

# **ASZMIC ROM**

## **Handbuch**

**profisoft GmbH**

Sutthäuser Str. 50-52

4500 Osnabrück

Tel. 0541-53905

EPROM 2532

J. ELRAD 83/8/41 Tooplerrill ASZ MIC.

8.7

1000 H

## Inhalt

Kapitel 1	Einführung und Überblick
Kapitel 2	Erste Schritte I
Kapitel 3	Erste Schritte II
Kapitel 4	Weitere DEBUG-Befehle
Kapitel 5	Textoperationen
Kapitel 6	Der Assembler
Kapitel 7	Programmausführung und Test
Kapitel 8	Graphik
Anhang 1	Gnerelle Informationen
Anhang 2	Die Shifttasten
Anhang 3	DEBUG-Befehle
Anhang 4	Systemadressen
Anhang 5	Z80-Befelssatz
Anhang 6	APPLICATION NOTES
	Beispiele

## Kapitel 1

### Einführung und Überblick

Wir beglückwünschen Sie zu Ihrem Kauf von ZX ASZMIC. Wenn Sie schon vorher mit Assembler- und Debugsystemen Bekanntschaft gemacht haben, können Sie sich mit den ersten der 3 Anhänge, die die Grundzüge von ASZMIC beinhalten, beschäftigen. Andernfalls geben Ihnen die nächsten 3 Kapitel, die eine Anzahl von Beispielen beihalten, eine Hilfe zur Anwendung von ASZMIC. Die Kapitel 5, 6 und 7 sind zum größten Teil eine Abhandlung über den Gebrauch; das Kapitel 8 zeigt Ihnen kurz die Graphikmöglichkeiten; und Anhang 4 bis 6 beinhalten gebräuchliche Informationen zum Nachschlagen.

Was genau ist ASZMIC? Es sollte ein möglichst billiges Assemblerprogramm sein. Aus diesem Grund gibt es Ihnen alle Möglichkeiten Programme zu schreiben, zu handhaben, zu assemblieren und laufen zu lassen, und sie in ihrer Entwicklung zu testen. Damit Sie Programme auf Ihrem mit ASZMIC verbundenen ZX 81 laufen lassen können, versuchen wir Ihnen gebräuchliche Routinen mit dem Monitor selber zu zeigen, um Ihnen zu helfen unabhängig von diesem Assemblerkurs Programme zu erstellen, und wir verwenden Bildschirmroutinen, um die Graphikmöglichkeiten, die in der Hardware Verwendung finden, zu demonstrieren. Anders als ähnliche Angebote ist ZX ASZMIC fast völlig unabhängig vom BASIC, und Sie sollten beachten, daß das Wesen Ihres Computers völlig verändert wurde. ZX ASZMIC entstand in Cambridge und verfolgt seitdem seine eigenen Wege. Wenn Sie schon Erfahrungen mit anderen Kleincomputersystemen gemacht haben, wird Ihnen vieles von ASZMIC bekannt vorkommen; aber wenn Sie bisher nur BASIC benutzt haben, wird Ihnen ASZMIC eine Fülle von Möglichkeiten eröffnen, die Sie nie mit Basic erreicht hätten, da jede Tätigkeit erst den Interpreter durchlaufen müßte. Dies hat sowohl gute als auch schlechte Seiten; ASZMIC kann bei der Ausführung eines Kommandos sagen wir leichter als andere Systeme zusammenbrechen. Glücklicherweise passiert dies nur sehr selten.

## 1.1 Einbau

ASZMIC ist ein direkter Ersatz für das BASIC ROM im ZX 81. Um es in Ihrem Computer einzubauen, müssen Sie zuerst das Gehäuse entfernen, indem Sie die 5 Schrauben von der Unterseite Ihres ZX 81 lösen; drei davon befinden sich unter den Gummifüßen. Dann müssen Sie die 3 Schrauben, die die beschriebene Platine auf dem Deckel befestigen, lösen, wobei Sie darauf achten müssen, nicht die Litze der Tastaturverbindung herauszuziehen, sondern sie auf die Platine zurückzufalten, damit sie dort bleiben und den Deckel auf einen sauberen Untergrund zu legen. Wenn Sie die Verbindungen doch herausziehen sollten, wird es Ihnen viel Mühe machen, sie wieder hineinzustecken.

Auf der anderen Seite der Platine befindet sich neben den Anschlüssen zum und vom Kassettenrekorder und dem Stromanschluß das ROM ganz am äußeren Rand (wenn die Anschlüsse nach oben zeigen, ist das ROM der zweite Chip von oben). Im ZX 81 Handbuch befindet sich eine entsprechende Zeichnung auf Seite 162. Trotz moderner Technik empfiehlt es sich vorsichtig zu sein und deshalb geerdete Aluminiumfolie oder einen Spülstein als Untergrund zu benutzen. Vermeiden Sie es außerdem, Nylonsachen zu tragen und erden Sie sich selbst bevor Sie irgendeinen Schaltkreis außerhalb seines Sockels berühren. Das BASIC ROM befindet sich wahrscheinlich auf einem 28er Sockel, obwohl es nur 24 Pinne besitzt, so daß über dem ROM vier Löcher frei sind. ASZMIC sollte nun so auf den Sockel gesetzt werden, daß ASZMIC möglichst nach außen gerückt wird (ähnlich dem ROM), da es ebenfalls 24 Pinne besitzt und somit auch 4 Löcher darüber frei bleiben. Heben Sie das ROM vorsichtig mit einem Schraubenzieher heraus, und legen Sie es an die Seite. Gehen Sie nicht zu hastig vor, um keine Pinne abzubrechen. Legen Sie es möglichst auf eine Folie. Biegen Sie nun die Pinne von ASZMIC etwas nach innen, und drücken Sie es in den Sockel, damit die Kontakte verbunden werden. (Drücken Sie es jedoch auch nicht zu fest herein.)



Vor dem Zusammenbau empfiehlt es sich zu prüfen, ob auch wirklich beim Einschalten ein blinkender Cursor erscheint. Wenn dies nicht der Fall ist versuchen Sie ein paar mal, ASZMIC neu zu platzieren.

Wenn Sie vorhaben sowohl ASZMIC als auch BASIC wahlweise zu benutzen, empfiehlt es sich eine Platine zu kaufen, auf der beide ROM's Platz finden. Nähere Informationen darüber können Sie von Capitel Computers Ltd erhalten. Bei der Entwicklung von ASZMIC schnitten wir ein Loch in den Deckel des ZX 81 (von 4 bis 6 cm mal 5cm in die Vorderseite, obwohl ein Schnitt von 1 mm in die Tastatur einen Ausfall zur Folge hat.) und platzieren einen speziellen Sockel auf dem des ZX 81 ROM's. Dies funktionierte erstaunlich gut, aber eine ROM-Platine sorgt dafür, daß Sie bei einer Übertragungskontrolle die Fähigkeit haben Speicherregionen zu schützen.

## 1.2 ZX 80-Einbau

Die Schaltkreise des ZX 80 sind zugänglicher als die des ZX 81; Sie brauchen nur die Plastiknieten zu entfernen, um den Deckel abzunehmen; das ROM befindet sich direkt auf der richtigen Seite. ASZMIC wird zum UHF-Modulator (Metallkiste) hin eingefügt. Es ist nicht möglich das G-Kommando auszuführen, und der Bildschirm wird bei EDIT-Operationen nicht konstant bleiben.

## Kapitel 2

### Erste Schritte I

#### 2.1 Nach dem Einschalten

Nachdem Sie den ZX 80/81 mit dem Netz verbunden haben, erscheint ein weißes Bild, außer einem kleinen Zeichen in der linken Ecke (Datenende) und einem blinkendem Cursor darüber. Die Blinkgeschwindigkeit gibt an, in welchem Modus sich ASZMIC befindet; ein schnelles Blinken bedeutet EDIT, ein langsames bedeutet DEBUG. Mit einem 16 K RAM befinden Sie sich im DEBUG-Modus; mit der 1 K RAM-Grundversion arbeitet das System im EDIT-Modus. Die Umschaltung zwischen EDIT- und DEBUG-Modus erfolgt durch die Tastenfolge Shift 9 und Shift E. Probieren Sie das Drücken von Shift 9 (DEBUG) und Shift E (EDIT) wahlweise aus, und betrachten Sie die Blinkdauer in dem jeweiligen Modus. Es besteht nur ein Unterschied zwischen EDIT- und DEBUG-Modus, der jedoch sehr wichtig ist. Wenn Sie N/L drücken, wird die gerade beendete Zeile im DEBUG-Modus dem Kommandointerpreter übergeben. Im EDIT-Modus beginnt eine neue Zeile.

#### 2.2 Editieren

Drücken Sie Shift E, um in den EDIT-Modus zu gelangen. Der Cursor beginnt schneller zu blinken. Nun drücken Sie A. Ein "A" erscheint auf dem Bildschirm, und der Cursor rückt eine Stelle nach rechts. Drücken Sie A erneut, wobei Sie dieses Mal Ihren Finger auf der Taste lassen. Nach einer halben Sekunde beginnt ASZMIC mit einem Takt von acht Zeichen pro Sekunde, A's zu schreiben. Nehmen Sie Ihren Finger von der Taste, bevor die Zeile voll ist. Dies demonstriert das in ASZMIC eingebaute Autorepeat. Es funktioniert mit allen Tasten. Drücken Sie nun N/L und darauf für etwa ein bis zwei Sekunden B, wieder N/L und darauf für etwa die gleiche Zeit C und zum Schluß erneut N/L. Wiederholen Sie diesen Prozeß, bis etwa 5 bis 6 Zeilen auf dem Bildschirm zu sehen sind.

Beachten Sie, daß der Bildschirm nach jedem N/L um eine Zeile nach oben rollt (scrolled). Erzeugen Sie jetzt eine Schlußzeile von sagen wir G's, ohne N/L am Schluß zu drücken. Stattdessen drücken Sie Shift 5 (Cursor nach links) und halten Ihre Finger weiterhin auf den Tasten. Der Cursor wandert nach links bis zum Zeilenanfang. Drücken Sie Shift 8 (Cursor nach rechts) kontinuierlich und beobachten Sie, wie der Cursor nach vorne bis zum Zeilenende wandert. Drücken Sie erneut Shift 5 (Cursor nach links) bis sich der Cursor in der Mitte der Zeile befindet. Drücken Sie darauf 12345. Diese Zahlen fügen sich in der Zeilenmitte ein. Was eigentlich geschieht, ist, daß alles unterhalb und rechts vom Cursor nach rechts geschoben wird; der Buchstabe, den Sie gedrückt haben, befindet sich unter dem Cursor, und der Cursor wandert eine Stelle nach rechts.

### 2.2.1 Löschen

Drücken Sie Shift Ø (RUBOUT). Die 5, die Sie eben erzeugt haben, verschwindet. Dies ist die erste Lösch Taste in ASZMIC, um Buchstabenfehler zu löschen. Die Zeichen links vom Cursor werden gelöscht, und alles hinter ihm wird um eine Stelle nach links gerückt. Drücken Sie nun zweimal Shift 5, um den Cursor über 3 zu plazieren. Drücken Sie Shift Q (EDIT RUBOUT), und beobachten Sie das Verschwinden der 3. Dies ist die zweite Taste zum Löschen; das editierende RUBOUT. Der Unterschied besteht darin, daß Shift Ø brauchbarer bei einer neuen Zeile ist, während Shift Q besser zum Editieren eines Files ist. Experimentieren Sie nun ein wenig nach Ihrem eigenen Geschmack. EDIT RUBOUT funktioniert nicht, wenn die Zeile nur aus einem Zeichen besteht. Sie können mit Shift Ø N/L's löschen. Merken Sie sich, daß Shift-Q auch die gepufferten Leerzeichen, die sich bei ASZMIC vor einem N/L befinden, was Probleme beim Assemblieren und Mischen erzeugen können, löschen kann.

### 2.2.2 Vertikale Cursorsteuerung

Schalten Sie Ihren ZX 80/81 aus und darauf wieder ein (RESET). Wir werden Sie noch sehr oft bitten, dies zu tun, da viele Beispiele von ASZMIC den Speicher der 1 K-Version erschöpfen.

Drücken Sie nun Shift T (TOP). Der Cursor und das Bild werden an die oberste Spitze gesetzt. Dies ist nützlich, wenn Sie innerhalb größerer Texte etwas suchen, und es ist noch wichtiger, da die oberste Zeile eine besondere Bedeutung hat. Es handelt sich um die Macro-Zeile, aber mehr darüber später. Drücken Sie nun Shift 9 (DEBUG), und beachten Sie, wie Sie runter zur Grundzeile springen und in den DEBUG-Modus wechseln. Drücken Sie Shift E (EDIT) und beachten Sie hierbei, daß nicht nur auf EDIT (schnelleres Blinken) umgeschaltet wird sondern auch zum Punkt, an dem Sie waren bevor Sie zuletzt Shift 9 gedrückt haben, gesprungen wird. Dies ermöglicht Ihnen zwischen EDIT- und DEBUG-Modus umzuschalten ohne die ganze Zeit nach Ihrer Position zu suchen.

Schalten Sie mit Shift 9 in den DEBUG-Modus um. Drücken Sie D Ø N/L. ASZMIC antwortet mit ØØØØ F5. Halten Sie Ihren Finger auf N/L bis ASZMIC ØØ3Ø 3E schreibt, und drücken Sie . (Punkt), gefolgt von N/L, gefolgt von E N/L, um zurück in den Edit-Modus zu gelangen. Eigentlich haben Sie die ersten 49 Monitorpositionen vollgeladen und verändert (Dump & Modify), aber wir haben dies nur gemacht, um mit dem bischen Text herumzuspielen. Drücken Sie jetzt Shift 7 (Cursor nach oben) und lassen Sie die Taste weiterhin gedrückt. Sie können nun beobachten, daß der Cursor nach oben bis zum Bildschirmanfang wandert, und schauen Sie dann, wie der Text nach unten rollt (scrolled) bis er sich auf einer Leerzeile befindet. Halten Sie nun Shift 6 (Cursor nach unten) gedrückt, und Sie werden sehen, daß der Cursor nach unten bis zur letzten Zeile genau über dem Zeichen des Datenendes (End-of-Data) wandert. Er bewegt sich nicht weiter. Drücken Sie nun eine Zeit lang Shift 4 (Page up - Seite hoch) und sehen Sie, wie Sie den Text nach oben durchlaufen. Bei Shift 3 (Page down - Seite runter) kehrt sich der Prozeß um.

### 2.2.3 Löschen einer Zeile

Steuern Sie den Cursor hoch nach ØØ2Ø 7E oder in die Nähe. Drücken Sie jetzt Shift 1 (Zeilenlöschen). Die Zeile verschwindet. Wiederholen Sie diesen Prozeß, und die nachfolgenden Zeilen verschwinden ebenfalls bis Sie wieder über dem Datende stehen (End-of-Data). Sie können diese letzte Zeile nicht mit Shift 1 löschen; Sie müssen sie durch überschreiben mit anderen Zeichen löschen.

Dies ist so, damit das Zeichen des Datenendes (End-of-Data), das ein "STOP" des "DUMP" darstellt, geschützt wird. Drücken Sie jetzt Shift 2 (löschen des Datenfiles). Der ganze Text vom Cursor bis zur Filemarkierung wird gelöscht.

#### 2.2.4 Files Und Filemarkierungen

Anders als das BASIC ROM, legt ASZMIC keinen speziellen Bildschirmfile an sondern es benutzt den größten Speicherteil als ein großes Textfeld, woraus der Bildschirm jeweils einen kleinen Ausschnitt zeigt. Eine Menge Operationen werden mit einem speziellen Steuerzeichen kontrolliert, dem £ (Sterling-) Zeichen, welches als Begrenzer fungiert. Da separate Textblöcke durch dieses £ als Files behandelt werden, nennt man dieses Zeichen Filemarkierung (filemark). Es benötigt sich selbst, um das Ende eines Files anzuzeigen, und es wird benötigt zum Drucken, Assemblieren, Saven auf Kassette, Mischen und für Löschoptionen bzgl. des Files, um ASZMIC zu sagen, wann es aufhören soll. Sie zeigen den Beginn eines Files mit einem dazu ähnlichen Namen an, welcher nicht innerhalb des Files erscheint. Es ist empfehlenswert eine Filemarkierung als erstes Zeichen dieses Namen zu benutzen. Dies :-

```
£NURSERY.RHYME
1
2
BUCKLE MY SHOE
3
4
KNOCK ON THE DOOR
£
```

ist ein Beispiel eines Files und kann gedruckt, gesaved, editiert und ähnliches werden unter Verwendung des Namen £NURSERY.RHYME. Ein Filename kann irgendeine Kombination aus alphanumerischen Zeichen (0-9 A-Z) und Punkten sein und wird mit irgendeinem anderem Zeichen beendet.

Setzen Sie den Cursor nach Home (Shift 9) und drücken Sie £ N/L. Dann benutzen Sie Shift E, um in den EDIT-Modus zu gelangen und den Cursor bis er blinkt auf das D der ursprünglichen D Ø Zeile an der Spitze von DUMP zu bewegen. Drücken Sie darauf Shift-2 (File löschen). Der gesamte Text vom Cursor bis zur Filemarkierung verschwindet.

### 2.2.5 Wenn der Speicherplatz knapp wird

Reseten (aus und wieder anschalten) Sie Ihren ZX 80/81 und drücken Sie Shift 9 (DEBUG, wie Ihnen wahrscheinlich jetzt bekannt ist). Drücken Sie D Ø 5ØØØ N/L. Der Bildschirm wird für mehrere Sekunden leer, da ASZMIC 625 Zeilen des formatierten DUMP's für Sie generieren muß. Unglücklicherweise beansprucht dies mehr als ein 16 K RAM-Pack an Text fassen kann. Wenn der Bildschirm wieder freigegeben wird, geben Sie ein Zeichen ein. Nichts passiert. Sie können den Cursor hin- und herbewegen (versuchen Sie es), aber Sie können keinen Text einfügen. Um ASZMIC weiter zu gebrauchen müssen Sie etwas Text löschen. Bewegen Sie den Cursor 4 bis 5 Zeilen nach oben (Shift 7), und halten Sie dann Shift 1 ständig gedrückt. Sie werden damit die letzten paar Zeilen löschen. Sie können nun Zeichen einfügen (100 oder mehr) bis Sie erneut keinen Speicherplatz mehr haben.

### 2.2.6 MACROS

Wir sollten diese Masse Text für etwas benutzen. Drücken Sie Shift T. Dies bringt den Cursor nach oben zur obersten Zeile (Top line). Drücken Sie nun E 40 aber, da Sie sich im DEBUG-Modus befinden, kein N/L um Kommandos auszuführen, sondern drücken Sie stattdessen Shift 9 (DEBUG), um den Cursor auf der Endzeile zu stationieren. Benutzen Sie nun Shift R (Shift macro).

Das Resultat davon ist, daß die oberste Zeile (top line) als ein Debug-Befehl ausgeführt wird, unabhängig vom Modus, in dem Sie sich befinden. E 40 ist ein DEBUG-Kommando, um in den EDIT-Modus zu schalten, und den Cursor beim ersten Erscheinen des Strings "40" an den Anfang der Zeile zu setzen; der schnell-blinkende Cursor befindet sich also auf der "4". Drücken Sie mehrmals Shift R und sehen Sie wie "40" bei DUMP erfolgreich heraus-gesucht wird. Sie können fast jedes DEBUG-Kommando oder oder Verkettungen davon an der obersten Zeile (top line) ausführen und so eine Menge Arbeit mit einem Tastendruck ausführen. (Es gibt eine andere Art von MACRO, ein MACRO-Kommando, das in ASZMIC eingebaut wurde, und wir werden bei der Besprechung über das Mischen von Files zurückkommen). Löschen Sie jetzt mit Shift 9 und N/L

das DUMP- Kommando (§ gilt nicht als solches und wird nicht beachtet vom Befehlsinterpreter), drücken Sie Shift T, und benutzen Sie dann Shift G, um den Cursor nach unten bis zum "D" von D Ø 5ØØØ zu bewegen, und drücken Sie zum Schluß Shift 2 (Löschen eines Files). Jetzt sollte alles gelöscht sein.

### 2.2.7 Mischen (Merging)

Reseten Sie den ZX 80/81. Drücken Sie Shift E (EDIT) und darauf folgendes:

```
>DØ1 N/L
DØ2 N/L
DØ3 N/L
DØ4 N/L
DØ5 N/L
& N/L
```

Drücken Sie nun Shift G (merge) einmal. Alles zwischen > und & wurde dupliziert. Versuchen Sie es noch einmal. Jedes Mal, wenn Sie Shift G drücken, werden weitere 5 Zeilen zur Cursorposition kopiert. Bewegen Sie den Cursor hoch zum "D" von einem DØ3, und drücken Sie Shift G. Die 5 Zeilen werden nun vor DØ3 eingefügt, weil sich der Cursor dort befindet. Drücken Sie jetzt Shift 9 (DEBUG).

Im EDIT-Modus ist Shift G eine reine Mischtaaste. Jetzt, im DEBUG-Modus, ist der Effekt völlig anders, weil ASZMIC jedesmal ein N/L zum Tastenfeld schreibt, was im DEBUG-Modus zur Kontrollé durch den Kommandointerpreter führt. Drücken Sie Shift G. Jede "D"-Zeile wird kopiert, aber da jede Zeile ein DUMP-Kommando ist, folgt ihr das entsprechende DUMP. Durch Drücken einer Taste führen wir 5 Zeilen der DEBUG-Kommandos aus. Dies wird MACRO-Kommando genannt.

Normalerweise mischt man im EDIT-Modus, um ein Stück eines Textes zu einem anderen zu kopieren. Sie identifizieren den zu bewegendenden Text durch > und &, setzen den Cursor an die Position, in die Sie hineinmischen wollen, drücken Shift G, und dann, wenn Sie den Ursprungsort löschen wollen, stellen Sie den Cursor auf > und drücken Shift 2 (Löschen eines Files).

Einfach, dehnbar und für die faulen Arbeiter, die ASZMIC geschrieben haben, leicht zu implementieren. Gehen Sie Anhang 4 für die Symbole Shift D & Shift F durch. Wenn Sie erscheinen haben Sie drei Misch Tasten. Shift-D entspricht Shift-G aber benutzt \* als Mischbeginnkennzeichen und Shift-F benutzt <.

### 2.3 Was kommt als nächstes?

Sie haben nun alle Shift-Tasten zur Kontrolle der EDIT-Funktionen kennengelernt. Wenn Sie sich Anhang 2 anschauen, werden Sie eine kurze Zusammenfassung über jedes Shift finden. Versuchen Sie selbst Text zu erzeugen und damit zu arbeiten. Dann schauen Sie sich Anhang 3 als Zusammenfassung aller DEBUG-Kommandos an. Wir werden ein bisschen geschlossen vorgehen und auch nicht alphabetisch, da einige der Befehle viel komplexer als andere in ihrer Ausführung sind.



## Kapitel 3

## Erste Schritte II

## 3 Einfache DEBUG-Befehle

Alles was Sie im EDIT-Modus machen können, können Sie auch im DEBUG-Modus ausführen. Der Unterschied besteht darin, daß im DEBUG-Modus nach jedem N/L der Befehlsinterpreter aufgerufen wird. Wenn Sie jedoch im DEBUG-Modus editieren, können Sie sich nicht darauf verlassen, mit Shift E (EDIT) zurück zu dem Punkt zu gelangen, wohin Sie vom Edit-Modus aus gelangen würden. Sie gelangen an den Ort zurück im Speicher, wo sich der Cursor be-  
fand, aber das Editieren ist hier völlig anders (so wie bei N/L, wo der Cursor nicht erscheint).

## 3.1 D für DUMP

Reseten Sie Ihren ZX81 (Ein- und Ausschalten). Drücken Sie Shift 9, um sicher zu gehen, daß Sie sich im DEBUG-Modus befinden. Drücken Sie D :4300 12 N/L, und auf dem Bildschirm erscheint:-

```
4300 00 00 00 00 00 00 00
4308 00 00 00 00 00
```

Dies ist ein Beispiel für ein DUMP-RANGE-Befehl. Es handelt sich um einen wichtigen neuen Gedanken, dem des Feldes. Das Kommando selber besteht nur aus dem Buchstaben D, der von einem Feld gefolgt wird, welches die Startadresse des DUMP's definiert, und dann folgt ein Feld, das die Anzahl der Bytes, die benötigt werden, festlegt. Der : (Doppelpunkt) sagt aus, daß 4300, das danach folgt, einen hexadezimalen Wert darstellt. Die Definition der einzelnen Feldarten von ASZMIC können Sie in Anhang 1 nachschlagen. Der Gebrauch definierter Felder ist in ASZMIC durchgehend gleich mit einer Ausnahme, die jetzt zur Sprache kommt.

Felder werden mit einem Komma oder durch ein oder mehrere Leerzeichen abgetrennt, aber nur mit einer der beiden Möglichkeiten. Sie können das erste Feld direkt hinter dem Buchstaben des DEBUG-Modus definieren, aber ein Leerzeichen danach ist übersichtlicher.

Es gibt noch eine Form von DUMP, das wir DUMP & MODIFY nennen. Drücken Sie D :4300 N/L.

ASZMIC antwortet mit:

4300 00

und der Cursor wartet auf der Eingabezeile. Drücken Sie N/L zwei oder drei mal. Mit jedem Druck erscheint die Adresse und ihr Inhalt, so wie hier:

4301 00

4302 00

4303 00

Wie kommen Sie wieder hier raus? Drücken Sie . (Punkt) und darauf N/L, und Sie gelangen wieder in den gewöhnlichen DEBUG-Modus zurück. Wie kommt man jetzt zum MODIFY? Drücken Sie D :4300 N/L erneut. Nun geben Sie 1 4 7 C 12 N/L ein und drücken darauf . (Punkt) N/L. Dann machen Sie einen DUMP RANGE der Form D :4300 6 N/L. Die Antwort lautet:

4300 01 04 07 0c 12 00

Vergleichen Sie dieses Resultat mit dem beim ersten D :4300 12, das Sie eingegeben haben. DUMP & MODIFY bildet eine Ausnahme unter den ASZMIC-Feldern und den dazugehörigen Regeln, die wir schon vorher erwähnt haben. Bei Zahlen wird auch ohne : (Doppelpunkt) als Vorzeichen eine hexadezimale Eingabe vorausgesetzt. Bei einer Dezimaleingabe müssen Sie O+Dezimalzahl verwenden. Für eine bessere Demonstration müssen Sie Ihren ZX81 reseten, in den DEBUG-Modus schalten, 15 oder 20 mal  $\&$  drücken, nach der Adresse von DSPBGN in Anhang 4 schauen und ihr 55 hinzufügen (z.B. 40B4 +55). Nach Beendigung der  $\&$ -Zeile mit N/L müssen Sie D :40B4 +55 N/L drücken und darauf 1C N/L 1D N/L usw. Nun können Sie sehen, daß  $\&$ -Zeichen von Zeichen, die durch den eingetippten Code beeinflusst werden, überschrieben werden.

## 3.2 C für COPY

Reseten Sie Ihren ZX81 (bei einem 16 K RAM ist das nicht immer notwendig). Schalten Sie in den DEBUG-Modus (Shift 9). Drücken Sie darauf folgende Befehle:

```
D: 4300 20 N/L
D: 4300 N/L
1 2 3 4 5 6 7 8 N/L
. N/L
D: 4300 20 N/L
C: 4300 :4308 N/L
D :4300 20 N/L
```

Holen Sie tief Luft, und schauen Sie auf Ihr Werk. Als erstes haben Sie 20 Adressen mit dem Inhalt 0 (wegen des Ausschaltens) gedumped. Darauf haben Sie ein DUMP & MODIFY mit den ersten 8 vollzogen und ein DUMP RANGE, um sich zu vergewissern, daß Sie modifiziert wurden. Darauf haben Sie 8 Bytes von der Adresse : 4300 zur Adresse :4308 (die Bytes wurden aufsteigend in die Adressen kopiert - z.B. 4308, 4309, ...) kopiert, und schließlich haben Sie die 20 Adressen wieder gedumped, um sich zu vergewissern, daß die Kopie gelungen ist.

Das C für COPY kommt von Zähler (Counter). Es ist klar, daß man sich bei überlappenden Speicherregionen versichern muß, daß alles richtig zur Zieladresse übertragen wird. Beim Kopieren geht ASZMIC vom Anfang bis zum Ende oder ähnliches durch, um sicher zu gehen, daß die Ursprungsadressen nicht vor der Kopie von dem Inhalt der Zieladressen überschrieben wurden.

## 3.3 F für FILL

Schalten Sie nach dem Reseten in den DEBUG-Modus. Drücken Sie:

```
F :4300 :4310 :AA N/L
```

Drücken Sie darauf D :4300 16 N/L, und betrachten Sie, daß Sie dieses Gebiet mit AA's gefüllt wurde. Das Format ist Evon bis Füllbyte. Der Befehl wird meistens dazu benutzt, Speicherregionen zu initialisieren. (Es ist nicht nötig, wenn mit Null gefüllt wird, es hinzuschreiben, da Null als Füllbyte angenommen wird,

wenn kein drittes Feld vorhanden ist.)

### 3.4 E für EDIT

E selbst bewirkt einen Wechsel in den EDIT-Modus. Wenn ihm aber ein Zeichen folgt, sucht ASZMIC nach dem Zeichen und setzt den Cursor an den Anfang dieser Zeile. Dies ist gut, um zu einem File bei einer Menge Text zu gelangen. Wir haben diesen Befehl schon in 2.2.6 MACROS benutzt, und Beispiele erübrigen sich somit. Schauen Sie sich die Unteroutine CMPSTR (gebraucht von E und vielen anderen ASZMIC-Befehlen an, um zu sehen, was ein gültiger Vergleich bewirkt.

### 3.5 H für unbedingter Sprung (HORRIBLE JUMP)

Die benötigte Form ist H Feld, und das Resultat ist ein Sprung zu der Adresse, die durch das Feld bestimmt wird, wenn es sich im Befehlsinterpreter befindet. Bei dieser Adresse befindet sich vermutlich ein Programm, das nach seiner Ausführung durch RET schließlich zu ASZMIC direkt zurückkehrt. Ihre Routine kann weitere Felder in der Zeile analysieren mit GETFLD, so daß wir eine Methode haben, Ihre eigenen Befehle zu verbinden. Wenn Sie ein Beispiel benötigen, schauen Sie nach der Adresse von INICON in Anhang 4 und fügen dieser Adresse ein H zu (erinnern Sie sich an :). INICON beginnt mit der Initialisierung von ASZMIC und erzeugt somit den gleichen Effekt wie Ein- und Ausschalten.

### 3.6 M für MACRO

Dies ist eine Erweiterung zu Shift 6. Reseten Sie Ihren ZX81 und schalten Sie in den DEBUG-Modus.

Drücken Sie:

=D Ø8 N/L

⌘ N/L

+D 16 8 N/L

⌘ N/L

Drücken Sie dann M= N/L mehrmals und M+ N/L einige Male. Die spezifizierten Dump's werden generiert. Das Zeichen nach M legt das Startzeichen des Befehls MACRO fest, anders ist es bei Shift G, was immer > als Startzeichen benötigt wird. So haben Sie eine große Auswahl an MACRO's zur Verfügung.

### 3.7 O für alte Register (old registers)

Im DEBUG-Modus müssen Sie O N/L drücken. Sie erhalten ein paar eingerückte Zeilen auf denen sich 6 4-stellige Hexadezimalzahlen befinden. Dies sind die Register, die im Feld REGIM festgehalten werden. Sehen Sie sich die Definition von O in Anhang 3 an, das sagt was was ist. Dies hat kaum eine Bedeutung, bis Sie das Programm starten, und es breaken, was zur Folge hat, daß die Registerinhalte nach REGIM übertragen werden. Es ist ein sehr guter Gedanke in die Shift Macro-Zeile ein O zu stellen, wenn ASZMIC gestartet wird, da die Shift Macro-Zeile nach jedem BREAK ausgeführt wird und die Registerinhalte nach einem solchen BREAK normalerweise interessant sind.

Wenn Sie Lust haben, können Sie in Anhang 4 die Adresse für REGIM nachschlagen und einige Adressen zwischen REGIM und REGIM +24 ändern, um den Effekt bei den DUMP-Zeiten der alten Register zu beobachten.

### 3.8 P für PRINT

Wenn Sie keinen Drucker besitzen, hat der Befehl keine Auswirkung. Wenn Sie jedoch einen besitzen, reseten Sie Ihren ZX81, und schalten Sie in den EDIT-Modus. Drücken Sie:

```
$PRINT.FILE
  ANY
  SORT
  OF
  RUBBISH
  WILL DO
  $
```

und drücken Sie Shift 9, um in den DEBUG-Modus zu gelangen.

## Kapitel 4

## Weitere Debug-Befehle

## 4 SAVEN &amp; LADEN

Es ist jetzt Zeitsich mit dem ASZMIC-Kassetteninterface zu beschäftigen. Es wird die gleiche Kassettenansteuerung benutzt wie beim BASIC-ROM, so daß es möglich ist, einige Tricks des Kapitels Text-Operationen zu verwenden, um ein in BASIC geschriebenes Programm zu laden, es zu ändern mit ASZMIC und zum Beispiel die REM-Zeilen mit Maschinencode vollzuschreiben, und es zurückzuschreiben, damit es wieder mit BASIC geladen werden kann. Der Nachteil dieser Kompatibilität ist die Langsamkeit des Interfaces, aber Sie sind das ja auch im Moment gewohnt.

Wenn ASZMIC einem File auf Kassette schreibt, druckt es zuerst eine Titelzeile zur Identifikation des Programmes oder des folgenden Files, wartet 5 Sekunden und schreibt dann den File nach draußen oder in die Speicherregion, die Sie angegeben haben. Wenn Sie von einer Kassette laden, erfaßt ASZMIC die Titelzeile, schreibt Sie auf den Bildschirm und spielt Ihnen nach 5 Sekunden Pause die Daten vor. Auf diese Weise können Sie einen Katalog von allem, was Sie auf Kassette gespeichert haben, anlegen, welcher sehr informativ und beruhigend ist, da Sie wissen, daß Ihre Daten nicht für immer verschwunden sind.

## 4.1 K für Kassette

Reseten Sie Ihren Zx 81, schalten Sie in den EDIT-Modus, und geben Sie ein File wie das folgende ein:

```

ALIMERICK
THERE WAS A OLD MAN OF DUMBREE
WHO THAUGHT LITTLE OWLS TO DRINK TEA
FOR HE SAID TO EAT MICE
IS NOT PROPRER OR NICE
THAT AMIABLE MAN OF DUMBREE
£

```

K&S "POETRY" 4LIMERICK

und wird wieder leer. Ein wenig später kommt ein weiteres 5-Sekundenstück, aber dieses Mal fügt sich die Zeile:

K&S "MEMSAVE" :4300 :4320

hinzu. Da es keinen File, der "NOTHING" heißt, auf der Kassette gibt, sollten Sie BREAK drücken, um zurück in den EDIT-Modus zu gelangen. Spulen Sie jetzt zurück, schalten Sie in den DEBUG-Modus, und wiederholen Sie die gleiche Prozedur wie oben, aber diesmal mit:

L "POETRY" N/L

um den ersten gesaveten File zu laden. Kontrollieren Sie, ob :4300 bis :4320 Nullen enthalten (D :4300 32 N/L) und laden Sie das zweite File mit:

L "MEMSAVE" N/L

Übertragen Sie :4300 nach :4320, um nachzuprüfen, daß Sie jetzt die Bytes "BB" enthalten.

Leicht?

es fehlt 4.3  
4.4

Gehen Sie jetzt in den DEBUG-Modus, verbinden Sie Ihren Kassettenrekorder wie beim BASIC, und tippen Sie ein:

K+S "POETRY" \$LIMERICK N/L

Die Folge K+S Leerzeichen " ist sehr wichtig. Wenn Sie es nicht so schreiben (mit Leerzeichen zwischen s und "), erkennt es ASZMIC nicht als Filenamen an, und Sie werden das File nicht mehr einlesen können.

Nachdem Sie N/L gedrückt haben, haben Sie 5 Sekunden Zeit, um Ihren Kassettenrekorder zu starten. Es schadet nichts, wenn Sie es früher machen. Der Bildschirm wird für eine halbe Sekunde leer und die Titelzeile wird ausgegeben. Das Bild kommt zurück für weitere 5 Sekunden, und dann wird das File selber ausgegeben. Das Bild kommt endgültig wieder, wenn das Saven beendet ist. Sie können das Saven jederzeit mit BREAK abbrechen. Stoppen Sie den Rekorder.

Geben Sie nun ein F :4300 :4320 :BB N/L, um ein Speichergebiet mit dem Code von BB zu füllen. Saven Sie anders, aber dieses Mal ein Speichergebiet:

k+S "HEMSAVE" :4300 :4320 N/L

Die Kassettenhandhabung verläuft wie oben. Spulen Sie zurück.

#### 4.2 L für Laden (LOAD)

Reseten Sie den ZX81 und schalten Sie in den DEBUG-Modus. Drücken Sie:

L "NOTHING" N/L

und starten Sie den Rekorder mit der gleichen Lautstärke wie beim BASIC, um das gesavete wieder abzuspielen. Nach einer Weile leuchtet der Bildschirm für 5 Sekunden mit:

Forts. 18



Drücken Sie P PRINT.FILE N/L, und das File wird gedruckt. Wenn Sie den Druckvorgang beenden wollen (z.B. wenn Sie vergessen haben, daß einzugeben) brauchen Sie nur die BREAK-Taste zu drücken.

Dies schließt die einfachen DEBUG-Befehle ab. Im nächsten Kapitel werden wir uns mit etwas mehr komplexeren und interessanteren Thema beschäftigen.

## 4.5 A für ASSEMBLE

Der ASZMIC-Assembler wurde aus einem 1 K-Assembler (in der Tat, 1 K), geschrieben für Nascom 1, entwickelt (und wird normalerweise mit gemischten Gefühlen "Dirty Dog"-Assembler genannt). Es handelt sich um einen 2-Weg-Assembler, der die vollständigen Zilog Mnemonics gebraucht. Wenn Sie nicht mit Zilogs Assemblersprache vertraut sind, ist es ein guter Gedanke, sich damit zu beschäftigen (z.B. durch den Kauf eines Buches darüber). Einige Quellen dazu befinden sich in Kapitel 26 des ZX81-Handbuchs.

Reseten Sie Ihren ZX81, schalten Sie in den EDIT-Modus, und drücken Sie folgende Tasten der Reihe nach (Wir setzen jetzt voraus, daß Sie wissen, daß jede Zeile mit N/L beendet wird):

**FILE**

ORG :4300

START LD HL,0

LD DE,0

LD B,10

LOOP INC DE

ADD HL,DE

LOOPEND DJNZ LOOP

RST 0

**f**

Dies ist ein Programm. Beginnen Sie jedes Programm mit der Anweisung ORG, um dem Assembler mitzuteilen, wo das Programm hin soll (Anfangsadresse). Es wird wahrscheinlich dort beginnen (TXTLIM), wenn Sie es ausgelassen haben, aber dieser Fehler sollte nicht vorkommen. Nun sind wir fertig, um es assemblieren zu lassen. Schalten Sie in den DEBUG-Modus und drücken Sie:

**A FILE 1**

Die "1" ist eine Möglichkeit zu sagen, daß Sie ein Assemblerlisting haben wollen. Fast augenblicklich kommt ASZMIC zurück zu Ihnen in den EDIT-Modus (Es können bis zu 300 Befehle pro Sekunde assembliert werden), um Ihnen ein Listing der für jeden Befehl generierten Codes und die jeweilige Speicheradresse, in die assembliert wurde, zu zeigen. Gehen Sie in den DEBUG-Modus, und machen Sie ei-

nen DUMP (D :4300 14), um zu sehen, was wirklich assembliert wurde. Dieser Maschinencode wird "object code" genannt. Löschen Sie jetzt den ganzen Text ohne den ZX81 zu reseten, da wir das Programm noch benötigen.

#### 4.6 J für JUMP

Wir kommen jetzt zur Ausführung des Programmes. Es bringt die Summe der Zahlen von 1 bis 10 ins HL-Register. Um zu sehen, was passiert, benutzen wir eine MACRO-Zeile, die automatisch ausführt, wenn eine BREAK-Bedingung heruntergezählt wurde. (BREAK bedeutet BREAKPOINT, Einzelschritt, RST 0 oder eine NMI-Unterbrechung von außerhalb). Drücken Sie Shift T, 0 (nicht 0) ohne N/L und dann Shift 9, um zurück in den DEBUG-Modus zu gelangen. Suchen Sie in Anhang 3 nach dem Befehl 0, wenn Sie ihn vergessen haben. Drücken Sie:

J :4300

Am Ende des Programmes haben Sie ein DUMP der Register. Schauen Sie auf HL. Es sollte :0037 beinhalten, was dezimal 10 entspricht.

#### 4.7 B für BREAKPOINT

Lassen Sie uns nun versuchen, daß Programm erneut laufen zu lassen, aber dieses mal mit einem eingefügten BREAKPOINT, um die Ausführung zu stoppen. Drücken Sie B LOOPEND N/L (die DJNZ Instruktion) und dann J START N/L. Nun schauen Sie auf die Register. Beide, das DE und das HL, sollten :0001 beinhalten, und B sollte :0A beinhalten, da der Breakpoint vor der Ausführung wirksam wird.

BREAKPOINTS arbeiten, indem Sie das BYTE in der Breakpointadresse save und an die Stelle eines RST 0 (:C7) setzen. Wenn Sie ein Breakpoint bei der normalen Ausführung setzen, wird das "normale" Byte zurückgesetzt (Kontrolle mit D LOOPEND 1).

Drücken Sie jetzt B N/L. Da Sie keine spezielle Adresse dem Breakpoint zugewiesen haben, wird er wieder an die gleiche Adresse zurückgesetzt (Kontrolle mit D LOOPEND 1 erneut). Drücken Sie B Ø N/L, um den Breakpoint zu löschen.

8

4.6 G für GO

Drücken Sie jetzt G N/L. Sie sind einen Befehl vorgerückt. Drücken Sie dies ein paar mal. Dies wird Einzelschritt (single step) genannt. Sie sehen wie sich der Programmzähler (PC) genauso wie die Register, mit denen Sie arbeiten, ändert. Da wir mit dem Befehl G keine Adressangabe machen, benutzt er die im Programmzähler gesavete Adresse innerhalb von REGIM (PC1 und REGIM).

Warum sollten wir jetzt nicht versuchen 20 Befehle auszuführen, bevor zum Monitor zurückgesprungen wird? Drücken Sie G START 2Ø N/L. Der "START" zeigt dem Befehl G, wo begonnen wird, und "2Ø" besagt, daß 20 Befehle vor dem Rücksprung ausgeführt werden. Sie werden sehen, daß der Programmzähler, welcher auf den als nächsten auszuführenden Befehl zeigt, :43Ø8 ist, HL :ØØ15 ist, DE :ØØ6 und B :Ø4 ist. Wenn Sie alles addieren, werden Sie sehen, daß dies der sechste Durchlauf der Schleife ist und in der Tat 20 Befehle ausgeführt wurden. Drücken Sie nun J N/L. Dies zeigt, daß J ohne eine Adresse das Programm bis zum Ende fortsetzt.

Wir erwähnen dies ein wenig unterschwellig ohne vorher darauf eingegangen zu sein. Sie gebrauchen die symbolischen Namen Ihres Programmes in den DEBUG-Befehlen, und sie funktionieren. Dies macht das Schreiben eines Programmes sehr einfach. Nebenbei, wenn Sie versuchen, Einzelschritte durch einen Breakpoint zu realisieren, dann müssen Sie sich damit nach den Breakpointregeln richten, die nicht genau die gleichen sind.

## 4.9 I für IMMEDIATE (sofort)

Wir schließen dieses Kapitel mit dem Blick auf die Möglichkeiten in ASZMIC Befehle sofort auszuführen, die glücklicherweise einheitlich sind, ab. Es ist die gleiche Art von interaktiven Möglichkeiten wie in BASIC, die erlauben, spezielle Assemblerbefehle, die sofort im Programmzusammenhang ausgeführt werden und wie im Programm selbst funktionieren, ohne Recompilierung zu gebrauchen.

Säubern Sie das Textgebiet, wenn Sie nur 1 K RAM besitzen. Drücken Sie I LD HL,1 N/L und betrachten Sie das HL-Register mit "0". Es beinhaltet :0001. Drücken Sie I EX DE,HL N/L, und betrachten Sie, daß das DE- und HL-Register ausgetauscht wird. Jedes Kommando kann mit I benutzt werden, und es können sogar Labels mit EQU definiert werden, obwohl relative Sprünge und andere Direktiven (ORG, DEFB, DEFW, DEFM) bedeutungslos sind und bei der Ausführung zum Zusammenbruch führen können.

Zum Schluß versuchen Sie LD HL,1+2+3+4+5-15 N/L und betrachten das HL-Register. Es sollte 0 sein, was zeigt, wie Sie einfache arithmetische Operationen mit Feldern ausführen können.

Wir weisen Sie darauf hin, sich in Anhang 3 die formale Definition der DEBUG-Kommandos anzuschauen, um zu sehen, was Sie die ganze Zeit gemacht haben. Es gibt noch das DEBUG-Kommando N, welches zur Verbindung bei einer speziellen Platine, auf der BASIC- und ASZMIC-ROM sind, zum Umschalten zwischen den ROM's gebraucht wird. Er ist in der Platinenbeschreibung näher erläutert.



Kap. 5

## Kapitel 5

### Textoperationen

Dieses und die nächsten beiden Kapitel erläutern die Möglichkeiten von ASZMIC, um die Definitionen der Anhänge und der vorherigen Kapitel auszuleuchten. Sie sollten es nicht allumfassende Beschreibung betrachten.

#### 5.0 Einführung

Der zweite Teil des ASZMIC-Speichers, der zwischen DSPGN & (TXT-LIM) liegt, wird zur Textaufbreitung und zum Editieren benutzt. Auf einem 16 K-System bedeutet dies, daß Sie ungefähr 12000 Zeichen benutzen können; mehr sind es noch, wenn Sie den TXTLIM-Zeiger nach unten verbiegen. ASZMIC behandelt dies als einen erlaubten Freiraum, in welchen Sie Zeichen nach Belieben einfügen können, und Ihr Fernsehbildschirm wird zu einem Fenster, das Sie durch die EDITOR-Shift-Tasten über dieses riesige Gebiet bewegen können. Es gibt keinen speziellen Bildschirmfile wie in BASIC, sondern das ganze Textgebiet ist der Bildschirmfile, und die BASIC-Begriffe Programm, Variable und Bildschirmfreiraum sind völlig unwichtig. Dies ist eine der Hauptstärken Ihres ZX81, die es erlaubt, daß der Großteil des Speichers auf dem Bildschirm gelistet werden kann. Wir trafen einige Abmachungen bzgl. dieses Freiraums, um Ihnen das Leben zu erleichtern.

#### 5.1 Files

Die erste Vereinbarung ist die Einführung einer Verwaltung, die den Text in leichter erkennbare Textgebiete unterteilt. Wir nennen diese Sektionen Files. Ein File wird durch ein Symbol, dessen erstes Zeichen das & ist, das auf der ersten Zeile des Files links postiert wird, gekennzeichnet (bzw. abgetrennt).

Dieses Symbol, plus dem vorangestellten `&`, wird zum Namen des Files. Das Ende eines Files wird ebenfalls durch ein `&` als erstes Zeichen einer Zeile gekennzeichnet. Wir bezeichnen die `&`-Zeichen fortan als FILEMARK (Filemarkierungen). Die Vergleichsroutine CMPSTR für Stringzeichen wurde codiert, um einen Stringvergleich, beginnend mit `&`, nur dann anzuerkennen, wenn der Zielstring der erste auf einer Zeile ist. Dies klingt ein bisschen kompliziert, aber es funktioniert in der Praxis wunderbar, weil bei jedem File, wenn es mit einem einheitlichen Filenamen eröffnet wird und es mit einem FILEMARK beendet wird, die ASZMIC-Kommandos unfehlbar das File, das Sie vom Textfeld interessiert, auswerfen und für Sie fortschreiten. Dieses Abkommen ist auch völlig in die DEBUG-Befehle integriert worden.

Das bedeutet, daß Sie so viele Files wie Sie mögen haben können, abgesehen von Einschränkung durch den Freiraum. Wir sollten erwähnen, daß DEBUG-Kommandos oft arbeiten, obwohl ein falscher Filename benutzt wurde (z.B. einen ohne `&`), aber darauf ist kein Verlaß. Das Vergessen des abschließenden Filemarks ist ein totales Disaster bei Kassettenoperationen und beim Drucken und ein totales Fiasko beim Assemblieren oder Mischen des endlosen Files.

## 5.2 Mischen & Löschen

Wir implementierten diese Funktionen als EDIT.Shift-Operationen, um eine größere Flexibilität zu erreichen und nach der Philosophie "Eher zeigen als erzählen" einen vollen Bildschirmeditor zu erhalten. Der Cursor wird zur Kennzeichnung der Startposition der Misch- und Löschoperationen benutzt; und eine Filemarkierung (File mark) wird zur Beendigung benutzt. Wenn eine Misch Taste gedrückt wurde, durchsucht ASZMIC von einer am wenigsten hinteren Shift-Macro-Zeile runter bis ein wenig über das aktuelle Ende des Datenzeigers hinaus, um das Startzeichen zum Mischen (`>` für Shift-G `<` für Shift-F `*` für Shift-D) und kopiert dann den gesuchten Text hin zur Cursorposition, bis ein Filemark im Text gefunden wird. Langes Mischen kann den Bildschirm für einen Moment ausschalten.

Auf der einen Seite ist dies ein konventionelles Mischen eines Files, auf der anderen wird es ein Text-MACRO. Sie können zum Beispiel in ein File von Unterroutinen von einer Kassette laden und sie in ein Programm mischen. Sie können genauso, wenn Sie ein Programm mit vielen Daten schreiben, etwas wie `> DEFB f` in das Textfeld codieren, und Sie können mit jedem Druck von Shift-G den String DEFB in Ihr Programm einfügen. Dies kann eine Menge/Tipperei sparen.

Wenn Sie den Ursprungsfile/string, welchen Sie gerade gemischt haben, löschen wollen, müssen Sie den Cursor genau auf dem Mischzeichen plazieren, Shift-2 drücken und schon wird gelöscht. Eine Filemarkierung (FILEMARK) stoppt immer eine Lösch- oder Mischoperation. Wenn vergessen wird, den Ursprungstext mit einem `f` (FILEMARK) zu beenden, wird Ihr Computer verzweifelt bis ins Unendliche mischen. Dies führt natürlich zum Systemabsturz. Beim Löschen wird bei einm fehlenden Filemark am Ende die Arbeit direkt abgeblockt.

### 5.3 Editieren

Sie haben die EDIT-Funktionen schon in Kapitel 2 kennengelernt, so daß wir nur noch einige Tips geben. Erinnern Sie sich an den Gebrauch des E-String-Befehls als Shift MACRO für das Suchen nach allen Vorkommen eines Strings in einem File. Drücken Sie N/L in der Mitte einer vorher beschriebenen Zeile und es entstehen 2 Zeilen; das Drücken von Rubout kann zum Verknüpfen von Zeilen durch das Löschen eines N/L benutzt werden. Versuchen Sie bitte nicht in den obersten 20 Zeilen des Textgebietes zu editieren außer mit einem Shift MACRO. Diese Zeilen dienen als Puffer, um einen sauberen Bildschirm zu erhalten, und das Löschen würde bedeuten, daß Sie einenzusammengebrochenen Bildschirm erzeugen. Wahrscheinlich führt ein Gebrauch der Taste Rubout, um das N/L ganz am Anfang des Bildschirms zu löschen, zu einer Verschiebung des Anfangs des unsichtbaren Datenzeigers, was katastrophale Folgen hat. Editieren außerhalb des Leerraums, den ASZMIC vor jedem N/L sicher stellt, kann unter Umständen zum Verlust des Cursors führen und Schönheitsfehler beim Assemblieren und Mischen verursachen. Das Schreiben von mehr als einigen Zeilen mit mehr als 36 Zeichen darin auf einer Bildschirmseite kann auch zum unsichtbarwerden des Cursors führen, obwohl nichts passiert.



Wenn Sie in einen Text Zeilen einfügen wollen, sollten Sie den Cursor auf die Zeile darüber, über dem gewünschten Punkt zum Einfügen, stellen und Shift-W drücken, um an das Ende der Zeile zu gelangen und N/L zu drücken, um eine neue Leerzeile zu schaffen. Shift-7 gefolgt von Shift-6 bringt Sie zurück zum Anfang der Zeile. Erinnern Sie sich, daß Shift-Q nach unten und vorwärts löscht, während Shift-O nach hinten löscht.

#### 5.4 Drucken

Damit der Drucker angesprochen wird müssen 2 BITS in ASSFLG gesetzt sein. Dieses Flag wird automatisch vom Assembler gesetzt und wird bei Beendigung des Assemblieren (außer bei Gebrauch von BREAK) auf Null gesetzt, so daß Sie zur Druckersteuerung das Flag selbst setzen müssen. BIT 1 gesetzt auf 1 schickt fast alles von ASZNIC, entgegen Ihres Wunsches, auf den Drucker und BIT 5 veranlaßt den Drucker 64 Zeichen pro Zeile zu schreiben.

Sie können den Drucker dazu bringen, die doppelte Anzahl an Zeichen zu schreiben als gewollt. Wenn Sie eine Zeile mit bis zu 32 Zeichen schreiben und mit Leerzeichen ausfüllen (kein N/L) bis 48 Zeichen und dann die 32 Zeichen auf genau die gleiche Art noch einmal eingeben, erhalten Sie nicht eine lange Zeile mit 2 Botschaften sondern eine Nachricht mit der doppelten Höhe der Zeichen beim Druck. Dies kann für Titel gut sein und zum Gebrauch der Mischmöglichkeiten für doppelt hohe Texte. Wenn Sie die Anzahl der Leerzeichen, falsch eingeben, so ist das Resultat interessant aber kaum leserlich. Mit dieser Technik können 64 Zeichen in einer Nachricht und 32 füllende Leerzeichen gedruckt werden.

Erinnern Sie sich an den Gebrauch der BREAK-Taste, um das Drucken abzubrechen, wenn Sie das abschließende Filemark vergessen haben sollten. Zum Schluß findet das Drucken indirekt statt, indem ASZMIC über eine Adresse im RAM zur Druckroutine springt. Wenn Sie en Inhalt von PRTJMP ändern, do können Sie Ihre eigene Druckroutine anstatt der des eingebauten Sinclair-Interface anspringen. Der Druck wird begonnen wobei auf dem Stapel noch der Stsrts der Zeile, die gedruckt werden soll, steht, und Ihre Routine muß dies löschen und wieder zurückkehren mit einem RET, wobei HL auf das erste Zeichen nach N/L, daß die Zeile beendet, zeigt.

## 5.5 Kassettenoperationen

Ihr Kassettenrekorder benötigt die gleiche Lautstärke und Einstellung wie in BASIC. ASZMIC arbeitet in dieser Hinsicht zuverlässiger, so daß Sie auch billigere Kassetten verwenden können. Wenn Sie eine neue Kassettensorte verwenden wollen, versuchen Sie eine Anzahl von Leerzeilen (K&S "NULL" :7000 :7FFF) aufzunehmen; besonders nach dem Einschalten, wenn der Speicher leer ist, geht es sehr gut. Versuchen Sie, es wieder zu laden, und drehen Sie langsam die Lautstärke ganz zurück, und betrachten Sie den Effekt, wenn Sie langsam lauter stellen. Der Bildschirm wechselt von "weiß und verschwommen" zu einer Anzahl von wohlgeordneten Streifen bis zur völligen Dunkelheit bei zu hoher Lautstärke. Die Lautstärke sollten Sie ziemlich in der Mitte einregulieren.

### 5.5.1 Kopf (HEADERS) & Files

Da es 2 Sorten von Files gibt, können wir auch Sie mit der Kasette manipulieren; den reinen Textfiles und den definierten Speicherregionen müssen wir jeweils eine getrennte Identifikation geben, da Speicherregionen keine Filenamen besitzen. Wenn Sie einen Savebefehl ausführen, gibt ASZMIC das Kommando selbst als einen Kopf aus und nach einer kurzen Pause folgt das File.

Bei der Wiedergabe stellt der Savebefehl eine Sequenz von Zeichen dar, welche ASZMIC erkennt und auf den Bildschirm für ein paar Sekunden darstellt. Danach kontrolliert ASZMIC den Rest des HEADERS, um herauszufinden, ob es sich um zu ladenden Text oder Daten handelt, und dem entsprechend wird reagiert. Nach dem Laden kehren Sie stets in den EDIT-Modus zurück. Es gibt noch einige Besonderheiten beim Laden von Speicherregionen. Die erste ist, daß auf die Region, in die das Programm (Daten) geladen wird, durch den Inhalt der OFFSET-Variable gezeigt, so daß Sie in andere Regionen vom Rekorder laden können. Als zweites können Sie den Inhalt von LABSTK & LABEND studieren und eine Symboltabelle auf Kassette schreiben, und wenn Sie der "Ende der Region"-Definition mit einem Leerzeichen folgen und mit L, erkennt ASZMIC das Laden als Symboltabelle an und setzt folglich LABEND. LABST ist unverändert geblieben, so daß Sie es wie vor dem Saven vorfinden. Dies kann manchmal für Wörterbücher und DEBUG-Zwecke nützlich sein.

Die Trennung von HEADERS und dem File auf Kassette gibt Ihnen die Möglichkeit, besser zu hantieren und zu überspielen. Die Details dazu sollten Sie sich selbst beibringen. Beachten Sie, daß wenn RDCASS vor dem Laufen des Bandwerkes bis zur Stille von 1/2 Sekunde vor einem File oder HEADER aufgerufen wird, es irgendwelchen Mist aufnehmen kann (abhängig von der Lautstärke und dem Bildschirminhalt zur Aufnahmezeit). Die Ruhepause jedoch dient zur Synchronisation.

Da ASZMIC alles nach K&S bis zum Ende der Datenmarkierung als HEADER behandelt, sollten die Savebefehle die Schlußbefehle im Textgebiet, wenn N/L gedrückt wurde, sein.

### 5.5.2 Das Schreiben von BASIC-Programmen

Es gibt noch eine weitere Möglichkeit mit dem Kassettenrekorder, die wir erwähnen wollen. Für diejenigen, die ~~keine~~ Platine besitzen, die sowohl ASZMIC- und BASIC-ROM beinhaltet, ist dies eine Möglichkeit zwischen beiden zu kommunizieren. Im Anhang befindet sich ein Listing eines Programmes, das den Inhalt eines Basicprogrammes mit einem einzelnen REM darin simuliert. Wenn Sie in die angezeigte Stelle ihren eigenen Code plazieren, so sorgen Sie dafür, daß es in der REM-Zeile erscheint. Dieses Programm ist dann auch in BASIC zu laden, und Sie können es mit USR 16514 starten. Sie sollten nicht auf das HL-Register in Ihrem Programm zugreifen. Sie benötigen zum Assemblieren die OFFSET-Möglichkeit. Auf einem 16 K-System `ORG :4000` und `OFFSET :3000` arbeitet es sehr gut, aber Sie müssen einen 9 Byte Kopf haben, da die ersten 9 Bytes der Variablen in BASIC nicht aufgenommen werden.

16384  
=4096H

Wenn Sie es andersrum machen (Basicpgm.'s nach ASZMIC), müssen Sie eine genügend große Speicherregion schaffen, um das BASIC-Programm festzuhalten. Spulen Sie zurück bis Sie sich kurz vor dem Beginn der aufzunehmenden Daten befinden. Wenn jetzt die BASIC-Aufnahme abgespielt wird, so wird diese in ASZMIC zurückgeladen. Sie sollten das Laden per Hand abbrechen, wenn das BASIC-Programm vollständig ist. Andernfalls können Sie ein kleines Programm, das RDCASS zum Holen der Bytes vom Rekorder und zum Speicher benutzt, schreiben. Dies geht wahrscheinlich schneller und zuverlässiger.

Am Anfang hatte ASZMIC ein super schnelles Kassetteninterface mit Kontrolle und einer Menge von Vorteilen, einen ASCII-Zeichensatz und ein RS232-Druckerinterface. Wir änderten dies alles zu Gunsten der Sinclair-Kompatibilität für Zeichen, Drucker und Kasette um, so daß wir hoffen, daß einige davon Gebrauch machen und es keine Verschwendung war.

## Kapitel 6

### Der Assembler

Bevor wir über die Assemblereinteilung von ASZMIC sprechen, sollten wir überlegen, was ein Assembler überhaupt macht. Anders als im BASIC sind die Assemblerbefehle ausnahmslos auf die derzeitigen Befehle der Z80-CPU bezogen. Die Z80-CPU hat einen Satz von logischen, arithmetischen und Befehlen zum Datentransfer, welche von ihr aufgerufen werden können ... es sind Operationen, die relativ leicht zu verstehen sind, obwohl es einige Zeit und Praxis erfordert "nützliche" Arbeit zu leisten. Unglücklicherweise sind die Anweisungen, die diese Operationen beim Lesen vom Speicher steuern, bedeutungsvoller für die digitale decodierende Logik in der Z80-CPU.

#### 6.1 Mnemoniks

Die erste Sache, die ein Assembler machen kann, ist, die gewohnte Z80-Schreibweise in für Ihren Rechner verständlichen Maschinencode umzuwandeln; zum Beispiel CCF als ein Ausdruck für Complement (Umkehr) Carry Flag ist leicht zu behalten als 3F. Der Assembler ermöglicht uns somit Z80-Befehle in Form von begreiflichen Bildern der Z80-CPU zu geben.

#### 6.2 Symbole

Die andere Sache, die ein Assembler ausführen kann, ist es uns zu erlauben, Symbole statt binärer Codes, die die Z80-CPU zur Daten- und Adressdarstellung benötigt, zu benutzen. Daten können dezimal oder im Buchstabenformat gebraucht werden. Die Z80-CPU arbeitet mit definierten Adressen im Speicher. Sehr oft wissen wir nicht, wie wir ein Programm codieren sollen und wo gewisse Sprungadressen und Unteroutine liegen. Wir können symbolische Namen, als LABELS bezeichnet, geben und erlegen dem Assembler die Bürde auf, diesen jeweils die Adressen zuzuordnen.

## 6.2 Symbole

Die andere Sache, die der Assembler für uns tun kann, ist, uns zu erlauben, statt der binären Form von Daten und Befehlen Symbole zu benutzen. Daten können dezimal oder im Buchstabenformat, das wir ebenfalls benutzen, eingegeben werden. Die Z80-CPU arbeitet mit definierten Speicheradressen. Sehr oft weiß man bei seinen Programmen nicht mehr, wo die Einsprungadressen der wichtigsten Routinen liegen. Aus diesem Grunde ist es möglich sie mit symbolischen Namen zu bezeichnen, die man übrigens als LABELS bezeichnet, und mit denen der Assembler sich seine Adressen selber sucht.

Einige der Assemblersymbole wie numerische Konstanten sind selbstdefiniert, und der Rest sind LABELS, die entweder ausdrücklich in einer Anweisung an den Assembler definiert werden können oder als das erste Feld eines Statements erscheinen können, wobei sie in diesem Fall die Adresse, ab der der Code, der von diesem Statement (Befehl) generiert wurde, sich im Speicher befindet, bestimmen. Diese Adresse wird die Position von Code genannt, der Assembler nimmt seine Informationen darüber vom Positionszähler. Dort befindet sich ein spezielles Symbol, das \$-Zeichen, welches der Assembler jedesmal interpretiert (umwandelt) und es aufführt als Wert für die Zählerposition. Die folgenden Befehle haben beim Assemblieren den gleichen Effekt:

```
LABLAB JP LAPLAP
      JP $
```

## 6.3 Wie macht er das?

Wenn der Assembler Ihr Programm das erste mal furcht, und er ein Label, das als Argument in einem Befehl benutzt wird, entdeckt, dann kann er, wenn das Label vorher definiert wurde, die Adresse oder den mit ihr verbundenen Wert dem Befehl hinzufügen, aber wenn das Label so etwas wie eine Sprunganweisung weiter ins Programm hinein ist, so hat der arme alte Assembler keinen Wert.

Die Lösung dieses Problems ist, daß der Assembler immer zweimal assembliert. Beim ersten Durchgang wird der Code generiert, aber in Wirklichkeit wird lediglich eine Tabelle (Symboltabelle) generiert, welche am Schluß einen fest definierten Wert für jedes Label enthält. Beim zweiten Durchlauf dient die Symboltabelle zur Unterstützung für gültige Symbolwerte und der auszuführende Code wird generiert, Listings ausgeworfen und Fehlermeldungen. Da wir über Symboltabellen sprechen, sind hier einige Definitionen. Ein Label, das als Argument vor seiner Definition auftaucht, nennt man Vorwärtsbeziehung, ein Label, das mehr als einmal definiert wurde, Dublikatdefinition und ein Label, auf das man sich bezieht, das aber nie definiert wurde innerhalb des Programms, ungenügende Definition, es sei denn man habe irgend einen Weg gefunden, es innerhalb der Symboltabelle (z.B. Direktbefehl) zu definieren, was externe Beziehung heißt.

#### 6.4 Mögliche Zusätze

Wenn Sie den Befehl A benutzen, um dem Befehlsinterpreter zu sagen, daß er den Assembler aufrufen soll, benennen Sie ein File, das assembliert werden soll und auch ein OPTIONS-Feld (Zusatzfeld), das dem Assembler erzählt, wie er arbeiten soll. Einige der Auswahlmöglichkeiten sind sehr einleuchtend. Sie können auswählen, daß ein Assemblerlisting generiert wird ... sehr wichtig zum Aus-testen Ihres Programmes. Sie können auswählen, ob Sie das Listing und die Fehlermeldungen auf dem Drucker ausgegeben haben wollen, sogar mit 64 Zeichen pro Zeile, wenn Sie mögen. Sie können das Generieren des Objectcode unterdrücken ... sehr nützlich bei den ersten Schritten zur Programmkontrolle und auch für andere Fälle.

Dann haben Sie noch einige weniger einleuchtende Auswahlmöglichkeiten. Sie befassen sich mit der Verbindung zwischen getrennten Programmen zu einem funktionierendem Ganzen. Normalerweise, wenn Sie anfangen zu assemblieren, wird der Inhalt der Symboltabelle auf Null gesetzt, aber, wenn Sie wollen, können Sie sie beibehalten. Warum?

Das ist möglich, um es zu ermöglichen, daß Ihr Programm extern-definierte Ansteuerungen gebrauchen kann. Ebenfalls können Sie auswählen, daß der Assembler sofort zum 2. Durchgang schreitet, welcher Ihnen schon bekannt sein dürfte. Warum das Ganze? Damit ist es möglich, viele verschiedene Programme zu einem funktionierendem Ganzen zusammenzufassen.

Wenn Sie in Anhang 3 unter dem Befehl A nachgucken, finden Sie eine Zusammenfassung aller möglichen OPTIONEN und die Zahlen, die sie darstellen. Das OPTION-Feld ist eine Anzahl von Zahlen für die Möglichkeiten, die Sie wollen. Dann, wenn ein Listing auf dem Drucker mit normalem Anstieg aber ohne Objectcode Ihr Wunsch ist, wird das Optionfeld 64+2+1 oder einfach 67, wenn Sie gut im Kopf rechnen können und keine Geduld haben.

## 6.5 OFFSETS

Es gibt eine Variable, die OFFSET genannt wurde, deren Adresse Sie in Anhang 4 finden und die eine besondere Bedeutung für das Assemblieren und das Laden von der Kassette hat. Der Wert dieser Variable ist normalerweise Null, und sie wird gebraucht als Verschiebe-Differenz vom Assembler.

Die Arbeitsweise ist folgende:- Normalerweise, wenn Sie ein Programm codieren, erwarten Sie, daß es an der Position, die Sie mit dem Befehl ORG festgelegt haben, assembliert wird, so daß es an dieser Position ausgeführt werden kann. Wir sagen, daß in diesem Fall der EXECUTION LOCATION COUNTER und der LOAD LOCATION COUNTER synchron sind. Aber was passiert, wenn Sie ein Programm schreiben, das nach der Assemblierung nicht ausgeführt werden soll, von dem verlangt wird, mit der Startadresse Null zu beginnen, wenn es stattdessen auf ein EPROM geschrieben werden soll. Sie müssen ein ORG 0 an den Anfang des Programmes setzen, so daß das Programm intern übereinstimmt, aber, wenn Sie es zurücklassen würden, würde der Assembler versuchen, den Objectcode an den Anfang des ASZMIC ROMs zu setzen. Dies macht zwar nichts, aber der Objectcode wäre verloren. Sie benötigen etwas, um ASZMIC klarzumachen, daß es den Objectcode irgendwo anders plazieren soll.



Dazu dient die OFFSET-Variable. Wenn ASZMIC den Objectcode auswirft, addiert es den Inhalt von OFFSET zur Ausführungsadresse, um eine Ladeadresse zu erzeugen, bei der jedes Objectbyte gelegen sein soll. Wenn Sie vor einer Assemblierung OFFSET auf :7000 setzen, dann wird der Code, von dem man erwartet, er werde an der Adresse 0 ausgeführt, in Wirklichkeit in den Speicher ab der Adresse :7000 geladen.

Erinnern Sie sich immer daran, OFFSET danach wieder auf 0 zu setzen. ASZMIC macht dies nicht automatisch für Sie, und Sie würden sonst daüber erstaunt sein, wohin Ihr nächstes assembliertes Programm verschwunden ist.

## 6.6 Das Assemblieren

Rufen Sie den Assembler mit A gefolgt vom Filenamen, den Sie vorgesehen haben, auf und legen Sie, es sei denn Sie wollen die Fehler im Objectcode übernehmen, kein Listing, keine Fehlermeldung auf dem Bildschirm und eine neue Symboltabelle haben, ein OPTION-Feld zur Kontrolle des Assemblierens an.

Sie müssen offensichtlich erst das zu assemblierende Programm mit Hilfe des Editors geschrieben haben, oder ein vorher geschriebenes File von Kassette laden. Versichern Sie sich, daß das File, das das Programm enthält, ASZMIC gerecht wird z.B. daß mit einem Filemark als erstem Zeichen der Zeile, die nach der letzten Programmzeile folgt, beendet wird. Außerdem sollen die Befehle innerhalb der einzelnen Programmzeilen den Z80-Regeln entsprechen (die einzige Ausnahme sind die EQU-Direktiven und das Format: 10000 anstatt 1000H für hexadezimale Zahlen). Diese Regeln sind, kurz gefaßt, daß das erste Zeichen eines Befehls ein Leerraum sein soll, es sei denn Sie wollen an dieser Stelle ein Label definieren, wobei jedes Label mit wenigstens einem Leerraum abgeschlossen werden muß. Darauf folgt der OP-Code und, wenn es der Befehl benötigt, kommen danach ein bis zwei Argumente. Wenn es zwei Argumente sind, müssen sie durch ein Komma ohne vor- oder nachgestelltes Leerzeichen getrennt werden.

Das Schlußfeld (Argument, OP-Code und Direktive), das Sie schreiben, sollte mit einem N/L, einem Leerzeichen oder einem ; (Semikolon) beendet sein. Wenn ein Semikolon gefunden wird, werden alle nachfolgenden Zeichen auf dieser Zeile als Bemerkung behandelt.

Wenn das erste Zeichen eines Befehls ein Semikolon ist, wird der gesamte Befehl als Bemerkung behandelt. Wenn ein Label, das in Spalte 1 des Befehls beginnt, mit einem = (Gleich-) Zeichen beendet wird, dann rechnet der Assembler damit, daß Sie eine EQU-Direktive verwenden, um das Label zu definieren und sucht nach einem Argument hinter dem = Zeichen (es können Leerzeichen dazwischen liegen, wenn Sie wollen), welches ausgewertet und dem Label zugewiesen wird.

Beachten Sie bitte, daß der Assembler beim Gebrauch eines RST  $\emptyset$ -Befehls an Stelle eines RET zurückspringt, so daß Sie das C-Kommando benutzen können, um das Ende der Region des Objectcodes in den IX- und IY-Registern sehen können.

#### 6.6.1 Direktiven

Der ASZMIC-Assembler hat zwei echte Direktiven, ORG und =, welche keinen auszuführenden Code generieren und drei Pseudodirektiven, DEFD, DEFW und DEFM, welche nicht ausführbaren Code generieren, aber einen Objectcode produzieren, welcher eine Darstellung von nach der Direktive ausgedrückten Werten ist. Wir berühren das = im letzten Paragraphen; deshalb sehen wir uns jetzt ORG an. ORG verursacht, daß das folgende Argument ausgewertet und zum Zähler für die Ausführungsadresse übertragen wird (zum IX-Register). Sie können es beim Programmstart dazu benutzen, zu definieren, wo das Programm die Ausführung (und wo sich der Objectcode befindet, wenn OFFSET auf  $\emptyset$  steht) erwartet und darauf wartet, daß in der Mitte des Programmes Leerzeichen erzeugt werden. Ein listiger Gebrauch von ORG in der Mitte des Programms ist ein Befehl folgender Eingabe: ORG  $\$+20$

was 20 Bytes in der Mitte des Programms für einen Buffer oder was auch immer reserviert. ZILOG benutzt eine DEFS-Direktive zu diesem Zweck, welche wir nicht in ASZMIC implementierten, weil es überflüssig war. Sie können ORG- oder = Direktiven nicht vor deren Definition benutzen. (Es gibt auch Wege dies zu umgehen.)

DEFB generiert ganz einfach ein Byte im Objectcodes mit dem Wert, Modulus 256, des zugehörigen Argumentes. DEFW generiert 2 Bytes, aber mit dem letzten als höherwertigem Byte, so wie es der Z80 erwartet.

DEFW :1234

produziert etwas, das im Speicher wie folgt aussieht :3412

DEFM sucht nach einem " (Anführungs-) Zeichen und wandelt dann jedes Zeichen danach in ein Byte im Speicher entsprechend den Sinclairwerten für jedes Zeichen um bis einem DEFM-Argument verwenden verwenden können. Versuchen Sie nicht zwei hintereinander zu codieren. Dies funktioniert nicht.

#### 6.6.2 OP-CODES und Argumente

OP-Codes sind die mnemonischen Darstellungen der Z80 Befehle. Sie können darüber in einem Z80-Assemblerhandbuch Ihrer Wahl lesen und sich daran erinnern, was sie darstellen und was sie machen, wenn Sie Anhang 5 durchgehen. Argumente sind in der Standard-ASZMIC-Form, die Sie inzwischen schon kennen sollten, anzugeben. Sie sind im "FIELDS"-Paragraph von Anhang 1 für Sie definiert.

Das einzige Argument, das Ihnen entgegentritt und das Sie gewöhnlich im gesamten ASZMIC nicht gebrauchen, ist das \$ (Dollar-) Zeichen, das den Wert des Zählers der Ausführungsadresse am Anfang des Befehls, in dem \$ erscheint, repräsentiert. Dies sollte Ihnen jedoch aus dem Buch über die Z80-Assembler-Programmierung bekannt sein.

### 6.6.3 Kommentare

Von Kommentaren wird immer wieder gesagt, daß sie eine sehr gute Sache seien, und sie sind besonders wünschenswert in Assemblerprogrammen, weil der Code versteckte Detailfunktionen besitzt (So kann es selbst erfahrenen und professionellen Programmierern passieren, daß sie zwölf Zeilen geschrieben haben und nicht mehr wissen, was die ersten sechs machen). Wir folgten dem Beispiel von ZILOG und gestatten es, einem Befehl mit einem Semikolon zu beenden und den Rest der Zeile mit Kommentaren zu füllen. Unglücklicherweise kann der Bildschirm nur 36 Zeichen pro Zeile haben (32 pro Zeile beim Drucker), so daß Kommentare die ans Ende angehängt werden kaum beachtet werden. Sie sind noch gebräuchlich im SOURCE CODE (Verzeihung, der SOURCE CODE ist der Name für die Programme, die Sie in den Assembler eingeben). OPTION Bit 5 macht den Verstümmelungseffekt rückgängig, und stellt auch den Drucker so um, daß Sie 64 Zeichen pro Zeile drucken können. Der ZX81 hat nicht genug Speicherplatz, um damit spielen zu können und hat ein teuflisch langsames Kassetteninterface, so daß die Hardware für große Programme ein wenig Unterstützung für die IN-line-Dokumentation anbietet. Es empfiehlt sich sogar unter Umständen, daß Sie versuchen jeden Teil des Programms zu beschreiben und die Arbeitsweise zu schildern. Sie werden bald vergessen, daß Sie glaubten, es ginge anders besser.

### 6.7 Fehler

In einem kleinem 4 K ROM gibt es eine Grenze der Fehleranalyse, welche der Assembler ausführt. Die Autoren hatten einige Probleme damit, zu entscheiden, nach welchen Fehlertypen gesucht werden soll. Sie kamen zu der Überzeugung, daß EUTHANASIA die beste Fehlerkontrolle ist. Spaß bei Seite, die Fehler, die Sie machen, fallen in drei Kategorien.

Sie verwechseln Labelnamen durch falsches Aufschreiben, codieren die selbe Unterroutine zweimal oder vergessen alles mit einzubeziehen; Sie fügen so viel zwischen einem relativen Sprung und seinem Ziel ein, daß der Sprung außerhalb des zulässigen Bereiches geht; und Sie machen Fehler mit dem OP-Code, indem Sie normalerweise in Spalte 1 anfangen zu codieren. Dies sind die drei Fehlerarten, die ASZMIC für Sie erkennt.

Fehler werden Ihnen angezeigt durch das Setzen einer NON-Blank Spalte 1. Wenn Sie nicht ein Assemblerlisting generieren, wird die fehlerhafte Zeile auf jeden Fall ausgegeben.

Das Aussehen des Zeichens in der ersten Spalte könnte Ihnen eine Menge über die Fehler erzählen, aber die Regeln sind so kompliziert, daß es einfacher ist, es festzustellen statt darüber großartig nachzudenken. Der erste Blick ist es zu sehen, ob irgendein Objectcode auf der Fehlerzeile erscheint. Wenn dies nicht der Fall ist, so handelt es sich um einen Objectcode-Fehler. Mögliche Ursachen sind das Codieren des OP-Codes ab Spalte 1, das Beenden mit einem Komma statt eines Leerzeichens oder die Falschschreibung (Wegen der ungewöhnlichen Ausführung, kann der Assembler nur sieben Achtel der möglichen Falschschreibungen entdecken). Schauen Sie danach, ob Sie in dem Befehl einen relativen Sprung benutzt haben. Wenn dem so ist, ist der Fehler bestimmt dort zu suchen. Am Ende liegt der Fehler bei Doppeldefinitionen oder bei den Labels, so daß es ratsam ist, auf die Labelargumente zu achten und herauszufinden, ob Sie definiert sind oder sogar mehrmals definiert wurden.

An dieser Stelle möchten wir hinzufügen, wenn Sie eine Zahl mit einem ungültigen Zeichen für die Zahlenbasis definieren, zeigt ASZMIC einen Fehler an.

## 6.8 Reihenfolge

Wir empfehlen folgende Reihenfolge bei einem Assemblerprogramm:

- 1) Editieren des Programmes
- 2) Saven auf Kasette
- 3) Es mit OPTION 64 zu assemblieren
- 4) Wenn Fehler vorhanden sind, sie zu identifizieren und nach 1) gehen
- 5) Es mit OPTION 65 zu assemblieren (67, wenn Sie einen Drucker besitzen)
- 6) Das Listing kontrollieren, um zu sehen, ob das Programm auch wirklich das ist, was Sie wollen. Wenn nicht, gehen Sie bitte nach 1).
- 7) Assemblieren ohne OPTIONS-Feld
- 8) DEBUG so benutzen wie es in Kapitel 7 beschrieben wird. Wenn Fehler auftreten, gehen Sie nach 1).
- 9) Saven Sie den Objectcode und möglicherweise die Symboltabelle auf Kasette (Erinnern Sie sich selbst an das L-Feld).

## 6.9 Listings

Die Hauptsache bei der Assemblerprogrammierung ist ein gutes Assemblerlisting. Es ist die fundamentale Hilfe, um Ihre Programme zum richtigen Funktionieren zu bringen. Wenn Sie keinen Drucker besitzen, ist das Leben für Sie viel härter. ASZMIC versucht Ihnen zu helfen, indem es erlaubt, Programmabels in DEBUG-Befehlen zu benutzen, Listings auf dem Bildschirm liefert, in die Sie anschließend save und speichern können auf Kasette, und der ASZMIC-Editor ist gut genug, um den Gebrauch von Bildschirm-listings als Notizblatt zu gestatten, so daß eine Weiterentwicklung doch möglich ist. Aber dies alles kann kein wahres Listing ersetzen.

Das Assemblerlisting besteht aus vier getrennten Feldern. Das erste ist das SINGLEBYTE-Error-Flag, welches in Spalte 1 gesetzt wird.

Das zweite ist die vierstellige Hexadezimaladresse, an der der Code von jedem Befehl liegt (das ist der Wert des Zählers der Ausführungsadresse am Anfang eines jeden Befehls); das dritte ist die hexadezimale Darstellung des ein bis vier Byte langen Codes von jedem Befehl (beachte, daß DEFM bis zu 5 Bytes vor einer Verstümmelung generieren kann); und das vierte ist so viel vom ursprünglichen Befehl wie eine 32-Zeichen-Zeile hineinpaßt.

Das Listing oder die Fehlermeldungen werden zum Drucker gegeben, wenn das Listing nicht von dem OPTIONS-Feld gewählt wurde. Es wird stattdessen zum Bildschirm geschickt, wenn BIT 1 des OPTIONS-Byte gesetzt ist.

## 6.10 Wörterbücher

Wenn Sie gerade mit der Assemblerprogrammierung begonnen haben, können Sie diese und die folgende Sektion auslassen. Sie beschreiben ziemlich exotische Funktionen, die Sie wahrscheinlich vorläufig nicht gebrauchen werden. Ein Wörterbuch ist ein Stück des Objectcodes, der häufig benutzte Routinen enthält, die für viele verschiedene und unabhängige Programme zu gebrauchen sind. Wenn Sie ein Programm schreiben, was eine solche Routine braucht, können Sie immer in den SOURCE-Code von Kassette laden (oder, wenn vorhanden, von Microdrive), es in Ihrem Programm zusammenmischen und dann das Ganze assemblieren, aber nach einiger Zeit finden Sie dies bestimmt unnötig. Wie viele Vergnügen macht es, Routinen (Dinge wie Eingabe und Ausgaberroutinen) Ihrer eigenen Sammlung immer wieder zu benutzen. Solch eine Sammlung von Routinen wird Wörterbuch genannt und charakterisiert sind sie durch Ihre Unabhängigkeit von externen Programmen. Ein Wörterbuch enthält Definitionen, aber keine externen Referenzen.

Wie würden Sie einen solchen Satz von Routinen mit ASZMIC benutzen? Zu Beginn würden Sie die Wörterbuchteile assemblieren und beides, Objectcode und Symboltabelle, save. Wenn Sie zum Assemblieren eines Programmes, was einige dieser Routinen benötigt, würden Sie zuerst in die Symboltabelle wie früher beschrieben laden und dann zu einer Assemblierung mit BIT 2 ("4") des OPTION-Feldsatzes springen. Dies bedeutet, daß Ihr Programm mit einer Symboltabelle, die die Adressen aller Namen innerhalb eines Wörterbuchprogrammes definiert, startet und Richtlinien; die ihr gemacht werden, nicht als Fehler ausgewiesen werden. Natürlich, wenn Sie zur Ausführung des Programmes kommen, sollten Sie sicher sein, daß Sie das Wörterbuch geladen haben. Andernfalls kommt es zu einem Disaster.

### 6.11 Kreuzansteuerungen

Die nächste Stufe in Ihrer Entwicklung als zukünftiger Assembler-programmierer ist es, so große Programme zu schreiben, daß Sie nicht genug Speicherplatz haben, um sie in der Gesamtheit zu assemblieren oder mit Freunden oder Kollegen zusammenzuarbeiten, um getrennte Teile des gleichen Programmes zu schreiben. In diesem Fall stehen Sie einer Situation gegenüber, die ganz anders als die mit den Wörterbüchern ist, weil jedes getrennt codierte und assemblierte Teil nicht nur Definitionen beinhaltet, die die anderen Teile gebrauchen können, sondern Verbindungen zu den anderen Teilen. Wie ist eine solche Situation zu behandeln? Es ist wahrscheinlich das Einfachste, ein Beispiel mit drei getrennt codierten aber von einander abhängigen Programmen anzuführen; PROGA, PROGB und PROGC. PROGA wird codiert und als seinen letzten Befehl hat es ENDA=4. PROGB beginnt mit einem ORG ENDA-Befehl.

Wenn Sie dann PROGA mit einem OPTION-Byte, das BIT 6 (kein Objectcode) einschließt, assemblieren und dann beides, PROGB und PROGC, mit einem OPTIONS-Byte assemblieren, das ein Setzen der beiden Bits, 6 und 2 (siehe Anhang 3), einschließt, dann haben Sie eine Symboltabelle, die Labeldefinitionen aller drei Programme, aufgebaut.



Assemblieren Sie dann alle drei Programme erneut, aber dieses mal gebrauchen Sie ein OPTIONS-Feld, welches sowohl BIT 7 als auch BIT 2 einschließt, und Sie erhalten als Ergebnis ein einziges Stück Objectcode, welches intern selbst zusammengesetzt ist, immer vorausgesetzt Sie haben keinen Fehler gemacht.

Was Sie gemacht haben war, daß Sie den Objectcode das erste mal als Sie die drei Programme in Folge assemblierten unterdrückt haben, so daß Sie am Ende eine Symboltabelle zurücklassen, die alle Symboldefinitionen aller drei Programme plus der zugehörigen Fehlermeldungen beinhalten, die, vorausgesetzt Sie verweisen nur auf die unbefriedigenden Kreuzbeziehungen, völlig irrelevant sind. Da PROG8 und PROG9 assembliert wurden, ausgewählt aus der SYMBOL Tabelle PRESERVATION OPTION (Erhaltung), wurden die Definitionen ENDA und ENDB in der Symboltabelle verfügbar zu der Zeit als der ORG-Befehl assembliert wurde, so daß die Programme automatisch zusammen im Speicher liegen. Das zweite mal assembliert wird jedes Programm, das wir als ein bewahrtes SYMBOL TABELLEN OPTION gebrauchen, damit alle Definitionen verfügbar sind. Aber da der Zweck des ersten Durchlaufs darin lag, die Symboltabelle, und dies haben wir schon gemacht, aufzubauen, benutzen wir auch BIT 7 (Durchgang 2) OPTION, um die Schaffung von Doppeldefinitionen zu verhindern.

## 6.12 VALETTE

Dies deckt fast alles ab, was wir über den Assembler sagen wollen. Das nächste Kapitel handelt vom Korrigieren von assembliertem Objectcode. Es ist ein guter Gedanke, Ihren Objectcode auf Kassette zu save, bevor Sie das erste mal versuchen, es auszuführen, da Fehler das System zum zusammenbrechen bringen. Wenn Sie Labels bei Ihren DEBUG-Befehlen benutzen wollen, sollten Sie auch die Symboltabelle save. Erinnern Sie sich daran, daß ASZMIC nach der Assemblierung in den EDIT-Modus zurückkehrt und beginnen Sie jedes Programm mit ORG.

## Kapitel 7

### Programmausführung und Test

Wenn Sie ein Programm geschrieben und assembliert haben, wollen Sie es natürlich auch laufen lassen. ASZMIC besitzt zwei Hauptkommandos zur Programmausführung und eines für die Ausführung mit Einzelschritten (wir sehen von den Möglichkeiten der Benutzung der Sofortbefehle CALL und JP ab). Wir sollten nun damit beginnen, nach den DEBUG-Befehlen für die Benutzung für Programmtests zu schauen.

#### 7.1 Der Befehl J

Dies ist der erste Befehl zum Programmtesten. Er kann durch eine Feldadresse genauer festgelegt werden, in welchem Fall die erste Instruktion, die ausgeführt wird, bei dieser Adresse festgesetzt wird oder bei sich selber, wobei in diesem Fall die Adresse in PC1 abgelegt wird, wenn als letztes eine vorkommende BREAK-Bedingung als Transferadresse benutzt wird. Der Befehl J kennzeichnet sich durch die Tatsache aus, daß alle Register vor der Ausführung von REGIM ab geladen werden und daß das I-Register auf 1 gesetzt wird, um die NMI-Handhabung und die Breakbedingungen zu erleichtern. Wenn Ihr Programm nicht die Registerinitialisierung aufruft, dann können Sie REGIM mit den Befehlen I oder D initialisieren.

Wenn Sie den Befehl J in Ihrem Programm benutzt haben und er einen Fehler beinhaltet, dann sind Ihre Chancen zu ASZMIC zurückzukommen, um herauszufinden, was passiert ist, gering, außer wenn Sie ein NMI-Interrupt eingefügt haben, um die Endlosschleife abzubrechen. Ihr erster Schritt als Schutzmaßnahme gegen solche Fälle ist der Breakpoint.

## 7.2 Breakpoint

Ein Breakpoint ist ein RST  $\emptyset$ . Er kann mit Hilfe des Befehls B eingefügt werden - in diesem Fall behandelt ASZMIC die Abspeicherung des Ursprungsbytes und manipuliert den gespeicherten Programmzeigerwert so, daß Sie mit dem Befehl weitermachen können, wenn Sie wollen - oder Sie können den Befehl B mit dem Befehl D oder durch das Plazieren im Ursprungscode einfügen, wobei Sie in diesem Fall einige SKIPPINGS oder Replazierungen, die Sie selbst benötigen, aufrufen. Der konventionelle Weg um einen Breakpoint zu benutzen, ist, Ihr Programm logisch aufzuteilen und dann den Befehl B dazu benutzen, am Ende des ersten Abschnitts einen Breakpoint zu setzen. Wenn das Programm bei der Ausführung am Breakpoint angelangt, können Sie mit den Befehlen B und O kontrollieren, ob alles Ihrem Wunsch entspricht und dann den Breakpoint an das Ende der nächsten Routine setzen und mit dem Befehl J zu dieser Routine zu springen. Den Vorgang sollten Sie wiederholen, bis Sie eine Routine mit Fehlern entdecken. Der fehlerhafte Programmteil ist meist der, von dem man es am wenigsten erwartet, und so ist es möglich, die Fehler zu entdecken und zu korrigieren. Erinnern Sie sich daran, daß RST  $\emptyset$  den Rücksprung bewirkt und das Ursprungsbyte ersetzt, wenn ein Breakpoint angetroffen wird; so daß, wenn Sie wiederum an diesem Punkt breaken wollen, die drei Kommandos G B J benutzen können, um nach dem Breakpoint weiterzumachen, ihn zu speichern und die Ausführung erneut zu beginnen. Wenn Sie einen Breakpoint festgelegt haben und ein anderes Programm in den Teil des Programmes, in dem sich der Breakpoint befindet, laden, dann wird nach der Bewegung des Breakpoints das Byte des alten in das neue Programm eingesetzt. Erinnern Sie sich daran, daß Sie keine Einzelschritte ohne SINGLE STEPPING innerhalb der Breakpointhandhabung machen können.

## 7.3 Der Befehl G

Wenn Sie einen Programmabschnitt mit Fehlern ausgemacht haben, können Sie mit dem Befehl G seine Funktionsweise untersuchen. Dies ist die ASZMIC-Möglichkeit des Singlesteps.

Dies hat den gleichen Effekt wie beim Befehl J, außer daß nach jedem Befehl automatisch gebreakt wird. Die Breakvorgänge verursachen, daß der Zustand in REGIM gesichert wird. Wenn Sie eine Anzahl von Befehlen vor dem Rücksprung nach ASZMIC durchführen wollen, können Sie einen Schrittzähler als ein zweites Feld im Befehl G festsetzen. Das erste Feld ist in diesem Fall die Adresse zu der gesprungen wird; auch dann, wenn es sich um den gespeicherten Programmzählerwert handelt.

Der Gebrauch eines Schrittzählers mit dem Befehl G kann sehr nützlich sein. Erinnern Sie sich daran, daß, obwohl Sie nicht nach ASZMIC zurückspringen bis der Schrittzähler erschöpft ist, wird ein Break mit der Abspeicherung des jeweiligen Zustandes nach jedem Befehl gesetzt, so daß die Ausführung viel mehr Zeit als normalerweise benötigt wird in Anspruch nimmt. Da der Singlestep mit dem ZX81-NMI gekoppelten Timer arbeitet, um ein Break zu simulieren, werden Breakpoint und Singlestep gleich behandelt, was Ihnen die Möglichkeit eröffnet, einen ROM-Breakpoint zu simulieren. Wenn Sie ein selbstgebranntes EPROM mit der Startadresse :2000 haben und Sie wollen einen Breakpoint bei :2122 setzen, gebrauchen Sie den B :2122-1 Befehl, und führen Sie die Eprom-Routine mit G :2000 32767 aus. ASZMIC führt dann SINGLE STEPS bis zum Befehl vor der Adresse :2122 aus, und dann macht es ohne Einzelschritte weiter.

Der Befehl G benutzt den ZX81-NMI-Interrupt-Timer auf eine eigentümliche Art und ist nicht völlig kompatibel mit Routinen, die den Bildschirm aufbauen, da Sie verlangen, daß das I-Register 14 beinhaltet. Ein wenig Scharfsinn bei den Breakpoints und den Singlesteps über fehlerfreie Gebiete wird Sie wahrscheinlich dazu befähigen, selbst ähnliche Probleme zu erkennen.

#### 7.4 Der Befehl O

Dieser Befehl, der die in REGIM gespeicherten Register auf dem Bildschirm zeigt, kommt beim Programmtesten vor. Die Register werden jeweils in 2 Zeilen folgendermaßen dargestellt:

```
PC HL HL'BC'DE'AF'
AF BC DE IX IY SP
```

8192  
= 2000H

Die '(Strich-) Register sind die Z80-Zweitregister. PC ist der Programmzähler, der Ihnen anzeigt, welcher Befehl als nächster ausgeführt wird. SP ist der Stackpointer, und der Rest sind die Standardregisterpaare. Es tut uns leid, daß wir nicht Überschriften über den Registern drucken konnten, aber es wird dazu genau so viel ROM benötigt wie für das Printerinterface, und außerdem ist der Vorgang dann zweimal so schnell, so daß wir meinen, es wäre besser auf die Dokumentation zu verzichten.

Es ist unglücklich, den O-Befehl als ein Shift MACRO zu platzieren, wenn Sie ein Programm im DEBUG-Modus bearbeiten (z.B. Drücken Sie Shift T und dann O an der obersten Zeile). Jedes Break, sei es beim Breakpoint oder Singlestep, gibt Ihnen automatisch einen Registerdump. Wenn Sie wichtige Variablen in Ihrem Programm, die Sie ebenfalls nach jedem einzelnen Befehl gedumped haben wollen, benutzen, können Sie Befehle mit der Shift-Macro-Zeile verknüpfen, indem Sie ;/ (Semikolon Strich) als Begrenzer zwischen jedem einzelnen Befehl verwenden und mit den Befehlen direkt hinter / beginnen. Diese Möglichkeit ist für DUMP-Befehle wirklich bedeutend. Aber die meisten Befehle, die funktionieren, benutzen keinen I-Befehl, da sie immer wieder zu Ihrem eigenen Ende springen.

## 7.5 Der Befehl I

Wir erzählten, daß einige CP/M-Debugger äquivalente I-Befehle haben, was beweist, daß es eine gute Idee ist, was viele Leute unabhängig entdecken werden, daß wir nicht die Ausführung von CP/M benutzt haben. Der Befehl I ermöglicht es Ihnen, eine Assemblerzeile zu schreiben, die sofort assembliert und ausgeführt wird. Vor der Ausführung werden die Register mit Hilfe des Befehls J geladen und nach der Assemblierung wird ein Breakpoint ausgeführt, so daß der Kontext erneut gespeichert wird, so daß der I-Befehl innerhalb des Programmkontextes ausgeführt wird. Vorausgesetzt Sie haben noch die ursprünglich assemblierte Symboltabelle, dann können Sie Labels Ihres Programmes im I-Befehl benutzen, so wie es auch mit anderen DEBUG-Befehlen möglich ist. Die winzige Einschränkung ist, daß ein I-Befehl den gespeicherten PC-Wert ändert, so daß Sie sich an die Adressen von J oder G erinnern müssen, wenn Sie I vor Ihnen benutzt haben, oder wenn Sie mit der Ausführung des ASZMIC-Stacks beginnen.

Der Gebrauch von I ist ein guter Weg, um einige unwichtige Unterlassungen in einem Programm zu berichtigen und mit dem Testen weiterzumachen.

Der Befehl I wird normalerweise zum Vorbereiten der Register gebraucht, aber Sie können auch jeden anderen Befehl damit ausführen. Wenn Sie keinen Leerraum zwischen I und dem Beginn des Assemblerbefehls lassen, dann wird vorausgesetzt, daß der Befehl mit einem Label beginnt, welches Sie mit = deklariert haben. Der einzige Haken ist, daß die Labeldefinitionsregeln in ASZFIC vorsehen, daß ein N/L als Begrenzer benutzt wird, so daß das I als Teil des Labels angesehen wird. Sie können jedes Label so definieren, daß es mit I beginnt.

## 7.6 Die Befehle D, F und C

Diese Befehle sind in Ihrer Funktion ziemlich außergewöhnlich, und wir erwähnen sie nur der Vollkommenheit halber. DUMP und MODIFY sind die Kernbefehle jedes DEBUG-Systems. Das Füllen ist zur Initialisierung von Buffern oder Arbeitsspeichern mit einem Wert zu gebrauchen. C kann, während der hauptsächlichste Gebrauch in der Neoadressierung des assemblierten Codes oder im Laden mit OFFSET liegt, oft nützlich sein, um komplexe Datenfelder im Speicher zu kopieren.

F und C sind ausgezeichnet geeignet zum Abstürzen des Systems, so daß Sie sich den Befehl noch mal anschauen sollten, bevor Sie N/L drücken.

## 7.7 VALETE

Ein Programm zu bearbeiten ist oft schwieriger, als eines zu schreiben, da Glück, Logik und Intuition unetnwirrbar miteinander verbunden sind. Logik und Eingebung (Intuition) kommt von Ihnen selber, aber wir wünschen Ihnen viel Glück bei Ihren Bemühungen.

## Kapitel 8

### Graphik

ASZMIC bietet Ihnen die Möglichkeit, hochauflösende Graphik zu erzeugen. Der Bildgenerator im BASIC-ROM benutzt eine festgelegte Anzahl von Rasterzeilen für jedes darzustellende Zeichen. Dies macht es Ihnen schwer, in BASIC ohne zusätzliche Hardware HI-RES-Graphik zu erzeugen.

ASZMIC besitzt einen speziellen Generator, der es erlaubt, die Parameter, welche den Bildschirm kontrollieren, selber zu bestimmen.

Sie können durch Maschinenprogramme eine 255 x 144 Bildschirmmatrix erzeugen; beim ZX81 kann während das Bild steht noch gerechnet werden.

#### 8.1 Der asynchrone Generator

Im Anhang wird dieses Programm KERNEL-Routine genannt. Sie hat die Funktion, jede 50stel Sekunde ein Bild zu generieren und die Tastatur zu kontrollieren.

Der Benutzer braucht nur den Bildschirm zu verändern, den Rest macht die KERNEL-Routine.

#### 8.2 PLOT

Die Plot-Routine wird durch CALL aufgerufen, wobei das B-Register die X und C die Y-Koordinate enthalten muß. Dann wird der Punkt an der entsprechende Stelle gesetzt. UNPLOT löscht den entsprechenden Punkt.

Diese Routine kontrolliert nicht, ob der Punkt innerhalb des Bildschirmbereichs liegt.

#### 8.3 LINE

Es gibt noch eine Routine, die die PLOT-Routine benutzt, um eine Linie zwischen zwei Punkten (X/Y) und (P/Q) zu ziehen. Die Register müssen D=Y, E=X, B=Q, C=P enthalten. ULINE macht auch hier genau das Gegenteil.

## 8.4 UPROG's

Dieser Name, den wir den Programmen, welche PLOT und LINE benutzen, gaben, ist sehr unschön. Wir bieten Ihnen zwei Beispiele an: STRUCTURES ergibt ein diamantförmiges Muster, MOIRE druckt dicht nebeneinander liegende Linien.

Eine sehr nützliche Methode zum Verschieben von weniger als 37 Bytes dreht in X-Richtung, ein Vielfaches von 36 in Y-Richtung. Dies geht schneller als löschen und neu zeichnen, ist außerdem einfacher. Sie können auch Text und Graphik mischen, wenn Sie den Text in einer Y-Koordinate, die ein Vielfaches von 8 von oben aus gesehen ist, 8 mal in den nachfolgende Zeilen drucken.

## 8.6 Andere Möglichkeiten

Sie können die Größe der vertikalen Pixel durch Ändern von PIXSIZE in der KERNEL-Routine verändern, die Anzahl der vertikalen Pixel durch Ändern von RASTERS, und die Anzahl der freien oberen Zeilen durch Ändern von TOPS.

Das Produkt plus TOPS plus NNN muß um die 3000 liegen, dann haben Sie wahrscheinlich ein synchrones Bild.

Also:

TOPS      beinhaltet die freien oberen Zeilen  
NNN      beinhaltet die Rechenzeit pro Einzelbild  
RASTERS beinhaltet die Anzahl der Zeilen  
PIXSIZE beinhaltet die vertikale Größe eines Pixels

## 8.7 Spezielle Operationen

$$1000 = 04096$$

Wenn ASZMIC neu anfängt, wird nach dem Initialisieren die Speicherstelle :1000 getestet und geprüft, ob darin ein JP-Befehl steht. Wenn ja, wird ein CALL :1000 ausgeführt.

Das bedeutet, Sie können ein eigenes ROM ans ASZMIC-ROM anfügen. Sie haben gesehen wie man DADDR dazu benutzen kann, um Befehlsabläufe einzuschieben, PRTJMP, um Ihre eigene Druckroutine zusammenzufügen und INTJMP, um BREAK-Bedingungen nach dem Speichern des Kontextes zu übernehmen. Es gibt noch eine Schlußbetrachtung, die wir noch nicht erwähnt haben. KEYJMP enthält normalerweise die Adresse KEYRET. Wenn Sie dort die Adresse Ihrer eigenen Routine einfügen, kann sie die Tastenüberprüfung übernehmen ( BC enthält das uncodierte Resultat von KEYBRE) und dann



entweder nach KEYRET zurückspringen oder nach LIX zurückkehren, wenn Sie alles erledigt haben.

Die Adresse SAVEMEM kann als Anspringspunkt benutzt werden, wenn Sie jemals ASZMIC reseten wollen, während es Daten enthält oder um ASZMIC für ein 16 K zu initialisieren. Das HL-Register wird mit der Adresse geladen, die die oberste Speicherstelle kennzeichnet und wird nach SAVEMEM gesprungen. Dies bedeutet "NEW" mit RANTOP-Veränderung.

Wenn Sie ein Programm bearbeiten, kann es sehr nützlich sein, einen externen Interrptknopf zu haben. Der Befehl J unterstützt dies, indem das I-Register mit 1 geladen wird, so daß ein einzelner NMI-Impuls den Effekt eines Breakpoints hat.

### 8.8 Der Schluß

Im Moment gibt es nur das reine ASZMIC-ROM auf dem Markt, aber wir hoffen, daß sie damit eine Menge anfangen können. Wir wünschen uns ebenfalls, daß ASZMIC Ihnen eine nützliche Hilfe zur Assemblerprogrammierung gibt. Wir hoffen, daß in einiger Zeit zusätzliche Hard- und Softwareentwicklungen auf dem Markt erscheinen. Ansonsten wünschen wir Ihnen viel Spaß mit ASZMIC.

## Anhang 1

### Generelle Informationen

#### Initialisierung

Nach dem Anschalten eines ZX80/81, der mit ASZNIC verbunden ist, wird der Speicherplatz in drei Teile geteilt. Der erste besteht aus den Systemvariablen, den Buffern, dem Stack und REGIM. Der zweite Teil, der zwischen DSPBGN und TXTLIM liegt, ist das Textfeld. Der dritte ist das Programm- und Datenfeld, welches zwischen TXTLIM und dem Speicheranfang liegt. Dieses Speichergebiet umfaßt ein Viertel des Speicherplatzes des gesamten Systems. Die Zeiger LABEND und LABSTK zeigen beide auf den Speicheranfang. Diese Zeiger definieren die Symboltabelle, und die Assembleroperationen sorgen dafür, daß LABEND die Adresse der obersten Stelle der Symboltabelle ist. LABSTK ist immer die oberste Stelle des Speichers. Das IV-Register enthält :4000, das I-Register 14, und der Interruptmodus ist auf 1 gesetzt.

Wenn irgendeine Taste gedrückt gehalten wird für mehr als eine halbe Sekunde, so beginnt das Autorepeat 8 Zeichen pro Sekunde zu produzieren.

Der Bildschirm besteht aus 34 Zeilen und 36 Spalten. Die ZX81-Hardware erzeugt automatisch eine neue Zeile für längere Zeilen, aber, da der EDIT-Bildschirm mit dem Zählen der N/L's arbeitet, ist dies eine Möglichkeit, die sparsam benutzt werden sollte.

## EDIT & DEBUG-Modus

ASZMIC hat 2 Modi: den EDIT-Modus, der durch einen schnellblinkenden Cursor zu erkennen ist und den DEBUG-Modus, dessen Cursor langsamer blinkt. Der Unterschied zwischen Ihnen ist, daß im DEBUG-Modus das Drücken eines N/L (Newline) eine beendete Zeile dem Kommandointerpreter übergibt. Wenn die ersten Buchstaben dieser Zeile im Gebiet A-P liegen, dann tritt ASZMIC in Aktion, da es sich bei der Zeile um einen Befehl handelt. Wenn dies nicht der Fall ist, wird sie ignoriert.

Tastenschläge, die nicht den Kommandointerpreter passieren, benutzen eine vertikale Synchronisationsuhr, die dafür sorgt, daß der Bildschirm während der Tasteneingabe nicht flackert. Der Befehlsinterpreter benutzt während seiner Tätigkeit keine solche Uhr. Der endgültige Effekt ist, daß Scroll- und Cursoroperationen ein schwaches Flackern erzeugen, daß aber der Bildschirm noch leserlich bleibt. Diese Möglichkeit besteht beim ZX80 nicht.

## Felder

Der Assembler und der Befehlsinterpreter gebrauchen beide eine Unteroutine, genannt GETFLD, die ein sehr zweckdienliches Feld für den Interpreter ist, genannt wird. Gültige Felder sind:

- a) Eine Dezimalzahl, die die Zeichen 0-9 (1-5 Stellen) beinhalten kann.
  - b) Eine hexadezimale Zahl, die durch einen : (Doppelpunkt), der vorangestellt wird, festgesetzt wird und die Zeichen 0-9 und A-F (1-4 Stellen) enthalten kann.
  - c) Ein \$-Zeichen, das bedeutet der derzeitige Inhalt des IX-Registers der (der als ein Adresszähler vom Assembler gebraucht wird).
  - d) Ein oder zwei Zeichen, die in Anführungszeichen (") eingeschlossen sind.
- (Die Punkte a) bis d) sind selbstdefinierte Felder.)

e) Ein Symbol. Dies ist ein Zeichenstring, der aus 3 oder mehr Zeichen innerhalb der Bedingung A-Z und 0-9 besteht mit einem Zeichen von A bis Z beginnt. Ein Symbol ist nur dann von Bedeutung, wenn es durch das Auftauchen innerhalb eines Labelfeldes eines assemblierten Befehls (durch A oder I) definiert wurde. Wenn einem Symbol ein ? (Fragezeichen) vorangeht, das auf ein Argument hinweist, dann sind die Regeln für eine Mindestanzahl an Zeichen und für das alphabetische Zeichen am Anfang hinfällig. Das ? ist nur dazu da, den nachfolgenden String als ein Symbol zu identifizieren und ist nicht Teil des Symbol selbst.

f) Jede Kombination der Punkte a) bis f) müssen durch das Zeichen + oder - getrennt werden. + verursacht eine Addition innerhalb der Felder, verursacht eine Subtraktion. Klammern werden nicht benutzt. (Eine linke Klammer setzt BFLAG auf nicht Null (1).)

Felder werden mit Leerzeichen oder Kommata abgetrennt, nur mit einer der beiden Möglichkeiten.

Beispiele:

"A\*"

12345

:12345

FUDGE

AC COUNTS.PAYABLE

\$

12345+"A\*"+:12345-FUDGE-?HL+ACCOUNTS.PAYABLE

Files

Ein File ist ein Stück eines Textgebietes, das durch einen Filenamen an seinem Anfang identifiziert ist und dessen erstes Zeichen der letzten Zeile ein Filemark (§) ist. Der Filename sollte als erstes Zeichen ein Filemark (§) haben und kann Zeichen von A-Z und 0-9 beinhalten.

Beispiel:

LAUTER UNSINN

~~A~~FILE1

DIES IST EIN

BEISPIEL

EINES FILES

~~A~~

NOCH MEHR UNSINN

Das File ~~A~~FILE1 ist mit den 3 Zeilen

DIES IST EIN

BEISPIEL

EINES FILES

definiert und sollte mit den Befehlen wie A (nicht zu empfehlen in diesem Fall), P,K und E benutzt werden. Der Benutzer kann so viele Files, wie er möchte definieren, vorausgesetzt jeder Filename ist einmalig.

#### Aufteilungen (PARTITIONS)

Der Anwender kann mehr File- oder Programmplatz für sich selbst schaffen, indem er den TXTLIM-Zeiger, der die Grenze zwischen Text- und Programmgebiet definiert, herbiegt. Benutzen Sie DUMP & MODIFY- oder IMMEDIATE-Befehle. Die Symboltabelle sollte mit LABEND (niedrigste Speicherstelle der Symboltabelle) und LABSTK (höchstgelegene Speicherstelle der Tabelle) Zeigern neu adressiert werden und der Inhalt des Speichergebietes, das dazwischen liegt, verschoben werden, wenn die Tabelle nicht ohnehin schon leer ist. Wenn Sie keine standardmäßige Speichereinheit für den ZX80/81 haben, dann ist die Art, in der der Speicherplatz aufgeteilt werden kann, abhängig von der Decodierung der Einheit, die Sie verwenden. Beachten Sie, daß die Ausführung eines mehr als 32K Programmes die ZX80/81-Bildschirmlogik aktivieren kann, weil 16-32 K & 48-64 K-Regionen gleich adressiert werden können.

## Der Gebrauch von ASZMIC-Routinen

Interne Routinen in ASZMIC können benutzt werden, um dem Anwender bei der Handhabung des I/O und verschiedener ENCODING- und DECODING-Funktionen zu helfen. Die folgenden werden normalerweise von ASZMIC selber benutzt (Systemvariablen bestehen intern aus IY=:4000, I=14, IM1 und 22 Bytes des verfügbaren Stacks).

Die Routinen schließen

GETFLD - decodieren eines Feldes  
PUTDE - codieren einer hexadezimalen Zahl  
WRITA - codieren eines einzelnen Hexbytes  
WSTRNG - druckt den Inhalt des Druckerpuffers auf dem Bildschirm  
NRM2 - druckt ein Zeichen auf den Bildschirm  
OUTFRM - druckt den Bildschirmrahmen aus  
KEYBRD - decodiert die Tastatur  
KEYINT - wandelt die Decodierung in ein Zeichen um  
RDCASS - liest ein Zeichen von Kassette  
WRCASS - schreibt ein Zeichen auf Kassette  
PRNT - druckt eine Zeile auf dem Drucker

und viele andere ein. Schauen Sie sich die Verwendungsbedingungen für Definitionen, das Aufrufen von Folgen und Beispielen an.

## Assembler

Der Assembler ist ein kleines, sehr schnelles Unterprogramm innerhalb von ASZMIC, das alle ZILOG-Standardmnemonics als Assemblerbefehle akzeptiert. Die Anweisungen DEFB, DEFW, DEFB & ORG sind auch enthalten. Die EQU-Direktive ist in einer nicht-standardisierten Form enthalten. Gebrauchen Sie anstatt "LABEL EQU Wert" "LABEL=Wert" ohne eingefügtes Leerzeichen. Die DEFS-Wertdirektive bleibt nicht bestehen: Gebrauchen Sie ORG \$+Wert stattdessen, was den gleichen Effekt der Speicherreservierung hat. Es gibt keine bedingte Compilation, Listinkontrolle oder Macrodirektive (daher ist die SET-Direktive nicht implementiert).

Ein Kommentarfeld kann nach jedem Assemblerbefehl angehängt werden, wenn ein ; (Semikolon) vorangegangen ist. Ein ; in Spalte 1 macht die gesamte Zeile zur Kommentarzeile.

Starten Sie immer jedes File, das assembliert werden soll, mit einem ORG.

ORG- und EQU-Direktiven sollten keine Vorwärtsbeziehungen gebrauchen. Assembleroptionen (siehe Befehl A in Anhang 3) können zur Kontrolle der Erzeugung des Objectcodes und der jeweiligen Listings gebraucht werden.

Fehler werden getestet nach Labelfehlern (nicht- oder doppelt-definierte Labels), OP-Code- und Sprungfehlern bei relativen Sprüngen bzgl. des Bereiches. Ein Fehler wird durch eine nicht leere erste Spalte angezeigt, und Fehlerzeilen werden sogar aufgelistet, wenn kein Listing gefordert wurde.

## Anhang 2

### Die Shifttasten

Dies sind die Tasten, welche die EDIT- und MACRO-Funktion von ASZMIC steuern.

#### Shift-Ø Das Drücken von RUBOUT

Der Cursor wird 1 Stelle nach links bewegt, der Buchstabe darunter glöscht und alle nachfolgenden Zeichen rechts davon werden eine Stelle nach links gerückt. Das Positionieren des Cursors an den Zeilenanfang und das anschließende Benutzen von Shift-Ø löscht das davorstehende N/L, so daß die beiden Zeilen verknüpft werden.

#### Shift-9 Aufruf des DEBUG-Modus

Shift-9 setzt ASZMIC in den DEBUG-Modus. Erinnern Sie sich an die gegenwärtige Cursorposition zum Gebrauch von Shift-E. Stellen Sie den Cursor auf die oberste Zeile der ersten Seite, und bewegen Sie den Bildschirmzeiger so, daß diese Zeile auf dem Bildschirm erscheint. Das DEBUG-Modus-Flag sorgt automatisch dafür, daß der Cursor nun langsam blinkt.

#### Shift-8 CURSOR nach rechts

Bewegt den Cursor 1 Zeichen nach rechts; aber niemals auf ein N/L.



#### Shift-7 Cursor hoch

Diese Taste setzt den Cursor eine Zeile über die gegenwärtige Position. Wenn es erforderlich ist, wird der Bildschirm nach unten gescrolled, so daß der Cursor auf dem Bildschirm zu sehen bleibt. Bewegen Sie den Cursor nicht auf oder hinter ein End-of-Data-Zeichen.

#### Shift-6 Cursor runter

Diese Taste funktioniert wie Shift-7, nur daß der Cursor nach unten bewegt wird und dem entsprechend hochgescrolled wird.

#### Shift-5 Cursor nach links

Der Cursor wird durch diese Taste 1 Zeichen nach links bewegt, aber niemals auf ein N/L.

#### Shift-4 Seiten-Rauf-Sprung

Bewegt den Bildschirm 27 Zeilen nach oben. Außerdem wird der Cursor auf die Mitte der Zeilen gesetzt. Bewegen Sie den "Display" nicht auf oder über ein End-of-Data-Zeichen.

#### Shift-3 Seiten-Runter-Sprung

Wird analog zu Seiten-Rauf-Sprung gebraucht, so daß der Bildschirm 27 Zeilen nach unten rückt. Wenn Sie zur letzten Seite kommen, dann wird der Cursor auf die unterste Zeile gesetzt.

#### Shift-2 Löschen eines Files

Der ganze Text, von der derzeitigen Cursorposition bis zum ersten Filemark (¶), das entdeckt wird, wird gelöscht. Es passiert nichts, wenn kein abschließendes Filemark entdeckt wird. Wenn beim Löschen die Cursorposition bis zur letzten Bildschirmseite bewegt wird, wird der Cursor auf die unterste Zeile gesetzt.

### Shift-1 Zeilenlöschen

Durch die Taste wird die Zeile, auf der sich der Cursor befindet, gelöscht. Sie setzt den Cursor an den Anfang der nächsten Zeile. Es passiert nichts, wenn sich der Cursor auf der untersten Zeile der letzten Seite befindet.

### Shift-T Geht zur obersten Zeile

Die Taste setzt den Cursor auf die oberste Bildschirmzeile, die Zeile der Shift-Macro-Definitionen.

### Shift-R Shift-Macro-Ausführung

Unabhängig vom derzeitigen ASZMIC-Modus (EDIT/DEBUG) wird der Inhalt der Shift-Macro-Zeile dem Befehlsinterpreter zur Ausführung übergeben. Cursor und Bildschirmseite sind unverändert, sieht man davon ab, daß bei der Ausführung der DEBUG-Befehle etwas verändert werden kann. Wenn die Zeile leer ist, passiert nichts.

### Shift-E Rückkehr in den EDIT-Modus

Durch diese Taste gelangt ASZMIC in den EDIT-Modus zurück. Der Cursor wird an die Stelle gesetzt, wo er sich befand, wenn HOME (Shift-9) als letztes gedrückt wurde. Es wird die Bildschirmseite falls nötig umgeändert, da der Cursor auf dem Bildschirm zu sehen bleiben muß. Ein schnelles Blinken des Cursors identifiziert den EDIT-Modus.

### Shift-W RIGHT JUSTIFY Cursor

Diese Taste bewegt den Cursor an die Position der derzeitigen Zeile, die am weitesten rechts ist. (Beachten Sie, daß es kein Gegenstück zur linken Seite gibt.)

## Shift-Q EDIT-Rubout

Diese Taste entspricht dem normalen Rubout, außer daß der Cursor nicht nach links gesetzt wird und daß dafür das Zeichen gelöscht wird, auf dem sich der Cursor befindet. (Es ist somit nicht möglich, das letzte Zeichen einer Zeile zu löschen.)

## Shift-G Mischen

Nach dem Drücken dieser Taste wird vom Anfang des Bildschirmfiles an gesucht, um ein Mischzeichen (›) zu finden. Der gesamte Text danach, außer dem nach einem Filemark, wird an die Textposition kopiert, an der sich der Cursor befindet. Es kommt zu einem Disaster, wenn das Mischzeichen und das Filemark fehlt. Im DEBUG-Modus wird, wenn ein N/L kopiert wird, die Zeile, die davon abgeschlossen wird, dem Befehlsinterpreter übergeben, so daß der kopierte Text ein Befehlsmakro wird. Im DEBUG-Modus hat Shift-G den gleichen Effekt wie der Befehl M› (siehe Anhang 3).

Die Shift-D und Shift-F Tasten sollten in Ihrem System implementiert werden. Sie sind wie Shift-G, aber benutzen \* und < als Startzeichen.

Die Eingabe eines normalen Zeichens verursacht, daß alles unter und rechts vom Cursor eine Stelle nach rechts gerückt wird und das Zeichen unter dem Cursor plaziert wird. Der Cursor wird ebenfalls um eine Stelle nach rechts verschoben. Der EOD-Zeiger wird um eins erhöht.

Die Eingabe eines N/L's verursacht, daß ein zusätzliches Leerzeichen an die Zeile angehängt wird, wenn das vorhergehende Zeichen keines war. Der Cursor wird nach dem N/L zum Anfang der nächsten Seite verschoben. Die Bildschirmseite wird eine Zeile nach unten bewegt (d.h. Der Text wird eine Zeile nach oben gescrolled). Der EOD-Zeiger wird wie gefordert um eins erhöht.

## Anhang 3

### DEBUG-Befehle

Immer dann, wenn ein M/L ins Textgebiet von ASZMIC geschrieben wird und man die Zeile im DEBUG-Modus beendet hat, wird sie dem Befehlsinterpreter übergeben, der den Befehl anhand des ersten Buchstaben einer Zeile identifiziert und die entsprechende Unteroutine aufruft. Wenn das erste Zeichen nicht zwischen A und P liegt, wird es ignoriert.

Verschiedene DEBUG-Befehle dürfen innerhalb einer einzigen Zeile verkettet werden, indem man die Zeichenfolge ;/ (Semikolon Strich) zur Trennung benutzt, und das nächste Kommando beginnt sofort nach dem Strich, z.B.

D 0 3;/D 5 3;/D :7000 10

Die Inhalte der Shift-Macro-Zeile werden auch dem Befehlsinterpreter übergeben, wenn Shift-R gedrückt wurde und diese Zeile wird auch ausgeführt, wenn ein BREAK-Befehl vorhanden ist. (Breakpoint, RST 0, Singlestep, externes NMI.)

## A... ASSEMBLE

### Eine FILENAMEN Möglichkeit

Das benannte File wird identifiziert und bis zu seinem abschließendem Filemark unter der Kontrolle des OPTION-Feldes assembliert.

Das OPTION-Feld wird in ein 8-BIT BYTE verwandelt, dessen Bits, wenn sie gesetzt sind, folgendes bedeuten:

- BIT 7.. (128).. zwei Durchläufe
- BIT 6.. ( 64).. keine Erzeugung des Objectcodes
- BIT 5.. ( 32).. Stufenmodus des Druckers. Keine Verstümmelung von Listingzeilen.
- BIT 2.. ( 4).. Erhalten und Addieren auf eine vorhergegangene Symboltabelle
- BIT 1.. ( 2).. direkte Listingausgabe auf dem Drucker
- BIT 0.. ( 1).. erzeugt Assemblerlisting

Ein OPTION-Feld, zur Erzeugung eines Listings auf dem Drucker ohne Objectcode, würde :43 (dezimal 67) oder 64+2+1 sein (Sie können es so schreiben).

Assemblerzeilen beginnen entweder mit einem ; in Spalte 1, in welchem Fall die Zeile als Kommentar behandelt wird, oder einem Nicht-Leerzeichen, in diesem Fall wird damit gerechnet, daß das erste Zeichen Teil eines Symbols ist, um es zu definieren, oder ein Leerraum. Darauf folgt dann ein OP-Code oder eine Assembleranweisung, die durch ein Leerzeichen getrennt ist, plus bis zu zwei Argumente, die durch ein Komma getrennt werden. Ein Kommentarfeld kann die Zeile beenden, wenn ihr ein ; (Semikolon) vorangegangen ist. Das File wird von einer Zeile mit einem Filemark in Spalte 1 beendet.

Beispiel:

**f**EXAMPLE

ORG :7000

NRM2=:492 ;KONTROLLWERT FÜR IHR SYSTEM IN ANHANG 4

START LD HL,\$+120 ;ADRESSE DER TABELLE

LD B,TABLEND-TABLE

LOOP LD A,(HL)

PUSH HL

PUSH BC

CALL NRM2

POP BC

POP HL

INC HL

DJNZ LOOP

; UND NUN ZURÜCK ZU ASM2IC

RST 0

;

ORG :7000+120

TABLE DEFM "TEST"

TABLEND=

**f**

A **f**EXAMPLE 1

Beachten Sie:

Wenn die Variable OFFSET nicht Null ist, wird ihr Wert dazu benutzt, den produzierten Objectcode neu zu adressieren, obwohl der Code selber durch die ORG-Adresse für die Ausführung festgelegt wird. OFFSET läuft um 0,d.h. ORG :8000 und OFFSET :C000 setzt den Objectcode an die Stelle :4000.

B... BREAKPOINT

B Adresse

B

Wenn eine Adresse festgelegt wird, dann wird der derzeitige Breakpoint zurückbewegt (gesavete Bytes werden zurück an die Stelle der derzeitigen Breakpointadresse gesetzt) und die festgesetzte Adresse wird die neue Breakpointadresse.

Das Byte an dieser Adresse wird gespeichert und ein RST Ø (:C7) tritt an diese Stelle. Wenn der Breakpoint bei der Programmausführung angetroffen wird, wird das gesavete Byte automatisch für das Wiederbeginnen der Ausführung wiedereingesetzt. Wenn der Befehl B ohne Adresse angegeben wird, dann wird ein RST an die Adresse des derzeitigen Breakpoints gesetzt.

#### C... COPY

C, von, bis, Bytezähler

Eine intelligente Kopieroperation von der festgelegten Anzahl an Bytes ab der ersten Adresse bis zur zweiten. Wenn die Ursprungs- und die Zieladresse sich überlappen, geht ASZMIC so vor, daß die Daten innerhalb der Zieladresse nicht verdorben werden.

#### D... DUMP

D Adresse Bytezähler

D Adresse

Im ersten Fall wird ein formatierter DUMP der angegebenen Bytezahl ab der Startadresse produziert. Es gibt 8 hexadezimale Bytes pro Zeile mit einer hexadezimalen Adresse. Lange DUMP's verursachen eine Leere des Bildschirms für eine Sekunde, während der DUMP erzeugt wird. Die Breaktaste unterbricht einen DUMP.

Das zweite Beispiel ist das von einem DUMP & MODIFY-Modus. Der Inhalt der angegebenen Adresse wird auf dem Bildschirm ausgegeben und ASZMIC wartet auf eine Eingabe. Folgende Bytes werden im Speicher ab der gleichzeitig eingegebenen Adresse plaziert. Eingaben werden mit einem . (Punkt) hinter der Adresse abgeschlossen. Dies ist der einzige Fall, bei dem hexadezimale Felder nicht mit einem vorgestelltem : (Doppelpunkt) geschrieben werden. Der Doppelpunkt wird vorausgesetzt, und dezimale Werte müssen als Ø+ dezimalem Feld eingegeben werden.

E... EDIT

E

E Symbol

Wenn kein Argument angegeben wurde, setzt sich ASZMIC selber in den EDIT-Modus (schnell blinkender Cursor). Hinzu zu fügen ist, daß, wenn ein Symbol angegeben wurde, nach dem String aufwärts vom Ende des Files gesucht wird, und der Cursor an dessen Anfang gesetzt wird. Die Bildschirmseite wird verändert, wenn es nötig ist. Es findet keine Reaktion statt, wenn das Symbol nicht gefunden wird. (Jedes Zeichen außer  $\text{f}$  .  $\emptyset$ -9 A-Z beendet die Symbolvergleichsoperation.  $\text{f}$  ist nur als erstes Zeichen erlaubt.)

F... Fill

F von, bis, Füllbyte

Innerhalb des angegebenen Bereichs werden die Adressen mit dem Füllbyte gefüllt.

G... GO nur für den ZX81

G

G Adresse

G Adresse Schrittzähler

Dies ist ASZMIC's SINGLE STEP-Möglichkeit. Wenn keine Argumente angegeben sind, wird der Inhalt von REGIM eingesetzt und die Ausführung des Befehls an der gesaveten Programmzähleradresse (PC1) findet statt. Ein SINGLE STEP-BREAK wird am Ende des Befehls erzeugt, der neue Kontext gespeichert in REGIM, (wenn die Variable INTJPM verändert wurde, um eine andere Adresse zu beinhalten als INTRET, ereignet sich ein Sprung zu INTJMP an dieser Stelle.) und die Shift-Macro-Zeile wird vor der Rücksprungkontrolle zu ASZMIC ausgeführt.

Wenn eine Adresse angegeben wurde, dann überschreibt sie die gespeicherte Programmzähleradresse (PC1) und wird die Adresse, an der die Befehle ausgeführt werden.



Wenn ein Schrittzähler angegeben wurde, dann geht die Operation wie darüber vonstatten, aber nachdem der Kontext gespeichert wurde und vor der MACRO-Ausführung, wird der gespeicherte Schrittzähler um eins erniedrigt und, wenn er noch positiv ist, dann wird der Kontext wieder eingesetzt und der nächste Befehl wird ausgeführt. Die Ausführungsrate ist hundert mal so groß wie die normale, so ein großer Schrittzähler benötigt etwas Zeit zum Durcharbeiten. Der größtmögliche Schrittzähler ist :7FFF (dezimal 32767). Der Befehl G arbeitet nicht auf dem ZX80.

Beachten Sie:

Die SINGLE STEP-Möglichkeit kann dazu benutzt werden, um einen ROM-Breakpoint zu simulieren. Setzen Sie den Breakpoint für die gewünschte Adresse (1), und gebrauchen Sie die G-Adresse 32767. Die Breakbenutzungslogik denkt sich dann, es hat einen Breakpoint erreicht und beendet den SINGLE STEP, wenn die Stoptaste erreicht wird.

H... Unbedingter Sprung

H Adresse

Es handelt sich um einen direkten Sprung zu der angegebenen Adresse im Kontext des ASZMIC Befehlsinterpreter. Das HL-Register zeigt auf die Befehlszeile nach der Adresse und die Routine, die angesprungen wird, kann mit einem einfachem RET abgeschlossen werden. Dies ist ein einfacher Weg, um Ihre eigenen Befehle zu verknüpfen.

I... IMMEDIATE

I Assemblerzeile

Eine ungewöhnliche Möglichkeit, die es sofort erlaubt zu assemblieren und die Assemblerbefehle auszuführen. Die Assemblerzeile folgt nach dem I (z.B. Wenn ein Leerzeichen dem I folgt, dann wurde kein Label festgelegt). Die Zeile wird assembliert zu Objectcode in den internen Stapel, gefolgt von einem BREAK und wird dann sofort in der internen Form des Befehl J ausgeführt. Es operiert im gespeicherten Programmkontext von REGIM.

Nach der Ausführung wird der neue Kontext wie für ein normales BREAK gespeichert. Labels sollten nur mit Hilfe des "LABEL=Wert"-Befehls in einem Sofortbefehl definiert werden, und die Anweisungen ORG, DEFM, DEFB, DEFW und die Befehle JR und DJNZ sollten vermieden werden.

J... JUMP

J

J Adresse

Dies ist wie beim Befehl G, außer, daß es keine SINGLE STEP-BREAK's bei der Ausführung gibt, die mit der Programmlogik weitermacht, außer wenn der Befehl B benutzt wurde, um einen Breakpoint irgendwo einzufügen. Der Effekt eines Breakpoints oder eines extern generierten NMI-Interrupts ist ähnlich dem eines SINGLE STEP INTERRUPTS.

K... Saven auf Kassette

K&S "i.d." &Filename

K&S "i.d." von bis

K&S "i.d." von bis L

ASZMIC benutzt das selbe Kassetteninterface wie der normale ZX80/81, aber die Weise, in der es gebraucht wird, ist völlig anders. ASZMIC kann entweder Files oder Speicherregionen (die vermutlich Programme oder Daten enthalten) saven. Wenn Sie K drücken, gibt es eine Wartepause von 5 Sekunden, um es Ihnen zu ermöglichen, den Kassettenrekorder einzuschalten; dann wird die Befehlszeile selbst auf Kassette zur Fileidentifizierung geschrieben; nach weiteren 5 Sekunden Pause wird das File oder das Speichergebiet herausgeschrieben.

Der Befehl muß mit der Zeichenfolge K Filemark S Leerraum Anführungszeichen, wie sie obengezeigt ist, um eine Titelzeile beim Laden zu erhalten, begonnen werden. Der in Anführungszeichen eingeschlossene String identifiziert den File für die Laderoutine, so wie es auch sonst beim ZX80/81 ist. Der Filename oder die Speicherregion, die gesaved wird, wird dann angezeigt. Wenn die Speicherbegrenzung von "Leerzeichen L" gefolgt wird, dann verändert der Ladebefehl LABEND, damit er den "von"-Wert, wenn die Region geladen wird, enthält.

L... LOAD

L "i.d."

Als Antwort auf diesen Befehl wird die Kassetteneingabe ständig überflogen, und wenn eine gültige Titelzeile gefunden wird, wird sie auf den Bildschirm geschrieben und nach 5 Sekunden sichtbar, dies erzeugt einen laufenden Inhaltskatalog der Kassette. Wenn "i.d." des Ladebefehl und die gesavete Titelzeile verglichen wird, dann wird die Titelzeile analysiert, um zum Schluß zu kommen, wenn ein File- oder Speicherladen verlangt wird und das folgende File geladen wird. In dem Fall, daß das Programm (die Daten) des Speicherladens in die auf der Titelzeile festgelegte Region geladen wird, es sei denn, die Variable OFFSET ist nicht auf den Wert Null gesetzt; in diesem Fall wird dieser Wert für eine Neuadressierung gebraucht. Wenn die Regionsbegrenzung in der Titelzeile von einem L gefolgt wird, dann wird die Region dazu benutzt, eine Symboltabelle zu sein und LABEND wird so verändert, daß es den "von"-Wert enthält. Dies nutzt aus, daß SAVE und LOAD in dem selben System Platz haben.

Beides, K- und L-Befehl, kann durch die Breaktaste abgebrochen werden.

M... MACRO

M Zeichen

Dies ist fast wie Shift-G, aber Sie können den Startidentifizierer selbst festlegen, statt des Gebrauchs von > als Begrenzer. Das Schlußsymbol ist wie immer {. Legen Sie M nicht mit sich selber fest, geben Sie immer ein Zeichen danach an, und erinnern Sie sich daran, daß es das erste Vorkommen des Zeichens, das den MACRO-START definiert (außer für die Shift-Macro-Zeile), ist.

N... NEW

(s. Kap 4, 9)

Lädt BC aus TXTLIN. Setzt HL auf :3CA (Basic NEW-Implementation). Springt nach TXTLIN. Es wird gebraucht in der Verbindung mit dem BASIC/ASZNIC-Board.

## O... OLD REGISTERS

O

REGIM ist ein Gebiet, das aus 2 Zeilen von je 6 vierstelligen hexadezimalen Zahlen besteht.

Die Register erscheinen in der Ordnung:

```
PC  HL  HL'  BC'  DE'  AF'
AF  BC  DE  IX  IY  SP
```

PC ist der Programmzähler, SP der Stackpointer, und der Strich (') zeigt die Schwesterregister an. Wir empfehlen, daß der Anwender ein O-Befehl in der Shift-Macro-Zeile platziert, da die Register dann auf dem Bildschirm ausgegeben werden, wenn ein BREAK benutzt wurde.

## P... PRINT

P &Filename

Der benannte File wird auf dem Drucker ausgedruckt bis ein abschließendes Filemark entdeckt wird als ersres Zeichen einer Zeile. Die Operation kann jeder Zeit durch BREAK abgebrochen werden.

Beachten Sie:

Wenn die Variable PRTJMP einen anderen Wert als PRTRET hat, dann wird die Adresse darin als die Adresse der Zeilendruckroutine benutzt. Dies macht die Implementation von anwendereigenen Druckroutinen nötig.

