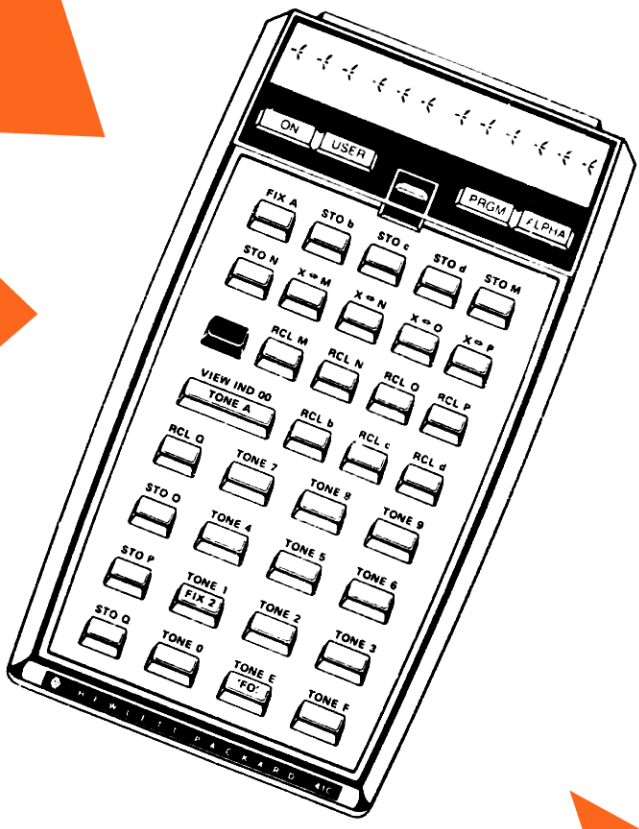
W. C. Wickes

Synthetische Programmierung auf dem HP-41C/CV

Deutsch von H. Dalkowski



Heldermann Verlag Berlin

W. C. Wickes

Synthetische Programmierung auf dem HP-41C/CV

Deutsch von H. Dalkowski

Fehlt denn nicht jedem was auf dem HP?
Dem dies, dem das, an Bytes fehlt es von je.
Vom Estrich zwar sind sie nicht aufzuraffen;
Doch Weisheit weiß die Tiefsten herzuschaffen.
In Tastenfolgen, Logikgründen
Sind Bytes gemünzt und ungemünzt zu finden,
und fragt ihr mich, wer sie zutage schafft:
Begabten Manns Natur- und Geisteskraft.

frei nach Faust II

W. C. Wickes
4517 NW Queens Avenue
Corvallis, OR 97330
U.S.A.

H. Dalkowski
Seidelbastweg 88a
D - 1000 Berlin 47

CIP - Kurztitelaufnahme der Deutschen Bibliothek

Wickes, William C.:

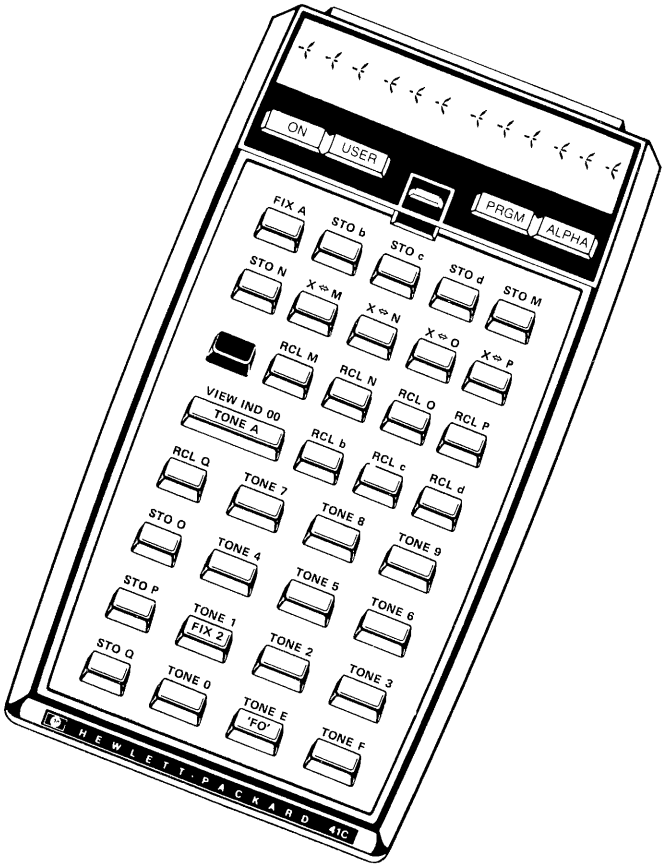
Synthetische Programmierung auf dem HP-41C CV / W.C.Wickes, Dt. von H. Dalkowski. - Berlin : Heldermann, 1982.
Einheitssacht.: Synthetic programming on the HP-41C CV <dt.>
ISBN 3-88538-800-6

ISBN 3-88538-800-6

W. C. Wickes

Synthetische Programmierung auf dem HP-41C/CV

Deutsch von H. Dalkowski



Heldermann Verlag Berlin

INHALTSVERZEICHNIS

Abbildungen

Tabellen

Programmverzeichnis

Vorwort

Es ist mir eine Freude, all denen zu danken, die mir mit ihren entscheidenden Beiträgen bei der Vorbereitung dieses Buches geholfen haben. Der Umschlagentwurf stammt von William Kolb, der auch das Manuskript vollständig redigierte und das ganze Projekt mit Zuspruch und Unterstützung begleitete. Tom Cadwallader, Keith Jarett und John McGechie, jeder ein bedeutender Wegbereiter der synthetischen Programmierung, lieferten, zusammen mit Tom James und Lee Vogel, wertvolle Ergänzungen und Anregungen. Charles Close gab wichtige Beiträge zu unserer genauen Kenntnis der Programmzeilen-Kodierung und anderer raffinierten HP-41C-Eigenschaften. Die Entdekkung des erweiterten Byte-Hüpfens durch Roger Hill war ein wesentlicher Schritt in Richtung auf ein 'freundlich gesinntes' synthetisches Programmieren. Jacob Schwartz erarbeitete die Barcode Zeichen-Tabelle im Anhang 3. Richard Nelson förderte durch seine vortrefflichen Leistungen in Form der Gründung und Unterhaltung des PPC die weltweite Verbindung, die für eine gemeinschaftliche Entwicklung der synthetischen Programmierung unentbehrlich ist. Professor Carroll Alley von der Universität Maryland ist der Hauptgefährte meiner Bemühungen um die synthetische Programmierung gewesen, geradewegs vom Eintreffen unserer jungfräulichen und noch mit Unvollkommenheiten behafteten HP-41C's an.

Das Buch ist meiner Frau Susan, der wohl duldsamsten Rechner-Witwe der Welt, gewidmet. Sie befaßte sich mit einem großen Teil der Schreib- und Korrekturarbeiten und bewahrte mich vor dem Verhungern, wenn ich meine vertraulichen Gespräche mit dem HP-41C und der Schreibmaschine hatte. Auch meinen Kindern Kenny und Lara, die mir, wann immer ich es wünschte, 'ihren' HP-41C zu benutzen erlaubten, bin ich dankbar.

Synthetische Programmierung auf dem HP-41CV (Addendum zur fünften Auflage)

Alle Funktionen und Techniken der synthetischen Programmierung, die in diesem Buch beschrieben werden, können auf dem HP-41CV , der erst nach der ersten Drucklegung dieses Buches auf den Markt kam, in gleicher Weise ausgeführt werden. Nur der Trick des 'Modul-Entfernens', der in Kapitel 3 benutzt wird, um den 'Byte-Hüpfer' zu erlangen, ist auf dem 41CV nicht möglich. Das folgende Verfahren kann (auf dem 41C übrigens ebensogut) stattdessen eingesetzt werden:

1. Schritte 2 und 4 auf Seite 37 unten bzw. Seite 38 oben ausführen.
2. PRGM-Modus an; Zeile 01 LBL "ABC" erzeugen.
3. CAT 1 (noch im PGRM-Modus) aufrufen und mit R/S sofort abstoppen, so daß 01 LBL "ABC" in der Anzeige erscheint. XEQ ALPHA "DEL" ALPHA 001 ausführen (es erscheint kurz '4094' und dann das .END.). BST tasten, was zu '4093 DEC' führt. Abermals BST , um '4092 X<>06' zu erreichen (Geduld!). Dies ist dieselbe Zeile wie die in Schritt 6 von Seite 38 beschriebene Zeile 06.
4. Man fahre nun mit den Anweisungen 7 bis 10 von Seite 38/39 fort, wobei zu beachten ist, daß die in Schritt 7 zu löschenden Zeilen jetzt '4089' und '4088' lauten. Ebenso fehlt LBL 01 aus Schritt 9.

Dieser Trick beruht auf einer Unvollkommenheit des Betriebssystems der beiden Rechner, die in zukünftigen Versionen beseitigt sein könnte. Für diesen Fall sollten sich 41CV-Besitzer die Benutzung eines Barcode-Lesers sichern, um wenigstens das Tastenzuweisungsprogramm "KA" aus Kapitel 5, mit dem man die synthetische Programmierung beginnen kann, einzulesen. - Auf die einleitende Vorführung des 'Gruselzwerges' auf Seite 2 müssen die Besitzer eines HP-41CV ganz verzichten.

GAU's UND ANDERE KATASTROPHEN

HP-41C 'GAU's (größte anzunehmende Unfälle) sind die Fälle, in denen die Anzeige einfriert oder erlischt. Die synthetische Programmierung birgt zwar solche Risiken. Kein Programm jedoch und keine Technik, die in diesem Buch beschrieben werden, verursachen eine Katastrophe, wenn man nur exakt den Anweisungen folgt - lediglich zufällige Fehler sind unvermeidlich. Wenn es dennoch einmal auf Ihrem Rechner zu einem GAU gekommen sein sollte - keine Sorge, es wird ihm kein Leid geschehen. Versuchen Sie zunächst die Batterie zu entfernen und dann wiedereinzusetzen. Wenn das fehlschlägt (versuchen Sie es einige Male), entfernen Sie alle Peripheriegeräte und eingesteckten Module, bevor Sie die Batterie entfernen. Als letzte Zuflucht (ich selbst mußte nie soweit gehen) entfernen Sie die Batterie eine ganze Nacht.

Kapitel 1

WARUM UND WOZU

"Es gibt mehr Ding' im himmlischen Vier Eins,
Als Eure Schulweisheit sich träumt, Hewpackio."
- W. Shakespeare möge verzeihen

1A. Synthetisches Programmieren?

Es gibt niemanden, vom ernsthaften Studenten der Informatik bis zum gelegentlichen Benutzer eines 4-Spezies-Rechners, den der HP-41C nicht beeindrucken muß. Dieser Apparat vereint höchst erstaunliche Rechenkraft mit dem Vorteil uneingeschränkter Beweglichkeit. Der angehende Käufer wird von einer langen Liste fest eingebauter Rechenfunktionen gefesselt; der erfahrene Besitzer entdeckt, daß der HP-41C eine immer größere Rolle beim Lösen seiner Probleme spielt, wenn er das Programmieren beherrscht und eigene Erfindungsgabe und eingebaute Funktionen zusammenwirken.

Und dennoch, selbst dann, wenn er alles gelernt hat, was das *Bedienungshandbuch* ihn lehren kann, erwartet ihn weiterer Hochgenuß: die Liste der HP-41C-Funktionen und Programmierfähigkeiten wird *nicht* durch die Eigenschaften begrenzt, die das *Handbuch* aufzählt. Tatsächlich existiert eine ganze Klasse von Funktionen und Programmiermöglichkeiten, die dazu benutzt werden kann, die Leistung des Rechners erheblich zu steigern, obwohl sich die neuen Funktionen zunächst nicht auf herkömmliche Weise über das Tastenfeld ausführen oder programmieren lassen. Die neuen Funktionen, die durch Erzeugung neuer Kombinationen normaler Programm-Bytes 'synthetisiert' werden, heißen 'synthetische Funktionen'; ihre Anwendung in Programmen gab Anlaß zu dem Begriff 'synthetisches Programmieren' und somit auch zum Titel dieses Buches.

Um den Appetit des Lesers zu wecken, hier eine Auswahl einiger typischen Anwendungen synthetischer Programmierung, die ohne die hier beschriebenen Techniken unmöglich oder unzweckmäßig sind:

*** Erzeugung 21 'neuer' Zeichen für den alltäglichen Gebrauch.

*** Umwandlung des Alpha-Registers in vier zusätzliche Daten-Register. Diese Register können während eines Programmlaufs als 'Notizblock' benutzt werden, um Daten, die von anderen Programmen in numerierten Daten-Registern gespeichert wurden, unversehrt zu lassen. Fernerhin kann der Inhalt dieser Register mit der Kartenleser-Funktion 'WSTS' ein- und ausgelesen werden.

*** Erweiterte Benutzer-Kontrolle über die 56 Benutzer- und Systemflags. Beispiel: zweimal tasten kann alle 56 Flags gleichzeitig löschen.

*** Automatisches 'SIZE-Suchen' in weniger als 2 Sekunden.

*** Schnelles Sortieren von Alpha-Daten.

*** Alphanumerische Zeichenketten-Verarbeitung.

*** Hinzufügung 6 neuer Tonfrequenzen und Variation der Tondauer.

*** Austausch von Programmzeilen mit gespeicherten Daten.

*** Verbesserte Kontrolle der Tastenzuweisung; hierin eingeschlossen die Zuweisung von 2-Byte-Funktionen (z.B. 'STO 65' einer Taste zugewiesen), automatisches Löschen aller Zuweisungen und Verdichten der Zuweisungsregister.

Eine einfache Übung soll Sie in die Welt der synthetischen Programmierung einführen und vielleicht dazu anregen, die Mühe aufzuwenden, den Rest des Buches zu lesen. Versuchen Sie den folgenden Hokuspokus:

1. Memory-Modul in den HP-41C einstecken.
2. Totallöschung ausführen.
3. SIZE 063 (bei Doppelmodul SIZE 127) festsetzen.
4. PRGM-Modus an.
5. Folgende Programmzeilen eintasten:

```
01  12345
02  STO IND 17
03  RDN
```

6. HP-41C ausschalten.
7. Memory-Modul entfernen; 60 Sekunden warten; Modul wiedereinsetzen.
8. HP-41C einschalten.
9. RTN tasten.
10. '1.435245455 EEX 59' eingeben.
11. SST tasten.
12. ALPHA an.

Woher kommt der 'Gruselzwerg'? Wenn Sie in den PRGM-Modus schalten und einmal BST tasten, werden Sie die Programmzeile '01 STO M' sehen. Diese 'synthetische' Programmzeile ist eine Kombination der Programm-Bytes 'IND 17' und 'RDN', die dadurch entstanden ist, daß Sie das 'STO' aus 'STO IND 17' durch Ziehen des Memory-Moduls entfernt haben. Im *Bedienungshandbuch* gibt es keinen Hinweis auf die Existenz der Funktion 'STO M', aber Sie werden diese Funktion und ihre Gefährten kennen und schätzen lernen, sobald Sie die synthetische Programmierung beherrschen.

1B. Zweck und Aufbau des Buches

Dieses Buch ist dazu bestimmt, einem jeden HP-41C-Benutzer, vom Novizen bis zum Experten, das Vergnügen synthetischer Programmierung zu vermitteln und ihn in ihre Mysterien einzuweihen. Es bildet eine Zusammenstellung der Verhaltensweisen des Rechners, die die synthetische Programmierung ermöglichen, ein Handbuch der Verfahren für die Erzeugung synthetischer Programmzeilen und eine Sammlung von Anwenderprogrammen, die sowohl praktischen Zwecken dient als auch die Benutzung exotischer Programmtechniken vorführt.

Kapitel Zwei beschreibt die interne Arbeitsweise des HP-41C aus einer begrifflichen Sicht, die einen Computer-Ingenieur vermutlich veranlassen wird, in die Knie zu gehen. Sie werden dort eine tiefere Einsicht in die Wirkungsweise der Rechner-Programmierung erlangen, als es durch das *Bedienungshandbuch* allein möglich ist. Bei der Einführung in die Grundlagen der synthetischen Programmierung in Kapitel Zwei werden Sie überdies ein Bild vom Betriebssystem des HP-41C erhalten, das Ihnen helfen wird, Ihre Programmierfähigkeit bestens zu entwickeln.

Kapitel Drei stellt die erste und wichtigste der synthetischen Funktionen, den 'Byte-Hüpfer', vor. Diese einzelne Tastenfunktion stößt die Tür zu einfachen Verfahren für die Erzeugung sämtlicher synthetischen Funktionen auf. Wir werden den Byte-Hüpfer 'erzeugen', indem wir den Trick des 'Modul-Entfernens', wie wir ihn zur Erlangung von 'STO M' benutzten, anwenden; ist dies einmal getan, werden wir uns nie wieder auf das Entfernen des Moduls zurückziehen müssen.

In Kapitel Vier wird eine neue Gruppe von HP-41C-Registern, die 'Zustandsregister', eingeführt. Der Zugriff auf diese Register, die das Alpha-Register, die 56 Flags, die Information über die Speicherzuweisung, den Programm-Adreßzeiger und den Unterprogramm-Rücksprungstapel einschließen, hat eine Unzahl von praktischen Anwendungen, ähnlich den Beispielen in Abschnitt 1A, zur Folge.

'Programme zu programmieren', ein Paket von HP-41C-Programmen und -Techniken, wird in Kapitel Fünf beschrieben. Der Hauptnutzen dieser Programme liegt in der Möglichkeit, andere Programme zu schreiben und zu entziffern.

Kapitel Sechs ist ein Kapitel von 'Standard Anwendungen', in welchem wir eine Anzahl von synthetischen Programmen antreffen, die an und für sich das Studium der vorangehenden Kapitel hinreichend rechtfertigen. Doch veranschaulichen diese Programme darüberhinaus allgemeine Techniken der synthetischen Programmierung, die sich auf ein weites Feld von Problemen, nur begrenzt durch Antrieb und Findigkeit des eingeweihten Lesers, anwenden lassen.

Schließlich lernen wir in Kapitel Sieben ein paar amüsante Kunstgriffe aus der 'Trickkiste des Gewerbes' kennen, die zwar nicht besonders praktisch sind aber das Herz des befähigten Rechner-Narren erfreuen werden. Eingeschlossen ist selbstverständlich ein übertriebenes Beispiel dafür, wie man enormen Forscherfleiß aufwenden

kann, um ein völlig nutzloses Ergebnis zu erzielen, nämlich: wie sich diese störrische Graugans rückwärts scheuchen läßt.

Es folgen Anhang 1 bis 3. Anhang 1 gibt einen kurzen Überblick über das dezimale, das binäre, das oktale und das hexadezimale Zahlensystem. Wenn Ihnen diese Notationen unvertraut oder Ihre Kenntnisse darüber vielleicht ein wenig eingerostet sind, dürfte es nützlich sein, Anhang 1 zu studieren, bevor Sie Kapitel Zwei in Angriff nehmen. Anhang 2 enthält den Barcode für die wichtigen Programme von Kapitel Fünf, "CODE", "REG", "KA" und "DECODE" sowie das umfangreiche 'Henkersprogramm' von Kapitel Sechs. Anhang 3 enthält spezielle Barcodes für bestimmte Zwecke der synthetischen Programmierung.

1C. Der Ursprung der synthetischen Programmierung

Es begann alles mit einem Zufall! Frühe Modelle des HP-41C hatten eine unbeabsichtigte Fehlerquelle, eine 'Unreinheit', in ihrer internen Verschlüsselung, die es z.B. erlaubte, den Befehl 'STO IND 01' für Werte von 719 bis 999 im Daten-Register R_{01} auszuführen. Dieser Befehl verursachte die Übertragung des Inhalts von Register X in den *Programm*-Speicher. Ich wollte wissen, was wohl geschähe, wenn ich diese Eigenschaft dazu benutzen würde, neue Programmzeilen durch solchermaßen ins Programm übertragene Zahlen zusammenzusetzen, daß sich Verknüpfungen von normalerweise unerreichbaren Kombinationen von Programm-Bytes ergäben. Um es kurz zu machen: es funktionierte. Nachdem die neuen Funktionen einmal im Speicher, aus dem sie auf Magnetkarten gelesen und mit jedem Programm verschmolzen werden konnten, aufgetaucht waren, ergaben sich praktische Anwendungen in Hülle und Fülle.

Nach der Entdeckung der synthetischen Funktionen waren die wichtigsten Fortschritte der synthetischen Programmierung:

1) die Entwicklung synthetischer Tastenzuweisungen, die den einfachen Aufruf der neuen Funktionen über die Tastatur ermöglichte und so die Notwendigkeit eines Hardware-Fehlers beseitigte;
2) die Entdeckung des Byte-Hüpfers, der wohl *die* grundlegende der synthetischen Funktionen ist. Weil der Byte-Hüpfer auf jedem HP-41C erzeugt werden kann, und weil er fast jede andere synthetische Programmzeile hervorzubringen gestattet, können wir in diesem Buch am 'Nullpunkt' beginnen und zeigen, wie sich ein HP-41C im 'Rucksackverfahren' auf volle synthetische Programmierfähigkeit hochfahren läßt.

1D. Keine Gefahr für den HP-41C

Synthetische Funktionen sind, wenn sie mit den in diesem Buch beschriebenen Verfahren in den HP-41C befördert werden, 'echte' Rechner-Operationen. Als solche stellen sie keine leibliche Bedrohung des HP-41C dar. Das einzige Risiko, welches

eigentlich nur als möglicher Verdruß denn als Gefahr betrachtet werden sollte, ist die Aussicht, durch gewisse Operationen mit synthetischen Funktionen entweder ein "MEMORY LOST" zu verursachen oder einen 'GAU', also jenen Zustand, in dem die Anzeige einfriert oder erlischt und das Tastenfeld kampfunfähig wird. Das erste Mißgeschick ruft heftiges Zähneknirschen und Händeringen hervor, krümmt dem HP-41C aber gewiß kein Haar. Das zweite Problem kann im wesentlichen (jedenfalls in 99,9% der Fälle) durch einfaches Entfernen und unverzügliches Wiedereinsetzen der Batterie, gefolgt von ein oder zwei Ein/Ausschaltversuchen, gelöst werden. Mir ist nur ein einziger Fall zu Ohren gekommen, bei welchem die Katastrophe so groß war, daß für die Wiederbelebung des Rechners die Batterie eine ganze Nacht über entnommen werden mußte; die Ursache jenes Betriebsunfalles ist jedoch unbekannt. Ich kann natürlich nichts garantieren, aber im Verlauf der Entwicklung der synthetischen Programmierung habe ich buchstäblich Dutzende von Malen den Speicher versehentlich gelöscht oder meinen Rechner in einen GAU geführt, trotzdem tickt der HP-41C unbeeindruckt weiter.

Wegen des Risikos einer unbeabsichtigten Speicherlöschung ist es jedoch für Hewlett-Packard selbst unzweckmäßig, die Benutzung synthetischer Funktionen zu 'unterstützen'. Daher sollten Sie Programme, die synthetische Zeilen enthalten, nicht an die Benutzerbibliothek (User's Library) senden.

1E. Einige Verabredungen

Die folgende Liste besteht aus besonderen Kennzeichnungen, die ich für dieses Buch festgelegt habe, um die Beschreibung von Rechner-Programmen, -Zeichen, -Zahlen, -Befehlen usw. zu vereinfachen.

1. Sie werden sicher schon den Gebrauch des Apostrophs statt der gewöhnlich benutzten Anführungszeichen bemerkt haben, also 'Beispiel' statt "Beispiel". Anführungszeichen habe ich dafür freigehalten, auf alphanumerische Zeichen und Textzeilen des HP-41C hinzuweisen, so wie etwa der Drucker Zeichen in Programmauflistungen ausweist. Auf diese Weise sollen Sie sich dessen bewußt werden, daß einer Programmzeile, die in Anführungszeichen eingeschlossen ist, in der HP-41C-Anzeige das Text-Symbol "ᵀ" vorangeht.

2. Um einen sauberen Druck zu gewährleisten, werden, wenn irgend möglich, Standardtypen der Schreibmaschine benutzt, um die Anzeige des HP-41C und die Druckerzeichen darzustellen. Die Identifizierung ist in der Regel gesichert, mit einer eventuellen Ausnahme, dem Gebrauch des Semikolons ";" , um das HP-41C-Symbol "ᵀ" darzustellen.

3. *Zahlen* in der Anzeige werden in der Form dargestellt, die sie annehmen, wenn sie mit 'ARCL' ins Alpha-Register gerufen werden. Zahlen im SCI- oder ENG-Format werden daher unter Benutzung von 'E' für den Exponenten abgebildet, also, um die möglicherweise verwirrenden Leerstellen auszuschalten, '1.23 E10' statt eigentlich '1.23 10'.

4. Einträge wie 'STO mn', 'DEL ℓmn' oder 'SF mn' werden als symbolische Operation mit der aufgeführten Funktion verstanden, wobei die Buchstaben ℓ, m, n usw. Ziffern bedeuten, die jeden mit dieser Funktion verbundenen normalen Wert annehmen können. Demgemäß können 'm' und 'n' in 'STO mn' jeden der Werte von 0 bis 9 annehmen.

5. Beim Auflisten langer hexadezimaler oder binärer Zahlen ist es oftmals günstig, die Ziffern für Erläuterungszwecke zu gruppieren. Diese Gruppierung wird durch Leerstellen oder in die Zahlen eingefügte Striche '|' vorgenommen, die natürlich nicht in der tatsächlichen Zahlenkodierung des HP-41C enthalten sind.

6. Ein Baustein des HP-41C-Arbeitsspeichers ist das 'Register', ein Block von sieben Zellen, in welchem entweder eine Zahl oder sieben Programm-Bytes abgelegt werden können. Register lassen sich wie folgt kennzeichnen.
   a) 'Register ℓmn' bezeichnet das Register, welches man unter der Speicheradresse 'ℓmn' (vgl. Abschnitt 2C) findet, wobei 'ℓmn' eine 3-ziffrige Hexadezimalzahl ist.
   b) 'Register α' , α ein Alpha-Zeichen, verweist auf eines der 16 'Zustandsregister', die in Kapitel 4 besprochen werden. Darunter befinden sich die bekannten RPN Stapelregister X, Y, Z, T und L . Die verbleibenden 11 Register werden mit M, N, O, P, Q, ⊢ , a, b, c, d und e bezeichnet, weil die entsprechenden synthetischen 'Zustandsregister-Zugriffsfunktionen' mit diesen Symbolen in der Anzeige erscheinen, z.B. 'STO M', 'RCL ⊢ ', 'ISG d' usw.
   c) 'R_{mn}' bedeutet *Daten*-Register Nr. 'mn', wobei 'mn' eine zwei- (gelegentlich auch 3)-ziffrige Dezimalzahl ist.

7. Programme werden in diesem Buch unmittelbar nach den Vorlagen, die der Drucker 82143 liefert, reproduziert, um Übertragungsfehler zu vermeiden. Leider gibt der Drucker die Zustandsregister-Zugriffsfunktionen für die Register M, N, O, P, Q und ⊢ mit anderen Symbolen wieder, als die Anzeige des HP-41C. Der Leser muß sich daher mit der folgenden Tabelle vertraut machen:

Tabelle 1 - 1

Symbole für Zustandsregister

Anzeige	Drucker
M	[
N	\
O	]
P	↑
Q	_
⊢	⁻

Überdies benutzt der Drucker in synthetischen Textzeilen das Symbol "♦" sowohl für das 'Null'-Byte '00' (welches in der HP-41C-Anzeige als "-" erscheint) als auch für das Byte '0A'. Da es in diesem Buch aber keine Textzeilen gibt, die das Byte '0A' enthalten, zeigt das Symbol "♦" stets das Byte '00' an.

8. Kurze Programme (Routinen) ohne Alpha-Marke werden durch eine rechterhand in Klammern gesetzte Kennzahl bezeichnet. Das Format dieser Kennzahl lautet '(Nr. des Abschnitts - Nr. der Routine)'.

9. Um das Eintasten der Befehle für einzelne Programmzeilen oder lauffähige Programme übersichtlich zu gestalten, werden die Anweisungen in den meisten Fällen in drei Spalten aufgegliedert. Die Einträge in der *linken* Spalte sind Kodes, die in den Speicher zu bringen sind: entweder Zahlen, die ins X-Register sollen, Alpha-Zeichen (eingeschlossen in Anführungszeichen), die ins Alpha-Register gehören, oder Programmzeilen (aufgeführt mit Programmzeilen-Nummer), die ins gegenwärtige Programm eingetastet werden müssen. Die *mittlere* Spalte enthält Tastenfolgen, die nicht ins Programm gelangen, wie z.B. 'GTO .123' oder 'DEL 005'. Die *rechte* Spalte zeigt, sofern zweckmäßig, die aus dem jeweiligen 'Mittelspalten'-Befehl entstehende HP-41C-Anzeige. Diese Anzeige wird in eckige Klammern [ ] eingeschlossen. Beispiel:

(eintasten)	(Operation)	(Anzeige)
	GTO .000	[00 REG 123]
01 STO 01		
	SST	[02 X<>Y]

weist Sie an, 'GTO .000' durchzuführen (um '00 REG 123' zu erhalten), Zeile '01 STO 01' einzutasten, dann einmal 'SST' auszuführen, um daraufhin Zeile '02 X<>Y' wahrzunehmen.

1F. Voraussetzungen

Um eine möglichst große Anzahl von HP-41C-Benutzern zu erreichen, ist dieses Buch für eine Minimalausstattung des HP-41C geschrieben. Erforderlich sind nur (1) ein HP-41C, (2) der zeitweilige Gebrauch eines Speichermoduls und (3) ein paar Stunden Zeit*), um den Stoff durchzuarbeiten. Kartenleser, Drucker und optischer Lesestift sind nicht notwendig, obwohl sie, wie auch bei jeder normalen Benutzung des HP-41C, eine wertvolle Unterstützung der synthetischen Programmierung bilden. Der Kartenleser z.B. bietet eine bequeme Möglichkeit, Ihre neuen Programme und Tastenzuweisungen gegen zufällige Speicherlöschung zu sichern. Der Drucker läßt sich sehr nützlich dazu verwenden, die Programme so, wie sie eingetastet wurden, aufzulisten und ein Ablaufprotokoll über jeden Schritt zu erstellen. Wenn Sie in der

*) ein paar Tage! (der Übersetzer)

glücklichen Lage sind, sogar Zugang zu einem Lesestift zu haben, können Sie eine Menge Tastenarbeit einsparen, indem Sie einfach die Barcodes aus Anhang 2 und 3 einlesen.

Kapitel 2 ist die schwerste 'Hürde' für Anfänger der synthetischen Programmierung, weil es eine Fülle ins Einzelne gehender Beschreibungen enthält, ohne den Spaß der Rechnerbedienung zu bieten. Ich schlage vor, daß Sie Kapitel 2 das erste Mal verhältnismäßig schnell durchlesen, gerade sorgfältig genug, um seinen Inhalt soweit zu beherrschen, daß Sie für die in Kapitel 3 beginnende fesselndere Tastenbetätigung vorbereitet sind. Wenn Sie dann durch die späteren Kapitel kommen, können Sie in den verschiedenen Abschnitten von Kapitel 2 weitere Einzelheiten nachschlagen.

1G. Hinweise

Die meisten der in diesem Buch beschriebenen Entdeckungen und Techniken sind zuerst in verschiedenen Heften des *PPC Calculator Journal* veröffentlicht worden. Das PPC (die Anfangsbuchstaben haben keine besondere Bedeutung) ist eine unabhängige, weltweite Verbindung von Rechner-Enthusiasten mit dem gemeinsamen Eifer, die programmierbaren Rechner von Hewlett-Packard kennen und gebrauchen zu lernen. Die Zeitschrift des PPC ist der Hauptträger des Informationsaustausches unter den über die ganze Welt verstreuten Wegbereitern der synthetischen Programmierung. Jeder entschlossene HP-41C-Benutzer wird, wenn er daran denkt, die Kenntnisse, die er in diesem Buch erwirbt, zu erweitern, feststellen, daß es sich auszahlt, dem Klub beizutreten und die *Zeitschrift* zu abonnieren. Verbindung mit mehreren tausend anderen Programmierern kann sehr viel Arbeit sparen. Anfragen sind zu richten an:

PPC - SP
2541 W. Camden Place
Santa Ana, CA 92704
U.S.A.

Hier eine Liste von Artikeln, die für die Entwicklung der synthetischen Programmierung bedeutsam waren (die Darstellungsform ist Band (V), Heft (N), Seite (P)):

Cadwallader, T. 'Improved Synthetic Key Assignments' V7N3P3
Close, C. 'Bug 2: A Practical Application' V7N3P8
Hewlett-Packard ('Corvallis Column') 'HP-41C Function Table' V6N4P11
---------- 'HP-41C Postfix Table' V6N5P11
---------- 'HP-41C Data & Program Structure' V6N6P19
Istok, G. 'Pseudo XROM's on the HP-41C' V7N2P32
Kennedy, J. 'The HP-41C Combined Hex Table' V6N5P27
McGechie, J. 'HP-41C Synthetic Key Assignments' V7N2P34
Nelson, R. 'Bugs in the Box' V6N5P27
Wickes, W. 'Direct Status Register Access on the HP-41C' V6N7P31
--------- 'Through the HP-41C with Gun and Camera' V6N8P27
--------- 'HP-41C Black Box Programs' V6N8P29
--------- 'Freedom From Bugs' V7N2P35
--------- 'Synthetic Key Assignments' V7N2P30
--------- 'Improved Black Box Programs' V7N2P35
--------- 'HP-41C Synthetic Function Routines' V7N4P26

---------- 'Byte-Jumping, or The Poor Man's Black Box' V7N4P26
---------- 'Direct Addressing of ROM Routines' V7N5P55
---------- 'Understanding BLDSPEC' V7N5P56

Es muß betont werden, daß die Entwicklung der synthetischen Programmierung andauert. Auch während dieses Buch geschrieben wurde, fanden wißbegierige Programmierer neue Tricks heraus und erweiterten unser Verständnis des HP-41C. Selbst das Schreiben dieses Buches vergrößerte meine eigene Einsicht in den Gegenstand und führte zu einer Reihe neuer Entdeckungen wie z.B. dem in Kapitel 5 beschriebenen Textgehilfen.

Kapitel 2

Im Innern des HP-41C

Dieses Kapitel wird gewissermaßen ein Ausflug nach Phantásien. Um Ihnen eine gebrauchsfähige Vorstellung von der Arbeitsweise des HP-41C zu vermitteln, werde ich ausgedachte, fast personifizierte Systeme, die die Tätigkeit des Rechners verkörpern, einführen. Diese Systeme können - müssen aber nicht - genaue elektronische Spiegelbilder im HP-41C-Gehäuse haben; Einzelheiten dieser Art sind nur für Elektronik-Ingenieure von Belang und fallen aus dem Rahmen dieses Buches. Entscheidend ist, daß der HP-41C sich so verhält, als wären diese Systeme vorhanden. Zuerst und vor allem müssen Sie sich das 'Gehirn' des Rechners als eine Einrichtung vorstellen, die wir 'Prozessor' nennen wollen. Dieser Prozessor ist für das Einlesen von Daten und Programmen in den Speicher verantwortlich und weist die anderen Systeme im Rechner an, das Gelesene auf bestimmte Weise zu verarbeiten. Gestützt auf Ihre Einbildungskraft können Sie sich den Prozessor als einen Aufseher vorstellen, der eine Reihe von Befehlen des Benutzers entgegennimmt und dann dem 'Personal', welches das System des HP-41C bildet, seine eigenen Anweisungen erteilt.

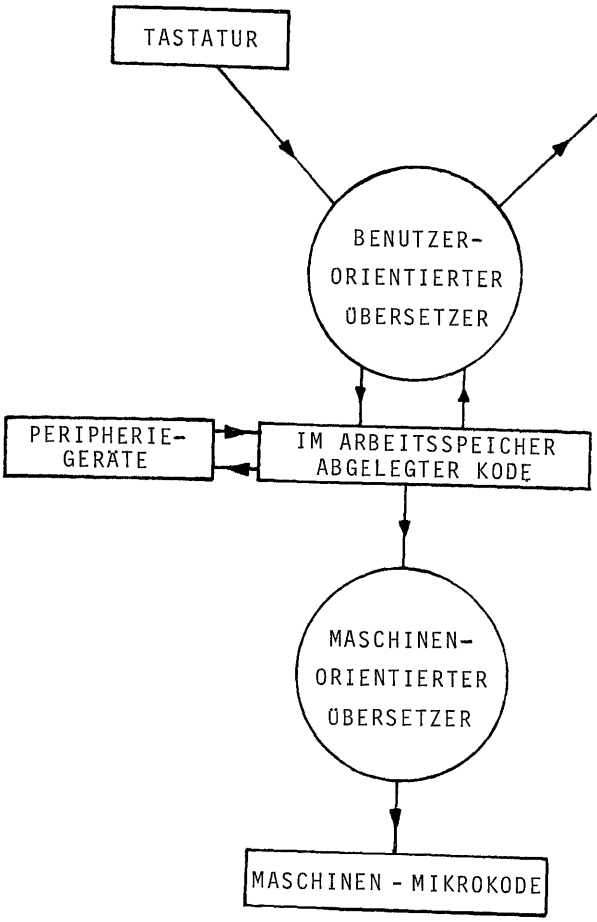
2A. Die Sprache des Rechners: Bits, Nybbles und Bytes

Ein Rätsel: Was haben die Zahl '1.435245455 E59', die Zeichenkette "⌶CREEPY" und das Programm

```
01 LBL 00
02 /
03 SQRT
04 X>Y?
05 X>Y?
06 LN
07 SIN
```

gemeinsam? Antwort: Alle drei werden auf gleiche Weise im Arbeitsspeicher des HP-41C abgelegt. Diesen offensichtlich zweifelhaften Vorgang verstehen, heißt, die Grundlage der gesamten Arbeitsspeicher-Gliederung und -Kodierung begreifen. Mit 'Arbeitsspeicher' meinen wir den Teil des HP-41C-Speichers, der unter der Kontrolle des Benutzers steht: die Daten- und Programm-Register, die Tastenzuweisungsregister, die RPN Stapelregister, das Alpha-Register usw.

Auf der untersten Stufe ist ein Rechner wirklich ein sehr einfaches Gerät. Er kann Zahlen speichern und zurückgeben, sie addieren, wenn's gewünscht wird, aber das ist auch schon alles. Um Befehle auszuführen, die dem Benutzer sehr einfach erscheinen, wie z.B. '+' oder 'LN', muß der Prozessor Folgen von Dutzenden elemen-




```
113 AVIEW
114 STO 05
115 GTO 99
116 *
117 STOP
```

STUFE 1: BENUTZER-EIN-
UND -AUSGABE

```
F4 21 45 5A 54 F6 49
50 53 20 54 53 F8 49
20 53 45 4B 43 49 57
```

STUFE 2: ARBEITSSPEICHER

```
...1..11.1
..1111.111
11.....11.
.....11111
```

STUFE 3: MASCHINEN - MIKROKODE

ABBILDUNG 2-1. DIE DREI STUFEN DER KODIERUNG IM HP-41C

tarer Schritte veranlassen. Die eigentliche Stärke eines Rechners liegt in seiner Fähigkeit, dem Benutzer die Auslösung dieses inneren Vorganges durch einfache Tastenfolgen zu ermöglichen. Ein programmierbarer Rechner kann zusätzlich die Tastenfolgen verschlüsseln und den Kode für wiederholte automatische Ausführung abspeichern.

Der HP-41C stellt gegenüber früheren Taschenrechnern insofern einen wichtigen Fortschritt dar, als Programm-Kodes dem Benutzer in unmittelbar lesbaren alphanumerischen Zeichen und Anweisungen angezeigt werden. Wir dürfen uns einen im HP-41C sitzenden unsichtbaren 'Übersetzer' vorstellen, der die gespeicherten Kodes abschnittsweise in anzeigefähige Zahlen oder Befehle umformt. Außerdem muß es noch einen zweiten Übersetzer geben, der dem Rechner die - 'Mikrokode' genannte - passende Folge von Elementarschritten zur Ausführung übergeben kann. In der Tat gibt es die in Zeichnung 2-1 veranschaulichten drei Stufen der 'Wiedergabe' desselben Kodes.

Auf 'Stufe 1' werden die Kodes in einer für den Benutzer direkt lesbaren Form zurückgereicht: Die Benutzer-Eingabe besteht aus einer durch Tastenbedienung bestehenden Kode-Erzeugung; die so gebildeten Kodes werden durch den 'benutzerorientierten Übersetzer' als Zahlen, Zeichen oder Programmzeilen in die Anzeige zurückgeschrieben. Auf 'Stufe 2' befinden sie sich in dem Zustand, der es erlaubt, sie im Speicher abzulegen oder auf Peripheriegeräte wie Kartenleser und Lesestift weiterzuleiten bzw. von dort zu lesen. Schließlich ist ein 'maschinenorientierter Übersetzer' dazu erforderlich, die Kodes in 'Stufe 3' zu bringen, also in einen Satz elementarer Maschinenbefehle, die zur Ausführung einer Operation gebraucht werden, zu verwandeln.

Wir sind hauptsächlich an Stufe 1 und 2 interessiert. 'Synthetische Programmierung' besteht darin, neue Kodes auf Stufe 2 zu erzeugen, indem man die Tastenfeldlogik, die den Benutzer auf die Eingabe des im *Handbuch* beschriebenen Befehlssatzes beschränkt, umgeht. Die entstehenden synthetischen Kodes können dann von beiden Übersetzern bearbeitet werden und liefern häufig praktische Ergebnisse wie z.B. neue Zeichen für die Anzeige und bis dahin unbekannte Programmfunktionen. Um das zustandezubringen, muß der Benutzer die 'Sprache' der Stufe 2 erlernen, so daß er unabhängig vom benutzerorientierten Übersetzer mit den gespeicherten Kodes umgehen kann.

Die Forderung, Kodes im HP-41C-Speicher ablegen zu können und für Peripheriegeräte lesbar und schreibfähig zu gestalten, legt die allgemeine Bauart, die der Kode annehmen muß, fest. Der Kartenleser z.B. verwendet für die Kode-Ablage Magnetkarten; die Karte selbst kann Information nur in Form geordneter Magnetfeld-Bereiche in ihrer Oxyd-Schicht speichern. Um eine reproduktionsfähige, zuverlässige Speicherung zu erreichen, muß die Ordnung so einfach wie möglich sein: Der Kode wird durch eine Reihe von magnetisierten und nicht-magnetisierten strichförmigen Bezirken dargestellt. Dieser Grundgedanke entspricht unmittelbar dem beim optischen Lesestift verwendeten Strich-Kode (Barcode). Wenn der Stift die Strich-Zeile abtastet, erkennt er breite und schmale Striche. Das Strich-Muster kann als eine lange Binärzahl aufgefaßt werden, in welchem die breiten Striche die 'Einsen' und die schmalen Striche die 'Nullen'

verkörpern. Abbildung 2-2 ist ein Barcode-Musterbeispiel, anhand dessen man sich ein Bild davon machen kann, wie Kodes auf einer Magnetkarte oder, tatsächlich, im HP-41C selbst abgelegt werden.

ABBILDUNG 2-2. HP-41C STRICH - KODE

Im Rechner werden die Einsen und Nullen durch Zustände mikroskopisch kleiner Transistoren dargestellt, aber der entscheidende Gedanke ist derselbe wie bei Lesestift und Kartenleser: Benutzer-Kodes werden als Teilstücke einer längeren Reihe binärer 'Bits' gespeichert. Damit der Kode einen Sinn gibt, muß der Prozessor wissen, wie die Reihe in taugliche Abschnitte aufzuspalten ist.

Betrachten wir noch einmal die Zahl '1.435245455 E59'. Nachzählen verrät uns, daß 14 'Informationsteilchen' zur Darstellung einer Dezimalzahl gebraucht werden: 10 Mantissen-Ziffern, 2 Exponenten-Ziffern, ein Mantissen- und ein Exponenten-Vorzeichen. Die Grundeinheiten oder Bausteine des Speicher-Kodes müssen in der Lage sein, jedes dieser Teilchen zu vertreten, d.h., sie müssen fähig sein, mindestens zehn verschiedene Werte anzunehmen, so daß jede der zehn Dezimalziffern dargestellt werden kann. Die Dezimalzahlen 0 bis 9 werden der Reihe nach durch die Binärzahlen 0000 bis 1001 wiedergegeben, woraus wir folgern können, daß die Grundeinheit aus vier zusammenhängenden binären Bits bestehen muß. Diese Einheit wird 'Nybble' genannt - sie besteht, wie wir gleich sehen werden, aus einem halben 'Byte' (fürwahr: Computer-Spezis können nur schlächt buchstabiren) *). Wir werden ein Nybble auch als eine 'Ziffer' ansehen, insoweit wir uns auf seine Rolle bei der Speicherung von Zahlen beziehen. Die obige Dezimalzahl wird demgemäß wie folgt mit 14 Nybbles verschlüsselt:

```
0000 0001 0100 0011 0101 0010 0100 0101 0100 0101 0101 0000 0101 1001
  +    1    4    3    5    2    4    5    4    5    5    +    5    9
```

(Die Leerstellen sind der Klarheit wegen eingefügt.) Das 'E' und der '.' bedürfen keiner ausdrücklichen Verschlüsselung, weil sich sowohl ihre Lage als auch ihr 'Wert' niemals verändern. Bei den Vorzeichen-Ziffern - von links aus gezählt die Nybbles eins und zwölf - benutzt der HP-41C '0000' für '+' und '1001' für '-'.

*) nämlich: 'bite off' - abbeißen, 'nibble' - abknabbern

Sie haben gewiß schon bemerkt, daß die vier zur Darstellung einer Dezimalzahl erforderlichen Bits Werte bis binär 1111, das entspricht dezimal 15, annehmen können. Ein Nybble ist also geeignet, 'hexadezimale' Zahlen ebensogut wie dezimale darzustellen. Diese Befähigung könnte vernachlässigt werden, wenn der HP-41C nur mit Dezimalzahlen umginge. Gleichwohl ist sogar ein Vier-Bit Nybble als Grundeinheit für das Verschlüsseln von Programmzeilen unzweckmäßig.

Der HP-41C verwendet 2 aufeinanderfolgende Nybbles, 8 Bits, genannt ein 'Byte', als Grundeinheit für die Programm-Kodierung. Binär 11111111 (hexadezimal FF) entspricht dezimal 255, so daß sich also 256 elementare Programm-Kodes bilden lassen, was selbst für einen Rechner mit der Fähigkeit des HP-41C ausreicht. Wenn der HP-41C ein Programm abarbeitet, dürfen wir uns vorstellen, daß er vom vollständigen Kode Stück für Stück 8-Bit-Happen zur Verwertung 'abbyßt'. Die Auflösung des Rätsels vom Beginn dieses Kapitels sollte jetzt langsam durchschimmern. Die Zahl '+1.435245455 E+59' ist als Folge von sieben Bytes, '01 43 52 45 45 50 59', abgelegt. Wenn sich diese Bytes im Programmspeicher befinden, stellen sie die Programmzeilen 'LBL 00', '/', 'SQRT','X>Y?', 'X>Y?', 'LN' bzw. 'SIN' dar. Der benutzerorientierte Übersetzer des HP-41C ist noch klüger, als wir ihn uns vielleicht gedacht haben. Die Übertragung in die Anzeige hängt nämlich nicht nur von dem vom Übersetzer gelesenen Kode ab, sondern auch von dem Modus (PRGM, ALPHA usw.), in welchem sich der Rechner gerade befindet. Das Rätsel deutet noch eine dritte Möglichkeit der Kode-Umsetzung an - befände sich unser Muster-Kode im Alpha-Register, würde er als Kette der sieben *Alpha-Zeichen* "≅CREEPY" angezeigt.

Das 'Schaufenster' des benutzerorientierten Übersetzers ist die Anzeige. Was Sie auch immer dort sehen ist die Darreichung irgendeines Registerinhaltes in alphanumerischer Gestalt. Wenn Sie den Rechner anschalten, ist er zunächst in einem 'Säumnis'-Modus, in welchem der Inhalt des X-Registers als Zahl in die Anzeige abgebildet wird, wobei jedes Zahl-Zeichen eine Ziffer aus dem X-Register wiedergibt. Schaltet man den HP-41C in den ALPHA-Modus, wird das Alpha-Register abgebildet, und zwar ein Zeichen je Byte. Führt man 'VIEW nn' aus, werden das Meldungsflag 50 gesetzt, um darauf aufmerksam zu machen, daß ein Register angesprochen ist, und der Inhalt von Register R_{nn} in die Anzeige abgebildet. Dieses Anzeige-Schema erlaubt Einblick in die verschiedenen Register, ohne den Inhalt des X-Registers zu zerstören. Ähnlich können wir mit 'AVIEW' ins Alpha-Register sehen. Ein 'CLD' löscht Flag 50 und stellt den Säumnis-Modus wieder her. Im PRGM-Modus wird in der Anzeige eine aus Programm-Bytes bestehende Programmzeile sichtbar. Während eines Programmlaufs bildet die 'fliegende Graugans', die durch ein 'VIEW' oder ein 'AVIEW' ersetzt werden kann, die aus dem Säumnis-Modus entstehende Anzeige. Abbildung 2-3 erläutert die hinter diesem Anzeigeverfahren versteckte Logik.

Wir haben gesehen, daß Benutzer-Programme und gespeicherte Daten als lange Folgen von Einsen und Nullen, *Bits* genannt, verschlüsselt werden. Im Datenspeicher kann jede mit Nybble bezeichnete zusammenhängende Gruppe von 4 Bits eine einzelne

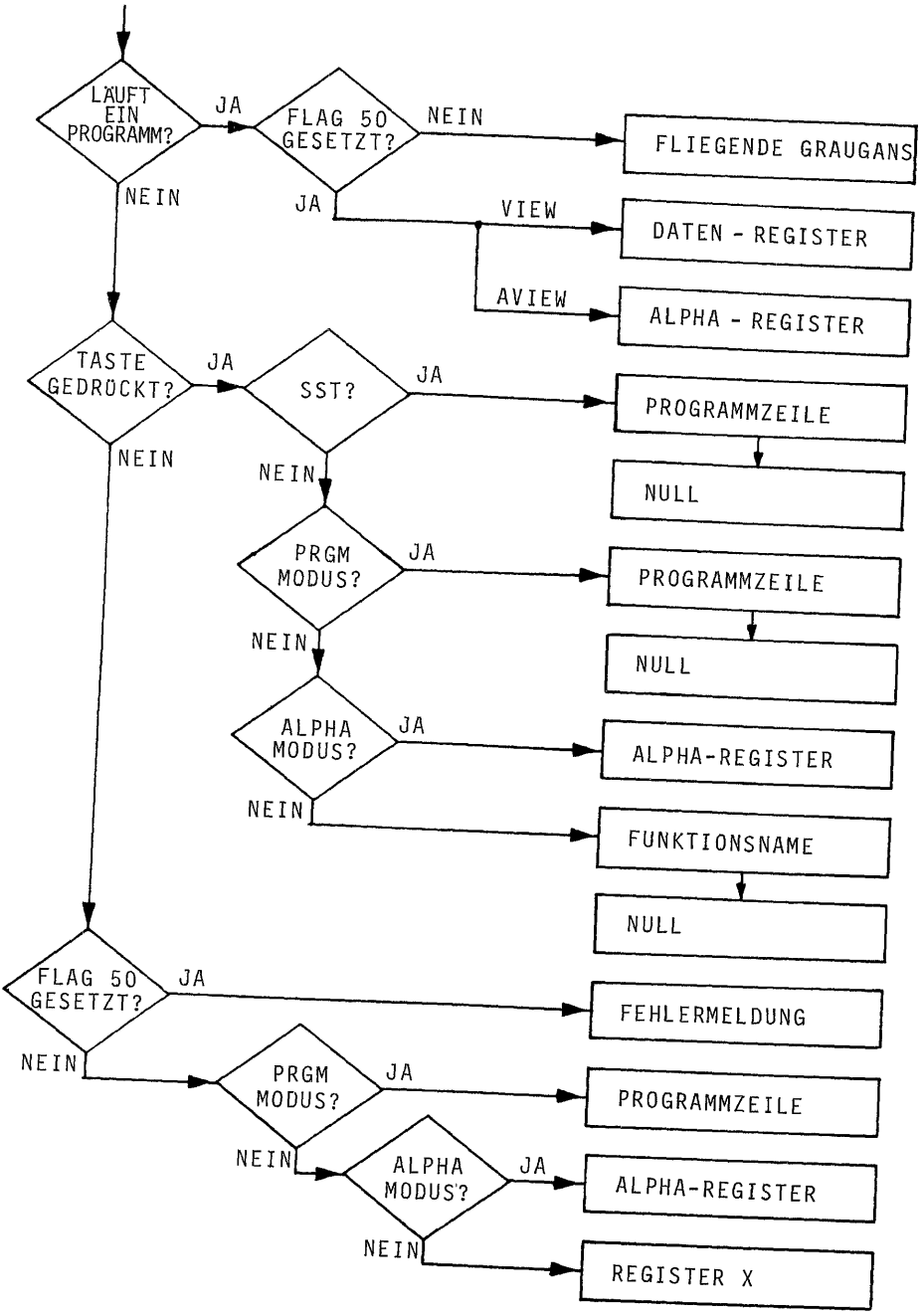


ABBILDUNG 2-3. DIE ANZEIGE - LOGIK

Dezimalziffer oder ein Vorzeichen für die Mantisse oder den Exponenten darstellen. Weil eine Zahl 14 Nybbles zur Darstellung benötigt, werden 14 Viererabschnitte (56 Bits) des Kodes zusammen in einer mit dem Ausdruck 'Register' bezeichneten Speicherstelle abgelegt oder von dort geholt. Der Befehl 'RCL 01' z.B. weist den Rechner an, die 56 Bits des Kodes, die er im mit R_{01} bezeichneten Speicher-Abschnitt findet, in einen anderen Speicher-Abschnitt, nämlich Register X, zu kopieren. Im Programmspeicher wird der Kode in Gestalt einzelner oder mehrerer Bytes auf einmal abgelegt bzw. von dort geholt. Im nächsten Abschnitt wird beschrieben, wie jedes der 256 möglichen Bytes eine eindeutige Anzahl von Programmbefehlen vertritt. Die Unterteilung des Speichers in Register ist im Programmspeicher weniger sinnfällig als im Datenspeicher, aber der in Abschnitt 2C beschriebene Adreßplan des HP-41C wird nichtsdestoweniger durch 7-Byte-Register gegliedert, so daß Register je nach Bedarf als Daten- oder Programmspeicher benutzt werden können.

2B. Die Byte-Tabelle

Bevor wir mit einer Erläuterung des im HP-41C verwendeten Adreßplanes beginnen, wollen wir das Verschlüsseln von Programmzeilen genauer betrachten. Die Grundeinheit der Programm-Verschlüsselung ist das Byte; jedes Byte kann 256 verschiedene Werte annehmen, von hexadezimal 00 bis FF. Es gibt jedoch sehr viel mehr als 256 verschiedene Programmzeilen - diese Vielfalt wird dadurch gewonnen, daß eine Programmzeile ein oder mehrere Bytes, bis zu einer Höchstzahl von 16 Stück, enthalten kann. Daher kann eine einzelne Programm*zeile* selbst dann, wenn die Anzeige nur einen Befehl vorweist, aus tatsächlich mehreren Bytes des gespeicherten Kodes bestehen.

Tabelle 2-1, die 'HP-41C Byte-Tabelle', zeigt die 256 anwendbaren Bytes in einem 16×16 Gitter. Diese Tabelle ist ein nützliches, für die synthetische Programmierung unentbehrliches Werkzeug, dessen verständiger Gebrauch für den Anfänger sehr wichtig ist. Es ist tatsächlich genau das 'Wörterbuch', mit dem der benutzerorientierte Übersetzer arbeitet. Die in der Kopfzeile der Tabelle eingetragenen Zahlen 0-F entsprechen dem ersten Nybble eines Bytes. Die den Spalten voranstehenden Zahlen 0-F liefern die zweite Ziffer des Bytes. Jedes durch Spalten und Zeilen gebildete Kästchen enthält eine Anzahl von 'Merkmalen', nämlich die verschiedenen zur Verfügung stehenden Möglichkeiten, das betreffende Byte, abhängig von seiner Lage im Speicher, zu deuten. Abbildung 2-4 zeigt ein Muster-Kästchen mit erdachten Einträgen, um alle Möglichkeiten vorzuführen.

Die erste Zahl im Kästchen, links oben in der Ecke, ist der Dezimalwert des Bytes. Diese Zahl ist zugleich der Wert, den die Drucker-Funktion 'ACCHR' benutzt, um das Druckerzeichen, welches im Kästchen rechts von der Dezimalzahl steht, auszugeben. (Die Dezimalwerte werden auch als Eingaben für das im Kapitel 5 beschriebene Tastenzuweisungsprogramm "KA" verwendet.) Z.B. finden wir in Kästchen 34 (Zeile 3, Spalte 4) die Dezimalzahl 52 (= 3×16 + 4) und das entsprechende Druckerzeichen "4" .

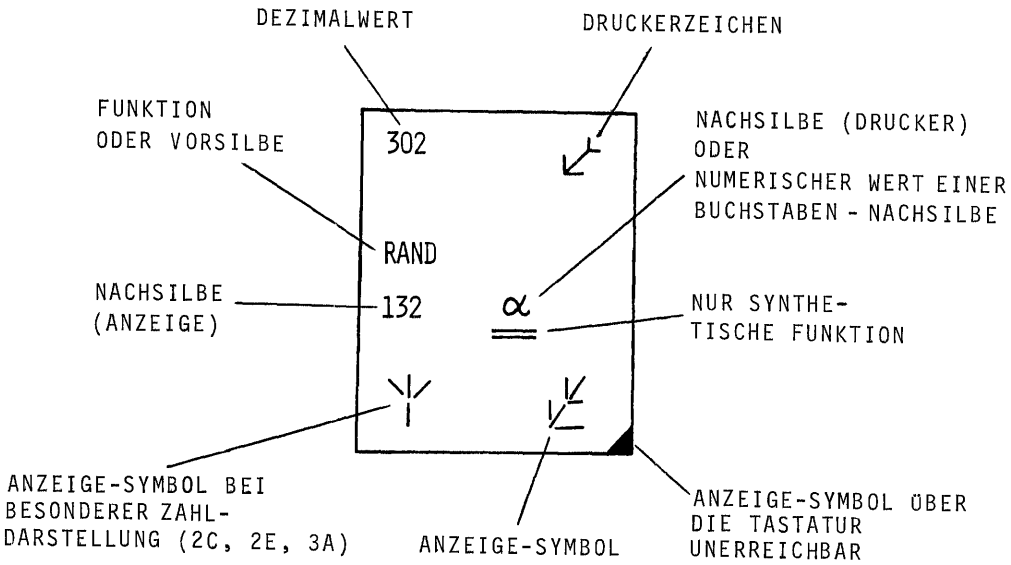


ABBILDUNG 2-4. EIN MUSTERKÄSTCHEN DER BYTE - TABELLE

Der nächste Eintrag in jedem Kästchen ist der Name einer HP-41C-Funktion. Die Bytes in Spalte 0-8 (ausgenommen Byte 1D und Byte 1E) bilden jedes für sich allein eine vollständige Programmzeile. Byte 34 wird als 'STO 04' angezeigt und ausgeführt, Byte 5C als 'ASIN' usw. Diese Bytes kann man 'Ein-Byte-Funktionen' oder 'selbständige' Bytes nennen, weil sie Vorgänge bewirken, die von den im Programm folgenden Bytes unabhängig sind.

Der HP-41C weicht von seinen Hewlett-Packard Vorgängern dadurch ab, daß er aus mehreren Bytes zusammengesetzte Programmzeilen zuläßt, im Gegensatz zu den bis dahin üblichen 'Ein-Byte-Schritten'! Die Bytes in Zeile 9, die Bytes A8-AE und die Bytes CE und CF sind 'Vorsilben'-Bytes für Zwei-Byte-Programmzeilen. Wenn der Prozessor auf ein solches Byte trifft, muß er auch das folgende Byte berücksichtigen, um den Programm-Befehl zu vervollständigen. Z.B. erfordert die durch Byte 90 gegebene Vorsilbe 'RCL' ein zweites Byte als 'Nachsilbe', um festzulegen, auf welches Register sich der Rückruf beziehen soll. Der Nachsilben-Wert jedes Bytes ist als Zahl oder Buchstabe unmittelbar unter dem Funktionsnamen im Byte-Kästchen aufgeführt. Um z.B. die Bytes '90 4C' zu entschlüsseln, beachten wir, daß laut Tabelle das erste Byte, Byte 90, die Vorsilbe 'RCL' bedeutet, also das Folgebyte, hier Byte 4C, als Nachsilbe zu berücksichtigen ist und im vorliegenden Fall demnach den Wert '76' hat.

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

TABELLE 2-1. DIE HP-41C BYTE-TABELLE

TABELLE 2-1. DIE HP-41C BYTE-TABELLE

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
8	128 ◆ DEG 00	129 × RAD 01	130 x̄ GRAD 02	131 ← ENTER 03	132 α STOP 04	133 β RTN 05	134 Γ BEEP 06	135 ↓ CLA 07	136 Δ ASHF 08	137 σ PSE 09	138 ◆ CLRG 10	139 ↘ AOFF 11	140 μ AON 12	141 ∠ OFF 13	142 ┌ PROMPT 14	143 ┼ ADV 15	8
9	144 θ RCL 16	145 Ω STO 17	146 δ STO+ 18	147 Ā STO- 19	148 ā STO* 20	149 Ä STO/ 21	150 ä ISG 22	151 Ö DSE 23	152 ö VIEW 24	153 Ü ΣREG 25	154 ü ASTO 26	155 ß ARCL 27	156 æ FIX 28	157 ≠ SCI 29	158 £ ENG 30	159 ▒ TONE 31	9
A	160 XROM 32	161 ! XROM 33	162 " XROM 34	163 # XROM 35	164 $ XROM 36	165 % XROM 37	166 & XROM 38	167 ' XROM 39	168 (SF 40	169) CF 41	170 * FS?C 42	171 + FC?C 43	172 , FS? 44	173 − FC? 45	174 . GTO IND XEQ IND 46	175 / SPARE 47	A
B	176 0 SPARE 48	177 1 GTO 00 49	178 2 GTO 01 50	179 3 GTO 02 51	180 4 GTO 03 52	181 5 GTO 04 53	182 6 GTO 05 54	183 7 GTO 06 55	184 8 GTO 07 56	185 9 GTO 08 57	186 : GTO 09 58	187 ; GTO 10 59	188 < GTO 11 60	189 = GTO 12 61	190 > GTO 13 62	191 ? GTO 14 63	B
C	192 @ GLOBAL 64	193 A GLOBAL 65	194 B GLOBAL 66	195 C GLOBAL 67	196 D GLOBAL 68	197 E GLOBAL 69	198 F GLOBAL 70	199 G GLOBAL 71	200 H GLOBAL 72	201 I GLOBAL 73	202 J GLOBAL 74	203 K GLOBAL 75	204 L GLOBAL 76	205 M GLOBAL 77	206 N X < > 78	207 O LBL 79	C
D	208 P GTO 80	209 Q GTO 81	210 R GTO 82	211 S GTO 83	212 T GTO 84	213 U GTO 85	214 V GTO 86	215 W GTO 87	216 X GTO 88	217 Y GTO 89	218 Z GTO 90	219 [GTO 91	220 \ GTO 92	221] GTO 93	222 ↑ GTO 94	223 _ GTO 95	D
E	224 ` XEQ 96	225 a XEQ 97	226 b XEQ 98	227 c XEQ 99	228 d XEQ 00 100	229 e XEQ 01 100	230 f XEQ A 102	231 g XEQ B 103	232 h XEQ C 104	233 i XEQ D 105	234 j XEQ E 106	235 k XEQ F 107	236 l XEQ G 108	237 m XEQ H 109	238 n XEQ I 110	239 o XEQ - J 111	E
F	240 p TEXT 0 T	241 q TEXT 1 Z	242 r TEXT 2 Y	243 s TEXT 3 X	244 t TEXT 4 L	245 u TEXT 5 M [	246 v TEXT 6 N \	247 w TEXT 7 O]	248 x TEXT 8 P ↑	249 y TEXT 9 Q _	250 z TEXT 10 ← →	251 { TEXT 11 a	252 | TEXT 12 b	253 → TEXT 13 c	254 Σ TEXT 14 d	255 ├ TEXT 15 e	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Also bilden die Bytes 90 4C die Zeile 'RCL 76'. Ähnlich liefern 92 1D die Zeile 'STO+29', A8 03 die Zeile 'SF 03' usw. Beachten Sie, daß bei den Bytes 00 ⊢ 63 Nachsilben-Wert und Dezimalwert des Bytes übereinstimmen. Benutzt man eines der ersten 5 Bytes von Zeile 7 als Nachsilbe, greift man auf die Stapelregister T, Z, Y, X und L (LAST X) zu, so daß also 91 70 die Zeile 'STO T', 98 73 die Zeile 'VIEW X' usw. bilden.

Einige der Bytes in Zeile 6 und 7 sind mit zwei Nachsilben-Werten, von denen einer oder beide doppelt unterstrichen sind, aufgeführt. Die Unterstreichung weist auf eine Nachsilbe hin, die nur durch die Kunstgriffe der synthetischen Programmierung zugänglich wird. Die unterschiedlichen Werte der Nachsilbe werden in Kapitel 4 erklärt.

Die Nachsilben-Werte der Zeilen 0-7 tauchen noch einmal in den Zeilen 8-F auf, wodurch wir anscheinend um 128 mögliche Nachsilben geprellt werden. Es liegt jedoch keine wirkliche Verdopplung vor: die *Nach*silben in der unteren Tabellenhälfte lassen nämlich den indirekten Aufruf der *Vor*silben-Funktionen zu. Z.B. liefert 91 52 die Zeile 'STO 82', 91 D2 dagegen 'STO *IND* 82'. Auf diese Weise kann jedes der Daten-Register R_{00}-R_{99} indirekt angesprochen werden. Andere Beispiele sind AA AA für 'FS?C IND 42'; 9D 8F für 'SCI IND 15'; 9F 86 für 'TONE IND 06'.

Byte 'AE' spielt eine doppelte Rolle als Vorsilbe. Wenn die Nachsilbe aus der oberen Tabellenhälfte stammt, wirkt AE als 'GTO IND'; kommt die Nachsilbe dagegen aus der unteren Tabellenhälfte, wird AE zu 'XEQ IND'. Z.B. entspricht AE 2A dem Befehl 'GTO IND 42', AE AA ergibt indessen 'XEQ IND 42'.

Die Bytes A0-A7 haben 'XROM' als 'Funktionsnamen'. Diese Bytes sind ebenfalls Vorsilben, aber nicht ganz in dem eben beschriebenen Sinne. Jede in einem Peripheriegerät ansässige Funktion, wie etwa 'WDTA' oder 'ACSPEC', einschließlich der nichtprogrammierbaren Funktionen wie 'WALL' oder 'LIST', sind mit einem eindeutigen Zwei-Byte-Kode, der in der Tabelle im Bereich zwischen A0 00 und A7 FF liegt, verbunden. Genauer gesagt: wir können das führende Nybble 'A' als die eigentliche Vorsilbe ansehen, während die verbleibenden 3 Nybbles gemeinsam als Nachsilbe eine bestimmte Peripherie-Funktion festlegen.

Die 'XROM'-Kodes, die in der Anzeige erscheinen, wenn ein Peripherie-Gerät abgekoppelt ist, können unmittelbar aus den Byte-Werten für die Peripherie-Funktionen abgeleitet werden. Die drei auf die Vorsilbe 'A' folgenden Nybbles müssen in zwei 6-Bit-Abschnitte zerlegt werden. Die zwei zugleich mit 'XROM' angezeigten Zahlen sind dann gerade die Dezimalwerte der beiden Teilstücke; z.B.:

'PRX' = hexadezimal A7 54 = binär 1010|0111 01|01 0100 = 'XROM 29,20'
(29, 20)

'WSTS' = hexadezimal A7 8A = binär 1010|0111 10|00 1010 = 'XROM 30,10'
(30, 10)

Einige Verwirrung könnten die Ein-Byte-Funktionen der Zeilen 0, 2 und 3 stiften. Um direkte Daten-Ablage in und -Rückgabe aus nicht weniger als 100 Registern zu ermöglichen, müssen 'STO' und 'RCL' aus zwei Bytes aufgebaut sein; sonst gin-

ge zuviel von der Hexadezimal-Tabelle für Funktionen wie 'STO 99' oder 'RCL 50' verloren. Andererseits verbraucht ein Programm mit vielen Zwei-Byte-Funktionen schneller den Speicherplatz. Der HP-41C schlägt einen Mittelweg ein, indem er für STO und RCL 00-15 Ein-Byte-Kodes zur Verfügung stellt, während er für R_{16}-R_{99} den Zugriff nur über eine Zwei-Byte Vor/Nachsilben-Verknüpfung gestattet. Gleichzeitig könnte man dadurch, daß man verschiedenen Bytes aus der Tabelle auch verschiedene numerische Nachsilben zuteilt, direkten Zugriff auf R_{100} - R_{255} gewähren, nur hätte das wiederum keinen Spielraum für die vielseitigen Eigenschaften des bestehenden Systems bezüglich der indirekten Adressierung gelassen.

Dieselbe Entscheidung zwischen Vielseitigkeit der Funktionen und Schutz der Programme gegen Speicherüberlauf tritt uns in der Verfügbarkeit von ('Kurzform'- oder) Ein-Byte-Marken als auch Zwei-Byte-Marken entgegen. Die Marken 00-14 sind in Zeile 0 der Tabelle vollständig verschlüsselt, wohingegen die Marken 15-99 je zwei Bytes benötigen: die Vorsilbe CF und eine Nachsilbe aus der oberen Tabellenhälfte. Die Zwei-Byte-Marken können sich auch der Nachsilben 66-6F und 7B-7F bedienen, nämlich um die sogenannten 'lokalen Alpha-Marken', 'LBL A' bis 'LBL J' und 'LBL a' bis 'LBL e' ins Leben zu rufen.

Die Dinge beginnen ein wenig rätselhafter zu werden, wenn wir die Tabelle abwärts gehen und in die mit Zeile B beginnenden Gebiete gelangen. Betrachten Sie z.B. Zeile E. Warum gibt es dort 16 verschiedene Vorsilben 'XEQ'? In diesem Tabellenbereich besteht die Funktion, die mit jenen Bytes beginnt, selbst aus zwei oder mehreren Bytes, so daß die Tabelle, so wie sie entworfen ist, unzulänglich für das Abbilden aller Einzelheiten wird. Wenden wir zunächst unsere Aufmerksamkeit den GTO's zu, und stellen wir dabei die 'Ein-Byte-GTO's aus Zeile B den 'Zwei-Byte-GTO's aus Zeile D gegenüber - erneut finden wir einen Kompromiß aus Vielseitigkeit und Speichernutzung, vergleichbar dem der Ein- und Zwei-Byte-STO's und -RCL's.

Wenn beispielsweise ein 'GTO 05' in ein Programm gelangt, werden diese beiden Bytes im Speicher wie folgt verschlüsselt:

1011 0110|0000 0000

Das erste Byte ist 'B6' und entspricht nach Tabelle dem 'GTO 05'. Was soll dann das zweite Byte? Wir begegnen hier einem der vielen unsichtbaren doch außergewöhnlichen Eigenschaften des HP-41C: dem 'schnellen Verzweigen'. Wenn ein laufendes Programm zum ersten Mal zur Zeile 'GTO 05' gelangt, muß der Prozessor das gegenwärtige Programm durchsuchen, bis er 'LBL 05' findet, ein (verhältnismäßig) langsamer Vorgang. Hat er einmal das Ziel gefunden, trägt er den Abstand zwischen dem 'GTO 05' und dem 'LBL 05' in das zweite ('freie') Byte des Zeilen-Kodes 'GTO 05' ein, so daß er in allen späteren Ausführungen des GTO's direkt zum LBL springen kann. Wir wollen die Einzelheiten der Verschlüsselung dieser Information erst im nächsten Abschnitt erörtern - hier genüge es zu erwähnen, daß das eine der Sprung-Information vorbehaltene Byte Sprünge bis zur Länge von 16 Registern, ent-

weder vorwärts oder rückwärts, erlaubt. Die Drei-Byte-GTO's der Zeile D haben 5 zusätzliche Bits zur Aufzeichnung des Sprung-Abstandes zur Verfügung, was Sprünge bis zu 512 Registern gestattet. Die Wahl des Programmierers zwischen Zwei- oder Drei-Byte-GTO's läuft folglich auf eine Wahl zwischen Programm-Geschwindigkeit und Programm-Länge hinaus: Wenn der Sprungabstand kleiner als 16 Register ist, ersparen die Zwei-Byte-GTO's und die entsprechenden Ein-Byte-LBL's zwei Bytes, ohne die Ausführungsgeschwindigkeit herabzusetzen. Bei größeren Sprüngen bedingen die Kurzformen jedoch beträchtlich mehr Ausführungszeit.

Die in Tabellenzeile E eingetragenen XEQ's sind genauso zusammengefügt wie die Drei-Byte-GTO's der Zeile D. Sie verrichten ihre Aufgabe in derselben Weise mit dem zusätzlichen Merkmal, daß die Adresse der XEQ-Zeile ebenso wie die Sprunglänge zum LBL hin aufgezeichnet wird: die Rückkehr-Adressen werden in zwei besonderen Registern als Teil eines 'Rücksprungstapels' (siehe Abschnitt 4F) abgelegt.

Fassen Sie Mut, wir sind fast durch mit der Tabelle! Zeile F ist die letzte: ihre Bytes weisen Programmzeilen als Alpha-Text aus. Wenn der Prozessor auf ein F (binär 1111) trifft, wird er munter und weiß, daß die Programmzeile Alpha-Text enthält. Die Anzahl der Textzeichen, die sich zwischen 1 und 15 bewegt, wird durch das zweite Nybble des Text-Bytes angezeigt. Im PRGM-Modus entsteht aus einem 'Fn'-Byte in der Anzeige das Text-Symbol "⊤", gefolgt von n Zeichen, die aus den nächsten 'n' Bytes des Programm-Speichers gewonnen werden. In einem laufenden Programm, oder bei einem SST, kopiert der Prozessor einfach die nächsten 'n' Bytes des Programms ins Alpha-Register und nimmt dann seine Tätigkeit mit dem Byte, welches dem letzten der 'n' Text-Bytes folgt, wieder auf. Beispiele:

$$\overline{}\text{A} = \text{F1 41}$$

$$\overline{}\text{BIG} = \text{F3 42 49 47}$$

$$\overline{}\text{THRILL} = \text{F6 54 48 52 49 4C 4C}$$

Für die Beherrschung der Textzeilen müssen noch die letzten Einträge in den Kästchen der Byte-Tabelle erläutert werden. Die rechte untere Ecke enthält das Alpha-Zeichen, welches in der Anzeige erscheint, wenn sich das zugehörige Byte entweder im Alpha-Register, in einer Programmzeile oder in einer globalen Alpha-Marke befindet. Die Anzeige ist imstande, 83 verschiedene Zeichen wiederzugeben. 59 davon bilden den normalen Zeichen-Satz und können unmittelbar eingetastet werden. Zwei weitere Zeichen, das Text-Symbol "⊤" und das Anschluß-Symbol "⊢", können zwar 'eingetastet' werden, jedoch nicht an beliebiger Stelle.

Die 19 Zeichen, deren Kästchen in der rechten unteren Ecke ein schwarzes Dreieck aufweisen, können nicht direkt eingegeben werden. Sie tauchen in der Anzeige bei Gebrauch der Drucker-Funktion 'BLDSPEC' auf. Die 'fliegende Graugans' "≯" sieht man während eines ablaufenden Programmes hartnäckig ihre Runden ziehen, aber um ihr Gegenstück "≮" aus dem Nest zu scheuchen, muß außergewöhnliche Mühe aufgewendet werden, denn das Byte 2C wird normalerweise als Komma angezeigt. Die Bytes 2C, 2E und 3A sind mit zwei Zeichen ausgewiesen. In der Regel wird das rechterhand ein-

getragene Zeichen erzeugt - beachten Sie in diesem Zusammenhang, daß diese drei Zeichen, ",", "." und ":", unter Verwendung der zwischen den LCD-Hauptsegmenten gelegenen besonderen Punkt/Komma-Segmente abgebildet werden. Das linkerhand eingetragene Zeichen kann vom Benutzer mit speziellen Zahl-Darstellungen, die in Abschnitt 7C beschrieben sind, gesteuert werden. Und schließlich: wenn einem Byte keines der bis hierhin erwähnten Zeichen zugeordnet ist, 'entartet' es zum vollen Bild, zu einer 'Sternexplosion' "⁕".*) Die Explosionszeichen sind außer für Byte 3A nicht in die Tabelle eingetragen.

Zwei Anmerkungen: Erstens, die 'Anschluß'-Operation wird durch Byte 7F verschlüsselt. Wenn dieses Byte allein steht, bedeutet es 'CLD', wenn es aber als zweites Byte einer Textzeile auftritt, bewirkt es den Anschluß der restlichen Bytes der Zeile an den gegenwärtigen Inhalt des Alpha-Registers. Das zweite Nybble 'n' aus 'Fn' hat in diesem Fall einen um 1 höheren Wert als die Anzahl der wirklich anzuhängenden Zeichen. "LEG" ist als 'F3 4C 45 47' verschlüsselt, "⊢LEG" dagegen als 'F4 7F 4C 45 47'. Zweitens, Byte FO, also 'TEXT 0', tritt gewöhnlich nicht in Benutzerprogrammen auf, außer als Nachsilbe 'IND T', indessen spielt es beim Verschlüsseln der Tastenzuweisungen eine Rolle.

Wir sind beim 'END'(e) angelangt. Die Bytes CO-CD, 'GLOBAL' also, spielen eine Doppelrolle - sie bestimmen sowohl 'END'-Zeilen als auch globale Alpha-Marken. Wenn das dritte Byte einer mit 'Ck' ($0 \leq k < E$) beginnenden Zeile ein Text-Byte 'Fn' ist, liegt eine globale Alpha-Marke vor. Andernfalls handelt es sich um ein Drei-Byte-'END'. In beiden Fällen gibt das zweite, dritte und vierte Nybble den Abstand der gegenwärtigen Zeile zum letzten 'END' oder zur letzten Alpha-Marke im Speicher an. Der Abstand ist genauso wie bei den Drei-Byte-GTO's (siehe Abschnitt 2C) verschlüsselt. Auf diese Weise sind alle globalen Zeilen miteinander verkettet; ein GTO-Alpha oder XEQ-Alpha beginnt die Suche in dieser globalen Kette am Ende des Programmspeichers, wo das ständig vorhandene .END. steht, und läuft rückwärts bis - falls nötig - zur ersten globalen Zeile im Speicher, die an ihren ersten zwei Bytes 'CO OO' erkannt wird. 'CAT 1' fördert die Marken und END's vorwärts, mit der ersten globalen Zeile beginnend, zutage.

In 'END'-Zeilen wird das dritte Byte dazu benutzt, Auskünfte über das gegenwärtige Programm bereitzustellen - ob es verdichtet wurde oder ob es das letzte im Speicher befindliche Programm ist, das END also das ständige .END. ist. In diesem dritten Byte zeigt das erste Nybble mit '0' ein normales END und mit '2' ein ständiges .END. an; im zweiten Nybble bedeutet '9' , daß das Programm verdichtet ist, und 'D' , daß es einer Verdichtung bedarf.

Die globalen Alpha-Marken sind die verzwicktesten aller HP-41C Programmzeilen. Das dritte Byte besteht aus einem 'Fn', in dem 'n' eine *um 1 größere* Hexadezimalzahl als die Anzahl der Zeichen der Marke ist. Das vierte Byte der Marke, welches

---

*) Das Zeichen ⁕ wird für das Anzeigesymbol ⊠ verwendet.

durch das 'Fn' bereitgestellt wird, enthält eine Verschlüsselung der Tastenzuweisung der Marke. '00' zeigt fehlende Tastenzuweisung an. Die verbleibenden 'n-1' Bytes enthalten schließlich den Namen der Marke. Beispiel:
LBL "ABC" = 'Cℓ mn F4 ab 41 42 43', wobei 'ℓmn' der Abstand zur letzten globalen Marke ist und 'ab' die zugewiesene Taste festlegt.

Es bleiben noch ein paar Einzelgänger der Byte-Tabelle zu untersuchen. Die Bytes 1D und 1E, Ausländer im Land der Ein-Byte-Funktionen, sind Vorsilben für GTO(Alpha) bzw. XEQ(Alpha). Wenn eines dieser Bytes eine Zeile anführt, folgt ihm ein 'Fn' Byte, das 'n' Bytes für den Namen der Marke freihält. Beispielsweise:

GTO "BLAZES" = 1D F6 42 4C 41 5A 45 53
XEQ "SPY" = 1E F3 53 50 59

Ferner gibt es noch den 'Unsichtbaren', Byte '00' - die 'Null'-Funktion. Dieses Byte bleibt dem Programmierer gewöhnlich verborgen, gleichwohl wird es vom HP-41C dafür benutzt, das Edieren zu erleichtern, und als Platzhalter für später einzusetzende Kodes verwendet. Beispielsweise wird eine Null automatisch vor die erste Ziffer einer Zahleneingabe gesetzt. Die Null dient so dazu, die Zahlenzeile von der vorangehenden Zeile, die auch eine Zahlenzeile sein könnte, zu trennen; die Null ist also in diesem Zusammenhang einem 'ENTER' gleichwertig. Bei Ausführung eines 'PACK' wird eine solche Null zusammen mit allen anderen im Programmspeicher vorhandenen überflüssigen Nullen entfernt, wenn sie sich als unnötig erweist.

Zuallerletzt die Bytes 1F, AF und B0: es sind 'Reserve'-Kodes, die keine Aufgabe als Vorsilbe oder Ein-Byte-Funktion erfüllen, sondern im Speicher lediglich als Nachsilbe auftreten.

2C. Die Register, mit Verlaub

Wir haben gesehen, daß der HP-41C Arbeitsspeicher und seine Nachbildungen auf Magnetkarten oder im Strich-Kode als eine lange Reihe binärer Bits angesehen werden kann, ähnlich einem Patronengurt, auf dem durch fehlende Geschosse ein Muster entstanden ist. Um Sinn hineinzubringen, gruppiert der Prozessor, wenn er die Reihe abtastet, die Bits in Nybbles und Bytes zum Entschlüsseln und in 7-Byte-Register zur Ablage von Daten und deren Rückruf. Damit der Prozessor nun weiß, welche Bits zu gruppieren sind, muß es einen Adreßplan zur Erkennung jeden Speicherabschnittes geben. Dieser Plan muß sowohl 'absolute' Adressierung erlauben, so daß der Prozessor Information in ortsfesten Bereichen wie dem Stapelregister wiederfinden kann, als auch 'relative' Adressierung, um sicherzustellen, daß Programmsprünge, wie sie bei 'GTO' und 'XEQ' stattfinden, nicht durch die Operation 'SIZE' verändert werden.

Weil der kleinste Baustein des Programmspeichers das Byte ist, und weil Daten-Register aus einer ganzen Zahl von Bytes zusammengesetzt sind, genügt es, über Einzeladressen bis hinab für Bytes zu verfügen, statt für jedes Nybble oder gar jedes Bit gesondert. Es sollte auch eine Adresse für jedes Register geben, damit der

Umgang mit Daten erleichtert und der Vorgang der Adreßsuche beschleunigt wird - wir benötigen etwas ähnliches wie eine Straßen-Adressierung, bei der Register- und Byte-Adresse dem Straßennamen bzw. der Haus-Nr. entsprechen. Diese einfachen Gedanken führen uns geradewegs zu dem im HP-41C tatsächlich verwendeten Adreßsystem. Jedes Byte im Arbeitsspeicher besitzt eine Adresse der Form:

[n a b c].

'abc' ist darin eine dreiziffrige Hexadezimalzahl, die ein einzelnes Register (absolut) kennzeichnet. Bei einem Daten-Register müssen wir zwischen seiner absoluten Adresse und seiner Daten-Register-Nummer, die eine relative Adresse ist, unterscheiden. Der Speicherplatz der im Daten-Register R_{00} abgelegten Zahl beispielsweise ist nicht festgelegt. Wenn ein neues 'SIZE' ausgeführt wird, werden die Inhalte des Speichers umgeordnet, um die Aufteilung zwischen Programm- und Datenspeicher zu verändern. Zum Vorteil für den Benutzer kann auf den ursprünglichen Inhalt von R_{00} immer noch mit 'STO 00' usw. zugegriffen werden, obwohl sich der Speicherplatz - also die absolute Adresse - des Inhalts geändert haben kann (siehe Abschnitt 4G).

Die verbleibende Ziffer 'n' der vierziffrigen Adresse ist die 'Byte-Nummer'. Jedes Register hat sieben Bytes, also kann 'n' einen der 7 Werte von 0 bis 6 annehmen. Wir erweitern nun unsere Vorstellung vom Prozessor, um den Adreß-'Zeiger', der stets die vierziffrige Adresse des gerade in der Verarbeitung befindlichen Programm-Bytes enthält, einzubeziehen. Die für den HP-41C getroffene Vereinbarung besteht darin, daß ein 'vorwärts' im Programmspeicher in Richtung aufsteigender Programmzeilen-Nummern *absteigenden* Adressen entspricht (siehe Abbildung 2-5). Bei Ausführung eines 'SST' wird die Byte-Nummer im Zeiger um die Anzahl der in der Programmzeile enthaltenen Bytes vermindert, wobei Byte 6 als das erste Byte eines Registers zählt und Byte 0 als das letzte. Wenn eine Registergrenze überschritten wird, wechselt 'n' von 0 auf 6, und 'abc' wird um 1 herabgesetzt. Die Daten-Register sind in entgegengesetzter Richtung numeriert, so daß z.B. dem Register R_{10} mit der (absoluten) Adresse '123' das Register R_{11} mit '124' folgt, darauf R_{12} mit '125' usw. Wenn wir den Zeiger auf ein Daten-Register setzen könnten und dann im PRGM-Modus Einzelschritte ausführten, würden wir sieben Programm-Bytes je Register sehen, beginnend mit einem aus Mantissen-Vorzeichen und erster Mantissen-Ziffer bestehenden Byte und endend mit den beiden Exponenten-Ziffern.

Es wurde schon dargelegt, daß die durch 'GTO' und 'XEQ' verursachten Programmverzweigungen in diesen Befehlen die Sprunglänge statt der absoluten Adresse der angesprungenen Marken in den führenden Programmzeilen festhalten. Das ist so eingerichtet, damit die durch ein 'SIZE' bewirkte oder durch Einfügen neuer Programmteile auf höherer Adresse verursachte Verschiebung von Programm-Register-Inhalten nicht auch noch eine Abänderung der gespeicherten Sprünge verlangt. Die Länge eines Sprunges wird als eine aus ganzen 7-Byte-Registern und verbleibenden Rest-Bytes gebildete Zahl ausgedrückt. Der Abstand wird von dem Byte, das den ersten Teil des

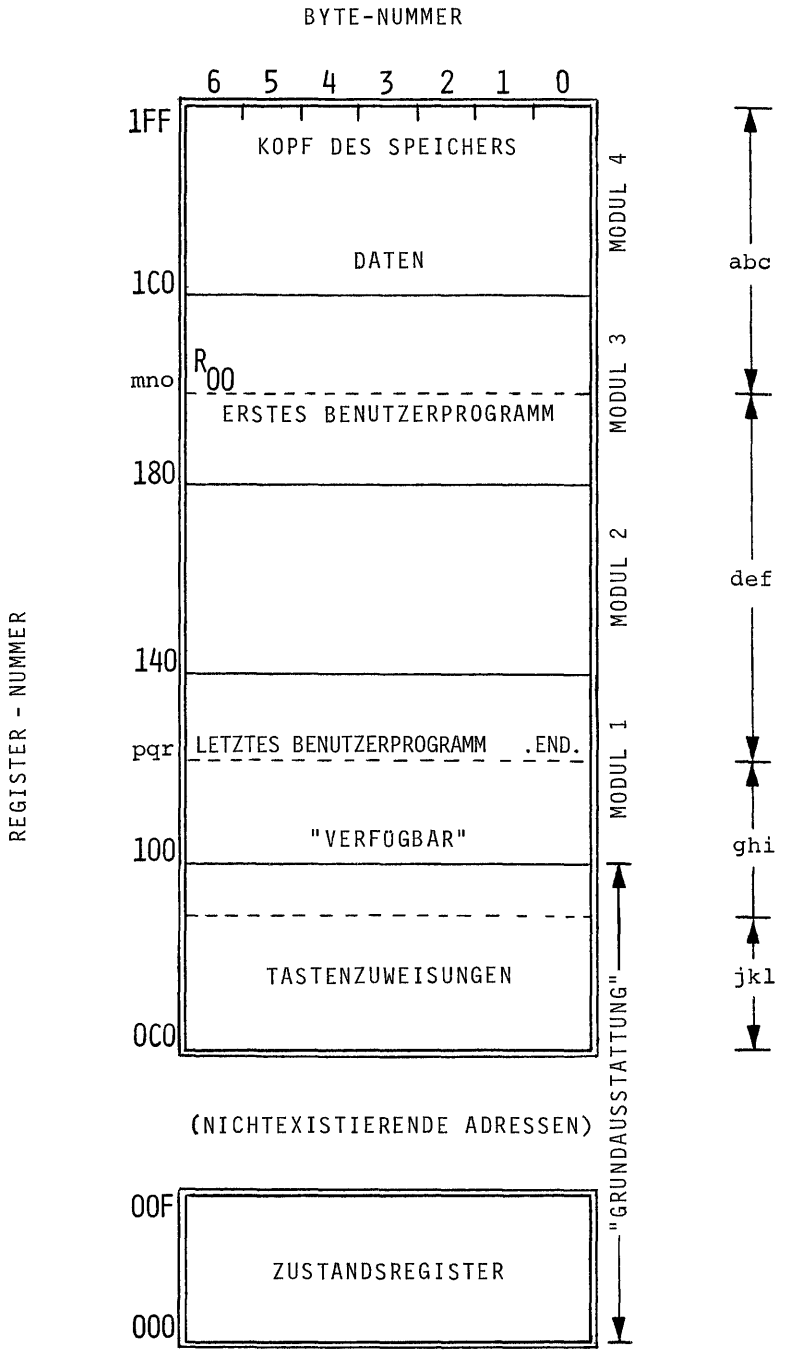


ABBILDUNG 2-5. DIE AUFTEILUNG DES HP-41C ARBEITSSPEICHERS

Sprunglängenkodes enthält, bis zu dem Byte, das der angesprungenen Marke unmittelbar vorangeht, gemessen. Um diese Verschlüsselung klarzumachen, wollen wir uns ein paar Beispiele ansehen. Nehmen wir zuerst die Routine

```
01 GTO 05               B6 22
02 "ABCDEFGHIJKLMNO"    FF 41 42 .... 4E 4F
03 LBL 05               06
04 "ABCDEFGHIJ"         FA 41 42 .... 49 4A
05 GTO 05               B6 82
```

(2C-1)

in der die Zahlen rechts von den Programmzeilen die Byte-Kodes dieser Zeilen sind. Vor der ersten Ausführung der Routine hätten die Kodes der Zeilen 01 und 05 'B6 00' gelautet. Das 'B6' bedeutet 'GTO 05'; das '00' weist auf unbekannte Sprunglänge hin. Nach der ersten Ausführung sehen die Kodes so wie oben eingetragen aus; jedes '00' ist durch den Abstandskode ersetzt. Schreiben wir die Bytes binär auf, können wir erkennen, wie die Bits gedeutet werden:

	Richtung	# Bytes	# Register
(Zeile 01) 22 =	0	010	0010
(Zeile 05) 82 =	1	000	0010

Ist das erste Bit 0, geht der Sprung vorwärts (auf eine niedrigere Adresse); ist das erste Bit 1, geht der Sprung rückwärts. Für Zwei-Byte-GTO's ist die Sprung-Information vollständig im zweiten Byte enthalten, also zählen wir den Sprungabstand von dort aus. Aus der '22' in Zeile 01 ermitteln wir: 2 Register + 2 Bytes = 16 Bytes, beginnend mit FF in Zeile 02, so daß der Zeiger auf das Zeichen "O", Byte '4F', von Zeile 02 springt. Der auf Ausführung wartende Befehl ist dann das 'LBL 05'. Für das GTO in Zeile 05 zählen wir rückwärts 2 Register + 0 Bytes = 14 Bytes, beginnend mit B6 in Zeile 05. Wieder landet der Zeiger auf dem Zeichen "O". Die größte Entfernung solcher Sprünge wird durch F Register + 7 Bytes = 112 Bytes oder 16 Register gegeben. Die Drei-Byte-GTO's und -XEQ's sind ähnlich wie die Zwei-Byte-GTO's aufgebaut, nur mit einer anderen Anordnung der Sprung-Information. Setzen wir diese Langformen in Routine 2C-1 ein:

```
01 GTO 45               D8 02 2D
02 "ABCDEFGHIJKLMNO"    FF 41 42 .... 4E 4F
03 LBL 45               CF 2D
04 "ABCDEFGHIJ"         FA 41 42 .... 49 4A
05 XEQ 45               E0 02 AD
```

(2C-2)

Wir brechen die Kodes wieder in Bits auf und gruppieren:

	Typ	# Bytes	# Register	Richtung	Marke
(Zeile 01) D8 02 2D =	1101	100	000000010	0	0101101
(Zeile 05) E0 02 AD =	1110	000	000000010	1	0101101

Für Marken-Nachsilben, die bis höchstens dezimal 99 reichen, werden nur 7 Bits benötigt; 3 weitere Bits werden für die Byte-Nummern 0-6 gebraucht. Die Darstellung des Zeilentyps verlangt 4 Bits (1101 für 'GTO', 1110 für 'XEQ'), und ein Bit gibt die Sprungrichtung an. Folglich bleiben 9 Bits für die Aufnahme der Anzahl zu überspringender Register. Somit können Sprünge bis zu 2^9 = 512 Registern ausgeführt werden, eine Entfernung die größer als der Speicher ist.

Bei einer GTO- oder XEQ-Zeile beginnt bereits das *erste* Byte mit der Sprungverschlüsselung; also müssen wir bei der Berechnung der Sprunglängen von diesem ersten Byte aus zählen. Für das 'GTO 45' in Zeile 01 errechnen wir 4 Bytes + 2 Register = 18 Bytes, was den Zeiger - vom Byte D8 ausgehend - wie vorher auf das Zeichen "0" setzt. Zeile '05 XEQ 45' veranlaßt den Zeiger, sich 0 Bytes + 2 Register = 14 Bytes weit von Byte E0 aus rückwärts zu bewegen.

Geradeso, wie es zur Erleichterung der Verschiebung von Speicherinhalten dienlich ist, Daten-Register-Nummern zu haben, die relative statt absolute Adressen sind, gibt es keine absolute Programm*zeilen-Nummer*, die fest zu irgendeinem Speicherplatz gehört. Die Zeilen-Nummer ist eine Größe, die jedesmal, wenn man sie braucht, also für die im PRGM-Modus angezeigten Programmschritte oder bei einem 'SST' oder 'BST' mit festgehaltener Taste, neu ausgerechnet wird. Sie werden schon bemerkt haben, daß beim ersten Umschalten in den PRGM-Modus nach einem Programmlauf oder beim Ausführen von 'BST' am Ende eines langen Programmes eine bemerkenswert lange Pause eintritt, bevor die Programmzeile gezeigt wird. Diese 'tote Zeit' verbraucht der Prozessor, wenn er die Zeilen-Nummer ausrechnet, was er nur vom Programmanfang aus kann, indem er nach vorn zurückkehrt, sich von dort aus vorwärts durch das Programm robbt und dabei den Zeilenzähler (der in einem besonderen Register enthalten ist) um 1 für jede vollständige Programmzeile heraufsetzt. Es wäre unnütz und zeitraubend für den Prozessor, sich während eines arbeitenden Programmes ständig auf dem laufenden über die Zeilen-Nummer zu halten, also muß er die gesamte Zeilen-Nummern-Berechnung jedesmal, wenn der Benutzer erneut in den PRGM-Modus schaltet, vollständig durchführen. Anschließende 'SST's sind schnell, aber ein 'BST' kann langsam sein, weil der Prozessor keinerlei Möglichkeit hat zu erkennen, ob das voranstehende Byte ein Einzelbyte oder die Nachsilbe einer Mehrbyte-Funktion ist. Er muß wieder zum Programmanfang zurück und Zeile um Zeile vorwärts zählen, bis er eine Zeilen-Nummer erreicht, die um eins *kleiner* als die der Startzeile ist.

2D. Die Speicheraufteilung

Abbildung 2-5 ist eine zeichnerische Darstellung des HP-41C Arbeitsspeichers, mit der wir uns alle Speicherregister als übereinander gestapelt vergegenwärtigen wollen. Die Zeichnung zeigt die 'Grundausstattung' sowie alle vier möglichen Speichererweiterungsmodule. Der Kopf der Zeichnung ist der 'Kopf des Speichers'; es ist das höchstadressierte verfügbare Daten-Register. Abwärtsgehend begegnen wir fallenden Daten-Register-Nummern und fallenden absoluten Adressen oder

steigenden Programmzeilen-Nummern. Die Byte-Nummern sind waagerecht angeordnet, und zwar das erste Byte, '6', jeden Registers links und das letzte, '0', rechts. Einzelschritt-Verarbeitung setzt den Adreßzeiger nach rechts die Bytes eines Registers entlang, dann nach links zum Byte '6' des nächstniedrigen Registers.

Das erste Daten-Register, R_{00} , und die erste Programmzeile des ersten Benutzerprogramms liegen im Speicher unmittelbar benachbart; zwischen ihnen befindet sich keine physikalische Grenze. Die augenblickliche absolute Adresse von R_{00} ist im HP-41C vermerkt, so daß der Prozessor stets weiß, welche Register der Datenspeicherung zugeteilt sind (nämlich jene oberhalb dieser Grenze) und welche Register für Programme freigehalten werden (jene unterhalb R_{00}). Das jeweilige 'SIZE' bestimmt die Anzahl der Register zwischen dem Kopf des Speichers und R_{00}. Wenn ein Speichererweiterungsmodul eingesetzt wird, werden seine 64 Register am Kopf des Speichers addiert, so daß die Anzahl der Daten-Register ohne Ausführung eines neuen 'SIZE' um 64 (hexadezimal 40) wächst. Wenn 'SIZE abc' ausgeführt wird, werden die Inhalte der Daten- und Programm-Register auf- oder abwärts gerückt, bis sich der ursprüngliche Inhalt von R_{00} im Register 'mno' (mno eine dreiziffrige Hexadezimalzahl), 'abc' Register vom Kopf des Speichers entfernt, befindet.

Register 'mno-1' ist das erste Register des Programmspeichers. Wenn wir ohne Programme im Speicher anfangen, enthalten die letzten drei Bytes des Registers 'mno-1' automatisch das ständige .END. . Dieses .END. ist notwendigerweise immer im Arbeitsspeicher vorhanden, weil es das erste Glied der Adressen-Kette, die alle globalen Marken und END's miteinander verbindet, bildet. Sobald wir mit dem Eintasten eines Programmes beginnen, überschreiben die ersten vier Bytes die übriggebliebenen Null-Bytes im Register 'mno-1'. Wenn weitere Programm-Bytes eingetastet werden, wird das .END. automatisch in die letzten drei Bytes des folgenden Programm-Registers verschoben, was Platz für die nächsten sieben Programm-Bytes schafft. Dieser Vorgang wiederholt sich, bis das Programm fertig ist oder alle verfügbaren Programm-Register voll sind. Wenn wir an irgendeiner Stelle ein 'END' eintasten, errichten wir eine Schranke im Speicher, die dazu dient, die vorangehenden Programmzeilen als ein in sich abgeschlossenes Programm abzusondern. Die 'END'-Zeile selbst ist diese Schranke, denn wenn man - SST verwendend - auf sie trifft oder wenn eine Programmsuche nach einer globalen Marke stattfindet, veranlaßt sie den Zeiger auf das nächste 'END' in der Marken-Kette zurückzuspringen oder auf Byte 'Omno', falls das gegenwärtige Programm das erste im Speicher ist.

Wenn wir insgesamt 'def' Programm-Register (einschließlich des .END.) vollgeschrieben haben, lautet die Adresse des Registers, welches das .END. enthält, pqr = mno - def. Denken Sie daran, daß diese ganze Register-Rechnung hexadezimal durchgeführt wird. 'pqr' kann im HP-41C niemals kleiner als hexadezimal 0C0 (dezimal 192) sein. Wegen der Wahl von '0C0' für den Boden des Programmspeichers bietet die 'Grundausstattung' Adressen im Bereich von 0C0 bis OFF.
Ist die erste Ziffer eines Registers eine '1', liegt das Register in einem Spei-

chererweiterungsmodul: Modul 1 - Register 100 bis 13F; Modul 2 - 140 bis 17F; Modul 3 - 180 bis 1BF; Modul 4 - 1C0 bis 1FF.

Es stehen jederzeit pqr - 0C0 Register, vermindert um die Anzahl der jeweils für Tastenzuweisungen verwendeten Register, für Programme zur Verfügung. Die vom Benutzer verordneten Tastenzuweisungen liegen in einem Registerblock, der bei 0C0 beginnt und im Speicher aufwärts läuft (die Einzelheiten der Verschlüsselung werden im Abschnitt 2E erläutert). Hat man 'jkl' Register mit Tastenzuweisungen belegt, sind noch ghi = mno - def - jkl - 0C0 Register für neue Programmzeilen oder weitere Zuweisungen verfügbar. Alles in allem besitzt der HP-41C

$$40\cdot(N+1) = abc + def + ghi + jkl$$

Register (hexadezimale Arithmetik!), wobei N die Anzahl der augenblicklich eingesteckten Speichermodule ist.

Unterhalb des Registers 0C0 gibt es eine Lücke in der Abbildung, die einen leeren Adreßbereich andeuten soll, weil es keine Register gibt, denen Adressen aus diesem Bereich entsprächen. Zwischen den Adressen 000 und 00F jedoch gibt es einen hochinteressanten Block von 16 Registern. Wir wollen diese Register 'Zustandsregister' nennen, weil ihr Inhalt vom Kartenleser mit der Funktion 'WSTS' ('Write Status') auf die Spur 1 einer Karte aufgezeichnet wird. Der Zugriff auf diese Register ist die Grundlage der synthetischen Programmierung; ihre Erkundung verdient ein ganzes Kapitel, Kapitel 4.

2E. Die Tastenzuweisungsregister

Die Tastenzuweisungsregister reichen vom Register 0C0 bis zu dem Register, welches das .END. enthält, dieses Register selbst aber nicht einschließend. Sie enthalten Kodes, die dem Prozessor mitteilen, welche Funktionen welchen Tasten zugewiesen sind (rufen Sie sich ins Gedächtnis zurück, daß die Zuweisungen der globalen Marken des Benutzers in der Marke selbst aufgezeichnet werden). Betrachten Sie die Tastenfolge

ASN ALPHA "LN" ALPHA 8 (weise 'LN' der Taste 8 zu).

Wenn wir in der Lage wären, den Adreßzeiger durch einen Taschenspielertrick im PRGM-Modus ins Register 0C0 zu setzen und den Inhalt aufzulisten, sähen wir folgendes (mit willkürlichen Zeilen-Nummern und rechterhand aufgeführten Byte-Kodes):

```
01 ""        F0
02 LBL 03    04
03 LN        50
04 RCL 05    25
```

(Zeile 01, Byte F0 oder 'TEXT 0', erscheint in der Anzeige als '01⊤'.) Vier Bytes

füllen ein Register nicht aus - es gibt noch drei unsichtbare Null-Bytes zwischen den Zeilen 01 und 02. Die Nullen verschwinden, wenn wir eine zweite Zuweisung vornehmen:

ASN ALPHA "LOG" ALPHA SHIFT 8 (weise 'LOG' der umgeschalteten Taste 8 zu).

Jetzt enthält Register 0C0:

```
01 ""        F0
02 LBL 03    04
03 LOG       56
04 RCL 13    2D
05 LBL 03    04
06 LN        50
07 RCL 05    25
```

Diese Folge von Zeilen hat als Programm keinen Sinn, obgleich wir die zugewiesenen Funktionen 'LOG' und 'LN' wiedererkennen. Die Bytes bilden vielmehr einen besonderen Kode. Das erste Byte, 'F0', weist das Register als Tastenzuweisungsregister aus und trennt es von benachbarten Zuweisungsregistern. Die nächsten drei Bytes bilden den Kode für die 'LOG'-Zuweisung, die als zweite Zuweisung getätigt wurde. Die letzten drei Bytes verschlüsseln die 'LN'-Zuweisung. In beiden Drei-Byte-Gruppen bestimmen die ersten beiden Bytes die zugewiesene Funktion, und das dritte Byte bezeichnet die Taste, der sie zugewiesen wurde. Bei der Zuweisung von HP-41C-Funktionen wird nur ein Byte für die Bestimmung dieser Funktion gebraucht, so daß das Byte '04' (LBL 03) bloß ein Füll-Byte bildet. Wenn dagegen eine Peripherie-Funktion zugewiesen wird, sind beide Funktionsbytes zur Darstellung dieser Funktion erforderlich. Hätten wir beispielsweise 'PRP' und 'WSTS' statt 'LN' und 'LOG' zugewiesen, enthielte Register 0C0 die Byte-Folge:

```
01 ""        F0
02 WSTS      A7 8A
03 RCL 13    2D
04 PRP       A7 4D
05 RCL 05    25
```

Der Kode für die eine Zuweisung tragende Taste entsteht folgendermaßen: Angenommen, wir weisen der Taste 'MN', also der Taste in Zeile M und Spalte N des Tastenfeldes, eine Funktion zu. Dann erhält das Byte, welches diese Taste im Zuweisungsregister vertritt, die Hexadezimaldarstellung 'XY', in der X = N-1 und Y = M ist. Die obige Zuweisung auf die Taste '8', die man im Tastenfeld unter Bezeichnung '53' auffindet, ergibt nach dieser Regel das Byte '25', das seinerseits für das Auftreten der Zeile 'RCL 05' im Zuweisungsregister verantwortlich ist. Weitere Beispiele: die 'COS'-Taste, Taste 24, wird durch das Byte '32' dargestellt und erscheint als Zeile 'STO 02'; 'R/S', Taste 84, wird als '38' verschlüsselt und als 'STO 08' angezeigt.

Der Kode für eine umgeschaltete Taste '-MN' entsteht nach der Rechenregel X = N-1, Y = M+8. So wird die Zuweisung auf die umgeschaltete Taste 8, die Taste -53, als 2D = 'RCL 13' verschlüsselt. Bei umgeschalteten Tasten der Zeile 8, in der M = 8 ist, erhalten wir für Y ein hexadezimales '10' (= 8+8), von dem wir die '1' nach X übertragen; 'VIEW' z.B., Taste -84, wird zu 40 = '+'. Die im Tastenfeld ursprünglich mit 42, 43 und 44 bezeichneten Tasten 'CHS', 'EEX' und '←' und ihre umgeschalteten Zweittasten -42, -43 und -44 liegen genau genommen in den Spalten N = 3, 4 bzw. 5, und das muß in der Byteformel für die Tastenzuweisung berücksichtigt werden, so als ob die 'ENTER'-Taste 41 auch noch eine imaginäre Taste 42 bedeckte. Abbildung 2-6 zeigt die Tastenzuweisungskodes in einer Tastenfeldschablone zum bequemen Nachschlagen. Die Kodes für die umgeschalteten Tasten sind oberhalb eingetragen.

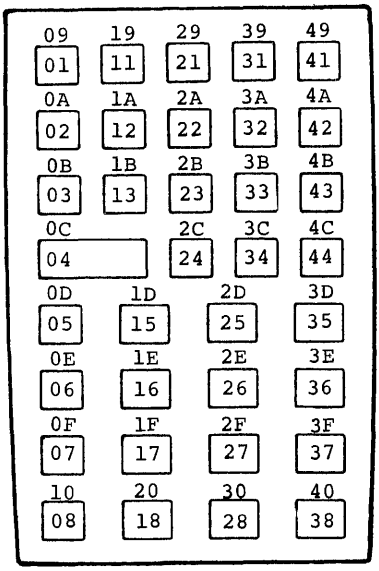


ABBILDUNG 2-6. DIE TASTENZUWEISUNGSBYTES

Sofern die zugewiesene Funktion eine Vorsilbe wie 'STO', 'ISG', 'GTO' usw. ist, zeigt die Auflistung des Tastenzuweisungsregisters Funktionsbyte und Tastenbestimmungsbyte in einer einzelnen Zeile verschmolzen an: das Tastenbestimmungsbyte spielt die Rolle einer Nachsilbe für die zugewiesene Vorsilbe.

Wenn eine nicht-programmierbare HP-41C-Funktion zugewiesen wird, stammt das Funktionsbyte aus Zeile 0 der Byte-Tabelle, so daß die entsprechende sichtbare Zeile eine der Kurzform-Marken oder 'NULL' ist. Tabelle 2-2 zeigt die entsprechenden Beziehungen:

Tabelle 2-2

Zuweisung nicht-programmierbarer HP-41C-Funktionen

Funktion	Byte	Programmzeile
CAT	00	NULL
@c*)	01	LBL 00
DEL	02	LBL 01
COPY	03	LBL 02
CLP	04	LBL 03
R/S	05	LBL 04
SIZE	06	LBL 05
BST	07	LBL 06
SST	08	LBL 07
ON	09	LBL 08
PACK	0A	LBL 09
←	0B	LBL 10
ALPHA/PRGM/USER	0C	LBL 11
2__	0D	LBL 12
SHIFT	0E	LBL 13
ASN	0F	LBL 14

Obwohl die meisten Einträge in Tabelle 2-2 normalen Zuweisungen entsprechen, beziehen sich die Bytes 01, 05, 0B, 0C, 0D und 0E auf Funktionen, die einer vorschriftsmäßigen Zuweisung nicht zugänglich sind. Benutzen wir hingegen das Tastenzuweisungsprogramm aus Kapitel 5, können wir diese Bytes in Tastenzuweisungsregister schmuggeln, was zu wunderlichen Wirkungen führt. Die 'Funktionen' '@c' und '2__' werden so bezeichnet, weil ein Druck auf die ihre Zuweisung tragende Taste in der Anzeige gerade diese Abbildungen hervorbringt. Der Aufruf von '@c' bewirkt manchmal gar nichts; zuweilen wird ein 'GTO..' ausgeführt. '2__' im Anschluß an den Eintrag einer zweiziffrigen Zahl veranlaßt den HP-41C, sich dem Benutzer für einige Zeit völlig zu 'verschließen'. Die Bytes 05, 0B und 0E lassen sich schon eher verwenden, denn sie erlauben uns, die Funktionen 'R/S', '←' bzw. 'SHIFT' anderweitig zuzuweisen. Der Druck auf eine mit 0B ins Leben gerufene Korrekturtaste löscht ohne Rücksicht darauf, ob sich der HP-41C im PRGM-Modus befindet oder nicht, die gegenwärtige Programmzeile. Das letzte Byte 0C schließlich nimmt den 'Schaukelstuhl-Tasten' 'ALPHA', 'PRGM' und 'USER' ihr Alleinvertretungsrecht. Die Auswahl der Funktion, die mit 0C zugewiesen wird, hängt von der Taste ab, die die Zuweisung empfangen soll (!): liegt die Taste in Zeile 1 oder 5, wird 'ALPHA' zugewiesen; Tasten in den Zeilen 2 und 6 erhalten 'PRGM' zugeteilt; 'USER' ergibt sich bei Zuweisung an Tasten in den verbleibenden Zeilen 3, 4, 7 und 8.

*) Das Zeichen @ wird für das Anzeigesymbol ▷ verwendet.

Kapitel 3

EXOTISCHES EDIEREN MIT DEM BYTE-HÜPFER

3A. Gewöhnliches Edieren

Jeder HP-41C-Programmierer kennt die einfachen Regeln, die das gewöhnliche Edieren beherrschen:

(1) Im PRGM-Modus wird eine eingetastete Zeile unmittelbar hinter die vorher in der Anzeige erschienene Programmzeile eingefügt. Bei allen folgenden Zeilen wird die Zeilen-Nummer um eins erhöht.

(2) Wenn man die Korrekturtaste drückt, wird die angezeigte Programmzeile gelöscht, und die Zeilen-Nummern der folgenden Zeilen werden um 1 herabgesetzt. In der Anzeige erscheint die der gelöschten Zeile vorangehende Zeile.

(3) Die Ausführung von 'DEL ℓmn' bewirkt die Löschung von 'ℓmn' Programmzeilen hintereinander, beginnend mit der angezeigten Zeile.

(4) 'PACK' veranlaßt eine Art haushälterischer Tätigkeit, indem unsichtbare Nullen beseitigt werden, um den zum Programmieren verfügbaren Raum möglichst groß zu halten.

Diese Verfahren gestatten ein einfaches und schnelles Edieren auf dem HP-41C. Doch für unsere Zwecke sind die in den Schritten (1) bis (4) gelieferten Auskünfte des Rechners unzureichend; wir müssen genau wissen, was im Speicher auf der Byte-Stufe vor sich geht, nicht auf Zeilen-Niveau. Wir wollen daher die Regeln wie folgt neu fassen:

(1) Die Programmzeile, die im PRGM-Modus gezeigt wird, ist diejenige, die mit dem ersten Nicht-Null-Byte beginnt, das dem Byte, auf welchem der Adreßzeiger augenblicklich steht, folgt. Wenn eine neue Programmzeile eingetastet wird, werden die Bytes, die die neue Zeile bilden, unmittelbar hinter das letzte Byte der vorher gezeigten Programmzeile gesetzt, wobei sie Null-Bytes überschreiben. Falls keine Null-Bytes vorhanden sind, falls also das Byte an der Einfügungsstelle nicht '00' ist, setzt der Prozessor selbständig 7 Nullen (oder, sofern erforderlich, ein Vielfaches davon) ein, bevor die neuen Programm-Bytes an ihren Platz gelangen. Die neuen Zeilen überschreiben dann soviele von den neuen Nullen, wie sie selbst an Platz brauchen, und lassen den Rest der Nullen (unsichtbar) im Programm zurück. Das Einbringen von insgesamt 7 Nullen vereinfacht den Vorgang der nach unten gerichteten Zeilenverschiebung im Speicher - jedes Benutzer-Programme enthaltende Register wird in das nächstniedrigere Register kopiert, wobei der Start beim .END. erfolgt und die Kopierarbeit aufwärts bis zu dem Register

geleistet wird, wo die Einfügung erfolgen soll. Ein von Hand ausgeführtes 'RTN', 'GTO.000' oder 'GTO.001' bewegt den Adreßzeiger auf das letzte Byte des vorangehenden Programms. Die ersten beiden Möglichkeiten, bei denen die Zeilen-Nummer auf '00' gesetzt wird, münden in einer Anzeige von '00 REG ℓmn' statt einer Programm-Zeile. In diesem Fall werden die eingetasteten Bytes unmittelbar hinter das Byte, auf welches der Adreßzeiger weist, eingefügt statt hinter der vor Ausführung stehenden Programmzeile.

(2) Wenn die Korrekturtaste im PRGM-Modus betätigt wird, werden die Bytes der angezeigten Zeile durch eine gleiche Anzahl von Nullen ersetzt. Der Adreßzeiger springt um eine Zeile zurück.

(3) Der 'DEL ℓmn'-Befehl ersetzt alle Bytes der folgenden 'ℓmn' Zeilen, die angezeigte Zeile eingeschlossen, durch Nullen. Sie können beobachten, wie ein auf eine Löschung sehr vieler Zeilen folgendes 'SST' ungewöhnlich lange mit der Ausführung zögert, was daher rührt, daß der Prozessor alle aus der Löschung stammenden Null-Bytes überfliegen muß, bis er ein Nicht-Null-Byte zum Anzeigen findet.

(4) Längeres Edieren kann eine ansehnliche Anzahl überflüssiger Nullen in ein Programm befördern. Der 'PACK'-Befehl entfernt alle unnötigen Nullen, indem er die Programm-Bytes im Speicher nach oben schiebt. Nullen innerhalb einer Mehrbyte-Programmzeile werden nicht beseitigt.

Wenn beim Edieren (Einfügen oder Löschen von Bytes) oder durch Verdichten mit 'PACK' Programm-Bytes im Speicher an andere Stellen gelangen, können verschiedene Sprunglängen-Kodes ungültig werden. Daher wird die Abstandsinformation in allen lokalen GTO's und XEQ's im Anschluß an diese Vorgänge in dem Programm, das gerade bearbeitet wird, auf Null gesetzt, so daß sie beim nächsten Programmlauf neu berechnet werden muß. Darüberhinaus müssen auch die relativen Adressen in der Globalmarken-Kette auf den neuesten Stand gebracht werden. Und schließlich wird in dem das Programm abschließenden END aufgezeichnet, ob das Programm der Verdichtung bedarf bzw. ob das geschehen ist.

Ein Beispiel für das Edieren eines Programms soll unsere umgearbeiteten Regeln klarmachen. Beginnen wir mit einem "MEMORY LOST" (Rechner bei gedrückter Korrekturtaste einschalten, dann Korrekturtaste loslassen), und tasten wir das folgende höchst einfache 'Programm' ein:

```
01 LBL 00
   .END.
```

Wenn wir alle Bytes dieses Programms aufschreiben, sieht das Ergebnis so aus:

Adresse	Zeilen-Nr.	Zeile	Byte-Kode
60EE	01	LBL 00	01
50EE			00
40EE			00
30EE			00
20EE		.END.	C0
10EE			00
00EE			29

Die Adressen ergeben sich aus der Tatsache, daß der HP-41C nach einem "MEMORY LOST" mit 47 (hexadezimal 2F) Programm-Registern 'erwacht' und der Programmspeicher mit Register 0C0 (vgl. Abbildung 2-5) anfängt: 0C0 + 02F - 1 = 0EE. Darum ist Register 0EE das oberste Programm-Register. In der .END.-Zeile zeigen die Nybbles '000' an, daß es sich um die höchste Globalmarke im Speicher handelt; die '29' kennzeichnet das ständige .END. in einem verdichteten Programm. Nehmen wir an, wir fügen hinter Zeile 01 drei '+'-Zeilen ein:

60EE	01	LBL 00	01
50EE	02	+	40
40EE	03	+	40
30EE	04	+	40
20EE		.END.	C0
10EE			00
00EE			29

Die Nullen sind durch die '40'er Bytes ersetzt worden. Jetzt löschen wir Zeile 03:

60EE	01	LBL 00	01
50EE	02	+	40
40EE			00
30EE	03	+	40
20EE		.END.	C0
10EE			00
00EE			2D

Die '40' auf der Adresse 40EE ist wieder zur Null geworden; das letzte Byte im .END. hat sich in ein '2D' verwandelt, um ein unverdichtetes Programm zu signalisieren. Wenn wir nun verdichteten, verschöbe sich die '40' von 30EE nach 40EE, doch entstünde ein anderes '00' auf 30EE, um das .END. an seinem Platz in den letzten drei Bytes zu halten. Wenn wir eine Ein-Byte-Zeile hinter Zeile 02 setzten, überschriebe sie einfach die Null auf 40EE. Wenn wir jedoch eine Zwei-Byte-Zeile, etwa 'STO 65', einfügen, erhalten wir

60EE	01	LBL 00	01
50EE	02	+	40
40EE	03	STO 65	91
30EE			41
20EE			00
10EE			00
00EE			00
60ED			00
50ED			00
40ED			00
30ED	04	+	40
20ED		.END.	C0
10ED			00
00ED			2D

Weil zwischen den Zeilen 02 und 03 nur ein Null-Byte zum Überschreiben zur Verfügung stand, sind 7 weitere Nullen eingesetzt worden. Zwei von ihnen wurden von den beiden Bytes des 'STO 65' überschrieben. Die '40' von 30EE ist nach unten auf 30ED gerückt, und das .END. ist in die letzten 3 Bytes des Registers 0ED gebracht worden. Folgt ein 'PACK', erhält das Programm die Gestalt:

60EE	01	LBL 00	01
50EE	02	+	40
40EE	03	STO 65	91
30EE			41
20EE	04	+	40
10EE			00
00EE			00
60ED			00
50ED			00
40ED			00
30ED			00
20ED		.END.	C0
10ED			00
00ED			29

Die Programmzeilen sind zusammengeschoben worden, aber weil dem Register 0EE ein Byte für die Unterbringung des .END. fehlte, verbleibt die Schlußmarke im Register 0ED.

3B. Der Byte-Hüpfer

Nunmehr bewaffnet mit ausreichender Kenntnis über die gewöhnlichen HP-41C-Operationen, können wir verwegen in brandneues Territorium vorstoßen. Es soll an dieser Stelle noch einmal betont werden, daß selbst dann, wenn einige der Arbeitsweisen, die wir jetzt gleich anwenden wollen, recht sonderbar erscheinen, für den HP-41C keine Gefahr besteht. Begleiten Sie mich durch das folgende Verfahren (HP-41CV Besitzer seien auf das Addendum im Vorwort verwiesen):

1. Setzen Sie einen Speichererweiterungsmodul in den HP-41C.
2. Totallöschung (HP-41C aus; Korrekturtaste gedrückt halten; HP-41C an; Korrekturtaste loslassen). Ein klarer Bruch mit der Vergangenheit!
3. 'SIZE 000' ausführen, um das .END. in den Modul zu legen.

4. ASN "X<>" + und ASN "Σ+" Σ+ füllt Register 0C0 mit zwei Zuweisungen.
5. HP-41C aus; Modul entfernen; etwa 60 Sekunden warten; Modul wiedereinsetzen; HP-41C an. (Wenn Sie einen zweiten Modul zur Hand haben, können Sie sich die 60 Sekunden Wartezeit ersparen, indem Sie den 'toten' Modul an die Stelle des entfernten setzen.) Jetzt hat sich das .END., welches wir in den Modul gebracht haben, 'verflüchtigt'. Hätten Sie den Rechner *vor* dem Wiedereinsetzen des Moduls eingeschaltet, wäre ein "MEMORY LOST" die Folge gewesen. Offensichtlich prüft der Prozessor nach, ob das Register, in welchem er das .END. vermutet, vorhanden ist, nicht aber, ob die .END.-Bytes dort wirklich an Ort und Stelle sind.
6. Wechsel in den PRGM-Modus; Sie sollten '00 REG 126' sehen (190, falls Sie einen Modul von doppeltem Speicherumfang verwendet haben). Drücken Sie nun einmal 'SST'. Nach einigen Sekunden werden Sie '01⌐' erblicken. Der Adreßzeiger ist jetzt im Register 0C0, dem ersten Zuweisungsregister! Weil das .END. fehlte, konnte den Zeiger nichts davon abhalten, fröhlich durch den leeren Speicher zu brausen, bis er schließlich dem ersten Nicht-Null-Byte begegnete, welches im vorliegenden Fall das aus den in Schritt 4 erfolgten Tastenzuweisungen herrührende Byte 'F0' war. Wenn Sie fünfmal 'SST' ausführen, sollte die nachstehend aufgeführte Befehlsfolge erscheinen:

```
02 LBL 03
03 Σ+
04 LBL 00
05 LBL 03
06 X<>06
```

(wenn Sie SST abermals drücken, gelangt der Zeiger in die Zustandsregister.) Sie werden diese Zeilen als die Kodes der in Schritt 4 ausgeführten Tastenzuweisungen erkennen.
7. Benutzen Sie BST, um zur Zeile 03 zurückzukehren. Seien Sie nicht beängstigt, wenn einige dieser SST's und BST's ein paar Sekunden Zeit beanspruchen. Drücken Sie anschließend zweimal die Korrekturtaste, um die Zeilen 03 und 02 zu löschen.
8. Tasten Sie ALPHA "A" ALPHA ein, was die Zeile '02⌐A' ergibt. (Statt "A" kann übrigens ebensogut jedes beliebige andere Einzelzeichen genommen werden.)
9. Führen Sie 'GTO..' aus; das 'GTO..' wird ein paar Sekunden in der Anzeige verweilen, gefolgt von einem schnellen "PACKING". Noch einmal SST, und dann Zeile '01 LBL 01' löschen. Zum zweiten und letzten Male in diesem Buch haben Sie ein Verfahren der synthetischen Programmierung, das sich auf den Trick des 'Modul-Entfernens' stützt, angewendet. Von nun an sind wir in der Lage, alle Ziele zu erreichen, ohne Zuflucht zu solch unerfreulicher Gefechtskunst nehmen zu müssen.

10. Drücken und halten Sie die Taste 'Σ+' im USER-Modus. Sie sollten 'XROM 05,01' angezeigt sehen. (Wenn diese Anzeige nicht erscheint, müssen Sie einen Fehler gemacht haben und die Schritte 1 bis 9 wiederholen.) Wir haben durch direktes Füllen eines Zuweisungsregisters eine funkelnagelneue Tastenzuweisung erzeugt, den sog. 'Byte-Hüpfer'.

Die genaueste Vorstellung von der Wirkungsweise des Byte-Hüpfers erhält man, wenn man ihn als *manuellen 'Programm-Textzeilen-Bearbeiter'* auffaßt. Um dies zu verstehen, rufen Sie sich ins Gedächtnis zurück, was geschieht, sobald eine Textzeile *automatisch* bearbeitet wird: Der Prozessor betrachtet das zweite Nybble des vorliegenden Programm-Bytes, also des Bytes 'Fn', welches signalisiert, daß eine Textzeile vorliegt, kopiert die nächsten 'n' Bytes des Programm-Kodes ins Alpha-Register und setzt den Zeiger um 'n' Bytes vor. Der Byte-Hüpfer ist eine Tastenfunktion, die solche Textverarbeitung von Hand durchzuführen erlaubt, und darf nicht mit der Einzelschritt-Verarbeitung einer Textzeile unter 'SST' verwechselt werden. Das Nybble 'F', welches den Vorgang auslöst, 'beschaffen' wir uns durch Betätigung der neuen USER-Taste, der wir ein "A" ('F1 41') zugewiesen haben. Um zu erkennen, warum diese Funktion interessant ist, tasten Sie die folgenden Zeilen ein:

```
01 STO 04        34                              (3B-1)
02 "ABCDEFG"     F7 41 42 43 44 45 46 47
```

Während Zeile 02 angezeigt wird, schalten Sie den PRGM-Modus aus, den USER-Modus an und betätigen den Byte-Hüpfer (Taste Σ+). Dann wieder in den PRGM-Modus, und Sie werden schrittweise

```
02 X<Y?      44
03 X>Y?      45
04 X<=Y?     46
05 Σ+        47
```

erblicken. Woher kommen diese Programmzeilen? Durch einen Blick auf die Byte-Werte der 'neuen' Programmzeilen überzeugen Sie sich leicht, daß es sich ganz einfach um die den Zeichen "D", "E", "F" und "G" aus der ursprünglichen Textzeile "ABCDEFG" entsprechenden Ein-Byte-Funktionen handelt. Wir begannen mit Zeile '02 "ABCDEFG"' in der Anzeige, also während der Adreßzeiger auf die '34' in Zeile 01 wies. Dann betätigten wir den Byte-Hüpfer, was den Prozessor glauben machte, eine Textzeile sei zu verarbeiten (verwechseln Sie diesen imaginären Text nicht mit dem echten Text in Zeile 02). Folgerichtig betrachtete er das zweite Nybble des vorliegenden Bytes, '34', kopierte die nächsten 4 Bytes ins Alpha-Register und rückte den Zeiger um 4 Bytes auf das Byte '43' vor. In dieser Zeigerstellung erscheint in der Anzeige die nächste Programmzeile, die hier die dem Byte '44' entsprechende Ein-Byte-Zeile '02 X<Y?' ist. Wenn Sie jetzt PRGM aus- und ALPHA einschalten, sehen Sie die vier Bytes "✱ABC", die durch das Kopieren der ersten 4 Bytes der Programmzeile 02 entstanden sind. Das Explosionszeichen ist das Byte 'F7'. Weil das Byte-Hüpfen von Hand ausgeübt wird, verändert es die laufende Programmzeilen-

Nummer nicht, obwohl sich der Zeiger bewegt, so daß die Zeile '02 X<Y?' dieselbe Zeilen-Nummer trägt wie die Zeile '02 "ABCDEFG"', von der aus der Sprung ausgeführt wurde.

Solange der Zeiger 'innerhalb' der Textzeile steht, arbeitet 'SST' wie gewöhnlich, jedoch schickt ihn ein 'BST' von jedem Byte aus zurück zur Zeile 'STO 04'. Denken Sie daran, daß ein 'BST' den Prozessor veranlaßt, vom Programmanfang aus die Zeilen vorwärts zu zählen, wobei er es natürlich ablehnt, mitten in eine Mehrbyte-Zeile zu springen.

Wir werden gleich sehen, welch Nutzen sich aus dem Byte-Hüpfen ziehen läßt. Um die Anweisungen zukünftig etwas abzukürzen, möchte ich zunächst eine neue Vorschrift einführen: 'JUMP .ℓmn' soll ausdrücken: 'Sprung mit dem Byte-Hüpfer von Zeile ℓmn aus'. Ausführlich:

'JUMP .ℓmn' bedeutet:

1. GTO .ℓmn (selbst dann, wenn die angezeigte Zeilen-Nummer schon ℓmn ist)
2. PRGM aus
3. Byte-Hüpfer betätigen
4. PRGM an

'JUMP' ohne Zeilen-Nummer heißt 'Byte-Hüpfen' von der gegenwärtigen Zeile aus, Schritt 1 entfällt.

Versuchen Sie jetzt unter Verwendung von Routine (3B-1) 'JUMP .002'. Nach Schritt 4 sehen Sie '02 X<Y?'. Drücken Sie einmal die Korrekturtaste und dann 'SST'. Sie müssen daraufhin '02 "ABC⁻EFG"' sehen. Mit "⁻" wird in der Anzeige eine Null dargestellt, in diesem Fall die bei der Löschung des Bytes "D" eingesetzte Null. Sie haben eine Programmzeile verändert, ohne die ganze Zeile löschen zu müssen! Das ist nur der Anfang - ändern Sie nun die Zeile 01 der Routine in 'STO 03' ab. (Wenn Sie während des Edierens irgendwelche Fehler machen, durch die unsichtbare Nullen in die Routine gelangen, bringen Sie sie mit 'PACK' wieder hinaus. Stünde beispielsweise der Zeile 02 eine Null voran, was wir in der Anzeige nicht sehen könnten, würde 'JUMP .002' nichts bewirken, denn das zweite Nybble einer Null ist '0'.) Als nächstes abermals 'JUMP .002', woraufhin Sie '02 /' erblicken, nämlich das "C"-Byte. Tasten Sie '03 LBL 00' ein und drücken Sie 'GTO .002'. Die Anzeige wird '02 "ABC⍰EFG"' vorweisen. Sie haben die auf das Byte "C" folgende Null durch das von 'LBL 00' herrührende Byte '01' ersetzt, welches in der Anzeige als "⍰" ('vollständiger Mensch') wiedergegeben wird, wenn die volle Textzeile erscheint. Die folgende Reihe von Anweisungen bringt die rechts von ihr abgebildeten Programmschritte hervor:

```
JUMP .002                    01 STO 03
eintasten:                   02 "ABC␣_)]"
     03 LBL 11               03 LBL 00
     04 DEC          ⟶       04 X>Y?
     05 RCL 09                05 X<=Y?
     06 ACOS                  06 Σ+
GTO .002                         .END.
```

Dies ist Ihr erstes ernsthaftes Beispiel 'synthetischer Programmierung', bei dem Byte-Kombinationen, die durch normale Tastenfeldbedienung nicht erreichbar wären, mit außergewöhnlichen Hilfsmitteln zusammengesetzt worden sind. Gemäß den in Abschnitt 3A beschriebenen Editionsregeln auf Byte-Stufe mußte der Prozessor, als wir 'LBL 11' einzufügen versuchten, 7 Nullen dazwischensetzen, um Platz für die neue Zeile zu schaffen. Das brachte auch Raum für 'DEC', 'RCL 09' und 'ACOS', doch gleichzeitig wurden die Bytes "ŦEFG" im Speicher abwärts geschoben, zweifellos außerhalb des 'Einflußbereiches' von Byte 'F7'. Sie 'gerieten' so zu 'echten' Programmzeilen, 03-06.

Jede solche mit dem Byte-Hüpfer erzeugte synthetische Textzeile arbeitet ganz normal, wovon Sie sich durch Einzelschrittausführung der neuen Zeile 02 (PRGM aus) und anschließenden Blick ins Alpha-Register überzeugen können. Wir können dies Verfahren dazu benutzen, jedes der 19 nicht-tastbaren HP-41C Anzeige-Symbole sowie das Anhangs- und das Textsymbol (doch nicht die Gänse) in eine Programm-Textzeile zu bringen. Jedes dieser Zeichen findet man in der oberen Hälfte der Byte-Tabelle, so daß es, wie eben vorgeführt, mittelbar durch Eintasten der entsprechenden Ein-Byte-Funktion ausgegeben werden kann. Darüberhinaus kann man auch jedes der 128 Druckerzeichen (diejenigen ohne Anzeige-Symbol erscheinen als Explosionszeichen) in eine Textzeile bringen, um sie dem Druck-Puffer unter Verwendung von 'ACA' (siehe Abschnitt 6E) zu übergeben.

In synthetischen Programmen werden häufig Programmzeilen verwendet, die an im Alpha-Register stehende Zeichen-Ketten 'Nullen anhängen'. Hier ein Beispiel für die Erzeugung solch einer Zeile - in diesem Fall, um 5 Nullen anzuhängen:

eintasten:

```
        01 ASTO 02
        02 "ŀABCDE"

  JUMP
  DEL 005
  DEL 001
```

Die Zeile '02 "ŀABCDE"' ist gewählt worden, um soviele Zeichen zum Löschen zur Verfügung zu haben, wie Nullen angehängt werden sollen. Das 'DEL 005' verwandelt diese Zeichen in Nullen. Das 'DEL 001' räumt das zur Steuerung des Byte-Hüpfers benötigte 'ASTO 02' wieder fort.

Das bis jetzt beschriebene Edieren mit dem Byte-Hüpfer ist insofern eingeschränkt, als das *erste* Zeichen in einer Textzeile nicht umgewandelt werden kann, ausgenommen in eine durch Löschen entstehende Null. In "ABCDEFG" z.B. kann das "A" nicht abgeändert werden, denn um ein neues Byte an die Stelle des "A" zu bringen, muß das dem auszutauschenden "A" voranstehende Byte 'F7' angezeigt werden; dieses Byte aber veranlaßt die Anzeige, darauf zu beharren, die gesamte Textzeile abzubilden. Eingefügte Bytes gelangen dann nur hinter der vollständigen Textzeile ins Programm.

Eine Vervollkommnung des Byte-Hüpfer-Verfahrens gestattet uns jedoch, willkürliche

Textzeilen mit nicht-tastbaren Zeichen in beliebigen oder allen Positionen zu erzeugen. Angenommen, wir wollen die Textzeile "(╫)", also die Bytes 'F3 28 23 29', erzeugen. In den vorangehenden Beispielen war es üblich, die Text-Bytes mit 'vorläufigen' Zeichen, die dann durch die erwünschten Bytes aus der Zeile gestoßen wurden, zu erzeugen. Diesmal wird es das Text-Byte selbst sein, welches wir erzeugen, jedoch innerhalb einer anderen Textzeile. Beginnen Sie mit der folgenden Routine:

```
01 STO 01      31
02 "BH"        F2 42 48
```

(3B-2)

Die Zeile '01 STO 01' liefert uns das Nybble '1' für einen Byte-Sprung über genau ein Byte; wir werden eine solche Zeile als 'Überwacher' bezeichnen, weil sie die Länge des Byte-Sprunges steuert. Aus ähnlichen Gründen werden wir eine vorläufige Textzeile wie '02 "BH"' den 'Erzeuger' nennen. 'Überwacher' und 'Erzeuger' werden getilgt, wenn die Edition mit dem Byte-Hüpfer abgeschlossen ist. Führen Sie jetzt 'JUMP .002' aus, und tasten Sie Zeile '03 "ABC"' ein. Das vollständige Programm lautet dann:

```
01 STO 01      31
02 "B*"        F2 42 F3
03 -           41
04 *           42
05 /           43
06 Σ-          00 00 00 48
```

Der Eintrag von Zeile '03 "ABC"' setzt ein 'F3' gleich rechts hinter die '42', wo es zum letzten Byte des Erzeugers wird. Die Bytes '41 42 43' ("ABC") 'passen' nicht mehr in den Erzeuger und erscheinen daher als drei unabhängige Programmzeilen 03-05. Schließlich haben wir noch das Byte '48', das ursprüngliche "H", welches durch die Einfügung von "ABC" aus dem Erzeuger gedrängt und zur Zeile '06 Σ-' wurde. Die drei aus der Einfügung übriggebliebenen Nullen sind wie gewöhnlich unsichtbar.

Als nächstes drücken Sie 'GTO .002' und tasten

```
03 RCL 08
04 RCL 03
05 RCL 09
```

ein, um die Bytes '28 23 29' unmittelbar hinter das Byte 'F3' zu setzen. Dann 'JUMP .002' ausführen und '03 Σ-' eintasten. Jetzt hat das Programm die Gestalt:

```
01 STO 01      31
02 "BH"        F2 42 48
03 "(╫)"       00 00 00 00 00 00 F3 28 23 29
04 -           00 00 00 00 41
05 *           42
06 /           43
07 Σ-          00 00 00 48
```

Die Einfügung von 'Σ-' wirft das Byte 'F3' aus dem Erzeuger, woraufhin dieses seine Rolle als 'TEXT 3' wiederaufnimmt und die Bytes '28 23 29' ergreift, um die Textzeile mit den Zeichen "(", "#" bzw. ")" zu vervollständigen. Die verschiedenen Nullen sind von den 7er-Gruppen, die während der drei Einfügungen ins Programm gelangten, übriggeblieben. Um 'aufzuräumen', löschen wir die Zeilen 01, 02 und 04-07 und führen 'PACK' aus. Nach einiger Übung werden Sie feststellen, daß das ganze Verfahren ziemlich geschwind geht.

Das Edieren mit dem Byte-Hüpfer ist keineswegs auf Textzeilen beschränkt, insbesondere das zuletzt beschriebene Verfahren mit 'Überwacher' und 'Erzeuger'. Um ein vergnügliches Beispiel zu haben, versuchen Sie folgendes (denken Sie dabei an das in Abschnitt 1E, Punkt 9, beschriebene Befehlsformat): Sie beginnen wieder mit Routine 3B-2

```
                  JUMP .002     [02 *          ]
03 TONE 1
                  GTO .002      [02 "B✱"       ]
03 LBL 09
                  JUMP .002     [02 *          ]
03 Σ-
                  SST           [04 TONE 0     ]
```

'TONE 0' sieht unauffällig aus, doch lassen Sie ihn erklingen, indem Sie ihn als Einzelschritt (PRGM aus) ausführen. Sie werden einen sehr tiefen Ton von über 2 Sekunden Dauer hören!

Wir können mit dem Überwacher-Erzeuger-Verfahren fast jede gewünschte Zusammenstellung von Vor- und Nachsilben zur Bildung synthetischer Zwei-Byte-Funktionen zustande bringen. Die wichtigste Gruppe solcher Funktionen sind die Zugriffsfunktionen auf die Zustandsregister, die wir im folgenden Kapitel 4 untersuchen wollen.

Kapitel 4

DIE ZUSTANDSREGISTER

4A. Sonderbare Nachsilben

Am Schluß von Abschnitt 2B hatten wir die Byte-Tabelle in aller Ausführlichkeit untersucht und fast jeden Eintrag erklärt. Es gibt indessen noch immer ein wesentliches Merkmal, das zu erforschen bleibt: Beachten Sie, daß die Bytes 64 und 65 doppelt unterstrichene Nachsilben haben und daß jedes der Bytes 66 bis 6F zwei Nachsilben-'Werte' besitzt, von denen einer unterstrichen ist. Die unterstrichenen Nachsilben erscheinen bei normaler HP-41C-Programmierung nie. Betrachten wir die als Buchstaben "A" bis "J" dargestellten Nachsilben der Bytes 66 bis 6F. Sie tauchen nur in Programmzeilen, die 'lokale Alpha-Marken' wie 'LBL C' oder 'XEQ F' enthalten, auf. Die Tastenfeldlogik hindert uns daran, diese Nachsilben anderen Vorsilben anzugliedern. Wenn wir z.B. 'STO' drücken, wird die ALPHA-Taste unwirksam, so daß nur noch numerische Nachsilben eingetragen werden können. Doch der Byte-Hüpfer hat uns von den Zwängen des Tastenfeldes befreit; versuchen wir also, ein 'STO A' zu erreichen:

```
01 STO 01
02 "BH"
                JUMP .002          [02 *        ]
03 STO 22 (beliebige Nachsilbe)
                GTO .002           [02 "B*"     ]
03 X<0?
                JUMP .002          [02 *        ]
```

An dieser Stelle sind wir soweit, die Vorsilbe 'STO' aus dem Erzeuger drängen zu können. Wir wollen dabei gleich noch ein 'RCL A' zusammenbasteln:

```
03 RCL 22 (drängt das 'STO' aus dem Erzeuger,
           setzt das 'RCL' hinein)
                GTO .002           [02 "B*"     ]
03 X<0?
                JUMP .002          [02 *        ]
03 Σ-
```

Das Programm lautet jetzt:

```
01 STO 01
02 "BH"
03 RCL A
04 6
05 STO A
06 6
07 Σ-
```

(4A-1)

Die '6' in den Zeilen 04 und 06 ist die der Nachsilbe '22', Byte '16', entsprechende 'Ein-Byte-Funktion'. Führen Sie nun

```
                    SIZE 103
                    PRGM (aus)
          12345
                    GTO .005
                    SST
                    CLX          [0.0000      ]
```

aus. Nach dem letzten Schritt ist die Zahl '12345' aus der Anzeige verschwunden. Um sie wiederzugewinnen, drücken Sie 'GTO .003' und 'SST'. Die Zahl kehrt zurück, was beweist, daß Sie im Register 'A' gespeichert war. Um zu erkennen, wohin sie wirklich gelangte, setzen Sie '102' in R_{00} und führen 'VIEW IND 00' aus. 'STO A' ist, wie Sie wohl schon erwartet haben, gleichwertig mit 'STO 102'. Wenn die numerischen Nachsilben über dezimal 99 hinaus fortgesetzt werden, vertritt die Nachsilbe 'A' den Dezimalwert 102. Synthetische Programmierung erlaubt uns also, den unmittelbaren Zugriff auf Daten-Register bis nach R_{111} (Nachsilbe 'J') auszudehnen; die Unterstreichung weist darauf hin, daß nur synthetische Funktionen diese Nachsilben erreichen.

Doch warum ist bei 111 Schluß? Es gibt noch eine weitere Reihe von Nachsilben; könnten wir sie nicht benutzen, uns bis zum Register 127 direkten Zutritt zu verschaffen? Beachten Sie zunächst, daß die Nachsilben 70, 71, 72, 73 und 74 schon dem Zugriff auf die Stapelregister T, Z, Y, X und L vorbehalten sind. Doch die verbleibenden Bytes winken uns zu: sie erweisen sich als die Schlüssel zu 'Pandoras Faß' der synthetischen Programmierung! Wir wollen das durch ein dramatisches Beispiel belegen: ausgehend von Routine 4A-1:

```
                    JUMP .002    [02 *        ]
          03 STO 22
                    GTO .002     [02 "B*"     ]
          03 AVIEW
                    JUMP .002    [02 *        ]
          03 Σ-
                    GTO .003     [03 STO d    ]
```

Stellen Sie jetzt PRGM aus, und tasten Sie der Reihe nach

'SF 00, SF 01, SF 02, SF 03, SF 04, FIX 9, SF 28, GRAD, USER, CLX, ALPHA'

ein, um verschiedenartige Indikatoren der Anzeige einzuschalten. Und nun drücken Sie einmal 'SST'. Durch die entvölkerte Anzeige wird ganz klar, daß die einfache Funktion 'STO d', bei Null im Register X, alle 56 HP-41C-Flags mit einem grimmigen Streich dahinrafft. Die Schlußfolgerung ist augenfällig - der Zwei-Byte-Kode '91 7E' oder 'STO d', den wir mit dem Byte-Hüpfer herstellten, erlaubt uns unmittelbaren Zugang zu einem besonderen Register, welches wir 'Register d' nennen werden und welches die 56 Flags enthält.

Die Inhalte der am Schluß von Abschnitt 2D erwähnten 'Zustandsregister' (Register 000-00F) werden durch die Kartenleser-Funktion 'WSTS' aufgezeichnet. Der Zustand

aller Benutzer- und System-Flags ist Teil der auf Statuskarten abgelegten Information, woraus wir schließen dürfen, daß das Register d eines der 16 Zustandsregister ist. Die Beobachtung, daß die Stapelregister, auf die ebenfalls mit Hilfe von Nachsilben aus Zeile 7 der Byte-Tabelle zugegriffen wird, Zustandsregister sind, führt uns in einem kühnen Sprung von Erleuchtung zu der Vermutung, daß alle Nachsilben der Zeile 7 sich auf Zustandsregister beziehen - 16 Nachsilben, 16 Register. Es bleibt nur noch, die Rolle der 16 Register zu bestimmen. Die Funktionen 'WSTS' und 'WALL' lassen erwarten, daß die Zustandsregister außerdem das Alpha-Register, den Adreßzeiger, die Rücksprungadressen von Unterprogrammen, die jeweilige Datenregister-Anzahl und die Lage der Statistikregister enthalten.

Wie die Byte-Tabelle zeigt, erscheinen die Nachsilben 75 bis 7F in der Anzeige als 'M', 'N', 'O', 'P', 'Q', '⊢ ', 'a', 'b', 'c', 'd' bzw. 'e'. Die zweiten Nachsilben-Werte der Bytes 75 bis 7A sind die Zeichen, die der Drucker für die entsprechenden Zustandsregister-Funktionen verwendet. Die als 'STO M' angezeigte Zeile würde demgemäß als 'STO [' gedruckt. Die Beziehungen zwischen Anzeige- und Druckersymbol sind auch in Tabelle 1-1 aufgeführt.

Um die Eigenschaften und Nutzungsmöglichkeiten der Zustandsregister zu untersuchen, werden wir Programmzeilen zusammensetzen, die uns Einblick in diese Register gewähren oder Veränderung ihrer Inhalte erlauben. Die Funktion 'X<>' ist hervorragend für diesen Zweck geeignet, weil sie sowohl als 'STO' als auch als 'RCL' wirksam werden kann. Weil eine Veränderung der Registerinhalte von a, b und c gelegentlich unangenehme Folgen zeitigen kann ('0, STO c' verursacht das ärgerlichste Ergebnis: "MEMORY LOST"), werden wir uns vorläufig auf eine 'mündliche' Beschreibung dieser Register beschränken. Als Übung sollten Sie versuchen, selbständig den Byte-Hüpfer zu verwenden, um die Routine 4A-2 ohne Führung zu erstellen. Schauen Sie in die Anweisungen des nächsten Absatzes, falls Sie sich verirren. Wie bei der Erzeugung von 'STO A' und 'RCL A' werden wir uns Tastenarbeit ersparen, indem wir die Vorsilben 'X<>' eine nach der anderen dazu benutzen, die jeweils vorangehende aus dem Erzeuger zu treiben. Sofern Sie die in Kapitel 3 getätigte Tastenzuweisung von 'X<>' aufgehoben haben, sollten Sie sie jetzt erneuern.

Den Anfang bildet Routine 3B-2. Wie üblich ist die Wahl einer vorübergehenden Nachsilbe für die Vorsilbe 'X<>' willkürlich (doch nehmen Sie nicht gerade 'X<>29, 30 oder 31', weil diese Nachsilben gleichzeitig Vorsilben sind). Die beste Wahl ist '00', denn '00' ist auch hexadezimal '00', also das Null-Byte, woraus uns der Vorteil erwächst, daß alle zurückbleibenden Nachsilben 'unsichtbar' sind und durch Verdichten fortgeschafft werden können.

Fangen Sie nun an:

```
                        JUMP .002     [02 *        ]
03 X<>00
                        GTO .002      [02 "B*"     ]
03 CLD      (e)
                        JUMP .002     [02 *        ]
```

```
03 X<>00
                    GTO .002      [02 "B✱"    ]
03 AVIEW    (d)
                    JUMP .002     [02 *       ]
03 X<>00
                    GTO .002      [02 "B✱"    ]
03 SIGN     (⊢)
                    JUMP .002     [02 *       ]
03 X<>00
                    GTO .002      [02 "B✱"    ]
03 X≠Y?     (Q)
                    JUMP .002     [02 *       ]
03 X<>00
                    GTO .002      [02 "B✱"    ]
03 X=Y?     (P)
                    JUMP .002     [02 *       ]
03 X<>00
                    GTO .002      [02 "B✱"    ]
03 CLX      (O)
                    JUMP .002     [02 *       ]
03 X<>00
                    GTO .002      [02 "B✱"    ]
03 LAST X   (N)
                    JUMP .002     [02 *       ]
03 X<>00
                    GTO .002      [02 "B✱"    ]
03 RDN      (M)
                    JUMP .002     [02 *       ]
03 Σ-
```

Nach dem Abarbeiten dieser Folge verbleibt uns, wenn wir noch '11 Σ-' löschen, die Routine

```
01 STO 01      06 X<>P
02 "BH"        07 X<>Q
03 X<>M        08 X<>⊢
04 X<>N        09 X<>d
05 X<>O        10 X<>e
```

(4A-2)

Wenn wir nun z.B. 'X<>M' ausführen wollen, schalten wir PRGM aus, drücken 'GTO .003' und dann 'SST'. Im Rest dieses Kapitels werden wir die Programmzeilen der voranstehenden Routine dazu benutzen, die Zustandsregister zu erforschen. Die praktischen Anwendungen der eben entdeckten Eigenschaften auf das Programmieren werden in Kapitel 6 besprochen. Abbildung 4-1 ist ein Schaubild, das die Nutzung der verschiedenen Teile der Zustandsregister in ähnlicher Weise wie in Abbildung 2-5 übersichtlich zusammenstellt.

4B. Das Alpha-Register

Ein Register aus dem HP-41C-Arbeitsspeicher ist 7 Bytes lang. Das 'Alpha-Register' bildet eine Ausnahme von dieser Regel, weil es bis zu 24 Bytes (Alpha-Zeichen) aufneh-

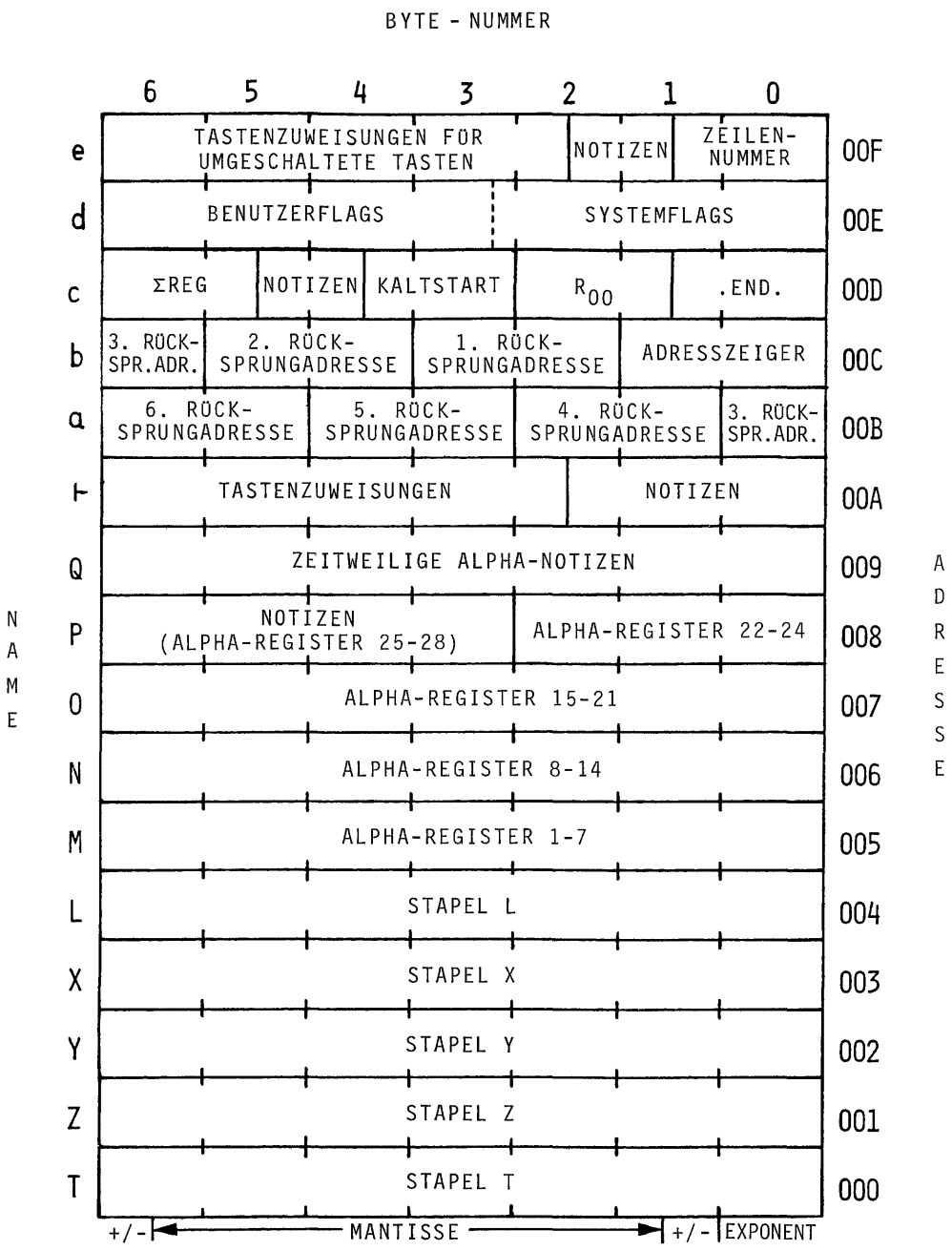


ABBILDUNG 4-1. DIE ZUSTANDSREGISTER

men kann. Tatsächlich besteht das Alpha-Register aus vier Zustandsregistern, den Registern M, N, O und P. 4 Register ergeben eine Summe von 28 Bytes; nur 24 Bytes davon können aber angezeigt oder mit 'ASTO' oder 'ARCL' umgesetzt werden.

Um den Aufbau des Alpha-Registers zu veranschaulichen, können wir Routine 4A-2 gebrauchen. Tasten Sie 24 Zeichen ins Alpha-Register, etwa

"ABCDEFGHIJKLMNOPQRSTUVWX".

Sofern Flag 26 gesetzt ist, hören Sie bei Eingabe von "X" einen Warnton, der Sie darauf aufmerksam macht, daß das Alpha-Register voll ist. Drücken Sie nun 'GTO .003', 'CLX', 'ALPHA (an)', 'SST', und Sie erblicken

"ABCDEFGHIJKLMNOPQ-------".

Die Überstreichung "-" ist das dem Null-Byte '00' entsprechende Anzeige-Symbol. Das 'CLX' hat Register X mit Nullen gefüllt; 'SST' führte 'X<>M' aus und brachte so die Nullen ins Register M, welches sich damit als dasjenige offenbart, welches die 7 äußersten rechten Bytes des Alpha-Registers enthält. Wenn Sie den ALPHA-Modus ausschalten und 'FIX 9' festsetzen, werden Sie '-2.5354555 E-42' im Register X sehen. Der Hexadezimalkode für die Zeichenkette "RSTUVWX" ist

'52 53 54 55 56 57 58'.

Diese Bytes stehen jetzt im Register X, wo der Prozessor tapfer versucht, eine Dezimalzahl anzuzeigen. Eine '5' bildet die Vorzeichen-Ziffer der Mantisse und eine '7' diejenige des Exponenten, so daß für beide Vorzeichen Minuszeichen entstehen (vgl. Abschnitt 5A). Die Mantissenziffern sind alle normale Dezimalziffern, woraus sich ihr Wert zu '2.535455565' ergibt; jedoch werden die letzten beiden Ziffern davon unterdrückt, um Platz für den Exponenten zu schaffen. Der Byte-Kode für einen negativen Exponenten '-xy' ist das (dezimale) Komplement des Exponenten: '100-xy'; im vorliegenden Fall also 42 = 100-58.

Fahren Sie mit Ihren Untersuchungen fort, indem Sie 'ALPHA (an)' und 'SST' tasten, um "ABCDEFGHIJRSTUVWX-------" festzustellen. 'SST' hat Zeile '04 X<>N' ausgeführt, so daß der ursprüngliche Inhalt von Register M, "RSTUVWX", ins Register N, welches die nächsten 7 Bytes des Alpha-Registers umfaßt, gebracht wurde. Um die Übung abzuschließen, tasten Sie 'ALPHA (aus)', 'GTO .003', 'SST' und 'ALPHA (an)'; die Anzeige gibt

"ABCDEFGHIJRSTUVWXKLMNOPQ"

wieder - die ursprünglichen Inhalte von M und N sind ausgetauscht worden. Alles in allem ist die ursprüngliche 24-Zeichen-Kette wie folgt aufgeteilt worden:

	xxxxABC	DEFGHIJ	KLMNOPQ	RSTUVWX
Register:	P	O	N	M

Wenn eine neue Zeichenkette ins Alpha-Register eingegeben wird, gelangt das erste eingetastete Zeichen ins letzte Byte (das Exponenten-Byte) des Registers M auf die Adresse 0005. Das nächste Zeichen geht ebenfalls nach 0005 und drückt dabei das vorherige Zeichen nach links ins Alpha-Register, also ins vorletzte Byte von Register M mit der Adresse 1005. Nachfolgende Zeichen setzen diesen Vorgang fort;

wenn Register M voll ist, wird beim nächsten Eintrag das erste auf der Adresse 6005 befindliche Zeichen in das letzte Byte des Registers N , Adresse 0006, geschoben, usw. bis u.U. in die Register O und P hinein. Wenn ein Zeichen ins vorvorletzte Byte von Register P, Adresse 2008, gelangt, ertönt der Warnton. Bei weiterer Eingabe werden die führenden Zeichen in die ersten 4 Bytes von Register P (oben mit "xxxx" angedeutet) abgedrängt, wobei sie endgültig aus der Anzeige verschwinden.

Drücken Sie jetzt 'GTO .006', schalten Sie in den Alpha-Modus, und hängen Sie die vier Zeichen "YZ=?" an die ursprünglichen 24 Stück an. "ABCD" verschwindet ganz, ist aber erstaunlicherweise noch im Register P zugegen: einmal 'SST' drücken, um 'X<>P' auszuführen, dann 'ALPHA (aus)', und Sie erblicken '-1.4243444 E-53'. Um diese Zahl in Zeichen zu übersetzen, tasten Sie 'GTO .003', 'ALPHA (an)', 'CLA', 'SST' ('X<>M'). Die Zeichen "ABCDEFG" kehren zurück - sie waren anfänglich der Inhalt von Register P, den wir auf dem Umweg über Register X nun ins Register M gebracht haben. Wenn wir den ganzen Vorgang wiederholen, indem wir erneut alle 28 Zeichen eintasten, diesmal jedoch den Alpha-Modus *vor* dem das 'X<>P' ausführenden 'SST' ausschalten, endigen wir mit "**CDEFG" . Der Prozessor benutzt gelegentlich die ersten 4 Bytes des Registers P für 'Notiz'-Zwecke, wobei er die ursprünglichen Inhalte dieser Bytes teilweise oder völlig tilgt. Offenkundig muß der Prozessor das Register P benutzen, wenn die ALPHA-Taste betätigt wird. Untersuchungen von Charles Close haben aufgedeckt, daß während eines Programmlaufs nur 'VIEW' und 'AVIEW' sowie Programmzeilen mit Zahlen-Eintrag den Verlust der führenden Bytes von P verursachen. Solange diese Befehle vermieden werden, können wir das gesamte 28-Byte Alpha-Register für Zeichen-Ketten-Verarbeitung ausnutzen. Bei der Aufzeichnung der gegenwärtigen Datenregister-Anzahl und der Lage der Statistik-Register auf Magnetkarte mit 'WSTS' z.B. verwendet der Prozessor die Bytes 1, 2 und 3 von P.

Es gibt zwei wichtige Anwendungsbereiche der Direktzugriffe auf die Register M, N, O und P. Erstens haben wir, wie in den vorangehenden Beispielen gezeigt wurde, eine neue Gruppe von Bearbeitungsverfahren für Zeichenketten gewonnen, die, wenn sie den herkömmlichen Befehlen 'ASTO', 'ARCL', 'APPEND' und 'ASHF' hinzugefügt wird, wirkungsvolles und schnelles Zeichen-Sortieren besorgt, was für die Handhabung der Anzeige, bei Spielen, zum Wörter-Umordnen usw. recht nützlich ist. Die zweite Anwendungsmöglichkeit ist der Gebrauch des Alpha-Registers als zusätzliche drei (oder gar vier, falls Register P dazukommt) Daten-Register mit denselben Eigenschaften wie gewöhnliche Daten-Register, doch dem Vorteil fester Speicherplatzzuordnung und nichtnormalisierten Rückrufes (vgl. Abschnitt 5B). Diese Anwendungen werden in den Kapiteln 5 und 6 im einzelnen untersucht.

4C. Register Q

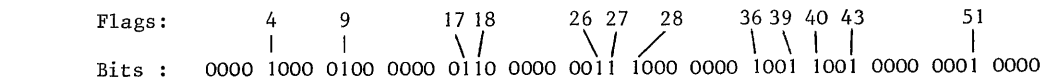
Register Q ist in erster Linie ein Notiz-Register für den Prozessor. Es wird so häufig benutzt, daß es als zusätzliches Daten-Register eigentlich unbrauchbar ist.

Für die synthetische Programmierung ist es hauptsächlich darum von Belang, weil es Alpha-Ketten, die nicht gleich ins Alpha-Register gelangen, vorübergehend aufnimmt. Solche Ketten entstehen während der Ausführung von Funktionen oder Programmen, die der Benutzer 'buchstabiert', oder während des Eintrags von Programm-Textzeilen. Benutzen Sie Routine 4A-2 z.B. folgendermaßen: 'XEQ' 'ALPHA' "GTO" 'ALPHA' '.007', 'SST', 'GTO .003', 'ALPHA', 'CLA', 'SST'. Das jetzt im Alpha-Register stehende "OTG" ist die Umkehrung der Buchstaben "G", "T" und "O", die Sie zur buchstabengetreuen Eingabe von "GTO" hinter 'XEQ' verwendet haben. Diese Eigenschaft, die Richtung zu wechseln, kann dazu ausgenutzt werden, die Erzeugung nicht-tastbarer Programm-Textzeilen zu vereinfachen (Abschnitt 5I).

4D. Das Flag-Register

Wir haben schon am Anfang dieses Kapitels entdeckt, daß Register d alle 56 HP-41C Benutzer- und System-Flags 'enthält'. Wenn wir uns daran erinnern, daß ein Register aus genau 56 Bits besteht, wird es augenfällig, daß *jedes der Flags gerade eines der Bits von Register d ist.* Das 'erste' (oder - abhängig von Ihrer persönlichen Vorstellung von einem Register - äußerste linke, höchste, wichtigste) Bit ist Flag 00, das zweite ist Flag 01 usw. bis zum letzten (56.) Bit, Flag Nr. 55.

Um ein Musterbeispiel für das Verhalten des Registers d zu haben, richten Sie Ihren HP-41C wie folgt ein: SF 04, SF 09, SF 17, SF 18, SF 26, USER (an), SF 28, FIX 9, RAD; alle anderen Benutzer-Flags gelöscht. Mit Routine 4A-2 können Sie jetzt die Inhalte des Registers d aufrufen: 'GTO .009', 'SST', 'ENTER', 'BST', 'SST', 'RDN', 'ALPHA', 'CLA', 'ARCL X'. Diese Befehlsfolge erlaubt uns, den Inhalt von Register d zu betrachten, ohne ihn zu verändern. Wir benutzen 'ARCL X', um alle 10 Mantissenziffern und die des Exponenten sehen zu können. Wenn wir berücksichtigen, daß die Vorzeichen-Ziffern positiver Exponenten und Mantissen den Wert Null haben, können wir aus der Alpha-Anzeige auf die Bytes im Register d schließen: '08 40 60 38 09 90 10', was ausgeschrieben und nach Nybbles gruppiert



ergibt. Jede '1' in dieser Reihe entspricht einem gesetzten Flag. Die erste '1' von links, im zweiten Nybble, ist z.B. Flag 04. Die nächste '1' ist Flag 09 usw. bis zur letzten '1', im zweiten Nybble von rechts, welche Flag 51, das 'SST'-Flag, bildet und nur vorübergehend gesetzt wurde, weil wir ein 'SST' ausführten, um 'X<>d' zu vollziehen. Damit Sie eine Kostprobe davon bekommen, was der Gebrauch des Flagregisters bringen kann, multiplizieren Sie die im Register X stehende Zahl '8.406038099 E10' mit '1 E30'. Führen Sie dann 'GTO .009' und 'SST' aus. Nein - Sie haben keine schwache Batterie, Sie haben nur gerade Flag 49, das Batterie-Anzeige-Flag, gesetzt. (Um es zu löschen, schalten Sie den HP-41C aus und wieder ein.) Dies ist ein Beispiel für die Beherrschung der Bytes von Register d durch den

Benutzer, denn die Zahlen, die Sie nach Register X brachten, gestatteten durch Registeraustausch die Überprüfung und Veränderung der System-Flags in vorbestimmter Weise. Stellen Sie sich jetzt den umgekehrten Vorgang vor - offene Kontrolle über die Benutzer-Flags, um beliebige Bytes im Register d zu erzeugen, von wo aus sie mit Zustandsregister-Zugriffsfunktionen ins Register X und anderswohin übertragen werden können (siehe Kapitel 5). Es war in der Tat die Durchführung dieses Planes, die ursprünglich die Entwicklung der ernsthaften synthetischen Programmierung einleitete.

4E. Die Tastenzuweisungsflags

Sobald eine Taste im USER-Modus gedrückt wird und dieser Taste eine andere als die Säumnis-Funktion (die 'USER-freie' Funktion) zugewiesen ist, muß der Prozessor die globalen Benutzer-Marken und die Tastenzuweisungsregister prüfen, um festzustellen, welches Programm oder welche Funktion auszuführen beabsichtigt ist. Um sich eine Menge fruchtlosen Suchens zu ersparen, unterhält der HP-41C 72 'Tastenzuweisungsflags', eines für jede Taste und jede Umschalttaste (hierbei zählt die ENTER-Taste doppelt). Wenn eine Taste im USER-Modus gedrückt wird, prüft der Prozessor zunächst das entsprechende Zuweisungsflag. Nur wenn es gesetzt ist, beginnt die Suche nach der Zuweisung.

Wie im Register d wird für jedes Zuweisungsflag ein Bit des Speichers verwendet. Weil 72 Bits nicht in ein einzelnes Register passen, werden die Zuweisungsflags zwischen den Registern ⊢ und e aufgeteilt. Die Flags für die einfachen Tasten sind die ersten 36 Bits des Registers ⊢ ; die Flags der umgeschalteten Tasten liegen gleichartig im Register e. Abbildung 4-2 zeigt die Beziehung zwischen Bit-Nummern und Tasten.

Um eines dieser Register einmal 'tätig' zu sehen, wollen wir Register ⊢ vorführen. Die einzigen Tastenzuweisungen, die wir bisher getätigt haben, wurden in Kapitel 3 gemacht; nämlich die Zuweisung des Byte-Hüpfers auf die Taste 'Σ+' und die der Funktion 'X<>' auf die Taste '+'. Wenn Sie in der Zwischenzeit zusätzliche Zuweisungen vorgenommen haben sollten, wird Ihr Ergebnis von dem, was gezeigt werden soll, abweichen, so daß die Löschung Ihrer nachträglichen Zuweisungen zu empfehlen ist.

Um den Inhalt von Register ⊢ mit Hilfe der Routine 4A-2 betrachten zu können, drücken Sie 'GTO .008', 'CLX', 'SST', 'FIX 7'. Sie müssen dann '0.0000021' sehen. Die 6 Nullen und das positive Vorzeichen machen sichtbar, daß die ersten 7×4 = 28 Bits der angezeigten Zahl (die den Inhalt des Registers ⊢ wiedergibt) auf Null gesetzt sind. Die Ziffer '2', binär 0010, zeigt, daß das Zuweisungsbit Nr. 31 gesetzt ist; die '1', binär 0001, stammt vom Zuweisungsbit Nr. 36 her. Wenn Sie sich Abbildung 4-2 ansehen, erkennen Sie, daß diese Bits gerade den Tasten '+' und 'Σ+', die wir mit Zuweisungen versehen hatten, entsprechen.

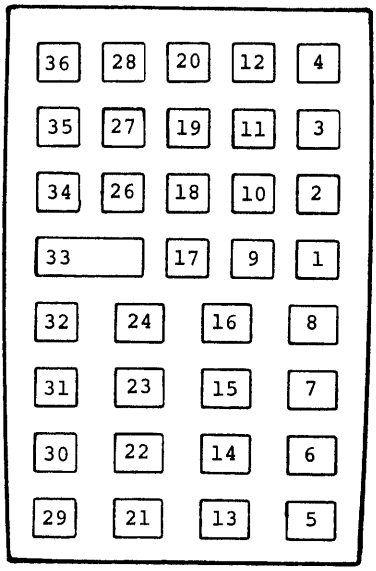


ABBILDUNG 4-2. DIE BITS DER TASTENZUWEISUNGSFLAGS

Die Überprüfung der Inhalte der Register ⊢ und e verschafft uns die Möglichkeit, ohne Einsatz des Druckers schnell festzustellen, welchen USER-Tasten Zuweisungen erteilt wurden. Wir haben 'FIX 7' festgesetzt, weil wir für diesen Zweck nur an den ersten 9 Nybbles des Registers ⊢ interessiert waren. Dieser Trick läßt sich nicht ganz allgemein anwenden - wenn zuviele Zuweisungen vorgenommen wurden, können die Zahlen, die aus 'RCL ⊢' oder 'RCL e' entstehen, hexadezimale Ziffern A-F enthalten, deren Anzeige-Bild schwierig zu entziffern sein kann (siehe Abschnitt 5A). Sie müssen dafür sorgen, daß die ursprünglichen Werte in die Register ⊢ und e zurückgespeichert werden; andernfalls verlieren Sie die Tastenzuweisungen einschließlich des Teiles des Arbeitsspeichers, der dazu benutzt wird, die Zuweisungen zu verschlüsseln. Wenn Sie jetzt die Taste '+' im USER-Modus drücken, führt der HP-41C '+' statt 'X<>' aus, weil die Null, die wir mit 'X<>⊢' ins Register ⊢ brachten, sämtliche Tastenzuweisungsflags gelöscht hat. Um sie wiederzuerlangen, drücken Sie (u.U. 'LASTX', falls Sie '+' tatsächlich ausgeführt haben) 'GTO .008' und 'SST'. Wenn Sie beim Hantieren mit den verschiedenen Befehlen den Inhalt von Register ⊢ oder e zufällig verloren haben, führen Sie über den Kartenleser ein 'WSTS' aus und lesen Sie die so erhaltene Statuskarte gleich wieder ein; Sie gewinnen dann alle ursprünglichen Tastenzuweisungen zurück.

Die letzten 3 Nybbles von Register e bilden den Speicherplatz für die (hexadezimal verschlüsselte) Programmzeilen-Nummer. Wenn die Nybbles der Zeilen-Nummer '000'

sind, wie nach einem 'GTO .000', einem von Hand ausgeführten 'RTN' oder einem mit 'END' abgeschlossenen Programmlauf, erscheint in der Anzeige '00 REG &mn'. Sobald ein Programm an einer anderen Stelle als dem 'END' anhält, wird die Zeilen-Nummer auf 'FFF' gesetzt. 'Erblickt' der Prozessor diese erdichtete Nummer, weiß er, daß er die tatsächliche Zeilen-Nummer neu errechnen muß, bevor er die gegenwärtige Programmzeile im PRGM-Modus oder nach Ausführung eines 'SST' vorweisen kann.

4F. Der Adreßzeiger und der Rücksprungstapel

In Kapitel 2 haben wir erfahren, daß sich der Adreßzeiger einer 4-ziffrigen Adresse bedient, die aus einer Byte-Nummer und einer 3-ziffrigen (hexadezimalen) Register-Nummer zusammengesetzt ist. Der Adreßzeiger selbst besteht aus den letzten 4 Nybbles von Register b. D.h.: wenn der Adreßzeiger auf Byte 'n' des Registers 'abc' gesetzt ist, dann enthält Register b die Zahl '0000000000nabc'. Wenn ein Unterprogramm aufgerufen wird, wird die Rücksprungadresse in den nächsten vier Nybbles links vom Adreßzeiger aufgezeichnet, während die neue laufende Adresse, dem Sprung des Unterprogrammes folgend, in die Nybbles des Zeigers gelangt. Beim Aufruf weiterer Unterprogramme werden die vorangehenden Rücksprungadressen nach links geschoben. Wenn Register b voll ist, gelangen die Adressen weiter ins Register a, so daß in den Registern a und b zusammen genügend Platz für die laufende und sechs offene Rücksprungadressen ist. Die Inhalte der Register haben stets die folgende Form:

yzäö uvwx qrst mn	op ijkl efgh abcd
Register a	Register b

worin 'efgh' die zuletzt aufgezeichnete Rücksprungadresse ist, 'ijkl' die vorletzte usw. Sobald ein 'RTN' oder 'END' ausgeführt wird, bewegt sich der 'Stapel' der Rücksprungadressen nach rechts, so daß die letzte Rücksprungadresse zum neuen Adreßzeiger wird, die vorletzte Rücksprungadresse wird zur letzten usw. Das einzig Verzwickte an diesem bezaubernden Entwurf ist die Tatsache, daß die Rücksprungadressen in einem Format geschrieben werden, welches leicht von dem des Adreßzeigers abweicht. Wenn z.B. ein Rücksprung auf die Adresse '3160' erfolgen soll, wird die Adresse im Rückkehrstapel als '0760' gespeichert. Die Gestalt '0760' ist tatsächlich nichts anderes als eine verdichtete Form von '3160'. Schreibt man alle Bits von '3160' auf, erhält man:

3160 = 0011 0001 0110 0000

In den Programmadressen des Benutzers (also an Speicherstellen im HP-41C selbst oder in Speichererweiterungsmodulen) sind das erste, das fünfte, das sechste und das siebente Bit immer Null, weil das erste Nybble nur Werte bis höchstens 6 (binär 0110) annehmen kann und das zweite Nybble stets 0000 oder 0001 ist. Dementsprechend werden in Rücksprungadressen die drei 'benutzten' Bits des ersten Nybbles in die drei 'unbenutzten' Bits des zweiten Nybbles gerückt. Die '3160' wird so zu:

0760 = 0000 0111 0110 0000

Die Zusammendrängung der Adresse macht das erste Nybble für die Speicherung zusätzlicher Information frei. Im einzelnen gilt: ist das erste Nybble Null, erfolgt der Rücksprung auf eine Programmadresse des Benutzers; ist es hingegen ungleich Null, weiß der Prozessor, daß er auf eine in einem Peripheriegerät ansässige Adresse zurückspringen muß. Z.B. beginnen alle Adressen des Speichers im Drucker mit dem Nybble '0110'.

Die nächstliegende Anwendung unserer Kenntnisse über den Aufbau des Rücksprungstapels und der synthetisch erzeugten Zugriffsmöglichkeiten auf den Stapel mit Funktionen wie 'STO b' oder 'RCL a' ist die, den Adreßzeiger auf beliebige Stellen im Speicher zu rücken, eingeschlossen die Tastenzuweisungsregister oder gar die Zustandsregister selbst. Bei Verwendung eines 'STO b' sind wir außerdem in der Lage, den Zeiger unmittelbar auf jedes Byte einer Mehrbyte-Programmzeile zu setzen, um auf diese Weise so wie mit dem Byte-Hüpfer zu edieren. Zur vollständigen Kontrolle über solche Vorgänge müssen wir allerdings erst noch eine Möglichkeit entwickeln, die uns erlaubt, beliebige hexadezimale 7-Byte-Kodes zu erzeugen und zu entziffern. Diese Fertigkeit werden wir in Kapitel 5 erwerben.

4G. Register c und Speicheraufteilung

Der letzte unerforschte Abschnitt des HP-41C Arbeitsspeichers ist Register c, welches mit aufregenden Nybbles und Bytes angefüllt ist. Die 14 hexadezimalen Ziffern dieses Registers sind wie folgt geordnet:

stu|vw|169|mno|pqr

Hierin sind die Ziffern, vertreten durch Buchstaben, so gruppiert, wie sie der Prozessor verarbeitet. Die Buchstaben sind so gewählt, daß sie mit denen der Speicheraufteilung in Abbildung 2-5 übereinstimmen. Die ersten drei Ziffern, 'stu', bilden die absolute Registeradresse des ersten der sechs durch die Funktion 'ΣREG' festgelegten Statistik-Register. Diese Adresse ändert sich jedesmal, wenn 'ΣREG' oder 'SIZE' ausgeführt wird. Ruft man 'Σ+', 'Σ-', 'CLΣ', 'SDEV' oder 'MEAN' auf, unterrichtet sich der Prozessor bei den Ziffern 'stu' darüber, welche Register augenblicklich zu Statistik-Registern ernannt sind.

Die nächsten zwei Ziffern, 'vw', werden vom Drucker für Notizzwecke benutzt. Die Ziffern Nr. 6, 7 und 8, die den expliziten Wert '169' haben, sind sehr interessant, freilich auf abartige Weise. Sie bilden die sog. 'Kaltstart-Konstante'. Der Prozessor prüft bei verschiedenen Gelegenheiten während des üblichen Betriebes, insbesondere beim Einschalten des HP-41C, ob diese drei Ziffern mit dem festen Wert '169' übereinstimmen. Stellt er eine Abweichung fest, nimmt er an, daß dem Speicher etwas Grausiges zugestoßen sei, und vollzieht dann einen unwiderruflichen Schritt: er löscht den gesamten Speicher und läßt "MEMORY LOST" anzeigen. So wird verständlich,

warum '0, STO c' die Speicherinhalte verloren gehen läßt: dieser Befehl tilgt die '169'.

Die Ziffern 'mno' und 'pqr' sind 3-ziffrige Register-Nummern, die den in Abbildung 2-5 gezeigten Registern 'mno' und 'pqr' entsprechen. 'mno' ist die absolute Adresse des mit der niedrigsten Nummer versehenen Daten-Registers, Register R_{00}. Der Prozessor muß sich bei jedem Aufruf eines Daten-Registers nach dieser Zahl richten. Das Daten-Register R_{ab} mit der relativen Adresse 'ab' ist das Speicher-Register mit der absoluten Adresse 'mno + ab' ('ab' in hexadezimale Darstellung umgesetzt).

Die Ziffern 'pqr' zeigen die augenblickliche Lage des ständigen '.END.' an. Weil die Kette der globalen Marken und END's beim '.END.' beginnt, fängt die durch ein 'GTO α' oder 'XEQ α' ausgelöste Suche im Register 'pqr' an.

Trotz der Gefahr, beim Umgang mit Register c die Speicherbelegung zu verlieren, gibt es viele wichtige Anwendungen, bei denen mit dem Inhalt von c hantiert wird; darunter vor allem die Kontrolle der Trennlinie zwischen Programm- und Datenspeicher und damit die Möglichkeit, Daten in Programm-Register zu bringen, was in Kapitel 5 beschrieben wird.

Kapitel 5

PROGRAMME ZUM PROGRAMMIEREN

Im Mittelpunkt dieses Kapitels steht die Entwicklung einer Reihe von HP-41C Programmen, die uns erlauben, beliebige hexadezimale 7-Byte-Kodes zu erzeugen, zu entziffern, abzuspeichern und zurückzurufen, woraus uns eine große Hilfe bei der synthetischen Programmierung erwächst. Die 'programmierenden Programme' sind dazu gedacht, bestimmte Aufgaben der synthetischen Programmierung zu erfüllen und gleichzeitig als Beispiele für die Benutzung synthetischer Funktionen zu dienen. Auch wenn Sie nicht jedes Programm dieses Kapitels brauchen werden, wird es für Sie lehrreich sein, die in den Routinen enthaltenen Techniken und Programmierungsmethoden sorgsam zu lesen. Es wird von jetzt an vorausgesetzt, daß Sie das in den Kapiteln 3 und 4 ausgearbeitete Byte-Hüpfer-Verfahren hinreichend gut beherrschen, um jede 2-Byte-Funktion wie etwa 'RCL M' oder 'X<>d' einzutasten, wenn es für ein Programm erforderlich ist. Wie schon bei der Erzeugung der Routine 4A-2 werden Sie es erleichternd finden, eine Reihe von synthetischen Funktionen Byte-hüpfend dadurch herzustellen, daß Sie sich vom Programmende ausgehend nach oben durcharbeiten und dabei jede in den Byte-Hüpfer gelangende Vorsilbe dazu benutzen, die vorangehende 'hinauszuwerfen'. Sobald alle synthetischen Funktionen gebildet sind, können Sie die normalen Funktionen auf herkömmliche Weise dazwischentasten.

In den Programmen werden drei entscheidende Techniken der synthetischen Programmierung vorgestellt:

1. *Mehrfachbenutzung der Flags.* Unmittelbarer Zugriff auf Register d erlaubt es, eine Anzahl von Benutzerflags für verschiedene Absichten gleichzeitig zu verwenden, indem man den vorliegenden Flagzustand mit 'RCL d' oder 'X<>d' sichert, sich der Flags für einen zweiten Zweck bedient und anschließend den ursprünglichen Zustand mit 'STO d' oder 'X<>d' wiederherstellt. Dies gestattet z.B. einem Programm, voneinander abweichende Anzeigeformate und trigonometrische Modi einzusetzen und am Schluß zu dem vom Benutzer bevorzugten Flagzustand zurückzukehren.

2. *Verwendung von Flags als Bits.* Wegen der uneingeschränkten Kontrolle über die Benutzerflags 0-29 gelingt die programmierbare Umwandlung binärer Zahlen von 30 Bit Länge in und aus dezimaler, oktaler und hexadezimaler Darstellung. Diese Fertigkeit ist ein unentbehrliches Werkzeug für die automatische Erzeugung und Entzifferung von Speicher-Bytes und Mehrbyte-Kodes.

3. Bearbeitungsverfahren für Zeichenketten. Die leistungsfähigen synthetischen Alpha-Register-Zugriffsfunktionen, welche in den Programmen dieses Kapitels dazu verwendet werden, einzelne Bytes zu 7-Byte-Kodes zusammenzusetzen oder solche Verbindungen auseinanderzupflücken, begegnen uns, wie wir sehen werden, im Alltag der synthetischen Programmierung immer wieder.

Der HP-41C ist dazu bestimmt, nur solche vom Programm oder vom Benutzer angebotenen Zahlen zu verarbeiten, die in normaler Dezimaldarstellung vorliegen. Wir dürfen daher erwarten, daß die Aufforderung an den Prozessor, Zahlen mit 'A' bis 'F' als Ziffern zu behandeln, mindestens einiges unerwartete Verhalten bewirkt. Bevor wir an die versprochenen Programme gehen, sollten wir darum erst lernen, wie die verschiedenen 7-Byte-Kodes von der Anzeige 'gedeutet' werden, und auch noch untersuchen, auf welche Weise Register-Befehle wie 'STO' und 'RCL' die Kodes beeinflussen.

5A. Mißratene Anzeigen

Wenn dem Prozessor eine Byte-Folge zum Anzeigen (PRGM aus) vorgelegt wird, muß er zunächst entscheiden, ob die Bytes als Zahl oder als Kette von Alpha-Zeichen anzuzeigen sind. Befindet sich der Rechner im Alpha-Modus oder ist ein 'AVIEW' befohlen, gibt es keine Wahl: alle Bytes des Alpha-Registers werden als Zeichen vorgewiesen. Doch wenn ALPHA aus ist, so daß in der Anzeige der Inhalt eines Daten-Registers steht, wird die Wahl durch das Vorzeichen der Mantisse bestimmt. Wir wissen schon, daß bei den Vorzeichen-Ziffern '0' (0000) und '9' (1001) die Registerinhalte als positive bzw. negative Zahlen abgebildet werden. Die einzige weitere 'normale' Vorzeichen-Ziffer ist die '1', bei der angenommen wird, daß das Register eine 6 Byte lange Kette aus Alpha-Zeichen, wie sie beispielsweise von einem 'ASTO'-Befehl herrühren kann, enthält. Jedes der verbleibenden 6 Bytes des Registers wird dann als Zeichen dargestellt; das erste Byte wird unterdrückt. Alle anderen Vorzeichen-Ziffern ('2' bis '8') veranlassen den Prozessor, den Registerinhalt als negative Zahl anzuzeigen, jedoch mit der Besonderheit, sie in vielen arithmetischen Operationen als "ALPHA DATA" zu behandeln.

Wenn der vorgelegte Registerinhalt Alpha-Daten enthält, worauf eine Vorzeichen-Ziffer '1' hinweist, wird die Anzeige durch Unterdrückung der Null-Bytes vereinfacht. Alle Null-Bytes werden dabei im Gegensatz zu einer mit Nullen durchsetzten Alpha-Register-Anzeige nicht nur zu Leerstellen 'erklärt', sondern beim Anzeigen völlig entfernt, indem alle anderen Zeichen zusammenrücken. Z.B. würden die mit '10 00 41 00 00 42 00' verschlüsselten Alpha-Daten *nach einem 'ARCL'* (vgl. dazu auch Abschnitt 5B) im Alpha-Register als "A--B-" erscheinen (führende Null-Bytes werden auch hierbei unterdrückt), bei einer Anzeige als Inhalt des Registers X jedoch einfach zu "AB" zusammenschrumpfen.

Zahlen-Anzeigen haben das zusätzliche Merkmal, von dem vom Benutzer gewählten Format, 'FIX' mit entsprechenden Rundungseffekten, 'ENG' und 'SCI', abhängig zu sein. Wir sollten uns jedoch dessen bewußt sein, daß diese Wahl nur die Anzeige beein-

flußt - die Zahl selbst bleibt als vollständiger 7-Byte-Kode unversehrt gespeichert.

In vielen Fällen, in denen Zahlen irgendwelche der Ziffern 'A' bis 'F' enthalten, tauchen bei der Anzeige dennoch nur gewöhnliche Dezimalziffern '0' bis '9' auf. Weil nämlich 'A' bis 'F' alle größer als '9' sind, findet bei jeder solchen Mantissenziffer der 'Übertrag' einer '1' in die zur Linken liegende Ziffer statt. Beispielsweise 'sollten' die Bytes '01 03 B0 0F 00 00 00' als die Zahl '1.03B00F' erscheinen, tatsächlich jedoch sieht man sie im 'FIX 6' Format als '1.041015'. Die Ziffer F ist, zwei Stellen verwendend, zur '15' geworden. Das B wurde zur '11' und der sich daraus ergebende Übertrag der zur Linken vorhandenen 3 zugeschlagen; '3B' wird somit als '41' angezeigt.

Eine Ausnahme von dieser allgemeinen Regel tritt ein, wenn die betreffende Zahl einen nicht-negativen Exponenten hat *und* in einem Format mit allen 10 Mantissenziffern gezeigt werden soll. Beide Bedingungen können nur für Exponenten zwischen 00 und 09 erfüllt werden. Wenn wir zum Format 'FIX 9' übergehen, erscheint die Zahl '1.03B00F' als '1.03;00?000'. Die Ziffern B und F werden als Einzelzeichen durch die Symbole ";" bzw. "?" wiedergegeben. Diese Zeichen findet man ebenso wie die Dezimalzahl-Zeichen "0" bis "9" in Zeile 3 der Byte-Tabelle. Entsprechend werden die Ziffern 'C', 'D' und 'E' in der Anzeige als "<", "=" bzw. ">" abgebildet. Eine Ziffer 'A' hingegen wird durch das in der linken unteren Ecke des Tabellenkästchens von Byte 3A eingetragene Explosionszeichen dargestellt. Wenn eine Zahl mit 'ARCL' ins Alpha-Register kopiert wird, bleiben die Sonderzeichen erhalten, mit Ausnahme der Explosion, die zum Alpha-Zeichen ":" wird.

Zahlen wie '1.03B00F' mit dem Exponenten Null werden mit 10 Mantissenziffern nur im FIX 9 Format angezeigt. Hätten wir '10.3B00F' ('1.03B00F E01') vorliegen, ergäbe sich eine Anzeige mit Sonderzeichen sowohl für FIX 8 wie für FIX 9. Allgemein erhalten wir für n = 9 - Exp. eine Sonderzeichen-Anzeige in den Formaten FIX 'n' bis FIX 9. Sobald der Exponent selbst irgendwelche Ziffern A-F enthält, werden die entsprechenden Sonderzeichen zu seiner Darstellung verwendet (das gilt in diesem Fall jedoch nicht für die Mantissenziffern, weil 10 Mantissenziffern nicht zusammen mit einem Exponenten angezeigt werden können.)

5B. Registertausch und Normaldarstellung

Weil wir uns auf den Umgang mit Daten-Registern, die beliebige 7-Byte-Kodes enthalten, vorbereiten, wollen wir eine Einteilung der Kodes vornehmen:

1. jeder 7-Byte-Kode, in welchem das erste Nybble '0001' ist, sei *'Alpha-Daten-Kette'* genannt.
2. jeder Kode, dessen erstes und zwölftes Nybble entweder '0000' oder '1001' lautet und dessen restliche zwölf Nybbles aus einer der Dezimalziffern '0' bis '9' bestehen, heiße *'Zahl'*.
3. jeder andere Kode soll von nun an als *'nicht-normalisierte Zahl'* oder kürzer 'NNZ' bezeichnet werden.

Daß diese Einteilung sinnvoll ist, sieht man leicht ein, denn die Register-Funktionen 'STO', 'RCL', 'X<>' und 'VIEW' greifen gerade auf 7-Byte-Kodes zu. 'STO' ist die einfachste von ihnen - 'STO mn', in der sich 'mn' auf *irgendein* direkt oder indirekt adressiertes Register bezieht, kopiert genau den Inhalt des Registers X in das angesprochene Register. Die anderen drei Register-Funktionen sind der synthetischen Programmierung leider nicht so wohlgesonnen. 'RCL pq', 'X<>pq' und 'VIEW pq', in denen 'pq' irgendeine Daten-Register-Nummer ist, bewirken, daß der Inhalt von R_{pq} vor dem Kopieren normalisiert wird. (Beim Austausch zwischen Zustandsregistern geschieht dies nicht.) Unter 'Normalisierung' verstehen wir, daß NNZ's verändert werden, und zwar entweder in gewöhnliche Dezimalzahlen (die keine ketzerischen Ziffern mehr, größer als 9, enthalten), wenn das ursprüngliche Mantissen-Vorzeichen 0 oder 9 war, oder in Alpha-Daten mit der Vorzeichen-Ziffer 1, wenn die ursprüngliche Vorzeichen-Ziffer zwischen 2 und 8 lag. Eine aus den Bytes '01 0C 00 D0 0E 00 FF' bestehende NNZ beispielsweise wird als '1.1201301 E??' angezeigt und nach 'STO 01', 'RCL 01' zu '1.120130140 E-35' normalisiert. Die ein regelwidriges Vorzeichen enthaltende NNZ '21 0C 42 34 7E 40 DD' erscheint als '-1.1242348 E==', Nach Normalisierung wechselt sie zur Alpha-Daten-Darstellung "µB4Σ@*" mit den Bytes '11 0C 42 34 7E 40 DD' über.

'ASTO' und 'ARCL' bewerkstelligen indessen eine andere Art von Registerwechsel. 'ASTO' nimmt die ersten 6 Bytes des Alpha-Registers, beginnend mit dem ersten Nicht-Null-Byte, her, setzt das Alpha-Ketten-Merkmal '10' davor und legt den so entstandenen 7-Byte-Kode im angesprochenen Register ab. Ein 'ARCL' wiederum kehrt diesen Vorgang um, wenn das angesprochene Register Alpha-Daten enthält, indem es das Byte '10' fallen läßt und die Alpha-Zeichen an das rechte Ende der im Alpha-Register befindlichen Alpha-Kette anhängt. Führende Nullen in den Alpha-Daten entfallen dabei; nachfolgende und dazwischenliegende hingegen bleiben erhalten. Beginnt man z.B. mit "ABC" im Alpha-Register und '10 44 45 46 47 48 49' ("DEFGHI") im X-Register, dann führt 'ARCL X' zu "ABCDEFGHI". Enthält das Y-Register den Kode '10 00 4A 00 00 4B 00' (angezeigt als "JK"), liefert ein anschließendes 'ARCL Y' die Alpha-Kette "ABCDEFGHIJ‾‾K‾". Falls schließlich das mit 'ARCL' angesprochene Register eine Zahl oder eine NNZ enthält, kopiert der Befehl nicht einfach die Bytes der Zahl ins Alpha-Register, sondern ändert vielmehr, wie es bei einer Zahlen-Anzeige geschieht, jede Ziffer in das der Zeile 3 der Byte-Tabelle entsprechende Anzeigesymbol ab. 'ARCL' normalisiert ebenfalls den Inhalt des angesprochenen Registers (außer bei Zustandsregistern).

5C. Der Start gelingt: "CODE"

Der Byte-Hüpfer ist bislang das einzige Werkzeug, welches wir zur Erzeugung ungewöhnlicher Programmzeilen und NNZ's (aus 'RCL M' usw. herrührend) zur Verfügung haben. Er läßt sich jedoch nur von Hand betätigen und ist auch bezüglich der erreichbaren Kodes beschränkt (Bytes aus der unteren Tabellenhälfte sind schwierig zu handhaben). Wir werden daher jetzt ein Programm mit Namen "CODE" schreiben, welches *irgendei-*

nen vom Benutzer bestimmten 7-Byte-Kode automatisch herstellt und in den beiden Registern X und M ablegt. Dieses Programm, zusammen mit der Routine "REG" (welche die Ablage des gewonnenen Kodes in einem beliebigen Register erlaubt), wird uns befähigen, jede gewünschte Folge von Programm-Bytes, NNZ's oder außergewöhnlichen Alpha-Zeichen-Ketten zu erzeugen. Fernerhin werden wir in der Lage sein, 'synthetische Tastenzuweisungen' vorzunehmen, die beliebige Zwei-Byte-Funktionen auf Benutzer-Tasten legen. Diese Möglichkeit wird die synthetischen Funktionen schließlich den normalen HP-41C- oder Peripherie-Funktionen völlig gleichstellen, denn wir werden sie mit einfachem Tastendruck von Hand ausführen oder an beliebiger Stelle als Programmbefehl einsetzen können.

"CODE" ist so entworfen, daß es mehreren Anforderungen genügt. Nach dem Aufruf sollte das Programm selbständig anhalten und den Benutzer zur Eingabe einfach einzutastender Kodes, die eine 14-ziffrige Hexadezimalzahl bestimmen, auffordern. Nach der Eingabe sollte es dann ohne weitere Eingriffe ablaufen und die gewünschte Zahl durch die zugehörigen Bytes verschlüsselt abliefern. Obwohl das Flag-Register zur Byte-Erzeugung benötigt wird, sollte das Programm den ursprünglichen Flagzustand am Ende wiederherstellen. Und endlich sollte "CODE" aus Gründen, die erst später (Abschnitt 6G) klar werden, keine gewöhnlichen Daten-Register benutzen. Es ist ziemlich schwierig, alle diese Forderungen zu berücksichtigen - die hier beschriebene Fassung von "CODE" ist aus vielen Verbesserungen der ursprünglichen Form (siehe 'HP-41C Black Box Programms', *PPC Calculator Journal*, V6N8P29) hervorgegangen.

```
01♦LBL "CODE"     23 FS? 07         45 FC?C 15        67 RCL J
02 "CODE=?"       24 CHS            46 SF 15          68 RCL \
03 AON            25 FS? 06         47 FS? 15         69 RCL [
04 STOP           26 CHS            48 CHS            70 STO \
05 AOFF           27 SF 06          49 FS? 14         71 R↑
06♦LBL "CO"       28 X<0?           50 CHS            72 STO [
07 "┣ABCDEFG"     29 CF 06          51 SF 14          73 "┣*"
08 .006           30 X<0?           52 X<0?           74 X<> \
09 STO L          31 SF 05          53 CF 14          75 R↑
10♦LBL "HB"       32♦LBL 01         54 X<0?           76 STO J
11 1              33 ST* X          55 SF 13          77 R↑
12 RCL J          34 FS?C 04        56♦LBL 01         78 STO \
13 X<> d          35 SF 00          57 FS?C 12        79 "┣**"
14 X<>Y           36 FS?C 05        58 SF 04          80 RCL Z
15 CF 00          37 SF 01          59 FS?C 13        81 STO [
16 CF 01          38 FS?C 06        60 SF 05          82 ISG L
17 CF 02          39 SF 02          61 FS?C 14        83 GTO "HB"
18 FS?C 03        40 FS?C 07        62 SF 06          84 RCL [
19 GTO 01         41 SF 03          63 FS?C 15        85 CLA
20 SF 04          42 FS?C 11        64 SF 07          86 STO [
21 FC?C 07        43 GTO 01         65 RDN            87 TONE 9
22 SF 07          44 SF 12          66 X<> d          88 AVIEW
                                                      89 END
```

"CODE"

191 BYTES

Gebrauchsanweisung für "CODE":

1. XEQ "CODE".
2. Nach Eingabe-Aufforderung "CODE=?" sind 14 Alpha-Zeichen zur Darstellung des gewünschten Kodes einzutasten, wobei die *Zeichen* "0" bis "9" und "A" bis "F" für die Nybbles '0' bis 'F' zu verwenden sind.
3. R/S.
4. Nach dem 'Piep' steht der angeforderte Kode in den Registern X und M (und wird mit 'AVIEW' angezeigt).

Um zu überprüfen, ob Ihre Fassung von "CODE" fehlerfrei ist, versuchen Sie folgende Beispiele:

Eingabe		Ausgabe (AVIEW)
"41 42 43 44 45 46 47"	⟶	"ABCDEFG"
"00 01 28 29 00 7F 7E"	⟶	"␀␁(␀)␀␡~"

[Der Rest dieses Abschnittes ist einer genauen Untersuchung der Wirkungsweise von "CODE" gewidmet und kann beim ersten Lesen überschlagen werden.]

Die Aufgabe von "CODE" ist es, 14 *Zeichen*, die der Benutzer beim Halt auf Zeile 04 eingegeben hat, in die entsprechenden *Bytes* umzusetzen. (Die globalen Marken "CO" und "HB" werden von anderen Programmen, wenn sie Teile von "CODE" als Unterprogramm aufrufen, angesprungen.) Um die Programmlänge auf ein Mindestmaß herabzusetzen, wird die Aufgabe von einer Routine bewältigt, die *ein Paar* von Eingabe-Zeichen in *ein* Ausgabe-Byte umsetzt. Die Routine wird siebenmal durchlaufen und benutzt das Alpha-Register, um die aufeinanderfolgenden Ausgabe-Bytes zusammenzuheften. Die Aufgabe wird durch die Forderung, keine Daten-Register zu verwenden, besonders verzwickt, weil lediglich das (die) Alpha-Register und der Rechenregisterstapel zum Jonglieren mit dem Eingabe-Kode, dem Ausgabe-Kode, dem ursprünglichen Inhalt des Flag-Registers und jeglicher für die Umsetzung anfallenden Arithmetik verbleiben.

Der entscheidende Teil der Umsetzung findet in den Zeilen 10-64 statt. Um zu verstehen, wie dort gearbeitet wird, wollen wir annehmen, daß ein Paar von Eingabe-Zeichen in die ersten zwei Bytes des Registers d gebracht worden sei. Nehmen wir z.B. die Zeichen "49", die in das Einzelbyte '49' verwandelt werden müssen. Die Anfangszeichen bestehen in Wirklichkeit aus den Bytes '34 39' - wir haben also, die beiden Dreien unbeachtet lassend, die '4' ins erste Nybble und die '9' ins zweite Nybble zu bringen:

34 39 ⟶ 49 39

woraufhin wir unsere Aufmerksamkeit ganz dem ersten Byte zuwenden können. Das Versetzen der Ziffern geschieht mit gewöhnlichen Flag-Befehlen - rufen Sie sich ins Gedächtnis zurück, daß die ersten vier Bits des Registers d die Flags 0-3 sind, die nächsten vier die Flags 4-7 usw. Die die Umwandlung ausführenden Programmzeilen sind die folgenden (rechts von jeder Zeile wird ihre Wirkung auf die ersten

16 Bits des Registers d gezeigt):

Programmzeile	Flags 00-15
(Anfangswert '34 39')	0011 0100 0011 1001
15 CF 00	"
16 CF 01	"
17 CF 02	0001 0100 0011 1001
18 FS?C 03	0000 0100 0011 1001
19 GTO 01	
32 LBL 01	
34 FS?C 04	"
35 SF 00	"
36 FS?C 05	0000 0000 0011 1001
37 SF 01	0100 0000 0011 1001
38 FS?C 06	"
39 SF 02	"
40 FS?C 07	"
41 SF 03	"
42 FS?C 11	"
43 GTO 01	"
56 LBL 01	
57 FS? 12	"
58 SF 04	0100 1000 0011 1001
59 FS? 13	"
60 SF 05	"
61 FS? 14	"
62 SF 06	"
63 FS? 15	"
64 SF 07	0100 1001 0011 1001 = 49 39

Die Einfachheit der voranstehenden Programmzeilen beruht auf der Tatsache, daß das zweite Nybble jedes der Zeichen "0" bis "9" mit dem binären Wert dieser Zeichen, aufgefaßt als Dezimalziffer, übereinstimmt. Leider gilt dies nicht für die Zeichen "A" bis "F"; sie erfordern ein umständlicheres Umsetzungsverfahren. Das Programm muß jedes Eingabezeichen daraufhin überprüfen, ob es 'größer' als "9" ist, und das verlangt besondere Bearbeitung. Die Prüfung des ersten Zeichens eines Eingabe-Paares findet in Zeile 18 statt. Die nachträgliche Umsetzung erfolgt in den Zeilen 20-31 (und verwendet Zeile 11). Das zweite Zeichen eines Eingabe-Paares wird in Zeile 42 geprüft; die Zeilen 33 und 44-55 veranlassen die zugehörige Umwandlung.

Der Rest von "CODE" umgibt die eben beschriebene Hauptroutine: das Programm muß zwei Eingabe-Zeichen-Bytes in die ersten zwei Bytes des Registers d schaffen, die Routine durchlaufen und dann ein Ausgabe-Byte aus dem Register d herausholen. In den Registern N und O werden die Eingabe-Zeichen abgelegt; Register M enthält den 'anwachsenden' Ausgabe-Kode. Es wird vielleicht spannend für Sie sein, die Zeilen 66-81 in Einzelschritten durchzugehen, um zu sehen, wie die führenden Bytes aus Register d den letzten Zeichen im Register M angehängt werden, während gleichzeitig der Eingabe-Kode in den Registern N und O um *zwei* Stellen nach links geschoben wird. Nach sieben Wiederholungen (Register L dient

als Zähler) enthält M die endgültigen Ausgabe-Bytes.

Wenn Eingabe-Zeichen ins Register d gelangen (Zeile 13), werden zahlreiche Systemflags gesetzt oder gelöscht (der halbe Spaß an "CODE" besteht darin, den verschiedenen Indikatoren der Anzeige beim Aufblinken und Verlöschen zuzusehen), einschließlich möglicherweise des PRGM-Modus-Flags, Flag 52. Wenn dieses Flag während eines gerade ablaufenden Programmes gesetzt wird, können gewisse Operationen den HP-41C zum Wüterich werden lassen, der sich selbst zu programmieren beginnt! (Siehe Abschnitt 7B.) Zu diesen Operationen gehören ganz gewöhnliche Zahleneingabe-Zeilen, so daß zur Verhinderung eines solchen Schicksalsschlages der Zeile '13 X<>d' vorangehend eine '1' schon in Zeile 11 eingegeben wird, was wiederum die - andernfalls verschwenderische - Einbeziehung der Zeile '14 X<>Y' erzwingt.

5D. Direkter Zugriff auf die Programm-Register

Das im letzten Abschnitt erarbeitete Programm "CODE" ist ein leistungsfähiges Werkzeug für die synthetische Programmierung, doch können wir damit bis jetzt nicht viel mehr anfangen, als Ketten von nicht eintastbaren Alpha-Zeichen erzeugen, weil die Ausgabe-Kodes nur in andere *Daten-Register* übertragen werden können. Aber bedenken Sie dies: Die Trennung des HP-41C-Speichers in Programm- und Daten-Register wird lediglich durch eine Zahl überwacht - nämlich die im Register c abgelegte Adresse von R_{00} . Die Änderung dieser Adresse erfolgt gewöhnlich mit der 'SIZE'-Funktion, die auch die Inhalte der Speicherregister so verschiebt, daß Programme bei Programmen und Daten bei Daten bleiben. Als ehrgeizige Programmierer lassen wir uns durch diese Kleinigkeit nicht abschrecken - synthetische Funktionen verschaffen uns Zugang zum Register c , und "CODE" ermöglicht uns, alle Bytes, die wir dorthin bringen möchten, zu erzeugen. Indem wir einen geeigneten Kode ins Register c setzen, können wir den 'Vorhang', der Programm- und Daten-Register auseinanderhält, ohne 'SIZE' zu benutzen, überall dort 'aufhängen', wo wir wollen, und dadurch die Inhalte von Daten-Registern in Programmzeilen überführen, oder umgekehrt!

Überdies werden wir, um im rechten Geiste weiterzufechten, ein Programm schreiben, welches den ganzen Vorgang des Speicherns in Programm-Register selbständig erledigt. Die allgemeine Marschrichtung ist die: Wir verwenden "CODE" zweimal; ein erstes Mal, um einen vorübergehenden Wert, der R_{00} als das Programm-Register festlegt, in welches wir einen besonderen Kode bringen wollen, ins Register c zu befördern; und ein zweites Mal, um den besonderen Kode, den wir in dem so neu bestimmten R_{00} ablegen wollen, zu erzeugen. Wir tauschen, 'X<>c' benutzend, den neuen Wert von c gegen den ursprünglichen aus, führen, während der besondere Kode in X liegt, 'STO 00' durch und leiten schließlich den ursprünglichen Inhalt von c nach dorthin zurück, so daß uns der Zugriff auf vorhandene Programme und Daten in der anfänglichen Aufteilung erhalten bleibt.

Der vorübergehend im Register c unterzubringende Kode lautet

'10 00 01 69 xy z1 00' .

Hierin ist 'xyz' die 3-ziffrige absolute Adresse des Programm-Registers, zu dem wir den Zugang suchen, und '169' die 'Kaltstart-Konstante', die benötigt wird, um "MEMORY LOST" zu verhüten. '100' ist nur der Einfachheit halber für die Adressen von 'ΣREG' und '.END.' gewählt - da sich diese Adressen bloß vorübergehend im Register c aufhalten, sind ihre Werte nicht von Bedeutung. Die '10' als Anfangsbyte macht den Kode handlich, weil sie ihn zu Alpha-Daten erklärt, die ohne Normalisierung gespeichert und abgerufen werden können. Die einzigen Variablen in dem für Register c vorgesehenen neuen Inhalt sind die Ziffern 'xyz', so daß wir die Ausführungszeit des Programms verkürzen können, indem wir "CODE" zur Erzeugung von nur 2 Bytes (hier 'xy z1') einsetzen. Das folgende Programm, "REG", ist eine Verwirklichung dieser Ideen.

```
01*LBL "RREG"          13 RCL [               25 RCL 00
02 SF 10               14 STO \               26 X<> c
03 GTO 01              15 "┣+++++++θ+×i"      27 RCL [
04*LBL "REG"           16 .001                28 X<> 00
05 CF 10               17 STO L               29 FS?C 10
06*LBL 01              18 XEQ "HB"            30 STO 00
07 "REG?"              19 "┣+"                31 X<>Y
08 AON                 20 RCL [               32 STO c
09 STOP                21 STO 00              33 X<>Y
10 AOFF                22 CLD                 34 "REG-"
11 ASTO 01             23 FC? 10              35 ARCL 01
12 "┣1"                24 XEQ "CODE"          36 AVIEW
                                              37 END
```

"REG"

101 BYTES

SIZE 002

Gebrauchsanweisung für "REG":

1. XEQ "REG". (Soll nur der Inhalt eines Registers aufgerufen werden, XEQ "RREG".)
2. Nach Eingabe-Aufforderung "REG?" sind drei Alpha-Zeichen zur Kennzeichnung einer absoluten Register-Adresse einzutasten, dann 'R/S'.
3. Nach Eingabe-Aufforderung "CODE=?" sind 14 Alpha-Zeichen zur Festlegung des zu speichernden Kodes einzutasten; dann 'R/S'.
4. Die Anzeige "REG-abc" (abc ist die Register-Adresse) meldet, daß die Programm-Ausführung abgeschlossen ist.

Die Zeile 15 in "REG" ist eine synthetische Programmzeile mit dem Kode

'FB 7F 00 00 00 00 00 00 10 00 01 69'.

Sie kann folgendermaßen durch Byte-Hüpfen erzeugt werden:

```
15 STO 07
16 STO 02
17 "├ABCDEFGHIJ"
                    JUMP .016          [16 X>Y?     ]
17 0
18 /
19 LBL 00
20 FRC
                    JUMP .016          [16 X>Y?     ]
                    SST, SST           [18 /        ]
                    DEL 001            [17 STO 02   ]
                    JUMP .017          [17 -        ]
                    DEL 005            [16 STO 02
                    GTO .018           [18 X<=Y?    ]
                    DEL 005            [17 "├------*-=*"]
                    GTO .015           [15 STO 07   ]
                    DEL 002            [14 STO N    ]
```

Die Marke "RREG" dient allein dazu, den Inhalt eines Registers nach X zu bringen. Denken Sie jedoch daran, daß das aufgerufene Register durch den Aufruf normalisiert wird.

Um ein Anwendungsbeispiel für "REG" zu haben, führen Sie 'SIZE 010' aus (falls keine Speichermodule eingesetzt sind - andernfalls 'SIZE 074' für einen Modul, 'SIZE 138' für zwei, 'SIZE 202' für drei oder 'SIZE 266' für vier). Benutzen Sie 'CAT 1', um den Adreßzeiger auf das erste Programm im Speicher zu setzen; drücken Sie anschließend 'RTN', damit der Zeiger an den Anfang des Programmes gelangt. Die Adresse des ersten Programm-Registers ist aufgrund unserer Vorbereitung 'OF5'. Tasten Sie jetzt im PRGM-Modus sieben 'ENTER'-Zeilen, die gerade Register OF5 füllen und die schon existierenden Programme im Speicher abwärts schieben, ein. Die synthetische Programmzeile "######" kann nunmehr wie folgt erzeugt werden:

```
                    XEQ "REG"          ["REG?"      ]
"OF5"
                    R/S                ["CODE=?"    ]
"F6232323232323"
                    R/S                ["REG-OF5"   ]
                    GTO auf das erste Programm (mit CAT 1)
                    GTO .001
                    PRGM (an)          [01 "######" ]
```

Wenn Sie die voranstehende Folge von Anweisungen wiederholen und dabei

"F6232323232323" durch "0191759FOA9676"

ersetzen, gelangen die folgenden Programmzeilen an den Kopf des Programmspeichers:

```
01 LBL 00
02 STO M
03 TONE 0
04 ISG N
```

('TONE 0' in Zeile 03 ist in Wirklichkeit 'TONE 10' - siehe Abschnitt 7A.)

Jede gewünschte synthetische Programmzeile kann in dieser Weise hergestellt werden. Um eine Zeile von mehr als 7 Bytes zu erzeugen, brauchen wir "REG" nur zweimal (oder sogar dreimal) anzuwenden und jeweils 7 Bytes des Kodes in benachbarte Register zu bringen.

5E. Synthetische Tastenzuweisungen

Eine besonders wirkungsvolle Anwendung von "REG" gelingt mit der Erzeugung 'synthetischer Tastenzuweisungen', das sind Zuweisungen synthetischer Zwei-Byte-Funktionen auf Benutzertasten, so daß sie von Hand ausgeführt oder unmittelbar ins Programm eingetastet werden können. Um diese Zuweisungen zustandezubringen, benutzen wir "REG" einfach dazu, besondere Kodes in die Tastenzuweisungsregister zu legen. Jeder Aufruf von "REG" vermag zwei Benutzertasten zu belegen. Der Vorgang wird am besten durch ein Beispiel erläutert: wir werden die Funktionen 'RCL M' und 'STO M' den Tasten 'TAN' (25) bzw. 'ATAN' (-25) zuweisen.

1. Löschen Sie alle vorhandenen Tastenzuweisungen mit Ausnahme des Byte-Hüpfers. (Diese durchgreifende Maßnahme ist im allgemeinen natürlich nicht nötig; tun Sie es diesmal dennoch.) Führen Sie ein 'PACK' aus, und weisen Sie dann irgendeine Funktion irgendeiner Taste zu.
2. Legen Sie nun *beliebige* HP-41C-Funktionen auf die beiden für unser Beispiel vorgesehenen Tasten ('TAN' und 'ATAN'). Dies bringt einen 'Strohmann'-Kode in das niedrigste Zuweisungsregister, Register 0C0, und setzt zudem die zugehörigen Tastenzuweisungsflags in den Registern ⊢ und e.
3. Bestimmen Sie den Kode, der den Strohmann in Register 0C0 überschreiben soll. Er ergibt sich aus dem in Abschnitt 2E beschriebenen Format der Tastenzuweisungsverschlüsselung. Die Kodes für die zuzuweisenden Funktionen entnehmen wir der Byte-Tabelle; die Tastenzuweisungsbytes ergeben sich aus der Abbildung 2-6. In unserem Falle erhalten wir:

```
F0        führt ein Tastenzuweisungsregister an
90 75     'RCL M'
42        Zuweisung auf die 'TAN'-Taste
91 75     'STO M'
4A        Zuweisung auf die 'ATAN'-Taste
```

4. Benutzen Sie "REG", um den vollständigen Zuweisungskode ins Register 0C0 zu bringen. In unserem Beispiel antworten wir auf die Aufforderung "REG?" mit "0C0" und auf "CODE=?" mit "F090754291754A". Sobald in der Anzeige "REG-0C0" erschienen ist, finden wir im USER-Modus 'RCL M' auf der 'TAN'-Taste und 'STO M' auf 'ATAN'.

Die synthetische Zuweisungstechnik ist in keiner Weise auf synthetische Funktionen beschränkt. Der wichtigste Grund für Tastenzuweisungen ist der, dem Anwender zu erlauben, häufig benutzte Tastenfolgen durch einfachen Druck auf eine Einzeltaste zu ersetzen. Gewöhnlich erfordert ein Einzelbefehl wie 'ST+IND X' 5-ma-

liges Tasten; mit herkömmlichen Mitteln können wir diesen Befehl günstigstenfalls auf 4-maliges Tasten vermindern, indem wir 'ST+' einer Taste zuweisen. Doch mit der Technik der synthetischen Tastenzuweisung ausgestattet haben wir keinen Grund, nicht die ganze Funktion 'ST+IND X' auf eine Taste zu legen (der Funktionskode hieße in diesem Fall '92 F3').

Überdies sind synthetische Zuweisungen nicht einmal auf HP-41C-Funktionen beschränkt; wir können ebensogut Peripherie-Funktionen auf die Tasten legen. Dafür entnehmen wir einfach den 'XROM'-Kode der gewünschten Funktion dem zum Peripherie-Gerät gehörigen Handbuch, setzen ihn mit Hilfe der in Abschnitt 2B beschriebenen Regel in hexadezimale Darstellung um und speichern die so ermittelten Byte-Kodes in ein Zuweisungsregister. Dieses Verfahren setzt den Anwender in die Lage, Programme, die Peripherie-Funktionen enthalten, selbst dann zu schreiben, wenn das entsprechende Gerät nicht verfügbar ist. Ja, wir können sogar nicht-programmierbare Funktionen wie 'LIST' oder 'WALL' in Programme einfügen. Die Kodes für nicht-programmierbare Kartenleser- und Drucker-Funktionen sind in Tabelle 5-1 aufgeführt.

Tabelle 5-1

Nicht-programmierbare Peripherie-Funktionen

Funktion	XROM-Zahl	Byte-Kode	Ausführung?
CARD READER	30,00	A7 80	verändert die Ziffernflags 36-39
VER	30,05	A7 85	normal
WALL	30,06	A7 86	normal
WPRV	30,09	A7 89	normal
-PRINTER-	29,00	A7 40	GAU
LIST	29,07	A7 47	führt 'LIST' von der nächsten Zeile an aus
PRP	29,13	A7 4D	NONEXISTENT

Wenn eine gewöhnliche HP-41C-Funktion einer Taste zugewiesen ist, wird nur eines der beiden Bytes einer Zuweisungsregister-Hälfte benötigt, um die Funktion zu kennzeichnen. Das erste der beiden Bytes ist in diesen Fällen stets '04' (LBL 03). Ist das erste Byte ein anderes als '04', weiß der Prozessor, daß die Zuweisung zu einer Zwei-Byte Peripherie-Funktion gehört. Falls das Gerät nicht angeschlossen ist, bringt der Druck auf die belegte Taste den zugehörigen 'XROM'-Kode in der Anzeige hervor. Im Falle synthetischer Zwei-Byte-Zuweisungen weicht das erste Byte ebenfalls von '04' ab, wieder mit dem Ergebnis einer 'XROM'-Anzeige bei Druck auf die Taste. In unserem Beispiel - Zuweisung von 'RCL M' auf die 'TAN'-Taste - taucht 'XROM 01,53' auf. Läßt man die Taste los, bevor "NULL" in der Anzeige erscheint, wird 'RCL M' ausgeführt.

Synthetische XROM-Kodes können in derselben Weise entschlüsselt werden wie gewöhnliche XROM's, nämlich nach der in Abschnitt 2B beschriebenen Regel. Um beispielsweise den zu 'RCL M' gehörigen XROM-Kode zu bestimmen, schreiben wir den Hexadezimalkode '90 75' für 'RCL M' binär auf , gruppieren die letzten 12 Bits

zu zwei Zahlen von je 6 Bits und wandeln diese in Dezimalzahlen um:

```
hexadezimal:    9     0     7     5
binär:       1001 |0000 01|11 0101|
dezimal:          01       53
```

Deshalb gilt 'RCL M' = 'XROM 01,53' und ähnlich 'STO M' = 'XROM 05,53'.

Um den Aufbau einer größeren Menge von Tastenzuweisungen zu erleichtern, ist es günstig, das synthetische Zuweisungsverfahren stärker als es durch die bloße Verwendung von "REG" möglich ist, vom HP-41C selbst steuern zu lassen. Bei Gebrauch des 'Tastenzuweisungsprogramms', "KA" (key assignment), das gleich aufgelistet wird, entfallen die von Hand auszuführenden Strohmann-Zuweisungen, das Nachschlagen in Tabelle 2-6, um die Tasten-Kodes festzustellen, sowie jede Notwendigkeit, sich um die gegenwärtigen Inhalte der Tastenzuweisungsregister zu sorgen. "KA" lehnt es ab, vorhandene Zuweisungen zu überschreiben, es sei denn, der Benutzer löscht sie auf eine Aufforderung hin.

Zusammen mit "KA" werden drei weitere Routinen aufgelistet. "KP" 'packt' die Tastenzuweisungsregister: obwohl zwei Tastenzuweisungen nur ein Tastenzuweisungsregister belegen, steht ein solches Register nach dem Löschen zweier beliebigen Zuweisungen i.a. nicht wieder zur Verfügung, es sei denn, die beiden Zuweisungen waren im selben Register eingetragen. Es ist daher möglich, eine Anzahl halbgefüllter Zuweisungsregister mitzuführen. "KP" rückt die Kodes in den Zuweisungsregistern zusammen und läßt höchstens ein halbgefülltes Register übrig, dann nämlich wenn eine ungerade Anzahl von Tasten belegt ist. Das halbgefüllte Register ist Register 0C0, so daß eine neue Zuweisung dieses Register auffüllt.

Das Programm "CA" (clear assignments) löscht selbständig alle Zuweisungen von Funktionen und Benutzerprogrammen, jedoch nicht die Tastenzuweisungsbytes in globalen Marken: wird nämlich ein Programm B einer Taste zugewiesen, die vorher mit einem weiter unten im Speicher befindlichen Programm A belegt war und durch "CA" freigemacht wurde, so wird diese neue Zuweisung mißachtet und die alte Zuweisung wiederbelebt, d.h. Programm A liegt wieder auf seiner Taste.

"EF" (end finder) ist eine Routine, die von "KA", "CA" und "KP" aufgerufen wird, um die Lage von '.END.' zu ermitteln. Ist 'xyz' die Anzahl von Registern, die das Register 0C0 vom '.END.'-Register trennt, dann bringt "EF" die Zahl '0.xyz' ins Register X, damit die Wiederholungen, die nötig sind, um die Zuweisungsregister der Reihe nach zu durchlaufen, überwacht werden können. Zeile 31 von "EF" setzt den Kode 'F0 00 00 00 0C 0C 0C' ins Register M. Diese NNZ wird von anderen Programmen ins Register c gebracht, so daß Register 0C0 vorübergehend zum Daten-Register R_{00} wird. Sobald sie sich im Register c befindet, verursacht ein Programm-Stop "MEMORY LOST". Damit dies vermieden wird, darf die Ausführung von "KA", "CA" oder "KP" nicht unterbrochen werden. Darüberhinaus müssen Sie dafür sorgen, *daß sich keine der vier Routinen als erste im Programmspeicher befindet*. Es muß wenigstens ein den Routinen voranstehendes

```
01↑LBL "EF'      09 CF 02         17 FS?C 10       25 193
02 RCL c         10 CF 03         18 SF 09         26 -
03 STO [         11 FS?C 07       19 FS?C 11       27 X<0?
04 "⊢↑↑↑↑X"      12 SF 05         20 SF 10         28 GTO 15
05 RCL [         13 FS?C 08       21 FS?C 12       29 1 E3
06 X<> d         14 SF 06         22 SF 11         30 /
07 CF 00         15 FS?C 09       23 X<> d         31 "↑↑↑μμμ"
08 CF 01         16 SF 07         24 DEC           32 END
```

"EF"

79 BYTES

```
01↑LBL "KA"      42 GTO 06        83 STO [         124 RCL [
02 XEQ "EF"      43 X<> Z         84 FS? 01        125 "μμμ"
03 CF 00         44 XEQ 02        85 "⊢↑↑↑"        126 RCL [
04 CF 01         45 XEQ 02        86 X<> [         127 X<> c
05 SF 03         46 36            87 X<> d         128 X<>Y
06 RCL [         47 STO a         88 FC? IND Y     129 STO 00
07 ENTER↑        48 RDN           89 GTO 01        130 X<>Y
08 X<> c         49 ENTER↑        90 X<> d         131 X<> c
09 X<>Y          50 X<0?          91 RCL \         132 "DONE"
10 X<> 00        51 SF 00         92 FIX 0         133 BEEP
11↑LBL 00        52 ABS           93 "CLEAR "      134 PROMPT
12 ""            53 ;i            94 ARCL T        135↑LBL 02
13 X<> [         54 *             95 TONE 4        136 X=0?
14 "⊢↑"          55 LASTX         96 PROMPT        137 "⊢↑↑↑↑"
15 X<> \         56 -             97 STO \         138 OCT
16 X=0?          57 INT           98 RDN           139  E3
17 GTO 01        58 ST- a         99 X<> d         140 /
18 "⊢------"     59 LASTX         100↑LBL 01       141 10
19 X<> \         60 FRC           101 SF IND Y     142 +
20 ISG Z         61 80            102 X<> d        143 X<> d
21 GTO 03        62 *             103 STO [        144 FS?C 19
22 STO 00        63 ST- a         104 "⊢↑↑↑↑"      145 SF 20
23 RDN           64 2             105 FC?C 01      146 FS?C 18
24 STO c         65 *             106 "⊢↑↑↑"       147 SF 19
25 GTO 15        66 +             107 RCL \        148 FS?C 17
26↑LBL 03        67 8             108 FS? 00       149 SF 18
27 X<> IND Z     68 FC? 00        109 STO e        150 FS?C 15
28 GTO 00        69 CLX           110 FC?C 00      151 SF 17
29↑LBL 01        70 +             111 STO '        152 FS?C 14
30 RDN           71 X<> a         112 CLA          153 SF 16
31 STO c         72 24            113 ARCL L       154 CF 07
32 CLA           73 X<=Y?         114 RCL a        155 SF 03
33↑LBL 05        74 SF 01         115 XEQ 02       156 X<> d
34 CF 22         75 FC? 01        116 FS?C 03      157 ARCL X
35 ASTO L        76 CLX           117 GTO 05       158 "⊢ABC"
36 "PRE↑POST↑KEY" 77 -            118↑LBL 06       159 0
37 TONE 8        78 FS? 00        119 ASTO X       160 X<> \
38 PROMPT        79 RCL e         120 ""           161 STO [
39 CLA           80 FC? 00        121 FC?C 22      162 RDN
40 ARCL L        81 RCL '         122 "⊢↑↑↑"       163 RDN
41 FC? 22        82 ASTO L        123 ARCL X       164 END
```

"KA"

336 BYTES

'END' geben, damit die durch 'GTO's ausgelösten Rückwärtssprünge richtig ablaufen. Sobald nämlich ein 'GTO' einen Sprung auf eine weiter oben im Speicher befindliche lokale Marke befiehlt, läuft die Suche zunächst nach unten bis zum 'END' und wird bei Mißerfolg am Anfang des Programms wieder aufgenommen, bis die Marke gefunden ist. Wenn aber der Programm/Daten-'Vorhang' ganz nach unten ins Register OCO geschoben wird, springt die Suche in den Bereich der nichtexistierenden Adressen, es sei denn, es gibt noch oberhalb des durchsuchten Programms ein 'END'.

Gebrauchsanweisung für "KA":

Das Programm darf nicht als erstes im Speicher stehen. Um eine oder zwei Zuweisungen zu tätigen:

1. XEQ "KA". Die Anzeige meldet sich mit "NONEXISTENT", falls keine Register mehr für Zuweisungen zur Verfügung stehen. *Versuchen Sie nicht, die Ausführung zu unterbrechen; wenn Sie das Programm einmal aufgerufen haben, tätigen Sie wenigstens eine Zuweisung.*
2. Nach Eingabe-Aufforderung "PRE↑POST↑KEY" sind drei Zahlen einzutasten:

   'Vorsilbe', 'ENTER↑', 'Nachsilbe', 'ENTER↑', 'Tasten-Kode', 'R/S'.

   Für 'Vorsilbe' und 'Nachsilbe' sind die *Dezimalwerte* der Vor- und Nachsilben-Bytes der zuzuweisenden Funktion aus der Byte-Tabelle zu nehmen. Beispielsweise müßten die Vorsilben '145' und '118' für die Funktion 'STO N' ('91 76') eingegeben werden. Der 'Tasten-Kode' ist der Zeilen-Spalten-Kode, der während gewöhnlicher Zuweisungen kurz in der Anzeige erscheint; also '12', wenn die Taste '1/X' belegt wird, '-54' für 'FS?' usw. Die Tasten 'CHS', 'EEX' und '←' (sowie ihre Umschalttasten) müssen mit '43', '44' bzw. '45' (negativ für Umschaltung) angesprochen werden, so als bedeckte 'ENTER↑' noch eine imaginäre Taste '42'.
3. Wenn die in Schritt 2 bestimmte Taste schon eine Zuweisung trägt, erklingt TONE 4, und die Anzeige fordert mit "CLEAR mn", worin 'mn' der Tasten-Kode ist, zur Löschung auf; dies ist von Hand auszuführen, und dann ist mit 'R/S' fortzufahren.
4. Schritt 2 wird für die zweite Zuweisung automatisch wiederholt. Wünscht man nur eine Zuweisung, muß die zweite Aufforderung mit 'R/S' beantwortet werden. Ist "KA" beendet, ertönt der BEEP, und die Anzeige meldet "DONE". Weitere Zuweisungen müssen wieder mit Schritt 1 begonnen werden.

```
01*LBL "CA"        08*LBL 00        15 ISG [
02 0               09 0             16 GTO 00
03 STO e           10 ENTER↑        17 RDN
04 STO '           11 X<> IND [     18*LBL 01
05 XEQ "EF"        12 X=Y?          19 RCL Z
06 X<> [           13 GTO 01        20 STO c
07 X<> c           14 RCL Z         21 END
```

"CA"

42 BYTES

Gebrauchsanweisung für "CA":

1. XEQ "CA". Wenn keine Funktionszuweisungen vorhanden sind und überdies kein Platz mehr dafür verfügbar wäre, wird "NONEXISTENT" angezeigt. *Unterbrechen Sie nicht den Programmlauf.*

```
01♦LBL "KP"     20 STO [      39 X<> \      58 XEQ 00     77 ASTO X
02 XEQ "EF"     21 ASHF       40♦LBL 01     59 X<> 00     78 ASHF
03 STO Y        22 X<> [      41 STO \      60 SIGN       79 ASTO L
04 RCL [        23 X=0?       42 FC?C 01    61 X≠0?       80 ""
05 X<> c        24 SF 01      43 "├♦♦♦"     62 GTO 03     81 ARCL X
06 ENTER↑       25 CLX        44 X<> \      63 CLX        82 CLX
07 CLA          26 "├♦"       45 CLA        64 LASTX      83 X<> [
08 CF 03        27 STO \      46 STO [      65 XEQ 00     84 STO IND T
09♦LBL 14       28 "├♦♦"      47♦LBL 09     66 STO IND T  85 ARCL L
10 FS?C 03      29 X<> \      48 ASTO X     67♦LBL 03     86 ISG T
11 GTO 07       30 "├♦"       49 CLA        68 RDN        87 GTO 14
12 CLX          31 X<> \      50 ARCL L     69 STO c      88 RTN
13 X<> IND Z    32 X=0?       51 ARCL X     70 TONE 9     89♦LBL 00
14 SF 25        33 GTO 01     52 SF 03      71 RTN        90 ""
15 X=0?         34 X<> \      53 ISG Z      72♦LBL 05     91 X<> [
16 FS?C 25      35 ASTO [     54 CF 03      73 X<> \      92 "├↑"
17 GTO 07       36 "├♦"       55 GTO 05     74 STO \      93 X<> \
18 ASTO L       37 STO \      56♦LBL 07     75 X=0?       94 "├↑↑↑↑↑↑"
19 CF 01        38 "├♦"       57 X<> [      76 GTO 14     95 X<> \
                                                          96 END
```

"KP"

190 BYTES

Gebrauchsanweisung für "KP":

1. XEQ "KP". Wenn keine Funktionszuweisungen vorhanden sind und überdies kein Platz mehr dafür verfügbar wäre, wird "NONEXISTENT" angezeigt. *Unterbrechen Sie nicht den Programmlauf.*

Einige der Zeilen in den Tastenzuweisungsprogrammen erfordern sorgfältiges Byte-Hüpfen. Es werden daher die für das sichere Eintasten solcher Zeilen nötigen Einzelschritte aufgelistet.

Die Zeilen 12 und 120 in "KA" und die Zeilen 80 und 90 in "KP" haben alle den Byte-Kode 'F1 F0'. Um eine von ihnen, etwa Zeile 12 aus "KA", zu erzeugen, gehen Sie so vor:

```
12 STO 01
13 "BH"
                  JUMP          [13 *           ]
14 GTO 07
15 STO IND T
                  PACK          [14 STO IND T   ]
                  JUMP .013     [13 *           ]
14 "A"
                  GTO .014      [14 -           ]
                  DEL 002       [13 "B✱"        ]
                  PACK
                  JUMP          [13 *           ]
14 X<>Y
                  GTO .015      [15 Σ-          ]
                  DEL 001       [14 "✱"         ]
```

```
                              GTO .012      [12 STO 01        ]
                              DEL 002       [11 LBL 00        ]
```

Zeile 125 in "KA" enthält den Kode 'F3 0C 0C 0C':

```
                125 STO 01
                126 "BH"
                              JUMP          [126 *            ]
                127 "ABC"
                              GTO .127      [127 -            ]
                              DEL 004       [126 "B*"         ]
                127 LBL 11
                128 LBL 11
                129 LBL 11
                              JUMP .126     [126 *            ]
                127 X<>Y
                              GTO .125      [125 STO 01       ]
                              DEL 002       [124 RCL M        ]
```

Der Kode für Zeile 04 von "EF" lautet 'F6 7F 00 00 00 00 02':

```
                04 STO 02
                05 "├ ABCDE"
                              JUMP          [05 -             ]
                              SST 3-mal     [08 X<Y?          ]
                09 LBL 01
                              JUMP .005     [05 -             ]
                              DEL 004       [04 STO 02        ]
                              DEL 001       [03 STO M         ]
                              GTO .005      [05 X>Y?          ]
                              DEL 001       [04 "├ ----*"     ]
```

Und schließlich Zeile 31 von "EF" mit 'F7 F0 00 00 00 0C 0C 0C':

```
                31 STO 01
                32 "BH"
                              JUMP          [32 *             ]
                33 GTO 10
                34 STO IND T
                              PACK          [33 STO IND T     ]
                              JUMP .032     [32 *             ]
                33 "ABCDEFG"
                              GTO .033      [33 -             ]
                              DEL 008       [32 "B*"          ]
                              PACK
                              GTO .033      [33 ""            ]
                34 +
                35 +
                36 +
                37 LBL 11
                38 LBL 11
                39 LBL 11
                              GTO .034      [34 +             ]
                              DEL 003       [33 ""            ]
                              JUMP .032     [32 *             ]
                33 X<>Y
                              GTO .031      [31 STO 01        ]
                              DEL 002       [30 /             ]
                              GTO .032      [32 Σ-            ]
                              DEL 001       [31 "*---µµµ"     ]
```

Als Beispiel für den Gebrauch von "KA" wollen wir die Funktionen 'RCL b' (Vorsilbe 144/Nachsilbe 124) und 'STO b' (145/124) zwei Tasten zuweisen. Diese Funktionen ermöglichen uns, den Adreßzeiger auf normalerweise unzugängliche Stellen im Speicher zu setzen. Versuchen Sie folgendes (führen Sie dabei die Anweisungen genau aus; wenn Sie nämlich zusätzliche Zeilen eintasten, während sich der Zeiger in den Zustandsregistern befindet, erfolgt "MEMORY LOST"):

```
                        XEQ "CODE"      ["CODE=?"           ]
"00000000000006"
                        R/S             ["7"                ]
                        ALPHA           ["7"                ]
"ABCDEFGHIJ"
                        ALPHA           [0.000.000,00       ]
                        STO b (benutzen Sie die belegte Taste)
                        PRGM, SST       [01 X<Y?            ]
                        DEL 003         [00 REG ---         ]
01 RCL 08
02 STO 15
03 RCL 09
                        PRGM (aus)
                        ALPHA           ["ABC(?)GHIJ"       ]
```

Sie haben tatsächlich das Alpha-Register ediert - der Befehl 'STO b' schickte den Zeiger ins letzte Byte von Register N auf die Adresse 0006. Das 'DEL 003' beseitigte die ersten drei Bytes von Register M, die Sie dann durch die Zeichen "(?)" ersetzten, indem Sie die entsprechenden Programmzeilen einfügten. Ähnlich würden Sie im PRGM-Modus neue Programmzeilen sehen, wenn Sie die Zeichen im Alpha-Register änderten.

5F. Erzeugung synthetischer Programmzeilen

Synthetische Programmzeilen lassen sich in vier Gruppen aufteilen: (1) synthetische Zwei-Byte-'Funktionen', gewöhnlich aus einer normalen Vorsilbe und einer Zustandsregister-Nachsilbe zusammengesetzt; (2) synthetische Textzeilen, die mindestens ein nicht-tastbares Zeichen enthalten; (3) andere ungewöhnliche Mehrbyte-Zeilen, insbesondere globale Marken, 'GTO's und 'XEQ's, deren Name nicht-tastbare Zeichen enthält; (4) 'En'-Zeilen, die im Gegensatz zu normalen Zeilen der Form '1 En' - wie sie bei Eingabe einer Potenz von 10 entstehen (z.B. '1 E3') - durch Tilgen des überflüssigen '1'-Bytes zu 'En' verkürzt sind. Von diesen vier sind die Arten 1 und 2 die gängigsten. Zeilen vom Typ 3 sind wegen der Fülle der normalerweise verfügbaren globalen Marken eigentlich nur Kuriositäten, die lediglich den fortgeschrittenen Programm-Klempner fesseln. Die Erzeugung von Zeilen des Typs 4 ist für die Hüter der reinen Lehre vorgesehen, für die ein einzelnes vergeudetes Byte eine Herausforderung ist. Wir werden Herstellungsverfahren für Zeilen des Typs 1 und 2 in allen Einzelheiten untersuchen und währendessen beiläufig die Bildung solcher des Typs 4 erlernen. Typ 3 werden wir wenig Aufmerksamkeit widmen; sie lassen sich im allgemeinen mit denselben Methoden, die man bei

Typ 2 anwendet, erzeugen.

Wir haben drei Wege eingeschlagen, um die Aufgabe, synthetische Programmzeilen zu erzeugen, zu lösen. Der am leichtesten zu beschreitende war der, ein herkömmliches Programm, "CODE", zu benutzen, um Bytes aufzubereiten und die nach Belieben erzeugten Byte-Folgen mit Hilfe von "REG" im Programmspeicher abzulegen. Dieses Verfahren der 'programmierten Programmierung' ist in seiner Anwendung unbeschränkt - mit ihm können wir alle vier Arten synthetischer Programmzeilen mit beliebigen Byte-Verknüpfungen hervorbringen. Der Preis für diese Bequemlichkeit und Leistungsstärke sind allerdings Programm-'Spesen' in Form von fast 300 ausgebuchten Bytes für "CODE" und "REG". Darüberhinaus benötigen wir ein Hilfsmittel, um die Adresse des Registers, in welchem die neuen Zeilen abgelegt werden sollen, herauszufinden, und das erfordert noch ein weiteres Programm (siehe Abschnitt 5J).

Die zweite Möglichkeit, Programmzeilen zu erzeugen, ist die, "KA" zu verwenden, um synthetische Funktionen auf Tasten zu legen, so daß sie nach Gutdünken in Programme eingefügt werden können. Diese Methode ist allerdings auf Zeilen des Typs 1 beschränkt. Weil "KA" ein sehr langes Programm ist, empfiehlt es sich, mit ihm eine Reihe von Zuweisungen auf einen Schlag zu tätigen und es dann aus dem Speicher zu verbannen.

Die Verfügbarkeit gewisser ausgefallenen Tastenzuweisungen, die auf fintenreicher Bedienung des HP-41C beruht, gibt eine dritte Gelegenheit, synthetische Zeilen zu gewinnen. Das Urbild solcher Zuweisungen ist der Byte-Hüpfer, mit dem wir die ganze synthetische Programmierung einleiteten. Wie wir im nächsten Abschnitt sehen werden, läuft die Verwendung tastenzugewiesener Befehle 'RCL e' und 'STO e' auf eine bedeutende Verbesserung des Byte-Hüpf-Verfahrens hinaus. Die Abschnitte 5H und 5I beschreiben zwei neue eigentümliche Zuweisungen für exotisches Edieren, den 'Textgehilfen' und den 'Q-Boten'. Der Textgehilfe dient dazu, beliebige Programmzeilen in Textzeilen umzuwandeln und umgekehrt. Der Q-Bote wird zusammen mit "CODE" verwendet, um wahlfreie 7-Byte-Kodes als Textzeilen in Programme zu bringen. Byte-Hüpfer und Textgehilfe beanspruchen sehr wenig Programmplatz, sind indessen in Bezug auf die Byte-Verknüpfungen, die sie - jeder allein - hervorzubringen vermögen, etwas begrenzt (gemeinsam eingesetzt können Byte-Hüpfer und Textgehilfe aber jede Byte-Kombination mit Ausnahme solcher, die E4 bis EF enthalten, zusammenstellen). Wird der Q-Bote vereint mit "CODE" benutzt, läßt sich jede Verbindung von sieben oder weniger Bytes herstellen.

Die voranstehenden Betrachtungen legen es nahe, ein 'Einheitstastenfeld der synthetischen Programmierung' mit Zuweisungen für die am häufigsten benutzten synthetischen Funktionen herzurichten, so wie es Abbildung 5-1 vorschlägt. Wenn das Bedürfnis nach anderen synthetischen Programmzeilen entsteht, setzen Sie den Byte-Hüpfer und/oder den Textgehilfen ein. Reichen diese nicht aus, wird "CODE" geladen und der Q-Bote benutzt. Schließlich kann man auch noch, falls nötig, "REG" damit beauftragen, irgendeine ausgefallene Byte-Verknüpfung, die den an-

deren Verfahren Trotz bietet, zu bilden.

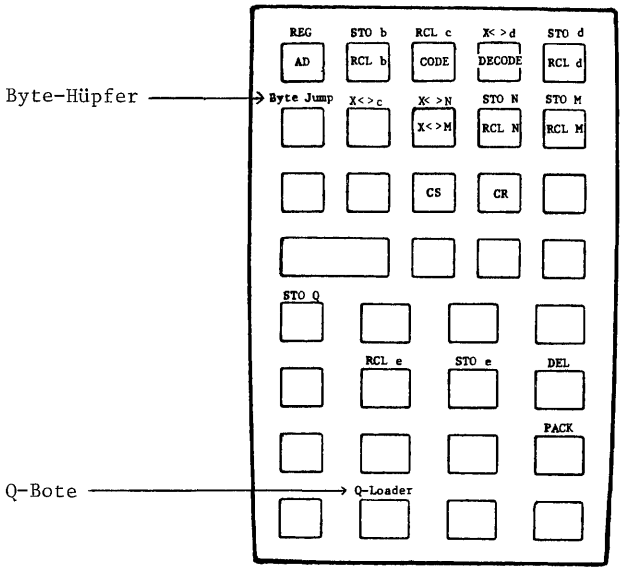


ABBILDUNG 5-1. EIN TASTENFELD FÜR DAS SYNTHETISCHE PROGRAMMIEREN

Eine Durchsicht der verschiedenen Programme dieses Buches deckt schnell auf, daß die folgenden synthetischen Funktionen hinlänglich oft verwendet werden, um ihre dauerhafte Tastenzuweisung zu rechtfertigen: STO M, RCL M, X<>M, STO N, RCL N und X<>N für bequemen Zugriff auf die Alpha-Register; STO b und RCL b, um den Adreßzeiger durch die Gegend zu jagen; X<>c und RCL c für die Handhabung von Daten-Register-Adressen; STO d, RCL d und X<>d für den Zugang zum Flag-Register; STO e und RCL e für 'erweitertes Byte-Hüpfen' (Abschnitt 5G); STO Q und der Q-Bote (siehe Abschnitt 5I); der Byte-Hüpfer. Wenn genügend Platz zur Verfügung steht, ist es günstig, "CODE", "REG", "CS" und "CR" (Abschnitt 5K) sowie "AD" (vielleicht sogar "DECODE" - siehe Abschnitt 5J) in den Speicher zu laden und Benutzertasten zuzuweisen. Abbildung 5-1 zeigt ein der synthetischen Programmierung angepaßtes Tastenfeld. Beachten Sie bitte, daß alle 'STO'- und die meisten 'X<>'-Befehle auf Umschalttasten gelegt sind, um die Gefahr zufälliger Speicherung in empfindliche Register zu vermindern.

Um das Tastenfeld aus Abbildung 5-1 einzurichten, bedient man sich am besten des Programms "KA". Tabelle 5-2 führt sowohl die erforderlichen Vorsilben, Nachsilben und Tasten-Kodes auf als auch die 'XROM'-Zahlen, die in der Anzeige erscheinen, wenn die jeweilige Taste gedrückt und festgehalten wird.

Tabelle 5-2

"KA"-Einträge für das Tastenfeld in Abbildung 5-1

Funktion	Vorsilbe	Nachsilbe	Tasten-Kode	XROM-Zahl
STO b	145	124	-12	05,60
RCL b	144	124	12	01,60
RCL c	144	125	-13	01,61
X<>d	206	126	-14	57,62
STO d	145	126	-15	05,62
RCL d	144	126	15	01,62
X<>c	206	125	-22	57,61
X<>N	206	118	-23	57,54
X<>M	206	117	23	57,53
RCL N	144	118	24	01,54
STO N	145	118	-24	05,54
RCL M	144	117	25	01,53
STO M	145	117	-25	05,53
STO Q	145	121	-51	05,57
RCL e	144	127	-62	01,63
STO e	145	127	-63	05,63
Q-Bote	4	25	-82	"OD"
Byte-Hüpfer	241	65	-21	05,01

Man kann die Funktionen aus Tabelle 5-2 ebensogut mit Hilfe der nachstehenden Folgen zuweisen:

1. Belegen Sie diese Tasten des HP-41C mit irgendwelchen seiner Funktionen:

$1/X$	y^x	X=Y?
10^x	$\sqrt{x}$	ASIN
LN	e^x	ACOS
%	TAN	ATAN
SIN	CLΣ	P-R
COS	BEEP	π

2. Rufen Sie neunmal "REG" auf, und beantworten Sie die Eingabe-Aufforderungen mit diesen Kodes:

Aufruf Nr.	"REG?"-Eintrag	"CODE=?"-Eintrag
1	0C0	F091763A91754A
2	0C1	F0F1410ACE762A
3	0C2	F0CE7E39CE7522
4	0C3	F0917C1990 7C11
5	0C4	F0CE7D1A907D29
6	0C5	F0917E49907E41
7	0C6	F0907542907632
8	0C7	F09179 0D041920
9	0C8	F0907F1E917F2E

5G. Erweitertes Byte-Hüpfen

Der Hauptmangel des Byte-hüpfenden Edierens ist das Unvermögen, das zweite Byte einer Mehrbyte-Programmzeile geradewegs zu ändern. Das führt dazu, daß wir auf die Benutzung eines 'Erzeugers' zurückgreifen müssen, also auf eine Strohmann-Textzeile, die Vorsilben-Bytes 'heimlich aufnimmt', so daß man hinter ihr Nachsilben ins Programm einfügen kann, welche in dem Augenblick an die Vorsilben geheftet

werden, wenn letztere den Erzeuger beim zweiten Byte-Hüpfen 'verlassen' müssen. Abgesehen von einer zweifachen Verwendung des Byte-Hüpfers läßt dieses Verfahren eine Menge unnützer Bytes zurück, die, einschließlich des Erzeugers selbst, gelöscht werden müssen.

Die Unfähigkeit, 'zweite Bytes' zu ändern, rührt daher, daß ins Programm eingefügte Bytes normalerweise an das letzte Byte der gerade angezeigten Zeile angeschlossen werden. Roger Hill war der erste, der darauf hinwies, daß diese Regel außer Kraft tritt, wenn die Zeilen-Nummer der gegenwärtigen Programmzeile '00' ist. In diesem Fall gelangen eingefügte Bytes unmittelbar hinter das Byte, auf dem gerade der Adreßzeiger steht.*) Das 'erweiterte Byte-Hüpfen' macht sich diese Eigenschaft zunutze und schafft so die bislang notwendige Erzeuger-Zeile ab.

Die einfachste Methode, die Zeilen-Nummer auf Null zu bringen ist die, 'RTN' zu tasten; allerdings setzt dies gleichzeitig den Zeiger an den Anfang des gegenwärtigen Programms zurück. Aber bereits die einfache Tastenfolge (PRGM aus) 'RTN', 'RCL e', 'GTO .ℓmn', 'STO e' ändert den Wert *jeder beliebigen* Programmzeilen-Nummer 'ℓmn' in '00' ab. Register e enthält die Zeilen-Nummer - an jeder Stelle gilt demnach: Gibt man den Inhalt von Register e, der nach einem 'RTN' entstanden ist, nach e zurück, wird damit die Zeilen-Nummer an dieser Stelle auf Null gesetzt. Ein Übergang in den PRGM-Modus im Anschluß an diese Tastenfolge liefert stets die Anzeige '00 REG pqr'. Sie können dasselbe Ergebnis auch mit 'GTO .ℓmn', 'RCL b', 'RTN', 'STO b' erzielen.

Wie schon für den alten Byte-Hüpfer führen wir auch hier zur vereinfachten Darstellung der Tastanweisungen einen neuen Befehl, 'eJUMP .ℓmn', ein, der die nachstehende Tastenfolge zusammenfassen soll:

'eJUMP .ℓmn' bedeutet			oder	
	1.	PRGM aus		1. PRGM aus
	2.	RTN		2. GTO .ℓmn
	3.	RCL e		3. Sprung mit dem Byte-Hüpfer
	4.	GTO .ℓmn		4. RCL b
	5.	Sprung mit dem Byte-Hüpfer		5. RTN
	6.	STO e		6. STO b
	7.	PRGM ein		7. PRGM ein

Bei umfangreichen Programmierarbeiten ist die erste Tastenfolge vorzuziehen. Wenn nämlich die Schritte 2 und 3 zu Beginn ausgeführt werden, können sie im weiteren Verlauf solange, wie der Inhalt von Register e unzerstört im Register X verbleibt (und solange keine Umschalttasten neu belegt werden), entfallen.

Um den Gebrauch des erweiterten Byte-Hüpfens vorzuführen, wollen wir ein 'RCL M' erzeugen.

---

*) vgl. auch Seite 35.

```
01 STO 01 (Überwacher)
02 RCL 99 (irgendein Daten-Register mit einer Adresse größer als 15)
                        eJUMP .002      [00 REG &mn     ]
                        DEL 001*        [00 REG &mn     ]
01 RDN
                        GTO .001        [01 STO 01      ]
                        DEL 001
                        SST             [01 RCL M       ]
```

*In diesem Fall ist DEL 001 nicht gleichwertig mit ← ; dagegen mit 'SST', ← .

Zwei-Byte-Funktionen - außer 'STO', 'RCL' und 'LBL' (die für '00' die Ein-Byte-Form 'STO 00', 'RCL 00' bzw. 'LBL 00' besitzen) - sind besonders einfach zu edieren, wenn man für sie '00' als Nachsilbe wählt, weil dann das dem 'eJUMP' folgende 'DEL 001' überflüssig wird:

```
01 STO 01
02 ISG 00
                        eJUMP .002      [00 REG &mn     ]
01 LASTX
                        GTO .001        [01 STO 01      ]
                        DEL 001
                        SST             [01 ISG N       ]
```

Das Verfahren ist nicht auf Zwei-Byte-Funktionen beschränkt - im folgenden erzeugen wir die als Zeile 90 des 'Henkersspieles' (Abschnitt 6C) auftretende Textzeile, die aus den 9 Bytes 'F9 40 40 40 40 40 40 43 4C 5F' besteht.

```
90 STO 01
91 "ABCDEFGHI"
                        eJUMP .091      [00 REG &mn     ]
                        DEL 009         [00 REG &mn     ]
01 +
02 +
03 +
04 +
05 +
06 +
07 /
08 %
09 DEC
                        GTO .090        [01 STO 01      ]
                        DEL 001
                        SST             ["@@@@@@CL_"    ]
```

5H. Der Textgehilfe

Die Erzeugung von Textzeilen ist ein spannender Vorgang. Wenn der Benutzer den Anfangsbuchstaben im PGRM-ALPHA-Modus eintastet, schreibt der Prozessor ein 'F1'-Text-Byte und gleich dahinter das Byte des Zeichens in den Speicher. Werden weitere Zeichen eingetastet, muß der Prozessor nicht nur diese hinzufügen, sondern auch jedesmal das Text-Byte auf den neuesten Stand bringen. Die Information, die

der Prozessor benötigt, um mit diesem Vorgang Schritt zu halten, wird während der Eingabe der Textzeile im Register Q aufgezeichnet. Das erste Nybble von Q ist die gerade vorliegende Anzahl von Zeichen in der Zeichen-Kette; die letzten vier Nybbles enthalten die Adresse des letzten eingegebenen Zeichens. Während die Zeile entsteht, ist Flag 45, das 'Daten-Eingabe-Flag' gesetzt. Ist Flag 45 durch irgendwelche Tasten, die die Eingabe von Zeichen beenden, gelöscht, kann man normalerweise keine Zeichen mehr der Zeile anfügen. Es gibt daher keine Möglichkeit, Zeichen in einer vorhandenen Textzeile abzuändern, es sei denn, man löscht die ganze Zeile und beginnt neu.

Aber die synthetische Programmierung führt uns ein weiteres Mal in dunkles, unerforschtes Gebiet. Wir können jedes Flag nach Belieben an- und ausschalten (siehe Abschnitt 6F), indem wir die passende NNZ ins Register d bringen. Insbesondere können wir bei Verwendung von 'STO d' gleichzeitig die Flags 45 (Daten-Eingabe), 48 (ALPHA) und 52 (PRGM) setzen und daraufhin tatsächlich Zeichen an eine vorhandene Textzeile anhängen. Fernerhin werden wir erkennen, daß sich jede Folge von Programm-Bytes in eine Textzeile verwandeln läßt und umgekehrt!

Doch ein Wort zur Warnung. Wir begeben uns in 'tiefe Wasser', so daß die nachstehenden Anweisungen sorgfältig ausgeführt werden müssen, um HP-41C-GAU's, die für seine Wiederbelebung die Entfernung der Batterie erzwingen, zu vermeiden. *Darüberhinaus werden sich nicht alle HP-41C's in genau gleicher Weise verhalten.* Seien Sie also auf Abenteuer eingestimmt.

Auf den Vorgang des Ablegens der entscheidenden NNZ im Register d wollen wir uns unter der Bezeichnung 'Textgehilfe' beziehen; die NNZ selbst soll 'ZTG' (Zahl des Textgehilfen) genannt werden. Eine geeignete 'ZTG' ist jede NNZ, die '0' als erstes Nybble und '488' als letzte drei Nybbles besitzt. Die '488' rührt von den drei Bits des Registers d, die den Flags 45, 48 und 52 entsprechen, her. Weil die Speicherung ins Register d alle 56 Flags beeinflußt, ist es naheliegend, die nicht weiter festgelegten Nybbles der 'ZTG' so zu wählen, daß ein benutzerfreundliches Anzeigeformat entsteht. Die in den nachstehenden Beispielen verwendete NNZ '00 00 02 3C 04 84 88' spielt die Rolle einer 'ZTG', die zugleich die Flags 26 (Tonsignal), 27 (USER-Modus), 28 und 29, das FIX 4-Format sowie den DEG-Modus setzt.

Benutzen Sie "CODE", um die 'ZTG' zu erzeugen:

```
                        XEQ "CODE"      ["CODE=?"   ]
"00000230048488"
                        R/S             ["✱⁄I✱✱  "]
                        ASTO 00
```

Das 'ASTO 00' sichert die 'ZTG' für fernere Verwendung. Wenn Sie aber die 'ZTG' aufrufen, nehmen Sie 'CLA', 'ARCL 00' und 'RCL M' statt einfach 'RCL 00', denn Sie müssen das Alpha-Ketten-Merkmal '10', welches 'ASTO 00' der NNZ vor-

angestellt hat (vgl. Abschnitt 5B), wieder fortschaffen.

Führen Sie jetzt 'GTO ..' aus, um ein neues Programm zu beginnen. Gehen Sie in den PRGM-Modus über, und tasten Sie die Textzeile "ABC" ein. Wenn Sie 'ALPHA' nach der Eingabe des "C" wieder ausschalten, ist die Zeile 'abgeschlossen'. Doch wenn Sie als nächstes auch den PGRM-Modus ausschalten und dann 'STO d' (mit der 'ZTG' im X-Register) ausführen, werden Sie die Zeile '01 "ABC"' *im PRGM-ALPHA-Modus* erblicken; und sobald Sie nun irgendein Zeichen eingeben, wird dieses (einschließlich der "_"-Aufforderung zu weiteren Eingaben) den Zeichen der Textzeile angehängt. Der Druck auf die Korrekturtaste löscht wie gewohnt die Zeichen der Kette von hinten, einschließlich des ursprünglichen "ABC", falls man es wünscht. Allerdings klappt dieser Trick nur, wenn der Inhalt des Registers Q unzerstört bleibt, wie es hier der Fall war, weil der anfängliche Eintrag abgeschlossen wurde. Im allgemeinen führt die Rückkehr zu einer edierten Textzeile mit Hilfe des Textgehilfen, sofern zwischendurch einige andere Befehle ausgeführt werden, statt zur Anfügung weiterer Zeichen mit hoher Wahrscheinlichkeit zu einem GAU. Der Prozessor bedarf einwandfreier Auskunft im Register Q, um den Aufbau einer Textzeile fortzusetzen, andernfalls gerät er hoffnungslos in die Irre.

Somit müssen wir zu einem noch seltsameren Verfahren Zuflucht nehmen. Statt zu versuchen, in einer bestehenden Textzeile etwas ans Ende zu hängen, werden wir das erste Nybble des Registers Q vorsätzlich auf Null setzen, indem wir die 'ZTG' unmittelbar, bevor wir sie im Register d ablegen, nach Q bringen. Merkwürdigerweise stellt nämlich der Prozessor mit der 'ZTG' in Q Textzeilen aus Bytes zusammen, *die schon im Speicher vorhanden sind.* Sobald das Anfangsbyte kein 'Fn' mehr ist, ersetzt es der Prozessor einfach durch ein 'Fn' und ändert 'n' jedesmal, wenn Zeichen zur Textzeile zugefügt oder von ihr fortgenommen werden.

All dies wird am besten durch Beispiele erklärt. Ein weiteres 'Kürzel' ist unentbehrlich:

'TG .ℓmn' bedeutet
1. PRGM aus
2. 'ZTG' ins Register X (z.B.: CLA, ARCL 00, RCL M)
3. GTO .ℓmn
4. STO Q
5. STO d

Löschen Sie jetzt das noch bestehende Programm, und tasten Sie ein:

```
01 "ABC"
02 X<Y?
03 X>Y?
04 X<=Y?
```

```
                    PACK
                    TG .001          [01 "ABC"       ]
"A"-Taste drücken                    [01 "A_"        ]
"A"-Taste drücken                    [01 "AB_"       ]
"A"-Taste drücken                    [01 "ABC_"      ]
"A"-Taste drücken                    [01 "ABCD_"     ]
"A"-Taste drücken                    [01 "ABCDE_"    ]
```

"A"-Taste drücken		[01 "ABCDEF_"]
	SST	[.END.]

Jeder Druck auf die "A"-Taste hat ein vorhandenes Programm-Byte ergriffen und der Textzeile einverleibt. Alle anderen 'Buchstaben-Tasten', also alle Tasten außer 'SST', 'BST', 'SHIFT', 'R/S' und '←' hätten dasselbe wie die "A"-Taste bewirkt. Hätten wir mehr als sechsmal die "A"-Taste betätigt, wäre das '.END.' selbst vom Text verschlungen worden. Doch einige HP-41C's nehmen bei diesem Verfahren nicht mehr als 7 Zeichen in die Textzeile auf; ein achter Tastendruck führt bei ihnen zu einem GAU. Bedauerlicherweise gibt es nur einen Weg, um festzustellen, was Sie für einen HP-41C haben, nämlich den, mit dem Textgehilfen eine Textzeile aus 8 Zeichen zu versuchen.

Die Korrekturtaste arbeitet mit dem Textgehilfen in eindeutiger Weise zusammen. Sobald man sie irgendwann nach der Eröffnung einer Textzeile (mit mindestens einem Zeichen) drückt, wird der Textzeile das äußerste rechte Zeichen 'entzogen', *verbleibt* aber als Einzelzeile (oder Vorsilbe) im Speicher. Sie haben noch Zeile '01 "ABCDEF"' im Speicher. Versuchen Sie daher folgendes:

	TG .001	[01 "ABCDEF"]
"A" sechsmal drücken		[01 "ABCDEF_"]
	←	[01 "ABCDE_"]
	SST	[02 X<=Y?]

Wie Sie sehen, ist das "F"-Byte nicht gelöscht, sondern nur aus der Textzeile gestoßen worden, um seine Rolle als Zeile '02 X≤Y?' wiederaufzunehmen. Wenn Sie nicht gleich die 'SST'-Taste gedrückt, sondern erst noch mit der Korrekturtaste "E", "D", "C" und "B" aus der Zeile entfernt hätten, wären die entsprechenden Bytes ebenfalls im Speicher zurückgeblieben. Doch eine Streichung mehr, nämlich die mit '01 "A_"' in der Anzeige, hätte das "A" und das 'F1'-Text-Byte völlig im Speicher gelöscht.

Wenn man die Korrekturtaste als *erste* Taste nach dem 'STO d' des Textgehilfen drückt, wird das erste Byte der angezeigten Zeile durch ein 'FF' ersetzt, und die nächsten 15 Bytes des Programms gelangen auf einen Schlag in eine Textzeile. Darauffolgende Betätigungen der Korrekturtaste kehren den Vorgang um, indem sie Schritt für Schritt den äußersten rechten Zeichen wieder ihre ursprüngliche Rolle zuweisen und die Textzeile entsprechend verkürzen.

Der Textgehilfe vereinfacht die Erzeugung von Textzeilen, die nicht-tastbare Zeichen enthalten, ganz erheblich. Jedes gewünschte Zeichen wird als Ein-Byte-Programmzeile eingetastet. Der ganzen Kette sollte eine 'Strohmann'-Ein-Byte-Zeile, die später in das Text-Byte übergeht, voranstehen. Haben wir die zukünftigen Zeichen erst einmal nach unserer Vorstellung versammelt, nehmen wir einfach den Textgehilfen, um die Ein-Byte-Zeilen in eine Textzeile zu verwandeln, so wie man Perlen auf eine Schnur reiht:

```
01 X<>Y (Strohmann)
02 LBL 11
03 X<=Y?
04 RCL 08
05 E↑X-1
06 RCL 09
                              TG .001
5 mal Buchstaben-Taste
                              ALPHA        [01 "µF(X)"   ]
```

Falls Sie eine Zeile von mehr als 7 Zeichen haben wollen und Ihr HP-41C die Zusammenarbeit verweigert, können Sie im Anschluß an den Textgehilfen sofort die Korrekturtaste drücken und dieselbe Taste gleich weiter dazu benutzen, die unerwünscht mit in die Zeile gelangten Zeichen zu entfernen, bis die Zeile schließlich so zurückbleibt, wie Sie sie zu erzeugen beabsichtigten.

Der Textgehilfe kann auch zur Erzeugung synthetischer Funktionen verwendet werden, indem man mit seiner Unterstützung eine vorhandene Vorsilbe in eine Textzeile bringt, eine neue Nachsilbe dahintersetzt und ihn dann ein zweites Mal benutzt, um die Vorsilbe wieder aus der Textzeile zu drängen. Dieses Verfahren ist für zwei-Byte-Funktionen gewöhnlich etwas umständlicher als die Verwendung des erweiterten Byte-Hüpfers. Doch werden wir im nächsten Abschnitt sehen, daß ein Zusammenspiel von Textgehilfen und Q-Boten einen sauberen Weg eröffnet, synthetische Zeilen unmittelbar aus im X-Register befindlichen NNZ's herzustellen.

Die Bildung von Programmzeilen des Typs 4 ist eine vergnügliche Übung mit dem Textgehilfen:

```
01 1 E25
                              TG .001      [01 1 E25     ]
eine Buchstaben-Taste drücken              [01 "*"       ]
Korrekturtaste drücken,    SST             [01  E25      ]
```

Dieser Vorgang läuft deswegen so reibungslos, weil die automatisch vor das '1'-Byte gesetzte Null als 'Strohmann'-Byte dienend in ein 'F1' überwechselt und dann zusammen mit der '1' gelöscht wird. Wollen Sie die '1' aus einer schon vorhandenen Zeile eines verdichteten Programms entfernen, müssen Sie erst ein Strohmann-Byte einschieben und vor der Benutzung des Textgehilfen auch noch ein 'PACK' ausführen.

5I. Der Q-Bote

Im Abschnitt 4C haben wir vorgeführt, wie das Register Q vorübergehend Alpha-Ketten, die weder im Programm noch im Alpha-Register auftauchen, aufnimmt. Mittels einer weiteren spitzfindigen HP-41C-Bedienung können wir dieses Verhalten dazu verwerten, bei der Erzeugung von Textzeilen bis zur Länge von sieben Zeichen Hilfe zu leisten. Wir benötigen jetzt die in Abschnitt 5F getätigten Tastenzuweisungen für 'STO Q' und den 'Q-Boten' (Vorsilbe 4, Nachsilbe 25).

Der 'Q-Bote' ist auf den ersten Blick nicht mehr als die einer Benutzertaste zugewiesene Zahl '9', also Byte '19'. Tatsächlich hätte auch jedes andere Byte zwischen '10' und '1C' als Q-Bote gearbeitet, doch ist unsere Zuweisung gut zu gebrauchen und liefert überdies die leicht lesbare Anzeige "0D", wenn die Taste gedrückt und gehalten wird. Wenn der Q-Bote bei 'PRGM aus' aufgerufen wird, schreibt er eine '9' ins X-Register. Im PRGM-Modus dagegen ediert er gleich zwei Programmzeilen: die erste ist der Eintrag einer '9', doch die zweite ist eine Textzeile, die das enthält, was auch immer an Zeichen im Register Q vorhanden war. Um sich das vor Augen zu führen, befehlen Sie 'PACK' (PRGM an), indem Sie 'XEQ "PACK"' buchstabengetreu ausführen und dann den Q-Boten aufrufen. Sie werden anschließend eine Zeile mit einer '9' gefolgt von einer Textzeile mit dem Inhalt "PACK" erblicken. Die Buchstaben "P-A-C-K" sind durch die Anweisung 'XEQ "PACK"' ins Register Q gelangt und von dort mit Hilfe des Q-Boten in die Textzeile übertragen worden.

Weil uns 'STO Q' zur Verfügung steht, sind wir nicht auf 'buchstabierfähige' Zeichenfolgen angewiesen - jede beliebige NNZ kann nach Q gebracht werden. Zur Vervollständigung unserer Vielseitigkeit wird "CODE" dienen, womit wir die NNZ's bequem herstellen können. Wir wollen jetzt eine Textzeile aus sieben willkürlichen Bytes zusammenstellen:

1. Verwenden Sie "CODE", um eine Kette aus sieben Zeichen Ihrer Wahl zu bilden; geben Sie die Bytes in umgekehrter Reihenfolge ein! Falls Sie nur 'n' Bytes ($n < 7$) haben möchten, müssen die ersten $7-n$ Bytes, die auf "CODE=?" hin eingetastet werden, Nullen ("00") sein. Berücksichtigen Sie auch, daß Textzeilen, die mit einer oder mehreren Nullen *aufhören*, nicht direkt mit diesem Verfahren erzeugt werden können.
2. Führen Sie ein 'GTO' aus, welches zu der Zeile führt, die der Stelle, an der die neue Textzeile entstehen soll, vorangeht.
3. Schalten Sie 'PRGM aus', und drücken Sie 'STO Q'.
4. Schalten Sie wieder 'PRGM ein', und rufen Sie den Q-Boten auf. Löschen Sie die Zeile mit der '9', und sehen Sie sich dann mit 'SST' die neue Textzeile an.

Eine Musteranwendung des Q-Boten soll hier die Erzeugung der Zeile 81 aus dem Programm 'Henkersspiel' in Abschnitt 6C sein. Der Kode für diese Zeile lautet 'F5 60 06 04 05 01':

```
                        XEQ "CODE"      ["CODE=?"       ]
"00000105040660"
                        R/S             ["XXXXT"        ]
                        GTO .000*
                        STO Q
                        PRGM ein
```

```
                    Q-Bote          [01 9           ]
                    DEL 001
                    SST             [01 "ᵀᵀᵀᵀᵀ"     ]
```

*Bei der tatsächlichen Herstellung des Programmes wäre dies ein 'GTO .080' gewesen.

Der Q-Bote bringt seine Textzeile an jeder gewünschten Stelle ins Programm ohne Rücksicht auf die Register-Grenzen oder die für die etwaige Benutzung von "REG" benötigten Adressen. Verwendet man den Textgehilfen, um Textzeilen, die durch den Q-Boten entstanden sind, in getrennte Programmzeilen zu verwandeln, lassen sich entweder synthetische Zeilen aus bis zu sieben Bytes oder gleichzeitig mehrere kürzere Zeilen herstellen. Wir werden die beiden Zeilen '01 "(#)"' und '02 ASTO M' zugleich erzeugen:

```
                    XEQ "CODE"      ["CODE=?"       ]
"0000759A292328"
                    R/S             ["**)#("        ]
                    GTO ..
                    STO Q
                    PRGM ein
                    Q-Bote          [01 9           ]
                    DEL 001
                    PACK
                    PRGM aus
                    TG .001
3 mal Buchstaben-Taste              [01 "(#)"       ]
                    SST             [02 ASTO M      ]
```

Soll keines der sieben Zeichen als Text in der Zeile zurückbleiben, läßt sich der Textgehilfe einsetzen, um das Text-Byte - so wie die '1' aus der Zeile '1 En' (in Abschnitt 5H) - aus der Kette zu entfernen.

Wir können über noch weitere Arten von Q-Boten verfügen, entsprechend den anderen Möglichkeiten, Programmtexte zu verwenden. Die folgenden Zuweisungen sind von Tom Cadwallader entwickelt worden:

Byte-Kode	"KA" Vor-/Nachsilbe	bringt den Q-Inhalt in eine Zeile vom Typ
04 1D	4/29	GTO (alpha)
04 1E	4/30	XEQ (alpha)
CD 00	205/00	LBL (alpha)

Die Notwendigkeit, bei der Verwendung des Q-Boten die Byte-Folgen rückwärts zu verschlüsseln, ist ein wenig unbequem. Wenn die angestrebte Kette jedoch nur aus 6 oder weniger Bytes besteht, kehrt die nachstehende Routine ihre Reihenfolge selbständig um.

```
01*LBL "REV"
02 ASTO [
03 SF 25
04 GTO IND [
05 RCL _
06 STO [
07 END
```

"REV"

20 BYTES

"REV" ist dazu ersonnen, eine Kette von bis zu 6 Zeichen im Alpha-Register umzudrehen. Um eine Kette aus dem X-Register umzudrehen, muß vor dem Aufruf 'XEQ "REV"' ein 'STO M' ausgeführt werden. Berücksichtigen Sie, daß "REV" nicht einwandfrei arbeitet, sobald irgendwo eine globale Marke steht, deren Name mit der umzudrehenden Alpha-Kette übereinstimmt; doch wird dies selten vorkommen. "REV" schlägt auch dann fehl, *wenn der Drucker angeschlossen ist.*

5J. Niederträchtige Erwiderungen des HP-41C

Wir sind soweit, daß wir dem HP-41C alles 'mitteilen' können, was wir wollen, indem wir "CODE" verwenden, um benutzerlesbare Zeichen in HP-41C-Byte-Kodes zu übersetzen. Doch zu einem richtigen Wechselgespräch gehört auch ein Programm, welches vorhandene Kodes hernimmt und sie in lesbare Zeichen umsetzt. Das nachstehend aufgelistete Programm "DECODE" kehrt die Tätigkeit von "CODE" genau um, indem es einen beliebigen 7-Byte-Code aus dem X-Register in 14 Zeichen, die im Alpha-Register abgelegt werden, umformt. Als zusätzlicher Luxus werden Doppelpunkte eingefügt, die die ausgegebenen Zeichen in Byte-Paare trennen. Die grundsätzliche Arbeit von "DECODE" ist ganz ähnlich der von "CODE". "DECODE" kann außerdem als Musterbeispiel für die in Abschnitt 4B beschriebene Verwendung von Register P als volle 7-Byte- Erweiterung des Alpha-Registers gelten.

```
01*LBL "DECODE"
02 CLA
03 ,006
04 STO L
05 X<>Y
06 GTO 14
07*LBL 13
08 STO ↑
09 "⊢:"
10 RCL ↑
11*LBL 14
12 ENTER↑
13 X<> d
14 CF 12
15 CF 13
16 CF 14
17 CF 15
18 FS?C 07
19 SF 15
20 FS?C 06
21 SF 14
22 FS?C 05
23 SF 13
24 FS? 04
25 SF 12
26 CF 08
27 CF 09
28 SF 10
29 SF 11
30 FS?C 04
31 XEQ 01
32 FS? 03
33 SF 07
34 FS? 02
35 SF 06
36 FS?C 01
37 SF 05
38 FS?C 00
39 SF 04
40 SF 02
41 SF 03
42 FS? 04
43 XEQ 02
44 FIX 5
45 ARCL L
46 X<> d
47 STO [
48 "⊢12"
49 X<> \
50 STO [
51 RCL ]
52 STO \
53 RCL ↑
54 STO ]
55 R↑
56 ISG L
57 GTO 13
58 0
59 STO ↑
60 TONE 9
61 AVIEW
62 RTN
63*LBL 01
64 FS? 13
65 CF 10
66 FS? 14
67 CF 10
68 FS? 10
69 RTN
70 CF 12
71 CF 11
72 SF 09
73 FC?C 15
74 SF 15
75 FC? 15
76 RTN
77 FC?C 14
78 SF 14
79 FC? 14
80 RTN
81 CF 13
82 RTN
83*LBL 02
84 FS? 05
85 CF 02
86 FS? 06
87 CF 02
88 FS? 02
89 RTN
90 CF 04
91 CF 03
92 SF 01
93 FC?C 07
94 SF 07
95 FC? 07
96 RTN
97 FC?C 06
98 SF 06
99 FC? 06
100 RTN
101 CF 05
102 END
```

"DECODE"

202 BYTES

Gebrauchsanweisung für "DECODE":

1. Legen Sie den zu entschlüsselnden Kode ins X-Register.
2. XEQ "DECODE".
3. Die 'Ausgabe-Zeichen' befinden sich im Alpha-Register und werden mit 'AVIEW' angezeigt.

Beispiele:

ALPHA "ABCDEFG" ALPHA, RCL M, XEQ "DECODE" ⟶ "41:42:43:44:45:46:47"
-1.234567891 E-56, XEQ "DECODE" ⟶ "91:23:45:67:89:19:44"

"DECODE" ist dazu gedacht, vollständige 7-Byte-Kodes zu behandeln, was in einem besonderen Fall, nämlich der Bestimmung der gegenwärtigen Lage des Adreßzeigers, 'zuviel des Guten' bedeutet. Ist 'RCL b' einer Taste zugewiesen, entziffern uns 'RCL b', 'XEQ "DECODE"' gewiß den Inhalt des Registers b. Gewöhnlich kümmern uns jedoch die ersten 5 Bytes des Registers b (Rücksprungadressen) wenig. "AD" (für 'Adresse') ist eine schnelle doch niederträchtige Routine, die die Kraft und Eleganz von "DECODE" der Ausführungsgeschwindigkeit opfert.

"AD" ist schnell, weil sie kurz ist, aber niederträchtig, weil sie die Anzeige-Eigenschaften von FIX 9 zur flinken Umwandlung von Hexadezimalzahlen in Zeichen bedenkenlos ausnutzt; genauer gesagt: weil die im Alpha-Register mit AVIEW betrachtete Ausgabe eine Ziffer 'A' als Doppelpunkt ":" und nicht als 'Explosion' darstellt. Schlagen Sie in Abschnitt 5A nach.

"AD" läßt die ursprüngliche Adresse im Register X zurück, so daß ein anschließendes 'STO b' den Adreßzeiger ordentlich zurücksetzt. Dieser Umstand erzwingt die andernfalls überflüssige Zeile '15 STOP', denn durch sie wird sichergestellt, daß ein Übergang in den PRGM-Modus nach dem 'STO b' die richtige Zeilen-Nummer beläßt (schlösse das 'END' die Ausführung von "AD" ab, endeten wir mit der Zeilen-Nummer '00').*)

```
01*LBL "AD"        09 "  "
02 STO [           10 X<> [
03 "┣***"          11 STO \
04 RCL d           12 ASTO L
05 FIX 9           13 RDN
06 ARCL [          14 VIEW L
07 STO d           15 STOP
08 X<> [           16 END
```

"AD"

39 BYTES

Gebrauchsanweisung für "AD":

1. RCL b (PRGM aus).
2. XEQ "AD".
3. Die Ausgabe besteht aus vier (mit 'AVIEW' angezeigten) Alpha-Zeichen, die

*) vgl. Seite 54.

den vier Ziffern des mit 'RCL b' beschafften Adreßzeigers entsprechen. Die Hexadezimalziffern, welche größer als '9' sind, werden folgendermaßen dargestellt: 'A' = ":", 'B' = ";", 'C' = "<", 'D' = "=", 'E' = ">" und 'F' = "?".

4. Um zum Ausgangsbyte (bei welchem 'RCL b' angeordnet wurde) zurückzukehren, drücken Sie 'STO b'.

5K. Kode-Speicherung

Zwei kurze Routinen werden unsere 'Bibliothek programmierender Programme' abrunden. Es gibt viele Gelegenheiten, bei denen man eine NNZ für späteren Rückruf in ein Daten-Register legen möchte, doch ist die beim Rückruf aus Daten-Registern erfolgende Normalisierung ein beträchtliches Hindernis.

Eine Möglichkeit, die NNZ ohne Normalisierung zurückzurufen, besteht darin, den Byte-Hüpfer mit ihrer Verlegung ins Alpha-Register zu beauftragen. Sei z.B. die uns interessierende NNZ im Register R_{00}, welches wir durch Entzifferung des Inhaltes von Register c als auf der Adresse '123' (hexadezimal) liegend erkennen. Dann bringen wir '1 E7' nach R_{01}, d.h. auf '124'. Als nächstes benutzen wir "CODE" zur Herstellung des Kodes '00 00 00 00 00 01 24' und führen 'STO b' aus. Wenn jetzt der Byte-Hüpfer aufgerufen wird, kopiert er wunschgemäß die NNZ von R_{00} nach Register M: die in R_{01} gespeicherte Zahl '1 E7' hat das Byte '07' auf die Adresse '0124' gelegt und sorgt so für einen Byte-Sprung der Länge sieben.

Dieses Verfahren ist natürlich ziemlich schwerfällig und kann nur von Hand durchgeführt werden. Zur automatischen Ablage und Rückgabe können wir die Routinen "CS" und "CR" benutzen. "CS" nimmt eine NNZ her, zerlegt sie in zwei Teile und wandelt jedes Stück einzeln in Alpha-Daten um, die dann herkömmlich abgespeichert werden. Dies erfordert zwei Daten-Register für die Ablage des gesamten NNZ-Kodes. "CR" kehrt den Vorgang um, indem zwei Alpha-Daten-Ketten geholt und zu einer 7-Byte-NNZ zusammengesetzt werden. Damit der Aufruf der beiden Routinen in augenfälliger Weise erfolgen kann, sollte man "CS" der Taste 'STO' und "CR" der Taste 'RCL' zuweisen. Sie laufen dann beide durch Druck auf die passende Taste und anschließende Eingabe (die während der sofort nach Aufruf beginnenden Pause erfolgen muß) der Nummer des gewünschten Daten-Registers ab. Jede Routine benutzt das gewählte und das nächsthöhere Daten-Register.

```
01*LBL "CS"        08 "┣++"          15 CLA
02 CLA             09 ,9             16 STO [
03 STO [           10 ST+ L          17 ASTO IND L
04 STO L           11 X<> \          18 CLA
05 RDN             12 ASHF           19 RDN
06 PSE             13 ASTO IND L     20 END
07 X<> L           14 ISG L
```

"CS"

40 BYTES

SIZE 002

Gebrauchsanweisung für "CS":

1. Aufruf über die Tastatur: XEQ "CS"; während der nachfolgenden Pause Daten-Register-Nummer 'mn' eingeben. Die NNZ wird in R_{mn} und R_{mn+1} abgelegt. Die Inhalte der Register X, Y und Z bleiben erhalten.
2. Aufruf als Unterprogramm: bei Aufruf muß die NNZ in X und 'mn' in Y liegen. Nach Ausführung sind die Inhalte der Register T und Z nach Z bzw. Y gerückt.

```
01*LBL "CR"          09 CLX
02 PSE               10 RCL IND L
03 STO L             11 STO \
04 CLA               12 "┣*****"
05 ARCL IND L        13 X<> \
06 "┣**"             14 CLA
07 ISG L             15 END
08 CLD
```

"CR"

37 BYTES

SIZE 002

Gebrauchsanweisung für "CR":

1. Aufruf über die Tastatur: XEQ "CR"; während der nachfolgenden Pause Daten-Register-Nummer 'mn' eingeben. Die NNZ aus R_{mn} und R_{mn+1} wird nach X gebracht und überschreibt dort 'mn'; der vor Ausführung von "CR" im Stapel befindliche Inhalt wird angehoben.
2. Aufruf als Unterprogramm: bei Aufruf muß 'mn' in X liegen. Nach Ausführung hat die NNZ 'mn' überschrieben; der restliche Stapelinhalt ist unversehrt.

Kapitel 6

ANWENDUNGEN

Dieses Kapitel soll ein 'Handbuch für mustergültige Anwendungen' der synthetischen Programmierung bilden. Es enthält zahlreiche HP-41C-Routinen, die - ähnlich den Programmen des Kapitels 5 - sowohl die erfindungsreiche Ausnutzung synthetischer Funktionen als auch ihre leistungsfähige Anwendung im Alltagsgeschäft beleuchten sollen. Die Routinen selbst verkörpern Zweck und Rechtfertigung der synthetischen Programmierung. Zunächst einmal befähigt die Verwendung synthetischer Funktionen den HP-41C dazu, viele wichtige Tätigkeiten schneller und mit weniger Programmspeicherplatz auszuführen, als er es mit den Standardfunktionen allein könnte. Beispiele dafür sind der 'SIZE-Sucher' (Abschnitt 6B) und die Behandlung von Alpha-Ketten (Abschnitt 6C). Zweitens bahnt uns die synthetische Programmierung neue Wege, die mit den normalen Funktionen überhaupt nie begangen werden könnten. Proben dafür sind die Bestimmung von Alpha-Zeichen und ihr Vergleich (Abschnitt 6D) sowie der unmittelbare Zugang zu den Programmen von Anwender-Modulen (Abschnitt 6H).

Die Sammlung der in diesem Kapitel beschriebenen Routinen bildet keineswegs ein vollständiges Verzeichnis der Nutzungsmöglichkeiten synthetischer Funktionen - es gibt keinen Katalog von Programmen, der die Fähigkeiten des HP-41C ausschöpfen könnte, schon gar nicht, wenn die synthetischen Funktionen mit einbezogen sind. Die Entwicklung von Methoden und Anwendungen der synthetischen Programmierung ist ein ständiger Vorgang. (Ein Musterbeispiel dafür ist die Entdeckung des in Abschnitt 5H beschriebenen Textgehilfen, die wir einem Druckfehler in einem vorläufigen Entwurf dieses Buches verdanken!) Wenn Sie den Stoff dieses Buches völlig durchgearbeitet haben, werden Sie in der Lage sein, die synthetischen Funktionen gewohnheitsmäßig in Ihre eigenen Programme einzubauen, so geschickt und mit wenig mehr Aufwand, als Sie das mit den Standardfunktionen tun. Vielleicht entdecken Sie sogar selbst ein paar neue 'Kunstgriffe des Gewerbes'. In dieser Hinsicht gilt die Losung 'betrachten Sie nichts als erwiesen'. Wenn Sie einen Einfall haben, erproben Sie ihn, wie abgelegen er auch immer scheinen möge. Es dauerte beispielsweise Monate weit verbreitet ausgeübter synthetischer Programmierung, bis schließlich irgendjemand bemerkte, daß das Register P als volle 7-Byte-Erweiterung des Alpha-Registers dienen kann. Weil die Anzeige höchstens 24 Zeichen vorweist, nahm man an, daß die linkerhand verschwindenden Zeichen für immer verloren seien. Doch siehe da, da waren sie - heimlich ins Register P geschlüpft.

6A. Zugriff auf das .END.

Ein Programm, welches gerade geschrieben wird, ist meistens das letzte im Speicher und enthält das '.END.'. Wird der Adreßzeiger zwischendurch auf ein anderes Programm gesetzt, gibt es nur zwei Möglichkeiten, ihn wieder auf das letzte zurückzuführen: die Verwendung von 'GTO' zusammen mit dem 'Buchstabieren' einer globalen Marke dieses Programms oder den Aufruf von 'CAT 1', um ans Ende des Programmspeichers zu gelangen. Gibt es noch keine globale Marke im letzten Programm, entfällt die erste Methode. Sind mehrere Speichererweiterungsmodule und viele Programme im HP-41C, kann der zweite Weg verdrießlich lang werden. Das Programm "EN" eröffnet eine dritte Möglichkeit, die Sie während mancher 'Edier-Stunde' als sehr bequem empfinden werden, besonders wenn Sie dabei die Programme aus Kapitel 5 verwenden, die den Adreßzeiger immer wieder versetzen.

```
01*LBL "EN"          10 SF 03
02 RCL c             11 X<> d
03 STO [             12 CLA
04 "┣♦♦♦♦♦"          13 STO [
05 X<> [             14 "┣♦♦"
06 X<> d             15 X<> \
07 CF 00             16 STO b
08 CF 01             17 END
09 SF 02
```

"EN"

45 BYTES

Gebrauchsanweisung für "EN":

1. XEQ "EN".
2. Nach Ausführung steht der Adreßzeiger am Anfang des Programmes, welches das '.END.' enthält.

Das '.END.' liegt in den Bytes 2, 1 und 0 des Registers, dessen Adresse von den letzten drei Nybbles des Registers c aufbewahrt wird. "EN" ergreift diese Adresse, sie heiße '&mn', in Zeile 02, verschiebt sie in die ersten zwei Bytes von Register d (Zeilen 03-06) und erzeugt mit ihr dort den Kode '3&mn' durch Löschen der Flags 0 & 1 bzw. Setzen der Flags 2 & 3 (Zeilen 07-10). Dieser Kode wird dann in die letzten beiden Bytes von Register N gebracht (Zeilen 11-14). Wenn er schließlich, in den Zeilen 15-16, nach Register b übertragen wird, springt der Adreßzeiger unverzüglich auf das dem .END. unmittelbar vorangehende Byte. Das Programm setzt seinen Lauf damit fort, daß es das .END. selbst ausführt, welches seinerseits die Ausführung anhält und den Zeiger an den Programmanfang zurücksetzt.

6B. Ermittlung von SIZE und andere Kunstgriffe

Eine elegante Vorführung dessen, was synthetische Funktionen an verbesserter HP-41C-Leistung erbringen, ist die folgende 'SIZE-Sucher'-Routine, die von Keith Jarett (*PPC Calculator Journal*, V7N5P57) geschrieben wurde.

```
01*LBL "S"          14 FS?C 13          27 CHS
02 "AB"             15 SF 11            28 64
03 RCL c            16 FS?C 14          29 MOD
04 X<> [            17 SF 13            30 SF 25
05 STO \            18 FS?C 15          31*LBL 14
06 ASHF             19 SF 14            32 VIEW IND X
07 "⊢***"           20 FS?C 16          33 FC? 25
08 X<> [            21 SF 15            34 RTN
09 X<> d            22 X<> d            35 64
10 FS?C 11          23 1 E3             36 +
11 SF 09            24 *                37 GTO 14
12 FS?C 12          25 INT              38 END
13 SF 10            26 DEC
```

"S" *)

76 BYTES

Gebrauchsanweisung für "S":

1. XEQ "S".
2. Nach Ausführung wird die gegenwärtige Daten-Register-Anzahl im Register X angezeigt.

Bedauerlicherweise gibt es keinen unmittelbaren Weg, um die augenblickliche Programm-/Datenspeicher-Aufteilung festzustellen. Nirgendwo im Speicher steht eine Adresse, die dem 'Kopf des Speichers' entspricht, weil er sich mit jedem Hinzufügen oder Entfernen von Speichererweiterungsmodulen ändert. Die einzige Möglichkeit, die Anzahl der Daten-Register ausfindig zu machen, ist die, nach und nach höher numerierte Daten-Register anzusprechen, bis eine "NONEXISTENT"-Meldung bekannt gibt, daß das letzte verfügbare Daten-Register vorüber ist. Dieser Weg kann auch von einem Programm beschritten werden, in welchem man sich das Fehler-Ignorierflag 25 zunutze macht, doch bei einer großen Anzahl von Daten-Registern kann es recht lange dauern, bis es am Ziel ist. Selbst die pfiffigsten solcher Programme laufen mindestens vier Sekunden. Die Routine "S" hält sich höchstens 1,5 Sekunden damit auf. Die Verbesserung erwächst aus einer Teilentzifferung des Registers c, welches einen Startwert liefert, der nur noch um Vielfache von 64, abhängig von der Anzahl der eingesteckten Module, erhöht zu werden braucht. Für die Bestimmung der Modul-Anzahl müssen höchstens vier Register geprüft werden.*)

Das 'Herz' des 'SIZE'-Suchers findet man in den Zeilen 10-26, die eine von Roger Hill entwickelte 3 Ziffern erfassende hexadezimal→dezimal Umwandlung bilden. Die drei Hexadezimalziffern von Belang sind die Ziffern 9, 10 und 11 aus Register c - die absolute Adresse von R_{00}. Die Zeilen 01-09 von "S" machen sie zu den Ziffern 3, 4 und 5 des Registers d, wo die Flags 8-19 liegen.

Betrachten Sie irgendeine R_{00}-Adresse, etwa '12A', die den Dezimalwert 256+32+10 = 298 hat; rufen Sie 'OCT' auf, um $298_{10} = 452_8$ (vgl. Anhang 1) festzustellen. Schreiben wir die beiden Zahlen '12A' und '452' gemäß der Verschlüs-

*) Anm. d. Übers.: für den HP-41CV gibt es eine auf 64 Bytes verkürzte Fassung:

```
27 SIZ          29 CHS
28 -            30 END
```

selung im HP-41C auf:

```
hexadezimal     12A = 0001 0010 1010    binär
oktal           452 = 0100 0101 0010    binär
```

Beachten Sie, daß die beiden Zahlen dieselbe Anzahl von Bits mit dem Wert '1' haben. Der Unterschied zwischen den beiden Darstellungen liegt darin, daß das erste Bit einer jeden Oktalziffer stets '0' ist, denn Oktalziffern haben einen Höchstwert von 7 (0111). Um nun ein hexadezimales Bit-Muster in ein oktales zu verwandeln, müssen wir lediglich die Werte gewisser Bits so nach links verschieben, daß 'Platz' für die am linken Ende 'oktaler Nybbles' stehenden '0'-Bits geschaffen wird. Hier ist ein zweites Beispiel, in welchem Pfeile die Verschiebung der Bits vom hexadezimalen ins oktale Muster verdeutlichen:

```
hexadezimal     1BC = 0001  1011  1100
                      ↙↙  ↙↙  ↙↓↓↓
oktal           674 = 0110  0111  0100
```

Diese Verschiebung läßt sich leicht durch offene Betätigung der Anwenderflags erreichen, wie man den Zeilen 10-21 von "S" entnimmt. Die Zeilen 22-26 vervollständigen die hexadezimal→dezimal Umwandlung, indem sie die drei Oktalziffern aus Register d im X-Register in eine ganze Dezimalzahl umsetzen.

Das Ergebnis 's' in X ist vorläufig noch die, dezimal ausgedrückte, absolute Adresse von R_{00}. Wenn N ($0 \leq N \leq 4$) Speichermodule im Rechner stecken, ist die Anzahl der Daten-Register gleich 256+N•64-s wobei 256+N•64 die dezimale Adresse des Speicherkopfes ist. Nun ist -s MOD 64 die kleinste positive Zahl t, die sich durch Addition ganzzahliger Vielfachen von 64 zu -s ergibt; sie stimmt mit (256+N•64-s)MOD 64, d.i. der Rest von 256+N•64-s bei Division durch 64, überein.*) Folglich liefern die Zeilen 27-29 den in Registern gemessenen Abstand zur nächsthöheren Speichermodul-Grenze. Die gesuchte Größe besteht also aus dieser Zahl t plus einem unbekannten Vielfachen von 64. Die Zeilen 30-37 bilden eine Methode, 'N' durch 'Versuch-und-Irrtum' zu bestimmen, indem 't' solange um 64 heraufgesetzt wird, bis ein vergebliches 'VIEW IND X' einen Fehler verursacht, der Flag 25 löscht.

Die hexadezimal→oktal→dezimal Umwandlung aus "S" kann in einer ganzen Reihe von Programmen verwendet werden. Nur eine geringfügige Abänderung von "S" ist nötig, um etwa die gegenwärtige Lage der Statistik-Register aus den ersten 3 Ziffern von Register c zu ermitteln. Ein noch anderes Beispiel ist die nächste Routine, "BYTE", die dazu gedacht ist, die Stelle, auf der augenblicklich der Adreßzeiger steht, als dezimale Byte-Zahl zu bestimmen, wobei vom Fuß des Speichers aus gezählt wird. In diesem Sinne ist '0OCO' das Byte '1'.

---

*) Anm. d. Übers.: dies schreibt man bündig in der Form

$$-s \equiv 256+N\cdot 64-s \pmod{64}$$

```
01*LBL "BYTE"       12 SF 15            23 LASTX
02 CLA              13 FS?C 18          24 FRC
03 STO [            14 SF 17            25 1 E3
04 "┣*****"         15 FS?C 19          26 *
05 X<> [            16 SF 18            27 DEC
06 X<> d            17 FS?C 20          28 7
07 FS?C 15          18 SF 19            29 *
08 SF 13            19 X<> d            30 +
09 FS?C 16          20 10               31 1343
10 SF 14            21 *                32 -
11 FS?C 17          22 INT              33 END
```

"BYTE"

69 BYTES

Gebrauchsanweisung für "BYTE":

1. 'RCL b' von Hand.
2. XEQ "BYTE".
3. Es wird die Byte-Nummer in dezimaler Darstellung ausgegeben.

Nach dem von Hand ausgeführten 'RCL b', das den augenblicklichen Adreßzeigerstand in die letzten beiden Bytes des X-Registers setzt, bringen die Zeilen 01-06 von "BYTE" diese Information von dort in die Nybbles Nr. 3-6 (Flags 08-23) des Registers d. Die erste Ziffer (Nybble Nr. 3) ist eine Byte-Nummer, die niemals 6 übersteigt; sie kann also unverändert bleiben ($6_{16} = 6_{10} = 6_8$; vgl. Anhang 1). Die verbleibenden 3 Ziffern (Nybbles 4-6) beinhalten eine absolute Registeradresse, die höchstens 1FF, oktal 777, sein kann (vorausgesetzt, es gibt mindestens ein Daten-Register). Daher ist Flag 12 stets Null. Die Zeilen 07-21 führen die hexadezimal→oktal Umwandlung durch und bringen eine Zahl der Form 'n,abc' ins X-Register; hierbei ist n die Byte-Nummer und 'abc' die Oktaldarstellung der Registeradresse. Die Zeile 22 trennt 'n' ab, während die oktal→dezimal Umsetzung von 'abc' in den Zeilen 23-27 stattfindet. Die Zeilen 28-30 berechnen n+7•abc. Weil aber vom Fuß des normalen Programmspeichers, Byte '00C0', aus gemessen werden soll, bereiten die Zeilen 31-32 die Ausgabe entsprechend auf, indem sie 1343 abziehen.

Es gibt zwei auf der Hand liegende Anwendungen von "BYTE", die beide den zweimaligen Aufruf der Routine erfordern. Es ist klar, daß die Kenntnis über die Byte-Nummer einer Einzeladresse längst nicht so nützlich ist wie die über den Byte-Abstand zweier Speicherplätze. "BYTE" bewahrt den ursprünglichen Inhalt von Register X, der dort vor dem Aufruf von 'RCL b' lagerte, im Register Y auf, so daß er nach Beendigung der Routine direkt über deren Ausgabewert steht. Daher liefert die Anweisungsfolge

```
GTO 'Punkt A'
RCL b
GTO 'Punkt B'
RCL b
XEQ "BYTE"
```

```
X<>Y
XEQ "BYTE"
-
```

die von Hand oder auch durch ein Programm ausgeführt werden kann, den Byte-Abstand zwischen 'Punkt A' und 'Punkt B' des Speichers. Die erste überzeugende Anwendung dieses Verfahrens ist die, als 'Punkt B' die erste Zeile eines Programms P zu wählen und als 'Punkt A' die erste Zeile des auf P unmittelbar folgenden Programms weiter unten im Speicher zu nehmen. Die ermittelte Byte-Differenz ist dann die Länge des Programms P, so wie sie auch von 'CAT 1' geliefert wird, wenn der Drucker angeschlossen ist. Die zweite sinnvolle Nutzung ist die, den Byte-Abstand zwischen einem 'GTO' und dem angesprungenen 'LBL' zu messen und zu sehen, ob er kleiner oder größer als 112 ist, um eine vernünftige Entscheidung über die Wahl des 'GTO'- bzw. 'LBL'-Typs zu treffen (vgl. Abschnitt 2C). Die einzige andere Möglichkeit, diese Ergebnisse zu erzielen, ist die, mühselig Zeile für Zeile die Programm-Bytes abzuzählen.

6C. Spaß und Spiel im Alpha-Register

Die vielleicht nützlichsten synthetischen Funktionen sind jene, die auf das Alpha-Register zugreifen, wie 'STO M', 'RCL N' oder 'X<>IND O'. Die Tatsache, daß die Nachsilben 'M', 'N', 'O' und 'P' der Vorsilbe einer jeden normalen Daten-Register-Funktion angeheftet werden können, eröffnet die Möglichkeit, das Alpha-Register wie vier (mit einiger Einschränkung bezüglich P) zusätzliche Daten-Register zu behandeln. Das ist offenkundig vorteilhaft, sobald der Speicherplatz begrenzt ist - der Einsatz des Alpha-Registers macht vier gewöhnliche Register für weitere Programm- oder Datenspeicherung frei. Den besten Gebrauch von diesen 'extra' Daten-Registern macht man, wenn man sie zu 'Notiz'-Zwecken verwendet (fast wie eine Erweiterung des RPN-Stapels), und vorübergehend beliebige Informationen in ihnen ablegt oder verarbeitet. Beispiele sind Schleifensteuerung und indirekte Adressierung ('ISG M', 'DSE O', 'STO+ IND N' usw.), Summationen ('STO+M', 'RCL-N' usw.) und zeitweilige Ablage numerischer Zwischenergebnisse. Berücksichtigen Sie, daß ein einfaches 'CLA' alle vier Zusatzregister gleichzeitig löscht. Die Alpha- (und die Stapel-) Register können nur dann indirekt angesprochen werden, wenn man den außergewöhnlichen Weg beschreitet, den Inhalt von Register c den gewünschten Verhältnissen anzupassen und eines der Zustandsregister (jedes außer T ist geeignet) zu R_{00} zu machen, doch das ist selten zweckmäßig.

Die Alpha-Register-Zugriffsfunktionen bringen, wenn sie vereint mit den gewöhnlichen Alpha-Funktionen 'APPEND', 'ASTO', 'ARCL', 'ASHF' und 'CLA' eingesetzt werden, eine Geschwindigkeit und Geschmeidigkeit in die Behandlung von Alpha-Ketten, welche die Möglichkeiten, die man mit den Standard-Funktionen allein hat, weit übertreffen. Betrachten wir als Beispiel zunächst die Aufgabe, ein einzelnes Zei-

chen aus einer Alpha-Kette auszusondern, wie es in einer Vielzahl von Wort-Ratespielen vorkommen kann. Hier eine Routine, die das Verlangte leistet, also das 'n-te' Zeichen (gezählt von links) einer Kette von höchstens sechs Zeichen allein im Alpha-Register zurückläßt, wobei nur herkömmliche Funktionen mitwirken:

```
01*LBL A        07 1            13 ASTO Y
02 7            08 +            14 ASHF
03 -            09 ASTO Y       15 ISG X
04 CHS          10*LBL 01       16 GTO 01
05 1 E3         11 "*"          17 AVIEW
06 /            12 ARCL Y       18 .END.
```

(6C-1)

Damit die Routine arbeitet, müssen der Benutzer oder ein Programm das 'n' ins Register X setzen. Dann sondert 'XEQ "A"' das 'n-te' Zeichen im Alpha-Register aus. Zwei Punkte lassen sie jedoch recht unbefriedigend erscheinen: erstens ist sie verhältnismäßig langsam, denn sie braucht, abhängig vom Wert 'n', 0.9 bis 2.1 Sekunden Laufzeit; zweitens kann man sie nicht unmittelbar auf Ketten von mehr als sechs Zeichen ausdehnen. Wenn die zu verarbeitenden Ketten mehr als sechs Zeichen haben können, hat das Programm keine Möglichkeit zu erkennen, wo das 'erste' Zeichen im Alpha-Register liegt. Diese Schwierigkeit kann einigermaßen dadurch überwunden werden, daß man die Zeichen von rechts nach links abzählt, so daß 'n=1' dem letzten (also äußersten rechten) Zeichen der Kette entspricht. Dann kann man die Kette in vier Teilketten der Länge 6 aufteilen, von denen schließlich die richtige, abhängig von 'n', von der Routine 6C-1 nach dem gewünschten Zeichen durchsucht wird. Doch die synthetischen Funktionen geben uns, wie angekündigt, bessere Methoden an die Hand.

Die unsichtbaren Grenzen zwischen den Registern M, N, O und P vereinfachen die Aufgabe, die Alpha-Ketten zu zerhacken, erheblich. Wir müssen lediglich ein Verfahren finden, welches die Ketten umherschiebt, so daß das gesuchte Zeichen an einer Registergrenze zu liegen kommt. Betrachten Sie die Folge 'CLX', 'FIX 4', 'ARCL X'. Nach Ausführung dieser Befehle enthält das Alpha-Register seinen ursprünglichen Inhalt, jedoch nach links verschoben, weil die 6 Zeichen "0,0000" (das Komma zählt mit) angehängt wurden. Hätten wir 'FIX 6' statt 'FIX 4' gewählt, wäre die ursprüngliche Kette um 8 Stellen nach links gerückt. Somit haben wir ein Verfahren zur Hand, welches nicht auf Wiederholungen beruht (daher sehr schnell ist) und die Alpha-Ketten um eine veränderliche Stellenzahl verschiebt. Es wird in der nachstehenden Fassung von "ISO" eingesetzt. Wenn Sie in Einzelschritten durch das Programm gehen, während sich der HP-41C im ALPHA-Modus befindet, sehen Sie, wie die Zeichen umhergeschoben und trennscharf gelöscht werden, um ein einzelnes Zeichen zurückzulassen.

```
01*LBL "ISO"        08 X<> J
02 10               09 "├↑"
03 -                10 X<> J
04 CHS              11 CLA
05 SCI IND X        12 STO [
06 ARCL X           13 AVIEW
07 CLX              14 END
```

"ISO"

30 BYTES

Gebrauchsanweisung für "ISO":

1. Beginnen Sie mit einer Kette von bis zu 10 Zeichen im Alpha-Register.
2. Legen Sie eine Zahl 'n' zwischen 1 und 10 ins X-Register.
3. XEQ "ISO".
4. Nach Ausführung bleibt das 'n-te' Zeichen der ursprünglichen Kette zurück. 'n' wird dabei von rechts gezählt.

Diese Routine ist, verglichen mit Routine 6C-1, sowohl kürzer als auch schneller, denn sie benötigt, unabhängig von 'n' nur 0.8 Sekunden zur Ausführung. Statt 'FIX IND X' wird 'SCI IND X' (Zeile 05) verwendet, wodurch Verschiebungen zwischen 4 (für n=10) und 13 (für n=1) Zeichen ermöglicht werden. "ISO" hat den Nachteil, den Anzeigemodus des HP-41C zu verändern, doch kann dieser Mangel leicht dadurch behoben werden, daß man 4 zusätzliche Programmbytes bewilligt und die Zeilen 05 und 06 durch die folgenden Befehle ersetzt:

```
05 X<>d
06 SCI IND d
07 ARCL d
08 X<>d
```

Ähnliche Vorgänge findet man in der nächsten Routine, "SUB", die dazu dient, ein Zeichen in einer Alpha-Kette, die ansonsten unversehrt bleibt, zu ersetzen:

```
01*LBL "SUB"    12 "├↑"         23 X<> J        34 ARCL X
02 10           13 X<> T        24 LASTX        35 R↑
03 -            14 X<> J        25 X<> ↑        36 STO d
04 CHS          15 "├======"    26 X<> T        37 CLX
05 RCL d        16 CLX          27 9            38 X<> J
06 SCI IND Y    17 X<> \        28 -            39 STO [
07 ARCL Y       18 STO [        29 CHS          40 CLX
08 RCL ↑        19 CLX          30 FIX 0        41 X<> ↑
09 STO L        20 X<> J        31 RND          42 STO \
10 CLX          21 STO \        32 CF 29        43 AVIEW
11 X<> J        22 X<> T        33 10↑X         44 END
```

"SUB"

86 BYTES

Gebrauchsanweisung für "SUB":

1. Beginnen Sie mit einer Alpha-Kette von bis zu 10 Zeichen.
2. Bringen Sie ein Alpha-Zeichen ins Register Y.

3. Legen Sie eine Zahl 'n' zwischen 1 und 10 ins X-Register.
4. XEQ "SUB".
5. Nach Ausführung hat das Zeichen aus Y das 'n-te' Zeichen in der Alpha-Kette ersetzt. 'n' wird von rechts gezählt.

Die Zeilen 30-34 von "SUB" enthalten eine andere Art der veränderlichen Zeichenverschiebung, die mit Hilfe der Funktion '10^X' vorgenommen wird, um eine Zahl, die aus 'x+1' Zeichen besteht, ins Register X zu bringen.

Das nachstehend aufgelistete 'Henkersspiel' ("HM" für Hangman) führt eine sinnreiche Anwendung der durch "ISO" und "SUB" ermöglichten Alpha-Ketten-Behandlungen vor. In den Zeilen 169-183 bzw. 114-168 findet man Lesarten dieser Routinen.

```
01*LBL "HM"       47 RCL d          93 ASTO 04        139 X<> \
02 0              48 AVIEW          94 GTO 01         140 STO [
03 STO d          49 STO d          95*LBL 03         141 CLX
04 .009           50*LBL 02         96 ISG 08         142 X<> ]
05 STO 07         51 FS? IND 06     97 GTO 05         143 STO \
06 FIX 0          52 GTO 04         98 "  **DONE**"   144 X<> T
07 SF 26          53 RCL 05         99 AVIEW          145 X<> ]
08 "WORD?"        54 CLA           100 TONE 3         146 LASTX
09 AON            55 ARCL 00       101 TONE 4         147 X<> †
10 STOP           56 ARCL 01       102 TONE 5         148 X<> T
11 "├             57 RCL 06        103 TONE 8         149 9
12 ASTO 00        58 INT           104 TONE 7         150 -
13 ASHF           59 XEQ 08        105 TONE 8         151 CHS
14 ASTO X         60 ASTO X        106 CLA            152 FIX 0
15 CLA            61 X=Y?          107 PSE            153 RND
16 ARCL X         62 XEQ 03        108 RCL 07         154 CF 29
17 "├††††"        63*LBL 04        109 INT            155 10†X
18 RCL \          64 ISG 06        110 ARCL X         156 ARCL X
19 CLA            65 GTO 02        111 "├ WRONG."     157 R↑
20 STO [          66 FS?C 19       112 AOFF           158 STO d
21 ASTO 01        67 GTO 01        113 PROMPT         159 CLX
22 "---"          68 ISG 07        114*LBL 05         160 X<> ]
23 ASTO 03        69 GTO 06        115 SF IND 06      161 STO [
24 ARCL 03        70 "ARRRRGGH..." 116 SF 19          162 CLX
25 ASTO 02        71 AVIEW         117 RCL 06         163 X<> †
26 CLA            72 TONE 0        118 INT            164 STO \
27 ASTO 04        73 TONE 0        119 CLA            165 ASTO 02
28 1,009          74 PSE           120 ARCL 02        166 ASHF
29 STO 08         75 "WORD IS: "   121 ARCL 03        167 ASTO 03
30 STO 06         76 ARCL 00       122 10             168 RTN
31 SF 19          77 ARCL 01       123 -              169*LBL 08
32 " "            78 AOFF          124 CHS            170 10
33 ASTO 05        79 PROMPT        125 RCL d          171 -
34 GTO 02         80*LBL 06        126 SCI IND Y      172 CHS
35*LBL 01         81 "┌α8×"        127 ARCL Y         173 X<> d
36 1,009          82 10            128 RCL †         174 SCI IND d
37 STO 06         83 RCL 07        129 STO L          175 ARCL d
38 CLA            84 INT           130 CLX            176 X<> d
39 ARCL 02        85 -             131 X<> ]          177 CLX
40 ARCL 03        86 XEQ 08        132 "├†"           178 X<> ]
41 "├ "           87 ASTO X        133 X<> T          179 "├†"
42 ARCL 04        88 RCL 07        134 X<> ]          180 X<> ]
43 TONE 9         89 INT           135 CLX            181 CLA
44 CLD            90 "@@@@@@CL_"   136 FIX 4         182 STO [
45 STOP           91 XEQ 08        137 ARCL X         183 END
46 ASTO 05        92 ARCL Y        138 CLX
```

"HM"

386 BYTES

SIZE 009

Gebrauchsanweisung für "HM":

1. XEQ "HM".
2. Der erste Spieler tastet ein Wort bis zu neun Buchstaben ein; R/S.
3. Wenn der (erste) Ton erklingt, erscheinen in der Anzeige ebensoviele Gedankenstriche, wie das unbekannte Wort Buchstaben enthält. Der zweite Spieler rät einen Buchstaben, drückt die entsprechende Taste und dann R/S.
4. Beim nächsten Ton ergibt sich eine Anzeige, bei der an allen Stellen, an denen der vermutete Buchstabe tatsächlich auftritt, die Gedankenstriche durch diesen Buchstaben ersetzt sind. Ist der Buchstabe kein Bestandteil des Wortes, wird dem rechts in der Anzeige entstehenden 'Galgen' "@" ein Teilstück zugefügt, und wenn er errichtet ist, der 'Delinquent' "⍋" daneben nach und nach zur Vollstreckung des Urteils "ARRRRGGH..." aufgestellt. Solange man noch nicht 'hängt', wird das Spiel mit Schritt 3 fortgesetzt.
5. Wenn das ganze Wort mit weniger als 10 falschen Vermutungen geraten wurde, wird "**DONE**" angezeigt und anschließend die Gesamtzahl der fehlgeschlagenen Versuche bekanntgegeben.
6. Beim 10. Mißgriff gibt es keine Gnade. Der zweite Spieler wird gehängt und der Trauergemeinde das unbekannte Wort aufgedeckt.

"HM" arbeitet mit Wörtern bis zu 9 Buchstaben. Falls der erste Spieler weniger als 9 Buchstaben eintastet, füllt das Programm das Wort mit Leerstellen auf (Zeilen 11-21), 'vermutet' anschließend selbst das Zeichen 'Leerstelle' (Byte '20') in derselben Weise, in der das der zweite Spieler täte, um diesem dann die richtige Anzahl unbekannter Buchstaben anzuzeigen.

Jetzt noch ein paar die synthetische Programmierung betreffenden Bemerkungen zu "HM": Die Zeilen 72 und 73 enthalten 'TONE 10', hexadezimal 'F9 0A', der mit dem Byte-Hüpfer erzeugt werden kann (Abschnitt 3B). Die Herstellung der Zeile 81,

'F5 60 06 04 05 01',

ist schon in Abschnitt 5I beschrieben worden; Zeile 90,

'F9 40 40 40 40 40 40 43 4C 5F',

wurde als Beispiel für erweitertes Byte-Hüpfen in Abschnitt 5G behandelt. Der Trick, mit dem die vermuteten Buchstaben 'graugansartig' durch die Anzeige getrieben werden (Zeilen 47-49), wird in Abschnitt 7B verraten.

Daten-Register werden von "HM" wie folgt verwendet:

R_{00} & R_{01}	zu erratendes Wort
R_{02} & R_{03}	gegenwärtiger Rate-Zustand des Wortes
R_{04}	'Galgen' und 'Delinquent'
R_{05}	zuletzt vermuteter Buchstabe
R_{06}	Schleifenzähler
R_{07}	Zähler für Fehlschläge
R_{08}	Zähler für richtige Vermutungen

6D. Zeichenerkennung

Obwohl ein Benutzer einfach nur auf eine alphanumerische Anzeige zu blicken braucht, um den Inhalt zu lesen, hat der HP-41C selbst keine Möglichkeit festzustellen, welche Zeichen - wenn überhaupt - im Alpha-Register liegen, außer durch arbeitsaufwendige Stück-für-Stück-Vergleiche mit bekannten Zeichen. Deshalb ist das Alphabetisieren einer Gruppe von Alpha-Ketten ein sich verbietend langsamer, Speicherplatz fressender Vorgang. Synthetische Funktionen können jedoch ein weiteres Mal die Tauglichkeit des HP-41C vergrößern und ihn zur 'Textverarbeitung' befähigen, weil sie die Umwandlung von Zeichen in Zahlen und umgekehrt gestatten.

Nehmen Sie an, wir möchten ein einzelnes Alpha-Zeichen bestimmen oder ihm einen numerischen Wert zuordnen. Weil es 256 Zeichen gibt (natürlich werden nicht alle unterschiedlich angezeigt), ist es naheliegend, eine Zahl aus dem Bereich zwischen 0 und 255 zu ermitteln, nämlich die, welche dem Byte-Kode des Zeichens dezimal entspricht. Die schon bekannte hexadezimal→oktal→dezimal Umwandlung aus Abschnitt 6B kann, wie das folgende Programm "CD" (für Character→Decimal) zeigt, für diesen Zweck verwendet werden:

```
01♦LBL "CD"          10 SF 09
02 "┣♦♦♦♦X"          11 FS?C 11
03 X<> [             12 SF 10
04 X<> d             13 FS?C 12
05 FS?C 08           14 SF 11
06 SF 06             15 X<> d
07 FS?C 09           16 DEC
08 SF 07             17 END
09 FS?C 10
```

"CD"

43 BYTES

(Zeile 2, 'F6 7F 00 00 00 00 02', ist dieselbe wie Zeile 4 im Programm "EF" in Abschnitt 5E.) Beispiele: "A", XEQ "CD" ergibt '65'; "$", XEQ "CD" ergibt '36'.

Die Umwandlung in anderer Richtung, "DC" (Decimal→Character), ist nur wenig schwieriger. Die Zeilen 03-06 von "DC" stellen sicher, daß die drei Oktalziffern der Eingabezahl stets in dieselbe Gruppe von Flags des Registers d übertragen werden, auch wenn die Eingabe nur eine ein- oder zweiziffrige Dezimalzahl ist.

```
01♦LBL "DC"     08 FS?C 19     15 SF 17     22 STO \
02 OCT          09 SF 20       16 FS?C 14   23 "┣A"
03  E3          10 FS?C 18     17 SF 16     24 X<> \
04 /            11 SF 19       18 X<> d     25 CLA
05 10           12 FS?C 17     19 STO [     26 X<> [
06 +            13 SF 18       20 "┣♦♦"     27 AVIEW
07 X<> d        14 FS?C 15     21 CLX       28 END
```

"DC"

58 BYTES

Beispiele: '37', XEQ "DC" ergibt "%"; '64', XEQ "DC" ergibt "@".

Die Aufgabe, eine Gruppe von Alpha-Ketten zu sortieren, erfordert ein verwickelteres Verfahren der Zeichenerkennung als dasjenige, welches durch "CD" gegeben ist. Weil der einzige Alpha-Vergleich, den der HP-41C durchführen kann, ein 'X=Y?' ist, benötigen wir numerische Gegenwerte für ganze Alpha-Ketten, so daß wir Vergleiche der Form 'X<Y?', wie sie für eine Alphabetisierung gebraucht werden, anstellen können. Ist solch ein Vergleich erst einmal ausgeführt, können herkömmliche Sortierverfahren verwendet werden, um eine Reihe von Alpha-Ketten zu ordnen. Ein schlichter Weg zur Erzeugung solcher numerischen Gegenwerte wäre der, "CD" auf jedes Zeichen einer Kette anzuwenden und die Ergebnisse zu einer einzigen Zahl zusammenzustellen. Beachten Sie aber, daß der Dezimalwert des Zeichens "Z" die Zahl 90 ist, der größte Wert einer Kette von 6 Buchstaben also 90^6 = 5.3 E11 wäre, was die größte ganze Zahl, mit der der HP-41C umgehen kann, übersteigt. Daher müßte die Umwandlungsroutine noch 64 vom Dezimalwert abziehen ("A" = 1, "B" = 2 usw. herstellend), bevor sie 6 Werte zu einer einzigen Zahl zusammensetzt. Überdies müßte das Verfahren zu eindeutigen Zuordnungen führen.

Die synthetische Programmierung bietet eine Methode, numerische Gegenwerte von Alpha-Ketten zu erzeugen, die viel kürzer und schneller ist als die, Zeichen für Zeichen einzeln umzusetzen. Die nächste Routine, "AL", alphabetisiert ein einzelnes Paar von Alpha-Daten-Ketten. Sie kann mit Routinen der gewöhnlichen Zahlen-Sortierung, die der Anwender auswählt, vereint werden, um eine Reihe von Alpha-Daten zu ordnen.

```
01*LBL "AL"          12 RTN               23 RTN
02 XEQ 01            13 X<> IND T         24*LBL 01
03 XEQ 01            14 X<> IND Z         25 "   "(F2 01 01)
04 X#Y?              15 X<> IND T         26 ARCL IND Y
05 GTO 03            16 RTN               27 "+***"
06 RDN               17*LBL 02            28 ASTO [
07 RDN               18 "   "(F3 01 01 01) 29 "+**"
08 XEQ 02            19 ARCL IND Y        30 RCL [
09 XEQ 02            20 ASHF              31 END
10*LBL 03            21 "+**"
11 X>Y?              22 RCL [
```

"AL"

70 BYTES

SIZE 002

Gebrauchsanweisung für "AL":

1. Die beiden zu ordnenden Alpha-Ketten müssen in zwei numerierten Daten-Registern liegen. Die Ketten dürfen 1 bis 6 Zeichen enthalten.
2. Legen Sie die Nummer des einen Daten-Registers in X, die des anderen in Y.
3. XEQ "AL".
4. "AL" befördert die Kette, die in alphabetischer Ordnung die erste ist, in das Register, dessen Adresse ursprünglich in Y war; die andere Kette kommt in das Register, dessen Adresse in X lag.

"AL" benutzt zuerst das Unterprogramm 01 (Zeilen 24-31), um die ersten vier

Zeichen der beiden Ketten in für den Vergleich geeignete Zahlen umzuändern. Eine 'Zahl' wird durch das erste Nybble, '0' oder '9', gekennzeichnet; außerdem sind numerische Vergleiche von Alpha-Ketten nur dann sinnvoll, wenn die zugeordneten Zahlen die gleichen Exponenten haben. Diese beiden Überlegungen schränken den Alpha-Vergleich soweit ein, daß nur vier Zeichen auf einmal gegenübergestellt werden können, denn wir brauchen ein Byte als Zahlenkennung ("AL" verwendet dafür den Byte-Kode '01') zur Linken der Kette und zwei Bytes, hier '00 00', zur Rechten, um die Exponenten zu vereinheitlichen. Dies läßt in einem 7-Byte-Register nur vier Bytes 'frei'. Vier Zeichen reichen aber im allgemeinen aus, um zwei Ketten voneinander zu unterscheiden; wenn die zwei restlichen Bytes der Kette auch noch herangezogen werden müssen, führt Unterprogramm 02 (Zeilen 17-23) diesen zusätzlichen Vergleich durch.

Der Trick, eine Alpha-Kette in eine Zahl zu verwandeln, läßt sich auch umkehren. Die folgende Routine, "MANT" genannt, führt vor, wie sich eine Zahl in Alpha-Zeichen umformen läßt, damit sie durch Befehle wie 'APPEND' oder 'ASTO' verändert werden kann. Im vorliegenden Fall wollen wir eine Zahl durch ihre Mantisse ersetzen, indem wir ihren Exponenten abspalten. Wir könnten dies auch mit den Funktionen 'LOG' und '10^X' tun, doch führt das gelegentlich zu Fehlern in der letzten oder gar den beiden letzten Ziffern der Mantisse. "MANT" liefert stets ein richtiges Ergebnis. Nach Ausführung der Routine ist die Zahl aus dem X-Register (einschließlich ihres Vorzeichens) durch ihre Mantisse ersetzt; die Inhalte von Y und Z sind unversehrt, diejenigen von T, L und vom Alpha-Register sind verloren.

```
01*LBL "MANT"        09  E50
02 STO [             10 *
03 CLX               11 LASTX
04 FIX 4             12 X>Y?
05 ARCL X            13 1/X
06 X<> [             14 /
07 "↓↑"              15 FIX 9
08 X<> \             16 END
```

"MANT"

35 BYTES

6E. Synthetische Textzeilen und der Drucker

Synthetische Textzeilen, die mit einem der in den Kapiteln 3 und 5 beschriebenen Verfahren erzeugt werden, sind insbesondere für den Gebrauch des Druckers nützlich. Jedes der 128 Standard-Druckzeichen kann in eine Programm-Textzeile gebracht werden, indem man das entsprechende Byte (welches man in der Byte-Tabelle findet) in die Zeile setzt. Dieses Vorgehen erübrigt die andernfalls wiederholt aufzurufende Drucker-Funktion 'ACCHR'. Versuchen Sie z.B. eine Routine zu schreiben, wel-

che die Zeichen "Big Deal #7" druckt. Mit Flag 13 und 'ACCHR' brauchen Sie insgesamt 40 Bytes. Doch das erstrebte Ziel kann mit nur 14 Bytes erreicht werden, wenn man eine synthetische Textzeile, die die kleinen Buchstaben und das Symbol "#" gleich enthält, schreibt:

```
01 "B** Dea* #7
02 PRA
```

Der Kode der Zeile 01 lautet 'FB 42 69 67 20 44 65 61 6C 20 23 37'. Er kann leicht mit dem Byte-Hüpfer oder dem Text-Gehilfen erzeugt werden:

```
01 +
02 *          ("B")
03 FRC        ("i")
04 X=0?       ("g")
05 RCL 00     (" ")
06 X<Y?       ("D")
07 LN1+X      ("e")
08 ABS        ("a")
09 HMS        ("l")
10 RCL 00     (" ")
11 RCL 03     ("#")
12 STO 07     ("7")
13 PRA
```

```
TG .001                [01"B** Dea* #7" ]
5 mal Korrekturtaste drücken.
```

Auf ähnliche Weise können wir die Drucker-Funktion 'BLDSPEC' ersetzen. Betrachten Sie das Sonderzeichen in Abbildung 6-1, in der das Punktmuster und die zugehörigen 'Werte' und 'Spalten-Druck-Zahlen', wie sie von den 'BLDSPEC'-Befehlen verarbeitet werden (vgl. Bedienungshandbuch des Druckers 82143A, S. 66-68), zu sehen sind.

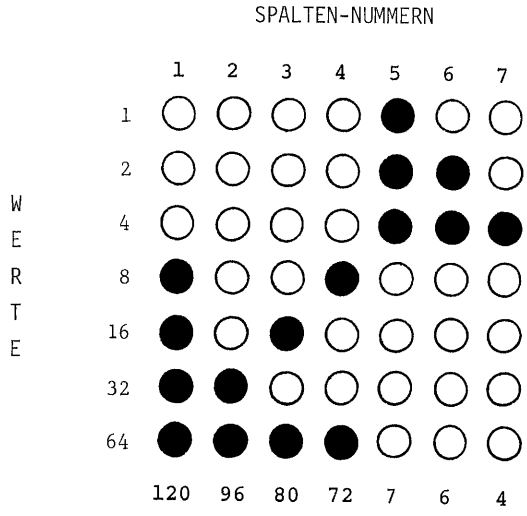


ABBILDUNG 6-1. EIN SONDERZEICHEN

Die Spalten-Druck-Zahlen sind genau die Dezimalwerte der Zahlen, die sich ergeben, wenn man jede Spalte als binäre 7-Bit-Zahl auffaßt, in der die Punkte Einsen und die Kreise Nullen entsprechen. Wenn 'BLDSPEC' mit Null in Register Y und der ersten Spalten-Druck-Zahl (120) im Register X das erste Mal ausgeführt wird, entsteht eine Alpha-Daten-Kette in X, die die sieben Bits der ersten Spalten-Druck-Zahl (1111000) als letzte sieben Bits der entstehenden Kette enthält:

X = 0001000000000000000000000000000000000000000000000|1111000

Sobald 'BLDSPEC' die nächste Spalten-Druck-Zahl (96) verarbeitet, werden deren sieben Bits in die letzten sieben Bitstellen von X gebracht, wobei der vorangegangene Eintrag nach links 'gedrängt' wird:

X = 0001000000000000000000000000000000000000000000|1111000|1100000

Nach Eingabe der letzten Spalten-Druck-Zahl enthält das X-Register:

	120	96	80	72	7	6	4
X =	0001 000|1 1110 00|11 0000 0|101 0000 |1001 000|0 0001 11|00 0011 0|000 0100						
(hexadez.)	1 1	E	3	0	5	0	9	0	1	C	3	0	4

Oberhalb der Binär-Darstellung von X sind die Spalten-Druck-Zahlen aufgeführt, unterhalb sehen Sie die dem Bit-Muster entsprechenden Byte-Kodes. Der gesamte Byte-Kode kann als 7-Zeichen-Textzeile 'F7 11 E3 05 09 01 C3 04' ins Programm, welches ihn ins Alpha-Register setzt, gebracht werden, von wo aus er mit einem 'RCL M' ins X-Register gelangt. Anschließend kann ihn die Drucker-Funktion 'ACSPEC' wie gewohnt verarbeiten.

Tatsächlich brauchen Sie die Spalten-Druck-Zahlen nicht einmal auszurechnen. Sie müssen lediglich eine binäre 49-Bit-Zahl, die das 7×7 Punkt-Gitter darstellt, aufschreiben, Einsen für Punkte, Nullen für Kreise. Beginnen Sie mit der linken unteren Ecke des Gitters als dem äußersten linken Bit, und laufen Sie in der ersten Spalte nach oben bis zur ersten Zeile durch, dann in der zweiten Spalte ebenfalls von unten nach oben usw. Sind alle Punkte verschlüsselt, stellen Sie die sieben Bits '0001000' davor, so daß Sie insgesamt 56 Bits haben. Gruppieren Sie diese in Ziffern zu je 4 Bits. Bilden Sie zum Schluß daraus eine 7-Zeichen-Textzeile unter Verwendung von Hexadezimalwerten. Weil die entstehenden Byte-Kodes aus beliebigen Stellen der Byte-Tabelle kommen können (also auch aus der unteren Hälfte), ist es am besten, die Textzeile mit Hilfe von "CODE" und dem Q-Boten zu erzeugen.

Diese Methode scheint vielleicht ein wenig umständlich, doch ist sie in Wirklichkeit kaum langatmiger als das ordnungsgemäße Verfahren mit 'BLDSPEC'. Die Ersparnis an Programmspeicherplatz ist jedenfalls, wie die folgende Gegenüberstellung zeigt, erheblich:

'normales' Programm:

```
01 0          10 BLDSPEC
02 ENTER      11 7
03 120        12 BLDSPEC
04 BLDSPEC    13 6
05 96         14 BLDSPEC
06 BLDSPEC    15 4
07 80         16 BLDSPEC
08 BLDSPEC    17 ACSPEC
09 72
```

30 Bytes

synthetisches Programm:

```
01 "▪▪▪▪▪▪▪"
02 RCL M
03 ACSPEC
```

12 Bytes

Natürlich könnten Sie die Folge der 'BLDSPEC'-Befehle auch von Hand ausführen und das entstehende Sonderzeichen dem Programm in einem Daten-Register zur Verfügung stellen, so daß Sie insgesamt gar nur 10 Bytes Speicherplatz (die Bytes des Daten-Registers mitgezählt) benötigen. Doch wenn das Programm über Magnetkarten in den Rechner gelangt, muß auch eine Daten-Karte eingelesen werden; darüberhinaus muß das Daten-Register solange, wie Sie das mit dem Sonderzeichen arbeitende Programm benutzen, gegen zerstörenden Zugriff durch andere Programme geschützt werden.

Falls Sie sich mit diesem Abschnitt herumplagen, werden Sie wahrscheinlich einen Drucker benutzen, um die Programmzeilen aufzulisten. Dabei erscheint die Zeile 01 des zuletzt beschriebenen Programms in der Form:

```
01 "ΩßσπΣα"
```

Es werden nur 5 Zeichen gezeigt, weil der Programm-Ausdruck einer Textzeile nur Zeichen aus der oberen Hälfte der Byte-Tabelle wiedergibt. Zeichen, die Bytes aus der unteren Tabellenhälfte entsprechen, sind unsichtbar. Überdies verwendet der Druckpuffer Bytes aus den Zeilen A, B, D und E für eigene Zwecke, die mit dem Druck von Sonderzeichen, der Zeichenbreite usw. zusammenhängen. Darum können Textzeilen, die Zeichen aus diesen vier Zeilen enthalten, ein sehr merkwürdiges Druckbild aufweisen. Enthielte beispielsweise eine Textzeile ein dem Byte-Kode 'D5' entsprechendes Zeichen, würde eine diese Textzeile umfassende Programm-Auflistung ein Druckbild hervorrufen, in welchem die dem Byte 'D5' folgenden Zeichen in doppelter Breite und in Kleinschreibung auftauchten.

6F. Nicht-normalisierte Zahlen und Kontrolle über ganze Gruppen von Flags

Die Verwendung synthetischer Textzeilen ist keineswegs auf die programmierte Erzeugung von Sonderzeichen im Alpha-Register beschränkt. Eine synthetische Textzeile von 7 Zeichen, der ein 'RCL M' folgt, setzt eine NNZ ins X-Register. Eine bemerkenswerte Verwendung so gewonnener NNZ's ist die der 'Gruppen-Kontrolle von Flags', die durch die Weiterbeförderung der NNZ ins Register d ermöglicht wird. Wir haben bereits beim Umgang mit dem Textgehilfen eine Anwendung solcher Gruppen-Kontrolle kennengelernt.

Die in den vorangegangenen Abschnitten beschriebenen Programme enthalten zahlreiche Beispiele für die Benutzung des Befehls 'X<>d', um den Anfangszustand des Flag-Registers wiederherzustellen, sobald es seine Aufgabe als 'binärer Verschlüsseler' erfüllt hat. Die Fähigkeit, willkürliche NNZ's zu erzeugen, gestattet es uns, alle 56 Flags auf einen Schlag zu setzen oder zu löschen. Die zugrundeliegende Befehlsfolge ist diese:

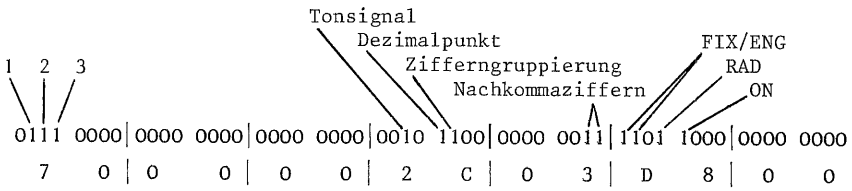
```
01 "xxxxxxx"
02 RCL M
03 STO d
```

(6F-1)

in der "xxxxxxx" die zur Erzeugung der NNZ benötigte synthetische Textzeile bedeutet. Die Routine 6F-1 verbraucht 12 Programm-Bytes; ebensoviele wären für nur 6 Programmzeilen der Form 'SF mn' oder 'CF mn' erforderlich. Im allgemeinen ist es daher wirkungsvoller, Routine 6F-1 statt einzelner 'SF'- und 'CF'-Zeilen einzusetzen, wenn mehr als 6 Flags gesetzt oder gelöscht werden sollen. Solche Gelegenheiten gibt es in Einleitungsroutinen, in denen bestimmte Flag-Zustände, erwünschte Anzeige-Formate oder trigonometrische Modi eingestellt werden müssen, häufig.

Zur Veranschaulichung wollen wir eine Routine schreiben, die die HP41-C-Flags wie folgt setzt: Flags 1, 2, 3, 26 (Tonsignal), 28 (Dezimalpunkt), 29 (Zifferngruppierung) an; Anzeige-Format 'FIX/ENG 3' (Flags 38, 39, 40 und 41 an); 'RAD'-Modus (Flag 43 an); Dauereinschaltung (Flag 44 an); die restlichen Flags gelöscht. Die 'FIX/ENG'-Anzeige haben wir insbesondere deswegen gewählt, weil sie ein Format bildet, welches ohne synthetische Programmierung nicht verfügbar ist. Im gewöhnlichen 'FIX'-Format (Flag 40 gesetzt, Flag 41 gelöscht) veranlassen Zahlen, welche für eine diesem Format angepaßte Darstellung zu groß oder zu klein sind, die Anzeige, ins 'SCI'-Format auszuweichen. Im 'FIX/ENG'-Format weicht sie dagegen ins 'ENG'-Format aus.

Um die den gewünschten Flagzustand herstellende synthetische Textzeile festzulegen, schreiben wir die Standorte aller 56 Flags als binäre 56-Bit-Zahl auf, mit Einsen für gesetzte und Nullen für gelöschte Flags, und gruppieren dann die Bits in hexadezimale Bytes zu je 8 Bits:



Wir sehen, daß die benötigte Textzeile 'F7 70 00 00 2C 03 D8 00' lautet. Dieser Byte-Kode ist eine Herausforderung an jedes der bislang untersuchten Verfahren zur Erzeugung synthetischer Textzeilen. Weil die Zeile nämlich 8 Bytes lang ist, kann sie z.B. nicht mit einem einzigen Aufruf von "REG" gebildet werden. Wir können jedoch eine 'Strohmann'-Textzeile von 7 Zeichen ins Programm bringen und sie im Speicher hin- und herjagen, indem wir über ihr im Speicher Bytes einfügen oder wegnehmen, bis alle ihre sieben Zeichen im selben Register liegen (bis also auf die Programm-Anzeige der Strohmann-Zeile hin ein 'RCL b' und ein XEQ "AD" eine Adresse liefern, die mit der Byte-Nummer '1' beginnt). Anschließend können wir dann "REG" dazu verwenden, den 'Stummel'-Kode '70 00 00 2C 03 D8 00' in dem Register, welches die 7 Strohmann-Zeichen enthält, abzulegen, während das im voranstehenden Register auf Byte-Nummer '0' stehende Textzeilen-Byte 'F7' unversehrt bleibt.

Weil das Byte 'D8' nicht als Einzelzeile eingetastet werden kann, muß man schon sehr listig mit dem Byte-Hüpfer oder dem Textgehilfen umgehen, wenn man die Textzeile mit ihnen edieren will. Wegen des Bytes '00' am Zeilenende verweigert auch der Q-Bote die Gefolgschaft. Würden wir ihn auf die Zeile ansetzen, erhielten wir eine Zeile von nur *sechs* Zeichen, denn der Q-Bote 'vergißt' führende Nullen im Eingabe-Kode. Doch eine vereinte Anstrengung von Byte-Hüpfer und Q-Boten bekommt das Kunststück in den Griff. Wir lassen zuerst den Q-Boten den Kode '01 D8 03 2C 00 00 70', in welchem wir das Byte '00' durch ein '01' ersetzt haben, befördern. Dann springen wir mit dem Byte-Hüpfer zu diesem '01' und löschen es:

```
                          XEQ "CODE"          ["CODE=?"      ]
"01 D8 03 2C 00 00 70"    R/S                 ["▫▪▪,¯¯▪"     ]
                          GTO neues Programm
01 STO 07                 PRGM aus
                          STO Q
                          PRGM an
                          Q-Bote              [02 9          ]
                          DEL 001             [01 STO 07     ]
                          PACK, SST           [02 "▪¯¯,▪▪▫"  ]
                          JUMP .002           [02 LBL 00     ]
                          DEL 001             [01 STO 07     ]
                          DEL 001             [00 REG abc    ]
```

Wir beenden das Ganze mit den Zeilen '02 RCL M' und '03 STO d', um die Routine 6F-1 abzurunden.

Der einzige Weg, auf dem wir viele der 'System'-Flags (das sind die Flags 30-35, 45-47 und 49-55) setzen können, ist der, NNZ's ins Register d zu bringen. Obwohl die Beherrschung dieser Flags gelegentlich zu recht amüsanten aber nicht besonders nützlichen Wirkungen führt (siehe Abschnitt 7B), gibt es doch ein Beispiel praktischer Verwendbarkeit, nämlich das Setzen des Daten-Eingabe-Flags 45 (was uns schon bei der Einführung des Textgehilfen in Form der ZTG begegnet ist).

Häufig, insbesondere wenn Berechnungen durchgeführt werden, die statistische Summationen enthalten, werden wir aufgefordert, Zahlen einzugeben, die sich nur in den letzten Ziffern unterscheiden, wie '123456', '123457', '123460' usw. Um das stets erneute Eintasten der ersten vier Ziffern '1234' zu vermeiden, könnten wir ein paar Programmschritte hinzufügen, die '123400' auf unsere Einträge addiert, so daß wir nur mehr die letzten beiden Ziffern jeder Zahl einzutasten brauchen. Doch wir können den Vorgang noch 'gefälliger' gestalten, indem wir den HP-41C bitten, die Zahlen '1234' selbst einzugeben *und anzuzeigen*, und zwar in der Weise, daß wir während unserer Eingabe der letzten beiden Ziffern die vollständige Zahl erblicken. Flag 45 erlaubt ein solches Vorgehen:

```
01 "*¯"     (F2 84 00)
02 RCL M
03 X<>d
04 1234
05 STOP
06 X<>Y
07 STO d
```

(6F-2)

Das erste Zeichen von Zeile 01, Byte '84', setzt die Flags 40 und 45, wenn es ins Register d gelangt. Sobald das Programm auf Zeile 05 mit gesetztem Flag 45 anhält, meint der Prozessor, der Rechner sei noch im Zustand der Dateneingabe. Beim Halt erscheint in der Anzeige '1234' (der Inhalt von Register X). Wenn wir eine Zahl eintasten, etwa '5', sehen wir das Anzeigebild '12345_', in welchem die Unterstreichung andeutet, daß die Eingabe weiterer Ziffern möglich ist. Die Zeilen 06 und 07 können nach Bedarf verwendet werden - sie dienen dazu, den anfänglichen Zustand der Flags wiederherzustellen, und lassen die zuletzt (vom HP-41C und vom Benutzer gemeinsam) eingegebene Zahl im Y-Register zurück.

Desselben Kunstgriffes bedienen wir uns in der Routine 6F-3 für die Eingabe von Alpha-Daten, so daß wir also auch Alpha-Zeichen einer bereits vorhandenen Kette anhängen können, während sie ganz angezeigt wird. Oder, um noch einen Schritt weiterzugehen: wir hängen Alpha-Zeichen einer Meldung oder Aufforderung der Anzeige an, doch nur die neu eingetasteten Zeichen bleiben zur Weiterverarbeitung im Alpha-Register zurück. Um sich das vor Augen zu führen, ersetzen Sie die ersten 5 Zeilen von "CODE" durch diese Folge:

```
01 LBL "CODE"
02 "Σ*"          (F2 04 80)
03 RCL M
04 X<>d
05 "CODE="
06 AVIEW
07 CLA
08 STOP
09 X<>d
```

(6F-3)

Die Zeilen 02-04 setzen die Flags 45 und 48 (ALPHA an). Die Zeilen 05-07 sorgen dafür, daß die Anzeige beim Programm-Halt auf Zeile 08 den Text "CODE=" aufweist, obwohl das Alpha-Register inzwischen gelöscht wurde. Wenn wir die Alpha-Zeichen des zu verschlüsselnden Kodes eintasten, gelangen sie auf gewohnte Weise ins Alpha-Register, doch in der Anzeige werden sie gleichzeitig dem "CODE=" angehängt. Wenn wir die Anzeige, während das Programm hält, durch Alpha aus/an löschen, verschwindet das Trugbild "CODE=", und nur die eingetasteten Zeichen bleiben in der Anzeige zurück. Das Ganze ist zwar keine tiefreichende Leistung, nichtsdestoweniger läßt es den HP-41C abermals ein wenig 'freundlicher' werden. Leider scheint es (jedenfalls bis jetzt) keine Möglichkeit zu geben, einer Alpha-Aufforderung zur Daten-Eingabe numerische Einträge anzuhängen.

Eine Mahnung zur Vorsicht ist in Bezug auf den Gebrauch von NNZ's im HP-41C auszusprechen. Die arithmetischen Routinen im Rechner sind dazu ausgelegt, nur mit normalen Dezimalzahlen umzugehen. Werden sie zur Verarbeitung von NNZ's herangezogen, kann das überraschende, dann und wann auch sehr ungelegene Wirkungen haben. Erzeugen Sie z.B. mit "CODE" die NNZ '00 00 01 00 00 00 00', die im 'SCI 5'-Format mit '0,00010 E0' angezeigt wird. Rufen Sie dann '1/X' auf und beobachten Sie, was sich ereignet. Die Anzeige erlischt für etwa 5 Sekunden, während derer das Tastenfeld blockiert ist, der HP-41C also auf keinen Tastendruck, 'ON' eingeschlossen, mehr antwortet. Die NNZ im Nenner verursacht die Verzögerung: der Divisionsvorgang, der bei 1/X stattfindet, setzt voraus, daß sowohl Zähler wie Nenner im richtigen 'SCI'-Format vorliegen (hier wäre das '01 00 00 00 00 09 96'). Die Division wird als eine Reihe von Subtraktionen ausgeführt - die beiden Exponenten werden voneinander subtrahiert, und dann wird der Nenner wiederholt vom Zähler abgezogen - in der Tat nichts anderes als die Umkehrung einer durch wiederholte Addition durchgeführten Multiplikation. Der Vorgang dauert nicht lange, wenn die Mantissen von Zähler und Nenner, wie sie sein sollten, beide von der Größenordnung Eins sind; doch in unserem Beispiel ist die Mantisse des Nenners nur 0,0001, so daß 10^4 Subtraktionen bis zum Abschluß der Division notwendig sind. 5 Sekunden sind nicht übermäßig lang, doch andere NNZ's könnten leicht 5.000 Sekunden oder mehr für eine Division verbrauchen. Andere Funktionen dauern sogar noch länger: 'LOG (0,0001 E0)' etwa braucht 45 Sekunden, also 9 mal soviel wie die auf den gleichen Wert angesetzte Funktion '1/X'. Die Normalisierung von Register-In-

halten, die stattfindet, wenn 'RCL'-Funktionen aufgerufen werden, ist insbesondere dazu vorgesehen, die Gefahr von Rechner-'Blockaden' auszuschließen, die von NNZ's verursacht werden könnten, welche durch das Einstecken von Speichererweiterungsmodulen in Daten-Register zu gelangen vermögen. Seien Sie also vorsichtig. Dennoch: wie bei den meisten anderen Bruchlandungen wird das Entfernen und Rücksetzen der Batterie Ihr Juwel wieder beleben.

6G. Wir versetzen die Spanische Wand

Wegen des sparsamen Verbrauchs von Programm-Bytes bei der Verwendung der Ein-Byte-Funktionen 'STO' und 'RCL' empfiehlt es sich, Programme, wenn irgend möglich, auf die Daten-Register R_{00}-R_{15} zugreifen zu lassen. Daher ist es die Regel, mehrere Programme im Speicher zu haben, deren jedes den gleichen Block von Daten-Registern benutzt, so daß die Ausführung eines Programms im allgemeinen die von anderen Programmen verwendeten oder abgelegten Daten zerstört. Will man das verhindern, kann man eine Daten-Übertragungsroutine schreiben, die die Daten von einem Block in einen anderen überführt und so den ersten Block für die Verwendung durch ein anderes Programm frei macht. Wenn die Anzahl der in einem solchen Block enthaltenen Register jedoch groß ist, läuft die Routine ziemlich langsam ab. Das Programm "CU" (für 'Curtain') bietet eine ganz andere, schnellere Lösung an.

```
01♦LBL "CU"      13 CLX          25♦LBL 12       37 DSE [
02 STO L         14 LASTX        26 FC?C IND Y   38 GTO 11
03 CLX           15 INT          27 SF IND Y     39♦LBL 14
04 RCL c         16 X=0?         28 FC? IND Y    40 X<> ]
05 STO [         17 GTO 14       29 CHS          41 X<> d
06 "┣♦♦♦♦"       18 2            30 X>0?         42 STO [
07 11            19 /            31 GTO 13       43 "┗ABC"
08 X<> [         20 RCL [        32 FC? IND Y    44 X<> \
09 X<> d         21 X<>Y         33 CHS          45 STO c
10 STO ]         22 FRC          34 DSE Y        46 RDN
11♦LBL 11        23 X=0?         35 GTO 12       47 END
12 RDN           24 GTO 13       36♦LBL 13
```

"CU"

87 BYTES

Gebrauchsanweisung für "CU":

1. Geben Sie eine ganze Zahl 'n' ins X-Register.
2. XEQ "CU".
3. Wenn $n > 0$ ist, wird R_n das neue R_{00}.
   Wenn $n < 0$ ist, wird R_{-n} das neue R_{00}.
   Alle anderen Daten-Register-Nummern werden sinngemäß verschoben.

"CU" nimmt die Zahl 'n' aus dem X-Register (die von Hand oder durch ein anderes Programm dorthin gebracht wurde) und addiert sie auf die im Register c liegende Adresse von R_{00}. Wenn 'n' positiv ist, werden die Daten-Register R_{00} bis R_{n-1} in Programm-Register 'umgewidmet', weil die geheimnisvolle 'Spanische

Wand', die im Speicher die Programme von den Daten trennt, aus ihrer Anfangslage unmittelbar unterhalb von R_{00} an eine neue Stelle unmittelbar unterhalb des Registers R_n versetzt wird; R_n wird zum 'neuen' R_{00}. Wenn 'n' negativ ist, wandert die Spanische Wand im Speicher nach unten, so daß 'n' bisherige Programm-Register zu Daten-Registern werden. All dies geschieht ohne Veränderung oder Bewegung der Inhalte der in den Vorgang einbezogenen Register.

Nehmen Sie an, es werde ein 'Programm 1', welches in den Registern R_{00}-R_{50} Daten für spätere Verarbeitung zurückläßt, ausgeführt. Doch zwischenzeitlich möchten wir ein 'Programm 2' laufen lassen, das sich der Register R_{00}-R_{25} für eigene Zwecke bedienen will. In diesem Fall rufen wir "CU" mit '51' im Register X auf (die Gesamtanzahl der verfügbaren Daten-Register sollte in diesem Fall mindestens 77 betragen). Nach der Ausführung von Programm 2 können wir den Rechner auf einen zweiten Lauf von Programm 1, bei dem dieses alle seine Daten wiederfindet, vorbereiten, indem wir jetzt "CU" mit '-51' im Register X aufrufen.

** WARNUNG: Der Versuch, die Spanische Wand über den Kopf des Speichers hinaus zu verschieben, indem "CU" für ein 'n', welches größer als die Gesamtanzahl der gerade verfügbaren Daten-Register ist, ausgeführt wird, oder die Kühnheit, sie auf Adressen vom Hexadezimalwert '010' bis '0C0' oder '000' zu setzen, führt unweigerlich zu einem "MEMORY LOST".

"CU" bewerkstelligt eine binäre Addition der Zahl 'n' auf die hexadezimalen Ziffern 9-11 des Registers c, welche dort die 'Adresse der Wand' bilden. Die entsprechenden Flags 32-43 des Registers d können nicht alle einzeln betätigt werden, so daß der Inhalt von c erst nach Register M übertragen und dort durch Anhängen von Nullen (Zeile 06) nach links geschoben werden muß. Dann setzen die Zeilen 08-09 diese Adresse ins Register d auf die Flags 00-11.

Die binäre Addition ist ein sehr einfacher Vorgang. Um 1 auf eine binäre Zahl zu addieren, brauchen wir nur den Wert des letzten Bits von 1 in 0 oder umgekehrt zu ändern. Ist das letzte Bit dabei eine 1 geworden, sind wir fertig. Ist es eine 0 geworden, gehen wir zum nächsten Bit zur Linken über. Wird dieses nächste Bit eine 1, sind wir jetzt fertig; wird es eine 0, gehen wir abermals nach links weiter zum nächsten Bit usf., bis wir schließlich zu einem Bit gelangen, das sich von 0 in 1 ändert. Die Subtraktion verläuft fast genauso - wir beginnen das Verfahren wieder beim äußersten rechten Bit, arbeiten uns nach links weiter und hören diesmal auf, sobald sich ein Bit von 1 in 0 ändert. Die Addition von 2 (binär 10) erfolgt in derselben Weise, nur daß wir dafür mit dem vorletzten Bit anfangen. Und allgemein beginnen wir mit dem von rechts gezählten (m+1)-ten Bit, wenn wir 2^m addieren wollen.

Die binäre Addition wird in "CU" in den Zeilen 11-35 durchgeführt. Die eingetastete Zahl 'n' wird durch wiederholte Division durch 2 (Zeilen 18-19) in binäre Bits zerlegt. Die dabei nach und nach entstehenden Bits werden entsprechend

der Prüfung in Zeile 30 der zu verändernden Adresse zugefügt oder von ihr abgezogen. Ist die Addition abgeschlossen, wird die nunmehr in den ersten 3 Bytes des Registers d stehende veränderte Adresse mit den ursprünglichen ersten 4 Bytes aus Register d, die im Register N warten, vereinigt (Zeilen 41-42). Der gesamte 7-Byte-Kode wird ins Register N gestoßen (Zeile 43) und endlich, in Zeile 45, nach Register c zurückgebracht. Nach Ausführung von "CU" sind die Inhalte der Stapelregister X und Y, die dort vor der Eingabe von 'n' lagen, wieder an Ort und Stelle.

Der HP-41C arbeitet auch dann in gewohnter Weise, wenn die durch ein 'SIZE' festgelegte Stellung der Spanischen Wand mit "CU" nach oben oder unten verschoben wurde. Allerdings sollte der Speicher bei nach oben versetzter Wand, welche Daten- zu Programm-Registern macht, nicht mit 'PACK' verdichtet werden, weil dies höchstwahrscheinlich die unterhalb der Wand abgelegten Daten unwiderruflich durch Beseitigung aller Null-Bytes zerstört. Diese Gefahr kann durch ein vorher an den Kopf des Programmspeichers gestelltes 'END', gefolgt von einem 'PACK', umgangen werden. Wenn die Wand anschließend versetzt wird, sind die entstehenden Programm-Register gegen ein 'PACK' geschützt. Das 'END', welches ein verdichtetes Programm vorspiegelt, bewahrt sie vor einem solchen Zugriff.

Eine zweite wichtige Anwendung von "CU" ist die, Daten auf Dauer in Programm-Bytes zu verwandeln, wodurch uns ein weiteres Mittel für die Erzeugung synthetischer Programmzeilen zur Verfügung steht. Dieses Verfahren ist äußerst nützlich, sobald mehrere aufeinanderfolgende Programm-Register oder gar ein ganzes Programm hinreichend viele synthetische Zeilen enthalten, um das Erstellen des gesamten Programmes mit "CODE" zu rechtfertigen. In diesem Fall verwenden wir "CODE", um die Byte-Kodes von je 7 Programm-Bytes zu erzeugen und sie der Reihenfolge nach in benachbarte Daten-Register zu speichern. Die letzten 7 Bytes gehören nach R_{00}, die vorletzten nach R_{01} usw. (Dies ist, beiläufig bemerkt, der Hauptgrund dafür gewesen, "CODE" so einzurichten, daß es keine numerierten Daten-Register verwendet.) Sobald die Verschlüsselung beendet ist, nehmen wir "CU", um die Wand oberhalb des höchsten Daten-Registers, welches die erzeugten Programm-Bytes enthält, aufzustellen. Die synthetischen Kodes erscheinen dann als Programmzeilen, beginnend am Kopf des Programmspeichers. Um die neuen Zeilen zu erreichen, rufen wir 'CAT 1' auf, unterbrechen die Funktion bei der ersten globalen Marke oder auf dem 'END' und lassen ein von Hand ausgeführtes 'RTN' folgen. Hier ein Beispiel:

```
                        XEQ "CODE"      ["CODE=?"       ]
"C000F400600401"
                        R/S             ["✱⁻✱⁻TIX "     ]
                        STO 01
                        XEQ "CODE"      ["CODE=?"       ]
"F32801297E8685"
                        R/S             ["✱(X)Σ✱✱"      ]
                        STO 00
2
```

```
XEQ "CU"
CAT 1 - R/S bei der ersten Marke oder dem  END
RTN
SST, um das neue Programm zu erblicken
```

```
01 LBL "┬X┼ "
02 "(┼)"
03 AVIEW
04 BEEP
05 RTN
```

An dieser Stelle wird sich die Marke "┬X┼" nicht im Katalog zeigen, weil sie noch kein Teilstück der globalen Kette geworden ist. Sie kann aber in die Kette eingesetzt werden, indem eine Programmzeile vorübergehend irgendwo zwischen den neuen Zeilen eingefügt und gleich wieder gelöscht wird, gefolgt von einem 'PACK'.

6H. Anwender-Module: Wir schleichen uns durch die Hintertür

Die 'ROM (Read-Only Memory)'-Anwender-Module bilden eine wesentliche Ausweitung des Speicherumfangs des HP-41C und schließen eine ausgedehnte Bibliothek vorprogrammierter Routinen in sich ein. Bedauerlicherweise leiden viele dieser Routinen an dem Mangel, wegen der in ihnen enthaltenen verschiedenen Unterbrechungen für Ein- und Ausgabe von Daten von Benutzerprogrammen nicht als Unterprogramme aufgerufen werden zu können. In vielen Fällen kann diese Einschränkung durch den Einsatz der folgenden Routine, die eine direkte Verzweigung zu jeder Stelle eines jeden ROM-Programmes erlaubt, überwunden werden:

```
01*LBL "ROM"          11 X<> [              21 X<> ]
02 SF 01              12 X<> \              22 X<> a
03 X<> a              13 X<> [              23 X<> \
04 X<> \              14 ARCL 00            24 STO b
05 CLX                15 "├-----"           25*LBL 00
06 RCL b              16 X<> ↑              26 X<> \
07 FC?C 01            17 X<> ]              27 CLA
08 GTO 00             18 "├++"              28 END
09 STO [              19 STO [
10 "├↑↑↑↑↑"           20 "├**"
```

"ROM"

73 BYTES

Gebrauchsanweisung für "ROM":

Vor dem Aufruf von "ROM" muß die absolute Adresse der Stelle des ROM-Programmes, an der mit seiner Ausführung begonnen werden soll, in R_{00} abgelegt werden. Dann muß Ihr Programm "ROM" als Unterprogramm aufrufen, nicht etwa unmittelbar das ROM-Programm im Modul. Der RPN-Stapel und die Daten-Register müssen so gefüllt sein, wie es das ROM-Programm am Einstiegspunkt erwartet. "ROM" überträgt die Ausführung an die festgelegte Stelle im Modul, von der aus das ROM-Programm ganz

normal arbeitet und dann die Ausführung an die Absprungstelle des Hauptprogrammes zurückgibt, wenn es dem rechtskräftigen 'RTN' oder 'END' begegnet. Obzwar ein ROM-Programm, welches unter seiner globalen Marke aufgerufen wird, gewöhnlich ein Unterprogramm sechster Stufe sein kann (während seiner Ausführung also bis zu sechs wartende Adressen im Rücksprungstapel zu liegen vermögen), darf "ROM" als Unterprogramm auf höchstens fünfter Stufe aufgerufen werden.

Der Verlust einer Unterprogramm-Ebene entspringt der Wirkungsweise von "ROM". Nach der Ausführung von Zeile 09 enthält das Alpha-Register ein Abbild des in den Registern a und b liegenden Rücksprungstapels:

R6 R5 R4 R3 R2 R1 A6

in welchem 'A6' die absolute Adresse des zweiten Bytes von Zeile 06 ist (wo das 'RCL b' erfolgte); 'R1' ist die Rücksprungadresse der Programmzeile, von der aus "ROM" als Unterprogramm aufgerufen wurde; 'R2' ist die zuvor aufgezeichnete Rücksprungadresse usw. Die Zeilen 10-20 schaufeln die Zeichen im Alpha-Register umher, bis die Register 0 und N schließlich einen neuen Rückkehrstapel der Form:

R5 R4 R3 R2 R1 ER A6

enthalten, in welchem 'ER' die aus R_{00} herbeigeholte Adresse der Einstiegsstelle in den Anwender-Modul darstellt. Beachten Sie, daß 'R6' verloren gegangen ist, woher sich die Einbuße einer Unterprogramm-Ebene erklärt, wenn man "ROM" aufruft. Der neue Rückkehrstapel wird durch die Zeilen 21-24 in die Register a und b gebracht. Bei Ausführung von Zeile '24 STO b' wird 'A6' zum Adreßzeiger, so daß die Ausführung mit Zeile 07 fortgesetzt wird. Diesmal ist Flag 01 gelöscht, woraufhin das Programm zur Zeile '25 LBL 00' springt. Die Zeilen 26-27 vervollständigen die 'Aufräumarbeiten im Hause', indem sie den RPN-Stapel in den Zustand, den er zum Zeitpunkt des Aufrufs von "ROM" hatte, zurücksetzen. Das 'END' von "ROM" 'senkt' den Rücksprungstapel, so daß 'ER', die Adresse aus dem Modul, zum Adreßzeiger wird, was somit zur erwünschten Übergabe der Ausführung an die ausgewählte Stelle des ROM-Programms führt. Wird dort ein 'END' oder 'RTN' angetroffen, kehrt die Ausführung zum Benutzerprogramm auf die rufende Adresse 'R1' zurück. Berücksichtigen Sie, daß das ROM-Programm selbst Unterprogramme enthalten kann, deren Aufruf zum Verlust der Rücksprungadressen 'R5', 'R4' usw. führt.

Das Verfahren zur bestimmung der richtigen Adresse zum Einstieg in das ROM-Programm ist denkbar einfach. Führen Sie zunächst ein 'GTO' auf irgendeine globale Marke innerhalb des betreffenden ROM-Programmes aus. Tasten Sie dann 'GTO .ℓmn', worin 'ℓmn' die Zeilen-Nummer der Zeile ist, mit der Sie die Programmausführung im Modul aufnehmen möchten. Dann drücken Sie 'RCL b', 'CLA', 'STO M', 'ASTO 00', was die entscheidende Adresse in der für die Verarbeitung

durch "ROM" geeigneten Form nach R_{00} bringt. Die Wahl von R_{00} ist willkürlich; wird dieses Register für andere Zwecke gebraucht, kann ebensogut jedes andere Register an seine Stelle treten, indem die Zeile 14 entsprechend geändert wird.

Als Beispiel für die Verwendung von "ROM" wollen wir den Programmteil "SSS" aus dem Anwender-Modul MATH 1 (HP 00041-15003) benutzen. Dieser Programmteil fordert zur Eingabe der Längen der drei Seiten eines Dreiecks von Hand auf und gibt dann die Seiten, die Winkel und den Flächeninhalt aus. Wenn der Drucker nicht angeschlossen ist, erfordert die Ausgabe die wiederholte Bedienung der Taste 'R/S', um dem Rechner jeden der sieben Ausgabewerte zu entlocken, was den Einsatz von "SSS" als selbständig ablaufendes Unterprogramm ausschließt.

Die Verwendung von "ROM" kann diese Schwierigkeiten überwinden, weil sie uns erlaubt, "SSS" an einer Stelle 'aufzurufen', die sowohl hinter den Eingabeunterbrechungen als auch hinter dem Befehl 'SF 21' (Zeile 65), welcher die Ausgabeunterbrechungen bei Abwesenheit des Druckers verursacht, liegt. Ein geeigneter Einstiegspunkt ist die Zeile '06 LBL 05'. An dieser Stelle setzt das Programm voraus, daß die Längen 'S1', 'S2' und 'S3' schon in den Registern R_{00}, R_{02} bzw. R_{04} liegen, so daß das aufrufende Programm nur noch für diese Speicherung zu sorgen braucht:

```
01↑LBL "MAIN"
02 25
03 STO 00
04 35
05 STO 02
06 45
07 STO 04
08 XEQ "ROM"
09 "ERLEDIGT"
10 AVIEW
11 END
```

In diesem Beispiel-Programm "MAIN" führen wir der Einfachheit halber die Seitenlängen 25, 35 und 45 explizit auf. Beachten Sie, daß "SSS" das Register R_{00} selbst verwendet, weswegen wir zur Ablage der ROM-Adresse ein anderes wählen müssen. Wir wollen jetzt das Programm zum Laufen bringen:

1. Ändern Sie Zeile 14 von "ROM" in '14 ARCL 10' um.
2. GTO "SSS"
3. CLA
4. GTO 05 (oder GTO .006)
5. RCL b
6. STO M
7. ASTO 10
8. XEQ "MAIN"

Sobald "ERLEDIGT" in der Anzeige erscheint, finden wir die Ergebnisse

A1 = 95,74 in R_{01}
A2 = 33,56 in R_{03}
A3 = 50,70 in R_{05}
AREA = 435,31 in X.

Für nachfolgende Ausführungen von "MAIN" können die Schritte 1 bis 7 entfallen, sofern nur die in R_{10} abgelegte ROM-Adresse unversehrt bleibt.

Kapitel 7

VERGNÜGLICHE ABARTIGKEITEN

Der Hauptzweck der synthetischen Programmierung besteht darin, die Programmierfähigkeiten des HP-41C zu erweitern. Die Anwendungsprogramme in Kapitel 6 sind die Ergebnisse einer unvoreingenommenen Benutzung der synthetischen Funktionen, verbunden mit einer Vielzahl von Versuchen, abenteuerlichen Einfällen, emsigen Nachforschungen usw. Es sollte daher nicht verwundern, daß diese Untersuchungen der im Innern des HP-41C ablaufenden Vorgänge auch eine Reihe von Absonderlichkeiten zutage gefördert haben, die zwar keine praktische Bedeutung erkennen lassen, nichtsdestoweniger zu unterhaltsamem Spiel anregen. Dieses Kapitel beschreibt mehrere solcher Merkwürdigkeiten.

7A. 128 Töne?

Bei herkömmlicher Bedienung kann der HP-41C die den Byte-Kodes '9F 00' bis '9F 09' entsprechenden 10 Töne 'TONE 0' bis 'TONE 9' hervorbringen. Am Schluß von Kapitel 3 haben wir jedoch schon erkannt, daß der Kode '9F 0A' einen neuen Ton erklingen läßt, und zwar von längerer Dauer und geringerer Frequenz als sie die 10 Standard-Töne aufweisen. '9F 0A' erscheint, wie wir schon beim Henkersspiel in Abschnitt 6C gesehen haben, als Programmzeile in der Form 'TONE 0'. Wir können die Verfahren der synthetischen Programmierung dazu verwenden, der Vorsilbe '9F' jede beliebige der 128 Nachsilben aus den Zeilen 0 bis 7 der Byte-Tabelle anzuheften. Es stellt sich heraus, daß fast jede Zusammenstellung einen anderen Ton hervorbringt. Und zwar gibt es 16 verschiedene Tonhöhen, was gerade den 16 möglichen Werten des zweiten Nybbles einer Tonzeilen-Nachsilbe entspricht. Zwei Ton-Kodes, die sich nur im ersten Nybble der Nachsilbe unterscheiden, erzeugen Töne von gleicher Höhe doch von in der Regel verschiedener Dauer.

Im Programm-Modus wird jede 'TONE'-Zeile mit einer Nachsilbe bis hexadezimal '65' (dezimal 101) als 'TONE n' abgebildet, wobei 'n' die letzte Ziffer des Dezimalwertes des Nachsilben-Bytes ist. Für höhere Nachsilben erscheinen die Zeilen als 'TONE α', worin 'α' eine Einzelzeichen-Nachsilbe ist, wie z.B. 'TONE D' für '9F 69' oder 'TONE P' für '9F 78'. Tabelle 7-1 zeigt die Ton-Dauer und -Höhe für jede der 128 möglichen Nachsilben. Die Tabelle ist von Richard Nelson (*PPC Calculator Journal*, V7N1P21, 1980) zusammengestellt worden. Da es einige Fälle gibt, die doppelt auftreten, haben wir tatsächlich nur 114 'verschiedene' Töne.

ZWEITE NACHSILBEN-ZIFFER

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
Freq.(Hz)	175	197	225	263	315	394	525	629	788	1051	105	113	121	131	143	158	
0	TONE 0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	0
	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	2.20	2.20	2.70	3.50	0.80	2.30	
1	TONE 6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	1
	2.00	0.34	1.50	0.33	0.50	1.00	0.45	0.84	0.30	0.55	5.00	3.50	2.00	4.10	0.30	2.40	
2	TONE 2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	2
	0.025	1.13	2.35	2.00	1.35	0.023	0.023	0.35	0.70	0.52	0.85	0.45	3.20	0.18	1.36	0.13	
3	TONE 8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	3
	0.54	0.37	2.10	1.95	0.28	0.15	0.80	0.77	0.65	0.058	0.42	0.41	3.30	0.39	0.97	0.30	
4	TONE 4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	4
	1.88	2.35	0.40	0.24	1.05	0.29	0.032	0.24	0.14	0.15	3.70	0.30	3.76	3.40	0.89	0.90	
5	TONE 0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	5
	0.085	0.22	1.75	0.74	0.28	1.25	0.50	0.14	0.58	0.050	2.70	0.42	3.21	2.95	0.30	2.40	
6	TONE 6	7	8	9	0	1	A	B	C	D	E	F	G	H	I	J	6
	0.65	2.32	0.43	1.25	0.12	1.00	0.99	0.84	0.70	0.52	0.23	0.45	3.62	0.33	2.10	0.35	
7	TONE T	Z	Y	X	L	M	N	O	P	Q	⊢	a	b	c	d	e	7
	1.70	0.65	1.45	0.52	1.25	1.30	0.24	0.84	0.14	0.33	0.25	4.60	0.76	4.00	3.50	2.90	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

ERSTE NACHSILBEN-ZIFFER

TABELLE 7-1. TON-FREQUENZEN, -ZAHLEN UND -DAUER (IN SEKUNDEN)

Falls Sie mit diesen Tönen Ihre Versuche anstellen wollen, können Sie das Programm "TONE" laufen lassen. Es erzeugt selbständig 127 'TONE'-Programmzeilen (alle Zusammenstellungen außer 'TONE 0', '9F 00', die von Hand dazugesetzt werden kann). Nach der Ausführung von "TONE" bestehen die ersten 127 Zeilen des ersten Programmes im Speicher aus 'TONE'-Zeilen, deren Zeilen-Nummern mit den Ton-Nummern, durchgezählt von 1 bis 127, übereinstimmen. "TONE" ruft "DC" (Abschnitt 6D) und "CU" (Abschnitt 6G) als Unterprogramme auf. Bevor Sie "TONE" laufen lassen, muß 'SIZE 045 (oder größer)' festgesetzt werden. Nach dem Programm-Lauf ist die Zahl der verfügbaren Daten-Register um 43 vermindert.

Die 'synthetischen Klänge' sind auch nicht 'musikalischer' als die Standard-Töne. Nichtsdestoweniger lassen die zusätzlichen Tonhöhen und die Vielfalt der Tonlängen eine durchaus belebtere Tonsprache des HP-41C zu.

```
01♦LBL "TONE"     10 XEQ 03         19 43             28 INT
02 ,13            11 "┣♦"           20 XEQ "CU"       29 XEQ "DC"
03 STO 43         12 RCL [          21 BEEP           30 ASTO X
04 42             13 STO IND 44     22 RTN            31 CLA
05 STO 44         14 DSE 44         23♦LBL 03         32 ARCL Y
06♦LBL 01         15 GTO 01         24 "┣"(F2 9F 7F)  33 ARCL X
07 CLA            16 "┣"(F2 7F 9F)  25 ASTO X         34 END
08 XEQ 03         17 RCL [          26 ISG 43
09 XEQ 03         18 STO 00         27 RCL 43
```

"TONE"
78 BYTES
SIZE 045

7B. Tricks mit System-Flags

Während wir die synthetische Programmierung entwickelt haben, sind wir verschiedenen Beispielen absichtlicher Veränderungen des Zustands von gewöhnlich unerreichbaren System-Flags begegnet, wobei überraschende Erscheinungen (wie das Setzen des Batterie-Kontroll-Flags in Abschnitt 4D) oder nützliche Wirkungen (wie beim Textgehilfen in Abschnitt 5H) zutage traten. Es können noch weitere zum Teil recht amüsante Wirkungen durch das Setzen von System-Flags erzielt werden. Hier eine Gruppe von Routinen, die für die Erforschung des Flag-Registers nützlich sind:

```
01♦LBL "SAVE"     10 RTN            19 SF IND X
02 0              11♦LBL "FL"       20 RCL d
03 RCL d          12 24             21 STO [
04 XEQ "CS"       13 -              22 "┣ABCD"
05 RTN            14 X<> d          23 X<> \
06♦LBL "RE"       15 STO [          24 STO d
07 0              16 "┣♦♦♦"         25 END
08 XEQ "CR"       17 RCL [
09 STO d          18 X<> d
```

"SAVE"
18 BYTES
SIZE 002

"RE"
16 BYTES
SIZE 002

"FL"
41 BYTES

Bevor Sie Ihre Versuche mit Sytem-Flags beginnen, sollten sie die Routine "SAVE", welche den gegenwärtigen Inhalt von Register d holt und ihn (vermittels "CS") in R_{00} und R_{01} ablegt, aufrufen. Sie können Ihren Rechner dann jederzeit in seinen anfänglichen Flag-Zustand zurückführen, indem Sie die Routine "RE" (welche ihrerseits "CR" aufruft) ausführen.

"FL" verschiebt den Inhalt von Register d 'nach links' in das Gebiet, in welchem wir die Benutzerflags beherrschen und alle Bits verändern können, so daß wir jedes ins Auge gefaßte Systemflag (bis hinauf zu Flag 53) in der 'verschobenen' Lage zu setzen oder zu löschen vermögen. Anschließend werden die Bytes von Register d wieder an ihre Ausgangsstelle gebracht; sobald die Routine anhält, ist das ausgewählte Flag gesetzt. Beispielsweise stellt '49, XEQ "FL"' die Batterie-Anzeige an; '47, XEQ "FL"' beendet den Lauf mit gesetzter 'SHIFT'-Anzeige (eine daraufhin betätigte Taste führt ihre Umschaltfunktion aus).

Das Setzen von Flag 30 zaubert einige außergewöhnliche 'Kataloge' herbei. Diese Geister-Kataloge haben keine besondere Bedeutung, doch ist es fesselnd, die verschiedenen 'Einträge' zu beobachten, während der Katalog 'durchgeblättert' wird. Laut Thomas Cadwallader erhält man Zugang zu verschiedenen Katalogen, wenn vor dem Setzen des Katalog-Flags verschiedene Anzeige-Formate festgelegt werden. Wenn Sie einen solchen Katalog sehen möchten, tasten Sie 'FIX 9, 30, XEQ "FL", R/S'. Beachten Sie, daß sich diese Verzeichnisse wie jeder richtige Katalog anhalten und schrittweise durchsehen lassen.

Was es auch immer wert sein mag, jedenfalls können wir jetzt den PGRM-Modus mit '52, XEQ "FL"' einschalten. Sie haben sich vielleicht darüber gewundert, warum wir in Programmen wie "CODE" und "DECODE" so rücksichtslos waren, jeden 'Kram' ins Register d zu bringen - wieso springt der HP-41C eigentlich nicht in den PRGM-Modus, wenn Flag 52 während eines laufenden Programms gesetzt wird?

Das liegt ganz einfach daran, daß der Prozessor den Zustand der verschiedenen Flags nur zu bestimmten Zeitpunkten, nicht ständig überprüft; es geschieht daher solange nichts Verdrießliches, als diese möglicherweise Gefahr bringenden Flags vor der Überprüfung gelöscht werden. Allerdings gibt es Fallgruben: Um einmal zu erleben, was geschehen kann, ändern Sie "FL" ab, indem Sie hinter der Zeile '24 STO d' die Zeile '25 1' einfügen und dann '52, XEQ "FL"' ausführen. Der Prozessor schaltet *tatsächlich* in den PRGM-Modus um, doch weil das Programm schon läuft, beginnt der HP-41C, sich selbst zu programmieren, indem er den verfügbaren Platz mit '1'-Zeilen füllt, bis der Speicher voll ist; alsdann erscheint in der Anzeige "PACKING, TRY AGAIN"! Offensichtlich wird Flag 52 in dem Moment, in welchem eine einen Zahlen-Eintrag enthaltende Programmzeile auftritt, 'überprüft'.

Das Meldungsflag, Flag 50, ist das vielleicht bemerkenswerteste der System-Flags. Jedesmal, wenn es gesetzt wird, 'friert die Anzeige ein', unabhängig davon, was sich gerade darin befindet. Damit Sie sich vier verschiedene Möglichkeiten ansehen können, tasten Sie (sobald Sie die im letzten Versuch erzeugten

'1'-Zeilen gelöscht haben) hinter der Zeile '23 X<>N' eine Zeile '24 STOP' ein. Führen Sie dann '50, XEQ "FL"' aus, und beobachten Sie die Anzeige, wenn das Programm anhält. Drücken Sie 'SST', aber nur ganz kurz. Die (im 'SCI 0'-Format) abgebildete Zahl bleibt bestehen, obwohl sich mehrere Indikatoren auf das 'SST' hin ändern können. Die Betätigung der Korrektur-Taste bringt die Anzeige wieder in ihr vorheriges Format zurück. Versuchen Sie abermals '50, XEQ "FL"', halten Sie die SST-Taste aber diesmal solange niedergedrückt, bis Sie '25 STO d' erblicken; dann lassen Sie los - die Zeile '25 STO d' bleibt in der Anzeige zurück. Als nächstes lassen Sie wieder '50, XEQ "FL"' ablaufen, führen jetzt jedoch das 'STO d' von Hand aus (Sie müssen dazu in den USER-Modus schalten und Ihre Tastenzuweisung benutzen). Diesmal vereist die Anzeige mit dem Bild 'XROM 05,62'. Schließlich rufen Sie '50, XEQ "FL"' noch einmal auf und fahren mit 'R/S' fort. Die fliegende Graugans ist zur Ruhe gebracht worden! Um unseren kleinen Gefährten an einer anderen Stelle ruhen zu lassen, löschen Sie die 'STOP'-Zeile, bringen stattdessen ein paar 'LBL 01'-Zeilen hinein und versuchen ein weiteres '50, XEQ "FL"'.

Wohlüberlegtes Löschen des Flags 50 kann ebenfalls eindrucksvolle Erscheinungen auslösen. Während eines ablaufenden Programms fliegt die Graugans durch die Anzeige, solange Flag 50 gelöscht ist. Wenn jedoch ein 'VIEW nn' oder ein 'AVIEW' befohlen werden, wird Flag 50 gesetzt und der Inhalt des jeweiligen Registers angezeigt. Ein 'CLD' löscht Flag 50 und versetzt wieder die Graugans in die Anzeige. Wenn wir dagegen Flag 50 während eines 'VIEW' löschen, ohne 'CLD' zu benutzen, nimmt der Prozessor zwar wieder die Säumnisanzeige auf, läßt diesmal jedoch das durch das 'VIEW' ausgelöste Bild als Laufschrift durch die Anzeige ziehen. Tatsächlich können wir die fliegende Graugans durch jedes andere Zeichen oder durch eine Kette von bis zu 12 Zeichen ersetzen. Am leichtesten gelingt dieser Kunstgriff dadurch, daß unser Programm unmittelbar vor der 'VIEW'-Zeile ein 'RCL d' durchführt, also zu dem Zeitpunkt, zu welchem Flag 50 noch gelöscht ist. Sowie das 'VIEW' ausgeführt ist, stellt ein 'STO d' den vorherigen Zustand von Flag 50 wieder her:

```
01 "ABCD"     05 0
02 RCL d      06 LBL 01
03 AVIEW      07 SIN
04 STO d      08 GTO 01
```

Wenn Sie diese Routine laufen lassen, sehen Sie die Kette "ABCD" ihre Runden in der Anzeige ziehen. Die Zeilen 05-08 bilden eine Endlos-Schleife, die unsere Ersatz-Gans fliegen läßt. Die Zeilen 01-04 können mit irgendeiner 12-Zeichen-Kette in ein beliebiges Programm eingefügt werden und auf diese Weise Ihre Programme, wenn sie laufen, ganz zu Ihren eigenen machen. Der Trick wird im 'Henkersspiel' im Abschnitt 6C verwendet, um den 'vermuteten Buchstaben' in bis dahin ungewohnter Weise anzuzeigen.

Hier nun eine Rätselfrage, mit der Sie Ihren HP-41C-Scharfsinn prüfen können. Es gibt 216 unabhängige LCD-Segmente in der HP-41C-Anzeige - 12 'Vollzeichen' zu je 14 Segmenten = 168 Segmente; dazu 12 mal die 'Doppelpunkt & Komma'-Zeichen mit zusammen 36 Segmenten; schließlich noch 12 (unabhängige) Zahlen- und Wort-Indikatoren; mithin insgesamt 216 Stück. Frage: Wieviele Segmente können 'im Stand' (der HP-41C also angeschaltet aber kein Programm durchlaufend) gleichzeitig angeregt sein? Wenn Sie die Antwort gefunden zu haben glauben und Ihre Anzahl in einer Anzeige vorführen können, vergleichen Sie sie mit dem Ergebnis der Routine "DI". Falls Sie mehr Segmente als "DI" vorweisen können, haben Sie mehr gelernt, als dieses Buch Sie lehren kann!

```
01*LBL "DI"
                02 "**0*!
"
03 RCL [
04 ":::"
05 ASTO Y
06 ARCL Y
07 ARCL Y
08 ARCL Y
09 AVIEW
10 X<> d
11 END
```

(F7 F8 00 00 10 00 21 E8) Zeile 02

(F6 80 3A 80 3A 80 3A)

"DI"

37 BYTES

7C. Wir scheuchen die Graugans rückwärts

Um Sie nunmehr vollends in die synthetische Programmierung zu verstricken, möchte ich Ihnen ein weiteres Beispiel der Art 'man behauptete, es könne nicht gelingen, und wir vollbrachten es doch (mit synthetischer Programmierung natürlich - wie auch anders?!)' vorführen. Soweit es nämlich die Gemeinschaft der Benutzer betraf, gehörte die Kenntnis vom Vorhandensein einer rückwärtsgerichteten Graugans und den Hilfsmitteln, sie in die Anzeige zu locken, zu den bestgehüteten Geheimnissen des HP-41C. (Selbstverständlich kann es besser gehütete Geheimnisse geben; aber die sind eben noch Geheimnisse.) Um dieses scheue Wesen dazu zu überreden, sich in der Anzeige blicken zu lassen, brauchen wir eine letzte neue synthetische Funktion, 'FIX 10'.

Die Anzeige von Zahlen im HP-41C wird vom Zustand der Flags 36-41 bestimmt. Beispielsweise ist das 'FIX'-Format durch ein gesetztes Flag 40 und ein gleichzeitig gelöschtes Flag 41 festgelegt. Die Ziffernflags 36-39 setzen die Anzahl der hinter dem Dezimalkomma (oder Dezimalpunkt) vorgewiesenen Ziffern fest. Die vier Flags bilden zusammen eine Bit-Gruppe, die eine Hexadezimalziffer aufnehmen kann; die Anzahl der angezeigten Stellen hinter dem Komma ist gleich dem Wert dieser Bit-Gruppe. Gewöhnlich wird das 'FIX'-Format zwischen 'FIX 0' und 'FIX 9' eingestellt, wobei die Werte der vier Binär-Ziffern zwischen '0000' und '1001' liegen.

Mit den Mitteln der synthetischen Programmierung können wir jeden gewünschten Hexadezimalwert in die Ziffernflags bringen, so daß uns also noch die Formate 'FIX 10' bis 'FIX 15' zur Verfügung stehen. Natürlich kann die Anzeige auch dann nicht mehr als 10 Ziffern vorweisen; und tatsächlich stimmen die durch 'FIX 11' bis 'FIX 15' hervorgerufenen Zahlen-Abbildungen mit dem 'FIX 0'-Format überein. Doch 'FIX 10' bringt ein neues Anzeige-Format hervor. In Zahlen mit positiven Exponenten werden nur die zehn Mantissen-Ziffern angezeigt; der Exponent wird unterdrückt. Demgemäß wird beispielsweise die Zahl '1,234567891 E56' im 'FIX 10'-Format als '1,234567891' abgebildet, wohingegen das 'FIX 9'-Format sie - wie bekannt - als '1,2345678 E56' anzeigen würde.

Es gibt viele Möglichkeiten, den HP-41C in das 'FIX 10'-Format zu überführen. Wie oben vorgeschlagen, könnten wir geradewegs den Dezimalwert 10 (1010, hexadezimal 'A') in die Ziffernflags setzen, indem wir eine passende NNZ ins Register d bringen. Ein anderer leicht zu beschreitender Weg ist der, 'FIX 8' festzusetzen, wodurch Flag 36 gesetzt und die Flags 37-39 gelöscht werden, und dann '38, XEQ "FL"' auszuführen. Oder wir benutzen die synthetische Funktion 'FIX 10' (Kode '9C 0A'); und zwar entweder als Programmzeile (sie erscheint darin als 'FIX 0') oder durch Zuweisung an eine Benutzertaste (die von "KA" zu verarbeitenden Vor-/Nachsilben sind 156/10). Schließlich könnten wir auch noch die NNZ '0A 00 00 00 00 00 00' ins Register X setzen und 'FIX IND X' ausführen.

'FIX 10' ist ein recht beschränktes Hilfsmittel dafür, nur die Mantisse einer Zahl mit positivem Exponenten anzuzeigen. Unglücklicherweise arbeitet es nämlich nicht ganz sauber: sobald der Exponent so beschaffen ist, daß er bei einer Division durch 14 einen der Reste 10, 11, 12 oder 13 hinterläßt*), werden einige der Mantissenziffern mit dem aus Zeile 2 statt aus Zeile 3 der Byte-Tabelle stammenden Anzeige-Symbol dargestellt. Das größte Drama spielt sich ab, wenn der Exponent einen Wert hat, der kongruent 13 modulo 14 ist (also 13, 27, 41,...) und zu allem Unglück auch noch die Flags 28 und 29 gelöscht sind. In diesem Fall erscheint nur noch die erste Mantissenziffer normal. '1,234567891 E13' entartet in diesem Modus zu '1"#$%&'()¦' - gerade noch zu entziffern, wenn Sie eine Byte-Tabelle zur Hand haben, doch wahrlich nicht sehr bequem.

Die Anzeige von Zeichen aus der Zeile 2 ist keineswegs auf die Dezimalziffern '0' bis '9' beschränkt. Wenn Sie in die Byte-Tabelle schauen, erkennen Sie, daß jedes der Bytes '2C', '2E' und '3A' *zwei* zu ihm gehörige Anzeige-Symbole besitzt. In Alpha-Anzeigen werden diese Bytes stets durch die (rechten) 'Satzzeichen' "," ,"." bzw. ":" dargestellt. Doch in Zahlen-Anzeigen werden die jeweiligen Ziffern durch das linke Zeichen sichtbar gemacht - und siehe da!, in einer Zahl mit geeignetem Exponenten und Mantissen-Ziffern 'C' und 'E' neh-

---

*) $E \equiv r \pmod{14}$, $r = 10,11,12,13$

men die Graugänse "≮" und "≯" Gestalt an. Das zu '3A' gehörige Zahlzeichen ist, wie wir schon in Abschnitt 5A herausgefunden hatten, die 'Explosion im Kasten'.

Um ein Gänse-Paar einzufangen, setzen Sie auf irgendeine Weise 'FIX 10' fest, und löschen Sie die Flags 28 und 29. Alsdann:

	XEQ "CODE"	["CODE=?"]
"0100E00C000013"		
	R/S	["⁻*μ⁻⁻*"]
	← drücken	[1 ≯ ≮]

In der Anzeige können sich höchstens neun Gänse auf einmal tummeln. Das erste Zeichen in einer Zahlen-Anzeige stammt nämlich stets aus Zeile 3 - das Unschuldigste von ihnen ist vielleicht das 'Semikolon' (Byte '3B'). Beispielsweise wird die NNZ 'OB CC CC CC CC CO 13' folgendermaßen abgebildet:

[≯ ≮ ≮ ≮ ≮ ≮ ≮ ≮ ≮ ≮]

Auf die Gefahr hin, Honig zu versüßen, wollen wir noch einmal zum Programm "CODE", welches den Inbegriff synthetischer Programmierung darstellt, zurückkehren. Was könnte wohl schicklicher sein, als die Graugans rückwärts fliegen zu lassen, während "CODE" abläuft? Zeile 07 von "CODE" (Zeile 11, falls Sie "CODE" den Vorschlägen in Abschnitt 6F entsprechend abgeändert haben sollten) lautet '07 "├ ABCDEFG"'. Die sieben anzuhängenden Zeichen sind völlig frei wählbar - sie könnten ebensogut aus einer 7-Byte-NNZ, die eine rückwärts fliegende Graugans enthält, bestehen. Ersetzen Sie also Zeile 07 durch:

07 "├**⁻⁻⁻⁻*"	F8 7F OB CO 00 00 00 00 13
08 RCL d	
09 FIX 0	9C OA
10 CF 28	
11 CF 29	
12 CF 21	(nur bei angeschlossenem Drucker nötig)
13 VIEW M	
14 STO d	

Während die Graugans rückwärts durch die Anzeige zieht, folgt ihr ein "≯" nach. Vielleicht wohl, daß

Pitschkleck! - Ein Fleck. Ein jäher Schreck. -
Erleichtert fliegt die Graugans weg. *)

* * *

Nach diesem entzückenden Vermerke
Geht es zu End' mit dem löblichen Werke. *)

Sie haben gewaltige Arbeit geleistet und sehr viel über den HP-41C gelernt. Künftig sollte 'synthetisches Programmieren' für Sie 'gewöhnliches Programmieren'

*) W. Busch würde, hoffe ich, gestattet haben.

sein. Sie haben jetzt einen Anspruch darauf, unter dem Titel NNB - 'Nicht-Normalisierter Benutzer' - angesprochen zu werden!

ANHANG 1 : ZAHLENSYSTEME

Jeder HP-41C-Benutzer kennt das dezimale Zahlensystem, in welchem die Zehn die grundlegende Einheit oder 'Basis' bildet. Betrachten Sie die folgende Buchstaben-Gruppe:

A B C D E F G H I J K L M.

Wenn wir die Buchstaben abzählen, kommen wir auf 'dreizehn' Stück; als Kürzel pflegen wir dafür 'dezimal'

13

zu schreiben. Diese Kurzbezeichnung im Dezimalsystem ergibt sich aus der wiederholten Verwendung einer begrenzten Menge von Symbolen, nämlich der Zahlzeichen 0,1,2,...,9; wir vermeiden es, für jede mögliche Zahl ein anderes Zeichen zu benutzen. Wenn wir das aus zwei Zahlzeichen bestehende Symbol '13' niederschreiben, ist der Wert jedes einzelnen Zahlzeichens von seiner Stellung im Gesamtsymbol '13' abhängig. Genau ausgedrückt: '13' bedeutet '1 mal zehn plus drei'. Jedes Zahlzeichen wird mit der zu einer ganzzahligen Potenz erhobenen Basiszahl des Systems multipliziert:

$$13 = (1 \times 10^1) + (3 \times 10^0)$$

Ein Zahlzeichen nennen wir Ziffer; '13' ist demgemäß eine 'zwei-ziffrige Zahl'. Eine 'N-ziffrige Zahl' sieht so aus:

$$ab...mn = (a \times 10^{N-1}) + (b \times 10^{N-2}) + ... + (m \times 10^1) + (n \times 10^0)$$

Die Ziffern a, b,... können Werte zwischen 0 und 9, also bis zu der um 1 verminderten Basiszahl des Systems, zehn, annehmen.

Die Zahl zehn ist für einen Mathematiker durch nichts geheiligt. Ebensogut können wir eine beliebige andere Zahl als Basis wählen. Versuchen wir es z.B. mit acht: In einem System mit der Basis acht, 'oktales' Zahlensystem genannt, ist der Höchstwert einer Ziffer sieben. Die Zahl dreizehn wird so dargestellt:

$$15_8 = (1 \times 8^1) + (5 \times 8^0) = 13_{10}$$

Wenn gleichzeitig mehrere Zahlensysteme nebeneinander verwendet werden, sollten mehrziffrige Zahlen mit Indizes versehen sein, damit man erkennen kann, in welchem System sie gedacht sind. Wir können dann solche Gleichungen hinschreiben und verstehen:

$$15_8 = 13_{10}$$
$$1295_{10} = 2417_8$$

Die im HP-41C eingebauten Funktionen 'OCT' und 'DEC' bieten eine leicht zu handhabende Möglichkeit, Zahlen des Dezimalsystems in solche des Oktalsystems

umzuwandeln bzw. umgekehrt. Es ist wichtig, sich darüber klar zu sein, daß diese Umwandlungen nur die *Darstellung* einer Zahl ändern, nicht etwa die Zahl selbst. 7654_8 HP-41C's bleiben die gleiche Anzahl, auch wenn wir stattdessen 4012_{10} HP-41C's schreiben.

Für unsere Untersuchungen des HP-41C sind zwei andere Systeme von Belang. Das eine ist das 'binäre Zahlensystem' mit der Basis zwei: Es benötigt nur zwei Symbole, '1' und '0'. Unsere Glückszahl dreizehn hat binär diese Gestalt:

$$1101_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 13_{10}$$

Das Binärsystem ist außerordentlich zweckmäßig für die Verwendung in elektronischen Rechenmaschinen, weil jede Ziffer nur zwei Werte annehmen kann, was sich mechanisch oder elektronisch besonders leicht verwirklichen läßt. Jede binäre Ziffer, üblicherweise auch 'Bit' genannt, kann als Zustand irgendeiner einfachen Art von Schalter dargestellt werden, wobei '1' die Bedeutung von 'an' und '0' die von 'aus' hat. Alle Vorgänge im Rechner geschehen binär. Die Umwandlung in eine dezimale Anzeige findet nur zur Bequemlichkeit für den Benutzer statt. Tatsächlich sind auch die Dezimalzahlen im Innern des HP-41C 'binär-dezimal' verschlüsselt ('BCD' für 'binary-coded-decimal'), und zwar in der Weise, daß eine Dezimalziffer in je vier binären Bits untergebracht ist. Die 13 wird beispielsweise so verschlüsselt:

$$13 \longrightarrow 0001\ 0011$$

Jede Vier-Bit-Gruppe kann aber eigentlich 16 verschiedene Zahlen darstellen (0000, 0001,...,1111), was uns zu dem letzten Zahlensystem, welches wir noch betrachten müssen, führt - zum 'hexadezimalen' System mit der Basis 16. In hexadezimaler Schreibweise kann jede Ziffer Werte zwischen Null und fünfzehn annehmen. Daher reichen die Symbole '0' bis '9' nicht aus, und wir fügen die Symbole 'A' bis 'F' hinzu:

A = zehn
B = elf
C = zwölf
D = dreizehn
E = vierzehn
F = fünfzehn

In diesem Sinne haben wir z.B.:

$$8A_{16} = (8 \times 16^1) + (10 \times 16^0)$$

$$1FF_{16} = 511_{10}$$

Beachten Sie noch einmal, daß die einzelnen Ziffern für sich genommen stets eindeutig zu verstehen sind. Erst wenn wir sie zu mehrziffrigen Zahlen zusammenstellen, müssen wir durch einen Index auf das Zahlensystem, in dem gerechnet wird, hinweisen.

Anhang 2 : Barcode Programme

"CODE"

SPEICHERPLATZBEDARF: 28 REGISTER

ROW 1 (1 : 2)

ROW 2 (2 : 7)

ROW 3 (7 : 9)

ROW 4 (10 : 15)

ROW 5 (15 : 21)

ROW 6 (22 : 29)

ROW 7 (30 : 37)

ROW 8 (37 : 43)

ROW 9 (44 : 51)

ROW 10 (51 : 59)

ROW 11 (59 : 66)

ROW 12 (66 : 73)

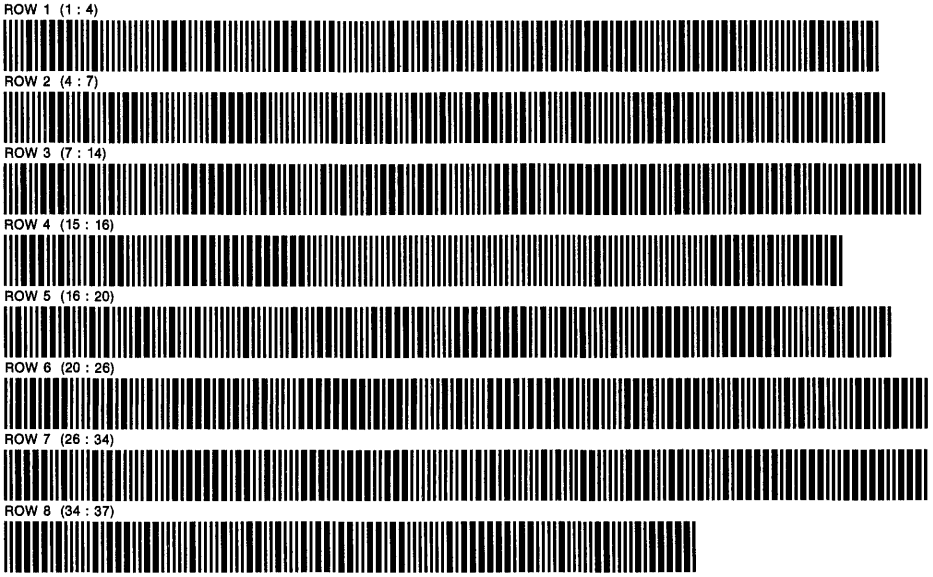
ROW 13 (73 : 79)

ROW 14 (79 : 84)

ROW 15 (85 : 89)

"REG"

SPEICHERPLATZBEDARF: 15 REGISTER



Wegen eines Versehens bei der Bearbeitung der Programme des Autors für die Barcode-Verschlüsselung ist die Zeile 34 des Programmes "REG" zu

'34 "B2-"' statt zu '34 "REG-"'

(wie im Text aufgelistet) geworden. Das "B2-" bezieht sich auf die im PPC-Klub übliche Bezeichnung des Hardware-Fehlers 'Bug 2' im HP-41C, der der Auslöser der synthetischen Programmierung war. Sie können die Zeile selbstverständlich abändern, sobald das Programm einmal in den Rechner gelesen wurde.

"KA" UND "EF"

SPEICHERPLATZBEDARF: 59 REGISTER

ROW 1 (1 : 4)

ROW 2 (4 : 12)

ROW 3 (12 : 18)

ROW 4 (18 : 22)

ROW 5 (23 : 30)

ROW 6 (31 : 36)

ROW 7 (36 : 40)

ROW 8 (40 : 45)

ROW 9 (46 : 54)

ROW 10 (55 : 64)

ROW 11 (65 : 74)

ROW 12 (74 : 81)

ROW 13 (82 : 86)

ROW 14 (87 : 93)

ROW 15 (93 : 97)

ROW 16 (98 : 104)

ROW 17 (104 : 108)

ROW 18 (108 : 115)

ROW 19 (115 : 121)

ROW 20 (122 : 125)

ROW 21 (126 : 132)

ROW 22 (132 : 139)

ROW 23 (139 : 146)

ROW 24 (147 : 153)

ROW 25 (153 : 158)

ROW 26 (158 : 165)

ROW 27 (165 : 168)

ROW 28 (169 : 175)

ROW 29 (175 : 181)

ROW 30 (182 : 188)

ROW 31 (189 : 195)

ROW 32 (195 : 196)

"DECODE"

SPEICHERPLATZBEDARF: 29 REGISTER

ROW 1 (1 : 3)

ROW 2 (3 : 9)

ROW 3 (10 : 17)

ROW 4 (17 : 23)

ROW 5 (24 : 30)

ROW 6 (30 : 36)

ROW 7 (36 : 42)

ROW 8 (43 : 48)

ROW 9 (48 : 54)

ROW 10 (54 : 62)

ROW 11 (63 : 70)

ROW 12 (70 : 77)

ROW 13 (77 : 85)

ROW 14 (85 : 92)

ROW 15 (92 : 99)

ROW 16 (99 : 103)

"HM"

SPEICHERPLATZBEDARF: 56 REGISTER

ROW 1 (1 : 4)

ROW 2 (5 : 10)

ROW 3 (11 : 11)

ROW 4 (12 : 17)

ROW 5 (17 : 23)

ROW 6 (23 : 28)

ROW 7 (29 : 36)

ROW 8 (36 : 42)

ROW 9 (42 : 50)

ROW 10 (51 : 59)

ROW 11 (59 : 65)

ROW 12 (66 : 70)

ROW 13 (70 : 74)

ROW 14 (75 : 77)

ROW 15 (77 : 83)

ROW 16 (84 : 90)

ROW 17 (90 : 93)

ROW 18 (94 : 98)

ROW 19 (98 : 103)

ROW 20 (103 : 111)

ROW 21 (111 : 116)

ROW 22 (116 : 125)

ROW 23 (125 : 132)

ROW 24 (132 : 139)

ROW 25 (139 : 146)

ROW 26 (147 : 155)

ROW 27 (156 : 163)

ROW 28 (164 : 172)

ROW 29 (173 : 179)

ROW 30 (179 : 183)

Anhang 3 : Die Barcode Zeichen-Tabelle

Dieses Buch stellt die Ergebnisse eines ganzen Jahres synthetischer Programmierung auf dem HP-41C vor. In dieser Zeit hat es sich klar herausgestellt, daß der optische Lesestift ein leistungsfähiges Werkzeug für die synthetische Programmierung ist. Diese Erkenntnis ist im wesentlichen der Arbeit von Jacob Schwartz (einem PPC-Mitglied, der für die äußerst saubere Gestaltung der Barcode-Vorlagen verantwortlich zeichnet) zu verdanken. Die meisten für die synthetische Programmierung entwickelten Lesestift-Techniken stecken noch in den Kinderschuhen; ein paar Beispiele werden Sie von dem Vorteil dieses Gerätes überzeugen.

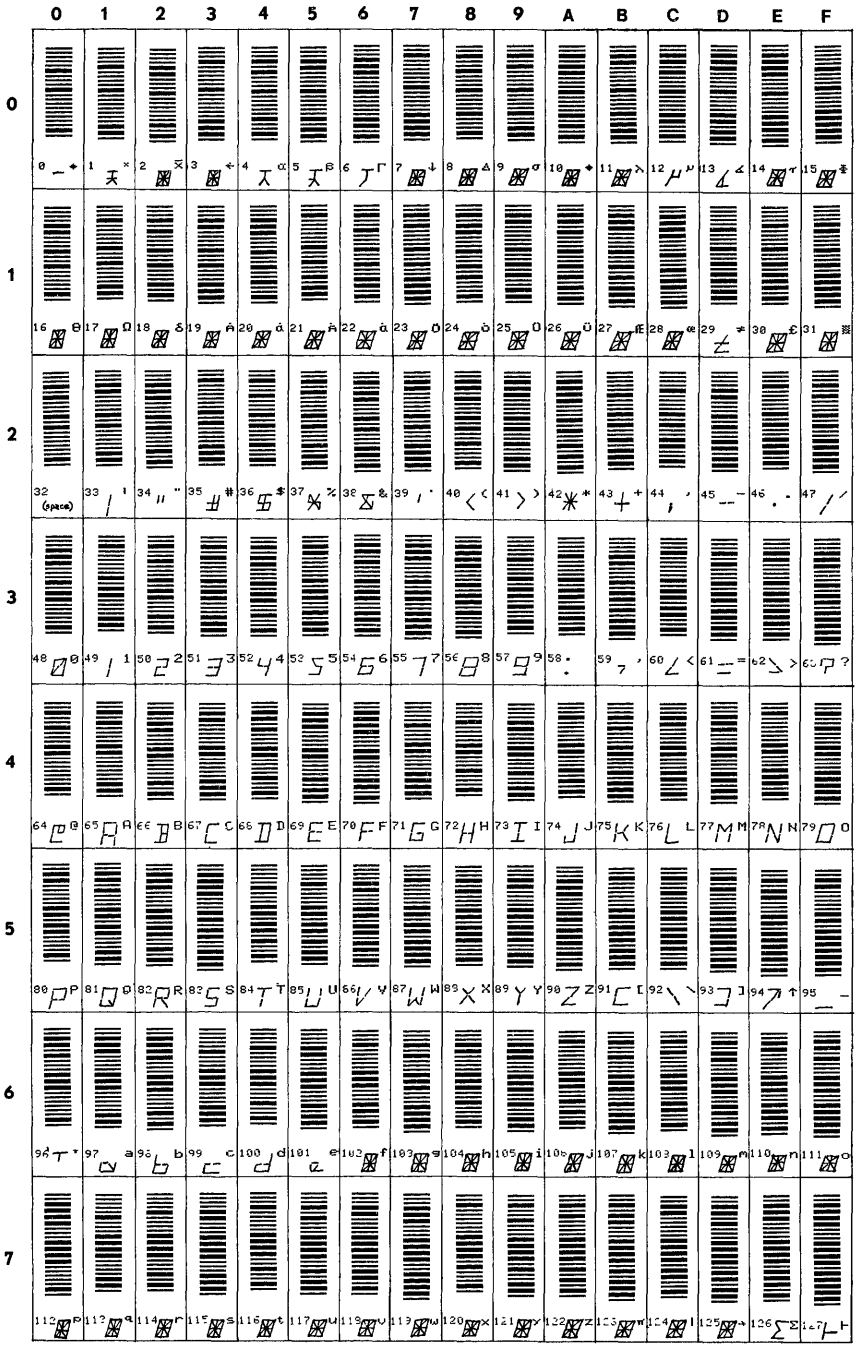
*** Benutzen Sie stets eine Schutzfolie, wenn Sie
die Strich-Kodes mit dem Lesestift abtasten!

Fürs erste können Sie Ihren Barcode-Vorlagen die Strich-Kodes für den Byte-Hüpfer und den Q-Boten hinzufügen:

BYTE-HÜPFER: 

Q-BOTE: 

Die auf der nächsten Seite stehende Barcode Zeichen-Tabelle wurde von Jacob Schwartz beigesteuert. Jedes Nicht-Standard Alpha-Zeichen aus der oberen Hälfte der Byte-Tabelle kann durch einfaches Abtasten des Strich-Kodes in der zugehörigen Tabellen-Stelle einer Alpha-Kette unmittelbar angefügt werden, und zwar sowohl im Alpha-Register als auch in einer Programm-Textzeile.



DIE BARCODE ZEICHEN-TABELLE

NACHWORT DES ÜBERSETZERS

Zum Schluß möchte ich dem Autor für das große Vergnügen, das er mir bei der Übersetzung des Buches bereitet hat, danken. Dieses Vergnügen wurde aus zwei Quellen gespeist: einmal aus dem Riesenspaß, unter der sachkundigen Leitung des Autors "mit Flinte und Kamera" den HP-41C zu durchstöbern, und zum anderen aus der kurzweiligen Lektüre des mit soviel Humor gewürzten Textes. Ich habe mich redlich bemüht, diesen Humor ins Deutsche hinüberzuretten. Von Sportgeist beseelt habe ich dabei um so manche Stelle wenn nicht gerungen so doch gefeilscht. Gelegentlich war, wie man unschwer erkennt, ein wenig Zwang nötig. Ich glaube, daß nur einige sprachraumbezogene Feinheiten völlig verloren gegangen sind. An ganz wenigen Stellen habe ich - etwas freier werdend, jedoch nie die "Werktreue" ernstlich vernachlässigend - auf Assoziationen gezielt, die nur beim deutschsprachigen Leser zu wecken sind.

Noch eine letzte Bemerkung: In der deutschen Ausgabe ist ein "Hoch" auf den Autor versteckt. Wer findet es?

Berlin, im Januar 1982. H.D.

ISBN 3-88538-800-6