BIBO-ASSEMBLER HANDBUCH

von

Erwin Reuß und Peter Bee

(c) 1986 COMPY-SHOP

VerLaTeXt von Martin Metz am

27. Juli 2004

Die Informationen im vorliegenden Handbuch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Die Autoren und Herausgeber dieses Handbuches können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen. Für Verbesserungsvorschläge und Hinweise auf Fehler sind die Autoren dankbar. Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien sowie der Übersetzung in fremde Sprachen. Die gewerbliche Nutzung der in diesem Handbuch gezeigten Modelle, Programme und Arbeiten ist nur mit der ausdrücklichen Genehmigung der Autoren erlaubt.

BIBO-ASSEMBLER HANDBUCH

(c) 1986 Compy-Shop Ohg, 4330 Mülheim Ruhr
Alle Rechte vorbehalten

Inhaltsverzeichnis

Kapitel 1

VORWORT

Viele Leute fragten uns immer wieder nach dem Assembler, mit dem unsere Programme geschrieben wurden, und wollten diesen Assembler auch kaufen. Nun denn, hier ist er.

Mit dem BIBO- ASSEMBLER wurden fast alle unsere Programme geschrieben, so zum Beispiel der 16K BIBOMON, die SPEEDY 1050 oder die COMPY-SHOP Grafikdemo's von Peter Sabath.

Auf den vielfachen Wunsch vieler unserer Kunden, haben wir nun diesen Assembler auch für alle ATARI Computer Besitzer nutzbar gemacht.

Was Ihnen hier nun vorliegt ist die Disketten Version des BIBO- ASSEMBLER's. Um mit diesem Assembler arbeiten zu können, brauchen Sie einen ATARI Computer der Serien 400/800. 600XL/800XL oder 800XE/130XE. Sie benötigen mindestens ein Diskettenlaufwerk und mindestens einen 48K großen Arbeitsspeicher. Haben Sie einen Computer der mit einer RAMDISK versehen ist, so kann der BIBO- ASSEMBLER auch mit dieser RAMDISK arbeiten. Sie werden feststellen, daß die Assemblierungszeiten, wenn Sie mit einer RAMDISK arbeiten, verschwindend gering sind.

Ein Wort noch zu Raubkopien. Der Assembler wird auf einer ungeschützten Diskette als File geliefert. Wir haben den Preis des BIBO- ASSEMBLER's sehr niedrig angesetzt. Gemessen an seinen Leistungen könnte er durch aus wesentlich mehr kosten. Wir haben alles in allem ca. 4 Monate intensive Arbeit in die Erstellung dieser Version gesteckt. Kaufen Sie deshalb keine Raubkopien! Raubkopierer schaden nicht nur uns, sondern auch Ihnen, da wir keine Hard- oder Software mehr entwickeln würden, sollten zu viele Raubkopien im Umlauf sein.

Raubkopierer können Ihnen auch keine Update Garantien bieten, so wie wir das machen!

Denken Sie daran, wenn Ihnen eine Raubkopie angeboten wird!

Kapitel 2

EINLADEN UND STARTEN

Wie bereits gesagt, wird der BIBO- ASSEMBLER auf einer ungeschützten Diskette als File geliefert. Sie sollten also als erstes eine Arbeitsdiskette erstellen. Dazu können Sie einmal eine Kopie der Original Diskette benutzen, oder Sie können den BIBO- ASSEMBLER mit jedem DOS Ihrer Wahl benutzen. Was finden Sie nun alles auf der BIBO- ASSEMBLER Masterdiskette? Mit dem BIBO-ASSEMBLER erhalten Sie das BIBO-DOS auf der BIBO- ASSEMBLER Masterdiskette. Dieses BIBO-DOS ist kompatibel zum ATARI DOS 2.5. Die gesamte BIBO- ASSEMBLER Masterdiskette ist im DOS 2.0 Format beschrieben.

2.1 BIBO-ASSEMBLER ARBEITSDISKETTE MIT BIBO-DOS

Kopieren Sie zuerst einmal die BIBO- ASSEMBLER Masterdiskette. Das entsprechende Kopierprogramm befindet sich auf der BIBO- Assembler Masterdiskette. Es trägt den Namen "SCOPY.COM". Sie können es vom BIBO-DOS aus mit der Funktion "L" aufrufen.

Benutzen Sie eine Kopie der BIBO- ASSEMBLER Masterdiskette zusammen mit einem XL oder XE Computer, müssen Sie vor dem Booten der Diskette die OPTION Taste drücken.

Benutzen Sie den BIBO- ASSEMBLER mit einem Computer der Serien 400/800, sorgen Sie bitte vor dem Booten dafür, daß kein Steckmodul mehr im Schacht ist.

Kapitel 3

DER EDITOR

Bei dem Editor des BIBO-ASSEMBLERS handelt sich es um einen sogenannten Zeileneditor. Den Zeileneditor kennen Sie sicher schon vom Basic her.

Hier muß vor jeder Zeile eine Zeilennummer stehen, die Werte innerhalb vorgeschriebener Grenzen annehmen kann. Soll eine Zeile in das Listing eingefügt werden, wird einfach eine Zeilennummer zwischen der vorherigen und der nachfolgenden Zeile eingegeben und schon befindet sich die Zeile an der gewünschten Stelle. Dies geht natürlich nur, wenn vorher die Abstände der Zeilennummern zueinander groß genug gewählt wurde. In der Regel wird ein Abstand der Zeilennummern zueinander von 10 benutzt, damit nachher noch bequem 9 Zeilen eingefügt werden können.

Die gesamte Bedienung des BIBO- ASSEMBLER Editors ähnelt also sehr dem Editor des Atari-Basic. Der BIBO- ASSEMBLER besitzt allerdings einige sehr nützliche Funktionen, die Ihnen der Editor des Standard Atari-Basic nicht bietet. Dies sind zum Beispiel Funktionen wie das automatische nummerieren und neu nummerieren von Zeilen, suchen und austauschen von Zeichen, kopieren, verschieben und löschen von einzelnen Zeilen oder ganzen Zeilenblöcken .

Die innere Struktur des BIBO- ASSEMBLERS sieht keine Eingabe von inversen Zeichen vor. Eine Ausnahme besteht allerdings bei der Eingabe von Texten die in Anführungszeichen (z.B. “TEXT“) eingeschlossen sind. Hier sind auch Inverszeichen zugelassen. Auch den CHRS(0), das ist das “HERZCHEN“, darf nicht eingegeben werden. Falls Sie dieses ASCII Zeichen dennoch benötigen sollten, benutzen Sie bitte die folgende Syntax:

```
.HX 00  ; Entspricht dem ASCII Zeichen fuer den Wert Null
```

Die Speicherverwaltung des Editors im BIBO- ASSEMBLER sieht für die Zeilennummer jeder Zeile zwei Bytes im Speicher vor. Damit sind theoretisch Zeilennummern von 0 bis 65535 (2^{16}) möglich. Aus programmtechnischen Gründen ist die Obergrenze allerdings auf 64000 festgelegt, was aber keinen großen Verlust bedeutet. Ein Assemblerlisting mit 64000 Zeilen ist schon utopisch und nur bei Programmen auf Großrechneranlagen möglich. Andererseits würden diese Unmenge an Zeilen gar nicht in einen 64k Speicher eines 6502 Mikroprozessorsystems passen.

3.1 Editor Befehle

Um alle Funktionen des Editors des BIBO- ASSEMBLER genau kennenzulernen, empfehlen wir, den Assembler in Ihren Rechner einzuladen und alle hier im Handbuch erklärten Funktionen gleich praktisch zu erproben.

Nach dem Einladen des BIBO- ASSEMBLERS in Ihren Rechner meldet sich der Editor mit:

```
Edit>
```

Von hier aus haben Sie die Kontrolle über alle Funktionen des Editors und des Assemblers. Sie können Kommandos im Direktmodus (LIS, DIR, MON...) oder Zeilennummern mit den entsprechenden Labels und Opcodes eingeben.

Alle Kommandos bestehen aus 3 Zeichen oder deren Abkürzungen. Werden mehr als 3 Zeichen eingegeben, wird das 4. Zeichen schon als Zusatz erkannt. Beispiel:

```
LIST
```

Der Befehl LIS wird richtig erkannt, der Editor sucht aber nach dem Label 'T', dem 4. Buchstaben. Ein großer Teil der Befehle kann auch durch den ersten Buchstaben und nachfolgendem Punkt abgekürzt werden:

```
L. = LIS
```

Geben Sie nun zunächst einmal ein:

```
SIZ <RETURN>
```

<RETURN> werden wir im folgenden nicht mehr erwähnen, es sollte selbstverständlich sein, nach jeder Eingabe die RETURN-Taste zu betätigen.

Der BIBO- ASSEMBLER zeigt Ihnen nun die belegten Speicherbereiche an. Da Sie noch nichts eingegeben haben ist die Länge von Source- Data, also die Länge des Programmlistings gleich Null. Die Anfangs- und Endadressen sind gleich. Diese Werte sind übrigens in Hexadezimaler Form angegeben. Jeder, der sich mit Assembler- Programmierung beschäftigt, sollte sich schnell mit Hexadezimalzahlen vertraut machen. Die einzigen Zahlen in diesem Assembler, die nie Hexadezimal werden können, sind die Zeilennummern.

Wir wollen nun ein paar Zeilen eingeben um die Funktionen des Editors praktisch auszuprobieren. Der Editor verfügt über eine automatische Zeilennummerierung. Das heißt, Sie müssen nicht immer die Zeilennummer von Hand eingeben. Das macht alles der Editor für Sie. Drücken Sie einfach die Taste <TAB> und Sie können Ihre erste Zeilennummer sehen:

```
Edit>
00010 _
```

Der Cursor wird hier durch den Unterstrich (_) dargestellt. Geben Sie weiter ein:

```
00010 START  LDA #0
```

Sie können nach dem Label 'START' einfach wieder die <TAB> Taste betätigen, ebenso nach dem Opcode LDA. So wird das Listing automatisch übersichtlich formatiert. Ein paar Worte noch zu den Labeln. Ein Label ist eine Markierung im Programm, mit der Sie einer Unterroutine einen Namen geben können, und diesen Namen auch mit einem Sprungbefehl "anspringen" können. Für diese Label wurde eine maximale Länge von 8 Zeichen reserviert. Sie können für ein Label aber auch mehr als 8 Zeichen eingeben (maximal 128), dies ist allerdings nicht sehr sinnvoll und benötigt sehr viel Speicherplatz. Erfahrungsgemäß reicht die Verwendung von 6 Zeichen für ein Label.

Übrigens: Es macht keinen Unterschied im Speicherbedarf einer Zeile ob Sie zwischen dem Label (START) und dem Opcode (LDA) einen oder mehrere Leerschritte eingeben.

Nach drücken der RETURN-Taste können Sie nun wieder die TAB-Taste betätigen und schon erscheint die nächste Zeilennummer, und sie können die nächste Zeile eingeben:

```
00020     STA $2C6
```

Wie sie sehen, können Sie ständig die TAB-Taste betätigen, um Ihr Programm formatiert und damit übersichtlich einzugeben. Die neue Zeilennummer erscheint nach betätigen der TAB-Taste nur, wenn Sie vorher die RETURN-Taste gedrückt haben. So kann man leicht mit den Cursor Steuertasten auf eine auf dem Bildschirm sichtbare Zeile gehen und mit der Tab-Funktion die richtige Stelle erreichen.

Vergessen Sie aber nicht nach Änderung der Zeile wieder die RETURN-Taste zu betätigen um diese Zeile in das Programmlisting zu übernehmen. Wem diese Art der automatischen Zeilennummerierung nicht gefällt, kann auch eine automatische Numerierung ohne daß man die Tab-Taste betätigen muß aktivieren. Der Befehl hierzu lautet:

```
Syntax:  NUM <Abstand>,<erste Zeile>
(von Number)
Beispiel: NUM 20,1000
```

Hiermit wird nach Eingabe einer Zeile und Betätigung der RETURN-Taste automatisch die nächste Zeilennummer ausgegeben. Der Abstand zur vorherigen Zeilennummer wird mit einer Zahl hinter dem Befehl NUM angegeben. Nach einladen des BIBO-ASSEMBLERS ist dieser Abstand automatisch auf 10 eingestellt. Wird noch eine zweite Zahl, getrennt durch ein Komma, eingegeben wird hierdurch festgelegt mit welcher Zeilennummer die automatische Numerierung beginnen soll. Wird keine der beiden Zahlen eingegeben, wird ein Abstand von 10 festgesetzt und die Numerierung beginnt mit der Zeile 10 oder mit der Zeile an der die Funktion vorher abgebrochen wurde. Ein Abbruch der NUM-Funktion ist jederzeit mit der BREAK-Taste möglich. Um die Auto-Number Funktion wieder zu aktivieren, muß auch wieder NUM eingegeben werden.

Falls Sie es aber vorziehen die Numerierung mit der TAB-Taste zu benutzen, (an diese gewöhnt man sich erfahrungsgemäß sehr schnell und möchte sie nicht mehr vermissen), können Sie auch hierfür den Abstand der Zeilennummern zueinander einstellen. Nach dem Einladen des BIBO- ASSEMBLERS ist auch dieser Abstand fest auf 10 mit der Anfangszeile 10 eingestellt. Der Befehl um diese Werte zu ändern lautet:

```
Syntax:  INC <Abstand>,<erste Zeile>
(von Increment, Erhoehung)
Beispiel: INC 20,1000
```

Der Abstand der Zeilennummern zueinander kann von l bis 255 eingestellt werden. Wird der Abstand weggelassen, wird automatisch der Abstand l gewählt. Diesen benötigt man des öfteren um Zeilen einzufügen. Wird zusätzlich zum Zeilenabstand noch eine weitere Zahl eingegeben, kann dadurch die erste Zeilennummer festgelegt werden, die beim Drücken der TAB-Taste als nächstes ausgegeben werden soll. Während der Eingabe von Zeilen in ein Assemblerlisting kann es immer wieder vorkommen, daß eine einzelne oder auch mehrere Zeilen gelöscht werden sollen. Wie beim BASIC Editor wird durch alleinige Eingabe einer Zeilennummer die zu dieser Nummer gehörende Zeile gelöscht. Sollen mehrere Zeilen gleichzeitig gelöscht werden, empfiehlt sich die Benutzung des Befehles:

```
Syntax:  DEL <erste Zeile>,<letzte Zeile>
(von delete, ausloeschen)
Beispiel: DEL 100,1000
```

Hierdurch werden alle Zeilen zwischen der Zeile 100 und der Zeile 1000 gelöscht. Die Zeilen mit der Nummer 100 oder 1000 müssen dabei selbst nicht vorhanden sein. Wird die letzte Zeilennummer weggelassen, wird natürlich auch nur die Zeile gelöscht, dessen Nummer mit der ersten Zeilennummer übereinstimmt. Sind einmal Zeilen mit dieser Funktion gelöscht worden, können diese nicht mehr zurückgeholt werden. Achten Sie also vorher darauf, welche Zeilen Sie löschen wollen.

Wollen Sie den kompletten Speicher löschen gibt es hierfür auch einen Befehl:

```
NEW
```

Hierdurch werden alle für den BIBO- ASSEMBLER wichtige Speicherstellen neu initialisiert, der HIMEM-Zeiger auf die höchste und der LOMEM-Zeiger auf die niedrigste freie Speicherstelle gesetzt und der Zeilenincrement (INC) auf 10,10 gesetzt. Bevor wir zur weiteren Beschreibung der einzelnen Befehle des BIBO-ASSEMBLERS kommen, erklären wir hier erst einmal die Eingabesyntax des Assemblers:

```
   A          B            C      D          E
+-------+-+----------+-+---+-+-------+-+-------------+
|00010  | |LABEL     | |LDA| |$4567  | |Bemerkung    |
+-------+-+----------+-+---+-+-------+-+-------------+
```

A - Zeilennummer: Kann Werte von 0 bis 64000 annehmen. Muß nicht 5-stellig eingegeben werden, wird allerdings beim Li- sten immer auf 5 Stellen aufgefüllt.

B - Label: Kann l bis 128 Zeichen lang sein. Das l. Zeichen muß immer ein Buchstabe sein (Ausnahme Local Label). Der Label muß direkt oder maximal l Leerschritt nach der Zeilennummer eingegeben werden.

C - Opcode: Besteht immer aus 3 Zeichen, dem 65(C)02 Opcode oder einem Pseudo-Opcode bestehend aus einem Punkt und zwei Buchstaben. Muß mindestens 2 Leerschritte von der Zeilennummer oder l Leerschritt vom Label entfernt eingegeben werden.

D - Zusatz: Adressangabe, Wert oder Filename. Muß direkt oder maximal l Leerschritt nach dem Opcode eingegeben werden.

E - Erklärung: Hier können alle Zeichen verwendet werden. Muß zurm Opcode (C) mindestens 2 Leerschritte oder dem Zusatz (D) mindestens l Leerschritt entfernt sein.

Die gesamte Zeile darf maximal 128 Zeichen lang werden.

Beachten Sie: Viele Erklärungen verbessern zwar die Übersicht über ein Assemblerlisting, sind aber auch sehr Speicherintensiv und verlängern die Assemblierzeit. Sie sollten deshalb bei langen Assemblerlistings sehr kurz gehalten werden. Wie im Basic können Sie auch im BIBO- ASSEMBLER Remarks (Erklärungen) einfügen, die eine ganze Zeile belegen. Hierfür muß direkt oder maximal ein Leerschritt hinter der Zeilennummer ein Semikolon (;) oder Sternchen (*) eingegeben werden. Diese Zeile wird beim Assemblieren einfach übersprungen.

```
Beispiel:

00100 ; Alle Zeichen hinter dem Semikolon gelten als Erklaerung
00200 * Auch ein Sternchen kann benutzt werden *
```

Um verschiedene Programmteile voneinander zu trennen, können Sie Leerzeilen einfügen, indem Sie nur ein Semikolon oder Sternchen hinter der Zeilennummer eingeben. Die elegantere Lösung hierfür ist, eine Trennzeile einzugeben.

Wird direkt hinter der Zeilennummer ein Minus-Zeichen (-) eingegeben, wird nachher beim Listen diese Zeile komplett mit dem Minus-Zeichen gefüllt.

```
Beispiel:

Eingabe:

02000 -
Daraus wird beim Listen:

02000 --------------------------------
```

Um die folgenden Funktionen direkt praktisch auszuprobieren empfehlen wir ein kurzes Listing aus dem Anhang abzutippen.

3.1.1 LIS

Einer der wichtigsten Befehle des BIBO- ASSEMBLERS ist der LIST-Befehl. Hiermit kann das einmal eingegebene Listing gesamt oder nur zum Teil wieder auf dem Bildschirm dargestellt werden.

```
Syntax:  LIS <erste Zeile>,<letzte Zeile>
(von Listen)
Beispiel: LIS 100,1000
```

Dieser Befehl kann als L. abgekürzt werden.

Das Listing kann durch Control-1 angehalten und fortgesetzt werden. Mit der BREAK-Taste wird das Listing abgebrochen. Wird nur LIS oder L. eingegeben

wird das gesamte Listing von der ersten bis zur letzten Zeile auf dem Bildschirm aufgelistet.

In unserem l. Beispiel werden die Zeilen von 100 bis 1000 gelistet. Soll nur eine einzelne Zeile gelistet werden, wird die letzte Zeile weggelassen:

```
LIS 1000
```

gibt nur die Zeile mit der Nummer 1000 (falls vorhanden) aus. Wird hinter der Zeilennummer ein Komma gesetzt:

```
LIS 1000,
```

werden alle Zeilen beginnend ab Zeile 1000 bis zum Ende des Listings ausgegeben. Eine weitere sehr nützliche Funktion des Listbefehles ist das Listing ab einem angegeben Label ausgeben zu lassen:

```
LIS START
```

Das Listing wird nach dem Label START abgesucht und falls vorhanden, beginnend mit dieser Zeile bis zum Ende des Programmes aufgelistet. Der Label muß auf jeden Fall vollständig vorhanden sein. Bei einer Eingabe von LIS STAR oder LIS STARTER wird die Zeile mit dem Label START nicht gefunden.

3.1.2 FIN

Das vollständige Listing kann mit folgender Funktion nach Zeichenfolgen abgesucht werden:

```
Syntax:  FIN <Zeichenkette>
(von find, finden)
Beispiel: FIN LDA
```

In unserem Beispiel wird das gesamte Listing nach der Zeichen- kombination 'LDA' abgesucht und, falls vorhanden, die Zeile in der sich diese Kombination befindet auf dem Bildschirm ausgegeben. Ist eine Buchstabenkombination in einer Zeile mehrfach vorhanden, wird diese Zeile nur einmal ausgegeben.

3.1.3 REP

Als erweiterte Suchfunktion haben Sie die Möglichkeit, die gefundene Zeichenfolge gegen eine neue Zeichenfolge auszutauschen:

```
Syntax:  REP "<Zeichenkette>"<Zeichenkette>"
(von replace, austauschen)
Beispiel: REP "LDA"LDX"
```

Hier werden alle Buchstabenkombinationen 'LDA' gegen 'LDX' aus- getauscht und die entsprechende Zeile auf dem Bildschirm dargestellt. Statt der Anführungszeichen können auch andere Zeichen (. , : - /) benutzt werden, wichtig ist nur, daß in einer Eingabezeile immer das gleiche Trennungssymbol benutzt wird. Der BIBO- ASSEMBLER benötigt diese Zeichen um den Anfang und das Ende eines Strings (Buchstabenkombination) zu erkennen. Soll also ein Anführungszeichen selbst ausgetauscht werden, muß als Trennzeichen natürlich ein anderes Zeichen verwendet werden.

Mit der obigen Funktion werden alle Zeichenketten in dem Listing ausgetauscht. Es besteht aber auch die Möglichkeit, einzelne Zeilen zu ändern.

Hinter dem letzten Trennungszeichen geben Sie einfach ein Fragezeichen ein:

```
REP "LDA"LDX"?
```

Wird in einer Zeile die Buchstabenkombination 'LDA' gefunden, wird diese auf dem Bildschirm ausgegeben und der Austauschvorgang angehalten. Der Cursor bleibt hinter dieser Zeile stehen und der Editor wartet auf eine Eingabe des Benutzers, ob diese Zeile geändert werden soll. Durch betätigen der RETURN-Taste wird nun 'LDA' gegen 'LDX' ausgetauscht und die geänderte Zeile wieder auf dem Bildschirm dargestellt. Soll die Zeile unverändert bleiben, drücken Sie einfach die SPACE-Taste oder 'N'. Ein Abbrechen der Austauschfunktion ist mit der ESC-Taste möglich.

ACHTUNG: Da der Editor normalerweise keine Inversen Zeichen annimmt, können diese auch mit der Replace-Funktion nicht ausgetauscht werden. Diese Zeilen müssen einzeln aufgelistet und geändert werden. Die maximale Länge der Austauschstrings ist auf 64 Zeichen begrenzt.

3.1.4 COP

Kopieren und Verschieben von Zeilen:

Einzelne oder mehrere Zeilen können dupliziert werden. Der Befehl hierzu lautet:

```
Syntax:   COP <nach Zeile>,<von Zeile>,<bis Zeile>
(von copy, kopieren)
Beispiel: COP 1000,100,200
```

In unserem Beispiel werden alle Zeilen zwischen 100 und 200 an die Position vor die Zeile 1000 kopiert. Diese Zeilen sind jetzt also doppelt vorhanden. Um Probleme mit der Zeilennumerierung auszuschließen. wurde nach dem Copy-Befehl automatisch ein Renumber durchgeführt.

Sollen wie im oberen Beispiel die Zeilen verschoben werden, also an die neue Stelle kopiert, aber an der ursprünglichen Stelle gelöscht werden, wenden Sie folgenden Befehl an:

```
Syntax:   MOV <nach Zeile>,<von Zeile>,<bis Zeile>
(von move. verschieben)
Beispiel: MOV 1000,100,200
```

Für diesen Befehl gilt die gleiche Syntax, wie für den Copy-Befehl, nur werden hier die ursprünglichen Zeilen 100 bis 200 gelöscht nachdem sie vor die Zeile 1000 kopiert wurden. Auch hierbei wird ein automatisches Renumber durchgeführt. Soll nur eine einzelne Zeile verschoben oder kopiert werden kann die dritte Zeilennummer (bis Zeile) weggelassen werden.

```
Beispiel: COP 1000,300
          MOV 100,1000
```

Bei den beiden vorher besprochenen Befehlen wurde bereits der Renumber angesprochen. Hierbei wird das Programmlisting neu durchnummeriert. Dies ist auf jeden Fall immer bei den Befehlen MOVE. COPY und bei der noch später beschriebenen Funktion MERGE notwendig und wird deshalb nach beenden der Funktion automatisch durchgeführt.

3.1.5 REN

Nun kann es bei der Eingabe eines Programmlistings schon einmal vorkommen, daß man zwischen zwei Zeilen noch mehrere Zeilen einfügen möchte, aber nicht genügend Zeilennummern zur Verfügung stehen. Für diesen Zweck können Sie das Programm manuell neu durchnumerieren.

```
Syntax: REN
(von renumber, neu numerieren)
```

Mit diesem Befehl wird das vollständige Listing beginnend mit der Zeilennummer 10 in Zehnerschritten neu durchnumeriert. Sie können außerdem die Anfangszeilennummer und die Schrittweite der einzelnen Zeilen bestimmen, wenn Sie folgendes eingeben:

```
Syntax:   REN <1. Zeilennummer>,<Schrittweite>
Beispiel: REN 1000,20
```

Hierbei wird in 20er Schritten beginnend mit der Nummer 1000 durchnumeriert. Sollte durch eine zu hohe Schrittweite oder Anfangsnummer während des Renumber- Vorganges die Zeilennummer 64000 überschritten werden, wird der Renumber abgebrochen und eine Fehlermeldung ausgegeben. Da hierbei schon ein Teil des Programmes neu numeriert wurde, sind die Zeilennummern unter Umständen nicht mehr in der richtigen Reihenfolge vorhanden. Im vorderen Teil des Listings können die Nummern höher sein als im hinteren. Solange in diesen Zeilen nichts geändert wird, ändert sich auch nichts an der Reihenfolge der Zeilen. In diesem Fall sollte auf jeden Fall das Programmlisting noch einmal mit einer kleineren Schrittweite oder Anfangsnummer neu durchnumeriert werden, bis diese Fehlermeldung nicht mehr auftritt. Die Renumber-Funktion eignet sich auch sehr gut um größeren Platz zwischen zwei Zeilen zu schaffen, ohne daß eine große Schrittweite gewählt werden muß. Angenommen, Sie wollen ab Zeile 1000 einen Programmteil eingeben, wissen allerdings noch nicht genau wie lang dieser Teil werden soll, müssen also alle dahinter liegenden Zeilen mit einer neuen Anfangsnummer durchnumerieren.

Das geschieht durch Eingabe einer dritten Zahl hinter dem Kommando REN:

```
Syntax:   REN <1. Zeilennummer>,<Schrittweite>,<Startzeile>
Beispiel: REN 5000,10,1000
```

Das Listing wird erst ab der Zeile 1000 neu durchnumeriert, der Programmteil vor dieser Zeile bleibt unberührt. Die Zeile mit der Nummer 1000 erhält somit die Nummer 5000, alle dahinter liegenden Zeilen werden mit der Schrittweite 10 neu numeriert. Jetzt haben Sie Ihr Ziel erreicht, zwischen 1000 und 4999 haben Sie genug Platz um Ihr neues Programmteil einzufügen.
ACHTUNG: Die l. Zeilennummer des neu zu numerierenden Programmteiles muß auf jeden Fall höher sein als die Startzeile. Wird diese Vorschrift nicht eingehalten, kann es vorkommen, daß Zeilennummern in die falsche Reihenfolge gebracht werden.

3.2 DER 2. EDITOR

Ihr BIBO- ASSEMBLER kann zur gleichen Zeit zwei unterschiedliche Programmlistings im Speicher verwalten. Somit haben Sie die Möglichkeit, Includefiles (Programmteile, die während des assemblierens nachgeladen werden) einzuladen, zu verändern und wieder abzuspeichern ohne daß Sie das Hauptprogramm vorher abspeichern müssen. Sie können auch das Listing im 2.Editor unabhängig vom Hauptprogramm editieren und assemblieren.

3.2.1 ED2

Sie rufen den 2.Editor auf mit:

```
ED2
```

3.2.2 SIZ

Die Daten des Haupteditors sind nun sozusagen im Speicher versteckt worden und können nicht verändert werden. Sie haben also einen vollständigen Editor vor sich, mit der einzigen Einschränkung, daß nicht mehr so viel Speicherplatz zur Verfügung steht, da die Daten des Haupteditor auch noch vorhanden sind. Wie die Speicherbelegung des BIBO- ASSEMBLERS nun aussieht, können Sie wieder mit dem Befehl

```
SIZ (S.)
```

erfahren. Source Data zeigt Ihnen nun drei Werte. Der erste Wert ist die Anfangsadresse der Daten im 2.Editor, der zweite Wert ist die Endadresse der Daten des 2.Editors und zugleich die Anfangsadresse der Daten im Haupteditor. Der dritte Wert schließlich ist die Endadresse der Daten des Haupteditors.

In diesem 2.Editor haben Sie die gleichen Möglichkeiten wie im Haupteditor. Alle bisher besprochenen und noch folgenden Funktionen können genau so angewendet werden. Auch der Assembler kann aufgerufen werden, die Daten im Haupteditor bleiben dabei unberücksichtigt.

3.2.3 RET

Zum Haupteditor kehren Sie wieder zurück mit:

```
RET
(von return, zurueckkehren)
```

Hierdurch werden allerdings die Daten des 2.Editors gelöscht. Vergessen Sie also nie, die Daten im 2.Editor vor Eingabe von RET abzuspeichern. Die Diskettenoperationen werden später erklärt.

3.2.4 MER

Ein weiterer Anwendungszweck des 2.Editors ist, zwei Listings zu einem Gesamtlisting zusammenzufügen.
Durch Eingabe von:

```
MER
(von merge, zusammenfuegen)
```

werden die Daten des 2.Editors in den Haupteditor kopiert und danach der 2.Editor automatisch verlassen. Das Listing, das sich vorher im 2.Editor befand, steht nun vor dem Programmlisting im Haupteditor. Da die Reihenfolge der Zeilennummern durch das Zusammenfügen der beiden Listings nicht mehr in Ordnung sein kann, wird mit dem Merge automatisch ein Renumber ausgeführt.

Sollen nur einzelne Zeilen vom 2.Editor in das Listing des Haupteditor kopiert werden, müssen alle Zeilen, die nicht kopiert werden sollen, im 2.Editor vor Aufruf der Funktion Merge gelöscht werden.

Wichtig: Mit NEW werden immer die Daten des 2.Editors und die des Haupteditors gelöscht. Soll nur der 2.Editor gelöscht werden, muß man zuerst mit RET in den Haupteditor zurückkehren und danach wieder mit ED2 den 2.Editor aufrufen.

Kapitel 4

DIE DISKETTENFUNKTIONEN DES EDITORS

Es ist selbstverständlich für einen Assembler, daß man das Listing, das man mühevoll eingegeben hat auch auf Diskette abspeichern und später wieder einladen kann. Der BIBO- ASSEMBLER bietet hierzu zwei verschiedene Möglichkeiten:

Das Listing kann als ASCII-Datei, also im Klartext mit allen Leerschritten und den 5-stelligen Zeilennummern oder in einem komprimierten, BIBO- ASSEMBLER eigenem Format abgespeichert und eingeladen werden.
Das BIBO- ASSEMBLER eigene Format hat den großen Vorteil, daß die Datei wesentlich kürzer als eine ASCII-Datei des gleichen Listings ist. Eine negative Eigenschaft der ASCII-Datei ist auch, daß der Ladevorgang sehr langsam ist, da jede Zeile beim Einlesen einsortiert werden muß. Man sollte also um Zeit und Diskettenplatz zu sparen Listings immer im BIBO- ASSEMBLER Format abspeichern.

4.0.5 SAV

Das im Speicher befindliche Listing wird im BIBO- ASSEMBLER Format abgespeichert mit:

```
Syntax:  SAV "<device: name.ext>"
(von save, abspeichern)
Beispiel: SAV "D1:TEST.ASM"
```

Das Anführungszeichen hinter dem Filenamen kann weggelassen werden. Wenn Diskettenlaufwerk l angesprochen werden soll, kann man natürlich die '1' in 'Dl:' weggelassen.
Wenn Sie sich im 2.Editor befinden, wird natürlich auch nur das Listing in diesem 2.Editor abgespeichert. Falls sie einen Rechner mit einer RAMDISK besitzen und das verwendete DOS diese RAMDISK verwalten kann, können Sie Ihr Listing natürlich auch unter der entsprechenden Laufwerksnummer in der RAMDISK abspeichern. Vergessen Sie aber nicht, vor ausschalten des Computers die Daten aus der RAMDISK auf eine Diskette umzukopieren.

WICHTIG: Bei Eingabe eines längeren Assemblerlistings empfiehlt es sich in regelmäßigen Zeitabständen die eingegebenen Daten auf Diskette (nicht RAM-DISK) abzuspeichern. Sollte einmal der Strom ausfallen, sind alle Dateien, die sich im Editor oder in der RAM-DISK befanden, verloren.

4.0.6 LOA

Ein mit der SAV-Funktion abgespeichertes Listing kann wieder eingeladen werden mit:

```
Syntax:   LOA "<device:name.ext>"
(von load, laden)
Beispiel: LOA "D1:TEST.ASM"
```

Auch hier kann das Anführungszeichen hinter dem Filenamen und die Laufwerksnummer, wenn es sich um Laufwerk l handelt, weggelassen werden.

Wird ein neues Programmlisting mit LOA eingeladen wird automatisch ein NEW ausgeführt, das heißt alle Daten, die sich im Editor (oder Editor 2) befanden, sind gelöscht worden. Wird eine Datei in den 2.Editor eingeladen, bleibt das Listing im Haupteditor natürlich erhalten.

4.0.7 ENT

Die zweite Speicher- bzw. Lademöglichkeit von Programmen ist das Listen und Entern von ASCII-Dateien. Mit dieser Art des Speicherformats können auch Listings. die mit anderen Assemblern erstellt und auf die Diskette als ASCII-Datei gelistet wurden in den Editor des BIBO- ASSEMBLERS geladen werden. Natürlich muß das Assemblerlisting schon mit Zeilennummern versehen sein. Die mit Enter eingeladenen ASCII-Dateien müssen natürlich noch auf die Syntax des BIBO- ASSEMBLERS angepaßt werden. Hierzu können Sie die schon erklärte Funktion REPlace benutzen.
Mit

```
Syntax:  ENT "<device:name.ext>"
(von enter)
Beispiel: ENT "D1:TEST.TXT"
```

kann eine ASCII-Datei in den Editor des BIBO- ASSEMBLERS geladen werden. Da mit dieser Ladefunktion jede Zeile direkt beim ENTern auf das BIBO-ASSEMBLER Format gebracht und einsortiert wird, kann das Einladen eines längeren Listings schon einige Minuten dauern. Ein blinkender Cursor zeigt mit jedem Aufblinken an, daß eine Zeile übernommen und in das Listing eingefügt wurde. Durch die interne Speicherorganisation des BIBO- ASSEMBLERS (diese wird später näher erklärt) blinkt der Cursor mit länger werdendem Listing immer langsamer. Sobald wieder Edit> oder Ed2> auf dem Bildschirm erscheint, ist der Ladevorgang beendet.

4.0.8 LST

Der umgekehrte Vorgang. also das Abspeichern eines Listings im Klartext (ASCII-Format) geschieht mit dem Befehl:

```
Syntax:  LST "<device:name.ext>"
(von listen)
Beispiel: LST "Dl:TEST.LST"
```

Mit diesem Befehl kann das Listing auch auf den Drucker ausgegeben werden. Hierzu mehr im Abschnitt “Druckeransteuerung“.

4.0.9 DIR

Mit einer weiteren Funktion können Sie sich das Inhaltsverzeichnis (Directory) von der Diskette (oder RAMDISK) ausgeben lassen:

```
Syntax:  DIR
(von directory)
```

Hiermit erscheint das Inhaltsverzeichnis von der Diskette, die sich in Laufwerk l befindet. Soll das Inhaltsverzeichnis eines anderen Laufwerks oder von der RAMDISK aufgerufen werden oder ein bestimmter Filename in einem Directory gesucht werden, geschieht dies durch zusätzliche Eingabe der Laufwerksnummer und des Filenamens.

```
Beispiele:
DIR "D8"
```

gibt das vollständige Directory von Laufwerk 8 aus. D8 ist zum Beispiel die RAMDISK in DOS 2.5.

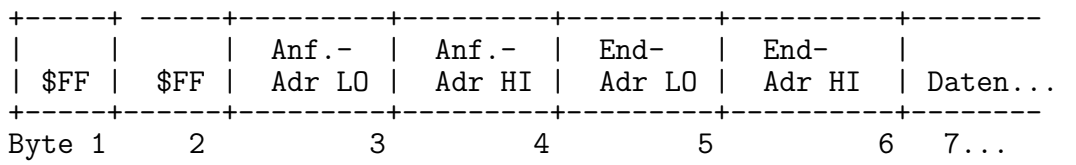
```
DIR "D:*.ASM"
```

zeigt alle Programmnamen auf der Diskette im Laufwerk l, die mit dem Extender ASM enden.

Zusätzlich zum abspeichern und laden von Programmlistings (Sourcedaten) ist es auch möglich Binärfiles (Objectdaten) abzuspeichern oder ins RAM einzuladen.

Diese Binärfiles haben einen Fileheader (Vorspann), wie er auch in .COM- oder .EXE-Files zu finden ist.

Es folgt eine Kurzbeschreibung über den Aufbau eines Binärfiles:



Die ersten beiden Bytes des Files müssen immer $FF $FF sein. Dieses ist die Kennung, daß es sich um ein Binärfile handelt und daß die nächsten vier Bytes die Anfangs- und Endadresse des Files bestimmen. Das dritte und vierte Byte geben die Startadresse an, in die das 7.Byte des Files geladen werden soll. Das fünfte und sechste Byte bestimmen die Endadresse bei der das letzte Byte abgelegt werden soll. Das siebte Byte ist also das erste Datenbyte des Binärfiles. Die Anzahl der Datenbytes errechnet sich aus Endadresse-Anfangsadresse+1. Die eins muß auf jeden Fall dazuaddiert werden, denn sind Anfangs- und Endadresse gleich, ist die Differenz dieser beiden Adressen Null. Da aber ein Byte erforderlich ist um diese Adresse mit Daten zu versorgen muß zu dieser Differenz ein Byte addiert werden. Diese Addition wird allerdings DOS - intern vorgenommen, der Benutzer muß nur wissen, wie man Binärfiles abspeichert oder lädt. Mit jedem DOS haben Sie die Möglichkeit Binärfiles zu erzeugen oder einzuladen. Mit DOS 2 oder 2.5 werden Binärfiles mit der Option K unter Angabe der gewünschten Anfangs- und Endadresse abgespeichert und mit der Option L wieder eingeladen.

4.0.10 BSA

Der BIBO- ASSEMBLER bietet bei Eingabe der Anfangs- und End-Adressen beim Abspeichern von Binärfiles wesentlich mehr Möglichkeiten als das DOS. So können Dezimale oder Hexadezimale Adressen wie auch Labels eingegeben werden. Die entsprechende zum Label gehörende Adresse befindet sich nach dem assemblieren eines Proarammes in der Symbol - Table, die später näher erklärt wird.

Das Abspeichern eines Binärfiles geschieht durch:

```
Syntax:  BSA "<device:name.ext>",Anfang,Ende
(von binary save)
Beispiel: BSA "D:TEST.COM",S4000,S4FFF
BSA "D:TEST.COM",16384,END
```

Hierbei müssen auf jeden Fall die Anführungszeichen nach dem Filenamen eingegeben werden. Die Anfangs- und Endadresse muß durch ein Komma getrennt werden, wird eine Adressangabe weggelassen, erscheint die Fehlermeldung 'Syntax Error'. Die Endadresse muß immer höher als die Anfangsadresse sein. Als Adressen können Dezimale Werte (16384,1536), Hexadezimale Werte ($4000,$600) oder Labels (START,END) eingesetzt werden. Labels können natürlich nur verwendet werden, wenn diese schon durch einen Assembliervorgang in die Symbol-Table eingetragen wurden. Es ist auch möglich die Adresse rechnerisch zu ermitteln. Hierbei können Labels oder Adressen miteinander addiert oder subtrahiert werden.

```
Beispiele:
START+$100
END-ANF+1
16384+START
```

Diese Möglichkeiten der Adressangabe werden noch bei anderen Befehlen benötigt, die wir später erklären werden. Dort werden wir die Möglichkeiten der Adressangabe noch näher erläutern und diese nur mit 'Adresse' kennzeichnen.

Ein mit BSA abgespeichertes Binärfile hat die Eigenschaft, die Speicherstelle selbst zu bestimmen, ab der es in der den Speicher eingeladen werden soll. Wenn Sie dieses File mit dem DOS einladen wollen, wird es auf jeden Fall immer in diese im Fileheader stehende Adresse eingeladen.

4.0.11 BLO

Diese Möglichkeit haben Sie natürlich auch vom BIBO- ASSEMBLER aus:

```
Syntax:  BLO "<device:name.ext>"
(von binary load)
Beispiel: BLO "D:TEST.COM"
```

Hiermit wird ein Binärfile in den originalen Speicherbereich eingeladen, aus dem es abgespeichert wurde. Nun kann es vorkommen, daß dieses zu ladende File ausgerechnet in den Bereich geladen werden soll, in dem das DOS, der Assembler oder der Bildschirmspeicher liegt. Das Binärfile kann deshalb auch an einen anderen Speicherbereich geladen werden. Hierzu wird nach dem Filenamen einfach die Adresse angegeben, in die das Binärfile geladen werden soll.

```
Beispiel:
BLO "D:TEST.COM",$5000
```

Sollte das Binärfile aus mehreren Datenblöcken bestehen, wie es beim Assemblieren erzeugt werden kann, wird nur der erste Datenblock in die angegebenen Speicherstelle eingelesen und danach der Ladevorgang abgebrochen.

Kapitel 5

DRUCKERANSTEUERUNG

Wie schon im Abschnitt 'Diskettenfunktionen' beschrieben, läßt sich das vollständige im Haupteditor oder Editor 2 befindliche Listing ausdrucken.

5.0.12 LST "P:"

Der Befehl hierzu ist der gleiche wie für das Abspeichern von Listings im ASCII-Format, nur das statt des Diskettenhandlers der Druckerhandler angesprochen werden muß.

```
LST "P:"
```

Hierbei wird das gesamte Listing mit allen ASCII-Zeichen zum Drucker geschickt, auch alle eventuell im Listing befindliche Invers- oder Controlzeichen. Diese können ein Teil des Ausdrucks eventuell durcheinander bringen. Es ist bei der Eingabe eines Programmlistings schon darauf zu achten, daß, wenn ein Listing einmal ausgedruckt werden soll, Steuerzeichen für den Drucker (Control-Codes), nicht als Klartext, sondern als Hex- oder Dezimalwerte eingegeben werden. Inverse Zeichen werden von vielen Druckern angenommen und in Schrägschrift (Italic) dargestellt, diese können beibehalten werden. Was Ihr Drucker bei inversen Zeichen ausgibt sollten Sie am besten einmal selbst testen.

Falls Sie das Listing mit allen Control- und Inverszeichen Ausdrucken wollen, empfehlen wir Ihnen das Programm 'Textfile-Printer'. Um dieses Programm nutzen zu können, muß das Listing mit LST "D:name.ext" auf eine Diskette gelistet werden, und kann vom Textfile-Printer nachher wieder eingeladen werden. Der Textfile-Printer ist ein selbständiges, nicht zum BIBO-ASSEMBLER gehörendes Programm und kann über den COMPY-SHOP bezogen werden. Es gibt dieses Programm zur Ansteuerung von Epson- und kompatiblen Druckern sowie für den ATARI 1029 Drucker.

5.0.13 PON

Als weitere Druckeroption besteht die Möglichkeit, alle Zeichen, die auf dem Bildschirm ausgegeben werden, auch parallel hierzu zum Drucker zu schicken. Sie aktivieren diese Funktion mit:

```
PON
(von Printer on)
```

Diese Funktion wird automatisch bei auftreten eines I/0-Errors abgeschaltet.

5.0.14 POF

Das Abschalten per Hand geschieht durch:

```
POF
(von Printer off)
```

Bei PON werden auf dem Drucker keine Control- und Inverszeichen ausgegeben und mit einem Leerschritt (Space) dargestellt. Hierdurch ist es möglich, bei aktiviertem Drucker auch im Monitor des BIBO- ASSEMBLERS Listings oder Hex-Dumps auszudrucken.

Kapitel 6

VERLASSEN DES ASSEMBLERS

6.0.15 DOS

Sie können den Assembler verlassen mit:

```
DOS
```

Sie befinden sich dadurch in der Kommandoebene des DOS oder das DUP.SYS des DOS wird nachgeladen und sie erhalten das DOS-Menü. Die Daten, die sich vor verlassen des Assemblers im Editor befanden, bleiben vollständig erhalten. Der Assembler kann wieder aktiviert werden mit RUN 9500 oder mit ASS wenn Sie das DOS der BIBO- ASSEMBLER Diskette verwenden.

Kapitel 7

ASSEMBLIEREM VON PROGRAMMEN

7.0.16 ASM

Das im Editor befindliche Programmlisting wird assembliert mit:

```
ASM oder A.
```

Bei dem Assembler handelt es sich um einen sogenannten Zweipass Assembler, das heißt, es werden zwei Durchgänge benötigt, um eine Objektdatei zu erzeugen. Die beiden Durchgänge werden mit PASS l und PASS 2 auf dem Bildschirm angezeigt. In PASS l werden die Speicherbereiche angezeigt, in die in PASS 2 die Daten abgelegt werden. Im 2. Durchgang wird das komplette Listing auf dem Bildschirm ausgegeben. Diese Bildschirmausgabe kann das Assemblieren sehr verlangsamen, deshalb sollte bei längeren Listings der Pseudo-Opcode .LI OFF in der ersten Zeile stehen, durch den die Bildschirmausgabe des Listings unterdrückt wird.

Was passiert nun beim Assemblieren in den beiden Durchgängen ?

Im PASS l wird das komplette Listing auf Eingabefehler überprüft, die Symboltabelle erstellt, die Werte der Labels errechnet und die Länge der Objektdatei ermittelt.

In PASS 2 werden die zu den Opcodes gehörenden Werte im RAM abgelegt oder als Objectfile auf die Diskette geschrieben und die im ersten Durchgang errechneten Werte der Labels in die Objectdaten eingefügt.

Das Assemblieren kann jederzeit durch drücken der OPTION-Taste abgebrochen werden. Der Abbruch wird mit 'ASM Stopped' angezeigt. Falls ein Fehler während des assemblierens auftritt, wird der Assembliervorgang ebenfalls gestoppt und die fehlerhafte Zeile angezeigt. Ein bereits geöffnetes Objectfile wird geschlossen, kann aber noch nicht die erforderliche Länge haben, die im Fileheader angegeben ist. Der Versuch solch ein abgebrochenes Objectfile einzuladen führt auf jeden Fall zu einer Fehlermeldung. Wird beim Assemblieren in einer Includedatei ein Fehler festgestellt, wird der Assembliervorgang ebenfalls abgebrochen und der Editor meldet sich zurück mit

```
Inc>
```

Die Includedatei befindet sich nun im 2.Editor und kann hier direkt bearbeitet werden. Sie können den Fehler beseitigen und das Includefile wieder abspeichern. Mit RET kehren Sie zum Haupteditor zurück. Falls Sie das Programm sofort nach der Änderung wieder assemblieren wollen, können Sie dieses nach abspeichern des Includefiles direkt aus dem Include heraus, der ASM-Befehl führt in diesem Fall automatisch ein RET aus. Sollte der Assembliervorgang erfolgreich abgeschlossen sein, meldet sich der Editor wieder zurück aus dem heraus Sie den Assembler aktiviert haben:

```
Edit>

oder

Ed2>
```

Nach erfolgter Assemblierung eines Programmes können Sie zwei weitere Befehle anwenden.

7.0.17 VAL

Um die beim Assemblieren ermittelte Adresse eines Labels abzufragen gibt es den Befehl:

```
Syntax:  VAL <Label>
(von Value, Wert)
Beispiel: VAL START
```

Hiermit wird die Symboltabelle nach dem Label abgesucht und die zu diesem Label gehörende Adresse in Hexa- und Dezimaler Form dargestellt.

Mit der VAL-Funktion sind auch Additionen und Subtraktionen sowie die Umwandlung von Hexadezimal- in Dezimalzahlen und umgekehrt möglich.

```
Beispiele:
VAL $1234
VAL START+$200
VAL 125-$10
```

7.0.18 SYM

Wird im 2.Durchgang beim Assemblieren das Listing auf dem Bildschirm ausgegeben, erscheint zum Schluß automatisch die Symboltabelle. Die vollständige Symboltabelle kann aber auch nachträglich auf dem Bildschirm aufgelistet werden mit:

```
SYM
```

Hierbei werden alle Labels alphabetisch geordnet und deren Werte angezeigt. Falls lokale Labels verwendet wurden, werden auch diese angezeigt.

Kapitel 8

TESTEN VON ASSEMBLIERTEN PROGRAMMEN

Um ein assembliertes Programm ablaufen zu lassen, können Sie den Monitor aufrufen und von dort aus das Programm mit der Funktion GO starten oder per Trace oder Einzelschritt durchtesten. Eine genaue Erklärung dieser Befehle erfolgt in der Beschreibung des Monitors im BIBO- ASSEMBLER. Eine weitere Möglichkeit ein Programm zu starten, besteht direkt von der Kommandoebene des Editors aus.

8.0.19 RUN

Der Befehl hierzu lautet:

```
Syntax:   RUN <Adresse>
oder:     RUN <Label>
Beispiel: RUN $4000
          RUN START
```

Der Programmablauf kann mit der BREAK-Taste unterbrochen werden falls die BREAK-Taste vom zu testenden Programm nicht gesperrt wird). Wird das Programm mit der BREAK-Taste angehalten oder stößt der Prozessor auf einen Software-Break (BRK) wird dieses auf dem Bildschirm angezeigt und akustisch gemeldet. Der Stand des Programmzählers und der Prozessorregister bei diesem Break wird zusätzlich angezeigt.

Kapitel 9

ADRESSIERUNGSARTEN

Der Mikroprozessor 6502 kennt insgesamt 13 Adressierungsarten. Hierzu kommen noch zwei Adressierungsarten die nur der 65C02 Prozessor kennt.

In diesem Kapitel werden wir nicht die Funktion der Adressierungsarten erklären sondern nur ihre Schreibweise im BIBO-ASSEMLBER. Die Erklärung der einzelnen Funktionen würde den Rahmen dieses Handbuches sprengen. Wir verweisen hier auf einige sehr gute Lehrbücher, die im Fachhandel zu beziehen sind.

1. Implizierte Adressierung:
   Opcodes die Prozessorintern Vorgänge auslösen.

   ```
   Beispiele:
   PHA
   TAX
   INY
   ```

2. Intermediate Adressierung:
   Die direkte Adressierung besteht aus dem Opcode und einem 8 Bit Wert. Dieser Wert kann dezimal, hexadezimal, als ASCII-Zeichen oder als Low- oder High-Byte einer Adresse oder Label eingegeben werden.

   Dezimal: Das Doppelkreuz (#) zeigt an, daß es sich nachfolgend um einen 8-Bit Wert handelt. Nach dem Doppelkreuz wird der dezimale Wert angegeben.

   ```
   Beispiele:
   LDA #0
   ORA #123
   EOR #255
   ```

   Hexadezimal: Dem Doppelkreuz muß das Dollar-Zeichen ($) und eine Hexadezimale Zahl folgen.

   ```
   Beispiele:
   LDA #$1F
   ORA #$10
   EOR #$FF
   ```

Bei Werten zwischen 0 und 9 kann das Dollar-Zeichen auch weggelassen werden, und der Wert dezimal angegeben werden.

```
LDA #$9 entspricht  LDA #9
```

ASCII: Ein ASCII-Wert wird durch ein Apostroph (') hinter dem Doppelkreuz angegeben.

```
Beispiele:
LDA #'A
LDX #'?
```

Hierbei können keine inversen Zeichen eingegeben werden. Sie können aber durch Addition von $80 (128) das ASCII-Zeichen invertieren.

```
LDA #'A+$80
```

Low–Byte einer Adresse: Das untere Byte einer 16-Bit Adresse oder Label wird als Wert angegeben.

```
Beispiele:
LDA #START
LDA #$1234 entspricht LDA #S34
```

High-Byte einer Adresse: Soll das obere Byte einer 16-Bit Adresse als Wert eingegeben werden, wird statt des Doppelkreuzes (#) der Schrägstrich (/) eingegeben.

```
Beispiele:
LDA /START
LDA /$1234 entspricht LDA #S12
```

3. Absolute Adressierung:
   Dem Opcode folgt eine 16-Bit Adresse oder ein Label, der auf eine 16-Bit Adresse zeigt.

```
Beispiele:
LDA 12345
STA $4000
STX LABEL
```

4. Zeropage Adressierung:
   Dem Opcode folgt eine 8-Bit Adresse oder ein Label, der auf eine 8-Bit Adresse zeigt.

```
Beispiele:
LDA 128
STA $80
STX LABEL
```

5. Accumulator Adressierung:
   Diese Adressierungsart bezieht sich nur auf den Accumulator des 6502 Prozessors.

   ```
   Beispiele:
   LSR
   ASL
   ROR
   ```

6. Indirekt Indizierte Adressierung:
   Der Inhalt des X-Registers wird zu einer 8-Bit Adresse addiert. Das 'X' muß innerhalb der Klammern '()' , durch ein Komma getrennt stehen!

   ```
   Beispiele:
   LDA ($45,X)
   LDA (LABEL,X)
   ```

7. Indirekt Indizierte Adressierung:
   Der Inhalt des Y-Registers wird zu der Adresse addiert, auf die eine Zeropageadresse zeigt. Das 'Y' muß sich außerhalb der Klammern '()' befinden!

   ```
   Beispiele:
   LDA ($58),Y
   STA (LABEL),Y
   ```

8. X-Indizierte Adressierung:
   Das X-Register wird zu einer 8-Bit Adresse addiert.

   ```
   Beispiele:
   LDA $80,X
   STA LABEL,X
   ```

9. X-Indizierte Adressierung:
   Das X-Register wird zu einer 16-Bit Adresse addiert.

   ```
   Beispiele:
   LDA $8000,X
   STA LABEL,X
   ```

10. Y-Indizierte Adressierung:
    Das Y-Register wird zu einer 8-Bit Adresse addiert.

    ```
    Beispiele:
    LDA $E0,Y
    STA LABEL,Y
    ```

11. Y-Indizierte Adressierung:
    Das Y-Register wird zu einer 16-Bit Adresse addiert.

```
Beispiele:
LDA $4000.Y
STA LABEL,Y
```

12. Indirekt Absolute Adressierung:
    Für diese Adressierungsart gibt es nur einen Befehl. Die Adresse kann 8- oder 16-Bit lang sein.

```
Beispiele:
JMP ($3000)
JMP ($45)
JMP (LABEL)
```

13. Relative Adressierung:
    Diese Adressierungsart gibt es nur bei Sprungbefehlen (Branches). Der relative Sprung kann maximal -126 oder +129 Bytes um den Befehl reichen.

```
Beispiele:
BMI $4010
BEQ *+10
BEQ LABEL
```

Die folgenden zwei Adressierungsarten können nur bei der Verwendung der 65C02-Opcodes benutzt werden.

14. Indirekte Zeropage Adressierung:
    Diese Adressierungsart bezieht sich nur auf eine 8-Bit Adresse.

```
Beispiele:
LDA ($45)
STA (LABEL)
```

15. Absolute X-Indizierte Indirekte Adressierung:
    Diese Adressierungsart bezieht sich immer auf eine absolute Adresse.

```
Beispiel:
JMP ($4000,X)
JMP (LABEL,X)
```

9.0.20 6502 UND 65C02 OPCODES

Abs-Absolut Zp-Zeropage Ind-Indirekt Accu-Accumulalor Inm-Intermediate

```
ADC Add with Carry
   Mit bertrag addieren
   ADC Abs
   ADC Zp
   ADC Inm
   ADC Abs,X
```

```
   ADC Abs,Y
   ADC (Ind,X)
   ADC (Ind),Y
   ADC Zp,X
C02: ADC (Ind)

AND Logisches UND
   AND Abs
   AMD 2p
   AND Inm
   AND Abs,X
   AND Abs,Y
   AND (Ind,X)
   AND (Ind),Y
   AND Zp,X
C02: AND (Ind)

ASL Arithmetic shift left
   Arithmetisch links schieben
   ASL Accu
   ASL Abs
   ASL Zp
   ASL Abs,X
   ASL Zp,X

BCC Branch on carry clear
   Verzweigen bei geloeschtem bertrag
   BCC Relativ

BCS Branch on carry set
   Verzweigen bei gesetztem Uebertrag
   BCC Relativ

BEQ Branch on equal or zero
   Verzweigen falls gleich Null
   BEQ Relativ

BIT Speicherbits testen
   BIT Abs
   BIT Zp
C02: BIT Zp,X
C02: BIT Abs,X

BMI Branch on minus
   Verzweigen falls negativ
   BMI Relativ

BNE Branch on not equal to zero
   Verzweigen falls ungleich Null
   BNE Relativ

BPL  Branch on plus
   Verzweigen falls positiv
```

```
  BPL Relativ

BRK  Break
   Unterbrechung durch Software
   BRK

BVC  Branch on overflow clear
   Verzweigung falls kein Ueberlauf
   BVC Relativ

BVS Branch on overflow set
   Verzweigung falls Ueberlauf
   BVS Relativ

CLC Clear carry
   Ubertragsflag loeschen
   CLC

CLD Clear decimal mode
   Dezimalmodus abschalten
   CLD

CLI Clear Interrupt mask
   Unterbrechungsmaske loeschen
   CLI

CLV Clear overflow flag
   Ueberlaufflag loeschen
   CLV

CMP Compare to accumulator
   Mit Akkumulator vergleichen
   CMP Abs
   CMP Zp
   CMP Inm
   CMP Abs,X
   CMP Abs,Y
   CMP (Ind,X)
   CMP (Ind),Y
   CMP Zp,X
C02: CMP (Ind)

CPX Compare to register X
   Mit Register X vergleichen
   CPX Abs
   CPX Zp
   CPX Inm

CPY Compare to Register Y
   Mit Register Y vergleichen
   CPY Abs
   CPY Zp
   CPY Inm
```

```
DEC Decrement Memory
   Speicher dekrementieren
   DEC Abs
   DEC Zp
   DEC Zp,X

DEX Decrement Register X
   Register X dekrementieren
   DEX

DEY Decrement Register Y
   Register Y dekrementieren
   DEY

EOR Exclusive or
   Exklusiv oder verknuepfen
   EOR Abs
   EOR Zp
   EOR Inm
   EOR Abs,X
   EOR Abs,Y
   EOR (Ind,X)
   EOR (Ind),Y
   EOR Zp,X
C02: EOR (Ind)

INC Increment Memory
   Speicher inkrementieren
   INC Abs
   INC Zp
   INC Abs,X
   INC Zp,X

INX Increment X register
   Register X inkrementieren
   INX

INY Increment Y register
   Register Y inkrementieren
   INY

JMP Jump to address
   Zur angegebenen Adresse springen
   JMP Abs
   JMP (Ind)
C02: JMP (Ind,X)

JSR Jump to Subroutine
   Sprung zum Unterprogramm
   JSR Abs

LDA Load accumulator
```

```
   Akkumulator laden
   LDA Abs
   LDA Zp
   LDA Inm
   LDA Abs,X
   LDA Abs,Y
   LDA (Ind,X)
   LDA (Ind),Y
   LDA Zp,X
C02: LDA (Ind)

LDX Load Register X
   Register X laden
   LDX Abs
   LDX Zp
   LDX Inm
   LDX Abs,Y
   LDX Zp,Y

LDY Load Register Y
   Register Y laden
   LDY Abs
   LDY Zp
   LDY Inm
   LDY Abs,X
   LDY Zp,X

LSR Logical shift left
   Logisch rechts schieben
   LSR Accu
   LSR Abs
   LSR Zp
   LSR Abs,X
   LSR Zp,X

NOP No Operation
   Keine Operation
   NOP

ORA Or with accumulator
Mit dem Akkumulator Oder-verknuepfen
   ORA Abs
   ORA Zp
   ORA Inm
   ORA Abs,X
   ORA Abs,Y
   ORA (Ind,X)
   ORA (Ind),Y
   ORA Zp,X
C02: ORA (Ind)

PHA Push accumulator
   Akkumulator auf den Stapel bringen
```

```
   PHA

PHP Push processor Status
   Statusregister auf den Stapel bringen
   PHP

PLA Pull accumulator
   Akkumulator vom Stapel holen
   PLA

PLP Pull processor Status
   Statusregister vom Stapel holen
   PLP

ROL Rotate left one bit
   Um ein Bit nach links rotieren
   ROL Accu
   ROL Abs
   ROL Zp
   ROL Abs,X
   ROL Zp,X

ROR Rotate right one bit
   Um ein Bit nach rechts rotieren
   ROR Accu
   ROR Abs
   ROR Zp
   ROR Abs,X
   ROR Zp,X

RTI Return from interrupt
   Rueckkehr aus einer Programmunterbrechung
   RTI

RTS Return from Subroutine
   Aus einem Unterprogramm zurueckkehren
   RTS

SBC Subtract with Carry
Mit Uebertrag subtrahieren
   SBC Abs
   SBC Zp
   SBC Inm
   SBC Abs,X
   SBC Abs,Y
   SBC (Ind,X)
   SBC (Ind),Y
   SBC Zp,X
C02: SBC (Ind)

SEC Set carry
Uebertragsflaq setzen
   SEC
```

```
SED Set decimal mode
   Dezimalmodus einschalten
   SED

SEI Set Interrupt mask
   Unterbrechungsmaske setzen
   SEI


STA Store accumulator in memory
   Akkumulatorinhalt im Speicher ablegen
   STA Abs
   STA Zp
   STA Abs,X
   STA Abs,Y
   STA (Ind,X)
   STA (Ind),Y
   STA Zp,X
C02: STA (Ind)

STX Store X register in memory
   Register X im Speicher ablegen
   STX Abs
   STX Zp
   STX Zp,Y

STY Store Y register in memory
   Register Y im Speicher ablegen
   STY Abs
   STY Zp
   STY Zp,X

TAX Transfer accumulator into X register
   Akkumulator in X Register uebertragen
   TAX

TAY Transfer accumulator into Y register
   Akkumulator in Y Register uebertragen
   TAY

TSX Transfer Stack register into X register.
   Stapelzeiger in X Register uebertragen
   TSX

TXA Transfer X register into accumulator
   Register X in Akkumulator uebertragen
   TXA

TXS Transfer X register into Stack register
   Register X in den Stapelzeiger uebertragen
   TXS
```

```
TYS Transfer Y register into stack register
   Register Y in den Stapelzeiger uebertragen
   TYS
```

9.0.21 65C02 OPCODES

```
BRA Branch always
   Immer verzweigen
   BRA Relativ

DEA Decrement accumulator
   Akkumulator dekrementieren
   DEA

INA Increment accumulator
   Akkumulator inkrementieren
   INA

PHX Push X register
   X Register auf den Stapel bringen
   PHX

PHY Push Y register
   Y Register auf den Stapel bringen
   PHY

PLX Pull X register
   X Register vom Stapel holen
   PLX

PLY Pull Y register
   Y Register vom Stapel holen
   PLY

STZ Store zero
   Speicherstelle mit Null fuellen
   STZ Abs
   STZ Zp
   STZ Abs,X
   STZ Zp,X

TRB Test and reset bit
   Bit testen und zuruecksetzen
   TRB Abs
   TRB Zp

TSB Test and set bit
   Bit testen und setzen
   TSB Abs
   TSB Zp
```

Kapitel 10

PSEUDO OPCODES

Pseudo-Opcodes dienen dazu, bestimmte Funktionen während des assemblierens auszuführen, wie zum Beispiel die Startadresse eines Programmes festzulegen, Includedateien einzuladen, Objectfiles zu erzeugen usw.

Die Pseudo-Opcodes unterscheiden sich von den normalen 65(C)02 Opcodes dadurch, daß diese nur aus zwei Buchstaben bestehen. Damit der Assembler diese erkennt, beginnen diese Pseudo-Opcodes mit einem Punkt “.“

10.0.22 .OR

```
.OR <Adresse> Originaladresse setzen.
Beispiele:
.OR $5000
.OK LABEL+$40
```

Die Originaladresse gibt die Basisadresse an, zu der das folgende Programm assembliert werden soll. Dieser Befehl kann in einem Listing öfter vorkommen, so daß Programmteile an verschiedene Adressen im Speicher abgelegt werden kann. Soll sich die Originaladresse auf einen Label beziehen, muß dieser Label bereits schon vorher definiert worden sein. Wird in einem Listing keine Originaladresse angegeben, ist diese Adresse automatisch $4000.

10.0.23 .TA

```
.TA <Adresse> Targetadresse setzen.
Beispiele:
.TA S3000
.TA LABEL-128
```

Die Targetadresse gibt an, ab welcher Speicherstelle die Obbjectdaten im Speicher abgelegt werden sollen. Dieser Befehl muß auf jeden Fall hinter die Zeile mit dem .OR - Befehl gesetzt werden. Der .OR - Befehl setzt immer die Original- und die Targetadresse auf den gleichen Anfangswert. Wird der Targetbefehl weggelassen, werden die Objectdaten zur Originaladresse in den Speicher abgelegt.

10.0.24 .OF

```
.OF "device:name.ext" Objectfile erzeugen.
Beispiel:
.OF "D:TEST.OBJ"
```

Die Objectdaten werden nicht mehr in den Speicher abgelegt sondern als Binärfile auf der Diskette abgespeichert. Das Objectfile kann auch aus mehreren Blöcken bestehen, wenn in einem Listing mehrere male der .OR - Befehl verwendet wird. Wird eine Targetadresse im Listing angegeben, erhält der Fileheader des Binärfiles auch die Targetadresse, die Daten des Binärfiles sind zur Originaladresse hin assembliert. Der .OF - Befehl muß immer der .TA- oder der .OR-Anweisung folgen. Diese Reihenfolge ist unbedingt einzuhalten.

```
Beispiele:
.OR $5000
.OF "D:TEST.COM"
```

Programm wird nach $5000 assembliert und abgespeichert.

```
.OR $AOOO
.TA S4000
.OF "D:TEST.OBJ"
```

Programm wird nach $AOOO assembliert. Das Object- file wird erzeugt, daß es nach $4000 eingeladen werden kann.

Kommt in einem Listing mehrfach der .OF - Befehl vor, wird das geöffnete Objectfile geschlossen und ein neues Objectfile erzeugt. Hat dieses Objectfile den gleichen Namen wie das vorherige File, wird dieses natürlich vom neuen Objectfile überschrieben.

10.0.25 .NO

```
.NO     Objectdaten nicht im Speicher ablegen.
```

Hiermit kann ein Programm zu Testzwecken assembliert werden. Es werden keine Daten im Speicher abgelegt. Dieser Befehl hat keinen Einfluß auf ein Objectfile. Wird die .NO Anweisung mitten in das Listing eingefügt, werden ab dieser Stelle keine Daten mehr im Speicher abgelegt.

10.0.26 .OB

```
.OB     Objectdaten im Speicher ablegen.
```

Hiermit wird der .NO - Befehl wieder aufgehoben. Alle Zeilen, die dem .OB-Befehl folgen werden wieder in den Speicher assembliert. Soll das vollständige Listing in den Speicher assembliert werden, muß dieser Befehl nicht eingegeben werden.

10.0.27 .LI ON

```
.LI ON   Listing einschalten.
```

In PASS 2 des Assembliervorganges wird jede, diesem Befehl folgende assemblierte Zeile auf dem Bildschirm ausgegeben. Diese Ausgabe enthält die Adresse in die die Daten geschrieben werden, 1 bis 3 Bytes, Zeilennummer, Label, Opcode und Erklärungen.

Diese Funktion wird automatisch aktiviert, wenn der Befehl in einem Listing nicht verwendet wird. Bedingt durch die langsame Textausgabe des ATARI- Betriebssystems wird bei eingeschaltetem Listing die Assemblierzeit eines Programmes um ein vielfaches verlängert.

10.0.28 .LI OFF

```
.LI OFF   Listing ausschalten.
```

Diese Funktion hebt das Listen des Programmes in PASS 2 beim Assemblieren auf. Alle dem Befehl folgende Zeilen werden nicht mehr auf dem Bildschirm ausgegeben. Um die größtmögliche Assemblierzeit eines Programmes zu erreichen, sollte als erste Zeile in einem Listing dieser Befehl stehen.

10.0.29 .EN

```
.EN     Assemblierung beenden.
```

Dieser Befehl kann dazu benutzt werden, den Assembliervorgang zu stoppen. Alle hinter diesem Befehl liegenden Programmzeilen werden nicht mehr assembliert. Am Ende eines Programmlisting wird automatisch der .EN-Befehl ausgeführt. Wird kein vorzeitiger Abbruch eines Assembliervorganges gewünscht, braucht dieser Befehl in einem Listing nicht eingegeben werden.

10.0.30 .HX

```
.HX <Data> Hexwerte einfuegen.
Beispiele:
.HX FF
.HX 0011223344556677 88
```

Die auf den .HX-Befehl folgenden Hexwerte werden in die Objektdatei eingefügt. Die einzelnen Hexzahlen müssen immer zweistellig sein und dürfen durch kein Zeichen getrennt sein.

10.0.31 .DA

```
.DA <Data> Daten einfuegen.
Beispiele:
.DA $45
.DA #123
.DA /LABEL
.DA "ASCII"
.DA LABEL,#$9B,#$FF
```

Dieses ist ein Universalbefehl mit dem 8- oder 16-Bit Daten in dezimaler und hexadezimaler Form, als ASCII-Zeichen oder als Low- oder High-Byte einer Adresse eingefügt werden können. Mehrere Daten können durch Komma getrennt eingegeben werden. ASCII-Zeichen müssen in Anführungszeichen stehen. Es können auch inverse ASCII-Zeichen eingegeben werden.

10.0.32 .AS

```
.AS <"STRING"> ASCII-Zeichen einfgen.
Beispiele:
.AS "Hallo"
.AS /TEST/
.AS -'Assembler'
```

ASCII-Zeichen, die sich zwischen den Anführungszeichen befinden, werden in den Objectcode eingefügt. Als Anfangs und Endkennung eines ASCII-Strings können beliebige Zeichen eingegeben werden. Wichtig ist nur, daß vor und hinter einem String das gleiche Zeichen benutzt wird. Nur wenn die Anführungszeichen (“) benutzt werden, können auch inverse ASCII-Zeichen eingegeben werden. Soll der gesamte Text Invers dargestellt werden, kann vor dem Trennungszeichen ein Minus-Zeichen (-) eingegeben werden.

10.0.33 .AT

```
.AT <"String"> ATASCII-Zeichen einfuegen.
Beispiele:
.AT "Test"
.AT /"HALLO" wie geht es/
.AT -"COMPUTER"
```

Für diesen Befehl gelten die gleichen Regeln wie für den vorausgegangenen Befehl .AS. Zwischen den Trennungszeichen befinden sich hier nur ATASCII-Zeichen, die vom Assembler automatisch in den Bildschirmcode umgewandelt werden.

10.0.34 .BL

```
.BL Laenge, Wert Speicherblock ueberspringen oder mit einem Wert
    fuellen.
Beispiele:
.BL 1000
.BL $200
.BL 100,255
.BL $80,$FF
```

Der erste Wert hinter der Anweisung .BL gibt die Anzahl der Speicherstellen an die übersprungen oder mit einem Wert gefüllt werden sollen. Der zweite Wert gibt an, mit welchem Wert die Speicherstellen gefüllt werden sollen. Wird kein zweiter Wert eingegeben, wird der Speicherbereich einfach übersprungen. Der zweite Wert kann nur 8-Bit groß sein, also Werte von 0-255 ($00-$FF) annehmen.

Bei einem Datenfile wird ein Speicherbereich, der übersprungen werden soll, mit den Wert 0 aufgefüllt.

10.0.35 .IN

```
.IN "device:name.ext" Includedatei in das Listing einfuegen.
Beispiel:
.IN "D:TEIL2.ASM"
```

Das Includefile wird vollständig mit allen Labels und Befehlen in das Hauptprogramm eingebunden. Die Zeilen einer Includedatei werden an die Stelle in die das Listing eingebunden, an der der Befehl .IN steht. Aus einem Includefile kann keine weitere Includedatei aufgerufen werden. Sollte der Speicherplatz knapp werden, kann das Hauptprogramm in mehrere kurze Listing aufgeteilt werden, die beim Assemblieren als Includefile nachgeladen werden.

10.0.36 .DF

```
.DF "device:name.ext" Datenfile in die Objectdaten einbinden.
Beispiel:
.DF "D:BYTES.DAT"
```

Die Daten eines Binärfiles werden an dieser Stelle in die Objectdaten eingebunden. Das Binärfile muß den Fileheader, bestehend aus $\$FF$ $\$FF$, der Anfangs- und der Endadresse, besitzen. Wird versucht ein anderes als ein Binärfile als Datenfile einzuladen, erscheint die Fehlermeldung “File Type Error“.

10.0.37 .EQ

```
.EQ <Adresse> Einem Label einen Wert zuweisen.
Beispiele:
LABEL  .EQ $45
LAB2   .EQ LABEL+1
COLOR = $2C6
```

Einem Label wird ein 8- oder 16-Bit Wert zugewiesen. Das Gleichheitszeichen (=) kann statt des .EQ eingegeben werden, dieses hat die gleiche Funktion. Wichtig: Label müssen direkt am beginn eines Programmlistings mit .EQ definiert werden. Sollte diese Regel nicht eingehalten werden, kann es vorkommen, daß Labeladressen falsch errechnet werden und das assemblierte Programm nicht ordnungsgemäß abläuft.

Kapitel 11

LOCAL LABELS

Local Labels sind Label, die mehrfach in einem Programmlisting vorkommen können. Diese bestehen nicht wie normale Label aus einer Buchstabenkombination sondern aus dem Punkt und einer dezimalen Zahl zwischen 0 und 99 (.12). Local Labels sollten immer dann benutzt werden, wenn Label nur innerhalb eines Programmteiles benutzt werden. Diese Label können von außerhalb nicht angesprungen werden. Ein Local Label benötigt in der Symbol-Tabelle weniger Speicherplatz als ein normaler Label und wird beim Assemblieren schneller aufgefunden. Aus diesen Gründen sollte, wo immer möglich, Local Labels benutzt werden. Da ein Local Label den Offset (Abstand) zu einem normalen Label angibt, kann der Abstand aus Programmtechnischen Gründen nur maximal 127 Bytes werden. Sollte dieser Bereich überschritten werden, erscheint die Fehlermeldung 'Local out of range' und es sollte wieder ein normaler Label benutzt werden.

```
Beispiel:
00010 START LDY #0
00020 .1    LDA $E000,Y
00030       STA $5000,Y
00040       INY
00050       BNE .1
00060 ;
00070 LOOP  LDX #$80
00080 .1    LDA $0,X
00090       STA $5100.X
00100       INX
00110       BNE .1
00120       RTS
```

In unserem Beispielprogramm wurde zweimal der gleiche .Label (.1) benutzt. Von der Zeile 110 wird aber auf jeden Fall immer zur Zeile 80 zurückgesprungen, da die beiden gleichen Label durch einen normalen Label getrennt sind.

Kapitel 12

DIE BEFEHLE DES MONITORS

Der BIBO- ASSEMBLER verfügt über einen leistungsfähigen Monitor. Dieser Monitor ist eine “abgespeckte“ Version des 16K-BIBOMON. Sie erreichen den Monitor vom Editor her durch Eingabe des Befehles MON. Der Monitor meldet sich dann mit dem Prompt MON> und erwartet eine Eingabe von Ihnen. Wenn Sie im Besitz eines 4K-BIBOMON oder gar eines 16K-BIBOMON sind, können Sie mit dem Befehl BIB auch direkt in die Benutzeroberfläche dieser Maschinensprache Monitore einspringen. Beide Monitore können dann über den Befehl Q wieder verlassen werden. Sie kehren dann wieder in den Editor des BIBO- Assembler zurück. Die folgenden Monitor Befehle stehen Ihnen im Monitor des BIBO-Assembler zur Verfügung.

12.0.38 BEFEHLSSATZ DES MONITORS

Befehl	Bedeutung
:	Schreibe in Adresse
;	Schreibe in Adresse
.	Speicher auflisten (von.bis)
+	Rechne HEX+HEX
-	Rechne HEX-HEX
,	Speicher auflisten (halbe Seite)
>	Suche nach (HEX, ASCII, ATASCII)
?	Register anzeige
=	Fuelle Speicherbereich
"	ASCII Code
G	GOTO - Programmstart
L	Disassembliere Speicher
M	Verschiebe Speicher
Q	Rueckkehr zum Editor
R	Sektor lesen
S	Programm verfolgen im Einzelschritt
T	Programm verfolgen
V	Vergleiche Speicherbereiche
W	Schreibe Sektor

12.0.39 DIE BEDEUTUNG DER BEFEHLE

```
BEFEHL: : - Schreibe in Adresse (STORE)
FORMAT: (Adresse):(Hex) (Hex)
BEISPIEL: 1000:AO 09
```

Erklärung: Dieser, und der nachfolgende Monitor Befehl, sind in Ihrer Anwendung völlig identisch, so daß eine Besprechung des Befehles nachfolgend gegeben wird.

```
BEFEHL: Schreibe in Adresse (STORE)
FORMAT: (Adresse);(Hex) (Hex)
BEISPIEL: 1000:AO 09
```

Erklärung: Wie bereits beim letzten Befehl erwähnt, sind dieser Monitor Befehl und der letzte völlig identisch. Sie dienen dazu, eine, oder mehrere Speicherstellen zu ändern. Die Anwendung dieser Befehle ist sehr leicht. Sie geben zuerst die Speicherstelle an, ab der die Änderung erfolgen soll, dann die neuen Werte für diese und die darauf folgenden Speicherstellen. Sie können bis zu 8 Speicherstellen mit einem Befehl ändern.

```
BEFEHL: . - Liste Speicher (MEMORY DUMP)
FORMAT: (Start).(Ende)
BEISPIEL: 1000.2000
```

Erklärung: Wenn Sie sich einen größeren Speicherbereich schnell ansehen wollen, benutzen Sie diesen Monitor Befehl. Auch seine Anwendung ist sehr leicht. Sie geben einfach die Startadresse des Speicherbereiches und die Endadresse des Speicherbereiches an, den Sie sehen möchten. Getrennt werden diese beiden Daten von einem Punkt. Sie können das Listing dann durch gleichzeitigen Druck von CONTROL und l (Taste l) anhalten, bzw. weiterlaufen lassen. Sie können dieses Listing jederzeit durch Druck der BREAK - Taste verlassen.

```
BEFEHL: + - Rechne HEX + HEX
FORMAT: (HEX)+(HEX)
BEISPIEL: 2DO+12
```

Erklärung: Es kommt oft vor, das Sie 2 Hexadezimal Zahlen miteinander addieren müssen. Der Monitor des BIBO- ASSEMBLER hilft Ihnen dabei.

```
BEFEHL: - - Rechne HEX - HEX
FORMAT: (HEX)-(HEX)
BEISPIEL: A9-9F
```

Erklärung: Auch dieser Monitor Befehl kann Ihnen Rechenarbeit abnehmen. Die Anwendungen dieser beiden Befehle ist so leicht, das wir hier nicht weiter darauf eingehen wollen.

```
BEFEHL: Liste halbe Seite
FORMAT: (Adresse),
ODER: (Adresse),,,,,
BEISPIEL: 1000,
ODER: 1000,,,,,
```

Erklärung: Dieser Monitor Befehl gibt Ihnen die Möglichkeit, immer nur eine halbe Seite des Arbeitsspeichers Ihres Computers auf dem Monitor zu sehen. Sie können aber auch mehrere Kommas hintereinander eingeben, so erhalten Sie ein laufendes Listing, das Sie wieder durch gleichzeitiges Drücken von CONTROL und der Taste l anhalten können, oder durch drücken der BREAK - Taste jederzeit unterbrechen können.

```
Befehl: > - Suche nach (BYTE, ASCII)
Format: (HEX VON).(HEX BIS)>(HEX)(HEX)
Format: (HEX VON).(HEX BIS)>"ASCII
Beispiel: 2000.9BFF>A9 09
Oder:     2000.9BFF>"TEST
Oder:     0>"TEST
```

Erklärung: Dieser Monitor Befehl ist sehr nützlich, um nach "verschollenen" Programmteilen zu suchen. Sie können mit diesem Befehl nach Hexadezimal Zahlen oder nach ASCII Zeichen (Text) suchen. Die Anwendung ist wieder sehr leicht. Sie geben die Startadresse des Speicherbereiches an, von dem aus gesucht werden soll. Danach folgt die Endadresse des Speicherbereiches. Beide Angaben werden durch einen PUNKT getrennt. Als nächstes folgt dann der Such Befehl. Suchen Sie nach ASCII Zeichen, müssen Sie das dem Monitor durch Eingabe der Anführungsstriche mitteilen (Beispiel 2). Geben Sie diese Anführungsstriche nicht ein, werden alle folgenden Eingaben als Hexadezimal Werte interpretiert. Eine Besonderheit stellt das dritte Beispiel dar. Hier wird der gesamte Speicherbereich, beginnend bei $0000, bis nach $FFFF nach der ASCII-Folge "TEST" abgesucht. Sie müssen also nicht immer die Endadresse angeben!

```
BEFEHL: ? - Register Inhalte anzeigen
FORMAT: ?
BEISPIEL: ?
```

Erklärung: Durch Eingabe des Fragezeichens erhalten Sie die aktuellen Werte der Register angezeigt.
Die Bedeutung der einzelnen Abkürzungen:

Als erstes sehen Sie den Programmcounter. Er gibt Ihnen die Speicherstelle der zuletzt ausgeführten Operation an. Danach folgen fünf Abkürzungen:

A= gibt den aktuellen Accumulator Inhalt wieder.
X= gibt den aktuellen Inhalt des X Registers wieder.
Y= gibt den aktuellen Inhalt des Y Registers wieder.
S= gibt den aktuellen Inhalt des Stack Pointers wieder.
P= gibt den aktuellen Processor Status wieder.

```
BEFEHL: = - Fuelle Speicherbereich
FORMAT: (HEX)<(VON).(BIS)=
BEISPIEL: 00<1000.9000=
```

Erklärung: Wollen Sie ganze Speicherbereiche sehr schnell mit einem Wert füllen, so können Sie das mit diesem Monitor Befehl sehr schnell und einfach machen. Die Anwendung: Zuerst geben Sie den Wert an, mit dem Sie den Speicherbereich füllen wollen. In unserem Beispiel ist das der Wert Null. Als nächstes kommt der sehr wichtige Trennungsbefehl <. Nun folgen noch die Angaben über den Speicherbereich, den Sie füllen wollen. Also die Start- und die Endadresse, getrennt durch einen Punkt. In unserem Beispiel wird der Speicherbereich von $1000 bis $9000 mit dem Wert $00 gefüllt.

```
BEFEHL: " - ASCII Code Eingabe
FORMAT: (Adresse)>"(ASCII)
ODER: (Adresse)<"(ASCII)

BEISPIEL: 0>"TEST
ODER: 1000<"TEST
```

Erklärung: Dieses ist kein eigentlicher Monitor-Befehl. Es teilt dem Monitor nur mit, das die Eingaben die als nächstes kommen, als ASCII Zeichen zu interpretieren sind. Eine genaue Anwendung dieser Funktion finden Sie bei dem SUCH Befehl (>). In unserem zweiten Beispiel wird das Wort TEST in die Speicherstellen nach $1000 geschrieben.

```
BEFEHL: G - Starte Programm
FORMAT: (Adresse)G
BEISPIEL: 4000G
```

Erklärung: Wenn Sie Ihr Programm in den Arbeitsspeicher assembliert haben, haben Sie zwei Möglichkeiten, es zu starten. Erstens vom EDITOR aus durch den EDITOR Befehl RUN. Die zweite Möglichkeit ist der direkte Sprung zu einer vorher angegebenen Adresse im Arbeitsspeicher Ihres Computers durch diesen Monitor Befehl. Die Kontrolle wird dabei an das Programm abgegeben, der Programmlauf kann durch drücken der BREAK - Taste unterbrochen werden. In diesem Falle meldet sich der Monitor mit einer Fehlermeldung wieder zurück.

```
BEFEHL: L - Disassembliere Speicher
FORMAT: (Adresse)L
ODER: (Adresse)LLLLL

BEISPIEL: 4000L
ODER: 4000LLLLL
```

Erklärung: Um im Arbeitsspeicher befindliche Programme für den Programmierer lesbarer darzustellen, gibt es im Monitor des BIBO- ASSEMBLER einen Disassembler. Sie können auch hier wieder wählen, ob Sie ein fortlaufendes Listing oder immer nur eine Bildschirmseite des Arbeitsspeichers disassembliert sehen möchten. Da der BIBO- ASSEMBLER auch den Befehlssatz des 65C02 Mikroprozessors der SPEEDY 1050 versteht, werden die Befehle für diesen Prozessor INVERS dargestellt. Hier noch einmal der Hinweis, das der Mikroprozessor Ihres ATARI Computers diese INVERS dargestellten Befehle nicht kennt, und somit "abstürzt", wenn er auf einen solchen Befehl trifft!

```
BEFEHL: M - Verschiebe Speicherbereiche (MOVE)
FORMAT: (ZIEL) (VON) . (BIS)M
BEISPIEL: 1000<C000.CFFFM
```

Erklärung: Mit dem Monitor Befehl M können Sie sehr schnell ganze Speicherbereiche, oder Blöcke verschieben. Auch die Anwendung dieses Befehles ist sehr einfach. Sie geben zuerst die Adresse des Zielspeichers, gefolgt von dem Trennungsbefehl an. Als nächstes kommt die Anfangsadresse des Speicherbereiches der bewegt (verschoben) werden soll. Es fehlt dann nur noch die Endadresse dieses Speicherbereiches. Die letzten beiden Angaben (Anfangs- und Endadresse des zu bewegenden Speicherbereiches) müssen durch einen PUNKT getrennt werden. Als letztes nun noch den MOVE Befehl M. In unserem Beispiel wird der Speicherbereich von $C000 bis $CFFF zum Speicherbereich beginnend ab $1000 bewegt (kopiert).

```
BEFEHL: Q - Verlasse Monitor (QUIT)
FORMAT: Q
BEISPIEL: Q
```

Erklärung: Durch Eingabe von Q können Sie den Monitor wieder verlassen und zum EDITOR zurück kehren.

```
BEFEHL: R - Lese Sektor (READ)
FORMAT: (Adresse)<(HEX)R
ODER: (Adresse)<(HEX).(HEX)R
BEISPIEL: 4000<4R
ODER: 400<4.29R
```

Erklärung: Mit Hilfe des Monitors des BIBO- ASSEMBLER's können Sie einzelne Sektoren, oder ganze Gruppen von Sektoren, in den Arbeitsspeicher Ihres Computers einlesen. Wie Sie aus unseren Beispielen ersehen können, ist auch die Anwendung dieses Befehles sehr einfach. Sie brauchen nur die Adresse des Speicherbereiches abgeben, wo der Monitor den Sektor ablegen soll, und die Sektornummer die gelesen werden soll. Getrennt werden diese beiden Hexadezimalzahlen durch den Trennungsbefehl <. Nach der Sektornummer kommt noch der Lesebefehl R. Wollen Sie mehrere Sektoren einlesen, so brauchen Sie nur den Start- und den Endsektor, getrennt durch einen PUNKT, anzugeben.

```
BEFEHL: S - Verfolge Programm im Einzelschritt
FORMAT: (Adresse)S
ODER: (Adresse)SSSSSSSSS
BEISPIEL: 4000S
ODER: 4000SSSSSS
```

Erklärung: Der Monitor des BIBO- ASSEMBLER bietet Ihnen zwei komfortable Möglichkeiten, Ihr Programm auszutesten. Als erstes haben wir hier die Einzelschritt Funktion. Nach Eingabe der Startadresse werden nun soviele Maschinen Anweisungen abgearbeitet, wie Sie angegeben haben. Die Anzahl der abzuarbeitenden Maschinen Anweisungen bestimmen Sie durch die Anzahl der hinter der Startadresse eingegebenen S (SINGLESTEP) Befehle. Die Kontrolle wird nicht an das Programm abgegeben. In unserem ersten Beispiel wird nur ein Maschinenbefehl ausgeführt, im zweiten Beispiel sechs. Nach der Ausführung jedes einzelnen Befehles erhalten Sie die Programmcounter (Speicheradresse), den Opcode, und die Registerinhalte angezeigt.

```
BEFEHL: T - Verfolge Programm (TRACE)
FORMAT: (Adresse)T
BEISPIEL: 4000T
```

Erklärung: Die zweite Möglichkeit ein Programm zu Testen stellt dieser Befehl dar. Konnten Sie mit dem S Befehl nur einzelne Programm Schritte ausführen, so können Sie mit dem TRACE-Befehl das komplette Programm testen. Auch hier erhalten Sie immer den Programmcounter (Speicheradresse), den Opcode, und die Registerinhalte angezeigt. Durch Druck auf die OPTION - Taste können Sie die Laufgeschwindigkeit erhöhen. Wir möchten Sie aber noch darauf hinweisen, daß ein Programm nicht verfolgt werden kann, wenn Einsprünge in das Betriebssystem gemacht werden!

```
BEFEHL: V - Vergleiche Speicherbereiche (VERIFY)
FORMAT: (ZIEL)<(VON).(BIS)V
BEISPIEL: 4000<C000.C080V
```

Erklärung: Dieser Befehl dient zum Vergleichen zweier Speicherblöcke. Die Anwendung ist die gleiche wie beim MOVE Befehl M. Sie geben zuerst die Speicheradresse des zu vergleichenden Speicherbereiches an. Danach folgt der Trennungsbefehl <. Als nächstes kommt die Startadresse des Originalspeicherbereiches, gefolgt von der Endadresse des Original Speicherbereiches. Getrennt werden diese wieder durch einen PUNKT. Als letzten geben Sie den Vergleichsbefehl V. Sollten sich Unterschiede innerhalb der beiden Speicherbereiche ergeben, so werden sowohl der Unterschiedliche, als auch der Originalinhalt der Speicherbereiche angezeigt.

```
BEFEHL: W - Schreibe Sektor (WRITE)
FORMAT: (Adresse)<4W
ODER: (Adresse)<4.29W
BEISPIEL: 400C<4W
ODER: 400C<4.29W
```

Erklärung: Dieser Monitor Befehl ist das genaue Gegenstück zum Lese-Sektor-Befehl R. Mit dem Lesebefehl R konnten Sie einen oder mehrere Sektoren einlesen, mit dem Schreibebefehl W können Sie diese nun wieder herausschreiben. Die Anwendung beider Befehle ist identisch. Sie geben zuerst die Bufferadresse, gefolgt von dem Trennungszeichen < an. Als nächstes brauchen Sie nur noch die Sektornummer gefolgt vom Schreibbefehl W einzugeben. Auch ganze Gruppen von Sektoren können so abgespeichert werden. Sie brauchen dazu lediglich den Start- und den Endsektor, getrennt durch einen PUNKT, anzugeben.

Nachdem Sie nun alle Monitor Befehle des BIBO- ASSEMBLER kennen gelernt haben, wird Ihnen die tägliche Programmierarbeit mit dem BIBO- ASSEMBLER leichter fallen. Sie werden feststellen, das es kaum einen anderen Assembler gibt, der Ihnen die Möglichkeiten des BIBO- ASSEMBLER und des eingebauten Monitors bietet.

Kapitel 13

Die Fehlermeldungen des Editors

- SYNTAX ERROR: Der eingegebene Befehl existiert nicht oder es wurde zu einem Befehl keine oder zu wenig Zeilennummern eingegeben. Bei Diskettenoperationen fehlt das Anführungszeichen vor dem Dateinamen. Bei Replace fehlen die Trennungszeichen oder sind nicht gleich.
- ED2 ALREADY USED ERROR: Editor 2 wird aufgerufen obwohl dieser bereits aktiviert ist.
- CAN'T MERGE ERROR: Es wurde versucht vom Haupteditor aus ein Merge auszuführen
- OUT OF RANGE ERROR: Es wurde eine Zeilennummer größer als 64000 eingegeben. Während eines Renumber wurde die Zeilennummer 64000 überschritten.
- MEMORY FULL: Das eingegebene Listing ist zu lang. Das zu ladende Programm paßt nicht in den freien Speicher.

Kapitel 14

DISKETTEN- UND I/0-FEHLER

- FILE TYPE ERROR: Es wurde versucht ein nicht im BIBO- ASSEMBLER Format abgespeichertes Programm mit Load einzuladen. Binärfile hat nicht das richtige Fileformat.
- I/O - ERROR #: Der Fehler wurde von einem Peripheriegerät (Diskettenstation, Drucker) oder vom Dos zurückgegeben. Die I/0-Fehlernummern, die während des arbeiten mit dem BIBO- ASSEMBLER auftreten können:

#128 - Break key abort. Während einer I/0-Operation wurde die Break-Taste qedrückt.

#130 - Non existent device. Es wurde versucht, ein Gerät anzusprechen, das nicht in der Handlertabelle eingetragen ist.

#138 - Device Timeout. Der Befehl wurde vom angesprochenen Gerät nicht innerhalb der vorgesehenen Zeitspanne beantwortet.

#139 - Device NAK. Das angesprochene Gerät kennt den Befehl nicht, oder es wurde ein falscher Parameter (z.B. Sektor 0) angegeben.

#144 - Device done error. Es wurde versucht auf eine schreibgeschützte Diskette zu schreiben oder es befindet sich keine Diskette im Laufwerk.

#160 - Drive number error. Es wurde eine illegale Laufwerknummer angegeben.

#161 - Too many open files. Für weitere Dateien sind keine freien Sektor-Buffer mehr verfügbar.

#162 - Disk full. Es können keine Daten mehr auf die Diskette geschrieben werden.

#163 - Fatal System I/O - error. Fehler im DOS.

#164 - File number mismatch. Fehlerhafte Sektorverkettung oder Filenummern vertauscht.

#165 - Filename error. Der Dateiname enthält illegale Zeichen. Zulässig sind nur Buchstaben, Zahlen, Stern und Fragezeichen.

#167 - File locked. Es wurde versucht auf eine gesperrte Datei zu schreiben.

#169 - Directory full. Das Inhaltsverzeichnis der Diskette kann keinen neuen Dateinamen mehr aufnehmen (maximal 64).

#170 - File not found. Es wurde versucht auf eine Datei zuzugreifen, die nicht im Inhaltsverzeichnis der Diskette eingetragen ist.

Kapitel 15

DIE SPEICHERVERWALTUNG DES BIBO- ASSEMBLERS

Im folgenden Kapitel werden wir Ihnen genau erklären, welche Speicherbereiche der Assembler und die Daten belegen und welche Bereiche für die Objekt-Daten beim Assemblieren frei sind.

Der BIBO- ASSEMBLER belegt den Speicherbereich von $9500 bis $BC00. Dies sind ca. 10 kByte, die auf jeden Fall den Speicherbereich belegen. Darüber ($BCOO-$BFFF) liegt die Displaylist und der Bildschirmspeicher. Auch dieser Speicherbereich ist belegt und kann nicht zur Ablage von Objectdaten benutzt werden.

Direkt unterhalb des BIBO- ASSEMBLERS werden die eingegebenen Programmzeilen abgelegt. Die untere Grenze dieses Speicherbereichs wächst mit länger werdenden Listing nach unten an. Wenn sie sich mit der Funktion SIZ die Adressbereiche ansehen, die das Listing (Source-Data) belegen, werden Sie also feststellen, daß sich die Anfangsadresse immer weiter nach unten verschiebt. Die erste Zeile des Listings beginnt also an dieser Anfangsadresse, das Ende des Listings liegt bei der Endadresse, die auch als HIMEM bezeichnet wird. HIMEM ist also die höchste Adresse im System, an der Daten des Programmlistings im Speicher abgelegt werden können. Dieser HIMEM-Zeiger kann mit einem Befehl verändert werden um zum Beispiel Speicherplatz oberhalb der Sourcedaten für die Objectdaten frei zu halten.

Der Befehl hierzu lautet:

```
HIM <Adresse>
(von himem)
```

Bei Eingabe dieses Befehles wird der Speicherinhalt (das Listing) automatisch gelöscht. Vergessen Sie also nie, vor Aufruf dieses Befehles, das Listing, das sich im Speicher befindet auf Diskette oder RAMDISK abzuspeichern. Nach einladen des BIBO- ASSEMBLERS befindet sich der HIMEM-Zeiger immer genau unterhalb des Assemblers.

Der Zeiger für die unterste freie Speicherstelle im RAM nennt sich LOMEM und liegt normalerweise direkt oberhalb des DOS. Der Wert des LOMEM-Zeigers hängt also vom verwendeten DOS und von der Konfiguration des verwendeten DOS ab. Ab hier wird beim Assemblieren die Symbol-Tabelle abgelegt. Diese Tabelle besteht aus den im Programmlisting befindlichen Label und zu den Labels gehörenden Daten.

Auch der LOMEM-Zeiger kann im Speicher verschoben werden:

```
LOH <Adresse>
(von lomem)
```

heißt der hierzu gehörende Befehl. Durch verändern des Lomem-Zeigers bleibt das Programmlisting unbeeinflußt, lediglich die Symbol-Tabelle wird gelöscht.

Bei Veränderung des LOMEM- oder des HIMEM- Zeigers ist darauf zu achten, das der Zeiger nicht in einen Bereich gelegt wird, in dem wichtige Daten liegen (DOS, Assembler, Bildschirmdaten).

Zwischen der Endadresse der Symbol-Tabelle und der Anfangsadresse der Source-Data ist nun Platz für die Objectdaten beim Assemblieren. Wird allerdings beim Assemblieren ein Include-File nachgeladen, verschiebt sich kurzzeitig die Anfangsadresse der Source-Daten nach unten. Außerdem kann es passieren, daß bei längeren Listings die Symbol-Tabelle die Anfangsadresse der Objectdaten überschreitet. In jedem Fall wird hierbei eine Fehlermeldung ausgegeben so daß keine wichtigen Daten zerstört werden. Sollte es also vorkommen, daß für die Objectdaten kein Platz mehr im Speicher bleibt, können die Objectdaten natürlich auch gleich beim Assemblieren als Binärfile auf Diskette oder RAMDISK abgespeichert werden. Diese Files können direkt vom DOS aus wieder eingeladen und gestartet werden.

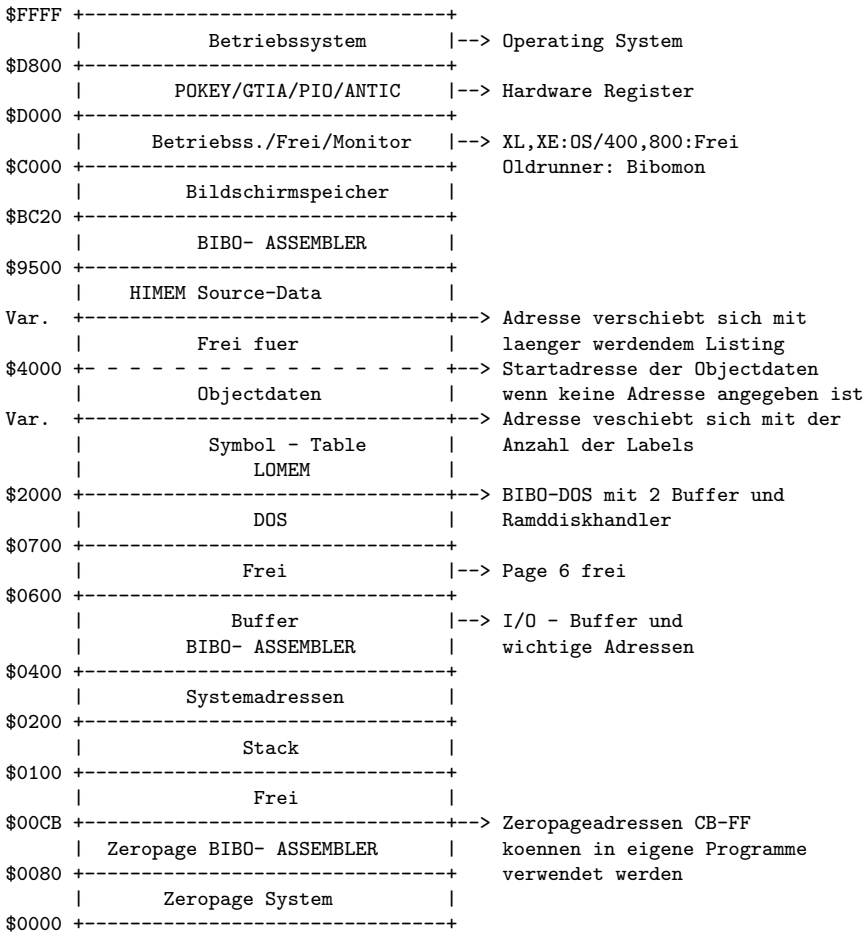
Kapitel 16

FEHLERMELDUNGEN DES ASSEMBLERS

- BAD INSTRUCTION ERROR: Der engegegebene Opcode oder Pseudo-Opcode existiert nicht.
- BAD LABEL ERROR: Der erste Buchstabe des Labels ist kein Buchstabe oder in einem Label befindet sich ein nicht alphanumerisches Zeichen.
- BAD LOCAL LABKL ERROR: Ein Local Label wurde vor einer .EQ (=1 Anweisung eingegeben.
- BAD ADDRESS ERROR: Hinter dem Opcode oder Pseudo-Opcode befindet sich keine oder eine ungültige Adressangabe.
- UNDEFINED LABEL ERROR: Der angeforderte Label befindet sich nicht in der Symbol Tabelle.
- DUPLICATE LABEL ERROR: Ein Label kommt in einem Assemblerlisting mehr als einmal vor.
- BRANCH TOO LONG ERROR: Ein Verzweigungsbefehl reicht weiter als +-127 Bytes.
- LOCAL OUT OF RANGE ERROR: Ein Local Label befindet sich mehr als 128 Bytes nach einem normalen Label.
- MEMORY STORAGE ERROR: Die Objectdaten werden in einen geschützten Speicherbereich geschrieben (Symbol Table/Source Data).
- LABEL NOT FOUKD ERROR: Vor einem .EQ (=) befindet sich kein Label.
- NESTED INCLUDE ERROR: In einer Includedatei wurde versucht eine weitere Includedatei aufzurufen.

Kapitel 17

Speicherübersicht



Alle Speicherbereiche des Assemblers dürfen unter keinen Umständen verändert werden.

Anhang A

Demo Programme

A.0.40 Demo Programm 1

```
00010 ******************
00020 *     DEMO1      *
00030 * ZEICHENAUSGABE *
00040 *     UEBER      *
00050 *   ROMROUTINE   *
00060 ******************
00070 ------------------------------
00080          .LI OFF
00090 ------------------------------
00100 HOLD     .EQ $F0        Hilfsregister fuer Y-Register
00110 ------------------------------
00120 START    LDY #1         ESC-Flag setzen
00130          STA $2FE
00140          LDY #0         Mit Zeichen Null beginnen
00150 .1       JSR CHAROUT    Zeichen ausgeben
00160          INY            naechstes Zeichen
00170          BNE .1         wiederholen
00180 ------------------------------
00190          STY $2FE       ESC-Flag zuruecksetzen
00200          RTS            PROGRAMMENDE
00210 ------------------------------
00220 CHAROUT  STY HOLD       Y-Register retten
00230          JSR ROMOUT     Zeichen ausgeben
00240          LDY HOLD       Y-Register resaturieren
00250          RTS            Zurueck
00260 ------------------------------
00270 ROMOUT   LDA $E407      Indirekt ueber Stack
00280          PHA            in die Zeichenausgabe
00290          LDA $E406      des Roms springen
00300          PHA
00310          TYA
00320          RTS
00330 ------------------------------
00340 *
```

```
00350 ;Programm wurde mit dem Textfile-
00360 ;Printer ausgedruckt
00370 *
00380 ------------------------------
```

A.0.41 Demo Programm 2

```
00010 *****************
00020 *    DEMO2      *
00030 *  TEXTAUSGABE  *
00040 *     UEBER     *
00050 *   ROMROUTINE  *
00060 *****************
00070 ------------------------------
00080          .LI OFF
00090 ------------------------------
00100 HOLD     .EQ $F0        Hilfsregister
00110 ------------------------------
00120 START    LDY #0
00130 LOOP     LDA TEXT,Y     Zeichen Holen
00140          BNE ZEICHNEN   Code Null?
00150          RTS            ja, Programmende
00160 *
00170 ZEICHEN  JSR CHAROUT    Ausgeben
00180          INY            Naechstes zeichen
00190          JMP LOOP
00200 ------------------------------
00210 CHAROUT  STY HOLD       Y-Register retten
00220          JSR ROMOUT     Zeichen ausgeben
00230          LDY HOLD       Y-Register restaurieren
00240          RTS
00250 ------------------------------
00260 ROMOUT   TAX
00270          LDA $E407      Indirekt ueber Stack
00280          PHA            in die Zeichenausgabe
00290          LDA $E406      des Roms springen
00300          PHA
00310          TXA
00320          RTS
00330 ------------------------------
00340 TEXT     .HX 7D   Clear Screen
00350          .AS "Dieser Text wurde ueber die Zeichen-"
00360          .HX 9B   Naechste Zeile
00370          .AS "ausgaberoutine des ROMs ausgeben. "
00380          .HX 00
00390 ------------------------------
00400 *
00410 ;Programm wurde mit dem Textfile-
00420 ;Printer ausgedruckt
00430 *
00440 ------------------------------
```

A.0.42 Demo Programm 3

```
00010 ******************
00020 *    DEMO3       *
00030 * SEKTOREN LESEN *
00040 *   UND AUF DEM  *
00050 *   BILDSCHIRM   *
00060 *    AUSGEBEN    *
00070 ******************
00080 ------------------------------
00090          .LI OFF
00100 ------------------------------
00110 SAVMSC   .EQ $58       Addresse des Bildschirmspeicher
00120 DUNIT    .EQ $301      Laufwerksnummer
00130 DCOM     .EQ $302      Diskkommando
00140 DBUF     .EQ $304      Datenpufferpointer
00150 DSEC     .EQ $30A      Sektornummer
00160 DISKIO   .EQ $E453     Einsprung in Disk I/O Rountine
00170 ------------------------------
00180 START    LDA SAVMSC    Bildschirm
00190          STA DBUF      als Datenpuffer
00200          LDA SAVMSC+1  fuer Sektordaten
00210          STA DBUF+1
00220          LDA #1        Laufwerksnummer
00230          STA DUNIT     und Sectornummer
00240          STA DSEC      auf 1 setzen
00250          LDA #0
00260          STA DSEC+1
00270          LDA #'R       Kommando $52 (READ Sektor)
00280          STA DCOM
00290 ------------------------------
00300 DISK     JSR DISKIO    Sektor in Buffer lesen
00310          BPL SECOK     Status positiv
00320          RTS           Nein. Programm abbrechen
00330 *
00340 SECOK    INC DSEC      Sektornummer
00350          BNE DISK      hochzaehlen
00360          INC DSEC+1    weitermachen...
00370          JMP DISK      .....
00380 ------------------------------
00390 *
00400 ;Programm wurde mit dem Textfile-
00410 ;Printer ausgedruckt
00420 *
00430 ------------------------------
```