538-802-2 (1985).

Jarett, K.: Erweiterte Funktionen des HP-41 – leicht gemacht. Deutsche Ausgabe von Heinz Dalkowski. 38.00 DM, ISBN 3-88538-803-0 (1985).

Meschede, W.: Plotten und Drucken auf dem HP-41C 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. 36.00 DM, ISBN 3-88538-806-5 (1985)

Wickes, W.C.: Synthetische Programmierung auf den HP-41C/CV. Deutsche Ausgabe von Heinz Dalkowski. 165 Seiten, 34.00 DM, ISBN 3-88538-800-6 (1983).

HP-41 Kombinierte Hex/Dezimale Byte Tabelle, 7 × 11.5 cm Plastikkarte, 6.00 DM (1983).

HP-41 Quick Reference Card, 7 × 15 cm Plastikkarte, 8.00 DM (1984).

Alle Produkte sind direkt vom Verlag erhältlich, die Plastikkarten nur auf diese Weise. Bitte richten Sie Ihre Bestellung an

Heldermann Verlag Berlin
Herderstr. 6-7
D-1000 Berlin 41

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. Als Hilfsmittel benutzt er den "Byte-Schnapper", der leichter als der "Byte-Hüpfer" handzuhaben 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 den Anfänger ist es die ideale Einführung. Die Übersetzung besorgte in gewohnt fachmännischer Weise Heinz Dalkowski.

ISBN 3 – 88538 – 802 – 2