ASDIS Manual

ASDIS

Assembler/Disassembler/Reverse-Assembler/Einzelschrittbetrieb
(c)Horst Kling

Allgemeines
ASDIS ist eine Sammlung von Funktionen, die alle das Programmieren in Assemblersprache unterstützen sollen. Bei der Entwicklung wurde besonderer Wert auf Absturzsicherheit und Einfachheit der Benutzung gelegt. Trotzdem ist es leider unumgänglich, dass Sie vor der Benutzung die Beschreibung gründlich durchlesen, denn dieses Programm wurde speziell auf den ZX81 zugeschnitten und unterscheidet sich in einigen Punkten von anderen Assemblern.

Hier in Sichworten die wichtigsten Leistungsmerkmale von ASDIS:

- Der interaktive Editor unterstützt das Schreiben eines Assemblerprogrammes und verhindert Syntaxfehler.
- Der Assembler erzeugt daraus den lauffähigen Maschinencode.
- Der Disassembler wandelt Maschinencode in lesbare Mnemoniks um.
- Der Reverse-Assembler wandelt Maschinencode in ein Assemblerlisting um, das dann mit dem Editor weiterbearbeitet werden kann.
- Mit den Einzelschrittbetrieb kann ein Maschinenprogramm schrittweise simuliert werden. Dabei hat man gleichzeitig Einblick in den Prozessor.
- Alle Tasten haben Autorepeat, wenn sie länger als ca 1/2 sec festgehalten werden.
- Beliebig viele Labels und Konstanten können definiert werden.
- Arbeitsspeicher (vergleichbar mit Basic-Variablen) können definiert werden.
- Das Programm wurde vollständig in Maschinencode geschrieben und hat deshalb kurze Reaktionszeiten.
- ASDIS gibt es als 16K- oder als 64K-Version, je nach Größe Ihres Speichers. Die 16K-Version belegt den Speicher ab etwa 25000, während die 64K-Version im Bereich von 8192 bis 16384 liegt. Basic und Assembler beeinflussen sich gegenseitig nicht, können also parallel benutzt werden.

Laden
Die Kassettenversion laden Sie mit dem Namen "ASDIS" oder einfach ''. Wenn es richtig eingelesen wurde, meldet das Programm sich mit diesem Namen und zeigt die Adresse an, an der der Assembler aufgerufen werden kann. Diese Adresse sollten Sie abschreiben, denn sobald man eine Taste berührt, wird NEW ausgeführt. Je nach RAM wurde der Maschinencode nach 8192(64K) oder nach ca 25000(16K) verschoben. Zusätzlich wurde bei der 16K-Version der RAMTOP darunter gelegt, so dass das Programm vor NEW geschützt ist. Deshalb können weitere Programme bzw. ein Assemblerlisting nachgeladen werden, welches sich dann mit dem Assembler bearbeiten lässt.

Listing
Das Listing ist der Speicherbereich, in den das Programm alles schreibt, was im Editor eingegeben wird (also Kommentare, Labels, Befehle, usw.).
Wenn das Programm gestartet wird, überprüft es zuerst, ob sich schon ein Assemblerlisting im Speicher befindet. Wenn das nicht der Fall ist, wird automatisch eines erzeugt.
Der Benutzer braucht sich um das Listing überhaupt nicht kümmern. Er sollte nur wissen, dass es immer in der Basic-Variablen A$ steht. Diese Variable darf also nicht verändert werden.
Ein geschriebenes Listing kann man mit Basic-SAVE auf Kassette speichern und mit LOAD zur Weiterbearbeitung wieder laden.

Mit CLEAR kann man ein Listing komplett löschen. Diese Anweisung ist für ASDIS dasselbe, wie NEW für das Basic.

Symbole
Symbole können bis zu 5 Zeichen lang sein, müssen immer mit einem Buchstaben beginnen und dürfen nur aus Buchstaben und Ziffern bestehen. Blanks innerhalb des Symbols sind nicht erlaubt. Folgende Symbole sollten Sie nicht benutzen, da der Assembler sie mit anderen Mnemonikteilen verwechselt:

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

Alle anderen Symbolnamen sind erlaubt.

ASDIS bietet 2 verschiedene Arten von Symbolen an: Marken und Labels. Der Unterschied besteht darin, dass Labels listingabhängige Symbole sind, deren Wert sich mit dem Listing ändern kann, während Marken benutzerdefinierte Symbole sind, die nur vom Programmierer geändert werden können.
Gemeinsam ist beiden, dass sie eine 16-Bit-Zahl darstellen. Im Programm können Sie beide Symbolarten völlig gleich verwenden, um z.B. einen Zahlenwert zu bekommen oder einen Sprung auszuführen. Solch eine Verwendung ist ein Aufruf.

Beispiel:
Der Befehl LD HL,(DFILE) ist erlaubt, unabhängig davon, ob DFILE ein Label oder eine Marke ist.

Zahlen
Zahlen kann man als Dezimalzahlen, Hexzahlen oder Symbole eingeben.
- Dezimalzahlen werden ohne Zusatz geschrieben.
- Hexzahlen müssen mit einem "$" beginnen.
- Symbole müssen mit einem Buchstaben beginnen.

Negative Zahlen und Rechenoperatoren sind nicht erlaubt.
Wenn ein eingegebenes Symbol nicht existiert, wird es je nach Situation entweder nicht akzeptiert oder stattdessen der Wert 0 angenommen.
Die Tatsache, dass alle 3 Eingabeformen immer völlig gleichwertig sind, sollten Sie besonders beachten, denn dadurch lassen sich viele Funktionen sehr einfach verwenden. Wenn also bei irgendeiner Funktion eine Zahl eingegeben werden soll, kann man stattdessen immer auch ein Symbol eingeben.

Funktionstasten
Immer wenn verschiedene Funktionen gleichzeitig zur Auswahl stehen, können sie durch einen einzelnen Tastendruck aufgerufen werden. In dieser Zeit reagieren alle gültigen Tasten wie Funktionstasten. Das ist z.B. im Hauptprogramm der Fall, wo man durch einen Tastendruck eine Funktion aufrufen kann oder in vielen Funktionen, wo durch Tastendruck Unterfunktionen erreichbar sind.
Alle Funktionen werden bei längerem Tastendruck automatisch wiederholt.
Es wurde versucht, denselben Tasten soweit möglich immer dieselben Funktionen zuzuordnen.
6 - vorwärts
7 - rückwärts
8 - Return
9 - Ausgabe an Drucker

Text- oder Zahleneingaben
Wenn eine Funktion weitere Eingaben erwartet, stellt sie einen entsprechenden Eingaberaum zur Verfügung, innerhalb dessen man die Eingabe machen kann.

(Wenn z.B. eine Zahl oder ein Label erwartet wird, kann man nur eine 5stellige Eingabe machen. Bei der Eingabe einer Befehlszeile werden maximal 15 Zeichen erwartet usw.) Diese Eingabe muss normalerweise linksbündig sein.
Bei allen Eingaben haben die Tasten N/L oder Shift und eine der Ziffern 0 bis 9 immer folgende Bedeutung:

N/L - Eingabe beenden
Shift 5 - Cursor (ein invertiertes Zeichen) nach links
Shift 8 - Cursor nach rechts
Shift 6 - das Zeichen, auf den der Cursor steht, wird gelöscht und alle folgenden werden verschoben (Delete)
Shift 7 - an der Stelle des Cursors wird ein Blank eingefügt, alle folgenden Zeichen werden verschoben (Insert)
Shift 9 - Umschaltung in invertierte Schreibweise. Nach N/L oder erneutem Shift wird wieder in normale Schreibweise geschalten.

Die invertierte Schreibweise ist nur für Kommentare und Textzeilen interessant.
Eine Rubout-Funktion gibt es nicht, dafür kann man alle Zeichen direkt überschreiben. Autorepeat bei längerem Tastendruck.

Hauptprogramm
Aus dem Hauptprogramm lassen sich 20 Funktionen direkt aufrufen. Die unteren 5 Bildzeilen zeigen dauernd die Kurzbezeichnungen dieser Funktionen an und die Tasten, mit welchen man sie starten kann. Wenn eine Funktion aufgerufen ist, steht anstelle der entsprechenden Taste ein invertiertes ">".
Wenn die Funktion beendet ist, kehrt sie ins Hauptprogramm zurück und der invertierte Winkel verschwindet wieder.

Funktionen des Hauptprogramms
1.EINGB:
Dies ist die Eingabefunktion (Editor), mit der man ein Listing schreiben kann. Da sie aber viele Unterfunktionen enthält, wird sie am Schluss gesondert beschrieben.

2.WERTE:
Werte können umgerechnet werden. Man gibt eine Zahl oder ein Symbol ein und der entsprechende Wert wird dezimal und hexadezimal angezeigt.

3.SCHIB:
Ein Datenblock kann von beliebiger Stelle im RAM in beliebiger Entfernung nach vorn oder hinten verschoben werden. Auch Überlappungen sind erlaubt. (Z.B. Verschiebung von 100 Bytes um 10 Adressen nach hinten oder vorn.) Es erscheint das Wort "VON" und man gibt ein, von wo ein Datenblock geholt werden soll. Nach N/L erscheint das Wort "NACH" und man gibt die Zieladrasse ein. Sobald man nach dem Wort "ZAHL" die Anzahl der Bytes eingegeben hat, die verschoben werden soll, wird die Funktion ausgeführt. Nach der Ausführung erscheint "OK". Sie wird nicht ausgeführt und ins Hauptprogramm zurückgekehrt, wenn eine der 3 Eingaben ungültig ist.

4.SUCHE:
Eine Folge von 1 bis 16 Bytes kann gesucht werden. Es werden bis zu 32 Hexzeichen ("0" ... "F") erwartet. Diese Zeichen stellen 1 bis 16 Bytes dar. Sonderzeichen (wie z.B. Blanks) dürfen deshalb nicht eingegeben werden. Die Zeile wird immer bis zum ersten nichthexadezimalen Zeichen als eingegeben angesehen. Nach "N/L" erscheint die Frage "AB WO" und man gibt die Adresse ein, ab der gesucht werden soll. Jeweils die Anfangsadresse eines gefundenen Blocks wird jetzt dezimal und hexadezimal angezeigt. Auf irgendeinen Tastendruck wird der nächste identische Block gesucht, mit "8" kann die Funktion

verlassen werden. Die zu suchende Folge wird immer an einer bestimmten Stelle im Speicher abgelegt, dort also auch immer gefunden.

5.INHLT:
Diese Funktion verschafft per Tabelle einen Überblick über den Inhalt des Assemblerlistings, wobei die einzelnen Worte folgendes bedeuten:

LAUFADR: Laufadresse, also die Adresse, an welcher das Programm nach dem Assemblieren lauffähig ist. (Siehe Funktion A)
ABLEGEN: Adresse, wo das Programm nach dem Assemblieren abgelegt werden soll. (Siehe Funktion A)
M-CODE : gibt die Länge des Maschinencodes an, der im Listing enthalten ist, also die Länge (in Bytes), die das reine Maschinenprogramm bisher hat (wenn man alle Labels, Kommentare, interne Codierung usw. abrechnet).
LISTING: gibt die Länge des Listings (in Bytes) an. Das ist die Länge der Variablen A$ (die durch Eingabe eines Assemblerprogramms vergrössert wurde) inklusive Labels, Kommentare, usw.
ZEILEN : Zahl der Zeilen, die das Listing enthält.
KOMMENT: Zahl der Kommentare
BEFEHLE: Zahl der Befehlszeilen
VARIABL: Zahl der definierten Arbeitsspeicher (Frei-, Text-, Daten-zeilen)
MARKEN : Zahl der definierten Marken.
LABELS : Zahl der verwendeten Labels.
AUFRUFE: Zeigt an, wieviele Symbole durch das Listing aufgerufen wurden. (Siehe Symbole)
FREI AB: Adresse des ersten Bytes das von Basic (also auch vom Listinq) nicht mehr benötigt wird.
BYTES : Anzahl der freien Bytes zwischen "FREI" und dem Maschinenstack. Zeigt also die Grösse des Speicherbereiches an, der für Maschinencode frei ist.

(Alle Werte werden dezimal und hexadezimal angezeiqt.)

6.RNAM.
Ein Symbol kann umbenannt werden. Es erscheint das Wort "ALT:" und man gibt die bisherige Schreibweise ein, danach erscheint "NEU:" und man gibt den neuen Namen ein. Wenn 2 richtig geschriebene Symbole eingegeben wurden, durchsucht das Programm das komplette Listing und ersetzt jedes Vorkommen des alten Symbols durch das neue und sagt anschliessend "OK".
Diese Funktion ist vor allem im Zusammenhang mit dem Reverse-Assembler interessant, wenn man automatisch erzeugte Symbole umbenennen will.

7.SYMBL:
Sämtliche Symbole werden aufgelistet und deren dezimale und hexadezimale Werte angezeigt. Zuerst erscheint das Wort "MARKEN" und nach den Marken das Wort "LABELS" und alle Labels. Es werden jeweils max. 18 Symbole angezeigt, bei Tastendruck erscheinen die nächsten. Normalerweise werden die Symbole relativ ungeordnet abgespeichert und deshalb auch ungeordnet angezeigt.
Wenn man aber Shift und 7 drückt statt 7, dann werden die Symbole vor der Anzeige zuerst alphabetisch geordnet (im Fast-Modus).

8.RETURN:
Kehrt ins Basic zurück.

9.DRUCK:
Ausgabe des oberen Bildschirmteils an den Drucker.

A.ASSMB:

Diese Taste startet den Assembler. Da es sich bei ASDIS um einen 2-Pass-Assembler handelt, dauert dieser Vorgang im allgemeinen nur Sekundenbruchteile. Aber auch Listings von 40KB werden immer in weniger als 60 sec assembliert. Im ersten Assembler-Pass werden die Werte aller Labels berechnet und im 2. Lauf werden diese Werte überall eingesetzt, wo das entsprechende Symbol aufgerufen wird.
Da der Editor so gestaltet ist, dass Syntaxfehler verhindert werden, können Fehler nur noch im Zusammenhang mit Symbolen auftreten:
Entweder wird durch einen Befehl ein Symbol aufgerufen, das nicht definiert wurde oder ein relativer Sprung ist so auf ein Label gerichtet, dass die maximale Sprungweite von +127 oder -128 überschritten wird. Es gibt also keine Fehlermeldungen.
Wenn ein Fehler entdeckt wird, wird der Assemblerlauf abgebrochen und direkt in die Eingabefunktion gesprungen (dies ist die einzige Funktion, die nicht immer ins Hauptprogramm zurückkkehrt).
Der Arbeitszeiger ist jetzt auf die Zeile gerichtet, in der der Fehler entdeckt wurde, man kann ihn also schnell beheben.
Wenn kein Fehler im Programm ist, wird der reine Maschinencoce automatisch an der Ablegeadresse (nicht Laufadresse) abgelegt und es erscheint "OK".
Anfänger im Programmieren mit Assembler sollten sich nicht durch die Möglichkeit erschrecken lassen, ein Programm an eine andere Stelle zu legen, als an die, wo das Proqramm laufen soll. Sie ist für Spezialfälle gedacht.
Im Zweifelsfall gilt immer: Laufadresse = Ablegeadresse.
Um zu zeiqen, wie einfach diese Funktion benutzt werden kann, ein Beispiel:
Basic-REM-Zeile mit der Zeilennummer 1 und vielen Blanks (oder irgendwelchen anderen Zeichen) eingeben - Assembler wieder starten - Ablege- und Laufadresse auf 16514 richten.
Wenn man jetzt ein geschriebenes Programm assemblieren lässt, wird es in diese REM-Zeile gelegt und ist dort lauffähig, kann also an der Adresse 16514 gestartet werden.
Diese Vorgehensweise ist für die meisten Fälle am sinnvollsten. Um bestimmte Programme zu schreiben, kann es aber wünschenswert sein, Lauf- und Ablegeadresse zu trennen.
Ein Beispiel: Sie wollen ein Programm schreiben, das später oberhalb des RAM-TOP laufen soll, wo jetzt z.B. der Assembler steht. Das ist ganz einfach möglich, indem Sie das Programm für irgendeinen anderen Speicherbereich im freien RAM schreiben. Und dort testen Sie das Programm auch aus. Lauf- und Ablegeadresse haben während dieser Zeit den Wert des Test-Speicherbereichs. Wenn das Programm fehlerfrei ist, geben Sie als Laufadresse die Anfangsadresse an, die Ihr Programm haben soll, die Ablegeadresse bleibt unverändert. Dann lassen Sie noch einmal assemblieren. Dadurch wird erreicht, dass das Programm jetzt an der ursprünglich beabsichtigten Stelle arbeitsfähig ist. Dort wird es aber nicht hingelegt, sondern wieder an die Stelle, an der es bisher getestet wurde. Es wird also nichts ungewollt überschrieben.
Es geht jetzt nur noch darum, diesen Maschinencode an seine Bestimmungsadresse zu bringen. Dazu braucht man noch ein Verschiebeprogramm. Aber zuerst empfiehlt es sich, das Listing und den Maschinencode abzuspeichern. B ist eine Spezialfunktion, die das Abspeichern von Maschinencode erleichtert.
Wenn man erreichen möchte, dass ein Listing nur überprüft aber nicht abgelegt wird, legt man als Ablegeadresse 0 fest, denn ins ROM kann bekanntlich nichts geschrieben werden.
Der Assembler läuft im Fast-Modus und kann mit "8" angehalten werden.

B.M-CODE:
REM-Zeilen haben für das Ablegen von Maschinencode einige Nachteile:
1. Man muss sie erst eintippen, was sehr mühselig sein kann.
2. Man muss immer darauf achten, dass die REM-Zeile mindestens so lang wie der Maschinencode ist.

3. Je nach Maschinencode kann so etwas wie ein ungewollter "Listschutz" entstehen, das Basic zeigt dann einfach grosse Teile des Basic-Programms nicht mehr an.
4. Wenn REM-Zeilen mit Maschinencode sehr lang werden, hängt sich das Bacic beim Listen sehr oft auf.

Diese Nachteile können durch die Funktion B vermieden werden, wobei man allerdings darauf verzichten muss, das Maschinenprogramm in eine Basic-Zeile zu schreiben.

Diese Funktion ersetzt das komplette Assemblerlisting durch den entsprechenden Maschinencode. D.h. die Variable A$, die bisher das Assembler-Listing enthält, wird mit dem entsprechenden Maschinencode überschrieben. Diese Variable wird ausserdem zu B$. A$ existiert dann nicht mehr.
Das funktoniert auch dann , wenn das Listing so gross geworden ist, dass es den gesamten Speicher belegt.
Wenn man ein Maschinenprogramm so ausgetestet hat, wie in Funktion A beschrieben, dann erzeugt diese Funktion den direkt abspeicherbaren Maschinencode.
Dieser Maschinencode muss nach dem Laden natürlich jeweils an die Stelle geschoben werden, für die er berechnet ist, deshalb empfiehlt es sich, an den Anfang eines Maschinenprogramms immer folgendes kleine Verschiebeprogramm zu schreiben:

```
     LD HL,($4010) 'Richtet HL auf 1.Variable (B$)
     INC HL        '
     LD C,(HL)     'Lädt in BC die Länge dieser Variablen
     INC HL        '
     LD B,(HL)     '
     LD DE,$0012   'Richtet HL auf 1. Programmbyte
     ADD HL,DE     '
     LD DE,LABEL   'Richtet DE auf Laufadresse
     LDIR          'Verschiebt das programm nach "LABEL"
     RET           'Rückkehr ins Basic
LABEL....          'Hier fängt das eigentliche Programm an.
```

Diese Verschiebefunktion lässt sich immer so aufrufen: RAND USR(PEEK 16400+256*PEEK 16401+3) und erspart eine zeitraubende PEEK- und POKE-Schleife. Dieses Programm funktioniert aber nur, wenn B$ die erste Variable im Variablenteil des Basic ist. Das lässt sich dadurch erreichen, dass man vor Neubeginn einer Programmeingabe erst CLEAR eingibt und dann ASDIS startet.
Bei der Verwendung von grossen Frei-Zeilen ist zu beachten, dass sie möglichst nicht ganz am Anfang des Listings stehen sollten, sonst könnte die Funktion B in Ausnahmesituationen zum Absturz führen. Im Zweifelsfall kann das immer dadurch verhindert werden, dass man vor einer Freizeile einen Kommentar einfügt, der so viele Zeichen enthält, wie durch die Frei-Zeile Bytes freigehalten werden.
Zur Sioherheit wird diese Funktion nicht sofort ausgeführt, sondern zeigt zunächst ein "?" und erwartet ein "J" zur Bestätigunq, andernfalls Return ins Hauptprogramm. Nach der Ausführung wird der Assembler automatisch verlassen.

C.RASS:
Mit dem Reverse-Assembler (umgekehrter Assembler) kann man Maschinencode, der irgendwo im Speicher steht, in ein Assemblerlisting umwandeln lassen. Man gibt die Adresse des 1. umzuwandelnden Befehls und die Menge der Befehle (nicht Bytes) an. Danach wird die entsprechende Zahl Befehle ins Listing aufgenommen. Alle vorkommenden 16-Bit-Zahlen werden durch Symbole ersetzt und überprüft, ob ein Symbol mit diesem Wert schon besteht. Wenn es nicht existiert, wird automatisch ein Label oder eine Marke erzeugt.

Datenbereiche werden von dieser Routine nicht erkannt. Der Reverse-Assembler erzeugt also keine Frei-, Daten- oder Text-Zeilen. Sollten solche Bereiche im Programm vorkommen, so muss man sie anschließend von Hand korrigieren.

D.DISAS:
Der Disassembler erwartet zunächst eine Anfangsadresse und zeigt dann 18 Befehle disassembliert an. Die Anzeige ist 3teilig: links Befehlsadresse, Mitte vollständige Mnemonik und rechts alle Codes aus denen der Befehl besteht. Eine Ausnahme machen die relativen Sprünge, denn es nützt ja nicht viel, wenn man die Weite eines Sprunges sieht, sondern man möchte sehen, wohin gesprungen wird, deshalb wird das Sprungziel berechnet und angezeigt.

Taste 5: Die oberen 18 Zeilen werden fliessend nach oben gescrollt und unten jeweils eine neue Zeile sichtbar.

Taste 6: Die nächsten 18 Befehle werden disassembliert.

Taste 7: Wenn man beim Disassemblieren etwas zu weit vorgerückt ist, möchte man wieder zurück, ohne eine neue Adresse einzutippen. Diese Funktion zieht einfach von der laufenden Adresse 70 ab und disassembliert die folgenden 18 Befehle. Man hat so die Möglichkeit, Maschinenprogramme bequem zu studieren.

Taste 8: Return ins Hauptprogramm.

Taste 9: Ausgabe an Drucker.

E.EINZL:
Der Einzelschrittbetrieb führt sämtliche Z80-Befehle fast ohne Absturzgefahr aus. Dazu unterscheidet das Programm intern zwei Befehlsarten:
1. Befehle die den Program-Counter (PC) ändern, werden in einer Spezialroutine simuliert (JP, JR, CALL, RET, usw.).
2. Alle anderen Anweisungen werden ausgeführt, nachdem ein Simulations-stackpointer sowie alle Registerinhalte neu geladen wurden. Nach der Ausführung werden die Registerinhalte wieder im Speicher abgelegt. Der Simulations-stack ist anfangs auf $FF80 (bzw. $7F80 bei 16K RAM) gerichtet, wo genügend Platz nach oben und unten ist, er kann aber auch an jede andere Stelle im freien RAM gerichtet werden.
In dieser Funktion sind immer sämtliche Registerinhalte und Flags sichtbar. Bei den Flags bedeutet "1" Flag gesetzt und "0" Flag gelöscht. Es wurden nur der Interrupt Vector (IV) und der Refresh Counter (RC) weggelassen, weil sie beim Sinclair kaum benutzt werden können, ohne dass die Hardware durcheinander gebracht wird.
Unterhalb des PC-Registers wird der 2. Registersatz des Z80 aufgelistet, auf den bekanntlich nur durch die Befehle EXX und EX AF,AF zugegriffen werden kann.

Bevor der Einzelschrittbetrieb beginnt, kann man die Inhalte aller Register festlegen. Um ein bestimmtes Register auszuwählen, bewegt man den Cursor mit den Tasten 6 und 7 auf und ab. Nach 5 oder Shift 8 (beliebig) kann das Register, neben dem der Cursor steht, verändert werden. Auch hier können (wie immer) Symbole statt Zahlen eingegeben werden. (Wenn man z.B. ein soeben assembliertes und abgelegtes Programm ab Label soundso testen will.)
Der Inhalt des PC entscheidet darüber, an welcher Stelle der Einzelschrittbetrieb aufgenommen wird. Sobald man die Registermanipulation (durch "8") verläßt, wird der Befehl noch nicht ausgeführt.
5: mit dieser Unterfunktion werden alle Befehle mit Ausnahme von CALL- und RST-Anweisung im Einzelschrittbetrieb ausgeführt. CALLs und RST-Anweisungen werden echt ausgeführt. Diese Funktion hat ihren Sinn im Testen rechenintensiver Programme, wo es aus Zeitgründen sinnlos wäre, sie im Einzelschrittbetrieb zu testen.

Unterprogramme, die man schon getestet hat und die mit CALL (oder RST) aufgerufen werden, kann man also auf einmal ausführen lassen, um schneller zur kritischen Stelle zu kommen, die man eigentlich testen möchte. Man sollte natürlich vor der Benutzung dieser Taste sicher sein, dass entsprechende Unterprogramme auch einwandfrei arbeiten.
6: Diese Taste bewirkt den eigentlichen Einzelschrittbetrieb. Sooft man diese Taste drückt, führt man eine Einzelschritt-Simulation aus. Der Bildschirm wird gelöscht, der ausgeführte und der nächste auszuführende Befehl werden disassembliert und die neuen Registerinhalte angezeigt.
7: Mit dieser Taste gibt man eine NOP-Anweisusng, also die Anweisung, den nächsten Befehel zu überspringen statt auszuführen. Das hat immer dann einen Sinn, wenn man einen normalen Programmablauf verlassen möchte, wie er durch Sprünge, CALLs oder RETs erzwungen würde oder wenn eine Anweisung nicht ausgeführt werden soll, weil sie z.B. wichtige Speicher (wie Stack oder System-Variablen) überschreiben würde.
Andere Beispiele für nicht ausführbare Befehle sind HALT, IM 0 und IM 2.
8: Rückkehr ins Hauptprogramm
9: Ausgabe an den Drucker

F.RUN:
Maschinenprogramme können sowohl aus dem Basic als auch aus dem Assembler aufgerufen werden. Es ist selbstverständlich, dass jedes Maschinenprogramm irgendeine Rückkehranweisung, also ein RET, enthalten muss. Bei Aufrufen aus dem Basic hat der Assembler keinerlei Einfluss auf den Programmlauf. D.h. die Registerinhalte zum Progammstart sind unbekannt und Programme können sowohl im Fast- als auch im Slow-Modus ausgeführt werden.
Der Programmstart aus dem Assembler bietet sich vor allem für das Austesten einzelner Routinen an, wenn es darum geht, anfangs Registerinhalte festzulegen - wie im Einzelschrittbetrieb - oder zu wissen, welche Registerinhalte der Prozessor nach Ende des Programmlaufs hat.
Zu Beginn der Funktion wird die Registereingabe gestartet. Der Inhalt des PC entscheidet über die Startadresse. Die Registereingabe wird durch '8' beendet, der erste ausführbare Befehl disassembliert und die Startanweisung "R" (RUN) erwartet. Wenn man stattdessen nochmals "8" eingibt, erfolgt die Rückkehr ins Hauptprogramm ohne Ausführung des Maschinencodes.
Sobald ein Maschinenprogramm aufgerufen ist, ist es für jedes Bit verantwortlich, mit dem es der Prozessor jetzt zu tun bekommt. Weder ROM noch Assembler haben irgendeine Einflussmöglichkeit bis zum RET. Es empfiehlt sich also, immer erst abzuspeichern.
Zur Programmausführung aus dem Assembler wird automatisch in den FAST-Modus umgeschaltet und der Bilsschirm gelöscht. Der Fast-Modus hat den Vorteil, dass alle, auch die Index-Rgister und der 2. Akku, verwendet werden können (solange man keine Routinen im ROM aufruft, die diese Register benötigen).
Nach Beendigung des Maschinenprogramms werden sämtliche Registerinhalte und Flags angezeigt, so wie sie vom ausgeführten Programm übergeben wurden.

G.VGL:
Mit der Vergleichsroutine kann man 2 Speicherblöcke auf Identität überprüfen. Es werden 2 Anfangsadressen erwartet und danach die Böcke Byte für Byte verglichen, bis ein Unterschied auftritt (bzw. der 1. Block die Adresse 0 erreicht). Bei einem Unterschied wird die entsprechende Adresse der beiden Blöcke dezimal und hexadezimal angezeigt und auf einen Tastendruck als Fortsetzungsanweisung oder Return ("8") gewartet.

H.LAUF:
Die bisherige Laufadresse wird angezeigt und kann geändert werden.

I.ABLEG:

Die bisherige Ablegeadressse wird angezeigt und kann geändert werden. Diese beiden Adressen brauchen nur einmal für jedes Programm festgelegt werden, wenn man es immer an derselben Stelle ablaufen lassen möchte.

J.DUMP:
Die Speicherinhalte des ZX81 werden als Hex-Dump ausgelistet, in dem man die Adresse eingibt und N/L drückt. Es werden auf 16 Zeilen 128 Bytes mit den Adressen angezeigt.
6: nächste 128 Bytes auslisten
7: vorhergehende 128 Bytes auslisten
8: Rückkehr ins Hauptprogramm
9: Ausgabe an den Drucker

K.MARKEN:
Alle Marken werden einzeln angezeigt und können gelöscht oder deren Wert geändert werden.
5: Markenwert ändern und nächste Marke anzeigen
6: keine Änderung und nächste Marke anzeigen
7: Marke löschen
8: Rückkehr ins Hauptprogramm

Eingabefunktion
Die Eingabefunktion ist so weit ausgebaut, dass man sie als ein Programm im Programm ansehen kann. Interaktiv ist sie, weil sie den Programmierer in folgenden Punkten unterstützt:
1. Sofortige Syntax-Prüfung, man kann also wie im Basic nur korrekte Zeilen eingeben.
2. Es ist immer eine Statusinformation sichtbar, die Hinweise darauf gibt, in welchen Zustand das Programm sich befindet bzw. welche Eingaben es erwartet.
3. Zeilennummern brauchen nicht eingegeben werden, die laufenden Aderessen werden automatisch berechnet und angezeigt.
4. An beliebiger Stelle kann man beliebig viele Zeilen einfügen oder löschen.
5. Den Arbeitszeiger kann man bequem auf jede Zeile richten.

Bildaufbau
Die Eingabefunktion hat einen speziellen Bildaufbau: der Bildschirm ist in 2 Teile aufgeteilt. Im oberen Teil von 16 Zeilen ist immer ein Teil des Assemblerlistings sichtbar. Er ist waagrecht wieder in 3 bis 4 Spalten gegliedert:
1. Die linken 4 Zeichen enthalten die jeweiligen Hex-Adressen der Zeilen.
2. Die nächsten 5 Zeichen rechts daneben sind der Symbolteil, hier werden eingegebene Labels angezeigt.
3. Bei Sonderzeilen zeigen die restlichen 23 Zeichen den eingegebenen Code an.
4. Bei normalen Assemblerzeilen ist der Codeteil nur 15 Zeichen lang (das genügt für die Darstellung sämtlicher Z80-Mnemoniks).
Die letzten 8 Zeichen zeigen den Hex-Code an, der den Mnemoniks entspricht.
In der 9. Zeile des oberen Teils sind die ersten 4 Zeichen immer invertiert dargestellt, um die Adresse der Zeile hervorzuheben, auf die der Arbeitszeiger gerichtet ist.
Der untere Bildteil von 8 Zeilen enthält den Raum für die Statusinformation und die Eingabe von Zeilen:
1. Das erste Zeichen der 17. Zeile enthält immer die Statusinformation
- einen invertierten Buchstaben.
2. Die nächsten 3 Zeichen sind ein Teil der Adresse, auf die der Arbeitszeiger gerichtet ist.
3. Die nächsten 5 Stellen sind der Symbolteil, in den man entweder ein Label schreibt oder die Angaben für eine Unterfunktion macht.
4. Der Rest der unteren 8 Zeilen kann für die Eingabe von Zeilen genutzt werden (Codeteil).

Verkürzte Darstellung
Sonderzeilen können also bis zu 8 Zeilen lang werden. Leider kann der ZX81 pro Bildzeile aber nur 32 Zeichen anzeigen. Um einen ungleichmässigen Bildaufbau zu verhindern, der dadurch entstehen würde, dass man auch Sonderzeilen, die mehr als 32 Zeichen benötigen, in voller Länge anzeigt, werden von allen Zeilen nur so viele Zeichen angezeigt, wie in einer Bildzeile Platz haben. Lange Zeilen werden also verkürzt dargestellt. Aber das Programm vergisst die restlichen Eingaben nicht. Man kann mit der Korrekturfunktion jede Zeile wieder in den unteren Bildteil holen, wo sie in voller Länge sichtbar und veränderbar ist.

Statusinformationen
Die möglichen Statusinformationen sind:

B Befehlszeile eingeben
C Codeteil
D Datenzeile eingeben
F Freizeile eingeben
G Grundzustand
K Kommentar eingeben
L Zeilen löschen
H Marke eingeben
R Arbeitszeiger rückwärts bewegen
S Symbolteil
T Textzeile eingeben
V Arbeitszeiger vorwärts bewegen
Z Arbeitszeiger auf bestimmte Zeile richten

Grundzustand der Eingabefunktion
Solange das invertierte "G" (Grundzustand) als Statusinformation sichtbar ist, hat man die Wahl, entweder durch einfachen Tastendruck eine Unterfunktion der Eingabefunktion aufzurufen oder in den Symbol- bzw. Codeteil zu verzweigen. Sobald
- eine Unterfunktion ausgeführt,
- eine Zeile eingegeben oder
- wenn ein Fehler entdeckt wurde,

kehrt das Programm automatisch wieder in diesen Grundzustand zurück.
Nur wenn eine Zeile eingegeben wurde, wird der Eingabeteil gelöscht. Der obere (Listings-) Teil wird durch Cursor-, Lösch- und Eingabefunktionen gesteuert. Wenn ein Fehler entdeckt wurde, wird das Bild überhaupt nicht geändert.

Symbolteil
Wenn im Grundzustand weder eine Ziffer noch ein Blank, "N/L" oder Strichpunkt gedrückt werden, erwartet das Programm eine Symboleingabe, was es durch die Statusinformation "S" anzeigt. Ausserdem erscheint das eingegebene Zeichen. Die Symboleingabe muss mit N/L abgeschlossen werden, worauf das Programm überprüft, ob das Symbol schon existiert und ob die Schreibweise korrekt ist. Sollte das Symbol schon zuvor definiert oder falsch geschrieben worden sein (siehe Symbole), dann kehrt das Programm in den Grundzustand zurück, andernfalls springt es in den Codeteil.
Man muss aber nicht immer Eingaben im Symbolteil machen, sondern kann ihn überspringen, indem man eine der 3 genannten Tasten drückt.
- "N/L" hat die Wirkung, dass der Inhalt des Symbolteils überprüft wird.
- Bei einem Druck auf Blank wird der Inhalt des Symbolteils gelöscht.
- Der Strichpunkt ermöglicht die Eingabe einer Kommentarzeile.

Codeteil
Sobald der Codeteil erreicht ist, erscheint die Statusinformation "C". Jetzt entscheidet der erste Tastendruck darüber, ob man eine Befehlszeile, Marke oder einen Arbeitsspeicher eingibt.
Mit Blank oder "N/L" kann man sofort wieder in den Grundzustand zurückkehren, mit dem Unterschied, dass "N/L" den Inhalt des Codeteils wie eine normale Befehlszeile betrachtet und entsprechend reagiert, während ein Blank sonst nichts bewirkt.

Unterfunktionen der Eingabefunktion
Im Grundzustand sind die Zifferntasten Funktionstasten.

0. Rubout. Die Zeile, auf die der Arbeitszeiger gerichtet ist, wird gelöscht.

1. Korrekturfunktion. Die Zeile, auf die der Arbeitszeiger gerichtet ist, wird im unteren Bildteil angezeigt und im Listing gelöscht. Man kann sie jetzt ändern oder auch unverändert wieder eingeben. Das Programm weiss aber nicht mehr, um was für eine Zeilenart es sich handelt, so dass man z.B. eine Befehlszeile in einen Kommentar umwandeln kann und umgekehrt.
Um zu verhindern, dass durch die Autorepeatfunktion aus Versehen mehrere Zeilen gelöscht werden, arbeitet die Korrekturfunktion nur, wenn das 1. Zeichen des Codeteils (in der 17. Zeile) ein Blank ist.

2. Löschen. Wenn die Statusinformation "L" sichtbar ist. kann man eine beliebige Anzahl Assemblerzeilen ab der Zeile löschen, auf die der Arbeitszeiger gerichtet ist.
Wenn man keine gültige Zahl oder 0 eingibt, wird nichts gelöscht.

3. Die Statusinformation "Z" erscheint und man kann den Arbeitszeiger auf eine beliebige Zeile richten, indem man deren Adresse (Label) angibt.

4. Bei Druck auf diese Taste wird der Arbeitszeiger um eine bestimmte Anzahl Zeilen vorgerückt. Bei Programmstart ist diese Zahl auf 16 eingestellt. Man kann sie aber beliebig ändern, indem man Shift und 4 drückt. Es erscheint die Statusinformation "V" und man kann angeben, in welcher Schrittweite ab jetzt vor- und rückwärts gerückt werden soll.

5. Die Taste 5 entspricht in ihrer Funktion der Taste 4, nur dass eben rückwärts gegangen wird. Die Statusinformation bei Shift 5 ist "R".

6. Der Arbeitszeiger wird um 1 Zeile vorwärts gerückt.

7. Es wird um eine Zeile zurückgegangen.

8. Rückkehr ins Hauptprogramm,

9. Ausgabe an den Drucker und Ausführung der Funktion 4. (Die Schrittweite wird dazu auf 16 eingestellt.)

Kommentar
Gibt man im Grundzustand ";" ein, verzweigt man damit in die Eingabe einer Kommentarzeile, was man an der Statusinformation "K" sieht. Kommentare sind selbständige Zeilen ohne Label, können bis 250 Zelchen lang sein und invertierte Zeichen enthalten. Sobald man "N/L" drückt, wird der untere Bildschirmteil bis zum letzten "Nicht-Blank" als Kommentar angesehen. Eine Kommentarzeile kann also nicht mit einem Blank aufhören oder aus lauter Blanks bestehen.

Kommentarzeilen dienen der Übersichtlichkeit des Listings und haben weder Einfluss auf die Zeilenadressen noch werden sie nach dem Assemblieren irgendwo abgelegt. In Listing sind Kommentare maximal bis zum 21. Zeichen sichtbar.

Befehlszeilen
Gibt man im Codeteil weder eine Ziffer noch ein Blank, Dollar-, Pfund-, Gleichheits- oder Anführungszeichen ein, dann erwartet das Programm eine Befehlszeile, was es durch die Statusinformation "B" anzeigt. Ausserdem wird das eingegebene Zeichen sichtbar.
Für die Eingabe der Mnemonik stehen die ersten 15 Zeichen hinter dem Symbolteil zur Verfügung. Dieser Platz reicht für jeden Befehl aus, da Daten und Symbole höchstens 5stellig sein dürfen, alle Eingaben linksbündig beginnen müssen und keine Rechenoperatoren erlaubt sind.
Die richtige Schreibweise der Befehle muss eingehalten werden, sonst wird die Zeile nicht akzeptiert und das Programm kehrt nach N/L in den Grundzustand zurück.
Jeder Befehl der Z80-Mnemonik besteht aus 1 bis 3 Kürzeln. Das erste bezeichnet immer den Befehl, die beiden anderen im allgemeinen Operatoren, mit denen gearbeitet wird. Hinter das 1. Kürzel muss auf jeden Fall ein Blank kommen. Wenn weitere Angaben erforderlich sind, muss hinter der letzten ebenfalls ein Blank stehen. Bei Verwendung von 2 Zusätzen müssen diese durch Punkt oder wahlweise Komma getrennt werden (beliebig).
Der Codeteil wird bei Befehlszeilen immer bis zum 2. Blank ab dem Symbolteil interpretiert.
Mnemonikteile, die eine Zahl darstellen, können immer durch Verwendung eines Symbols ersetzt werden. Obwohl Symbole immer 16-Bit-Zahlen darstellen, können sie auch für Befehle mit 8-Bit-Daten verwendet werden. Wenn z.B. das Symbol ZAHL den Wert $643F hat. dann wird der Assemblerbefehl "LD B,ZAHL" so assembliert, dass nur das Low-Byte (also hier $3F) eingesetzt wird.
Bei 8-Bit-Index-Befehlen dürfen in der Klammer keine Symbole vorkommen (Bsp.: LD (IX+DIST),34 geht nicht, aber LD (IX+0),ZAHL).
Eine Ausnahme machen die relativen Sprünge. Wenn man ein Symbol angibt (z.B. JR NC,ENDE), dann wird immer angenommen, dass es sich dabei um das Sprungziel handelt und die notwendige Sprungweite wird berechnet (Es wird also nicht - wie bei anderen 8-Bit-Daten - einfach die niederwertige Hälfte des Symbols eingesetzt). Ob die Sprungweite mit einem relativen Sprung möglich ist, oder ob ein absoluter Sprung nötig ist, zeigt sich, wenn man assemblieren lässt.

Hat man einen Befehl richtig eingegeben und die Eingabe mit "N/L" abgeschlossen, wird er sofort übersetzt und ins Listing eingefügt. Die sofortige Befehlscodierung hat einige Vorteile:
- Syntaxfehler werden sofort erkannt.
- Der eigentliche Assemblerlauf braucht nur noch die Symbole überprüfen und ist daher recht schnell (es handelt sich also genaugenommen um einen 3-Pass-Assembler).
- Fertig codierte Befehle belegen weniger Speicherplatz als die Mnemoniks (Bsp.: EX (SP),HL 10 Bytes --- E9 1 Byte)

Arbeitsspeicher
Die übrigen Sonderzeilen sind im Wesentlichen Hilfsmittel um Arbeitsspeicher einzurichten. Da auch Sonderzeilen Labels haben können, ist es möglich, sie ähnlich zu benutzen wie die Variablen im Basic.
Allerdings muss beim Assembler der Benutzer selbst die Länge der Variablen kontrollieren.

Frei-Zeile
Wenn man als erstes Zeichen im Codeteil ein "$" oder eine Ziffer eingibt, erscheint die Statusinformation "F" und das Programm erwartet eine Zahl von 1 bis 255.

Diese Eingabe ist vergleichbar mit dem Basic-Befehl DIM, mit dem Unterschied, dass beim DIM-Befehl der entsprechende Speicherbereich mit Nullen angefüllt wird, während der Assembler nur einen Speicherbereich freihält, ohne ihn zu verändern. Im Listing steht in der entsprechenden Zeile die dezimale und hexadezimale Länge des eingerichteten Arbeitsspeichers. Beim Assembler sind Arbeitsspeicher nicht an einen bestimmten Datentyp (Text oder Zahlen) gebunden, es liegt im Ermessen des Programmierers, wie die Speicher benutzt werden. Im folgenden Anwendungsbeispiel werden so viele Bytes wie VAR2 angibt, von der Stelle, die VAR1 angibt, in den Arbeitsspeicher VAR3 geschoben. Nach der Ausführung lässt sich das Ergebnis mit der Dump- oder Disassembler-Funktion ab "VAR3" betrachten.

```
5000 LABEL LD HL,(VAR1) 2AOF50
5003       LD DE,VAR3   111250
5006       LD A,(VAR2)  3A1150
5009       LD C,A       4F
500A       LD B,0       0600
500C       LDIR         EDBO
500E       RET          C9
500F VAR1  002-$02 (2-Byte-Variable)
5011 VAR2  001=S01 (1-Byte-variable)
5012 VAR3  255=$FF (255-Byte-Arbeitsspeicher)
```

Daten-Zeile

Es ist aber auch möglich, einen Arbeitsspeicher zu definieren und gleichzeitig festzulegen, was drin stehen soll, indem man entweder das Pfund- (Daten) oder das Anführungszeichen (Text) drückt. Diese Zeilen entsprechen dann den Basic-Statements LET X = ... und LET X$ = "...".
Nach dem Pfundzeichen erscheint die Statusinformation "D" (Datenzeile) und es kann eine Zeile mit bis zu 246 Hexzeichen ("0" ... "F") eingegeben werden (also 123 Bytes), die keinerlei Sonderzeichen wie Blanks oder "$" enthalten darf. Die Zeile wird nicht akzeptiert, wenn man andere Zeichen oder eine ungerade Anzahl von Zeichen eingibt.
Diese Zeilenart kann man nicht nur als Variablendefinition, sondern auch als Pseudo-Hexmonitor benutzen, wenn man z.B. einen bestimmten Maschinencode eintippen und an eine bestimmte Stelle legen möchte. Es ist ja möglich, beliebig viele Datenzeilen aneinander zu hängen und mit der Ablegeadresse zu entscheiden, wo dieser Code dann hingelegt werden soll. Im Listing sind von solchen Zeilen jeweils max. 10 Bytes sichtbar.

Text-Zeile

Nach den Anführungszeichen erscheint die Statusinformation "T" und man kann eine 246 Zeichen lange Zeile eingeben, wobei alle Zeichen erlaubt sind, die über die Tastatur erreichbar sind. Wie bei Kommentaren wird die Zeile bis zum letzten Nicht-Blank als gültig angesehen. Im Listing sind von Textzeilen max. 21 Zeichen zu sehen.
Folgendes Programm, das eine Text- und eine Datenzeile verwendet, löscht zuerst den Bildschirm (durch Aufruf der CLS-Routine, deren Adresse man mittels einer Marke definieren muss) und schreibt (mit Hilfe des RST 10) den Text, der im Arbeitsspeicher VAR2 steht, auf den Bildschirm in der Länge, die in VAR1 angegeben ist.

```
5000 LABEL  CALL CLS        CD2A0A
5003        LD HL,VAR2      211250
5006        LD BC,(VAR1)    ED4B1050
500A LOOP   LD A,(HL)       7E
500B        RST 10          07
500C        INC HL          23
500D        DJNZ LOOP       10FB
500F        RET             C9
```

```
5010 VAR1   0012 (Datenzeile)
5012 VAR2   DIES IST EIN TEXT. (Textzeile)
5024 ;KOMMENTAR
```

Rufen Sie dieses Programm aus dem Basic auf.

Marken

Ein Gleichheitszeichen zu Beginn des Codeteils ermöglicht die Definition einer Marke. Dazu muss natürlich im Symbolteil eine Eingabe gemacht worden sein. Es erscheint die Statusinformation "M" und eine Zahl wird erwartet. Wenn man eine Zahl eingibt, ist damit eine Marke definiert, die sich z.B. als Konstante oder externes Label (wie im obigen Beispiel CLS) benutzen lässt.