Schneider Computer Division
Silvastraße 1
8939 Türkheim

Schneider DEVPAC

Assembler/Disassembler

SW 216

Erste Ausgabe 1984
Originalausgabe in Englisch

Deutsche Bearbeitung: ESCON GmbH, Freising

Schneider DEVPAC

Der komplette HiSoft Assembler, Disassembler, Editor und Monitor für den Schneider CPC464

Herausgegeben von der
Schneider Computer Division
Silvastraße 1
8939 Türkheim 1

Ablauf G3S3.7
Anzeige M3S1.1
Arbeitsbereich M3S2.6
Argumente G3S3.2
ASCII M3S2.11, M3S2.6
Assemblieren 0.4, G3S3.7, G3AN3.3, G3AN3.4
Assembler 0.1, M3S2.15
Assembler Kommandos G3S2.3, G3S2.9, G3AN2.1
Assembler Anweisung 0.3, G3S2.3, G3S2.5, G3S2.8, G3AN2.1, M3S2.6
Assemblierungsliste 0.4, 0.7, G3S2.3
Assembler-Befehl G3S2.3
Auflisten G3S2.10
Ausdrucke G3S2.5, G3S2.8
Ausführen Code M3S2.3

BC G3S3.9
Bedingte Pseudo-Befehle G3S2.9
Beispiel G3S3.9
Befehlszähler G3S2.2, G3S2.3, G3S2.5, G3S2.7
Bildschirmaufbereitung G3S3.4
Blattvorschub G3S2.9

Cassettengeschwindigkeit G3S3.7

DEFB G3S2.8, M3S2.6
DEFM G3S2.8
DEFS G3S2.8
DEFW G3S2.8
Datenbereiche M3S2.6
Debugger G3S3.9
Durchnumerieren G3AN3.3

Editieren G3S3.5
Editor G3S3.1
Einfügen, einbinden G3S2.1, G3S2.10, G3AN3.1, G3AN3.5
Einfüge-Modus G3S3.3, G3S3.6
Einzelschritt M3S2.11, M3S2.13
Einzelschrittverfahren M3S2.3, M3S2.5, M3S2.10
Ersetzen G3S3.4, G3S3.6
EQU G3S2.4, G3S2.5, G3S2.8

Fehler-Nummern G3AN1.1
Fehler G3S2.2, G3S2.4
Finden G3S3.4, G3S3.5, G3S3.6

Help G3S3.8
Hexadezimal G3S2.10, M3S1.2, M3S2.1, M3S2.2

IF G3S2.9
Intelligentes Kopieren M3S2.2

Kommandos G3S2.4, G3S3.2
Komprimierung G3S3.1
Kopfzeile G3S2.9

Laden MONA3 ..... M3S1.1
Laden GENA3 ..... G3S1.1
List G3S3.4, M3S2.4, M3S2.14

Maschinen-Code 0.1
Magnetband G3S2.10, M3S2.9
Magnetband Kommandos G3S3.6
Modus ändern G3S3.6

Namen G3S2.3, M3S2.7
Neu numeriert G3S3.4

Registerveränderung M3S2.15
Relative Adressierung G3S2.7
Relativer Abstand M3S2.4, M3S2.8
Reservierte Wörter G3AN2.1
Run G3AN3.4

Speicherzeiger (Pointer) M3S2.1, M3S2.10, M3S2.9, M3S2.4, M3S2.13, M3S2.14
Speicherveränderung M3S2.14
Stack M3S2.6
Suchen Zeichenkette M3S2.1
Symboltabelle G3S1.1, G3S2.1, G3S2.3, G3S2.5, G3AN1.1

Term G3S2.6
TEXTEND G3S3.9, M3S2.7
Text-Ausgabe G3S3.3
Text-Aufbereitung G3S3.4
Text einfügen G3S3.3
Textdatei G3S3.1, G3S3.3, G3S3.9, M3S2.6, M3S2.7

Übereinstimmung G3S3.7
Überschreiben Speicher M3S2.5
Unterbrechung/Unterbr.punkt M3S2.9, M3S2.10, M3S2.13, M3S2.3, M3S2.5

Zeichenmodus G3S3.9
Zeilenendemarkierung G3S2.4, G3S2.8

Vorwort

Das DEVPAC von HiSoft ist weitgehend als das umfangreichste Software-Werkzeug für die Programmierung im Maschinencode anerkannt. Es umfaßt nahezu alle Möglichkeiten bedeutend größerer Systeme und ist das Ergebnis jahrelanger Entwicklung und Erfahrung.

Wenn Sie als Benutzer fest eingebaute Funktionen verwenden wollen, mit denen Sie im BASIC die Hardware ansprechen können, sollten Sie als Ergänzung das Firmware-Handbuch „SW158" von Schneider erwerben.

Die Seitennummern sind in der Form

(Kapitel).(Seite)

angegeben.

Darstellung in diesem Handbuch

Beachten Sie bitte die Art der Darstellung in den Veröffentlichungen der Schneider Computer Division, um die unterschiedlichen Verarbeitungsmöglichkeiten und Ablauffolgen in den Computerbefehlen zu erkennen.

Text, der über die Tastatur eingegeben wird, wird in OCR-B dargestellt:

```
10 FOR N=1 TO 50
```

Eingaben, die zur leichteren Lesbarkeit zwar möglich, aber nicht unbedingt notwendig sind, werden in Helvetica 75 dargestellt.

Funktionstasten werden in eckigen Klammern eingeschlossen

[P] Als Kommando, ohne ein Zeichen auszugeben
[ESC]

Allgemein erklärende und beschreibende Texte werden in kaum unterschiedlicher Art dargestellt, d. h. Century, Palatino, Times usw.

Das Software-Paket für den CPC 464

– HiSoft DEVPAC –

Einführung

Willkommen im Land der Assembler-Programmierung. Ein Gebiet, das für geduldige Programmierer eine große Belohnung (kleine und schnelle Programme), aber auch viele Fallgruben bereithält, in die der unvorsichtige Programmierer stürzen kann (wird).

In diesem Kapitel wollen wir versuchen, Ihnen einen Vorgeschmack zu geben, welche Vorteile Sie durch sorgfältige Programmierung im Assembler erreichen können. Falls Sie in dieser Art der Programmierung bereits Erfahrung haben, können Sie ruhig dieses Kapitel überspringen und in die anderen Kapitel einsteigen, in denen technische Einzelheiten des Softwarepakets enthalten sind.

Das Software-Paket besteht aus zwei aufeinander abgestimmten Programmen:

GENA3 – ein Z80 Assembler und
MONA3 – ein Disassembler/Debugger

Was sollen diese Bezeichnungen bedeuten?

Wie Ihnen zweifellos bekannt sein wird, „denkt" ein Computer binär, d. h. der gesamte Speicher und sämtliche Befehle bestehen nur aus einer Kette von „1" und „0" oder EIN und AUS bzw. Strom/kein Strom.

Eine höhere Programmiersprache, wie beispielsweise BASIC oder PASCAL, faßt mehrere Maschinenbefehle zusammen und bildet daraus englisch-ähnliche Kommandos, wie etwa **PRINT**. Der Vorteil dabei ist, daß diese höheren Anweisungen leicht verständlich und daher ziemlich einfach in der Anwendung sind. Der Nachteil ist, daß eine höhere Anweisung gleichzusetzen ist mit mehreren hundert, manchmal sogar mehreren tausend Maschinenbefehlen. So daß Programme in höheren Programmiersprachen langsamer ablaufen und einen größeren Speicherbedarf benötigen, obwohl sie eigentlich sehr einfach zu erstellen sind. Wenn wir also kleine, schnelle Programme schreiben wollen, erscheint es notwendig, Programme in Maschinencode zu erstellen, die der Computer direkt verarbeiten kann. Wie wir oben sagten, bestehen diese Befehle jedoch nur aus einer Kette von „1" oder „0" (oder Bits,

Binary-digits) und kein Mensch hat das Muster für alle Befehle im Kopf (oder möchte nachschlagen). Das müssen Sie auch nicht: Die Assemblersprache rettet Sie davor, indem sie für jeden Maschinencodebefehl eine auf dem Englischen basierende und an der Anweisung orientierte Verschlüsselung bereithält.

So können Sie ein Programm in einer niederen Sprache erstellen, indem Sie Merknamen als Befehle benützen, die jeweils eindeutig einem Maschinencodebefehl entsprechen. Dann müssen Sie Ihr Assemblerprogramm in den Maschinencode umsetzen (assemblieren). Dazu verwenden wir einen Assembler. GENA3 ist ein Assembler für Z80-Mikroprozessoren und enthält einen eigenen Datenaufbereiter (Editor), der es Ihnen ermöglicht, das Assemblerprogramm zu erstellen und dann auf einfache Weise im Dialog zu assemblieren (in den Maschinencode umzusetzen).

Gelegentlich wollen Sie vielleicht einen Teil eines Maschinencodes anschauen und versuchen, zu verstehen, wie er abläuft. Dazu wollen Sie den Maschinencode umsetzen in die entsprechende Assemblerbefehle, damit er leichter verständlich ist. Ein Programm, das diese Rückwärtsumsetzung durchführt, wird als Disassembler bezeichnet.

Sie hätten vielleicht gern noch zusätzliche Möglichkeiten, die Ihnen helfen, das Maschinenprogramm zu verstehen; wie ein Befehl den Zustand des Prozessors verändert – so daß Sie im Einzelschritt das Programm Befehl für Befehl ablaufen lassen können, um zu erfahren, wie die Z80-Register und Anzeiger (Arbeitsbereich des Prozessors) durch die Ausführung eines Befehls sich ändern. Sie möchten vielleicht eine Anzahl von Befehlen ablaufen lassen und dann anhalten, um den Zustand des Rechners zu untersuchen, d. h. Sie wollen einen Unterbrechungspunkt im Programm setzen, an dem die Ausführung anhält. Ein Programm, das alle diese Möglichkeiten bietet, wird als Debugger bezeichnet und hilft Ihnen, die schrecklichen kleinen Fehler auszumerzen.

MONA3 ist ein kombinierter Disassembler/Debugger, der Ihnen viele mächtige Kommandos zur Verfügung stellt, die helfen, Maschinencodeprogramme zu erforschen und zu verstehen.

Soviel darüber, was DEVPAC für Sie bereit hält.

Etwas durcheinander? Wir hoffen nicht!
Ungeduldig? Gut, dann wollen wir ein Assemblerprogramm erstellen:

Legen Sie Ihre DEVPAC-Kassette in den Datarecorder des CPC464, drücken Sie die Taste **[PLAY]** und geben Sie über die Tastatur

RUN" [ENTER]

ein.

Nach kurzer Zeit erscheint

```
Load Address?
```

auf dem Bildschirm. Lassen Sie den Datarecorder in dem Zustand, wie er ist und geben Sie über die Tastatur

1000 [ENTER]

ein.

Nun wird der Assembler in den Speicher des CPC 464 geladen und zwar ab Adresse **1000**. Nach dem Laden (es dauert ungefähr 135 Sekunden) läuft der Assembler automatisch an, gibt eine Bereitschaftsmeldung und das Zeichen '>' am Bildschirm aus, dem der normale Cursor folgt. Drücken Sie **[CAPS LOCK]** und geben Sie

```
I10,10[ENTER]
```

ein.

Sie erhalten dann auf der linken Seite des Bildschirms die Zahl **10**, gefolgt von einem Zwischenraum. Wir können nun einige Assemblerbefehle eingeben, indem wir jede Zeile mit **[ENTER]** abschließen. Die Zeilen werden dann mit der links angegebenen Zeilennummer in das Programm eingestellt.

Geben Sie nun ein:

```
ENT $ [ENTER]
LD B,26 [ENTER]
LD A,"A" [ENTER]
LOOP:CALL#BB5A [ENTER]
INC A [ENTER]
DJNZ LOOP [ENTER]
RET [ENTER]
```

Abschließend geben Sie **[CTRL]C** ein – d. h. halten Sie die Taste **[CTRL]** gedrückt und drücken Sie **C**. Sie sind nun wieder im Kommando-Modus des Editors, wieder mit dem Anforderungszeichen '>'.

Listen Sie Ihr Programm auf, indem Sie

L [ENTER] eingeben.

Beachten Sie, wie das Programm ordentlich in Spalten aufgelistet wird.

Was macht nun das Programm und wie bringen wir es zum Ablauf?

Nun wollen wir das Programm zeilenweise untersuchen:

```
10 ENT $
```

Dies teilt dem Editor lediglich mit, daß bei Ausführung das Programm an dieser Stelle zum Ablauf beginnen soll. **ENT** erzeugt keinen Maschinencode, es wird als ein Assemblerkommando bezeichnet.

```
20 LD B,26
```

Dies stellt den Wert 26 in eines der Z80-Register und zwar in Register B.

```
30 LD A,"A"
```

Dies stellt den Wert des ASCII-Zeichens **'A'** in das Z80-Register A

```
40 LOOP:CALL#BB5A
```

Ein **CALL**-Befehl wird benutzt, um eine Unterroutine irgendwo im Rechner aufzurufen. Die Subroutine an der Stelle **#BB5A** ruft eine Routine im Betriebssystem, die auf dem Bildschirm jeweils das Zeichen darstellt, das sich im A-Register befindet und dann auf den Befehl zurückkehrt, der dem **CALL** folgt. Es sollte jetzt also **'A'** ausgegeben werden.

```
50 INC A
```

Dies erhöht den derzeitigen Wert des Registers A um 1. Da alle ASCII-Zeichen einen aufsteigenden Wert haben, steht **B** im Register A

```
60 DJNZ LOOP
```

Dies ist ein ziemlich mächtiger Befehl. Er führt folgendes aus. Vermindere den Wert im Register B um 1; untersuche dann den Wert im Register B auf Null und verzweige nach **LOOP**, falls der Wert nicht Null ist.

Wir können mit diesem Befehl somit eine Folge von Befehlen öfter wiederholen, ungefähr so wie eine **FOR...NEXT**-Schleife.

Da hier das Register B den Wert 26 hatte, wird es durch **DJNZ** auf 25 vermindert und (weil B nicht 0) nach **LOOP** verzweigt. Nun gibt der Befehl bei **LOOP** wieder den Wert in Register A aus und **'B'** wird angezeigt.

Dann wird A wieder erhöht (**'C'**) und wir kommen erneut zu **DJNZ LOOP**. Wir bleiben also 26 mal in der Schleife bis das Register B gleich 0 ist; dann laufen wir durch zu

```
70 RET
```

Dies kehrt zum Editor zurück (**RETURN**).

Allgemein kehrt **RET** wieder aus einer Unterroutine zurück. In unserem Fall sind wir nicht in einer Unterroutine. Wenn wir jedoch das Programm ausführen, so wird es aus der Sicht des Editors als Unterroutine angesehen und kehrt zum Editor zurück.

Wie erreichen wir nun die Ausführung?
Im Augenblick haben wir nur den Quellentext des Programms; wir haben ihn noch nicht in Maschinencode umgesetzt. Wir machen dies mit dem Editor-Kommando 'A' (für Assemblieren).

Geben Sie

A [ENTER] ein.

Es erscheint die Meldung

```
Table size
```

In den meisten Fällen brauchen Sie hier keinen Wert einzugeben, sondern nur **[ENTER]** zu drücken.

Nun erscheint die Meldung:

```
Options:
```

Es gibt eine Vielzahl von Assembleroptionen, für uns genügt hier die Standardzuweisung und wir drücken deshalb wieder **[ENTER]**. Am Bildschirm wird jetzt eine Auflistung angezeigt, die Assemblerliste. In der Hauptsache zeigt sie den Maschinencode, welcher Ihrem Assemblerprogramm entspricht und wo das Programm abgespeichert wurde. Die ersten 4 Zeichen geben die Speicheradresse (in hexadezimaler Form, Basis 16) Ihres Maschinencodes an, gefolgt vom tatsächlichen Maschinencode.

So erkennen Sie, daß aus `LD B,26` ein `061A` generiert wurde. Die Darstellung `061A` wiederum ist eine andere Darstellung der tatsächlich gesetzten Bits, wobei jedem Zeichen 4 Bits entsprechen; d. h. `0` ist `0000`, `6` ist `0110`, `1` ist `0001` und `A` ist `1010`. Sie brauchen sich davon aber nicht verwirren zu lassen; das Wichtigste ist, daß Sie nun im Speicher einen Maschinencode erzeugt haben. Den können Sie jetzt ganz einfach ablaufen lassen, indem Sie

`R` **[ENTER]**

eingeben.

Was passiert? Sie sollten jetzt das Alphabet angezeigt sehen. Lassen Sie es nochmal ablaufen (drücken `R` **[ENTER]** und nochmals ... Einfach?

Jetzt ändern Sie das Programm. Geben Sie ein:

```
20 LD B,60 [ENTER]
A [ENTER]
```

Drücken Sie nach Table size? und Options? nur **[ENTER]** und lassen Sie das neue Programm durch

`R` **[ENTER]** ablaufen.

Diesmal erhalten Sie 60 Zeichen angezeigt. Assembler wurden geschaffen, um die Aufgabe der maschinennahen Programmierung zu vereinfachen und GENA3 enthält viele Hilfsmittel, die es Ihnen ermöglichen, leicht verständliche Programme zu erstellen.

Wir löschen nun das obige Programm (mit `D10,70` **[ENTER]**) und geben es neu in einer besser lesbaren Form ein. Beachten Sie, daß vom Lösch-Kommando lediglich das Assemblerprogramm betroffen wurde; es wurde nicht der Maschinencode im Speicher gelöscht (versuchen Sie erneut das R-Kommando).

Jetzt geben wir das neue Programm ein:

```
I10,10 [ENTER]
LAENGE: EQU 26 [ENTER]
ERSTES: EQU "A" [ENTER]
AUSGEB: EQU #BB5A [ENTER]
[ENTER]
ENT $ [ENTER]
LD B, LAENGE [ENTER]
LD A, ERSTES [ENTER]
SCHLEIFE: CALL AUSGEB [ENTER]
INC A [ENTER]
DJNZ SCHLEIFE [ENTER]
RET [ENTER]
[CTRL/C]
```

Listen Sie nun das Programm auf (mit `L` **[ENTER]**), es ist jetzt besser lesbar. Wir haben das Assembler-Kommando EQU angewendet (Assembler-Kommandos erzeugen, wie schon gesagt, keinen Code), um bestimmten variablen Namen einen gezielten Wert zuzuordnen. Zwar sind es nicht immer wirkliche Variablen, weil man

sie, vom vorgegebenen Problem ausgehend, nicht immer wieder verändern kann; sie geben dem Programm eine größere Aussagekraft. Assemblieren Sie nun das Programm (`A` **[ENTER]** und nach `Table size?` und `Options?` ebenfalls **[ENTER]**) und starten Sie es mit `R` **[ENTER]**. Das Ergebnis ist dasselbe, wie mit dem vorigen einfacheren Programm.

Wenn Sie in Bezug auf die Asemblerprogrammierung ein Neuling sind, sollten Sie sich ein gutes Buch kaufen (oder ausleihen) und in Verbindung mit dem Rest des Handbuchs durcharbeiten. Das klassische Buch dazu ist 'Programmierung des Z80' von Rodney Zaks, das jedoch für echte Anfänger schwer verständlich sein wird.

Durch die Entwicklung von billigen Mikrocomputern sind auch Bücher zur Assemblersprache immer öfter und umfangreicher erhältlich – denken Sie aber immer daran, daß Sie Bücher über den Z80-Assembler benötigen und nicht etwa über den 6502 oder 68000 usw.

Löschen Sie nun Ihr Programm mit

`D10,110` **[ENTER]**

und geben Sie folgendes ein:

```
I10,10 [ENTER]
ENT $ [ENTER]
COUNT: EQUI 10000 [ENTER]
PLOT: EQU #BBEA [ENTER]
MODE: EQU #BC0E [ENTER]
GRPEN: EQU # BBDE [ENTER]
[ENTER]
LD A,1 [ENTER]
CALL MODE [ENTER]
LD HL,300 [ENTER]
LD (RANUM),HL [ENTER]
LD BC,COUNT [ENTER]
[ENTER]
LOOP: PUSH BC [ENTER]
BIT 4,C [ENTER]
JP Z,NOSET [ENTER]
LD A,C [ENTER]
CALL GRPEN [ENTER]
NOSET:CALL RANDOM [ENTER]
PUSH HL [ENTER]
CALL RANDOM [ENTER]
POP DE [ENTER]
CALL PLOT [ENTER]
POP BC [ENTER]
DEC BC [ENTER]
LD A,B [ENTER]
OR C [ENTER]
JP NZ, LOOP [ENTER]
```

```
RET [ENTER]
[ENTER]
RANDOM: LD DE,(RANUM) [ENTER]
LD H,E [ENTER]
LD L,-3 [ENTER]
LD A,D [ENTER]
SBC HL,DE [ENTER]
SBC A,0 [ENTER]
SBC HL,DE [ENTER]
SBC A,0 [ENTER]
LD E,A [ENTER]
LD D,0 [ENTER]
SBC HL,DE [ENTER]
JP NC,RAN1 [ENTER]
INC HL [ENTER]
RAN1: LD (RANUM),HL [ENTER]
LD A,H [ENTER]
AND %00000001 [ENTER]
LD H,A [ENTER]
RET [ENTER]
[ENTER]
RANUM: DEFS 2 [ENTER]
[CTRL/C]
```

Nun assemblieren wir dieses Programm (**A [ENTER]**) – Sie können die Option 4 angeben (**4 [ENTER]**) nach der Options-Anforderung, um die Assemblerliste zu unterdrücken – und führen es (**R [ENTER]**) aus.

Sie werden feststellen, daß die Plotter-Geschwindigkeit ungefähr 1500 Punkte pro Sekunde beträgt. Wir könnten das Programm durch direkten Zugriff auf den Bildschirmspeicher sogar noch schneller machen. Doch wollen wir nun ein entsprechendes BASIC-Programm eingeben. Um aus dem Assembler zu kommen, geben wir

B [ENTER] ein.

Wir geben nun folgende Zeilen, jeweils mit **[ENTER]** abgeschlossen, ein:

```
NEW [ENTER]
AUTO [ENTER]
CLS [ENTER]
J%=1 [ENTER]
FOR I%=1 TO 10000 [ENTER]
X%=RND*639 : Y%=RND*399 [ENTER]
IF (I%MOD 20)=0 THEN J%=INT(RND*3)+1 [ENTER]
PLOT X%,Y%,J% [ENTER]
NEXT I% [ENTER]
[ESC]
RUN [ENTER]
```

Das Ergebnis ist bei weitem langsamer als bei der Maschinencode-Routine, annähernd um das 20-fache. Unbeachtet dessen ist es für BASIC dennoch eine gute Geschwindigkeit und normalerweise ist zu erwarten, daß Maschinencode-Programme bis zu 1000 mal schneller ablaufen, als entsprechende BASIC-Programme.

Wie Sie jedoch sehen, ist ein Maschinencodeprogramm bedeutend umfangreicher und schwieriger zu verstehen. Sie müssen in jeder Beziehung einen Kompromiß eingehen.

Informationshalber möchten wir hier die Zeiten für den „Sternenhimmel" angeben: BASIC 150 Sekunden, PASCAL 22 Sekunden, Maschinencode 8 Sekunden.

Wir hoffen, daß dieses Kapitel Ihnen den Mund wässrig gemacht hat, die Assemblersprache anzuwenden und legen Ihnen dringend nahe, das Handbuch langsam und sorgfältig durchzuarbeiten und die Programmbeispiele auszuprobieren.

Im Anhang 3 finden Sie ein ausführliches Beispiel zur Anwendung des Assemblers sowie im Kapitel 2.11 des MONA3-Handbuchs ein Beispiel zur Anwendung des Disassemblers/Debuggers.

Kapitel 1

Assembler und Editor – GENA3

GENA3 ist ein mächtiger und leicht anzuwendender Z80-Assembler, der sich in seiner Definition vom Standard-Zilog Assembler nur wenig unterscheidet; im Gegensatz zu vielen anderen Mikroprozessor-Assemblern ist GENA3 ein umfangreiches, professsionelles Softwareprodukt. Es wird Ihnen daher dringend nahegelegt, die folgenden Kapitel zusammen mit dem Beispiel in Anhang 3 sorgfältig zu studieren, bevor Sie versuchen, den Assembler anzuwenden. Wenn Sie ein vollkommener Computerneuling sind, sollten Sie zuerst den Anhang 3 und das Einführungsbeispiel durcharbeiten.

GENA3 belegt annähernd 7k Bytes wenn es geladen ist. Es hat einen eigenen, eingebauten Zeileneditor.

Dieser legt die Textdatei direkt hinter dem eigenen Objektcode an. Die Symboltabelle wird hinter der Textdatei angelegt. So ist beim Laden von GENA3 zu beachten, daß genügend Platz vorhanden ist, den Assembler selbst anzulegen, sowie die maximale Symboltabelle und die Textdatei, die Sie in der laufenden Anwendung bearbeiten wollen. Gewöhnlich wird GENA3 daher in einen niedrigen Speicherbereich geladen.

Um GENA3 in Ihren Computer zu laden, geben Sie einfach

```
RUN" [ENTER]
```

ein und drücken die Taste **[PLAY]** am Cassettenteil. Dies startet automatisch und verursacht das Laden eines kurzen BASIC-Programms, das die Meldung

```
Load address?
```

ausgibt.

Sie müssen eine Zahl zwischen `1000` und `30000` eingeben und **[ENTER]** drücken. Die Zahl, die Sie eingeben, bestimmt die Adresse, ab der der Assembler geladen wird – eine gute Adresse ist `1000`. Nach Eingabe dieser Zahl erscheint

```
Please wait ... loading GENA 3...
```

auf dem Bildschirm, und der Datacorder wird erneut automatisch gestartet. Jetzt wird der Assembler in den Computer geladen; das dauert ungefähr 135 Sekunden. Nach dem Ladevorgang erscheint als Bereitschaftsbestätigung eine „Hilfs"-Anzeige, in der alle Kommandos aufgelistet sind, die in Großbuchstaben aufgerufen werden können.

Achtung: Wenn Sie beabsichtigen, den Assembler GENA3 zusammen mit dem Disassembler/Debugger MONA3 gleichzeitig im Computer zu halten, sollten Sie immer GENA3 in den niedrigen Speicherbereich (z. B. `1000`) und MONA3 in den höheren Speicherbereich (z. B. `30000`) laden. In diesem Fall sollte GENA3 zum Schluß geladen werden.

Kapitel 2

Einzelheiten über GENA3

2.0 Wie GENA3 arbeitet

GENA3 ist ein schneller 2-Phasen-Z80-Assembler, der leicht an Z80-Systeme anzupassen ist. Der Assembler verarbeitet alle Standard-Z80-Befehle und beinhaltet zusätzliche Erweiterungen, die bedingtes Assemblieren, Assembler-Anweisungen und eine Binärbaum-Symboltabelle ermöglichen.

Wenn Sie eine Assemblierung anfordern (Eingabe des Kommandos `'A'` im Editor – siehe Kapitel 3) erscheint erst die Frage `'Table size:'`, die Sie mit einer Zahl beantworten müssen. Diese Zahl bestimmt die Speichergröße, die während der Assemblierung für die Symboltabelle zugeordnet wird. Wenn Sie durch einfaches Drücken der **[ENTER]** Taste diese Festlegung ihrem Computer überlassen, wählt GENA3 einen Bereich für die Symboltabelle, der abhängig von der Größe der Textdatei ausreichend sein müßte – im Normalfall gelingt dies hervorragend. Wenn Sie jedoch die 'include'-Option anwenden, müßten Sie einen größeren Speicherbereich als im Normalfall für die Symboltabelle bestimmen, da der Assembler nicht den Umfang der Datei vorhersehen kann, die eingefügt werden soll.

Nach `'Table size:'` werden Sie gefragt, ob Sie `'Options'` setzen wollen:

Geben Sie die Option-Nummer ein; bei mehreren gewünschten Optionen müssen Sie deren Nummern addieren und die Summe eingeben. Folgende Optionen sind möglich

Option-Nr.	Bedeutung
1	Erstellen einer Liste der Symboltabelle am Ende der Phase 2
2	Die Generierung des Objektcodes soll unterdrückt werden
4	Das Erstellen einer Assemblierungsliste soll unterdrückt werden
8	Die Assemblierungsliste soll auf dem Drucker ausgegeben werden
16	Der generierte Objektcode soll hinter der Symboltabelle abgelegt werden. Der Befehlszähler muß durch Angabe in der Assembleranweisung 'ORG' noch gesetzt werden, so daß der Objektcode zwar in einem Speicherbereich abgelegt, aber festgelegt wird, wo er ablaufen soll
32	Unterdrückung der Prüfung zur Ablage des Objektcodes – nützlich zur Beschleunigung der Assemblierung

Beispiel: Mit Option 36 (4 + 32) kann eine schnelle Assemblierung erreicht werden, da keine Assemblierungsliste erstellt und keine Prüfung durchgeführt wird, wo der Objektcode abgelegt wird.

Achtung: Bei Angabe der Option 16 wird die Steuerungsanweisung 'ENT' für den Assembler außer Kraft gesetzt. Um in diesem Fall die Lage des Objektcodes zu bestimmen, kann durch das Editor-Kommando **'X'** das Ende des Textes (die zweite ausgegebene Nummer) abgefragt werden. Zu diesem Wert muß der zugewiesene Speicherbereich für die Symboltabelle, sowie der Wert 2 addiert werden.

Die Assemblierung findet in 2 Phasen statt. Während des ersten Durchlaufs (Phase 1) prüft GENA3 auf Fehler und erstellt die Symboltabelle, im 2. Durchlauf wird (wenn Option 2 nicht gesetzt ist) der Objektcode generiert. Während des 1. Durchlaufs wird weder auf dem Bildschirm noch auf dem Drucker etwas ausgegeben, außer es trat ein Fehler auf. In diesem Fall wird die betreffende Zeile mit einer Fehlernummer darunter ausgegeben (siehe Anhang 1). Die Assemblierung wird unterbrochen. Durch Drücken von **[CTRL/C]** kann in den Editor verzweigt werden, jede andere Taste bewirkt ein Fortfahren der Assemblierung mit der nächsten Zeile. Nach Beendigung des 1. Durchlaufs erfolgt die Meldung:

```
Pass 1 errors: nn
```

Wenn Fehler erkannt wurden, wird die Assemblierung gestoppt und der 2. Durchlauf nicht durchgeführt. Wenn im Operandenfeld Namen benutzt wurden, die nicht im Namenfeld auftreten, wird für jeden nicht definierten Namen die Meldung **'*WARNING* label absent'** ausgegeben.

Während des 2. Durchlaufs wird der Objektcode generiert (außer Option 2 ist gesetzt) und eine Assemblierungsliste erstellt (außer Option 4 ist gesetzt oder die Assembler-Anweisung ***L-** ist inkraft).

Die Assemblerliste hat folgende Form

Spalte 1 (Stelle 1)	Spalte 2 (Stelle 6)	Spalte 3 (Stelle 15)	Spalte 4 (Stelle 21)	Spalte 5 (Stelle 28)	Spalte 6 (Stelle 33)
C000	210100	25	name	LD	HL,1

Bedeutung der Spalten

1 Wert des Befehlszählers bei Verarbeitungsbeginn dieser Zeile, außer es treten die Assembler-Anweisungen **ORG, ENT** oder **EQU** (siehe Abschnitt 2.6) in dieser Zeile auf. In diesen Fällen wird der Wert des Operandenfeldes der entsprechenden Anweisung in Spalte 1 eingetragen.
Der Wert in Spalte 1 wird normalerweise in hexadezimaler Form dargestellt. Durch Angabe des Assembler-Kommandos ***D+** (siehe Abschnitt 2.8) wird in Dezimalform ausgegeben.

2 Diese Spalte kann bis zu 8 Zeichen (entspricht 4 Bytes) enthalten und stellt den aus dieser Zeile generierten Objektcode dar.

3 Zeilennummer von 1 bis 32767

4 Falls eingetragen die ersten 6 Zeichen des für diese Zeile vergebenen Namens

5 Assemblerbefehl

6 Operanden (wenn nötig) und Bemerkungen

Die Ausgabe der Assemblierungsliste kann durch Drücken einer Taste angehalten werden. Ein nachfolgendes Drücken der Taste **[CTRL]C** bewirkt ein Verzweigen in den Editor, durch Drücken jeder anderen Taste wird die Ausgabe fortgesetzt.

Die einzigen Fehler, die während des 2. Durchlaufs auftreten können sind ***ERROR* 10'** (siehe Anhang 1) und **'BAD ORG!'** (tritt auf, wenn der Objektcode GENA3 oder die Textdatei oder die Symboltabelle überschreiben würde. Die Überprüfung auf diesen Tatbestand kann durch Setzen der Option **32** ausgeschaltet werden).

★ERROR★ 10 ist nicht verhängsnisvoll und Sie können mit dem 1. Durchlauf fortfahren, um weitere Fehler zu erkennen. Dagegen ist **'BAD ORG!'** schwerwiegend und die Steuerung geht direkt an den Editor.

Die Warnung **'**File Closed**'** kann auftreten, wenn eine Assembler-Anweisung **'ORG'** auftritt, während mit dem Assembler-Kommando ***T** der generierte Objektcode auf Band ausgegeben werden soll.

Es wird das ***T**-Kommando ausgeschaltet und die Objektcode-Banddatei geschlossen.

Am Ende des 2. Durchlaufs wird die Meldung

```
Pass 2 errors: nn
```

ausgegeben und es folgen Warnungen, wenn Namen nicht definiert wurden (siehe oben). Anschließend wird die Meldung

```
Table used: xxxxx from yyyyy
```

ausgegeben.

Diese Information besagt, welcher Bereich der Symboltabelle belegt wurde und wieviel zugewiesen wurde.

Wenn die Assembler-Anweisung **'ENT'** richtig verwendet wurde, erscheint jetzt die Meldung

```
'Executes : nnnnn'
```

Dies zeigt den Befehlszähler des Objektcodes an, um ihn mit dem Editor-Kommando **'R'** zu starten. Geben Sie das Kommando **'R'** nur ein, wenn die Assemblierung erfolgreich beendet wurde und die Meldung **'Executes : nnnnn'** erschienen ist.

Abschließend wird die Steuerung dem Editor übergeben.

2.1 Format der Assembler-Befehle

Jede Textzeilt, die durch GENA3 verarbeitet werden soll, muß folgendes Format haben, wobei bestimmte Felder wahlweise vorkommen können:

```
NAMENSFELD:   OPERATION   OPERANDEN   KOMMENTAR
```

Beispiel:

```
ANFANG:       LD          HL,name     ;lade HI mit name
```

Zwischenräume und Tabulatorzeichen (durch den Editor eingefügt) werden grundsätzlich ignoriert.

Die Zeile wird wie folgt verarbeitet:

Das erste Zeichen einer Zeile wird untersucht und wenn dies einem der Zeichen **';'** oder **'*'** oder Zeilenendemarkierung entspricht, wird die Zeile wie folgt behandelt

bei **';'** wird die gesamte Zeile als Kommentar betrachtet, d. h. tatsächlich ignoriert.

bei **'*'** wird das/werden die nächste(n) Zeichen als Assembler-Kommando (siehe Abschnitt 2.8) erwartet und alle Zeichen nach dem Kommando als Kommentar betrachtet.

bei <Zeilenendemarkierung> Leerzeile wird ignoriert

Wenn als erstes Zeichen ein Zeichen auftritt, das keinem der obengenannten entspricht, wird die Zeile nach einem nachfolgenden Tabulatorzeichen oder Doppelpunkt **':'** untersucht. Wenn zuerst ein Doppelpunkt gefunden wurde, werden alle vorhergehenden Zeichen zur Bildung eines Namens herangezogen, ansonsten wird angenommen, daß es sich um das erste Zeichen eines Assembler-Befehls handelt, wenn es nicht ein Tabulatorzeichen ist.

Nach Verarbeitung eines Namens, oder falls kein Name angegeben wurde, sucht der Assembler nach dem nächsten Zeichen, das nicht <Zwischenraum> und nicht <Tabulatorzeichen> ist und erwartet, daß dieses nun entweder eine <Zeilenendemarkierung> ist oder der Beginn eines Z80-Befehls (siehe Anhang 2), mit einer Länge bis zu vier Zeichen und abgeschlossen durch einen Zwischenraum/Tabulatorzeichen oder Zeilenendemarkierung. Wenn der Assemblerbefehl gültig ist, und einen oder mehrere Operanden erfordert, werden Zwischenräume/Tabulator übersprungen und die Operanden verarbeitet. Namendefinitionen können allein in einer Zeile stehen; dies erhöht die Lesbarkeit der Auflistung. Kommentare können irgendwo nach dem Operanden stehen, oder wenn ein Befehl keine Operanden benötigt, direkt nach dem Befehlsfeld.

Befehle müssen immer durch einen Zwischenraum oder **[TAB]** oder **[ENTER]** abgeschlossen werden.

2.2 Namen

Ein Name steht symbolisch für eine 16 Bit lange Information. Diese Information kann eine Adresse spezifizieren (eines anderen Befehls oder eines Datenbereichs) oder sie kann über die Assembler-Anweisung **'EQU'**(siehe Abschnitt 2.6) als Konstante verwendet werden.

Wenn einem Namen ein Wert zugeordnet wird, der mehr als 8 Bits beansprucht und dann in einem Zusammenhang verwendet wird, in dem nur eine 8-Bit-Konstante zulässig ist, erfolgt eine Fehlermeldung; z. B.

```
label EQU  #1234
      LD   A,label
```

verursacht den Fehler ***ERROR* 10,** wenn im 2. Durchlauf der zweite Befehl verarbeitet wird.

Ein Name kann aus einer beliebigen Anzahl von gültigen Zeichen bestehen (siehe unten), wobei jedoch nur die ersten 6 Zeichen signifikant sind und die einzelnen Namen in diesen ersten 6 Stellen eindeutig sein müssen (*ERROR* 4). Ein Name muß durch einen Doppelpunkt ':' abgeschlossen werden und darf selbst kein reserviertes Wort (siehe Anhang 2) darstellen. Als Namensteil ist ein reserviertes Wort jedoch zulässig.

Die Zeichen, aus denen sich ein gültiger Name zusammensetzen kann, sind 0 – 9, $, „A – z". Beachten Sie, daß „A – z" alle Groß- und Kleinbuchstaben einschließlich der Zeichen [, ,],^, ' und _ umfaßt. Ein Name muß immer mit einem Buchstaben beginnen!

Einige Beispiele von gültigen Namen sind:

```
SCHLEIFE:
schleife:
ein-langer-name:
L[1]:
a:
LDIR:(LDIR ist kein reserviertes Wort)
zwei^5:
```

2.3 Befehlszähler

Der Assembler führt einen Befehlszähler mit, so daß einem Symbol im Namensfeld eine Adresse zugeordnet und in die Symboltabelle eingetragen werden kann. Dieser Befehlszähler kann über die Assembler-Anweisung 'ORG' auf jeden beliebigen Wert gesetzt werden (siehe Abschnitt 2.6).

Das Symbol '$' kann dazu benutzt werden, um den gegenwärtigen Wert des Befehlszählers anzusprechen, d. h. LD HL,$+5 würde einen Code generieren, der das Registerpaar HL mit einem Wert lädt, der dem gegenwärtigen Befehlszählerstand +5 entspricht.

2.4 Symbol – Tabelle

Wenn ein Name zum erstenmal erkannt wurde, wird er in eine Tabelle eingetragen, zusammen mit zwei Informationsfeldern, über die er alphabetisch eingeordnet werden kann. Beim ersten Auftreten des Namens im Namensfeld wird sein Wert (bestimmt durch den Befehlszähler oder durch einen Ausdruck in der Assembleranweisung 'EQU') in die Tabelle eingetragen. Ansonsten wird immer dann der Wert eingetragen, wenn im folgenden der Name im Namensfeld gefunden wird.

Diese Art von Symboltabelle wird als Binärbaum-Symboltabelle bezeichnet. Ihre Struktur erlaubt das Eintragen in die Tabelle, sowie das Wiederauffinden des Namens in kürzester Zeit – bei umfangreichen Programmen eine äußerst wichtige Tatsache.

Die Länge eines Eintrags in der Tabelle variiert von 8 Bytes bis 13 Bytes, abhängig von der Länge des Namens.

Wenn während des 1. Durchlaufs festgestellt wird, daß ein Name mehrfach (im Namensfeld) definiert wurde, wird ein Fehler `'*ERROR* 4'` ausgegeben, da der Assembler nicht wissen kann, welchen Wert er zuordnen soll.

Wenn ein Name nicht definiert ist oder kein Wert zugeordnet wurde, wird am Ende der Assemblierung eine Meldung `'*WARNING* symbol absent'` ausgegeben. Das Fehlen einer Namensdefinition hindert den Assembler nicht, weiterzumachen.

Es ist zu beachten, daß nur die ersten 6 Zeichen eines Namens in die Tabelle eingetragen werden, um den Tabellenbereich klein zu halten. Am Ende der Assemblierung erhalten Sie eine Mitteilung, die aussagt, wieviel Speicher während der Assemblierung von der Symboltabelle belegt wurde – Sie können die Speicherzuordnung ändern, indem Sie nach Meldung `'Table'` zu Beginn der Assemblierung den gewünschten Wert eingeben (siehe Abschnitt 2.0).

2.5 Ausdrücke

Ein Ausdruck ist ein Operand, bestehend aus einem oder mehreren Termen (Formelteil), welche durch einen Operator getrennt sind. Nachfolgend die Erklärung für Term und Operator:

TERM

Dezimal Konstante	z. B. `1029`
Hexadezimal Konstante	z. B. `#405`
Binär Konstante	z. B. `%10001010`
Zeichen Konstante	z. B. `'A'`
Namen	z. B. `L1029`

Zusätzlich kann '$' verwendet werden, um den momentanen Stand des Befehlszählers zu benennen.

OPERATOR:

+	Addition
–	Subtraktion
&	logisches Und
©	logisches Oder
!	logisches XOR
★	Multiplikation
/	DIVISION
?	MOD Funktion `(a?b=a-(a/b)*b)`

Beachten Sie bitte, daß eine Hexadezimalzahl mit `'#'`, eine Binärzahl mit `'%'` beginnen und eine Zeichenkonstante in Anführungszeichen `'"'` eingeschlossen sein muß. Wenn eine Zahl (dezimal, hexa oder binär) gelesen wird, nimmt GENA3 die 16 niedrigstwertigen Bits der Zahl (d. h. MOD 65536); z. B. `70016` wird als `4480` und `#5A2C4` wird als `#A2C4` gelesen.

Es stehen zwar eine Vielzahl von Operanden zur Verfügung, es wird jedoch keiner bevorrechtigt – Ausdrücke werden strikt von links nach rechts berechnet. Die Operatoren '★', '/' und ? werden nur der Bequemlichkeit halber hinzugefügt und nicht als Teil einer voll gültigen Berechnungsroutine, die dann GENA3 erheblich aufblähen würde.

Wenn ein Ausdruck in ein Klammerpaar eingeschlossen ist, wird er als Speicheradresse interpretiert; wie z. B. in dem Befehl `LD HL,(LOC+5)`, der das Registerpaar mit dem 16-Bit-Wert der Speicherstelle 'loc+5' laden würde.

Bestimmte Z80-Befehle (`'JR'` und `'DJNZ'`)erwarten Operanden, die einen 8-Bit-Wert und keinen 16-Bit-Wert darstellen. Dies wird als relative Adressierung bezeichnet. Wenn relative Adressen angegeben wurden, dann subtrahiert GENA3 automatisch den Wert des Befehlszählers des nächsten Befehls vom Wert des Operanden im laufenden Befehl, um die relative Adresse des laufenden Befehls zu ermitteln. Der erlaubte Bereich für relative Adressen reicht vom Wert des Befehlszählers des nächsten Befehls −128 bis zum Wert des Befehlszählers des nächsten Befehls +127.

Wenn Sie dagegen einen relativen Abstand vom momentanen Stand des Befehlszählers angeben wollen, dann sollten Sie das Symbol '$' (ein reserviertes Wort) gefolgt von der gewünschten Distanz verwenden. Da dies nun relativ zum laufenden Befehlszählerstand ist, muß der Wert der Verschiebung innerhalb des Bereiches −126 bis +129 liegen.

Beispiele gültiger Ausdrücke:

```
#5000-feldname
%1001101!%1011          ergibt %1000110
#3456?#1000             ergibt #456
4 + 5 * 3 - 8           ergibt 19
$ - name + 8
2345/7 - 1              ergibt 334
"A" + 128
"y" - ";"+7
(5*name -#1000 &%1111)
17@%1000                ergibt 25
```

Beachten Sie bitte, daß zwischen Termen und Operatoren und umgekehrt Zwischenräume erlaubt sind, jedoch nicht innerhalb von Termen. Wenn das Ergebnis einer Multiplikation den Wert `32767` überschreitet, wird der Fehler `'*ERROR* 15'`, bei einer Division der Fehler `'*ERROR* 14'` ausgegeben; ansonsten wird ein Überlauf ignoriert. Alle arithmetischen Befehle benutzen das Zweierkomplement, so daß Zahlen über `32767` als negativ betrachtet werden, z. B. `60000= -5536 (60000 - 65536)`

2.6 Assembler-Anweisungen

GENA3 erkennt bestimmte Pseudo-Befehle. Diese Assembler-Anweisungen, wie sie genannt werden, haben keinen Einfluß auf den Z80 Prozessor zur Ausführungszeit, d. h. sie werden nicht in Maschinencode umgesetzt, sie weisen lediglich den Assembler während der Assemblierung darauf hin, bestimmte Aktionen durchzuführen. Diese Aktion bewirkt keine Veränderung des von GENA3 erstellten Objektcodes in irgendeiner Weise.

Pseudo-Befehle werden genauso verarbeitet wie Ausführungsbefehle, sie können einen Namen besitzen (bei **'EQU'** sogar notwendigerweise) und ein Kommentar kann folgen. Folgende Assembler-Anweisungen sind vorhanden:

ORG Ausdruck
setzt den Befehlszähler auf den Wert des 'Ausdrucks'. Wenn weder Option 2 noch Option 16 gesetzt ist und eine **ORG**-Anweisung würde das GENA3-Programm oder die Textdatei oder die Symboltabelle überschreiben, dann wird die Meldung **'Bad ORG!'** ausgegeben und die Assemblierung abgebrochen. Siehe Abschnitt 2.0 für weitere Details und den Zusammenhang zwischen Option 2 und Option 16 mit der **ORG**-Anweisung.

EQU Ausdruck
Dieser Anweisung muß ein Name vorausgehen und setzt den Wert des Namens auf den Wert des Ausdrucks. Der Ausdruck darf keinen Namen enthalten, dem nicht vorher schon ein Wert zugewiesen wurde **(*ERROR* 13)**.

DEFB Ausdruck,Ausdruck,...
Jeder Ausdruck muß einen 8-Bit-Wert annehmen; das Byte, auf das der Befehlszähler gerade verweist, wird auf den Wert des 'Ausdrucks' gesetzt und der Befehlszähler um 1 erhöht. Dies wiederholt sich für jeden Ausdruck.

DEFW Ausdruck,Ausdruck,...
Setzt das 'Wort' (2 Bytes), auf das der Befehlszähler gerade verweist, auf den Wert des 'Ausdrucks' und der Befehlszähler wird um 2 erhöht. Dies wiederholt sich für jeden Ausdruck. Zuerst wird das rechte Byte des Wortes abgespeichert, dann folgt das linke.

DEFS Ausdruck
Erhöht den Befehlszähler um den Wert des Ausdrucks, gleichbedeutend einer Reservierung eines Speicherbereichs. Der zugewiesene Bereich wird mit Nullen gelöscht.

DEFM "S"
Definiert den Inhalt eines n Bytes langen Speicherbereichs mit den dem String "S" entsprechenden ASCII-Zeichen. n ist die Länge des Strings, der theoretisch 1 – 255 Stellen lang sein kann. Tatsächlich wird die Länge des Strings begrenzt durch die Zeilenlänge, die der Editor erlaubt. Das erste Zeichen im Operandenfeld (oben das Zeichen ") wird als Begrenzer interpretiert und der String wird definiert mit den Zeichen, die zwischen den beiden Begrenzern stehen. Die Zeilenendemarkierung wird auch als abschließender Begrenzer erkannt.

ENT Ausdruck
Setzt die Start-Adresse für die Ausführung auf den Wert des Ausdrucks. Diese Anweisung wird in Verbindung mit dem Editor-Kommando **'R'** (siehe Kapitel 3) benutzt. Wenn nicht angegeben, wird kein Ersatzwert eingetragen.

2.7 Bedingte Pseudo-Befehle

Bedingte Pseudo-Befehle gestatten es dem Programmierer, bestimmte Abschnitte des Quellentextes in die Assemblierung einfließen zu lassen oder auszuschließen. Dies wird durch die Anwendung von **'IF', 'THEN'** und **'ELSE'** ermöglicht.

IF Ausdruck
Der 'Ausdruck' wird ausgewertet. Wenn das Ergebnis Null ist, werden die nachfolgenden Zeilen bis zu **'ELSE'** oder **'END'** vom Assembler ignoriert. Wenn der Wert von 'Ausdruck' ungleich Null ist, wird die Assemblierung normal fortgesetzt.

ELSE
Dieser Pseudo-Befehl dient lediglich dazu, das Ignorieren von Befehlsfolgen während der Assemblierung einzuschalten bzw. aufzuheben. Wenn vor dem **'ELSE'** die Befehle assembliert wurden, schaltet **'ELSE'** auf "ÜBERLESEN" und umgekehrt.

END
'END' schaltet lediglich das Überlesen/Ignorieren aus.
Es ist zu beachten, daß die bedingten Pseudo-Befehle nicht geschachtelt auftreten dürfen; es wird keine Prüfung auf Verschachtelung durchgeführt. Bei jedem Versuch dies doch zu tun, ist das Ergebnis nicht vorhersehbar.

2.8 Assembler – Kommandos

Assembler-Kommandos haben, wie die Assembler-Anweisungen, keinerlei Einfluß auf den Z80-Prozessor zur Ausführungszeit, da sie nicht in Maschinencode umgesetzt werden. Im Gegensatz zu Assembler-Anweisungen haben sie aber auch keinen Einfluß auf den generierten Objektcode, da Assembler-Kommandos lediglich den Ausdruck der Assemblierungsliste modifizieren.

Ein Assembler-Kommando ist eine Zeile im Quellentext, die mit einem Stern **'*'** beginnt. Der Buchstabe nach dem ersten Stern bestimmt die Funktion des Kommandos und muß groß geschrieben werden. Der Rest der Zeile kann irgendein Text sein, außer bei den Kommandos **'L', 'T'** und **'D'**, bei denen noch ein nachfolgendes **'+'** oder **'-'** erwartet wird.

Folgende Kommandos sind möglich:

***E**
Veranlaßt die Ausgabe von 3 Leerzeilen am Bildschirm, bzw. Blattvorschub, wenn die Ausgabe auf den Drucker gelegt wurde. Dies ist nützlich zum Absetzen zusammengehöriger Teile.

***Hs**
Mit diesem Kommando wird erreicht, daß der String **s** als Kopfzeile definiert und nach jedem Blattvorschub (***E**) gedruckt wird. ***H** führt automatisch ***E** durch.

***S**
veranlaßt, daß das Ausdrucken an dieser Zeile angehalten wird. Durch Drücken irgendeiner Taste wird die Auflistung fortgesetzt. Die Verwendung dieses Kommandos ist nützlich, um Adressen in der Mitte der Liste zu lesen. Es ist zu beachten, daß ***S** das Ausdrucken nicht unterbricht, wenn das Kommando ***L-** aktiv ist.

***L-**
unterdrückt die Auflistung, beginnend mit dieser Zeile.

***L+**
veranlaßt, daß wieder eine Auflistung, beginnend mit dieser Zeile, durchgeführt wird.

***D+**
veranlaßt, daß der Befehlszähler in dezimaler Form am Anfang jeder Zeile, anstatt in hexadezimaler Form ausgegeben wird.

***D-**
kehrt die Ausgabe des Befehlszählers wieder in hexadezimale Form um.

***F** Dateiname
Dieses sehr wirkungsvolle Kommando erlaubt Ihnen, Text von einem Magnetband zu assemblieren. Die Textdatei wird blockweise in einen Puffer gelesen und dann aus dem Puffer assembliert.

Dies erlaubt Ihnen, einen umfangreichen Objektcode zu generieren, ohne daß der Speicher durch den Quellentext nennenswert vermindert wird.

Der Dateiname (bis zu 8 Zeichen) der Textdatei, die Sie an diesem Punkt einfügen wollen, kann wahlweise angegeben werden und muß durch einen Zwischenraum getrennt sein. Wenn kein Dateiname angegeben wird, wird die erste Textdatei, die auf dem Band gefunden wird, eingefügt.

Jede Textdatei, die Sie auf diese Weise einfügen wollen, muß zuvor mit dem Kommando **'P'** des Editors auf das Band gesichert worden sein. Jedesmal, wenn der Assembler ein ***F**-Kommando findet, fordert er Sie auf **'Press PLAY then any key'**, d. h. Drücken Sie die Wiedergabe-Taste auf Ihrem Rekorder und anschließend irgendeine Taste auf der Eingabe-Tastatur. Dies wird sowohl im 1. als auch im 2. Durchlauf geschehen, da der eingefügte Text in jedem Durchlauf geprüft werden muß. Auf dem Band wird dann zum Einfügen nach einer Datei mit dem gewünschten Dateinamen bzw. nach der 1. Datei gesucht. Wenn eine Datei zum Einfügen gefunden wurde, der Dateiname jedoch nicht dem gewünschten Dateinamen entspricht, so wird die Meldung **'Found filename'** ausgegeben und die Suche wird fortgesetzt; ansonsten erscheint die Meldung **'Loading filename'**, die Datei wird blockweise geladen und eingefügt.

Siehe Beispiel im Anhang 3, wie dieses Kommando angewendet wird.

***T+**
Dieses Kommando veranlaßt, daß der Objectcode während der Assemblierung auf Band ausgegeben wird. Diese Ausgabe kann durch ***T-**, ORG-Anweisung oder Abschluß der Assemblierung beendet werden. Der Code, der auf diese Weise auf dem Band gesichert wurde, kann über das Debugger-Programm MONA3 zurückgeladen werden.

***T-**
schließt die Banddatei mit dem laufenden Objektcode.

Assembler-Kommandos, außer ***F**, werden erst im 2. Durchlauf ausgewertet.

Wenn die Assemblierung durch einen Pseudo-Befehl abgeschaltet wurde, sind auch alle Assembler-Kommandos wirkungslos.

Kapitel 3

Der integrierte Editor (Dateiaufbereiter)

3.1 Einführung in den Editor

Der Editor, der mit allen Möglichkeiten von GENA3 ausgeliefert wird, ist ein einfacher, zeilenorientierter Aufbereiter. Er wurde so entwickelt, daß er mit allen Z80 Betriebssystemen betrieben werden kann, während er in der Bedienung einfach ist und die Möglichkeit bietet, Programme schnell und effizient einzugeben und zu ändern. Um die Größe der Textdatei gering zu halten, werden zusammenhängende Zwischenräume durch den Editor komprimiert. Die Vorgehensweise ist dabei wie folgt:

Wenn eine Zeile über die Tastatur eingegeben wird, wird sie zeichenweise in einen Assembler-internen Puffer gestellt. Am Zeilenende (d. h. wenn Sie die **[ENTER]**-Taste drücken), wird sie vom Puffer in die Textdatei übertragen. Während dieser Übertragung werden bestimmte Zwischenräume komprimiert. Die Zeile wird, beginnend beim ersten Zeichen, auf Zwischenraum untersucht. Wenn dies gleich ein Zwischenraum ist, wird ein Tabulatorzeichen in die Textdatei gestellt und die nachfolgenden Zwischenräume überlesen; ansonsten wird zeichenweise bis zu einem Zwischenraum vom Puffer in die Textdatei übertragen. Die Verfahrensweise ist nun die gleiche, als wäre dieser Zwischenraum der erste in der Zeile. Dies wird nun wiederholt mit dem Ergebnis, daß Tabulatorzeichen eingefügt sind am Zeilenbeginn oder zwischen Namen und Befehl, und zwischen Befehl und Operanden. Wenn dabei eine Zeilenendemarkierung erkannt wird, wird natürlich die Übertragung beendet und die Steuerung wird wieder dem Editor übergeben.

Diese Komprimierungsart bietet somit dem Benutzer die Möglichkeit, durch einfache Eingabe der **[TAB]**-Taste eine spaltengerechte Textdatei zu erstellen und dabei gleichzeitig noch Speicherplatz zu sparen.

Es ist zu beachten, daß im Kommentar keine Zwischenräume komprimiert werden und im Namen, im Befehl und den Operanden kein Zwischenraum vorhanden sein darf.

Wenn GENA3 gestartet wird, wird automatisch der Editor geladen und meldet sich mit:

```
Hisoft GENA3 Assembler
Copyright Hisoft 1984
All rights reserved
```

gefolgt von dem Zeichen ' > ', nach dessen Ausgabe der Editor immer empfangsbereit ist (Kommando-Modus).

Sie können nun eine Kommandozeile nach dem folgenden allgemeingültigen Format eingeben:

C N1,N2,S1,S2 [ENTER]

C entspricht einem Kommando, das ausgeführt werden soll (siehe nachfolgenden Abschnitt 3.2)

N1 entspricht einer Zahl von 1 – 32767

N2 entspricht einer Zahl von 1 – 32767

S1 entspricht einer Zeichenkette mit maximal 20 Stellen

S2 entspricht einer Zeichenkette mit maximal 20 Stellen

Das Komma trennt hier die einzelnen Argumente (dies kann jedoch geändert werden, siehe' **S** 'Kommando); Zwischenräume werden ignoriert, ausgenommen bei den Zeichenketten. Keines der Argumente ist obligatorisch, obwohl manche Kommandos ohne Angabe von **N1** und **N2** nicht arbeiten (z. B. das' **D** 'elete-Kommando). Der Editor merkt sich die früher eingegebenen Zahlen und Zeichenketten und setzt diese vorausgehenden Werte, wo sie notwendig sind, ein, wenn sie in einem Kommando fehlen. Zu Beginn ist **N1** und **N2** auf 10 gesetzt und die Zeichenkette ist leer. Wenn Sie eine ungültige Kommandozeile eingeben, wie z. B. **F-1,100,HALLO** dann wird die Zeile ignoriert und es erscheint die Meldung' **Pardon?** '; dann müssen Sie erneut eine Kommandozeile eingeben, z. B. **F1,100,HALLO**. Die oben genannte Fehlermeldung wird auch ausgegeben, wenn die Zeichenkette für **S2** länger als 20 Stellen ist; die Zeichenkette **S1** wird gegebenenfalls nach der 20. Stelle abgeschnitten.

Kommandos können in Groß- oder Kleinbuchstaben eingegeben werden.

Beim Erstellen einer Kommandozeile können alle benötigten Kontroll-Funktionen, die nachfolgend beschrieben werden, angewendet werden; z. B. **[CTRL/X]** um eine Zeile bis zum Beginn zu löschen, **[TAB]** um den Cursor auf die nächste Tabulatorstelle zu setzen usw.

Der folgende Abschnitt gibt detaillierte Auskunft über die verschiedenen Kommandos des Editors. Beachten Sie dabei, daß, wenn ein Argument in den Zeichen '< >' eingeschlossen ist, dieses Argument immer angegeben werden muß, um das entsprechende Kommando auch durchführen zu können.

3.2 Die Editor-Kommandos

3.2.1 Text-Eingabe

Text kann in die Textdatei durch Angabe einer Zeilennummer, eines Zwischenraums mit dem nachfolgenden Text oder durch die Benutzung des I-Kommandos eingegeben werden; wenn Sie nur eine Zeilennummer und anschließend die Taste **[ENTER]** eingeben (d. h. ohne Text), ist zu beachten, daß die entsprechende Zeile, falls vorhanden, in der Textdatei gelöscht wird. Immer, wenn Sie einen Text eingeben, sollten Sie die Kontroll-Funktionen **[CTRL/X]** (Löschen der Zeile bis zum Beginn), **[TAB]** Sprung zur nächsten Tabulatorposition) und **[CTRL/X]** (Rückkehr in den Kommando-Modus) anwenden.

Durch die **DELETE**-Taste wird an den Zeilenanfang zurückgekehrt und dabei die Zeile gelöscht. Der Text wird in einen GENA3-internen Puffer gestellt. Wenn dieser Puffer vollständig gefüllt ist, dürfen Sie keinen weiteren Text mehr eingeben, sondern müssen **[DEL]** oder **[CTRL/X]** drücken, um Platz im Puffer frei zu machen.

Wenn der Editor während der Texteingabe feststellt, daß sich das Ende der Textdatei dem Ende des RAM-Speichers nähert, gibt er die Meldung `'Bad Memory'` aus. Dies bedeutet, daß kein Text mehr eingegeben werden kann und daß die zu bearbeitende Textdatei auf Band gesichert werden muß, um sie später wieder zu laden.

Kommando: `I` n,m

Dieses Kommando schafft eine Lücke für den automatischen Eingabe-Modus. Sie erhalten Zeilennummern vorgegeben, die mit n beginnen, in der Schrittweite von m. Sie geben den gewünschten Text in die zur Verfügung gestellte Zeile ein, gegebenenfalls benützen Sie die verschiedenen Kontroll-Codes und beenden die Zeile mit der Taste **[ENTER]**. Um diesen Modus zu verlassen drücken Sie **[CTRL/X]**.

Wenn Sie eine Zeile mit einer bereits vorhandenen Zeilennummer eingeben, ersetzt diese nach Drücken der **[ENTER]**-Taste die ursprüngliche Zeile. Wenn das automatische Hochzählen der Zeilennummer die Nummer 32767 überschreitet, wird der Einfüge-Modus automatisch verlassen.

Wenn Sie während der Texteingabe das Ende der Bildschirmzeile erreichen und Sie noch keine 64 Zeichen (Puffergröße) eingegeben haben, wird der Bildschirminhalt hochgeschoben und Sie können auf der nächsten Zeile weiterschreiben.

3.2.2 Text-Ausgabe

Kommando: `L` n,m

Hiermit wird der Text am Ausgabegerät von Zeile n bis Zeile m aufgelistet. Der Ersatzwert für n ist immer 1 und der Ersatzwert von m ist immer 32767, d. h. die Ersatzwerte werden nicht von Argumenten vorausgehender Kommandos übernommen. Die Spaltenausrichtung erfolgt automatisch mit einer klaren Aufteilung der einzelnen Felder. Die Anzahl der Bildschirmzeilen, die jeweils angezeigt werden, ist immer 24; die Auflistung wird dann angehalten (wenn noch nicht die Zeilennummer m erreicht ist). Um wieder in den Editor zurückzukehren, müssen Sie **[CTRL/C]** drücken; durch Eingabe jeder anderen Taste wird weitergeblättert.

3.2.3 Text-Aufbereitung

Wenn einmal ein Text erstellt wurde, wird es immer wieder notwendig sein, einige Zeilen zu ändern. Es gibt eine Vielzahl von Kommandos, mit denen Zeilen verbessert, gelöscht, an eine andere Stelle übertragen oder neu durchnumeriert werden können. Zusätzlich zu den zeilenorientierten Kommandos, die nachfolgend beschrieben werden, gibt es noch eine einfache Form der Bildschirmaufbereitung, die vor allem dann nützlich ist, wenn Sie bemerken, daß nur ein Name oder Befehl falsch geschrieben ist. Sie setzen dann den Cursor mit Hilfe der Cursor-Steuerungstasten an die gewünschte Zeile und kopieren den Text durch Drücken der **[COPY]**-Taste in den Puffer des Editors (einschließlich Zeilennummer) bis zu der Stelle, an der Sie ändern wollen. Nun tippen Sie die Korrektur über die Tastatur ein und kopieren wieder den Rest der Zeile. Mit der **[ENTER]**-Taste wird die Eingabe beendet. Es ist klar, daß diese Möglichkeit sorgfältig angewendet werden muß, besonders im Hinblick auf die **[DEL]**-Taste, die die Zeichen im internen Puffer löscht, welche jedoch nicht unbedingt die Zeichen hinter dem Cursor sein müssen.

Benutzen Sie die Cursor-Tasten nicht im **I**- und **E**-Kommando.

7 Aufbereitungs-Kommandos:

Kommando: **D** <n,m>

Alle Zeilen von n bis einschließlich m werden in der Textdatei gelöscht. Wenn m kleiner als n ist oder weniger als 2 Argumente angegeben werden, erfolgt keine Durchführung, um Flüchtigkeitsfehlern vorzubeugen. Eine einzelne Zeile kann gelöscht werden, wenn man m = n setzt; dies kann aber auch erreicht werden, indem man nur die Zeilennummer und **[ENTER]** eingibt.

Kommando: **M** n,m

Dieses Kommando bewirkt, daß der Text mit der Zeilennummer n in die Zeile mit der Nummer m übertragen wird. Dabei wird die Zeile m gegebenenfalls überschrieben, die Zeile n wird gelöscht. Wenn die Zeilennummer n nicht vorhanden ist, wird das Kommando ohne Aktion beendet.

Kommando: **N** <n,m>

Mit diesem Kommando können die Zeilennummern neu vergeben werden. Dabei wird, beginnend bei der ersten Zeile mit Nummer n in einer Schrittweite von m durchnumeriert. Falls durch die neue Numerierung eine Zeilennummer größer als 32767 gebildet würde, bleiben die ursprünglichen Zeilennummern erhalten.

Kommando: **F** n,m,f,s

Der Text im Zeilenbereich x ($n < x < m$) wird durchsucht, ob in ihm die Zeichenkette f enthalten ist; gegebenenfalls wird dann die Zeile ausgegeben und in den Editier-Modus (siehe nachfolgendes Kommando 'E') verzweigt. Sie können dann die innerhalb des Editier-Modus zugelassenen Kommandos anwenden, um weitere nachfolgende Zeilen zu finden, in denen die Zeichenkette f im genannten Bereich auftritt, oder sie durch die Zeichenkette s in der laufenden Zeile zu ersetzen und weiterzusuchen. Für weitere Einzelheiten siehe unter Kommando 'E'.

Es ist zu beachten, daß wenn der Zeilenbereich und/oder die beiden Zeichenketten bereits in einem vorausgehenden Kommando angegeben waren, zur Suche lediglich noch **"F"** einzugeben ist.

Kommando: **E** n

Dieses Kommando bearbeitet die Zeile mit der Nummer n.

Wenn n nicht vorhanden ist, wird das Kommando abgebrochen, ansonsten wird die Zeile in den Puffer kopiert und mit Zeilennummer am Bildschirm ausgegeben. Die Zeilennummer wird darunter ein zweites Mal ausgegeben und in den Editier-Modus verzweigt. Alle nachfolgenden Bearbeitungen finden im Puffer statt und nicht in der Textdatei selbst, so daß man jederzeit wieder auf die Originalzeile zurückgreifen kann.

In diesem Modus wird ein gedanklicher Zeiger, beginnend beim 1. Zeichen, über die Zeile bewegt und verschiedene Unterkommandos unterstützen die Aufbereitung.

Folgende Unterkommandos sind möglich:

'_' (Zwischenraum) erhöht den Zeigerwert um 1, d. h. der Zeiger steht auf dem nächsten Zeichen.

[DEL] vermindert den Zeigerwert um 1, d. h. der Zeiger steht auf dem vorhergehenden Zeichen

[ENTER] beendet die Aufbereitung dieser Zeile und schreibt sie mit allen Veränderungen in die Textdatei zurück

Q verzichtet auf die Bearbeitung dieser Zeile, d. h. das Kommando wird (unter Verzicht auf alle evtl. durchgeführten Änderungen) beendet und die Zeile bleibt im gleichen Zustand, wie vor Ausführung des **'E'**-Kommandos

R lädt erneut die Zeile von der Textdatei in den Puffer, d. h. alle vorangegangenen Änderungen wurden rückgängig gemacht und die Zeile steht im Originalzustand wieder zur Verfügung

L gibt den Rest der Zeile aus, d. h. alle Zeichen nach der momentanen Position des Zeigers. Sie bleiben im Editier-Modus, der Zeiger wird jedoch wieder auf den Zeilenanfang gestellt

K Lösche das Zeichen, auf dem der Zeiger gerade steht

Z Lösche alle Zeichen, beginnend mit der Stelle, auf der der Zeiger steht, bis zum Zeilenende

F Suche das nächste Vorkommen der Zeichenkette f, die vorausgehend bereits angegeben wurde (siehe 'F'-Kommando). Dieses Unterkommando wird (unter Berücksichtigung der Änderungen) automatisch beendet, wenn in derselben Zeile diese Zeichenkette nicht mehr vorkommt. Wenn innerhalb des vorgegebenen Zeilenbereichs in einer der nachfolgenden Zeilen die gesuchte Zeichenkette enthalten ist, wird mit der gefundenen Zeile der Editier-Modus wieder aufgerufen. Der (gedachte) Zeiger steht immer auf der ersten Stelle der gefundenen Zeichenkette.

S Ersetze die gefundene Zeichenkette f mit der vorausgehend definierten Zeichenkette s und führe dann das Unterkommando 'F' aus, d. h. suche das nächste Auftreten von f. So wird, zusammen mit dem oben genannten Unterkommando 'F', schrittweise die Textdatei durchsucht, mit der Möglichkeit, die gefundene Zeichenkette wahlweise durch die vorgegebene zu ersetzen (siehe als Beispiel Abschnitt 3.3)

I Einfügen von Zeichen an der Stelle, auf die der Zeiger verweist. Sie bleiben in diesem Unterkommando, bis Sie die **[ENTER]**-Taste drücken. Danach befinden Sie sich im Editier-Modus und der Zeiger steht auf dem ersten Zeichen nach der Einfügung. Wenn Sie während des Unterkommandos die Taste **[DEL]** drücken, wird das Zeichen links vom Zeiger aus dem Puffer gelöscht, während bei Eingabe der Taste **[TAB]** der Zeiger (unter Einfügen von Zwischenräumen) auf den nächsten Spaltenbeginn verweist. Während Sie „EINFÜGEN" ausgewählt haben, wird der Cursor als „★" dargestellt.

X stellt den Zeiger auf das Zeilenende und verzweigt automatisch in das Unterkommando 'I' (wie oben beschrieben).

C Aufruf der Änderungsfunktion innerhalb des Editier-Modus. Sie bleiben bis zur Eingabe der **[ENTER]**-Taste in dieser Funktion. Sie können das Zeichen, auf das der Zeiger verweist, überschreiben; dabei wird der Zeiger um 1 erhöht. Bei Erreichen des Zeilenendes wird die Änderungsfunktion abgeschlossen und Sie gelangen wieder in den Editiermodus. Der Zeiger verweist in jedem Fall auf die nächste Stelle nach der letzten Änderung. Die Taste **[DEL]** bewirkt während dieser Funktion lediglich ein Vermindern des Zeigerwertes um 1, d. h. der Zeiger bewegt sich nach links, während die Taste **[TAB]** ignoriert wird.

Während Sie sich im „ÄNDERN" befinden, wir der Cursor als '+' dargestellt.

3.2.4 Magnetband-Kommandos

Mit den Kommandos **'P', 'V'** und **'G'** kann eine Textdatei auf Band gesichert oder vom Band geladen werden.

Zum Sichern von Objektcode wird das Kommando 'O' verwendet.

Kommando: **P** n,m,s

Der Zeilenbereich mit der Untergrenze n und der Obergrenze m wird unter dem Dateinamen s auf Band gesichert; wenn angegeben, muß n kleiner m sein. Es ist zu beachten, daß diese Argumente durch ein vorheriges Kommando gesetzt sein könnten. Vor Eingabe des Kommandos ist sicherzustellen, daß der Datacorder eingeschaltet und die Taste **[REC/PLAY]** gedrückt ist.

Kommando: G„s

Auf dem Magnetband wird nach einer Datei mit dem Namen s gesucht; Wenn s ein leerer String ist, wird die erste Textdatei geladen. Wenn die Datei gefunden wurde, so wird sie am Ende des laufenden Textes angefügt.

Nach Eingabe des Kommandos 'G' muß die **[PLAY]**-Taste am Rekorder gedrückt werden. Es beginnt jetzt die Suche nach einer Datei mit dem angegebenen Namen, bzw. nach der ersten Textdatei, wenn kein Name angegeben war (Leerstring). Wenn die Datei gefunden wurde, wird die Meldung 'LOADING FILENAME' ausgegeben, ansonsten wird 'FOUND FILENAME' angezeigt und die Suche fortgesetzt.

Es ist zu beachten, daß wenn bereits eine Textdatei im Speicher steht, die Datei vom Band geladen und am Ende der bestehenden Datei angefügt wird und die gesamte Datei neu durchnumeriert, beginnend mit 1 und einer Schrittweite von 1.

Kommando: V„s

Dieses Kommando prüft eine Datei im Speicher mit einer Datei mit dem Namen s auf Band auf Übereinstimmung.

Abhängig vom Erfolg wird das Ergebnis 'VERIFIED'(Übereinstimmung) bzw. 'FAILED'(Fehler) ausgegeben.

Kommando: O„s

Dieses Kommando gibt den Objektcode der letzten Assemblierung unter dem Namen s auf Band aus.

Um den Objektcode zurückzuladen ist MONA3 oder BASIC aufzurufen.

Kommando: Tn

Verändert die Geschwindigkeit für alle nachfolgenden Magnetbandaufzeichnungen. 'T'gefolgt von **[ENTER]** wählt eine langsame Geschwindigkeit, wogegen 'T'gefolgt von einer Zahl größer 0 und **[ENTER]** eine höhere Geschwindigkeit auswählt.

3.2.5 Assemblieren und Ablauf über den Editor

Kommando: A

Dieses Kommando startet eine Assemblierung mit der ersten Zeile der Textdatei.

Für weitere Details siehe Kapitel 2.

Kommando: R

Wenn der Quellentext ohne Fehler umgewandelt und mit der Assembler-Anweisung ENT eine Startadresse benannt wurde, kann mit dem Kommando R die Ausführung des Objektprogramms eingeleitet werden. Das Objektprogramm kann über die Assembler-Instruktion RET (#C9) so oft zum Editor zurückkehren, solange der Stack bei Programmende genauso positioniert ist, wie er zu Beginn des Programms war.

Es ist zu beachten, daß die Angabe der Option 16 bei der Assemblierung die ENT-Anweisung aufhebt.

3.2.6 Andere Kommandos

Kommando: H

Zeigt einen Bildschirm, auf dem alle in GENA3 möglichen Kommandos in Großbuchstaben dargestellt sind.

Kommando: B

Mit diesem Kommando wird ins Betriebssystem zurückgekehrt. Um wieder in den Assembler zu gelangen, ist ein Kaltstart (Originalladeadresse + 2) oder ein Warmstart (Originalladeadresse + 4) durchzuführen.

Kommando: S„d

Mit diesem Kommando läßt sich das Begrenzungszeichen, welches zum Trennen der einzelnen Argumente in Kommandozeilen verwendet wird, austauschen. Bei Aufruf des Editors wird das Komma ',' als Begrenzer verwendet. Mit dem Kommando 'S' und der Angabe eines Strings d kann dies geändert werden (wenn d mehr als ein Zeichen enthält, wird auf das erste Zeichen Bezug genommen).

Es ist zu beachten, daß, wenn einmal ein neuer Begrenzer benannt wurde, dieser auch verwendet werden muß (auch im 'S'-Kommando), bis ein neuer definiert wird. Zwischenraum kann nicht als Begrenzer angegeben werden.

Kommando: C

Das Kommando 'C' gibt die derzeitigen Inhalte von N1, N2, S1 und S2 aus; dies sind die Standard-Werte für die zwei Zeilennummern und die Zeichenketten.

Die Anwendung dieses Kommandos ist immer dann sinnvoll, wenn in einem weiteren Kommando mit Ersatzwerten operiert werden soll, um den Inhalt obiger Elemente zu überprüfen.

Kommando: Zn,m

Mit dem 'Z'-Kommando kann der Bereich von Zeile n bis einschließlich Zeile m ausgedruckt werden. Wenn beide Argumente n und m fehlen, wird die gesamte Textdatei ausgedruckt.

Wenn kein Drucker angeschlossen ist, wird 'No Printer!' angezeigt und keine Aktion vorgenommen. Sie können das Ausdrucken unterbrechen, wenn Sie am Zeilenende irgendeine Taste drücken. Wenn Sie dann **[CTRL/C]** drücken, wird das Ausdrucken abgebrochen; bei Eingabe jeder anderen Taste wird das Ausdrucken fortgesetzt.

Kommando: X

Mit dem Kommando 'X' wird die Start- und Endadresse der Textdatei auf Dezimalbasis angezeigt. Dies ist nützlich, wenn der Text in BASIC gesichert werden soll, oder wenn angezeigt werden soll, wieviel freier Speicher nach der Textdatei noch zur Verfügung steht.

'X' wird auch in Verbindung mit MONA3 verwendet, wenn über den Disassembler (MONA3) eine Textdatei zur Eingabe in GENA3 ersetzt werden soll. Um die Textdatei dem Assembler bekannt zu machen, müssen wir die Textdatei nach **'TEXTSTART'** von GENA3 übertragen oder die Textdatei direkt ab dieser Adresse generieren. Wenn nun 'X' benutzt wird, verweist die erste ausgegebene Adresse auf den Anfang der Textdatei, die auch leer sein kann. Dies ist jedoch die Adresse, an der die von MONA3 erstellte Textdatei beginnen muß.

Teilen Sie nun dem Disassembler diese Adresse mit und merken Sie sich die **'TEXTEND**-ADRESSE', die der Disassembler ausgibt. Diese Adresse muß GENA3's **'TEXTEND'** zugeordnet werden. Dies geschieht in der Weise, daß **'TEXTEND'** an der Adresse 7 relativ zur Startadresse von GENA3 abgespeichert wird (wenn z. B. GENA3 ab 1000 geladen wird, würde die **'TEXTEND'**-Adresse in 1007 und 1008 (das niedrigste Byte zuerst) mit dem Befehle **'POKE'** abgespeichert werden.

GENA3 kann dann über einen Warmstart (in diesem Fall ab 1004) zur Aufbereitung und Assemblierung der Textdatei aufgerufen werden.

Kommando: **W**

Dieses Kommando schaltet wechselweise (Flip-Flop) zwischen 40- oder 80-Zeichen-Modus um.

Kommando: **J**

Verzweigt in den Debugger MONA3, wenn er geladen und mindestens einmal benutzt wurde. Ansonsten wird das Kommando abgebrochen.

3.3 Beispiel zum Benutzen des Editors

Nehmen wir an, Sie haben folgendes Programm (mit Kommando **I10,10**) eingetippt

```
10 *h 16 BIT Zufallszahlen
20
30 ;INPUT: HL enthaelt vorhergehende Zufallszahl oder
   Ursprung
40 ;OUTPUT:HL enthaelt  nene Zufallszahl
50
60 Random:  PUSH AF ;Register sichern
70          PUSH BC
80          PUSH HL
90          ADD HL,HL ;* 2
100         ADD HL,HL ;* 4
110         ADD HL,HL ;* 8
120         ADD HL,HL ;*16
130         ADD Hl,HL ;*32
140         ADD HL,HL ;*64
150         PIP BC ;alte Zufallszahl
160         ADD Hl,DE
170         LD DE,41
180         ADD HL,DE
190         POP BC ;Register zurueckspeichern
200         POP AF
210         REY
```

Dieses Programm weist eine Reihe von Fehlern auf, die nachfolgend aufgeführt sind:

Zeile 10 im Assembler-Kommando ★**H** wurde ein kleines h verwendet
Zeile 40 'nene' anstatt 'neue'
Zeile 150 'PIP' anstatt 'POP'
Zeile 160 Es soll ein Kommentar eingefügt werden (kein Formfehler)
Zeile 210 'REY' sollte 'RET' heißen

Außerdem sollten zwischen den Zeilen 140 und 150 zwei zusätzliche Zeilen **ADD HL,HL** eingefügt werden und das Registerpaar **DE** in den Zeilen 160 bis 180 sollte in das Registerpaar **BC** geändert werden. Um all diese Berichtigungen vorzunehmen, verfahren wir wie folgt:

E10 [ENTER] dann _ (leer) **C** (Berichtigungsmodus) **H [ENTER] [ENTER]**
F40,40,nene,neue [ENTER] dann das Unterkommando 'S'
I142,2 [ENTER]
142 ADD HL,HL ;★128
144 ADD HL,HL ;★256
[CTRL/C]
F150,150,PIP,POP[ENTER] dann das Unterkommando 'S'
E 160[ENTER] dann **X__;★257 + 41[ENTER] [ENTER]**
F160,180,DE,BC[ENTER] wiederholte Anwendung des Unterkommandos 'S'
E210[ENTER] _____ (10 Leerstellen) **C** (Änderungsmodus) **T [ENTER] [ENTER]**
N10,10[ENTER] zum neuen Durchnumerieren

Es wird Ihnen dringend empfohlen, das obige Beispiel unter Benutzung des Editors durchzuarbeiten.

Anhang 1

Fehler-Nummern und ihre Bedeutung

*ERROR*1	In dieser Zeile tritt ein Fehler im Zusammenhang auf
*ERROR*2	Der Befehl wird nicht anerkannt
*ERROR*3	Die Anweisung ist falsch zusammengesetzt
*ERROR*4	Ein Name wurde mehr als einmal definiert
*ERROR*5	Diese Zeile enthält ein ungültiges Zeichen, d. h. ein Zeichen ist in einem bestimmten Zusammenhang ungültig
*ERROR*6	Einer der Operanden in dieser Zeile ist ungültig
*ERROR*7	Ein Name in dieser Zeile ist ein reserviertes Wort
*ERROR*8	Register passen nicht zusammen
*ERROR*9	Zu viele Register in dieser Zeile
*ERROR*10	Ein Ausdruck, der einen Wert bis zu 8 Bits annehmen sollte, überschreitet 8 Bits
*ERROR*11	Die Befehle JP(IX ± n) und JP(IY ± n) sind ungültig
*ERROR*12	Fehler im Aufbau einer Assembler-Steueranweisung
*ERROR*13	Ungültiger Vorwärtsverweis, d. h. ein EQU wurde auf einen Namen bezogen, der noch nicht definiert wurde
*ERROR*14	Division durch Null
*ERROR*15	Überlauf bei einer Multiplikation
Bad ORG!	ORG wurde einer Adresse zugewiesen, die GENA, die Textdatei oder die Symboltabelle zerstören würde. Die Steuerung geht an den Editor.
Out of Table space!	Wenn während des 1. Durchlaufs ungenügend Speicherplatz für die Symboltabelle zugelassen wurde, erscheint diese Meldung und die Steuerung geht sofort an den Editor.
Bad Memory!	Diese Meldung erscheint, wenn kein Platz mehr für weitere Texteingaben vorhanden ist, d. h. das Textende ist nahe an der RAM-Grenze. Sie müssen die laufende Textdatei ganz oder teilweise auf Band sichern.

Anhang 2

Reservierte Wörter, Befehle usw.

Es folgt eine Aufstellung von Wörtern, die für GENA3 reserviert sind. Diese Wörter dürfen nicht als Namen, wohl aber als Teile von Namen verwendet werden. Beachten Sie, daß alle reservierten Wörter sich aus Großbuchstaben zusammensetzen.

```
A     B     C     D     E     H     L     I     R     $
AF          AF          BC          DE          HL          IX
IY          SP          NC          Z           NZ          M
P           PE          PO
```

Es folgt nun eine Aufstellung der gültigen Z80-Befehle, Assembler-Steuerungsanweisungen und Assembler-Kommandos. Beachten Sie, daß alles in Großbuchstaben eingegeben werden muß.

```
ADC   ADD   AND   BIT   CALL  CCF   CP    CPD   CPDR
CPI   CPIR  CPL   DAA   DEC   DI    DJNZ  EI    EX
EXX   HALT  IM    IN    INC   IND   INDR  INI   INIR
JP    JR    LD    LDD   LDDR  LDI   LDIR  NEG   NOP
OR    OTDR  OTIR  OUT   OUTD  OUTI  POP   PUSH  RES
RET   RETI  RETN  RL    RLA   RLC   RLCA  RLD   RR
RRA   RRC   RRCA  RRD   RST   SBC   SCF   SET   SLA
SRA   SRL   SUB   XOR

DEFB  DEFM  DEFS  DEFW  ELSE  END   ENT   EQU   IF
ORG
*D    *E    *H    *L    *S    *C    *F    *T
```

Anhang 3

Übungsbeispiel

Es folgt ein Beispiel, wie die Arbeit mit GENA3 typisch abläuft. Wenn Sie ein Neuling in der Welt der Assembler-Programmierung sind oder wenn Sie sich einfach ein bißchen unsicher fühlen, wie der Editor/Assembler anzuwenden ist, dann legen wir Ihnen nahe, dieses Beispiel sorgfältig durchzuarbeiten.

Beachten Sie bitte, daß jede Zeile in der üblichen Weise durch Drücken der **[ENTER]**-Taste abgeschlossen wird.

Problemstellung:

Es soll eine schnelle Multiplikationsroutine geschrieben und getestet werden. Die Textdatei soll dann mit dem Editor-Kommando 'P' auf Band gesichert werden, so daß Sie leicht in zukünftige Programme eingefügt (included) werden kann.

Vorgehensweise:

1. Schreiben der Multiplikationsroutine als Unterprogramm und sichern auf Band mit dem Editorkommando 'P', so daß es während der Arbeit leicht wieder zurückgespeichert werden kann, falls Fehler auftreten.
2. Austesten der Multiplikationsroutine
3. Sichern der verbesserten Routine auf Band mit dem Editor-Kommando 'P', so daß sie in andere Programme eingefügt werden kann.

Schritt 1 – Schreiben der Multiplikationsroutine

Zur Texteingabe benutzen wir das Editor-Kommando 'I'. Sie können dabei die **TAB**-Taste verwenden, obwohl dies nicht notwendig ist, außer Sie wünschen eine spaltengerechte Auflistung.

Die in unserem Assembler-Beispiel generierten Adressen müssen nicht unbedingt mit denen auf Ihrer Maschine übereinstimmen, sie beziehen sich lediglich auf dieses Beispiel.

```
> I10,10 [ENTER]
10 ;eine schnelle Multiplikationsroutine
20 ;es wird HL mit DE multipliziert
30 ;das Ergebnis wird in HL abgestellt
40 ;das Kennzeichen C wird gesetzt,
50 ;wenn ein Ueberlauf erfolgt
60
70 ORG #7F00
80
90 Mult: OR A
```

```
100 SBC HL,DE;HL>DE?
110 ADD HL,DE
120 JR NC,Mu1;ja
130 EX DE,HL
140 Mu1:OR D
150 SCF ;Ueberlauf wenn
160 RET NZ;DE>255
170 OR E;mal 0?
180 LD E,D
190 JR NZ,MU4;nein
200 EX DE,HL;0
210 RET
220
230 ;Hauptroutine
240
250 Mu2: EX DE,HL
260 ADD HL,DE
270 EX DE,HL
280 Mu3: ADD HL,HL
290 RET C;Ueberlauf
300 Mu4:RRA
310 JR NC,Mu3
320 OR A
330 JR NZ,Mu2
340 ADD HL, DE
350 RET
360 [CTRL/C]
>P10,350,Mult[ENTER]
```

Die oben stehenden Eingaben erstellen den Text der Routine und sichern ihn auf Band.

Denken Sie daran, daß Ihr Datacorder aufnahmebereit ist, bevor Sie das 'P'-Kommando eingeben.

Schritt 2 – Austesten der Routine

Als erstes schauen wir, ob der Text korrekt assembliert wird. Wir benutzen dazu die Option 6, so daß keine Liste erstellt und kein Objektcode generiert wird.

```
>A
Table size:[ENTER] (die Größe der Symboltabelle wird von GENA3 bestimmt)
Options: 6 [ENTER]

Hisoft GENA3 Assembler. Page 1.
Pass 1 errors: 00
Pass 2 errors: 00
*WARNING* MU4 absent

Table used: 74 from 156
>
```

Wir sehen an dieser Assemblierung, daß wir in Zeile 190 **Mu4** anstatt **MU4**, den Namen, zu dem wir verzweigen wollten, eingegeben haben. So verbessern wir Zeile 190:

```
>F190,190,MU4,Mu4 [ENTER]
   190        JR     NZ,  nun das Unterkommando 'S'
>
```

Nun Assemblieren wir erneut und stellen fest, daß kein Fehler mehr gefunden wurde. Nun müssen wir noch einen Programmteil schreiben, um die Routine testen zu können:

```
> N300,10    Neu durchnumerieren, so daß wir mehr Text schreiben können
>I10,10
10 ;einige Befehle, um die
20 ;Multiplikationsroutine zu testen
30
40 LD HL, 50
50 LD DE, 20
60 CALL Mult ;Multiplizieren
70 LD A,H ;abholen des Ergebnisses
80 CALL Aout
90 LD A,L
100 CALL Aout
110 RET ;Ruecksprung in den Editor
120
130 ;Routine zum Bereitstellen von A in hex
140
150 Aout: PUSH AF
160 RRCA
170 RRCA
180 RRCA
190 RECA
200 CALL Nibble
210 POP AF
220 Nibble:AND %1111
230 ADD A,#90
240 DAA
250 ADC A,#40
260 DAA
270 ;Eingabe-Abruf
280 CALL #BB5A
290 RET
300 [CTRL]C
>
```

Nun assemblieren wir die Testroutine zusammen mit der Multiplikationsroutine.

```
>A
Table size: [ENTER]
Options: 6 [ENTER]
Hisoft GENA3 Assembler. Page 1.
7EAC 190 RECA
*ERROR* 02
hit any key to continue
Pass 1 errors: 01
Table used: 88 from 201
>
```

Wir haben einen Fehler in unserer Routine; **RECA** in Zeile 190 müßte heißen **RRCA**; so geben wir folgendes ein:

```
>E190
190RECA
190 _ _ _ _ _ _ _ _ _ (9 Leerstellen)C(Ändern) R [ENTER] [ENTER]
>
```

Wir assemblieren erneut mit der Option 4 (keine Liste), was ohne Fehler ausgeführt werden müßte. In diesem Fall sind wir jetzt in der Lage, unsere Multiplikationsroutine zu testen. Wir müssen jedoch dem Editor mitteilen, ab wo er den Code durchführen kann. Dies geschieht mit der Steuerungsanweisung ENT:

```
>35 ENT $ [ENTER]
```

Nun assemblieren wir erneut und die Assemblierung müßte mit der Meldung

```
Table used: 88 from 202
Executes: 9847
>
```

oder entsprechend beendet werden. Nun können wir unseren Code mit dem Editor-Kommando 'R' ausführen. Wir erwarten bei der Multiplikation 50 mal 20 als Ergebnis 1000, das in hexa#3E8 lautet.

```
>R [ENTER] 0032>
```

Es funktioniert nicht! Warum?

Lassen Sie sich die Zeilen 380 – 500 anzeigen (L380,500). Auf Zeile 430 finden Sie dann den Befehl OR D, dem dann der Befehl RET NZ folgt.

Was passiert, ist ein logisches Oder (OR) zwischen dem Register D und dem Akkumulator A, mit dem Setzen des Fehlerkennzeichens C, wenn das Ergebnis ungleich null ist. Der Sinn ist eine Prüfung auf DE<256, so daß bei der Multiplikation kein Überlauf entstehen kann. Dies geschieht durch Prüfung von D auf Null. Aber das OR läuft in diesem Fall nur richtig ab, wenn der Akkumulator Null enthält, was keinesfalls garantiert ist. Wir müssen sicher sein, daß A vor dem OR D Null enthält, ansonsten ist das Ergebnis nicht vorhersehbar. Bei Prüfung des Programms erkennen wir, daß das OR A in Zeile 380 auf XOR A geändert werden könnte, so daß die Flags für den Befehl SBC HL,DE gesetzt sind und A auf Null gesetzt wird.

So tippen wir:

```
>E380 [ENTER]
380 Mult: OR A
380 _ _ _ _ _ _ _ _ (8 Leerstellen)I(Einfügemodus)X[ENTER] [ENTER]
>
```

Nun assemblieren wir wieder (Option 4) und führen den Code mit 'R' aus. Das Ergebnis müßte nun korrekt erscheinen – #3E8.

Wir können die Routine noch weiter prüfen, indem wir die Zeilen 40 und 50 so ändern, daß unterschiedliche Zahlen multipliziert werden, dann assemblieren und ausführen. Sie werden feststellen, daß die Routine einewandfrei arbeitet.

Nun haben wir eine korrekte Routine, die wir auf Band sichern können.

```
>P300,999Mult [ENTER]
```

Denken Sie daran, daß der Rekorder aufzeichnungsbereit sein muß, bevor Sie **[ENTER]** drücken. Wenn die Routine in der Form einmal gesichert wurde, kann sie jederzeit in ein Programm eingefügt werden.

Dies geschieht wie folgt:

```
500 RET
510
520 ;fuege die Routine hier ein
530
540 * F Mult
550
560 ;die naechste Routine
```

Wenn der obige Text assembliert wird, wird der Assembler Sie auffordern'Press PLAY...'(Abspieltaste drücken), wenn er auf Zeile 540 stößt. Dies geschieht im 1. und 2. Durchlauf. Deshalb muß die Speicherung von Mult für beide Fälle jeweils bereit sein. Dies bedeutet im Normalfall ein Zurückspielen des Bandes nach dem 1. Durchlauf. Sie können jedoch die Routine Mult zweimal hintereinander auf Band sichern und für den 1. Durchlauf die eine Sicherung und für den 2. Durchlauf die andere verwenden.

Studieren Sie das obige Beispiel sorgfältig und probieren Sie es selbst aus.

Kapitel 1

Der DISASSEMBLER/ DEBUGGER – MONA3

Disassembler und Monitor

MONA3 liegt in einer verschiebbaren Form vor; Sie laden es einfach auf die gewünschte Adresse und führen ihn durch Angabe dieser Adresse aus. Wenn Sie MONA3 ein weiteres Mal aufrufen (wenn Sie beispielsweise von MONA3 auf BASIC verzweigt hatten), müssen Sie MONA3 mit einer um dezimal 2 erhöhten Adresse aufrufen.

Um MONA3 in Ihren Computer zu laden, tippen Sie einfach:

```
RUN"[ENTER]
```

und drücken **[PLAY]** an Ihrem Cassettenteil. Zunächst wird ein kurzes BASIC-Programm geladen. Dies läuft automatisch an und gibt die Meldung

```
Load Address?
```

aus.

Sie müssen eine Dezimalzahl zwischen **1000** und **30000** eingeben und **[ENTER]** drücken. Die Zahl, die Sie eingeben, stellt die Ladeadresse für den Disassembler dar – eine gute Adresse ist **30000**.

Nach Eingabe der Zahl erscheint die Meldung

```
Please wait...Loading MONA3..
```

auf dem Bildschirm und der Datacorder läuft erneut automatisch an. Jetzt wird der Disassembler in den Computer geladen; dies dauert ungefähr 100 Sekunden. Wenn MONA3 geladen ist, läuft es selbst an und gibt zur Bestätigung der Arbeitsbereitschaft (Kommando-Modus) ein ' > ' am Zeilenanfang aus (siehe Kapitel 2).

Wenn Sie beabsichtigen, gleichzeitig den Assembler GENA3 und den Disassembler/Debugger MONA3 in Ihren Computer zu laden, beachten Sie bitte, daß Sie immer das Programm, das Sie in den unteren Speicherbereich laden wollen, zuletzt laden. Sinnvoller Weise wird GENA3 in den unteren Speicherbereich (z. B. ab **30000**) geladen. In diesem Fall muß GENA3 zum Schluß geladen werden.

Die Kommandos werden direkt ausgeführt, sie müssen also nicht mit **[ENTER]** abgeschlossen werden. Ungültige Kommandos werden ignoriert. Die gesamte Anzeige wird nach jedem Kommando wiederholt, so daß Sie immer das Ergebnis eines bestimmten Kommandos beobachten können.

Viele Kommandos benötigen als Eingabe eine Hexadezimalzahl (0 – 9 und A – F oder a – f). Wenn Sie eine Hexazahl eingeben und beenden diese Eingabe mit einem nicht-hexa Zeichen, so wird, wenn dieses Beendigungszeichen ein gültiges Kommando ist, dieses Kommando (nach evtl. noch ausstehenden) durchgeführt.

Wenn das Beendigungszeichen ein Minus '–' ist, dann wird die Hexazahl in negativer Form (Zweierkomplement) wiederholt, d. h.

1800– ergibt **E800**

Wenn Sie bei einer Hexazahl mehr als 4 Ziffern eingeben, dann werden nur die vier zuletzt eingegebenen Ziffern beachtet und am Schirm angezeigt.

Kapitel 2

Der Kommando-Vorrat

Nachfolgend werden die möglichen Kommandos für MONA3 beschrieben. Wenn in diesem Kapitel eine Hexadezimalzahl mit **[ENTER]** abgeschlossen wird, so kann dies auch durch Eingabe eines nicht-hexa Zeichens geschehen. Außerdem wird zur Darstellung eines Zwischenraums dort das Zeichen '_' geschrieben, wo er notwendig ist.

[CTRL] D

schaltet die Zahlenbasis, in der die Adressen ausgegeben werden, zwischen hexadezimal (16) und dezimal (10) wechselweise um. Beim ersten Aufruf von MONA3 werden die Adressen hexadezimal ausgegeben; durch Eingabe von **[CTRL] D** wird auf Dezimalausgabe umgeschaltet, und erneut **[CTRL] D** kehrt wieder auf hexa um, usw. Dies betrifft sämtliche Adressen, die über MONA3 ausgegeben werden, einschließlich derer, die vom Disassembler generiert werden. Nicht umgeformt werden dagegen Speicherinhalte; 8-Bit-Zahlen werden immer hexadezimal ausgegeben.

[CTRL] A

gibt eine neue Seite aus, beginnend mit der Adresse, auf die der Speicherzeiger (Pointer) verweist. Dies ist oft nützlich, um von der momentanen Stellung aus zu sehen, welche Befehle folgen. Durch erneute Eingabe von **[CTRL] A** kommen Sie wieder in den Kommando-Modus; jede andere Eingabe gibt eine weitere Seite der Dis-Assemblierung aus.

'→' Cursor rechts

erhöht den Speicherzeiger um 1, so daß die 32 Byte lange Speicheranzeige sich an einer um 1 größeren Adresse ausrichtet als die vorhergehende.

'←' Cursor links

vermindert den Speicherzeiger um 1

'↑' Cursor hoch

vermindert den Speicherzeiger um 8, so daß man schneller im Speicher zurückschreiten kann.

'↓' Cursor runter

erhöht den Speicherzeiger um 8, um schnell im Speicherbereich vorwärts zu schreiten.

'G'

sucht nach einer angegebenen Zeichenkette. Sie werden durch einen Doppelpunkt (:) aufgefordert, das erste Byte, nach dem Sie suchen wollen, einzugeben und

[ENTER] zu drücken. Sie können jetzt byteweise Ihre Suchkette eingeben, immer mit der **[ENTER]**-Taste abgeschlossen. Wenn die Zeichenkette vollständig eingegeben ist, drücken Sie nur noch **[ENTER]**. Dies schließt die Definition der Zeichenkette, nach der Sie suchen, ab und der Suchprozeß nach dem 1. Auftreten der Zeichenkette, beginnend mit der Adresse, auf die der Speicherzeiger gerade verweist, läuft ab. Wenn die gesuchte Zeichenkette gefunden wird, verweist der Speicherzeiger auf das erste Zeichen der gefundenen Zeichenkette.

Nehmen wir an, Sie wollen den Speicher, beginnend bei `#8000`, nach der Zeichenkette `#3E#FF` (2 Bytes) durchsuchen, dann verfahren Sie wie folgt:

`M:8000[ENTER]`	setzen des Speicherzeigers auf `#8000`
`G:3E [ENTER]`	Angabe des 1. Bytes der Suchkette
`FF [ENTER]`	Angabe des (der) nächsten Bytes der Kette
[ENTER]	Abschluß der Suchkette

Nach dem letzten **[ENTER]** (oder jedes nicht-hexa Zeichens) startet das Kommando `'G'` einen Suchprozess ab `#8000` nach dem ersten Auftreten von `#3E#FF`.

Wenn diese gefunden wird, sind Sie wieder im Kommando-Modus. Sie können dann weitere Ausprägungen der Suchkette mit dem Kommando `'N'` finden.

`'H'`

Wandelt eine Dezimalzahl in ihr hexadezimales Äquivalent um.

Sie werden mit einem Doppelpunkt (:) aufgefordert, eine Dezimalzahl einzugeben, die durch ein nicht-numerisches Zeichen (jedes Zeichen außer 0...9) beendet wird. Dann erscheint ' = ' auf der gleichen Zeile gefolgt vom Hexa-Wert für die entsprechende Dezimalzahl.

Drücken Sie irgendeine Taste, um wieder in den Kommando-Modus zu gelangen.

Beispiel:

`H:41472__=A200` hier wurde der Zwischenraum zur Beendigung der Dezimalzahl eingegeben

`'I'`

Intelligentes Kopieren

Dieses Kommando wird angewendet, um einen Speicherblock an einen anderen Platz zu kopieren; intelligent ist es in der Art, daß es an eine Stelle kopiert werden kann, die den früheren Block zerstören würde.

'I' fordert die Beginn- und Endadresse (`'First:'`, `'Last:'`, beide inkl.) des zu kopierenden Speicherblocks, sowie die Anfangsadresse der Kopie (`'To'`) an. Alle Anforderungen müssen durch eine Hexadezimalzahl beantwortet werden. Wenn die Beginnadresse größer ist als die Endadresse, wird das Kommando abgebrochen, ansonsten wird anforderungsgemäß kopiert.

'J'

führe den Code aus ab einer angegebenen Adresse

Dieses Kommando fordert mit einem Doppelpunkt (:) eine Hexa-Zahl an – wenn diese einmal eingegeben ist, werden sämtliche (noch ausstehenden) Anforderungen zurückgesetzt, der Bildschirm gelöscht und die Steuerung an die angeforderte Adresse übergeben. Wenn Sie nach Durchführen des Codes wieder in den Kommando-Modus gelangen möchten, müssen Sie Unterbrechungspunkte (siehe '!'–Kommando) dort setzen, wo Sie zur Ausgabe zurückkehren möchten.

Beispiel:
J : B000[ENTER] führt den Code ab **#B000** aus.

Sie können dieses Kommando abbrechen, wenn Sie vor Beendigung der Adresseneingabe **[ESC]** anwenden.

[CTRL]C

veranlaßt das Fortfahren der Programmausführung ab der Adresse, auf die momentan der Befehlszähler verweist.

Dieses Kommando wird wahrscheinlich am häufigsten in Verbindung mit dem '!'-Kommando verwendet – ein Beispiel soll die Anwendung verdeutlichen:

Nehmen wir an, Sie gehen mittels **[CTRL] S** schrittweise durch den Code, der nachstehend aufgelistet ist und haben die Adresse **#8920** erreicht. Sie sind jedoch nicht daran interessiert, schrittweise durch das Unterprogramm ab **#9000** zu gehen, sondern wollen lediglich den Zustand der Flagbytes sehen, wie sie sich nach dem Aufruf des Unterprogramms auf **#8800** darstellen.

```
891E 3EFF              LD   A,-1
8920 DC0090            CALL #9000
8923 2A0080            LD   HL,(#8000)
8926 7E                LD   A,(HL)
8927 111488            LD   DE,#8814
892A CD0088            CALL #8800
892D 2003              JR   NZ,lab1
892F 320280            LD   (#8002),A
8932 211488 lab1:      LD   HL,#8814
```

Verfahren Sie wie folgt:

Setzen Sie einen Unterbrechungspunkt mit dem '!'-Kommando an der Stelle **#892D** (denken Sie daran, erst mit **'M'** den Speicheranzeiger zu setzen, bevor Sie das **[CTRL] C**-Kommando geben).

Die Ausführung startet mit der Adresse, auf die der Befehlszähler verweist – in unserem Fall **#8920** – und läuft durch, bis der eingegebene Unterbrechungspunkt (**#892D**) erreicht ist. Es folgt eine erneute Anzeige und Sie können den Stand der Flags usw. überprüfen. Dann können Sie im Einzelschritt wieder durch den Code gehen.

'L'

Liste einen Speicherblock auf, beginnend mit der Adresse, auf die der Speicherzeiger verweist. 'L' löscht den Bildschirm und ein 160 Byte langer Bereich wird in hexadezimaler Form und im ASCII-Zeichensatz dargestellt. Die Adressen werden abhängig vom momentanen Status von **[CTRL] D** (siehe oben) entweder dezimal oder haxadezimal angezeigt. Die Anzeige erfolgt in 20 Reihen á 8 Bytes. Am Ende jeder Reihe stehen die ASCII-Zeichen. Um alle ASCII-Zeichen darstellen zu können, wird die Wertigkeit von allen Zeichen über 127 um 128 vermindert, und Zeichen mit Wertigkeiten von 0 – 31 werden mit Punkt (.) dargestellt.

Am Ende einer Seite haben Sie jeweils die Möglichkeit, durch Drücken von **[ESC]** in den Kommando-Modus zurückzukehren oder durch Drücken einer beliebigen anderen Taste die nächste Seite mit 160 Bytes anzufordern.

'M'

Setzen des Speicherzeigers auf die angegebene Adresse.

Sie werden mit einem Doppelpunkt (:) aufgefordert, eine hexadezimale Adresse einzugeben. Der Speicherzeiger wird dann so gesetzt, daß er auf die eingegebene Adresse verweist. Ebenso wird die Speicheranzeige auf dem Bildschirm auf diesen Wert gesetzt.

'M' ist immer vor Speicherauflistung, Codeeingaben usw. zum Setzen einer Bezugsadresse notwendig.

'N'

Suche das nächste Auftreten der hexadezimalen Zeichenkette, die zuletzt durch das Kommando 'G' spezifiziert wurde.

'N' beginnt mit der Suche bei der Adresse, auf die der Speicherzeiger verweist. Wenn das nächste Auftreten der Zeichenkette gefunden wurde, wird die Speicheranzeige auf dem Bildschirm auf diese Adresse gesetzt.

'O'

Bestimmen eines relativen Abstands.

Dieses Kommando zieht das Byte, auf das der Speicherzeiger gerade verweist, als Basis zur Bestimmung eines relativen Abstands heran und verändert die Speicheranzeige entsprechend.

Beispiel:
Nehmen wir an, der Speicherzeiger verweist auf #6800 und der Inhalt von #67FF und #6800 wäre #20 und #16, was als Befehl JR NZ,$+24 interpretiert werden könnte. Um herauszufinden, wohin der Befehl verzweigen würde, wenn die Nicht-Null-Bedingung auftritt, drücken Sie einfach 'O', wenn der Speicherzeiger auf das Byte #16 verweist. Die Anzeige wird dann auf #6817 die angegebene Verzweigung, gestellt. Beachten Sie, daß relative Abstände größer als #7F (127 dez.) vom Z80-Prozessor als negativ behandelt werden, 'O' beachtet dies auch in seiner Berechnung. Siehe auch unter dem 'U'-Kommando, das in Verbindung mit 'O' angewendet wird.

'P'

Überschreibt den Speicher zwischen zwei vorgegebenen Grenzen mit einem vorgegebenen Byte. 'P' fordert Wert für **'First:'** (Beginn), **'Last:'** (Ende) und **'With:'** (Byteinhalt) an. Beantworten Sie die Anforderungen jeweils mit der Eingabe von Hexadezimalzahlen, d. h. Beginn- und Endeadressen (beide inkl.) des Blocks, den Sie überschreiben wollen und das Byte, mit dessen Inhalt Sie den Speicherblock überschreiben wollen.

Beispiel:

```
P
First:7000 [ENTER]
Last:77FF [ENTER]
With:55 [ENTER]
```

überschreibt die Stellen **#7000** bis **#77FF** (einschließlich) mit dem Byte **#55** ('U'). Wenn die Startadresse größer ist als die Endadresse, wird 'P' abgebrochen.

'R'

liest einen Objektcode vom Band – die Datei kann über das **'W'**-Kommando von MONA3 oder die Kommandos **'O'** oder 'T' von GENA3 erstellt worden sein. Sie werden aufgefordert, einen Dateinamen einzugeben (drücken Sie nur **[ENTER]**, wenn Sie den Dateinamen nicht wissen) und dann die Adresse, ab der der Code geladen werden soll. Die Adresse müssen Sie angeben.

'>'

Setzen eines Unterbrechungspunktes nach dem laufenden Befehl und Fortfahren mit der Ausführung.

Beispiel:

```
9000 B7          OR A
9001 C20098      CALL NZ,#9800
9004 01000       LD   BC,0
9800 21FFFF      LD   HL,-1
```

Sie gehen im Einzelschritt durch den oben angeführten Code und erreichen **#9001** mit einem Wert ungleich Null im Register A. Somit befindet sich das Null-Kennzeichen (Zero Flag) nach dem Befehl **OR A** im **NZ**-Status (nicht Null). Wenn Sie nun **[CTRL] S** eingeben, um im Einzelschritt weiterzumachen, wird auf **#9800** (die Adresse der Unterroutine) verzweigt. Wenn Sie nicht im Einzelschritt durch die Unterroutine wollen, verwenden Sie das '>'-Kommando auf Adresse **#9001** und der **CALL** wird automatisch durchgeführt und die Ausführung hält auf der Adresse **#9004** an, wo Sie wieder im Einzelschritt fortfahren können.

Bedenken Sie, daß '>' einen Unterbrechungspunkt nach dem laufenden Befehl setzt und dann das **[CTRL] C**-Kommando durchführt.

Ein ausführliches Beispiel des Einzelschritt-Verfahrens finden Sie beim **[CTRL] S**-Kommando.

'S'

setzt den Speicherzeiger auf die Adresse, die sich gerade im Stack (Sicherstellungsbereich, bezeichnet mit SP) befindet. Dies ist nützlich, um beispielsweise die Rückkehradresse einer aufgerufenen Unterroutine ausfindig zu machen, oder ähnliches.

'T'

dis-assemblieren eines Code-Blocks, wahlweise auf dem Drucker.

Sie werden zunächst aufgefordert, die Adressen für **'First:'** (Beginn) und **'Last:'** (Ende) des Codes, den Sie dis-assemblieren wollen, in hexadezimaler Form einzugeben. Wenn die Startadresse größer ist als die Endadresse, wird das Kommando abgebrochen. Nach Eingabe der Adresse werden Sie gefragt **'Printer?'** (Ausgabe auf Drucker); wenn Sie mit 'Y' antworten, weisen Sie für die Disasssemblierung den Drucker zu, jedes andere Zeichen veranlaßt, daß die Ausgabe auf dem Bildschirm erfolgt.

Nun werden Sie mit **'Text:'** aufgefordert, eine hexadezimale Adresse einzugeben, ab der die erzeugte Textdatei abgestellt werden soll. Wenn Sie nicht wünschen, daß eine Textdatei erstellt wird, drücken Sie nach der Aufforderung nur **[ENTER]**. Wenn Sie eine Adresse angeben, wird vom Disassembler eine Textdatei angelegt, die bei der eingegebenen Adresse beginnt und die eine Form aufweist, die eine Weiterverarbeitung über GENA3 ermöglicht. Wenn Sie die Textdatei mit GENA3 verwenden wollen, müssen Sie sie entweder bei der Adresse, die Sie im GENA3-Editor-Kommando 'X' verwenden, anlegen, oder sie an diese Adresse übertragen, da GENA3 die Textdatei an dieser Adresse erwartet. Sie müssen GENA3 auch das Ende der Textdatei mitteilen. Verwenden Sie dazu die 'END-of-Text'-Adresse, die Ihnen der Disassembler zur Verfügung stellt (siehe unten) und versorgen Sie damit **TEXTEND** von GENA3 – siehe dazu Abschnitt 3.2 im GENA3-Handbuch).

Sie müssen dann, um die Textdatei zu erhalten, GENA3 über den Warmstart-Einsprungpunkt aufrufen.

Wenn zu irgendeinem Zeitpunkt während der Erstellung der Textdatei der Text MONA3 überschreiben würde, wird die Disassemblierung abgebrochen. Drücken Sie dann irgendeine Taste, um in den Kommando-Modus zu gelangen.

Wenn Sie eine Adresse für die Textdatei angegeben haben, müssen Sie noch eine **'Workspace:'** Adresse (Arbeitsbereich) bestimmen. Dies ist der Beginn eines freien Speicherbereichs, der benötigt wird, um eine einfache Symboltabelle anzulegen, die jeden Namen, der vom Disassembler generiert wird, aufnimmt. Der benötigte Speicherplatz beträgt für jeden generierten Namen 2 Bytes. Sie können die Angabe dieser Adresse nicht umgehen.

Danach werden Sie wiederholt nach **'First:'** und **'Last:'** (einschließlich)-Adressen von möglichen Datenbereichen innerhalb des zu bearbeitenden Blocks gefragt. Diese Datenbereiche sind Speicherplätze, die nicht als Z80-Befehle interpretiert werden sollen. Es werden dafür vom Disassembler **DEFB**-Steuerungsanweisungen für den Assembler generiert. Wenn der Wert eines Datenbytes zwischen **32** und **127** einschließlich (**#20** und **#7F**) liegt, wird eine ASCII-Umschlüsselung durchge-

führt. d. h. #41 wird nach DEFB auf 'A' geändert. Wenn Sie die Angaben von Datenbereichen beendet haben, oder überhaupt keine angeben wollen, drücken Sie einfach **[ENTER]** als Antwort auf die beiden Anforderungen. Das 'T'-Kommando verwendet einen Bereich hinter MONA3, um die Adressen der Datenbereiche abzuspeichern. Sie können daher soviele Datenbereiche angeben, wie Speicherplatz vorhanden ist. Jeder Eintrag belegt 4 Bytes des Speichers. Beachten Sie, daß der Aufruf von 'T' alle vorher gesetzten Unterbrechungspunkte aufhebt – siehe '!'-Kommando.

Der Bildschirm wird jetzt gelöscht. Wenn Sie die Generierung einer Textdatei angefordert haben, wird jetzt eine kurze Unterbrechung auftreten, (abhängig von der Größe des Speicherabschnitts, den Sie mit dem Disassembler bearbeiten wollen), während der die Symboltabelle erstellt wird. Danach wird über Bildschirm oder Drucker die Liste der Disassemblierung ausgegeben. Sie können am Ende einer Zeile die Ausgabe der Liste unterbrechen, indem Sie irgendeine Taste drücken. Wenn Sie nachfolgend **[ESC]** drücken, gelangen Sie in den Kommando-Modus, durch Drücken jeder anderen Taste läuft die Disassemblierung weiter. Wenn ein ungültiger Operationscode festgestellt wird, wird er als NOP generiert und in der Liste mit einem Stern '★' nach dem Operationscode gekennzeichnet.

Am Ende der Disassemblierung wird die Ausgabe unterbrochen, und die Meldung 'End of Text xxxxx' wird angezeigt, falls Sie die Erstellung einer Textdatei gefordert haben. xxxx ist die Adresse (in hexa oder dezimal), die Sie mit POKE in den GENA3-Bereich für TEXTEND stellen müssen (das niedrigstwertige Byte zuerst), damit der Assembler die erstelite Textdatei bei einem Warmstart aufnehmen kann. Wenn die Assemblierung beendet ist, drücken Sie irgendeine Taste, um in den Kommando-Modus zu gelangen.

Namen werden überall dort, wo sie von Bedeutung sind (z. B. in C30078) in der Form 'Lxxxx' generiert, wobei xxxx die absolute Hexadezimaladresse des Namens darstellt; jedoch nur, wenn sich die Adresse innerhalb der für die Disassemblierung gegebenen Grenzen befindet. Wenn die Adresse außerhalb dieses Bereichs liegt, wird kein Name generiert, sondern es wird lediglich die hexadezimale oder dezimale Adresse angegeben. Wenn wir beispielsweise den Bereich zwischen #7000 und #8000 mit dem Disassembler bearbeiten, so wird der Befehl C30078 umgesetzt in

```
JP
L7800
```

Im anderen Fall, wenn wir den Bereich #9000 bis #9800 bearbeiten, wird der Befehl

```
C30078
```

in

```
JP
#7800        oder
JP
30720        umgesetzt, wenn wir die Dezimaldarstellung benutzen.
```

Wenn während der Disassemblierung auf eine bestimmte Adresse durch einen Befehl Bezug genommen wird, so erscheint der Name im Namensfeld des entsprechenden Befehls (vor dem Befehlscode), aber nur, wenn eine Textdatei angefordert wurde.

Beispiel: T

```
First: 8B [ENTER]
Last:9E [ENTER]
Printer? Y
Text:[ENTER]
First: 95 [ENTER]
Last:9E [ENTER]
First: [ENTER]
Last: [ENTER]
008B FE16                       CP   #16
008D 3801                       JR   C,L0090
008F 23                         INC  HL
0090 37      L0090              SCF
0091 225D5C                     LD   (#5C5D),HL
0094 C9                         RET
0095 BF524E                     DEFB #BF,"R","N"
0098 C4494E                     DEFB #C4,"I","N"
009B 4B4559                     DEFB "K","E","Y"
009E A4                         DEFB #A4
```

'U'

wird in Verbindung mit dem Kommando 'O' verwendet.

'O' setzt, wie beschrieben, die Speicheranzeige auf einen relativen Abstand, d. h. der Effekt eines JR- oder DJNZ-Befehls wird angezeigt.

'U' wird verwendet, um die Speicheranzeige wieder auf den Stand zu setzen, wie er vor dem letzten 'O' war.

Beispiel:

```
 7200 47          71F3 77
 7201 20          71F4 C9
>7202 F2<        >71F5 F5<
 7203 06          71F6 C5
 Anzeige 1        Anzeige 2
```

Sie sind in der Anzeige 1 und wollen wissen, wohin der relative Sprung 20F2 verzweigt. So drücken Sie 'O' und die Speicheranzeige bringt Anzeige 2.

Nun untersuchen Sie den Code ab #71F5 und wollen in den ursprünglichen Code, der dem relativen Sprung folgt, zurückkehren, um zu sehen was passiert, wenn das Null-Kennzeichen (Zero-Flag) gesetzt ist. Drücken Sie 'U' und es erscheint wieder Anzeige 1.

Beachten Sie bitte, daß Sie mit 'U' immer nur zum Auftreten des letzten 'O' zurückkehren können. Alle früheren 'O' sind nicht mehr bekannt.

'V'

wird in Verbindung mit dem 'X'-Kommando angewendet.

'V' entspricht in der Veränderung der Speicheranzeige dem 'U'-Kommando, außer daß die Speicheranzeige auf die Eingabe des letzten 'X'-Kommandos zurückgesetzt wird.

Beispiel:

```
 8702 AF          842D 18
 8703 CD          842E A2
>8704 2F<        >842F E5<
 8705 44          8430 21
 Anzeige 1        Anzeige 2
```

Sie sind in Anzeige 1 und wollen sich die Unterroutine auf #842F anschauen. Sie drücken 'X' und es erscheint die Anzeige 2.

Wie bei 'U' können Sie mit diesem Kommando nur zu der Adresse der letzten 'X'-Eingabe zurückkehren, alle früheren Adressen, bei denen 'X' eingegeben wurde, sind nicht mehr bekannt.

'W'

Schreibt einen Speicherblock unter einem angegebenen Dateinamen auf Band. Es wird ein Dateiname und dann die erste und letzte (inkl.) Adresse des Speicherblocks angefordert, den Sie sichern wollen.

'!'

setzt im Speicherzeiger einen Unterbrechungspunkt.

Ein Unterbrechungspunkt ist, soweit er MONA3 betrifft, lediglich ein CALL Befehl auf eine Routine in MONA3, die in den Kommando-Modus verzweigt, um dem Programmierer die Möglichkeit zu geben, die Ausführung eines Programms anzuhalten und den Inhalt von Z80-Registern, Kennzeichen (Flags) oder bedeutsame Speicherstellen zu überprüfen. Wenn Sie also den Ablauf eines Programms beispielsweise bei #9876 unterbrechen wollen, benutzen Sie das Kommando 'M' um den Speicherzeiger auf #9876 zu setzen und geben dann das Kommando '!', um an dieser Adresse den Ablauf zu unterbrechen.

Die 3 Bytes des Codes, der ursprünglich an #9876 stand, werden gesichert und durch einen CALL-Befehl ersetzt, der den Ablauf unterbricht, wenn die Ausführung diesen Befehl erreicht. Dann werden die 3 gesicherten Bytes wieder zurückgestellt auf #9876. Durch Drücken einer Taste sind Sie im Kommando-Modus. Alle Register und Flags sind nun in dem Zustand, wie er sich direkt vor der Unterbrechung darstellt. Sie können jetzt alle Möglichkeiten von MONA3 anwenden.

Zur Beachtung:

MONA3 benutzt den Bereich hinter dem eigenen Code, um Informationen über die Unterbrechung abzuspeichern. Das bedeutet, daß Sie soviele Unterbrechungen setzen können, wie Speicher verfügbar ist. Jede Unterbrechung erfordert einen Speicherplatz von 5 Bytes. Wenn eine Unterbrechung bearbeitet wird, speichert MONA3 automatisch den ursprünglichen Speicherinhalt zurück, wie er vor Setzen der Unterbrechung war.

Da das 'T'-Kommando ebenfalls den Speicherbereich (nach MONA3) benutzt, sind alle Unterbrechungspunkte verloren, wenn das 'T'-Kommando benutzt wird.

Unterbrechungspunkte können nur im RAM gesetzt werden. Da sich ein Unterbrechungspunkt in einem 3 Byte **CALL**-Befehl ausdrückt, ist in bestimmten Ausnahmefällen ein bestimmtes Maß an Vorsicht erforderlich, z. B. folgender Code:

```
8000 3E          8008 00
8001 01          8009 00
8002 18          800A 06
8003 06          800B 02
>8004 AF<        800C 18
8005 0E          800D F7
8006 FF          800E 06
8007 01          800F 44
```

Wenn Sie auf **#8004** einen Unterbrechungspunkt setzen und die Ausführung ab **#8000** beginnen, dann wird Register A mit dem Wert 1 geladen, nach **#800A** verzweigt, Register B mit dem Wert 2 geladen und nach **#8005** verzweigt. Da **#8005** mit dem niedrigwertigen Byte des **CALL**-Befehls für den Unterbrechungspunkt überschrieben wurde, haben wir nun einen zerstörten Code und das Ergebnis ist nicht vorhersehbar.

Diese spezielle Situation ist zwar ziemlich ungewöhnlich, dennoch sollten Sie sie bedenken. In diesem Fall bietet das Einzelschrittverfahren die Lösung.

Sehen Sie sich dazu das Beispiel mit weiteren Einzelheiten zum Einzelschritt im **[CTRL] S**-Kommando weiter hinten an.

Ebenso ist zu beachten, daß bei Erkennen eines Unterbrechungspunktes die Ausführung anhält und auf das Drücken einer Taste wartet. Erst dann kommen Sie in den Kommando-Modus.

'X'

wird verwendet, um den Speicherzeiger auf die Verzweigadresse eines absoluten **CALL**- oder **JP**-Befehls zu setzen.

'X' nimmt die 16-Bit-Adresse aus dem Byte, auf das der Speicheranzeiger verweist und dem folgenden Byte, und verändert die Speicheranzeige. Denken Sie daran, daß die niedrigwertige Hälfte der Adresse durch das erste Byte und die hochwertige Hälfte der Adresse durch das zweite Byte (INTEL-Format) angegeben wird.

Beispiel:

Sie wollen sich die Routine, die durch den **CALL CD0563** aufgerufen wird, anschauen.

Setzen Sie den Speicherzeiger (mit 'M') so, daß er auf die **05** im **CALL**-Befehl verweist und drücken Sie 'X'. Die Speicheranzeige wird nun auf die Stellen um **#6305** verändert.

Siehe auch das 'V'-Kommando in Verbindung mit dem 'X'.

'Y'

Eingabe von ASCII-Zeichen unter Benutzen des Speicherzeigers.

'Y' stellt eine neue Zeile zur Verfügung, in der Sie direkt über die Tastatur ASCII-Zeichen eingeben können. Diese Zeichen werden wiederholt und die entsprechende Hexadezimaldarstellung wird in den Speicher eingetragen, beginnend mit der Adresse, auf die der Speicherzeiger verweist. Die Zeichenkette muß durch **[ESC]** beendet werden, Zeichenlöschungen können durch **[DEL]** vorgenommen werden. Nach Beendigung der Eingabe der ASCII-Zeichen verweist der Speicherzeiger auf die nächste Stelle nach der eingegebenen Zeichenkette.

[CTRL] S

Einzelschritt

Vor Anwendung von **[CTRL] S** (oder '>') muß sowohl der Befehlszähler (PC) als auch der Speicherzeiger auf die Adresse des Befehls gesetzt werden, den Sie ausführen wollen.

[CTRL] S führt lediglich den laufenden Befehl aus.

Beachten Sie, daß Sie sich zwar im gesamten Speicher (RAM oder ROM) im Einzelschritt bewegen können, Sie kommen aber im Einzelschritt nicht über die **EXX**- oder **EX AF,AF**-Befehle.

Es folgt nun ein ausführliches Beispiel, das die Anwendung einer Vielzahl von Befehlen, die im Debugger MONA3 möglich sind, klären soll – es wird Ihnen empfohlen, dieses Beispiel sorgfältig zu studieren und es selbst auszuprobieren.

Nehmen wir an, wir haben 3 Abschnitte (Section's) von Code, wie nachfolgend aufgeführt, im Speicher. Der erste Abschnitt (Section 1) ist das Hauptprogramm, das **HL** und **DE** mit Zahlen versorgt und dann eine Routine aufruft, die die beiden Zahlen multipliziert (Section 2), das Ergebnis in **HL** abstellt und schließlich zweimal eine Routine aufruft, die das Ergebnis auf dem Bildschirm ausgibt (Section 3).

```
7080 2A0072          LD   HL,(#7200)   ;SECTION 1
7083 ED5B0272        LD   DE,(#7202)
7087 CD0071          CALL Mult
708A 7C              LD   A,H
708B CD1D71          CALL Aout
708E 7D              LD   A,L
708F CD1D71          CALL Aout
7092 210000          LD   HL,0
7100 AF        Mult: XOR  A             ;SECTION 2
7101 ED52            SBC  HL,DE
7103 19              ADD  HL,DE
7104 3001            JR   NC,Mu1
7106 EB              EX   DE,HL
7107 B2         Mu1: OR   D
7108 37              SCF
7109 C0              RET  NZ
```

```
710A B3              OR   E
710B 5A              LD   E,D
710C 2007            JR   NZ,Mu4
710E EB              EX   DE,HL
710F C9              RET
7110 EB        Mu2:  EX   DE,HL
7111 19              ADD  HL,DE
7112 EB              EX   DE,HL
7113 29        Mu3:  ADD  HL,HL
7114 D8              RET  C
7115 1F        Mu4:  RRA
7116 30FB            JR   NC,Mu3
7118 B7              OR   A
7119 20F5            JR   NZ,Mu2
711B 19              ADD  HL,DE
711C C9              RET
711D F5        Aout: PUSH AF              ;SECTION 3
711E 0F              RRCA
711F 0F              RRCA
7120 0F              RRCA
7121 0F              RRCA
7122 CD2671          CALL Nibble
7125 F1              POP  AF
7126 E60F    Nibble: AND  %1111
7128 C690            ADD  A,#90
712A 27              DAA
712B CE40            ADC  A,#40
712D 27              DAA
712E CD5ABB          CALL #BB5A
7131 C9              RET
 .
 .
7200 1B2A            DEFW 10779
7202 0200            DEFW 2
```

Nun wollen wir den obigen Code überprüfen, entweder um zu sehen, ob er funktioniert, oder wie er abläuft. Wir können dies mit nachstehender Kommandofolge erreichen – es muß erwähnt werden, daß dies nur eine von vielen Möglichkeiten ist, sich durch den Code durchzuarbeiten und nicht notwendiger Weise die wirkungsvollste darstellt; sie sollte jedoch dazu dienen, das Einzelschrittverfahren darzustellen:

M:7080[ENTER]	setze den Speicherzeiger auf **#7080**
7080	setze den Befehlszähler auf **#7080**
[CTRL] S	Einzelschritt
[CTRL] S	Einzelschritt
[CTRL] S	folge dem **CALL**
M:7115[ENTER]	überspringe die Vorverarbeitung der Zahlen
!	setze einen Unterbrechungspunkt
[CTRL] C [ENTER]*)	setze die Ausführung ab **#7100** bis zum Unterbrechungspunkt fort
[CTRL] S	Einzelschritt
[CTRL] S	Einzelschritt
[CTRL] S	Einzelschritt
[CTRL] S	Einzelschritt
[CTRL] S	Einzelschritt
[CTRL] S	Einzelschritt
[CTRL] S	Einzelschritt
[CTRL] S	
[CTRL] S	
[CTRL] S	Rückkehr von der Multiplikationsroutine
[CTRL] S	Einzelschritt
[CTRL] S	folge dem **CALL**
M:7128[ENTER]	setze den Speicherzeiger auf ein interessierendes Bit
!	setze einen Unterbrechungspunkt
[CTRL] C [ENTER]*)	Setze die Ausführung fort ab **#711D** bis zum Unterbrechungspunkt
[CTRL] S	Einzelschritt
[CTRL] S	Einzelschritt
[CTRL] S	Einzelschritt
[CTRL] S	Einzelschritt
S	Anzeige der Rücksprungadresse
!	setze hier einen Unterbrechungspunkt
[CTRL] C [ENTER]*)	und führe weiter aus
[CTRL] S	Einzelschritt
S	Rückkehr von der **Aout**-Routine
!	
[CTRL] C [ENTER]*)	
[CTRL] S	Einzelschritt
>	Durchführung des gesamten **CALL** bis **Aout**

*) **[ENTER]** an dieser Stelle ist ein Beispiel dafür, daß irgendeine Taste gedrückt werden muß, um eine erneute Anzeige zu erhalten.

Arbeiten Sie das obige Beispiel bitte durch, indem Sie zuerst den Code der Routine eingeben (siehe nachfolgend 'Speicherveränderung') oder GENA3 benutzen, und dann die obigen Kommandos ablaufen lassen. Sie werden bemerken, daß das Beispiel eine unschätzbare Hilfe darstellt, um zu verstehen, wie ein Programmablauf verfolgt werden kann.

[CTRL] L

Dieses Kommando entspricht exakt dem 'L'-Kommando, mit der Ausnahme, daß die Ausgabe auf dem Drucker anstatt auf dem Bildschirm erfolgt. Denken Sie daran, daß Sie am Ende einer Seite durch Drücken von **[ESC]** wieder in den Kommando-Modus zurückkehren und jede andere Taste (außer **[CTRL] X**) eine neue Seite bereit stellt.

'Speicherveränderung'

Der Inhalt einer Adresse, auf die der Speicherzeiger verweist, kann verändert werden, indem man eine Hexadezimalzahl gefolgt von einem Beendigungszeichen (nicht-hexa Zeichen → alle Zeichen außer 0 – 9, A – F) eingibt. Die beiden niedrigstwertigen Hexadezimalziffern (wenn Sie nur 1 Zeichen eingeben, wird es links mit 0 ergänzt) werden in die Adresse gespeichert, auf die der Speicherzeiger verweist.

Anschließend wird das Kommando (falls möglich), welches dem Beendigungszeichen entspricht, durchgeführt. Stellt das Beendigungszeichen kein gültiges Kommando dar, wird es ignoriert.

Beispiele:

F2→	#F2	wird eingespeichert und der Speicherzeiger um 1 erhöht
123↓	#23	wird eingespeichert und der Speicherzeiger um 8 erhöht
EM:E00	#0E	wird eingespeichert in die Speicherstelle, auf die der Speicherzeiger verweist und dann wird der Speicherzeiger auf #E00 geändert. Beachten Sie, daß ein Zwischenraum '_' benutzt wurde, um das 'M'-Kommando hier abzuschließen
8C0	#8C	wird abgespeichert und dann der Speicherzeiger verändert (wegen der Angabe des '0' Kommandos), um auf die Adresse mit dem relativen Abstand von #8C zu verweisen, d. h. sein momentaner Wert ist –115
2A5D_	#5D	wird eingespeichert und der Speicherzeiger wird nicht verändert, da das Beendigungszeichen ein Zwischenraum ist und kein gültiges Kommando

Registerveränderung

Wenn als Antwort auf eine '>' Anforderung eine Hexadezimalzahl eingegeben wird, die mit einem Punkt '.' abgeschlossen wird, so wird die angegebene Zahl in das Z80-Register eingespeichert, welches durch den rechten Pfeil '>' gerade adressiert ist.

Beim Eintritt in MONA3 zeigt '>' auf den Befehlszähler (PC) und so wird durch die Angabe des Punktes '.' als Beendigungszeichen für eine vorausgehende Hexadezimalzahl der Befehlszähler verändert. Wenn Sie irgend ein anderes Register verändern wollen, dann geben Sie den Punkt '.' allein ein (nicht als Beendigungszeichen) und der Zeiger '>' wird in einer Schleife über die Register PC bis AF geführt. Beachten Sie, daß es auf diese Weise nicht möglich ist, den Stackanzeiger (SP) oder die IR-Register zu adressieren (und somit zu verändern).

Beispiele:

Nehmen wir an, der Registerzeiger '>' adressiert ursprünglich den Befehlszähler (PC)

.	zeigt auf IY
.	zeigt auf IX
0.	setzt IX auf 0
.	zeigt auf HL
123.	setzt HL auf #123
.	zeigt auf DE
.	zeigt auf BC
E2A7.	setzt BC auf #E2A7
.	zeigt auf AF
FF00.	setzt A auf #FF und löscht alle Flags
.	zeigt auf PC
8000.	setzt den Befehlszähler PC auf #8000

[CTRL] J

Dieses Kommando verzeigt in den Assembler GENA3, wenn er geladen ist und mindestens einmal angewendet wurde; ansonsten wird das Kommando ignoriert.

[CTRL] X

Kehrt zu BASIC bzw. zu dem Programm zurück, von dem aus MONA3 aufgerufen wurde.