Der einfache Einstieg in die faszinierende Computertechnik

Buch 4

Elektor Verlag

Junior-Computer 4

Der einfache Einstieg in die faszinierende Computertechnik

A. Nachtmann
G. Nachbar

ISBN 3-921608-25-2

Elektor Verlag GmbH, 5133 Gangelt 1

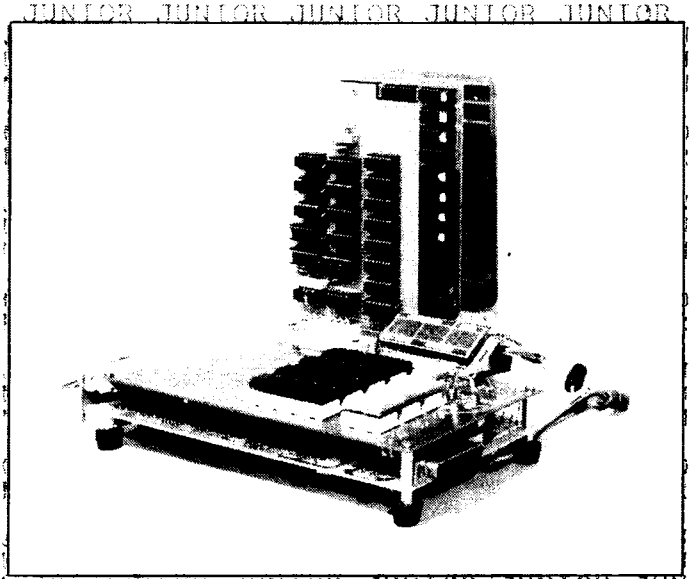
Junior-Computer 4 . . .

. . . ein Buch, das ganz im Zeichen der Software steht. Wie lassen sich (eigene) Programme schnell, nein, superschnell in den Junior-Computer eingeben? Wie arbeiten die neuen Systemprogramme? Die Antworten auf diese und andere Fragen finden Sie in diesem Buch!
Zuerst stellt Kapitel 13 ein brandneues Systemprogramm vor. Es trägt den Namen "PME" und vereinigt die Vorzüge des Standard-Editors mit den Eigenschaften und Möglichkeiten von PM. Das Eingeben und assemblieren von langen Anwenderprogrammen wird damit fast zum Kinderspiel.
Es folgen dann drei Kapitel, die in allen Einzelheiten, Byte für Byte, die Systemprogramme PM (Kapitel 14), PME (Kapitel 15) und TM einschließlich der beiden Kassetten-"Super-Subroutinen" (Kapitel 16) erklären.
Eine Entscheidung, die wir zugunsten einer ausführlicheren Beschreibung der Systemprogramme getroffen haben, findet sicherlich auch Ihre Zustimmung: Es fehlt das in Buch 3 angekündigte Kapitel über den VIA. Wenn Sie sich eingehender mit dem 6522-VIA beschäftigen wollen, steht genügend gute Literatur* hierüber zur Verfügung.
Die kommentierten Programm-Listings der beschriebenen Systemprogramme sind im Anhang, hinten im Buch, zusammengefaßt.

A. Nachtmann
G.H. Nachbar

*Literatur zum VIA 6522:
R6522 Datenblatt (Rockwell)
SY6522/SY6522A Datenblatt (Synertek)
Section 6, R6500 Hardware Manual (Rockwell)

Inhalt



13

Superschnell editieren und assemblieren

Der PM-Editor

Das Entwickeln eines Programms ist eine wesentlich interessantere Beschäftigung als das nachfolgende Eingeben des Programms in den Computer. Deshalb sollte letzteres möglichst schnell und einfach vor sich gehen, damit für ersteres mehr Zeit bleibt. Mit dem in Buch 2 beschriebenen Editor und Assembler lassen sich Programme schon recht bequem und sicher in den Arbeitsspeicher des Computers laden. Diese "Handarbeit" ist nur ein einziges Mal notwendig, denn durch Aufzeichnen auf eine Tonbandkassette wird die eingetippte Software computergerecht dauergespeichert. Wie sich die Prozedur der Eingabe weiter vereinfachen und beschleunigen läßt, das zeigt dieses Kapitel.

In Buch 3 kam zum Standard-Monitor das Systemprogramm PM hinzu. Hier folgt nun ein ähnlicher Schritt: Wir gehen vom Standard-Editor zum PM-Editor über. Aus seinem Namen (abgekürzt PME) geht bereits hervor, daß dieser Editor auf dem Systemprogramm PM aufbaut. Die große ASCII-Tastatur löst auch beim Editieren und Assemblieren die kleine hexadezimale Tastatur ab; an die Stelle des Siebensegmentdisplays treten der Bildschirm und der Drucker. Auch PME arbeitet noch mit hexadezimalen Labeln. Insbesondere im Anfangsstadium der Programmentwicklung führt das schneller zum Ziel als das Arbeiten mit einem "großen" Editor und Assembler.

Was die Bedienung betrifft, so kommen verschiedene neue Tastenfunktionen hinzu; sie dienen hauptsächlich der schnellen

und bequemen Eingabe. Der Bildschirm gibt nicht nur eine einzige Instruktion, sondern eine ganze Liste von Instruktionen gleichzeitig wieder. Und nicht zuletzt sei vermerkt, daß sich die Anzahl der notwendigen Tastenbetätigungen im Vergleich zum Standard-Editor um ca. 30% verringert.
In diesem Kapitel wollen wir PME gründlich kennenlernen. Dabei werden wir auch verschiedene Programme besprechen, die von PM-Subroutinen Gebrauch machen. Mit ihnen wird die Reihe von Programmbeispielen aus Kapitel 12 (Buch 3) fortgesetzt.
Das Durcharbeiten dieses Kapitels und das Erlernen des Umgangs mit PME erfordert einen verhältnismäßig hohen Zeitaufwand. Diese Investition ist jedoch, wie die Zeit schnell lehren wird, mit Sicherheit lohnend.

Der Junior wächst weiter

Das äußere Wachstum ist zwar abgeschlossen; die Hardware wurde bereits in Buch 3 vollendet.
Das innere Wachstum schreitet dagegen fort: die Software wird weiter ausgebaut.
Sowohl im TM- als auch im PM-EPROM sind noch freie Speicherkapazitäten von ungefähr dreiviertel Kbyte vorhanden. Diese freien EPROM-Kapazitäten können natürlich nicht ungenutzt bleiben. Was das PM-EPROM betrifft, so sind die noch verfügbaren Speicherzellen nun für das Systemprogramm PME reserviert. PME ist ein Editor, der auf dem bereits vertrauten Systemprogramm PM basiert. Das bedeutet: höherer Bedienungskomfort durch die ASCII-Tastatur und bessere Übersicht dank des Bildschirms.
Zuerst wollen wir etwas näher erläutern, weshalb PME entwickelt wurde. Daß hierfür in einem der beiden EPROMs noch Platz vorhanden war, ist natürlich allein kein stichhaltiger Grund. Eine Reihe anderer Systemprogramme sind denkbar, die nicht weniger nützlich erscheinen. Die freie Speicherkapazität im TM-EPROM wird deshalb wahrscheinlich ebenfalls nicht lange brachliegen.

Weshalb PME?

Wir verfügen bis jetzt nur über den Standard-Editor, beschrieben in den Kapiteln 5 und 8 aus Buch 2. Der Standard-Editor erwies sich beim Erstellen von Programmen als überaus wichtige Hilfe, denn er ermöglicht das Arbeiten mit hexadezimalen Labeln. Die Alternative hierzu sind Label in Form von Worten, die aus maximal sechs Zeichen bestehen. Solche Label begegnen uns in den Listings am Schluß von Buch 2 und am Ende dieses Buchs. Erstellt wurden diese Listings mit einem "großen" Assembler/

Editor, einem Systemprogramm, das wesentlich umfangreicher als der noch freie Bereich von dreiviertel Kbyte im PM-EPROM ist. Wollte man den Junior damit ausrüsten, dann wäre ein weiteres EPROM notwendig. Es gibt aber noch andere Gründe, die einen "großen" Editor für den Junior weniger zweckmäßig erscheinen lassen:
Grundlage für die Bearbeitung eines Anwenderprogramms ist eine möglichst vollständige, schriftliche Dokumentation. Alle (echten!) Label müssen bekannt sein; ferner müssen alle verwendeten Speicherplätze im Voraus mit einem Namen (wie z.B. POINTL oder BYTES) bezeichnet werden. Man nennt dies auch "Deklarieren". Das gleiche gilt für Subroutinen, deren Adresse festliegt, für I/O-Einheiten und so weiter.
Nehmen wir an, daß wir um vier Uhr mit dem Schreiben eines Programms beginnen und um fünf Uhr wissen möchten, ob dieses Programm läuft. Ferner wollen wir unterstellen, daß anschließend noch der eine oder andere Programmierfehler beseitigt werden muß. Wir werden dann mit hoher Wahrscheinlichkeit folgendes feststellen: Zum einen muß beim Arbeiten mit einem "großen" Editor/Assembler viel Vorarbeit geleistet werden, zum anderen besteht die Gefahr, daß sich ein Teil dieser Vorarbeit als nutzlos erweist.
Leistet dagegen ein mit hexadezimalen Labeln arbeitender Editor bei der Eingabe Hilfestellung, zum Beispiel der Standard-Editor des Juniors oder PME, dann ist das Programm um halbfünf im Computer. Bis halbsechs sind wir vermutlich damit beschäftigt, das Programm weiter zu verbessern und zu vervollständigen.
Schlußfolgerung: Wir arbeiten mit hexadezimalen Labeln. Besonders während der Entstehungsphase eines Anwenderprogramms bietet das erhebliche Vorteile. Außerdem ist kein neuer Assembler notwendig; der vorhandene Assembler aus Kapitel 9 in Buch 2 bleibt voll einsatzfähig. Mit der Verwendung hexadezimaler Label ist auf der anderen Seite ein kleiner Nachteil verbunden, der aber kaum ins Gewicht fällt: Es dürfen keine Label gewählt werden, die identisch mit dem rechten Byte (ADL) von absoluten Adressen sind. Das sind Adressen, die bereits vor dem Editieren und Assemblieren festliegen. Ferner ist die Anzahl der Label nicht unendlich groß: etwas mehr als achtzig stehen zur Auswahl. Doch das reicht im allgemeinen auch für extrem lange Programme aus.

So viel zum wichtigsten Vorteil des alten und auch des neuen Editors. Die relativ eng begrenzten Möglichkeiten des Standard-Editors sind gleichzeitig die Existenzberechtigung für PME. Führen wir uns die Grenzen des Standard-Editors noch einmal vor Augen:

- Zuerst seien die relativ wenigen Tastenfunktionen sowie das Siebensegmentdisplay genannt, das nur eine einzige Instruktion gleichzeitig wiedergeben kann. Für den Standard-Junior-Computer reicht dies aus, doch nach den in Buch 3 beschriebenen Erweiterungen der Hardware (ASCII-Tastatur und Bildschirm) erscheint auch ein anderer Editor-Standard wünschenswert.
- Dann die Fehlermeldungen. Der Standard-Editor kennt ausschließlich die Fehlermeldung "EEEEEE"; sie ist nur auf dem Display sichtbar, solange die zur Fehlermeldung Anlaß gebende Taste gedrückt bleibt. Es kann deshalb vorkommen, daß man das "EEEEEE" versehentlich übersieht. Die Fehlermeldung des Standard-Editors gibt keinerlei Auf-

schluß über die Art des aufgetretenen Fehlers.

- Keine Fehlermeldung in folgendem Fall: Der Standard-Editor zeigt keine Reaktion, wenn der mit BEGAD und ENDAD spezifizierte Speicherbereich nach der Eingabe von zu vielen Labeln und Instruktionen überläuft. Ein Beispiel für diesen Fall ist das Programm PLAY (Buch 2, Seite 55 . . . 58). Der verfügbare Bereich ØØØØ . . . ØØEØ reicht zwar für die assemblierte Version von PLAY aus (sie enthält keine Label mehr), nicht jedoch für die Fassung des Programms vor dem Assemblieren. Der Speicherbereich ist um einige Plätze zu kurz: die abschließende "77" landet auf Speicherplatz ØØE3 (BEGADH!). Das läßt sich hier dadurch ändern, daß man das Label 9Ø (PLAY) streicht. Dieses Label wird nämlich gar nicht angesprungen! Für die Version, die noch die Label enthält, ist der Bereich dann groß genug, . . . wenn nicht ein anderer Umstand hinzukäme: Das erste Label, das während des Assemblierens gefunden wird, überschreitet die Zahl der verfügbaren Speicherplätze (siehe Bild 1). Fazit: Wir setzen PLAY auf Speicherseite Ø2, dort ist reichlich Platz vorhanden.
  Übrigens: Wir können nun frei über einen zusammenhängenden RAM-Bereich von 1,5 kByte (Adressen Ø2ØØ . . . Ø7FF) verfügen. Die Gefahr des Überlaufens ist damit im Vergleich zur Standard-Ausführung des Juniors stark herabgesetzt.
- Viel Tipparbeit. Die mit Abstand am häufigsten vorkommende Tastenfunktion ist INPUT. Meistens schließt sich eine einzugebende Instruktion oder ein Label unmittelbar an die vorangegangene Instruktion oder Label an. Ist es da wirklich notwendig, vor jeder Instruktion und jedem Label noch einmal die INPUT-Taste zu drücken? Nein, man kann das Editor-Programm auch so gestalten, daß die INPUT-Tastenfunktion beim Eingeben von numerischen Daten (Ø . . . F) automatisch wirksam ist. Nur wenn eine andere Tastenfunktion, z.B. INSERT oder SEARCH gewählt wird, tritt INPUT außer Kraft. Dieses Verfahren reduziert den Umfang der Tipparbeit nicht unerheblich.

Willkommen PME!

Beim Standard-Editor erscheinen abwechselnd zwei Informationen unterschiedlicher Art auf dem Display: Die Wiedergabe der Tastenbetätigung wird jedesmal von der Reaktion des Junior-Computers überschrieben. Auf eine Gesamtübersicht aller bisherigen Tastenbetätigungen und Computerreaktionen muß hier zwangsläufig verzichtet werden. Auf den Bildschirm des Terminals (siehe Buch 3) können dagegen mit Hilfe von PM bei weitem mehr Zeichen pro Zeile und wesentlich mehr als nur eine einzige Zeile gleichzeitig geschrieben werden. Ferner erzeugt der UART, ein Baustein des Elekterminals, Hardware-Echos der Tastenbetätigungen auf dem Bildschirm. Das Drücken einer Taste wird auf dem Bildschirm festgehalten, ohne daß dazu Software notwendig ist.
Ergebnis dieser Überlegungen: Auf dem Bildschirm oder auf Papier können die auszudruckenden Informationen in zwei Spalten angeordnet werden. Eine Spalte ist für die Tastenaktionen des Operators bestimmt, die andere gibt die Reaktionen des Junior-Computers über sein Sprachrohr PME wieder. Oder anders ausgedrückt: Eine Spalte spiegelt unsere Aktivitäten,

während in der anderen die Antworten des Computers erscheinen.
Zwei Spalten also: Links die Computer-Reaktionen, rechts die menschlichen Aktionen. Eine Aktion des Operators wird auf der betreffenden Zeile in der rechten Spalte sichtbar gemacht, während für die Antwort des Computers die linke Hälfte der nächsten Zeile reserviert ist. Eine Ausnahme von diesem Schema machen nur die Label. Wie wir noch sehen werden, werden sie über die volle Breite des Bildschirms geschrieben.
Nun noch einmal zu den Fehlermeldungen. Bereits bei PM haben wir unterschiedliche Rückmeldearten kennengelernt: "JUNIOR" und "WHAT?". Auch bei PME begegnen wir allgemeinen Rückmeldungen und Fehlermeldungen. Die Fehlermeldungen sind jedoch näher spezifiziert, so daß sich ein Fehler schneller aufspüren und beseitigen läßt.
Auch die Adressen werden ausgedruckt. Da auf dem Bildschirm genügend Platz vorhanden ist, kann auch jede zu einem Opcode gehörende Adresse im Bild erscheinen. Die aktuelle Adresse läßt zum Beispiel erkennen, wie weit der Speicher schon gefüllt ist.
PME kann aber noch mehr. Zum Beispiel: schnellere und bequemere Eingabe von BEGAD und ENDAD, vier Startadressen statt zwei, eine ganze Reihe zusätzlicher Tastenfunktionen, weniger Zeitaufwand für das Assemblieren, Liefern von Informationen über Labeladressen und anderes mehr.
Nach diesen einführenden Betrachtungen wollen wir uns nun den Details des PM-Editors widmen.

PME - Betrachtung aus der Nähe

A. Allgemeines

Das Systemprogramm PME ist im bisher freien Teil des PM-EPROMs untergebracht. Dieses EPROM trägt die Bezeichnung IC5 und befindet sich auf der Interfacekarte. Die Bestellnummer beim Elektor-Software-Service (ESS) für PM ohne PME lautete ESS 507, für PM zusammen mit PME und noch einigen weiteren Bytes (siehe an anderer Stelle in diesem Buch) lautet sie ESS 507N. Seit Juni 1981 werden auch die unter der Bestellnummer ESS 507 an den Elektor-Software-Service eingeschickten EPROMs stillschweigend zusätzlich mit PME programmiert. Junior-Computer-Freunde der ersten Stunde, in deren PM-EPROM PME und die erwähnten zusätzlichen Bytes fehlen, können ihr EPROM nachprogrammieren lassen. Näheres ist der jeweils neuesten Ausgabe der Zeitschrift "Elektor" zu entnehmen.
Das Programm PME füllt die Speicherplätze 14F8 . . . 17FD des PM-EPROMs. PME macht von verschiedenen PM-Subroutinen Gebrauch. Das bedeutet, daß PME über PM gestartet werden muß; dazu ist abhängig von der jeweiligen Situation eine von vier Startadressen zu benutzen. Das Starten von PME über den Standard-Monitor ist zwar möglich, es kann jedoch Komplikationen mit sich bringen.

B. Kaltstart

Startadresse: $ 1500

Der Kaltstart von PME ist dann erforderlich, wenn ein Programm von Grund auf neu editiert werden soll. Für Anwenderprogramme, die schon

einmal editiert oder sogar assembliert wurden, sind andere Startadressen zuständig. Davon wird später die Rede sein.
Was ist zu tun, um PME kalt zu starten?
Zuerst wird PM gestartet:
RST 1 Ø Ø Ø GO RUB OUT (=RES)
Damit befinden wir uns im Systemprogramm PM. Jetzt kann der Kaltstart von PME folgen:
1 5 Ø Ø SP R
(Zur Erinnerung: "SP" bedeutet "Zwischenraumtaste", nicht etwa Buchstabentaste "S" und "P"!).
PME ist nun gestartet. Es folgt die Rückmeldung:
BEGAD, ENDAD

Der nächste Schritt: Eingabe der Beginnadresse BEGAD und der Endadresse ENDAD des Speicherabschnitts, in den das Programm geladen werden soll. Bei einem Kaltstart muß dies immer ein RAM-Bereich sein. Die Bedienfolge ist die gleiche wie bei PM-Tastenfunktion M (Hexdump). Zuerst geben wir BEGAD ein, drücken die Kommataste und lassen dann ENDAD folgen. Schließlich wird noch Taste CR gedrückt. Auch bei PME können die links stehenden Nullen weggelassen werden.
Ein Beispiel. Wir belegen den altvertrauten RAM-Abschnitt Ø2ØØ . . . Ø3FF:
BEGAD, ENDAD: 2ØØ,3FF CR
PM EDITOR
Ø2ØØ 77

Die Rückmeldung "PM EDITOR" bedeutet, daß wir mit der Eingabe von Instruktionen und Labeln beginnen können. Die "77" bei BEGAD kennen wir schon vom Standard-Editor: Dies ist das EOF-Zeichen, das während der Eingabe von Instruktionen oder Labeln zu höheren Adressen hin durchgeschoben wird. Es ist daher nicht verwunderlich, daß die erste Instruktion oder das erste Label nach einem Kaltstart mit Hilfe der Insert-Taste I eingegeben werden muß (siehe Abschnitt G, Nummer ⑤ dieses Kapitels).
Der Kaltstart ist bei PME mit weniger Tipparbeit verbunden als beim Standard-Editor. Wir haben keine Last mehr mit ENDADL=$ØØE2 usw. Das ist eine wesentliche Erleichterung.
Noch eine Anmerkung: Wenn man ENDAD niedriger als BEGAD wählt, wird im Gegensatz zur PM-Tastenfunktion M kein Fehler angezeigt. ENDAD muß stets höher als BEGAD liegen, das dürfte eigentlich selbstverständlich sein.

C. Warmstart

Startadresse: $1533

Auch den Begriff "Warmstart" und seine Bedeutung kennen wir bereits vom Standard-Editor. Der Warmstart ermöglicht die Rückkehr zum Editor, wobei die Inhalte der nachfolgend genannten Adreßpointer vom Vorgeschehen bestimmt werden:

BEGAD : Beginnadreßpointer
ENDAD : Endadreßpointer
CEND : variabler Endadreßpointer
CURAD : Displaypointer

Anmerkung: CURAD zeigt beim Standard-Editor auf die Adresse mit dem Opcode der Instruktion oder mit dem Label, das auf dem Display sichtbar ist. Bei PME zeigt CURAD auf die Adresse mit dem Opcode der Instruktion, die von PME als letzte auf die linke Spalte des Bildschirms oder des Druckerpapiers gesetzt wurde.

Nach Eingeben der Startadresse 1533 (über PM!) und Drücken von Taste R folgt von PME die Rückmeldung:

PM EDITOR
XXXX YY YY YY

XXXX steht hier für den Inhalt von CURAD und YY YY YY für die Instruktion (oder das Label), deren Opcode (oder FF) sich in CURAD befindet. Abhängig von der Länge der Instruktion werden zwei, vier oder sechs Zeichen gedruckt. Im Gegensatz zum Kaltstart (siehe Abschnitt B) wird in diesem Fall kein EOF-Zeichen "77" auf BEGAD gesetzt.

D. Lauwarmer Start

Startadresse: $1667

Diese Art des Editor-Starts ist neu; sie existiert beim Standard-Editor nicht. Lauwarm liegt bekanntlich in der Mitte zwischen Warm und Kalt. Ähnlich verhält es sich auch hier. Wie beim Kaltstart müssen zuerst BEGAD und ENDAD spezifiziert werden; daraufhin wird die erste Instruktion (deren Opcode in BEGAD steht) ausgedruckt. Mit dem Warmstart hat der lauwarme Start gemeinsam, daß in BEGAD kein EOF-Zeichen "77" geschrieben wird. Während beim Kaltstart der variable Endadreßpointer CEND zu Beginn auf die nach BEGAD folgende Adresse gerichtet ist, wird beim lauwarmen Start CEND gleich ENDAD gesetzt.

Der lauwarme Start von PME hat folgende Sequenz:

1667 SP R (SP = Zwischenraumtaste!)
BEGAD, ENDAD: 1CØØ, 1FFF CR
PM EDITOR
1CØØ 85

Als Beispiel haben wir für BEGAD 1CØØ und für ENDAD 1FFF gewählt. Ist dies nicht ein EPROM-Bereich? Das stimmt! Der lauwarme Start ist nämlich vor allem für die Betrachtung fertiger Programme gedacht, unabhängig davon, ob diese in einem RAM oder einem EPROM stehen. Insbesondere die Tastenfunktionen SP (Abschnitt G, Nummer ①), Z (Abschnitt G, Nummer ②), L (Abschnitt G, Nummer ⑧), P (Abschnitt G, Nummer ⑨), und S (Abschnitt G, Nummer ⑥ und ⑦) sind dafür geeignet.

Der Starteingang "Lauwarm" von PME hat also für die Analyse und die Dokumentation von Eigen- und Fremdprogrammen (einschließlich der Systemprogramme in den EPROMs) erhöhte Bedeutung.

E. Warmer CEND-Start

Startadresse: $17C5

In Kapitel 11 aus Buch 3 haben wir erfahren, wann man Anwenderprogramme, die noch Label enthalten, zweckmäßigerweise auf Band setzt. Das ist mit Hilfe von TM (Taste SEF) oder mit PM möglich. Im zweiten Fall

muß außer einer Programmnummer ID auch eine Startadresse SA eingegeben werden, die gleich BEGAD ist, sowie ferner eine Endadresse, die mit CEND übereinstimmt. CEND ist nämlich gleich der Adresse, die um eine höher als die Adresse des "Schlußlichts" 77 liegt.
Wenn wir ein auf Band stehendes Programm mit Hilfe von PME bearbeiten wollen, müssen wir dieses Programm zuerst vom Band lesen. Dies geschieht mit PM, Taste G. Anschließend wird PME warm gestartet. Das bedeutet, daß die Inhalte von BEGAD, ENDAD und CEND mit den Werten des vom Band gelesenen Programms übereinstimmen müssen. Ferner muß CURAD einen sinnvollen Wert haben. In Kapitel 11 ist beschrieben, wie hier vorzugehen ist: Es sind verschiedene Adressen zu notieren usw.; alles in allem ist das Verfahren verhältnismäßig umständlich.
Machen wir vom warmen CEND-Start des Systemprogramms PME Gebrauch, dann gehört auch dies größtenteils der Vergangenheit an. Natürlich müssen Programmnummer ID und Beginnadresse BEGAD auch weiterhin bekannt sein. ENDAD kann man jedoch neu definieren, zum Beispiel höher wählen als ursprünglich, weil ja wahrscheinlich Instruktionen und Label hinzukommen. Letzteres ist auch notwendig, wenn man mit ID=FF mehrere editierte Programme einliest, die nahtlos aneinander anschließen sollen (zwischenliegende EOF-Zeichen "77" werden entfernt).
Was passiert nach dem warmen CEND-Start von PME? Es geschieht folgendes: Sind BEGAD und ENDAD eingegeben (das läuft ebenso wie beim kalten und lauwarmen Start ab), dann sucht PME den Speicherplatz, der den Pseudo-Opcode 77 enthält. Sobald dieser gefunden ist, wird seine Adresse um eins erhöht und CEND gleich dieser um eins erhöhten Adresse gesetzt. Es folgt die Rückmeldung:

PM EDITOR
XXXX 77

CEND ist jetzt auf Adresse (XXXX + 1) gerichtet.

Der PME-Eingang für den warmen CEND-Start kann in jedem Fall für die Neuedition von Programmen mit Labeln verwendet werden, die mit SA=BEGAD und EA=CEND auf Band geschrieben wurden. Hierfür ist dieser Starteingang in erster Linie bestimmt. Es gibt aber noch weitere Möglichkeiten. Sie nutzen die Tatsache, daß das EOF-Zeichen "77" nach dem Assemblieren eines Programms nicht verlorengegangen ist. Auch wenn die Label entfernt sind, bildet die "77" das Schlußlicht. Es steht dann auf dem Speicherplatz, der sich unmittelbar der letzten Instruktion des Programms anschließt. Wenn wir aus irgendeinem Grund ein bereits assembliertes Programm später erneut editieren wollen (entfernte Label können wieder hinzugefügt werden!), so kann dies über den warmen CEND-Start von PME geschehen. Voraussetzung ist nur, daß bei der Speicherung auf Band die abschließende "77" nicht verlorenging. Verhindern läßt sich das ganz einfach dadurch, daß man Endadresse EA um eine höher als die Endadresse wählt, die zur letzten Instruktion des Programms gehört. Tastenfunktion SEF von TM sorgt dafür automatisch.
Für den warmen CEND-Start von PME ist das Vorhandensein der "77" unbedingt erforderlich. Fehlt das EOF-Zeichen, so folgt auch keine Rückmeldung "PM EDITOR"; PME gerät dann in eine Sackgasse. In diesem Fall muß die RST-Taste gedrückt, PM neu gestartet und anschließend ein anderer Starteingang von PME gewählt werden. Welcher Starteingang dafür

infrage kommt, zeigt Praxisbeispiel 5, das später in diesem Kapitel folgt.

F. Die BRK-Taste

Die Funktion der BRK-Taste kennen wir bereits vom Systemprogramm PM. Mit ihr konnte dort das Ausdrucken von Texten abgebrochen werden. Auch bei PME hat die BRK-Taste die Funktion einer Notbremse. Während bei PM die Tastenfunktion M (Hexdump) Anlaß zur Betätigung der BRK-Taste geben kann, bietet bei PME hauptsächlich die Tastenfunktion L (siehe Abschnitt G, Nummer ⑧) einen Grund dafür. Nach Drücken von Taste L innerhalb von PME wird das Anwenderprogramm auf dem Bildschirm mit erhöhter Geschwindigkeit durchlaufen. Ist ungefähr die Stelle erreicht, die genauer betrachtet werden soll, dann drückt man die BRK-Taste und danach ein mal oder auch wiederholt die P-Taste (siehe Abschnitt G, Nummer ⑨).
Nach Betätigen der BRK-Taste innerhalb von PME folgt die Rückmeldung:
PM EDITOR
Anmerkung: Wenn PME über den kalten, lauwarmen oder den warmen CEND-Eingang gestartet wird, ist der BRK-Sprungvektor zunächst noch durch PM spezifiziert. Unterbricht man die Meldung "BEGAD, ENDAD:" durch Drücken der BRK-Taste, dann folgt die Rückmeldung "JUNIOR".

G. Die Tastenfunktionen von PME

① Taste SP (Zwischenraum) (Skip)

Instruktions-Plustaste

Dies ist eine vom Standard-Editor her wohlbekannte Tastenfunktion. Wir gehen davon aus, daß PME auf die linke Hälfte des TV-Schirms oder des Druckerpapiers eine Instruktion geschrieben hat. Bei PME erscheint gleichzeitig die Adresse, die den Opcode der Instruktion enthält. Der Schreibpositionszeiger des Elekterminals (oder die Wagenposition des Druckers) weist auf die erste Position der rechten Spalte. Die Zeile ist dabei die gleiche, auf der die Instruktion einschließlich der Opcode-Adresse steht. Drücken wir jetzt die Leertaste (Zwischenraumtaste) SP, dann folgt in der rechten Spalte keine Reaktion. Das liegt in diesem Fall daran, daß die Leertaste gedrückt wurde; normalerweise werden, wie bereits besprochen, hier die Tastenaktionen sichtbar.
Das Drücken der Zwischenraumtaste SP hat eine andere Wirkung: Auf die nächste Zeile wird in der linken Spalte die nachfolgende Instruktion des Programms gedruckt, oder es erscheint FF XX ØØ, falls es sich um ein Label handelt. An der Adresse der neuen Instruktion bzw. des Labels ist die Länge der vorangegangenen Instruktion erkennbar. Die Adresse liegt um eine, zwei oder drei Positionen höher als die Adresse der letzten Instruktion.
Wenn sämtliche Instruktionen des Programms durchlaufen sind, oder anders ausgedrückt, wenn durch fortlaufendes Drücken der Leertaste schließlich die "77" erreicht ist, wird nach einem weiteren Druck auf Taste SP folgendes geschrieben:
XXXX 77 SP
DONE
YYYY ZZ ZZ ZZ

Vor der nächsten Instruktion steht die Rückmeldung "DONE" als Zeichen dafür, daß der variable Endadreßzeiger CEND passiert wurde. Die Adresse YYYY liegt um eine höher als die Adresse XXXX, da das Schlußlicht "77" einen Speicherplatz füllt. "ZZ ZZ ZZ" ist höchstwahrscheinlich eine willkürliche Ziffernfolge. Was hier steht, hängt von der Vorgeschichte ab: Während des laufenden Computerbetriebs kann der Editor ein zweites Mal kalt gestartet worden sein. In jedem Fall gehört "ZZ ZZ ZZ" nicht zu dem Programm, mit dem man gerade beschäftigt ist. Die Anzahl der "Z" hängt übrigens von der Länge der echten oder unechten Instruktion ab. Aus der Besprechnung der Subroutine OPLEN/LENACC (Buch 2, Kapitel 8) wissen wir, daß jede Kombination XX von zwei Datennibbles vom Prozessor als Opcode interpretiert wird.
Noch eine Anmerkung: Wenn man nach Skippen der "77" mehr als einmal die Leertaste drückt, wird nach der Meldung "DONE" jedesmal die folgende echte oder unechte Instruktion gedruckt. Das Systemprogramm PME sorgt nicht für eine Blockade ("bis hierher und nicht weiter"), sondern weist mit der Meldung "DONE" nur darauf hin, daß die Fortsetzung des Skippens nicht zweckmäßig ist.

② Taste Z (Back)

Instruktions-Minustaste

Neu!

Eine Verbesserung, durch die sich PM vom Standard-Monitor unterscheidet, ist die zur Plustaste zusätzlich vorhandene Minustaste. Bei PME kommt zur Skiptaste (Instruktions-Plustaste) des Standard-Monitors die Instruktions-Minustaste Z hinzu.
Die Wirkung von Z wird an folgendem Beispiel deutlich:

```
Ø2AC FF 17 ØØ   Z   (Label 17)
Ø2AB CA         Z   (DEX)
Ø2A9 A9 FF      Z   (LDA # FF)
Ø2A6 2Ø 14 ØØ       (JSR-Label 14)
        usw.
```

Wenn man durch wiederholtes Drücken von Z schließlich die Beginnadresse BEGAD erreicht, wird bei jeder weiteren Betätigung von Z die erste Instruktion oder das erste Label noch einmal ausgedruckt. BEGAD wirkt sich hier als Blockierung in Richtung niedrigerer Adressen aus.

③ Taste K (Kill; Delete)

Instruktionen oder Label entfernen

Dies ist wieder eine bereits bekannte Tastenfunktion. Drücken von K hat zur Folge, daß die als letzte ausgedruckte Instruktion aus dem Speicher gelöscht wird; sie wird von der folgenden Instruktion überschrieben. Das Programm verkürzt sich dadurch, abhängig von der Länge der überschriebenen Instruktion, um ein, zwei oder drei Byte. Ausgedruckt wird die Instruktion, die im Speicher der entfernten Instruktion folgt. Wurde die letzte Instruktion gelöscht, dann ist dies die "77", die unter Adresse CEND-1 im Speicher steht.

Ein Beispiel:

```
Ø2A6 2Ø 14 ØØ    SP
Ø2A9 A9 FF       K
Ø2A9 CA          SP
Ø2AA FF 17 ØØ    SP
Ø2AD 77          SP
DONE
Ø2AE XX XX XX
```

Wir sehen, daß die Instruktionen CA und FF 17 ØØ um zwei Speicherplätze in Richtung niedrigerer Adressen gerückt sind.

④ Taste T (Top of File)

Zurück zum Anfang

Neu!

Tastenfunktion T ist zwar nicht revolutionär, aber doch recht nützlich: Sie sorgt dafür, daß die Instruktion oder das Label ausgedruckt wird, das auf der ersten Programmadresse (BEGAD) steht. Das ist unter anderem dann von Nutzen, wenn ein Programm mit einer der Tasten SP, L oder P durchlaufen werden soll.
Ein Beispiel:

```
Ø2AE XX XX XX    T
Ø2ØØ FF 1Ø ØØ
```

BEGAD ist hier die Adresse Ø2ØØ; das Programm beginnt dort mit dem Label 1Ø. Beim Standard-Editor konnte man BEGAD durch SEARCH FF 1Ø oder auch durch Skippen finden. PME bietet neben Tastenfunktion T noch die Alternative "S FF 1Ø ØØ" (siehe Abschnitt G, Nummer ⑥ und ⑦), doch damit ist mehr Tipparbeit verbunden. Ihren Umfang wollen wir aber, das hatten wir uns zum Ziel gesetzt, so gering wie möglich halten.

⑤ Taste I (Insert)

Instruktionen oder Label einfügen

Diese Tastenfunktion ist nicht neu; sie steht auch auf der Liste der Tastenfunktionen, die zum Standard-Editor gehört. Drückt man Taste I und gibt dann die einzufügende Instruktion ein, so wird diese Instruktion im Speicher unmittelbar vor die zuletzt gedruckte Instruktion gesetzt. Das geschieht jedoch erst, nachdem anschließend noch so oft eine numerische Taste gedrückt wurde, wie es der Länge der einzufügenden Instruktion entspricht. Die zuletzt gedruckte Instruktion und alle nachfolgenden Instruktionen werden daraufhin um das benötigte Stück in Richtung höherer Adressen verschoben.
Wenn nach Drücken von Taste I und vor der vollständigen Eingabe der Instruktion (zwei, vier oder sechs numerische Tasten) eine nicht numerische Taste gedrückt wird, folgt die Fehlermeldung:

```
ILLEGAL KEY
```

PME druckt dann noch einmal die Instruktion, die wir "letzte Instruktion" nannten. Der Fehler kann versehentlich passieren, man kann aber auch falsch eingegebene Daten auf diese Weise noch rechtzeitig korrigieren: Die

Eingabe läßt sich nach der Rückmeldung "ILLEGAL KEY" wiederholen. Eingeleitet werden muß die Wiederholung durch erneutes Drücken von Taste I.
Ein Beispiel dient wieder zur Verdeutlichung: Ausgangspunkt ist die Situation, die unmittelbar nach dem Entfernen der Instruktion A9 FF (siehe Abschnitt G, Nummer ③) bestand. In Speicherplatz Ø2A9 steht nun die Instruktion CA. Wir wollen die frühere Instruktion A9 FF durch A9 ØØ ersetzen. Dazu drücken wir die Tasten I, A, 9, Ø und Ø:

Ø2A9 CA IA9ØØ
Ø2A9 A9 ØØ SP
Ø2AB CA SP
Ø2AC FF 17 ØØ

Wie man sieht, sind sämtliche auf die Instruktion A9 ØØ folgenden Instruktionen um zwei Plätze in Richtung höherer Adressen aufgerückt. Übrigens: Ist Ihnen aufgefallen, daß PME zwischen die einzelnen Bytes einer Instruktion beim Ausdrucken einen Zwischenraum setzt, und zwar sowohl auf der linken als auch auf der rechten Bildschirmhälfte? Der Bildschirminhalt wird dadurch noch übersichtlicher; die einzelnen Instruktionen sind leichter lesbar.
Zusammen mit der Tastenfunktion INPUT (siehe Abschnitt G, Nummer ⑩) dient INSERT zur Eingabe von Instruktionen und Labeln. Treten INSERT oder INPUT in Aktion, so wächst der Umfang des Bereichs, den das Programm im Speicher einnimmt. Dieser Bereich beginnt bei BEGAD und reicht bis zu der Adresse, unter der das Schlußlicht "77" steht (CEND).
Der für das Programm (einschließlich der Label!) reservierte Speicherbereich hat jedoch keinen unendlich großen Umfang, denn wir haben ihn durch ENDAD begrenzt. Das geschah andererseits nicht ohne Grund. Während ENDAD beim Standard-Editor keine Rolle spielt, liegen die Dinge beim Assembler und bei PME anders. Hier erscheint eine Fehlermeldung auf dem Bildschirm, sobald die Programmlänge den gewählten Bereich überschreitet.

Wie schon angedeutet (nähere Einzelheiten folgen noch), benutzen wir den Assembler aus den Kapiteln 5 und 9 (Buch 2) auch für Programme, die wir mit Hilfe von PME eingegeben haben. Während der ersten Assemblierphase werden alle gefundenen Label (Opcode FF) in dem dafür reservierten Speicherbereich "notiert" und anschließend aus dem Programm entfernt. Das Programm verkürzt sich dadurch um jeweils drei Byte. Bevor das erste Label entfernt werden kann, muß es erst "notiert" werden. Wenn für dieses Label Platz vorhanden ist, dann finden automatisch auch alle anderen Label des Programms einen Platz.
Das erste Label müssen wir daher auf Speicherplatz ENDAD und den beiden darunterliegenden Speicherplätzen unterbringen können ("darunter" hinsichtlich der Adresse). Diese Speicherplätze dürfen deshalb nicht mit Instruktionen oder Labeln des zu editierenden Programms und auch nicht mit dem EOF-Zeichen "77" belegt werden. Daraus folgt, daß CEND wie in Bild 1 dargestellt die niedrigste Position einnimmt, die möglich ist; CEND zeigt dann auf die höchstmögliche Adresse.
Jedesmal wenn mit INSERT oder INPUT eine Instruktion oder ein Label eingegeben wird, vergleicht PME den neuen Inhalt von CEND mit dem

1

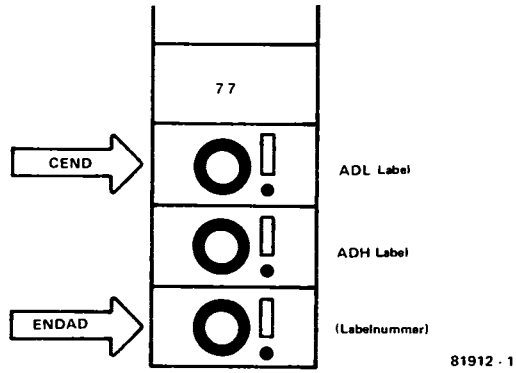


Bild 1. Die "untersten" vier Speicherplätze (mit den vier höchsten Adressen) des durch BEGAD und ENDAD festgelegten Speicherbereichs können nicht mit dem zu editierenden Programm belegt werden. Wenn der Inhalt von CEND kleiner ist als der Inhalt von ENDAD minus zwei, liefert PME die Fehlermeldung "FULL". Das Verbotsschild vor den untersten drei Speicherplätzen gilt nur für die Dauer der Editierung. Nach der ersten Assemblierphase wird das zuerst gefundene Label auf diese Speicherplätze gesetzt (in der Reihenfolge Labelnummer, ADH, ADL).

Inhalt von CEND aus Bild 1. Sind beide Inhalte gleich, dann folgt die Rückmeldung "FULL". Die zuletzt eingegebene Instruktion wird nicht mehr in den Speicher gesetzt, denn für sie ist bereits kein Platz mehr vorhanden. Trotzdem wird nach der Rückmeldung "FULL" noch eine Instruktion ausgedruckt. Bei INSERT ist dies die Instruktion, die als letzte vor der neuen, nicht mehr akzeptierten Instruktion eingegeben wurde. Mit anderen Worten: Der Inhalt des Displaypointers CURAD ist unverändert geblieben. Hat man dagegen die nicht mehr akzeptierte Instruktion über INPUT eingegeben, so erhöht sich der Inhalt von CURAD entsprechend der Länge der zuletzt ausgedruckten Instruktion. Zum Unterschied zwischen INPUT und INSERT siehe auch Tabelle 2.
Anmerkung: Aus Bild 1 kann man entnehmen, daß der letzte für das zu editierende Programm verfügbare Speicherplatz die Adresse ENDAD-4 hat. Es kann vorkommen, daß notgedrungen mehr Speicherplätze unbenutzt bleiben als die vier Speicherplätze in Bild 1 (von der "77" bis einschließlich ENDAD). Zum Beispiel in dem Fall, daß Speicherplatz ENDAD-4 zwar noch frei ist (alle Plätze von BEGAD bis einschließlich ENDAD-3 sind dann bereits mit Instruktionen oder Labeln besetzt), aber nachfolgend noch eine Zwei- oder Drei-Byte-Instruktion eingegeben wird.
Noch eine Anmerkung: Auch wenn eine eingegebene Instruktion nicht mehr akzeptiert wurde und sich der Inhalt des variablen Endadreßpointers CEND entsprechend erhöht hat, bleibt das EOF-Zeichen "77" an seinem Platz!

⑥ Taste S (Search)

⑦ Taste Y (Yes)

Die Tastenfunktion SEARCH kennen wir bereits vom Standard-Editor.

Dort mußten zwei Byte eingegeben werden; es waren somit vier numerische Tasten zu drücken. Bei den beiden Byte konnte es sich entweder um eine Zwei-Byte-Instruktion, um ein Label oder um die ersten beiden Byte einer Drei-Byte-Instruktion handeln. Die Suche nach der Zwei-Byte-Konfiguration stoppte, sobald dieses Muster zum ersten Mal auftrat; der Standard-Editor ließ nicht erkennen, ob die gleiche Konfiguration im fraglichen Bereich mehrfach vorhanden war. Ferner konnten Ein-Byte-Instruktionen überhaupt nicht aufgespürt werden.
PME kennt solche Einschränkungen nicht. Hier hat SEARCH zwei wichtige Eigenschaften, die dem Standard-Editor fehlen.
Die erste: Es wird nicht nach einer Zwei-Byte-Konfiguration, sondern nach einer Instruktion gesucht, die wahlweise ein, zwei oder drei Byte lang sein kann.
Die zweite: Wenn man feststellen will, ob eine gesuchte (und gefundene) Instruktion wiederholt in dem durch BEGAD und CEND begrenzten Speicherbereich vorkommt, so bietet PME hierzu die Möglichkeit. Es läßt sich somit nicht nur jede beliebige Instruktion und jedes Label aufspüren; feststellbar sind auch die Adressen der Speicherplätze, in denen die gleiche Instruktion steht (jedes Label ist hoffentlich nur ein mal vorhanden!).
Wie gehen wir in der Praxis vor?

1. Als erstes wird Taste S gedrückt; das "Echo" des Tastendrucks erscheint auf der rechten Bildschirmhälfte.
2. Anschließend geben wir die gesuchte Instruktion (oder das gesuchte Label) ein, indem wir abhängig von seiner Länge zwei, vier oder sechs numerische Tasten drücken. Erst nachdem die Eingabe vollständig ist, geht PME auf die Suche. Ausgedruckt wird die Instruktion (oder das Label) bereits während der Eingabe. Die Instruktion erscheint auf der rechten Seite neben dem S, wobei PME selbsttätig Leerräume zwischen die Bytes einfügt.

Drückt man nach der S-Taste versehentlich oder mit Absicht eine nicht numerische Taste, dann reagiert PME mit der Rückmeldung
ILLEGAL KEY
Um SEARCH nach dieser Rückmeldung fortsetzen zu können, muß Taste S noch einmal gedrückt werden. Man kann absichtlich eine nicht numerische Taste drücken, wenn bei der Eingabe der gesuchten Instruktion (bzw. des gesuchten Labels) ein Fehler unterlaufen ist. Auf jeden Fall werden alle eingegebenen Bytes mit eingefügten Zwischenräumen ausgedruckt.

3. Sobald die Instruktion (oder das Label) vollständig eingegeben ist, wird sie von PME mit der niedrigsten Adresse (BEGAD) beginnend gesucht. Nun gibt es zwei Möglichkeiten: Entweder steht die gesuchte Instruktion in dem von BEGAD und CEND begrenzten Speicherbereich, oder sie kommt dort nicht vor. Im ersten Fall druckt PME auf der linken Seite der nächsten Zeile die gesuchte Instruktion zusammen mit der Adresse aus. Im zweiten Fall folgt die Rückmeldung

DONE
XXXX ZZ ZZ ZZ
Dabei ist XXXX die auf CEND folgende Adresse und ZZ ZZ ZZ die Instruktion, deren Opcode in XXXX steht (diese Instruktion kann natürlich auch aus weniger als drei Byte bestehen).

4. Die folgenden Tastenaktionen sind nur sinnvoll, wenn die Suche erfolgreich war. Dann stehen zwei verschiedene Wege offen:

4a. Der vorgegebene Speicherbereich soll nach weiteren, gleichartigen Instruktionen abgesucht werden. Hierzu wird Taste Y gedrückt der Buchstabe Y erscheint auf der rechten Bildschirmhälfte. PME druckt daraufhin die gesuchte Instruktion, sofern sie noch mindestens ein weiteres Mal vorhanden ist, zusammen mit der zugehörigen Adresse auf der linken Bildschirmhälfte aus. Diese Adresse ist selbstverständlich nicht identisch mit der Adresse der zuvor gefundenen Instruktion; sie muß zwangsläufig höher liegen.
Wurde dagegen keine weitere Instruktion des gesuchten Typs gefunden, dann folgt die Rückmeldung
DONE
XXXX ZZ ZZ ZZ
(siehe Punkt 3)

4b. Der vorgegebene Speicherbereich soll nicht nach weiteren Instruktionen dieses Typs abgesucht werden. Dann drückt man eine beliebige Taste der ASCII-Tastatur, die bei ihrer Betätigung einen ASCII-Kode erzeugt, natürlich mit Ausnahme der Taste Y. Es folgt die Rückmeldung
DONE
XXXX ZZ ZZ ZZ
ZZ ZZ ZZ ist die vorher gefundene Instruktion (Punkt 3) und XXXX die zugehörige Adresse.
Zur Praxis der "Instruktionssuche" noch ein paar wichtige Anmerkungen:

a. Man kann die Suche in jeder beliebigen Phase abbrechen. Die nach der Rückmeldung "DONE" mit der zugehörigen Adresse ausgedruckte Instruktion zeigt gleichzeitig an, ob nicht weiter gesucht wurde oder ob die weitere Suche erfolglos war.

b. Die Suchaktion (lies: Search-Tastenroutine) ist erst dann vollständig beendet, wenn die Rückmeldung "DONE" gefolgt von einer Instruktion plus Adresse erscheint. Dabei ist gleichgültig, ob "DONE" infolge einer ergebnislosen Suche oder durch Abbrechen des Suchvorgangs (Drücken einer anderen Taste als Y) entstand.

c. Die Y-Taste hat nur dann die beschriebene Funktion, wenn vorher S gedrückt und danach die zu suchende Instruktion eingegeben wurde. Drückt man Taste Y ohne vorher die Funktion S aufgerufen zu haben oder aber nach unvollständiger Eingabe der Instruktion, dann folgt die Rückmeldung "ILLEGAL KEY".

d. Wird die Suche wie beschrieben durch Drücken einer beliebigen Software-Taste (mit Ausnahme von Y) abgebrochen, so folgt keine Rückmeldung "ILLEGAL KEY". Die normalerweise bei PME vorhandenen Tastenfunktionen – auch S, jedoch mit Ausnahme von Y – können während der Suchaktion (also nach Drücken von Taste S und vor einer Rückmeldung "DONE") nicht aktiviert werden. Andere Tasten, die im normalen Modus von PME ohne Funktion sind und deren Betätigung zur Rückmeldung "ILLEGAL KEY" führt, haben dagegen zeitweise eine Funktion: Dann, wenn Taste S gedrückt wurde und die anschließend eingegebene Instruktion mindestens ein mal gefunden ist. Einzelheiten kommen bei der Beschreibung der PME-Software in Kapitel 15 zur Sprache.

Tabelle 1. Beispiel für das Arbeiten mit dem Warmstart-Eingang von PME sowie für die Anwendung verschiedener PME-Tastenfunktionen.

```
JUNIOR
1667
1667 20 R
BEGAD,ENDAD: 1C00,1FFF
PM EDITOR
1C00 85 F3          S85 FA
1C08 85 FA          Y
1C83 85 FA          Y
1CB0 85 FA          Y
1CDD 85 FA          Y
1D3D 85 FA          Y
1E37 85 FA          Y
1FDA 85 FA          Y
1FEA 85 FA          Y
DONE
2000 0A             SC8
1CBF C8             Y
1CEA C8             Y
1D55 C8             Y
1E0E C8             Y
1E56 C8             Y
1F70 C8             Y
1FB6 C8             Y
1FBF C8             Y
1FC4 C8             Y
DONE
2000 0A             T
1C00 85 F3          P
1C02 68
1C03 85 F1
1C05 68
1C06 85 EF
1C08 85 FA
1C0A 68
1C0B 85 F0
1C0D 85 FB
1C0F 84 F4
1C11 86 F5
1C13 BA
1C14 86 F2
1C16 A2 01
1C18 86 FF
1C1A 4C 33 1C       S8D 83 1A
1C1F 8D 83 1A       Y
DONE
2000 0A             S8E 83 1A
DONE
2000 0A             S8C 83 1A
DONE
2000 0A
```

Höchste Zeit für ein praktisches Beispiel:
Wir betrachten dazu das Listing in Tabelle 1. Dem Listing läßt sich entnehmen, daß wir PME "lauwarm" gestartet haben. Die Adreßpointer BEGAD und ENDAD haben wir so gewählt, das der Adressenbereich des Standard-EPROMs auf der Junior-Computer-Basisplatine zwischen ihnen liegt. Nach Drücken von CR (in Tabelle 1 nicht ausgedruckt) erscheint die Rückmeldung "PM EDITOR" gefolgt von der ersten Instruktion mit der zu ihr gehörenden Adresse: "1CØØ 85 F3".
Wir wollen prüfen, wie oft die Instruktion STAZ-POINTL innerhalb des Standard-Monitorprogramms vorkommt und unter welchen Adressen sie dort steht. Dazu geben wir ein: S85FA. Den Leerraum zwischen der "85" und "FA" fügt PME selbsttätig ein. Er muß bei der Eingabe weggelassen werden, da sonst die Rückmeldung "ILLEGAL KEY" erscheint. Danach müßte die Suchaktion von neuem gestartet werden.
Schon vor Beginn der Suchaktion wußten wir, daß die gesuchte Instruktion im spezifizierten Adressenbereich mindestens ein mal vorhanden sein muß; Speicherplatz ØØFA ist nämlich der Displaybuffer POINTL. Die Antwort von PME läßt nicht lange auf sich warten: Zum ersten Mal begegnen wir der gesuchten Instruktion bei Adresse 1CØ8. Die folgenden acht Betätigungen der Taste Y liefern sieben weitere Adressen, unter denen diese Instruktion zu finden ist. Schließlich erscheint die Rückmeldung "DONE", gefolgt von Adresse 2ØØØ mit zugehörigem Inhalt. Letzteres ist nicht verwunderlich, denn beim "lauwarmen" Start von PME wird CEND gleich ENDAD gesetzt. Als Endadresse ENDAD hatten wir aber 1FFF gewählt.
In Tabelle 1 ist protokolliert, daß wir nun auf die Suche nach der Instruktion C8 (INY) gehen. Diese Instruktion kommt im Standard-Monitor neun mal vor. Die Richtigkeit der ausgedruckten Adressen läßt sich übrigens leicht an Hand des Listings in Buch 2, Seiten 214 . . . 222 kontrollieren.
Setzen wir das Studium von Tabelle 1 fort: Nach Drücken von Taste T wird die erste Adresse (1CØØ) zusammen mit der dort stehenden Instruktion ausgedruckt. Das ist völlig richtig. Über die Funktion und Wirkung von Taste P werden wir in Abschnitt G, Nummer ⑨ noch Näheres erfahren; das sei hier zunächst einmal zurückgestellt.
Der letzte Teil von Tabelle 1 demonstriert, wie sich Programme mit Hilfe von PME analysieren lassen: Wir wollen herausfinden, wann und in welcher Weise der Standard-Monitor das Richtungsregister PBDD beeinflußt. Die Adresse von PBDD ist $1A83. Infrage kommt dafür nur eine Store-Instruktion; drei Möglichkeiten sind gegeben:

8D 83 1A STA-$1A83
8E 83 1A STX-$1A83
8C 83 1A STY-$1A83

Zuerst klären wir, ob die Instruktion STA-$1A83 im Monitorprogramm vorhanden ist. Wie Tabelle 1 zeigt, kommt sie dort ein mal vor. Die Suche nach STX-$1A83 und STY-$1A83 verläuft dagegen ergebnislos. Schlußfolgerung: Die einzige Instruktion des Standard-Monitors, die das Richtungsregister PBDD beeinflußt, steht dort unter einer zur RESET-Startroutine gehörenden Adresse.

⑧ Taste L (List)

Ausdrucken von Programmen

ebenfalls neu!

Bereits zu PM gehört eine Reihe von Kommandos, die sich in die Gruppe der Anzeige- und Druckbefehle einreihen lassen. Ein Beispiel dafür ist das Kommando, mit dem das Ausdrucken eines Hexdump veranlaßt werden kann. Bei PME haben wir bisher nur Kommandos kennengelernt, die zur Anzeige oder zum Ausdrucken von einzelnen Instruktionen, eventuell zusammen mit einer Rückmeldung des Computers, führen. Von ihnen unterscheiden sich die Tastenfunktionen L und P (letztere siehe nachfolgend unter ⑨): sie bewirken das Ausdrucken einer Liste von Instruktionen. Ein Druck auf Taste L innerhalb von PME hat zur Folge, daß, beginnend mit der ersten Instruktion (oder dem ersten Label) in Speicherplatz BEGAD, sämtliche zwischen BEGAD und CEND stehenden Instruktionen und Label der Reihe nach ausgedruckt werden. Sie erscheinen untereinander auf der linken Hälfte des Bildschirms bzw. des Druckerpapiers. Wenn der gesamte Inhalt des spezifizierten Speicherbereichs auf dem Bildschirm oder dem Papier steht (der Displaypointer CURAD verschiebt sich nach jeder ausgedruckten Instruktion um einen Schritt in Richtung höherer Adressen), folgt die Rückmeldung "DONE". Danach wird noch die Instruktion ausgedruckt, die unter der Adresse steht, auf die CURAD nach Passieren von CEND zeigt.
Die Aktivierung der Tastenfunktion L bietet sich vor allem dann an, wenn man sich einen schnellen Überblick über das gesamte Programm verschaffen will. Soll ein bestimmter Programmteil näher betrachtet werden, dann kann man nach Erscheinen dieses Teils die BRK-Taste drücken (Rückmeldung: "PM EDITOR"). PME setzt das Auflisten fort, sobald Taste P gedrückt wird.

⑨ Taste P (Print)

Ausdrucken von Instruktionsblöcken

Diese Tastenfunktion ist mit der vorstehend besprochenen Funktion L eng verwandt. Ausgangspunkt ist hier die von PME zuletzt gedruckte Instruktion; sie steht in der linken Spalte. Durch welches Kommando diese Instruktion auf den Bildschirm bzw. auf Papier gesetzt wurde, spielt dabei keine Rolle. Wir drücken nun die Taste P. In der gleichen Zeile wie die zuletzt ausgedruckte Instruktion, jedoch auf der rechten Seite, erscheint daraufhin der Buchstabe P. Anschließend werden die folgenden 15 Instruktionen auf der linken Seite ausgedruckt, so daß zusammen mit der ersten Zeile (in der das P steht) insgesamt 16 Instruktionen sichtbar sind. Setzt man als Datensichtgerät das Elekterminal ein, so ist dies der Inhalt von genau einer Bildschirmseite.
Die Anzahl von 16 ausgedruckten Instruktionen trifft allerdings nur zu, solange PME nicht auf den Pseudo-Opcode "77" stößt. Tritt dieses EOF-Zeichen auf, dann wird das Ausdrucken des Instruktionsblocks abgebrochen. Die "Instruktion" 77 erscheint dabei als letzte auf dem Bildschirm bzw. dem Papier. Ein Beispiel für das Arbeiten mit Tastenfunktion P ist ebenfalls in Tabelle 1 enthalten.

⑩ INPUT

Tasten Ø . . . 9 und A . . . F

Es wurde bereits mehrfach erwähnt: Um die Funktion INPUT aufzurufen, ist das Drücken einer besonderen Taste nicht notwendig; die numerischen Daten können unmittelbar eingegeben werden. Anders verhielt es sich bei den Funktionen INSERT (Taste I) und SEARCH (Taste S), wo es ebenfalls um die Eingabe numerischer Daten ging. Dort mußte zuerst I bzw. S gedrückt werden.

Doch zurück zu INPUT. Wenn PME nach Ausführen einer Tastenfunktion auf neue Anweisungen wartet (das läßt sich am Stand des Cursors auf dem Bildschirm bzw. an der Schreibposition des Druckers erkennen; sie sind dann auf den Zeilenanfang der rechten Spalte gerichtet), und wenn während dieser Wartestellung numerische Tasten betätigt werden, so aktiviert PME selbsttätig die Funktion INPUT.

Was leistet INPUT? Diese Funktion sorgt dafür, daß eine einzugebende Instruktion oder ein Label nach vollständiger Eingabe unter einer bzw. mehreren aufeinanderfolgenden Adressen in den Speicher gesetzt wird, wobei sich die Adressen unmittelbar an die zuletzt (in der rechten Spalte) ausgedruckten Adresse anschließt.

Was die Praxis betrifft, so bestehen zwischen den Funktionen INPUT und INSERT sowohl einige Parallelen als auch Unterschiede. Auch bei INPUT wird die Instruktion oder das Label erst nach der vollständigen Eingabe im RAM gespeichert. Auch hier führt das Drücken einer nicht numerischen Taste zur Rückmeldung "ILLEGAL KEY". In diesem Fall muß die Eingabe von Anfang an wiederholt werden. Ebenso wie bei INSERT wird die Instruktion auch bei INPUT auf der folgenden Zeile der linken Spalte ausgedruckt, nachdem sie vollständig eingegeben ist. PME sorgt für die Leerräume zwischen den Bytes, sowohl in der linken als auch in der rechten Spalte. Es dürfen keine Leerräume von Hand gesetzt werden; das würde zur Rückmeldung "ILLEGAL KEY" führen und eine Wiederholung der Eingabe notwendig machen!

Da durch das Ausführen einer INPUT-Routine eine Instruktion oder ein Label zum bisherigen Speicherinhalt hinzukommt, verschieben sich die "77" sowie der Inhalt von CEND in Richtung höherer Adressen. Der Abstand zwischen der alten und der neuen Adresse hängt von der Länge der gerade eingegebenen Instruktion ab. Jede hinzukommende Instruktion (oder Label) vergrößert den von BEGAD und CEND abgesteckten Speicherbereich, so daß auch hier der verfügbare Bereich nach einiger Zeit überschritten werden kann. Bild 1 macht dies deutlich; man vergleiche die in Zusammenhang mit INSERT gegebenen Erläuterungen. Ist kein Platz mehr für die gerade eingegebene Instruktion vorhanden, dann erscheint die Fehlermeldung

FULL
XXXX ZZ ZZ ZZ

Beim Ausdrucken der Adresse (XXXX) und der zugehörigen Instruktion (ZZ ZZ ZZ) wird hier die Erhöhung von CURAD infolge der Eingabe der nicht mehr gespeicherten Instruktion berücksichtigt, während bei der Tastenfunktion INSERT die vor der Blockierung ausgedruckte Adresse mit zugehöriger Instruktion noch einmal erschien.

Tabelle 2. Ein nur aus Ein-Byte-Instruktionen bestehendes Phantasieprogramm, einmal mit Tastenfunktion INPUT (links) und einmal mit INSERT eingegeben (rechts). Unterschiedlich ist unter anderem die Reihenfolge, in der die Instruktionen nach der Eingabe im Speicher stehen.

```
JUNIOR

1500
1500 20 R
BEGAD,ENDAD: 200,20F
PM EDITOR
0200 77          TCA
0200 CA          E8
0201 E8          C8
0202 C8          EA
0203 EA          48
0204 48          08
0205 08          68
0206 68          28
0207 28          88
0208 88          0A
0209 0A          18
020A 18          D8
020B D8          58
FULL
020C 77          T
020D CA          P
020E E8
020F C8
0210 EA
0211 48
0212 08
0213 68
0214 28
0215 88
0216 0A
0217 18
0218 D8
0219 77
```

```
JUNIOR

1500
1500 20 R
BEGAD,ENDAD: 200,20F
PM EDITOR
0200 77          TCA
0200 CA          TE8
0200 E8          TC8
0200 C8          TEA
0200 EA          T48
0200 48          1U
ILLEGAL KEY
0200 48          T08
0200 08          T68
0200 68          T28
0200 28          T88
0200 88          T0A
0200 0A          T18
0200 18          TD8
0200 D8          T58
FULL
0200 D8          T
0200 D8          P
0201 18
0202 0A
0203 88
0204 28
0205 68
0206 08
0207 48
0208 EA
0209 C8
020A E8
020B CA
020C 77
```

Auch bei INPUT wird der variable Endadreßpointer CURAD entsprechend der Länge der überzähligen Instruktion erhöht, obwohl diese Instruktion nicht mehr in den Speicher gelangt. Ebenso wie bei INSERT bleibt das EOF-Zeichen "77" auch bei INPUT auf seinem Platz.
Wir wollen nun Tabelle 2 betrachten. Diese Tabelle enthält zwei Listings, die den Unterschied zwischen INSERT und INPUT deutlich zum Vorschein kommen lassen. In beiden Fällen spezifizieren wir mit BEGAD und ENDAD einen Speicherbereich, der 17 Plätze umfaßt (ENDAD wird mitgezählt!). Es geht um ein Phantasieprogramm, das nur aus Ein-Byte-

Instruktionen besteht. Das Programm soll einmal mit INPUT und ein zweites Mal mit INSERT in den Speicher gesetzt werden. Links in Tabelle 2 wurde, abgesehen vom obligatorischen INSERT am Anfang, von INPUT Gebrauch gemacht, während die rechte Hälfte dieser Tabelle die Eingabe des gleichen Programms mit INSERT wiedergibt.
Die letzte Instruktion ist in beiden Fällen unter Adresse Ø2ØB zu finden; für die Instruktion mit dem Opcode 58 (CLI) ist kein Platz mehr vorhanden. Die Regel, daß die Adresse des letzten verfügbaren Speicherplatzes gleich ENDAD minus vier ist, beweist auch hier ihre Gültigkeit: $ØF-$ØB ist gleich 15 – 11 = 4. In Speicherplatz Ø2CØ steht die "77" (das ergibt sich auch aus den in Tabelle 2 durch Funktion ausgedruckten Instruktionen), und die drei Speicherplätze Ø2ØD, Ø2ØE und Ø2ØF stehen für das "Notieren" des ersten Label während der ersten Assemblierphase zur Verfügung.
Tabelle 2 zeigt deutlich, daß die Reihenfolge, in der die Instruktionen gespeichert werden, bei INPUT und INSERT unterschiedlich ist. Hinzufügen und Einfügen unterscheiden sich durch eine umgekehrte Reihenfolge der Speicherung. Die unterschiedlichen Instruktionen, die jeweils auf die Rückmeldung "FULL" folgen, sind in Tabelle 2 ebenfalls sichtbar. Ferner sei noch angemerkt, daß während der Eingabe mit INSERT (rechts Hälfte von Tabelle 2) zur Demonstration einmal eine falsche Taste ("1" anstelle von "I") gedrückt wurde. Nach Drücken von Taste "U" erscheint "ILLEGAL KEY"; anschließend wird die Eingabe des Byte "Ø8" neu begonnen und diesmal erfolgreich zu Ende geführt.

⑪ Taste X (EXecute)

Assemblieren auf Tastendruck

Ebenso wie der Standard-Editor arbeitet auch das Systemprogramm PME mit hexadezimalen Labeln. Der Assembler, den wir bereits in Buch 2 kennengelernt haben, kann ohne irgendwelche Änderungen auch für solche Programme verwendet werden, die mit Hilfe von PME erstellt wurden. Bekanntlich entfernt der Assembler ein Label nach dem anderen aus dem Programm, nachdem er die Adresse und die Labelnummer in dem bei ENDAD beginnenden Speicherbereich abgelegt hat.
Der Assembler kann auch unverändert weiterverarbeitet werden, weil er zu den "stillen" Programmen gehört: Während seines Durchlaufs wird das Display nicht angesteuert; irgendwelche Tasten müssen nicht gedrückt werden. Erst nach dem Assemblieren leuchtet das Display wieder auf.
Die Startadresse des Assemblers lautete und lautet immer noch $1F51. Früher mußte nach dem Editieren die RST-Taste gedrückt werden, dann war diese Startadresse einzugeben, und schließlich wurde die GO-Taste gedrückt. Alternativ dazu konnte man den NMI-Sprungvektor auf die Startadresse des Assemblers richten und dann bei Bedarf die ST-Taste drücken. Mit PME ändert sich hier einiges: das verdanken wir der PME-Tastenfunktion X.
Drückt man Taste X, so springt das Programm nach ein paar vorbereitenden Aktionen zum Assembler. Eine der vorbereitenden Aktionen ist das Richten des NMI-Sprungvektors auf eine Adresse innerhalb von PME; sie ist das Sprungziel nach dem Durchlauf durch den Assembler. Einige Zeit nach Drücken von X (die Dauer hängt von der Länge des zu assemblieren-

den Programms ab) leuchtet das Siebensegmentdisplay wieder auf und zeigt damit an, daß das Assemblieren beendet ist. Wird nun Taste ST auf der Standard-Tastatur gedrückt, so führt der dadurch ausgelöste NMI zurück zu PME.
Anschließend geschieht etwas recht Bemerkenswertes: Sämtliche Label werden auf dem Bildschirm bzw. auf Papier ausgedruckt! Es erscheinen dort sowohl die Labelnummern als auch die zugehörigen Adressen. Das kann zum Beispiel so aussehen:

LAB $1Ø: $Ø2ØØ LAB $12: $Ø2Ø8 LAB $14: $Ø2ØC LAB $13: $Ø213
LAB $15: $Ø22B
PM EDITOR
XXXX ZZ ZZ ZZ

Das Ausdrucken der Label ist deshalb möglich, weil diese ja auch noch nach dem Assemblieren im Speicher stehen. Drei Speicherplätze sind für jedes Label reserviert; der Label-Speicherbereich beginnt bei ENDAD und erstreckt sich in Richtung absteigender Adressen.
Wie wir gesehen haben, werden maximal vier Label in jeder Zeile ausgedruckt. Die Reihenfolge, in der sie dort stehen, entspricht der Reihenfolge, in der sie während der ersten Assemblierphase gefunden, "notiert" und entfernt wurden. Das erste ausgedruckte Label trägt die niedrigste Adresse, die Adressen der folgenden Label liegen jeweils höher. Weshalb dies so und nicht anders geschieht, kann man in Kapitel 9 (Buch 2) nachlesen, das die Assembler-Software ausführlich erklärt. Ob die Reihenfolge der Labelnummern mit der Folge der natürlichen Zahlen (1, 2, 3, 4 usw.) übereinstimmt, hängt vom Programmierer ab. Man ist nicht an eine bestimmte Reihenfolge gebunden; eine willkürliche Zuordnung der Labelnummern ist allerdings der Übersichtlichkeit des Programms weniger dienlich. Im angeführten Beispiel treten der Reihe nach folgende Labelnummern auf: 1Ø, 12, 14, 13, 15. Die Labelnummer 11 fehlt hier, vielleicht weil das zu assemblierende Programm eine absolute Operandenadresse XX11, wahrscheinlich mit XX = ØØ, enthält.
Nachdem alle Label ausgedruckt sind (in diesem Sonderfall ist der Bildschirm bzw. das Papier nicht in eine rechte und eine linke Hälfte geteilt!), folgt auf der nächsten Zeile die Rückmeldung "PM EDITOR". Anschließend wird auf die übernächste Zeile die erste Instruktion des assemblierten Programms gesetzt; XXXX ist folglich die Adresse BEGAD. Den Schlußpunkt nach dem Assemblieren und Ausdrucken der Label bildet somit ein Sprung zum Warmstarteingang von PME.

Wir haben nun sämtliche Tastenfunktionen von PME kennengelernt. Blättern wir zurück, so stellen wir fest, daß PME auch in dieser Hinsicht einiges zu bieten hat. Im folgenden Abschnitt dieses Kapitels wird der Umgang mit PME in der Praxis geübt. Gleichzeitig setzen wir fort, was wir bereits in Kapitel 12 (Buch 3) begonnen haben: das Nutzbarmachen von PM-Subroutinen für eigene Programme.

PME in der Praxis

. . . denn grau ist alle Theorie!

1. 8-Bit-Hexadezimal-nach-Dezimal-Umwandlung

In Kapitel 5 aus Buch 2 (Seiten 9 . . . 26) wurde an einem praktischen Beispiel der Umgang mit dem Standard-Editor demonstriert. Als Beispiel diente ein Programm, (DISPL zusammen mit verschiedenen Subroutinen), das den dezimalen Wert einer aus acht Bit bestehenden hexadezimalen Zahl auf das Display schreibt. Um die hexadezimale Zahl einzugeben, mußten zwei numerische Tasten gedrückt werden. Das Programm selbst (lies: die Problemanalyse und das daraus folgende Umsetzen in ein Programm) wurde seinerzeit nicht erläutert; auch hier wollen wir darauf verzichten. Die zweite Hälfte dieses der Praxis gewidmeten Abschnitts beschäftigt sich nämlich noch ausführlich mit einer erweiterten Version des erwähnten Programms aus Buch 2. Zunächst wollen wir uns ausschließlich mit den Unterschieden und Besonderheiten beschäftigen, die PME im Vergleich zum Standard-Editor aufweist.

Wir betrachten dazu nebeneinander das Listing auf den Seiten 25 und 26 von Buch 2 sowie Tabelle 3. Letztere beginnt mit dem Start von PM (Rückmeldung "JUNIOR"). Anschließend wird PME kalt gestartet. Die erste "Instruktion" ist das Label 1Ø, das wir mit Hilfe der Insert-Taste eingeben. Das ist notwendig, weil anderenfalls das EOF-Zeichen "77" auf Speicherplatz ØØØØ stehenbleibt. Für den variablen Endadreßpointer ist zwar ohne Bedeutung, ob die erste Instruktion mit Input oder Insert in den Speicher gelangt; CEND verschiebt sich in jedem Fall in Richtung höherer Adressen. Programme mit dem Pseudo-Opcode "77" zu beginnen (ØØØØ ist gleichzeitig die Startadresse!) bedeutet jedoch eine unnötige Komplizierung.

Alle folgenden Instruktionen und Label werden mit Input in den Computer eingegeben: es werden ausschließlich numerische Tasten gedrückt. In Tabelle 3 stoßen wir dann nach einigen Zeilen auf ein K; die folgende Instruktion wird mit Hilfe von Taste I in den Speicher gesetzt. Der Grund ist die versehentliche Eingabe von "85 F7" anstelle von "85 D7".

Das Eintippen des gesamten Programms dauert nur wenige Minuten, wobei Tippfehler viel schneller erkannt und leichter beseitigt werden können als beim Standard-Editor. Dank des Elekterminals sind auf einem Bildschirm sechzehn Zeilen gleichzeitig sichtbar. Ist ein Drucker vorhanden, dann läßt sich auf einen Blick sogar das gesamte Programm übersehen, solange das Listing nicht mehrere Meter lang ist!

Als nächster Schritt folgt das Drücken von Taste X; das Programm wird nun assembliert. Hat der Computer diese Arbeit erledigt, so leuchtet das Display auf. Danach drücken wir Taste ST ("STOP", "NMI") auf der Standard-Tastatur, was zur Folge hat, daß PME sämtliche Label ausdruckt und sich abschließend mit "PM EDITOR" und der ersten Instruktion des assemblierten Programms zurückmeldet. Um ein vollständiges Listing des assemblierten Programms zu erhalten, wird noch drei mal Taste P gedrückt.

Mit ein wenig Geschicklichkeit und Übung ist Tabelle 3 innerhalb von fünf Minuten fertiggesestellt. Nur das Einlesen eines Programms gleicher Länge vom Band erfordert noch weniger Zeit . . .

Tabelle 3. Dieses Listing entstand bei der Eingabe des Programms DISPL und seinen Subroutinen. Siehe Praxisbeispiel 1 im Text sowie die Seiten 25 und 26 in Buch 2.

```
JUNIOR

1500
1500 20 R
BEGAD,ENDAD: 200,3FF
PM EDITOR
0200 77              IFF 10 00
0200 FF 10 00        A9 00
0203 A9 00           85 F9
0205 85 F9           85 FA
0207 85 FA           85 FB
0209 85 FB           FF 11 00
020B FF 11 00        20 6F 1D
020E 20 6F 1D        10 10
0211 10 10           85 F9
0213 85 F9           85 F7
0215 85 F7           K
0215 77              I85 D7
0215 85 D7           20 12 00
0217 20 12 00        4C 11 00
021A 4C 11 00        FF 12 00
021D FF 12 00        20 14 00
0220 20 14 00        85 FA
0223 85 FA           84 D7
0225 84 D7           20 14 00
0227 20 14 00        A2 04
022A A2 04           FF 13 00
022C FF 13 00        0A
022F 0A              CA
0230 CA              D0 13
0231 D0 13           05 FA
0233 05 FA           85 FA
0235 85 FA           84 FB
0237 84 FB           60
0239 60              FF 14 00
023A FF 14 00        A0 00
023D A0 00           84 D8
023F 84 D8           20 15 00
0241 20 15 00        18
0244 18              A5 D7
0245 A5 D7           69 0A
0247 69 0A           60
0249 60              FF 15 00
024A FF 15 00        38
024D 38              A5 D7
024E A5 D7           E9 0A
0250 E9 0A           85 D7
0252 85 D7           A5 D8
0254 A5 D8           E9 00
```

```
0256 E9 00          30 16
0258 30 16          C8
025A C8             4C 15 00
025B 4C 15 00       FF 16 00
025E FF 16 00       60
0261 60             X
LAB $10: $0200 LAB $11: $0208 LAB $12: $0217 LAB $13: $0223
LAB $14: $022E LAB $15: $023B LAB $16: $024C
PM EDITOR
0200 A9 00          P
0202 85 F9
0204 85 FA
0206 85 FB
0208 20 6F 1D
020B 10 F3
020D 85 F9
020F 85 D7
0211 20 17 02
0214 4C 08 02
0217 20 2E 02
021A 85 FA
021C 84 D7
021E 20 2E 02
0221 A2 04
0223 0A             P
0224 CA
0225 D0 FC
0227 05 FA
0229 85 FA
022B 84 FB
022D 60
022E A0 00
0230 84 D8
0232 20 3B 02
0235 18
0236 A5 D7
0238 69 0A
023A 60
023B 38
023C A5 D7          P
023E E9 0A
0240 85 D7
0242 A5 D8
0244 E9 00
0246 30 04
0248 C8
0249 4C 3B 02
024C 60
024D 77
```

2. 16-Bit-Hexadezimal-nach-Dezimal-Wandlung

Es soll ein Programm erstellt werden, das eine hexadezimale, 16 Bit umfassende Zahl in die dezimale Form umsetzt und diese äquivalente dezimale Zahl ausdruckt. Jede umzuwandelnde hexadezimale Zahl soll zusammen mit ihrem dezimalen Äquivalent am Anfang einer neuen Zeile ausgedruckt werden. Die Vollständigkeit der Eingabe (sämtliche Ziffern der Hex-Zahl) sind eingegeben) soll dem Junior-Computer durch Drücken der Taste ":" (Doppelpunkt) signalisiert werden. Der dezimale Wert soll auf der gleichen Zeile hinter dem Doppelpunkt erscheinen. Wir fordern ferner, daß auch hexadezimale Zahlen, die 4, 8 oder 12 Bit lang sind (eine, zwei oder drei numerische Tasten) vom Programm verarbeitet werden. Drückt man (versehentlich oder absichtlich) mehr als vier numerische Tasten, dann sollen die letzten vier Ziffern Gültigkeit haben.
Das ist in Kurzform die Aufgabe, die die Software lösen soll. Wie kann diese Software aussehen? Um die Frage zu beantworten, betrachten wir zuerst den Aufbau einer beliebigen dezimalen Zahl. In der Zahl 1981 steckt 1×10^3, also ein Tausender. Zieht man von dieser Zahl 2×10^3 ab, so ist das Ergebnis eine negative Zahl. Ferner stecken in 1981 neun Hunderter (9×10^2), acht Zehner (8×10^1) und ein Einer (1×10^0).
Die größte durch 16 Bit darstellbare, hexadezimale Zahl ist $FFFF; sie ist äquivalent mit der dezimalen Zahl 65535. Anders betrachtet: Die höchste Zehner-Potenz, die im dezimalen Äquivalent einer 16 Bit umfassenden, hexadezimalen Zahl vorkommen kann, ist 10^4. Wir stellen weiterhin fest, daß

$10\,000_{10}$ mit $271Ø,
$1\,000_{10}$ mit $Ø3E8,
100_{10} mit $ØØ64 und
10_{10} mit $ØØØA äquivalent ist.

Nach vorstehender Überlegung bietet sich für die Umwandlung der hexadezimalen Zahl in ihr dezimales Äquivalent folgendes Verfahren an: Von der eingegebenen hexadezimalen Zahl ziehen wir so oft die Zahl $271Ø (dezimal: 10 000) ab, bis das Ergebnis eine negative Zahl ist. Zur negativen Zahl addieren wir wieder $271Ø. Die Anzahl der Subtraktionen, die möglich waren, ohne daß das Ergebnis negativ wurde, wird in einem Register festgehalten und ausgedruckt. Es leuchtet ein, daß nach Beendigung des beschriebenen Algorithmus (sich wiederholende, gleichartige Rechenprozedur) keine Zehntausender mehr in der übrig gebliebenen Zahl stecken.
Nun wird der gleiche Algorithmus auf die übrig gebliebene Zahl mit Ø3E8 als Subtrahend angewandt: von der Restzahl ziehen wir so oft $Ø3E8 (dezimal: 1 000) ab, bis das Ergebnis negativ ist. Die neue Restzahl (sie kann natürlich auch Null sein) erhalten wir durch anschließende Addition von Ø3E8. Auch die Hunderter und die Zehner des gesuchten dezimalen Äquivalents werden nach der gleichen Methode ermittelt. Übrig bleibt ein einstelliger Rest, der einen Wert zwischen Ø und 9 haben kann. Er wird unmittelbar, also ohne Anwendung von Rechenprozeduren ausgedruckt.
Bevor wir das eigentliche Programm besprechen, das die gestellte Aufgabe erfüllt, wollen wir die dort verwendeten PM-Subroutinen aufzählen:

- **CRLF**, Startadresse $11E8. Zwei aufeinanderfolgende grafische Kommandos; sie bewirken, daß eine neue Zeile begonnen wird.

- **RECCHA,** Startadresse $12AE. Wartet auf das Drücken einer Taste und setzt deren ASCII-Kode in den Akku.
- **HEXNUM,** Adresse $126F. Verarbeitet eingegebene numerische Daten. Nach Umwandlung des ASCII-Kodes in ein Datennibble wird dieses von rechts in den Eingangspuffer INL geschoben. Die Puffer INH und INL enthalten Daten, die dem ASCII-Kode der vier zuletzt gedrückten numerischen Tasten entsprechen. Dabei ist ein links stehendes Nibble stets "älter" als ein Nibble, das rechts steht. Wenn es sich um nicht numerische Daten (Daten $\neq$ Ø . . . 9 und A . . . F) handelt, folgt die Rückmeldung "WHAT?".
- **RESIN,** Startadresse $1268. Setzt den Inhalt der Puffer INL und INH auf Null.
- **PRNIBL,** Startadresse $129B. Ist identisch mit der zweiten Hälfte der Subroutine PRBYT. Druckt das rechte Datennibble des Akku-Inhalts aus.

Nähere Einzelheiten dieser Subroutinen werden noch in Kapitel 14 besprochen.

Nun zum Programm. Die Hauptroutine heißt entsprechend dem gesteckten Ziel HEXDEC; sie ist in Bild 2 zu finden. Zuerst wird der Beginn einer neuen Zeile angesteuert (CRLF), dann wartet das Programm auf das Drücken einer Taste. Wird eine Taste gedrückt, so muß zuerst geprüft werden, ob es sich um die Doppelpunkttaste handelt. Trifft dies zu, so ist die Eingabe der umzuwandelnden Hexadezimalzahl vollständig; es kann dann die schon beschriebene Rechen- und Druckprozedur beginnen.

Was geschieht, wenn nicht der Doppelpunkt, sondern eine andere Taste gedrückt wurde? Der Sprungbefehl BNE führt in diesem Fall zum Label DATA und zur Subroutine HEXNUM. Gehört die gedrückte Taste nicht zu den numerischen Tasten, so sorgt HEXNUM für die Fehlermeldung "WHAT?" und ferner für das Nullsetzen der Z-Flag. In diesem Fall führt der auf HEXNUM folgende BNE zum Label NEW: Die Inhalte der Puffer INL und INH werden nullgesetzt; es folgt ein Rücksprung nach HEXDEC. Die Eingabe der Hexadezimalzahl muß wiederholt werden.

Wenn dagegen die gedrückte Taste zur Gruppe der numerischen Tasten gehört, dann sorgt HEXNUM dafür, daß das der gedrückten Taste entsprechende Datennibble zum rechten Nibble von INL wird. Die Inhalte von INH und INL sind so organisiert: In INH stehen stets die ersten beiden Datennibble und in INL die letzten beiden Datennibble, die von den vier zuletzt gedrückten Tasten stammen. Ferner ist ein linkes Datennibble immer "älter" als ein rechtes Datennibble. Nicht benutzte Nibbles sind ØØ.

Wir wollen nun das Geschehen nach Drücken der Doppelpunkttaste betrachten: Dafür ist der Programmteil unter dem linken BNE in Bild 2 zuständig. Vier mal nacheinander werden die Speicherplätze POWERL ($ØØØØ) und POWERH ($ØØØ1) mit Daten geladen, die mit der Frage in Zusammenhang stehen, ob gerade die Zehntausender, Tausender, Hunderter oder Zehner des dezimalen Äquivalents bestimmt werden. Vier mal nacheinander wird die Subroutine AMOUNT angesprungen, die sowohl die ersten vier Ziffern der äquivalenten dezimalen Zahl bestimmt als auch für das Ausdrucken der Ziffern sorgt. Die letzte Ziffer (Einer) steht in INL; sie wird von Subroutine PRNIBL ausgedruckt. Bleibt noch der Sprung zurück nach HEXDEC über NEW (Nullsetzen der Puffer); danach wartet das

2

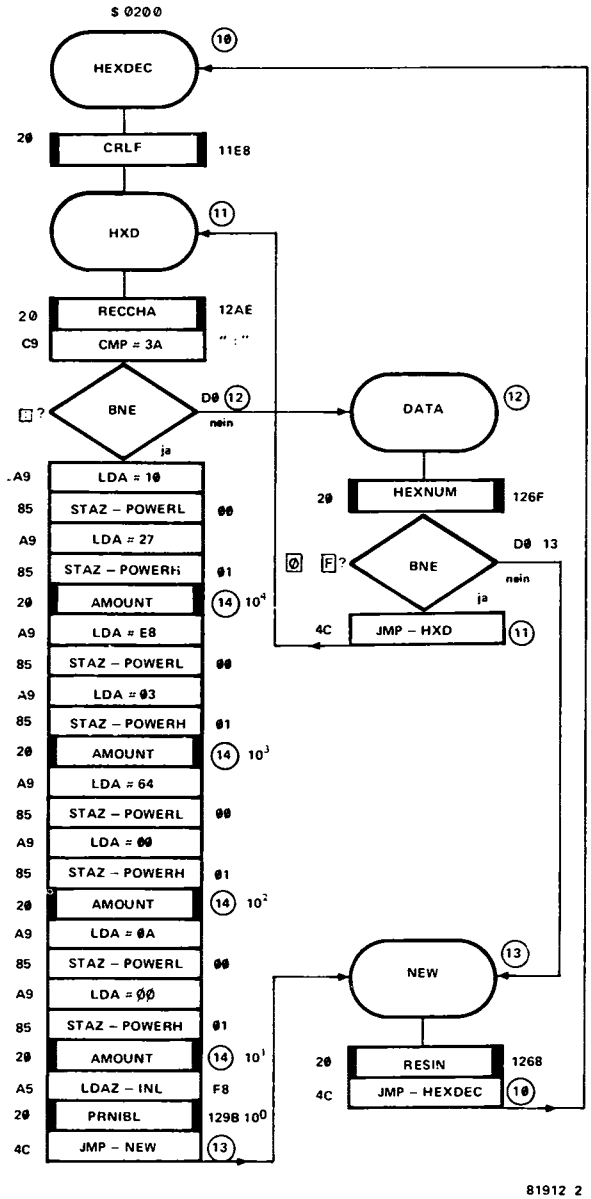


Bild 2. Das Programm HEXDEC. Dieses Programm wandelt hexadezimale Zahlen, die aus maximal vier Ziffern bestehen können, in die dezimale Form um und druckt das Ergebnis aus. Das Bild zeigt das Hauptprogramm von HEXDEC.

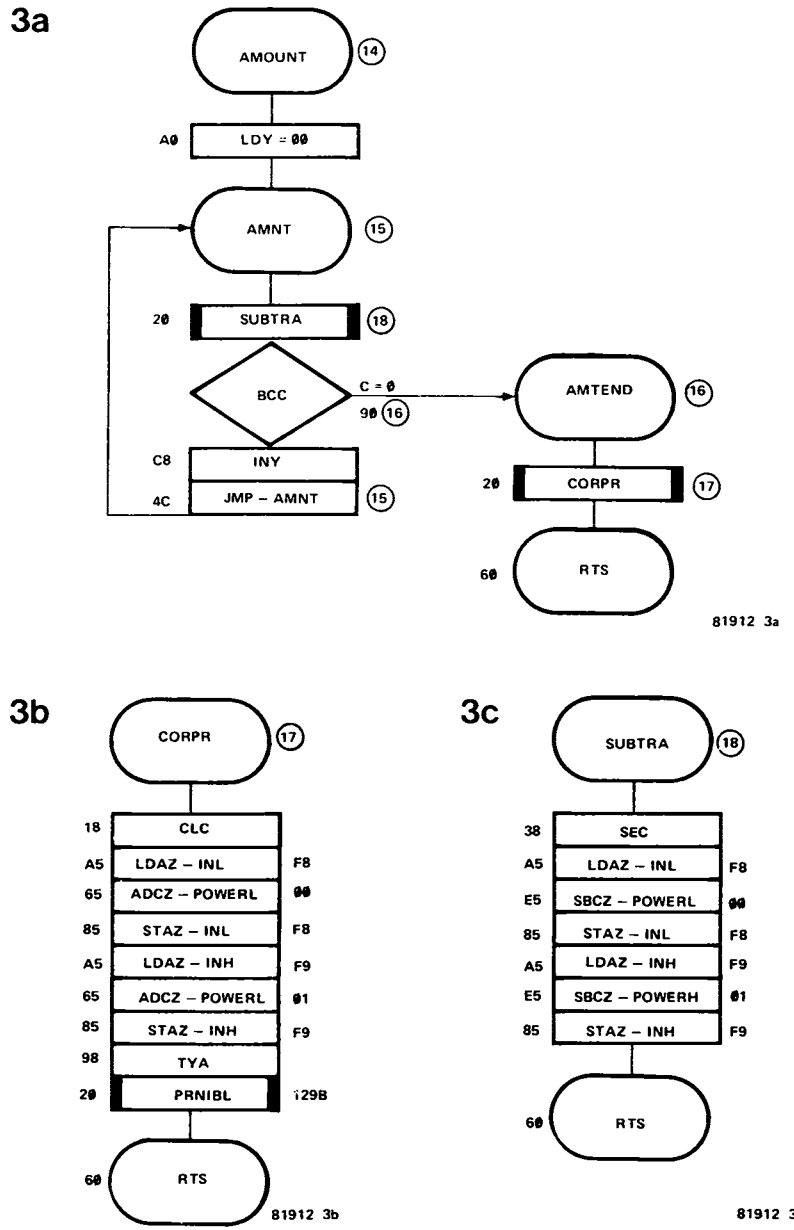


Bild 3. Die Subroutine AMOUNT in Bild 3a bestimmt, wie oft die einzelnen Zehnerpotenzen in der eingegebenen hexadezimalen Zahl enthalten sind. Die zu AMOUNT gehörenden Subroutinen CORPR und SUBTRA sind in Bild 3b und 3c dargestellt.

Programm auf die Eingabe einer weiteren umzuwandelnden Hexadezimalzahl.
Wenden wir uns nun den Details der Subroutine AMOUNT zu; sie ist in Bild 3a dargestellt. Am Anfang der Subroutine steht das Nullsetzen des Y-Registers. Anschließend, nach dem Label AMNT, wird die Subroutine SUBTRA angesprungen. Diese Subroutine ist in Bild 3c zu finden. Von der Zahl (INH, INL) wird die Zahl (POWERH, POWERL) abgezogen. Ist das Ergebnis positiv oder Null, dann ist die Carry-Flag gesetzt (logisch 1). Bei negativem Ergebnis ist sie nicht gesetzt (Ø). Der Zustand der Carry-Flag läßt sich leicht auswerten; es gilt: Carry = $\overline{\text{Borrow}}$ und Borrow = $\overline{\text{Carry}}$. Wenn das Ergebnis negativ ist, muß "geborgt" werden, was gleichbedeutend mit Borrow = 1 bzw. Carry = Ø ist. Erhalten wir ein Ergebnis, das größer oder gleich Null ist, so entfällt das "Borgen".
Bei nicht negativem Resultat folgen eine Inkrementierung des Y-Registers (INY) und eine weitere Subtraktion: Sprung zurück nach AMNT. Ist dagegen das Subtraktionsergebnis negativ, so springt das Programm nach AMTEND. Der Inhalt des Y-Registers ist gleich der Anzahl der Subtraktionen mit nicht negativem Ergebnis.
Was von AMOUNT noch übrig bleibt, ist schnell erklärt. Nach dem Label AMTEND durchläuft der Computer die Subroutine CORPR aus Bild 3b. Zuerst wird zur Zahl (INH, INL) die Zahl (POWERH, POWERL) addiert, so daß der Inhalt von INH/INL gleich der letzten gesuchten Ziffer (Einer) ist. Der Inhalt des Y-Registers wird in den Akku kopiert (TYA), und schließlich druckt PRNIBL die letzte gefundene Ziffer aus.
Wir laden die Programme aus den Bildern 2 und 3 mit Hilfe von PME in den Junior-Computer. Tabelle 4 ist das "Hard Copy" aller Tastenaktionen und zugehörigen PME-Reaktionen. Nach dem Start von PM wird PME kalt gestartet. Dann folgt die Eingabe der Instruktionen, wobei die erste Instruktion wieder mit Tastenfunktion I in den Speicher gelangt. Ein Druck auf Taste X versetzt den Assembler in Aktion. Sobald nach dem Assemblieren Taste ST der Standard-Tastatur betätigt wird, erscheinen sämtliche Label im Bild. Den Schlußpunkt bildet die Rückmeldung "PM EDITOR" zusammen mit der ersten Instruktion auf Speicherplatz ØØØØ.
Das Programm soll gestartet werden. Zuerst müssen wir PME verlassen, denn auch nach dem Ausdrucken der Label befinden wir uns noch dort. Was ist zu tun? Wir drücken nacheinander die Tasten RST, 1, drei mal Ø, GO und RUB OUT (=RES): PM ist gestartet. Das ist erforderlich, weil das Programm HEXDEC von PM-Subroutinen Gebrauch macht. Die I/O muß in Übereinstimmung damit festgelegt werden. Der Start mit Hilfe des Standard-Monitors ist hier nicht zulässig!
Nach dem Start von HEXDEC werden verschiedene hexadezimale Zahlen eingegeben und, jeweils nach Setzen des Doppelpunkts, wird das zugehörige dezimale Äquivalent ausgedruckt. Aus Tabelle 4 geht auch hervor, daß die Zahl ØØØØØ erscheint, wenn ausschließlich die Doppelpunkttaste gedrückt wird.
Das Programm läuft; es erfüllt seine Aufgabe erwartungsgemäß. Zum Schluß wollen wir das Programm zu Dokumentationszwecken in assemblierter Form ausdrucken lassen. Wir starten dazu PME warm. Wie aus Tabelle 4 ersichtlich, lautet die Startadresse nicht 1533, sondern 153D. Das ist hier möglich, weil der zwischen den Adressen 1533 und 153C ste-

Tabelle 4. Hier spiegelt sich das Geschehen während der Eingabe des Programms HEXDEC wieder. Siehe auch Bilder 2 und 3 sowie Praxisbeispiel 2 im Text.

```
SIXTEEN BIT HEXADECIMAL TO DECIMAL CONVERSION
HEXDEC
JUNIOR

1500
1500 20 R
BEGAD,ENDAD: 200,3FF
PM EDITOR
0200 77              IFF 10 00
0200 FF 10 00        20 E8 11
0203 20 E8 11        FF 11 00
0206 FF 11 00        20 AE 12
0209 20 AE 12        C9 3A
020C C9 3A           D0 12
020E D0 12           A9 10
0210 A9 10           85 00
0212 85 00           A9 27
0214 A9 27           85 01
0216 85 01           20 14 00
0218 20 14 00        A9 E8
021B A9 E8           85 00
021D 85 00           A9 03
021F A9 03           85 01
0221 85 01           20 14 00
0223 20 14 00        A9 64
0226 A9 64           85 00
0228 85 00           A9 00
022A A9 00           85 01
022C 85 01           20 14 00
022E 20 14 00        A9 0A
0231 A9 0A           85 00
0233 85 00           A9 00
0235 A9 00           85 01
0237 85 01           20 14 00
0239 20 14 00        A5 F8
023C A5 F8           20 9B 12
023E 20 9B 12        4C 13 00
0241 4C 13 00        FF 12 00
0244 FF 12 00        20 6F 12
0247 20 6F 12        D0 13
024A D0 13           4C 11 00
024C 4C 11 00        FF 13 00
024F FF 13 00        20 68 12
0252 20 68 12        4C 10 00
0255 4C 10 00        FF 14 00
0258 FF 14 00        A0 00
025B A0 00           FF 15 00
025D FF 15 00        20 18 00
0260 20 18 00        90 16
```

```
0263 90 16          C8
0265 C8             4C 15 00
0266 4C 15 00       FF 16 00
0269 FF 16 00       20 17 00
026C 20 17 00       60
026F 60             FF 17 00
0270 FF 17 00       18
0273 18             A5 F8
0274 A5 F8          65 00
0276 65 00          85 F8
0278 85 F8          A5 F9
027A A5 F9          65 01
027C 65 01          85 F9
027E 85 F9          98
0280 98             20 9B 12
0281 20 9B 12       60
0284 60             FF 18 00
0285 FF 18 00       38
0288 38             A5 F8
0289 A5 F8          E5 00
028B E5 00          85 F8
028D 85 F8          A5 F9
028F A5 F9          E5 01
0291 E5 01          85 F9
0293 85 F9          60
0295 60
0296 77             X
LAB $10: $0200 LAB $11: $0203 LAB $12: $023E LAB $13: $0246
LAB $14: $024C LAB $15: $024E LAB $16: $0257 LAB $17: $025B
LAB $18: $026D
PM EDITOR
0200 20 E8 11
JUNIOR

200
0200 20 R
FFFF:65535
FFF:04095
FF:00255
F:00015
ABCD:43981
1000:04096
T
WHAT?

CF16:53014
14F8:05368
17FF:06143
1024:04132
2710:10000
03E8:01000
0064:00100
```

```
000A:00010
:00000

JUNIOR

153D
153D A0 R
PM EDITOR
0200 20 E8 11       P
0203 20 AE 12
0206 C9 3A
0208 D0 34
020A A9 10
020C 85 00
020E A9 27
0210 85 01
0212 20 4C 02
0215 A9 E8
0217 85 00
0219 A9 03
021B 85 01
021D 20 4C 02
0220 A9 64
0222 85 00          P
0224 A9 00
0226 85 01
0228 20 4C 02
022B A9 0A
022D 85 00
022F A9 00
0231 85 01
0233 20 4C 02
0236 A5 F8
0238 20 9B 12
023B 4C 46 02
023E 20 6F 12
0241 D0 03
0243 4C 03 02
0246 20 68 12       P
0249 4C 00 02
024C A0 00
024E 20 6D 02
0251 90 04
0253 C8
0254 4C 4E 02
0257 20 5B 02
025A 60
025B 18
025C A5 F8
025E 65 00
0260 85 F8
0262 A5 F9
0264 65 01
0266 85 F9          P
0268 98
0269 20 9B 12
026C 60
026D 38
026E A5 F8
0270 E5 00
0272 85 F8
0274 A5 F9
0276 E5 01
0278 85 F9
027A 60
027B 77
```

hende Programmteil nur den BRK-Sprungvektor spezifiziert und den Stackpointer (siehe Kapitel 15) nullsetzt. Von der BRK-Taste wird aber kein Gebrauch gemacht (hätten wir BRK gedrückt, dann wäre als Zeichen für den Rücksprung zu PM die Rückmeldung "JUNIOR" erschienen). Der letzte Teil von Tabelle 4 ist das aus drei vollständigen Datenblöcken und einem unvollständigen Datenblock bestehende Listing von HEXDEC, wobei zu jedem vollständigen Datenblock (ausgedruckt nach Drücken von Taste P) 16 Instruktionen gehören.

3. Dezimale Addition

Bei diesem Programm steht der praktische Nutzen stark im Hintergrund, denn für die dezimale Addition von zwei Zahlen benötigt man natürlich keinen Mikrocomputer; ein einfacher Taschenrechner erfüllt diese Aufgabe mit gleicher Zuverlässigkeit. Das nachfolgend beschriebene dezimale Additionsprogramm soll vielmehr das Verständnis weiter vertiefen, das für das Arbeiten mit PM-Subroutinen und das Ausschöpfen der von PME gebotenen Möglichkeiten notwendig ist.
Grundlage für die Addition ist eine vom Operator eingegebene, höchstens sechsstellige dezimale Zahl oder eine aus früheren Additionen als Ergebnis hervorgegangene, höchstens achtstellige dezimale Zahl. Zu dieser Zahl wird eine weitere, vom Operator einzugebende dezimale Zahl addiert. Das Ergebnis ist eine neue kumulative Zahl; zu ihr kann anschließend eine weitere Zahl addiert werden, und so weiter.
Die erste vom Operator einzugebende und aus maximal sechs Ziffern bestehende Zahl wird eingegeben, indem dieser bis zu sechs mal eine der numerischen Tasten Ø . . . 9 und zum Schluß die Taste "." (Punkt) drückt. Dann gibt der Operator die ebenfalls höchstens sechs Stellen umfassende Zahl ein, die zur ersten Zahl addiert werden soll. Danach wird die Taste "P" ("Plus") gedrückt. Wir haben die Taste "P" anstelle der sonst gebräuchlichen Taste "+" gewählt, da so das gleichzeitige Drücken der Shift-Taste entfällt. Wird eine Taste gedrückt, die nicht zu den Tasten ".", "P" und Ø . . . 9 gehört, reagiert der Computer mit der Fehlermeldung "WHAT?".
In der Praxis sieht die Folge von Aktionen und Reaktionen so aus:

```
123456.
    123456
654321P
+   654321
=ØØ777777
1P
+   ØØØØØ1
=ØØ777778
```

Die Aktionen des Operators sind hier wieder im Normaldruck wiedergegeben, während die Computer-Reaktionen fett gedruckt wurden. Übrigens erscheinen die Tastenaktionen bei diesem Programm nicht so auf dem Bildschirm bzw. auf Papier, wie wir es bisher gewohnt waren: Jede mit einem "." oder einem "P" abgeschlossene Zeile wird von der nachfolgenden Reaktionszeile des Computers überschrieben. Für den Drucker bedeutet dies, daß er hier nicht ohne weiteres Lesbares von sich gibt. Auf dem Papier bleibt natürlich die alte Zeile stehen, während der Zeileninhalt auf

dem Bildschirm beim Überschreiben mit einer neuen Zeile gelöscht wird. Die meisten Drucker besitzen aber einen Schalter, mit dem sich das Hardware-Echo der Tastenaktionen abschalten läßt. Das ist der Schalter "Half/Full Duplex"; er wird in Stellung "Full Duplex" gesetzt. Durch das Fehlen der Tastenaktionszeile erhält die ausgedruckte Rechnung eine Form, wie wir sie aus der ersten Schulzeit kennen.

Das dezimale Additionsprogramm, das den Namen DECADD erhielt, ist in Bild 4 angegeben. Im Programm und seinen Subroutinen begegnen wir einer Reihe von Speicherplätzen. Ihre Namen, Adressen und Funktionen gehen aus folgender Aufstellung hervor:

INL	Adresse $ØØF8	eingegebene Zahl, 10^0 und 10^1
INH	Adresse $ØØF9	eingegebene Zahl, 10^2 und 10^3
POINTL	Adresse $ØØFA	eingegebene Zahl, 10^4 und 10^5
DECA	Adresse $ØØDE	kumulative Zahl, 10^0 und 10^1
DECB	Adresse $ØØDF	kumulative Zahl, 10^2 und 10^3
DECC	Adresse $ØØEØ	kumulative Zahl, 10^4 und 10^5
POINTH	Adresse $ØØFB	kumulative Zahl, 10^6 und 10^7

Die höchste kumulative Zahl, die eine Addition als Ergebnis liefert, ist folglich 99 999 999.

In DECADD und seine Subroutinen sind folgende PM-Subroutinen eingebaut:

- **CRLF, RECCHA**; siehe Praxisbeispiel 2.
- **PRCHA**, Adresse $1334. Druckt ein Zeichen oder führt ein grafisches Kommando aus. Der ASCII-Kode des Zeichens oder Kommandos steht im Akku.
- **PRSP**, Adresse $11F3. Druckt einen Leerraum (Zwischenraum).
- **PRBYT**, Adresse $128F. Druckt nacheinander das linke und das rechte Nibble des Akku-Inhalts nach Umsetzen der Nibbles in den entsprechenden ASCII-Kode.
- **MESSY**, Adresse $11D6. Druckt den Text, der in einer Lock-Up-Tabelle steht. Welcher Text gedruckt wird, bestimmen der Anfangswert von Y und die Position des EOT-Zeichens $Ø3 in der Look-Up-Tabelle.

Nähere Einzelheiten zu den genannten Subroutinen sind Kapitel 14 zu entnehmen.

Zu Beginn des Programms DECADD (Bild 4) werden die drei Puffer auf Null gesetzt, die die einzugebende Zahl aufnehmen. Ferner wird eine neue Zeile begonnen (CRLF). Danach wartet das Programm auf das Drücken einer Taste (RECCHA). Nur die Tasten ".", "P" und Ø . . . 9 haben eine Funktion; das Drücken der anderen Tasten wird vom Programm mit einer Fehlermeldung beantwortet.

Gesetzt den Fall, daß eine der Tasten Ø . . . 9 gedrückt wird. In DECADD gelangen wir dann zum Label DATA und damit zur Subroutine DECNUM. Das ist sozusagen die dezimale Version von HEXNUM. Gültige Daten liefert DECNUM nur, wenn eine der Tasten Ø . . . 9 gedrückt wird. In allen anderen Fällen folgt die Rückmeldung "WHAT?"; gleichzeitig wird die Z-Flag nullgesetzt. Durch Drücken einer der genannten numerischen Tasten entsteht ein Datennibble, das von rechts in den Puffer INL geschoben wird. Die in den drei Zahlenpuffern bereits vorhandenen Nibbles rücken hierbei um eine Stelle nach links. In POINTL, INH und INL stehen daher immer die Daten, die von den sechs zuletzt gedrückten Tasten

4

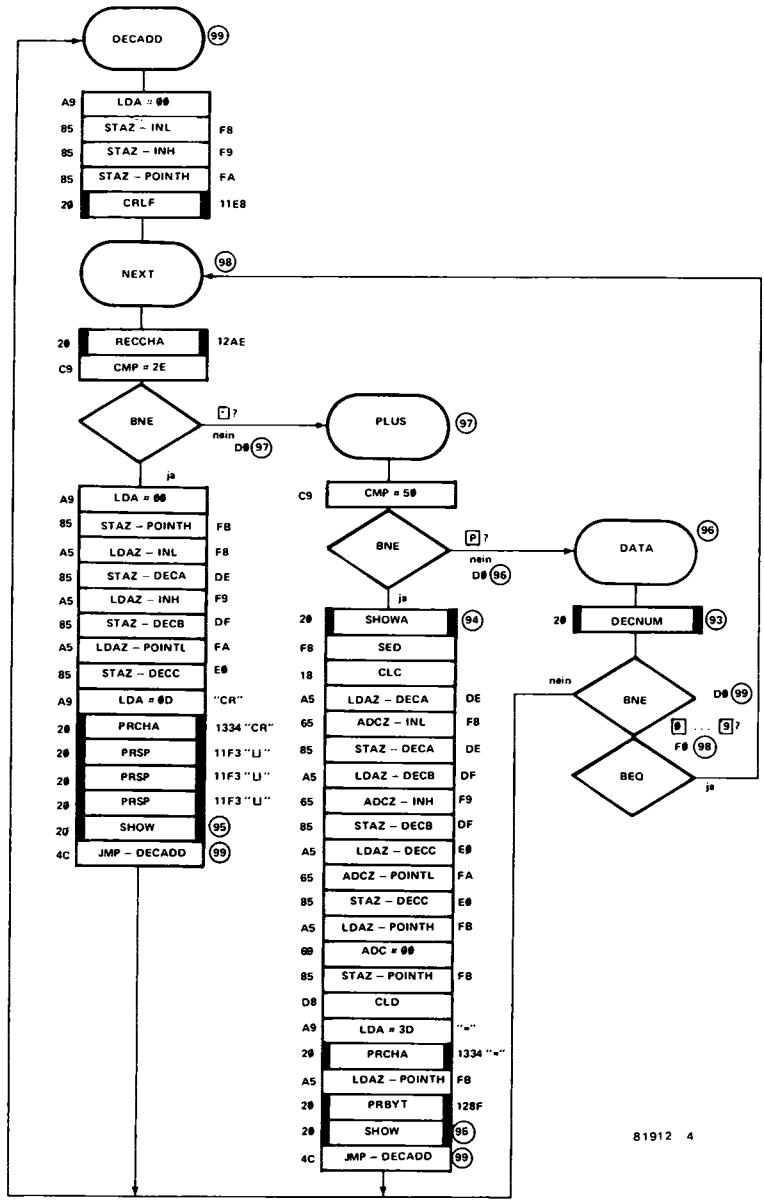


Bild 4. Das Programm DECADD; mit ihm lassen sich eine höchstens sechsstellige dezimale Zahl und eine maximal achtstellige dezimale Zahl addieren. Die zweite Zahl kann auch das Ergebnis vorangegangener Additionen sein (sogenannte kumulative Zahl).

stammen. Ein linkes Nibble ist stets "älter" als ein rechtes Nibble. Das ursprüngliche linke Nibble von POINTL geht beim Einschieben eines neuen Nibble verloren. Werden weniger als sechs Ziffern eingegeben, dann sind die nicht benutzten Nibbles Null. Nachdem DECNUM durchlaufen ist, entscheidet die Z-Flag, ob das Programm noch einmal von Anfang an startet (infolge einer fehlerhaften Eingabe), oder ob es auf das Drücken der nächsten Taste wartet. Im zweiten Fall befinden wir uns beim Label NEXT. Was geschieht, wenn die Eingabe der ersten Zahl durch Drücken von Taste "." beendet wird? Dann wird zuerst die "Schreibtafel" gewischt: der Puffer POINTH, der später die ersten beiden Ziffern des Additionsergebnisses (= kumulative Zahl) aufnimmt, wird nullgesetzt. Es folgt das Kopieren der eingegebenen Zahl in die Puffer, die für die Aufnahme der kumulativen Zahl bestimmt sind, wobei natürlich POINT wegen der geringeren Stellenzahl unberührt bleibt.
Nun sind verschiedene grafische Aktionen an der Reihe. An die Rückkehr zum Beginn der gleichen Zeile (CR) schließt sich das Ausdrucken von drei Leerräumen an (drei mal nacheinander PRSP), gefolgt von Subroutine SHOW, die die eingegebene Zahl ausdruckt. Danach kehren wir zum Programmanfang (DECADD) zurück. Wie Bild 5a zeigt, ist SHOW sehr

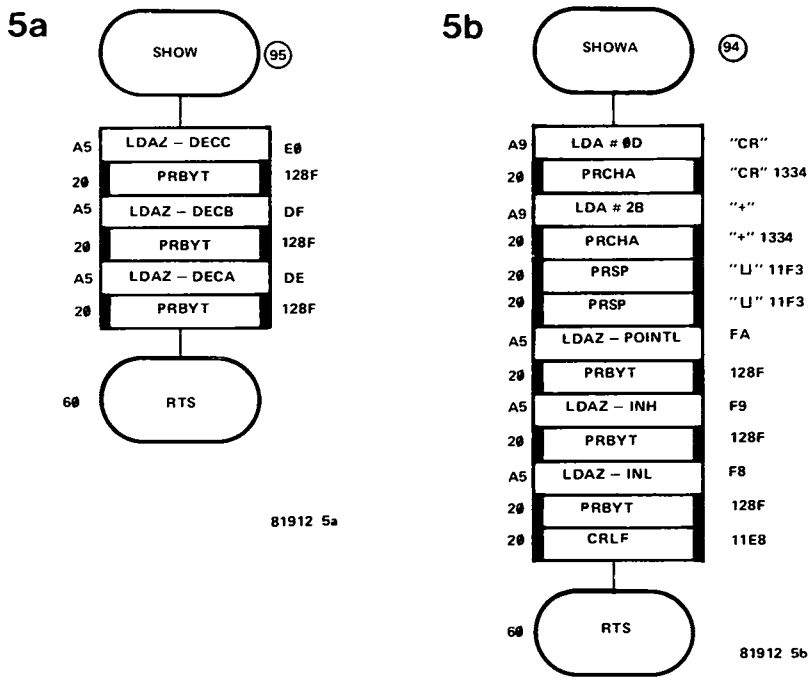
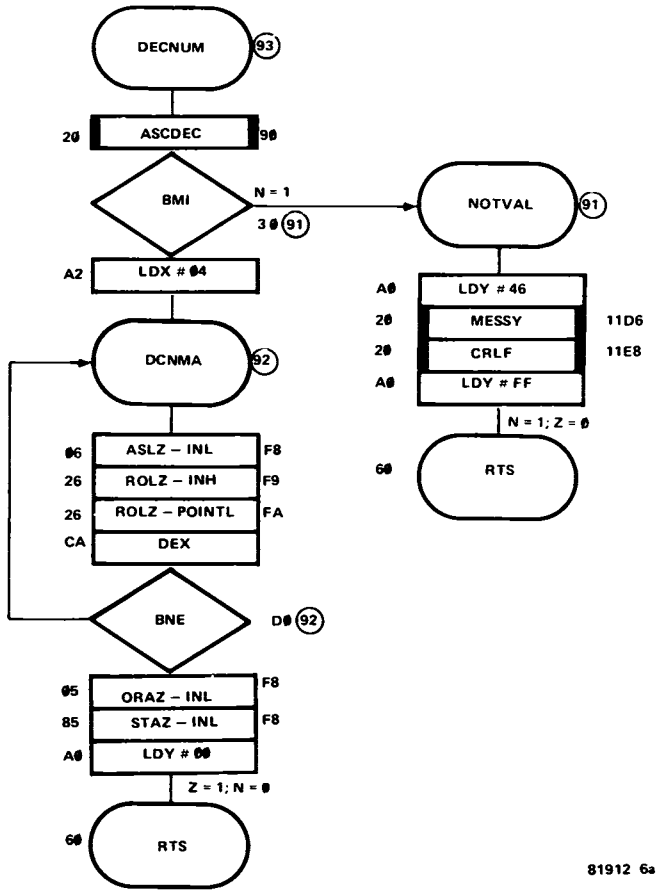


Bild 5. Subroutine SHOW (5a) sorgt in DECADD (Bild 4) für das Ausdrucken der rechten sechs Ziffern der kumulativen Zahl. Eine ähnliche Aufgabe hat Subroutine SHOWA (5b): sie macht die Zahl sichtbar, die zur kumulativen Zahl addiert werden soll.

6a



einfach aufgebaut: sie besteht aus drei Ladeoperationen, auf die jeweils ein Aufruf von PRBYT folgt.
Wir gehen jetzt der Frage nach, was nach der Eingabe der zweiten Zahl und anschließendem Drücken von Taste "P" geschieht. Am Anfang dieses Programmzweiges in Bild 4 steht die Subroutine SHOWA; sie ist in Bild 5b dargestellt. Auch diese Subroutine besteht im wesentlichen aus Druckaktionen. In der gleichen Zeile werden nacheinander das Zeichen "+", zwei Leerräume und die gerade eingegebene zweite Zahl ausgedruckt; danach folgt ein Sprung zum Beginn der nächsten Zeile. Die ausgedruckte Zahl muß zur kumulativen Zahl addiert werden, wobei das Ergebnis gleichzeitig die neue kumulative Zahl ist. Von der Carry-Flag hängt ab (ADC #00), ob sich der Inhalt von Überlaufpuffer POINTH ändert. Wie gefordert führt der Computer die Addition dezimal aus: am Anfang des eigentlichen Rechenprogramms steht der Befehl SED, am Schluß wird die dezimale Rechenweise durch CLD wieder aufgehoben. Letzteres ist notwendig, weil ver-

6b

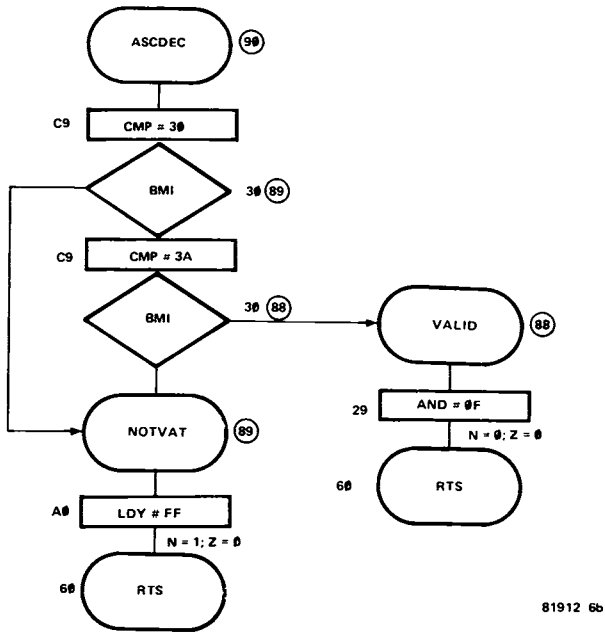


Bild 6. Die Subroutinen DECNUM (6a) und ASCDEC (6b), die sowohl in DECADD (Bild 4) als auch in DECHEX (Bild 7) eingebaut sind, verarbeiten die eingegebenen dezimalen Ziffern Ø . . . 9 und speichern sie in den Zahlenpuffern POINTL, INH und INL. Gleichzeitig sperren sie die Verarbeitung anderer Ziffern, z.B. A . . . F.

schiedene Subroutinen von PM bei gesetzter D-Flag nicht korrekt ausgeführt werden. In Kapitel 12 aus Buch 3 wurden diese Zusammenhänge ausführlich besprochen.
Hat der Computer die Addition erledigt, so wird auf der folgenden Zeile das Zeichen "=" sowie der Inhalt von POINTH zusammen mit den Inhalten der übrigen drei Puffer ausgedruckt, in denen die kumulative Zahl steht. Der letzte Schritt ist der Rücksprung nach DECADD, so daß sofort eine neue Addition begonnen werden kann.
Noch nicht besprochen haben wir die Subroutine DECNUM; sie ist in Bild 6a dargestellt. DECNUM beginnt mit einer anderen Subroutine: ASCDEC in Bild 6b. Der ASCII-Kode der gedrückten Zifferntaste steht im Akku, wobei den Ziffern Ø . . . 9 die ASCII-Kodezahlen 3Ø . . . 39 entsprechen. ASCDEC bewirkt, daß beim Drücken einer nicht zu dieser Gruppe gehörenden Taste die N-Flag gesetzt und die Z-Flag rückgesetzt ist. Gehört dagegen die gedrückte Taste zur Gruppe der dezimalen Zifferntasten

Tabelle 5. Eingabe des Programms DECADD und seiner Subroutinen mit Hilfe von PME. Zu dieser Tabelle gehören die Bilder 4, 5 und 6 sowie Praxisbeispiel 3 im Text.

```
SIX DIGIT DECIMAL ADDITION
UP TO 99,999,999
DECADD
JUNIOR

1500
1500 20 R
BEGAD,ENDAD: 200,3FF
PM EDITOR
0200 77              IFF 99 00
0200 FF 99 00        A9 00
0203 A9 00           85 F8
0205 85 F8           85 F9
0207 85 F9           85 FA
0209 85 FA           20 E8 11
020B 20 E8 11        FF 98 00
020E FF 98 00        20 AE 12
0211 20 AE 12        C9 2E
0214 C9 2E           D0 97
0216 D0 97           A9 00
0218 A9 00           85 FB
021A 85 FB           A5 F8
021C A5 F8           85 DE
021E 85 DE           A5 F9
0220 A5 F9           85 DF
0222 85 DF           A5 FA
0224 A5 FA           85 E0
0226 85 E0           A9 0D
0228 A9 0D           20 34 13
022A 20 34 13        20 F3 11
022D 20 F3 11        20 F3 11
0230 20 F3 11        20 F3 11
0233 20 F3 11        20 95 00
0236 20 95 00        4C 99 00
0239 4C 99 00        FF 97 00
023C FF 97 00        C9 50
023F C9 50           D0 96
0241 D0 96           20 94 00
0243 20 94 00        F8
0246 F8              18
0247 18              A5 DE
0248 A5 DE           65 F8
024A 65 F8           85 DE
024C 85 DE           A5 DF
024E A5 DF           65 F9
0250 65 F9           85 DF
0252 85 DF           A5 E0
0254 A5 E0           65 FA
0256 65 FA           85 E0
```

```
0258 85 E0          A5 FB
025A A5 FB          69 00
025C 69 00          85 FB
025E 85 FB          D8
0260 D8             A9 3D
0261 A9 3D          20 34 13
0263 20 34 13       A5 FB
0266 A5 FB          20 8F 12
0268 20 8F 12       20 95 00
026B 20 95 00       4C 99 00
026E 4C 99 00       FF 96 00
0271 FF 96 00       20 93 00
0274 20 93 00       D0 99
0277 D0 99          F0 98
0279 F0 98          FF 95 00
027B FF 95 00       A5 E0
027E A5 E0          20 8F 12
0280 20 8F 12       A5 DF
0283 A5 DF          20 8F 12
0285 20 8F 12       A5 DE
0288 A5 DE          20 8F 12
028A 20 8F 12       60
028D 60             FF 94 00
028E FF 94 00       A9 0D
0291 A9 0D          20 34 13
0293 20 34 13       A9 2B
0296 A9 2B          20 34 13
0298 20 34 13       20 F3 11
029B 20 F3 11       20 F3 11
029E 20 F3 11       A5 FA
02A1 A5 FA          20 8F 12
02A3 20 8F 12       A5 F9
02A6 A5 F9          20 8F 12
02A8 20 8F 12       A5 F8
02AB A5 F8          20 8F 12
02AD 20 8F 12       20 E8 11
02B0 20 E8 11       60
02B3 60             FF 93 00
02B4 FF 93 00       20 90 00
02B7 20 90 00       30 91
02BA 30 91          A2 04
02BC A2 04          FF 92 00
02BE FF 92 00       06 F8
02C1 06 F8          26 F9
02C3 26 F9          26 FA
02C5 26 FA          CA
02C7 CA             D0 92
02C8 D0 92          05 F8
02CA 05 F8          85 F8
02CC 85 F8          A0 00
02CE A0 00          60
02D0 60             FF 91 00
```

```
02D1 FF 91 00       A0 46
02D4 A0 46          20 D6 11
02D6 20 D6 11       20 E8 11
02D9 20 E8 11       A0 FF
02DC A0 FF          60
02DE 60             FF 90 00
02DF FF 90 00       C9 30
02E2 C9 30          30 89
02E4 30 89          C9 3A
02E6 C9 3A          30 88
02E8 30 88          FF 89 00
02EA FF 89 00       A0 FF
02ED A0 FF          60
02EF 60             FF 88 00
02F0 FF 88 00       29 0F
02F3 29 0F          60
02F5 60
02F6 77             X
LAB $99: $0200 LAB $98: $020B LAB $97: $0236 LAB $96: $0268
LAB $95: $026F LAB $94: $027F LAB $93: $02A2 LAB $92: $02A9
LAB $91: $02B9 LAB $90: $02C4 LAB $89: $02CC LAB $88: $02CF
PM EDITOR
0200 A9 00          P
0202 85 F8
0204 85 F9
0206 85 FA
0208 20 E8 11
020B 20 AE 12
020E C9 2E
0210 D0 24
0212 A9 00
0214 85 FB
0216 A5 F8
0218 85 DE
021A A5 F9
021C 85 DF
021E A5 FA
0220 85 E0          P
0222 A9 0D
0224 20 34 13
0227 20 F3 11
022A 20 F3 11
022D 20 F3 11
0230 20 6F 02
0233 4C 00 02
0236 C9 50
0238 D0 2E
023A 20 7F 02
023D F8
023E 18
023F A5 DE
0241 65 F8
```

```
0243 85 DE          P
0245 A5 DF
0247 65 F9
0249 85 DF
024B A5 E0
024D 65 FA
024F 85 E0
0251 A5 FB
0253 69 00
0255 85 FB
0257 D8
0258 A9 3D
025A 20 34 13
025D A5 FB
025F 20 8F 12
0262 20 6F 02       P
0265 4C 00 02
0268 20 A2 02
026B D0 93
026D F0 9C
026F A5 E0
0271 20 8F 12
0274 A5 DF
0276 20 8F 12
0279 A5 DE
027B 20 8F 12
027E 60
027F A9 0D
0281 20 34 13
0284 A9 2B
0286 20 34 13       P
0289 20 F3 11
028C 20 F3 11
028F A5 FA
0291 20 8F 12
0294 A5 F9
0296 20 8F 12
0299 A5 F8
029B 20 8F 12
029E 20 E8 11
02A1 60
02A2 20 C4 02
02A5 30 12
02A7 A2 04
02A9 06 F8
02AB 26 F9          P
02AD 26 FA
02AF CA
02B0 D0 F7
02B2 05 F8
02B4 85 F8
02B6 A0 00
02B8 60
02B9 A0 46
02BB 20 D6 11
02BE 20 E8 11
02C1 A0 FF
02C3 60
02C4 C9 30
02C6 30 04
02C8 C9 3A          P
02CA 30 03
02CC A0 FF
02CE 60
02CF 29 0F
02D1 60
02D2 77
```

(Label VALID), so wird der ASCII-Kode durch die Instruktion AND #ØF in ein Datennibble gleicher Wertigkeit umgesetzt.
Zurück zu DECNUM. Die auf ASCDEC folgende Verzweigung führt nach Drücken einer nicht zu den dezimalen Ziffern gehörenden Taste zum Label NOTVAL. Nach Ausdrucken der Rückmeldung "WHAT?" (MESSY, mit Y=46; siehe Kapitel 14) wird die N-Flag gesetzt; die Z-Flag ist dann Ø. Das Drücken einer dezimalen Zifferntaste hat dagegen die schon erwähnten Auswirkungen auf den Inhalt von INL, INH und POINTL: Sämtliche Datennibbles rücken um eine Position nach links und machen so Platz für das neue Datennibble (ORAZ-INL und STAZ-INL). Zum Schluß wird die Z-Flag gesetzt und die N-Flag gelöscht.
Wir wollen nun das Programm DECADD mit seinen Subroutinen editieren und anschließend assemblieren. Aus Tabelle 5 geht hervor, was zu tun ist. Nach dem Start von PM (Rückmeldung "JUNIOR") und dem Kaltstart von PME werden sämtliche Instruktionen eingetippt. Die erste Instruktion gelangt wie immer mit Tastenfunktion I (INSERT) in den Computer. Nach Drücken von Taste X wird das Programm assembliert. Ein weiterer Knopfdruck (Taste ST auf der Standard-Tastatur) läßt sämtliche Label im Bild erscheinen; es folgt automatisch ein Sprung zurück zum Warmstarteingang von PME. In seiner assemblierten Form wird das Programm durch wiederholtes Drücken von Taste P ausgedruckt.
Da der Drucker die Tastenaktionen und die Computer-Reaktionen jeweils in der gleichen Zeile übereinander schreibt, wurde in Tabelle 5 auf die Dokumentation der Ausführung von DECADD verzichtet. Natürlich wäre es möglich gewesen, das Programm so zu modifizieren, daß jedesmal vor Ausdrucken einer Computer-Reaktion eine neue Zeile begonnen wird. Dazu braucht nur ein CR-Kommando durch den Sprungbefehl JSR-CRLF ersetzt zu werden. Wir schlagen vor, daß Sie, lieber Leser, diese Kommandos im noch nicht assemblierten Programm selbst ausfindig machen und entsprechend ändern. Ein Verfahren für den Austausch einzelner Instruktionen ist in Praxisbeispiel 5 (siehe nachfolgend in diesem Kapitel) beschrieben.

4. Von Dezimal nach Hexadezimal

Das Programm HEXDEC (siehe Bild 2 und Praxisbeispiel 2) wandelte Hexadezimalzahlen in Dezimalzahlen um. Natürlich läßt sich auch ein Programm realisieren, das Dezimalzahlen in die hexadezimale Form umwandelt. Genau das leistet das Programm DECHEX in Bild 7. Die Verwandschaft dieses Programms mit seinem Gegenstück in Bild 2 fällt unmittelbar ins Auge. Für die Eingabe der dezimalen Zahl (Label DATA in Bild 7) ist Subroutine DECNUM (siehe auch Bild 6a) zuständig, die wir schon vom dezimalen Additionsprogramm DECADD (Bild 4) kennen. DECNUM kann maximal sechs Ziffern verarbeiten, so daß die umzuwandelnde dezimale Zahl im Prinzip sechsstellig sein kann. Wir wollen uns jedoch auf die dezimalen Zahlen beschränken, die kleiner oder gleich 65535 ($FFFF) sind, da nur sie in der Mikrocomputer-Praxis eine Bedeutung haben.
Das Verfahren der Zahlenumwandlung ist bei DECHEX und HEXDEC identisch. Es wird nacheinander festgestellt (Subroutine AMOUNT), wie oft die Zahlen 4096 (= 16^3 oder $1ØØØ), 256 (= 16^2 oder $1ØØ) und 16 (= 16^1 oder $1Ø) in der eingegebenen Zahl bzw. im jeweils übrig

7

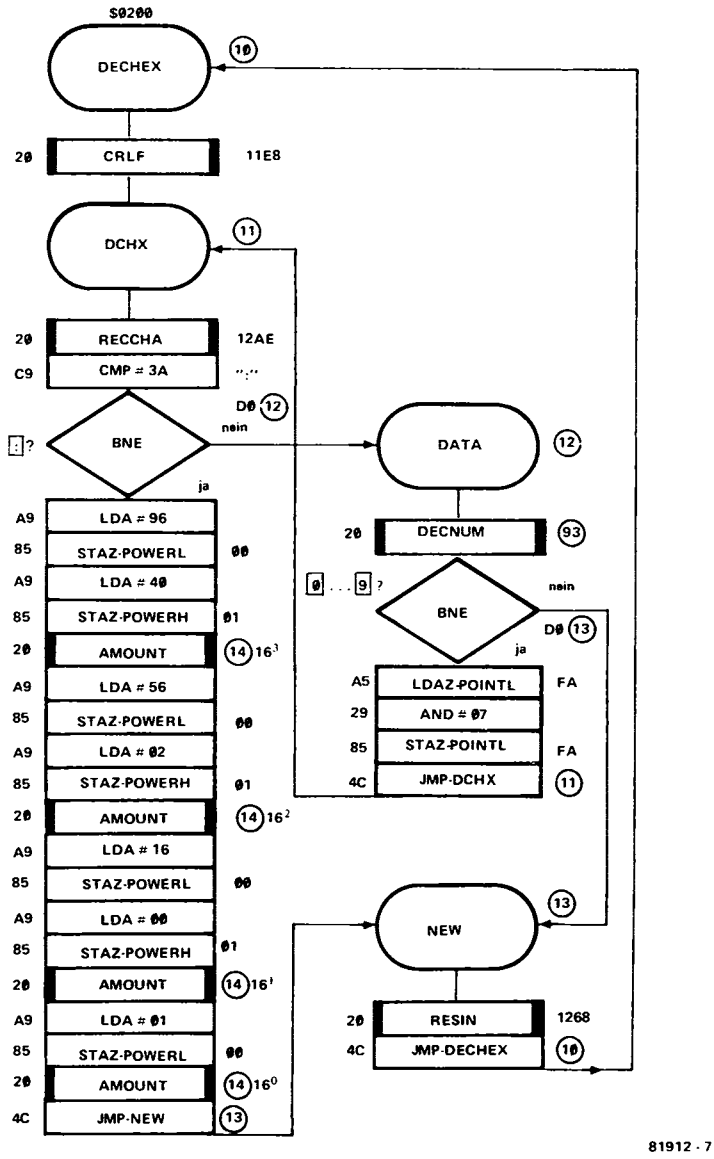


Bild 7. DECHEX ist das Gegenstück zu HEXDEC: Mit DECHEX können dezimale Zahlen, die kleiner als 65536 sind, in Hexadezimalzahlen umgewandelt werden. Schon äußerlich ist die Verwandschaft mit HEXDEC in Bild 2 erkennbar. Die von DECHEX benutzten Subroutinen DECNUM und ASCHEX zeigen die Bilder 6a und 6b, die Subroutine AMOUNT steht in Bild 3a.

bleibenden Rest enthalten sind. Schließlich ermittelt das Programm noch die Anzahl der Einer (16^0 oder $1).
In dem Fall, daß die eingegebene Zahl größer als 65535 ist, liefert DECHEX ein falsches Ergebnis. Das Programm prüft nämlich nicht, ob auch die Zahl 65536 (16^4 oder $10000) in der umzuwandelnden Zahl steckt. Dafür wird dann das Ergebnis, das beim Teilen durch 4096 entsteht, größer als 15 ($F). DECHEX enthält deshalb Vorkehrungen, die unsinnige Ergebnisse weitgehend verhindern: Im linken Nibble von POINTL stehen die Hunderttausender der eingegebenen Zahl, im rechten die Zehntausender. Das Maskieren des Inhalts von POINTL mit Ø7 (AND # Ø7) nach der Eingabe (Label DATA) sorgt dafür, daß die umzuwandelnde Zahl (nach Drücken von ":") keine Hunderttausender mehr enthält und die Anzahl der Zehntausender nicht höher als 7 werden kann.
Die schon von HEXDEC bekannte Subroutine AMOUNT (Bild 3a) läßt sich ohne Änderung auch in DECHEX verwenden. Dagegen müssen die von AMOUNT aufgerufenen Subroutinen CORPR und SUBTRA an ihre Aufgabe in DECHEX angepaßt werden. Die geänderten Versionen sind in den Bildern 8a und 8b dargestellt. Neben der Umstellung auf dezimales Addieren (CORPR) und dezimales Subtrahieren (SUBTRA) ist auch eine Anpassung des Inhalts von POINTL abhängig von der Carry-Flag notwendig (ADC # ØØ in CORPR; SBC # ØØ in SUBTRA).

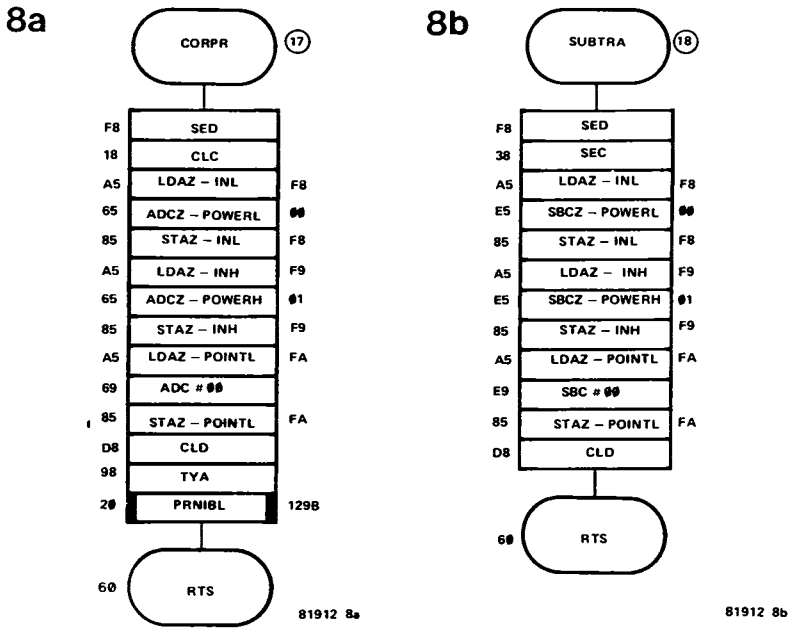


Bild 8. In DECHEX sind die Subroutinen CORPR und SUBTRA aus Bild 3a und 3b nicht ohne weiteres verwendbar, da hier dezimal gerechnet wird und auch der Inhalt von Zahlenpuffer POINTL angepaßt werden muß. Dieses Bild zeigt die auf DECHEX zugeschnittenen Versionen.

Tabelle 6. So wurde mit PME das Programm DECHEX eingegeben, gelistet und anschließend gestartet. Siehe hierzu die Bilder 8, 3a, 8a, 8b, 6a und 6b sowie ferner Praxisbeispiel 4 im Text.

```
JUNIOR

1500
1500 20 R
BEGAD,ENDAD: 200,3FF
PM EDITOR
0200 77              IFF 10 00
0200 FF 10 00        20 E8 11
0203 20 E8 11        FF 11 00
0206 FF 11 00        20 AE 12
0209 20 AE 12        C9 3A
020C C9 3A           D0 12
020E D0 12           A9 96
0210 A9 96           85 00
0212 85 00           A9 40
0214 A9 40           85 01
0216 85 01           20 14 00
0218 20 14 00        A9 56
021B A9 56           85 00
021D 85 00           A9 02
021F A9 02           85 01
0221 85 01           20 14 00
0223 20 14 00        A9 16
0226 A9 16           85 00
0228 85 00           A9 00
022A A9 00           85 01
022C 85 01           20 14 00
022E 20 14 00        A9 01
0231 A9 01           85 00
0233 85 00           20 14 00
0235 20 14 00        4C 13 00
0238 4C 13 00        FF 12 00
023B FF 12 00        20 93 00
023E 20 93 00        D0 13
0241 D0 13           A5 FA
0243 A5 FA           29 07
0245 29 07           85 FA
0247 85 FA           4C 11 00
0249 4C 11 00        FF 13 00
024C FF 13 00        20 68 12
024F 20 68 12        4C 10 00
0252 4C 10 00        FF 14 00
0255 FF 14 00        A0 00
0258 A0 00           FF 15 00
025A FF 15 00        20 18 00
025D 20 18 00        90 16
0260 90 16           C8
0262 C8              4C 15 00
```

```
0263 4C 15 00       FF 16 00
0266 FF 16 00       20 17 00
0269 20 17 00       60
026C 60             FF 17 00
026D FF 17 00       F8
0270 F8             18
0271 18             A5 F8
0272 A5 F8          65 00
0274 65 00          85 F8
0276 85 F8          A5 F9
0278 A5 F9          65 01
027A 65 01          85 F9
027C 85 F9          A5 FA
027E A5 FA          69 00
0280 69 00          85 FA
0282 85 FA          D8
0284 D8             98
0285 98             20 9B 12
0286 20 9B 12       60
0289 60             FF 18 00
028A FF 18 00       F8
028D F8             38
028E 38             A5 F8
028F A5 F8          E5 00
0291 E5 00          85 F8
0293 85 F8          A5 F9
0295 A5 F9          E5 01
0297 E5 01          85 F9
0299 85 F9          A5 FA
029B A5 FA          E9 00
029D E9 00          85 FA
029F 85 FA          D8
02A1 D8             60
02A2 60             FF 93 00
02A3 FF 93 00       20 90 00
02A6 20 90 00       30 91
02A9 30 91          A2 04
02AB A2 04          FF 92 00
02AD FF 92 00       06 F8
02B0 06 F8          26 F9
02B2 26 F9          26 FA
02B4 26 FA          CA
02B6 CA             D0 92
02B7 D0 92          05 F8
02B9 05 F8          85 F8
02BB 85 F8          A0 00
02BD A0 00          60
02BF 60             FF 91 00
02C0 FF 91 00       A0 46
02C3 A0 46          20 D6 11
02C5 20 D6 11       20 E8 11
02C8 20 E8 11       A0 FF
```

```
02CB A0 FF          60
02CD 60             FF 90 00
02CE FF 90 00       C9 30
02D1 C9 30          30 89
02D3 30 89          C9 3A
02D5 C9 3A          30 88
02D7 30 88          FF 89 00
02D9 FF 89 00       A0 FF
02DC A0 FF          60
02DE 60             FF 88 00
02DF FF 88 00       29 0F
02E2 29 0F          60
02E4 60
02E5 77             X
LAB $10: $0200 LAB $11: $0203 LAB $12: $0235 LAB $13: $0243
LAB $14: $0249 LAB $15: $024B LAB $16: $0254 LAB $17: $0258
LAB $18: $0272 LAB $93: $0288 LAB $92: $028F LAB $91: $029F
LAB $90: $02AA LAB $89: $02B2 LAB $88: $02B5
PM EDITOR
0200 20 E8 11       P
0203 20 AE 12
0206 C9 3A
0208 D0 2B
020A A9 96
020C 85 00
020E A9 40
0210 85 01
0212 20 49 02
0215 A9 56
0217 85 00
0219 A9 02
021B 85 01
021D 20 49 02
0220 A9 16
0222 85 00          P
0224 A9 00
0226 85 01
0228 20 49 02
022B A9 01
022D 85 00
022F 20 49 02
0232 4C 43 02
0235 20 88 02
0238 D0 09
023A A5 FA
023C 29 07
023E 85 FA
0240 4C 03 02
0243 20 68 12
0246 4C 00 02       P
0249 A0 00
024B 20 72 02
```

```
024E 90 04
0250 C8
0251 4C 4B 02
0254 20 58 02
0257 60
0258 F8
0259 18
025A A5 F8
025C 65 00
025E 85 F8
0260 A5 F9
0262 65 01
0264 85 F9          P
0266 A5 FA
0268 69 00
026A 85 FA
026C D8
026D 98
026E 20 9B 12
0271 60
0272 F8
0273 38
0274 A5 F8
0276 E5 00
0278 85 F8
027A A5 F9
027C E5 01
027E 85 F9          P
0280 A5 FA
0282 E9 00
0284 85 FA
0286 D8
0287 60
0288 20 AA 02
028B 30 12
028D A2 04
028F 06 F8
0291 26 F9
0293 26 FA
0295 CA
0296 D0 F7
0298 05 F8
029A 85 F8          P
029C A0 00
029E 60
029F A0 46
02A1 20 D6 11
02A4 20 E8 11
02A7 A0 FF
02A9 60
02AA C9 30
02AC 30 04
02AE C9 3A
02B0 30 03
02B2 A0 FF
02B4 60
02B5 29 0F
02B7 60              P
02B8 77
JUNIOR

200
0200 20 R
4096:1000
256:0100
8192:2000
65535:FFFF
10000:2710
1000:03E8
100;
WHAT?

100:0064
4095:CFFF
255:00FF
15:000F
12345:3039
```

Die Praxis des Editierens und Assemblierens spiegelt sich in Tabelle 6 wieder. Nacheinander werden folgende Programmteile eingegeben:
DECHEC, Label 1Ø . . . 13 (Bild 7)
AMOUNT, Label 14 . . . 16 (Bild 3a)
CORPR, Label 17 (Bild 8a)
SUBTRA, Label 18 (Bild 8b)
DECNUM, Label 93 . . . 91 (Bild 6a)
ASCDEC, Label 9Ø . . . 88 (Bild 6b)
Der weitere Ablauf ist inzwischen schon Routine geworden: Assemblieren (X), Rücksprung nach PME (ST) im Anschluß an das Ausdrucken der Label, Ausdrucken des assemblierten Programms (P, P, P . . .) und Starten des Programms nach Rückkehr zu PM.

5. Warmer CEND-Start

PME und die Tonbandkassette

In Kapitel 11 aus Buch 2 haben wir gezeigt, wie man mit TM ein noch nicht assembliertes Programm auf Band konserviert, um es später überarbeiten und bei Bedarf ergänzen zu können. Vor allem beim Schreiben von längeren Programmen ist dies eine wertvolle Hilfe. Man kann die Arbeit beliebig oft für kürzere oder auch längere Zeit unterbrechen: Nach dem Lesen des noch nicht assemblierten Programms vom Band läßt sich mit einem warmen CEND-Start der gleiche Zustand wiederherstellen, der vor der Konservierung vorhanden war.
Um zu demonstrieren, wie PME hier mitwirken kann, haben wir mit Absicht ein besonders kurzes Programm ausgewählt: das Programm ASCII aus Kapitel 12, Buch 2. Bild 9 zeigt ASCII hier noch einmal. Wegen seiner geringen Länge braucht dieses Programm eigentlich nicht editiert und auch nicht assembliert zu werden. Label 1Ø ist nämlich bekannt, es ist identisch mit Startadresse BEGAD! Hier kommt es jedoch auf etwas anderes an: die Vertiefung des Umgangs mit PME und allen seinen Möglichkeiten.
Das Listing von allem, was wir mit dem Programm ASCII unternommen haben, ist die Tabelle 7. Das dort dokumentierte Geschehen wollen wir nun Punkt für Punkt betrachten:

① Systemprogramm PM wird gestartet.
② Es folgt der Kaltstart von PME. Die Startadresse von ASCII lautet $Ø2ØØ.
③ Die Instruktionen werden eingegeben. Wir beginnen mit dem "Inserten" von Label 1Ø. Alles weitere geht aus Bild 9 hervor.
④ Alle Instruktionen sind im Computer. Das Programm wird nach Drücken von Taste L ausgedruckt.
⑤ Das noch nicht assemblierte Programm soll auf Band gesetzt werden. Dazu springen wir zurück nach PM (Rückmeldung "JUNIOR"), legen eine Kassette in das Aufnahmegerät und drücken nacheinander die Tasten S, 8, 8, ", ", 2, Ø, Ø, ", ", 2, 1 und E. Nachdem das Band in Stellung "Aufnahme" gestartet ist, wird CR gedrückt. Nun schreibt der Computer das Programm auf Band und meldet sich danach mit "READY" zurück. Als Programmnummer haben wir "88" gewählt, SA ist gleich BEGAD und EA gleich CEND, eine Adresse, die um eine höher als die Adresse mit der "77" liegt (siehe Listing, Punkt 4).

9

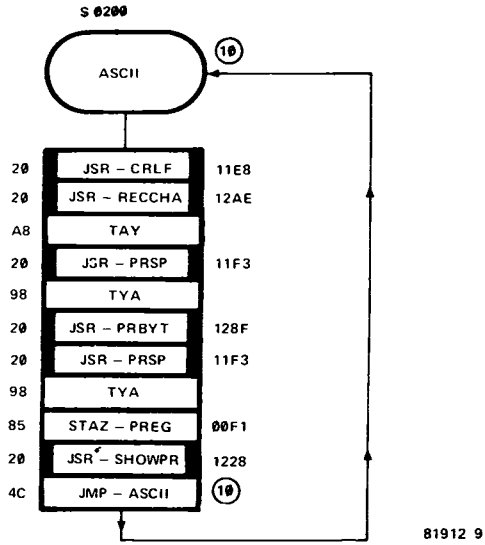


Bild 9. An diesem Programm mit Namen ASCII, das aus Kapitel 12 (Buch 3) stammt, wird im Text der Umgang mit dem warmen CEND-Start von PME demonstriert (Praxisbeispiel 5). Dort ist auch erklärt, weshalb Startadresse BEGAD ($0200) das Label 10 erhielt.

⑥ PME wird warm gestartet.

⑦ Das Programm wird assembliert (Tasten X und ST drücken). Ein einziges Label ist vorhanden; dieses wird ausgedruckt.

⑧ Nach Drücken von P erscheint das Programm in seiner assemblierten Form.

⑨ Um das Programm zu starten, springen wir zurück nach PM. Nach der Eingabe der Startadresse und Drücken von Taste R läuft das Programm. Es soll den ASCII-Kode des Buchstaben A ausdrucken. Das Ergebnis stimmt! Nun wollen wir den ASCII-Kode der Zahl 6 wissen. Die "6" wird jedoch nicht wie erwartet auf der nächsten Zeile ausgedrückt, und auch ihr ASCII-Kode bleibt im Verborgenen. Auch nach Drücken von A, B und C tut das Programm nicht was es soll.
Offensichtlich ist uns bei der Eingabe ein Fehler unterlaufen, denn in Kapitel 12 (Buch 3) arbeitete das Programm noch einwandfrei. Wir wenden die Methode des genauen Hinsehens auf das Listing an und stellen fest, daß das Label der Stein des Anstoßes ist: Der Sprungbefehl am Ende muß natürlich "4C 10 00" und nicht "4C 11 00" lauten. Ein Label 11 ist nicht vorhanden, so daß das Programm nach einmaligem einwandfreien Durchlauf (mit dem Buchstaben A) seinen Dienst versagen mußte.
Schlußfolgerung: Wir lesen das nicht assemblierte Programm noch einmal vom Band, nehmen die erforderlichen Änderungen vor und

Tabelle 7. Schwarz auf Weiß hat der Drucker in dieser Tabelle dokumentiert, was das Programm ASCII über sich ergehen lassen mußte. Dieses einem früheren Kapitel entnommene Programm diente als Übungsobjekt für das Arbeiten mit dem warmen CEND-Start von PME. Siehe auch Bild 9 und Praxisbeispiel 5 im Text.

```
① JUNIOR
② 1500
  1500 20 R
  BEGAD,ENDAD: 200,2FF
  PM EDITOR
③ 0200 77              IFF 10 00
  0200 FF 10 00        20 E8 11
  0203 20 E8 11        20 AE 12
  0206 20 AE 12        A8
  0209 A8              20 F3 11
  020A 20 F3 11        98
  020D 98              20 8F 12
  020E 20 8F 12        20 F3 11
  0211 20 F3 11        98
  0214 98              85 F1
  0215 85 F1           20 28 12
  0217 20 28 12        4C 11 00
④ 021A 4C 11 00        L
  0200 FF 10 00
  0203 20 E8 11
  0206 20 AE 12
  0209 A8
  020A 20 F3 11
  020D 98
  020E 20 8F 12
  0211 20 F3 11
  0214 98
  0215 85 F1
  0217 20 28 12
  021A 4C 11 00
  021D 77
  DONE
  021E E8
⑤ JUNIOR
  S88,200,21E
  READY
⑥ 1533
  1533 A9 R
  PM EDITOR
⑦ 021E E8              X
  LAB $10: $0200
  PM EDITOR
⑧ 0200 20 E8 11        P
  0203 20 AE 12
  0206 A8
  0207 20 F3 11
  020A 98
  020B 20 8F 12
```

```
   020E 20 F3 11
   0211 98
   0212 85 F1
   0214 20 28 12
   0217 4C 11 00
   021A 77
⑨ JUNIOR
   200
   0200 20 R
   A 41 01000001L6ABC
⑩ JUNIOR
   G88
   READY
⑪ 17C5
   17C5 20 R
   BEGAD,ENDAD: 200,2FF
   PM EDITOR
⑫ 021D 77              S4C 11 00
   021A 4C 11 00       K
   DONE
   021A 4C 11 00       K
   021A 77             I4C 10 00
   021A 4C 10 00
⑬ 021D 77              X
   LAB S10: $0200
   PM EDITOR
⑭ 0200 20 E8 11        P
   0203 20 AE 12
   0206 A8
   0207 20 F3 11
   020A 98
   020B 20 8F 12
   020E 20 F3 11
   0211 98
   0212 85 F1
   0214 20 28 12
   0217 4C 00 02
   021A 77
⑮ JUNIOR
   200
   0200 20 R
   A 41 01000001
   B 42 01000010
   C 43 01000011
   D 44 01000100
   1 31 00110001
   2 32 00110010
   3 33 00110011
   4 34 00110100
   K 4B 01001011
   Z 5A 01011010
```

assemblieren es wieder. Danach folgt ein neuer Probelauf.

⑩ Nach Drücken von Taste RST wird PM gestartet und das Programm ASCII in seiner nicht assemblierten Form vom Band gelesen.

⑪ PME wird über den warmen CEND-Eingang gestartet. Die Adressen von BEGAD und ENDAD bleiben unverändert. Nur wenn zu vermuten ist, daß durch die Korrektur einige Instruktionen hinzukommen, wird ENDAD entsprechend erhöht. Da CEND den gleichen Inhalt hat wie unmittelbar vor dem Schreiben des Programms auf Band, können wir mit dem Editieren fortfahren.

⑫ Die zu ändernde Instruktion wird mit SEARCH gesucht. Dann drücken wir Taste K, um die fehlerhafte Instruktion zu löschen. Doch halt: Erst muß eine beliebige Taste außer Y gedrückt werden, damit die Funktion SEARCH aufgehoben ist. Das geschah durch die erste Betätigung von Taste K. Nun wird K noch einmal gedrückt, anschließend wird die korrekte Instruktion "4C 1Ø ØØ" mit Taste I eingefügt.

⑬ Das geänderte Programm wird assembliert.

⑭ Das assemblierte Programm wird ausgedruckt.

⑮ Wir springen zurück nach PM und starten das Programm ASCII zum zweiten Mal. Jetzt läuft es so oft wir wollen.

Anmerkungen und Tips

- Beim kalten, lauwarmen und warmen CEND-Start ist folgendes zu beachten: In der Startphase, also vor Drücken von Taste CR, ist der BRK-Sprungvektor noch von PM bestimmt. Drückt man während des Ausdruckens von "BEGAD, ENDAD:" auf die BRK-Taste, dann folgt beim Loslassen die Rückmeldung "JUNIOR"; wir befinden uns wieder in PM. Ist die Eingabe von BEGAD und ENDAD fehlerhaft, dann erscheint die Rückmeldung "WHAT?". Anschließend wird noch einmal "BEGAD, ENDAD:" ausgedruckt. Warum dies geschieht, erklären wir in den Kapiteln 14 und 15. Eine Fehlermeldung erscheint nicht, wenn ENDAD niedriger als BEGAD ist; darauf haben wir bereits hingewiesen.
- Eine Anmerkung zur Eingabe von Adreßoperanden, die zu assemblieren sind: In Kapitel 5 (Buch 2) wurde gesagt, daß dem Opcode (2Ø für JSR, 4C für JMP) die Labelnummer des Sprungziels sowie ein zweites Operandenbyte folgen muß; letzteres ist das Begrenzer-Byte (Labelbegrenzer). Sowohl in Kapitel 5 als auch in den hier gegebenen Beispielen wurde dafür konsequent das Byte ØØ gewählt. Das Gleiche gilt für das dritte Label-Byte, denn "ØØ" fällt besonders leicht in Auge. Es ist jedoch nicht zwingend notwendig, als drittes Byte "ØØ" zu wählen. Dies ergibt sich aus den Vorgängen während der ersten Assemblierphase; siehe hierzu Kapitel 9 (Buch 2). Wichtig ist nur, daß für das zweite Operandenbyte eines JSR oder JMP ein Speicherplatz reserviert wird: Eine Instruktion gelangt erst in den Speicher, nachdem so viele Bytes eingegeben sind, wie es der Länge der Instruktion entspricht. PME macht dies durch seine Reaktion auf der nächsten Zeile deutlich; die gerade eingegebene Instruktion wird auf der linken Seite ausgedruckt.
- Eine Labelnummer darf niemals gleich dem rechten Byte ADL einer absoluten Adresse sein, das natürlich nicht assembliert zu werden

braucht. Es empfiehlt sich, die im Programm vorkommenden absoluten Adressen zu notieren, bevor man die Labelnummern wählt.

- Wenn bei Sprunginstruktionen der absolute Offset bekannt ist, sollte man trotzdem darauf verzichten, ihn in die Instruktion einzusetzen. Die Gefahr ist nämlich groß, daß dieser Offset im Programm auch als Labelnummer vorkommt.
- Jede Labelnummer darf nur ein einziges Mal verwendet werden. Obwohl die Reihenfolge der Labelnummern im Prinzip beliebig ist, kann auch hier ein wenig Ordnung nicht schaden. Von Vorteil ist eine auf- oder absteigende Zahlenfolge (die eventuell als Labelnummern unzulässigen Zahlen werden überschlagen) sowie die Reihenfolge, in der die Label während des Editierens in den Speicher gelangen. Dann läßt sich leicht überprüfen, ob der Offset einer Branch-Instruktion überschritten wird. Notfalls "brancht" man zu einem Label im erlaubten Bereich, dem ein absoluter Sprung folgt.

So weit die Tips, die uns noch wichtig erschienen. Zum Teil waren es Wiederholungsübungen aus Kapitel 5, Buch 2. Ihre Übungen, liebe Leserin und lieber Leser, die Sie mit PME unternehmen, sind jedoch alles andere als Wiederholungsübungen. Nur die Begegnung mit dem System der hexadezimalen Label und der Labelnummern wiederholt sich bei PME noch einmal. Wir wünschen Ihnen viel Freude mit PME und natürlich mit den durch seine Hilfe entstandenen Programmen.

14

1268-Byte PM-Software

Das Junior-TV-Programm

In Kapitel 12 aus Buch 3 haben wir die einzelnen Tastenfunktionen des Systemprogramms PM und den Umgang mit ihnen in der Praxis kennengelernt. Dieses Kapitel knüpft daran an: Hier geht es um das "Innenleben" von PM.
Weshalb wollen wir tiefer in die PM-Materie eindringen? Dafür gibt es zwei gute Gründe: Zum einen hat PM für die Entwicklung eigener Programme exemplarischen Charakter. Auch an ihm lassen sich Lösungswege studieren, die in gleicher oder ähnlicher Form vielfältig begehbar sind. Zum anderen können die verschiedenen PM-Subroutinen in der eigenen "Software-Küche" als gebrauchsfertige Zutaten verwendet werden, wenn ihre Eigenschaften und Möglichkeiten bekannt sind. Das demonstrieren unter anderem die in die Kapitel 12 und 13 eingestreuten Programmbeispiele.
Mit diesem Kapitel verhält es sich ähnlich wie mit vergleichbaren Kapiteln aus Buch 2: Man kann es überschlagen, wenn man auf ein tieferes Eindringen in die Materie verzichten will. Andererseits sind aber der Vertiefung im Rahmen dieses Buches schon aus Platzgründen Grenzen gesetzt. Die PM-Software ist nämlich noch umfangreicher als die in Buch 2 besprochene Standard-Monitor-Software. Wir müssen uns deshalb auf eine Punkt-für-Punkt-Beschreibung der einzelnen Details beschränken. Ferner setzen wir voraus, daß inzwischen einige Kenntnisse und Erfahrungen mit der 6502-Maschinensprache vorhanden sind, auf die wir jetzt zurückgreifen können.

1 In der Regel gehören zu jedem Systemprogramm eine Hauptroutine, eine oder mehrere Vorroutinen sowie natürlich eine Anzahl von Subroutinen. Die Hauptroutine eines Systemprogramms hat meistens folgende Struktur: Sie wartet auf die Betätigung von Bedienelementen, z.B. von Tasten, führt die zugehörigen Funktionen aus und kehrt anschließend zum Wartezustand zurück, so daß eine geschlossene Programmschleife entsteht.
Vorroutinen werden nur initial, das heißt vorbereitend durchlaufen. Sie schaffen die Voraussetzungen, die für den Start des eigentlichen Programms notwendig sind. Die Vorroutine von PM (Bild 1a und 1b) ist mit der Reset-Routine des Standard-Monitors vergleichbar: sie läuft beim ersten bzw. zweiten Start von PM ab (vgl. Buch 3, S. 140 Mitte). Der auf

1a

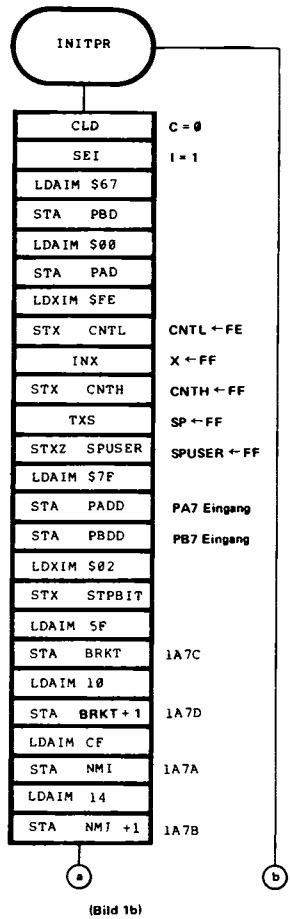


Bild 1a. Erster Teil der PM-Vorroutine. Die Vorroutine schafft die Voraussetzungen, die für den Start der Hauptroutine notwendig sind.

das Label LABJUN (Bild 1b) folgende Teil wird nach Drücken der BRK-Taste und im Anschluß an die Rückmeldung "JUNIOR" durchlaufen, während die Routine STEP (ebenfalls in Bild 1b) nach Ausführen einer einzelnen Instruktion im Step-Modus oder, allgemeiner betrachtet, nach Auftreten eines NMI abläuft.
Am Anfang der **Vorroutine INITPR** (Bild 1a) stehen verschiedene elementare Operationen: Der Prozessor wird veranlaßt, binär zu rechnen; IRQ-Interrupts werden durch SEI blockiert. INITPR legt ferner die I/O-Ports fest. PADD und PBDD werden mit 7F geladen, so daß bis auf PA7 und PB7 alle Portleitungen als Ausgänge geschaltet sind. Damit ist Portleitung PB7 in Bereitschaft, Daten von der Bandkassette zu empfangen, PA7 kann ebenfalls serielle Daten empfangen, und PBØ, geschaltet als Ausgang, kann serielle Daten senden. Da in PAD der Wert ØØ gesetzt wird, liegt an den Ausgängen PAØ . . . PA6 logisch 1. Dies hätte zur Folge, daß sämtliche Segmente aller sechs Displays aufleuchten (vgl. Kapitel 7), wenn nicht der Displaymultiplexer infolge der 67 in PBD in neutraler Stellung stehen würde. PBD=67 bedeutet, daß PB1=1, PB2=1, PB3=Ø und PB4=Ø ist. Der nicht angeschlosse Ausgang "3" von Dekoder IC7 auf der Basiskarte ist dann logisch Ø; siehe Buch 2, S. 130, Bild 10. Durch Setzen von PB5 und PB6 auf logisch 1 werden die Anzeige-LEDs und Steuerungen der beiden Kassettenlaufwerke abgeschaltet (siehe Schaltung der Interfacekarte in Buch 3). Ferner wird durch die logische 1 an Ausgang PBØ erreicht, daß an dieser für die serielle Datenausgabe bestimmten Leitung das Ruhesignal liegt. Es muß logisch 1 sein, solange der Junior-Computer keine Daten sendet; der logische Zustand des RS232-Ausgangs ist dann logisch Ø.
Damit sind die Vorbereitungen noch längst nicht abgeschlossen. Es werden noch die Stackpointer auf FF gesetzt, und der Speicherplatz STBIT erhält den Wert Ø2. Letzteres hat zur Folge, daß am Schluß jedes vom Computer ausgegebenen ASCII-Kodes zwei Stopbit stehen. Schließlich wird noch der BRK-Sprungvektor auf die Startadresse von LABJUN (Bild 1b; Rückmeldung "JUNIOR") und der NMI-Sprungvektor auf die Startadresse von STEP (ebenfalls in Bild 1b) gerichtet. Übrig bleibt in Bild 1a noch das Laden von CNTL mit FE und CNTH mit FF. Der Grund dafür wird nachfolgend deutlich.

2 Wir sind in Bild 1b angekommen. Das Programm in Bild 1a ist nach dem ersten Start von PM vollständig abgearbeitet. Wann folgt auf dem ersten Start der zweite Start? Aus Kapitel 12 wissen wir, daß für den zweiten Start das Drücken der Taste RUB OUT (= RES) notwendig ist. Dadurch wird der hierzu gehörende ASCII-Kode 7F erzeugt. Die Datenübertragung beginnt mit dem Startbit: in Bild 2 ist der Verlauf des Signals skizziert, das vom Startbit eingeleitet wird. Am Anfang der Routine STRTBT (Bild 1a) steht eine Warteschleife, die mit Hilfe einer BIT-Instruktion das logische Signal am seriellen Eingang PA7 überwacht. Die BIT-Instruktion sorgt dafür, daß die N-Flag gleich Bit 7 des Speicherinhalts ist, auf den die BIT-Instruktion angewendet wird. Sobald der Beginn eines Startbit festgestellt ist, verzweigt das Programm zur Subroutine COMTIM; sie ist in Bild 3 dargestellt. COMTIM wartet zuerst das Ende des Startbit ab, das durch die Rückkehr des Signals an PA7 nach logisch 1 gekennzeichnet ist. Auf das Startbit muß deshalb eine logische 1 folgen, weil der

1b

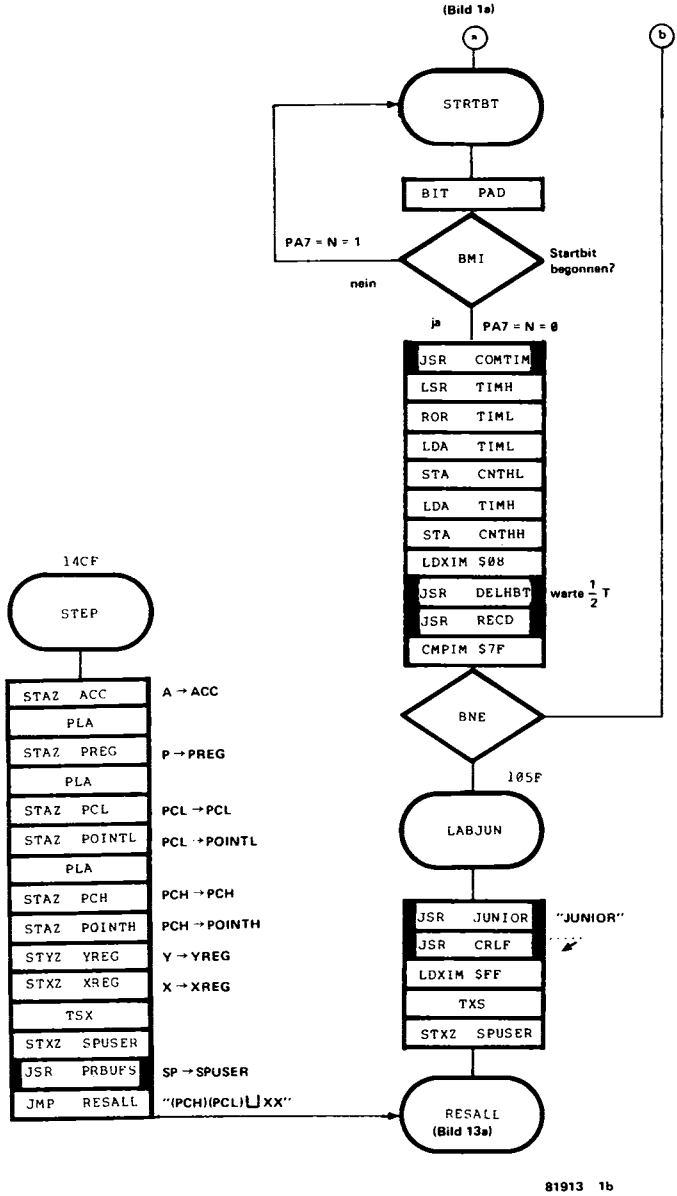


Bild 1b. Zweiter Teil der PM-Vorroutine einschließlich der STEP-Vorroutine. Vergleichbar ist die PM-Vorroutine mit der RESET-Routine des Standard-Monitors.

2

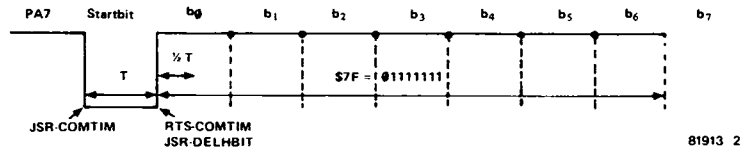


Bild 2. Impulsdiagramm des seriellen ASCII-Kodes 7F. Zwischen dem Ende des Startbit und der Abtastung des ersten Bitwerts liegt eine halbe Bitzeit, so daß alle Bit in ihrer Mitte abgetastet werden.

3

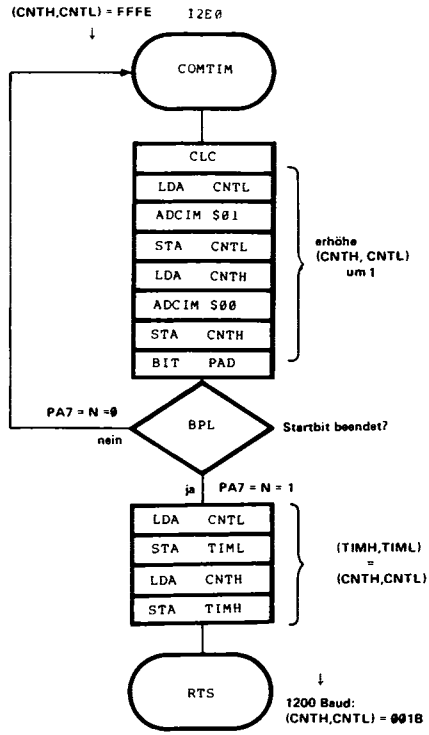


Bild 3. Subroutine COMTIM mißt die Dauer des ersten ankommenden Startbit. Da der Inhalt der Speicherplätze CNTH/CNTL hierzu proportional ist, steht damit die Bitzeit T des empfangenen Signals fest.

ASCII-Kode 7F mit sieben mal logisch 1 beginnt (siehe Bild 2).
Während der Dauer des Startbit wird zu der in CNTH und CNTL stehenden 16-Bit-Zahl periodisch die Zahl 1 addiert. Jede einzelne Addition nimmt eine bestimmte konstante Zeit in Anspruch. Der Anfangswert von (CNTH, CNTL) ist FFFE. Je länger nun die elementare Bitzeit T des empfangenen

seriellen Signals ist, desto höher liegt der Endwert von (CNTH, CNTL). Die Bitzeit wird von der Datenübertragungsgeschwindigkeit (Baudrate) bestimmt; sie hängt in diesem Fall von der Frequenz des Taktgenerators und von den Eigenschaften des UART im peripheren Gerät ab. Der Zweck der periodischen Inkrementierung von (CNTH, CNTL) ist unter anderem die selbsttätige Anpassung der Geschwindigkeit, mit der der Junior-Computer Daten sendet, an die Übertragungsgeschwindigkeit des angeschlossenen peripheren Geräts.
Beim Bau des Elekterminals für den Junior-Computer (Kapitel 12, Buch 3) haben wir auf den Baudrate-Schalter verzichtet, so daß das Elekterminal mit der konstanten Datenübertragungsgeschwindigkeit 1200 Baud arbeitet Dieser Baudrate entspricht ein Wert von 1B in CNTL und ein Wert von ØØ in CNTH. Man kann das leicht überprüfen, indem man die Inhalte der Speicherplätze 1A5A (CNTL) und 1A5B (CNTH) sichtbar macht. Der Junior-Computer ist auf die beschriebene Weise in der Lage, sich innerhalb weiter Grenzen an die Übertragungsgeschwindigkeiten peripherer Geräte anzupassen. Die Grenze ist dort gesetzt, wo der Inhalt von CNTH infolge einer extrem niedrigen Baudrate den Wert FF übersteigt.
Am Schluß vom COMTIM wird der Endstand von (CNTH, CNTL) in die Speicherzellen TIMH und TIML kopiert.

3 Die Inhalte von TIMH und TIML werden unmittelbar nach Ende der Subroutine COMTIM weiterverarbeitet. Sämtliche Bit von (TIMH, TIML) rücken um eine Stelle weiter nach rechts. Das geschieht durch Schieben der Bit von TIMH nach rechts (LSR) und Drehen der Bit von TIML nach rechts (ROR). Die Folge ist, daß die Zahl (CNTHH, CTNHL) ungefähr die Hälfte der Zahl (CNTH, CNTL) beträgt; exakt halbiert wird sie nur bei geraden Zahlen. Die Speicherplätze CNTH und CNTL enthalten folglich eine Zahl, die zur Bitzeit T in einem bestimmten Verhältnis steht, während die Zahl in den Speicherplätzen CNTHH und CNTHL zur halben Bitzeit proportional ist. Der Sinn dieser Operation wird aus dem folgenden Punkt 4 zusammen mit Punkt 6 deutlich.

4 Bevor wir die Betrachtung der PM-Vorroutine in Bild 1b fortsetzen, sollen zuerst einige elementare Subroutinen besprochen werden. Dazu gehören die **Subroutinen DELHBT** und **DELBIT**, die in Bild 4 dargestellt sind. Wir gehen davon aus, daß Bitzeit T in Form des Inhalts von CNTH, CNTL festliegt und daß das Gleiche für die halbe Bitzeit und den Inhalt der Speicherplätze CNTHH und CNTHL gilt (siehe Punkt 3). Die Subroutinen DELHBT und DELBIT sorgen dafür, daß bestimmte andere Programmschritte um die halbe Bitzeit T/2 bzw. um die ganze Bitzeit T verzögert werden: Solange die Verzweigung BCS hinter dem Label CNTDN (Count Down; Abwärtszählen) in Bild 4 auf "Springen" steht, wird die Zahl (TIMH, TIML) periodisch um 1 vermindert. Der Anfangswert von (TIMH, TIML) ist gleich dem Endwert der Zahl (CNTH, CNTL) nach Ende von COMTIM (Eingang DELBIT) oder gleich dem durch 2 geteilten Endwert von (CNTH, CNTL), also gleich der Zahl (CNTHH, CNTHL) (Eingang DELHBT). Die periodische Dekrementierung dauert so lange an, bis die BCS-Verzweigung nicht mehr auf "Springen" steht; Carry-Flag C ist dann Ø, Borrow=$\overline{C}$ ist gleich 1. In TIMH, TIML steht nun die negative Zahl

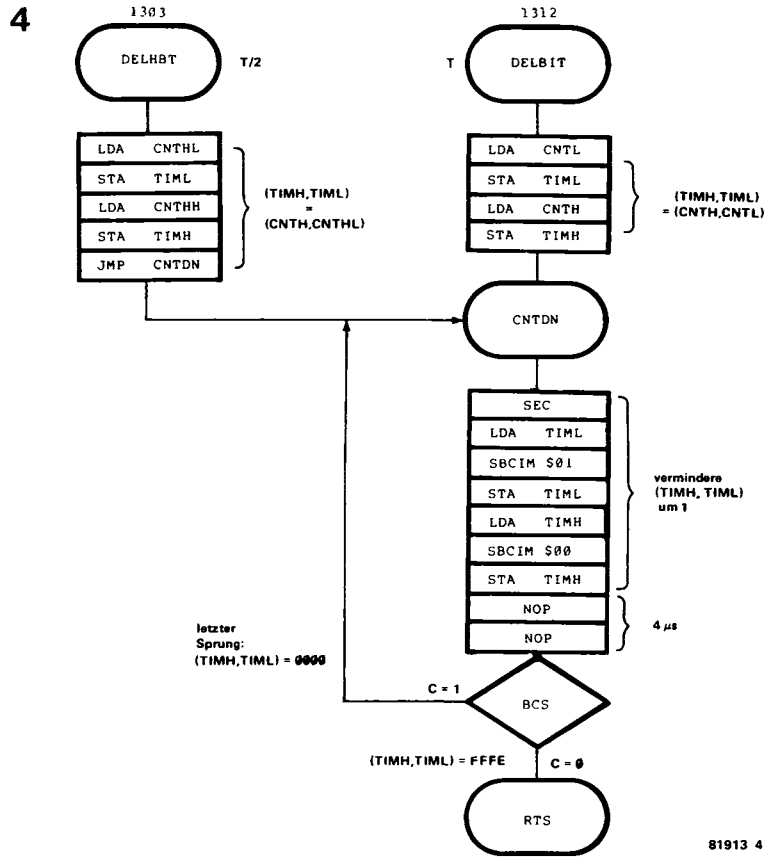


Bild 4. Die Subroutinen DELHBT und DELBIT sorgen für die notwendigen Verzögerungen um eine halbe bzw. ganze Bitzeit.

FFFE. **Diese Zahl ist gleich dem Anfangswert von (CNTH, CNTL) zu Beginn der Subroutine COMTIM** (siehe Punkt 2). Mit anderen Worten: **Wenn wir dafür sorgen, daß der Durchlauf des Programmteils zwischen dem Label CNTDN und der Verzweigung BCS die gleiche Zeit dauert wie das Abarbeiten des zwischen Label COMTIM (Bild 3) und Verzweigung BPL liegenden Programmteils, erhalten wir eine Verzögerung von einer halben bzw. einer ganzen Bitzeit.**

Von Label COMTIM bis einschließlich Verzweigung BPL (Bild 3) benötigt der Prozessor 29 µs, während er den Programmteil zwischen Label CNTDN und Verzweigung BCS (Bild 4) in 25 µs durchlaufen würde, wenn nicht zwei eingefügte NOP-Instruktionen diesen Programmteil auf ebenfalls 29 µs verlängerten.

Der schon in Punkt 2 erwähnte Endstand ØØ1B von (CNTH, CNTL) bedeutet, daß der in diesen Speicherzellen stehende Anfangswert FFFE 29 mal

dekrementiert wurde, so daß insgesamt 29 • 29 µs = 841 µs verstrichen sind. Der Aufruf der Subroutine COMTIM dauert weitere 6 µs, und ferner wird der Beginn des Startbit höchstwahrscheinlich nicht exakt an seiner Flanke festgestellt (Abbruch der Warteschleife mit BIT-PAD, oben in Bild 1b). Wir können daher mit einer Bitzeit von annähernd 850 µs rechnen. Da eine Baudrate von 1200 bit/s gleichbedeutend mit einer Bitzeit von 833 µs ist und das Systemprogramm PM wesentlich größere Abweichungen zuläßt, ist das gesteckte Ziel erreicht.

5 Der zweite Exkurs betrifft die **Subroutine RECCHA** in Bild 5. Diese Subroutine dient zur Erkennung eines zum Junior-Computer gesendeten ASCII-Kodes; dieser wird von RECCHA in den Akku gesetzt. Der Junior-Computer empfängt unter anderem immer dann ein solches serielles Bitmuster, wenn auf der ASCII-Tastatur eine Softwaretaste gedrückt wird. Da hierbei auch ein Hardware-Echo entsteht (siehe Kapitel 12), erscheint das zugehörige Zeichen auch auf dem Bildschirm oder auf dem Papier des Druckers.
Am Anfang von RECCHA (Bild 5) steht eine Warteschleife: BIT-PAD mit dem nachfolgenden BMI. Erst wenn ein Startbit beginnt (PA7 = Ø), läuft das Programm weiter. Der Wert des X-Registers wird dann in Speicherplatz TEMPB gesetzt. Wie aus dem rechten unteren Teil von Bild 5 (oberhalb des RTS) hervorgeht, erhält das X-Register seinen ursprünglichen Inhalt später wieder zurück. **Der Inhalt des X-Registers geht deshalb während des Durchlaufs durch RECCHA nicht verloren.** Innerhalb von RECCHA dient das X-Register als Bitzähler; sein Anfangswert ist Ø8. Zuerst wird eine halbe Bitzeit und anschließend, nach Label RECA, zusätzlich eine ganze Bitzeit gewartet. Danach befinden wir uns inmitten der Periode, während der das erste ASCII-Bit bØ gesendet wird (siehe Bild 2). Mit der Instruktion BIT-PAD und der folgenden Verzweigung BPL stellt das Programm fest, ob dieses Bit Ø oder 1 ist. Die Carry-Flag wird entsprechend dem Bitwert auf Ø oder 1 gesetzt; das Carry-Bit wird mit einer ROR-Instruktion von links in Speicherplatz CHA geschoben. Sämtliche Bit in CHA rücken dabei um eine Stelle nach rechts weiter.
Der Inhalt von CHA verändert sich während der einzelnen Phasen des Zeichenempfangs wie folgt:

nach X = Ø8:	bØ	X	X	X	X	X	X	X
nach X = Ø7:	b1	bØ	X	X	X	X	X	X
nach X = Ø6:	b2	b1	bØ	X	X	X	X	X
nach X = Ø5:	b3	b2	b1	bØ	X	X	X	X
nach X = Ø4:	b4	b3	b2	b1	bØ	X	X	X
nach X = Ø3:	b5	b4	b3	b2	b1	bØ	X	X
nach X = Ø2:	b6	b5	b4	b3	b2	b1	bØ	X
nach X = Ø1:	b7	b6	b5	b4	b3	b2	b1	bØ

Sämtliche Bit stehen jetzt in der richtigen Reihenfolge in CHA, so daß sie weiterverarbeitet werden können. Nach Label RECC wartet das Programm noch eine Bitzeit bis zum ersten Stopbit. Danach wird der Inhalt von CHA in den Akku kopiert, und die Instruktion AND # 7F setzt Bit 7 auf Ø. Dieses Bit ist das von der Tastatur, oder richtiger, vom UART stammende Paritätsbit. In Kapitel 12 war zu lesen, daß wir daran nicht interessiert sind.

5

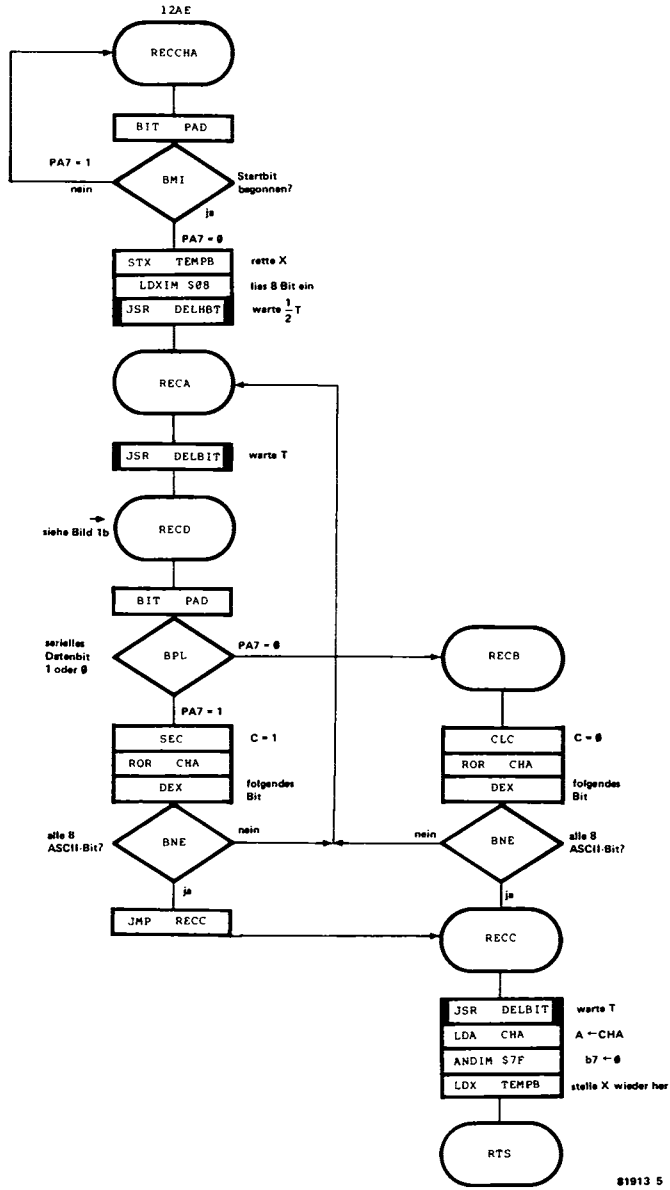


Bild 5. Subroutine RECCHA hat die Aufgabe, die von der Peripherie kommenden seriellen ASCII-Kodes in parallele Signale umzuwandeln und diese in den Akku zu setzen.

Das Label RECD ist in Bild 5 eingefügt, weil die Vorroutine (siehe Bild 1b und nachfolgender Punkt 6) hierher springt. In dieser Programmphase wurde das Startbit bereits erkannt. Ferner muß im X-Register vor dem Sprung nach RECD der Wert Ø8 stehen.

6 An dieser Stelle setzen wir Punkt 3 fort und besprechen den übrigen Teil der PM-Vorroutine (zweite Hälfte, Bild 1b). Nachdem die Auswertung des Startbit erledigt ist und in den verschiedenen Bitzeit-Speicherplätzen der erforderliche Wert steht, wird in das X-Register die Zahl Ø8 gesetzt. Nach Ablauf einer halben Bitzeit springt das Programm zu dem Teil der Subroutine RECCHA, der für das Einlesen des auf das Startbit folgenden ASCII-Datenbit zuständig ist (siehe Bild 5 und Punkt 5). Anschließend wird geprüft, ob der empfangene ASCII-Kode der Kode 7F ist. Trifft dies zu, weil Taste RES = RUB OUT gedrückt wurde, dann beginnt beim Label LABJUN die zweite Startphase von PM. Anderenfalls folgt ein Rücksprung nach INITPR in Bild 1a.
Der Durchlauf durch den bei Label LABJUN beginnenden Programmteil schließt ein, daß die Rückmeldung "JUNIOR" ausgedruckt wird und daß der nächste von PM geschriebene Text auf einer neuen Zeile steht (CRLF). Die nähere Betrachtung der beiden Subroutinen JUNIOR und CRLF heben wir uns für einen späteren Zeitpunkt auf (Punkte 11 und 12).
In Anbetracht der Tatsache, daß das Programm auch beim Drücken der "grafischen Notbremse" (gemeint ist natürlich die BRK-Taste) zum Label LABJUN springt, kann man sich leicht vorstellen, daß unter bestimmten Voraussetzungen die Gefahr des Stack-Überlaufs besteht. Der Stackpointer wird deshalb ebenso wie die Speicherzelle SPUSER auf FF gesetzt. Damit sind wir am Endpunkt der PM-Vorroutine, dem Label RESALL angekommen. Dieses Label ist gleichzeitig der Startpunkt der PM-Hauptroutine.

7 Noch eine weitere Vorroutine von PM führt zum Label RESALL: die Routine STEP, die ebenfalls in Bild 1b dargestellt ist. Sie wird während der Rückkehr zu PM durchlaufen, nachdem der Prozessor im Step-Modus eine einzelne Instruktion ausgeführt hat. Das Gleiche gilt im allgemeinen auch für den Fall, daß ein NMI aufgetreten ist. Der NMI-Sprungvektor wurde nämlich innerhalb der Vorroutine INITPR (siehe Bild 1a) auf die Adresse des Labels STEP gerichtet.
Vorroutine STEP zeigt weitgehende Ähnlichkeit mit der SAVE-Routine des Standard-Monitors. Alle Registerinhalte werden in die bekannten Speicherplätze gerettet (siehe Kapitel 3 und 7). Den Abschluß bildet der Sprung zur Subroutine PRBUFS, deren Details später unter Punkt 10 besprochen werden. An dieser Stelle sei nur vermerkt, daß die Adresse (PCH, PCL) am Anfang einer neuen Zeile ausgedruckt wird, gefolgt von einem Zwischenraum und dem Inhalt des zu dieser Adresse gehörenden Speicherplatzes. Während des Step-Modus ist der Inhalt dieses Speicherplatzes gleich dem Opcode der nächsten auszuführenden Instruktion; sie wird nach Drücken von Taste R erledigt.
Auch nach einer BRK-Instruktion oder einem Hardware-IRQ kann man zum PM-Eingang STEP springen. In diesem Fall muß der IRQ-Sprungvektor entsprechend spezifiziert sein.

Damit haben wir die Vorroutinen von PM sowie bereits einige PM-Subroutinen näher kennengelernt. Bevor wir zur Besprechung der PM-Hauptroutine übergehen, wollen wir noch einige weitere Subroutinen betrachten. Sie stehen mit den Druck-Aktivitäten des Junior-Computers in engem Zusammenhang.

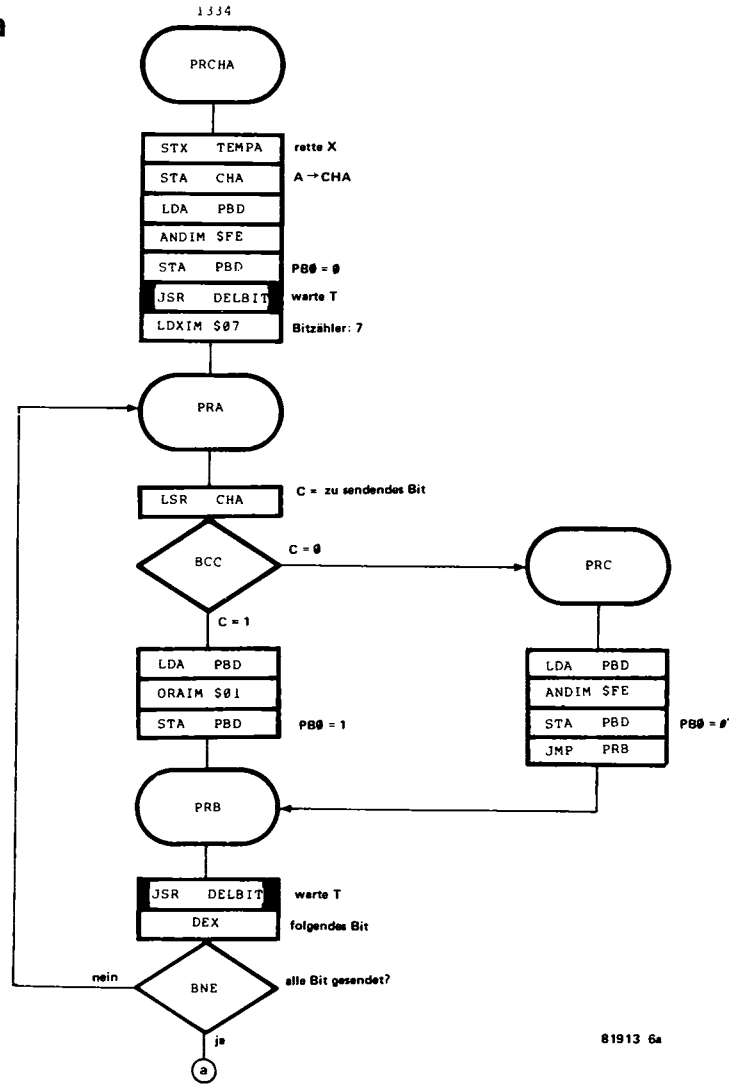
8 Das Gegenstück zu RECCHA, der elementaren Empfangssubroutine für ASCII-Zeichen, ist die **Sendesubroutine PRCHA.** Ihr Flußdiagramm geht aus den Bildern 6a und 6b hervor. PRCHA sendet einen im Akku stehenden ASCII-Kode in serieller Form an die Peripherie (Terminal, Drucker usw.).
Die Subroutine rettet zuerst den Inhalt des X-Registers in Speicherplatz TEMPA. Unmittelbar vor dem Rücksprung (RTS) wird der ursprüngliche Inhalt in das X-Register zurücktransportiert (Bild 6b), so daß sich hier durch PRCHA nichts verändert. Ebenso wie bei RECCHA spielt auch bei PRCHA der Speicherplatz CHA eine Rolle: Der Inhalt des Akkus wird in CHA kopiert und dort verschiedenen Bit-Operationen unterzogen. Durch Maskieren von Bit Ø wird PBØ logisch Ø (AND # FE, oder anders geschrieben: AND # 1111 111Ø). Danach wartet die Subroutine eine Bitzeit, bis das Startbit gesendet ist. In das X-Register, das als Bitzähler dient, wird der Wert Ø7 gesetzt.
Nun erhält die Carry-Flag den Wert des zu sendenden Bit (unmittelbar nach Label PRA in Bild 6a), indem der Prozessor sieben mal die Instruktion LSR-CHA ausführt. Portleitung PBØ wird auf das gleiche logische Signal wie die Carry-Flag gesetzt, während die Signale auf den übrigen Portleitungen PB1 . . . PB7 unverändert bleiben.
Der weitere Ablauf sieht wie folgt aus:

PRCHA zu Beginn:	Ø	b6	b5	b4	b3	b2	b1	bØ	C = X
nach X = Ø7:	Ø	Ø	b6	b5	b4	b3	b2	b1	C = bØ
nach X = Ø6:	Ø	Ø	Ø	b6	b5	b4	b3	b2	C = b1
nach X = Ø5:	Ø	Ø	Ø	Ø	b6	b5	b4	b3	C = b2
nach X = Ø4:	Ø	Ø	Ø	Ø	Ø	b6	b5	b4	C = b3
nach X = Ø3:	Ø	Ø	Ø	Ø	Ø	Ø	b6	b5	C = b4
nach X = Ø2:	Ø	Ø	Ø	Ø	Ø	Ø	Ø	b6	C = b5
nach X = Ø1:	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	C = b6

Es ist deutlich, daß Bit b7 vor der seriellen Datenübertragung Ø ist und nicht mit den anderen Bit ausgesendet wird. Weil PM vom Paritätsbit keinen Gebrauch macht, gehen nur die sieben ASCII-Bit auf die Reise.
Nun ist Bild 6b an der Reihe, in dem es um die beiden Stopbit sowie um die Abfrage der BRK-Taste geht. Die Anzahl der Stopbit, die in Speicherplatz STPBIT (siehe Bild 1a) steht, wird in das X-Register gesetzt. Anschließend wartet die Subroutine zwei Bitzeiten T ab; Portleitung PBØ liegt während dieser Zeit auf logisch Ø.
Damit ist das Senden des ASCII-Zeichens abgeschlossen. Der noch übrig bleibende Teil der Subroutine in Bild 6b steht in Zusammenhang mit der BRK-Taste. Aus Kapitel 12 (Buch 3) wissen wir, daß mit ihr das Ausdrucken von ASCII-Zeichen unterbrochen werden kann. Nach jedem vollständig gesendeten ASCII-Kode wird die BRK-Taste abgefragt. Ist sie nicht gedrückt, dann folgt nur noch die Wiederherstellung des ursprünglichen Inhalts von Register X. Bei gedrückter BRK-Taste wartet die Subroutine

zunächst das Loslassen der Taste ab. Danach folgt ein indirekter Sprung (in Bild 6b mit dem nicht offiziellen Opcode JMI bezeichnet), wobei die effektive Sprungadresse (BRK-Sprungvektor) in den Speicherplätzen BRKT ($1A7C) und BRKT + 1 ($1A7D) steht. Die effektive Sprungadresse ist gleich der Adresse des Labels LABJUN (siehe Bild 1a), so daß nach Drücken und Loslassen der BRK-Taste die Rückmeldung "JUNIOR" erscheint.

6a



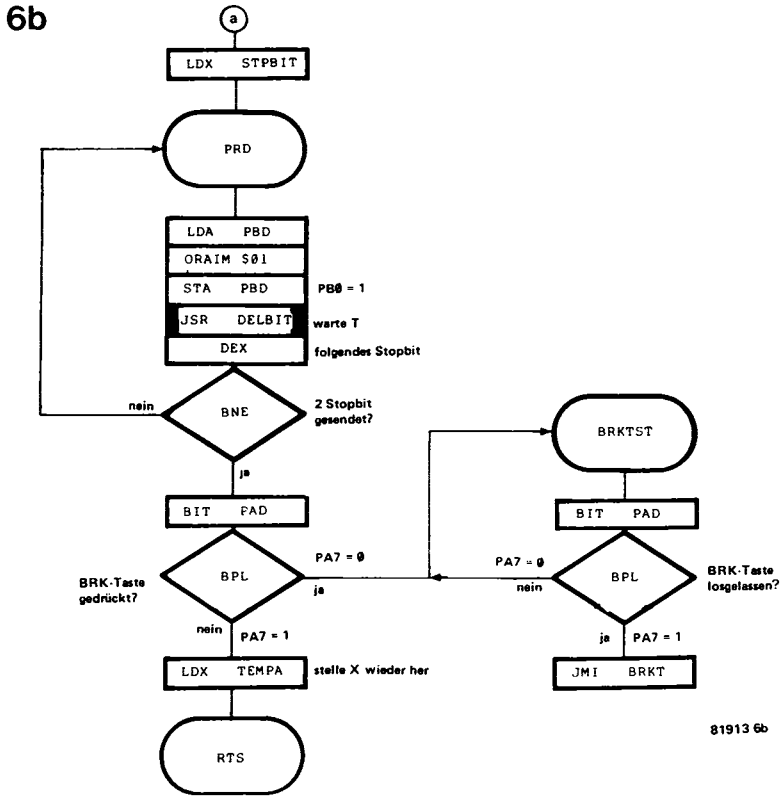


Bild 6. Als elementare Druck-Subroutine druckt PRCHA das ASCII-Zeichen, dessen Kode im Akku steht, auf dem Bildschirm oder dem Druckerpapier aus. PRCHA umfaßt die Flußdiagramme in den Bildern 6a und 6b.

9 Die **Subroutine PRBYT** sendet im Akku stehende **8-bit-Daten** an die Peripherie; sie ist in Bild 7a dargestellt. Der Akku-Inhalt wird in Form von **zwei** ASCII-Kodes auf die Reise geschickt: jedem der beiden Nibble entspricht ein ASCII-Kode. Vorher müssen jedoch die beiden Nibble in ASCII-Kodes umgewandelt werden, denn ohne Umwandlung würde beispielsweise die im Akku stehende 8-bit-Zahl $20 das Ausdrucken eines Zwischenraums und nicht des Zahlenwerts zur Folge haben. Subroutine NIBASC in Bild 7b erledigt das Notwendige:
Den Datennibbles 00 . . . 09 (hexadezimale und dezimale numerische Tasten) entsprechen die ASCII-Kodes $30 . . . $39, während der ASCII-Kode der Datennibbles 0A . . . 0F (hexadezimale numerische Tasten) gleich $41 . . . $46 ist. Dieser Zusammenhang läßt sich aus der Tabelle in Buch 3, Anhang 7 ablesen. Gehen wir davon aus, daß der Inhalt des Akkus zu

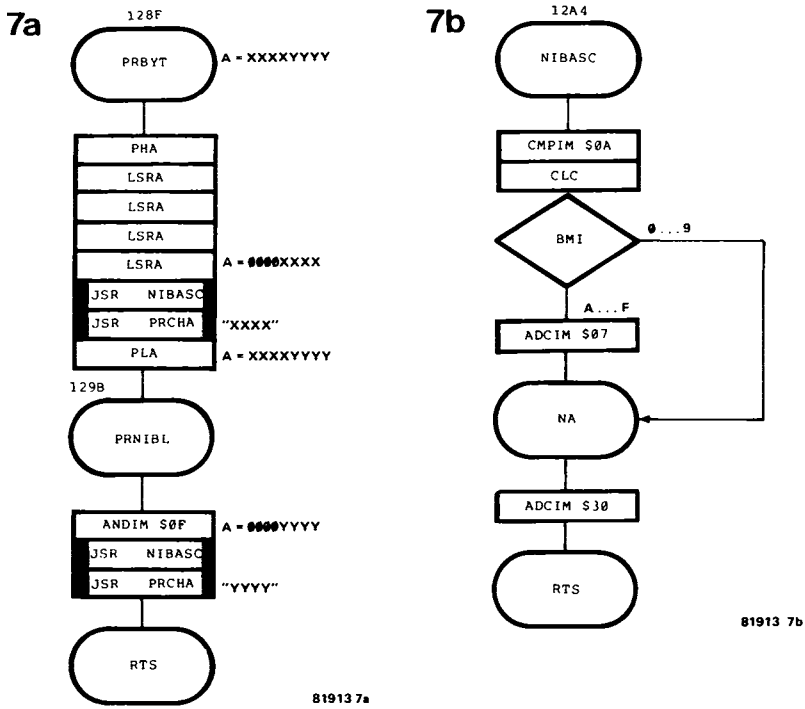


Bild 7. Das Ausdrucken von Datenbytes in Form von zwei Datennibbles ist der Zweck der Subroutine PRBYT (Bild 7a). Vorher werden die Datennibbles von Subroutine NIBASC in den entsprechenden ASCII-Kode umgewandelt.

Beginn von NIBASC gleich ØX mit X = Ø . . . F ist, dann muß zum Akku-Inhalt im ersten Fall $3Ø und im zweiten Fall $37 addiert werden. Im ersten Fall erhalten wir den ASCII-Kode der Ziffern Ø . . . 9, im zweiten den der Buchstaben A . . . F. Genau dies geschieht beim Durchlaufen von NIBASC. Mit den Instruktionen CMP und BMI wird zwischen den beiden Zahlengruppen Ø . . . 9 und A . . . F unterschieden. Die Zahlen der ersten Gruppe werden um $3Ø und die Zahlen der zweiten Gruppe um $Ø7 und $3Ø erhöht.
Zurück zur Subroutine PRBYT in Bild 7a. Sie beginnt mit dem "Einfrieren" des Akku-Inhalts (PHA) und schiebt danach die dort stehenden Bit um vier Stellen nach rechts. Das ursprüngliche linke Nibble wird hierdurch zum rechten Nibble, das linke Nibble ist jetzt ØØ.
Es folgt der Sprung nach NIBASC (Bild 7b): das linke Nibble wird in den zugehörigen ASCII-Kode umgewandelt. Nach dem Rücksprung sorgt Subroutine PRCHA (Bild 6a/b, Punkt 8 dieses Kapitels) für das Ausdrucken des betreffenden Zeichens. Schließlich erhält der Akku seinen ursprünglichen Inhalt zurück (PLA).

8

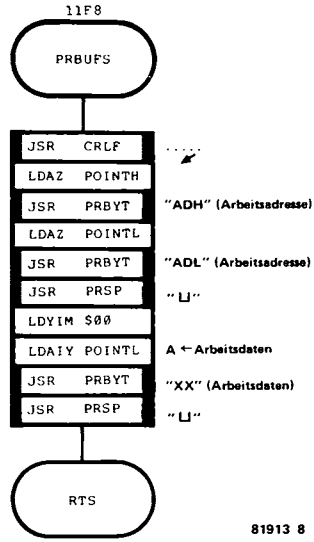


Bild 8. Subroutine PRBUFS druckt Arbeitsadressen mit den in den zugehörigen Speicherplätzen stehenden Daten (Arbeitsdaten) aus.

Wir sind beim Label PRNIBL angekommen, das die Startmarke für das Ausdrucken des rechten Akku-Nibble darstellt. Dieses Label kann man immer dann anspringen, wenn nur ein einziges Nibble (nämlich das rechte) ausgedruckt werden soll. Durch Maskieren des linken Nibble ist der Akku-Inhalt in der für das Durchlaufen der Subroutinen NIBASC und PRCH notwendigen Form. Den Schlußpunkt bildet wie immer der RTS, der Rücksprung zur Hauptroutine.

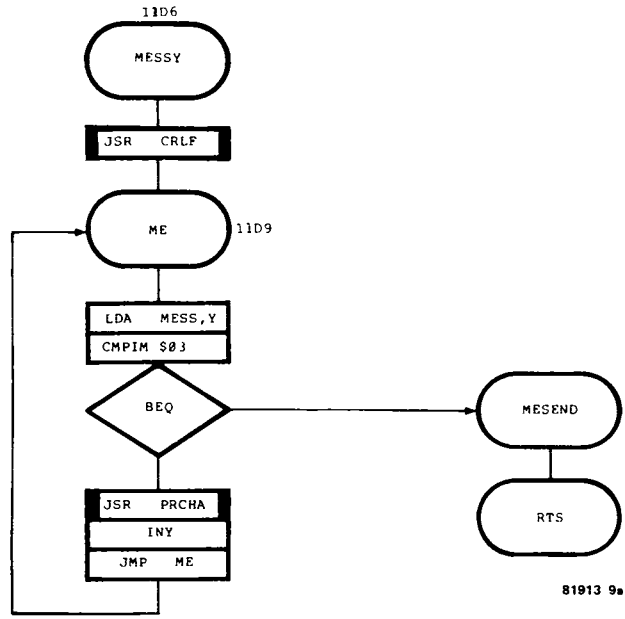
10 Die **Subroutine PRBUFS** in Bild 8 druckt die Adresse aus, die wir **"Arbeitsadresse"** genannt haben. Diese Adresse steht in den Puffern POINTH (linkes Adreßbyte) und POINTL (rechtes Adreßbyte). Gleichzeitig wird der Inhalt des Speicherplatzes ausgedruckt, dessen Adresse in POINTH, POINTL steht; dort befinden sich die **Arbeitsdaten.** Bild 8 ist schnell erklärt: Subroutine PRBUFS startet mit dem Beginn einer neuen Zeile (CRLF, siehe Punkt 12). Dann druckt die Subroutine PRBYT, die im vorangegangenen Abschnitt (Punkt 9) besprochen wurde, das linke Byte ADH und das rechte Byte ADL der Arbeitsadresse aus. Es folgt ein Zwischenraum (PRSP, siehe Punkt 12), der das Lesen der ausgedruckten Informationen erleichtert. Der Inhalt des Speicherplatzes, dessen Adresse in POINTH, POINTL steht, wird mit LDA-(POINTL),Y in den Akku geladen, wobei Y = $00 ist. Zum Schluß werden die beiden Datennibble zusammen mit einem angehängten Leerraum ausgedruckt (PRBYT und PRSP).

11 **Subroutine MESSY** in Bild 9a dient zum Ausdrucken von Zeichenketten (Strings), zum Beispiel von Rück- und Fehlermeldungen wie "JUNIOR" und "WHAT?" Texte wie "SA:" und "EA:" gehören ebenfalls dazu. Der zu druckende Text ist in Form von ASCII-Kodes in aufeinanderfolgenden Speicherplätzen der Tabelle MESS gespeichert, die die Adressen 13BD . . . 141D umfaßt (siehe Listing am Schluß dieses Buchs). Die Anzahl der Speicherplätze, die zwischen dem Tabellenanfang (Adresse 13BD) und einem bestimmten Tabellenplatz liegen, steht im Y-Register. Jeder Text wird von einem EOT-Zeichen (ASCII-Kode $Ø3) abgeschlossen. Soll ein Text ausgedruckt werden, so wird der Y-Anfangswert des betreffenden Textes in das Y-Register gesetzt. Danach inkrementiert man den Inhalt von Y so oft, bis das EOT-Zeichen erreicht ist.
Subroutine MESSY (Bild 9a) beginnt mit einer anderen Subroutine: CRLF (siehe Punkt 12) sorgt für einen neuen Zeilenanfang. Nach Passieren des Labels ME lädt die Instruktion LDA-MESS,Y den Kode des auszudruckenden Zeichens in den Akku. Der Anfangswert von Y muß bereits vor dem Sprung nach MESSY in diesem Register stehen. Es folgt ein Vergleich mit dem EOT-Zeichen $Ø3. Fällt er positiv aus, so ist die Subroutine beendet (RTS). Anderenfalls wird der im Akku stehende ASCII-Kode mit PRCHA ausgedruckt, Y wird inkrementiert, und das Spiel kann neu beginnen (JMP-ME). Unter bestimmten Voraussetzungen durchläuft der Prozessor nur den Teil von MESSY, der beim Label ME beginnt. In diesem Fall springt die Schreibposition nicht zu einem neuen Zeilenanfang.
Ein Beispiel für die Anwendung von MESSY ist in Bild 9b dargestellt: es sind die **Subroutinen JUNIOR, EDITOR und ASSEM.** Wie bereits erwähnt, wird der Anfangswert des Textes ("JUNIOR", "EDITOR" oder "ASSEMBLER") vor dem Sprung nach MESSY in das Y-Register gesetzt. Nach dem Rücksprung sorgt Subroutine CRLF dafür, daß weitere Druckaktivitäten auf einer neuen Zeile beginnen. Übrigens: Die Subroutinen EDITOR und ASSEM gehören nicht zum Systemprogramm PM.

12 Die **Subroutinen CRLF und PRSP** in Bild 10 werden überdurchschnittlich oft aufgerufen, denn diese grafischen Subroutinen tragen maßgeblich zur Formgebung und somit zur Übersichtlichkeit der ausgedruckten Informationen bei. Ihr Aufbau ist schnell erklärt:
Der ASCII-Kode des gewünschten grafischen Kommandos wird in den Akku geladen, anschließend wird der Akku-Inhalt mit PRCHA "ausgedruckt". Das Kommando CR (Carriage Return = Zurück zum Anfang der **gleichen** Zeile) hat den ASCII-Kode ØD, und der ASCII-Kode des Kommandos LF (Line Feed = Gehe zur **gleichen** horizontalen Position in der **nächsten** Zeile) lautet ØA. Auch das Drucken eines Zwischenraums (PRSP) ist unkompliziert: In den Akku wird der Kode 2Ø geladen; danach folgt ein Sprung zum Label CLEND, so daß Subroutine PRCHA den Zwischenraum ausdruckt.

13 Die **Subroutine HEXNUM** (Bild 11) spielt dann eine Schlüsselrolle, wenn es zu entscheiden gilt, ob ein vom Junior-Computer empfangener ASCII-Kode von einer gedrückten numerischen Taste oder von einer anderen Taste stammt. Treffen von der Tastatur numerische Daten ein, so werden diese nach der Rückwandlung aus dem ASCII-Kode in die

9a



9b

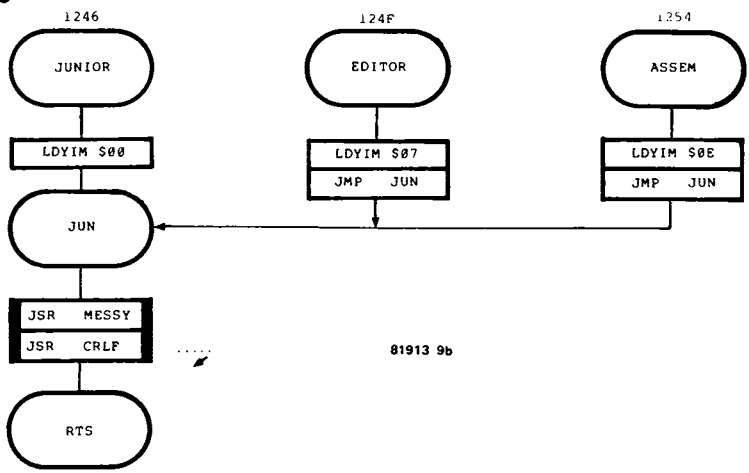


Bild 9. Rück- und Fehlermeldungen werden von Subroutine MESSY (Bild 9a) ausgedruckt. Dazu gehören auch Texte wie zum Beispiel "SA:" und "EA:". Die Subroutinen JUNIOR, EDITOR und ASSEM (Bild 9b) lassen mit Hilfe von MESSY die Worte "JUNIOR", "EDITOR" und "ASSEMBLER" auf dem Bildschirm bzw. auf Papier erscheinen. Die beiden zuletzt genannte Rückmeldungen treten nicht bei PM, sondern bei anderen Systemprogrammen auf.

10

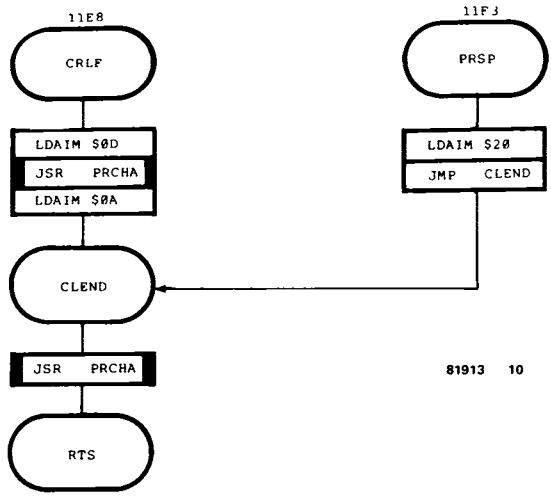


Bild 10. Die Subroutinen CRLF und PRSP gehören zur Gruppe der grafischen Subroutinen. Während CRLF die Schreibposition an den Anfang der nächsten Zeile führt, druckt PRSP einen Leerraum (Zwischenraum) aus.

Datenpuffer INL/INH gesetzt und von dort aus weiterverarbeitet. Handelt es sich nicht um numerische Daten, so druckt der Computer die Frage "WHAT?" aus.
Bevor wir HEXNUM im Detail besprechen, wollen wir einen Seitenblick auf Subroutine ASHETT in Bild 12 werfen. Diese Subroutine ist der Teil von HEXNUM, der die numerischen Daten erkennt und die Rückwandlung aus dem ASCII-Kode vornimmt. ASHETT geht davon aus, daß der empfangene ASCII-Kode im Akku steht. Akzeptiert werden die ASCII-Kodes $3Ø . . . $39 (Tasten Ø . . . 9) und $41 . . . $46 (Tasten A . . . F); alle übrigen ASCII-Kodes werden von ASHETT zurückgewiesen. Die Unterscheidung zwischen zugelassenen und nicht zugelassenen Kodes wird mit vier aufeinanderfolgenden CMP-Instruktionen und vier daran anschließenden BMI-Instruktionen getroffen. Wenn ein nicht zugelassener Kode (eines nicht numerischen Zeichens) festgestellt wurde, folgt der Rücksprung (RTS) aus der Subroutine. Hierbei ist die N-Flag 1, die Z-Flag ist folglich Ø. Der ASCII-Kode eines zugelassenen Zeichens muß zuerst in ein Datennibble umgewandelt werden. Dies geschieht innerhalb von ASHETT nach dem Label VALIT.
Bei den Daten ØØ . . . Ø9 kann die Umwandlung einfach durch Nullsetzen des linken Akku-Nibble (AND #ØF) vorgenommen werden, während bei den Daten ØA . . . ØF zuvor noch die Zahl Ø9 zum Akku-Inhalt addiert werden muß. Da bei der Rückkehr aus ASHETT nach Erkennen eines zugelassenen ASCII-Kodes das linke Akku-Nibble gleich Ø und das rechte stets ungleich Ø ist, sind die N-Flag und die Z-Flag in diesem Fall Ø.
Zurück zur Subroutine HEXNUM in Bild 11. Sie beginnt mit ASHETT

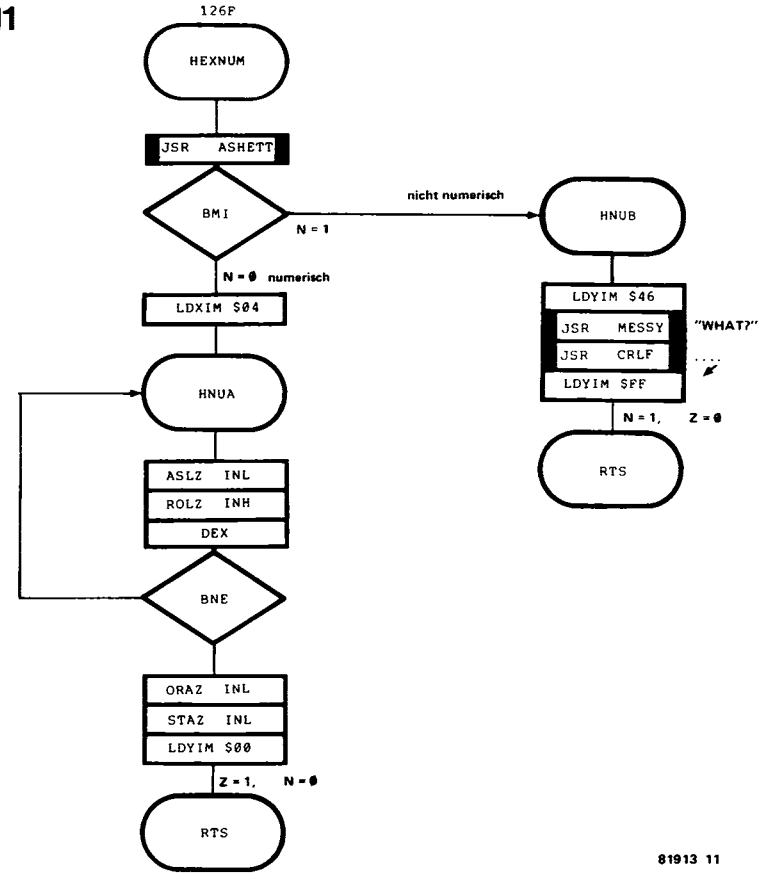


Bild 11. HEXNUM ist der Name der Subroutine, die die Eingabe von numerischen Daten verarbeitet. Das empfangene ASCII-Zeichen wird in ein Datennibble umgewandelt und in die Datenpuffer INH/INL gesetzt.

(Bild 12), darauf folgt ein BMI. Ein nicht numerischer ASCII-Kode führt zum Label HNUB, dem Startpunkt für die Fehlermeldung "WHAT?". Zuerst wird das Y-Register mit dem dafür notwendigen Wert 46 geladen, dann springt das Programm nach MESSY (siehe Punkt 11 und Bild 9a). Spätere Druck-Aktivitäten beginnen auf einer neuen Zeile (CRLF), und vor dem Rücksprung (RTS) erhält Y noch den Wert FF. Die N-Flag ist deshalb 1, die Z-Flag ist Ø.
Wenn der empfangene ASCII-Kode zu den Kodes der numerischen Zeichen gehört, wird das daraus erzeugte Datennibble zu den vorher empfangenen und bereits in den Datenpuffern INH/INL stehenden Nibbles hinzugefügt: Sämtliche Bit werden vier mal nach links geschoben und nach links

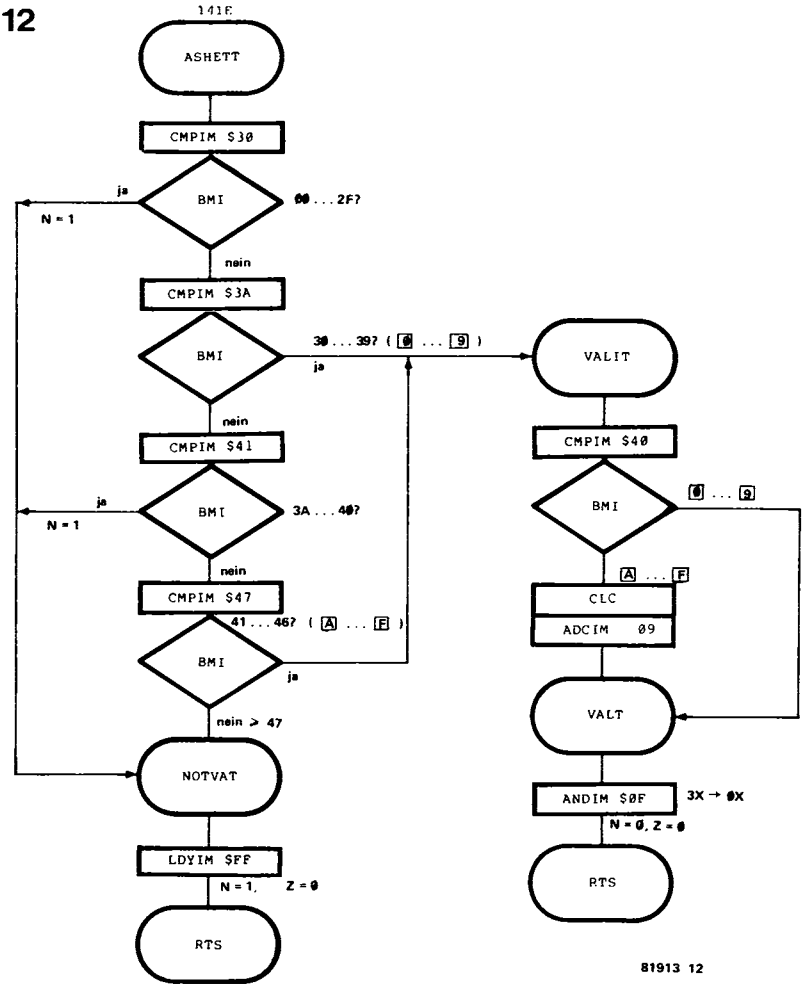


Bild 12. Subroutine ASHETT klärt innerhalb von HEXNUM (Bild 11) die Frage, ob eine der numerischen Tasten 0 . . . F oder eine andere Taste gedrückt wurde. Letzteres führt zur Fehlermeldung "WHAT?".

gedreht. Da dieses Einschiebeverfahren bereits beim Standard-Monitor angewendet und dort auch ausführlich besprochen wurde (siehe Buch 2), sollen hier einige Stichpunkte genügen. Nach Einschieben eines empfangenen Nibble sehen die Inhalte der beiden Datenpuffer-Speicherzellen wie folgt aus:

- Das rechte Nibble von INL ist das gerade empfangene Datennibble.
- Das linke Nibble von INL ist das frühere rechte Nibble von INL.

- Das rechte Nibble von INH ist das frühere linke Nibble von INL.
- Das linke Nibble von INH ist das frühere rechte Nibble von INH.
- Das frühere linke Nibble von INH ist nicht mehr vorhanden.

Mit anderen Worten: Der Inhalt von INH und INL stammt stets von den vier zuletzt gedrückten numerischen Tasten, es sei denn, daß die Puffer zwischenzeitlich auf Null gesetzt wurden (Subroutine RESIN, siehe Punkt 14). Waren weniger als vier numerische Tasten gedrückt worden, dann sind die nicht benutzten Nibble von IHN/INL Null.
Unmittelbar vor dem Rücksprung aus diesem Zweig von HEXNUM erhält das Y-Register den Wert ØØ. Im Gegensatz zur Situation nach Empfang eines nicht numerischen Zeichens (rechter Zweig von HEXNUM in Bild 11) ist daher jetzt die Z-Flag 1, und die N-Flag ist Ø.

14 Wir kommen nun zur **Hauptroutine von PM** und zu den noch verbleibenden PM-Subroutinen. Die Bilder 13a, b und c zeigen das detaillierte Flußdiagramm der PM-Hauptroutine. Ein zentraler Punkt ist das Label RESALL, das links oben in Bild 13a gezeichnet ist. Auf diesen "Kilometerstein" folgen die **Subroutine RESPAR** (Bild 14a) und die **Subroutine RESIN** (Bild 14b). Während RESPAR die beiden Adreßpuffer PARA und PARB (beide 16 bit) auf Null setzt, bewirkt RESIN das Gleiche bei den Datenpuffer INH und INL. Beides geschieht immer dann, wenn der Inhalt dieser Puffer entbehrlich geworden ist: nach dem Ende einer Tastenaktion.
Nun stoßen wir auf das Label READCH, das ebenfalls eine wichtige Markierung im Ablauf der PM-Hauptroutine darstellt. Wichtig ist dieses Label für die Datenverarbeitung, beispielsweise für die Eingabe von Arbeitsadressen oder Arbeitsdaten. Unmittelbar nach dem Label READCH springt PM zur Subroutine RECCHA (siehe Bild 5 und Punkt 5): der Junior-Computer wartet (erneut) auf das Drücken einer Taste. Dann beschäftigt sich die PM-Hauptroutine mit der Beantwortung folgender Fragen: Wurde Taste X gedrückt? Wenn ja, so führe die X-Tastenroutine aus. Oder ist Taste Y die gedrückte Taste? Wenn ja, so führe die Y-Tastenroutine aus. Oder war es die Z-Taste? Dann führe die Z-Tastenroutine aus, und so weiter.
Übrigens: Die "Meilensteine" der PM-Hauptroutine, nämlich RESALL (Puffer löschen) und READCH (gedrückte Taste identifizieren), werden während der Ausführung der zu den Tasten M, G und S gehörenden Routinen nicht angesprungen. Bei diesen Tastenroutinen ist das Warten auf das Drücken von Tasten (Eingabe von Parametern) Bestandteil der Tastenroutinen selbst. Mit anderen Worten: Von Subroutine RECCHA macht PM außer nach dem Label READCH auch an anderen Stellen Gebrauch.

15 Die **Plustastenroutine** beginnt in Bild 13a beim Label PLU. In Kapitel 12 war zu lesen, daß beim Drücken der Plustaste die Arbeitsadresse um eine erhöht wird und die neue Arbeitsadresse zusammen mit den zugehörigen Daten auf einer neuen Zeile erscheint. Dazu springt das Programm zuerst nach **Subroutine INCPNT**, die in Bild 15a gezeichnet ist. Auf die Arbeitsadresse zeigt der Adreßpointer (POINTH, POINTL). Um den Adreßpointer zu inkrementieren (INCPNT, Bild 15a), wird die inzwischen wohlbekannte Methode angewendet. Es folgt das Ausdrucken

der neuen Arbeitsadresse zusammen mit den zugehörigen Daten (Subroutine PRBUFS; siehe Punkt 10 und Bild 8). Den Abschluß bildet der Rücksprung zum Label RESALL.

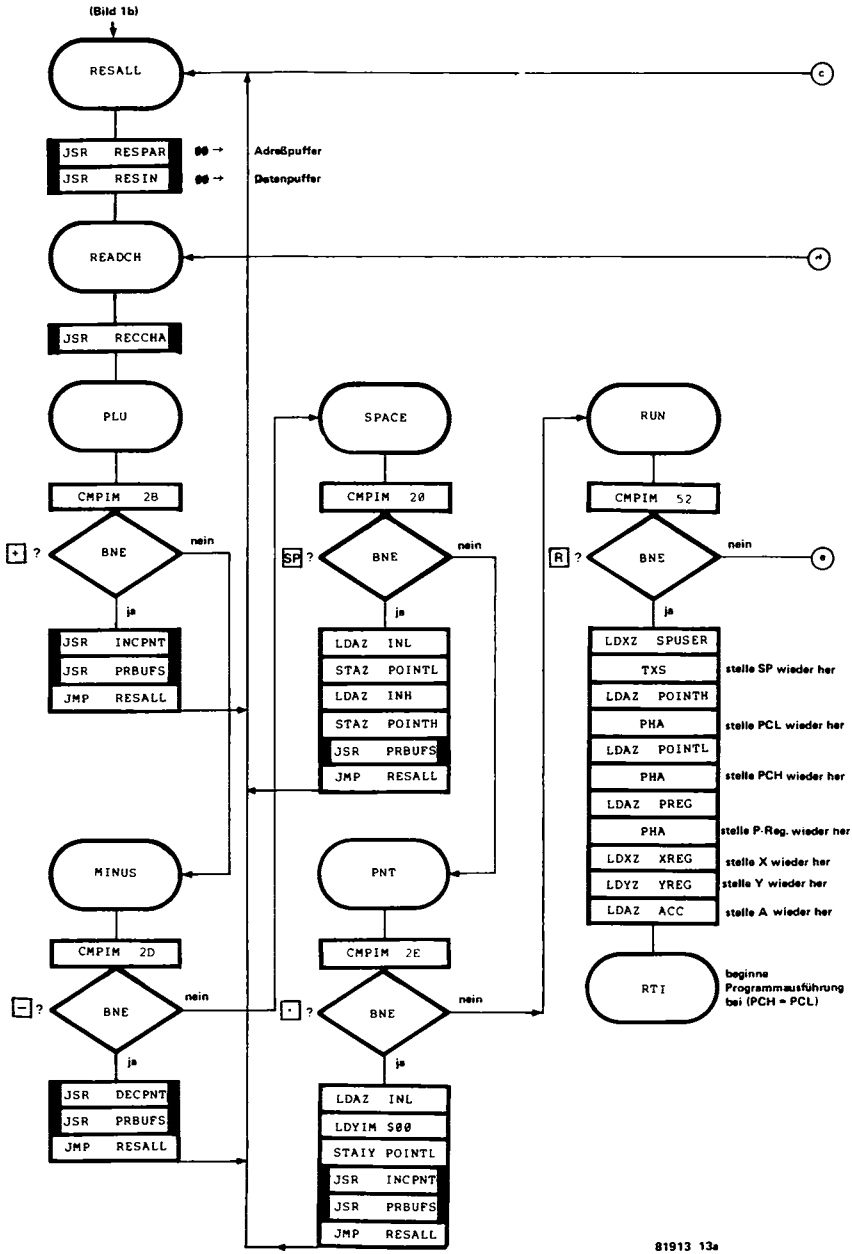
16 Die **Minustastenroutine** besteht aus den Instruktionen, die in Bild 13a auf das Label MINUS folgen. Die Arbeitsadresse muß hier um eine herabgesetzt werden; die neue Arbeitsadresse ist zusammen mit ihrem Inhalt auf einer neuen Zeile auszudrucken. Das geschieht ähnlich wie bei der unter Punkt 15 beschriebenen Plustastenroutine. Hier wird jedoch statt INCPNT die **Subroutine DECPNT** (Bild 15b) angesprungen: sie setzt Adreßpointer (POINTH, POINTL) um eins herab. Im übrigen ist die Minustastenroutine mit der Plustastenroutine identisch.

17 Zur **Leertastenroutine** gehören die Instruktionen in Bild 13a, die unmittelbar an das Label SPACE anschließen. Durch Drücken der Leertaste soll eine Arbeitsadresse ausgedruckt werden, die zuvor über die Tastatur (eine bis vier numerische Tasten) eingegeben wurde und nun in den Datenpuffern INL und INH steht. Hierzu wird der Inhalt von INL in POINTL (rechtes Byte der Arbeitsadresse) und der Inhalt von INH in POINTH (linkes Byte der Arbeitsadresse) kopiert. Die Arbeitsadresse kann nun ausgedruckt werden; hierzu springt das Programm zur Subroutine PRBUFS (Bild 8 und Punkt 10). Schlußpunkt ist wieder der Rücksprung nach RESALL.

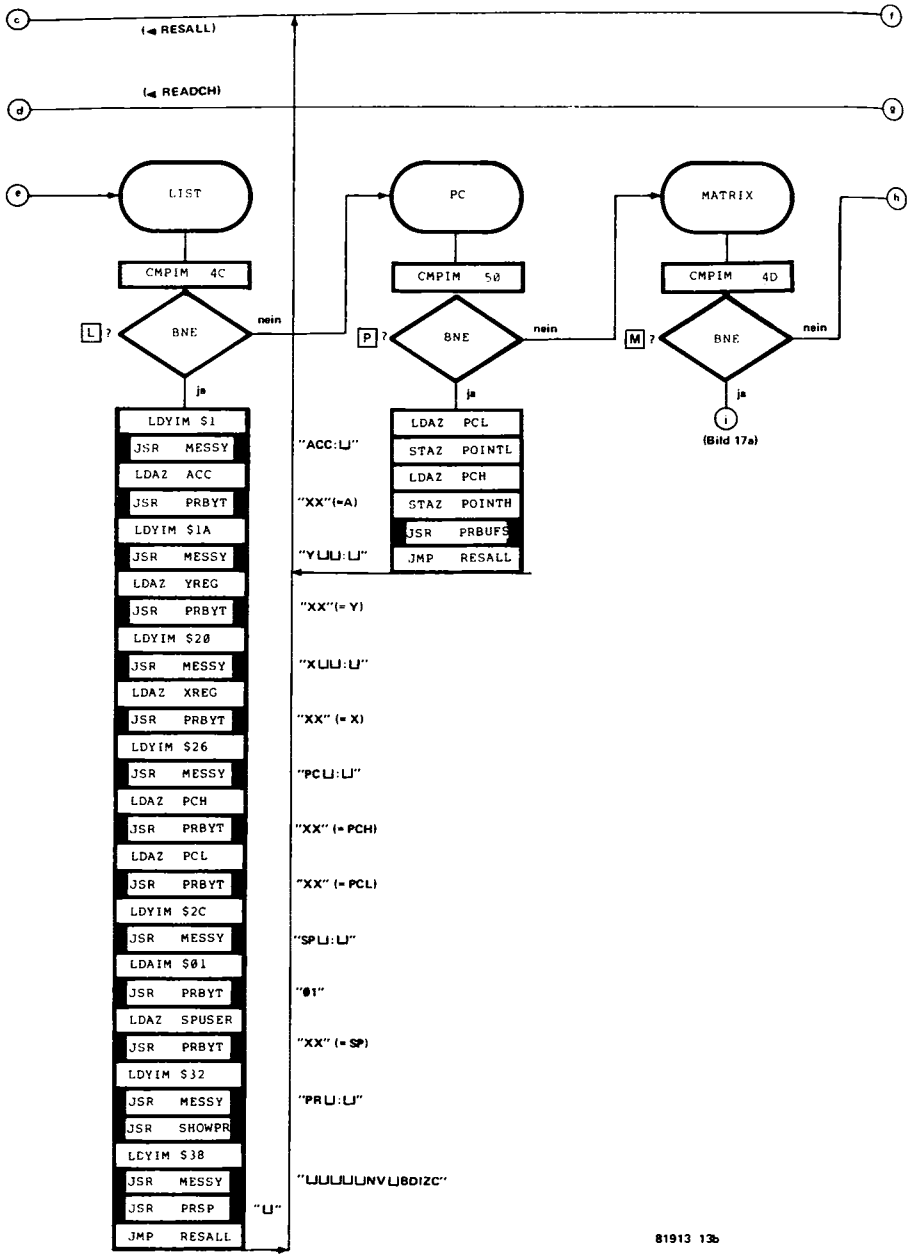
18 **Die Punkttastenroutine** umfaßt die in Bild 13a auf das Label PNT folgenden Instruktionen. Es müssen Daten unter einer Arbeitsadresse in den Speicher geladen werden, wobei die Arbeitsadresse bereits vorher mit der Leertaste, der Plus- oder der Minustaste bestimmt wurde. Am Anfang steht das Laden des Akku mit dem Inhalt des Datenpuffers INL. INH wird hier nicht benötigt, da nur ein einzelnes Datenbyte (8 bit!) transportiert werden soll. Das Datenbyte wurde vorher über eine oder zwei numerische Tasten eingegeben. Drückt man nacheinander mehr als zweimal eine numerische Taste, dann werden nur die letzten beiden Tastenbetätigungen ausgewertet. Nachdem die Arbeitsdaten in den Akku gesetzt sind, wandert der Akku-Inhalt durch die Instruktion STA-(POINTL),Y in den zur Arbeitsadresse gehörenden Speicherplatz. Danach wird die nächste Dateneingabe vorbereitet, indem Subroutine INCPNT die Arbeitsadresse um eine erhöht und Subroutine PRBUFS diese neue Arbeitsadresse mit ihrem Inhalt ausdruckt.

19 **Die R-Tastenroutine** (sämtliche Instruktionen nach Label RUN in Bild 13a) stimmt exakt mit der Tastenroutine überein, die zur GO-Taste des Standard-Monitors gehört (siehe Buch 2, S. 118, Bild 5a). Alle Register werden mit den Daten geladen, die dort vor dem letzten Sprung nach PM standen (STEP-Eingang; siehe Bild 1b). Der Rücksprung RTI am Schluß der R-Tastenroutine bewirkt, daß das Programm bei der im Program Counter stehenden Adresse (PCH, PCL) startet, oder daß beim Arbeiten im Step-Modus die nächste Instruktion ausgeführt wird. Der Opcode der Instruktion befindet sich im Speicherplatz mit der Adresse (PCH, PCL).

13a


(Bild 1b)
RESALL
JSR RESPAR
JSR RESIN
Adreßpuffer
Datenpuffer
READCH
JSR RECCHA
PLU
CMPIM 2B
+ ?
BNE
nein
ja
JSR INCPNT
JSR PRBUFS
JMP RESALL
MINUS
CMPIM 2D
- ?
BNE
nein
ja
JSR DECPNT
JSR PRBUFS
JMP RESALL
SPACE
CMPIM 20
SP ?
BNE
nein
ja
LDAZ INL
STAZ POINTL
LDAZ INH
STAZ POINTH
JSR PRBUFS
JMP RESALL
PNT
CMPIM 2E
. ?
BNE
nein
ja
LDAZ INL
LDYIM $00
STAIY POINTL
JSR INCPNT
JSR PRBUFS
JMP RESALL
RUN
CMPIM 52
R ?
BNE
nein
ja
LDXZ SPUSER
TXS
stelle SP wieder her
LDAZ POINTH
PHA
stelle PCL wieder her
LDAZ POINTL
PHA
stelle PCH wieder her
LDAZ PREG
PHA
stelle P-Reg. wieder her
LDXZ XREG
stelle X wieder her
LDYZ YREG
stelle Y wieder her
LDAZ ACC
stelle A wieder her
RTI
beginne Programmausführung bei (PCH = PCL)
81913 13a

13b


(◄ RESALL)
(◄ READCH)
LIST
CMPIM 4C
L ?
BNE
nein
ja
PC
CMPIM 50
P ?
BNE
nein
ja
MATRIX
CMPIM 4D
M ?
BNE
nein
ja
(Bild 17a)
LDYIM $1
JSR MESSY
"ACC:␣"
LDAZ ACC
JSR PRBYT
"XX"(=A)
LDYIM $1A
JSR MESSY
"Y␣␣:␣"
LDAZ YREG
JSR PRBYT
"XX"(= Y)
LDYIM $20
JSR MESSY
"X␣␣:␣"
LDAZ XREG
JSR PRBYT
"XX" (= X)
LDYIM $26
JSR MESSY
"PC␣:␣"
LDAZ PCH
JSR PRBYT
"XX" (= PCH)
LDAZ PCL
JSR PRBYT
"XX" (= PCL)
LDYIM $2C
JSR MESSY
"SP␣:␣"
LDAIM $01
JSR PRBYT
"01"
LDAZ SPUSER
JSR PRBYT
"XX" (= SP)
LDYIM $32
JSR MESSY
"PR␣:␣"
JSR SHOWPR
LDYIM $38
JSR MESSY
"␣␣␣␣␣NV␣BDIZC"
JSR PRSP
"␣"
JMP RESALL
LDAZ PCL
STAZ POINTL
LDAZ PCH
STAZ POINTH
JSR PRBUFS
JMP RESALL
81913 13b

13c

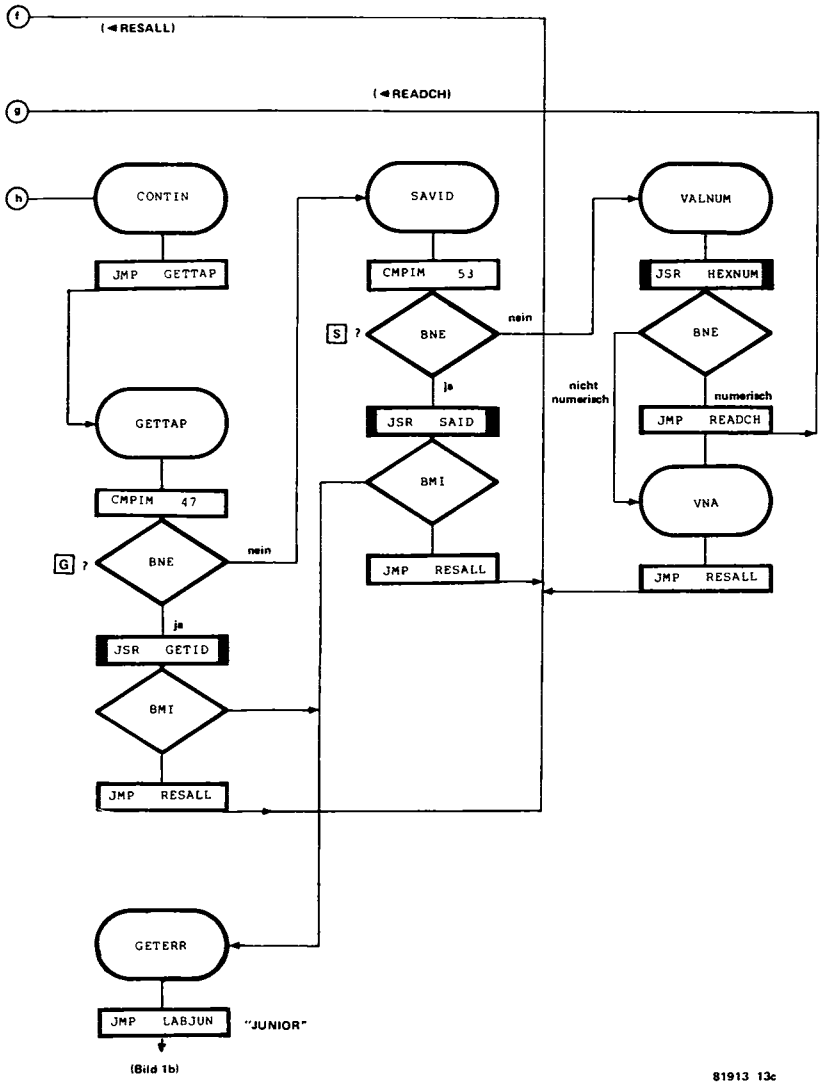


Bild 13. Dies ist die PM-Hauptroutine; ihr Flußdiagramm erstreckt sich über die Bilder 13a, 13b und 13c. Wegen der Vielzahl der zu Tastenroutine M gehörenden Instruktionen wurde diese Routine separat in Bild 17a und 17b gezeichnet.

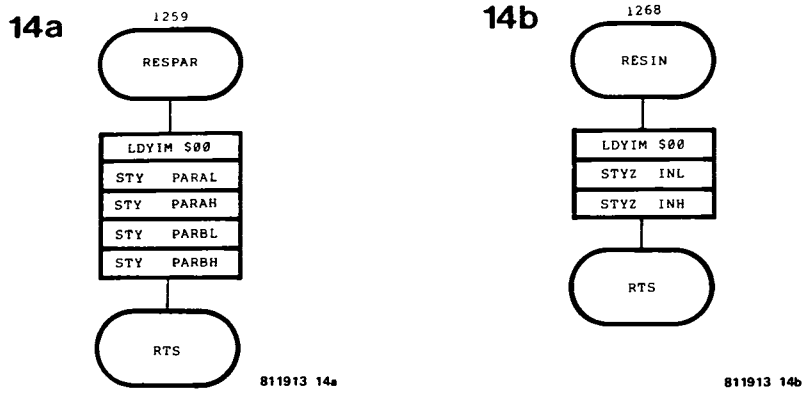


Bild 14. Die Subroutine RESPAR (Bild 14a) löscht die zum Speichern der ersten und letzten Adresse dienenden Adreßpuffer PARAH/PARAL und PARBH/PARBL, während RESIN (Bild 14b) die Datenpuffer INH und INL auf Null setzt.

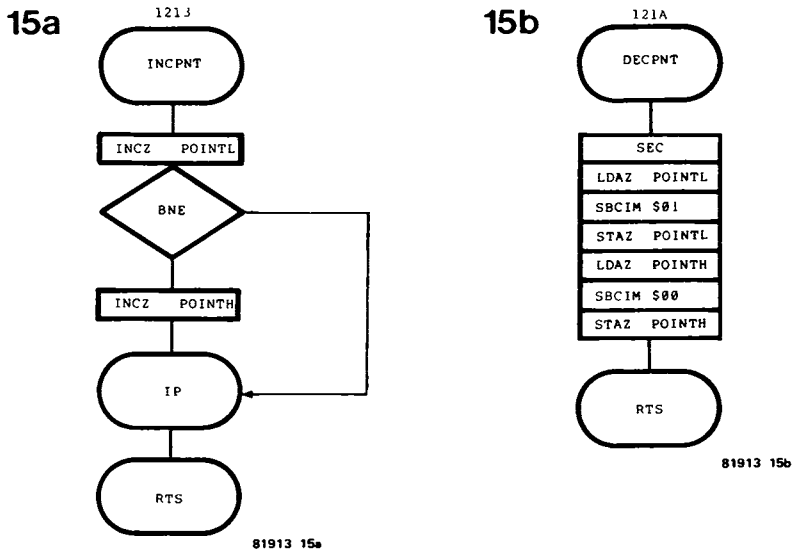


Bild 15. Der Adreßpointer POINT wird von Subroutine INCPNT in Bild 15a um eins erhöht (inkrementiert), von Subroutine DECPNT (Bild 15b) wird er um eins herabgesetzt (dekrementiert).

20 Bevor wir die L-Tastenroutine besprechen können, müssen wir zuerst eine dort eingebaute, spezielle Subroutine betrachten: **die Subroutine SHOWPR** in Bild 16. Sie sorgt dafür, daß der Inhalt des P-Registers bitweise ausgedruckt wird.
SHOWPR beginnt mit dem Kopieren von Speicherplatz PREG, in den der Inhalt des P-Registers gerettet wurde (dort stehen die Flags), nach Speicherplatz PRTEMP. Das als Bitzähler verwendete X-Register erhält den Anfangswert Ø8. Welche Flags mit den einzelnen Bit von Speicherplatz PREG (und nun auch von PRTEMP) identisch sind, kann man dem Programmiermodell der 6502-CPU in Buch 1, Seite 61, entnehmen.
Nach Label SPRA von SHOWPR wird mit der Instruktion ASL-PRTEMP acht mal nacheinander das siebente Bit von PRTEMP in das Carry-Bit eingeschoben. Abhängig davon, ab das Carry-Bit 1 oder Ø ist, druckt Subroutine PRNIBL (siehe Punkt 9 und Bild 7a, untere Hälfte) eine 1 oder

16

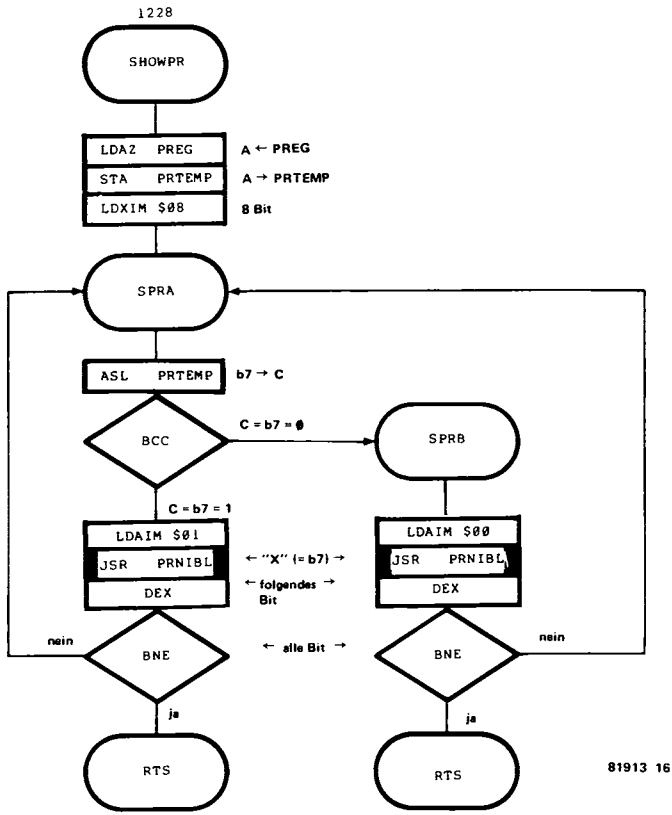


Bild 16. Subroutine SHOWPR, die von der L-Tastenroutine aufgerufen wird, druckt den Inhalt des P-Registers bitweise aus. Dort stehen nämlich die einzelnen Flags, die unter anderem als Verzweigungskriterien dienen.

eine Ø aus. Es folgen Vorbereitungen für das Ausdrucken des nächsten Bit, sofern noch ein Bit vorhanden ist und Verzweigung BNE deshalb auf Springen steht.
Hier noch eine Übersicht über den Gesamtablauf:
X = Ø8: N-Flag ausgedruckt
X = Ø7: V-Flag ausgedruckt
X = Ø6: willkürliches Bit ausgedruckt
X = Ø5: BRK-Flag ausgedruckt
X = Ø4: D-Flag ausgedruckt
X = Ø3: I-Flag ausgedruckt
X = Ø2: Z-Flag ausgedruckt
X = Ø1: C-Flag ausgedruckt

21 Aufgabe der **L-Tastenroutine** ist es, die Inhalte der CPU-Register in übersichtlicher Form auszudrucken. Diese Tastenroutine umfaßt eine größere Anzahl von Instruktionen, zu denen zahlreiche Sprünge nach grafischen Subroutinen gehören. In Bild 13b ist die beim Label LIST beginnende L-Tastenroutine hieran leicht erkennbar. Die Besprechung dieser Subroutine können wir kurz halten, wenn wir das in Buch 3, Seite 145 angeführte Beispiel für die Anwendung von Tastenfunktion L zur Hand nehmen. Die während der einzelnen Phasen ausgedruckten Texte und Zwischenräume sind in Bild 13b in Anführungszeichen gesetzt. Diese Anführungszeichen werden nicht ausgedruckt! Alle innerhalb der L-Tastenroutine vorkommenden Subroutinen wurden bereits besprochen: MESSY unter Punkt 11, PRBYT unter Punkt 9, SHOWPR unter dem vorangegangenen Punkt 20 und PRSP unter Punkt 12.

22 **Die P-Tastenroutine** besteht aus den sechs Instruktionen unterhalb des Labels PC in Bild 13b. Beim Drücken der Taste P wird die Arbeitsadresse gleich der im Programmzähler stehenden Adresse (PCH, PCL) gesetzt und ausgedruckt. Maßgebend ist der Inhalt des Programmzählers PC unmittelbar vor dem letzten Sprung nach PM über seinen STEP-Eingang (vergleiche Punkt 7). Die Funktion der P-Taste bei PM ist mit der Funktion von Taste PC des Standard-Monitors identisch. Die Inhalte der Speicherplätze PCL und PCH werden in POINTL bzw. POINTH kopiert. Anschließend druckt Subroutine PRBUFS die so entstandene neue Arbeitsadresse aus.

23 **Die M-Tastenroutine** veranlaßt das Ausdrucken eines Hexdump. Diese Tastenroutine, die beim Label MATRIX in Bild 13b beginnt und deren Hauptteile in den Bildern 17a und 17b gezeichnet sind, umfaßt verhältnismäßig viele Instruktionen. Links oben in Bild 17a startet die Routine mit der Subroutine MESSY; diese druckt den Text "HEXDUMP:" aus. Als nächste folgt die Subroutine INPAR, die wir zuerst näher betrachten wollen, bevor wir die Besprechung der M-Tastenroutine unter Punkt 25 fortsetzen.

24 **Die Subroutine INPAR** (Bild 18) wird entweder in voller Länge oder zu einem Teil (ab Label IPB) während der Tastenroutinen M, G und S durchlaufen. Das sind die Tastenfunktionen, bei denen zusätz-

17a

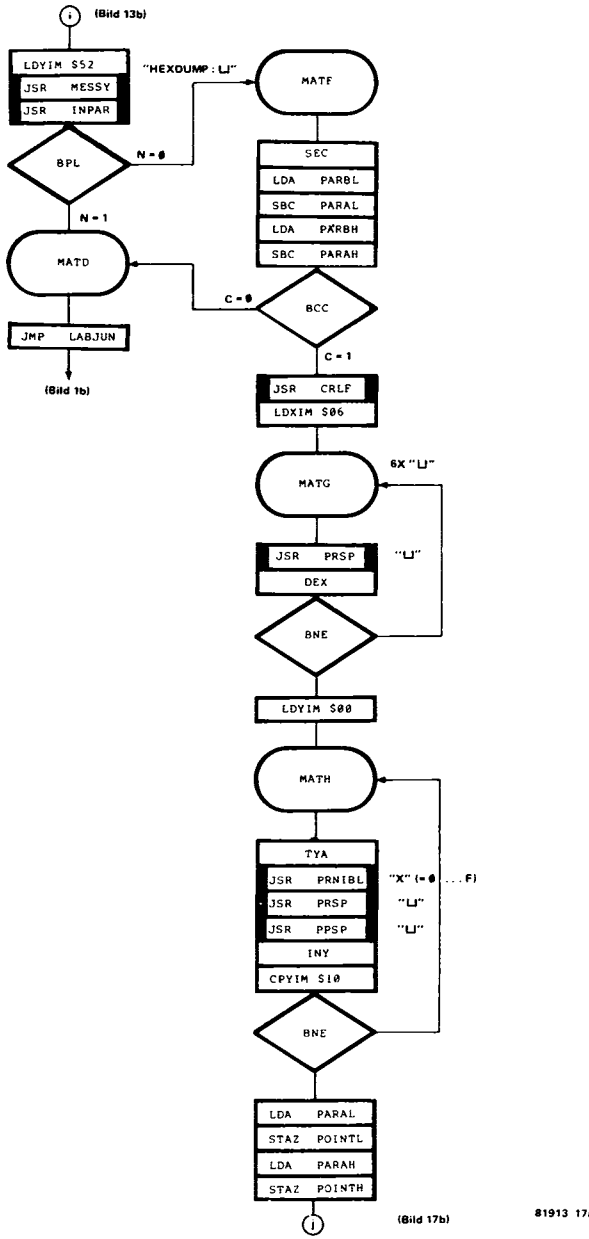
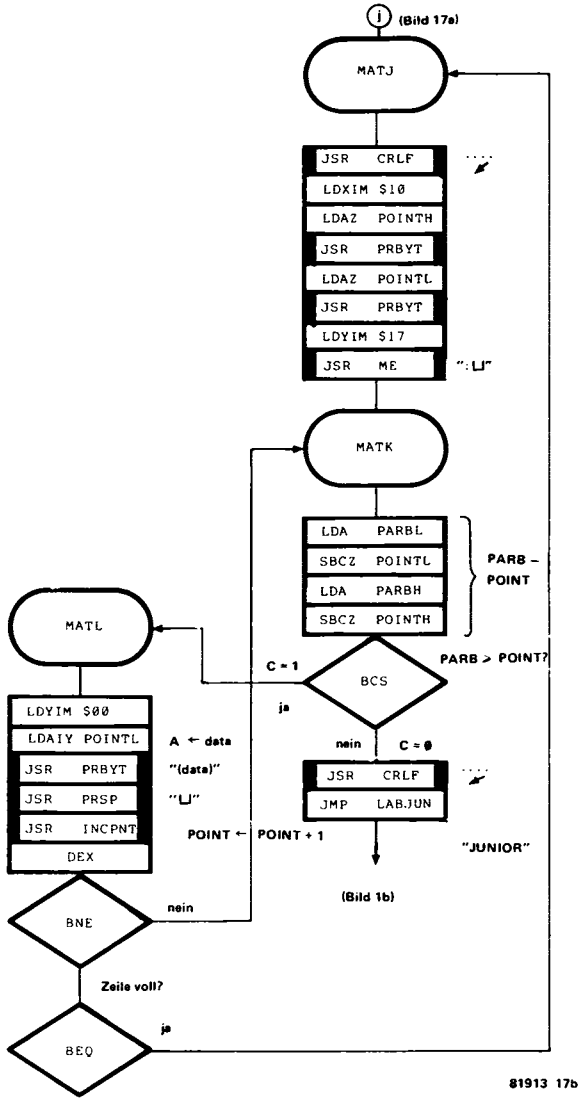


Bild 17. Bis auf die ersten beiden Instruktionen (sie sind in Bild 13b zu finden) ist hier die M-Tastenroutine dargestellt. Mit ihr druckt der Junior-Computer auf Tastendruck das Hexdump eines beliebigen Speicherbereichs aus.

17b



lich Parameter eingegeben werden müssen. In Kapitel 12, Buch 2, wurde der praktische Umgang mit den Tastenfunktionen M, G und S ausführlich behandelt.
Subroutine INPAR ermöglicht die reibungslose Eingabe der Parameter. Aus Bild 18 geht hervor, daß am Anfang von INPAR die Subroutine RECCHA steht: das Programm wartet auf das Drücken einer Taste. Sobald dies geschieht, wird zuerst die gedrückte Taste identifiziert. Wenn nicht

die Kommataste, sondern eine andere Taste gedrückt wurde, dann müssen entweder zulässige Daten (mit den numerischen Tasten) oder unzulässige Daten (mit den übrigen Tasten ohne die Kommataste) eingegeben worden sein. Die Subroutine HEXNUM (siehe Punkt 13 und Bild 11) trifft die Entscheidung darüber und setzt die numerischen Daten Ø . . . F in die Datenpuffer INL und INH.
Wenn die Kommataste gedrückt wird, ist die erste Dateneingabe beendet. Die Subroutine verzweigt dann zum Label IPA. Die eingegebenen Daten werden in die **Adreßpuffer PARAL und PARAH** gesetzt. Eingegeben wurde beispielsweise die erste Adresse eines auszudruckenden Hexdump (siehe Punkt 25) oder die Startadresse eines auf Band zu speichernden Datenblocks (Subroutine SAID, siehe Punkt 28). Steht diese erste Adresse in PARAL/PARAH, so löscht Subroutine RESIN die Datenpuffer INH und INL.
Wir sind nun in Bild 18 beim Label IPB angekommen. Auch hier wartet das Programm zunächst auf das Drücken einer Taste (Subroutine RECCHA). Geschieht dies, dann klärt die CMP-Instruktion, ob die CR-Taste als Zeichen dafür gedrückt wurde, daß die Parametereingabe beendet ist. Anderenfalls verarbeitet Subroutine HEXNUM die Eingabe. Wurde dagegen die CR-Taste gedrückt, so befinden wir uns nun beim Label IPC. Die zuletzt eingegebenen Daten (zweite Adresse) wandern in die **Adreßpuffer PARBL und PARBH**. Anders als nach Drücken der Kommataste bleiben in diesem Fall die Inhalte von INH und INL unverändert. Das Y-Register wird am Schluß von INPAR auf Null gesetzt, so daß die Z-Flag 1 und die N-Flag Ø ist.

25 Nun wenden wir uns wieder der M-Tastenroutine in Bild 17a zu, deren Besprechung wir unterbrochen hatten. Nach der Rückkehr aus INPAR (Punkt 24) bestimmt die N-Flag den weiteren Verlauf. Wurde innerhalb von INPAR eine unzulässige Taste gedrückt, dann erreichen wir über das Label MATD die Routine LABJUN in Bild 1b. Nach der Rückmeldung "JUNIOR" kommen wir schließlich an der Endstation RESALL der PM-Hauptroutine an. Von hier aus können wir durch Drücken von Taste M einen neuen Anlauf nehmen und noch einmal von vorn beginnen. War dagegen die Eingabe formal korrekt, so folgt ein Sprung zum Label MATF. Aus den daran anschließenden Instruktionen kann sich ergeben, daß die Parametereingabe trotzdem fehlerhaft war: Von der in PARAH, PARAL stehenden 16-bit-Zahl wird die 16-bit-Zahl subtrahiert, die in PARBH, PARBL steht. Ist das Ergebnis negativ (Carry-Flag = Ø), so folgt ein Sprung über Label MATD zu Routine LABJUN; sie druckt die Fehlermeldung "JUNIOR" aus. Die zweite Adresse (PARB) darf nämlich nicht niedriger als die erste Adresse (PARA) liegen. Das gilt sowohl für das Ausdrucken eines Hexdump als auch für das Schreiben von Daten auf Band (siehe Punkt 28).
Liegt die zweite Adresse höher als die erste (C = 1), dann kann das Ausdrucken des Hexdump beginnen. Die Instruktionen zwischen den Labeln MATG und MATH in Bild 17a sorgen dafür, daß auf einer neuen Zeile zuerst sechs Leerräume ausgedruckt werden. Der Grund dafür ist auf den ersten Blick deutlich, wenn man ein fertiges Hexdump zur Hand nimmt.

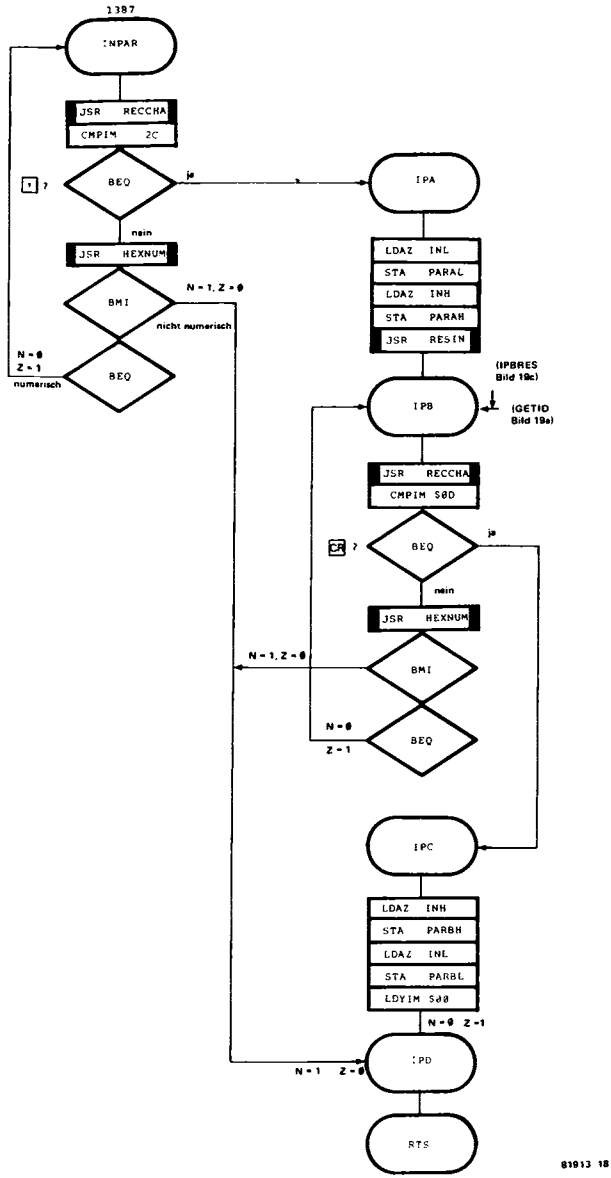


Bild 18. Subroutine INPAR nimmt die Eingabe der Parameter bei den Tastenfunktionen M, G und S entgegen. In bestimmten Fällen wird nur der letzte Teil ab Label IPB durchlaufen.

Auf die sechs Leerräume folgen in der ersten Zeile die Ziffern Ø . . . F. Sie bezeichnen die sechzehn Spalten des Hexdump; zwischen den einzelnen Ziffern stehen deshalb jeweils zwei Leerräume. Die letzten Instruktionen des in Bild 17a gezeichneten Teils von Tastenroutine M kopieren den Inhalt des Puffers PARA, der die erste Adresse enthält, in Adressenpuffer POINT.

26 Bild 17b zeigt die zweite Hälfte der M-Tastenroutine. Nach Label MATJ springt die Schreibposition zuerst an den Anfang einer neuen Zeile (CRLF). An dieser Stelle muß die Adresse erscheinen, die zum ersten in dieser Zeile ausgedruckten Speicherplatzinhalt gehört. Er steht in (PARAH, PARAL) und zunächst gleichzeitig in (POINTH, POINTL).
Am Anfang der ersten Zeile wird also die erste Adresse, gefolgt von einem Doppelpunkt und einem Zwischenraum, sowie den zugehörigen Daten ausgedruckt. Subroutine ME unterscheidet sich von Subroutine MESSY nur durch das Fehlen der CRLF-Routine; siehe Bild 9a und Punkt 11. Ferner wird im X-Register, das den Anfangswert $1Ø erhält, die Anzahl der eventuell noch in der betreffenden Zeile auszudruckenden Daten festgehalten.
Die Instruktionen nach Label MATK prüfen, ob bereits sämtliche Daten des Hexdump ausgedruckt werden. Dafür ist der Unterschied zwischen dem Inhalt von Adreßpointer POINT und der in PARB stehenden letzten Hexdumpadresse maßgebend. Die Carry-Flag gibt darüber Auskunft, wann das Ende erreicht ist. Liegt die in POINT stehende Adresse niedriger als die Adresse in PARB, so werden die Daten nach der Verzweigung zum Label MATL in den Akku geladen und anschließend von Subroutine PRBYT ausgedruckt. Hinzu kommt noch ein Zwischenraum (PRSP). Anschließend wird die in POINT stehende Adresse um eine erhöht (Subroutine INCPNT, siehe Punkt 15 und Bild 15a); der Inhalt des X-Registers wird um eins herabgesetzt. Damit sind die notwendigen Vorbereitungen getroffen, um das nächste Byte auszudrucken. Ob dies in der gleichen oder bereits am Anfang der nächsten Zeile geschieht, bestimmt die Z-Flag; sie stellt die Weichen bei den Verzweigungen BNE und BEQ. Die Frage, ob noch weitere Bytes auszudrucken sind, entscheidet sich wieder nach dem Label MATK. Es wird entweder direkt erreicht (gleiche Zeile; BNE auf Sprung) oder nach der Erledigung der Instruktionen, die dem Label MATJ folgen (neue Zeile: BEQ auf Sprung).
Sind sämtliche Daten ausgedruckt, so wird erst ein neuer Zeilenbeginn und dann das Label LABJUN angesteuert: die Rückmeldung "JUNIOR" ist der Schlußpunkt unter dem Hexdump. Da vor dem Sprung nach LABJUN noch Subroutine CRLF aufgerufen wird und die Rückmeldung "JUNIOR" auch stets am Anfang einer neuen Zeile erscheint (siehe Subroutine MESSY und JUNIOR), befindet sich zwischen dem Hexdump und dem Wort "JUNIOR" eine Leerzeile.
Noch eine Anmerkung: Wenn die Anzahl der auszudruckenden Byte durch 16 teilbar ist und deshalb auch in der letzten Zeile des Hexdump 16 Byte stehen, wird am Anfang der folgenden leeren Zeile trotzdem eine Adresse ausgedruckt. Dies ergibt sich aus dem Programmablauf nach Label MATL in Bild 17b. Beim Beginn einer neuen Zeile springt die Routine über den BEQ sofort nach Label MATJ, um die nächste Reihenadresse auszudruk-

ken; sie "weiß" zu diesem Zeitpunkt noch nicht, daß keine weiteren Daten mehr vorhanden sind.

27 Für die Besprechung der **G-Tastenroutine** verlassen wir Bild 13b und betrachten nun den dritten Teil der PM-Hauptroutine in Bild 13c. Zur-G-Tastenroutine gehören die Instruktionen, die an das Label GETTAP anschließen. **Subroutine GETID** (Bild 19a) leistet hier die Hauptarbeit; ihr wollen wir uns deshalb zuerst zuwenden. Unmittelbar nach Label GETID folgt ein Sprung zur Subroutine IPB. Diese Subroutine ist nicht neu, sondern sie ist Teil der Subroutine INPAR in Bild 18 und wurde bereits unter Punkt 24 besprochen.
Um eine Datensendung vom Band zu lesen, muß nicht nur Taste G gedrückt werden, sondern der Junior-Computer wartet außerdem auf die Eingabe einer Programmnummer ID. Danach ist die CR-Taste zu drücken. Am Ende von IPB, das gleichzeitig das Ende von INPAR ist, läßt sich an der N-Flag ablesen, ob die Programmnummer vorschriftsmäßig eingegeben wurde. Der gleich nach JSR-IPB folgende BMI (Bild 19a) führt bei falscher Eingabe sofort über Label GA zum Rücksprung.
Haben wir Programmnummer ID vorschriftsmäßig eingegeben, dann wird der Inhalt des Akku in Speicherplatz ID gesetzt (vergleiche Kapitel 16). Kurz vor Verlassen von IPB, nämlich zwischen dem Label IPC und dem RTS, ist der Akku-Inhalt gleich dem Inhalt des rechten Datenpuffers INL. Übrigens: Subroutine IPB wird bei korrekter Programmnummereingabe erst beim Drücken der CR-Taste verlassen.
Wurde als Programmnummer FF gewählt, dann muß zusätzlich eine Startadresse SA eingegeben werden. Das Programm springt dann in Bild 19a zum Label IDPAR. Nach der Rückmeldung "SA:" und dem Durchlauf durch IPBRES (Bild 19c) folgt ein weiterer Sprung nach IPB, dem letzten Teil von INPAR. Dort nimmt der Computer die Startadresse SA entgegen und wartet anschließend auf das Drücken der CR-Taste. Der Umweg über IPBRES ist notwendig, um die Datenpuffer INL und INH zu löschen. Eine andere Möglichkeit wäre ein Aufruf von Subroutine RESIN unmittelbar vor dem Label IPB in Bild 18 gewesen.
Vorausgesetzt, daß im Anschluß an den zweiten Sprung nach IPB alles ordnungsgemäß verlaufen ist, werden die Inhalte von INH und INL in die Speicherplätze SAH und SAL kopiert (siehe Kapitel 16). Dann springt das Programm zum Label GB, wo ebenso wie im Fall ID ungleich FF das eigentliche Lesen der auf Band stehenden Daten beginnt. Zuständig ist dafür die Subroutine RDTAPE, die wir jedoch erst in Kapitel 16 besprechen. Die Input/Output-Leitungen sind am Schluß von RDTAPE anders geschaltet als bisher. **Subroutine RESTTY** in Bild 19b stellt deshalb nach der Rückkehr aus RDTAPE den ursprünglichen Zustand (siehe Bild 1a) wieder her. Übrig bleiben noch das Ausdrucken der Rückmeldung "READY", das Setzen der Z-Flag (N-Flag = Ø) und der Rücksprung aus GETID zu Routine GETTAP in Bild 13c.
Wenn wir GETID bei gesetzter N-Flag verlassen, folgt über das Label GETERR ein Sprung zum Label LABJUN, so daß die Fehlermeldung "JUNIOR" erscheint. N=Ø bedeutet dagegen, daß uns bei der Bedienung (siehe Buch 3) keine Fehler unterlaufen sind. Wir kehren dann zum zentralen Start- und Zielpunkt RESALL der PM-Hauptroutine zurück.

19a

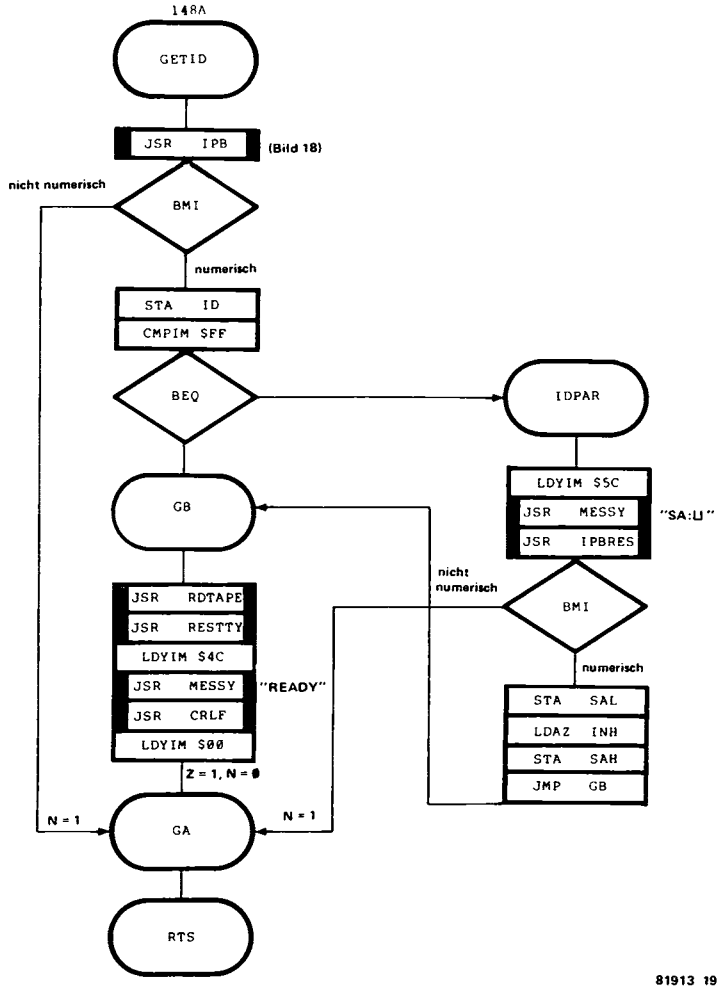
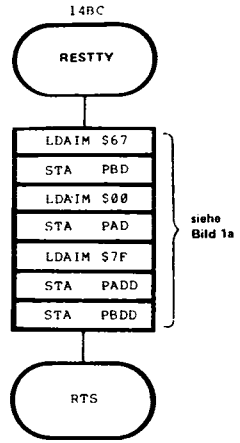


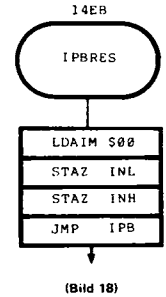
Bild 19. Die meiste Arbeit bei Aktivierung der G-Tastenroutine leistet Subroutine GETID in Bild 19a. Subroutine RESTTY (BILD 19b) stellt den ursprünglichen Zustand der I/O-Ports nach Durchlaufen der Kassetten-Subroutinen RDTAPE und DUMP (vgl. Bild 20) wieder her. IPBRES (Bild 19c) löscht die Datenpuffer INL/INH vor dem Sprung nach IPB.

19b



81913 19b

19c



81913 19c

28 **Die S-Tastenroutine,** die ebenfalls die Eingabe von Parametern erfordert, beginnt in Bild 13c beim Label SAVID. In Buch 3, Kapitel 11, wurde ausführlich beschrieben, wie und in welcher Reihenfolge die einzelnen Parameter einzugeben sind. Hier deshalb nur eine kurze Aufzählung der Bedienfolge:
Taste S – erste Dateneingabe – Komma – zweiten Dateneingabe – Komma – dritte Dateneingabe – Komma – Taste CR.
Bei der ersten Dateneingabe muß eine Programmnummer ID eingegeben werden, die zwischen Ø1 und FE liegt;
bei der zweiten Dateneingabe ist Startadresse SA einzugeben;
die dritte Dateneingabe ist die Eingabe der Endadresse EA (=LA+1).
Kernstück der S-Tastenroutine ist **Subroutine SAID** in Bild 20, die wir deshalb zuerst besprechen wollen. SAID prüft zunächst, ob die Kommataste oder eine andere Taste gedrückt wurde. Trifft letzteres zu, dann ist die Dateneingabe noch nicht abgeschlossen. Subroutine HEXNUM unterscheidet ankommende numerische Daten von unzulässigen, nicht numerischen Daten. Numerische Daten werden von HEXNUM in die Datenpuffer INH und INL gesetzt, während im anderen Fall ein Sprung zum Label SIB mit N=1 und Z=Ø und somit zu einer Fehlermeldung führt. Nach Drücken der Kommataste springt die Routine zum Label SIC. Der Inhalt von INL wird in ID gesetzt. Dann wird geprüft, ob als Programmnummer entgegen der Bedienvorschrift ØØ oder FF eingegeben wurde. Liegt ein solcher Verstoß vor, so löst dies mit N=1 und Z=Ø über Label SIA eine Fehlermeldung aus.
Für die Verarbeitung der zweiten und dritten Eingabe sowie der abschließenden Betätigung der CR-Taste ist Subroutine INPAR (Bild 18) zuständig, die wir bereits unter Punkt 24 besprochen haben. Bevor INPAR in Aktion tritt, werden noch von RESIN die Datenpuffer gelöscht.
Nach Drücken der CR-Taste beginnt das Senden der Daten zum aufnehmenden Kassettengerät. In die Startadressenpuffer SAH und SAL werden die Inhalte der Puffer PARAH und PARAL kopiert, in denen die erste

20

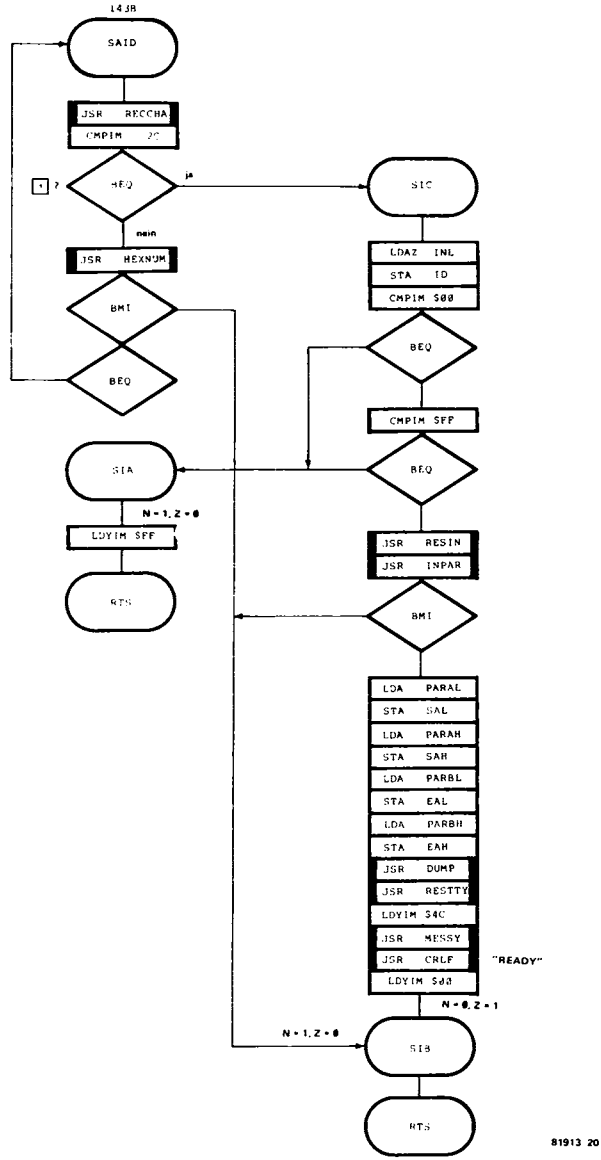


Bild 20. Hauptbestandteil der S-Tastenroutine ist die Subroutine SAID. Zu ihr gehört auch ein Sprung nach TM-Subroutine DUMP, von der die Datenblöcke zum Kassettenrekorder gesendet werden.

Adresse steht. Mit der in PARBH und PARBL stehenden letzten Adresse geschieht Ähnliches: Einige LDA- und STA-Befehle kopieren sie in Endadressenpuffer EAH/EAL. Sind diese Reisevorbereitungen getroffen, dann kann der Datenzug zum Bandgerät abfahren. Die "Lokomitive" ist dabei die Subroutine DUMP, die wir allerdings erst in Kapitel 16 besprechen. Ebenso wie RDTAPE (vgl. Punkt 22) ändert auch DUMP die Programmierung der I/O-Ports. Auf DUMP folgt deshalb Subroutine RESTTY, die die I/O-Leitungen wieder in den alten Zustand schaltet.
Den Abschluß von SAID bildet Rückmeldung "READY". Ferner wird noch das Y-Register auf ØØ gesetzt, beim Rücksprung ist daher die N-Flag Ø und die Z-Flag ist 1.
Zurück zur S-Tastenroutine in Bild 13c. Nach der Rückkehr aus SAID zeigt die N-Flag den ordnungsgemäßen oder fehlerhaften Ablauf der Eingabevorgänge an. Im ersten Fall ist wieder das Label RESALL die Endstation, im zweiten erreichen wir über GETERR das Label LABJUN, so daß die Rückmeldung "JUNIOR" erscheint. Sie hat hier den Charakter einer Fehlermeldung.

29 Übrig bleibt von der PM-Hauptroutine nur noch **die Daten-Tastenroutine**; sie umfaßt die Instruktionen, die in Bild 13c mit dem Label VALNUM überschrieben sind. Sämtliche Tasten, die bei PM eine spezielle Bedeutung tragen, haben wir inzwischen besprochen:
Taste + unter Punkt 15,
Taste – unter Punkt 16,
Taste SP unter Punkt 17,
Taste • unter Punkt 18,
Taste R unter Punkt 19,
Taste L unter Punkt 21,
Taste M unter Punkt 23, 25 und 26,
Taste G unter Punkt 27 und
Taste S unter Punkt 28.
Während die in vorstehender Liste fehlenden nicht numerischen Tasten bei PM keine Bedeutung haben, können über die numerischen Tasten bei Bedarf Daten eingegeben werden. Handelt es sich dabei um Arbeitsadressen oder Arbeitsdaten, so übernimmt die Daten-Tastenroutine ihre Verarbeitung. Die Parametereingabe bei den Tastenfunktionen M, G und S wird dagegen nicht von RECCHA innerhalb der RESALL-READCH-Schleife verarbeitet; das wurde bereits bei der Besprechung dieser Tastenfunktionen deutlich.
Numerische Daten verarbeitet Subroutine HEXNUM, die in Bild 11 gezeichnet ist und unter Punkt 13 besprochen wurde. Solange HEXNUM läuft, beantwortet der Junior-Computer jeden Druck auf eine nicht numerische Taste mit der Fehlermeldung "WHAT?". In diesem Fall springt das Programm zum Label VNA und von dort zu RESALL, dem zentralen Start- und Zielpunkt von PM. Bei der Eingabe von numerischen Daten werden diese von HEXNUM aus dem ASCII-Kode in Datennibbles umgewandelt und in die Datenpuffer INH und INL gesetzt. Danach folgt ein Sprung zum Label READCH; siehe Bild 13a. Die Datenpuffer INH und INL werden in diesem Fall nicht gelöscht (RESIN, unmittelbar nach Label REALL), da die Dateneingabe noch unvollständig sein kann.

Die Beschreibung der PM-Software ist damit abgeschlossen. Hinzugefügt werden soll noch eine ergänzende Anmerkung zur Funktion der BRK-Taste innerhalb von PM.

30 In Buch 3, Kapitel 12 sowie unter Punkt 8 in diesem Kapitel haben wir gesagt, daß das Drücken der BRK-Taste innerhalb von PM nur dann eine Wirkung hat, wenn der Junior-Computer mit dem Ausdrucken von Daten oder Texten beschäftigt ist; der Druckvorgang wird dann abgebrochen. Betroffen ist hiervon stets die elementare Druck-Subroutine PRCHA.
In der Praxis kann man dagegen beobachten, daß anscheinend auch bei anderer Gelegenheit ein Druck auf die BRK-Taste zur Rückmeldung "JUNIOR" führt: dann nämlich, wenn Subroutine RECCHA auf das Drücken einer Taste wartet. Dies läßt sich jedoch wie folgt erklären: Wenn für die Dauer von mindestens neun aufeinanderfolgenden Bitzeiten die BRK-Taste gedrückt bleibt, hat die Tastatur den ASCII-Kode ØØ zum Junior-Computer gesandt. Da der ASCII-Kode ØØ nicht zum "Wortschatz" von PM gehört, wird er mit der Fehlermeldung "WHAT?" beantwortet. Im gleichen Moment, in dem das Ausdrucken von "WHAT?" beginnt, ist aber die BRK-Taste noch gedrückt. Der gerade eingeleitete Druckvorgang wird sofort wieder abgebrochen, und nach Loslassen der BRK-Taste erscheint die Rückmeldung "JUNIOR".

15

766-Byte PME-Software

Der PM-Editor

Bekanntschaft mit dem Systemprogramm PME haben wir bereits in Buch 3, Kapitel 13 geschlossen. Dort haben wir erfahren, was der neue Editor leistet und wie man seine Möglichkeiten nutzt. Nach jenem mehr äußerlichen und praxisbezogenen Kennenlernen wollen wir uns nun hinter den Kulissen vom PME umschauen.
Der PM-Editor baut zum Teil auf Subroutinen anderer Systemprogramme auf, er kann jedoch auch mit neuen, eigenen Subroutinen aufwarten. Auch diese Subroutinen bieten sich für den "Einbau" in andere Software an. Obwohl der Umfang von PME, gemessen an der Zahl der Byte, längst nicht an PM und TM heranreicht, wollen wir auch hier die Form der kompakten Punkt-für-Punkt-Beschreibung wählen. Dieser Rahmen paßt zum PM-Editor, denn auch er hat eine kompakte innere Struktur. Trotzdem (oder gerade deshalb?) ist PME, wie wir uns in Kapitel 13 überzeugen konnten, äußerst vielseitig und leistungsfähig.

1 Beginnen wollen wir mit einer **allgemeinen Übersicht** über den Aufbau von PME. Dieses **"Blockschema"**, man könnte es auch das Programm des Programms nennen, ist gleichzeitig das Programm dieses Kapitels. Zu finden ist es in Bild 1.
Betrachten wir Bild 1, so stellen wir bei PME folgende grundlegende Struktur fest: Einer Hauptroutine geht eine in bestimmten Fällen initial zu durchlaufende Vorroutine voran. Wie ein breiter Fluß ("Flußdiagramm"!) aus dem Zustrom von Nebenflüssen und Bächen entsteht, so hat die PME-Vorroutine ihren Ursprung in drei verschiedenen Quellen: dem Kaltstart, dem lauwarmen Start und der warmen CEND-Startroutine. Ein vierter

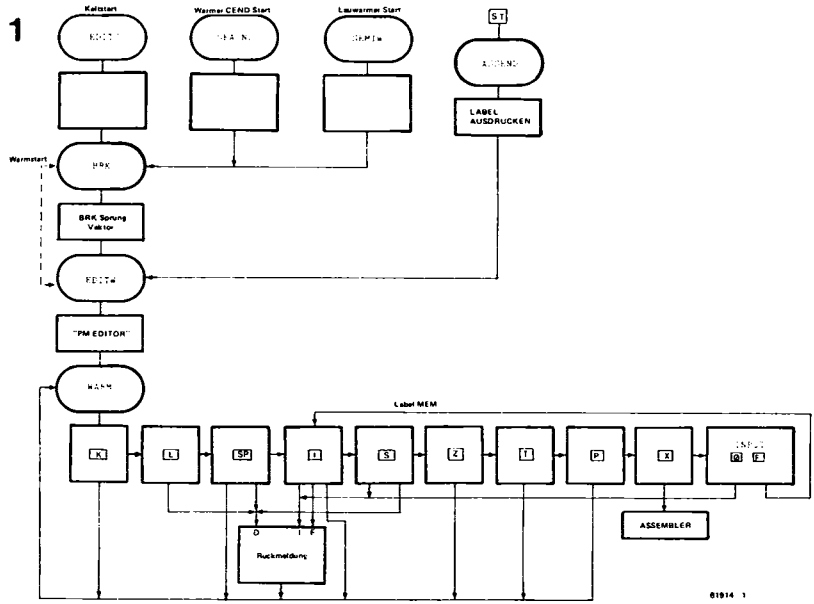


Bild 1. Blockschema des Systemprogramms PME. Wie sich Nebenflüsse zu einem Strom vereinen, so treffen die verschiedenen Vorroutinen beim zentralen Label WARM zusammen.

"Nebenfluß" ist die beim Label ASSEND beginnende Routine; man muß sie eigentlich als Zwischenroutine bezeichnen, weil sie erst nach dem Assemblieren eines Programms durchlaufen wird. Das geschieht, nebenbei bemerkt, durch Drücken von Taste ST auf der Standard-Tastatur.
Während des Verbleibs in PME befindet sich das Programm meistens innerhalb der PME-Hauptroutine. Zu ihr gehören in Bild 1 sämtliche Blöcke hinter dem Label WARM. Was innerhalb der PME-Hauptroutine vor sich geht, erscheint uns bekannt. Dem Drücken einer Taste folgt das gewohnte Frage- und Antwortspiel: War es Taste A? Wenn ja, führe die A-Tastenroutine aus. Wenn nicht, dann war es vielleicht Taste B? In diesem Fall führe die B-Tastenroutine aus. Oder war es Taste C? Dann führe die C-Tastenroutine aus, und so weiter. Nach der Ausführung der zur gedrückten Taste gehörenden Routine folgt ein Rücksprung zum zentralen Label WARM der PME-Hauptroutine. Das Programm wartet dann auf den nächsten Tastendruck, um anschließend die zugehörige Tastenroutine zu durchlaufen.
Manchmal liegt auf dem Rückweg zum Label WARM noch eine Zwischenstation in Form einer Rückmeldung wie "DONE", "ILLEGAL KEY" oder "FULL". Nach der X-Tastenroutine kehrt das Programm nicht zum Label WARM zurück, sondern springt zum Assembler; das Label WARM wird erst über einen Umweg (Taste ST, Label ASSEND, Label EDITW) erreicht. Der beim Label MEM beginnende Teil der I-Tastenroutine ist gleichzeitig der letzte Teil der Input-Tastenroutine.

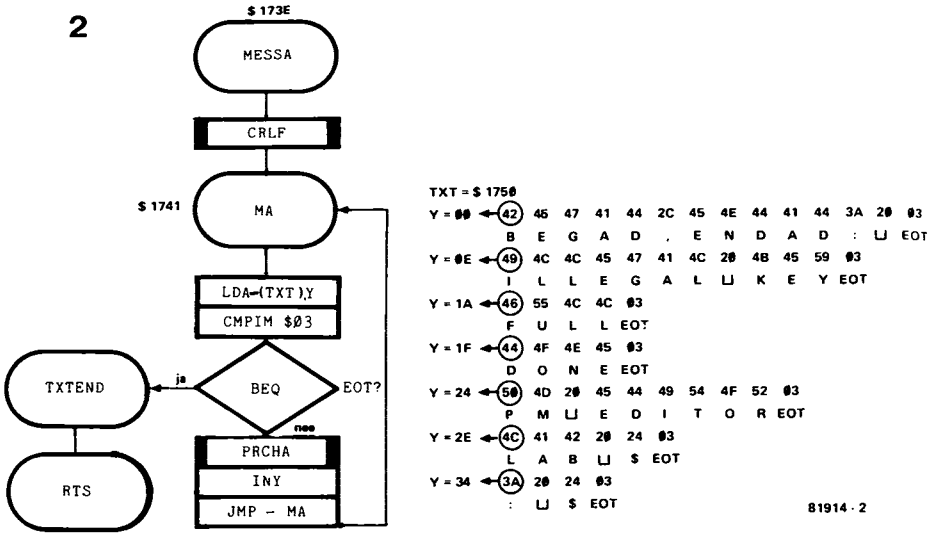


Bild 2. Die PME-Subroutine MESSA ist identisch mit der PM-Subroutine MESSY; in der Look-Up-Tabelle stehen jedoch die für PME benötigten Texte.

2 **Subroutine MESSA** in Bild 2 dient dem PM-Editor zum Ausdrucken von Texten. Diese Subroutine ist weitgehend mit Subroutine MESSY identisch, die in Kapitel 14, Abschnitt 11 beschrieben wurde und dort in Bild 9a dargestellt ist. MESSA unterscheidet sich dadurch von MESSY, daß hier nicht MESS, sondern TXT die erste Adresse der Texttabelle ist. In dieser Tabelle stehen folgende Texte: "BEGAD, ENDAD", "ILLEGAL KEY", "FULL", "DONE", "PM EDITOR", "LAB $" und ": $". Den zu druckenden Text bestimmt der Wert, der vor dem Sprung zur Subroutine MESSA im Y-Register steht.

3 **Die Subroutine PRINS** in Bild 3 druckt eine Instruktion aus, wobei der Instruktion die Adresse vorangeht, unter der der Opcode der Instruktion gespeichert ist. Auf die Instruktion folgen mehrere Zwischenräume; ihre Anzahl hängt von der Instruktionslänge und somit von der Anzahl der zur Instruktion gehörenden Byte ab. Von Subroutine PRINS wird stets die Instruktion ausgedruckt, auf deren Opcode-Adresse der Adreßpointer CURAD zeigt. In den Kapiteln 5 und 8 (Buch 2) wurde CURAD "Displaypointer" genannt, doch hier bei PME ist die Bezeichnung "Instruktionspointer" zutreffender. CURAD steht stets mit der letzten Instruktion in Zusammenhang, die auf der linken Seite des TV-Schirms oder des Druckerpapiers erschienen ist oder erscheinen wird.
Das Flußdiagramm von Subroutine PRINS (Bild 3) beginnt mit CRLF (siehe Kapitel 14, Punkt 12), so daß zuerst der Anfang einer neuen Zeile in der linken Spalte angesteuert wird. Dann folgt eine bereits vom Standard-Editor bekannte Subroutine: OPLEN wurde in Buch 2, Seite 179 ff. be-

3

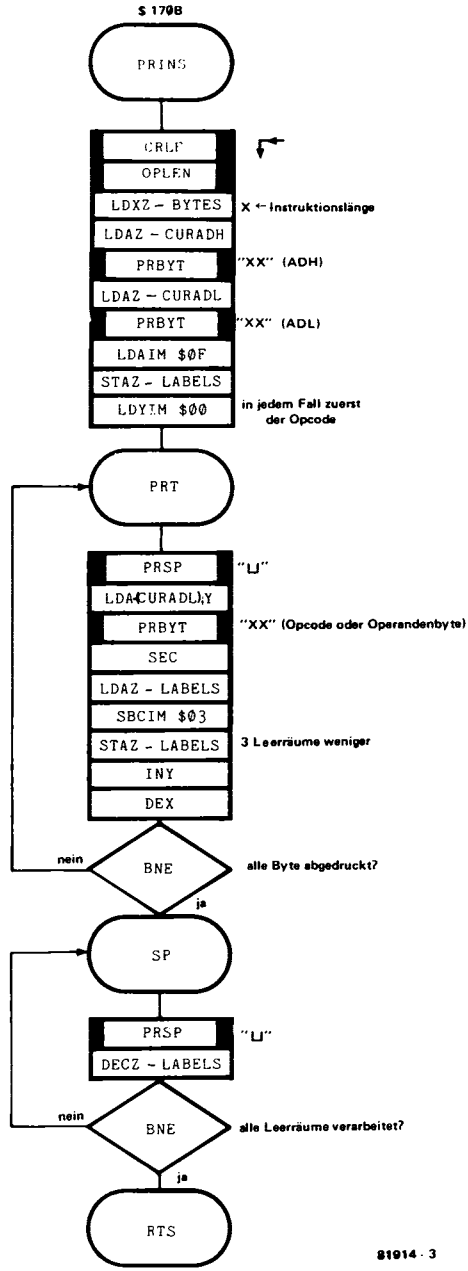


Bild 3. PME-Subroutine PRINS druckt die von Instruktionspointer CURAD angewiesene Instruktion mit der Adresse ihres Opcodes in der linken Spalte aus. Danach folgen noch so viele Leerräume, daß die nächste Tasteneingabe linksbündig in der rechten Spalte erscheint.

schrieben. Der Opcode der Instruktion, auf die der Instruktionspointer CURAD zeigt, wird von Subroutine OPLEN in den Akku geladen. Danach stellt OPLEN fest, ob dieser Opcode zu einer Ein-, Zwei- oder Drei-Byte-Instruktion gehört und setzt das Ergebnis in Speicherplatz BYTES.
Bei Subroutine PRINS wird nach OPLEN der Inhalt von BYTES in das X-Register übertragen, so daß dort die Anzahl der auszudruckenden Byte steht. Bevor dies geschieht, müssen jedoch die Inhalte von CURADH und CURADL ausgedruckt werden, denn sie enthalten die Opcode-Adresse. Das erledigt Subroutine PRBYT, die in Kapitel 14, Abschnitt 9 besprochen wurde. Vor Erreichen des Labels PRT wird noch der Wert ØF in Speicherplatz LABELS gesetzt, was in Zusammenhang mit der Anzahl der Leerräume steht, die noch im Anschluß an die Instruktion auszudrucken sind.
Der Teil von PRINS zwischen den Labeln PRT und SP wird abhängig von der Instruktionslänge und somit abhängig vom Inhalt des X-Registers einmal, zweimal oder dreimal nacheinander durchlaufen. Beim ersten Durchlauf ist der Inhalt von Y gleich ØØ, beim eventuell folgenden zweiten Durchlauf ist er Ø1 und beim dritten Durchlauf, sofern dieser stattfindet, ist er Ø3. Jeder Durchlauf beginnt mit dem Ausdrucken eines Leerraums (Subroutine PRSP, siehe Kapitel 14, Punkt 12). Danach wird der Akku mit dem Inhalt des Speicherplatzes geladen, dessen Adresse sich aus dem Inhalt von CURADH/CURADL und dem Wert des Y-Registers ergibt. Die Dreckarbeit erledigt auch hier die Subroutine PRBYT. Ferner wird der Inhalt von Speicherplatz LABELS bei jedem Durchlauf zwischen den Labeln PRT und SP liegenden Teil von PRINS um Ø3 herabgesetzt. Jedes ausgedruckte Byte verkürzt nämlich den Raum, der noch im Anschluß an die Instruktion bis zum Erreichen der rechten Spalte mit Leerräume gefüllt werden muß.
Mit dem Ausdrucken dieser Leerräume beschäftigt sich der letzte Teil von PRINS nach dem Label SP. Die Anzahl steht in Speicherplatz LABELS: sie hängt, wie schon erklärt, von der Instruktionslänge ab. Auf die ausgedruckte Instruktion folgen 12 Leerräume, wenn es sich um eine Ein-Byte-Instruktion handelt. Bei Zwei-Byte-Instruktionen sind dagegen nur 9 Leerräume und bei Drei-Byte-Instruktionen nur 6 Leerräume notwendig.
Das Ausdrucken der Leerräume könnte übrigens bei der P- und L-Tastenroutine entfallen, denn dort wird für eine Reihe von Zeilen kein Gebrauch von der rechten Bildschirm- bzw. Druckerpapier-Spalte gemacht. Auf die dadurch bei den genannten Tastenroutinen erzielbare, geringfügig höhere Schreibgeschwindigkeit wurde jedoch verzichtet, weil der Aufwand an zusätzlichen Subroutinen-Bytes zu hoch erschien.

4 **Der Kaltstart von PME** hat zur Folge, daß die Instruktionen der PME-Vorroutine vom Label EDITC bis zum Label WARM (Bild 4a) durchlaufen werden. Während der erste zwischen den Labeln EDITC und BRK liegende Teil ausschließlich nach einem Kaltstart abgearbeitet wird, durchläuft der Prozessor den Zweiten, auf Label BRK folgenden Teil der Vorroutine auch in bestimmten anderen Fällen.
Die Vorroutine in Bild 4a beginnt mit dem Aufruf der Subroutine RESIN, die wir bereits in Kapitel 14, Punkt 14 kennenlernten: Sie löscht die Inhalte der Datenpuffer INH und INL. Dann folgt der Aufruf von Sub-

4a

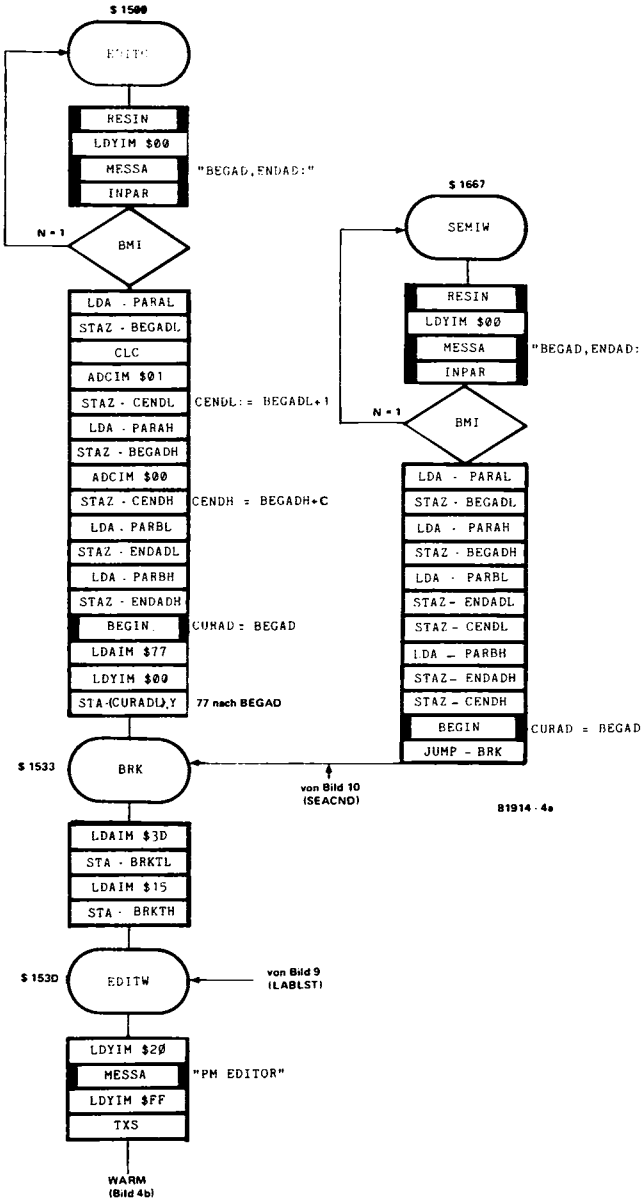


Bild 4a. Flußdiagramm der PME-Vorroutinen EDITC und SEMIW einschließlich des gemeinsamen Teils nach Label BRK.

routine MESSA (siehe Punkt 2 dieses Kapitels); MESSA druckt den Text "BEGAD, ENDAD:" aus. Subroutine INPAR, die unter anderem bereits in der M-Tastenroutine von PM (Ausdrucken eines Hexdump) vorkam, nimmt die Eingabe der Parameter entgegen. Zuerst ist Beginnadresse BEGAD einzugeben, dann die Kommataste zu drücken, daran schließt sich die Eingabe von Endadresse ENDAD an, und den Schlußpunkt setzt ein Druck auf Taste CR. Ausführlich beschrieben wurde Subroutine INPAR in Kapitel 14 unter Punkt 24.
Wenn auf irgendeine Weise bei der Eingabe ein Fehler entstand, dann ist am Ende von INPAR die N-Flag "1" und die Z-Flag "Ø". Die BMI-Verzweigung hinter INPAR (oben in Bild 4a) führt nach einem solchen Fehlstart zurück zum Label EDITC, so daß zum zweiten Mal der Text "BEGAD, ENDAD:" erscheint. In verschiedenen Fällen wird vorher auch noch die Fehlermeldung "WHAT?" sichtbar, weil, wie aus Bild 18 in Kapitel 14 hervorgeht, innerhalb von INPAR zweimal die Subroutine HEXNUM aufgerufen wird. HEXNUM, beschrieben in Kapitel 14, Abschnitt 13 und dargestellt in Bild 11 des gleichen Kapitels, beantwortet nämlich die Eingabe nicht numerischer Daten mit "WHAT?".
War die Eingabe von BEGAD und ENDAD korrekt, dann wird die in den Adreßpuffern PARAH und PARAL stehende erste Adresse nach BEGADH/BEGADL kopiert. Das gleiche geschieht mit der in PARBH/PARBL stehenden zweiten Adresse: sie wandert in die Speicherplätze ENDADH und ENDADL. Ferner werden die Speicherplätze CENDH und CENDL so geladen, daß der variable Endadreßpointer CEND auf die Adresse gerichtet ist, die um eine Adresse höher als Beginnadresse BEGAD liegt. Das ist notwendig, weil in BEGAD das EOF-Zeichen 77 stehen muß. Je mehr Instruktionen mit Tastenfunktion Input hinzugefügt oder mit Insert eingefügt werden, desto weiter verschiebt sich das EOF-Zeichen 77 gleichzeitig mit dem Inhalt von CEND zu den höheren Adressen. Die umgekehrte Richtung wird eingeschlagen, sobald die Tastenfunktionen Delete oder Kill (siehe Punkt 6) in Aktion treten.
Bevor Schlußlicht "77" in BEGAD gesetzt werden kann, muß Instruktionspointer CURAD auf BEGAD zeigen. Das bewirkt Subroutine BEGIN, die wir bereits aus Buch 2, Kapitel 8 kennen.
Nach Abarbeiten der PME-Vorroutine in Bild 4a wird die in BEGAD stehende Instruktion ausgedruckt. Ging ein Kaltstart voran, dann muß als erste die Instruktion mit dem Pseudo-Opcode 77, das EOF-Zeichen, erscheinen.

5 Nur der zweite Teil der PME-Vorroutine in Bild 4a wird unter anderem nach einem **Warmstart von PME** durchlaufen; er beginnt beim Label BRK. Wählt man diesen PME-Eingang, dann hängen die Werte von BEGAD, ENDAD, CURAD und CEND vom vorherigen Computer-Geschehen ab, oder sie wurden während der Vorroutinen SEMIW (siehe Punkt 19), SEACND (Punkt 20) oder der Zwischenroutine LABLST (Punkt 17, Zwischenlabel EDITW) festgelegt. Nach einem Warmstart muß in jedem Fall der BRK-Sprungvektor neu bestimmt werden, denn vor dem Warmstart befanden wir uns in PM; dort aber ist der BRK-Sprungvektor auf das Label LABJUN mit der Rückmeldung "JUNIOR" (siehe Kapitel 14) gerichtet. Deshalb wird in Bild 4a nach dem Label BRK als

erstes der BRK-Sprungvektor auf die zum Label EDITW gehörende Adresse $153D gerichtet. Label EDITW ist der Startpunkt für einige Instruktionen, die für das Ausdrucken der Rückmeldung "PM EDITOR" und für das Rücksetzen des Stackpointers auf FF sorgen. Letzteres kann für den Ablauf nach Drücken der BRK-Taste vorteilhaft sein, denn es sind Situationen denkbar, in denen nicht nur der Druckvorgang unterbrochen werden muß, sondern auch die Gefahr des Stack-Überlaufs besteht (genau genommen läuft der Stack nicht "über"; wenn Adresse Ø1ØØ erreicht ist, folgt wieder der erste Stack-Speicherplatz mit der Adresse Ø1FF).
Um PME warm zu starten, kann an die Stelle von Startadresse $1533 (Label BRK) auch Adresse $153D (Label EDITW) treten. Dies bedeutet allerdings, daß der vorher innerhalb von PM spezifizierte BRK-Sprungvektor unverändert bleibt. Das gilt übrigens bis zum Erreichen des Labels EDITW auch für den Durchlauf durch die Vorroutinen EDITC und SEMIW in Bild 4a und die Vorroutine SEACND in Bild 10. Drücken und anschließendes Loslassen der BRK-Taste führt während des Ausdruckens von "BEGAD, ENDAD:" oder während des Verbleibs in INPAR (einschließlich der beiden Sprünge nach RECCHA; siehe Kapitel 14, Punkt 30) zur Rückmeldung "JUNIOR". Dagegen hat das Drücken und Wiederloslassen der BRK-Taste die Rückmeldung "PM EDITOR" zur Folge, wenn diese Tastenaktion nach Erreichen des Labels BRK vorgenommen wird.

6 Wir kommen nun zur **Hauptroutine des Systemprogramms PME**, und zwar zum ersten, in Bild 4b dargestellten Teil. Beginnen wollen wir mit der Besprechung der **K-Tastenroutine.** Dazu gehören die Instruktionen, die unter dem zentralen Label WARM stehen (links oben in Bild 4b). Als erstes druckt Tastenroutine K die Instruktion aus, auf deren Adresse der Instruktionspointer CURAD zeigt. Gedruckt wird mit Subroutine PRINS, die wir schon unter Punkt 3 in diesem Kapitel besprochen haben und die in Bild 3 dargestellt ist. Nach PRINS folgt Subroutine RECCHA: Warte auf das Drücken einer Taste und setze den ASCII-Kode der gedrückten Taste in den Akku. Einzelheiten über RECCHA sind in Kapitel 14 unter Punkt 5 zu finden.
Wenn RECCHA feststellt, daß Taste K gedrückt wurde, springt PME zur Subroutine UP. Diese zum Standard-Editor gehörende Subroutine sorgt dafür, daß die von CURAD angewiesene Instruktion von den im Speicher folgenden Instruktionen überschrieben wird. Sämtliche nachfolgenden Instruktionen rücken um das Stück in Richtung niedrigerer Adressen, das die zu löschende Instruktion einnimmt.
An Subroutine UP schließt sich eine weitere Subroutine des Standard-Editors an: RECEND. Der Inhalt des variablen Endadreßpointers CEND wird von RECEND um den in Speicherplatz BYTES stehenden Betrag herabgesetzt. In BYTES ist nämlich ein Wert enthalten, der die Länge der überschriebenen Instruktion angibt. Von BYTES wird übrigens auch innerhalb der Subroutine UP Gebrauch gemacht. Die Subroutinen OPLEN oder LENACC werden hier nicht benötigt, da die überschriebene Instruktion als letzte ausgedruckt wurde. Bekanntlich steht in Speicherplatz BYTES immer die Anzahl der Byte, die die zuletzt ausgedruckte Instruktion umfaßt (vgl. Subroutine PRINS; Punkt 3, Bild 3 in diesem Kapitel).
Subroutine UP: Siehe Buch 2, Kapitel 8, Seite 176 und Bild 17/18.

4b

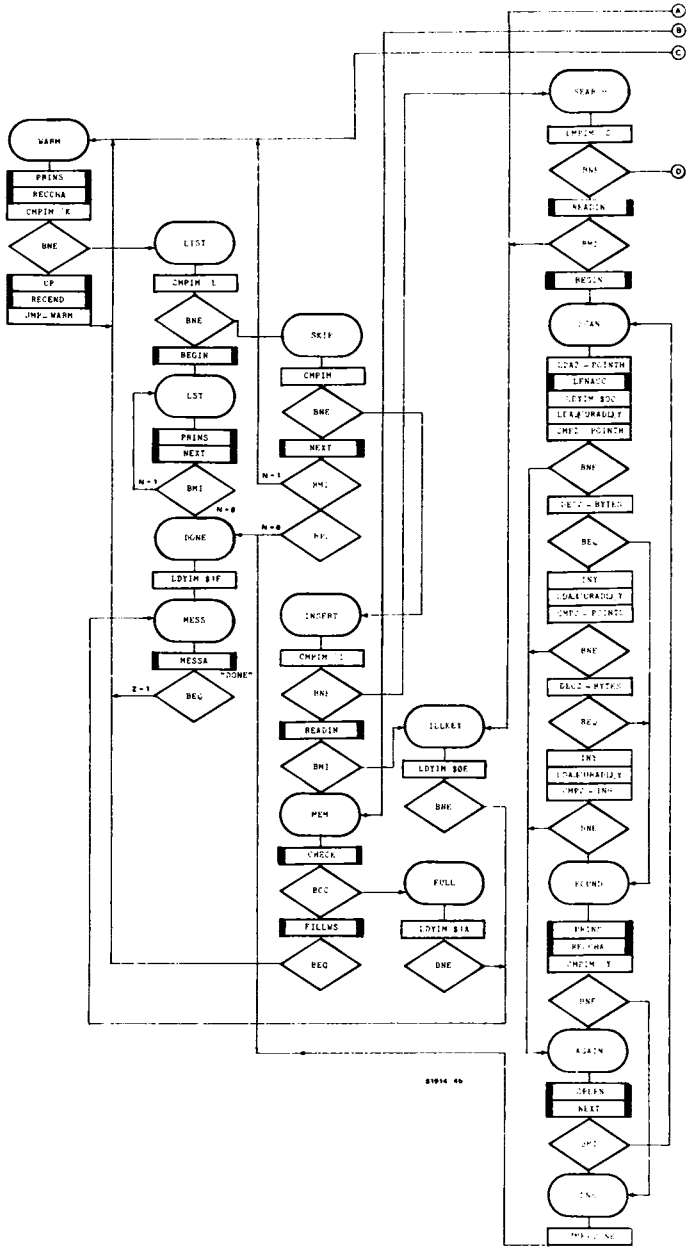


Bild 4b. Erster Teil der PME-Hauptroutine mit den Tastenroutinen K, L, SP (Zwischenraumtaste, nicht "S" und "P"!), I und S.

Subroutine RECEND: Siehe Buch 2, Kapitel 8, Seite 173 und Bild 14.
Den Abschluß der K-Tastenroutine bildet der Rücksprung zum zentralen Label WARM der PME-Hauptroutine. Was getan werden sollte, ist damit erledigt: Die als letzte ausgedruckte Instruktion steht nicht mehr im Speicher, die nachrückenden Instruktionen haben das "Loch" gefüllt. Da der Inhalt von CURAD unverändert bleibt, wird nach der Rückkehr zum Label WARM die Instruktion ausgedruckt, die an die Stelle der gelöschten Instruktion getreten ist.

7 **Die L-Tastenroutine** umfaßt die Instruktionen, die sich in Bild 4b zwischen dem Label LIST und dem Rückmeldungslabel DONE befinden. Aufgabe von Tastenroutine L ist das Ausdrucken einer Instruktionsliste, wobei der Druckvorgang dann beendet werden muß, wenn der Inhalt des Instruktionspointers CURAD mit dem Inhalt von Endadreßpointer CEND übereinstimmt.
Auf das Feststellen der gedrückten L-Taste folgt ein Sprung zur Subroutine BEGIN, so daß die erste Instruktion des auszudruckenden Blocks auf dem Bildschirm bzw. Druckerpapier erscheint; ihr Opcode steht unter Adresse BEGAD im Speicher. Der übrige Teil der L-Tastenroutine ist eine Schleife: sie besteht aus den Subroutinen PRINS und NEXT sowie einer BMI-Verzweigung. Subroutine NEXT entstammt dem Standard-Editor; siehe Buch 2, Seite 165 und Bild 10. Instruktionspointer CURAD wird von NEXT um den in Speicherplatz BYTES stehenden Betrag erhöht, denn BYTES gibt die Länge der zuvor von PRINS ausgedruckten Instruktion an. Außerdem vergleicht Subroutine NEXT den Inhalt von CURAD mit dem Inhalt von CEND. Abhängig vom Ergebnis des Vergleichs werden die N-Flag und die C-Flag wie folgt gesetzt:
CEND $>$ CURAD $\rightarrow$ N = 1, C = Ø
CEND $\leqslant$ CURAD $\rightarrow$ N = Ø, C = 1
Für die N-Flag gilt dies nebenbei bemerkt nur, solange das Ergebnis der beim Vergleich durchgeführten Subtraktion nicht negativer als −127 ist. Anstelle der Carry-Flag kann nach Durchlaufen von Subroutine NEXT auch die N-Flag als Entscheidungskriterium für Verzweigungen dienen.
Nachdem das Schlußlicht, das EOF-Zeichen 77, ausgedruckt ist (es steht unter der Adresse CEND − 1 im Speicher) und noch einmal NEXT durchlaufen wurde, ist CURAD gleich CEND. Der Pseudo-Opcode 77 hat nämlich die Länge von einem BYTE. Die N-Flag wird daraufhin Ø, so daß die PRINS/NEXT-Schleife unterbrochen ist. Der Programmablauf wird beim Label DONE fortgesetzt. Nach Ausdrucken der Rückmeldung "DONE" durch Subroutine MESSA ist die Z-Flag 1, da dann das abschließende EOT-Zeichen im Akku steht (siehe Bild 1). Es folgt ein Sprung zum zentralen Label WARM der PME-Hauptroutine.
Der Rücksprung zum Label WARM bedeutet, daß Subroutine PRINS anschließend noch eine weitere Instruktion ausdruckt. Diese Instruktion, deren Opcode unter Adresse CEND im Speicher steht, gehört nicht mehr zu dem von Subroutine LIST ausgedruckten Instruktionsblock. Sie ist deshalb auch kein Bestandteil des zu editierenden Programms.

8 **Die Leertastenroutine** ist die nächste Station unserer Wanderung durch PME; in Bild 4b beginnt sie beim Label SKIP. Drücken wir

die Leertaste (wir haben sie auch "Zwischenraumtaste" genannt), so wird die Instruktion ausgedruckt, die der zuletzt ausgedruckten Instruktion im Speicher folgt. Im Gegensatz zur L-Tastenroutine wird also von der Leertastenroutine nur eine einzige Instruktion ausgedruckt. Die Adresse dieser Instruktion weist der Instruktionspointer CURAD an. Nach Erkennen der gedrückten Leertaste durchläuft das Programm zuerst die Subroutine NEXT, die bereits im vorangegangenen Abschnitt (Punkt 7) erwähnt wurde. Auf NEXT folgen die Verzweigungen BMI und BPL. Weil diese beiden "Programmweichen" von der N-Flag gestellt werden, steht stets eine von ihnen auf "Springen". Bei N = 1 ist Label WARM das Sprungziel, bei N = Ø wird Label WARM erst über die Rückmeldung "DONE" erreicht. Eine neue Instruktion wird in beiden Fällen ausgedruckt. Solange CURAD noch nicht gleich CEND ist, erscheint die unter der nächsten Adresse im Speicher stehende Instruktion auf dem Bildschirm bzw. Druckerpapier. Ist dagegen CURAD = CEND, so wird nach der Rückmeldung "DONE" die Instruktion ausgedruckt, deren Opcode unter Adresse CEND im Speicher steht. In dieser Hinsicht besteht kein Unterschied zwischen der Leertastenroutine und der L-Tastenroutine (Punkt 7).

9 Bevor wir die Besprechung der PME-Hauptroutine mit der I-Tastenroutine (siehe Punkt 11) fortsetzen, wenden wir uns hier und im nächsten Abschnitt drei PME-Subroutinen zu. Die erste Subroutine dieser Zwischenbetrachtung ist **Subroutine READIN/READ** in Bild 5. Ihre Funktion läßt sich mit der Funktion von Subroutine RDINST des Standard-Editors vergleichen (siehe Buch 2, Seite 166 und Bild 11). READIN/READ verarbeitet nämlich die über die Tastatur eingegebenen Instruktionen, die bei Aktivierung der Insert-, Search- oder Input-Tastenroutine (siehe Punkt 11, 12 und 18 dieses Kapitels) in den Computer gelangen. Ebenso wie innerhalb von RDINST abhängig von der Instruktionslänge ein-, zwei- oder dreimal die Subroutine GETBYT aufgerufen wird, so springt READIN/READ ein-, zwei- oder dreimal zur **Subroutine BYTIN/BYT**. Letztere ist in Bild 6 dargestellt; sie soll zuerst betrachtet werden.
Der Name weist bereits auf die Aufgabe von Subroutine BYTIN/BYT hin: Die Information, die durch Drücken von zwei numerischen Tasten (Ø . . . F) in den Computer gelangt, wird in ein linkes und ein rechtes Datennibble umgewandelt und, zusammengefügt zu einem Datenbyte, in den Akku gesetzt. Aus Bild 6 geht hervor, daß BYTIN mit einem Sprung zur Subroutine RECCHA beginnt: PME ist so lange in Wartestellung, bis eine Taste gedrückt wird. Die nachfolgende PM-Subroutine ASHETT setzt den ASCII-Kode einer gedrückten numerischen Taste in das entsprechende Datennibble um. Gehört die gedrückte Taste nicht zur Gruppe der numerischen Tasten, dann wird ASHETT mit gesetzter N-Flag verlassen.
Einzelheiten zu ASHETT siehe Kapitel 14, Punkt 13 und Bild 12.
Bei N = Ø steht nach dem Durchlauf von ASHETT das aus der Tasteninformation gebildete Nibble rechts im Akku. Vier aufeinanderfolgende Schiebebefehle rücken dieses Nibble im Akku nach links, so daß das ursprünglich rechts im Akku stehende Nibble zum linken Akku-Nibble wird. Danach wandert der Akku-Inhalt vorübergehend in den Speicherplatz NIBBLE. Es folgt ein weiterer Sprung nach RECCHA und anschließend nach ASHETT: das zweite Datennibble entsteht aus der zweiten Tasten-

5

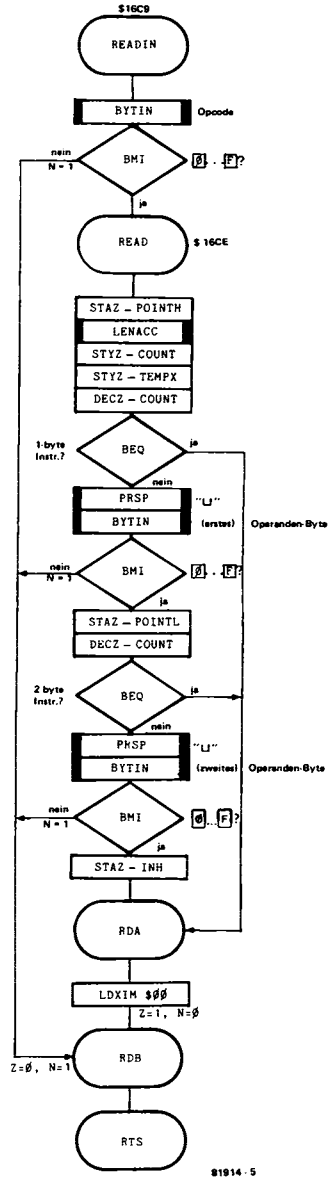


Bild 5. Die Subroutine READIN/READ liest eine eingegebene Instruktion ein. Sie ist Bestandteil der I-, S- und Input-Tastenroutine.

6

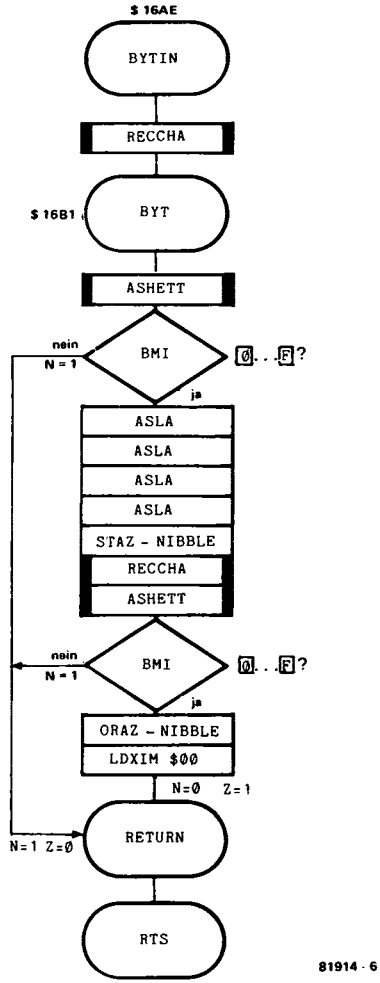


Bild 6. Subroutine BYTIN/BYT liest ein Daten-Byte (zwei Daten-Nibble) ein, die von zwei gedrückten numerischen Tasten stammen. BYTIN/BYT wird von READIN/READ in Bild 5 mindestens einmal aufgerufen.

information. Die Instruktion ORAZ-NIBBLE setzt nun das erste und das zweite Nibble zu einem im Akku stehenden Byte zusammen. Nachdem die Z-Flag gesetzt und die N-Flag rückgesetzt ist, steht dem Rücksprung aus BYTIN/BYT nichts mehr im Weg. In dem Fall, daß versehentlich (oder auch mit Absicht) eine nicht numerische Taste gedrückt wurde, wird BYTIN/BYT mit N = 1 und Z = Ø verlassen.

Subroutine BYT ist mit Subroutine BYTIN bis auf den ersten Sprung nach RECCHA identisch. Von BYT macht die Input-Tastenroutine Gebrauch, die unter Punkt 18 besprochen wird.
Subroutine READIN/READ in Bild 5 beginnt mit der gerade besprochenen Subroutine BYTIN. Danach stehen die ersten beiden Datennibble im Akku; sie verkörpern den Opcode der von der Tastatur kommenden Instruktion. Voraussetzung ist natürlich, daß die N-Flag nach Subroutine BYTIN nicht gesetzt ist, denn in diesem Fall folgt unmittelbar der Rücksprung.
Nach Passieren von Label READ wird der Opcode der eingegebenen Instruktion in Speicherplatz POINTH gesetzt. Subroutine LENACC stellt anschließend die zu erwartende Länge der Instruktion fest und entscheidet damit, wie oft BYTIN innerhalb von READIN durchlaufen werden muß. Die Instruktionslänge, die am Ende von LENACC im Y-Register steht, wird in die Speicherplätze TEMPX und COUNT kopiert.
Wenn die Instruktion zwei oder drei Byte umfaßt, druckt Subroutine PRSP einen Zwischenraum aus, und Subroutine BYTIN wartet anschließend auf die Eingabe des ersten Operanden-Byte. Dieses Byte wird in Puffer POINTL abgelegt. Ist die Instruktion drei Byte lang, dann wiederholt sich das Spiel; das zweite Operanden-Byte findet seinen Platz in Puffer INH. Schließlich erhält das X-Register noch den Wert ØØ (Label RDA in Bild 5), so daß Z = 1 und N = Ø ist. Wurde dagegen zu irgendeinem Zeitpunkt eine nicht numerische Taste gedrückt, dann folgt der Rücksprung aus READIN/READ mit N = 1 und Z = Ø.
Subroutine READ ist mit Subroutine READIN bis auf den ersten Sprung nach BYTIN identisch. Von READ macht die Input-Tastenroutine Gebrauch, die unter Punkt 18 besprochen wird.

10 Die dritte und letzte Subroutine in unserer Zwischenbetrachtung ist **die Subroutine CHECK**. Sie klärt die Frage, ob für eine neue, an- oder einzufügende Instruktion noch genügend Platz im Speicher vorhanden ist. Bild 1 in Kapitel 13 machte bereits deutlich, daß insgesamt vier Speicherplätze frei bleiben müssen. Drei dieser vier Speicherplätze nehmen das erste gefundene Label während der ersten Assemblierphase auf, während auf dem vierten Speicherplatz das EOF-Zeichen 77 steht. Dies bedeutet, daß der variable Endadreßpointer CEND dann seinen höchstmöglichen Stand erreicht, wenn er auf die Adresse zeigt, die um zwei Adressen niedriger als Endadresse ENDAD liegt. Würde CEND nach Ein- oder Anfügen der neuen Instruktion einen höheren Stand einnehmen, so reicht der Platz für die neue Instruktion nicht mehr aus. Es bleibt dann nichts anderes übrig als Endadresse ENDAD zu erhöhen, sofern dies möglich ist.
Nach Durchlaufen von Subroutine CHECK zeigt die Carry-Flag an, wie es um die verfügbare Speicherkapazität bestellt ist. Das Flußdiagramm von CHECK ist in Bild 7 gezeichnet. Am Anfang steht Subroutine ADCEND, eine Subroutine des Standard-Editors. ADCEND, dargestellt in Buch 2, Kapitel 8, Seite 170, erhöht den variablen Adreßpointer CEND um die Anzahl der Byte, die die neue Instruktion aufweist. Die Anzahl der Byte steht vor dem Sprung nach CHECK in Speicherplatz BYTES.
Nun wird von ENDAD zuerst der Wert Ø2 und dann der Wert von CEND abgezogen. Vom Ergebnis der Subtraktion hängt der Zustand der Carry-

7

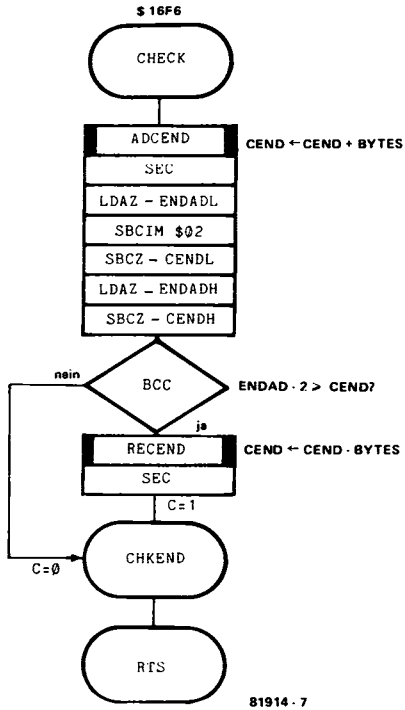


Bild 7. Bevor eine neue Instruktion ein- oder angefügt werden kann, prüft Subroutine CHECK, ob dafür in dem durch BEGAD und ENDAD begrenzten Bereich noch Platz vorhanden ist.

Flag ab: C = Ø bedeutet, daß ENDAD niedriger als CEND + 2 ist. In diesem Fall reicht der Platz für die neue Instruktion nicht mehr aus, so daß mit C = Ø der Rücksprung folgt. Wenn dagegen das Subtraktionsergebnis eine positive Zahl oder Null ist, wird die Carry-Flag gesetzt; für die neue Instruktion ist dann noch Platz vorhanden. Vor dem Rücksprung mit C = 1 wird noch Subroutine RECEND, eine Verwandte von ADCEND aufgerufen. Näheres über RECEND steht in Buch 2 auf Seite 173. Der variable Endadreßpointer CEND erhält durch RECEND seinen ursprünglichen Wert zurück. Das ist notwendig, weil (bei C = 1) unmittelbar im Anschluß an CHECK das Laden der neuen Instruktion in den Speicher beginnt (siehe I-Tastenroutine, Punkt 11).
Anmerkung: Wenn Subroutine CHECK mit C = Ø verlassen wird, findet keine Korrektur von CEND durch Subroutine RECEND statt. Einerseits bleibt dann das EOF-Zeichen 77 an seinem Platz; die neue Instruktion gelangt ja nicht in den Programmspeicher, da die Kapazität nicht ausreicht. Andererseits zeigt aber CEND auf eine höhere Adresse als vorher, also

nicht mehr auf die Adresse, die um eine höher als die Adresse mit dem EOF-Zeichen 77 liegt. Wie sich dies ändern läßt, ist unter Punkt 20 beschrieben.

11 Kehren wir nun zum ersten Teil der PME-Hauptroutine in Bild 4b zurück. Die nächste Routine ist **die I-Tastenroutine**; sie beginnt beim Label INSERT. Wenn Taste I gedrückt wurde, dann wird zuerst Subroutine READIN aufgerufen, die wir bereits unter Punkt 9 besprochen haben. Nach dem Rücksprung aus READIN ist eine der beiden folgenden Situationen gegeben:
Entweder N = Ø, Z = 1:
In POINTH steht der Opcode der einzufügenden Instruktion;
in POINTL steht gegebenenfalls der erste Operand der einzufügenden Instruktion;
in INH steht gegebenenfalls der zweite Operand der einzufügenden Instruktion.
Oder N = 1, Z = Ø:
Während der Eingabe wurde eine nicht numerische Taste gedrückt.
Im zweiten Fall führt die BMI-Verzweigung, die sich an Subroutine READIN anschließt, zum Label ILLKEY. Nach Ausdrucken der Fehlermeldung "ILLEGAL KEY" folgt der Rücksprung zum zentralen Label WARM.
Wurde dagegen die Instruktion ordnungsgemäß eingegeben, dann passieren wir nun das Label MEM. Subroutine CHECK wird aufgerufen; sie stellt wie unter Punkt 10 beschrieben fest, ob für die eingegebene Instruktion noch genügend Platz vorhanden ist. Sollte das nicht der Fall sein, so ist das Label FULL das nächste Sprungziel, und "FULL" lautet dann auch die nächste Rückmeldung. Wenn der Platz im Speicher ausreicht, springt PME zur Subroutine FILLWS. Diese Subroutine gehört ebenfalls zum Standard-Editor: siehe Buch 2, Kapitel 8, Bild 12. FILLWS sorgt dafür, daß für die neue Instruktion Platz geschaffen wird (Subroutine DOWN). Nachdem der variable Endadreßpointer CEND angepaßt ist, werden in einen, zwei oder drei Speicherplätze die Inhalte von POINTH (Opcode), POINTL (erster Operand) und INH (zweiter Operand) geladen. Die Anzahl der Speicherplätze hängt natürlich von der Länge der neuen Instruktion ab. Da am Schluß von Subroutine FILLWS die Z-Flag gesetzt ist, führt die anschließende BEQ-Verzweigung zum zentralen PME-Label WARM. Instruktionspointer CURAD blieb unverändert, so daß nach Label WARM die eingefügte Instruktion ausgedruckt wird.
Der Teil der I-Tastenroutine, der beim Label MEM beginnt, ist gleichzeitig der zweite Teil der Input-Tastenroutine (siehe Punkt 18).

12 **Die S-Tastenroutine** umfaßt eine längere Liste von Instruktionen: sie reicht in Bild 4b vom Label SEARCH bis zu der unter dem Label DNE stehenden Instruktion. Wir beginnen beim Label SEARCH. Nach Erkennen der gedrückten S-Taste folgt ein Sprung zur Subroutine READIN aus Punkt 9. Das "Muster" der zu suchenden Instruktion steht daher in Speicherplatz POINTH (Opcode) und abhängig von der Instruktionslänge auch in POINTL (erster Operand) und INH (zweiter Operand). Wenn eine nicht numerische Taste gedrückt wurde, führt die BMI-Ver-

zweigung hinter READIN zum Label ILLKEY, so daß die Fehlermeldung "ILLEGAL KEY" erscheint.
Das Label SCAN ist der Startpunkt der eigentlichen Suchaktion. Der Opcode der aufzuspürenden Instruktion wird in den Akku geladen, und Subroutine LENACC bestimmt die Länge der Instruktion. Die Anzahl der Byte ist nach LENACC in Speicherplatz BYTES enthalten. Fast hätten wir es vergessen: Vor Erreichen des Labels SCAN wurde noch Subroutine BEGIN aufgerufen, so daß Instruktionspointer CURAD auf Beginnadresse BEGAD gerichtet ist. Die Suchaktion startet deshalb bei der ersten Instruktion oder dem ersten Label des abzusuchenden Bereichs. Nach LENACC wird der Opcode der ersten Instruktion in den Akku geladen (LDA-(CURADL),Y) und mit dem Opcode des in POINTH stehenden "Musters" verglichen. Stimmen die beiden Opcodes nicht überein, dann steht bereits fest, daß dies nicht die gesuchte Instruktion ist. Die BNE-Verzweigung führt zum Label AGAIN, so daß die nächste im Speicher stehende Instruktion an der Reihe ist.

Bei Übereinstimmung der beiden Instruktionen geschieht dagegen folgendes: Durch Herabsetzen des in BYTES stehenden Werts um Eins und einen anschließenden BEQ wird geprüft, ob das "Muster" eine Ein-Byte-Instruktion ist oder ob zu ihr mehrere Byte gehören. Ist die zu suchende Instruktion tatsächlich nur ein Byte lang, so springt das Programm zum Label FOUND: die gesuchte Instruktion ist gefunden, sie wird zusammen mit ihrer Adresse ausgedruckt. Handelt es sich jedoch um eine Mehr-Byte-Instruktion, dann wird das nächste im Speicher stehende Byte in den Akku geladen. Dieses Byte muß nicht zwangsläufig der erste oder einzige Operand der überprüften Instruktion sein, weil diese ja möglicherweise aus nur einem einzigen Byte (dem Opcode) bestehen kann. Auf jeden Fall wird aber das zweite im Speicher stehende Byte mit dem ersten oder einzigen Operand der zu suchenden Instruktion, oder genauer,: mit dem in POINTL stehenden "Muster" verglichen. Auch hier gibt es wieder zwei Möglichkeiten: Keine Übereinstimmung? Dann springe zum Label AGAIN. Übereinstimmung? In diesem Fall prüfe, ob die zu suchende Instruktion (das "Muster") aus zwei Byte besteht. Wenn ja: springe nach FOUND; lautet die Antwort nein, so lade das nächste im Speicher stehende Byte in den Akku und vergleiche es mit dem zweiten Operand des "Musters". Wenn auch diese beiden Byte gleich sind, haben wir ohne Sprünge das Label FOUND erreicht.
Zwischenbemerkung: Wir hätten den zwischen den Labeln SCAN und FOUND liegenden Teil der Routine auch anders gestalten können. Nach der Feststellung, daß beide Opcodes übereinstimmen, wäre auch ein Vergleich der beiden Instruktionslängen mit anschließendem Sprung nach AGAIN bei Ungleichheit möglich gewesen. Dadurch hätten wir zwar Zeit gespart, dies wäre jedoch auf Kosten der für PME insgesamt benötigten Speicherkapazität gegangen. Außerdem ist der von uns gewählte Algorithmus wirklich "wasserdicht".
Zurück zur S-Tastenroutine. Nach Label FOUND wird Subroutine PRINS aufgerufen; sie druckt die gesuchte und inzwischen gefundene Instruktion zusammen mit ihrer Adresse aus. Wie schon in Kapitel 13 beschrieben, ist nun die Fortsetzung der Suchaktion nach weiteren, gleichartigen Instruktionen möglich. Dazu muß Taste Y gedrückt werden. Auf PRINS folgt

deshalb ein Sprung zur Subroutine RECCHA. Ist die gedrückte Taste die Y-Taste, dann erreichen wir auf direktem Weg das Label AGAIN, um die nächste im Speicher stehende Instruktion zu überprüfen. Nach Drücken einer anderen Taste führt die BNE-Verzweigung zum Label DNE. Ausgedruckt wird dann die Rückmeldung "DONE"; sie ist das Zeichen dafür, daß wir die S-Tastenroutine verlassen haben.
Auf Label AGAIN folgt ein Aufruf der Subroutinen OPLEN und NEXT. Während OPLEN die Länge der zuletzt überprüften Instruktion feststellt, gibt NEXT dem Instruktionspointer CURAD einen solchen Wert, daß dieser auf die im Speicher folgende Instruktion zeigt. Ob tatsächlich noch eine weitere Instruktion vorhanden ist, gibt die N-Flag nach der Rückkehr aus NEXT an. Dies wurde ausführlich im Zusammenhang mit der L-Tastenroutine unter Punkt 7 besprochen. Sind keine weiteren Instruktionen vorhanden, die noch überprüft werden könnten, dann ist das Label DNE mit der Rückmeldung "DONE" das nächste Ziel.
Aus dem Flußdiagramm der S-Tastenroutine in Bild 4b geht hervor, daß diese Routine nur über das Label DNE verlassen werden kann. Das stimmt mit dem überein, was in Kapitel 13 steht: Erst nach Erscheinen der Rückmeldung "DONE" ist die Suchaktion beendet.

13 **Die Z-Tastenroutine** besteht aus den Instruktionen, die links in Bild 4c unter dem Label BACK stehen. Wenn die Routine feststellt, daß die Z-Taste gedrückt ist, wird der Inhalt von BEGAD in Adreßpointer TABLE kopiert. TABLE spielte bisher nur beim Assembler eine Rolle (siehe Buch 2, Kapitel 9). Wir benötigen diesen Speicherplatz hier jedoch als Hilfsadreßpointer, wenn wir die Länge der Instruktion bestimmen, die der zuletzt ausgedruckten Instruktion im Speicher vorangeht. Die Länge dieser Instruktion muß vom Inhalt des Instruktionspointers CURAD abgezogen werden, damit nach dem Rücksprung zum Label WARM die vorangehende Instruktion ausgedruckt werden kann. Dies ist ja die Funktion der Z-Taste innerhalb des Systemprogramms PME.
Nachdem der Inhalt von BEGAD auch in TABLE steht, werden die Inhalte von TABLE und CURAD miteinander verglichen. Dieser Vergleich stellt fest, ob die zuletzt ausgedruckte Instruktion die erste Instruktion des editierten Speicherbereichs ist. Trifft dies zu, dann existiert für PME keine vorangehende Instruktion; es folgt ein Sprung über Label BCKB zurück zum Label WARM. Da CURAD unverändert bleibt, wird nun die letzte Instruktion noch einmal ausgedruckt.
Ganz anders verläuft das Programm, wenn CURAD nicht auf BEGAD, sondern auf eine andere Adresse gerichtet ist: Jetzt wird die Schleife um das Label BCKA durchlaufen. Dabei wird der Inhalt von TABLE so lange um den Betrag erhöht, den die Länge der gerade untersuchten Instruktion angibt, bis TABLE und CURAD übereinstimmen. Anschließend muß CURAD um den Betrag herabgesetzt werden, der als letzter in Speicherplatz BYTES steht. Um das genannte Ziel zu erreichen, wird nach Label BCKA zuerst der Opcode der zu untersuchenden Instruktion in den Akku geladen; anschließend bestimmt Subroutine LENACC die Länge dieser Instruktion. Den Inhalt von TABLE erhöht **Subroutine INCTAB** (Bild 8b) um den Betrag der Instruktionslänge.
Dann folgt der Vergleich des erhöhten Inhalts von TABLE mit dem Inhalt

4c

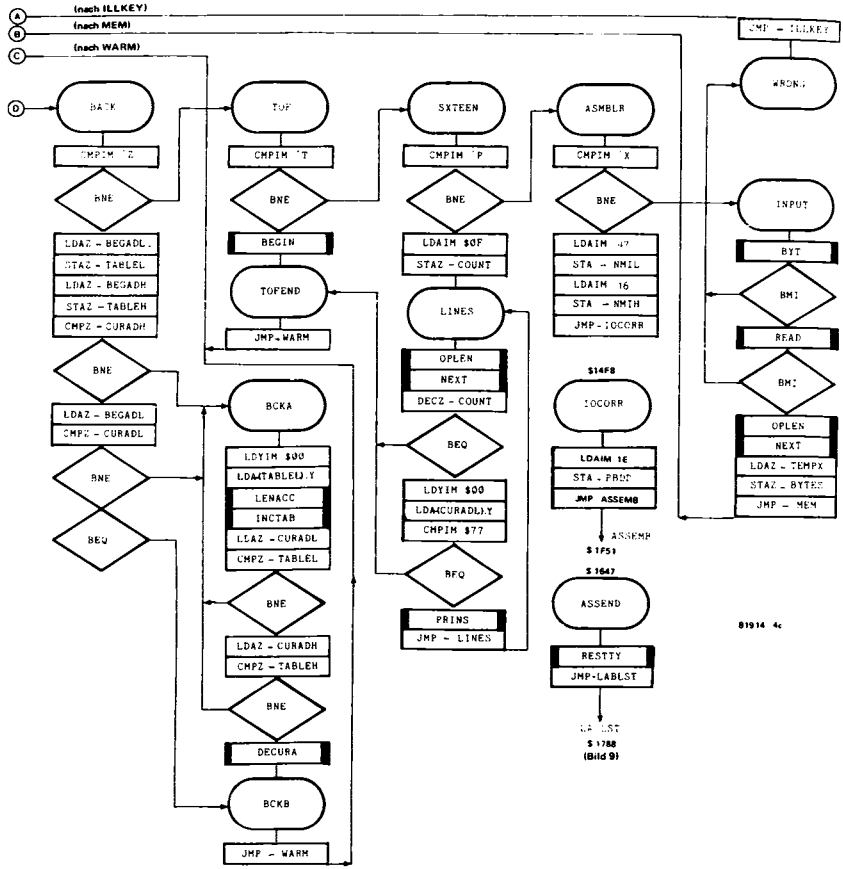


Bild 4c. Zweiter Teil der PME-Hauptroutine; er umfaßt die Tastenroutinen Z, T, P und X sowie die Input-Tastenroutine (Tasten 0 . . . F).

von CURAD. Stimmen beide noch nicht überein, so wird die beim Label BCKA beginnende Schleife noch einmal durchlaufen. Bei gleichem Stand von TABLE und CURAD springt das Programm dagegen zur **Subroutine DECURA** in Bild 8a. Sie setzt den Stand von Instruktionspointer CURAD um den in Speicherplatz BYTES stehenden Betrag herab. Damit sind alle notwendigen Voraussetzungen erfüllt, die vor dem Rücksprung zum zentralen Label WARM geschaffen werden mußten: Nach Passieren von Zwischenlabel BCKB und dem Rücksprung nach WARM wird die vorangegangene Instruktion ausgedruckt.

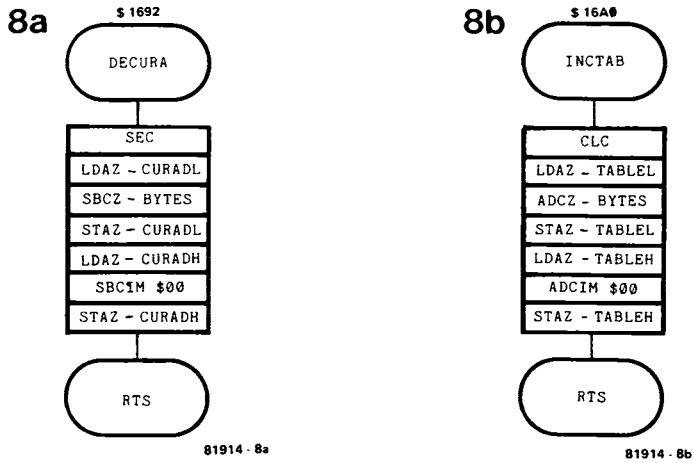


Bild 8. Diese beiden Subroutinen spielen dann eine Rolle, wenn durch Drücken der Z-Taste die Instruktionen hinsichtlich ihrer Adresse rückwärts durchlaufen werden.

14 **Die T-Tastenroutine** bedarf nur einer kurzen Erläuterung. Zu ihr gehören die Instruktionen, die in Bild 4c auf das Label TOF folgen. Wenn Taste T gedrückt wird, springt PME zur Subroutine BEGIN. Instruktionspointer CURAD wird auf BEGAD gerichtet, so daß nach der Rückkehr zum zentralen Label WARM über das Zwischenlabel TOFEND die erste Instruktion auf dem Bildschirm oder Druckerpapier erscheint. Natürlich kann das editierte Programm auch mit einem Label beginnen.

15 Wir kommen zur **P-Tastenroutine**, die alle Instruktionen umfaßt, die sich in Bild 4c an das Label SXTEEN anschließen. Nach Drücken der P-Taste erhält der als Instruktionszähler verwendete Speicherplatz COUNT den Wert $ØF (dezimal: 15). Hierdurch werden außer der zuletzt gedruckten Instruktion (das P erscheint in der gleichen Zeile auf der rechten Seite) 15 weitere Instruktionen ausgedruckt. Insgesamt stehen dann 16 Instruktionen auf dem Bildschirm bzw. Druckerpapier. Während 14 Instruktionen beim Durchlaufen der Schleife ausgedruckt werden, die beim Label LINES beginnt, ist für die 15. Instruktion der auf das Label WARM folgende Aufruf von Subroutine PRINS zuständig.
Im einzelnen: Nach Label LINES bestimmt Subroutine OPLEN die Länge der bereits ausgedruckten Instruktion. Der Inhalt von Instruktionspointer CURAD wird von Subroutine NEXT entsprechend angepaßt; anschließend wird COUNT um Eins herabgesetzt. Wenn COUNT dadurch seinen Endwert Null erreicht, springt das Programm infolge des BEQ über Zwischenlabel TOFEND zurück zum Label WARM; es wird dann die 15. Instruktion ausgedruckt. Solange COUNT noch nicht auf Null steht, folgt ein Vergleich zwischen dem Opcode der auszudruckenden Instruktion mit der Zahl 77 (dem EOF-Zeichen). Die Routine stellt dadurch fest, ob bereits das Ende des editierten Programms erreicht wurde. Ist das der Fall, dann wird das

EOF-Zeichen 77 als letzte "Instruktion" ausgedruckt: Rücksprung über TOFEND nach WARM.
Besteht kein Anlaß zu diesem vorzeitigen Ende, so druckt nach dem Vergleich mit dem EOF-Zeichen die auf den zweiten BEQ folgende Subroutine PRINS die Instruktion aus; anschließend springt PME zurück zum Label LINES. Die ersten vier auf Label LINES folgenden Instruktionen werden 15 mal durchlaufen, weil erst danach der Inhalt von COUNT Null ist. Der restliche Teil der Schleife hat 14 Durchläufe zu verzeichnen, so daß Subroutine PRINS 14 mal ohne den Umweg über Label WARM aufgerufen wird. Zum Label WARM springt die Routine beim 15. Durchlauf der Schleife; der nach Label WARM folgende Aufruf von PRINS druckt die letzte Instruktion aus. Voraussetzung ist natürlich, daß nicht inzwischen das EOF-Zeichen 77 erreicht wurde.

16 **Die X-Tastenroutine** besteht aus den Instruktionen in Bild 4c, die sich unmittelbar an das Label ASMBLR anschließen, sowie aus den drei auf Label IOCORR folgenden Instruktionen. Aufgabe der X-Tastenroutine ist die Vorbereitung auf das Assemblieren des editierten Programms. Nach Feststellen der gedrückten X-Taste wird der NMI-Sprungvektor auf die zum Label ASSEND gehörende Adresse gerichtet. Sobald das Programm assembliert ist und danach die Taste ST/NMI auf der Standard-Tastatur gedrückt wird, folgt der Rücksprung nach PME zum Label ASSEND. Was danach geschieht, ist im nächsten Abschnitt (Punkt 17) beschrieben.
Nach der Spezifizierung des NMI-Sprungvektors springt die Routine zum Label IOCORR. Das Richtungsregister PBDD des Ports B erhält den gleichen Inhalt, wie er nach Durchlaufen der RESET-Vorroutine des Standard-Monitors vorhanden ist: siehe Buch 2, Kapitel 7. Die übrigen Register der I/O-Einheit brauchen nicht verändert zu werden. Anschließend steht für PME der Weg offen, zum Assembler zu springen (JMP-ASSEMB), so daß das editierte Programm jetzt assembliert wird. Wie dies im einzelnen vor sich geht, ist ausführlich in Buch 2, Kapitel 9 beschrieben.

17 **Nach dem Assemblieren** beginnt die Erprobungsphase des editierten Programms. Bevor es sich der ersten Bewährungsprobe stellt, werden noch sämtliche im Programm enthaltene Label ausgedruckt. Wie PME dies bewerkstelligt, das wird nachfolgend erklärt.
Eingeleitet wurde die Prozedur des Assemblierens durch Drücken der X-Taste (siehe Kapitel 13). Wenn die sechs Siebensegment-Displays auf der Basisplatine wieder aufleuchten, ist das Assemblieren beendet, es muß dann die Taste ST auf der Standard-Tastatur gedrückt werden. Der dadurch ausgelöste NMI führt zum Label ASSEND rechts unten in Bild 4c. Als erste wird dort Subroutine RESTTY aufgerufen, die innerhalb von PM den ursprünglichen Zustand der I/O-Register nach Durchlaufen der Kassetten-Subroutinen (DUMP oder RDTAPE) wiederherstellt. Einzelheiten über RESTTY sind unter Punkt 27 und 28 in Kapitel 14 zu finden. Nach der Rückkehr vom Assembler braucht zwar nur das Register PBDD korrigiert zu werden (vgl. Punkt 16), die notwendigen Einzelinstruktionen würden jedoch mehr Byte füllen als der Aufruf von RESTTY.
Die Labelliste wird von der **Subroutine LABLST** in Bild 9 ausgedruckt, zu

der PME nun springt. Während dieser Routine erhält der Adreßpointer TABLE, der wie unter Punkt 13 beschrieben zwischenzeitlich als Hilfsadreßpointer für andere Zwecke "entliehen" wurde, seine ursprüngliche Funktion als Teil des Tabellen-Endadreßpointers zurück. Zu Beginn der ersten Assemblierphase wird in TABLE der Wert (ENDAD) – FF gesetzt. Der Anfangswert des Speicherplatzes LABELS ist FF. Tabellenpointer TABLE + LABELS bestimmt, auf welchen Speicherplatz des Labelspeicherbereichs eine Labelnummer mit den zugehörigen Adreßteilen ADH und ADL gespeichert wird. In seiner Ausgangsstellung ist der Tabellenpointer auf Endadresse ENDAD gerichtet. Nach Verarbeitung eines Labels (Finden und Speichern) wird der Inhalt von Speicherplatz LABELS um $Ø3 herabgesetzt. Soweit einige Wiederholungen aus Buch 2, Kapitel 9 als Gedächtnisstütze.

Zurück zur Routine LABLST in Bild 9 dieses Kapitels. Die Zwischenwerte von Speicherplatz LABELS stehen hier im Y-Register; in dieses Register wird daher zuerst der Wert FF geladen. Der Inhalt von TABLE ist, wie schon erwähnt, gleich (ENDAD) – FF. Die auf Label LBLSTB folgenden Instruktionen dienen zum Ausdrucken der verschiedenen Label-Bestandteile in der richtigen Reihenfolge, wobei jedesmal nach Drucken einer Komponente das Y-Register dekrementiert wird (DEY). Vor Label LBLSTB wurde noch die Druckposition an den Anfang einer neuen Zeile geführt (CRLF), und das als Labelzähler dienende X-Register erhielt den Wert $Ø4, so daß in jeder Zeile maximal vier Label stehen.

Zweimal nach Label LBLSTB wird der Inhalt des Y-Registers in Speicherplatz TEMPX zwischengespeichert. Das ist notwendig, weil PME zweimal einen Text ausdrucken muß und hierzu ebenfalls das Y-Register benötigt. Dem Ausdrucken des ersten Textes ("LAB $") folgt das Laden der Labelnummer in den Akku, die während der ersten Assemblierphase gefunden und gespeichert wurde. Zuständig ist hierfür die Instruktion LDA-(TABLEL), Y, wobei der Inhalt von Y gleich FF minus einem ganzzahligen Vielfachen von Ø3 ist. Während des ersten Durchlaufs durch LBLSTB ist dieser Faktor gleich Null, während des letzten Durchlaufs ist er gleich der Gesamtanzahl der Label.

Nach Ausdrucken der Labelnummer durch PRBYT folgt der Text ": $". Danach wird zuerst das linke Adreßbyte ADH und dann das rechte Adreßbyte ADL des Labels ausgedruckt. Es folgt ein Zwischenraum (PRSP), der die auf gleicher Zeile stehenden Label voneinander trennt, und anschließend wird geprüft, ob noch weitere Label auszudrucken sind (CPYZ-LABELS und BEQ). Ist dies der Fall, so wird X dekrementiert (DEX). Der im X-Register stehende Wert gibt darüber Auskunft, wieviele Label noch auf die Zeile passen. Abhängig davon springt die Routine entweder zurück zum Label LBLSTA (neuer Zeilenanfang) oder zum Label LBLSTB (Weiterführen der gleichen Zeile).

Wenn sämtliche Label ausgedruckt sind, folgt ein Sprung zum Label LBLSTC. Nach Setzen von Instruktionspointer CURAD auf die erste Instruktion des assemblierten Programms durch Subroutine BEGIN springt die Routine zum Label EDITW in Bild 4a. Auf dem Bildschirm oder Druckerpapier erscheint nun die Rückmeldung "PM EDITOR" und daran anschließend die erste Instruktion des assemblierten Programms.

Noch einmal zurück zur Instruktion CPYZ-LABELS vor dem ersten BEQ

9

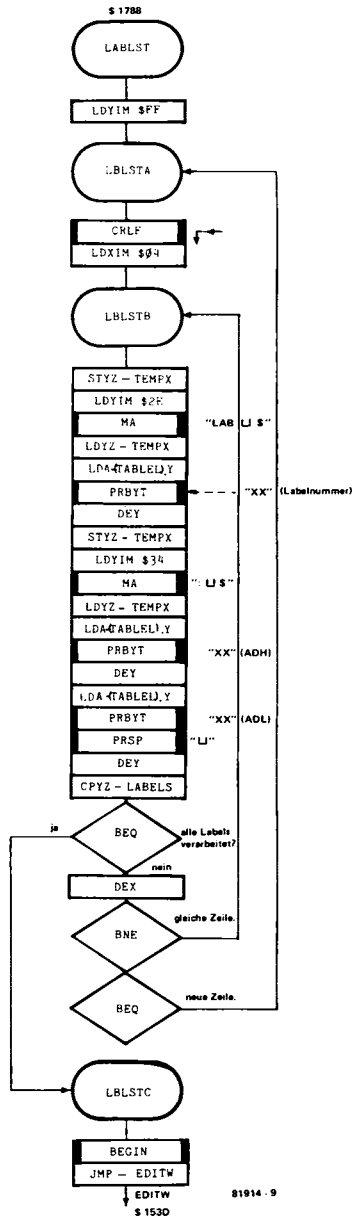


Bild 9. Subroutine LABLST druckt eine Liste der Label aus, die während der Assemblierung eines Programms gefunden und verarbeitet wurden. In jeder Zeile erscheinen maximal vier Label mit ihrer Labelnummer und ihrer Adresse. Die Reihenfolge richtet sich nach der Reihenfolge der Adressen.

in Bild 9. Was geschieht hier im einzelnen? Es geschieht folgendes: Jedesmal wenn der Assembler ein Label findet und speichert, setzt er den Inhalt von Speicherplatz LABELS (Anfangswert: FF) um $Ø3 herab. Innerhalb von Subroutine LABLST in Bild 9 wird der Inhalt des Y-Registers dreimal um $Ø1 herabgesetzt (DEY). Der Endstand des Y-Registers nach Ausdrucken sämtlicher Label stimmt deshalb mit dem Stand von LABELS überein, den LABELS nach Finden und Speichern sämtlicher im assemblierten Programm vorkommender Label hat. Siehe hierzu auch Buch 2, Seite 191, Bild 4.

18 Am Schluß unserer Betrachtungen der PME-Hauptroutine steht die **Input-Tastenroutine**. Wie schon in Kapitel 13 beschrieben, braucht keine (nicht numerische) Funktionstaste gedrückt zu werden, um die Input-Tastenroutine zu aktivieren. Das Eingeben und Anfügen von Instruktionen ist einfach durch Drücken von mindestens zwei numerischen Tasten möglich. Wenn nach Drücken einer numerischen Taste das Label INPUT, das letzte Label der PME-Hauptroutine in Bild 4c, erreicht wird, steht im Akku der ASCII-Kode dieser Taste. Numerische Daten werden nämlich immer als Teil einer in den Speicher zu setzenden Instruktion verarbeitet, es sei denn, daß vorher Taste I oder Taste S gedrückt wurde. In den beiden zuletzt genannten Fällen wird nicht die Input-, sondern die I- bzw. S-Tastenroutine durchlaufen (siehe Punkt 11 und 12 dieses Kapitels).
Zur Input-Tastenroutine gehören die Instruktionen in Bild 4c, über denen das Label INPUT steht. Sie beginnt mit einem Aufruf von Subroutine BYT, die bis auf den ersten Sprung nach RECCHA mit Subroutine BYTIN identisch ist. RECCHA wird hier nicht benötigt, weil diese Tastenabfrageroutine bereits nach dem zentralen Label WARM aufgerufen wurde (siehe Bild 4b; BYTIN/BYT: siehe Bild 6).
Nach BYT steht der Opcode der neu hinzukommenden Instruktion im Akku. Wenn eine nicht numerische Taste gedrückt wurde, führt der auf BYT folgende BMI über Zwischenlabel WRONG zum Label ILLKEY, so daß die Fehlermeldung "ILLEGAL KEY" erscheint. Wurde dagegen eine numerische Taste gedrückt, so folgt ein Aufruf von Subroutine READ; sie ist abgesehen von den ersten beiden Instruktionen mit Subroutine READIN identisch (siehe Bild 5 und Punkt 9). READ setzt zuerst den gerade eingelesenen Opcode in Speicherplatz POINTH. Danach wird der restliche Teil der Instruktion eingelesen, wobei auch hier ein BMI zur Fehlermeldung "ILLEGAL KEY" bei der Eingabe nicht numerische Daten führt.
Anschließend ist wieder das inzwischen vertraute Subroutinenpaar OPLEN/NEXT an der Reihe: Instruktionspointer CURAD wird um den Betrag erhöht, der der Länge der zuletzt ausgedruckten Instruktion (nicht der neu hinzukommenden Instruktion!) entspricht. Die neue Instruktion wird also, sofern sie Aufnahme in den Speicher findet, dort unmittelbar an die zuletzt ausgedruckte Instruktion angereiht. Speicherplatz BYTES enthält die Länge der neuen Instruktion, nachdem der Inhalt von TEMPX, zustandegekommen durch READ/READIN, in BYTES kopiert ist. Danach folgt ein Sprung zum Label MEM der I-Tastenroutine in Bild 4b. Was dort noch geschieht, ist bereits unter Punkt 11 beschrieben worden.
Eine letzte Randbemerkung: Instruktionspointer CURAD wird unabhängig

davon erhöht, ob für die neue Instruktion im Speicher Platz vorhanden ist oder nicht. Deshalb wird nach der Rückmeldung "FULL" die Instruktion ausgedruckt, auf die CURAD nach seiner Erhöhung zeigt.

19 Die zum **lauwarmen Start von PME** gehörende Vorroutine ist in Bild 4a gezeichnet; sie beginnt beim Label SEMIW. Bis zum BMI stimmt diese Vorroutine vollständig mit Vorroutine EDITC (siehe Punkt 4) überein: Von Subroutine RESIN werden die Eingangspuffer gelöscht, und nach der Rückmeldung "BEGAD, ENDAD:" folgt die Eingabe dieser Parameter über die Tastatur (Subroutine INPAR). Die BMI-Verzweigung führt zum Anfang zurück, wenn die Eingabe fehlerhaft war. Nach Korrekter Eingabe werden in die Speicherplätze BEGADL, BEGADH, ENDADL und ENDADH die eingegebenen Parameter geladen. Ferner wird der variable Endadreßpointer CEND auf Endadresse ENDAD gerichtet, und Instruktionspointer CURAD zeigt infolge des Aufrufs von Subroutine BEGIN auf Beginnadresse BEGAD. Im Anschluß an den Sprung zum Label BRK der Subroutine EDITC wird zuerst der BRK-Sprungvektor spezifiziert und dann die Rückmeldung "PM EDITOR" ausgedruckt. Danach erreicht auch diese Vorroutine das zentrale Label WARM der PME-Hauptroutine.
Der lauwarme Starteingang SEMIW von PME ist insbesondere für die Analyse von betriebsfertigen Programmen geeignet. Dabei spielt es keine Rolle, ob diese Programme in einem EPROM oder im RAM stehen. Die Tastenfunktionen SP, S, L und P leisten hier wertvolle Hilfe. Bei der Besprechung der zugehörigen Tastenroutinen in diesem Kapitel wurde deutlich, daß die Editierung eines im Speicher (RAM oder EPROM) stehenden Programms bei der Adresse endet, auf die der variable Endadreßpointer CEND zeigt. Beim Eintritt in PME über den lauwarmen Starteingang wird jedoch CEND automatisch gleich ENDAD gesetzt, ENDAD aber ist frei wählbar!
Übrigens: Um einen lauwarmen Start ging es bereits beim Standard-Editor in Buch 2, Kapitel 5. Dort mußte allerdings eine ganze Serie von Puffern einzeln in "Handarbeit" geladen werden.

20 Wir kommen zum Schluß dieses Kapitels. Die Revue der PME-Routinen beschließt die Vorroutine, die zum **warmen CEND-Starteingang** gehört. Aus ihrem Flußdiagramm in Bild 10 (Label SEACND) geht hervor, daß auch sie bis zum ersten BMI mit Vorroutine EDITC in Bild 4a identisch ist. Unsere Betrachtung kann deshalb gleich beim Label SCNDA einsetzen.
Nachdem Instruktionspointer CURAD auf die erste Adresse BEGAD gerichtet ist (JSR-BEGIN), wird der Opcode der untersuchten Instruktion in den Akku geladen und dort mit dem Pseudo-Opcode 77 verglichen. Handelt es sich tatsächlich um das EOF-Zeichen 77, so springt die Routine zum Label SCNDB. Anderenfalls wird der Inhalt von CURAD um die Länge der zuletzt untersuchten Instruktion erhöht (LENACC und NEXT), so daß die nächste Instruktion auf Übereinstimmung mit dem Pseudo-Opcode 77 überprüft werden kann.
Wenn das EOF-Zeichen 77 gefunden ist, zeigt CURAD auf deren Adresse. Die Adresse, die um eine höher liegt als die Adresse mit dem Pseudo-

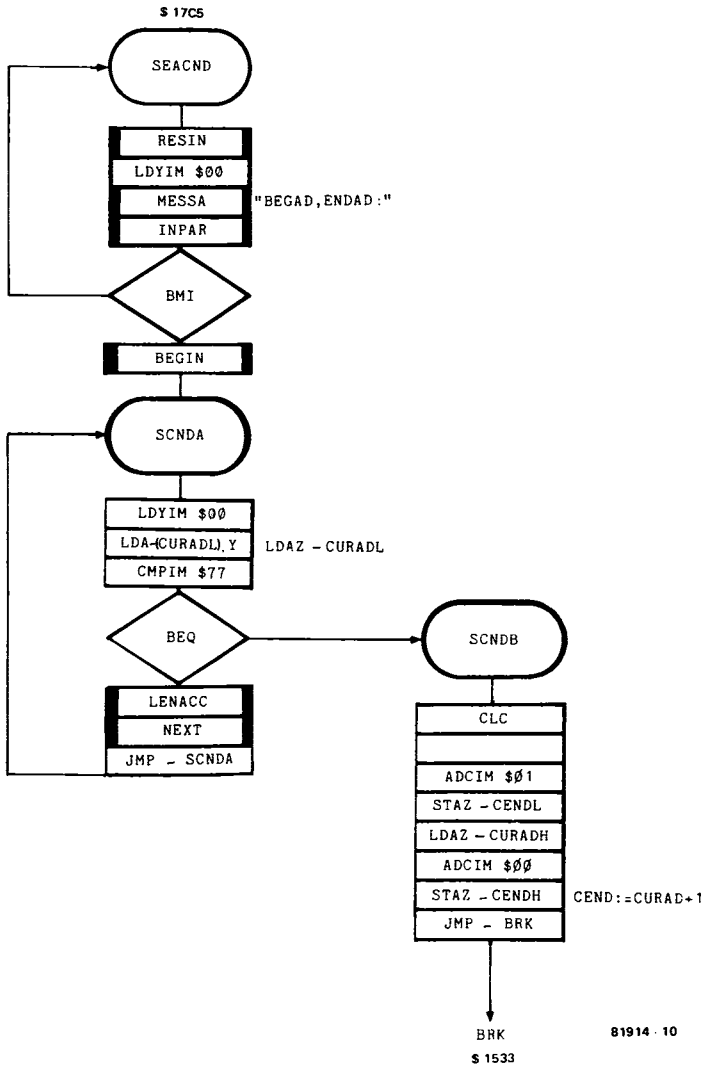


Bild 10. Vorroutine SEACND wird nach Eintritt in PME über den warmen CEND-Eingang durchlaufen. Dabei wird der ursprüngliche Wert des variablen Endadreßpointers CEND wiederhergestellt, sofern nicht das EOF-Zeichen 77 fehlt.

Opcode 77, wird nun in den variablen Endadreßpointer CEND gesetzt. Nach dem Sprung zum Label BRK in Bild 4a folgt die Festlegung des BRK-Sprungvektors, und anschließend erscheint die Rückmeldung "PM EDITOR". Damit sind wir wieder beim zentralen Label WARM der PME-

Hauptroutine angekommen. Ein vom Band gelesenes, bereits assembliertes oder noch nicht assembliertes Programm hat seine ursprünglichen BEGAD- und ENDAD-Werte zurückgehalten. Der Wert von ENDAD muß aber nicht unbedingt gleich dem ursprünglichen Wert sein: Man kann nach Eintritt in PME über den warmen CEND-Starteingang für ENDAD eine höhere Adresse eingeben, wenn man Instruktionen ein- oder anfügen will. Natürlich ist es nicht sinnvoll, für ENDAD den höchsten überhaupt möglichen Wert zu wählen, wenn das editierte Programm nur relativ wenige Instruktionen umfaßt.
Der Endwert von Instruktionspointer CURAD ist die Adresse des Speicherplatzes, in dem das EOF-Zeichen 77 steht, also nicht die in CEND stehende Adresse! Unter den Subroutinen-Instruktionen, die in Bild 10 mit dem Label SCNDB überschrieben sind, befindet sich nämlich keine Instruktion, die den Inhalt von CURAD verändert.
Dem Flußdiagramm in Bild 10 läßt sich übrigens auch entnehmen, was beim Fehlen des EOF-Zeichens 77 geschieht: Die Schleife um das Label SCNDA wird unendlich oft durchlaufen.
Wer die Beschreibungen der Input- und der I-Tastenroutine genau studiert hat, wird noch eine andere Verwendungsmöglichkeit für den warmen CEND-Starteingang von PME entdecken. Wenn nämlich eine ein- oder anzufügende Instruktion keine Aufnahme mehr in den Speicher findet (Rückmeldung "FULL"), ist zwar der Inhalt von CEND erhöht worden, das EOF-Zeichen 77 bleibt jedoch an seinem Platz (siehe Punkt 10, letzter Teil, Subroutine CHECK). Solange sich ENDAD weiter zu höheren Adressen hinausschieben läßt, weil die Speicherkapazität noch nicht erschöpft ist, kann man über den warmen CEND-Starteingang von PME einen höheren Wert für ENDAD (bei gleichem Wert für BEGAD!) eingeben. Danach paßt sich nämlich CEND automatisch an die Adresse des Speicherplatzes an, in dem der Pseudo-Opcode 77 steht.

16

1152 Byte TM-Software

Das Kassetten-Management

"Daten vom, zum und am laufenden Band", so lautete das Motto des 11. Kapitels in Buch 3. Es ging dort nicht um das Fließband eines Industriebetriebs, sondern um das Magnetband des Kassettenrekorders, das dem Junior-Computer als residenter Datenspeicher dient. Während in Kapitel 11 das Verfahren des Datentransports zum und vom Band im Vordergrund stand, gehen wir in diesem Kapitel der Frage nach, wie die Software die verschiedenen Aufgaben der Verpackung und des Transports von Daten bewältigt.

Was verbirgt sich hinter dem Namen TAPE MONITOR, abgekürzt TM? Um dies zu ergründen, beschäftigen wir uns zuerst ausführlich mit "Super-Subroutine" DUMP. Diese Subroutine wird immer dann aufgerufen, wenn Datensendungen auf Band zu speichern sind: nicht nur TM, sondern auch das Systemprogramm PM macht von ihr Gebrauch. Ferner läßt sich Subroutine DUMP vorteilhaft auch in andere, nicht systemeigene Software einbauen.

Das Gegenstück zu DUMP ist RDTAPE, eine weitere "Super-Subroutine". RDTAPE ist zuständig für den Datentransport in umgekehrter Richtung; sie liest Daten vom Band in den Junior-Computer ein. Da Subroutine RDTAPE ebenso wichtig und ebenso vielseitig verwendbar wie DUMP ist, nimmt auch RDTAPE einen verhältnismäßig breiten Raum in diesem Kapitel ein.

Systemprogramm TM stellt sozusagen eine Parallele zum Systemprogramm PM dar. Der Unterschied zwischen beiden Systemprogrammen liegt im wesentlichen darin, daß PM die Erweite-

rung der Peripherie (Bildschirm oder Drucker, ASCII-Tastatur) voraussetzt, während sich TM mit der Standard-Ausstattung (Siebensegment-Displays und Hex-Tastatur auf der Basiskarte) begnügt. Die Tastenfunktionen von PM und TM sind dagegen fast identisch.
Wie schon in den vorangegangenen Kapiteln dieses Buchs werden auch nachfolgend die einzelnen Software-Komponenten von TM Punkt für Punkt besprochen. Im übertragenen Sinn startet nun der letzte Teil einer "Datensendung", die in Buch 1 mit den "Synchronisationszeichen" begann und am Schluß dieses Kapitels mit dem "EOT-Zeichen" endet.

Bevor wir dieses Kapitel beginnen, hier ein kurzer Überblick: Unter Punkt 1 bis 9 beschäftigt uns Subroutine DUMP mit ihren Unterroutinen (man könnte sie auch "Sub-Subroutinen" nennen). Danach folgt in Punkt 10 bis 19 die ausführliche Besprechung von Subroutine RDTAPE, ebenfalls mit den zugehörigen Unterroutinen. Erst danach wenden wir uns der TM-Hauptroutine zu (Punkt 20. . . 28). Am Schluß des Kapitels steht noch eine kurze Beschreibung der PLL-Testsoftware. Wir beginnen nun mit DUMP.

1 Das detaillierte Flußdiagramm der **Subroutine DUMP** ist in den Bildern 1a (erste Hälfte) und 1b (zweite Hälfte) gezeichnet. Der erste Teil zwischen den Labeln DUMP und DUMPT (Bild 1a) besteht aus Instruktionen, die in die Speicherplätze HIGHER, LOWER, FIRST und SECOND bestimmte Werte laden. Die Inhalte der genannten Speicherplätze stehen mit der Elementarzeit der Datenübertragung, der Bitzeit und der relativen Dauer des 2400- und 3600-Hz-Signals in engem Zusammenhang. Näheres werden wir in Punkt 3 und Punkt 4 besprechen.
Erst nach dem Label DUMPT in Bild 1a werden die I/O-Leitungen so programmiert, wie es die Datenübertragung zum Band erfordert. Eigentlich sollte man erwarten, daß dies zu allererst geschieht. Das Abweichen vom gewohnten Schema hat natürlich einen Grund: Subroutine DUMP wird nicht nur von den Systemprogrammen TM und PM aufgerufen, sondern kann, wie schon in der Einleitung erwähnt, auch für Anwenderprogramme genutzt werden. Wenn ein solches Programm mit einer vom Junior-Standard abweichenden Datenübertragungsgeschwindigkeit arbeitet, sind nur die acht Lade- und Speicher-Instruktionen mit entsprechend angepaßten Werten in das Anwender-Programm aufzunehmen. Sprungziel ist dann nicht das Label DUMP am Anfang der Routine, sondern das Zwischenlabel DUMPT. Unter Punkt 9 in diesem Kapitel ist beschrieben, wie man die Schreib- und Lesegeschwindigkeit des Junior-Computers an die Gegebenheiten des KIM-Systems anpaßt.
Die ersten neun auf das Label DUMPT folgenden Instruktionen (Bild 1a)

programmieren die I/O-Ports in der für das Schreiben auf Band notwendigen Weise. Sämtliche acht Leitungen von Port B werden als Ausgang geschaltet, und in das Register PBD wird der Wert $47 geladen. Für Port B wirkt sich dies im einzelnen wie folgt aus:

- Die als Kassetten-Dateneingang und -ausgang benutzte Leitung PB7 ist als Ausgang geschaltet; an diesem Ausgang liegt (zunächst) logisch Ø.
- Portleitung PB6 ist als Relais-Steuerausgang geschaltet; er ist logisch 1. Aus der Schaltung des Kassetteninterface in Buch 3, Seite 13 geht hervor, daß in diesem Fall kein Strom durch die Spule des INPUT-Relais fließt und die grüne INPUT-LED D4 nicht aufleuchtet.
- Portleitung PB5 ist ebenfalls als Relais-Steuerausgang geschaltet. Das Signal an diesem Ausgang ist logisch Ø, so daß Transistor T3 (siehe Schaltung in Buch 3, Seite 13) leitet. Durch OUTPUT-Relais Re2 fließt Strom; gleichzeitig leuchtet die rote OUTPUT-LED D5 auf.
- Displaymultiplexer IC7 (siehe Buch 2, Seite 130, Bild 9) schaltet die sechs Siebensegment-Displays ab, da an seinen Eingängen DCBA das Signal PB4PB3PB2PB1 = ØØ11 liegt. Signaländerungen an Port A, der im Normalbetrieb die Display-Segmente steuert, bleiben daher wirkungslos. Das Drücken einer Taste auf der Hex-Tastatur hat ebenfalls keine Wirkung mehr. Beim Schreiben von Daten auf Band sind somit die Siebensegment-Displays abgeschaltet; die Tastatur ist gesperrt. Nur die rote OUTPUT-LED leuchtet auf.
- Portleitung PBØ ist als serieller Datenausgang geschaltet, obwohl während des Durchlaufs durch Subroutine DUMP über diese Leitung keine Daten übertragen werden, PBØ liegt auf logisch 1, dem Ruhesignal während der Übertragungspausen. Die vom Junior-Computer zum Kassettenrekorder gesendeten Signale laufen über Port PB7.

Nun zu Port A. Die normalerweise für die Segmentsteuerung benutzten Portleitungen PAØ . . . PA6 sind als Ausgang geschaltet; Portleitung PA7 arbeitet als serieller Dateneingang. Der Wert $ØØ, der in PAD steht, würde alle Display-Segmente aufleuchten lassen, wenn nicht das Signal auf den Leitungen von Port B dies verhinderte. Solange an Port PA7 keine Signale von einem peripheren Gerät eintreffen, liegt an diesem Port das Ruhesignal logisch 1. Im Zusammenhang mit Subroutine DUMP ist dies jedoch nebensächlich, da während ihres Ablaufs keine Daten von der Peripherie zum Junior-Computer gesendet werden.

Wenden wir uns nun wieder dem Flußdiagramm in Bild 1a zu. Hier sei zuerst angemerkt, daß der Inhalt von Speicherplatz GANG stets mit dem Inhalt von I/O-Register PBD übereinstimmt. Die Funktion dieses Speicherplatzes wird im nächsten Abschnitt (Punkt 2) erklärt. Ferner erhalten die Speicherplätze CHKH und CHKL, die bei der Kontrolle auf fehlerfreie Datenübertragung mitwirken, den Anfangswert $ØØ. Es folgt die eigentliche Vorbereitung für das Schreiben von Daten auf Band: Die Inhalte der Speicherplätze SAL und SAH, die die erste Adresse des zu sendenden Datenblocks enthalten, werden in POINTL bzw. POINTH kopiert; POINTL und POINTH bilden zusammen den Adreßpointer POINT. Die letzten Instruktionen vor dem Label SYNCS in Bild 1a setzen schließlich den Wert $FF in den als Zähler für die Synchronisationszeichen dienenden Speicherplatz SYNCNT.

Die Besprechung von Subroutine DUMP wird unter Punkt 7 fortgesetzt.

1a

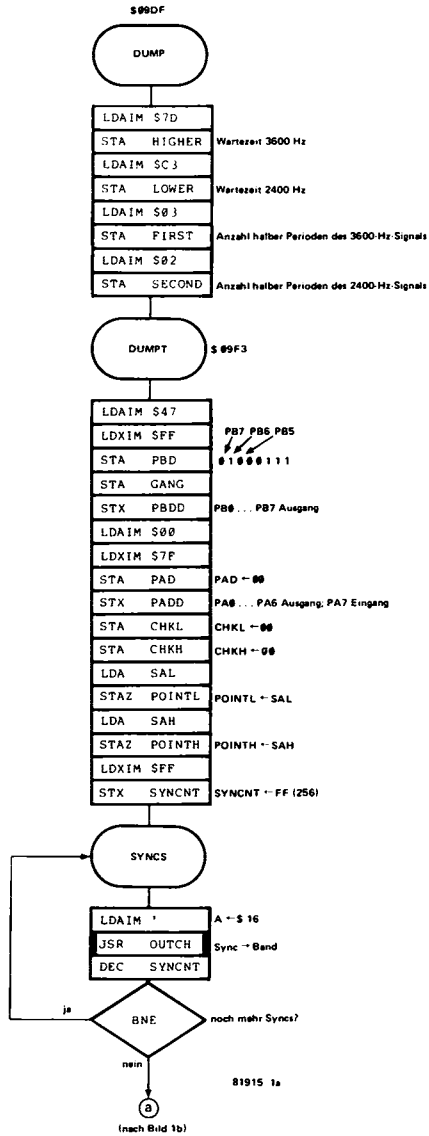


Bild 1a. Erster Teil von TM-"Super-Subroutine" DUMP/DUMPT. Hier werden hauptsächlich Vorbereitungen für das Schreiben einer Datensendung auf Band getroffen.

1b

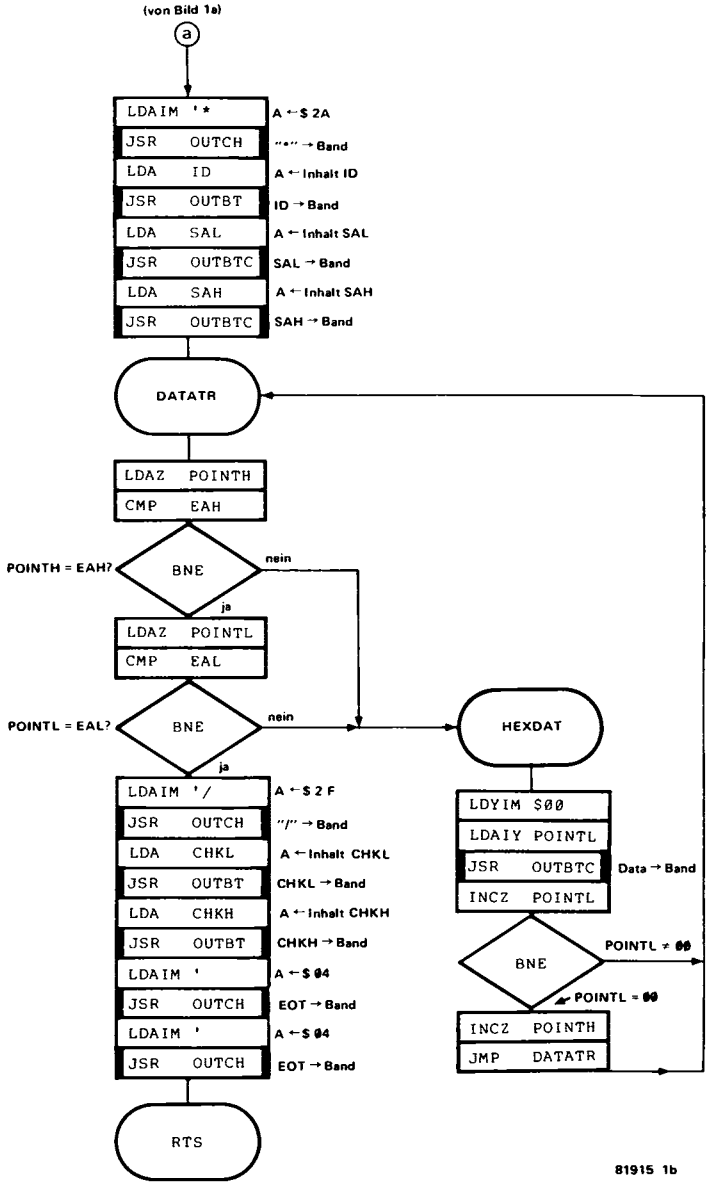


Bild 1b. Zweiter Teil der TM-Routine DUMP/DUMPT. Für das Schreiben des Datenblocks ist die Schleife um das Label HEXDAT zuständig. Die übrigen Instruktionen setzen den sogenannten Rahmen auf Band.

2 Inzwischen wollen wir verschiedene von Subroutine DUMP aufzurufende Unterroutinen betrachten, die die auf Band zu setzenden Daten entsprechend den Erfordernissen "verpacken". In Buch 3 wurde das Verfahren des Datentransports und der Datenkonservierung, das beim Junior-Computer angewendet wird, ausführlich besprochen. Deshalb fassen wir an dieser Stelle nur noch einmal kurz zusammen:

- Portleitung PB7 ist der "heiße Draht" für den Informationsfluß vom Junior-Computer zum Kassettenrekorder.
- Jedes einzelne Datenbit erhält vor Antritt der Reise zum Band die Gestalt eines tonfrequenten Signals, das mit einem höherfrequenten Teilsignal (3600 Hz) beginnt und mit einem Teilsignal niedrigerer Frequenz (2400 Hz) endet.
- Um tonfrequente Signale zu erzeugen, muß der logische Zustand von Port PB7 zu bestimmten Zeitpunkten invertiert werden (1 wird Ø und Ø wird 1); diese Zeiten sind von der momentanen Frequenz des tonfrequenten Signals abhängig.
- Ist ein zu sendendes Datenbit **logisch 1**, so wird dieses in ein tonfrequentes Signal "übersetzt", das aus **drei halben Perioden eines 3600-Hz-Rechtecksignals gefolgt von vier halben Perioden eines 2400-Hz-Rechtecksignals** besteht.
- Einem Datenbit, das **logisch Ø** ist, entspricht ein tonfrequentes Signal mit **sechs halben Perioden des 3600-Hz-Rechtecksignals gefolgt von zwei halben Perioden des 2400-Hz-Rechtecksignals.**

An der Umwandlung von Datenbits in Tonsignale sind die beiden folgenden Subroutinen maßgeblich beteiligt:

a. **Subroutine HIGH** in Bild 2a; sie liefert ein aus drei halben 3600-Hz-Perioden bestehendes Signal,
b. **Subroutine LOW** in Bild 2b; das von ihr erzeugte Signal besteht aus zwei halben 2400-Hz-Perioden.

Um ein Bit zum Kassettenrekorder zu senden, das logisch 1 ist, muß daher einmal die Subroutine HIGH und unmittelbar danach zweimal die Subroutine LOW aufgerufen werden, Bei einer logischen Ø verhält es sich umgekehrt: Subroutine HIGH wird zweimal aufgerufen, direkt gefolgt von einem Aufruf der Subroutine LOW. Ferner ist die Feststellung wichtig, daß unabhängig vom Wert des zu sendenden Bit stets zuerst Subroutine HIGH und dann Subroutine LOW an der Reihe ist.

Betrachten wir zuerst das Flußdiagramm von Subroutine HIGH in Bild 2a. Die Subroutine besteht im wesentlichen aus einer Verzögerungsschleife, wobei die Gesamtverzögerung drei halbe Perioden eines symmetrischen 3600-Hz-Rechtecksignals beträgt. Jedesmal nach Ablauf einer halben Periode muß das logische Signal von Portleitung PB7 invertiert werden. Die Anzahl der Verzögerungen steht in Speicherplatz FIRST; sein Inhalt wird am Anfang von Subroutine HIGH in das X-Register kopiert.

Das Signal, dessen Dauer einer einzelnen halben 3600-Hz-Periode (HIGH) oder 2400-Hz-Periode (LOW, siehe Punkt 3) entspricht, wird von dem im PIA-Baustein integrierten Intervall-Timer geliefert. Eine Anleitung zur Programmierung des Intervall-Timers haben wir bereits in Buch 2, Kapitel 6 gegeben. Hier wird der Timer durch Laden des Werts $7D in Speicherplatz CNTA gestartet, so daß der Teilfaktor gleich 1 ist und beim Time Out kein IRQ auftritt. Beim Erreichen des Time Out wird die aus den Instruktionen

BIT-RDFLAG und BPL (Bild 2a) bestehende Warteschleife durchbrochen und der Timer neu gestartet. Die Invertierung des logischen Signals auf Portleitung PB7 folgt kurze Zeit später. Wie schon erwähnt, ist der Inhalt von Speicherplatz GANG stets mit dem Inhalt des I/O-Registers PBD identisch. Die Instruktion EOR #8Ø invertiert Bit 7 des Akku-Inhalts, die übrigen Bit bleiben dabei unverändert. Wem die Funktion des Befehls EOR nicht mehr geläufig ist, der schlage Buch 1, Seite 66 auf und betrachte dort die dritte Zeichnung in Bild 2.

An dieser Stelle muß noch eine andere Frage beantwortet werden: Weshalb wurde nicht für Subroutine HIGH (Bild 2a) die Instruktionsfolge
LDA-PBD
EOR #8Ø
STA-PBD
anstelle der Instruktionsfolge
LDA-GANG
EOR #8Ø
STA-PBD
STA-GANG gewählt?
Die Antwort ist nicht schwierig: Weil beim Lesen einer als Ausgang geschalteten Portleitung stets eine logische 1 das Ergebnis der Leseoperation ist! Ein tonfrequentes Signal läßt sich also mit den obenstehenden "Ersatz"-Instruktionen nicht erzeugen. Durch Kopieren des Inhalts von Register PBD in Speicherplatz GANG unmittelbar nach jeder Änderung des Registerinhalts ist dieses Problem gelöst.
Ein weiteres Detail von Subroutine HIGH in Bild 2a, das einer Erklärung bedarf, ist der im X-Register stehende Wert. Zu Beginn von HIGH wird in dieses Register der Wert $Ø3 geladen, so daß der Intervall-Timer während des Ablaufs von HIGH dreimal startet. Das logische Signal auf Portleitung PB7 wird folglich ebenfalls dreimal invertiert. Um drei halbe Perioden eines 3600-Hz-Rechtecksignals zu erzeugen, muß aber das Signal auf Portleitung PB7 viermal invertiert werden. Diese Erscheinung ist auch als "Lattenzauneffekt" bekannt: Für einen Lattenzaun von 1 m Länge und einem Lattenabstand von 10 cm benötigt man nicht 10, sondern 11 Latten! Subroutine HIGH trägt dem wie folgt Rechnung: Während des letzten Durchlaufs durch die Instruktionen nach dem Zwischenlabel HIG und vor der letzten Dekrementierung des X-Registers (DEX) ist dessen Inhalt gleich Ø1. Nach Starten des Intervall-Timers, Invertieren von PB7 und Ausführen des Befehls DEX wird Subroutine HIGH verlassen (RTS). Somit stimmt die Anzahl der Ladeoperationen, die während des Ablaufs von Subroutine HIGH auf das Timer-Register CNTA abgewendet werden, mit der Anzahl der Halbperioden überein. Die nach Verlassen von HIGH bis zum letzten Time Out verstreichende Zeit ist die letzte halbe Periode des von HIGH erzeugten Signals. Der nächste Timer-Start, der bereits innerhalb des nächsten Ablaufs von HIGH (nach einem zweiten Aufruf von HIGH) oder des Ablaufs von LOW liegt, findet kurz vor der nächstfolgenden Invertierung von PB7 statt. Hierdurch wird das vorangegangene Teilsignal (3600 Hz oder 2400 Hz) vervollständigt; gleichzeitig beginnt damit das nächste Teilsignal. Für den folgenden Ablauf von Subroutine HIGH in Bild 2 bedeutet dies, daß die Verzweigung BPL erst dann nicht mehr auf "Sprung nach HIG" steht, wenn der zur letzten vorangegangenen Halb-

2a

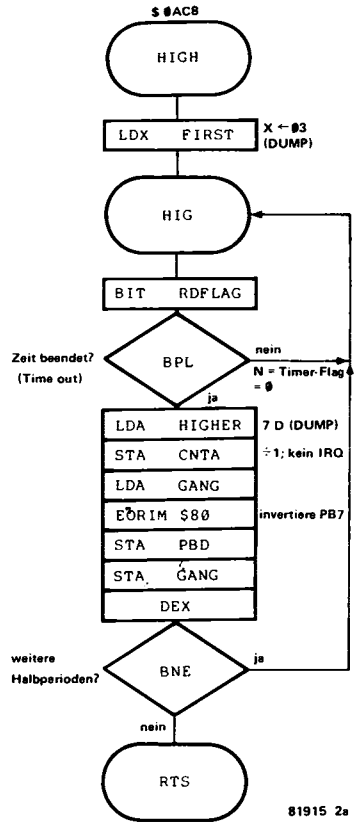


Bild 2a. Subroutine HIGH erzeugt das 3600-Hz-Teilsignal des zum Kassettenrekorder geschickten NF-Signals. Beim Bitwert Ø wird sie zweimal nacheinander durchlaufen.

periode gehörende Time Out aufgetreten ist.
Rechnerisch ergibt sich für die Dauer der Verzögerung folgendes: Beim Start wird der Intervall-Timer mit dem Wert $7D (= 125_{10}) geladen, so daß bis zum Time Out $7 \cdot 16\,\mu s + 13\,\mu s + 1\,\mu s = 126\,\mu s$ vergehen. Die Zeitdifferenz zwischen dem Timer-Start und dem Invertieren von PB7 soll als konstant angenommen werden; sie wird zum Ausführen der Instruktionen LDA-GANG, EOR #8Ø und STA-PBD benötigt und beträgt $10\,\mu s$. Das Zeitintervall zwischen zwei aufeinanderfolgenden Invertierungen von PB7 ist somit gleich dem Zeitintervall zwischen zwei aufeinanderfolgenden Starts des Intervall-Timers. Welche Dauer hat dieses Zeitintervall? Es setzt sich zusammen aus den $126\,\mu s$ des Timers, $8\,\mu s$ für die Ausführung der Instruktionen LDA-HIGHER und STA-CNTA sowie höchstwahrscheinlich einigen weiteren Mikrosekunden, weil das Verlassen der Verzögerungsschleife BIT-RDFLAG/BPL nicht mit dem Time Out des Timers zusam-

2b

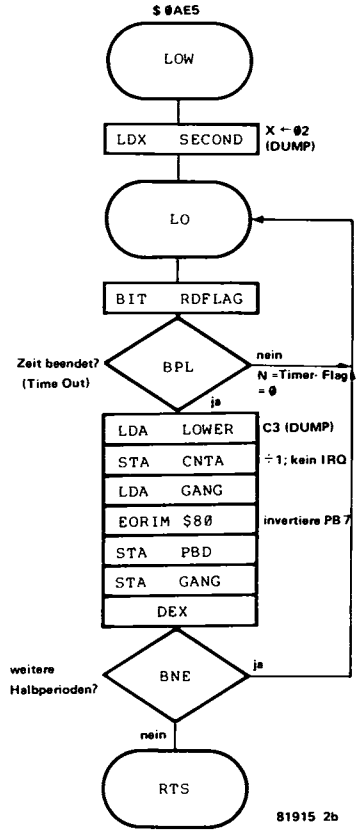


Bild 2b. Subroutine LOW erzeugt das 2400-Hz-Signal, das stets den zweiten Teil eines auf Band gesetzten Bit bildet. LOW wird beim Bitwert 1 zweimal nacheinander durchlaufen.

menfallen muß. Maximal können zur Gesamtdauer 7 µs hinzukommen; diese 7 µs entsprechen einem vollständigen Durchlauf durch die Verzögerungsschleife.
Die Zeit zwischen zwei aufeinanderfolgenden Invertierungen des logischen Signals auf Portleitung PB7 liegt somit zwischen 134 µs und 141 µs. Die Periodendauer eines 3600-Hz-Signals beträgt 277,8 µs; eine halbe Periode dauert folglich 138,9 µs. Die Differenz zwischen Theorie und Software-Praxis ist jedoch ohne jede Bedeutung, da die Lese-Software bei weitem höhere Toleranzen zuläßt.

3 Über **Subroutine LOW** in Bild 2b braucht nun nicht mehr viel gesagt zu werden, denn sie ist mit der soeben besprochenen Subroutine HIGH weitgehendst identisch. Im Unterschied zu HIGH muß LOW zwei halbe Perioden eines 2400-Hz-Rechtecksignals liefern; in das X-Register

wird deshalb der in Speicherplatz SECOND stehende Wert $Ø2 gesetzt. Die Verzögerungsschleife BIT-RDFLAG/BPL mit X = $Ø2 wird dann durchbrochen, wenn die letzte halbe Periode des vorangegangenen Teilsignals (3600 Hz, Subroutine HIGH oder 2400 Hz, Subroutine LOW) beendet ist. Der nächste Start des Intervall-Timers folgt 8 . . . 15 µs später, weil bis zum Feststellen des Time Out maximal 7 µs vergehen können. Nach weiteren 10 µs wird PB7 invertiert, der Inhalt des X-Registers wird dekrementiert (DEX), und das beschriebene Spiel wiederholt sich noch einmal. Dem Time-Out, der auf den letzten Start des Timers folgt, kommt der Rücksprung RTS zuvor. Dieser Time Out wird innerhalb der Verzögerungsschleife BIT-RDFLAG/BPL festgestellt, die zum unmittelbar anschließenden oder nächstfolgenden Aufruf von HIGH oder LOW gehört.
Und so verhält es sich bei Subroutine LOW mit den Zeiten: In den Intervall-Timer wird der in Speicherplatz LOWER stehende Wert $C3 (= 195_{10}) geladen. Bis zum Time Out vergehen deshalb 12 · 16 µs + 3 µs + 1 µs = 196 µs. Hinzu kommen 8 µs, die für die Ausführung der Instruktionen LDA-LOWER und STA-CNTA benötigt werden (vergleiche Subroutine HIGH). Bis zum Feststellen des Time Out können maximal weitere 7 µs vergehen, so daß die Gesamtdauer zwischen 204 µs und 211 µs liegt. Da die exakte Dauer einer halben 2400-Hz-Periode 208,3 µs beträgt, sind die gestellten Forderungen an die Genauigkeit auch bei Subroutine LOW erfüllt.

4 **Zwei allgemeine Anmerkungen zu HIGH und LOW:** Zuerst sei darauf hingewiesen, daß bei den Subroutinen HIGH und LOW das "Umpolen" von Portleitung PB7 nach Ablauf von jeweils einer halben Periode die vorrangige Aufgabe ist. Die Signalspannung hat dabei nebensächliche Bedeutung, denn die Dateninformationen "Ø" und "1" stecken in der Frequenz der beiden einander abwechselnden Tonsignale sowie in deren relativer Dauer, nicht aber in der Signalamplitude oder der Signalform.
Die Zweite Anmerkung: Einem Aufruf von Subroutine HIGH folgt stets entweder ein weiterer Aufruf von HIGH oder ein Aufruf von Subroutine LOW. Das Senden eines Datenbit zum Band beginnt nämlich mit mindestens einem Durchlauf von HIGH und endet mit mindestens einem Durchlauf von LOW. Der Time Out nach der letzten Halbperiode des 3600-Hz-Signals (HIGH) fällt deshalb immer in die Zeit der BIT-RDFLAG/BPL-Verzögerungsschleife, die zum nächsten Durchlauf von Subroutine HIGH bzw. LOW gehört.

Bei Subroutine LOW verhält sich dies anders. Wenn der einzige bzw. letzte Durchlauf (abhängig vom Bit-Wert) von LOW beendet ist, muß noch die letzte halbe Periode des 2400-Hz-Signals abgeschlossen werden. Das geschieht innerhalb der BIT-RDFLAG/BPL-Verzögerungsschleife des nächstfolgenden Durchlaufs von Subroutine HIGH. In dieser Verzögerungsschleife verbleibt der Prozessor jedoch nicht so lange, wie dies beim Übergang von HIGH nach LOW (siehe oben) der Fall ist. Ursache sind die zusätzlichen Instruktionen, die das Senden des nächsten Datenbit (Label ONE in Bild 3, siehe Punkt 5), Datennibble (Label NIBOUT) oder ASCII-Zeichens (Label OUTCH in Bild 3) vorbereiten. Diese Instruktionen müssen abgearbeitet sein, bevor der Time Out des Intervall-Timers erreicht ist. Zu freien Verfügung verbleiben bis dahin ungefähr 200 µs, so daß rund

3

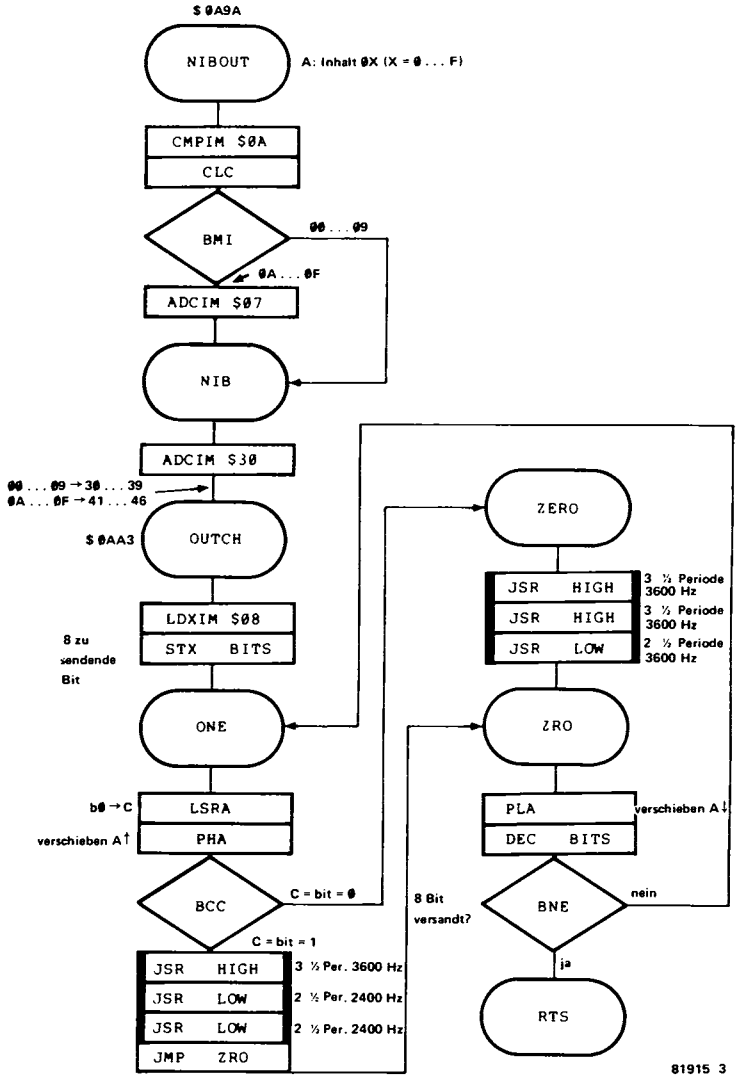


Bild 3. Die zweite Hälfte der Subroutine NIBOUT, beginnend beim Label OUTCH, sendet ein einzelnes ASCII-Zeichen zum Band. Vom Label NIBOUT an wird die Routine durchlaufen, wenn zuvor ein Daten-Nibble in den zugehörigen ASCII-Kode umgewandelt werden muß.

50 Instruktionen von je 4 µs Ausführungszeit eingeschoben werden können. Das aber ist in jedem Fall ausreichend.
Nach diesen beiden Anmerkungen wollen wir nun erforschen, wie die Software die einzelnen Bit, Byte und ASCII-Zeichen versandfertig "verpackt" und auf die Reise zur Bandkassette schickt.

5 **Subroutine NIBOUT/OUTCH** in Bild 3 sendet die acht Bit eines im Akku stehenden ASCII-Kodes nacheinander zum Kassettenrekorder. Das Senden der Bit erledigen die Instruktionen, die auf Zwischenlabel OUTCH folgen. Datennibble müssen zuvor noch in ASCII-Kodes umgewandelt werden; in diesem Fall beginnt das Abarbeiten der Routine beim Label NIBOUT. Zuerst soll der erste Teil der Routine zwischen den Labeln NIBOUT und OUTCH betrachtet werden (siehe Bild 3).
Wir gehen davon aus, daß das zu sendende Datenbyte bereits in zwei Datennibble aufgespalten ist (vgl. Punkt 6). Subroutine NIBOUT setzt bei jedem Aufruf ein Datennibble auf Band. Das Datennibble wird wie folgt in den entsprechenden ASCII-Kode umgewandelt: Unmittelbar vor dem Sprung nach NIBOUT steht im Akku der Wert $ØX, wobei X = Ø . . . F sein kann. Wie diese Zahl in den Akku gelangt, ist unter Punkt 6 nachzulesen und aus Bild 4 (Subroutine OUTBTC/OUTBT) zu entnehmen. Den Nibble-Werten Ø . . . 9 entsprechen die ASCII-Kodes 3Ø . . . 39 und den Werten A . . . F entsprechen die ASCII-Kodes 41 . . . 46. Im ersten Fall muß zum Akku-Inhalt der Wert $3Ø und im zweiten Fall der Wert $37 addiert werden. Die Instruktion CMP #ØA unterscheidet zusammen mit dem BMI die erste Gruppe von der zweiten und sorgt dafür, daß stets der richtige Wert addiert wird. Als Ergebnis steht der ASCII-Kode des Datennibbles im Akku.
Beim Label OUTCH beginnt die eigentliche Senderoutine. Zuerst wird der Wert $Ø8 in den als Bitzähler dienenden Speicherplatz BITS geladen. Danach durchläuft der Prozessor achtmal die beim Label ONE beginnende und beim nachfolgenden BCC verzweigende Schleife, wobei jedesmal ein Bit zum Band gesendet wird. Im einzelnen geschieht hierbei folgendes: Zu Beginn rücken sämtliche im Akku stehende Bits um einen Platz nach rechts (LSRA); Bit bØ wird dabei zum Carry-Bit. Dann wandert der verschobene Akku-Inhalt vorübergehend in den Stack (PHA). Die Carry-Flag, in der das Bit bØ steht, entscheidet den weiteren Verlauf der Routine. Ist die Carry-Flag 1, so werden nacheinander einmal die Subroutine HIGH und zweimal die Subroutine LOW aufgerufen. Bei C = Ø folgt dagegen ein Sprung zum Label ZERO, was zwei Aufrufe von HIGH und nur einen Aufruf von LOW bedeutet. In beiden Fällen wird nach dem Label ZRO der verschobene Akku-Inhalt vom Stack geholt (PLA), der Inhalt von Speicherplatz BITS wird dekrementiert, und die Routine springt, sofern noch mindestens ein zu sendendes Bit vorhanden ist, zum Label ONE zurück.
Hinweis: Die Subroutinen HIGH und LOW wurden unter den Punkten 2, 3 und 4 in diesem Kapitel beschrieben.

6 **Subroutine OUTBTC/OUTBT**, dargestellt in Bild 4, sorgt dafür, daß Datenbytes in Form von zwei aufeinanderfolgenden Datennibble zum Band gesendet werden. Die dem Label OUTBT folgenden Instruktionen erledigen diese Aufgabe. In bestimmten Fällen ist jedoch vorher

4

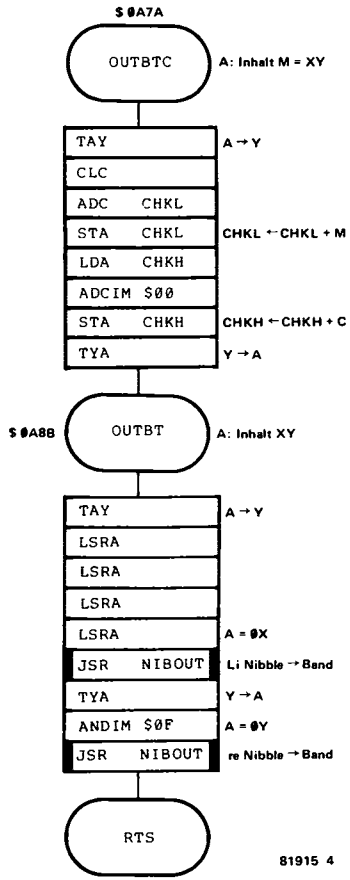


Bild 4. Der Teil der Subroutine OUTBTC, der beim Label OUTBT beginnt, setzt zwei Daten-Nibble auf Band, die zusammen ein Daten-Byte bilden. Die Routine wird vom Label OUTBTC an durchlaufen, wenn der Wert des Daten-Byte zu der in den Speicherplätzen CHKH/CHKL stehenden Kontrollsumme addiert werden muß.

noch etwas anderes notwendig: Die 8-bit-Zahl, die das Datenbyte verkörpert, muß zu der 16-bit-Zahl addiert werden, die die Inhalte der Speicherplätze CHKH und CHKL bilden. Diese Speicherplätze enthalten die Kontrollsumme aller Datenbytes; ihre Funktion wurde bereits in Buch 3, Kapitel 11 beschrieben. Auf den aktuellen Stand wird der Inhalt von CHKH/CHKL durch die Instruktionen gebracht, die in Bild 4 zwischen den Labeln OUTBTC und OUTBT stehen.
Das zu sendende Datenbyte steht vor Aufruf der Subroutine OUTBTC im Akku. Es wird am Anfang der Routine in das Y-Register kopiert (TAY) und dann zu der in den Speicherplätzen CHKH, CHKL stehenden Zahl addiert. Die angewendete Rechenoperation ist die bekannte Standard-

16-bit-Addition. Nach Wiederherstellen des ursprünglichen Akku-Inhalts (TYA) ist das Label OUTBT erreicht.
Der nun folgende Teil der Subroutine beginnt ebenfalls mit dem Kopieren des Akku-Inhalts in das Y-Register. Vier LSRA-Instruktionen schließen sich an, so daß das linke Nibble des neuen Akku-Inhalts Null und das rechte Nibble gleich dem ursprünglichen linken Nibble ist. Dieses Nibble wird von Subroutine NIBOUT (siehe Bild 3 und Punkt 5) zum Kassettenrekorder gesendet. Danach erhält der Akku seinen ursprünglichen Inhalt zurück (TYA), und das linke Nibble wird durch die Instruktion AND #ØF zu Null gemacht. Nachdem Subroutine NIBOUT auch das zweite Nibble zum Band gesendet hat, wird die Routine mit dem Rücksprung beendet.

7 An dieser Stelle setzen wir Punkt 1 fort. In der Besprechung von **Subroutine DUMP/DUMPT** waren wir bis zum Label SYNCS (siehe Bild 1a) vorgestoßen. Die auf dieses Label folgenden Instruktionen senden die jedem Datenblock vorangehenden 255 Synchronisationszeichen zum Band. Hier läuft im einzelnen folgendes ab: Laden des Akku mit dem ASCII-Kode $16 des Synchronisationszeichens, Senden dieses ASCII-Kodes zum Band (Subroutine OUTCH, siehe Punkt 5 und Bild 3) und Dekrementieren des Synchronisationszeichenzählers SYNCNT. Dies alles geschieht 255 mal. Beim 256. Durchlaufen der Schleife wird der in Bild 1a gezeichnete Teil von Subroutine DUMP verlassen. Die Fortsetzung dieser "Super-Subroutine" ist in Bild 1b zu finden.
Aus Buch 3, Kapitel 11 wissen wir, daß auf die Synchronisationszeichen das Datenblockbeginnzeichen folgen muß. Die Instruktionen oben in Bild 1b laden deshalb den Akku mit $2A, dem ASCII-Kode des Zeichens "*" (Stern). Subroutine OUTCH sendet anschließend diesen ASCII-Kode zum Band. Vervollständigt wird der Vorspann der Datensendung durch die Programmnummer ID sowie die Adreßbytes SAH und SAL der Datenblock-Startadresse. Da die Startadresse in die Kontrollsumme (Inhalt der Speicherplätze CHKH/CHKL) eingeht, folgt nach Laden von SAL und SAH jeweils ein Aufruf von Subroutine OUTBTC (siehe Bild 4 und Punkt 6).

8 Der **letzte Teil von Subroutine DUMP/DUMPT**, der beim Zwischenlabel DATATR (Bild 1b) beginnt, ist für den Transport des Datenblocks zum Band und für den Nachspann der Datensendung zuständig. Der Datenblock umfaßt sämtliche Daten von Startadresse SA bis einschließlich der letzten Adresse LA. Endadresse EA liegt bekanntlich um eine Adresse höher als LA. Während des Transport der einzelnen Bytes weist Adreßpointer POINT die jeweils aktuelle Adresse an. POINT zeigt zu Beginn der Datensendung auf Startadresse SA (siehe Bild 1a und Punkt 1).
Die drei unmittelbar auf Label DATATR folgenden Instruktionen prüfen, ob POINT bereits auf Endadresse EA zeigt. Wenn dies der Fall ist, dann wird der Nachspann abgewickelt, der jede Datensendung beendet. Zeigt POINT noch nicht auf EA, so springt die Routine zum Label HEXDAT. Der Inhalt des von POINT angewiesenen Speicherplatzes wird in den Akku geladen und nach Anpassen von CHKL und gegebenenfalls auch CHKH zum Band transportiert: Aufruf von Subroutine OUTBTC. Danach wird POINT auf den nächsten Speicherplatz gerichtet. Der Rücksprung zum

Label DATATR bedeutet, daß die Byte-Transportaktion von vorn beginnen kann.
Sobald der Datenblock vollständig auf dem Band steht, springt die Routine nicht mehr zum Label HEXDAT. Es werden nun die Instruktionen abgearbeitet, die den Nachspann auf das Band setzen. Dazu gehören der Reihe nach das Datenblock-Endzeichen "/" (ASCII-Kode $2F), der Endstand von CHKH und CHKL sowie zwei Datensende-Endzeichen EOT (ASCII-Kode $Ø4). Damit ist die Datensendung abgeschlossen. Der Rücksprung RTS aus der "Super-Subroutine" DUMP setzt auch den Schlußpunkt unter deren Besprechung, wenn nicht noch eine kleine Ergänzung notwendig wäre:

9 Subroutine DUMP bzw. DUMPT läßt sich, wie schon erwähnt, auch als "Baustein" für eigene Software nutzen. Sofern man die gleiche Übertragungsgeschwindigkeit wählt, mit der die Junior-Computer-Systemprogramme Daten zum Band senden, genügt im fremden Programm die Instruktion
JSR-DUMP.
Eine abweichende Übertragungsgeschwindigkeit macht die Änderung des Inhalts von mindestens zwei der vier Speicherplätze HIGHER, LOWER, FIRST und SECOND notwendig. Im allgemeinen wird diese Geschwindigkeit niedriger als der Junior-Standard liegen, denn mit zunehmender Übertragungsgeschwindigkeit sinkt die Zuverlässigkeit der Datenübertragung stark ab. Das Sprungziel nach Anpassung von HIGHER, LOWER, FIRST und SECOND ist dann nicht mehr DUMP, sondern DUMPT!
Für den Spezialfall, daß der Junior-Computer mit der Datenübertragungsgeschwindigkeit des Mikrocomputers KIM arbeiten soll, ist die in Bild 5 gezeichnete **Subroutine DUMKIM** notwendig. Hier werden zuerst die vier genannten Speicherplätze mit den notwendigen Werten geladen, dann folgt ein Sprung zum Label DUMPT in Bild 1a. Aus Bild 5 geht hervor, daß die Inhalte von HIGHER und LOWER unverändert geblieben sind. Der Grund dafür ist die gleiche Form des für ein einzelnes Bit auf Band geschriebenen Signals beim JUNIOR und beim KIM (siehe Buch 3, Seite 89, Bild 6). Die Inhalte von FIRST und SECOND unterscheiden sich dagegen jeweils um den Faktor 6 von den ursprünglichen Werten, weil die Übertragungsgeschwindigkeit des KIM im Vergleich zum JUNIOR nur ein Sechstel beträgt.

10 Wir kommen nun zur Besprechung der **zweiten TM-"Super-Subroutine": RDTAPE.** Diese Subroutine liest die vorher vom Programmierer spezifizierte Datensendung vom Band und setzt gleichzeitig den darin enthaltenen Datenblock in den Speicher des Junior-Computers. Das detaillierte Flußdiagramm von RDTAPE ist in den Bildern 6a (erster Teil) und 6b (zweiter Teil) gezeichnet. Zuerst beschäftigen wir uns mit den Instruktionen am Anfang der Routine, oben in Bild 6a.
Hauptaufgabe der ersten sieben Instruktionen nach dem Label RDTAPE ist die Programmierung der I/O-Ports: In Register PBD wird der Wert $32 und in Register PBDD der Wert $7E gesetzt. Mit diesen Werten sind die I/O-Ports wie folgt programmiert:

- Die als Kassetten-Dateneingang und -ausgang benutzte Portleitung PB7

5

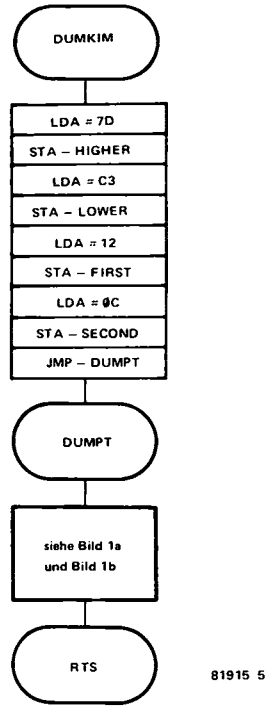


Bild 5. Subroutine DUMKIM (sie ist nicht in der TM-Routine enthalten) ist bis auf einen Unterschied identisch mit Subroutine DUMP: Die Übertragungsgeschwindigkeit entspricht der Schreib- und Lesegeschwindigkeit des Mikrocomputers KIM in seiner Normalversion. Die "Hypertape"-Software des KIM arbeitet dagegen mit der gleichen Übertragungsgeschwindigkeit wie das Systemprogramm TM des Junior-Computers.

ist als Eingang geschaltet; an diesem Eingang liegt (zunächst) logisch Ø.

- Die als serieller Datenausgang benutzte Portleitung PBØ ist, im Gegensatz zu Subroutine DUMP, hier als Eingang geschaltet; ihre ursprüngliche Funktion ist damit aufgehoben.
- Portleitung PB6 dient als Relais-Steuerausgang, an ihm liegt logisch Ø. Transistor T2 (siehe Interface-Schaltung in Buch 3, Seite 13) leitet daher. Die grüne OUTPUT-LED D5 leuchtet auf, und INPUT-Relais Re1 schaltet den INPUT-Kassettenrekorder ein.
- Portleitung PB5 dient ebenfalls als Relais-Steuerausgang. Diese Portleitung ist logisch 1, so daß Transistor T3 des Kassetten-Interface sperrt. Durch OUTPUT-LED D5 und OUTPUT-Relais Re2 fließt kein Strom.
- Displaymultiplexer IC7 (siehe Buch 2, Seite 130, Bild 9) ist wegen PB4PB3PB2PB1 = 1ØØ1 so geschaltet, daß Siebensegment-Display Di6 (das Display auf der Basiskarte ganz rechts) aufleuchten kann. In Buch 3, Kapitel 11 war zu lesen, daß auch Display Di5 beim Lesen vom Band aufleuchtet. Näheres darüber ist bei der Besprechung von Subroutine CHARVU/VU unter Punkt 15 zu erfahren.

6a

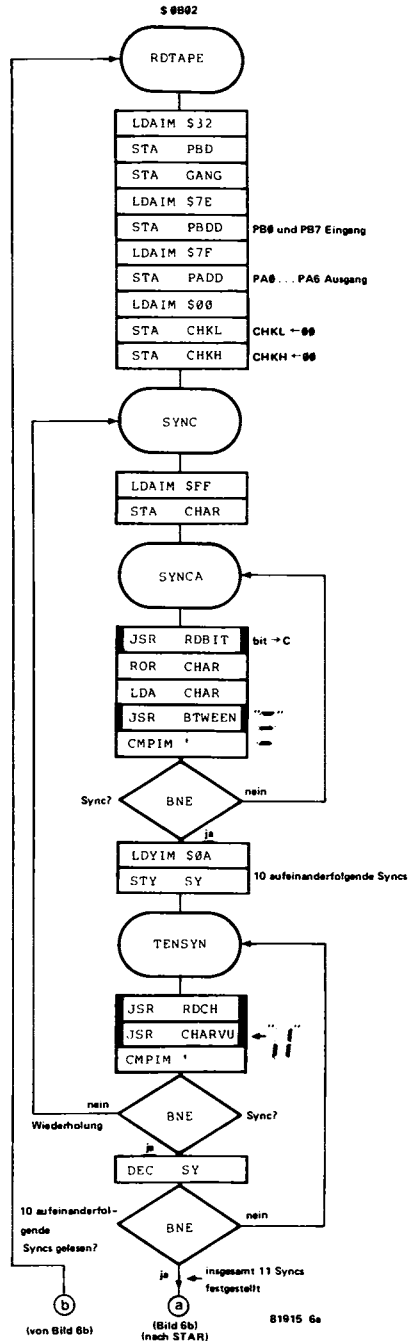


Bild 6a. Erster Teil der "Super-Subroutine" RDTAPE, dem Gegenstück zu DUMP/ DUMPT. Hier werden die Vorbereitungen für das Lesen eines Datenblocks vom Band getroffen.

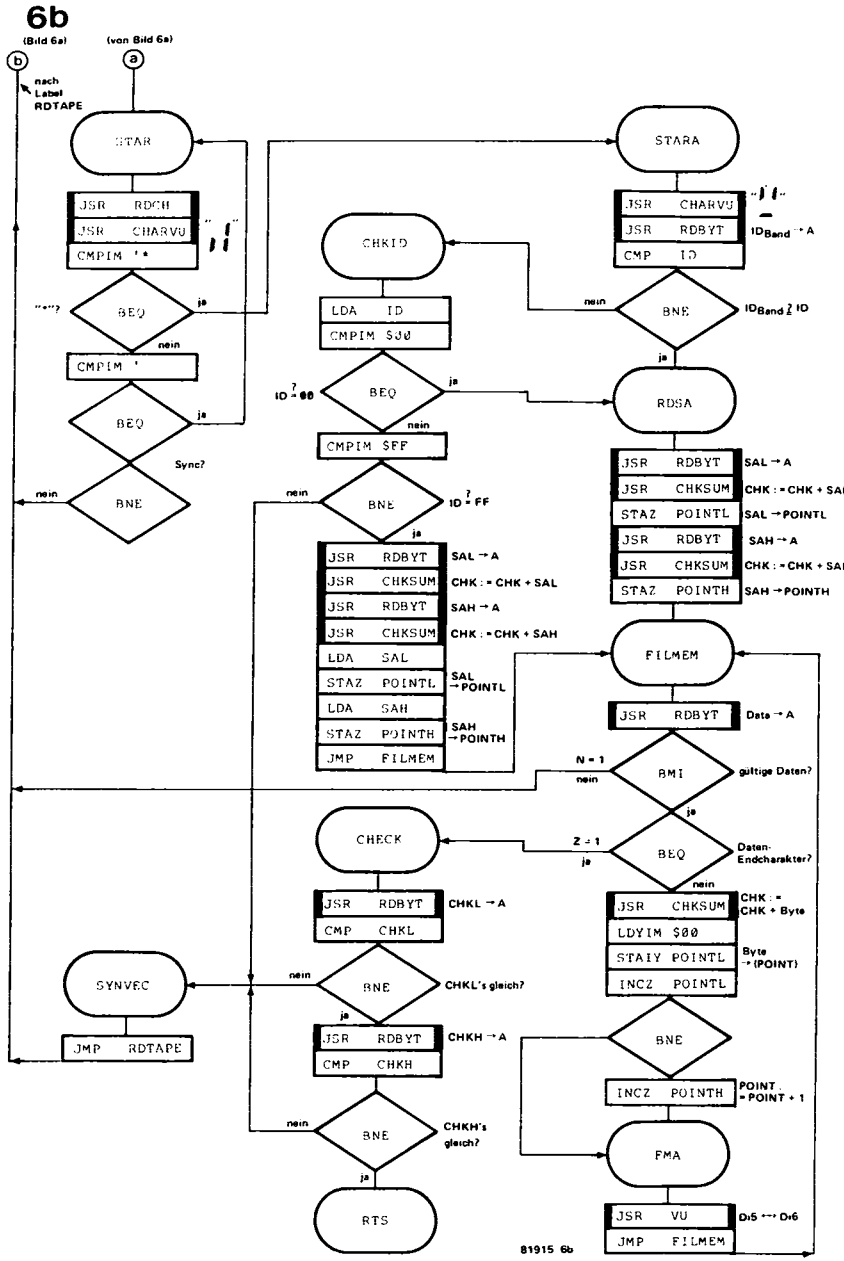


Bild 6b. Zweiter Teil von RDTAPE. Daß das Lesen der Daten vom Band nicht ganz unkompliziert ist, spiegelt sich im Flußdiagramm von RDTAPE deutlich wider.

Der in Register PADD des Ports A geladene Wert $7F hat zur Folge, daß die Portleitungen PAØ . . . PA6, die die Display-Segmente steuern, als Ausgänge geschaltet sind. Ferner wird hierdurch die als serieller Dateneingang für das Terminal dienende Portleitung PA7 bestimmungsgemäß als Eingang programmiert. Für Subroutine RDTAPE ist letzteres jedoch bedeutungslos, da RDTAPE eventuell auf dieser Leitung ankommende Daten nicht verarbeitet. Die vom Kassettenrekorder kommenden Datensignale gelangen nicht über Portleitung PA7, sondern über Portleitung PB7 in den Junior-Computer.
Unter den Instruktionen zu Beginn von Subroutine RDTAPE suchen wir vergeblich nach einer Instruktion STA-PAD. Eine solche Instruktion fehlt hier, weil in Register PAD erst innerhalb von Subroutine BTWEEN (siehe Punkt 16) oder innerhalb von Routine CHARVU/VU (Punkt 15) ein Bitmuster gesetzt wird. Dieses Bitmuster kann sich im Verlauf von RDTAPE ändern.
Bevor wir die Besprechung von Kassetten-Leseroutine RDTAPE fortsetzen, wollen wir uns zunächst den von ihr aufgerufenen Subroutinen zuwenden. In Punkt 17 kehren wir zu RDTAPE zurück.

11 Eine elementare Subroutine von RDTAPE ist **Subroutine RDBIT** in Bild 7. Wie bereits der Name verrät, liest RDBIT die einzelnen Bits einer Datensendung vom Band; hierbei wird das Ergebnis jeder Leseaktion (logisch Ø oder 1) von der Carry-Flag angezeigt. Wegen der relativ schwierigen Aufgabe, die RDBIT zu verrichten hat, fällt auch die Besprechung von RDBIT etwas länger als gewohnt aus.
Bevor wir die Einzelheiten von Subroutine RDBIT betrachten, wollen wir zuerst einen Blick auf die sechs in Bild 8 gezeichneten Impulsdiagramme werfen. Den beiden Signalen, die dort mit den Buchstaben a und d bezeichnet sind, waren wir bereits in Buch 3, Kapitel 10 und 11 begegnet. Es sind die PLL-Ausgangssignale für logisch Ø und logisch 1, wobei der Idealfall angenommen wurde, daß kein PLL-Jitter auftritt. Tatsächlich aber sehen diese Signale ungefähr so aus, wie unter b (logisch Ø) und e (logisch 1) in Bild 8 skizziert ist. Den PLL-Jitter deuten dort die Schwingungen an, die jeden Signalsprung begleiten; sie sind mit dem Prellen eines mechanischen Schalters vergleichbar (siehe Buch 2, Seite 128, Bild 8). In Wirklichkeit sind zwar die Einschwingvorgänge beim Umschalten des PLL noch wesentlich komplizierter, die in Bild 8 gegebene Darstellung reicht aber in diesem Zusammenhang aus. Wichtig ist hier hauptsächlich die Tatsache, daß sich der exakte Zeitpunkt eines Frequenzsprungs von 3600 Hz nach 2400 Hz (und umgekehrt) nicht eindeutig rekonstruieren läßt.
Das PLL-Ausgangssignal wird noch invertiert, bevor es auf die als Eingang geschaltete Portleitung PB7 gelangt. Die Signale für logisch Ø und logisch 1 an PB7 sind in Bild 8 unter c und f dargestellt.
Über einen längeren Zeitraum betrachtet besteht das an PB7 liegende Eingangssignal aus der Aneinanderreihung von Teilsignalen, wie sie die Impulsdiagramme c und f in Bild 8 zeigen. Es ist deutlich, daß unabhängig von der Bitfolge der Datensendung ein ständiger Wechsel zwischen 3600-Hz- und 2400-Hz-Perioden stattfindet. Jede 3600-Hz-Periode gehört zum Beginn eines Datenbit, jede 2400-Hz-Periode signalisiert dessen Ende. Für das Verständnis von RDBIT ist die Kenntnis dieser Tatsache besonders

7

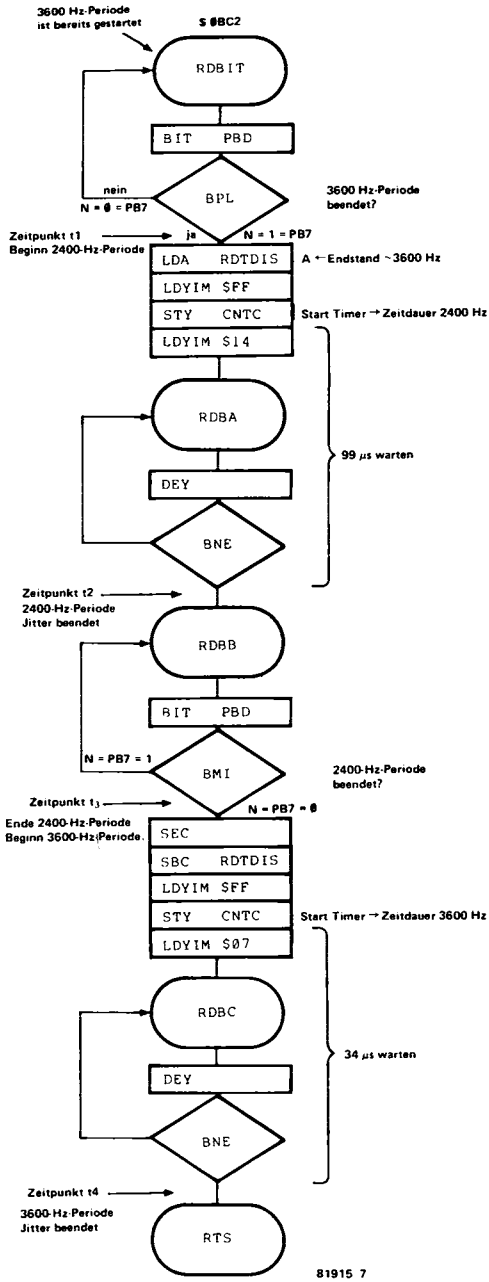


Bild 7. Subroutine RDBIT liest ein einzelnes Bit vom Band. Auskunft über den Wert des gelesenen Bit (log. 0 oder 1) gibt die Carry-Flag.

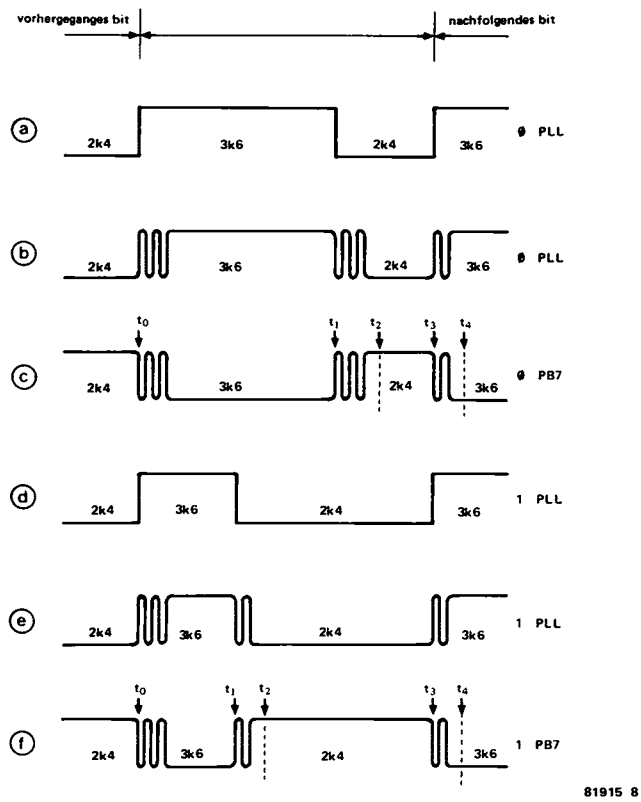


Bild 8. Impulsdiagramme der verschiedenen Signale, die die PLL-Schaltung beim Lesen von logisch 0 und 1 im Idealfall (a und d) und in der Praxis (b und e) liefert. Von der Software wird das an Port PB7 liegende, invertierte Signal verarbeitet.

wichtig.
Subroutine RDBIT, dargestellt in Bild 7, beginnt mit einer Warteschleife: die Instruktionen BIT-PBD und BPL. Der Prozessor durchläuft diese Schleife so lange, bis das Signal auf PB7 und damit auch die N-Flag logisch 1 wird. Aus den Impulsdiagrammen c und f in Bild 8 geht hervor, daß das dann geschieht, wenn eine 2400-Hz-Periode die vorangegangene 3600-Hz-Periode ablöst. Der Zeitpunkt, zu dem dieser Wechsel eingeleitet wird, ist in Bild 8 mit t_1 bezeichnet. Zurück zu Bild 7: Auf die Warteschleife folgt mit der Instruktion LDA-RDTDIS das Lesen des Timer-Datenregisters. Die "Momentaufnahme" des Timer-Stands wird damit in den Akku gesetzt.
Das Lesen des Timer-Datenregisters ist natürlich nur dann sinnvoll, wenn der Timer irgendwann vorher gestartet wurde. In der Tat ist das geschehen. Gelesen wurde das Timer-Register am Ende der 3600-Hz-Periode, also zum

Zeitpunkt t_1. Der letzte Start des Timers fand zu Beginn der 3600-Hz-Periode, zum Zeitpunkt t_0 statt. Dieser Zeitpunkt liegt aber vor dem Sprung nach Subroutine RDBIT. Die Erklärung ist einfach: Der Timer wurde während der letzten Ausführungsphase von Subroutine RDBIT gestartet, die zum **vorangegangenen** Aufruf von RDBIT gehört. Hier stoßen wir auf ein charakteristisches Merkmal dieser Bit-Leseroutine: Während jedes Durchlaufs durch RDBIT wird ein Bit eingelesen und die Carry-Flag gesetzt oder rückgesetzt. Die letzte Phase von RDBIT schließt Vorbereitungen für den nächsten Aufruf von RDBIT ein oder, anders ausgedrückt, die erste Phase von RDBIT wurde während der letzten vorangegangenen Phase vorbereitet. Möglich ist das nur dank des fortlaufenden, ununterbrochenen Wechselspiels der Tonfrequenzen auf dem Band: 3600 Hz - 2400 Hz - 3600 Hz - 2400 Hz - usw.
Wenden wir uns wieder Bild 7 zu. Nach Lesen und Speichern des 3600-Hz-Timer-Endstands in den Akku wird der Intervall-Timer neu gestartet. Die Instruktionen LDY#FF und STY-CNTC bewirken, daß ausgehend vom Anfangswert $FF der Inhalt des Timer-Datenregisters alle 64 µs um eins herabgesetzt wird; siehe Kapitel 6 in Buch 2. Nach dem Time Out tritt kein IRQ auf.
In Bild 7 sind wir nun beim Label RDBA angelangt. Der Zweck der hier beginnenden Warteschleife (DEY und BNE mit vorangegangenem LDY#14) ist es, den PLL-Jitter unschädlich zu machen. Die Verzögerungsdauer wurde so gewählt, daß der Prozessor die Ausführung der Routine erst nach Erreichen von Zeitpunkt t_2 (siehe Bild 8, Diagramme c und f) fortsetzt. Ohne diese Maßnahme würde die nächste Warteschleife, die beim Label RDBB beginnt und das Ende der 2400-Hz-Periode detektiert, zu früh durchbrochen. Die Folge eines solchen "Fehlfinish" wäre die Verfälschung der gelesenen Daten.
Sobald zum Zeitpunkt t_3 die 2400-Hz-Periode beendet ist, steht der direkt auf Instruktion BIT-PBD folgende BMI nicht mehr auf Sprung. Wir stoßen nun auf die Instruktionen SEC und SBC-RDTDIS, die einer gesonderten Erklärung bedürfen und deshalb im nächsten Abschnitt (Punkt 12) behandelt werden. Hier soll die Feststellung genügen, daß nach Ausführung der genannten Instruktionen die Carry-Flag den Wert (Ø oder 1) des gerade vom Band gelesenen Bit angibt. Danach wird der Intervall-Timer auf die gleiche Weise neu gestartet, wie dies bereits am Anfang von RDBIT geschah. Hiermit ist die schon beschriebene Vorbereitung für den nächsten Aufruf von RDBIT getroffen. Der Intervall-Timer startet also immer (annähernd) zum Zeitpunkt t_3.
Die letzten Instruktionen von RDBIT, nach Label RDBC, bewirken ebenfalls eine Verzögerung. Damit ist sichergestellt, daß die erste Warteschleife des nachfolgenden Aufrufs von RDBIT zu einem Zeitpunkt erreicht wird, zu dem kein PLL-Jitter mehr auftritt. Dies ist der Zeitpunkt t_4.
Man beachte, daß Zeitpunkt t_3 des eingelesenen Bit identisch ist mit Zeitpunkt t_0 des folgenden einzulesenden Bit; siehe Diagramme c und f in Bild 8!

12 Mit den Instruktionen SEC und SBC-RDTDIS innerhalb der Routine RDBIT (Bild 7) hat es folgende Bewandtnis: Der Befehl SEC sorgt bekanntlich dafür, daß vor einer Subtraktion die Carry-Flag

gesetzt und somit borrow = Ø ist; eventuelle "alte Schulden" sind dadurch getilgt. Die Instruktion SBC-RDTDIS subtrahiert vom Akku-Inhalt, also vom Stand des Timers am Ende der 3600-Hz-Periode, den Stand des Timers, den dieser am Ende der 2400-Hz-Periode aufweist. Den Betrag, der vom Akku-Inhalt abgezogen werden muß, liefert die in der Instruktion SBC-RDTDIS enthaltene Leseoperation.
Sowohl zu Beginn einer 3600-Hz-Periode als auch zu Beginn einer 2400-Hz-Periode startet mit dem Anfangswert $FF das Rückwärtszählen des Timers; alle 64 µs wird der Timer-Stand um eins herabgesetzt. Der Endstand des Timers ist folglich um so niedriger, je länger eine 3600-Hz- bzw. 2400-Hz-Periode dauert.
Welche Information liefert das Subtraktionsergebnis? Die Antwort ist nicht schwierig: Wenn der Akku-Inhalt größer oder gleich dem Inhalt des Timer-Datenregisters zum Ausführungszeitpunkt von Instruktion RDTDIS ist, hat dies zur Folge, daß die Carry-Flag gesetzt wird (C = 1). Das aber ist dann der Fall, wenn die 3600-Hz-Periode kürzer als die 2400-Hz-Periode war. Letzteres bedeutet aber, so geht aus Bild 8 hervor, daß das vom Band gelesene Bit eine logische 1 ist.
Analog dazu läßt sich die gleiche Überlegung auch für den Fall anstellen, daß als Ergebnis der Subtraktion die Carry-Flag rückgesetzt wird (C = Ø); das vom Band gelesene Bit ist dann eine logische Ø. Die Subtraktion SBC-RDTDIS nimmt genau betrachtet einen Zeitvergleich zwischen der 3600-Hz- und der 2400-Hz-Periode vor, dessen Ergebnis die Carry-Flag anzeigt:
C = 1 → Bitwert = logisch 1
C = Ø → Bitwert = logisch Ø
Die Wirkung der Subtraktion demonstrieren recht anschaulich die beiden aus Domino-Steinen geschichteten Säulen in Bild 9: Jede der beiden Säulen A und B besteht aus 255 ($FF!) Steinen. Während der 3600-Hz-Periode wird von Säule A alle 64 µs ein Stein abgehoben; das Gleiche geschieht mit Säule B während der 2400-Hz-Periode. War das vom Band gelesene Bit eine logische 1, dann überragt Säule A die Säule B. Umgekehrt bleiben von Säule B mehr Steine übrig als von Säule A, wenn das vom Band gelesene Bit eine logische Ø war. Angemerkt muß hier noch werden, daß die zur Unterdrückung des Jitter dienenden Warteschleifen keinen Einfluß auf die Höhe der Säulen A und B (die Timer-Endstände) haben.
Da das nominale Zeitverhältnis zwischen der 3600-Hz- und der 2400-Hz-Periode entweder 2:1 oder 1:2 beträgt, fällt das Ergebnis der Subtraktion entweder "deutlich positiv" oder "deutlich negativ" aus. Das gilt auch dann, wenn die Bandgeschwindigkeit bei der Wiedergabe von der Geschwindigkeit bei der Aufnahme relativ stark abweicht.
Rechnerisch ergibt sich für jedes Bit eine Bitzeit von insgesamt ca. 1250 µs:
Bitwert Ø: 833 µs mit 3600 Hz und 417 µs mit 2400 Hz
Bitwert 1: 417 µs mit 3600 Hz und 833 µs mit 2400 Hz
Der Bitzeit 1250 µs entspricht die Baudrate 800 bit/s.
Während eines Zeitintervalls von 833 µs wird der Timer-Stand 13 mal um eins vermindert, bis zum Ablauf des 417-µs-Intervalls geschieht dies 6 mal. Weil zu Beginn jedes Intervalls der Wert $FF (dezimal 255) in das Timer-Datenregister gesetzt wird, ist noch ein weiter Spielraum bis zum Time Out

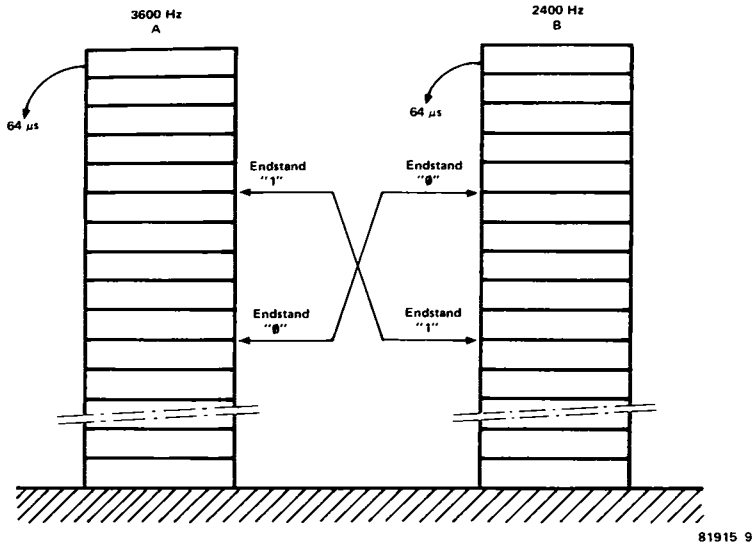


Bild 9. Die beiden Säulen veranschaulichen die Arbeitsweise von Subroutine RDBIT (Bild 7). Abhängig von der Signalfrequenz werden im 64-µs-Takt entweder von Säule A oder von Säule B einzelne Steine abgetragen. Entscheidend für das Ergebnis ist die entstandene Höhendifferenz zwischen A und B.

vorhanden. Mit anderen Worten: Die beiden Säulen in Bild 9 werden bei weitem nicht vollständig abgetragen. Auch dann bleibt noch ein stattlicher Rest stehen, wenn man mit der um den Faktor 6 langsameren Datenübertragungsgeschwindigkeit des KIM arbeitet. Das Ergebnis der Subtraktion ist positiv, unabhängig davon, ob man von der Zahl 255 die Zahl 13 oder die Zahl 78 (= 6 · 13) abzieht.
Eine andere Frage zu Subroutine RDBIT ist noch offen. Für das erste vom Band gelesene Bit existiert kein vorangegangener Aufruf von RDBIT, so daß der Beginn der ersten 3600-Hz-Periode nicht detektiert werden kann und der Intervall-Timer nicht startet. Das Lesen des Datenblocks wird davon jedoch nicht beeinträchtigt, denn das erste Bit gehört zum ersten vom Band gelesenen Synchronisationszeichen. Wenn das erste Synchronisationszeichen fehlerhaft gelesen wird (das Gegenteil ist nicht ausgeschlossen!), so bleiben noch 254 weitere Synchronisationszeichen übrig.
Zum Schluß dieses Abschnitts noch eine Anmerkung, die in ähnlicher Form schon unter Punkt 4 (Subroutinen HIGH und LOW von Schreibroutine DUMP) zu lesen war: Die Zeit, die für das Ausführen der letzten Instruktion von RDBIT (nach dem Timer-Start) notwendig ist, darf zusammen mit der Zeit für das Ausführen von Instruktionen vor dem nächsten Aufruf von RDBIT nicht so lang sein, daß die 3600-Hz-Periode bereits verstrichen ist. Verfügbar sind für die letzten vier Instruktionen von RDBIT (einschließlich RTS) und die vor dem nächsten Aufruf von RDBIT zu erledigenden Instruktionen insgesamt rund 400 µs.

13 Die auf dem Band stehenden Bit sind Bestandteile von ASCII-Zeichen; zu jedem ASCII-Zeichen gehören 8 aufeinander folgende Bit. Das Zusammensetzen von jeweils 8 Bit zu einem ASCII-Zeichen übernimmt beim Lesen vom Band **die Subroutine RDCH** in Bild 10.
Von Subroutine DUMP waren die ASCII-Zeichen in der Weise auf das Band geschrieben worden, daß zuerst das niederwertigste Bit bØ und zuletzt das höchstwertige Bit b7 aufgezeichnet wurde (siehe Bild 3 und Subroutine OUTCH, Punkt 5). In der gleichen Reihenfolge werden die Bit nun vom Band gelesen.
Subroutine RDCH (Bild 10) beginnt mit dem Laden des Werts Ø8 in das X-Register. Danach werden achtmal nacheinander folgende Instruktionen ausgeführt: JSR-RDBIT (lies ein Bit vom Band), ROR-CHAR (schiebe sämtliche Bit von Speicherplatz CHAR eine Stelle nach rechts und setze b7 gleich C) und DEX (Vorbereitung für das nächste Bit).
Wenn alle Bit eingelesen sind, folgen die Instruktionen ROL-CHAR und LSR-CHAR. Dadurch ist sichergestellt, daß Bit b7 von CHAR Null ist. Dieses Bit kann als Paritätsbit verwendet werden; der Junior-Computer macht davon jedoch keinen Gebrauch. Schon beim Schreiben der ASCII-Zeichen auf das Band wurde deshalb Bit b7 auf Null gesetzt, und auch die

10

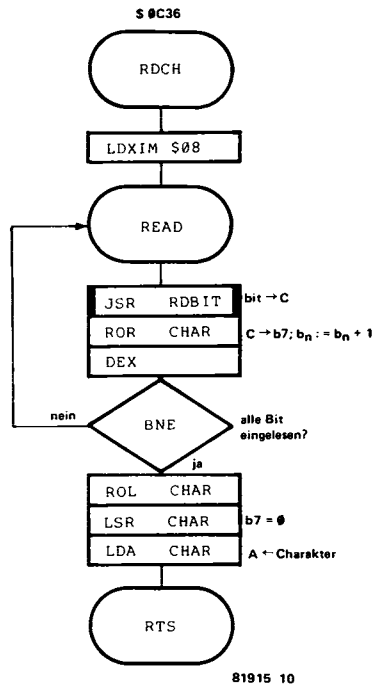


Bild 10. Subroutine RDCH liest ein einzelnes ASCII-Zeichen vom Band und setzt dessen Kode in den Akku.

Inhalte von CHKH und CHKL kamen ohne Paritätsbit zustande. Das zweite Nullsetzen von b7 während des Lesens hat den Zweck, etwaige Unstimmigkeiten der Werte in CHKH und CHKL mit den neu errechneten Prüfwerten in dem Fall zu vermeiden, daß durch irgendwelche Umstände doch einmal ein vom Band gelesenes Bit b7 logisch 1 geworden ist. Wenn dagegen eine Datensendung gelesen wird, bei der dem Bit b7 die Rolle des Paritätsbit zufällt (Aufnahme mit einem anderen Mikrocomputer), dann kann das konsequente Nullsetzen von b7 innerhalb von RDCH das Verlassen von Subroutine RDTAPE unmöglich machen. Dies ist sogar wahrscheinlich, weil die errechneten Endwerte von CHKH und CHKL mit den auf Band stehenden Werten in aller Regel nicht übereinstimmen werden.
Die letzte Instruktion von Subroutine RDCH kopiert noch den Inhalt des Speicherplatzes CHAR in den Akku (LDA-CHAR); dann folgt der Rücksprung RTS.

14 Ein weiterer Schritt bei der Rückgewinnung der Daten, die ja vor dem Schreiben auf Band als Bytes im Speicher des Junior-Computers standen, ist das Zusammensetzen von zwei Daten-Nibble zu einem Daten-Byte. Diesem Zweck dient **Subroutine RDBYT** in Bild 11.
Subroutine DUMP setzte während der Aufnahme zuerst das linke und danach das rechte Nibble der einzelnen Daten-Bytes auf das Band (siehe Punkt 6). Wie nicht anders zu erwarten ist, werden die Nibble in der gleichen Reihenfolge auch wieder vom Band gelesen. Nach dieser Vorbemerkung betrachten wir das Flußdiagramm von RDBYT in Bild 11.
RDBYT beginnt mit einer anderen Subroutine: RDCH, beschrieben unter Punkt 13 und dargestellt in Bild 10, liest einen ASCII-Kode vom Band und setzt ihn in den Akku. Entweder ist dieser Kode ein Daten-Nibble, oder es ist der Kode des Datenblock-Endzeichens "/". Im zweiten Fall folgt mit gesetzter Z-Flag (Z = 1) der Rücksprung aus Subroutine RDBYT. Handelt es sich nicht um das Datenblock-Endzeichen, so verzweigt die Subroutine zum Label RBB. Hier wird zuerst die **Subroutine ASCHEX** aus Bild 12 aufgerufen; ihre Besprechung wollen wir an dieser Stelle einschieben.
Der ASCII-Kode eines Daten-Nibble muß in den zugehörigen Wert "übersetzt" werden, damit aus zwei Nibble ein Daten-Byte entstehen kann. Subroutine ASCHEX nimmt die Umwandlung vor, nachdem sie geprüft hat, ob im Akku tatsächlich ein ASCII-Kode der Ziffern Ø . . . 9 oder A . . . F steht. Den Ziffern Ø . . . 9 entsprechen die ASCII-Kodes 3Ø . . . 39 und den Ziffern A . . . F (die Buchstaben A . . . F sind hier Hex-Ziffern!) entsprechen die ASCII-Kodes 41 . . . 46.
ASCHEX beginnt deshalb mit einem Frage- und Antwortspiel, das aus vier CMP-Instruktionen mit jeweils nachfolgendem BMI besteht. Enthält der Akku einen ASCII-Kode, der nicht zu den genannten Ziffern gehört, so ist das Label NOTVAL das nächste Ziel: ASCHEX wird dann mit gesetzter N-Flag (N = 1) verlassen. ASCII-Kodes von hexadezimalen Ziffern führen dagegen zum Label VALID. Die an VALID anschließenden Instruktionen wandeln den im Akku stehenden ASCII-Kode in den zugehörigen Wert um. Bei der Zifferngruppe Ø . . . 9 genügt das Nullsetzen des linken Nibble durch AND $ØF, während bei den Ziffern A . . . F vorher noch $Ø9 zum Akku-Inhalt addiert wird. Danach folgt mit N = Ø der Rücksprung aus

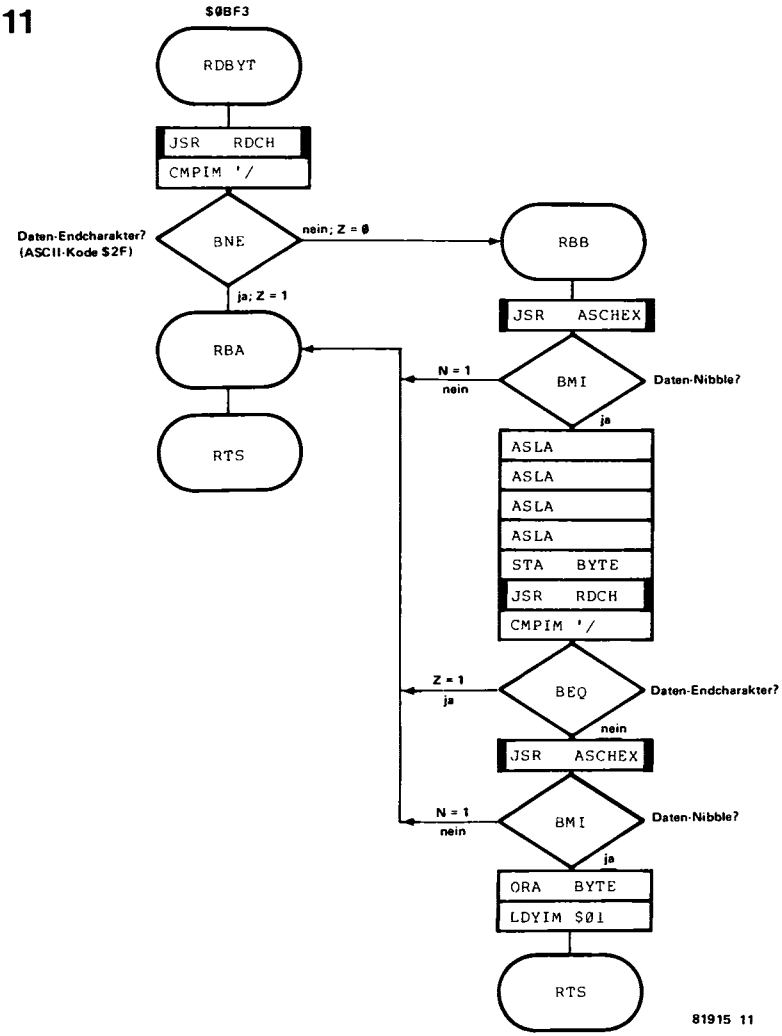


Bild 11. Ein einzelnes Daten-Byte wird von Subroutine RDBYT vom Band gelesen; das Daten-Byte besteht aus zwei nacheinander gelesenen Nibble.

Subroutine ASCHEX.
Zurück zu Subroutine RDBYT in Bild 11. Die BMI-Verzweigung nach dem ersten Aufruf von Subroutine ASCHEX führt bei gesetzter N-Flag (kein ASCII-Kode eines Daten-Nibble im Akku) unmittelbar zum Rücksprung aus RDBYT. Anderenfalls folgen vier Instruktionen ASLA, die das im Akku stehende Daten-Nibble zum linken Nibble des Akku-Inhalts machen. Nachdem der neue Akku-Inhalt in Speicherplatz BYTE steht (STA-BYTE),

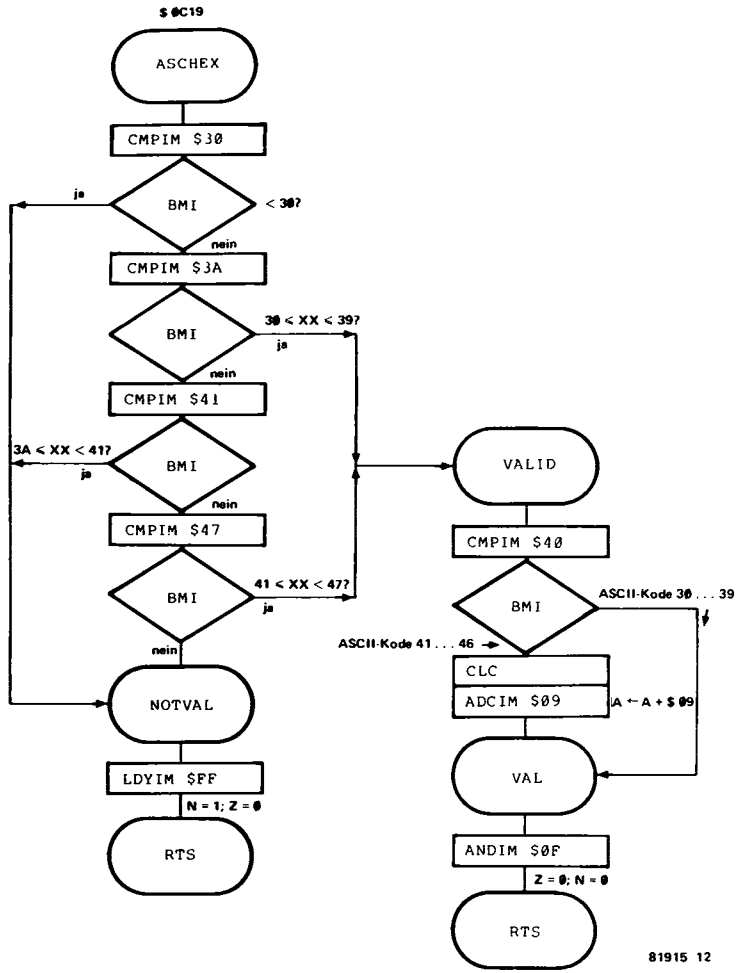


Bild 12. "Sub-Subroutine" (oder noch genauer: "Sub-Sub-Subroutine") ASCHEX wird von Subroutine RDBYT (Bild 11) zweimal aufgerufen. ASCHEX wandelt einen ASCII-Kode in ein Daten-Nibble um.

wird Subroutine RDCH aufgerufen und mit ihr der nächste ASCII-Kode eingelesen. Wenn dies der Kode des Datenblock-Endzeichens ist, folgt über Label RBA mit Z = 1 der Rücksprung aus Subroutine RDBYT. In allen anderen Fällen werden noch einmal die Dienste von Subroutine ASCHEX zu Hilfe genommen. Sofern ASCHEX den eingelesenen ASCII-Kode als Kode einer Hex-Ziffer identifiziert, macht die Instruktion ORA-BYTE das zweite Nibble zum rechten Daten-Nibble. Das aus zwei vom Band gelesenen ASCII-Kodes wiedergewonnene Daten-Byte steht nun im Akku. Vor

dem Rücksprung aus Subroutine RDBYT wird noch der Wert $Ø1 in das Y-Register geladen, so daß sowohl N = Ø als auch Z = Ø ist.

15 Den "Sub-Subroutinen" von RDTAPE sind noch zwei Subroutinen zuzurechnen, die in Zusammenhang mit der Display-Steuerung vor und während des Lesens vom Band stehen; siehe hierzu Bild 7 in Kapitel 11 (Buch 3). Die erste Subroutine ist die **Subroutine CHARVU/VU**, dargestellt in Bild 13 dieses Kapitels. Meistens wird diese Subroutine von Anfang an durchlaufen. Nur in zwei Fällen beginnt das Abarbeiten der Instruktionen erst beim Label VU: innerhalb von Subroutine BTWEEN (Bild 14 und Punkt 16) sowie während des Einlesens eines gesuchten und gefundenen Datenblocks in den Arbeitsspeicher.
Subroutine CHARVU rettet zuerst den Akku-Inhalt auf den Stack (PHA). Anschließend werden mit EOR#7F die Akku-Bits bØ . . . b6 invertiert und mit STA-PAD in Portregister PAD geladen. Damit ist der Akku-Inhalt zum neuen Segment-Muster geworden.
Stichwort Segment-Muster: Bild 15 zeigt zur Orientierung alle 128 Konfigurationen, in denen die Segmente der Siebensegment-Displays aufleuchten können. Die in der gleichen Zeile stehenden Muster haben das höherwertige Nibble (H) gemeinsam, bei den in gleicher Spalte stehenden Mustern ist das niederwertige Nibble (L) identisch.
Zurück zu Subroutine CHARVU: Nach Instruktion STA-PAD wird der ursprüngliche Akku-Inhalt mit dem Befehl PLA wiederhergestellt. Er wurde auf den Stack gerettet, weil der im Akku stehende ASCII-Kode noch weiterverarbeitet werden muß. Aus dem gleichen Grund ist auch die erste Instruktion nach Label VU ein PHA. Die nächsten vier Instruktionen bewirken folgendes:

A = PBD	:	Ø	Ø	1	1	Ø	Ø	1	Ø
EOR#Ø2	:	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø
Ergebnis	:	Ø	Ø	1	1	Ø	Ø	Ø	Ø

(A = PBD = GANG)

Am Anfang von "Super-Subroutine" RDTAPE war in Portregister PBD der Wert $32 geladen worden. Dieser Wert in PBD schaltet, wie unter Punkt 10 beschrieben wurde, das Display Di6 ein. Das Ergebnis der Instruktion EOR#Ø2 ist aber die Invertierung von Bit PB1, so daß Display-Multiplexer IC7 (siehe Buch 2, Seite 130, Bild 9) nun das Display Di5 aufleuchten läßt. Beim nächsten Durchlauf durch die beim Label VU beginnenden Instruktionen wird Bit PB1 erneut invertiert; das Zurückschalten auf Display Di6 ist diesmal die Folge. Wenn Subroutine CHARVU bzw. VU nicht nur einmal, sondern mehrfach nacheinander ohne längere Zwischenpausen aufgerufen wird, scheinen die beiden Displays Di5 und Di6 gleichzeitig aufzuleuchten! Dieses Multiplexen der Displays haben wir übrigens schon in Kapitel 7 (Buch 2) kennengelernt.

16 **Subroutine BTWEEN** in Bild 14 ist die zweite Subroutine, die an der Display-Steuerung innerhalb von Systemprogramm TM beteiligt ist. Sie wird immer dann aufgerufen, wenn das "Suchmuster" (drei waagerechte Striche übereinander; siehe Buch 3, Seite 92, Bild 7) auf den Displays Di5 und Di6 erscheinen soll. Nach Retten des Akku-Inhalts auf

13

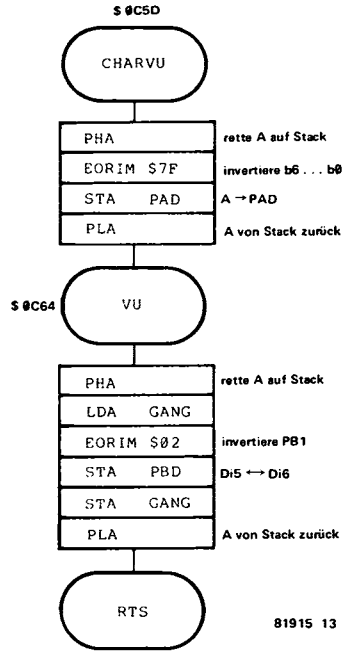


Bild 13. Der beim Label VU beginnende Teil von Subroutine CHARVU schaltet zwischen den Displays Di5 und Di6 hin und her. Da VU bzw. CHARVU in regelmäßigen Zeitabständen aufgerufen wird, leuchten beide Displays anscheinend gleichzeitig auf. Beginnt die Routine beim Label CHARVU, so werden außerdem noch die Bits b0 . . . b6 des Segment-Musters invertiert.

14

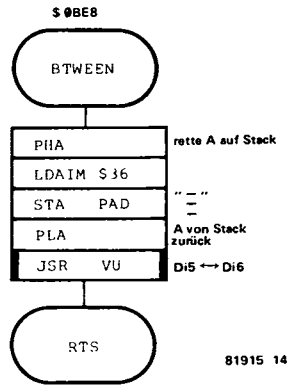


Bild 14. Subroutine BTWEEN läßt das "Suchmuster" auf den Displays erscheinen: dies sind die drei waagerechten Segmente. Solange noch keine Synchronisationszeichen vom Band kommen, zeigt Di6 dieses Muster ständig, Di5 dagegen nur gelegentlich. Das gleichzeitige und gleichmäßige Aufleuchten beider Displays signalisiert, daß Synchronisationszeichen eingelesen werden.

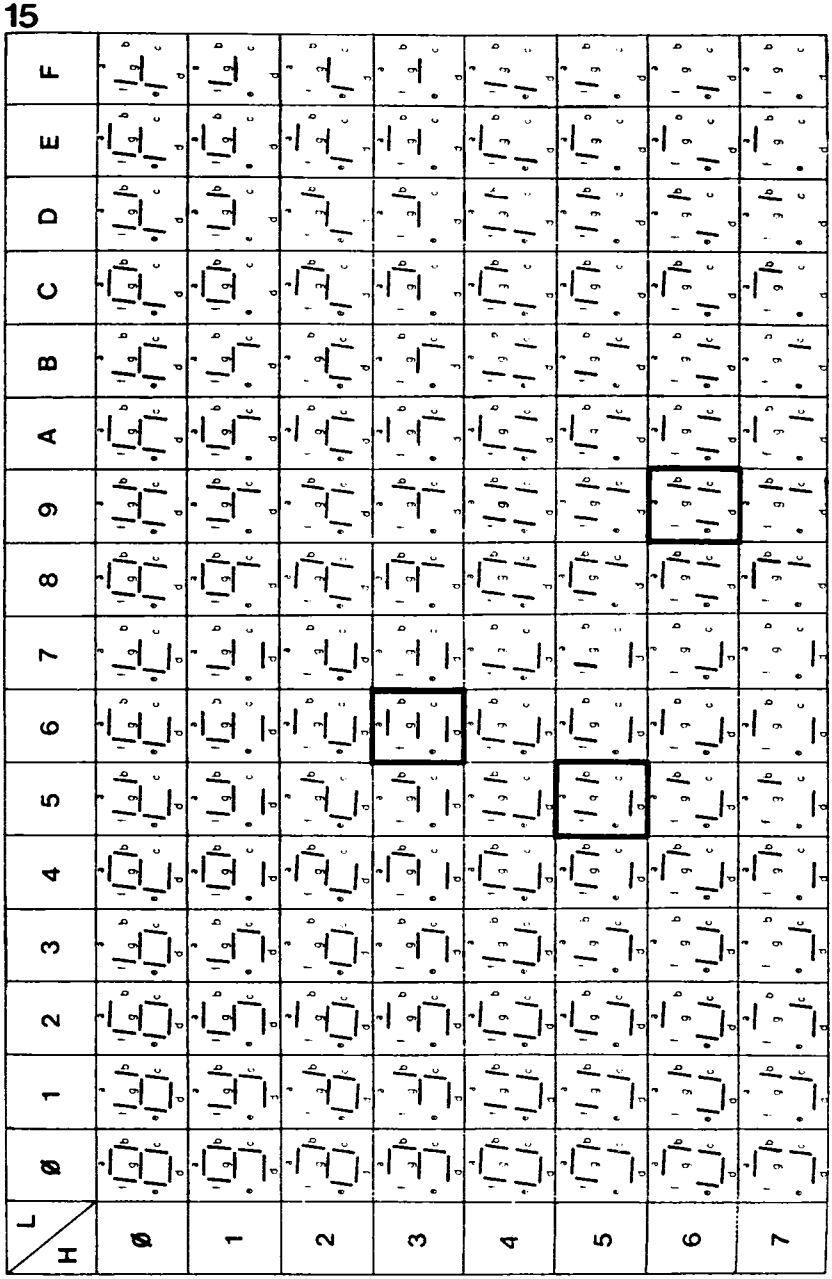


Bild 15. Übersicht über alle 128 Zeichen, die auf einem Siebensegment-Display des Junior-Computers darstellbar sind. H ist das linke, L das rechte Nibble des zugehörigen hexadezimalen Kode.

den Stack wird in Portregister PAD der Wert $36 gesetzt; das zugehörige Segment-Muster läßt sich aus Bild 15 entnehmen. Anschließend stellt der Befehl PLA den ursprünglichen Akku-Inhalt wieder her. Der Sprung zur Subroutine VU hat zur Folge, daß die Displays Di5 und Di6 wie beschrieben einander abwechseln. Das von den aufleuchtenden Segmenten gebildete Muster bleibt so lange unverändert, bis Portregister PAD durch einen Aufruf von Subroutine CHARVU einen neuen Inhalt erhält.

17 (Fortsetzung von Punkt 10)
Wir kehren nun zu "Super-Subroutine" **RDTAPE** zurück, deren Besprechung wir in Punkt 10 beim Label SYNC (siehe Bild 6a) unterbrochen hatten. Nach Label SYNC wird zuerst in Speicherplatz CHAR der Wert $FF gesetzt. Dieser Speicherplatz hat innerhalb von Subroutine RDCH eine bestimmte Funktion: siehe Punkt 13. Der Anfangswert von CHAR wird jedoch nicht von Subroutine RDCH selbst, sondern von RDTAPE festgelegt. In Kürze kommen wir darauf noch einmal zurück.
Als nächstes folgt das Label SYNCA. Subroutine RDBIT (besprochen unter Punkt 10 und 11) liest das erste vom Band kommende Bit ein. Dieses Bit wird durch ROR-CHAR zum Bit b7 von Speicherplatz CHAR und durch LDA-CHAR auch zum Bit b7 des Akku-Inhalts. Anschließend läßt Subroutine BTWEEN das "Suchmuster" (die drei waagerechten Segmente) auf Display Di5 oder Di6 erscheinen; siehe Punkt 16. Die Instruktion CMP# 16 vergleicht den Akku-Inhalt mit dem Kode des Synchronisationszeichens, der bekanntlich $16 lautet.
Natürlich kann nach Einlesen eines einzigen Bit noch kein vollständiges, vom Band stammendes Synchronisationszeichen in CHAR stehen, denn der Inhalt von CHAR ist abhängig vom Wert des eingelesenen Bit entweder
Ø 1 1 1 1 1 1 1 oder
1 1 1 1 1 1 1 1 (= $FF).
Das Synchronisationszeichen hat jedoch das Bitmuster
Ø Ø Ø 1 Ø 1 1 Ø.
Das Laden des Anfangswerts $FF in Speicherplatz CHAR hat folgenden Grund: Es verhindert das Auftreten eines Synchronisationszeichens, solange noch nicht acht Bit nacheinander eingelesen sind. Ohne diese Maßnahme steht in CHAR zunächst ein willkürliches Bitmuster, das zusammen mit einigen eingelesenen Bit bereits das Synchronisationszeichen ergeben könnte.
Ist ein Synchronisationszeichen vollständig eingelesen, so wird in den Speicherplatz SY der Wert $ØA gesetzt. Das steht damit in Zusammenhang, daß zuerst zehn ununterbrochen aufeinanderfolgende Synchronisationszeichen vom Band gelesen werden müssen, bevor RDTAPE den Beginn einer Datensendung erkennt. Wenn die BNE-Verzweigung nach Label TENSYN auf Springen steht, folgt ein Rücksprung zum Label SYNC. Der Versuch, zehn Synchronisationszeichen vom Band zu lesen, startet dann noch einmal.
Innerhalb der Programmschleife um das Label TENSYN wird außer der Subroutine RDCH auch die Subroutine CHARVU aufgerufen. CHARVU (siehe Punkt 15) läßt das Display aufleuchten: Wenn nicht von der vorangegangenen Subroutine RDCH ein vom Synchronisationszeichen abweichendes Bitmuster eingelesen wurde, steht im Akku der Wert $16. Nach

Invertierung der Bits bØ . . . b6 lädt CHARVU den Wert $69 in Portregister PAD. Aus Bild 15 geht hervor, daß zu $69 die Segment-Konfiguration gehört, die in Buch 3, Seite 92, Bild 7 mit ② bezeichnet ist. Das Segment-Muster ② erscheint sowohl auf dem Display Di5 als auch auf Di6, weil bei jedem Durchlauf durch CHARVU ein Wechsel zwischen diesen beiden Displays stattfindet. Solange Synchronisationszeichen vom Band eintreffen, wird Subroutine CHARVU ständig erneut aufgerufen. Entweder geschieht dies innerhalb der Programmschleife um das Label TENSYN oder aber während der nachfolgenden Programmschleife um das Label STAR (Bild 6b oben). Dabei werden die Displays Di5 und Di6 so schnell abwechselnd ein- und ausgeschaltet, daß sie für das Auge des Betrachters gleichzeitig aufleuchten.

18 Wenn 10 Synchronisationszeichen ohne Unterbrechung registriert wurden (insgesamt müssen 11 Synchronisationszeichen vom Band gelesen werden; Label TENSYN wird erst nach dem ersten Zeichen erreicht), befinden wir uns beim Label STAR, oben in Bild 6b. Unterbrochen wird die hier beginnende Schleife erst, nachdem alle noch folgenden Synchronisationszeichen (maximal 255 – 11 = 244) den Wiedergabekopf des Kassettenrekorders passiert haben. Anschließend stehen zwei Möglichkeiten offen: Entweder wird das Datenblock-Beginnzeichen "*" eingelesen, so daß die Routine zum Label STARA verzweigt. Oder das eingelesene Zeichen ist weder ein Synchronisationszeichen noch das Datenblock-Beginnzeichen. In diesem Fall lief offensichtlich etwas fehl; es folgt mit dem Rücksprung zum Label RDTAPE ein neuer Anlauf.
Zum Label STARA in Bild 6b gelangen wir nur, wenn das Datenblock-Beginnzeichen "*" erkannt wurde; sein ASCII-Kode lautet $2A. Durch Invertieren der Bits bØ . . . b6 und Laden des so veränderten Bitmusters in Portregister PAD (Subroutine CHARVU, siehe Punkt 15) entsteht auf den Displays das in Bild 15 unter dem Kode $55 angegebene Segment-Muster. Dies ist das dritte Muster, das Bild 7 in Kapitel 11 (Buch 3, Seite 92) zeigt.
Wie lange dieses Muster auf den Displays sichtbar bleibt, entscheidet sich während der nächsten Phase von Subroutine RDTAPE. Sie beginnt damit, daß RDTAPE die Programmnummer ID einliest. Nach Abschluß von Subroutine RDBYT steht die zum Datenblock gehörende Programmnummer im Akku. Durch einen Vergleich mit der vom Programmierer eingegebenen und in Speicherplatz ID stehenden Programmnummer stellt RDTAPE fest, ob es sich um den gewünschten Datenblock handelt. Trifft dies zu, dann wird RDTAPE beim Label RDSA fortgesetzt. Stimmen die eingegebene und die vom Band gelesene Programmnummer nicht überein, so wird zuerst geprüft, ob eine der beiden speziellen Programmnummern ØØ oder FF eingegeben wurde. Dazu verzweigt die Routine zum Label CHKID.
Wurde als Programmnummer weder ØØ noch FF gewählt, dann ist der Rücksprung über Zwischenlabel SYNVEC zum Anfang von Subroutine RDTAPE unausweichlich. Die gerade vom Band gelesene Datensendung ist offenbar nicht die gewünschte, so daß die nächste auf dem Band stehende Datensendung abgewartet werden muß. In diesem Fall erscheint auch nur kurzzeitig das Muster auf den Displays, das in Bild 15 dem Kode $55 entspricht. Das Muster ist während eines Zeitabschnitts sichtbar, der mit

dem Aufruf von Subroutine CHARVU (nach Label STARA) beginnt und mit dem Aufruf von Subroutine BTWEEN (Bild 6a) endet, nachdem das Programm über Zwischenlabel SYNVEC zum Label RDTAPE zurückgekehrt ist.
Wir wollen kurz die **Subroutine CHKSUM** in Bild 16 betrachten, bevor wir die Besprechung von RDTAPE fortsetzen. Subroutine CHKSUM addiert zu der 16-bit-Zahl CHK = (CHKH, CHKL) den Betrag, der unmittelbar nach dem Aufruf von CHKSUM im Akku steht. Die 8-bit-Zahl, die sich dort zu diesem Zeitpunkt befindet, wird von Subroutine CHKSUM nicht verändert; hierfür sorgen die Instruktionen PHA und PLA am Anfang bzw. Ende von CHKSUM.
Zurück zum zweiten Teil von "Super-Subroutine" RDTAPE in Bild 6b. Nach der Betrachtung des Falls, daß eine nicht gewünschte Programmnummer vom Band gelesen wurde, bleiben noch die folgenden drei Möglichkeiten offen:

1. **Programmnummer (= Ø1 . . . FE) gefunden:** Die Instruktionen vom Label RDSA bis zum Label FILMEM werden ausgeführt. RDBYT liest das rechte Startadreßbyte SAL ein, und CHKSUM addiert diese 8-bit-Zahl zur Prüfsumme CHK. Das Gleiche geschieht anschließend mit dem linken Startadreßbyte SAH. Ferner wird Adreßpointer POINT auf die zum Datenblock gehörende, vom Band gelesene Startadresse SA gerichtet.
2. **Programmnummer ØØ eingegeben:** Auch in diesem Fall ist Label RDSA mit den Instruktionen bis zum Label FILMEM das nächste Ziel; siehe unter 1. Mit ID = ØØ bleibt nämlich die vom Band gelesene Programmnummer unberücksichtigt, während die vom Band gelesene Startadresse ihre Funktion beibehält.

16

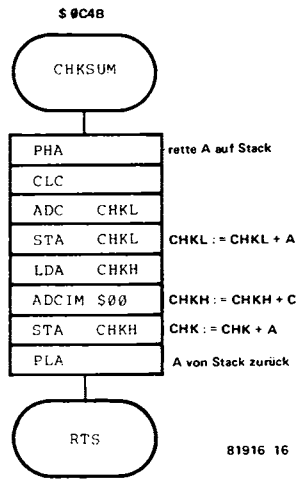


Bild 16. Aufgabe von Subroutine CHKSUM ist die Addition der 8-bit-Zahl, die das gerade eingelesene Byte darstellt, zu der in den Speicherplätzen CHKH/CHKL stehenden 16-bit-Zahl.

3. **Programmnummer FF eingegeben:** Die vom Band gelesene Startadresse SA geht durch zweimaligen Aufruf der Subroutinen RDBYT und CHKSUM in die Prüfsumme CHK ein (zuerst das niederwertige und danach das höherwertige Byte). Anschließend wird Adreßpointer POINT auf die vom Programmierer eingegebene Startadresse SA gerichtet, die in Speicherplatz SAL/SAH steht. Die Programmnummer FF bedeutet nämlich, daß sowohl die auf dem Band stehende Programmnummer als auch die dort stehende Startadresse wirkungslos ist. Trotzdem wird die vom Band gelesene Startadresse in die Prüfsumme CHK einbezogen, denn anderenfalls würden hier natürlich Unstimmigkeiten auftreten. Nach Spezifizieren von POINT springt die Routine zum Label FILMEM.

19 Die auf Label FILMEM folgenden Instruktionen der Subroutine RDTAPE (Bild 6b) lesen den gewünschten Datenblock vom Band und laden ihn in den Arbeitsspeicher des Junior-Computers. Zuerst wird Subroutine RDBYT aufgerufen; sie wurde unter Punkt 14 besprochen und ist in den Bildern 11 und 12 dargestellt. Erkennt RDBYT, daß das eingelesene Byte nicht zu den zulässigen Daten gehört (N-Flag = 1), dann bricht ein Rücksprung zum Label RDTAPE das Lesen des Datenblocks ab. Handelt es sich um das Datenblock-Endzeichen "/", so folgt ein Sprung zum Label CHECK. In allen übrigen Fällen werden zugelassene Daten vom Band gelesen; sie werden zuerst von Subroutine CHKSUM zur Prüfsumme CHK addiert. Die nächsten Instruktionen laden das eingelesene Byte in den Speicherplatz, dessen Adresse der Adreßpointer POINT angibt. Zu Beginn der Einleseaktion zeigt POINT auf Startadresse SA; sie wurde entweder ebenfalls vom Band gelesen oder vom Programmierer eingegeben (vgl. Punkt 18).
Nach Laden des Byte in den von POINT angewiesenen Speicherplatz wird der Inhalt von POINT um eins erhöht. An Label FMA, das wir jetzt passieren, schließt sich ein Aufruf von Subroutine VU an. Das hat zur Folge, daß ein Display-Wechsel von Di5 nach Di6 bzw. umgekehrt stattfindet. In Portregister PAD steht übrigens noch immer das Segment-Muster, das Subroutine CHARVU nach dem Label STARA dort hineingesetzt hatte; es ist das Muster mit dem Kode $55 (siehe Bild 15). Sichtbar ist dieses Muster sowohl auf Di5 als auch auf Di6, weil die beiden Displays nach jedem Einlesen eines Byte von Subroutine VU (siehe Bild 13 und Punkt 15) umgeschaltet werden. Um das nächste Byte einzulesen, springt die Routine nach dem Aufruf von Subroutine VU zurück zum Label FILMEM.
Übrig bleiben von RDTAPE in Bild 6b noch die auf Label CHECK folgenden Instruktionen. Sie vergleichen die auf dem Band aufgezeichnete rechnerische Summe aller Byte-Werte mit der während des Einlesens ermittelten Summe. Nach Erkennen des Datenblock-Endzeichens "/" geschieht dies zuerst mit den niederwertigen Bytes CHKL und dann, sofern die niederwertigen Bytes übereinstimmen, mit den höherwertigen Bytes CHKH der beiden 16-bit-Zahlen. Eine Differenz zwischen der auf dem Band stehenden und der neu errechneten Prüfsumme führt über Zwischenlabel SYNVEC zur Rückkehr an den Anfang der Subroutine RDTAPE. Nur wenn beide Werte gleich sind, wird mit dem Rücksprung RTS die Sub-

routine RDTAPE verlassen.
Stellt Subroutine RDTAPE Unstimmigkeiten zwischen den beiden Prüfsummen fest, so wurde zwar vorher der gelesene Datenblock in den Arbeitsspeicher des Computers geladen, Subroutine RDTAPE wird aber nicht verlassen. Dies ist auch auf den Displays sichtbar: das Muster mit dem Kode $55 (siehe Bild 15) wird vom Muster mit dem Kode $63 abgelöst. Dagegen verlischt das Display, sobald RDTAPE nach einem von PM ausgegangenen Aufruf beendet ist. Das erfolgreiche Laden eines Datenblocks vom Band signalisiert bei PM außerdem die Rückmeldung "READY". Wenn RDTAPE von TM aufgerufen wurde, erscheint nach erfolgreichem Laden eines Datenblocks die Rückmeldung "id XX" auf den Siebensegment-Displays.

20 **Die Hauptroutine des Systemprogramms TM** ist in den Bildern 17a, 17b und 17c dargestellt. Ihre Struktur weicht nicht wesentlicht von der Struktur der übrigen Junior-Computer-Systemprogramme ab. Am Anfang der TM-Hauptroutine steht eine Vorroutine; sie beginnt beim Label TPINIT und endet beim Label TPTXT, dem zentralen Label der TM-Hauptroutine. TPTXT ist nämlich das Sprungziel nach Ablauf der PAR-Tastenroutine (siehe Punkt 24) und nach Eingabe von Daten durch den Programmierer, die für einen der Speicherplätze ID, SAH, SAL, EAH, EAL, BEGADH, BEGADL, ENDADH und ENDADL bestimmt sind. Label TPTXT wird auch nach Ablauf der GET-Tastenroutine (Punkt 22) und der SAVE-Tastenroutine (Punkt 23) angesteuert, nachdem vorher der zweite, beim Label TPI beginnende Teil der TM-Vorroutine durchlaufen wurde. Am Schluß der SEF-Tastenroutine (Punkt 25) und der EDIT-Tastenroutine (ebenfalls Punkt 25) wird das Systemprogramm TM verlassen. Sprungziel ist hier der EDITOR; er ist in dem EPROM untergebracht, das sich auf der Basisplatine des Junior-Computers befindet.
Unmittelbar nach dem Label TPTXT setzt die TM-Hauptroutine den Namen und den Inhalt des gerade aktuellen Speicherplatzes auf die Displays; anschließend wartet sie auf das Drücken einer Taste. Abhängig davon, welche Taste gedrückt wird, folgt entweder das Abarbeiten einer Funktionstastenroutine oder der Datentastenroutine. Danach kann das gleiche Spiel von neuem beginnen.
Ein kurzer Blick auf das Flußdiagramm von TM läßt erkennen, daß der mit DISCNT bezeichnete Speicherplatz recht häufig in Erscheinung tritt. Dieser Speicherplatz dient dem Systemprogramm TM als **Parameter-Zähler.** Der Inhalt von DISCNT bestimmt den Speicherplatz, in den die vom Programmierer einzugebenden Daten (die Parameter) geladen werden. Ferner hängt vom Inhalt des Speicherplatzes DISCNT ab, welcher Parameter-Speicherplatz mit seinem Namen und seinem Inhalt auf den Displays erscheint. Die einzelnen Werte, die in DISCNT stehen können, haben folgende Bedeutung:
DISCNT = ØØ → Anzeige/Eingabe von ID
DISCNT = Ø1 → Anzeige/Eingabe von SAH
DISCNT = Ø2 → Anzeige/Eingabe von SAL
DISCNT = Ø3 → Anzeige/Eingabe von EAH
DISCNT = Ø4 → Anzeige/Eingabe von EAL
DISCNT = Ø5 → Anzeige/Eingabe von BEGADH

DISCNT = Ø6 → Anzeige/Eingabe von BEGADL
DISCNT = Ø7 → Anzeige/Eingabe von ENDADH
DISCNT = Ø8 → Anzeige/Eingabe von ENDADL
Nach dieser Übersicht über die Hauptroutine von TM kommen wir nun zu den Details.

21 **Die Vorroutine von TM** umfaßt die Instruktionen in Bild 17a, die sich zwischen den Labeln TPINIT und TPTXT befinden. Zuerst werden die Inhalte der fünf Speicherplätze gelöscht, die unmittelbar an der Datenspeicherung auf Band beteiligt sind: ID, SAH, SAL, EAH, und EAL. Nach Label TPI wird auch in den Parameter-Zähler DISCNT der Wert $ØØ gesetzt. Das gilt übrigens auch dann, wenn Label TPI aus einer der beiden anderen Richtungen erreicht wird; auch in diesen Fällen steht nämlich im X-Register der Wert $ØØ. Für die Anzeige auf den Displays bedeutet dies, daß sie nach jedem Passieren von Label TPI den Namen und den Inhalt von Speicherplatz ID wiedergeben.
Die nächste Aktion der TM-Vorroutine ist das Laden von Portregister PBD mit dem Wert $Ø6. Dadurch werden die Displays abgeschaltet, und auch die Tastatur wird nicht mehr abgefragt. Anschließend erklärt der in Portregister PBDD geladene Wert $1E die Portleitungen PB1 . . . PB4 zu Ausgängen. Was die Gründe für diese und die noch bis zum Label GET folgenden Instruktionen betrifft, so verweisen wir auf Kapitel 7 in Buch 2. Die nach dem zentralen Label TPTXT aufgerufene Subroutine TAPDIS zeigt weitgehende Ähnlichkeit mit der vertrauten Subroutine SCANDS; siehe hierzu Bild 19 und Punkt 27. In jedem Fall steht beim Erreichen des Labels GET der Wert der innerhalb von Subroutine GETKEY gedrückten und erkannten Taste im Akku.

22 Zur **GET-Tastenroutine** gehören die Instruktionen, die sich in Bild 17a unterhalb des Labels GET befinden. Das Drücken der GET-Taste hat einen Sprung zur Subroutine RDTAPE zur Folge; sie wurde unter den Punkten 10 . . . 19 dieses Kapitels besprochen. Sobald die vom Programmierer gewählte Datensendung den Wiedergabekopf des Kassettenrekorders passiert, wird von Subroutine RDTAPE der zugehörige Datenblock in den Arbeitsspeicher des Junior-Computers geladen. Vorher wurde vom Programmierer die Programmnummer ID dieses Datenblocks und im Fall ID = FF auch eine Startadresse SAH/SAL eingegeben.
Nach dem Rücksprung aus Subroutine RDTAPE prüft die Hauptroutine (Bild 17a), ob der eingelesene Datenblock mit FF als Programmnummer in den Computer geladen wurde. Wenn dies nicht zutrifft (Label GETB), wird der Inhalt des X-Registers durch den Befehl INX auf $ØØ gesetzt; danach folgt der Rücksprung über Label TPI zum Label TPTXT. Auf den Displays erscheint jetzt der Text "id XX". Der Umweg über Label TPI zum Label TPTXT ist notwendig, um die I/O-Ports wieder für die TM-Hauptroutine zu programmieren. Hier müssen nämlich die Portleitungen anders geschaltet sein als innerhalb der "Super-Subroutinen" DUMP/DUMPT und RDTAPE.
Wenn die Programmnummer FF gewählt wurde, verzweigt die TM-Hauptroutine zum Label GETA. Danach wird zuerst **die Subroutine ADJPNT** aufgerufen; sie ist in Bild 18 dargestellt. ADJPNT hat nur die Funktion,

17a

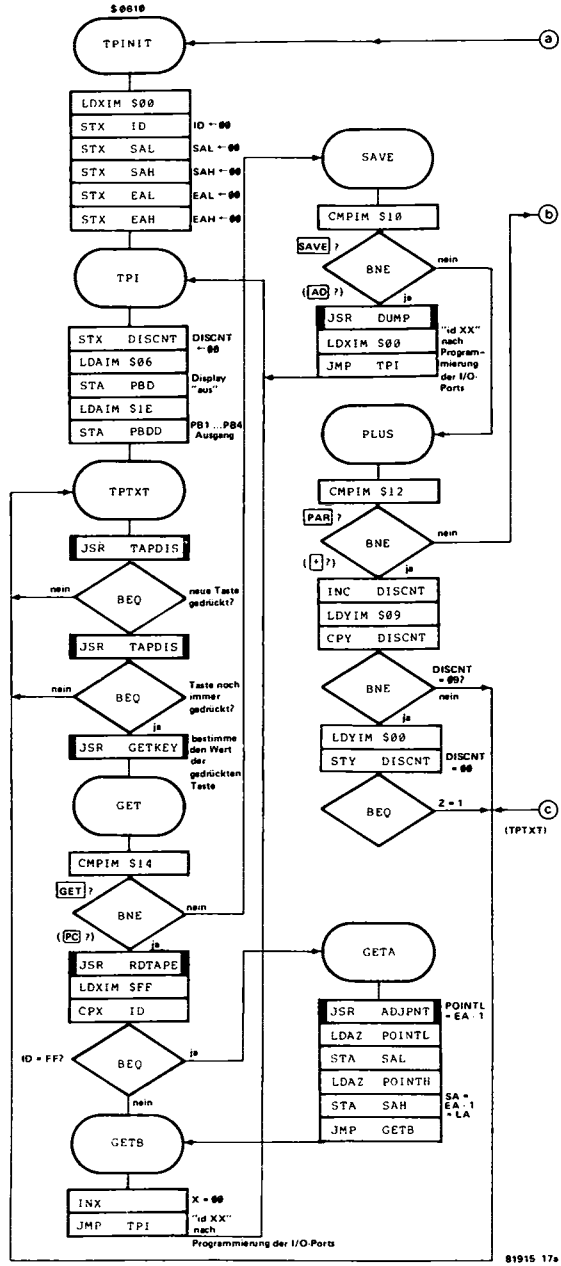


Bild 17a. Erster Teil der Hauptroutine von Systemprogramm TM.

17b

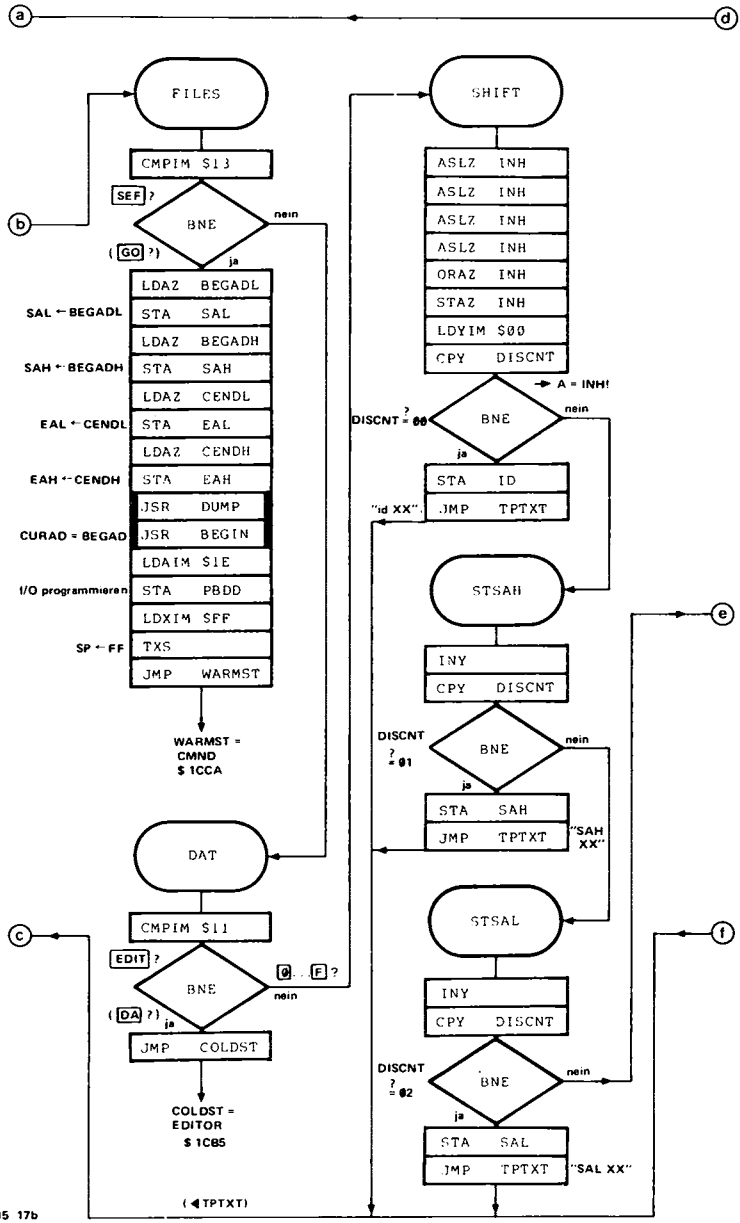


Bild 17b. Zweiter Teil der Hauptroutine von TM.

17c

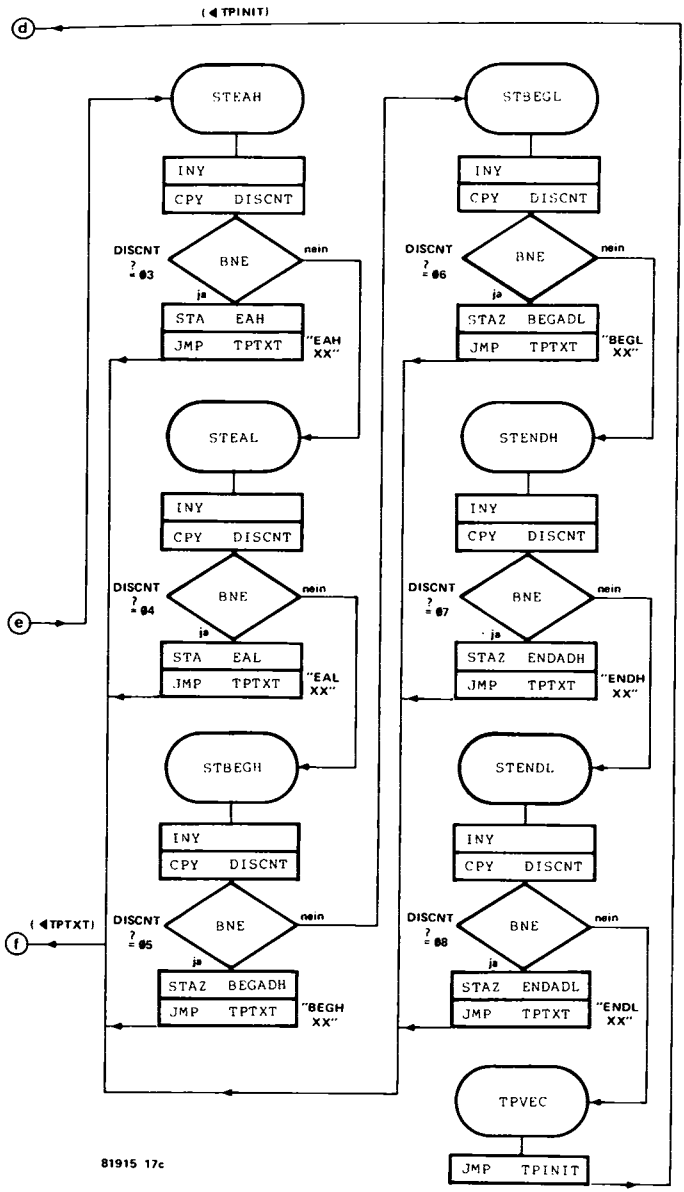


Bild 17c. Dritter Teil der TM-Hauptroutine.

18

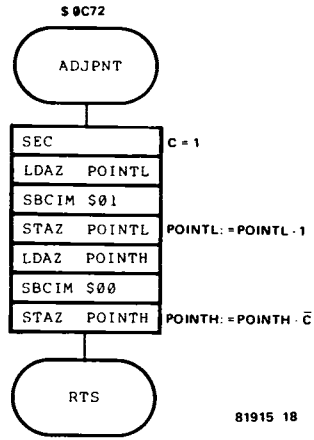


Bild 18. Subroutine ADJPNT setzt Adreßpointer POINT um eins herab, wenn dies für den nahtlosen Anschluß verschiedener vom Band gelesener Datenblöcke notwendig ist.

den Inhalt von Adreßpointer POINT um eins herabzusetzen. POINT zeigt während des Ablaufs von Subroutine RDTAPE auf den Speicherplatz, in den das gerade vom Band gelesene Daten-Byte gelesen werden soll. Anschließend wird dort der Adreßpointer POINT um eins erhöht. Steht der Datenblock vollständig im Speicher, so zeigt POINT auf die Adresse, die um eine höher als die letzte mit einem Daten-Byte vom Band geladene Adresse liegt. Der gleiche Sachverhalt in Kurzschreibweise: POINT = EA = LA + 1. Die Endadresse EA steht nicht auf dem Band, sondern ergibt sich aus Startadresse SA (SA steht entweder auf dem Band oder wurde bei ID = FF vom Programmierer eingegeben) und der Anzahl der Daten-Bytes plus eins.
Am Ende von Subroutine ADJPNT zeigt der Adreßpointer POINT nicht auf Endadresse EA, sondern auf die letzte Adresse LA des eingelesenen Datenblocks. Der Zweck dieser Operation ist folgender: Die auf Subroutine ADJPNT folgenden vier Instruktionen (siehe Bild 17a) machen die nun in POINT stehende letzte Adresse LA des eingelesenen Datenblocks zur neuen Startadresse SA. Sie ist die Startadresse für den Fall, daß ein weiterer Datenblock vom Band gelesen wird und dieser Datenblock an den ersten ohne Unterbrechung anschließen soll. Im allgemeinen wird es sich bei den aneinander anzufügenden Datenblöcken um Programme oder Programmteile handeln, die noch nicht assembliert sind. Auf die beschriebene Weise werden die zwischen den Blöcken liegenden EOF-Zeichen ($77) entfernt. Näheres über den Umgang mit der Programmnummer FF ist in Buch 3, Kapitel 11 zu finden; in diesem Zusammenhang sei insbesondere auf Bild 11 des genannten Kapitels hingewiesen.
Aus dem Flußdiagramm der GET-Tastenroutine wird deutlich, daß der Programmierer vor dem Laden des nächsten Datenblocks, der zum bereits im Speicher stehenden Datenblock hinzugefügt werden soll, nichts in den

Computer eingeben muß. Die Software hat nämlich nicht nur die Startadresse automatisch angepaßt, sondern auch in Speicherplatz ID steht immer noch die Programmnummer FF. Es ist nur dafür Sorge zu tragen, daß der nächste vollständig vom Band gelesene Datenblock tatsächlich der Datenblock ist, der mit dem ersten verbunden werden soll! Mit FF als Programmnummer liest der Junior-Computer bekanntlich den nächsten Datenblock ein, den ihm der Kassettenrekorder anbietet (siehe Buch 3, Kapitel 11).

23 **Die SAVE-Tastenroutine** besteht aus den fünf Instruktionen, über denen in Bild 17a das Label SAVE steht. Ein Druck auf die SAVE-Taste hat den Aufruf von Subroutine DUMP zur Folge; sie wurde ausführlich unter Punkt 1 . . . 9 dieses Kapitels beschrieben. Der zuvor spezifizierte Datenblock wird auf Band gespeichert, vorausgesetzt natürlich, daß eine Kassette in den Rekorder eingelegt ist und außerdem die Aufnahme-Taste am Kassettenrekorder gedrückt wurde. Vorher sind noch in die Speicherplätze ID, SAH, SAL, EAH und EAL die für den Datentransport notwendigen Parameter einzugeben.
Im Anschluß an Subroutine DUMP wird in das X-Register der Wert $ØØ geladen, so daß nach Erreichen des zentralen Labels TPTXT über das Label TPI der Text "id XX" auf den Displays erscheint. Dabei ist "XX" gleich der vom Programmierer spezifizierten Programmnummer ID. Der Umweg über Label TPI ist ebenso wie bei Tastenroutine GET (Punkt 22) deshalb notwendig, weil die I/O-Port-Programmierung nach Durchlaufen von Subroutine DUMP wieder an die Erfordernisse der TM-Hauptroutine angepaßt werden muß.

24 **Die PAR-Tastenroutine** umfaßt die in Bild 17a auf das Label PLUS folgenden Instruktionen. Nach Drücken der PAR-Taste wird Parameter-Zähler DISCNT um eins erhöht. Wenn hierdurch der Zähler-Inhalt den Wert $Ø9 erreicht, wird er wieder auf $ØØ rückgesetzt. Nach Drücken der PAR-Taste erscheint deshalb der jeweils nächste Parameter mit seinem Namen und dem Inhalt des zugehörigen Speicherplatzes auf den Displays. Hierbei folgt auf den letzten Parameter ("ENDLXX") wieder der erste ("id XX"). Die Reihenfolge der Parameter ist in Punkt 20 angegeben.

25 **Die SEF-Tastenroutine** beginnt beim Label FILES, links oben in Bild 17b. Ein Druck auf die SEF-Taste bewirkt, daß Startadresse SA gleich BEGAD und Endadresse EA gleich CEND gesetzt wird. CEND ist der variable Endadreß-Pointer des auf Band zu schreibenden, noch nicht assemblierten Programms. Dann folgt der Aufruf von Subroutine DUMP. Vor Drücken der SEF-Taste muß Programmnummer ID spezifiziert worden sein.
Nach der Rückkehr aus DUMP wird das Systemprogramm TM in Richtung des warmen EDITOR-Starteingangs verlassen. Zuvor richtet noch Subroutine BEGIN den Display-Pointer CURAD, auch Instruktions-Pointer genannt, auf die Adresse BEGAD; sie ist identisch mit Startadresse SA. Ferner wird Portregister PBDD in der Weise geladen, daß hier der gleiche Zustand wie nach der RESET-Vorroutine des Standard-Monitors vorhan-

den ist. Zu guter Letzt erhält noch der Stack-Pointer den Wert $FF; dann folgt der Sprung zum Warmstart-Eingang des EDITOR.
Zur **EDIT-Tastenroutine** (Bild 17b, Label DAT) ist nur anzumerken, daß das Systemprogramm auch hier in Richtung EDITOR verlassen wird. In diesem Fall ist das Sprungziel jedoch der Kaltstart-Eingang des EDITOR.

26 Wir kommen nun zu den numerischen Tasten Ø . . . F und damit zur Besprechung der **Datentasten-Routine**. Zu ihr gehört alles das, was von der TM-Hauptroutine noch übrig bleibt: alle Instruktionen unter dem Label SHIFT in Bild 17b (rechte Spalte) sowie die Instruktionen in Bild 17c.
Die Datentasten-Routine arbeitet wie folgt: Die von einer gedrückten numerischen Taste kommende Information wird als neues rechtes Nibble in Datenpuffer INH gesetzt. Das dort bereits stehende rechte Nibble räumt seinen Platz und wird zum linken Nibble. Was hierbei im einzelnen geschieht, wurde schon verschiedentlich an anderer Stelle erklärt, so daß wir jetzt darauf verzichten können.
Der veränderte Inhalt des Datenpuffers INH muß in einen der neun Parameter-Speicherplätze kopiert werden. In welchen Parameter-Speicherplatz die eingegebene Information gelangt, das hängt vom Stand des Parameter-Zählers DISCNT ab. Die Datentasten-Routine vergleicht deshalb den Inhalt von DISCNT einzeln mit den Werten $ØØ . . . $Ø8 und lädt die inzwischen im Akku stehende Tasteninformation in Speicherplatz

ID	(Label SHIFT),	wenn DISCNT = ØØ,
SAH	(Label STSAH),	wenn DISCNT = Ø1,
SAL	(Label STSAL),	wenn DISCNT = Ø2,
EAH	(Label STEAH),	wenn DISCNT = Ø3,
EAL	(Label STEAL),	wenn DISCNT = Ø4,
BEGADH	(Label STBEGH),	wenn DISCNT = Ø5,
BEGADL	(Label STBEGL),	wenn DISCNT = Ø6,
ENDADH	(Label STENDH),	wenn DISCNT = Ø7,
ENDADL	(Label STENDL),	wenn DISCNT = Ø8.

Danach folgt die Rückkehr zum zentralen Label TPTXT der TM-Hauptroutine, so daß die über eine oder zwei numerische Tasten eingegebene Information auf den Displays sichtbar werden kann.
Die Besprechung der TM-Hauptroutine ist damit abgeschlossen. Wir müssen nun noch eine Subroutine und eine "Sub-Subroutine" betrachten.

27 Die Subroutine TAPDIS in Bild 19 macht den Namen und Inhalt eines der neun Parameter-Speicherplätze ID . . . ENDADL auf den Displays sichtbar. Während auf den Displays Di1 . . . Di4 der Parameter-Name erscheint, geben die Displays Di5 und Di6 den zugehörigen Speicherplatz-Inhalt wieder. Wie schon in Kapitel 7 (Buch 2) ausführlich beschrieben, hängt vom Inhalt des X-Registers ab, welches der sechs Displays momentan aufleuchtet. Dabei besteht folgender Zusammenhang:
X = Ø8 → Display Di1
X = ØA → Display Di2
X = ØC → Display Di3
X = ØE → Display Di4
X = 1Ø → Display Di5
X = 12 → Display Di6

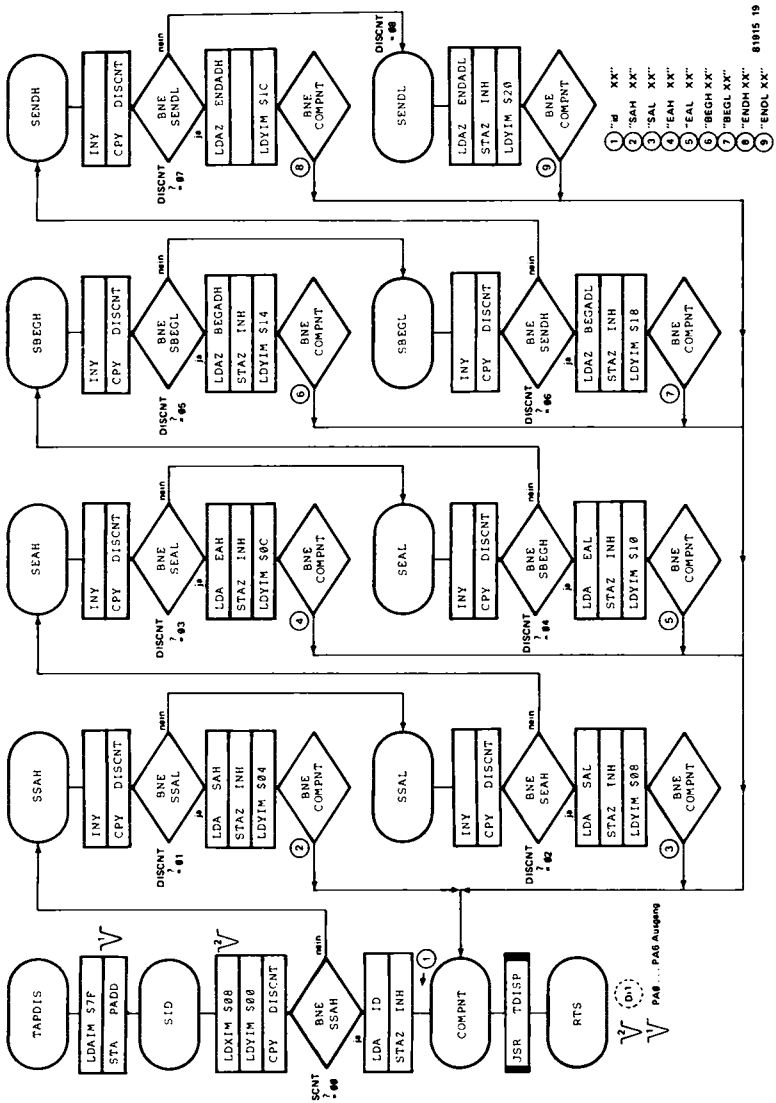


Bild 19. TM-Subroutine TAPDIS stellt den Parameter fest, der auf den Displays dargestellt werden muß. Die Darstellung selbst ist Aufgabe der Subroutine TDISP in Bild 20.

Subroutine TAPDIS (Bild 19) schaltet zuerst die Portleitungen PAØ . . . . PA6 als Ausgang. Diese Portleitungen steuern die Segmente der Siebensegment-Displays. Nach dem Label SID wird in das X-Register der Wert $Ø8 gesetzt, so daß Display D1 während der Subroutine TDISP als erstes aufleuchtet. Vor dem Aufruf von TDISP muß noch geklärt werden, welcher Parameter-Speicherplatz ausgelesen werden soll. Dazu wird der Inhalt von Parameter-Zähler DISCNT einzeln mit den Werten ØØ . . . Ø8 verglichen. Der jeweilige Vergleichswert steht hierbei im Y-Register. Anschließend erhält das Y-Register eine andere Aufgabe: Es dient der nachfolgenden Subroutine TDISP als Index-Register für die von ihr benutzte Look-Up-Tabelle.
Bei Erreichen des Labels COMPNT in Bild 19, also vor dem Aufruf von Subroutine TDISP, ist abhängig von Parameter-Zähler DISCNT eine der folgenden Situationen gegeben:

DISCNT = ØØ: INH ← Inhalt ID, Index Y = ØØ
DISCNT = Ø1: INH ← Inhalt SAH, Index Y = Ø4
DISCNT = Ø2: INH ← Inhalt SAL, Index Y = Ø8
DISCNT = Ø3: INH ← Inhalt EAH, Index Y = ØC
DISCNT = Ø4: INH ← Inhalt EAL, Index Y = 1Ø
DISCNT = Ø5: INH ← Inhalt BEGADH, Index Y = 14
DISCNT = Ø6: INH ← Inhalt BEGADL, Index Y = 18
DISCNT = Ø7: INH ← Inhalt ENDADH, Index Y = 1C
DISCNT = Ø8: INH ← Inhalt ENDADL, Index Y = 2Ø

28

Subroutine TDISP ist in Bild 20 dargestellt. Diese Subroutine läßt den Namen (Di1 . . . Di4) und den Inhalt (Di5, Di6) des von Parameter-Zähler DISCNT bestimmten Speicherplatzes auf den Siebensegment-Displays erscheinen. Ferner überwacht Subroutine TDISP die Tastatur; sie stellt fest, ob eine weitere Taste gedrückt wird.
Die Programmschleife um das Label TDISP ist mit Subroutine CONVD des Standard-Monitors vergleichbar; siehe Buch 2, Kapitel 7, Bild 14. Zuerst wird der Akku mit Daten geladen, die aus der Look-Up-Tabelle TLOOK stammen. Der vor Aufruf von TDISP in das Y-Register geladene Index (siehe Punkt 27) bestimmt die Auswahl der Daten aus Tabelle TLOOK. Das in den Akku geladene Byte dient als Segment-Muster für die Anzeige auf dem betreffenden Siebensegment-Display. Aus dem PM/TM-Listing am Schluß dieses Buchs (Anhang 3) sowie aus Bild 15 geht der Zusammenhang zwischen den hexadezimalen Daten und den Segment-Mustern hervor.
Nach Umladen des Segment-Muster-Index in den Akku (TYA) wird das Y-Register in einer Verzögerungsschleife verwendet. Während der Verzögerungsdauer leuchtet eins der Displays Di1, Di2, Di3 oder Di4 auf. Anschließend wandert der Index zurück in das Y-Register (TAY); er wird dort um eins erhöht (INY), um das Byte des folgenden Segment-Musters aus der Look-Up-Tabelle zu lesen. Inzwischen ist auch das X-Register auf die Aktivierung des nächsten Displays vorbereitet, es sei denn, daß als nächstes das Display Di5 an der Reihe ist (CPX # 1Ø). Dann nämlich folgt ein Sprung zu einer anderen Subroutine: zur Subroutine LDAINH mit der Startadresse $1DA7.
Schlagen wir das Listing des Standard-Monitors in Buch 2 (Anhang 2) auf, so finden wir die Adresse $1DA7 auf Seite 218 innerhalb der Subroutine

20

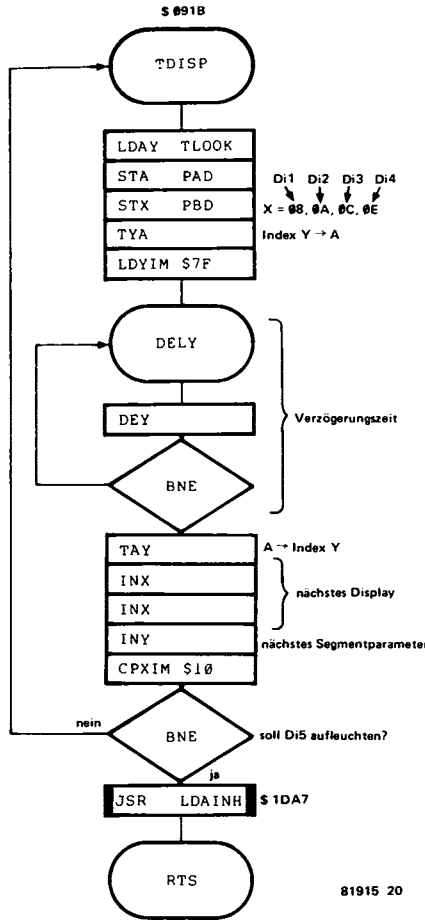


Bild 20. Dafür, daß der Name eines Parameters und der Inhalt des zugehörigen Speicherplatzes auf den Displays erscheint, sorgt Subroutine TDISP. Ferner prüft diese Subroutine, ob eine Taste gedrückt ist.

SCAND/SCANDS wieder. Subroutine LDAINH ist nämlich mit dem Teil von Subroutine SCAND/SCANDS des Standard-Monitors identisch, der den Inhalt von Datenpuffer INH auf den Displays Di5 und Di6 anzeigt. Bild 21 verdeutlicht die einzelnen Phasen von Subroutine SCAND/SCANDS: der Einstieg dort schließt genau passend an den Ausstieg aus Subroutine TDISP an. Während letztere den Parameter-Namen auf die Displays Di1 . . . Di4 setzt, macht Subroutine LDAINH die zugehörigen Daten auf den Displays Di5 und Di6 sichtbar.
Nach der Rückkehr aus SCAND/SCANDS gibt der Inhalt des Akkus

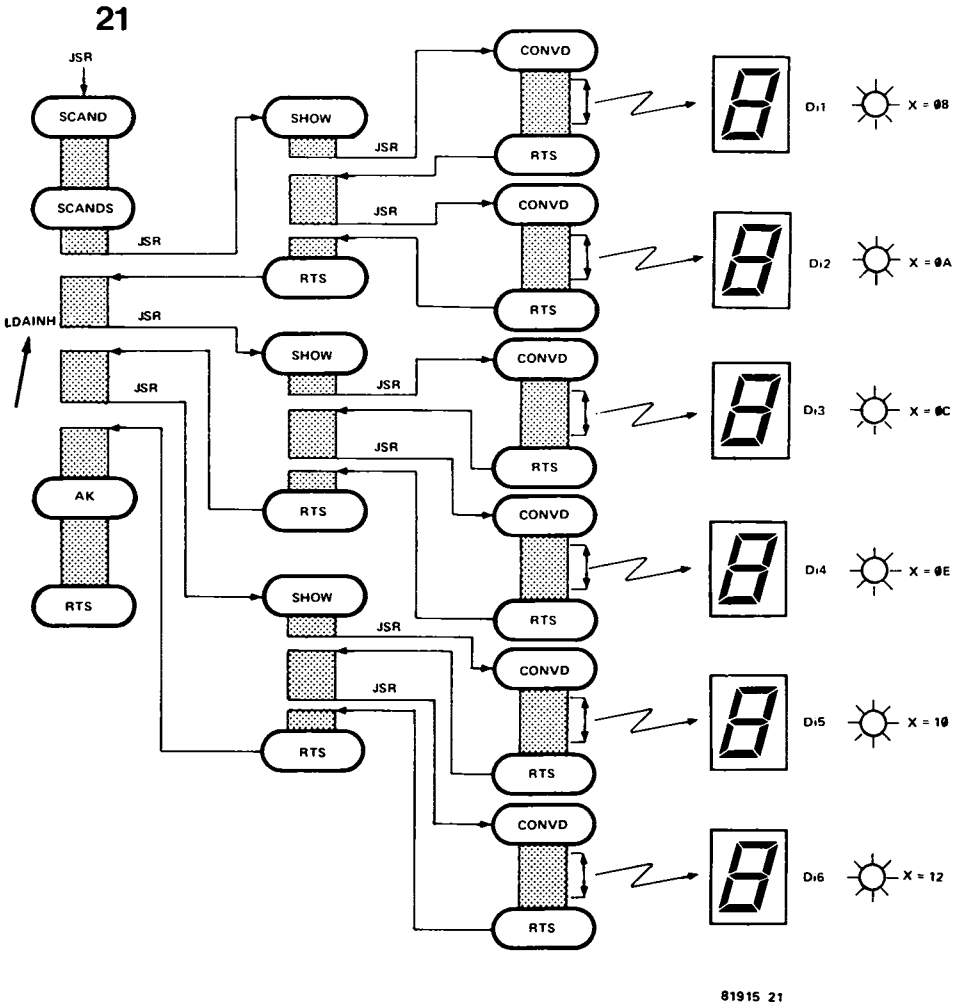


Bild 21. Die innerhalb von TDISP aufgerufene Subroutine LDAINH stimmt mit dem letzten Teil der zum Standard-Monitor gehörenden Subroutine SCAND/SCANDS überein.

darüber Auskunft, ob inzwischen eine Taste gedrückt wurde: dafür sorgt der letzte, beim Label AK beginnende Teil der Subroutine (siehe Bild 21).

Wir sind am Schluß unserer Besprechung von Systemprogramm TM angekommen. Bevor wir dieses Kapitel endgültig beschließen, wollen wir uns noch einmal mit dem "Ohr" des Junior-Computers beschäftigen: Stichwort PLL.

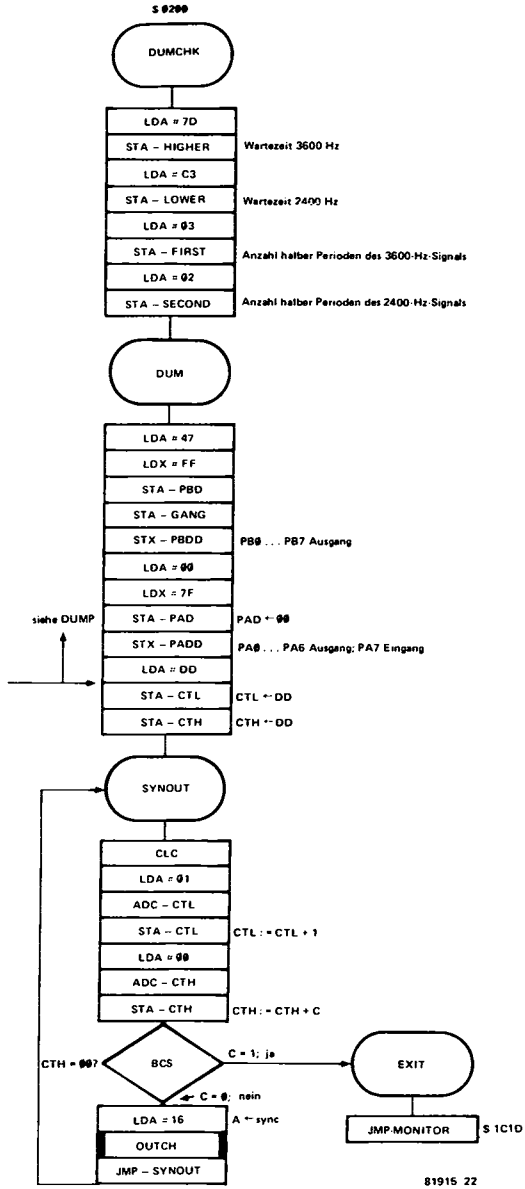


Bild 22. Das Schreib-Hilfsprogramm DUMCHK setzt für die Dauer von etwa 90 Sekunden fortlaufend Synchronisationszeichen auf Band.

29 In Buch 3, Kapitel 11 wurden zwei Verfahren beschrieben, nach denen man die PLL-Schaltung ordnungsgemäß einstellen kann. Zu diesem Zweck sind dort die Hexdumps von **drei Testprogrammen** angegeben. Für die Einstellung mit Hilfe der Siebensegment-Displays ist sowohl ein Schreib- als auch ein Leseprogramm erforderlich; stellt man die PLL-Schaltung mit einem Oszilloskop ein, so wird nur ein Schreibprogramm benötigt. Einzelheiten, die die Praxis betreffen, sind im genannten Kapitel (Buch 3, Seite 104 . . . 113) nachzulesen. An dieser Stelle wollen wir die Arbeitsweise der drei Testprogramme aus der Nähe betrachten.

Damit die Siebensegment-Displays signalisieren können, ob Poti P1 auf der Interface-Karte richtig oder falsch eingestellt ist, werden zuerst für die Dauer von etwa 90 Sekunden Synchronisationszeichen auf Band gesetzt. Das ist die Aufgabe des Programms DUMCHK in Bild 22, dessen Listing bereits in Kapitel 11 (erste Hälfte von Tabelle 2 in Buch 3, Seite 106) abgedruckt wurde. Das Programm DUMCHK hat die Startadresse Ø2ØØ.

Der erste Teil von Hilfsprogramm DUMCHK (siehe Flußdiagramm in Bild 22) ist identisch mit dem ersten Teil der Subroutine DUMP/DUMPT in Bild 1a. Erst zwei Instruktionen vor dem Label SYNOUT sind Unterschiede feststellbar: In die beiden Speicherplätze CTH und CTL wird der Wert $DD gesetzt. Beim Label SYNOUT beginnt eine Zählschleife. Jeder Durchlauf durch diese Schleife erhöht die in CTH (linkes Byte) und CTL (rechtes Byte) stehende 16-bit-Zahl um eins und schreibt, sofern nicht bei Verzweigung BCS die Carry-Flag gesetzt ist, mit Subroutine OUTCH ein Synchronisationszeichen ($16) auf das Band. Ist die Carry-Flag bei Erreichen des BCS gesetzt, so folgt ein Sprung über Label EXIT zum Standard-Monitor. Auf den Siebensegment-Displays erscheint wieder die Startadresse Ø2ØØ.

Verzweigung BCS steht auf Sprung, sobald die in den Speicherplätzen CTH/CTL enthaltene 16-bit-Zahl den Wert $FFFF überschreitet. Da dort vor dem ersten Schleifendurchlauf die Zahl $DDDD stand, wird die Schleife insgesamt 8738 mal vollständig durchlaufen. Es werden somit 8738 Synchronisationszeichen auf Band geschrieben. Jedes einzelne Synchronisationszeichen dauert 1250 μs (siehe Punkt 3 dieses Kapitels), so daß bis zum Sprung in Richtung Standard-Monitor etwa 87 Sekunden vergehen.

Das zugehörige Leseprogramm trägt den Namen RDCHK, es hat die Startadresse Ø251. Das Flußdiagramm von RDCHK ist in Bild 23 zu finden, während das Hexdump bereits in Kapitel 11 (Buch 3, Seite 106, zweiter Teil von Tabelle 2) abgedruckt war. Der Teil des Leseprogramms RDCHK, der zwischen den Labeln RDCHK und SYB steht, stimmt mit dem ersten Teil der Subroutine RDTAPE (Bild 6a, Label RDTAPE bis TENSYN) überein. Nach dem Label SYB wird von Subroutine RDCH ein Zeichen vom Band gelesen und in den Akku gesetzt. Dann wird geprüft, ob dieses Zeichen ein Synchronisationszeichen ist. Abhängig vom Ergebnis folgt entweder ein Sprung zum Label SYB oder zum Label SYNCHK.

Die Programme DUMCHK und RDCHK in Bild 23 bzw. 24 gehören zu dem PLL-Einstellverfahren, bei dem das Siebensegment-Display anzeigt, ob der Junior-Computer die auf dem Band stehenden Zeichen richtig oder falsch liest. Was während des Lesevorgangs auf den Displays Di5 und Di6

23

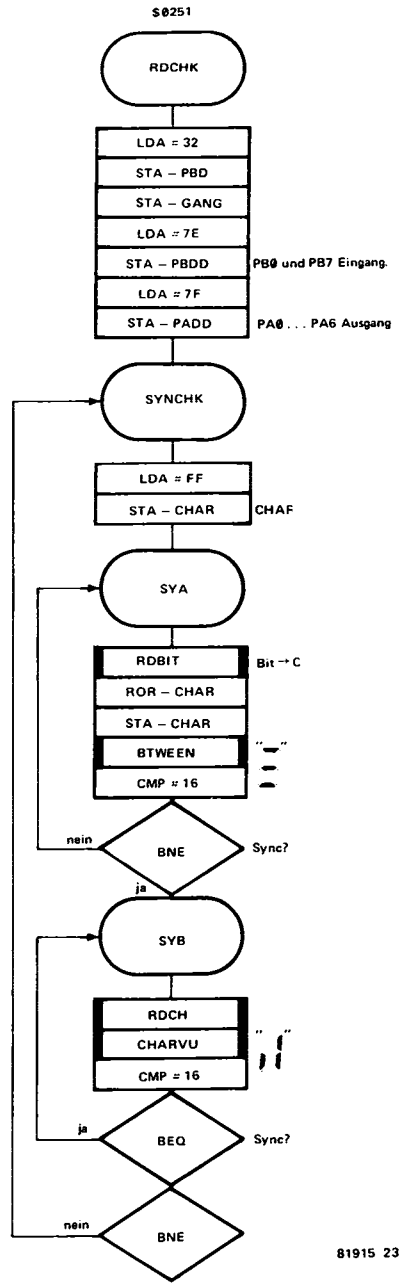


Bild 23. Mit Lese-Hilfsprogramm RDCHK und den von DUMCHK (Bild 22) auf Band geschriebenen Synchronisationszeichen läßt sich das PLL-System ohne weitere Hilfsmittel richtig einstellen.

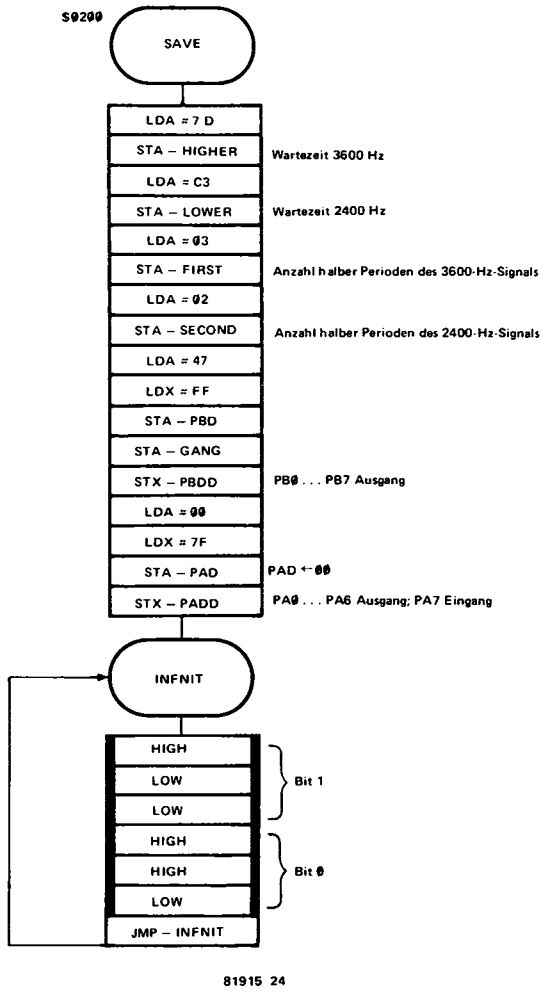


Bild 24. Schreib-Hilfsprogramm SAVE, das ununterbrochen logische Nullen und Einsen auf das Band schreibt, ermöglicht die PLL-Einstellung mit dem Oszilloskop.

sichtbar ist, bestimmen zwei Instruktionen des Leseprogramms RDCHK in Bild 23: Nach dem Aufruf von Subroutine BTWEEN erscheint das "Suchmuster"; die drei waagerechten Segmente leuchten auf. Dagegen dient der bis auf Bit b7 invertierte ASCII-Kode des gerade eingelesenen Zeichens zur Steuerung der Segmente, sobald Subroutine CHARVU aufgerufen wird. Wenn der Computer Synchronisationszeichen vom Band liest, was natürlich die richtige Einstellung des PLL-Systems voraussetzt,

wird die Schleife um das Label SYB durchlaufen; auf den Displays ist dann das "Synchronisationsmuster" sichtbar.
Weitere Möglichkeiten sind das Lesen anderer Zeichen, weil diese auf dem Band stehen, oder das fehlerhafte Lesen infolge falsch eingestelltem PLL. In beiden Fällen dient nach dem Aufruf von Subroutine CHARVU ein willkürlicher und deshalb wahrscheinlich falscher Akku-Inhalt als Segmentmuster. Für das Auge ist dies jedoch nicht erkennbar, weil einige Mikrosekunden später der Rücksprung zum Label SYNCHK folgt und die Subroutine BTWEEN das "Suchmuster" auf den Displays erscheinen läßt. Der gerade beschriebene Fall kann übrigens nur dann auftreten, wenn irgendwann zuvor mindestens ein Synchronisationszeichen fehlerfrei vom Band gelesen wurde. Bis dahin wird nämlich die Warteschleife um das Label SYA nicht verlassen.

30 Das Programm SAVE in Bild 24 wird zum Einstellen des PLL-Systems mit Hilfe eines Oszilloskops benötigt. Bereits in Buch 3 (Seite 107, Tabelle 3) war das Hexdump dieses Programms abgedruckt.
Programm SAVE setzt ein Testsignal auf Band, das aus einer beliebig langen Reihe von logischen Nullen und Einsen besteht. Dabei folgt stets auf eine Null eine Eins und umgekehrt; Nullen und Einsen wechseln einander ab. Der erste Teil des Programms vom Label SAVE bis zum Label INFINIT ist identisch mit dem ersten Teil von Subroutine DUMP/DUMPT. Beim Label INFINIT beginnt eine unendlich lange Schleife; hier werden in der notwendigen Reihenfolge die Subroutinen HIGH und LOW aufgerufen. Da die Schleife in sich geschlossen ist, wird so lange abwechselnd eine logische Null und eine logische Eins auf Band gesetzt, bis ein Druck auf die RST-Taste die Schleife unterbricht. Erst dann ist das Schreiben des Testsignals auf das Band beendet.

Beendet ist hiermit auch das 16. Kapitel der Junior-Computer-Buchreihe. Zwar mag die Fülle der "Super-Subroutinen", Subroutinen und "Sub-Subroutinen" zunächst verwirrend erscheinen, wir hoffen jedoch, daß dieses Kapitel dazu beitrug, einen gangbaren Pfad durch das Subroutinen-Dickicht zu schlagen.

Listing des Systemprogramms PME

Auf den folgenden Seiten ist das Listing des Systemprogramms PME abgedruckt. Die Instruktionen der Subroutinen BINAR und PMBINA, mit denen die (zeitweise) Rückkehr zum binären Rechenmodus möglich ist, sind auf den Seiten 191/192 zu finden. Über weitere Einzelheiten informiert Anhang 4 in diesem Buch.

Einige wichtige PME-Adressen:

EDITC	$1500
BRK	$1533
EDITW	$153D
SEMIW	$1667
SEACND	$17C5
BINAR	$17F6
PMBINA	$17FA

```
0001: 14F8                    ORG   $14F8
0002:
0003:
0004:                 SOURCE LISTING OF THE PM EDITOR
0005:
0006:
0007:                 WRITTEN BY G. NACHBAR
0008:
0009:                 DATE : 25 JUNE 1981
0010:
0011:
0012:                 THE PM EDITOR USES HEXADECIMAL LABELS
0013:
0014:                 *********************************
0015:                 POINTERS AND TEMPS IN PAGE ZERO
0016:                 *********************************
0017:
0018: 14F8            BEGADL *      $00E2
0019: 14F8            BEGADH *      $00E3
0020: 14F8            ENDADL *      $00E4
0021: 14F8            ENDADH *      $00E5
0022: 14F8            CURADL *      $00E6
0023: 14F8            CURADH *      $00E7
0024: 14F8            CENDL  *      $00E8
0025: 14F8            CENDH  *      $00E9
0026: 14F8            TABLEL *      $00EC
0027: 14F8            TABLEH *      $00ED
0028: 14F8            LABELS *      $00EE
0029: 14F8            BYTES  *      $00F6
0030: 14F8            COUNT  *      $00F7
0031: 14F8            INH    *      $00F9
0032: 14F8            POINTL *      $00FA
0033: 14F8            POINTH *      $00FB
0034: 14F8            TEMPX  *      $00FD
0035: 14F8            NIBBLE *      $00FE
0036:
0037:                 ************************************
0038:                 PME'S POINTERS AND TEMPS IN PAGE 1A
0039:                 ************************************
0040:
0041: 14F8            PARAL  *      $1A63
0042: 14F8            PARAH  *      $1A64
0043: 14F8            PARBL  *      $1A65
0044: 14F8            PARBH  *      $1A66
0045: 14F8            NMIL   *      $1A7A
0046: 14F8            NMIH   *      $1A7B
0047: 14F8            BRKTL  *      $1A7C
0048: 14F8            BRKTH  *      $1A7D
0049: 14F8            PBDD   *      $1A83
0050:
0051:
0052:                 *********************************
0053:                 ADDRESSES IN THE STANDARD EPROM
0054:                 *********************************
0055:
0056:
0057: 14F8            SAVE   *      $1C00
0058: 14F8            BEGIN  *      $1ED3
0059: 14F8            OPLEN  *      $1E5C
0060: 14F8            LENACC *      $1E60
0061: 14F8            ADCEND *      $1EDC
0062: 14F8            RECEND *      $1EEA
0063: 14F8            UP     *      $1E83
0064: 14F8            FILLWS *      $1E47
0065: 14F8            NEXT   *      $1EF8
0066: 14F8            ASSEMB *      $1F51
0067:
0068:
```

```
0069:                   ******************
0070:                   SUBROUTINES IN PM
0071:                   ******************
0072:
0073:
0074: 14F8              INPAR   *       $1387
0075: 14F8              RECCHA  *       $12AE
0076: 14F8              PRCHA   *       $1334
0077: 14F8              PRBYT   *       $128F
0078: 14F8              PRSP    *       $11F3
0079: 14F8              ASHETT  *       $141E
0080: 14F8              CRLF    *       $11E8
0081: 14F8              RESIN   *       $1268
0082: 14F8              RESTTY  *       $14BC
0083:
0084: 14F8              STEP    *       $14CF
0085:
0086:
0087:
0088:                   IOCORR: ADAPTS THE PBDD TO THE VERSION D SITUATION,
0089:                   PRIOR TO ASSEMBLING
0090:
0091: 14F8 A9 1E        IOCORR LDAIM $1E
0092: 14FA 8D 83 1A            STA   PBDD    PB0 IS INPUT
0093: 14FD 4C 51 1F            JMP   ASSEMB ASSEMBLE THE PRCGRAM
0094:
0095:                   EDITC: COLD START ENTRY OF PME
0096:
0097: 1500 20 68 12    EDITC   JSR   RESIN   RESET INPUT BUFFERS
0098: 1503 A0 00               LDYIM $00
0099: 1505 20 3E 17            JSR   MESSA   "BEGAD,ENDAD:"
0100: 1508 20 87 13            JSR   INPAR   GET PARAMETERS
0101: 150B 30 F3               BMI   EDITC   TRY IT AGAIN, IF NOT DONE PROPERLY
0102: 150D AD 63 1A            LDA   PARAL   LOAD BEGADL
0103: 1510 85 E2               STAZ  BEGADL
0104: 1512 18                  CLC
0105: 1513 69 01               ADCIM $01
0106: 1515 85 E8               STAZ  CENDL   CENDL = BEGADL+1
0107: 1517 AD 64 1A            LDA   PARAH
0108: 151A 85 E3               STAZ  BEGADH LOAD BEGADH
0109: 151C 69 00               ADCIM $00
0110: 151E 85 E9               STAZ  CENDH   CENDH = BEGADH+AARRY
0111: 1520 AD 65 1A            LDA   PARBL
0112: 1523 85 E4               STAZ  ENDADL LOAD ENDAD
0113: 1525 AD 66 1A            LDA   PARBH
0114: 1528 85 E5               STAZ  ENDADH LOAD ENDADH
0115: 152A 20 D3 1E            JSR   BEGIN   PRINT POINTER CURAD
0116: 152D A9 77               LDAIM $77     EQUALS BEGAD
0117: 152F A0 00               LDYIM $00
0118: 1531 91 E6               STAIY CURADL EQF 77 ON FIRST ADDRESS BEGAD
0119:
0120:                   BRK: WARM START ENTRY OF PME, INCLUDING THE SPECIFI-
0121:                   CATION OF THE BRK JUMP VECTOR
0122:
0123: 1533 A9 3D       BRK     LDAIM $3D
0124: 1535 8D 7C 1A            STA   BRKTL
0125: 1538 A9 15               LDAIM $15
0126: 153A 8D 7D 1A            STA   BRKTH
0127:
0128:                   EDITW: MAY BE USED AS A WARM START ENTRY OF PME
0129:                   PROVIDED THE BRK JUMP VECTOR HAS BEEN SPECIFIED
0130:                   BEFOREHAND.
0131: 153D A0 20        EDITW  LDYIM $20
0132: 153F 20 3E 17            JSR   MESSA   "PM EDITOR"
0133: 1542 A0 FF               LDYIM $FF
0134: 1544 9A                  TXS           RESET STACK POINTER
0135:
0136:                   THE CENTRAL LABEL "WARM" OF PME STARTS STARTS
0137:                   WITH THE K-KEY ROUTINE
```

```
0138:                   ("DELETE")
0139:
0140: 1545 20 0B 17  WARM    JSR    PRINS  PRINT CURRENT INSTRUCTION
0141: 1548 20 AE 12          JSR    RECCHA WAIT FOR A DEPRESSED KEY
0142: 154B C9 4B             CMPIM 'K      IS IT THE K-KEY?
0143: 154D D0 09             BNE    LIST   CONTINUE IF NOT
0144: 154F 20 83 1E          JSR    UP     ADAPT WORKSPACE
0145: 1552 20 EA 1E          JSR    RECEND ADAPT CURRENT END ADDRESS POINTER
0146: 1555 4C 45 15          JMP    WARM   READY
0147:                   LABEL LIST: L-KEY ROUTINE
0148:
0149: 1558 C9 4C       LIST  CMPIM 'L      L-KEY DEPRESSED?
0150: 155A D0 12             BNE    SKIP   IF NOT CONTINUE
0151: 155C 20 D3 1E          JSR    BEGIN  START AT BEGAD
0152: 155F 20 0B 17  LST     JSR    PRINS  PRINT CURRENT INSTRUCTION
0153: 1562 20 F8 1E          JSR    NEXT   ADAPT INSTRUCTION POINTER
0154: 1565 30 F8             BMI    LST
0155: 1567 A0 1F       DONE  LDYIM $1F
0156: 1569 20 3E 17    MESS  JSR    MESSA  PRINT "DONE"
0157: 156C F0 D7             BEQ    WARM   READY
0158:
0159:                   LABEL SKIP: SPACE BAR KEY ROUTINE
0160:                   LABEL INSERT: I-KEY ROUTINE
0161:
0162: 156E C9 20       SKIP  CMPIM '        SPACE BAR DEPRESSED?
0163: 1570 D0 07             BNE    INSERT CONTINUE IF NOT
0164: 1572 20 F8 1E          JSR    NEXT.  ADAPT INSTRUCTION POINTER
0165: 1575 30 CE             BMI    WARM   READY
0166: 1577 10 EE             BPL    DONE   PRINT "DONE" IF NO FURTHER INSTR. AVAILABLE
0167: 1579 C9 49       INSERT CMPIM 'I      I-KEY DEPRESSED?
0168: 157B D0 17             BNE    SEARCH IF NOT CONTINUE
0169: 157D 20 C9 16          JSR    READIN READ NEW INSTRUCTION
0170: 1580 30 0A             BMI    ILLKEY PRINT "ILLEGAL KEY" IF NEW ISTR. ISN'T VALID
0171: 1582 20 F6 16    MEM   JSR    CHECK
0172: 1585 90 09             BCC    FULL   PRINT "FULL" IF MEMORY IS FULL
0173: 1587 20 47 1E          JSR    FILLWS LOAD NEW INSTR. IN MEMORY
0174: 158A F0 B9             BEQ    WARM   READY
0175: 158C A0 0E       ILLKEY LDYIM $0E
0176: 158E D0 D9             BNE    MESS   PRINT "ILLEGAL KEY"
0177: 1590 A0 1A       FULL  LDYIM $1A
0178: 1592 D0 D5             BNE    MESS   PRINT "FULL"
0179:
0180:                   LABEL SEARCH: S-KEY ROUTINE
0181:                   ANY BYTE PATTERN (INSTRUCTION) MAY BE SEARCHED FOR.
0182:                   IF THE INSTRUCTION SEARCHED FOR IS FOUND,
0183:                   ONE MAY SEARCH FOR THE SAME INSTRUCTION ON A HIGHER
0184:                   ADDRESS, BY PRESSING THE KEY "Y". THE SEARCHING PRO-
0185:                   CESS ALWAYS HAS TO BE CONCLUDED BY THE MESSAGE "DONE".
0186:                   THIS OCCURS EITHER BECAUSE ALL OR NO INSTRUCTIONS ARE
0187:                   FOUND, OR BY PRESSING AN ARBITRARY SOFTWARE-KEY (I.E.
0188:                   A KEY GENERATING AN ASCII CODE WHEN DEPRESSED),
0189:                   WITH THE EXEPTION OF THE 'Y'-KEY.
0190: 1594 C9 53       SEARCH CMPIM 'S      S-KEY DEPRESSED?
0191: 1596 D0 40             BNE    BACK   IF NOT CONTINUE
0192: 1598 20 C9 16          JSR    READIN READ INSTR. TO BE SEARCHED FOR
0193: 159B 30 EF             BMI    ILLKEY "ILLEGAL KEY" IF NOT PROPERLY DONE
0194: 159D 20 D3 1E          JSR    BEGIN  START AT BEGAD
0195: 15A0 A5 FB       SCAN  LDAZ   POINTH GET OPCODE OF SEARCH INSTR.
0196: 15A2 20 60 1E          JSR    LENACC GET LENGTH OF SEARCH INSTR.
0197: 15A5 A0 00             LDYIM $00
0198: 15A7 B1 E6             LDAIY CURADL GET OPCODE OF THE CURRENT INSTR.
0199: 15A9 C5 FB             CMPZ   POINTH COMPARE IT AGAINST OPCODE OF THE SEARCH INST.
0200: 15AB D0 20             BNE    AGAIN  IF NO MATCH THEN NEXT INSTR.
0201: 15AD C6 F6             DECZ   BYTES
0202: 15AF F0 12             BEQ    FOUND  CONTINUE, IF INSTR. LENGTH IS 2 OR 3
0203: 15B1 C8                INY
0204: 15B2 B1 E6             LDAIY CURADL GET NEXT BYTE IN MEMORY
0205: 15B4 C5 FA             CMPZ   POINTL COMP. IT AGAINST 1ST OPERAND BYTE
0206: 15B6 D0 15             BNE    AGAIN  IF NO MATCH NEXT INSTR.
```

```
0207: 15B8 C6 F6            DECZ  BYTES
0208: 15BA F0 07            BEQ   FOUND  CONTINUE IF LENGTH OF CURR. INSTR. IS 3
0209: 15BC C8               INY
0210: 15BD B1 E6            LDAIY CURADL GET NEXT BYTE IN MEMORY
0211: 15BF C5 F9            CMPZ  INH    COMP. IT AGAINST 2ND OPERAND BYTE
0212: 15C1 D0 0A            BNE   AGAIN  IF NO MATCH NEXT INSTR.
0213: 15C3 20 0B 17  FOUND  JSR   PRINS  PRINT OUT THE SPECIFIED INSTRUCTION
0214: 15C6 20 AE 12         JSR   RECCHA WAIT FOR A DEPRESSED KEY
0215: 15C9 C9 59            CMPIM 'Y     Y-KEY DEPRESSED?
0216: 15CB D0 08            BNE   DNE    IF NOT PRINT "DONE"
0217: 15CD 20 5C 1E  AGAIN  JSR   OPLEN  GET LENGTH OF THE CURR. INSTR.
0218: 15D0 20 F8 1E         JSR   NEXT   ADAPT INSTR. POINTER
0219: 15D3 30 CB            BMI   SCAN   CONTINUE SEARCH IF STILL INSTR. AVAILABLE
0220: 15D5 4C 67 15  DNE    JMP   DONE   ELSE PRINT "DONE"
0221:
0222:                       LABEL BACK: Z-KEY ROUTINE
0223:
0224: 15D8 C9 5A     BACK   CMPIM 'Z     Z-KEY DEPRESSED?
0225: 15DA D0 30            BNE   TOF    IF NOT CONTINUE
0226: 15DC A5 E2            LDAZ  BEGADL
0227: 15DE 85 EC            STAZ  TABLEL
0228: 15E0 A5 E3            LDAZ  BEGADH
0229: 15E2 85 ED            STAZ  TABLEH TABLE:=BEGAD
0230: 15E4 C5 E7            CMPZ  CURADH  TABLEH:=CURADH?
0231: 15E6 D0 08            BNE   BCKA   IF NOT INCREASE TABLE
0232: 15E8 A5 E2            LDAZ  BEGADL
0233: 15EA C5 E6            CMPZ  CURADL TABLEL:=CURADL?
0234: 15EC D0 02            BNE   BCKA   IF NOT INCREASE TABLE
0235: 15EE F0 19            BEQ   BCKB   ELSE PRINT FIRST INSTRUCTION
0236: 15F0 A0 00     BCKA   LDYIM $00
0237: 15F2 B1 EC            LDAIY TABLEL GET LENGTH OF INSTR. AT
0238: 15F4 20 60 1E         JSR   LENACC WHICH POINT IS POINTED
0239: 15F7 20 A0 16         JSR   INCTAB INCREASE POINT BY BYTES
0240: 15FA A5 E6            LDAZ  CURADL
0241: 15FC C5 EC            CMPZ  TABLEL TABLEL:=CURADL?
0242: 15FE D0 F0            BNE   BCKA   IF NOT INCREASE TABLE
0243: 1600 A5 E7            LDAZ  CURADH
0244: 1602 C5 ED            CMPZ  TABLEH TABLEH:=CURADH
0245: 1604 D0 EA            BNE   BCKA   IF NOT INCREASE TABLE
0246: 1606 20 92 16         JSR   DECURA DECREASE INSTRUCTION POINTER BY BYTES
0247: 1609 4C 45 15  BCKB   JMP   WARM
0248:
0249:                       LABEL TOF: T-KEY ROUTINE
0250:                       LABEL SXTEEN: P-KEY ROUTINE
0251:
0252: 160C C9 54     TOF    CMPIM 'T     T-KEY DEPRESSED?
0253: 160E D0 06            BNE   SXTEEN IF NOT CONTINUE
0254: 1610 20 D3 1E         JSR   BEGIN  INSTR. POINTER = BEGAD
0255: 1613 4C 45 15  TOFEND JMP   WARM   READY
0256: 1616 C9 50     SXTEEN CMPIM 'P     P-KEY DEPRESSED?
0257: 1618 D0 1C            BNE   ASMBLR IF NOT CONTINUE
0258: 161A A9 0F            LDAIM $0F    15 INSTR. TO BE PRINTED
0259: 161C 85 F7            STAZ  COUNT
0260: 161E 20 5C 1E  LINES  JSR   OPLEN  GET LENGTH OF CURRENT INSTR.
0261: 1621 20 F8 1E         JSR   NEXT   ADAPT INSTR. POINTER
0262: 1624 C6 F7            DECZ  COUNT
0263: 1626 F0 EB            BEQ   TOFEND READY IF 15 INSTR. HAVE BEEN PRINTED
0264: 1628 A0 00            LDYIM $00
0265: 162A B1 E6            LDAIY CURADL
0266: 162C C9 77            CMPIM $77
0267: 162E F0 E3            BEQ   TOFEND READY IF OPCODE IS 77
0268: 1630 20 0B 17         JSR   PRINS  PRINT INSTRUCTION
0269: 1633 4C 1E 16         JMP   LINES  NEXT INSTRUCTION
0270:                       LABEL ASMBLR: X-KEY ROUTINE
0271:                       LABEL ASSEND: RESTORE I/O AFTER ASSEMBLING
0272:
0273: 1636 C9 58     ASMBLR CMPIM 'X     X-KEY DEPRESSED?
0274: 1638 D0 13            BNE   INPUT  IF NOT CONTINUE
0275: 163A A9 47            LDAIM ASSEND SPECIFY NMI JUMP VECTOR
```

```
0276: 163C 8D 7A 1A          STA   NMIL
0277: 163F A9 16             LDAIM ASSEND /256
0278: 1641 8D 7B 1A          STA   NMIH
0279: 1644 4C F8 14          JMP   IOCORR PREPARE I/O PRIOR TO ASSEMBLING
0280: 1647 20 BC 14  ASSEND  JSR   RESTTY RESTORE I/O: FM SITUATION
0281: 164A 4C 88 17          JMP   LABLST LIST THE HEXADECIMAL LABELS
0282:
0283:                        LABEL INPUT: I-KEY ROUTINE
0284:                        NOTE: NO NON-NUMERICAL KEY HAS TO BE DEPRESSED
0285:                        PME AUTOMATICALLY ASSUMES THE INPUT KEY FUNCTION
0286:                        HAS BEEN OPTED FOR AS SOON AS THE DATA BELONGING
0287:                        TO THE NEW INSTR. HAS BEEN SPECIFIED
0288:
0289: 164D 20 B1 16  INPUT   JSR   BYT    GET THE 2ND NIBBLE OF THE 1ST BYTE
0290: 1650 30 12             BMI   WRONG
0291: 1652 20 CE 16          JSR   READ   GET THE OTHER BYTE(S)
0292: 1655 30 0D             BMI   WRONG
0293: 1657 20 5C 1E          JSR   OPLEN  GET LENGTH OF THE CURRRENT INSTR.
0294: 165A 20 F8 1E          JSR   NEXT   ADAPT INSTR. POINTER
0295: 165D A5 FD             LDAZ  TEMPX  GET LENGTH OF NEW INSTR.
0296: 165F 85 F6             STAZ  BYTES  AND STORE IT IN BYTES
0297: 1661 4C 82 15          JMP   MEM    LOAD MEMORY WITH NEW INSTRUCTION
0298: 1664 4C 8C 15  WRONG   JMP   ILLKEY PRINT "ILLEGAL KEY"
0299:
0300:                        END OF THE PME MAIN ROUTINE
0301:
0302:
0303:
0304:                        SEMI WARM START ("LUKE WARM") ENTRY OF PME
0305:                        FOLLOWING THE SPECIFICATION BY THE USER
0306:                        OF BEGAD AND ENDAD THE INSTRUCTION POINTER
0307:                        CURAD IS POINTED AT BEGAD AND THE CURRENT
0308:                        END ADDRESS POINTER CEND IS POINTED AT ENDAD.
0309:                        AFTER THIS PME IS ENTERED BY THE WARM START ENTRY
0310:
0311:
0312: 1667 20 68 12  SEMIW   JSR   RESIN  RESET INPUT BUFFERS
0313: 166A A0 00             LDYIM $00
0314: 166C 20 3E 17          JSR   MESSA  "BEGAD,ENDAD:"
0315: 166F 20 87 13          JSR   INPAR  GET BEGAD AND ENDAD
0316: 1672 30 F3             BMI   SEMIW  TRY IT AGAIN IF NOT DONE PROPERLY
0317: 1674 AD 63 1A          LDA   PARAL
0318: 1677 85 E2             STAZ  BEGADL
0319: 1679 AD 64 1A          LDA   PARAH
0320: 167C 85 E3             STAZ  BEGADH
0321: 167E AD 65 1A          LDA   PARBL
0322: 1681 85 E4             STAZ  ENDADL
0323: 1683 85 E8             STAZ  CENDL
0324: 1685 AD 66 1A          LDA   PARBH
0325: 1688 85 E5             STAZ  ENDADH
0326: 168A 85 E9             STAZ  CENDH  CEND = ENDAD
0327: 168C 20 D3 1E          JSR   BEGIN  CURAD = BEGAD
0328: 168F 4C 33 15          JMP   BRK    JUMP TO WARM START
0329:
0330:
0331:
0332:                        ******************
0333:                        SUBROUTINES OF PME
0334:                        ******************
0335:
0336:
0337:                        DECURA
0338:                        THE INSTRUCTION POINTER IS DECREASED BY THE
0339:                        NUMBER OF BYTES - BEING THE LENGTH OF THE
0340:                        INSTRUCTION - WHICH PRECEEDS THE CURRENT
0341:                        INSTRUCTION IN MEMORY.
0342:
0343: 1692 38      DECURA  SEC
0344: 1693 A5 E6           LDAZ  CURADL
```

```
0345: 1695 E5 F6            SBCZ  BYTES
0346: 1697 85 E6            STAZ  CURADL  CURADL:=CURADL-BYTES
0347: 1699 A5 E7            LDAZ  CURADH
0348: 169B E9 00            SBCIM $00
0349: 169D 85 E7            STAZ  CURADH  CURADH:=CURADH-BORROW
0350: 169F 60               RTS
0351:
0352:
0353:                  SUBROUTINR INCTAB
0354:                  INCRESES THE POINTER TABLE BY THE AMOUNT DETERMINED
0355:                  BY THE CONTENTS OF BYTES (INSTRUCTION LENGTH)
0356:
0357: 16A0 18         INCTAB CLC
0358: 16A1 A5 EC            LDAZ  TABLEL
0359: 16A3 65 F6            ADCZ  BYTES
0360: 16A5 85 EC            STAZ  TABLEL  TABLE:=TABLE + BYTES
0361: 16A7 A5 ED            LDAZ  TABLEH
0362: 16A9 69 00            ADCIM $00
0363: 16AB 85 ED            STAZ  TABLEH
0364: 16AD 60               RTS
0365:
0366:                  THE SUBROUTINE BYTIN  LOADS THE ACCU WITH
0367:                  WITH DATA WHICH BELONGS TO TWO NUMERICAL KEYS
0368:                  DEPRESSED. RETURNING FROM SUBROUTINE N=1 AND
0369:                  Z=0 IF A NON-NUMERICAL KEY WAS DEPRESSED. TWO
0370:                  SUCCESIVELY DEPRESSED NUMERICAL KEYS WILL
0371:                  RESULT INTO N=0 AND Z=1
0372:
0373: 16AE 20 AE 12  BYTIN  JSR   RECCHA WAIT FOR A DEPRESSED KEY
0374: 16B1 20 1E 14  BYT    JSR   ASHETT CONVERT IT TO A DATA NIBBLE
0375: 16B4 30 12            BMI   RETURN ERROR EXIT IF KEY <> 0...F
0376: 16B6 0A               ASLA        ENTERED DATA IS NEW HIGHER DATA NIBB
0377: 16B7 0A               ASLA
0378: 16B8 0A               ASLA
0379: 16B9 0A               ASLA
0380: 16BA 85 FE            STAZ  NIBBLE SAVE HIGHER DATA NIBBLE
0381: 16BC 20 AE 12         JSR   RECCHA WAIT FOR A DEPRESSED KEY
0382: 16BF 20 1E 14         JSR   ASHETT CONVERT IT TO A DATA NIBBLE
0383: 16C2 30 04            BMI   RETURN ERROR EXIT IF KEY <> 0...F
0384: 16C4 05 FE            ORAZ  NIBBLE ISERT NEW LOWER DATA NIBBLE
0385: 16C6 A2 00            LDXIM $00     RESET N-FLAG, SET Z-FLAG
0386: 16C8 60         RETURN RTS
0387:
0388:
0389:                  THE SUBROUTINE READIN LOADS EITHER ONE,
0390:                  TWO OR THREE DATA BUFFERS DEPENDING
0391:                  OF THE INSTRUCTION LENGTH SPECIFIED BY
0392:                  THE OPCODE.
0393:                  NORMAL EXIT: Z=1, N=0
0394:                  ERROR EXIT: Z=0, N=1
0395:
0396: 16C9 20 AE 16  READIN JSR   BYTIN  WAIT FOR OPCODE
0397: 16CC 30 27            BMI   RDB    ERROR IF KEY <> 0...F
0398: 16CE 85 FB      READ  STAZ  POINTH STORE OPCODE IN POINTH
0399: 16D0 20 60 1E         JSR   LENACC GET INSTRUCTION LENGTH
0400: 16D3 84 F7            STYZ  COUNT  AND COPY IT INTO COUNT
0401: 16D5 84 FD            STYZ  TEMPX  AS WELL AS IN TEMPX
0402: 16D7 C6 F7            DECZ  COUNT
0403: 16D9 F0 18            BEQ   RDA    RETURN IF ONE BYTE INSTR.
0404: 16DB 20 F3 11         JSR   PRSP   PRINT A SPACE
0405: 16DE 20 AE 16         JSR   BYTIN  WAIT FOR (1ST) OPERAND
0406: 16E1 30 12            BMI   RDB    ERROR IF KEY <> 0...F
0407: 16E3 85 FA            STAZ  POINTL STORE (1ST) OPERND IN POINTL
0408: 16E5 C6 F7            DECZ  COUNT
0409: 16E7 F0 0A            BEQ   RDA    RETURN IF TWO BYTE INSTR.
0410: 16E9 20 F3 11         JSR   PRSP   PRINT A SPACE
0411: 16EC 20 AE 16         JSR   BYTIN  WAIT FOR 2ND OPERAND
0412: 16EF 30 04            BMI   RDB    ERROR IF KEY <> 0...F
0413: 16F1 85 F9            STAZ  INH    STORE 2ND OPERAND IN INH
```

```
0414: 16F3 A2 00      RDA     LDXIM $00     Z=1, N=0
0415: 16F5 60         RDB     RTS
0416:
0417:
0418:                 THE SUBROUTINE CHECK VERIFIES IF THERE IS STILL
0419:                 ROOM IN THE WORKSPACE AREA FOR A NEW INSTRUCTION.
0420:                 THE WORKSPACE IS LIMITED BY BEGAD AND ENDAD.
0421:                 CHECK RETURNS WITH:
0422:                 C=0 IF THERE IS NO ROOM FOR THE INSTR.
0423:                 C=1 IF THERE IS ROOM FOR THE NEW INSTRUCTION
0424:
0425: 16F6 20 DC 1E  CHECK  JSR   ADCEND ADJUST CURRENT ENDADDRESS POINTER
0426: 16F9 38                 SEC
0427: 16FA A5 E4              LDAZ  ENDADL
0428: 16FC E9 02              SBCIM $02
0429: 16FE E5 E8              SBCZ  CENDL
0430: 1700 A5 E5              LDAZ  ENDADH CARRY DETERMINED BY ENDAD-2-CEND
0431: 1702 E5 E9              SBCZ  CENDH
0432: 1704 90 04              BCC   CHKEND EXIT IF NO ROOM ANYMORE
0433: 1706 20 EA 1E           JSR   RECEND READJUST CURRENT END ADDRESS POINTER
0434: 1709 38                 SEC
0435: 170A 60         CHKEND  RTS
0436:
0437:
0438:                 THE SUBROUTINE PRINS PRINTS AN INSTRUCTION
0439:                 SPECIFIED BY THE INSTRUCTION POINTER CURAD.
0440:                 THE CONTENTS OF THE INSTRUCTION POINTER IS ALSO
0441:                 PRINTED. BY PRINTING A CERTAIN NUMBER OF SPACES
0442:                 THE POSITION OF THE CARRIAGE IS IDENTICAL TO THE
0443:                 FIRST POSITION OF THE SYSTEM COMMAND COLUMN.
0444:
0445:
0446: 170B 20 E8 11  PRINS  JSR   CRLF   PRINT A NEW LINE
0447: 170E 20 5C 1E           JSR   OPLEN  GET LENGTH OF INSTRUCTION
0448: 1711 A6 F6              LDXZ  BYTES  TO BE PRINTED
0449: 1713 A5 E7              LDAZ  CURADH
0450: 1715 20 8F 12           JSR   PRBYT  PRINT HIGHER ADDRESS BYTE
0451: 1718 A5 E6              LDAZ  CURADL
0452: 171A 20 8F 12           JSR   PRBYT  PRINT LOWER ADDRESS BYTE
0453: 171D A9 0F              LDAIM $0F
0454: 171F 85 EE              STAZ  LABELS MAXIMUM NUMBER OF SPACES
0455: 1721 A0 00              LDYIM $00
0456: 1723 20 F3 11  PRT      JSR   PRSP   PRINT A SPACE
0457: 1726 B1 E6              LDAIY CURADL GET BYTE TO BE PRINTED
0458: 1728 20 8F 12           JSR   PRBYT  AND PRINT IT
0459: 172B 38                 SEC
0460: 172C A5 EE              LDAZ  LABELS DECREASE NUMBER OF SPACES
0461: 172E E9 03              SBCIM $03    TO BE PRINTED
0462: 1730 85 EE              STAZ  LABELS BY 3
0463: 1732 C8                 INY          SET UP FOR NEXT BYTE
0464: 1733 CA                 DEX          TO BE PRINTED
0465: 1734 D0 ED              BNE   PRT    IF THERE IS ANYONE
0466: 1736 20 F3 11  SP       JSR   PRSP   PRINT A NUMBER OF SPACES
0467: 1739 C6 EE              DECZ  LABELS
0468: 173B D0 F9              BNE   SP
0469: 173D 60                 RTS          RETURN
0470:
0471:
0472:                 THE SUBROUTINE MESSA IS IDENTICAL WITH THE
0473:                 PM SUBROUTINE MESSY, EXEPT FOR A DIFFERENT
0474:                 LOOKUP TABLE
0475:
0476:
0477: 173E 20 E8 11  MESSA  JSR   CRLF   PRINT A NEW LINE
0478: 1741 B9 50 17  MA       LDAY  TXT    GET CHARACTER TO BE PRINTED
0479: 1744 C9 03              CMPIM $03    EOT CHARACTER?
0480: 1746 F0 07              BEQ   TXTEND IF YES RETURN
0481: 1748 20 34 13           JSR   PRCHA  PRINT A CHARACTER
0482: 174B C8                 INY          SET UP FOR NEXT CHARACTER
```

```
0483: 174C 4C 41 17          JMP   MA
0484: 174F 60        TXTEND  RTS           READY
0485:
0486:
0487:                        LOOKUP TABLE TXT
0488:
0489: 1750 42        TXT     =     'B      Y=00
0490: 1751 45                =     'E
0491: 1752 47                =     'G
0492: 1753 41                =     'A
0493: 1754 44                =     'D
0494: 1755 2C                =     ',
0495: 1756 45                =     'E
0496: 1757 4E                =     'N
0497: 1758 44                =     'D
0498: 1759 41                =     'A
0499: 175A 44                =     'D
0500: 175B 3A                =     ':
0501: 175C 20                =     '
0502: 175D 03                =     $03
0503: 175E 49                =     'I      Y=0E
0504: 175F 4C                =     'L
0505: 1760 4C                =     'L
0506: 1761 45                =     'E
0507: 1762 47                =     'G
0508: 1763 41                =     'A
0509: 1764 4C                =     'L
0510: 1765 20                =     '
0511: 1766 4B                =     'K
0512: 1767 45                =     'E
0513: 1768 59                =     'Y
0514: 1769 03                =     $03
0515: 176A 46                =     'F      Y=1A
0516: 176B 55                =     'U
0517: 176C 4C                =     'L
0518: 176D 4C                =     'L
0519: 176E 03                =     $03
0520: 176F 44                =     'D      Y=1F
0521: 1770 4F                =     'O
0522: 1771 4E                =     'N
0523: 1772 45                =     'E
0524: 1773 03                =     $03
0525: 1774 50                =     'P      Y=24
0526: 1775 4D                =     'M
0527: 1776 20                =     '
0528: 1777 45                =     'E
0529: 1778 44                =     'D
0530: 1779 49                =     'I
0531: 177A 54                =     'T
0532: 177B 4F                =     'O
0533: 177C 52                =     'R
0534: 177D 03                =     $03
0535: 177E 4C                =     'L      Y=2E
0536: 177F 41                =     'A
0537: 1780 42                =     'B
0538: 1781 20                =     '
0539: 1782 24                =     '$
0540: 1783 03                =     $03
0541: 1784 3A                =     ':      Y=34
0542: 1785 20                =     '
0543: 1786 24                =     '$
0544: 1787 03                =     $03
0545:
0546:                        THE INSTRUCTIONS FOLLOWING THE LABEL LABLST
0547:                        DEAL WITH THE PRINTING OF ALL HEXADECIMAL LABELS
0548:                        AFTER THE PROGRAM HAS BEEN ASSEMBLED
0549:
0550:
0551: 1788 A0 FF     LABLST LDYIM $FF
```

```
0552: 178A 20 E8 11  LBLSTA JSR   CRLF    PRINT ON A NEW LINE
0553: 178D A2 04             LDXIM $04    4 LABELS ON EACH LINE
0554: 178F 84 FD     LBLSTB STYZ  TEMPX   SAVE Y
0555: 1791 A0 2E             LDYIM $2E    "LAB $"
0556: 1793 20 41 17          JSR   MA
0557: 1796 A4 FD             LDYZ  TEMPX  RESTORE Y
0558: 1798 B1 EC             LDAIY TABLEL GET LABEL NUMBER
0559: 179A 20 8F 12          JSR   PRBYT  PRINT LABEL NUMBER
0560: 179D 88                DEY
0561: 179E 84 FD             STYZ  TEMPX  SAVE Y
0562: 17A0 A0 34             LDYIM $34    ": $"
0563: 17A2 20 41 17          JSR   MA
0564: 17A5 A4 FD             LDYZ  TEMPX  RESTORE Y
0565: 17A7 B1 EC             LDAIY TABLEL GET HIGHER ADDRESS BYTE
0566: 17A9 20 8F 12          JSR   PRBYT  AND PRINT IT
0567: 17AC 88                DEY
0568: 17AD B1 EC             LDAIY TABLEL GET LOWER ADDRESS BYTE
0569: 17AF 20 8F 12          JSR   PRBYT  AND PRINT IT
0570: 17B2 20 F3 11          JSR   PRSP   PRINT A SPACE
0571: 17B5 88                DEY          SETUP FOR NEXT LABEL
0572: 17B6 C4 EE             CPYZ  LABELS ALL LABELS PRINTED
0573: 17B8 F0 05             BEQ   LBLSTC IF NOT SET UP FOR NEXT LABEL
0574: 17BA CA                DEX
0575: 17BB D0 D2             BNE   LBLSTB ON THE SAME LINE
0576: 17BD F0 CB             BEQ   LBLSTA OR ON A NEW LINE
0577: 17BF 20 D3 1E  LBLSTC JSR   BEGIN  INSTRUCTION POINTER = BEGAD
0578: 17C2 4C 30 15          JMP   EDITW  WARM START ENTRY OF PME WITHOUT
0579:                                     BRK JUMP VECTOR SPECIFICATION
0580:
0581:
0582:
0583:                        THE WARM CEND ENTRY (LABEL SEACND) OF PME
0584:                        RESTORES THE POSITION OF THE CURRENT END
0585:                        ADDRESS POINTER CEND, AFTER SPECIFICATION,
0586:                        BY THE USER, OF BEGAD AND ENDAD, AND AFTER FINDING A
0587:                        PSEUDO OPCODE 77, I.E. AN EOF CHARACTER. THIS IS
0588:                        FOLLOWED BY A JUMP TO THE WARM START ENTRY OF PME.
0589:
0590:
0591: 17C5 20 68 12  SEACND JSR   RESIN  RESET INPUT BUFFERS
0592: 17C8 A0 00             LDYIM $00
0593: 17CA 20 3E 17          JSR   MESSA  "BEGAD,ENDAD"
0594: 17CD 20 87 13          JSR   INPAR  GET BEGAD AND ENDAD
0595: 17D0 30 F3             BMI   SEACND TRY IT AGAIN IF NOT PROPERLY DONE
0596: 17D2 20 D3 1E          JSR   BEGIN  INSTR. POINTER=BEGAD
0597: 17D5 A0 00     SCNDA   LDYIM $00
0598: 17D7 B1 E6             LDAIY CURADL GET CURRENT OPCODE
0599: 17D9 C9 77             CMPIM $77    IS IT OPCODE 77?
0600: 17DB F0 09             BEQ   SCNDB  IF NOT CONTINUE
0601: 17DD 20 60 1E          JSR   LENACC GET CURRENT INSTR. LENGTH
0602: 17E0 20 F8 1E          JSR   NEXT   ADJUST INSTR. POINTER
0603: 17E3 4C D5 17          JMP   SCNDA  AND CHECK AGAIN FOR OPCODE 77
0604: 17E6 18        SCNDB   CLC
0605: 17E7 A5 E6             LDAZ  CURADL
0606: 17E9 69 01             ADCIM $01
0607: 17EB 85 E8             STAZ  CENDL  CENDL:=CURADL+1
0608: 17ED A5 E7             LDAZ  CURADH
0609: 17EF 69 00             ADCIM $00
0610: 17F1 85 E9             STAZ  CENDH  CENDH:=CURADH+CARRY
0611: 17F3 4C 33 15          JMP   BRK    WARM START ENTRY OF PME
0612:
0613:
0614:                        THE INSTRUCTIONS FOLLOWING THE LABEL BINAR
0615:                        AND THE INSTRUCTIONS FOLLOWING THE LABEL PMBINA ARE
0616:                        AN EXTENSION OF THE STANDARD MONITOR AND OF PM
0617:                        RESPECTIVELY. IN CASE OF SINGLE STEPPING THROUGH
0618:                        AN USER PROGRAM WITH DECIMAL ARITHMETIC, THERE
0619:                        WILL BE A TEMPORARY SWTCH BACK ON BINARY ARITHMETIC
0620:                        PROVIDED THE NMI JUMP VECTOR HAS BEEN SPECIFIED
```

```
0621:                   ON 17F6 OR 17FA RESPECTIVELY.
0622:
0623:
0624: 17F6 D8           BINAR   CLD             BINARY ARITHMETIC
0625: 17F7 4C 00 1C             JMP     SAVE    SAVE INPUT PROPER
0626: 17FA D8           PMBINA  CLD             BINARY ARITHMETIC
0627: 17FB 4C CF 14             JMP     STEP    SAVE INPUT PROPER
0628:
0629:
0630:                   ******* END OF THE PME PROGRAM *******
0631:
0632:
0633:                   VICINUS ELABORABAT PROGRAMMAM XXV VI MCMLXXXI
0634:                   VIR NOCTIS IMPOSIT PROGRMMAM IN MACHINAM
0635:
ID=

-T

    SYMBOL TABLE 3000 3276
    ADCENL 1EDC    AGAINH 15CD    ASHET\ 141E    ASMBLZ 1636
    ASSEMJ 1F51    ASSEND 1647    BACK   15D8    BCKA   15F0
    BCKB   1609    BEGADH 00E3    BEGADL 00E2    BEGINH 1ED3
    BINAR  17F6    BRKTHH 1A7D    BRKTLH 1A7C    BRK    1533
    BYTESH 00F6    BYTIN  16AE    BYT    16B1    CENDHH 00E9
    CENDLH 00E8    CHECK  16F6    CHKEND 170A    COUNT  00F7
    CRLF   11E8    CURADH 00E7    CURADL 00E6    DECURA 1692
    DNE    15D5    DONE   1567    EDITC  1500    EDITW  153D
    ENDADH 00E5    ENDADL 00E4    FILLWS 1E47    FOUND  15C3
    FULL   1590    ILLKEY 158C    INCTAB 16A0    INH    00F9
    INPAR  1387    INPUT  164D    INSERT 1579    IOCORR 14F8
    LABELS 00EE    LABLST 1788    LBLSTA 178A    LBLSTB 178F
    LBLSTC 17BF    LENACC 1E60    LINES  161E    LIST   1558
    LST    155F    MA     1741    MEM    1582    MESS   1569
    MESSA  173E    NEXT   1EF8    NIBBLE 00FE    NMIH   1A7B
    NMIL   1A7A    OPLEN  1E5C    PARAH  1A64    PARAL  1A63
    PARBH  1A66    PARBL  1A65    PBDD   1A83    PMBINA 17FA
    POINTH 00FB    POINTL 00FA    PRBYT  128F    PRCHA  1334
    PRINS  1708    PRSP   11F3    PRT    1723    RDA    16F3
    RDB    16F5    READ   16CE    READIN 16C9    RECCHA 12AE
    RECEND 1EEA    RESIN  1268    RESTTY 14BC    RETURN 16C8
    SAVE   1C00    SCAN   15A0    SCNDA  17D5    SCNDB  17E6
    SEACND 17C5    SEARCH 1594    SEMIW  1667    SKIP   156E
    SP     1736    STEP   14CF    SXTEEN 1616    TABLEH 00ED
    TABLEL 00EC    TEMPX  00FD    TOFEND 1613    TOF    160C
    TXTEND 174F    TXT    1750    UP     1E83    WARM   1545
    WRONG  1664
-E
7EE4 0636
-
```

Anhang 2

Hexdump des Systemprogramms PME

Diese Software ist im PM-EPROM zusätzlich untergebracht; sie belegt die Adressen $14F8 . . . $17FF. Während PME die Adressen $14F8 . . . $17F5 umfaßt, gehören die Adressen $17F6 . . . $17FD zu den beiden Routinen, die im Anhang 4 beschrieben sind. Das Hexdump von PM steht auf den Seiten 197 und 198 in Buch 3. Zeile 14FØ ist sowohl dort als auch hier vorhanden.

```
       Ø  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
14FØ: F9 4C A2 13 FF FF FF FF A9 1E 8D 83 1A 4C 51 1F
15ØØ: 2Ø 68 12 AØ ØØ 2Ø 3E 17 2Ø 87 13 3Ø F3 AD 63 1A
1510: 85 E2 18 69 Ø1 85 E8 AD 64 1A 85 E3 69 ØØ 85 E9
1520: AD 65 1A 85 E4 AD 66 1A 85 E5 2Ø D3 1E A9 77 AØ
1530: ØØ 91 E6 A9 3D 8D 7C 1A A9 15 8D 7D 1A AØ 24 2Ø
1540: 3E 17 A2 FF 9A 2Ø ØB 17 2Ø AE 12 C9 4B DØ Ø9 2Ø
1550: 83 1E 2Ø EA 1E 4C 45 15 C9 4C DØ 12 2Ø D3 1E 2Ø
1560: ØB 17 2Ø F8 1E 3Ø F8 AØ 1F 2Ø 3E 17 FØ D7 C9 2Ø
1570: DØ Ø7 2Ø F8 1E 3Ø CE 1Ø EE C9 49 DØ 17 2Ø C9 16
1580: 3Ø ØA 2Ø F6 16 9Ø Ø9 2Ø 47 1E FØ B9 AØ ØE DØ D9
1590: AØ 1A DØ D5 C9 53 DØ 4Ø 2Ø C9 16 3Ø EF 2Ø D3 1E
15AØ: A5 FB 2Ø 6Ø 1E AØ ØØ B1 E6 C5 FB DØ 2Ø C6 F6 FØ
15BØ: 12 C8 B1 E6 C5 FA DØ 15 C6 F6 FØ Ø7 C8 B1 E6 C5
15CØ: F9 DØ ØA 2Ø ØB 17 2Ø AE 12 C9 59 DØ Ø8 2Ø 5C 1E
15DØ: 2Ø F8 1E 3Ø CB 4C 67 15 C9 5A DØ 3Ø A5 E2 85 EC
15EØ: A5 E3 85 ED C5 E7 DØ Ø8 A5 E2 C5 E6 DØ Ø2 FØ 19
15FØ: AØ ØØ B1 EC 2Ø 6Ø 1E 2Ø AØ 16 A5 E6 C5 EC DØ FØ
16ØØ: A5 E7 C5 ED DØ EA 2Ø 92 16 4C 45 15 C9 54 DØ Ø6
1610: 2Ø D3 1E 4C 45 15 C9 5Ø DØ 1C A9 ØF 85 F7 2Ø 5C
1620: 1E 2Ø F8 1E C6 F7 FØ EB AØ ØØ B1 E6 C9 77 FØ E3
1630: 2Ø ØB 17 4C 1E 16 C9 58 DØ 13 A9 47 8D 7A 1A A9
1640: 16 8D 7B 1A 4C F8 14 2Ø BC 14 4C 88 17 2Ø B1 16
1650: 3Ø 12 2Ø CE 16 3Ø ØD 2Ø 5C 1E 2Ø F8 1E A5 FD 85
1660: F6 4C 82 15 4C 8C 15 2Ø 68 12 AØ ØØ 2Ø 3E 17 2Ø
1670: 87 13 3Ø F3 AD 63 1A 85 E2 AD 64 1A 85 E3 AD 65
1680: 1A 85 E4 85 E8 AD 66 1A 85 E5 85 E9 2Ø D3 1E 4C
1690: 33 15 38 A5 E6 E5 F6 85 E6 A5 E7 E9 ØØ 85 E7 6Ø
16AØ: 18 A5 EC 65 F6 85 EC A5 ED 69 ØØ 85 ED 6Ø 2Ø AE
```

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
16B0: 12 20 1E 14 30 12 0A 0A 0A 0A 85 FE 20 AE 12 20
16C0: 1E 14 30 04 05 FE A2 00 60 20 AE 16 30 27 85 FB
16D0: 20 60 1E 84 F7 84 FD C6 F7 F0 18 20 F3 11 20 AE
16E0: 16 30 12 85 FA C6 F7 F0 0A 20 F3 11 20 AE 16 30
16F0: 04 85 F9 A2 00 60 20 DC 1E 38 A5 E4 E9 02 E5 E8
1700: A5 E5 E5 E9 90 04 20 EA 1E 38 60 20 E8 11 20 5C
1710: 1E A6 F6 A5 E7 20 8F 12 A5 E6 20 8F 12 A9 0F 85
1720: EE A0 00 20 F3 11 B1 E6 20 8F 12 38 A5 EE E9 03
1730: 85 EE C8 CA D0 ED 20 F3 11 C6 EE D0 F9 60 20 E8
1740: 11 B9 50 17 C9 03 F0 07 20 34 13 C8 4C 41 17 60
1750: 42 45 47 41 44 2C 45 4E 44 41 44 3A 20 03 49 4C
1760: 4C 45 47 41 4C 20 4B 45 59 03 46 55 4C 4C 03 44
1770: 4F 4E 45 03 50 4D 20 45 44 49 54 4F 52 03 4C 41
1780: 42 20 24 03 3A 20 24 03 A0 FF 20 E8 11 A2 04 84
1790: FD A0 2E 20 41 17 A4 FD B1 EC 20 8F 12 88 84 FD
17A0: A0 34 20 41 17 A4 FD B1 EC 20 8F 12 88 B1 EC 20
17B0: 8F 12 20 F3 11 88 C4 EE F0 05 CA D0 D2 F0 CB 20
17C0: D3 1E 4C 3D 15 20 68 12 A0 00 20 3E 17 20 87 13
17D0: 30 F3 20 D3 1E A0 00 B1 E6 C9 77 F0 09 20 60 1E
17E0: 20 F8 1E 4C D5 17 18 A5 E6 69 01 85 E8 A5 E7 69
17F0: 00 85 E9 4C 33 15 D8 4C 00 1C D8 4C CF 14 77 77
```

Anhang 3

Listing des Systemprogramms TM und Listing des Systemprogramms PM

Auf jeder Buchseite sind 69 Assembler-Zeilen abgedruckt. An den allgemeinen Teil (die Deklarationen) schließen sich nacheinander die folgenden Abschnitte an:

- Das Hauptprogramm TM,
- die Subroutinen von TM,
- die "Super-Subroutine" DUMP/DUMPT,
- die Subroutinen von DUMP/DUMPT,
- die "Super-Subroutine" RDTAPE,
- die Subroutinen von RDTAPE,
- das Hauptprogramm PM,
- die Subroutinen von PM,
- die Look-Up-Tabelle MESS von PM,
- die STEP-Vorroutine von PM,
- die Subroutine IPBRES von PM,
- und zum Schluß eine "Symbol-Tabelle", in der sämtliche Label sowie alle nicht anonymen Speicherplätze angegeben sind.

```
0001: 0800                ORG   $0800  SOFTWARE OF JUNIOR COMPUTER 3,4
0002:
0003:
0004:                SOURCE LISTING OF THE TAPE/PRINTER MONITOR
0005:
0006:                WRITTEN BY A. NACHTMANN
0007:
0008:
0009:                DATE: 17 DEC. 1980
0010:
0011:
0012:
0013:                ************************************
0014:                *** DEFINITIONS OF DUMPT & RDTAPE ***
0015:                ************************************
0016:
0017:
0018:                DUMPT STORES A PROGRAM ON TAPE
0019:                1) STORE A PROGRAM ON TAPE
0020:                2) ID IS THE IDENTITY NUMBER
0021:                3) SA IS THE START ADDRESS
0022:                4) EA IS THE END ADDRESS + 1
0023:                5) ID = 00 & ID = FF SHOULD NOT BE USED
0024:
0025:                RDTAPE READS A PROGRAM FROM TAPE
0026:                1) READ A PROGRAM FROM TAPE WITH A CERTAIN ID
0027:                2) DISCARD DATA IN MEMORY BEGINNING AT SA
0028:                3) ID = 00 & ID = FF ARE SPECIAL IDS
0029:
0030:
0031:
0032:                THE DATA FORMAT ON TAPE IS ASCII:
0033:
0034:                255SYN<>*<>ID<>SAL<>SAH<>DATA<>/<>CHKL<>CHKH<>EOT<>EOT
0035:
0036:                I/O DEFINITION
0037:
0038: 0800           PAD    *      $1A80  PORT A DATA REGISTER
0039: 0800           PADD   *      $1A81  PORT A DATA DIRECTION
0040: 0800           PBD    *      $1A82  PORT B DATA REGISTER
0041: 0800           PBDD   *      $1A83  PORT B DATA DIRECTION
0042:
0043:
0044:                TIMER REGISTERS
0045:
0046: 0800           CNTA   *      $1AF4  CLK1T, DISABLE TIMER IRQ
0047: 0800           CNTC   *      $1AF6  CLK64T, DISABLE TIMER IRQ
0048: 0800           RDFLAG *      $1AD5  READ FLAG REGISTER, B7 IS TIMER FLAG
0049: 0800           RDTDIS *      $1AD4  READ CONTENTS OF TIMER CELL, IRQ DISABLED
0050:
0051:
0052:                TEMPORARY DATA BUFFERS
0053:
0054: 0800           SY     *      $1A69  SYN COUNTER
0055: 0800           BYTE   *      $1A6A  BYTE FROM TAPE
0056: 0800           CHAR   *      $1A6B  CHARACTER FROM TAPE
0057: 0800           HIGHER *      $1A6C  3600 HZ HALF PERIODE DELAY
0058: 0800           LOWER  *      $1A6D  2400 HZ HALF PERIODE DELAY
0059: 0800           CHKL   *      $1A6E  CHECK SUM LOW
0060: 0800           CHKH   *      $1A6F  CHECK SUM HIGH
0061: 0800           SAL    *      $1A70  START ADDRESS LOW
0062: 0800           SAH    *      $1A71  START ADDRESS HIGH
0063: 0800           EAL    *      $1A72  END ADDRESS LOW + 1
0064: 0800           EAH    *      $1A73  END ADDRESS HIGH
0065: 0800           SYNCNT *      $1A74
0066: 0800           BITS   *      $1A75  AMOUNT OF BITS
0067: 0800           FIRST  *      $1A76  HALF PERIODE AMOUNT OF 3600 HZ
0068: 0800           SECOND *      $1A77  HALF PERIODE AMOUNT OF 2400 HZ
0069: 0800           GANG   *      $1A78  TEMP OF PBD-BITS
```

```
0070: 0800            ID     *      $1A79  ID OF THE DUMPT PROGRAM
0071: 0800            NMI    *      $1A7A  NMI VECTOR
0072:
0073:                 ****************************************
0074:                 *** DEFINITIONS OF THE TTY MONITOR ***
0075:                 ****************************************
0076:
0077:
0078:                 I/O DEFINITION: SEE DUMPT & RDTAPE
0079:
0080:
0081:                 CPU REGISTERS
0082:
0083: 0800            PCL    *      $00EF  PROGRAM COUNTER
0084: 0800            PCH    *      $00F0
0085: 0800            PREG   *      $00F1  STATUS REGISTER
0086: 0800            SPUSER *      $00F2  USER STACK POINTER
0087: 0800            ACC    *      $00F3  ACCUMULATOR
0088: 0800            YREG   *      $00F4  INDEX Y
0089: 0800            XREG   *      $00F5  INDEX X
0090:
0091:
0092:                 INPUT BUFFERS
0093:
0094: 0800            INL    *      $00F8  LOWER BYTE
0095: 0800            INH    *      $00F9  UPPER BYTE
0096:
0097:
0098:                 ADDRESS POINTER
0099:
0100: 0800            POINTL *      $00FA
0101: 0800            POINTH *      $00FB
0102:
0103:
0104:                 TEMPORARAY DATA BUFFERS
0105:
0106: 0800            TEMP   *      $00FC
0107: 0800            TEMPX  *      $00FD
0108: 0800            STPBIT *      $1A59  NUMBER OF STOP BITS + 1
0109: 0800            CNTL   *      $1A5A  BIT TIME BUFFER
0110: 0800            CNTH   *      $1A5B
0111: 0800            CNTHL  *      $1A5C  HALF BIT TIME BUFFER
0112: 0800            CNTHH  *      $1A5D
0113: 0800            TIML   *      $1A5E  COUNT DOWN BUFFER
0114: 0800            TIMH   *      $1A5F
0115: 0800            TEMPA  *      $1A60  TEMPS
0116: 0800            TEMPB  *      $1A61
0117: 0800            CHA    *      $1A62  CHARACTER BUFFER
0118: 0800            PARAL  *      $1A63  PARAMETER BUFFERS
0119: 0800            PARAH  *      $1A64
0120: 0800            PARBL  *      $1A65
0121: 0800            PARBH  *      $1A66
0122: 0800            PRTEMP *      $1A67  TTY BUFFER
0123: 0800            BRKT   *      $1A7C  BREAK TEST VECTOR
0124:
0125:
0126:
0127:
0128:                 ************************************
0129:                 *** BUFFERS & EXTERNAL ADDRESSES ***
0130:                 ************************************
0131:
0132: 0800            DISCNT *      $1A68  DISPLAY COUNTER
0133: 0800            COLDST *      $1CB5  EDITOR COLD START
0134: 0800            WARMST *      $1CCA  EDITOR WARM START
0135: 0800            BEGIN  *      $1ED3  EDITOR SUBROUTINE
0136: 0800            RESET  *      $1C1D  RESET OF VERSION D
0137: 0800            GETKEY *      $1DF9  COMPUTE THE KEY VALUE
0138: 0800            LDAINH *      $1DA7  PART OF SCAND/SCANDS
```

```
0139: 0800            BEGADL *      $00E2  BEGIN ADDRESS POINTER
0140: 0800            BEGADH *      $00E3
0141: 0800            ENDADL *      $00E4  END ADDRESS POINTER
0142: 0800            ENDADH *      $00E5
0143: 0800            CENDL  *      $00E8  CURRENT END ADDRESS POINTER
0144: 0800            CENDH  *      $00E9
0145:
0146:                 ************************
0147:                 *** TAPE MANAGEMENT ***
0148:                 ************************
0149:
0150:
0151:                 >PC KEY: READ A DATA BLOCK WITH ID=01...FE FROM TAPE
0152:                          1) IF ID = 00, NO SA MAY BE ENTERED
0153:                          2) IF ID = FF, SAH & SAL SHOULD BE ENTERED
0154:                 >AD KEY: SAVE A DATA BLOCK FROM SA TO EA-1 ON TAPE
0155:                          1) ENTER ID = 01...FE
0156:                          2) ENTER SAH & SAL
0157:                          3) ENTER EAH & EAL + 1
0158:                 >DA KEY: 1) EDITOR COLD START ENTRY
0159:                          2) FILE IS DEFINED BY BEGH,BEGL & ENDH,ENDL
0160:                 >+  KEY: DISPLAY ID-SAH-SAL-EAH-EAL-BEGH-BEGL-ENDH-
0161:                                  ENDL-ID-SAH...
0162:                 >GO KEY: SAVE AN EDITOR FILE BETWEEN BEGAD AND CEND
0163:                            ON TAPE; THEN JUMP TO WARM START ENTRY OF
0164:                            THE EDITOR AND DISPLAY THE FIRST INSTRUCTION
0165:
0166:
0167:
0168: 0800 20 72 0C  GETA   JSR   ADJPNT DECREMENT FILE POINTER
0169: 0803 A5 FA            LDAZ  POINTL POINT := SA
0170: 0805 8D 70 1A         STA   SAL
0171: 0808 A5 FB            LDAZ  POINTH
0172: 080A 8D 71 1A         STA   SAH
0173: 080D 4C 4E 08         JMP   GETB
0174:
0175: 0810 A2 00      TPINIT LDXIM $00
0176: 0812 8E 79 1A         STX   ID      RESET ALL TAPE PARAMETERS
0177: 0815 8E 70 1A         STX   SAL
0178: 0818 8E 71 1A         STX   SAH
0179: 081B 8E 72 1A         STX   EAL
0180: 081E 8E 73 1A         STX   EAH
0181:
0182: 0821 8E 68 1A  TPI     STX   DISCNT RESET DISPLAY COUNTER
0183: 0824 A9 06            LDAIM $06
0184: 0826 8D 82 1A         STA   PBD     DISPLAY OFF
0185: 0829 A9 1E            LDAIM $1E
0186: 082B 8D 83 1A         STA   PBDD
0187:
0188: 082E 20 36 09  TPTXT   JSR   TAPDIS DISPLAY TAPE PARAMETERS
0189: 0831 D0 FB            BNE   TPTXT   KEY RELEASED?
0190:
0191: 0833 20 36 09  TTXT    JSR   TAPDIS
0192: 0836 F0 FB            BEQ   TTXT    ANY KEY DEPRESSED?
0193: 0838 20 36 09         JSR   TAPDIS DEBOUNCE KEY
0194: 083B F0 F6            BEQ   TTXT    KEY STILL DEPRESSED?
0195: 083D 20 F9 1D         JSR   GETKEY RETURN WITH KEY IN ACCU
0196:
0197: 0840 C9 14      GET    CMPIM $14     PC KEY?
0198: 0842 D0 0E            BNE   SAVE
0199: 0844 20 02 0B         JSR   RDTAPE READ DATA FROM TAPE
0200: 0847 A2 FF            LDXIM $FF
0201: 0849 EC 79 1A         CPX   ID      CHECK, IF ID = FF
0202: 084C F0 B2            BEQ   GETA    IF YES, ADJUST FILE POINTER
0203:
0204: 084E E8         GETB   INX           RESET DISCNT
0205: 084F 4C 21 08         JMP   TPI     SHOW 'ID  XX'
0206:
0207: 0852 C9 10      SAVE   CMPIM $10     AD KEY?
```

```
0208: 0854 D0 08           BNE   PLUS
0209: 0856 20 DF 09        JSR   DUMP    WRITE DATA ON TAPE
0210: 0859 A2 00           LDXIM $00
0211: 085B 4C 21 08        JMP   TPI     SHOW 'ID  XX'
0212:
0213: 085E C9 12      PLUS CMPIM $12     + KEY?
0214: 0860 D0 11           BNE   FILES
0215: 0862 EE 68 1A        INC   DISCNT SET UP PARAMETER COUNTER
0216: 0865 A0 09           LDYIM $09     LIMIT COUNT
0217: 0867 CC 68 1A        CPY   DISCNT
0218: 086A D0 C2           BNE   TPTXT
0219: 086C A0 00           LDYIM $00     RESET PARAMETER COUNTER
0220: 086E 8C 68 1A        STY   DISCNT
0221: 0871 F0 BB           BEQ   TPTXT
0222:
0223: 0873 C9 13      FILES CMPIM $13    GO KEY?
0224: 0875 D0 25           BNE   DAT
0225: 0877 A5 E2           LDAZ  BEGADL
0226: 0879 8D 70 1A        STA   SAL     BEGAD := SA
0227: 087C A5 E3           LDAZ  BEGADH
0228: 087E 8D 71 1A        STA   SAH
0229: 0881 A5 E8           LDAZ  CENDL   CEND := EA
0230: 0883 8D 72 1A        STA   EAL
0231: 0886 A5 E9           LDAZ  CENDH
0232: 0888 8D 73 1A        STA   EAH
0233:
0234: 088B 20 DF 09        JSR   DUMP    WRITE DATA = FILE ON TAPE
0235: 088E 20 D3 1E        JSR   BEGIN   SHOW THE FIRST INSTRUKTION
0236: 0891 A9 1E           LDAIM $1E     DIFINE I/O
0237: 0893 8D 83 1A        STA   PBDD
0238: 0896 A2 FF           LDXIM $FF     RESET STACK
0239: 0898 9A              TXS
0240: 0899 4C CA 1C        JMP   WARMST RETURN TO EDITOR
0241:
0242: 089C C9 11      DAT  CMPIM $11     DA KEY?
0243: 089E D0 03           BNE   SHIFT   IT WAS A DATA KEY
0244: 08A0 4C B5 1C        JMP   COLDST EDITOR COLD START ENTRY
0245:
0246: 08A3 06 F9      SHIFT ASLZ  INH     SHIFT KEY INTO DISPLAY BUFFER
0247: 08A5 06 F9           ASLZ  INH
0248: 08A7 06 F9           ASLZ  INH
0249: 08A9 06 F9           ASLZ  INH
0250: 08AB 05 F9           ORAZ  INH
0251: 08AD 85 F9           STAZ  INH
0252: 08AF A0 00           LDYIM $00     RESET PARAMETER COUNTER
0253: 08B1 CC 68 1A        CPY   DISCNT ID TO DISPLAY?
0254: 08B4 D0 06           BNE   STSAH
0255: 08B6 8D 79 1A        STA   ID      SAVE ID
0256: 08B9 4C 2E 08        JMP   TPTXT
0257:
0258: 08BC C8         STSAH INY
0259: 08BD CC 68 1A        CPY   DISCNT SAH TO DISPLAY?
0260: 08C0 D0 06           BNE   STSAL
0261: 08C2 8D 71 1A        STA   SAH     SAVE SAH
0262: 08C5 4C 2E 08        JMP   TPTXT
0263:
0264: 08C8 C8         STSAL INY
0265: 08C9 CC 68 1A        CPY   DISCNT SAL TO DISPLAY?
0266: 08CC D0 06           BNE   STEAH
0267: 08CE 8D 70 1A        STA   SAL     SAVE SAL
0268: 08D1 4C 2E 08        JMP   TPTXT
0269:
0270: 08D4 C8         STEAH INY
0271: 08D5 CC 68 1A        CPY   DISCNT EAH TO DISPLAY?
0272: 08D8 D0 06           BNE   STEAL
0273: 08DA 8D 73 1A        STA   EAH     SAVE EAH
0274: 08DD 4C 2E 08        JMP   TPTXT
0275:
0276: 08E0 C8         STEAL INY
```

```
0277: 08E1 CC 68 1A          CPY   DISCNT EAL TO DISPLAY?
0278: 08E4 D0 06             BNE   STBEGH
0279: 08E6 8D 72 1A          STA   EAL    SAVE EAL
0280: 08E9 4C 2E 08          JMP   TPTXT
0281:
0282: 08EC C8          STBEGH INY
0283: 08ED CC 68 1A          CPY   DISCNT BEGH TO DISPLAY?
0284: 08F0 D0 05             BNE   STBEGL
0285: 08F2 85 E3             STAZ  BEGADH SAVE BEGADH
0286: 08F4 4C 2E 08          JMP   TPTXT
0287:
0288: 08F7 C8          STBEGL INY
0289: 08F8 CC 68 1A          CPY   DISCNT BEGADL TO DISPLAY?
0290: 08FB D0 05             BNE   STENDH
0291: 08FD 85 E2             STAZ  BEGADL SAVE BEGADL
0292: 08FF 4C 2E 08          JMP   TPTXT
0293:
0294: 0902 C8          STENDH INY
0295: 0903 CC 68 1A          CPY   DISCNT ENDADH TO DISPLAY?
0296: 0906 D0 05             BNE   STENDL
0297: 0908 85 E5             STAZ  ENDADH SAVE ENDADH
0298: 090A 4C 2E 08          JMP   TPTXT
0299:
0300: 090D C8          STENDL INY
0301: 090E CC 68 1A          CPY   DISCNT ENDADL TO DISPLAY?
0302: 0911 D0 05             BNE   TPVEC  ERROR EXIT
0303: 0913 85 E4             STAZ  ENDADL SAVE ENDADL
0304: 0915 4C 2E 08          JMP   TPTXT
0305:
0306:
0307: 0918 4C 10 08    TPVEC JMP   TPINIT
0308:
0309:                  SUBROUTINES OF 'TAPE MANAGEMENT'
0310:
0311: 091B B9 BB 09    TDISP LDAY  TLOOK  TEXT LOOKUP
0312: 091E 8D 80 1A          STA   PAD    OUTPUT TEXT PATTERN
0313: 0921 8E 82 1A          STX   PBD    DISPLAY ENABLE
0314: 0924 98                TYA         SAVE INDEX Y
0315: 0925 A0 7F             LDYIM $7F
0316:
0317: 0927 88          DELY  DEY         DELAY
0318: 0928 D0 FD             BNE   DELY
0319: 092A A8                TAY         RESTORE INDEX Y
0320: 092B E8                INX         SET UP FOR NEXT DISPLAY
0321: 092C E8                INX
0322: 092D C8                INY         ADJUST TEXT POINTER
0323: 092E E0 10             CPXIM $10   4 DISPLAYS SCANNED?
0324: 0930 D0 E9             BNE   TDISP
0325: 0932 20 A7 1D          JSR   LDAINH DISPLAY DATA BUFFER INH
0326: 0935 60                RTS
0327:
0328: 0936 A9 7F       TAPDIS LDAIM $7F
0329: 0938 8D 81 1A          STA   PADD   I/O DEFINITION
0330:
0331: 093B A2 08       SID   LDXIM $08    INIT DISPLAY SWITCH
0332: 093D A0 00             LDYIM $00
0333: 093F CC 68 1A          CPY   DISCNT ID TO DISPLAY?
0334: 0942 D0 09             BNE   SSAH
0335: 0944 AD 79 1A          LDA   ID     ID TO DISPLAY
0336: 0947 85 F9             STAZ  INH
0337:
0338: 0949 20 1B 09    COMPNT JSR  TDISP  SHOW ANY PARAMETER
0339: 094C 60                RTS
0340:
0341: 094D C8          SSAH  INY
0342: 094E CC 68 1A          CPY   DISCNT SAH TO DISPLAY?
0343: 0951 D0 09             BNE   SSAL
0344: 0953 AD 71 1A          LDA   SAH    SAH TO DISPLAY
0345: 0956 85 F9             STAZ  INH
```

```
0346: 0958 A0 04            LDYIM $04    LOOKUP POINTER
0347: 095A D0 ED            BNE   COMPNT SHOW SAH
0348:
0349: 095C C8         SSAL  INY
0350: 095D CC 68 1A         CPY   DISCNT SAL TO DISPLAY?
0351: 0960 D0 09            BNE   SEAH
0352: 0962 AD 70 1A         LDA   SAL    SAL TO DISPLAY
0353: 0965 85 F9            STAZ  INH
0354: 0967 A0 08            LDYIM $08    LOOKUP POINTER
0355: 0969 D0 DE            BNE   COMPNT SHOW SAL
0356:
0357: 096B C8         SEAH  INY
0358: 096C CC 68 1A         CPY   DISCNT EAH TO DISPLAY?
0359: 096F D0 09            BNE   SEAL
0360: 0971 AD 73 1A         LDA   EAH    EAH TO DISPLAY
0361: 0974 85 F9            STAZ  INH
0362: 0976 A0 0C            LDYIM $0C    LOOKUP POINTER
0363: 0978 D0 CF            BNE   COMPNT SHOW EAH
0364:
0365: 097A C8         SEAL  INY
0366: 097B CC 68 1A         CPY   DISCNT
0367: 097E D0 09            BNE   SBEGH
0368: 0980 AD 72 1A         LDA   EAL
0369: 0983 85 F9            STAZ  INH
0370: 0985 A0 10            LDYIM $10    LOOKUP POINTER
0371: 0987 D0 C0            BNE   COMPNT SHOW EAL
0372:
0373: 0989 C8         SBEGH INY
0374: 098A CC 68 1A         CPY   DISCNT
0375: 098D D0 08            BNE   SBEGL
0376: 098F A5 E3            LDAZ  BEGADH
0377: 0991 85 F9            STAZ  INH
0378: 0993 A0 14            LDYIM $14
0379: 0995 D0 B2            BNE   COMPNT
0380:
0381: 0997 C8         SBEGL INY
0382: 0998 CC 68 1A         CPY   DISCNT
0383: 099B D0 08            BNE   SENDH
0384: 099D A5 E2            LDAZ  BEGADL
0385: 099F 85 F9            STAZ  INH
0386: 09A1 A0 18            LDYIM $18
0387: 09A3 D0 A4            BNE   COMPNT
0388:
0389: 09A5 C8         SENDH INY
0390: 09A6 CC 68 1A         CPY   DISCNT
0391: 09A9 D0 08            BNE   SENDL
0392: 09AB A5 E5            LDAZ  ENDADH
0393: 09AD 85 F9            STAZ  INH
0394: 09AF A0 1C            LDYIM $1C
0395: 09B1 D0 96            BNE   COMPNT
0396:
0397: 09B3 A5 E4      SENDL LDAZ  ENDADL
0398: 09B5 85 F9            STAZ  INH
0399: 09B7 A0 20            LDYIM $20
0400: 09B9 D0 8E            BNE   COMPNT
0401:
0402:
0403:
0404:                  7 SEGMENT TEXT LOOKUP TABLE
0405:
0406: 09BB 66         TLOOK =     $66    ID PATTERN
0407: 09BC 21               =     $21
0408: 09BD 7F               =     $7F
0409: 09BE 7F               =     $7F
0410:
0411: 09BF 52               =     $52    SAH PATTERN
0412: 09C0 08               =     $08
0413: 09C1 09               =     $09
0414: 09C2 7F               =     $7F
```

```
0415:
0416: 09C3 52                =      $52     SAL PATTERN
0417: 09C4 08                =      $08
0418: 09C5 47                =      $47
0419: 09C6 7F                =      $7F
0420:
0421: 09C7 06                =      $06     EAH PATTERN
0422: 09C8 08                =      $08
0423: 09C9 09                =      $09
0424: 09CA 7F                =      $7F
0425:
0426: 09CB 06                =      $06     EAL PATTERN
0427: 09CC 08                =      $08
0428: 09CD 47                =      $47
0429: 09CE 7F                =      $7F
0430:
0431: 09CF 03                =      $03     BEGH PATTERN
0432: 09D0 06                =      $06
0433: 09D1 42                =      $42
0434: 09D2 09                =      $09
0435:
0436: 09D3 03                =      $03     BEGL PATTERN
0437: 09D4 06                =      $06
0438: 09D5 42                =      $42
0439: 09D6 47                =      $47
0440:
0441: 09D7 06                =      $06     ENDH PATTERN
0442: 09D8 2B                =      $2B
0443: 09D9 21                =      $21
0444: 09DA 09                =      $09
0445:
0446: 09DB 06                =      $06     ENDL PATTERN
0447: 09DC 2B                =      $2B
0448: 09DD 21                =      $21
0449: 09DE 47                =      $47
0450:                  ******************************
0451:                  * DUMP/DUMPT IS A SUBROUTINE *
0452:                  ******************************
0453:
0454: 09DF A9 7D      DUMP    LDAIM $7D     3600HZ HALF PERIODE DELAY
0455: 09E1 8D 6C 1A           STA   HIGHER
0456: 09E4 A9 C3              LDAIM $C3     2400 HZ HALF PERIODE DELAY
0457: 09E6 8D 6D 1A           STA   LOWER
0458: 09E9 A9 03              LDAIM $03     TOGGLE 3 TIMES AT 3600 HZ
0459: 09EB 8D 76 1A           STA   FIRST
0460: 09EE A9 02              LDAIM $02     TOGGLE 2 TIMES AT 2400 HZ
0461: 09F0 8D 77 1A           STA   SECOND
0462:
0463: 09F3 A9 47      DUMPT   LDAIM $47     PBD PATTERN
0464: 09F5 A2 FF              LDXIM $FF     PBDD PATTERN
0465: 09F7 8D 82 1A           STA   PBD     INPUT OFF,OUTPUT ON,STROBE DISABLED
0466: 09FA 8D 78 1A           STA   GANG    SAVE BITS OF PBD
0467: 09FD 8E 83 1A           STX   PBDD    PB7...PB0 IS OUTPUT
0468: 0A00 A9 00              LDAIM $00     PAD PATTERN
0469: 0A02 A2 7F              LDXIM $7F     PADD PATTERN
0470: 0A04 8D 80 1A           STA   PAD     SEGMENTS ON BUT DISABLED
0471: 0A07 8E 81 1A           STX   PADD    ALL LINES OUTPUT EXEPT PA7
0472: 0A0A 8D 6E 1A           STA   CHKL    RESET CHECK SUM
0473: 0A0D 8D 6F 1A           STA   CHKH
0474: 0A10 AD 70 1A           LDA   SAL     INITIALIZE DUMPT POINTER
0475: 0A13 85 FA              STAZ  POINTL
0476: 0A15 AD 71 1A           LDA   SAH
0477: 0A18 85 FB              STAZ  POINTH
0478: 0A1A A2 FF              LDXIM $FF     SET SYNC COUNTER
0479: 0A1C 8E 74 1A           STX   SYNCNT
0480:
0481: 0A1F A9 16      SYNCS   LDAIM '       SYN CHARACTER
0482: 0A21 20 A3 0A           JSR   OUTCH   OUTPUT 255 SYN CHARACTERS
0483: 0A24 CE 74 1A           DEC   SYNCNT
```

```
0484: 0A27 D0 F6            BNE   SYNCS
0485:
0486: 0A29 A9 2A            LDAIM '*      OUTPUT START CHARACTER
0487: 0A2B 20 A3 0A         JSR   OUTCH
0488: 0A2E AD 79 1A         LDA   ID      OUTPUT ID
0489: 0A31 20 8B 0A         JSR   OUTBT
0490: 0A34 AD 70 1A         LDA   SAL     OUTPUT START ADDRESS
0491: 0A37 20 7A 0A         JSR   OUTBTC AND START CHECK SUM COMPUTATION
0492: 0A3A AD 71 1A         LDA   SAH
0493: 0A3D 20 7A 0A         JSR   OUTBTC
0494:
0495: 0A40 A5 FB      DATATR LDAZ  POINTH
0496: 0A42 CD 73 1A         CMP   EAH     ENTIRE FILE TRANSMITTED?
0497: 0A45 D0 23            BNE   HEXDAT
0498: 0A47 A5 FA            LDAZ  POINTL
0499: 0A49 CD 72 1A         CMP   EAL
0500: 0A4C D0 1C            BNE   HEXDAT
0501:
0502: 0A4E A9 2F            LDAIM '/      OUTPUT END OF DATA CHARACTER
0503: 0A50 20 A3 0A         JSR   OUTCH   STOP WITH CHECK SUM COMPUTATION
0504: 0A53 AD 6E 1A         LDA   CHKL    OUTPUT CHECK SUM
0505: 0A56 20 8B 0A         JSR   OUTBT
0506: 0A59 AD 6F 1A         LDA   CHKH
0507: 0A5C 20 8B 0A         JSR   OUTBT
0508: 0A5F A9 04            LDAIM '       EOT CHARACTER
0509: 0A61 20 A3 0A         JSR   OUTCH   OUTPUT EOT CHARACTER
0510: 0A64 A9 04            LDAIM '       EOT CHARACTER
0511: 0A66 20 A3 0A         JSR   OUTCH
0512:
0513: 0A69 60               RTS           RETURN TO CALLER
0514:
0515: 0A6A A0 00      HEXDAT LDYIM $00
0516: 0A6C B1 FA            LDAIY POINTL FETCH CURRENT DATA BYTE
0517: 0A6E 20 7A 0A         JSR   OUTBTC TRANSMIT CURRENT DATA BYTE
0518: 0A71 E6 FA            INCZ  POINTL AND COMPUT CHECK SUM
0519: 0A73 D0 CB            BNE   DATATR SET UP FOR NEXT DATA BYTE
0520: 0A75 E6 FB            INCZ  POINTH
0521: 0A77 4C 40 0A         JMP   DATATR
0522:
0523:                  *** END OF DUMP/DUMPT ***
0524:                  DUMP'S SUBROUTINES
0525:
0526:
0527:                  OUTBTC OUTPUTS A HEX BYTE AS TWO ASCII CHARACTERS
0528:                  TO THE TAPE RECORDER. ALSO THE CHECK SUM IS COM-
0529:                  PUTED.
0530:
0531:                  OUTBT OUTPUTS A HEX BYTE AS TWO ASCCI CHARACTERS
0532:                  TO THE TAPE RECORDER WITHOUT CHECK SUM COMPUTATION.
0533:
0534:                  NIBOUT CONVERTS A HEX NIBBLE TO AN ASCII CHARACTER
0535:                  AND TRANSMITS THE 8 BIT ASCII WORD TO THE TAPE.
0536:
0537:                  HIGH AND LOW PRODUCE THE DELAYS OF THE 3600 HZ AND
0538:                  THE 2400 HZ FREQUENCY.
0539:
0540:
0541:                  *** OUTBTC/OUTBT ***
0542:
0543: 0A7A A8         OUTBTC TAY           SAVE ACCU
0544: 0A7B 18                CLC
0545: 0A7C 6D 6E 1A          ADC   CHKL    CHECK SUM COMPUTATION
0546: 0A7F 8D 6E 1A          STA   CHKL
0547: 0A82 AD 6F 1A          LDA   CHKH
0548: 0A85 69 00             ADCIM $00     CHK:= CHK + BYTE
0549: 0A87 8D 6F 1A          STA   CHKH
0550: 0A8A 98                TYA           GET ACCU AGAIN
0551:
0552: 0A8B A8         OUTBT  TAY           SAVE ACCU TEMP
```

```
0553: 0A8C 4A              LSRA          GET UPPER NIBBLE
0554: 0A8D 4A              LSRA
0555: 0A8E 4A              LSRA
0556: 0A8F 4A              LSRA
0557: 0A90 20 9A 0A        JSR   NIBOUT OUTPUT UPPER NIBBLE AS ASCII CHAR.
0558: 0A93 98              TYA           GET BYTE AGAIN
0559: 0A94 29 0F           ANDIM $0F     GET LOWER NIBBLE
0560: 0A96 20 9A 0A        JSR   NIBOUT OUTPUT LOWER NIBBLE AS ASCII CHAR.
0561: 0A99 60              RTS
0562:
0563:                 *** END OF OUTBTC/OUTBT ***
0564:
0565:
0566:                 *** NIBOUT/OUTCH ***
0567:
0568: 0A9A C9 0A      NIBOUT CMPIM $0A     CONVERT A NIBBLE TO AN ASCII CHAR.
0569: 0A9C 18               CLC
0570: 0A9D 30 02           BMI   NIB
0571: 0A9F 69 07           ADCIM $07
0572:
0573: 0AA1 69 30      NIB    ADCIM $30
0574:
0575: 0AA3 A2 08      OUTCH  LDXIM $08     SET UP FOR 8 BITS
0576: 0AA5 8E 75 1A          STX   BITS
0577:
0578: 0AA8 4A         ONE    LSRA          SHIFT OUT BIT BY BIT
0579: 0AA9 48                PHA           SAVE CHARACTER
0580: 0AAA 90 0C             BCC   ZERO
0581: 0AAC 20 C8 0A          JSR   HIGH    START AT 3600 HZ
0582: 0AAF 20 E5 0A          JSR   LOW
0583: 0AB2 20 E5 0A          JSR   LOW     END AT 2400 HZ
0584: 0AB5 4C C1 0A          JMP   ZRO
0585:
0586: 0AB8 20 C8 0A   ZERO   JSR   HIGH    START AT 3600 HZ
0587: 0ABB 20 C8 0A          JSR   HIGH
0588: 0ABE 20 E5 0A          JSR   LOW     END AT 2400 HZ
0589:
0590: 0AC1 68         ZRO    PLA           GET CHARACTER AGAIN
0591: 0AC2 CE 75 1A          DEC   BITS    ALL BITS SHIFTED OUT
0592: 0AC5 D0 E1             BNE   ONE
0593: 0AC7 60                RTS
0594:
0595:                 *** END OF NIBOUT/OUTCH ***
0596:
0597:
0598:                 *** HIGH ***
0599:
0600: 0AC8 AE 76 1A   HIGH   LDX   FIRST   AMOUNT OF HALF PERIODES
0601:
0602: 0ACB 2C D5 1A   HIG    BIT   RDFLAG TIME OUT?
0603: 0ACE 10 FB             BPL   HIG
0604: 0AD0 AD 6C 1A          LDA   HIGHER SET UP TIMER FOR SHORT PERIODE
0605: 0AD3 8D F4 1A          STA   CNTA    DISABLE TIMER IRQ, CLK1T
0606: 0AD6 AD 78 1A          LDA   GANG
0607: 0AD9 49 80             EORIM $80     TOGGLE OUTPUT
0608: 0ADB 8D 82 1A          STA   PBD
0609: 0ADE 8D 78 1A          STA   GANG    SAVE STATUS QUO
0610: 0AE1 CA                DEX
0611: 0AE2 D0 E7             BNE   HIG
0612: 0AE4 60                RTS
0613:
0614:                 *** END OF HIGH ***
0615:
0616:
0617:                 *** LOW ***
0618:
0619: 0AE5 AE 77 1A   LOW    LDX   SECOND AMOUNT OF HALF PERIODES
0620:
0621: 0AE8 2C D5 1A   LO     BIT   RDFLAG TIME OUT?
```

```
0622: 0AEB 10 FB            BPL   LO
0623: 0AED AD 6D 1A         LDA   LOWER  SET UP TIMER FOR LONG PERIODE
0624: 0AF0 8D F4 1A         STA   CNTA   DISABLE TIMER IRQ, CLK1T
0625: 0AF3 AD 78 1A         LDA   GANG
0626: 0AF6 49 80            EORIM $80    TOGGLE OUTPUT
0627: 0AF8 8D 82 1A         STA   PBD
0628: 0AFB 8D 78 1A         STA   GANG   SAVE STATUS QUO
0629: 0AFE CA               DEX
0630: 0AFF D0 E7            BNE   LO
0631: 0B01 60               RTS
0632:
0633:                  *** END OF LOW ***
0634:
0635:                  ***************************
0636:                  * RDTAPE IS A SUBROUTINE *
0637:                  ***************************
0638:
0639:                  ID = 00: IGNORE ID ON TAPE; DISCARD DATA BLOCK
0640:                  AT SAL, SAH COMING FROM TAPE
0641:
0642:                  ID = FF: IGNORE ID ON TAPE; DISCARD DATA BLOCK
0643:                  AT SAL, SAH STORED IN JUNIORS MEMORY
0644:
0645: 0B02 A9 32      RDTAPE LDAIM $32    INPUT RECORDER IS ON
0646: 0B04 8D 82 1A         STA   PBD    OUTPUT RECORDER IS OFF, PB7 IS ENABLED
0647: 0B07 8D 78 1A         STA   GANG   STROBE '9' IS ENABLED
0648: 0B0A A9 7E            LDAIM $7E    PB0 IS INPUT
0649: 0B0C 8D 83 1A         STA   PBDD   PB7 IS INPUT
0650: 0B0F A9 7F            LDAIM $7F    PA6...PA0 IS OUTPUT
0651: 0B11 8D 81 1A         STA   PADD
0652: 0B14 A9 00            LDAIM $00    SEGMENTS ON
0653: 0B16 8D 6E 1A         STA   CHKL   RESET CHECK SUM
0654: 0B19 8D 6F 1A         STA   CHKH
0655:
0656: 0B1C A9 FF      SYNC  LDAIM $FF    RESET FOR SYN CHARACTER
0657: 0B1E 8D 6B 1A         STA   CHAR
0658:
0659: 0B21 20 C2 0B   SYNCA JSR   RDBIT  READ A BIT FROM TAPE
0660: 0B24 6E 6B 1A         ROR   CHAR   RIGHT SHIFT
0661: 0B27 AD 6B 1A         LDA   CHAR   GET CURRENT CHARACTER
0662: 0B2A 20 E8 0B         JSR   BTWEEN DISPLAY NOT SYN CHARACTER
0663: 0B2D C9 16            CMPIM '      SYN CHARACTER?
0664: 0B2F D0 F0            BNE   SYNCA  IF NOT, RESYNC
0665: 0B31 A0 0A            LDYIM $0A    TRY IT FOR 10 SYNS AT LEAST
0666: 0B33 8C 69 1A         STY   SY     SYN COUNTER
0667:
0668: 0B36 20 36 0C   TENSYN JSR   RDCH
0669: 0B39 20 5D 0C         JSR   CHARVU DISPLAY SYN CHARACTER
0670: 0B3C C9 16            CMPIM '      STILL SYN CHARACTER?
0671: 0B3E D0 DC            BNE   SYNC   IF NOT, RETURN
0672: 0B40 CE 69 1A         DEC   SY     10 SYNS RECEIVED?
0673: 0B43 D0 F1            BNE   TENSYN RETURN IF LESS THAN 10 SYNS
0674:
0675: 0B45 20 36 0C   STAR  JSR   RDCH   WAIT FOR '*' CHARACTER
0676: 0B48 20 5D 0C         JSR   CHARVU
0677: 0B4B C9 2A            CMPIM '*
0678: 0B4D F0 06            BEQ   STARA
0679: 0B4F C9 16            CMPIM '      STILL SYN CHARACTER?
0680: 0B51 F0 F2            BEQ   STAR   IF YES, THEN WAIT
0681: 0B53 D0 AD            BNE   RDTAPE IF NOT, THEN RESYNC
0682:
0683: 0B55 20 5D 0C   STARA JSR   CHARVU DISPLAY '*'
0684: 0B58 20 F3 0B         JSR   RDBYT  READ ID FROM TAPE
0685: 0B5B CD 79 1A         CMP   ID     REQUESTED ID?
0686: 0B5E D0 3E            BNE   CHKID
0687:
0688: 0B60 20 F3 0B   RDSA  JSR   RDBYT  READ SAL FROM TAPE
0689: 0B63 20 4B 0C         JSR   CHKSUM CHECK SUM COMPUTATION
0690: 0B66 85 FA            STAZ  POINTL SET UP STORE POINTER
```

```
0691: 0B68 20 F3 0B          JSR   RDBYT   READ SAH FROM TAPE
0692: 0B6B 20 4B 0C          JSR   CHKSUM
0693: 0B6E 85 FB             STAZ  POINTH
0694:
0695: 0B70 20 F3 0B  FILMEM  JSR   RDBYT   READ DATA BYTE FROM TAPE
0696: 0B73 30 8D             BMI   RDTAPE  NOT VALID HEX CHARACTER
0697: 0B75 F0 13             BEQ   CHECK   END OF DATA CHARACTER?
0698: 0B77 20 4B 0C          JSR   CHKSUM
0699: 0B7A A0 00             LDYIM $00
0700: 0B7C 91 FA             STAIY POINTL  STORE BYTE IN MEMORY
0701: 0B7E E6 FA             INCZ  POINTL  SET POINTER FOR NEXT BYTE
0702: 0B80 D0 02             BNE   FMA
0703: 0B82 E6 FB             INCZ  POINTH
0704:
0705: 0B84 20 64 0C  FMA     JSR   VU      DISPLAY '*'
0706: 0B87 4C 70 0B          JMP   FILMEM  READ NEXT DATA BYTE FROM TAPE
0707:
0708: 0B8A 20 F3 0B  CHECK   JSR   RDBYT   READ CHECK SUM FROM TAPE
0709: 0B8D CD 6E 1A          CMP   CHKL    AND COMPARE IT
0710: 0B90 D0 09             BNE   SYNVEC
0711: 0B92 20 F3 0B          JSR   RDBYT
0712: 0B95 CD 6F 1A          CMP   CHKH
0713: 0B98 D0 01             BNE   SYNVEC
0714: 0B9A 60                RTS           RETURN TO CALLER
0715:
0716: 0B9B 4C 02 0B  SYNVEC  JMP   RDTAPE
0717:
0718: 0B9E AD 79 1A  CHKID   LDA   ID
0719: 0BA1 C9 00             CMPIM $00     ID = 00?
0720: 0BA3 F0 BB             BEQ   RDSA
0721: 0BA5 C9 FF             CMPIM $FF     ID = FF?
0722: 0BA7 D0 F2             BNE   SYNVEC
0723: 0BA9 20 F3 0B          JSR   RDBYT   READ SA FROM TAPE, BUT IGNORE IT
0724: 0BAC 20 4B 0C          JSR   CHKSUM
0725: 0BAF 20 F3 0B          JSR   RDBYT
0726: 0BB2 20 4B 0C          JSR   CHKSUM
0727: 0BB5 AD 70 1A          LDA   SAL     USE SA STORED IN BUFFER
0728: 0BB8 85 FA             STAZ  POINTL
0729: 0BBA AD 71 1A          LDA   SAH
0730: 0BBD 85 FB             STAZ  POINTH
0731: 0BBF 4C 70 0B          JMP   FILMEM
0732:
0733:                    *** END OF RDTAPE ***
0734:
0735:                    SUBROUTINES OF RDTAPE
0736:
0737:                    *** RDBIT ***
0738:
0739:                    RDBIT READS 1 BIT FROM TAPE.
0740:                    LOG 1: IT RETURNS WITH C = 1
0741:                    LOG 0: IT RETUNS WITH C = 0
0742:
0743: 0BC2 2C 82 1A  RDBIT  BIT   PBD     3600 HZ?
0744: 0BC5 10 FB             BPL   RDBIT
0745: 0BC7 AD D4 1A          LDA   RDTDIS  GET COUNT DOWN
0746: 0BCA A0 FF             LDYIM $FF
0747: 0BCC 8C F6 1A          STY   CNTC    START TIMER FOR 2400 HZ COUNT DOWN
0748: 0BCF A0 14             LDYIM $14
0749:
0750: 0BD1 88        RDBA    DEY           DELAY JITTER TIME
0751: 0BD2 D0 FD             BNE   RDBA
0752:
0753: 0BD4 2C 82 1A  RDBB    BIT   PBD     2400 HZ?
0754: 0BD7 30 FB             BMI   RDBB
0755: 0BD9 38                SEC
0756: 0BDA ED D4 1A          SBC   RDTDIS  SET OR RESET C-FLAG
0757: 0BDD A0 FF             LDYIM $FF
0758: 0BDF 8C F6 1A          STY   CNTC    START TIMER FOR 3600 HZ COUNT DOWN
0759: 0BE2 A0 07             LDYIM $07     DELAY FOR JITTER
```

```
0760:
0761: 0BE4 88          RDBC    DEY
0762: 0BE5 D0 FD                BNE     RDBC
0763: 0BE7 60                   RTS
0764:
0765:                   *** END OF RDBIT ***
0766:
0767:
0768:
0769:                   *** BTWEEN ***
0770:
0771:                   DISPLAY THE BETWEEN CHARACTER
0772:
0773: 0BE8 48          BTWEEN  PHA             SAVE ACCU
0774: 0BE9 A9 36                LDAIM $36       OUTPUT BETWEEN CHARACTER
0775: 0BEB 8D 80 1A             STA     PAD
0776: 0BEE 68                   PLA             GET ACCU AGAIN
0777: 0BEF 20 64 0C             JSR     VU      DON'T CHANGE BETWEEN
0778: 0BF2 60                   RTS
0779:
0780:                   *** END OF BTWEEN ***
0781:                   *** RDBYT ***
0782:
0783:
0784:                   READ 1 HEX BYTE = 2 ASCII CHARACTERS FROM TAPE.
0785:                   COMPOSE THE HEX VALUES OF THE CHARACTERS IN ACCU
0786:                   AND RETURN.
0787:
0788:                   1) N = 1: A NOT VALID HEX CHARACTER WAS TRANSMITTED
0789:                   2) Z = 1: END OF DATA TRANSMISSION
0790:                   3) Z = 0: VALID HEX BYTE IN ACCU
0791:                   4) THE N-FLAG HAS ALWAYS PRIORITY
0792:
0793: 0BF3 20 36 0C  RDBYT  JSR     RDCH    READ ANY ASCII CHARACTER FROM TAPE
0794: 0BF6 C9 2F                CMPIM '/        END OF DATA CHARACTER?
0795: 0BF8 D0 01                BNE     RBB
0796:
0797: 0BFA 60          RBA     RTS             ERROR EXIT
0798:
0799: 0BFB 20 19 0C    RBB     JSR     ASCHEX  ASCII HEX CONVERSION
0800: 0BFE 30 FA                BMI     RBA     NOT VALID CHARACTER
0801: 0C00 0A                   ASLA            SHIFT NIBBLE TO LEFT
0802: 0C01 0A                   ASLA
0803: 0C02 0A                   ASLA
0804: 0C03 0A                   ASLA
0805: 0C04 8D 6A 1A             STA     BYTE    SAVE HIGH ORDER NIBBLE
0806: 0C07 20 36 0C             JSR     RDCH    READ NEXT CHARACTER
0807: 0C0A C9 2F                CMPIM '/        END OF DATA CHARACTER
0808: 0C0C F0 EC                BEQ     RBA
0809: 0C0E 20 19 0C             JSR     ASCHEX  ASCII HEX CONVERSION
0810: 0C11 30 E7                BMI     RBA     NOT VALID CHARACTER
0811: 0C13 0D 6A 1A             ORA     BYTE    BYTE = HIGH ORDER AND LOW ORDER NIBBLE
0812: 0C16 A0 01                LDYIM $01       BE SHURE THAT CHARACTER
0813: 0C18 60                   RTS             NORMAL EXIT
0814:
0815:                   *** END OF RDBYT ***
0816:                   *** ASCHEX ***
0817:
0818:
0819:                   CONVERT AN ASCII CHARACTER TO A HEX DATA NIBBLE.
0820:
0821:                   1) RETURN WITH CONVERTED HEX NUMBER IN ACCU
0822:                   2) N = 1, IF NOT VALID HEX NUMBER
0823:                   3) Z = 1, IF VALID HEX NUMBER
0824:                   4) *** ASCHEX IS ALSO USED IN THE PRINTER SOFTWARE ***
0825:
0826: 0C19 C9 30     ASCHEX CMPIM $30    IGNORE 00...2F
0827: 0C1B 30 0C                BMI     NOTVAL
0828: 0C1D C9 3A                CMPIM $3A
```

```
0829: 0C1F 30 0B            BMI   VALID
0830: 0C21 C9 41            CMPIM $41     IGNORE 3A...40
0831: 0C23 30 04            BMI   NOTVAL
0832: 0C25 C9 47            CMPIM $47     IGNORE 47...7F
0833: 0C27 30 03            BMI   VALID
0834:
0835: 0C29 A0 FF      NOTVAL LDYIM $FF     SET N-FLAG
0836: 0C2B 60                RTS           ERROR EXIT
0837:
0838: 0C2C C9 40      VALID  CMPIM $40     ASCII HEX CONVERSION
0839: 0C2E 30 03            BMI   VAL
0840: 0C30 18                CLC
0841: 0C31 69 09            ADCIM $09
0842:
0843: 0C33 29 0F      VAL    ANDIM $0F     HEX DATA IS LOW ORDER NIBBLE IN ACCU
0844: 0C35 60                RTS
0845:
0846:                *** END OF ASCHEX ***
0847:                *** RDCH ***
0848:
0849:
0850:                READ AN ASCII CHARACTER FROM TAPE AND
0851:                STORE IT IN ACCU
0852:
0853: 0C36 A2 08      RDCH   LDXIM $08     SET UP FOR 8 BITS
0854:
0855: 0C38 20 C2 0B  READ   JSR   RDBIT    READ A BIT FROM TAPE
0856: 0C3B 6E 6B 1A          ROR   CHAR    SHIFT BIT INTO CHARACTER
0857: 0C3E CA                DEX           ALL BITS READ?
0858: 0C3F D0 F7            BNE   READ
0859: 0C41 2E 6B 1A          ROL   CHAR    B7 MUST BE ZERO
0860: 0C44 4E 6B 1A          LSR   CHAR
0861: 0C47 AD 6B 1A          LDA   CHAR    RECEIVED CHARACTER TO ACCU
0862: 0C4A 60                RTS
0863:
0864:                *** END OF RDCH ***
0865:
0866:                *** CHKSUM ***
0867:
0868:
0869:                COMPUTE CHECK SUM OF RECEIVED DATA
0870:
0871: 0C4B 48        CHKSUM PHA           SAVE ACCU
0872: 0C4C 18                CLC
0873: 0C4D 6D 6E 1A          ADC   CHKL    CHK := CHK + BYTE
0874: 0C50 8D 6E 1A          STA   CHKL
0875: 0C53 AD 6F 1A          LDA   CHKH
0876: 0C56 69 00             ADCIM $00
0877: 0C58 8D 6F 1A          STA   CHKH
0878: 0C5B 68                PLA           GET ACCU AGAIN
0879: 0C5C 60                RTS
0880:
0881:                *** END OF CHKSUM ***
0882:
0883:
0884:                *** CHARVU ***
0885:
0886:
0887:                OUTPUT ANY CHARACTER TO 7 SEGMENT DISPLAY
0888:                CHARACTER CHANGE IS ENABLED/DISABLED
0889:
0890: 0C5D 48        CHARVU PHA           SAVE ACCU
0891: 0C5E 49 7F             EORIM $7F     OUTPUT INVERTED CHAR. TO SEGMENTS
0892: 0C60 8D 80 1A          STA   PAD
0893: 0C63 68                PLA           RESTORE ACCU
0894:
0895: 0C64 48        VU     PHA
0896: 0C65 AD 78 1A          LDA   GANG
0897: 0C68 49 02             EORIM $02     CHANGE DISPLAYS
```

```
0898: 0C6A 8D 82 1A          STA   PBD
0899: 0C6D 8D 78 1A          STA   GANG    SAVE STATUS QUO
0900: 0C70 68                PLA
0901: 0C71 60                RTS
0902:
0903:                  *** END OF CHARVU ***
0904:
0905:
0906:                  ADJPNT ADDRESS POINTER ADJUSTMENT
0907: 0C72 38          ADJPNT SEC           16 BIT SUBTRACTION
0908: 0C73 A5 FA             LDAZ  POINTL
0909: 0C75 E9 01             SBCIM $01
0910: 0C77 85 FA             STAZ  POINTL
0911: 0C79 A5 FB             LDAZ  POINTH
0912: 0C7B E9 00             SBCIM $00
0913: 0C7D 85 FB             STAZ  POINTH
0914: 0C7F 60                RTS
0915:
0916:                  *** END OF ADJPNT ***
ID=B4
R
X

0001: 1000                   ORG   $1000
0002:
0003:
0004:                  ******************************************
0005:                  *** MAIN PROGRAM OF THE PRINTER MONITOR ***
0006:                  ******************************************
0007:
0008:
0009:                  COMMANDS OF THE PRINTER  ONITOR:
0010:
0011:                  > '-'  DECREMENT THE CURRENT ADDRESS BY ONE
0012:                  > '+'  INCREMENT THE CURRENT ADDRESS BY ONE
0013:                  > 'SPACE'  1) PRINT THE ADDRESS IN THE INPUT BUFFER
0014:                             2) SHOW THE DATA OF THIS ADDRESS
0015:                  > '.'  STORE INPUT DATA AT THE CURRENT ADDRESS
0016:                  > 'R'   START PROGRAM EXECUTION AT THE CURRENT ADDRESS
0017:                  > 'L'  LIST THE CONTENTS OF ALL CPU-REGISTERS
0018:                  > 'P'  PRINT OUT THE LAST PROGRAM CONTER
0019:                  > 'M'  PRINT A HEXDUMP SPECIFIED BY THE INPUT PARAMETER
0020:                  > 'G' READ A DATA BLOCK WITH A CERTAIN ID FROM TAPE
0021:                  > 'S' STORE A DATA BLOCK BETWEEN SA AND EA-1 ON TAPE
0022:
0023:
0024:
0025: 1000 D8          INITPR CLD           RESET SEQUENCE OF THE PRINTER PROGRAM
0026: 1001 78                 SEI           IRQ LINE IS DISABLED
0027: 1002 A9 67               LDAIM $67
0028: 1004 8D 82 1A            STA   PBD    PBD: 01100111
0029: 1007 A9 00               LDAIM $00
0030: 1009 8D 80 1A            STA   PAD    PAD: 00000000
0031: 100C A2 FE               LDXIM $FE    RESET BIT TIME COUNTER
0032: 100E 8E 5A 1A            STX   CNTL
0033: 1011 E8                  INX
0034: 1012 8E 5B 1A            STX   CNTH
0035: 1015 9A                  TXS          RESET COMPUTER STACK POINTER
0036: 1016 86 F2               STXZ  SPUSER RESET USER STACK POINTER
0037: 1018 A9 7F               LDAIM $7F
0038: 101A 8D 81 1A            STA   PADD   PADD: 01111111
0039: 101D 8D 83 1A            STA   PBDD   PBDD: 01111111
0040: 1020 A2 02               LDXIM $02
0041: 1022 8E 59 1A            STX   STPBIT TRANSMIT NONE PARITY & ONE STOP BIT
0042: 1025 A9 5F               LDAIM LABJUN
0043: 1027 8D 7C 1A            STA   BRKT   SETUP BREAK VECTOR
0044: 102A A9 10               LDAIM LABJUN /256
0045: 102C 8D 7D 1A            STA   BRKT   +01
0046: 102F A9 CF               LDAIM STEP   SETUP STEP BY STEP VECTOR
```

```
0047: 1031 8D 7A 1A          STA   NMI
0048: 1034 A9 14             LDAIM STEP   /256
0049: 1036 8D 7B 1A          STA   NMI    +01
0050:
0051: 1039 2C 80 1A  STRTBT  BIT   PAD    WAIT FOR A START BIT
0052: 103C 30 FB             BMI   STRTBT
0053: 103E 20 E0 12          JSR   COMTIM COMPUTE THE START BIT TIME
0054: 1041 4E 5F 1A          LSR   TIMH   DIVIDE BY 2
0055: 1044 6E 5E 1A          ROR   TIML
0056: 1047 AD 5E 1A          LDA   TIML
0057: 104A 8D 5C 1A          STA   CNTHL  SAVE HALF START BIT TIME
0058: 104D AD 5F 1A          LDA   TIMH
0059: 1050 8D 5D 1A          STA   CNTHH
0060: 1053 A2 08             LDXIM $08
0061: 1055 20 03 13          JSR   DELHBT
0062: 1058 20 BE 12          JSR   RECD   GET THE REST OF THE CHARACTER
0063: 105B C9 7F             CMPIM $7F    WAS IT THE RUBOUT CHARACTER?
0064: 105D D0 A1             BNE   INITPR
0065:
0066: 105F 20 46 12  LABJUN  JSR   JUNIOR PRINT 'JUNIOR'
0067: 1062 20 E8 11          JSR   CRLF
0068: 1065 A2 FF             LDXIM $FF
0069: 1067 9A                TXS          RESET STACK POINTER WHEN A BREAK OCCURS
0070: 1068 86 F2             STXZ  SPUSER
0071:
0072: 106A 20 59 12  RESALL  JSR   RESPAR RESET PARAMETER BUFFER
0073: 106D 20 68 12          JSR   RESIN  RESET INPUT BUFFER
0074:
0075: 1070 20 AE 12  READCH  JSR   RECCHA WAIT FOR A CHARACTER
0076:
0077: 1073 C9 2B     PLU     CMPIM '+
0078: 1075 D0 09             BNE   MINUS
0079: 1077 20 13 12          JSR   INCPNT INCREMENT CURRENT ADDRESS
0080: 107A 20 F8 11          JSR   PRBUFS OPEN NEXT CELL
0081: 107D 4C 6A 10          JMP   RESALL
0082:
0083: 1080 C9 2D     MINUS   CMPIM '-
0084: 1082 D0 09             BNE   SPACE
0085: 1084 20 1A 12          JSR   DECPNT DECREMENT CURRENT ADDRESS
0086: 1087 20 F8 11          JSR   PRBUFS OPEN PREVIOUS CELL
0087: 108A 4C 6A 10          JMP   RESALL
0088:
0089: 108D C9 20     SPACE   CMPIM ' 
0090: 108F D0 0E             BNE   PNT
0091: 1091 A5 F8             LDAZ  INL    OUTPUT THE ADDRESS
0092: 1093 85 FA             STAZ  POINTL IN THE INPUT BUFFER
0093: 1095 A5 F9             LDAZ  INH
0094: 1097 85 FB             STAZ  POINTH
0095: 1099 20 F8 11          JSR   PRBUFS OPEN CURRENT CELL
0096: 109C 4C 6A 10          JMP   RESALL
0097:
0098: 109F C9 2E     PNT     CMPIM '.
0099: 10A1 D0 0F             BNE   RUN
0100: 10A3 A5 F8             LDAZ  INL
0101: 10A5 A0 00             LDYIM $00
0102: 10A7 91 FA             STAIY POINTL STORE CURRENT DATA BYTE IN MEMORY
0103: 10A9 20 13 12          JSR   INCPNT
0104: 10AC 20 F8 11          JSR   PRBUFS OPEN NEXT CELL
0105: 10AF 4C 6A 10          JMP   RESALL
0106:
0107: 10B2 C9 52     RUN     CMPIM 'R
0108: 10B4 D0 13             BNE   LIST
0109: 10B6 A6 F2             LDXZ  SPUSER START PROGRAM EXECUTION
0110: 10B8 9A                TXS          AT THE CURRENT DISPLAYED ADDRESS
0111: 10B9 A5 FB             LDAZ  POINTH
0112: 10BB 48                PHA
0113: 10BC A5 FA             LDAZ  POINTL
0114: 10BE 48                PHA
0115: 10BF A5 F1             LDAZ  PREG
```

```
0116: 10C1 48              PHA
0117: 10C2 A6 F5           LDXZ  XREG
0118: 10C4 A4 F4           LDYZ  YREG
0119: 10C6 A5 F3           LDAZ  ACC
0120: 10C8 40              RTI
0121:
0122: 10C9 C9 4C     LIST  CMPIM 'L
0123: 10CB D0 52           BNE   PC
0124: 10CD A0 14           LDYIM $14
0125: 10CF 20 D6 11        JSR   MESSY  ACC:
0126: 10D2 A5 F3           LDAZ  ACC
0127: 10D4 20 8F 12        JSR   PRBYT
0128: 10D7 A0 1A           LDYIM $1A
0129: 10D9 20 D6 11        JSR   MESSY  Y  :
0130: 10DC A5 F4           LDAZ  YREG
0131: 10DE 20 8F 12        JSR   PRBYT
0132: 10E1 A0 20           LDYIM $20
0133: 10E3 20 D6 11        JSR   MESSY  X  :
0134: 10E6 A5 F5           LDAZ  XREG
0135: 10E8 20 8F 12        JSR   PRBYT
0136: 10EB A0 26           LDYIM $26
0137: 10ED 20 D6 11        JSR   MESSY  PC :
0138: 10F0 A5 F0           LDAZ  PCH
0139: 10F2 20 8F 12        JSR   PRBYT
0140: 10F5 A5 EF           LDAZ  PCL
0141: 10F7 20 8F 12        JSR   PRBYT
0142: 10FA A0 2C           LDYIM $2C
0143: 10FC 20 D6 11        JSR   MESSY  SP :
0144: 10FF A9 01           LDAIM $01
0145: 1101 20 8F 12        JSR   PRBYT
0146: 1104 A5 F2           LDAZ  SPUSER
0147: 1106 20 8F 12        JSR   PRBYT
0148: 1109 A0 32           LDYIM $32
0149: 110B 20 D6 11        JSR   MESSY  PR :
0150: 110E 20 28 12        JSR   SHOWPR PRINT OUT FLAGS
0151: 1111 A0 38           LDYIM $38
0152: 1113 20 D6 11        JSR   MESSY  NV BDIZC
0153: 1116 20 F3 11        JSR   PRSP
0154: 1119 4C 6A 10        JMP   RESALL
0155:
0156: 111C 4C B0 11  CONTIN JMP   GETTAP
0157:
0158:
0159: 111F C9 50     PC    CMPIM 'P
0160: 1121 D0 0E           BNE   MATRIX
0161: 1123 A5 EF           LDAZ  PCL    RESTORE LAST PROGRAM COUNTER
0162: 1125 85 FA           STAZ  POINTL
0163: 1127 A5 F0           LDAZ  PCH
0164: 1129 85 FB           STAZ  POINTH
0165: 112B 20 F8 11        JSR   PRBUFS OPEN CURRENT CELL
0166: 112E 4C 6A 10        JMP   RESALL
0167:
0168: 1131 C9 4D     MATRIX CMPIM 'M
0169: 1133 D0 E7           BNE   CONTIN
0170: 1135 A0 52           LDYIM $52
0171: 1137 20 D6 11        JSR   MESSY  HEXDUMP:
0172: 113A 20 87 13        JSR   INPAR  READ PARAMETERS
0173: 113D 10 03           BPL   MATF
0174:
0175: 113F 4C 5F 10  MATD  JMP   LABJUN RETURN, IF INVALID CHARACTER
0176:
0177: 1142 38        MATF  SEC          VALID INPUT PARAMETERS?
0178: 1143 AD 65 1A        LDA   PARBL
0179: 1146 ED 63 1A        SBC   PARAL  PARA < PARB?
0180: 1149 AD 66 1A        LDA   PARBH
0181: 114C ED 64 1A        SBC   PARAH
0182: 114F 90 EE           BCC   MATD
0183: 1151 20 E8 11        JSR   CRLF
0184: 1154 A2 06           LDXIM $06
```

```
0185:
0186: 1156 20 F3 11  MATG   JSR   PRSP
0187: 1159 CA               DEX
0188: 115A D0 FA            BNE   MATG
0189: 115C A0 00            LDYIM $00     RESET COLUMN COUNTER
0190:
0191: 115E 98        MATH   TYA
0192: 115F 20 9B 12         JSR   PRNIBL OUTPUT COLUMNS
0193: 1162 20 F3 11         JSR   PRSP
0194: 1165 20 F3 11         JSR   PRSP
0195: 1168 C8               INY
0196: 1169 C0 10            CPYIM $10     PRINT COLUMS 0...F
0197: 116B D0 F1            BNE   MATH
0198: 116D AD 63 1A         LDA   PARAL
0199: 1170 85 FA            STAZ  POINTL SETUP MATRIX POINTER
0200: 1172 AD 64 1A         LDA   PARAH
0201: 1175 85 FB            STAZ  POINTH
0202:
0203: 1177 20 E8 11  MATJ   JSR   CRLF
0204: 117A A2 10            LDXIM $10
0205: 117C A5 FB            LDAZ  POINTH
0206: 117E 20 8F 12         JSR   PRBYT  OUTPUT CURRENT MATRIX ADDRESS
0207: 1181 A5 FA            LDAZ  POINTL
0208: 1183 20 8F 12         JSR   PRBYT
0209: 1186 A0 17            LDYIM $17
0210: 1188 20 D9 11         JSR   ME
0211:
0212: 118B 38        MATK   SEC
0213: 118C AD 65 1A         LDA   PARBL  HEXDUMP FINISHED?
0214: 118F E5 FA            SBCZ  POINTL
0215: 1191 AD 66 1A         LDA   PARBH
0216: 1194 E5 FB            SBCZ  POINTH
0217: 1196 B0 06            BCS   MATL
0218: 1198 20 E8 11         JSR   CRLF
0219: 119B 4C 5F 10         JMP   LABJUN HEXDUMP IS FINISHED
0220:
0221: 119E A0 00     MATL   LDYIM $00
0222: 11A0 B1 FA            LDAIY POINTL FETCH CURRENT DATA BYTE
0223: 11A2 20 8F 12         JSR   PRBYT  OUTPUT CURRENT DATA BYTE
0224: 11A5 20 F3 11         JSR   PRSP
0225: 11A8 20 13 12         JSR   INCPNT
0226: 11AB CA               DEX
0227: 11AC D0 DD            BNE   MATK
0228: 11AE F0 C7            BEQ   MATJ
0229:
0230: 11B0 C9 47     GETTAP CMPIM 'G
0231: 11B2 D0 0B            BNE   SAVID
0232: 11B4 20 8A 14         JSR   GETID  READ DATA FROM TAPE SPEC. BY ID
0233: 11B7 30 03            BMI   GETERR ILLEGAL CHARACTER?
0234: 11B9 4C 6A 10         JMP   RESALL NORMAL EXIT
0235:
0236: 11BC 4C 5F 10  GETERR JMP   LABJUN
0237:
0238: 11BF C9 53     SAVID  CMPIM 'S
0239: 11C1 D0 08            BNE   VALNUM NO COMMAND KEY WAS DEPRESSED
0240: 11C3 20 3B 14         JSR   SAID   STORE DATA ON TAPE SPECIFIED BY THE PARAMETERS
0241: 11C6 30 F4            BMI   GETERR ILLEGAL PARAMETER WAS ENTERED
0242: 11C8 4C 6A 10         JMP   RESALL
0243:
0244: 11CB 20 6F 12  VALNUM JSR   HEXNUM INPUT DATA TO BUFFER
0245: 11CE D0 03            BNE   VNA
0246: 11D0 4C 70 10         JMP   READCH
0247:
0248: 11D3 4C 6A 10  VNA    JMP   RESALL
0249:
0250:                ****************************************
0251:                *** SUBROUTINES OF THE PRINTER MONITOR ***
0252:                ****************************************
0253:
```

```
0254:
0255:                    MESSY PRINTS A MESSAGE, POINTED BY Y REGISTER
0256:
0257: 11D6 20 E8 11  MESSY  JSR   CRLF    PRINT A CR&LF
0258:
0259: 11D9 B9 BD 13  ME     LDAY  MESS    LOAD CHARACTERS
0260: 11DC C9 03             CMPIM $03    ETX CHARACTER ?
0261: 11DE F0 07             BEQ   MESEND
0262: 11E0 20 34 13          JSR   PRCHA  CHARACTER TO TTY
0263: 11E3 C8                INY
0264: 11E4 4C D9 11          JMP   ME
0265:
0266: 11E7 60        MESEND RTS
0267:
0268:
0269:                    CRLF PRINT CARRIAGE RETURN & LINE FEED
0270:                    PRSP PRINT A SPACE
0271:
0272: 11E8 A9 0D     CRLF   LDAIM $0D
0273: 11EA 20 34 13         JSR   PRCHA  OUTPUT CR
0274: 11ED A9 0A            LDAIM $0A
0275:
0276: 11EF 20 34 13  CLEND  JSR   PRCHA  OUTPUT LF
0277: 11F2 60               RTS
0278:
0279: 11F3 A9 20     PRSP   LDAIM $20
0280: 11F5 4C EF 11         JMP   CLEND  OUTPUT A SPACE
0281:
0282:
0283:                    PRBUFS OUTPUT ADDRESS AND DATA
0284:
0285: 11F8 20 E8 11  PRBUFS JSR   CRLF
0286: 11FB A5 FB            LDAZ  POINTH
0287: 11FD 20 8F 12         JSR   PRBYT  OUTPUT HIGH ORDER ADDR
0288: 1200 A5 FA            LDAZ  POINTL
0289: 1202 20 8F 12         JSR   PRBYT  OUTPUT LOW ORDER ADDR
0290: 1205 20 F3 11         JSR   PRSP
0291: 1208 A0 00            LDYIM $00
0292: 120A B1 FA            LDAIY POINTL FETCH DATA FROM MEMORY
0293: 120C 20 8F 12         JSR   PRBYT
0294: 120F 20 F3 11         JSR   PRSP
0295: 1212 60               RTS
0296:
0297:
0298:                    INCPNT INCRMENT ADDR POINTER BY ONE
0299:
0300: 1213 E6 FA     INCPNT INCZ  POINTL
0301: 1215 D0 02            BNE   IP
0302: 1217 E6 FB            INCZ  POINTH
0303:
0304: 1219 60        IP     RTS
0305:
0306:
0307:                    DECPNT DECREMENT ADDR POINTER BY ONE
0308:
0309: 121A 38        DECPNT SEC
0310: 121B A5 FA            LDAZ  POINTL 16 BIT SUBTRACTION
0311: 121D E9 01            SBCIM $01
0312: 121F 85 FA            STAZ  POINTL
0313: 1221 A5 FB            LDAZ  POINTH
0314: 1223 E9 00            SBCIM $00
0315: 1225 85 FB            STAZ  POINTH
0316: 1227 60               RTS
0317:
0318:
0319:                    SHOWPR SHOW THE CONTENTS OF THE P REGISTER
0320:
0321: 1228 A5 F1     SHOWPR LDAZ  PREG
0322: 122A 8D 67 1A         STA   PRTEMP
```

```
0323: 122D A2 08           LDXIM $08    BIT COUNTER
0324:
0325: 122F 0E 67 1A  SPRA  ASL   PRTEMP SHIFT OUT THE BITS
0326: 1232 90 09           BCC   SPRB   IS IT A 0 OR 1 ?
0327: 1234 A9 01           LDAIM $01
0328: 1236 20 9B 12        JSR   PRNIBL PRINT A '1'
0329: 1239 CA              DEX
0330: 123A D0 F3           BNE   SPRA   ALL BITS PRINTED ?
0331: 123C 60              RTS
0332:
0333: 123D A9 00     SPRB  LDAIM $00
0334: 123F 20 9B 12        JSR   PRNIBL PRINT A '0'
0335: 1242 CA              DEX
0336: 1243 D0 EA           BNE   SPRA   ALL BITS PRINTED ?
0337: 1245 60              RTS
0338:
0339:
0340:                JUNIOR PRINT 'JUNIOR'
0341:                EDITOR PRINT 'EDITOR'
0342:                ASSEM  PRINT 'ASSEMBLER'
0343:
0344: 1246 A0 00     JUNIOR LDYIM $00
0345:
0346: 1248 20 D6 11  JUN    JSR   MESSY
0347: 124B 20 E8 11        JSR   CRLF
0348: 124E 60              RTS
0349:
0350: 124F A0 07     EDITOR LDYIM $07
0351: 1251 4C 48 12        JMP   JUN
0352:
0353: 1254 A0 0E     ASSEM  LDYIM $0E
0354: 1256 4C 48 12        JMP   JUN
0355:
0356:
0357:                RESET SUBROUTINES
0358:
0359: 1259 A0 00     RESPAR LDYIM $00
0360: 125B 8C 63 1A        STY   PARAL
0361: 125E 8C 64 1A        STY   PARAH
0362: 1261 8C 65 1A        STY   PARBL
0363: 1264 8C 66 1A        STY   PARBH
0364: 1267 60              RTS
0365:
0366: 1268 A0 00     RESIN  LDYIM $00
0367: 126A 84 F8           STYZ  INL
0368: 126C 84 F9           STYZ  INH
0369: 126E 60              RTS
0370:
0371:                HEXNUM >CONVERT AN ASCII CHAR. TO A HEX NIBBLE
0372:                       >SHIFT HEX DATA NIBBLE INTO BUFFER
0373:                       >PRINT 'WHAT?' IF CHARACTER WAS NOT VALID
0374:                       >RETURN WITH Z=1, IF VALID CHARACTER
0375:                       >RETURN WITH N=1, IF NOT VALID CHARACTER
0376:
0377: 126F 20 1E 14  HEXNUM JSR   ASHETT ASCII HEX CONVERSION
0378: 1272 30 10           BMI   HNUB   NOT VALID ?
0379: 1274 A2 04           LDXIM $04    SET NIBBLE COUNTER
0380:
0381: 1276 06 F8     HNUA   ASLZ  INL
0382: 1278 26 F9            ROLZ  INH
0383: 127A CA               DEX
0384: 127B D0 F9            BNE   HNUA
0385: 127D 05 F8            ORAZ  INL    NIBBLE TO INPUT BUFFER
0386: 127F 85 F8            STAZ  INL
0387: 1281 A0 00            LDYIM $00    SET Z FLAG
0388: 1283 60               RTS
0389:
0390: 1284 A0 46     HNUB   LDYIM $46
0391: 1286 20 D6 11         JSR   MESSY  'WHAT?'
```

```
0392: 1289 20 E8 11          JSR   CRLF
0393: 128C A0 FF             LDYIM $FF     SET N FLAG
0394: 128E 60                RTS
0395:
0396:
0397:                  PRBYT >CONVERT AN BYTE STORED IN ACCU TO
0398:                        TWO ASCII CHARACTERS AND PRINT THEM
0399:
0400: 128F 48          PRBYT  PHA           SAVE BYTE
0401: 1290 4A                 LSRA          GET UPPER NIBBLE
0402: 1291 4A                 LSRA
0403: 1292 4A                 LSRA
0404: 1293 4A                 LSRA
0405: 1294 20 A4 12           JSR   NIBASC NIBBLE TO ASCII CONVERSION
0406: 1297 20 34 13           JSR   PRCHA  PRINT UPPER NIBBLE
0407: 129A 68                 PLA          GET BYTE AGAIN
0408:
0409: 129B 29 0F       PRNIBL ANDIM $0F     GET LOWER NIBBLE
0410: 129D 20 A4 12           JSR   NIBASC
0411: 12A0 20 34 13           JSR   PRCHA  PRINT LOWER NIBBLE
0412: 12A3 60                 RTS
0413:
0414:
0415:                  NIBASC >CONVERT A HEX DATA NIBBLE TO AN ASCII CHARACTER
0416:
0417: 12A4 C9 0A       NIBASC CMPIM $0A     NUMBER OR LETTER ?
0418: 12A6 18                 CLC
0419: 12A7 30 02              BMI   NA
0420: 12A9 69 07              ADCIM $07
0421:
0422: 12AB 69 30       NA     ADCIM $30
0423: 12AD 60                 RTS
0424:
0425:
0426:                  RECCHA >RECEIVE 1 ASCII CHARACTER FROM PRINTER
0427:                        >RETURN WITH ASCII CHARACTER IN ACCU
0428:                        >SAVE X REGISTER
0429:
0430: 12AE 2C 80 1A    RECCHA BIT   PAD     WAIT FOR START BIT
0431: 12B1 30 FB              BMI   RECCHA
0432: 12B3 8E 61 1A           STX   TEMPB   SAVE INDEX X
0433: 12B6 A2 08              LDXIM $08     BIT COUNTER IS 8
0434: 12B8 20 03 13           JSR   DELHBT  DELAY HALF BIT TIME
0435:
0436: 12BB 20 12 13    RECA   JSR   DELBIT  DELAY ONE BIT TIME
0437:
0438: 12BE 2C 80 1A    RECD   BIT   PAD     ONE/ZERO CHECK
0439: 12C1 10 0A              BPL   RECB    BRANCH IF ZERO
0440: 12C3 38                 SEC           BIT IS '1'
0441: 12C4 6E 62 1A           ROR   CHA     ROTATE CARRY INTO CHARACTER
0442: 12C7 CA                 DEX           SET UP FOR NEXT BIT
0443: 12C8 D0 F1              BNE   RECA    ALL BITS READ?
0444: 12CA 4C D4 12           JMP   RECC
0445:
0446: 12CD 18          RECB   CLC           BIT IS '0'
0447: 12CE 6E 62 1A           ROR   CHA
0448: 12D1 CA                 DEX
0449: 12D2 D0 E7              BNE   RECA
0450:
0451: 12D4 20 12 13    RECC   JSR   DELBIT  WAIT FOR STOP BIT TIME
0452: 12D7 AD 62 1A           LDA   CHA
0453: 12DA 29 7F              ANDIM $7F     BIT 7 MUST BE ZERO
0454: 12DC AE 61 1A           LDX   TEMPB   RESTORE INDEX X
0455: 12DF 60                 RTS
0456:
0457:
0458:                  COMTIM >COMPUTE BIT TIME
0459:
0460: 12E0 18          COMTIM CLC
```

```
0461: 12E1 AD 5A 1A          LDA   CNTL    16 BIT ADD
0462: 12E4 69 01             ADCIM $01
0463: 12E6 8D 5A 1A          STA   CNTL
0464: 12E9 AD 5B 1A          LDA   CNTH
0465: 12EC 69 00             ADCIM $00
0466: 12EE 8D 5B 1A          STA   CNTH
0467: 12F1 2C 80 1A          BIT   PAD     START BIT FINISHED ?
0468: 12F4 10 EA             BPL   COMTIM
0469: 12F6 AD 5A 1A          LDA   CNTL    SET UP FOR
0470: 12F9 8D 5E 1A          STA   TIML    HALF BIT TIME COMPUTATION
0471: 12FC AD 5B 1A          LDA   CNTH
0472: 12FF 8D 5F 1A          STA   TIMH
0473: 1302 60                RTS
0474:
0475:
0476:                  DELBIT >DELAY 1 BIT TIME
0477:                  DELHBT >DELAY 1/2 BIT TIME
0478:
0479: 1303 AD 5C 1A  DELHBT LDA   CNTHL   FETCH 1/2 BIT TIME
0480: 1306 8D 5E 1A          STA   TIML
0481: 1309 AD 5D 1A          LDA   CNTHH
0482: 130C 8D 5F 1A          STA   TIMH
0483: 130F 4C 1E 13          JMP   CNTDN   START WITH BIT COUNT DOWN
0484:
0485: 1312 AD 5A 1A  DELBIT LDA   CNTL    FETCH 1 BIT TIME
0486: 1315 8D 5E 1A          STA   TIML
0487: 1318 AD 5B 1A          LDA   CNTH
0488: 131B 8D 5F 1A          STA   TIMH
0489:
0490: 131E 38        CNTDN   SEC
0491: 131F AD 5E 1A          LDA   TIML    16 BIT SUBTRACTION
0492: 1322 E9 01             SBCIM $01     COUNT DOWN
0493: 1324 8D 5E 1A          STA   TIML
0494: 1327 AD 5F 1A          LDA   TIMH
0495: 132A E9 00             SBCIM $00
0496: 132C 8D 5F 1A          STA   TIMH
0497: 132F EA                NOP           EQUALIZE 4 MICRO SEC
0498: 1330 EA                NOP
0499: 1331 B0 EB             BCS   CNTDN   COUNT DOWN FINISHED ?
0500: 1333 60                RTS
0501:
0502:                  PRCHA >TRANSMIT AN ASCII CHARACTER STORED IN
0503:                         ACCU TO PRINTER
0504:                        >SAVE INDEX X
0505:
0506: 1334 8E 60 1A  PRCHA   STX   TEMPA   SAVE INDEX X
0507: 1337 8D 62 1A          STA   CHA
0508: 133A AD 82 1A          LDA   PBD
0509: 133D 29 FE             ANDIM $FE     TRNSMIT START BIT
0510: 133F 8D 82 1A          STA   PBD
0511: 1342 20 12 13          JSR   DELBIT  DELAY OF START BIT
0512: 1345 A2 07             LDXIM $07     SET UP FOR 7 DATA BITS
0513:
0514: 1347 4E 62 1A  PRA     LSR   CHA     SHIFT OUT CHARACTER
0515: 134A 90 30             BCC   PRC     BRANCH IF '0'
0516: 134C AD 82 1A          LDA   PBD
0517: 134F 09 01             ORAIM $01     OUTPUT A LOG '1'
0518: 1351 8D 82 1A          STA   PBD
0519:
0520: 1354 20 12 13  PRB     JSR   DELBIT  DELAY 1 BIT TIME
0521: 1357 CA                DEX           SET UP FOR NEXT BIT
0522: 1358 D0 ED             BNE   PRA     ALL BITS TRANSMITTED ?
0523: 135A AE 59 1A          LDX   STPBIT  X := AMOUNT OF STOP BITS + 1
0524:
0525: 135D AD 82 1A  PRD     LDA   PBD
0526: 1360 09 01             ORAIM $01     FIRST NONE PARITY
0527: 1362 8D 82 1A          STA   PBD     AND THEN 1 STOP BIT
0528: 1365 20 12 13          JSR   DELBIT
0529: 1368 CA                DEX
```

```
0530: 1369 D0 F2            BNE   PRD
0531: 136B 2C 80 1A         BIT   PAD     TEST FOR BREAK
0532: 136E 10 04            BPL   BRKTST
0533: 1370 AE 60 1A         LDX   TEMPA   RESTORE INDEX X
0534: 1373 60               RTS
0535:
0536: 1374 2C 80 1A  BRKTST BIT   PAD     KEY RELEASED ?
0537: 1377 10 FB            BPL   BRKTST
0538: 1379 6C 7C 1A         JMI   BRKT    JUMP TO AN USER SELECTABLE VECTOR
0539:
0540: 137C AD 82 1A  PRC    LDA   PBD
0541: 137F 29 FE            ANDIM $FE     OUTPUT A LOG '0'
0542: 1381 8D 82 1A         STA   PBD
0543: 1384 4C 54 13         JMP   PRB
0544:
0545:                INPAR >PARAMETER INPUT OF MATRIX
0546:
0547: 1387 20 AE 12  INPAR  JSR   RECCHA  WAIT FOR A CHARACTER
0548: 138A C9 2C            CMPIM ',      IS IT A COLON ?
0549: 138C F0 07            BEQ   IPA
0550: 138E 20 6F 12         JSR   HEXNUM  FILLUP INPUT BUFFER
0551: 1391 30 29            BMI   IPD     RETURN, IF NOT VALID
0552: 1393 F0 F2            BEQ   INPAR   ELSE CONTINUE
0553:
0554: 1395 A5 F8     IPA    LDAZ  INL     INPUT TO PARAMETER BUFFER
0555: 1397 8D 63 1A         STA   PARAL
0556: 139A A5 F9            LDAZ  INH
0557: 139C 8D 64 1A         STA   PARAH
0558: 139F 20 68 12         JSR   RESIN
0559:
0560: 13A2 20 AE 12  IPB    JSR   RECCHA  WAIT FOR A CHARACTER
0561: 13A5 C9 0D            CMPIM $0D     WAS IT A CARRIAGE RETURN ?
0562: 13A7 F0 07            BEQ   IPC
0563: 13A9 20 6F 12         JSR   HEXNUM
0564: 13AC 30 0E            BMI   IPD     VALID CHARACTER ?
0565: 13AE F0 F2            BEQ   IPB
0566:
0567: 13B0 A5 F9     IPC    LDAZ  INH     INPUT TO PARAMETER BUFFER
0568: 13B2 8D 66 1A         STA   PARBH
0569: 13B5 A5 F8            LDAZ  INL
0570: 13B7 8D 65 1A         STA   PARBL
0571: 13BA A0 00            LDYIM $00
0572:
0573: 13BC 60        IPD    RTS
0574:
0575:
0576:                STRING LOOKUP TABLE
0577:
0578: 13BD 4A        MESS   =     'J       Y = 00
0579: 13BE 55               =     'U
0580: 13BF 4E               =     'N
0581: 13C0 49               =     'I
0582: 13C1 4F               =     'O
0583: 13C2 52               =     'R
0584: 13C3 03               =     $03
0585: 13C4 45               =     'E       Y = 07
0586: 13C5 44               =     'D
0587: 13C6 49               =     'I
0588: 13C7 54               =     'T
0589: 13C8 4F               =     'O
0590: 13C9 52               =     'R
0591: 13CA 03               =     $03
0592: 13CB 41               =     'A       Y = 0E
0593: 13CC 53               =     'S
0594: 13CD 53               =     'S
0595: 13CE 45               =     'E
0596: 13CF 4D               =     'M
0597: 13D0 03               =     $03
0598: 13D1 41               =     'A       Y = 14
```

```
0599: 13D2 43          =     'C
0600: 13D3 43          =     'C
0601: 13D4 3A          =     ':      Y = 17
0602: 13D5 20          =     '
0603: 13D6 03          =     $03
0604: 13D7 59          =     'Y      Y = 1A
0605: 13D8 20          =     '
0606: 13D9 20          =     '
0607: 13DA 3A          =     ':
0608: 13DB 20          =     '
0609: 13DC 03          =     $03
0610: 13DD 58          =     'X      Y = 20
0611: 13DE 20          =     '
0612: 13DF 20          =     '
0613: 13E0 3A          =     ':
0614: 13E1 20          =     '
0615: 13E2 03          =     $03
0616: 13E3 50          =     'P      Y = 26
0617: 13E4 43          =     'C
0618: 13E5 20          =     '
0619: 13E6 3A          =     ':
0620: 13E7 20          =     '
0621: 13E8 03          =     $03
0622: 13E9 53          =     'S      Y = 2C
0623: 13EA 50          =     'P
0624: 13EB 20          =     '
0625: 13EC 3A          =     ':
0626: 13ED 20          =     '
0627: 13EE 03          =     $03
0628: 13EF 50          =     'P      Y = 32
0629: 13F0 52          =     'R
0630: 13F1 20          =     '
0631: 13F2 3A          =     ':
0632: 13F3 20          =     '
0633: 13F4 03          =     $03
0634: 13F5 20          =     '       Y = 38
0635: 13F6 20          =     '
0636: 13F7 20          =     '
0637: 13F8 20          =     '
0638: 13F9 20          =     '
0639: 13FA 4E          =     'N
0640: 13FB 56          =     'V
0641: 13FC 20          =     '
0642: 13FD 42          =     'B
0643: 13FE 44          =     'D
0644: 13FF 49          =     'I
0645: 1400 5A          =     'Z
0646: 1401 43          =     'C
0647: 1402 03          =     $03
0648: 1403 57          =     'W      Y = 46
0649: 1404 48          =     'H
0650: 1405 41          =     'A
0651: 1406 54          =     'T
0652: 1407 3F          =     '?
0653: 1408 03          =     $03
0654: 1409 52          =     'R      Y = 4C
0655: 140A 45          =     'E
0656: 140B 41          =     'A
0657: 140C 44          =     'D
0658: 140D 59          =     'Y
0659: 140E 03          =     $03
0660: 140F 48          =     'H      Y = 52
0661: 1410 45          =     'E
0662: 1411 58          =     'X
0663: 1412 44          =     'D
0664: 1413 55          =     'U
0665: 1414 4D          =     'M
0666: 1415 50          =     'P
0667: 1416 3A          =     ':
```

```
0668: 1417 20                =       '
0669: 1418 03                =       $03
0670: 1419 53                =       'S      Y = 5C
0671: 141A 41                =       'A
0672: 141B 3A                =       ':
0673: 141C 20                =       '
0674: 141D 03                =       $03
0675:
0676:                    ASHETT = ASCHEX
0677:
0678:
0679:                    CONVERT AN ASCII CHARACTER TO A HEX DATA NIBBLE.
0680:
0681:                    1) RETURN WITH CONVERTED HEX NUMBER IN ACCU
0682:                    2) N = 1, IF NOT VALID HEX NUMBER
0683:                    3) Z = 1, IF VALID HEX NUMBER
0684:
0685: 141E C9 30      ASHETT CMPIM $30     IGNORE 00...2F
0686: 1420 30 0C             BMI   NOTVAT
0687: 1422 C9 3A             CMPIM $3A
0688: 1424 30 0B             BMI   VALIT
0689: 1426 C9 41             CMPIM $41     IGNORE 3A...40
0690: 1428 30 04             BMI   NOTVAT
0691: 142A C9 47             CMPIM $47     IGNORE 47...7F
0692: 142C 30 03             BMI   VALIT
0693:
0694: 142E A0 FF      NOTVAT LDYIM $FF     SET N-FLAG
0695: 1430 60                RTS           ERROR EXIT
0696:
0697: 1431 C9 40      VALIT  CMPIM $40     ASCII HEX CONVERSION
0698: 1433 30 03             BMI   VALT
0699: 1435 18                CLC
0700: 1436 69 09             ADCIM $09
0701:
0702: 1438 29 0F      VALT   ANDIM $0F
0703: 143A 60                RTS
0704:
0705: 143B 20 AE 12   SAID   JSR   RECCHA WAIT FOR A CHARACTER
0706: 143E C9 2C             CMPIM ','
0707: 1440 F0 07             BEQ   SIC     IT WAS A DELIMITER
0708: 1442 20 6F 12          JSR   HEXNUM READ PARAMETER = ID
0709: 1445 30 3F             BMI   SIB
0710: 1447 F0 F2             BEQ   SAID
0711:
0712: 1449 A5 F8      SIC    LDAZ  INL     SAVE ID
0713: 144B 8D 79 1A          STA   ID
0714: 144E C9 00             CMPIM $00     ID = 00 & FF ARE NOT VALID
0715: 1450 F0 35             BEQ   SIA
0716: 1452 C9 FF             CMPIM $FF
0717: 1454 F0 31             BEQ   SIA
0718: 1456 20 68 12          JSR   RESIN   RESET INPUT BUFFER
0719: 1459 20 87 13          JSR   INPAR   READ SA AND EA
0720: 145C 30 28             BMI   SIB     NOT VALID CHARACTER
0721: 145E AD 63 1A          LDA   PARAL   SAVE ALL PARAMETERS
0722: 1461 8D 70 1A          STA   SAL
0723: 1464 AD 64 1A          LDA   PARAH
0724: 1467 8D 71 1A          STA   SAH
0725: 146A AD 65 1A          LDA   PARBL
0726: 146D 8D 72 1A          STA   EAL
0727: 1470 AD 66 1A          LDA   PARBH
0728: 1473 8D 73 1A          STA   EAH
0729: 1476 20 DF 09          JSR   DUMP    STORE DATA ON TAPE
0730: 1479 20 BC 14          JSR   RESTTY I/O RESET
0731: 147C A0 4C             LDYIM $4C     'READY'
0732: 147E 20 D6 11          JSR   MESSY
0733: 1481 20 E8 11          JSR   CRLF
0734: 1484 A0 00             LDYIM $00     NORMAL EXIT
0735:
0736: 1486 60          SIB    RTS
```

```
0737:
0738: 1487 A0 FF      SIA     LDYIM $FF     ERROR EXIT
0739: 1489 60                 RTS
0740:
0741:
0742: 148A 20 A2 13   GETID   JSR   IPB      READ ID
0743: 148D 30 17              BMI   GA       NOT VALID PARAMETER
0744: 148F 8D 79 1A           STA   ID       SAVE ID
0745: 1492 C9 FF              CMPIM $FF      SPECIAL ID ?
0746: 1494 F0 11              BEQ   IDPAR
0747:
0748: 1496 20 02 0B   GB      JSR   RDTAPE  READ DATA FROM TAPE
0749: 1499 20 BC 14           JSR   RESTTY  I/O RESET
0750: 149C A0 4C              LDYIM $4C      'READY'
0751: 149E 20 D6 11           JSR   MESSY
0752: 14A1 20 E8 11           JSR   CRLF
0753: 14A4 A0 00              LDYIM $00      NORMAL EXIT
0754:
0755: 14A6 60         GA      RTS
0756:
0757: 14A7 A0 5C      IDPAR   LDYIM $5C      'SA: '
0758: 14A9 20 D6 11           JSR   MESSY
0759: 14AC 20 EB 14           JSR   IPBRES  READ A SPECIAL ID
0760: 14AF 30 F5              BMI   GA       NOT VALID PARAMETER
0761: 14B1 8D 70 1A           STA   SAL      SAVE START ADDRESS
0762: 14B4 A5 F9              LDAZ  INH
0763: 14B6 8D 71 1A           STA   SAH
0764: 14B9 4C 96 14           JMP   GB
0765:
0766:
0767:
0768: 14BC A9 67      RESTTY  LDAIM $67
0769: 14BE 8D 82 1A           STA   PBD
0770: 14C1 A9 00              LDAIM $00
0771: 14C3 8D 80 1A           STA   PAD
0772: 14C6 A9 7F              LDAIM $7F
0773: 14C8 8D 81 1A           STA   PADD
0774: 14CB 8D 83 1A           STA   PBDD
0775: 14CE 60                 RTS
0776:                 ************************************************
0777:                 *** STEP BY STEP PROGRAM OF THE PRINTER MONITOR ***
0778:                 ************************************************
0779:
0780:
0781:                 > THE NMI VECTOR FOR STEP BY STEP IS SET AUTOMATICALLY
0782:                 > STEP SWITCH: ON POSITION
0783:                 > K4 AND K5 DISABLE THE SYNC SIGNAL OF THE PROCESSOR
0784:                 > A HARDWARE MODIFICATION IS REQIRED (SEE BOOK 3)
0785:
0786:
0787: 14CF 85 F3    STEP    STAZ  ACC      SAVE ACCU
0788: 14D1 68                 PLA          GET PREG
0789: 14D2 85 F1              STAZ  PREG
0790: 14D4 68                 PLA          GET PCL
0791: 14D5 85 EF              STAZ  PCL
0792: 14D7 85 FA              STAZ  POINTL PC OF THE NEXT INSTRUCTION
0793: 14D9 68                 PLA          GET PCH
0794: 14DA 85 F0              STAZ  PCH
0795: 14DC 85 FB              STAZ  POINTH
0796: 14DE 84 F4              STYZ  YREG
0797: 14E0 86 F5              STXZ  XREG
0798: 14E2 BA                 TSX          GET OLD STACK POINTER
0799: 14E3 86 F2              STXZ  SPUSER AND SAVE IT
0800: 14E5 20 F8 11           JSR   PRBUFS OPEN NEXT CELL
0801: 14E8 4C 6A 10           JMP   RESALL BACK TO MONITOR
0802:
0803:
0804:
```

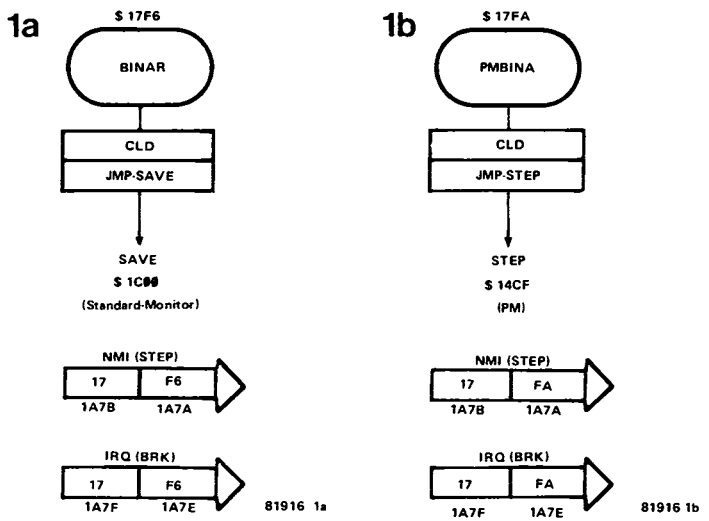
```
0805:                 *** SPECIAL ID ***
0806:
0807: 14EB A9 00      IPBRES LDAIM $00
0808: 14ED 85 F8             STAZ  INL     RESET INPUT BUFFER
0809: 14EF 85 F9             STAZ  INH
0810: 14F1 4C A2 13          JMP   IPB     CONTINUE
```

SYMBOL TABLE 3000 3594

ACC	00F3	ADJPNT	0C72	ASCHEX	0C19	ASHETT	141E
ASSEM	1254	BEGADH	00E3	BEGADL	00E2	BEGIN	1ED3
BITS	1A75	BRKT	1A7C	BRKTST	1374	BTWEEN	0BE8
BYTE	1A6A	CENDH	00E9	CENDL	00E8	CHAR	1A6B
CHARVU	0C5D	CHA	1A62	CHECK	0B8A	CHKH	1A6F
CHKID	0B9E	CHKL	1A6E	CHKSUM	0C4B	CLEND	11EF
CNTA	1AF4	CNTC	1AF6	CNTDN	131E	CNTH	1A5B
CNTHH	1A5D	CNTHL	1A5C	CNTL	1A5A	COLDST	1CB5
COMPNT	0949	COMTIM	12E0	CONTIN	111C	CRLF	11E8
DATATR	0A40	DAT	089C	DECPNT	121A	DELBIT	1312
DELHBT	1303	DELY	0927	DISCNT	1A68	DUMP	09DF
DUMPT	09F3	EAH	1A73	EAL	1A72	EDITOR	124F
ENDADH	00E5	ENDADL	00E4	FILES	0873	FILMEM	0B70
FIRST	1A76	FMA	0B84	GA	14A6	GANG	1A78
GB	1496	GETA	0800	GETB	084E	GETERR	11BC
GETID	148A	GETKEY	1DF9	GETTAP	11B0	GET	0840
HEXDAT	0A6A	HEXNUM	126F	HIGH	0AC8	HIGHER	1A6C
HIG	0ACB	HNUA	1276	HNUB	1284	ID	1A79
IDPAR	14A7	INCPNT	1213	INH	00F9	INITPR	1000
INL	00F8	INPAR	1387	IP	1219	IPA	1395
IPBRES	14EB	IPB	13A2	IPC	13B0	IPD	13BC
JUNIOR	1246	JUN	1248	LABJUN	105F	LDAINH	1DA7
LIST	10C9	LO	0AE8	LOWER	1A6D	LOW	0AE5
MATD	113F	MATF	1142	MATG	1156	MATH	115E
MATJ	1177	MATK	118B	MATL	119E	MATRIX	1131
ME	11D9	MESEND	11E7	MESS	13BD	MESSY	11D6
MINUS	1080	NA	12AB	NIBASC	12A4	NIBOUT	0A9A
NIB	0AA1	NMI	1A7A	NOTVAL	0C29	NOTVAT	142E
ONE	0AA8	OUTBT	0A8B	OUTBTC	0A7A	OUTCH	0AA3
PADD	1A81	PAD	1A80	PARAH	1A64	PARAL	1A63
PARBH	1A66	PARBL	1A65	PBDD	1A83	PBD	1A82
PC	111F	PCH	00F0	PCL	00EF	PLUS	085E
PLU	1073	PNT	109F	POINTH	00FB	POINTL	00FA
PRA	1347	PRBUFS	11F8	PRBYT	128F	PRB	1354
PRCHA	1334	PRC	137C	PRD	135D	PREG	00F1
PRNIBL	129B	PRSP	11F3	PRTEMP	1A67	RBA	0BFA
RBB	0BFB	RDBA	0BD1	RDBB	0BD4	RDBC	0BE4
RDBIT	0BC2	RDBYT	0BF3	RDCH	0C36	RDFLAG	1AD5
RDSA	0B60	RDTAPE	0B02	RDTDIS	1AD4	READ	0C38
READCH	1070	RECA	12BB	RECB	12CD	RECC	12D4
RECCHA	12AE	RECD	12BE	RESALL	106A	RESET	1C1D
RESIN	1268	RESPAR	1259	RESTTY	14BC	RUN	10B2
SAH	1A71	SAID	143B	SAL	1A70	SAVE	0852
SAVID	11BF	SBEGH	0989	SBEGL	0997	SEAH	096B
SEAL	097A	SECOND	1A77	SENDH	09A5	SENDL	09B3
SHIFT	08A3	SHOWPR	1228	SIA	1487	SIB	1486
SIC	1449	SID	093B	SPACE	108D	SPRA	122F
SPRB	123D	SPUSER	00F2	SSAH	094D	SSAL	095C
STAR	0B45	STARA	0B55	STBEGH	08EC	STBEGL	08F7
STEAH	08D4	STEAL	08E0	STENDH	0902	STENDL	090D
STEP	14CF	STPBIT	1A59	STRTBT	1039	STSAH	08BC
STSAL	08C8	SY	1A69	SYNC	0B1C	SYNCA	0B21
SYNCNT	1A74	SYNCS	0A1F	SYNVEC	0B9B	TAPDIS	0936
TDISP	091B	TEMP	00FC	TEMPA	1A60	TEMPB	1A61
TEMPX	00FD	TENSYN	0B36	TIMH	1A5F	TIML	1A5E
TLOOK	09BB	TPINIT	0810	TPI	0821	TPTXT	082E
TPVEC	0918	TTXT	0833	VALID	0C2C	VALIT	1431
VALNUM	11CB	VALT	1438	VAL	0C33	VNA	11D3
VU	0C64	WARMST	1CCA	XREG	00F5	YREG	00F4
ZERO	0AB8	ZRO	0AC1				

EPROM-Ergänzung des Standard-Monitors und EPROM-Ergänzung des Systemprogramms PM

In Buch 3 wird an verschiedenen Stellen der Umweg beschrieben, der nach dem Übergang auf den dezimalen Rechenmodus beschritten werden muß, um den Standard-Monitor oder das Systemprogramm PM zu erreichen. Dort bestand dieser Umweg aus einem Hilfsprogramm, das in das PIA-RAM gesetzt wurde (siehe Buch 3, Anhang 2 sowie Kapitel 12, Seite 169).
Der Umweg ist im genannten Fall nach wie vor notwendig. Die zugehörigen Instruktionen brauchen jedoch nicht mehr in das PIA-RAM gesetzt zu werden, sondern sie sind im PM/PME-EPROM (ESS 507N) permanent vorhanden. Näheres geht aus den untenstehenden Bildern 1a und 1b hervor. Da das PIA-RAM für diesen Zweck nicht mehr benötigt wird, kann die Leitung zum Anschluß K6 der Aufsteckplatine entfallen (siehe Buch 3, Kapitel 12, Bild 17). Die Routinen BINAR und PMBINA befinden sich jetzt im PM/PME-EPROM; Signal K4 sorgt dafür, daß sie jederzeit durchlaufen werden können. Ebenso wie Signal K4 muß auch das Signal K7 stets an der Aufsteckplatine anliegen!



Sachwortverzeichnis

Viele verbinden mit dem Stichwort "Computer" die Vorstellung: weniger Arbeit – mehr Freizeit. Aktueller und realistischer ist da schon die Formel: mehr Arbeit in gleicher Zeit. Die steigende Zahl der µComputer-Fans läßt noch einen anderen Schluß zu: mehr Arbeit in der Frei-Zeit!
Vor dem ersten Tastendruck steht allerdings die Qual der Wahl, denn das Angebot an Fix-und-fertig- oder Selbstbau-Computern, an Büchern und Zeitschriften ist sehr groß. Es soll schon Leute gegeben haben, die den Wald vor lauter Bäumen nicht sahen; dann entweder gar nicht anfingen oder zu Hause feststellen, daß der neue Home-Computer wohl doch ein Mißgriff war.

Elektor möchte deshalb allen Newcomern auf diesem Gebiet unter die Arme greifen:
- Mit dem Junior-Computer wird in Buch 1 ein preisgünstiger Ein-Platinen-Selbstbau-Mikrocomputer als Lernsystem angeboten.
- In Buch 2 ist die Rede von Monitor, Editor, Assemblieren und Disassemblieren.
- Buch 3 beschreibt die Schaltung der Interface-Karte, die richtige Verwendung und Bedienung der Erweiterungen.

In Buch 4 ist es soweit: Mit der Software-Intelligenz der Interface-Karte rückt der Junior-Computer endgültig in die Klasse der Personal-Computer auf. Dafür sorgen in erster Linie die Programme "Tape-Management" und "Printer-Management". Sie erfüllen jedoch nur dann ihren Zweck, wenn der Anwender sie optimal nutzen kann. Dies ist möglich, weil das Buch die einzelnen Programme bis ins Detail beschreibt. Einzelne Programme werden Befehl für Befehl besprochen, wobei ausführliche Flußdiagramme die "Denkweise" der Software offenlegen. Damit machen Sie sich für Ihre eigene Programmierung sämtliche Tricks des Juniors zu eigen.
Das gesamte System ist soweit ausbaufähig, wie es die spezielle oder allgemeine Anwendung fordert.
Also dann: Frisch an's Werk.

Elektor Verlag GmbH, D-5133 Gangelt 1

ISBN 3-921608-25-2