DECKER & COMPUTER

POSTFACH 967, 7000 STUTTGART 1, TEL 0711/225314

ZX-ASZMIC

NEUES DEUTSCHES HANDBUCH

ZX-ASZMIC

SEMBLY LANGUAGE
VELOPMENT SYSTEM

ISBN 3-89191-023-1

ZX-ASZMIC

HARTUNG GORRE VERLAG
KONSTANZ
ISBN 3-89191-023-1

File-Name : ASZMIC.MAN

Titel : Neues deutsches Handbuch fuer das ZX-ASZMIC-ROM

Quelle : 7.7.85 Last entry : 17.7.85

Dieses Handbuch wurde erstellt fuer das ZX-ASZMIC-ROM der Firma Comprocsys Ltd., England. Dieses ROM ist das geistige Eigentum von Herrn C. Frazer Johnson, vertreten durch seine Firma Comprocsys Ltd., England.

Das ZX-ASZMIC-ROM ist ein Betriebssystem mit Editor, Assembler und Debugger. Dieses Handbuch soll den Besitzern des ZX-ASZMIC-ROMs die Funktionen und die Handhabung des ROMs erklaeren.

Die Rechte fuer dieses neue, deutsche Handbuch liegen ausschliesslich bei Aribert Deckers, Stuttgart.

Die Rechte fuer das englischsprachige Original-Handbuch fuer das ZX-ASZMIC-ROM liegen ausschliesslich bei Comprocsys Ltd., England.

Die Rechte fuer das im Anhang nachgedruckte "Z80-Micro-Reference-Manual" liegen bei der MOSTEK Corporation, Carrolton, Texas, USA, bzw bei der deutschen UTC MOSTEK GmbH, Neuhausen.

Wir danken der Firma MOSTEK fuer die Genehmigung zum Nachdruck des "Z80-Micro-Reference-Manual".

S E R V I C E

Vertrieb, Beratung und Service erfolgen durch die Firma

Decker & Computer
Postfach 967
D-7000 Stuttgart -1
Telefon : 0711/225314

Es ist unser Ziel, gut verstaendliche und fehlerfreie Handbuecher und Anleitungen zu liefern. Aber trotz aller Muehe ist es leider fast unmoeglich, ein voellig fehlerfreies Buch zu schreiben.

Wenn Sie also einen Fehler oder eine unverstaendliche Stelle finden :

Rufen Sie bitte an oder schreiben Sie uns. (Bitte frankierten Rueckumschlag beifuegen !)

Wir werden dann versuchen, die betreffenden Stellen zu korrigieren bzw den Sachverhalt besser zu beschreiben. Wir danken Ihnen fuer Ihre Hilfe.

INHALT

KAPITEL I

ZX-ASZMIC-ROM in Deutschland

Die Firma Decker & Computer, Stuttgart, hat vom englischen Hersteller Comprocsys Ltd. den Vertrieb fuer das ASZMIC-ROM in Deutschland erhalten. Weil vom Hersteller nur ein englisches Handbuch zur Verfuegung stand und es in deutscher Sprache bislang nur das Handbuch der bisherigen Vertriebsfirma gab, entschloss man sich, ein deutsches Handbuch vollstaendig neu zu erstellen.

Bis jetzt waren einzelne Handbuecher - also ohne das ROM - weder beim Hersteller noch ueber die Vertriebsfirmen erhaeltlich. Um aber die sehr grosse Zahl der bisherigen Benutzer zu unterstuetzen, denen nur das alte deutsche oder das englische Handbuch zur Verfuegung stand, wird das neue Handbuch auch einzeln verkauft !

Um Ihnen einen Ueberblick ueber die neue Preisgestaltung zu geben, hier ein Auszug aus der Preisliste :

ZX-ASZMIC : ROM ohne Handbuch ..... DM 80,--
ZX-ASZMIC : ROM mit neuem deutschem Handbuch ... DM 100,--

neues Handbuch : einzeln ..... DM 30,--
neues Handbuch : im Tausch gegen altes Handbuch ..... DM 20,--
(englisch oder deutsch)

Diese Preise gelten zuzueglich einer Versandpauschale :
bei Vorkasse : DM 6,-- , bei Nachnahme : DM 9,--

Es ist das erklaerte Ziel der Firma Decker & Computer, die Besitzer der Computer ZX 80 und ZX 81 durch hochwertige Hardware, Programme und Handbuecher zu unterstuetzen.

KAPITEL II

Was ist der ASZMIC ?

Der ZX-ASZMIC ist ein ROM, das anstelle des BASIC-ROMs in den ZX 81 eingesetzt wird. Er enthaelt nicht nur einen Assembler, sondern ein vollstaendiges Betriebssystem mit einem Bildschirm-orientierten Editor. Damit kann man nicht nur Maschinen-Code-Unterprogramme programmieren (die mit "REM-Zeilen" in BASIC-Programme eingebunden werden), sondern auch komplizierte Steuerungen direkt in Assembler programmieren und anschliessend als reinen Maschinen-Code ausfuehren lassen.

Zum Lernen der Maschinen-Programmierung und der Z80-Assemblersprache ist das ASZMIC-ROM ein preiswertes Hilfsmittel : Man benoetigt nur den billigsten Rechner, den es auf dem Markt gibt : den ZX 81 oder einen ZX 80 mit Slow-Modus.

Eine kleine Uebersicht der Eigenschaften des ASZMIC-ROMs

* 35 Zeilen zu je 36 Zeichen
* bildschirmorientierter Editor
* verwendet Zilog-Mnemonics
* Kontrolle der Symbol-Tabelle
* Dump, Modify, Fill, Copy
* Debug verwendet Programmnamen
* Register anzeigen und aendern
* Programm-Verschiebung
* Kommando- und Editor-Macros
* Repeat-Funktion auf allen Tasten
* 18 Funktionstasten zum Editieren
* benutzt normalen Sinclair-Drucker (32 oder 64 Zeichen pro Zeile)
* Breakpoints
* automatisches Dumpen bei Break
* Vorwaerts-Referenzen und Externals
* Listings und Fehler-Flags
* benutzt normale Sinclair-Aufzeichnung
* Files koennen gemischt werden (merge)
* ORG, EQU, DEFB, DEFW, DEFM Anweisungen
* Single-Step (Einzelschritt)
* interpretierender Assembler
* direkte Ausfuehrung eines Programms
* Inhaltsverzeichnis (Katalog) der Cassette
* Hochaufloesende Grafik 255x144 Punkte
* Routinen zur Unterstuetzung des Monitors

Dieses Handbuch wurde geschrieben, um Ihnen alles zu erklaeren.

KAPITEL III

Wie baue ich das ASZMIC-ROM ein ?

Wenn Sie einen ZX80 haben, koennen Sie das ASZMIC-ROM ebenfalls verwenden (Version E04), unterliegen aber einer Reihe von Einschraenkungen. Wenden Sie sich bitte sofort an Decker & Computer !

Der ZX81 hat sein BASIC in einem ROM, das darum auch BASIC-ROM heisst. Dieses ROM muss ausgebaut werden. An seine Stelle ist der ASZMIC einzusetzen.

Einbau in einen ZX 81

- Als erstes schalten Sie den Rechner und das angeschlossene Zubehoer ab.
- Dann entfernen Sie alle Anschluesse (Kabel und Steckanschlusse) vom Rechner, so dass Sie diesen oeffnen koennen.
- Drehen Sie den Rechner auf den Ruecken : Sie sehen im Boden des Rechners mehrere Loecher. Wenn noch nicht allzuviel mit dem Geraet geschehen ist, befindet sich in jedem Loch eine Schraube. Drehen Sie jede dieser Schrauben heraus.
- Nehmen Sie die Bodenplatte des Rechners ab : jetzt sehen Sie die Platine von unten. Diese Platine traegt das BASIC-ROM.
- Loesen Sie die Schrauben, mit denen die Platine am Deckel des Rechner-Gehaeuses befestigt ist. Seien Sie dabei sehr vorsichtig, weil die Tastatur-Anschluesse (ein flaches, milchig aussehendes Plastik-Kabel) sehr empfindlich sind und leicht reissen.
- Drehen Sie die Platine um (achten Sie auf die Kabel !).

- Jetzt kommt der Augenblick der Wahrheit : Sie sehen 2 grosse ICs (mit je 40 Fuessen). Das eine ist die ULA, das andere der Prozessor.

  Fall a) Sie haben einen ZX 81 mit einer gruenen Platine

  Zwischen ULA und Prozessor, in Richtung auf den Kuehlkoerper hin zum Platinenrand verschoben, ist ein 24-poliges IC. Das ist das BASIC-ROM. Frage : Steckt das ROM in einem IC-Sockel (und kann herausgezogen werden) oder ist es eingeloetet ?

  Fall b) Sie haben einen ZX 81 mit einer roten Platine (Timex)

  Hier ist der Prozessor gleich neben der ULA und das ROM befindet sich zwischen Prozessor und den 2 Steckern fuer den Anschluss der Tastaturfolien.

  Fuer Fall a) und Fall b) :

  1) Das ROM ist eingeloetet : loeten Sie es aus, saeubern die Platine und loeten einen 28-poligen IC-Sockel ein.
     Wenn Sie das ROM nicht selbst ausloeten koennen, bitte bei Decker & Computer melden !

  2) Das ROM kann herausgezogen werden : Achten Sie auf die Kerbe an einer der beiden Stirnseiten des ROMs. Sie zeigt in Richtung der beiden 40-poligen ICs. Diese Kerbe ist wichtig :

Das ASZMIC-ROM hat ebenfalls eine Kerbe und die muss beim Einsetzen in genau die gleiche Richtung zeigen.

3) Sonderfall : Es ist kein 24-poliges ROM, sondern ein 28-poliges EPROM (so aehnlich, wie das ASZMIC-ROM) :

Bitte melden Sie sich unbedingt bei Decker & Computer !

- Kleiderordnung : Sofern Sie Kleidung aus Kunstfaser oder Wolle tragen, ist es jetzt Zeit, auf Baumwolle zu wechseln.

- Stellen Sie Ihr Werkzeug, den Rechner und das ASZMIC-ROM auf eine Unterlage aus Aluminium (Kuechenfolie) oder einem anderen Metall (zum Beispiel Werkbank). Dadurch haben alle benoetigen Teile das gleiche elektrische Potential.

- Entfernen Sie das BASIC-ROM. Dazu nehmen Sie einen duennen Schraubenzieher, den Sie zwischen IC-Sockel und ROM schieben. Hebeln Sie das ROM von beiden Seiten (!) vorsichtig heraus. Achten Sie besonders auf die Beinchen, die zuletzt herauskommen. Diese verbiegen sich besonders leicht (wegen der grossen Hebelwirkung), wenn man beide Seiten ungleich herausdrueckt.

- Nehmen Sie das ASZMIC-ROM vorsichtig aus seiner Verpackung. Es hat 24 Fuesse. Setzen Sie das ROM in den IC-Sockel des BASIC-ROMs. Jene Stirnseite, die keine (!) Kerbe hat, muss zum nahen Platinenrand hin zeigen. Die Fuesse dieser Stirnseite sind buendig mit dem Sockel einzusetzen : Wenn der Sockel 28 Fuesse hat, bleiben also ueber der Stirnseite mit der Kerbe 4 Loecher frei !

- Nachdem Sie das ASZMIC-ROM eingesetzt haben, koennen Sie den Rechner wieder zusammenbauen : Platine an den Gehaeuse-Deckel schrauben und dann Boden aufsetzen und festschrauben.

KAPITEL IV

Die ersten Versuche zum Editieren

Nach dem Einschalten :

Schliessen Sie Ihren Computer an den Fernseher und das Netzteil an. Schalten Sie den Rechner ein. Links unten auf dem Schirm sollten Sie jetzt ein kleines Zeichen sehen. Darueber blinkt der Cursor. Das kleine Zeichen heisst END-OF-DATA und bedeutet, dass hier die Daten zuende sind. Die Bedeutung dieses Zeichens werden wir spaeter erklaeren.

Die Frequenz ("Geschwindigkeit") des Blinkens gibt an, in welchem Zustand (Modus) sich der ASZMIC befindet :

schnelles Blinken : EDIT-MODUS

langsames Blinken : DEBUG-MODUS

Ist ein 16k-RAM angeschlossen, beginnt der ZX-ASZMIC im DEBUG-MODUS.

Ist kein RAM angeschlossen (haben Sie nur das interne 1k RAM zur Verfuegung), sind Sie automatisch im EDIT-MODUS.

Um zwischen diesen Betriebsarten umschalten zu koennen, gibt es Befehle :

SHIFT 9 und SHIFT E

DEBUG EDIT

Probieren Sie das, indem Sie beide Befehle abwechselnd geben und sich ansehen, was geschieht.

Es gibt nur einen Unterschied zwischen beiden Modi - und der ist wichtig !

- Wenn Sie im DEBUG-Modus <Newline> eingeben (das schreiben wir auch als /NL/), wird die damit beendete Zeile dem Kommando-Interpreter zur Ausfuehrung uebergeben.

- Wenn Sie dagegen <Newline> im EDIT-MODUS eingeben, wird damit lediglich die Zeile beendet und Sie befinden sich automatisch am Anfang der naechsten.

Das Editieren :

Druecken Sie

SHIFT E

und gelangen damit in den EDIT-MODUS. Der Cursor blinkt schnell. Nun tippen Sie

A

Ein "A" erscheint auf dem Bildschirm und der Cursor spaziert einen Schritt nach rechts. Tippen Sie nochmals

A

und halten die Taste gedrueckt. Nach rund einer halben Sekunde wird der ASZMIC leuter "A"s auf den Schirm schreiben (pro Sekunde 8 mal). Nehmen Sie den Finger von der Taste, bevor die Zeile vollgeschrieben ist.

Wie Sie gerade gesehen haben, ist eine automatische Repeat-Funktion eingebaut.

Geben Sie jetzt <Newline> ein, druecken dann "B" fuer rund 2 Sekunden, druecken <Newline>, druecken dann "C" fuer mehrere Sekunden und schliessen mit <Newline> ab. Machen Sie so weiter, bis Sie 5 oder 6 Zeilen auf dem Schirm sehen.

Der Inhalt des Bildschirms wird bei jedem <Newline> um eine Zeile nach oben geschoben.
Geben Sie nun eine Zeile ein, die Sie aber nicht mit <Newline> beenden. Der Cursor steht hinter dem letzten Zeichen. Druecken Sie

SHIFT 5 (Cursor nach links)

Wenn Sie die Tasten gedrueckt halten, wandert der Cursor wandert links, bis zum Beginn der Zeile. Druecken Sie dann

SHIFT 8 (Cursor nach rechts)

Hier bewegt sich der Cursor bis zum Ende der Zeile.

Bewegen Sie den Cursor so, dass er irgendwo in der Mitte der Zeile steht. Geben Sie nun

123456

ein. Die Ziffern werden ab Cursorposition in den Text der Zeile eingefuegt. Genau betrachtet, verschiebt der Rechner alle Zeichen rechts vom Cursor um eine Position nach rechts und setzt an den nunmehr freien Platz das neu eingegebene Zeichen ein. Dann wird der Cursor um eins nach rechts gestellt.

Loeschen (Rubout)

Druecken Sie

SHIFT O

Das ist Loeschen (Rubout) : die "6", die Sie eben noch eingefuegt hatten, verschwindet wieder und die Zeile wird um eine Stueck kuerzer. Beachten Sie, dass hierbei das Zeichen links (!) vom Cursor geloescht wurde.

Stellen Sie den Cursor jetzt auf die "3" und druecken

SHIFT Q

Das loescht die "3". Man kann also nicht nur das Zeichen links, sondern auch jenes unter dem Cursor loeschen !

Die Verwendung dieser oder jener Loeschmethode haengt ganz von Ihnen ab. Es ist Geschmackssache und eine Frage der Uebung, ob man nun diese oder jene benutzt.

Achtung : SHIFT Q funktioniert nicht, wenn nur noch ein Zeichen in der Zeile uebrig ist.

<Newline> (Zeilenvorschub) kann man mit SHIFT O loeschen.

Pferdefuss : SHIFT Q kann auch die "Fueller" vor einem <Newline> loeschen, die der ASZMIC dort automatisch einfuegt. Das kann zu Schwierigkeiten beim Assemblieren und Mergen fuehren !

Vertikale Bewegungen des Cursors

Machen Sie einen RESET am Computer. Dazu druecken Sie auf die Reset-Taste. Wenn Sie sich noch keine eingebaut haben, dann wird's hoechste Zeit ! (Bis dahin : Schalten Sie kurz die Stromversorgung aus und wieder ein.)

Das Resetten werden Sie noch oft brauchen, da es die einfachste Moeglichkeit ist, den Speicher ganz schnell zu loeschen.

Druecken Sie SHIFT T

"T" steht hier fuer TOP (also "oben"). Der Cursor und die ganze Anzeige werden an den Anfang springen. Das ist nuetzliche, wenn Sie laengere Texte durchgehen wollen. Das wichtigste fuer uns ist aber, dass die oberste Zeile eine ganz besondere Bedeutung hat : es handelt sich um die MACRO-Zeile (mehr darueber spaeter).

Druecken Sie nun

SHIFT 9

Das ist - Sie wissen es schon - die Umschaltung in den DEBUG-MODUS. Sie sehen, dass Sie ploetzlich am Fuss des Bildschirms sind und sich im DEBUG-Modus befinden. Druecken Sie

SHIFT E

und Sie sind nicht nur wieder im EDIT-Modus, sondern haben auch wieder genau jene stelle erreicht, an der Sie standen, als Sie mit "SHIFT 9" umgeschaltet hatten.

Sie koenen also zwischen EDIT und DEBUG hin- und her-schalten, ohne lange nach jener Stelle suchen zu muessen, an der Sie sich zuvor befunden hatten.

Gehen Sie in den DEBUG-Modus mit "SHIFT 9". Druecken Sie

D 0 <Newline>

Der ASZMIC wird Ihnen mit

0000 F5

antworten. Halten Sie Ihren Finger solange auf der <Newline>-Taste, bis ASZMIC mit

0030 3E

geantwortet hat. Geben Sie jetzt ein :

.<Newline>

Das ist ein Punkt mit anschliessendem <Newline>. Und nun : wir sind wieder im EDIT-Modus. Punkt-Kommandos sind sehr wichtig. Achten Sie also darauf !

Was Sie eben im DEBUG gemacht haben ? Sie haben dem Rechner befohlen, Ihnen die ersten 49 Adressen (im Adressbereich des Prozessors) anzuzeigen. Das nennt man "DUMP". Frei uebersetzt koennte man sagen : "ausschuetten". Ein "dump" ist uebrigens auch eine Muellhalde - bei unserer weiteren Arbeit wird der Rechner sich auch oft als solche erweisen ...

Druecken Sie jetzt

SHIFT 7

und schieben damit den Cursor hoch. Wenn Sie die Taste laenger druecken, erreicht der Cursor schliesslich die oberste Zeile. Danach wird Text von oben nach unten gescrolled (geschoben). Sie koennen in einem Text also auch von unten nach oben laufen : der nicht sichtbare Text wird automatisch nachgeschoben und auf dem Bildschirm sichtbar.

Kehren Sie um und druecken dazu

SHIFT 6

Jetzt laeuft der Cursor erst in die untereste Zeile, bevor der Rechner mit dem Hochschieben des Texts beginnt.

Hat der Cursor die letzte Zeile erreicht - und schliesslich das END-OF-DATA-Zeichen, bleibt er stehen.

Druecken Sie

SHIFT 4

und der Text wird nicht Zeile fuer Zeile durchlaufen, sondern Seite fuer Seite. "SHIFT 4" ist der Befehl fuer "Page up" ("1 Seite nach oben").

Das Gegenstueck ist

SHIFT 3

"Page down" ("1 Seite nach unten").

Das Loeschen einer ganzen Zeile

Stellen Sie den Cursor auf 0020 7E (oder so). Druecken Sie nun

SHIFT 1

("Delete line"). Die Zeile wird verschwinden. Wiederholen Sie das und Stueck fuer Stueck werden die Zeilen verschwinden - bis Sie wieder ganz unten sind : in der Zeile vor dem END-OF-DATA-Zeichen. Diese Zeile koennen Sie nicht mit "SHIFT 1" loeschen. Sie muss Zeichen fuer Zeichen geloescht werden. Das dient zum Schutz des END-OF-DATA-Zeichens. Das ist naemlich das Signal, das hier den DUMP stoppen muss.

Druecken Sie

SHIFT 2

Jetzt ist der ganze Text vom ersten bis zum letzten Zeichen geloescht.

Files und File-Markierungen (File-Marks)

Das BASIC-ROM erzeugt fuer die Anzeige des Bildinhalts auf dem Fernseher oder Monitor ein sogenanntes "Display-File". Alles, was in diesem Display-File ist, wird angezeigt. Um etwas anzeigen zu koennen, muss es in das Display-File transportiert werden.

Der ASZMIC arbeitet anders : er betrachtet den groessten Teil des RAMs als Text. Darueber kann er ein Fenster verschieben und darin immer einen Ausschnitt auf dem Bildschirm anzeigen. Ein grosser Teil der Operationen werden mit dem Pfund-Zeichen ^. Dieses Zeichen ist ein Trenner :

> Sind zum Beispiel mehrere Textbloecke im RAM, zwischen denen jeweils das ^ steht, dann werden die Bloecke dadurch von einander getrennt und vom ASZMIC als einzelne Files betrachtet.

Weil Textbloecke als Files behandelt werden, heisst ^ auch Filemark oder Filemarke. Die Filemarke an sich bedeutet das "Ende eines Files" und wird benoetigt, zum Drucken, Assemblieren, Saven, Mergen und das Loeschen von Files. Die Filemarke gibt dann jeweils an, dass der betreffende Block beendet ist und die laufende Operation beendet werden muss.

Der Anfang eines Files wird mit einem Namen markiert. Dieser Filename gilt fuer diesen besonderen Textblock und darf deshalb auch nicht im Textblock selbst vorkommen. Es ist daher aeusserst sinnvoll, die Filemarke als erstes Zeichen des Namens zu verwenden (dann erscheint der Name im Text kein 2. Mal).

Beispiel :

```
^MEIN.COMPUTER
1
2
MEIN COMPUTER IST DER EINZIGE
AUF DER WELT, DER SICH NICHT
IRREN KANN.
3
4
ABER DAFUER LUEGT ER.
^
```

Das ist ein File, das man editieren kann, saven, drucken und so weiter. Dazu muss man den Namen ^MEIN.COMPUTER angeben.

Ein Filename kann bestehen aus Ziffern (0 bis 9), Buchstaben (A bis Z) und dem Punkt. Ein Filename hoert auf mit irgendeinem anderen Zeichen.

Setzen Sie den Cursor auf die HOME-Position : Das ist jene Stelle, an die Sie kommen, wenn Sie in den DEBUG-Modus umschalten. Druecken Sie

SHIFT 9 ^ <Newline>

Nun gehen Sie in den EDIT-Modus (na, wie geht das noch ?) und bewegen den Cursor, bis er auf dem D Ihres Befehles "D 0" in der Zeile am Anfang des DUMPs blinkt.

Dann druecken Sie

SHIFT 2

Der Platz geht aus

Machen Sie einen Reset und druecken

SHIFT 9

Das bringt Sie in den DEBUG-Modus. Jetzt geben Sie ein :

D 0 5000 <Newline>

Der Bildschirm wird fuer mehrere Sekunden verschwimmen : der ASZMIC bereitet 625 Zeilen eines formatierten DUMPs vor. Um diese Menge Text unterzubringen, brauchte er aber mehr als 16k RAM - selbst wenn der ASZMIC den Speicher nicht in 3 Teile fuer den Text und 1 Teil fuer Programme zerlegen wuerde.

Warten Sie, bis der Bildschirm wieder klar wird : Versuchen Sie, ein Zeichen einzugeben. Nichts geschieht. Sie koennen mit dem Cursor umherfahren (probieren Sie es !), aber keinen neuen Text einfuegen.

Um weitermachen zu koennen, muss erst Text geloescht werden : Bringen Sie den Cursor 4 oder 5 Zeilen aufwaerts und halten dann "SHIFT 1" laenger gedrueckt. Damit werden die letzten Zeilen eliminiert. Danach koennen wieder Zeichen eingegeben werden (ca 100) - bis der freie Platz wieder aufgebraucht ist.

Macros

Diese Riesenmenge Text koennen wir auch fuer etwas gebrauchen. Druecken Sie

SHIFT T

Das schafft den Cursor in die oberste Zeile. Nun

E 40

tippen. Das darf aber nicht mit <Newline> beendet werden : Weil wir uns im DEBUG-Modus befinden, wuerde das als Kommando betrachtet und sofort aus efuehrt. Statt dessen stellen Sie den Cursor in die Fusszeile :

SHIFT 9

Dann kommt

SHIFT R

(SHIFT Macro).

Damit wird die oberste Zeile als DEBUG-Befehl betrachtet und ausgefuehrt - unabhaengig davos, in welchem Modus wir uns befinden !

"E 40" ist ein DEBUG-Befehl, um in den EDIT-Modus zu gelangen und den Cursor auf den Beginn des ersten Auftretens der Zeichenkette "40" stellen. Das heisst : Der Cursor erscheint auf der "4" von "40" und blinkt schnell.

Druecken Sie "SHIFT R" mehrfach und erleben, wie jedes Auftreten der "40" im Text herausgesucht wird.

Sie koennen fast jeden DEBUG-Befehl oder eine Verkettung mehrere solcher Befehle in die oberste Zeile schreiben und mit einem einzigen

von Makro, der [illegible] ASZMIC eingebaut ist - er wird uns beim Mergen begegnen.)

Loeschen Sie den DUMP mit

SHIFT 9 ^ <Newline>

(^ ist kein gueltiger DEBUG-Befehl und wird vom Kommando-Interpreter ignoriert.) Das reicht noch nicht (warum ?) - also druecken Sie

SHIFT T

und halten dann

SHIFT 6

unten, um das "D" von "D 0 5000" zu erreichen. Jetzt mit "SHIFT 2" alles loeschen (Befehl zum Loeschen eines Files).

Mergen

Machen Sie einen Reset und gehen mit "SHIFT E" in den EDIT-Modus. Geben Sie ein :

```
>D 0 1 <Newline>
D 0 2 <Newline>
D 0 3 <Newline>
D 0 4 <Newline>
D 0 5 <Newline>
^<Newline>
```

Nun druecken Sie einmal

SHIFT G

(Mergen). Alles zwischen dem ">" und dem "^" ist verdoppelt worden. Versuchen Sie es noch einmal : Bei jedem "SHIFT G" werden an der Cursor-Position 5 neue Zeilen als Kopie. Stellen Sie den Cursor auf das "D" eines der Befehle "D 0 3" und druecken dann "SHIFT G". Die 5 Zeilen sind nun vor dem "D 0 3" eingefuegt, weil der Cursor sich genau dort befunden hatte.

Druecken Sie jetzt "SHIFT 9" fuer DEBUG. Im EDIT-Modus ist "SHIFT G" ein einfacher Befehl zum Mergen. Hier, im DEBUG-Modus, hat er eine ganz andere Wirkung : Jedesmal, wenn der ASZMIC im DEBUG-Modus ein <Newline> in den Textbereich schreibt, wird die Befehlsgewalt automatisch dem Kommando-Interpreter uebergeben. Sehen wir uns das einmal an : Druecken Sie

SHIFT G

Jede Zeile mit dem "D" wurde kopiert, aber weil jede dieser Zeilen ein Befehl zum DUMPen ist, wurde nach der Zeile dieser DUMP auch ausgegeben.

Mit dem Druecken einer Taste werden 5 Zeilen mit DEBUG-Anweisungen ausgefuehrt. Das Ganze nennt man einen Macro-Befehl.

Im EDIT-Modus wird das Mergen ueblicherweise benutzt, um Textstuecke von einer Stelle an eine andere zu kopieren. Dazu werden die Passage mit dem Keil ">" und dem Pfund "^" markiert. Anschliessend stellt man den Cursor an die gewuenschte Position und kopiert dorthin. Will man den Text aber nicht kopieren, sondern verschieben, dann muss der markierte Text nach der Operation geloescht werden. Er ist als File markiert und kann daher

mit "SHIFT 2" eliminiert werden.
Andere Editoren haben fuer das Kopieren einen Befehl und fuer das Verschieben einen anderen. Beim ASZMIC gibt es aber nur den fuer das Kopieren. Darum muss man sich beim Verschieben damit behelfen, nach dem Kopiervorgang das Original einfach zu loeschen. Das ist zwar langwieriger, war aber bei der Entwicklung des ASZMIC leichter zu schreiben und erfordert auch weniger Platz im ROM.

Sehen Sie sie die spaeter noch erwaehnten Befehle "SHIFT D" und SHIFT F" an. Damit haben Sie 3 Tasten zum Mergen. "SHIFT D" arbeitet wie "SHIFT G" und verwendet den Stern "*" als Markierungsbeginn; "SHIFT F" benutzt den Keil "<". Laut englischem Originalhandbuch koennen diese Funktionen arbeiten - oder auch nicht ...

Kurze Uebersicht ueber die Funktionstasten

SHIFT 9	DEBUG-Modus	Umschalten in anderen Modus
SHIFT E	EDIT-Modus	
SHIFT 0	Rubout links	Loeschen
SHIFT Q	Rubout rechts	
SHIFT 1	Zeile loeschen	
SHIFT 2	File loeschen	
SHIFT 5	Cursor links	Bewegen des Cursors
SHIFT 8	Cursor rechts	
SHIFT 6	Cursor runter	
SHIFT 7	Cursor rauf	
SHIFT 4	Seite rauf	Bewegen ueber eine Seite
SHIFT 3	Seite runter	
SHIFT R	Macro ausfuehren	Macro
SHIFT G	Mergen	Mergen
SHIFT D	Mergen	#
SHIFT F	Mergen	#

Sonderzeichen	Pfund	^	Markierung fuer Beginn eines Files
			Ende-Markierung fuer Mergen
	Keil	>	Anfangsmarkierung fuer Mergen mit SHIFT G
	Keil	<	Anfangsmarkierung fuer Mergen mit SHIFT F #
	Stern	*	Anfangsmarkierung fuer Mergen mit SHIFT D #
	END-OF-DATA		Ende-Markierung aller Daten

wenn Ihr ASZMIC es tut.

KAPITEL V

Die ersten Versuche zum Debuggen

Alles, was Sie im EDIT-Modus tun koennen, ist auch im DEBUG-Modus moeglich. Der Unterschied zwischen beiden besteht darin, dass im DEBUG-Modus bei Eingabe eines <Newline> der Kommando-Interpreter den Befehl sofort ausfuehrt. Wenn Sie allerdings im DEBUG-Modus editiert haben, koennen Sie nicht darauf verlassen, wie bisher einfach mit "SHIFT E" an ihre zuletzt im EDIT-Modus bearbeitete Stelle zurueckkehren zu koennen. Sie werden zwar an jene Stelle im Speicher zurueckkehren, wo der Cursor WAR. Aber wenn Sie dort etwas anderes hineingeschrieben haben, stehen Sie jetzt zum Beispiel auf einem <Newline>-Zeichen (und dann erscheint der Cursor nicht).

D fuer DUMP

Machen Sie einen Reset. Um ganz sicher zu sein, dass der Rechner im DEBUG-Modus ist, druecken Sie

SHIFT 9

Geben Sie nun ein

D :4300 12<Newline>

und der Rechner wird Ihnen eine Anzeige der Form

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

ausgeben. Dies ist ein DUMP. Neu ist die "12" vor dem <Newline>. Sie bewirkt den Ausdruck der 12 Zeilen an jener Startadresse 4300.
Unser besonderes Augenmerk richten wir auf jene "4300". Es ist naemlich nicht nur moeglich, hier eine Zahl einzugeben. Dieser Platz, an dem die Zahl steht, ist ein "Feld". Im Feld nach dem "D" kann auch etwas stehen, aus dem der ASZMIC eine Adresse berechnen kann - mehr darueber am Ende dieses Kapitels.
Wenn Sie es noch nicht gemerkt haben : der Doppelpunkt vor der "4300" hat auch eine Funktion. Er bedeutet, dass die nachfolgende Zahl hexadezimal ist. Ohne Doppelpunkt wuerde der ASZMIC die "4300" als dezimal betrachten.

Das Blank (Leerzeichen) nach dem "D" ist uebrigens nicht notwendig. Es ist jedoch uebersichtlicher, wenn man sich an diese Schreibweise haelt.

Felder (Expressions, Ausdruecke) sind sehr wichtig und werden vom ASZMIC gleich behandelt. Man kann Expressions hintereinander schreiben (eingeben). Dabei muss man sie natuerlich voneinander trennen. Dazu nimmt man entweder Blanks oder Kommas. Man kann aber nur immer eins von beiden nehmen, nicht also durch Blank UND Komma trennen.

Die Form

D Adresse Bereich<Newline>

ist der DUMP ueber einen Bereich. Es gibt eine andere Form :

D Adresse <Newline>

Hier antwortet der Computer mit z.B. auf

D :4300<Newline>

mit

```
4300 00
```

Der Cursor steht hinter der "00" und der ASZMIC wartet auf eine Eingabe. Druecken Sie <Newline> mehrere Male. Bei jeder Eingabe gibt der Rechner weitere Zahlen aus :

```
4300 00
4301 00
4302 00
4303 00
```

Die 4300 und die folgenden Adressen sind Adressen im Speicher. Was dahinter steht ist der Inhalt der betreffenden Speicherzelle. So koennen Sie sich den Inhalt des ganzen Speichers ansehen, aber dafuer ist dieser Befehl nicht gedacht. Zuerst einmal muessen Sie aus diesem DUMP wieder heraus. Dazu geben Sie einen Punkt ein, gefolgt von <Newline>

```
.<Newline>
```

Schon sind Sie wieder im DEBUG. Geben Sie nun ein :

```
D :4300<Newline>
```

Weiter geht's mit

```
1 4 7 C 12<Newline>
.<Newline>      (Punkt und Newline)
```

Die zweite Zeile ist klar : damit kommt man aus dem DUMP wieder heraus. Die Wirkung der ersten Zeile sehen Sie sich mit dem bekannten

```
D :4300 6<Newline>
```

an :

```
4300 01 04 07 0C 12 00
```

Sie bemerken, dass vorher nur "00" angegeben wurde, jetzt aber die Zahlen erscheinen, die Sie eingegeben hatten. Aus der "1" wurde "01" und aus "C" wurde "0C" etc. Der erste DUMP-Befehl ist ein DUMP-Bereich-Befehl, der neue ein "DUMP-Modify". Das "Modify" ist englisch fuer "Modifizieren". Weil der deutsche Ausdruck "Modifizieren" extrem selten verwendet wird und "Modify" schlicht und einfach ein ueblicher Ausdruck, der auch von (fast) jedem verstanden wird, bleiben wir dabei.

Fassen wir zusammen :

D Adresse Bereich<Newline>	ist der DUMP ueber einen Bereich.
D Adresse <Newline>	ist DUMP-Modify.

Wenn Sie sich das Ergebnis noch einmal und ganz genau ansehen, dann ist es doch seltsam : Sie haben Zahlen und Buchstaben eingeben und der Rechner hat sie automatisch als hexadezimal angenommen - obwohl doch vorhin ausdruecklich gesagt wurde, hexadizimale Zahlen muessten durch einen Doppelpunkt definiert werden. DAS IST DIE AUSNAHME, von der oben die Rede war. Sie wurde aus praktischen Gruenden gemacht : es ist sehr, sehr selten, dass man beim Modify mit Dezimalzahlen arbeitet. Darum geht der Rechner von Hexzahlen aus. Natuerlich gibt es die Moeglichkeit, auch dezimale Zahlen einzugeben. Dazu nimmt man statt

```
hexzahl
```



```
ß + dezimalzahl
- - ----------
```

stellt also das "ß+" vor die Zahl. Sie koennen zum Beispiel auch "7+4", also eine Expression eingeben.

Um Ihnen ein weiteres Beispiel fuer DUMP zu geben : machen Sie Reset und gehen in den DEBUG-Modus, tippen ca 20 mal ^ (Pfund) ein. Nun schauen Sie im Anhang nach, welche Adresse der DSPBGN (Display-Begin) hat und addieren 55 dazu. Das ergibt beispielsweise :40B4+55.
Beenden Sie sie Zeile mit den ^ mit einem <Newline> und tippen :

```
D :40B4+55<Newline>
```

gefolgt von

```
1C<Newline>
1D<Newline>
```

etc. Sie sehen jetzt, dass die Pfund-Zeichen von jenen Zeichen ueberschrieben werden, deren Code Sie eingegeben haben.

Beachten Sie im Uebrigen, dass die 40B4+55 ein arithmetischer Ausdruck sind und der Computer sie selbst ausrechnet. Hier sind entsprechend der Leistung des Unterprogramms zur Berechnung dieser Ausdruecke natuerlich Grenzen gesetzt.

Zusammenfassung : Es gibt 2 DUMP-Befehle :

D Adresse Bereich<Newline>	ist der DUMP ueber einen Bereich.
D Adresse <Newline>	ist DUMP-Modify.

Adresse ist dabei ein arithmetischer Ausdruck.

C fuer COPY

Machen Sie ein Reset. Wenn Sie ein 16k-RAM haben, ist das zwar nicht immer unbedingt notwendig, schafft aber eindeutige Verhaeltnisse.

Gehen Sie in den DEBUG-Modus und geben ein :

```
D :4300 20<Newline>
D :4300<Newline>
1 2 3 4 5 6 7 8<Newline>
.<Newline>
D :4300 20<Newline>
C: 4300 :4308 8<Newline>

D :4300 20<Newline>
```

So, das sehen Sie sich einmal an : Als erstes haben Sie den Inhalt der 20 Speicherzellen ab :4300 angesehen. Dann wurde mit DUMP-Modify in die 8 Zellen beginnend bei 4300 von 01 hex bis 08 hex hineingeschrieben und der Modify-Modus wieder verlassen. Das Ergebnis wurde angesehen. Dann wurde der Inhalt der Zellen ab :4300 kopiert in jene ab :4308. Die Laenge des kopierten Bereiches betraegt 8 Zellen. Zum Schluss wurde das Ergebnis angesehen.

Das Format der Kopier-Anweisung ist ganz einfach :

```
C Adresse des Originals   Adresse des Ziels   Laenge<Newline>
- --------------------   ----------------   ------ --------
```

Wenn Sie das ausprobieren und von :4300 bis :4305 kopieren wollen an Ziel :4302 und folgende, dann gibt es doch ein Hindernis : Wenn der Inhalt von :4300 kopiert worden ist in :4302, dann ist doch der alte Inhalt von :4302 ueberschrieben worden ! Also muss man rueckwaerts arbeiten und erst Inhalt von :4305 nach :4307 schaffen, dann Inhalt von :4304 nach :4306 und so weiter. Um Ihnen die Muehe zu ersparen, kontrolliert der ASMIC solche Ueberlappungen und macht den Arbeitsvorgang automatisch so, dass keine Fehler entstehen !

F fuer FILL oder Fuellen

Wie ueblich : Reset und hinein in den DEBUG ! Nachdem Sie gesehen haben, dass mit dem DUMP-Modify der Inhalt von Speicherstellen angesehen und ueberschrieben werden kann, ist es naheliegend, dass man oft mehr als nur ein paar Bytes im Speicher vollschreiben muss. Wenn man groessere Bereiche hat, die immer mit dem gleichen Muster (Inhalt) beschrieben werden sollen, waere DUMP-Modify allerdings sehr muehselig. Darum gibt es einen Befehl

F Anfangs-Adresse End-Adresse Inhalt<Newline>

Damit (probieren Sie es aus !) kann man von der angegebenen Anfangsadresse bis zur Endadresse den Speicher mit dem gewuenschten Inhalt fuellen. Nehmen Sie :

F :4300 :4310 :AA<Newline>

Beachten Sie vor dem Inhalt den Doppelpunkt !!

E fuer EDIT

Das hier kennen Sie schon : Wird im DEBUG-Modus das "E" eingegeben, so kehren Sie in den EDIT-Modus zurueck.

Folgt nach dem "E" ein Symbol, dann durchsucht der ASZMIC den TExt nach diesem Symbol und stellt den Cursor dorthin. Das war so bei unserem Beispiel mit den Macros. Sehen Sie sich dazu auch die Beschreibung der Subroutine CMPSTR an : Sie dient dazu, beim E-Kommando und vielen anderen Befehlen einen Vergleich zu pruefen.

H fuer HINEIN !

Das ist nun kein englischer Ausdruck. Warum ist das so ? Wenn nur eine

beschraenkte Anzahl von Zeichen zur Verfuegung steht, die man als Befehle nehmen will, so gibt es Buchstaben und Sonderzeichen. Buchstaben kennen Sie bereits : Das "E" gehoert dazu. Die SHIFT-Befehle und das Punkt-Kommando gehoeren zu den Sonderzeichen. Die SHIFT-Befehle sind nicht druckbar, wohl aber der Punkt. Um sich die Funktion der Befehle gut merken zu koennen, nimmt man ueblicherweise einen Namen (wie DUMP oder FILL) und benutzt dann ein "D" oder ein "F" als Abkuerzung fuer diesen Befehl (dann braucht man weniger Buchstaben und der Befehlsinterpreter ist einfacher zu schreiben). Leider hapert es mit der Sprache : die Befehle heissen nicht so, dass man von A bis Z das Alphabet voll ausnutzen kann. Dann kommen eben solche "Ausreisser" zustande wie "H". Im englischen Original wurde "H" mit "HORRIBLE JUMP", also "fuerchterlicher Sprung" uebertragen, was aber eine ebenso wacklige Eselsbruecke ist, wie das HINEIN !.

Format des Be[illegible]s :

H Startadresse<Newline>

Wenn man ein Programm im Speicher hat, dann moechte man es auch ausfuehren. Dazu muss man den Prozessor zwingen, an die Startadresse des Programms zu springen. Der H-Befehl laedt ein Register mit dieser Startadresse und weist den Prozessor dann an, an den Inhalt dieses Registers zu springen.

Versuchen Sie ein kleines Programm zu schreiben, das Sie mit der entsprechenden RETURN-Anweisung RET beenden, so dass der Prozessor wieder in den ASZMIC zurueckfindet. Nehmen Sie beispielsweise im Anhang die Routine INICON und springen mit "H" dort hinein. Der ASZMIC wird dort hineinspringen und das ganze System wird neu initialisiert. Das hat die gleiche Wirkung wie ein Reset oder Ziehen des Steckers.

M fuer MACRO

Dies ist eine Erweiterung von "SHIFT G". Machen Sie Reset und gehen in den DEBUG-Modus.

Tippen Sie :

```
=D 0 8<Newline>
^<Newline>
+D 16 8 <Newline>
^<Newline>
```

Geben Sie nun mehrmals

M=<Newline>

und mehrmals

M+<Newline>

ein. Das erzeugt die vorgegebenen DUMPs. Das Zeichen nach dem "M" gibt das Startzeichen fuer den Kommando-Macro an - im Gegensatz zum "SHIFT G", bei dem der Macro immer mit ">" beginnt.

Das erlaubt die Benutzung einer ganzen Reihe von Macros. So ist jener DUMP, der mit dem "=" Gleichheitszeichen angefordert wird vom Macro mit dem "=" aufgerufen worden und jener DUMP mit dem "+" davor vom Macro mit dem "+" nach dem "M".

O fuer Old Register

Geben Sie DEBUG-MODUS

O<Newline>

ein. Sie werden mehrere Zeilen mit beeindruckenden Zahlen sehen. IN jeder befinden sich 6 4-stellige Hex-Zahlen. Das sind die Register, die im REGIM (Register Image = Register-Abbild) genannten Bereich im Speicher gelagert werden. Normalerweise ist das alles ohne Bedeutung. Wenn Sie aber ein Programm ausprobieren wollen und es noch nicht einwandfrei funktioniert, koennen Sie Schritt fuer Schritt mitverfolgen, was der Prozessor in seinen Registern treibt.

Es ist sehr nuetzlich, bei einer Sitzung mit dem ASZMIC [illegible]"O" in die SHIFT-Macro-Zeile zu schreiben : die Macro-Zeile wird bei jeder BREAK-Bedingung ausgefuehrt. Da man ueblicherweise einen BREAK macht, um die Register anzusehen, wird das hier also automatisch ausgefuehrt.

Sie koennen den "O"-Befehl ausprobieren und dazu im Speicher den REGIM-Bereich von REGIM bis REGIM+24 modifizieren. Dann sehen Sie ihn hinterher mit "O" an.

P fuer PRINT

Dieser Befehl wird absolut nichts tun, wenn Sie kein Sinclair-Drucker angeschlossen haben. Falls doch : Reset und in den EDIT-Modus gehen. Geben Sie ein :

```
^PRINT.FILE
JEDER MURKS
WIRD
REICHEN
^
```

Mit "SHIFT 9" gehen Sie in den DEBUG-Modus. Nun tippen Sie :

P ^PRINT.FILE<Newline>

und das File wird ausgedruckt. Wenn Sie den Druckvorgang vorzeitig abbrechen wollen (wenn Sie beispielsweise bei der Eingabe das ^-Zeichen am Ende des Textes vergessen hatten), so brauchen Sie nur die BREAK-Taste zu druecken.

Weitere Funktionen des DEBUG

Um ueberhaupt sinnvoll mit dem Computer arbeiten zu koennen, braucht man eine Moeglichkeit, auf einem Band oder einer Cassette all das zu speichern, was entweder eingegeben oder beim Assemblieren erzeugt wurde.

Der ZX 81 ist mit einem Anschluss fuer einen Cassetten-Recorder versehen. Da lag es nahe, diesen Anschluss auch vom ASZMIC aus zu verwenden. Damit man auch die vom BASIC-ROM aus gesavete Programme mit dem ASZMIC lesen kann, wird die gleiche Baudrate verwendet. Der Preis fuer diese Kompatibilitaet ist das Schneckentempo fuer die Band-Operationen.

So kann man mit dem BASIC-ROM Programme auf's Band schreiben und mit dem ASZMIC lesen. Dann erzeugt man Maschinencode-Routinen, fuegt sie in die REM-Zeilen ein und schreibt wieder auf's Band zurueck. Das ergibt Programme, die das BASIC-ROM lesen und ausfuehren kann !
Dies ist allerdings nur mit einer Reihe von Tricks moeglich.
Wenn der ASZMIC ein File auf Cassette schreibt, kommt als erstes eine Titel-Zeile zum Identifieren des nachfolgenden Files oder Programms. Dann wartet der ASZMIC 5 Sekunden, bevor er das File oder den von Ihnen angegebenen Speicherbereich auf's Band schreibt.

Beim Laden findet der ASZMIC den Titel und schreibt ihn auf den Bildschirm. In der 5-Sekunden-Pause haben Sie dann Gelegenheit, diesen Titel zu lesen. So erhalten Sie einen Katalog des Bandinhalts. Ein Vorteil dabei ist auch, dass Sie so sehen, ob der Rechner abgestuerzt ist oder noch laedt.

K fuer Kassette

Machen Sie einen Reset und gehen in den EDIT-Modus.Dann geben Sie ein File

ein, wie zum Be[illegible]el dieses hier :

```
^TUERSCHILD
DER WEG ZUR HOELLE IST GLATT UND EBEN
UND GLAENZT VON GOLD UND SILBER IM LICHT

DER WEG ZUM PARADIES ABER IST HART UND
STEINIG, DUNKEL UND VOLL UNRAT

DRUM FUERCHTE DICH NICHT, OH WANDERER
DER DU DIESES HAUS BETRITTST, DENN SIEHE :

DU BIST AUF DEM RECHTEN WEG

ARIBERT DECKERS
^
```

Nun gehen Sie in den DEBUG-Modus und schliessen Ihren Cassetten-Recorder genauso an, wie es es fuer das BASIC tun. Dann tippen Sie :

K^S "APHORISMEN" ^TUERSCHILD<Newline>

Dabei ist es aeusserst wichtig, dass zwischen dem "S" und dem Gaensefuesschen nur ein Blank steht - sonst koennte der ASZMIC die Zeile hinterher nicht als Titel erkennen und wuerde Ihr File ignorieren, es also nicht lesen koennen.

Also : K ^ S <Blank> "

Nachdem Sie <Newline> gedrueckt haben, muessen Sie in den folgenden 5 Sekunden den Cassetten-Recorder starten. (Es schadet natuerlich nicht, wenn Sie ihn zu frueh loslaufen lassen.) Die Anzeige verschwindet fuer ca 1/2 Sekunde und die Titel-Zeile wird auf's Band geschrieben. Nun folgt eine Pause von 5 Sekunden, in der das Display stabil ist, danach verschwindet es wieder und das File wird gesaved.

Wenn alles auf Band ist, erscheint die Anzeige wieder. Mit der <BREAK>-Taste kann man das Saven jederzeit abbrechen.

Geben Sie ein :

F :4300 :4320 :BB<Newline>

Damit wird im Speicher ein Bereich mit :BB gefuellt. Um das nun auf Band zu sichern, muss angegeben werden, dass ein Speicherbereich zu saven ist :

K ^ S <Blank> "SPEICHERVERSUCH" :4300 :4320 <Newline>

Die Bedienung des Recorders erfolgt wie bei dem File vorhin.

Spulen Sie das Band zurueck.

L fuer LADEN (oder LESEN oder LOAD)

Wie ueblich : Reset und dann in den DEBUG-Modus. Tippen Sie ein :

L "GARNIX" <Newline>

Starten Sie den Recorder und spielen ab ("PLAY"). Das tun Sie bitte mit der gleichen Lautstaerke-Einstellung, die sich beim BASIC als am besten geeignet erwiesen hat. Nach einiger Zeit erhellt sich der Schirm und es

erscheint :

K^S "APHORISMEN" ^TUERSCHILD

Das bleibt fuer 5 Sekunden (das ist die Pause !). Danach verschwindet die Anzeige (der Prœssor ist mit Laden beschaeftigt und kann den Bildschirm nicht bedienen). Nach einer gewissen Zeit kommt wieder fuer 5 Sekunden eine Meldung, dieses Mal wird zusaetzlich angezeigt :

K^S "SPEICHERVERSUCH" :4300 :4320

Weil es auf dem Band kein File gibt mit dem Namen "GARNIX", koennen Sie entweder warten und warten ... oder Sie brechen gleich ab mit der <Break>-Taste. Spulen Sie das Band zurueck und versuchen ein 2. Mal zu laden, jetzt aber mit :

L "APHORISMEN" <Newline>

Das bringt Ihnen das zuerst gesicherte File in den Speicher. Pruefen Sie nach, ob auch alles stimmt.

Machen sie einen Reset, gehen in den DEBUG und spulen noch einmal das Band zurueck. Jetzt geht es an das Laden in den Speicherbereich. Pruefen Sie mit DUMP nach, ob von :4300 bis :4320 auch wirklich "00" stehen - dann laden Sie :

L "SPEICHERVERSUCH" <Newline>

DUMPen Sie ab :4300 und sehen nach, ob auch wirklich jene :BB geladen wurden, die Sie aufs Band geschrieben hatten.

A fuer ASSEMBLIEREN

Jetzt haben Sie alle Werkzeuge : sie koennen Editieren, Saven, Laden und mit dem DEBUG umgehen. Also geht's an die Hauptarbeit, das Assemblieren.

Da Sie sich mit dem Z80 und seinem Befehlsatz beschaeftigen muessen, brauchen Sie fuer diese Dinge gute Handbuecher. Als sehr gut hat sich zum Beispiel das Buch Rodney Zaks "Programmierung des Z80" erwiesen. Mr. Zaks hat einen eigenen Verlag, den Sybex Verlag, der diese Buecher in deutscher Uebersetzung auf den Markt gebracht hat.

Alle Buecher von Verlagen haben das gleiche Problem : sie sind nicht von den Herstellern der ICs gemacht. Fuer die ICs sind aber nur jene Angaben gueltig, die von den jeweiligen Herstellern selbst gemacht werden. Wer 100% zuverlaessige Daten braucht, haelt sich daher immer an die Datenbuecher, Handbuecher und Applikationsbuecher der Hersteller. Der Z80 wird von den Firmen Zilog, Mostek, SGS und NEC produziert. Die Distributoren dieser Hersteller koennen Ihnen die notwendigen Handbuecher beschaffen. Leider ist der Z80 inzwischen sehr betagt (muss man ja wohl zugeben !) und die neuen Prœssoren fuellen einen Grossteil der Datenbuecher. Darum sollte man sich aeltere Datenbuecher besorgen : dort wird meist viel besser erklaert und es werden auch mehr und bessere Beispiele gebracht. Heute ist der Z80 Standard und man beschraenkt sich auf extrem knappe Darstellungen - immer auf Kosten der Verstaendlichkeit.

Um Ihnen eine Uebersicht ueber die Befehle des Z80 zu geben, folgt im Anhang ein Nachdruck des "Z80 Micro-Reference Manual" von Mostek. Eigentlich sollte das Original diesem ASZMIC-Handbuch beigelegt werden, aber leider wird dieses Mini-Handbuch nicht mehr produziert. Um die Schrift gut lesbar zu machen, wurde die Original-Vorlage vergroessert.

Es ist fuer uns drucktechnisch nicht moeglich, ein Leporello zu falten. Wenn Sie wo[illegible], koennen Sie das "Z80-Micro-Reference-Manual" heraustrennen und so binden, wie es fuer Sie am guenstigsten ist.

Falls Sie es nicht schon getan haben : Sehen Sie sich das "Z80-Micro-Reference-Manual" einmal an. Das, was darin steht, wird Ihr Arbeitsgebiet werden. Wenn Sie glauben "fit" genug zu sein und sich auch genug bedrucktes Papier (Buecher) beschafft haben, wagen wir uns weiter :

Machen Sie einen Reset, gehen in den EDITor und geben folgendes ein (natuerlich die <Newline> nach den Zeilen nicht vergessen !) :

```
^File
 ORG :4300
START LD HL,0
 LD DE,0
 LD B,10
LOOP INC DE
 ADD HL,DE
LOOPEND DJNZ LOOP
 RST 0
^
```

Das ist ein Programm. Wir nehmen uns jede einzelne Zeile vor und erklaeren ihre Bedeutung :

^File	Name des Programms
ORG :4300	Hier beginnt das Programm. Jedes Programm muss eine Anweisung dafuer haben, wo es anfangen soll. Sonst streikt der Assembler.
START LD HL,0	Das Wort "START" ist ein Label, also ein Name fuer diese Adresse im Programm.
LD DE,0	DE mit 0 laden
LD B,10	B mit 10 dezimal laden
LOOP INC DE	Wieder ein Label : LOOP heisst Schleife und ist als Abkuerzung wesentlich besser als das deutsche Wort. Andere Kurzel sind LOP, LUP oder nur LP
ADD HL,DE	HL und DE addieren
LOOPEND DJNZ LOOP	Ende der Schleife. Wenn Register B nicht 0 ist, wird zurueckgesprungen zum Label LOOP (wo immer das auch stehen mag)
RST 0	Call auf Adresse 0, also ein Neustart !
^	Ende

Wenn Sie nicht die ORG-Anweisung benutzen, wird der ASZMIC moeglicherweise beim Inhalt von TXTLIM beginnen, aber das haengt von der Version des ASZMIC ab und ist daher zu vermeiden.

Nachdem das Programm eingegeben wurde, beginnen Sie das Assembly :

Gehen Sie in den DEBUG und tippen :

A ^FILE 1

Das "A" ist der Befehl zum Assemblieren und "^FILE" gibt den Namen des Programms an, das assembliert werden soll. Die "1" ist eine "Option", also eine zusaetzliche Angabe. Denken Sie immer daran, dass so eine Option ist wie die Angabe, ob sie einen 4-Zylinder oder einen 6-Zylinder-Motor haben wollen - und ohne Motor faehrt der Wagen nicht ! Wenn Sie also keine Option angeben, wird Ihnen ein vom Programm vorgegebener Wert zwangsweise geliefert. Diese Zuweisung nennt man im Englischen "default". Das kann man mit "vorgegeben", "vorbelegt" etc uebersetzt werden.

Die Option mit dem Wert "1" weist den ASZMIC an, ein Assembly-Listing zu produzieren.

Das Programm ist sehr kurz und der ASMIC ist auch sofort wieder "da" und zeigt Ihnen ein Listing mit den Object-Codes zu jedem Befehl. Dazu wird auch die Adresse angegeben, wo dieser Befehl, d.h. der Object-Code im Speicher steht.

Gehen Sie in den DEBUG-Modus und DUMPen das Programm ab :4300

D :4300 14<Newline>

und sehen nach, ob es auch tatsaechlich da ist.

Der Text, die "Source" ("Quellfile") ist nicht mehr noetig : loeschen Sie ihn. Das Programm lassen Sie im Speicher, wir brauchen es noch - also tunlichst keinen Reset machen !

J fuer JUMP

Das Programm soll ausgefuehrt werden. Es erzeugt die Summe der Zahlen von 1 bis 10 im HL-Register-Paar. Um den Rechner bei der Arbeit zu beobachten, benutzen wir einen SHIFT-Macro, der immer dann ausgefuehrt wird, wenn eine BREAK (Unterbrechungs)-Bedingung erfuellt ist. Dazu gehoeren Breakpoint, Single Step (Einzelschritt), RST 0 -Code oder ein extern erzeugter Interrupt ueber NMI.

Druecken Sie

<SHIFT T> 0 <SHIFT 9>

Das tun Sie ohne <Newline> und gelangen also wieder in den DEBUG. Es ist natuerlich das O-Kommando, also keine 0 (Null) statt des "O" eintippen !

Tippen Sie :

J :4300

Damit springt der Prozessor in das Programm und Sie erhalten nach Ablauf des Programms einen DUMP der Register. Achten Sie auf HL : Es sollte :0037 enthalten, was 55 dezimal entspricht. DE sollte :000A enthalten, also 10 dezimal.

B fuer BREAKPOINT

Das Programm ist einfach an uns "vorbeigesaust" und das Endergebnis wurde erst am Ende "serviert". Das geht gut, solange das Programm in Ordnung ist. Sicherheitshalber werden wir ein Programm unterbrechen, um auch "mittendrin" die Register zusehen zu koennen. Dazu wird ein BREAKPOINT eingefuegt. Geben Sie ein :

B LOOPEND<Newline>

Wie Sie sehen, ist LOOPEND ein Label, der ASZMIC rechnet also selbst die tatsaechliche Speicherstelle aus. Springen Sie mit

J START<Newline>

in das Programm und sehen sich die Register an. Sowohl HL als auch DE sollten :0001 enthalten und B immer noch :0A. Der BREAKPOINT wird wirksam, bevor B um 1 verringert wird, d.h. bevor der DJNZ-Befehl ausgefuehrt wird.

BREAKPOINTs arbeiten nur im RAM. Sie ersetzen einfach im Speicher jenes Byte an der Adresse, wo der Breakpoint wirksam werden soll durch den Code fuer einen RST 0 -Befehl (also :C7). In einem ROM kann dieser Austausch nicht stattfinden, weil in ein ROM nicht geschrieben werden kann. Man kann aber mit Tricks arbeiten und einen Breakpoint simulieren - darueber spaeter mehr.

Erreicht der Rechner beim Progamm-Ablauf das RST 0, so springt der Prozessor an die Adresse 0 und der ASZMIC "fischt ihn dann wieder auf".

Wenn ein BREAKPOINT bei normalem Programmablauf erreicht wird, ersetzt der ASZMIC das ausgetauschte Byte durch den urspruenglichen Inhalt. Pruefen Sie das mit

D LOOPEND

Geben Sie ein :

B <Newline>

Die Logik dieses Befehls besagt, dass hier keine Adresse angegeben wurde, also muss der vorher zugewiesene BREAKPOINT wieder eingefuegt werden. Das koennen sie

D LOOPEND 1

nachpruefen.

G fuer GO

GO heisst gehen, also weitergehen oder weitermachen. Das ist so, wie der BASIC-Befehl CONTinue.

Tippen Sie

G<Newline>

Das bringt Sie einen Befehl weiter. Tun sie das mehrere Male. Das nennt man Single-Stepping oder Einzelschritt-Betrieb. Sie sehen, wie sich der PC (Programm-Zaehler) und die betroffenen Register schrittweise aendern.

Weil beim G-Kommando keine Adresse angegeben wurde, benutzt der ASZMIC jene Adresse, die im Speicher als Abbild der Register (REGIM) abgelegt worden ist.

Nun, warum nicht versuchen, 20 Befehle auszufuehren, bevor es zurueck in den Monitor (das ist der ASZMIC) geht ? Tippen Sie

G START 20<Newline>

Das benutzte Label START gibt dem ASZMIC an, wo er beginnen soll und die "20", wie viele Schritte ausgefuehrt werden sollen, bevor es zurueckgeht in den ASZMIC.

Sie werden sehen, dass der PC (Programm-Zaehler), der auf die naechste auszufuehrende Instruktion zeigt, :4308 ist. HL ist :0015 und DE :0006. B enthaelt :04. Wenn sie nachdenken, dann werden Sie bemerken, dass dies der 6. Durchlauf durch die Schleife ist, und dass tatsaechlich 20 Befehle abgearbeitet wurden.

Geben Sie

J <Newline>

ein. Daran sehen sie, wie der J-Befehl das Programm weiterfuehrt, wenn keine Adresse angegeben wurde.

Anmerkung : Wenn Sie Single-Steppen und dabei auf einen BREAKPOINT stossen, dann werden Sie die Logik (das Programm) zum Ausfuehren eines BREAKPOINT Single-Steppen, was ganz bestimmt nicht das ist, was Sie eigentlich tun wollten. Bedenken Sie, dass hier 2 Funktionen des ASZMIC miteinander kollidieren koennen. Der ASZMIC merkt naemlich nicht, dass durch das Einsetzen eines BREAKPOINT im RAM ein RST 0 eingefuegt wurde : er geht auch den Befehl RST 0 und die nachfolgenden im Single-Step durch.

I fuer IMMEDIATE

Fuer das Wort "immediate" gibt es keine passende Uebersetzung mit einem "I" als Anfang. "Immediate" heisst "sofort" und ist im allgemeinen technischen Sprachgebrauch ein gaengiges Wort, so dass es hier auch beibehalten wird.

Sie koennen im BASIC Befehle direkt ausfuehren lassen :

```
PRINT 2**3  <Newline>
```

weist den Rechner an, das Ergebnis sofort auf dem Schirm auszudrucken, ohne dass Sie erst ein Programm eingeben und mit RUN starten muessen.

Der ASZMIC akzeptiert Befehle in Assembler und fuehrt sie (als eine Ergaenzung Ihres Programms) in der Umgebung (dem Kontext) Ihres Programms aus. Es muss also nicht erst neu assembliert werden.

Wenn Sie nur ein 1k-RAM haben, loeschen sie jetzt bitte den Textbereich.

Tippen Sie :

```
I LD,1<Newline>
```

und sehen sich den Inhalt des HL-Registers mit dem O-Befehl an. Es enthaelt :0001. Tippen Sie

```
I EX DE,HL<Newline>
```

und beachten Sie, dass der Inhalt der Register DE und HL vertauscht wurde.

In einem Immediate-Befehl koennen sie jeden Assembler-Befehl geben. Sie koennen sogar Label mit der EQU-Zuweisung festlegen. Relative Spruenge und andere Anweisungen, wie ORG, DEFB, DEFW, DEFM, sind bedeutungslos und koennen das System zum Absturz bringen, wenn man mit dem I-BEfehl ihre Ausfuehrung erzwingen will.

Geben sie zum Schluss

```
LD HL,1+2+3+4+5-15<Newline>
```

ein und sehen sich HL im DUMP an. Es sollte 0 enthalten. Das ist eine Probe fuer eine arithmetische Anweisung in einer Expression.

Der N-Befehl

Der ASZMIC beherrscht auch einen Befehl mit Namen "N". Dieser Befehl hat nur dann einen Sinn, wenn eine besondere Platine fuer beide ROMS, das ASZMIC-ROM und das BASIC-ROM, angeschlossen wird. Dann wird mit dem N-

Anleitung zu dieser Platine ("External Card") erklaert.

Initialisierung

Wenn Sie den Rechner einschalten, wird der Speicher in 3 Bereiche aufgegliedert. Der erste enthaelt Systemvariable, Pufer, den Stack und das Abbild der Register (REGIM). Der zweite, der Textbereich, liegt zwischen DSPBGN und (TXTLIM). Der dritte (fuer Text und Daten) schliesslich liegt zwischen (TXTLIM) und der Oberkante des Speichers. Die Zeiger LABEND und LABSTK zeigen beide auf die Oberkante des Speichers.Diese Zeiger definieren die Symbol-Tabelle und Assemblierungs-Vorgaenge bewirken, dass der Inhalt von LABEND jene Adresse ist, die das aktuelle untere Ende der Symbol-Tabelle ist. Der Inhalt von LABSTK ist immer die Oberkante des verfuegbaren Speichers. Weiter gilt :

```
IY-Register   :4000   (also die Startadresse des RAMs)
I-Register    14
IM            1       (Interrupt-Modus ist auf 1 gesetzt)
```

Probleme mit der Bilderzeugung

Im DEBUG wird beim Eingeben eines <Newline> die damit beendete Zeile dem Kommando-Interpreter uebergeben. Wenn der erste Buchstabe in der Zeile von einer von A bis P ist, wird eine Funktion ausgefuehrt - sonst wird sie ignoriert.

Wenn bei einem Tastendruck der Kommando-Interpreter nicht aufgerufen wird, um einen Befehl auszufuehren, wird durch einen Taktgenerator (vertikale Synchronisierung) dafuer gesorgt, dass der Bildschirm stabil bleibt, waehrend der Tastendruck bearbeitet wird.
Der Kommando-Interpreter benutzt den Generator nicht.
Die Gesamtwirkung ist, dass Scrollen und Cursor-Bewegungen ein Flackern bewirken, aber dennoch ein lesbares Display behalten. Bei einem ZX 80 geht das nur mit Slow-Modus.

Kommando-Interpreter und Assembler benutzen beide ein Unterprogramm namens GETFLD. Das ist ein allgemeiner Interpreter fuer Felder.

Beachten Sie, dass die Hardware des ZX 80/81 erzwingt, dass bei einem Signal M1=0 und A15=1, eine Bedienung des Bildschirms erfolgt und der RAM-Bereich von 48k bis 64k ausgelesen wird, obwohl (zum Beispiel im BASIC das Display-File) der darzustellende Text im Bereich von 16k bis 32k liegt. Darum koennen oberhalb von 32k keine Maschinenprogramme ablaufen.

Sie sollten sich jetzt den Anhang mit den genauen Definitionen der einzelnen DEBUG-Befehle ansehen. Hier, in diesem Kapitel fuehren wir sie nur in den Gebrauch dieser Befehle ein, die volle Bedeutung zu erklaeren, waere an dieser Stelle zu viel auf einmal.

Tabelle der DEBUG-Funktionen

A	Assemblieren	A Filename Option<Newline>
	Optionen : 128	2. Durchlauf erzwingen
	64	Erzeugung des Object-Codes unterdruecken
	32	64 Zeichen pro Druckzeile / keine Laengenbegrenzung
	16	gibt es nicht
	8	gibt es nicht
	4	alte Symbol-Tabelle erhalten und ergaenzen
	2	Listing zum Drucker senden
	1	Listing erzeugen
B	Breakpoint	B<Newline>
		B adresse<Newline>
C	Kopieren	C Adresse-des-Originals Zieladresse Laenge<Newline>
D	DUMP	D Adresse<Newline>
		D Adresse Bytezaehler<Newline>
E	EDIT	E<Newline>
		E Symbol<Newline>
F	Fuellen	F Startadresse Endadresse Byte<Newline>
G	GO	G<Newline>
		G Adresse<Newline>
		G Adresse Schrittzaehler<Newline>
H	HINEIN !	H Adresse<Newline>
I	Immediate	I Assemblerzeile<Newline>
J	JUMP	J<Newline>
		J Adresse<Newline>
K	Saven Cassette	K^S "Kennung" ^Filename<Newline>
		K^S "Kennung" Startadresse Endadresse<Newline>
		K^S "Kennung" Startadresse Endadresse L<Newline>
L	Laden Cassette	L "Kennung"<Newline>
M	MACRO	M Buchstabe<Newline>
N	NEW	N<Newline>
O	Old Register	O<Newline>
P	Print	P ^Filename<Newline>

Genaue Definition eines Feldes

Der Kommando-Interpreter und der Assembler benutzen ein Unterprogramm namens GETFLD (siehe auch spaeter). Das ist ein allgemeiner Interpreter fuer Felder. Gueltige Felder sind :

a) 1-stellige bis 5-stellige Dezimalzahl mit den Ziffern 0 bis 9 darin

b) 1-stellige bis 4-stellige hexadezimale Zahl mit einem Doppelpunkt davor und den Buchstaben A bis F und den Ziffern 0 bis 9 darin

c) $ (das Dollar-Zeichen)
gibt den aktuellen Stand des IX-Register an (wird beim Assembler als Adresszaehler benutzt)

d) eins oder zwei Zeichen, die von ""Gaensefuesschen umgeben sind

Diese 4 gueltigen Felder von a) bis d) sind selbst-definierend.

e) ein Symbol. Das ist eine Zeichenkette aus 3 oder mehr Zeichen (Buchstaben von A bis Z und Ziffern von 0 bis 9, sowie dem Punkt). Ein Symbol muss mit einem Buchstaben beginnen.
Ein Symbol hat nur dann eine Bedeutung, wenn es in einem Label einer assemblierten Anweisung (mit dem A-Befehl oder dem I-Befehl) definiert wurde.
Wenn ein Symbol mit einem ? beginnt (Fragezeichen) UND WENN DIESES SYMBOL ALS ARGUMENT bei einer Referenz benutzt wird, dann werden die Bedingungen fuer die Mindestlaenge der Zeichenkette und fuer den Buchstaben als erstes Zeichen gelockert. Das Fragezeichen ist nur dazu da, die nachfolgende Zeichenkette als Symbol zu kennzeichnen und gilt nicht als Teil des Symbols.

f) Eine Kombination der Moeglichkeiten vaon a) bis f) getrennt durch ein + oder ein - (Plus-Zeichen oder Minus-Zeichen).
Das + bewirkt eine Addition der Elemente des Feldes.
Das - bewirkt eine Subtraktion der Elemente.
Klammern werden nicht benutzt.
Eine linke Klammer (bewirkt,dass BFLAG auf 0 gesetzt wird.

Felder werden mit einem Blank oder einem Komma beendet, nicht aber mit beidem gleichzeitig.

Beispiele fuer Felder :

```
"A*"
12345
:1234
KAESE
RECHNUNGS.NUMMER
$
12345+"A*"+:1234-KAESE+$-?HL+RECHNUNGS.NUMMER
```

K A P I T E L VI

Ueber den Umgang mit Text

Um mit dem ASZMIC vernuenftig arbeiten zu koennen, muss man verstehen, was der ASZMIC im Speicher tut. Beginnen wir mit der Aufteilung der 64k, auf die der Prozessor direkt zugreifen kann.

```
 0k -  4k 0000 0FFF ASZMIC
 4k -  8k 1000 1FFF obere Haelfte des BASIC-ROMs ($)
 8k - 16k 2000 3FFF Luecke

16k - 17k 4000 43FF 1k-RAM
16k - 32k 4000 7FFF 16k-RAM

32k - 48k 8000 BFFF Luecke

48k - 64k C000 FFFF Spiegelung des Bereiches von 4000 bis 7FFF
                    fuer die Anzeige auf dem Bildschirm
```

($) Bei Verwendung der External Card

Fuer die Bearbeitung von Texten steht ein Bereich zur Verfuegung, der von DSPBGN (DiSPlayBeGiN) bis zu (TXTLIM) (TeXTLIMit) reicht. Die Klammer () um TXTLIM bedeutet, dass TXTLIM die Adresse einer Variablen ist. Der Inhalt dieser Adresse ist gemeint, also das, was dort steht : "von DSPBGN bis zum Inhalt von TXTLIM". Die Schreibweise mit den Klammern ist ueblich und auch kuerzer zu schreiben.

In einem 16k-System stehen fuer den Text ca 12000 Byte zur Verfuegung. (Im englischen Original gibt es Aussagen, die "1/4 des Speichers" besagen. Der Begriff des Speichers wird doppeldeutig verwendet : 1. als Angabe fuer das RAM, 2. fuer den Gesamtbereich des Prozessors.)

Der Platz kann fuer laengere Files oder Programme vergroessert werden, indem man den Zeiger TXTLIM verschiebt. TXTLIM definiert die Grenze zwischen Textbereich und Programm-Bereich. Benutzen Sie dazu den DUMP-Befehl mit Modify oder den I-Befehl. Die Symbol-Tabelle kann verschoben werden, indem man LABEND (zeigt auf das untere Ende der Symbol-Tabelle) und LABSTK (zeigt auf das obere Ende der Tabelle) aendert. Dann wird der Inhalt der Tabelle verschoben - es sei denn, sie war ohnehin leer.

Der Fernsehschirm ist ein Fenster, das sich mit EDIT-Befehlen fast beliebig ueber dem Textbereich verschieben laesst. Es gibt kein spezielles Display-File wie beim BASIC. Statt dessen ist der ganze Text so angelegt, dass er direkt auf dem Schirm ausgegeben werden kann. Die uebliche Trennung des BASIC in Programm-Bereich, Variable und Display-File gibt es hier nicht, sie ist ohne Belang.

Herr Johnsons Software ermoeglicht hier eine Ausnutzung einer der wesentlichen Staerken des ZX 81 : der Grossteil des Speichers kann auch direkt auf dem Bildschirm dargestellt werden.

Wir zwingen diesem Bereich einige neue Konventionen (Vereinbarungen, Definitionen) auf und machen damit das Leben fuer Sie ein wenig leichter.

Files

Die erste Definition betrifft den Text. Im Textbereich koennen Files angelegt werden. Files werden gekennzeichnet durch einen Filenamen am Anfang und eine Filemarke ^ am Ende. Mit dem Filenamen koennen die Files im Textbereich einzeln identifiziert werden.

- Der Filename hat ein ^ (Pfund-Zeichen) als erstes Zeichen und darf die Zeichen A bis Z und 0 bis 9, sowie den Punkt enthalten.
- Das ^ Pfund-Zeichen muss das erste Zeichen im File sein, muss also in der ersten Zeile des Files ganz links stehen.
- Das Ende des Files wird ebenfalls durch ein ^ markiert. Dieses ^ allein und ganz links in einer Zeile.
- Der NAME des Files besteht auch aus dem ^ !
- Das ^ bezeichnen wir als Filemarke.

Im ASZMIC gibt es eine Routine, um 2 Zeichenketten zu vergleichen : CMPSTR (CoMPare STRing). Sie ist so geschrieben, dass sie den Vergleich von Zeichenketten mit einem ^ darin nur dann anerkennt, wenn die zu vergleichende Kette (das Ziel) ganz vorne in einer Zeile steht.

Das hoert sich kompliziert an, funktioniert aber in der Praxis ganz wunderbar : Sie geben jedem File einen eindeutigen Namen und beenden es mit einer Filemarke. Die Befehle des ASZMIC koennen das Sie interessierende File zweifelsfrei unter allen anderen im Textbereich und erkennen und es fuer Sie aufbereiten.

Das bedeutet, dass Sie beliebig viele Files anlegen koennen. Die einzige Einschraenkung liegt in der Laenge des physikalisch vorhandenen Speichers.

Achtung : Der DEBUG wird oft auch dann arbeiten, wenn Sie einen ungueltigen Filenamen angegeben haben, also einen, der nicht mit einer Filemarke beginnt. Fuer die Folgen dieser "Bearbeitung" gibt es aber keine Garantien !

Wenn Sie die Filemarke zum Beenden eines Files vergessen haben und drucken wollen oder Laden/Saven wollen, geraten Sie in gewaltige Schwierigkeiten. Das Assemblieren oder Mergen eines ohne Filemarke abgeschlossenen - also endlosen - Files erweist sich als Katastrophe.

Mergen und Loeschen

Diese Funktionen wurden dem EDITor als SHIFT-Funktionen zugewiesen - und nicht dem DEBUG zugeordnet. Dies geschah, um flexibler zu sein und gemaess der Ueberlegung "Es ist besser, etwas zu zeigen, als nur darueber zu reden" einen ausgereiften Bildschirm-Editor zu erhalten.

Der Cursor wird benutzt, um den Startpunkt fuer Loesch- und Merge-Operationen zu markieren. Die Operation gilt bis zur naechsten vorgefundenen Filemarke.
Wenn eine Befehlstaste fuer das Mergen gedrueckt wurde, durchsucht der ASZMIC den Textbereich : er faengt kurz hinter der SHIFT-Macro-Zeile an und geht bis kurz hinter das Ende des Textes, also hinter das END-OF-DATA-Zeichen. Hat ASZMIC die Markierung fuer das Mergen (fuer SHIFT G das ">", fuer SHIFT F der "<" und fuer SHIFT D den "*") gefunden, so fuegt er eine Kopie des Quell-Texts (ab dieser Marke) dorthin, wo der Cursor steht. Das geht so lange, bis die Ende-Markierung fuer das Mergen

gefunden wird, die Filemarke.
Weil das einige Zeit benoetigt, kann sich bei laengeren Texten der Bildschirm verdunkeln.

Das eine Extrem des Mergens ist das normale Einfuegen eines Files, das andere ein Text-Macro. So koennen sie beispielsweise Mengen von Unterprogrammen von Cassette lesen und im Rechner zu in ein Gesamtprogramm einfuegen. Wenn Sie ein Programm mit grossen Datenmengen haben, schreiben Sie

```
> DEFB ^
```

ganz oben in den Text und holen das bei jedem "SHIFT G" herunter an die aktuelle Position. Das erspart unnoetiges Tippen.

Wenn sie das Source-File oder die Zeichenkette loeschen wollen, die sie mit Merge transportiert haben, stellen Sie den Cursor auf die Merge-Markierung und druecken "SHIFT 2" - schon ist geloescht.

Eine Filemarke ist stets das Ende fuer's Mergen oder Loeschen. Fehlt bei einer Quelle die Ende-Marke, so hoert der Computer nicht auf, einzufuegen und stuerzt ab. Dabei kann man ihn auch nicht unterbrechen. Der ASZMIC hat beim Mergen leider keine Ueberwachungsfunktion eingebaut, die beim Erreichen einer Grenze "die Reissleine zieht".

Loeschen ohne Ende-Marke wird sofort verweigert.

Editieren

Die EDIT-Funktionen wurden bereits ausgiebig erklaert - darum nur noch einige Hinweise.

Sie erinnern Sich doch noch an den Macro mit der Zeichen-Kette mit dem "E" ? Man kann mit einem SHIFT-Macro alle Vorkommen einer bestimmten Zeichenkette in einem Text heraussuchen lassen.

Das Eingeben von <Newline> inmitten einer vorher geschriebenen Zeile macht daraus 2 Zeilen.

Mit dem "Rubout" SHIFT 0 kann man 2 Zeilen aneinander haengen.

Bitte versuchen Sie, nicht in den obersten 20 Zeilen des Textbereiches zu editieren, es sei denn, mit einem SHIFT-Macro. Diese Zeilen sind naemlich Fuellmaterial und sorgen fuer einen gut lesbaren Bildschirm. Wenn man sie loescht, so ist das Ergebnis ein nicht zulaessiges Display.

Im Besonderen kann das Loeschen der <Newline>-Zeichen (mit SHIFT 0) ganz am Anfang des Displays die lebenswichtige - aber unsichtbare - Markierung BEGINN-DES-TEXTs freilegen. Das hat katastrophale Folgen.

Wenn das Blank geloescht wird, das der ASZMIC automatisch vor jedes <Newline>-Zeichen setzt, kann der Cursor verlorengehen. Beim Assemblieren und beim Mergen gibt's dann "kosmetische" Fehler.

Worden mehrere ueberlange Zeilen (jede mehr als 36 Zeichen) geschrieben, die gemeinsam in ein Fenster muessen - also auf dem Bildschirm angezeigt werden sollen - kommt der ASZMIC in Schwierigkeiten mit dem Cursor. Es bleiben aber keine boesen Folgen.

Sollen in einem Text Zeilen eingefuegt werden, stellen Sie den Cursor am Besten in die darueberliegende Zeile und gehen mit "SHIFT W" an deren Ende. Durch Eingeben von <Newline> wird eine Leerzeile eingefuegt, in

"SHIFT 7" mit nachfolgendem "SHIFT 6" bringt sie an den Anfang einer Zeile - das hatten wir bereits besprochen.

Erinnern Sie sich daran, dass "SHIFT Q" vorwaerts loescht und "SHIFT 0" rueckwaerts.

Drucken

Die Steuerung des Druckers arbeitet mit Hilfe von 2 Bits in der Variablen ASSFLG (ASSembly FLaG). Dieser Merker ASSFLG wird vom Assembler automatisch gesetzt (er dient zum Ueberwachen der Optionen) und bei Beendigung des Assemblierens auf "00" zurueckgesetzt.

Bei Unterbrechung mit <BREAK> wird das ASSFLG nicht zuverlaessig betreut und der Drucker unter Umstaenden nicht abgeschaltet.

Darum muss der Drucker von Ihnen hoechstselbst ueberwacht werden und Sie muessen auf die beiden besonderen Bits notfalls von Hand setzen !

Bit 1 auf "1" gesetzt, reicht die Ausgaben vom ASZMIC zum Drucker.

Bit 5 auf "1" gesetzt, bewirkt den Druck mit 64 Zeichen pro Zeile.

Sie koennen den Drucker ueberlisten, damit er die Zeichen doppelt so hoch druckt (z.B. fuer Ueberschriften). Dazu schreiben Sie eine Zeile von 32 Zeichen Laenge und fuellen sie mit Blanks auf 48 Zeichen auf. In der gleichen Zeile (kein <Newline> !) schreiben Sie jenen Text noch einmal (IN GENAU DER GLEICHEN ZEICHENFOLGE - Mergen !). Das beschert Ihnen nicht etwa eine lange Zeile mit 2 Texten, sondern eine einzige Zeile mit doppelt hohen Zeichen.

Unterlaeuft Ihnen bei den Blanks ein Fehler, so ist das Ergebnis interessant, aber kaum lesbar.

Diese Methode eignet sich auch fuer 64 Zeichen in einer Zeile. Dazu wird aber nicht - wie oben - mit 16 Blanks aufgefuellt, sondern mit 32.

Wenn Sie in einen Ausdruck geraten und die Filemarke als Ende-Zeichen fehlt : nicht die <BREAK>-Taste vergessen !

Noch eines zum Schluss : Das Drucken erfolgt indirekt. Der ASZMIC springt nicht direkt in das Unterprogramm zum Drucken, sondern sucht die Adresse der Subroutine ueber die Variable PRTJMP. Wenn Sie den Inhalt von PRTJMP austauschen gegen die Startadresse einer von Ihnen selbst geschriebenen Routine, dann wird Ihre Routine benutzt. So koennen Sie eigene Interfaces bauen und auch komplizierte Drucker verwenden.

Der Druck wird begonnen mit der Startadresse der Druckzeile auf dem Stack. Ihre Routine muss das beseitigen und mit einem RET zurueckkehren, wobei HL auf das erste Zeichen nach dem <Newline>-Zeichen zeigt, das die zu druckende Zeile beendet hat.

Arbeiten mit der Cassette

Ihre Cassetten-Recorder sollte mit den gleichen Einstellungen fuer Lautstaerke und Tonhoehe arbeiten wie beim BASIC.

Der ASZMIC erzeugt einen Ton und gibt diesen fortwaehrend aus, bis die eigentliche Aufnahme des Files beginnt. So wird bei Cassetten-Recordern mit automatischer Verstaerkungsregelung diese Regelung rechtzeitig "eingestimmt" und kommt damit nicht jene Schwierigkeiten, die sie beim

Empfang vom BASIC aus hat. Die Zuverlaessigkeit der Aufzeichnungen wird also erhoeht.

Wenn Sie eine neue Bandsorte haben und die optimale Lautstaerke herausfinden wollen, senden Sie am Besten eine Folge von Blanks :

```
K ^ S <Blank> "NULL" :7000 :7FFF <Newline>
- - - ------- ------ ----- ----- ---------
```

Weil beim Starten des Rechners (nach einem Reset) der Textbereich leer ist, koennen Sie den Reset hierfuer besonders gut verwenden.

Versuchen Sie, die Aufnahme zu lesen und beginnen dazu ganz leise. Wenn Sie den Recorder lauter stellen, wird der Bildschirm von einem verschwommenen Zustand uebergehen in eine Reihe genau abgetrennter Baender und schliesslich in einen hauptsaechlich schwarzen Schirm. Die guenstigste Einstellung ist jene in der Mitte der Baender.

HEADER und Files

Weil es 2 Arten von Files gibt, die auf Band gesichert werden koennen, muss dafuer gesorgt werden, dass jedes File eine besondere Markierung erhaelt - Bereiche im Speicher erhalten naemlich beim Saven keinen Namen.

Wenn Sie dem ASZMIC einen Save-Befehl geben, schreibt er diesen Befehl als Kopf (HEADER) auf das Band. Nach einer Pause folgt dann das File.

Beim Wiedereinlesen gibt der Aufbau des Save-Kommandos (das ja auf dem Band mit aufgezeichnet wurde !) eine eindeutige Zeichenfolge wieder, die der ASZMIC erkennt. Das bewirkt, dass ASZMIC dieses Kommando zum Identifizieren dessen, was auf dem Band ist, fuer ein paar Sekunden auf dem Bildschirm anzeigt.

Zum Laden gibt man dem ASZMIC einen Namen an. Stimmt dieser mit dem Namen im Header (im Kopf des Files) ueberein, so wird der Rest des Headers gepruefft und festgestellt, ob ein Text oder ein Speicherbereich geladen werden soll. Dementsprechend wird dann das File geladen.

Nach dem Laden kommen Sie automatisch in den EDITor.

Es gibt beim Laden eines Speicherbereiches einige Besonderheiten :

1. Wenn Sie einen Text auf Band haben, dann koennen Sie ihn beim Laden mittels Mergen an eine andere Stelle holen. Bei einem Speicherbereich ist es moeglich ein Stueck auf Band zu schreiben und hinterher an eine andere Stelle laden zu lassen. Die Variable OFFSET ("Verschiebung") bestimmt, wohin der Bereich geladen wird. Wenn Sie OFFSET aendern, bestimmen Sie, wohin geladen werden soll.

2. Wenn Sie die Variablen LABSTK und LABEND untersuchen und eine Symbol-Tabelle auf Band gesichert haben, so koennen Sie der Definition von "Ende des Bereichs" mit einem Blank folgen und einem L. ASZMIC wird das zu ladende als Symboltabelle akzeptieren und und LABEND dementsprechend setzen. LABSTK bleibt hingegen unveraendert, so dass Sie nicht nur die gleiche Speichergroesse benutzen muessen, sondern auch die Symboltabelle vor dem Saven nicht verschieben durften !

   Symboltabelle sind fuer Bibliotheken nuetzlich oder zum Debuggen.

Die physikalische Trennung von Header und File auf dem Band gibt Ihnen die Moeglichkeit, auch komplizertere Dinge zu tun und beim "Jonglieren" und Ueberschreiben. Das ist zum Wohl [illegible]

nicht zufrieden sind, solange sie ihr System nicht bis zum Aeussersten misbraucht haben. Ganz einfach geht's auch : zum Retten einer schlechten Aufnahme. Die Einzelheiten muessen Sie selbst herausfinden. (Das sagt das englische Handbuch und mehr wissen wir auch nicht. D&C).

Beachten Sie : Wenn die Routine RDCASS (ReaD CASSette) aufgerufen wird, bevor das Band die 1/2-Sekunden-Pause (mit "Schweigen") vor dem Header erreicht hat, moeglicherweise Unsinn liest. Das haengt ab von der Lautstaerke-Einstellung und dem Bildschirm-Inhalt zur Zeit der Aufnahme. Das Schweigen wird jedoch synchronisieren an einer Grenze eines Bytes, so dass das File zwar verschoben ist, aber nicht ruiniert.

Der ASZMIC betrachtet alles, was nach dem

```
K ^ S
- - -
```

bis zum END-OF-DATA-Zeichen folgt, als den Header. Darum sollte der SAVE-Befehl der letzte Befehl im Textbereich sein, wenn <Newline> gedrueckt wird.

Das Erzeugen von BASIC-Programmen

Es gibt eine sehr praktische Art, den Cassettenrecorder zu misbrauchen. Fuer all jene, die nicht die External Card haben und so ASZMIC und BASIC-ROM gleichzeitig benutzen koennen, ist der Cassettenrecorder das uebliche Werkzeug, um zwischen beiden uebertragen zu koennen. Im Anhang finden Sie ein Listing eines Programms fuer allgemeine Anwendungen : es simuliert den Kontext (die Umgebung) eines BASIC-Programms mit einem einzigen REM darin. Wenn Sie nun an der angegebenen Stelle Maschinencode einfuegen, wird in die REM-Zeile aufgenommen. Die Bandaufnahme mit diesem Programm kann dann vom BASIC aus geladen werden.

In unserem Beispiel wird die Kennung "123456789" benutzt. Sie koennen weitere BASIC-Zeilen hinzufuegen. Das Unterprogramm in der REM-Zeile wird schlicht mit USR (16514) aufgerufen. Ihr Maschinenprogramm sollte das HL'-Register nicht beeinflussen.

Zum Assemblieren des Programms muessen Sie mit OFFSET verschieben. Bei einem 16k-System sollte die ORG-Anweisung lauten :

```
ORG :4000
```

Dazu gehoert der Zusatz

```
OFFSET : 3000
```

Das funktioniert ganz gut. Sie muessen aber einen 9-Byte langen Header haben, weil beim BASIC die ersten 9 Bytes der Variablen nicht auf's Band geschrieben werden.

Wenn Sie den umgekehrten Weg gehen wollen und im ASZMIC Programme des BASIC laden wollen, dann bedarf das einer Vorbereitung : Sie muessen einen Bereich im Speicher nehmen, der gross ist, das BASIC-Programm aufzunehmen. Diesen Bereich saven Sie auf Band. Sobald mit der Ausgabe des Bereichs an den Recorder begonnen wurde, brechen Sie mit <BREAK> ab. der Header ist naemlich das einzige, was wir brauchen.

Nun spulen Sie zurueck, bis sie gerade vor jener Stelle sind, ab der die Daten aufgezeichnet wurden. Und genau hier gehen Sie in BASIC und saven dann vom BASIC aus (!) das Programm. (Zugegeben : das ist umstaendlich).

Beim Laden wird zuerst der Header gelesen und dann das Programm. Dann

Empfang vom BASIC aus hat. Die Zuverlaessigkeit der Aufz[illegible]hnungen wird also erhoeht.

Wenn Sie eine neue Bandsorte haben und die optimale Lautstaerke herausfinden wollen, senden Sie am Besten eine Folge von Blanks :

```
K ^ S <Blank> "NULL" :7000 :7FFF <Newline>
- - - ------- ------ ----- ----- --------
```

Weil beim Starten des Rechners (nach einem Reset) der Textbereich leer ist, koennen Sie den Reset hierfuer besonders gut verwenden.

Versuchen Sie, die Aufnahme zu lesen und beginnen dazu ganz leise. Wenn Sie den Recorder lauter stellen, wird der Bildschirm von einem verschwommenen Zustand uebergehen in eine Reihe genau abgetrennter Baender und schliesslich in einen hauptsaechlich schwarzen Schirm. Die guenstigste Einstellung ist jene in der Mitte der Baender.

HEADER und Files

Weil es 2 Arten von Files gibt, die auf Band gesichert werden koennen, muss dafuer gesorgt werden, dass jedes File eine besondere Markierung erhaelt - Bereiche im Speicher erhalten naemlich beim Saven keinen Namen.

Wenn Sie dem ASZMIC einen Save-Befehl geben, schreibt er diesen Befehl als Kopf (HEADER) auf das Band. Nach einer Pause folgt dann das File.

Beim Wiedereinlesen gibt der Aufbau des Save-Kommandos (das ja auf dem Band mit aufgezeichnet wurde !) eine eindeutige Zeichenfolge wieder, die der ASZMIC erkennt. Das bewirkt, dass ASZMIC dieses Kommando zum Identifizieren dessen, was auf dem Band ist, fuer ein paar Sekunden auf dem Bildschirm anzeigt.

Zum Laden gibt man dem ASZMIC einen Namen an. Stimmt dieser mit dem Namen im Header (im Kopf des Files) ueberein, so wird der Rest des Headers geprueft und festgestellt, ob ein Text oder ein Speicherbereich geladen werden soll. Dementsprechend wird dann das File geladen.

Nach dem Laden kommen Sie automatisch in den EDITor.

Es gibt beim Laden eines Speicherbereiches einige Besonderheiten :

1. Wenn Sie einen Text auf Band haben, dann koennen Sie ihn beim Laden mittels Mergen an eine andere Stelle holen. Bei einem Speicherbereich ist es moeglich ein Stueck auf Band zu schreiben und hinterher an eine andere Stelle laden zu lassen. Die Variable OFFSET ("Verschiebung") bestimmt, wohin der Bereich geladen wird. Wenn Sie OFFSET aendern, bestimmen Sie, wohin geladen werden soll.

2. Wenn Sie die Variablen LABSTK und LABEND untersuchen und eine Symbol-Tabelle auf Band gesichert haben, so koennen Sie der Definition von "Ende des Bereichs" mit einem Blank folgen und einem L. ASZMIC wird das zu ladende als Symboltabelle akzeptieren und und LABEND dementsprechend setzen. LABSTK bleibt hingegen unveraendert, so dass Sie nicht nur die gleiche Speichergroesse benutzen muessen, sondern auch die Symboltabelle vor dem Saven nicht verschieben durften !

   Symboltabelle sind fuer Bibliotheken nuetzlich oder zum Debuggen.

Die physikalische Trennung von Header und File auf dem Band gibt Ihnen die Moeglichkeit, auch kompliziertere Dinge zu tun und beim "Jonglieren" und Haberschreiben. Das ist zum Wohl jener hartnaeckigen Seelen, die

nicht zufrieden sind, solange sie ihr System nicht bis zum Aeussersten misbraucht hab[illegible] Ganz einfach geht's auch : zum Retten einer schlechten Aufnahme. Die Einzelheiten muessen Sie selbst herausfinden. (Das sagt das englische Handbuch und mehr wissen wir auch nicht. D&C).

Beachten Sie : Wenn die Routine RDCASS (ReaD CASSette) aufgerufen wird, bevor das Band die 1/2-Sekunden-Pause (mit "Schweigen") vor dem Header erreicht hat, moeglicherweise Unsinn liest. Das haengt ab von der Lautstaerke-Einstellung und dem Bildschirm-Inhalt zur Zeit der Aufnahme. Das Schweigen wird jedoch synchronisieren an einer Grenze eines Bytes, so dass das File zwar verschoben ist, aber nicht ruiniert.

Der ASZMIC betrachtet alles, was nach dem

```
K ^ S
- - -
```

bis zum END-OF-DATA-Zeichen folgt, als den Header. Darum sollte der SAVE-Befehl der letzte Befehl im Textbereich sein, wenn <Newline> gedrueckt wird.

Das Erzeugen von BASIC-Programmen

Es gibt eine sehr praktische Art, den Cassettenrecorder zu misbrauchen. Fuer all jene, die nicht die External Card haben und so ASZMIC und BASIC-ROM gleichzeitig benutzen koennen, ist der Cassettenrecorder das uebliche Werkzeug, um zwischen beiden uebertragen zu koennen. Im Anhang finden Sie ein Listing eines Programms fuer allgemeine Anwendungen : es simuliert den Kontext (die Umgebung) eines BASIC-Programms mit einem einzigen REM darin. Wenn Sie nun an der angegebenen Stelle Maschinencode einfuegen, wird in die REM-Zeile aufgenommen. Die Bandaufnahme mit diesem Programm kann dann vom BASIC aus geladen werden.

In unserem Beispiel wird die Kennung "123456789" benutzt. Sie koennen weitere BASIC-Zeilen hinzufuegen. Das Unterprogramm in der REM-Zeile wird schlicht mit USR (16514) aufgerufen. Ihr Maschinenprogramm sollte das HL'-Register nicht beeinflussen.

Zum Assemblieren des Programms muessen Sie mit OFFSET verschieben. Bei einem 16k-System sollte die ORG-Anweisung lauten :

```
ORG :4000
```

Dazu gehoert der Zusatz

```
OFFSET : 3000
```

Das funktioniert ganz gut. Sie muessen aber einen 9-Byte langen Header haben, weil beim BASIC die ersten 9 Bytes der Variablen nicht auf's Band geschrieben werden.

Wenn Sie den umgekehrten Weg gehen wollen und im ASZMIC Programme des BASIC laden wollen, dann bedarf das einer Vorbereitung : Sie muessen einen Bereich im Speicher nehmen, der gross ist, das BASIC-Programm aufzunehmen. Diesen Bereich saven Sie auf Band. Sobald mit der Ausgabe des Bereichs an den Recorder begonnen wurde, brechen Sie mit <BREAK> ab. der Header ist naemlich das einzige, was wir brauchen.

Nun spulen Sie zurueck, bis sie gerade vor jener Stelle sind, ab der die Daten aufgezeichnet wurden. Und genau hier gehen Sie in BASIC und saven dann vom BASIC aus (!) das Programm. (Zugegeben : das ist umstaendlich).

Beim Laden wird zuerst der Header gelesen und dann das Programm. Dann

Empfang vom BASIC aus hat. Die Zuverlaessigkeit der Aufzeichnungen wird also erhoeht.

Wenn Sie eine neue Bandeorte haben und die optimale Lautstaerke herausfinden wollen, senden Sie am Besten eine Folge von Blanks :

K ^ S <Blank> "NULL" :7000 :7FFF <Newline>

Weil beim Starten des Rechners (nach einem Reset) der Textbereich leer ist, koennen Sie den Reset hierfuer besonders gut verwenden.

Versuchen Sie, die Aufnahme zu lesen und beginnen dazu ganz leise. Wenn Sie den Recorder lauter stellen, wird der Bildschirm von einem verschwommenen Zustand uebergehen in eine Reihe genau abgetrennter Baender und schliesslich in einen hauptsaechlich schwarzen Schirm. Die guenstigste Einstellung ist jene in der Mitte der Baender.

HEADER und Files

Weil es 2 Arten von Files gibt, die auf Band gesichert werden koennen, muss dafuer gesorgt werden, dass jedes File eine besondere Markierung erhaelt - Bereiche im Speicher erhalten naemlich beim Saven keinen Namen.

Wenn Sie dem ASZMIC einen Save-Befehl geben, schreibt er diesen Befehl als Kopf (HEADER) auf das Band. Nach einer Pause folgt dann das File.

Beim Wiedereinlesen gibt der Aufbau des Save-Kommandos (das ja auf dem Band mit aufgezeichnet wurde !) eine eindeutige Zeichenfolge wieder, die der ASZMIC erkennt. Das bewirkt, dass ASZMIC dieses Kommando zum Identifizieren dessen, was auf dem Band ist, fuer ein paar Sekunden auf dem Bildschirm anzeigt.

Zum Laden gibt man dem ASZMIC einen Namen an. Stimmt dieser mit dem Namen im Header (im Kopf des Files) ueberein, so wird der Rest des Headers geprueft und festgestellt, ob ein Text oder ein Speicherbereich geladen werden soll. Dementsprechend wird dann das File geladen.

Nach dem Laden kommen Sie automatisch in den EDITor.

Es gibt beim Laden eines Speicherbereiches einige Besonderheiten :

1. Wenn Sie einen Text auf Band haben, dann koennen Sie ihn beim Laden mittels Mergen an eine andere Stelle holen. Bei einem Speicherbereich ist es moeglich ein Stueck auf Band zu schreiben und hinterher an eine andere Stelle laden zu lassen. Die Variable OFFSET ("Verschiebung") bestimmt, wohin der Bereich geladen wird. Wenn Sie OFFSET aendern, bestimmen Sie, wohin geladen werden soll.

2. Wenn Sie die Variablen LABSTK und LABEND untersuchen und eine Symbol-Tabelle auf Band gesichert haben, so koennen Sie der Definition von "Ende des Bereiche" mit einem Blank folgen und einem L. ASZMIC wird das zu ladende als Symboltabelle akzeptieren und und LABEND dementsprechend setzen. LABSTK bleibt hingegen unveraendert, so dass Sie nicht nur die gleiche Speichergroesse benutzen muessen, sondern auch die Symboltabelle vor dem Saven nicht verschieben durften !

   Symboltabelle sind fuer Bibliotheken nuetzlich oder zum Debuggen.

Die physikalische Trennung von Header und File auf dem Band gibt Ihnen die Moeglichkeit, auch kompliziertere Dinge zu tun und beim "Jonglieren" und Ueberschreiben. Das ist zum Wohl jener hartnaeckigen Seelen, die

nicht zufrieden sind, solange sie ihr System nicht bis zum Aeussersten misbraucht haben. Ganz einfach geht's auch : zum Retten einer schlechten Aufnahme. Die Einzelheiten muessen Sie selbst herausfinden. (Das sagt das englische Handbuch und mehr wissen wir auch nicht. D&C).

Beachten Sie : Wenn die Routine RDCASS (ReaD CASSette) aufgerufen wird, bevor das Band die 1/2-Sekunden-Pause (mit "Schweigen") vor dem Header erreicht hat, moeglicherweise Unsinn liest. Das haengt ab von der Lautstaerke-Einstellung und dem Bildschirm-Inhalt zur Zeit der Aufnahme. Das Schweigen wird jedoch synchronisieren an einer Grenze eines Bytes, so dass das File zwar verschoben ist, aber nicht ruiniert.

Der ASZMIC betrachtet alles, was nach dem

K ^ S

bis zum END-OF-DATA-Zeichen folgt, als den Header. Darum sollte der SAVE-Befehl der letzte Befehl im Textbereich sein, wenn <Newline> gedrueckt wird.

Das Erzeugen von BASIC-Programmen

Es gibt eine sehr praktische Art, den Cassettenrecorder zu misbrauchen. Fuer all jene, die nicht die External Card haben und so ASZMIC und BASIC-ROM gleichzeitig benutzen koennen, ist der Cassettenrecorder das uebliche Werkzeug, um zwischen beiden uebertragen zu koennen. Im Anhang finden Sie ein Listing eines Programms fuer allgemeine Anwendungen : es simuliert den Kontext (die Umgebung) eines BASIC-Programms mit einem einzigen REM darin. Wenn Sie nun an der angegebenen Stelle Maschinencode einfuegen, wird in die REM-Zeile aufgenommen. Die Bandaufnahme mit diesem Programm kann dann vom BASIC aus geladen werden.

In unserem Beispiel wird die Kennung "123456789" benutzt. Sie koennen weitere BASIC-Zeilen hinzufuegen. Das Unterprogramm in der REM-Zeile wird schlicht mit USR (16514) aufgerufen. Ihr Maschinenprogramm sollte das HL'-Register nicht beeinflussen.

Zum Assemblieren des Programms muessen Sie mit OFFSET verschieben. Bei einem 16k-System sollte die ORG-Anweisung lauten :

ORG :4000

Dazu gehoert der Zusatz

OFFSET : 3000

Das funktioniert ganz gut. Sie muessen aber einen 9-Byte langen Header haben, weil beim BASIC die ersten 9 Bytes der Variablen nicht auf's Band geschrieben werden.

Wenn Sie den umgekehrten Weg gehen wollen und im ASZMIC Programme des BASIC laden wollen, dann bedarf das einer Vorbereitung : Sie muessen einen Bereich im Speicher nehmen, der gross ist, das BASIC-Programm aufzunehmen. Diesen Bereich saven Sie auf Band. Sobald mit der Ausgabe des Bereichs an den Recorder begonnen wurde, brechen Sie mit <BREAK> ab. der Header ist naemlich das einzige, was wir brauchen.

Nun spulen Sie zurueck, bis sie gerade vor jener Stelle sind, ab der die Daten aufgezeichnet wurden. Und genau hier gehen Sie in BASIC und saven dann vom BASIC aus (!) das Programm. (Zugegeben : das ist umstaendlich).

Beim Laden wird zuerst der Header gelesen und dann das Programm. Dann

muss den Ladevorgang von Hand mit <BREAK> abbrechen.

Eine Alternative waere ein kleines Programm, das mit Hilfe von RDCASS einzelne Bytes vom Band lesen und sie im Speicher ablegen kann. Diese Methode ist wahrscheinlich schneller und leichter zu kontrollieren.

Als ASZMIC entwickelt wurde, versah Mr. Johnson ihn zuerst mit einem sehr schnellen Aufzeichnungsverfahren, einem zusaetzlichen ASCII-Zeichensatz und einer seriellen Steuerung fuer Drucker - und anderen netten Dingen. All das wurde jedoch aufgegeben, um mit den Sinclair-Geraeten kompatibel zu sein, was die darstellbaren Zeichen, den Drucker und die Cassetten-Aufzeichnung angeht. Hoffentlich war das eine richtige Entscheidung - sonst ware die Muehe umsonst gewesen.

KAPITEL VII

Der Assembler

Das wesentliche Element des ASZMIC ist der Assembler. Bevor wir uns an dessen Funktionen wagen, sehen wir uns an, was ein Assembler ganz allgemein tut.

Beim BASIC ist es egal, auf welchem Prozessor es installiert ist : die Befehle, die wir programmieren muessen, sind die gleichen. Der BASIC-Interpreter setzt die BASIC-Befehle dann in Anweisungen fuer den Prozessor um.

Wenn man seine Programme mit hoher Geschwindigkeit abarbeiten lassen will oder sie vielleicht in ein EPROM brennen moechte, muss man sie so geben, dass der Prozessor sie sofort versteht. Weil es aber nicht sehr anregend ist, einzelne Bytes in den Speicher zu befoerdern (wie man es tut, um die beruechtigten REM-Zeilen beim ZX 80/81 mit MAschinencode zu fuellen), schreibt man in einer Sprache, die "Assembler" heisst.

Dann nimmt man ein Programm, das auch durchaus in BASIC geschrieben sein kann. Dieses Programm ist in der Lage, in Assembler geschriebene Programme in Maschinencode zu uebersetzen. Diesen Arbeitsvorgang nennt man "Assemblieren". Sie muessen also beachten, dass "Assembler" nicht nur ein Programm ist, sondern auch eine Programmiersprache.

Diese Programmiersprache gibt allen Befehlen des Prozessors Namen, zum Beispiel

```
LD  A,0
```

Das heisst : "Lade den Akkumulator mit :00."

Die Befehle des Assemblers sind also genau jene, die der Prozessor selbst hat ! Um sich die Arbeit zu erleichtern, kann man auch Luxus einbauen und sich Macros und aehnliches definieren. Der Assembler muss dann die Macros in Befehle fuer den Prozessor umsetzen. Es gibt aber auch Befehle - wie das schon vorgestellte "ORG" - die der Prozessor gar nicht verstehen kann. Diese Anweisungen sind Direktiven fuer den Assembler und sagen diesem, WIE und WO das Programm fuer den Prozessor arbeiten soll.

Um Schreibarbeit zu sparen, sind die Assembler-Befehle Abkuerzungen, die mehr oder weniger verstaendlich die Funktion des Befehls beschreiben. Weil die Prozessoren und darum auch die Assembler in den USA entwickelt wurden, handelt es sich um Abkuerzungen englischer Worte. Man nennt sie "Mnemonics", das soll heisen, dass sie an die Funktion erinnern. Abgesehen davon, dass das oft sehr geschmeichelt ist, ist schon der Ausdruck "Mnemonics" eine auesserst stupide Erfindung !

Symbole

Man muss dem Prozessor nicht nur Befehle geben koennen, sondern auch Daten in den Speicher schreiben zu koennen - so, wie man es mit DATA-Anweisungen im BASIC tut. Ob man die Zahlen hexadezimal oder dezimal eingibt, ist unwichtig - Hauptsache ist, dass man nicht gezwungen ist, Bitmuster als Binaerzahlen eingeben zu muessen.

Wie Sie schon frueher in diesem Buch gesehen haben, kann man Felder verwenden. Diese werden vom ASZMIC fuer den Prozessor brauchbare Bitmuster umgesetzt. Man kann auch zwangsweise angeben, welchen Wert

eine Konstante haben soll.

Es gibt eine besondere Variable. Ihr Symbol ist das $-Zeichen (Dollar). Nehmen wir ein Beispiel :

```
PENG   JP   PENG
```

Das ist eine Endlosschleife. Der Befehl "JP PENG" zwingt den Prozessor, an die Adresse PENG zu springen. Das ist genau die Start-Adresse des Befehls, der gerade ausgefuehrt wird. Der ASZMIC assembliert diese Zeile und fuer ihn ist die Adresse PENG die gerade aktuelle Adresse (fuer die der Maschinencode erzeugt wird) - er traegt sie in einer internen Variable ein. Diese "aktuelle Adresse" hat fuer den ASZMIC den Namen $, also das Dollar-Zeichen.

Wenn man statt der obigen Zeile

```
JP $
```

nimmt, so hat das die gleiche Wirkung.

Wie arbeitet der Assembler ?

Der Assembler macht 2 Durchlaeufe (englisch :"passes") und heisst darum im allgemeinen Sprachgebrauch "2-Pass-Assembler". Beim ersten Durchlauf werden alle Adressen, Konstanten und Label in einer Tabelle notiert. Beim zweiten Durchlauf werden die Befehle uebersetzt und mit Hilfe der Tabelle alle notwendigen Werte eingesetzt. Das heisst : der lauffaehige Maschinencode ("Object-Code") wird in den Speicher geschrieben. Ausserdem werden Fehlermeldungen ausgegeben und - entsprechend den Optionen - Listings erzeugt.

Es gibt Label, die nicht im neu geschriebenen Programm liegen : Wenn Sie eine Routine aus dem ASMIC verwenden (wie RDCASS), so geben Sie deren Adresse an und im Nachfolgenden den Namen (also RDCASS). Das ist dann eine "externe Referenz". Vergessen Sie die Zuweisung, so ist dieses Label immer noch vorhanden, wird aber nicht "ausgefuellt".

Optionen

Wenn sie den Asembler mit dem A-Kommando aufrufen, geben Sie ihm auch den Namen jenes Source-Files an, das assembliert werden soll. Zusaetzlich gibt es ein Feld fuer Optionen, die die Arbeitsweise vorgeben :

```
A ^ filename optionen
- - -------- --------
```

Wie Sie schon ahnen, sind die Optionen das A und O bei einem Assembler. Es geht wird nicht "einfach" Object-Code produziert, sondern es geht vor allem auch darum, Listings zu erzeugen (oder nicht) Object-Code zu erzeugen (oder nicht) etc etc.

Die Optionen sind in ein einziges Byte gepresst. Man muss sie dezimal angeben. Wie beim CHR-Befehl addiert man die dezimalen Zahlen, die den einzelnen Optionen entsprechen. Wenn Sie ein Listing erzeugen lassen wollen, nehmen Sie "1". Soll das Listing zum Drucker gesandt werden, kommt die "2" hinzu. "Listing erzeugen und drucken" bedeutet, dass die "3" (=2+1) als Kennung der Option angegeben wird.

Zum Testen, ob ein Programm ueberhaupt Sinn ergibt, brauchen Sie ein Listing - die Erzeugung des Object-Codes unterdruecken Sie am besten. Oft ist das auch eine Frage des freien Platzes im Speicher.

Es gibt auch Optionen, deren Funktion nicht sofort ersichtlich ist. Sie betreffen das Zusammenhaengen (Verknuepfen, Verketten, "Linken") von Programmen zu einem funktionsfaehigen Ganzen.

Normalerweise wird beim Start des Assemblierens die Symboltabelle auf Null gesetzt und - falls es eine noch eine im Speicher gab - ist sie jetzt verschwunden. Das kann sehr hinderlich sein, wenn man externe Referenzen benutzen will.

Der erste Durchlauf des Assemblers erzeugt die Symbol-Tabelle. Wenn man diesen Durchlauf verhindert, ergibt das Schlimmes. Kann man aber den zweiten Durchlauf zwangsweise starten und eine Symboltabelle bereitstellen, so kann man nicht nur eines, sondern auch mehrere Programme kombinieren und assemblieren. Die Verknuepfung eines Teils mit Label etc aus einem anderen Programm nennt man "Cross-Referenz", also "Kreuz-Referenz" - aber das ist ein entsetzliches Wort.

Offsets

Es gibt eine Variable, die OFFSET heisst - ihre Adresse ist im Anhang angegeben. Ihre Bedeutung fuer das Laden von Cassette wurde bereits erklaert. Beim Assemblieren hat es eine aehnliche Bedeutung. Normalerweise ist sie auf Null gesetzt, kann aber zum Verschieben eines Programms mit einem Wert belegt werden.

Wenn ein Programm assembliert wird, so geht man davon aus, dass es auch an der Adresse ablaeuft, die mit ORG vorgegeben wurde. Also wird beim Assemblieren der Object-Code an jene Adresse im Speicher befoerdert, die mit ORG angegeben wurde. Soll das Programm in einem EPROM arbeiten, das ab Adresse 0 liegt, so wuerde der ASZMIC beim Assemblieren den Object-Code im ZX 81 ab Adresse 0 ablegen - also den ASZMIC ueberschreiben wollen, was natuerlich Unsinn ist.

Also muss man den ASZMIC anweisen, den Object-Code woanders unterbringen, damit es weder einen Schaden anrichtet, noch selbst verlorengeht. Der Wert von OFFSET wird als eine Konstante bei jeder Adresse addiert - oder bei jeder Konstanten, jedem Label, die sich aus der aktuellen Adresse im Programm ergeben. Setzen Sie zum Beispiel ORG auf :0 und OFFSET auf :7000. Dann wird der Object-Code, der ab :0 lauffaehig ist ab :7000 in den speicher geschrieben oder kann von Cassette dorthin gelesen werden.

Das Zuweisen eines Wertes fuer OFFSET bedeutet auch, dass Sie OFFSET hinterher wieder restaurieren : War es vorher :0, so muss es wieder auf :0 gesetzt werden.

Der Assembliervorgang

1. Sie haben Source-Code geschrieben oder eingelesen - er befindet sich im Computer.
2. Der Source-Code entspricht den Konventionen :
   a) Er beginnt mit einem Filenamen, dessen erstes Zeichen die Filemarke ist.
   b) Es wird durch eine Filemarke beendet.
   c) Die Assembler-Befehle des Source-Code entsprechen den von ZILOG und MOSTEK definierten Konventionen (Ausnahme : die EQU-Zuweisung und die Schreibweise :0000 statt 0000H bei hexadezimalen Zahlen) :
      Kurz :
      - Label stehen ab der ersten Spalte in einer Zeile.
      - Beginnt eine Zeile mit einem Blank, so gilt sie als Befehlszeile.
      - Nach einem Label muss mindestens 1 Blank stehen.
      - Nach dem Label (oder einem fuehrenden Blank) steht der Befehl in Assembler

- Wenn der Befehl oder Zuweisungen eine oder mehrere Operanden erfordern, stehen diese nach dem Befehl.
- Bei zwei Operanden werden diese durch ein Komma getrennt. Es gibt keine vorlaufenden oder nachfolgenden Blanks.
- Eine Befehlszeile wird mit einem <Newline> beendet.
- Ist in einer Zeile ein Semikolon enthalten, so werden alle Zeichen nach diesem Semikolon als Kommentar betrachtet und vom Assembler ignoriert.
- Das Semikolon fuer Kommentare kann in der ersten Spalte stehen oder nach dem Assembler-Befehl.
- Wenn ein Label, das in Spalte 1 beginnt, mit einem "=" (Gleichheitszeichen) beendet wird, ist das eine EQU-Zuweisung. Folglich muss dahinter ein Argument stehen, das ausgewertet und diesem Label zugewiesen werden kann.

Beachten Sie, dass der Assemblier-Vorgang mit einem RST 0 zurueckkehrt, also die letzte Adresse auf den Stack befoerdert. Das erlaubt Ihnen, mit dem O-Kommando das IX- und das IY-Register anzusehen - dort ist die Schluss-Adresse des Object-Codes aufbewahrt. Das ist keine besondere Funktion, sondern ergibt sich aus der Konstruktion des ASZMIC : die letzte Adresse ist nichts weiter als die aktuelle Adresse, die der Assembler stets mitfuehrt !

3. Sie rufen den Assembler auf mit

```
A ^ filename optionen
- - -------- --------
```

Anweisungen

Um dem Assembler Angaben ueber die Startadresse des Object-Codes zu machen, gibt es die ORG-Anweisung.

Eine andere Anweisung ist =. Bei ZILOG oder Mostek stuende hier EQU. Damit wird einer Variablen ein Wert zugewiesen :

```
                label=wert

Beispiel :      PENG=:4000
```

Ausserdem gibt es 3 weitere Anweisungen. Sie erzeugen im Object-Code Daten, also keine ausfuehrbaren Befehle.

```
DEFB wert
DEFW wert
DEFM text
```

DEFB (DEFine Byte) erzeugt ein Byte, DEFW (DEFine Word) 2 Byte.

Lassen Sie sich nicht durch diese Bezeichnungen irrefuehren ! Sie sind vollkommen willkuerlich, auch wenn selbsternannte Spezialisten Ihnen einen Riesensermon erzaehlen, das sei Standard und "das weiss man". Es gibt naemlich ueberhaupt keinen Standard fuer diese Dinge !

Bei DEFB :00 ist alles einfach. Die Zuweisung erfolgt 1:1. Bei DEFW wird durch die Struktur des Z80 erzwungen, dass bei

```
label DEFB :1234
```

im Speicher bei steht

```
label     :34
label+1   :12
```


Also : [illegible] :34 :12

Das ist kein Fehler, sondern beruht darauf, dass der Z80 erst das niederwertige Byte liest und dann das hoeherwertige. Beim Lesen wird mit der unteren Adresse begonnen - und dort steht ganz richtig :34 !

DEFM (DEFine Message) setzt den nachfolgenden Text direkt ein. Das erleichtert den Einbau von Meldungen. Man muss also nicht jeden Buchstaben eines Texts einzeln codieren, sondern kann die ganze Meldung direkt als lesbaren Text eintippen.
Der Text muss in "" Gaensefuesschen stehen. Das bedeutet, dass im Text selbst keine Gaensefuesschen stehen duerfen. Sie koennen jedoch sehr einfach mit DEFB erzeugt werden : lassen sie Ihren Text im DEFM bis zum " laufen und nehmen dann DEFB, Dann geht's mit DEFM weiter.

Die ORG-Anweisung hat einen kleinen Nebeneffekt. Mit ORG wird die Startadresse des Programms vorgegeben. Man kann aber eine Expression einsetzen ! Geht man einen Schritt weiter, so kann man mit dem $ Dollar arbeiten :

```
ORG $+20
```

In diesem Beispiel wird mit dem $ die aktuelle Adresse in eine Expression eingebaut. Beim Start muss unbedingt eine absolute Adresse vorgegeben werden. Danach steht der Wert fuer $ fest. Wenn Sie mitten im Programm sind und "ORG $+20" anweisen, dann ueberspringt der Assembler eine Luecke. Da koennen dann andere Teile eingebaut werden. Das entspricht dem DEFS (DEFine Space = definiere einen Platz) von ZILOG. Bei ORG- oder =-Anweisungen koennen keine Vorgriffe gemacht werden auf spaetere Zuweisungen oder Berechnungen in Expressions.

Kommentare

Kommentare sind eine lebensnotwendige Angelegenheit : Schreiben Sie ein Programm und Sie werden bei der x-ten Zeile nicht mehr wissen, womit sie begonnen hatten. Wie schon beschrieben, folgen Kommentare nach einem Semikolon. Eine besondere Leistung des ASZMIC besteht darin, ueberlange Zeilen zuzulassen. Beim Drucken eines Listings werden nur 32 Zeichen pro Zeile zugelassen - der Rest wird abgeschnitten. Weil der erzeugte Object-Code und die Adresse davor in einer Zeile 16 Zeichen benoetigen, bleibt nicht viel uebrig ! Man kann jedoch eine Option benutzen und Bit 5 von ASSFLG die Begrenzung aufheben. Das setzt gleichzeitig die Zeilenbreite des Druckers auf 64 Zeichen.

Weil der ZX 81 nur wenig Speicher hat und ein Schnecken-Interface zum Cassetten-Recorder, bietet die Hardware wenig Hilfe fuer eine gute Kommentierung innerhalb eines Programms. Trotzdem sollten Sie versuchen, wenigstens ein Minimum an "Dokumentation" einzubauen. Sie werden das zu schaetzen lernen !

Fehler

Das ASZMIC-ROM ist nur 4k gross. Um das Programm darin unterbringen zu koennen, wurden grosse Anstrengungen unternommen. Fuer lange Routinen und Texte zur Analyse von Fehlern und deren Meldung. So blieb nur die Moeglichkeit des "elektronischen Darwinismus" :

"Nur ein Programm, das stimmt, ueberlebt auch"

3 Typen von Fehlern werden vom ASZMIC erkannt :

1. Label werden falsch geschrieben
   ein Label wird doppelt vergeben
   ein Label wird vergessen

2. fuer einen relativen Sprung an ein bestimmtes Label muesste eine zu grosse Distanz uebersprungen werden

3. Ihnen unterlaufen Fehler bei den Assembler-Befehlen (Sie schreiben zum Beispiel in die erste Spalte, wo nur Label beginnen duerfen.)

Fehler werden durch angezeigt, indem die 1. Spalte einer Zeile durch ein Zeichen belegt wird, das kein Blank ist. Wenn Sie kein Listing produzieren, wird die Zeile zumindest angezeigt.

Die Art des Zeichens in der 1. Spalte koennte Ihnen eine Menge ueber den Fehler sagen - aber die Ablaeufe sind so kompliziert, dass es leichter ist, durch logisches Denken hinter die Ursachen zu kommen.

1. Schauen Sie nach, ob ein Object-Code in der Fehler-Zeile erscheint. Wenn nicht, dann duerfte es sich um einen falschen Assembler-Befehl handeln oder der Befehl beginnt in der 1. Spalte oder er wird mit einem Komma anstelle eines Blanks beendet. Wegen seiner besonderen Konstruktion kann der ASZMIC nur einen Teil der falsch geschriebenen Asembler-Befehle erkennen.

2. Wenn es in der Anweisung einen relativen Sprung gibt, dann ist es fast sicher, dass die maximale Reichweite eines Sprunges ueberschritten wurde.

3. Schliesslich bleiben : doppelte Definitionen, nichterfuellte Referenzen fuer Label. Also durchsuchen Sie den Source-Text, ob das Label ueberhaupt definiert wurde, ob es doppelt definiert wurde.

Zusaetzlich meldet sich der ASZMIC, wenn es bei einer Zahl ein falsches Zeichen fuer die Zahlenbasis gibt.

Der Arbeitsablauf bis zu einem fertigen Programm

1. Editieren Sie das Programm
2. Sichern Sie es sofort auf Band
3. Assemblieren Sie mit Option 64
4. Wenn Fehler auftauchen, finden Sie diese heraus und fangen wieder von vorne an : ab zum EDITor !
5. Es wurden keine Fehler gemeldet :
   Assemblieren Sie nochmals, dieses Mal mit Option 65 (67, wenn Sie einen Drucker haben).
6. Pruefen Sie das Listing, ob auch wirklich das herausgekommen ist, was Sie haben wollten. Wenn nicht : wieder bei 1 anfangen !
7. Assemblieren sie nochmals : dieses Mal ohne Optionen
8. Pruefen Sie die Funktionen des Programm durch : jede einzelne !!!
   Dazu nehmen Sie den DEBUG. Bei Fehlern : weiter bei 1.
9. Sichern Sie den Object-Code auf Band, gegebenenfalls auch die Symbol-Tabelle (Denken Sie an das Feld beim L-Befehl !)

Listings

Listings sind eigentlich unnoetig - wenn das Programm funktioniert. Genau : WENN es funktioniert. Bis dahin ist es ein weiter Weg und die Listings sind das wichtigste Hilfsmittel zum Debuggen. Haben Sie keinen Drucker, erwartet Sie ein hartes Leben. Ausserdem braucht man Listings, um Programme aneinander anpassen zu koennen !

Der ASZMIC unterstuetzt Sie bei der Arbeit : Er erlaubt das Ausgeben von Listings auf den Bildschirm. Darin koennen Sie Filenamen und Filemarken editieren und das Ganze saven und laden. Der EDITor reicht aus, das Bildschirm-Listing als Notizblock zu benutzen und trotzdem weiterzuentwickeln. Ein Bildschirm-Listing kann dennoch niemals ein Listing-auf-Papier ersetzen.

Das Assembly-Listing besteht aus 4 Segmenten :

Der erste ist das "Single-Byte Error-Flag". Sie kennen es : es steht in der 1. Spalte.
Dann folgt die Adresse des Object-Codes : 4 Bytes in hex.
Die 1 bis 4 Bytes Object-Code bilden das naechste Segment. DEFM kann uebrigens 5 Bytes erzeugen, die nicht der Begrenzung zum Opfer fallen.
Den Rest bilden so viele Zeichen des urspruenglichen Assembler-Befehls, wie jetzt noch in eine 32 Zeichen breite Zeile passen.

Das Listing - oder die Fehlermeldungen, wenn die Optionen kein Listing vorsehen - wird zum Drucker geschickt, wenn die Option mit Bit 1 den Drucker als Ziel angibt.

Bibliotheken

Am Anfang haben Sie noch keine Uebung und jedes Programm ist fuer Sie etwas Neues. Spaeter stellen Sie dann fest, dass Sie das Rad immer neu erfinden muessen und viel leichter ist, sich hin und wieder Teile von alten Programmen in neue einbauen zu lassen. Eine Methode besteht darin, alle Sources auf Band zu haben und das Brauchbare mit Merge in das neue Programm einzulesen. Dann wird assembliert und alles laeuft gut.

Ein anderer Weg unterlaesst das Assemblieren und nimmt gleich den fertigen Object-Code. Das kostet vor allem beim Editieren und Assemblieren viel weniger Platz, erfordert aber eine gute Dokumentation !

Eine Bibliothek besteht aus Moduln, so zum Beispiel Drucker-Ansteuerungen, Bildschirm-Ausgaben, Routinen fuer bestimmte Berechnungen etc etc. Manche Moduln haben Unterprogramme, manche nicht. Erfordert ein Modul Unterprogramme oder das Vorhandensein eines bestimmten 2. Moduls, so muss man diese Unterprogramme und zusaetzliche Moduln natuerlich auch laden. Ein Beispiel ist ein Druckertreiber. Er erfordert die Umsetzung in ASCII-Zeichen. Schreiben Sie einen Drucker-Treiber fuer verschiedene Zeichensaetze - und es kann immer nur ein Zeichensatz im Speicher vorhanden sein - so ist folglich die Tabelle fuer den Zeichensatz ein Modul und der Drucker-Treiber muss stets zusammen mit einem dieser Moduln zusammengefuegt werden. Es gibt Programme, denen man sagt, welchen Zeichensatz man will und welchen Drucker man will und und und .. Dann stellt das Programm automatisch die richtigen Treiber-Routinen und die Tabellen zusammen. Das nennt man "Generieren" und das Programm dazu ist ein "Generator". Alle erforderlichen Teile holt sich der Generator aus einer Bibliothek. Der ZX 81 ist leider zu klein, um solchen Luxus zu ermoeglichen.

Das Arbeiten mit Moduln erfordert, dass es zu jedem Modul auch immer eine Symbol-Tabelle gibt. Sie muessen also beim Assemblieren eines Moduls nicht nur den Object-Code, sondern auch diese Tabelle auf Band sichern. Wollen Sie dann ein Programm assemblieren, das Moduln mitbenutzt, so sind deren Symbol-Tabellen vor Assemblierung zu laden - sonst findet der Assembler die Referenzen nicht ! Der Assemblierungslauf muss mit Option 4 (Bit 2 gesetzt) erfolgen.

Zum Ablaufen des gesamten Programms muessen Sie alle Moduln laden - entweder einzeln oder als ein zusammenhaengendes File.

Zu grosse Programme

Wenn die Programme immer leistungsfaehiger werden - werden sie auch immer laenger. Schliesslich sind sie so lang, dass sie nicht mehr als ein Stueck assembliert werden koennen. Dann zerlegt man sie in Segmente. Je logischer diese aufgebaut sind, um so einfacher ist der Umgang damit.

Nehmen wir an, Sie haetten ein Programm in die Segmante A, B und C zerlegt, bzw von Anfang an diese 3 getrennte Segmente geschrieben.

Jetzt beginnen Sie mit der Assemblierung von Segment A. Den Object-Code brauchen wir nicht - er stoert nur und wird daher ueber die Option Bit 6 unterdrueckt. Das liefert Ihnen eine Symbol-Tabelle und ein paar Fehlermeldungen.

Nun muss Segment B - das direkt auf A folgen soll - richtig im Speicher liegen. Mit einer festen ORG-Anweisung waere das moeglich, aber viel zu umstaendlich. Wir brauchen nur in die letzte Zeile von Segment A zu schreiben :

```
ENDA=$
```

Diese Zuweisung ergibt keinen Code. Sie erzeugt aber ein Label in der Symbol-Tabelle. Wenn Sie Segment B mit der Anweisung

```
ORG ENDA
```

beginnen, muss eine Symboltabelle dieses ENDA enthalten und schon ist das Problem geloest.

Segment B muss analog enden mit ENDB=$ und so im Segment C den Beginn mit ORG ENDB ermoeglichen.

Also assemblieren Sie anschliessend Segment B mit Option Bit 6 (kein Object-Code) und mit Option Bit 2 (Symboltabelle nicht neu anlegen, sondern ergaenzen).

Dann fahren Sie fort mit Segment C.

So waechst Ihre Symboltabelle. Wenn Sie jetzt fuer Segment B die Source laden und mit der Symbol-Tabelle assemblieren (mit Bit 7 =2. Durchlauf erzwingen und mit Bit 2 = vorhandene Symbol-Tabelle verwenden) erhalten Sie das erste Segment des Object-Codes. Dieses Segment muessen Sie aus dem Speicher auslagern, also auf Band sichern.

Dann verfahren Sie mit Segment B und mit Segment C ebenso. Zum Schluss haben Sie 3 Segmente und eine grosse Symbol-Tabelle, alle 4 einzeln auf Band. Die Segmente laden Sie dann und haengen Sie aneinander.

Weil anfangs laufend Referenzen auf Adressen etc in anderen Segmenten gemacht wurden, gab es Fehlermeldungen. Die interessieren nicht, weil die Symbol-Tabelle ja erst zusammengesetzt werden musste. Mit dieser Tabelle wurden dann die Segmente des Object-Codes erzeugt. Der dritte Arbeitsgang, das Laden der Segmente, ergibt dann ein ablauffaehiges Programm.

KAPITEL VIII

Das Testen von Programmen und das Abarbeiten

Wenn ein Programm editiert und assembliert worden ist, muss es getestet werden und schliesslich auch seine Dienste tun. Das Testen und Beseitigen der Fehler nennt man "debuggen" ($) Der ASZMIC enthaelt DEBUG-Funktionen, die nun beschrieben werden sollen.

($) Anmerkung : Ein Kaefer heisst auf Englisch "bug". Bei einem der damals wie auch heute recht zahleichen Zusammenbrueche von Rechnern wurde "anno Tobak" in einem Computer ein grosser Kaefer entdeckt, der einen Kurzschluss verursacht hatte. Als Ausfallursache wurde darum "bug" (Kaefer) angegeben. "Debugging" heisst "entkaefern" und ist ein Jargon fuer die Beseitigung von Fehlern, sowohl in der Hardware, als auch in der Software. Weil dieser Begriff dem Nicheingeweihten voellig mysterioes erscheint (wer kann schon eine Verbindung zwischen einem Kaefer und einem Computer herstellen ?), wird in vielen Buechern die Kaefergeschichte erzaehlt. Wenn Sie sie schon gekannt haben - bitte verzeihen Sie uns.

Der J-Befehl

Das ist der wichtigste Befehl zum Testen. Sie koennen ein Feld mit einer Adresse angeben. Der ASZMIC betrachtet das als Startadresse des Programms und springt dorthin. Der J-Befehl ohne Feld geht davon aus, dass in PC1 eine Adresse abgelegt wurde, als zuletzt eine BREAK-Bedingung erfuellt wurde. An diese Adresse wird gesprungen.

Das Besondere des J-Befehls ist, dass alle Register im Prozessor aus dem Inhalt von REGIM, dem Abbild der Register im Speicher, gefuellt werden - und dann in das Programm gesprungen wird. Ferner wird das I-Register auf 1 gesetzt, um die Abarbeitung eines NMI und BREAK-Bedingungen zu erlauben. Wenn Ihr Programm keine Vorbelegung der Register vornimmt, koennen Sie das Abbild der Register im Speicher vorbelegen und dazu den I-Befehl oder den D-Befehl nehmen.

Das geht alles geradeaus. Wenn Sie mit J in Ihr Programm springen und dieses Programm hat einen Fehler, dann haben Sie schlechte Karten, es sei denn Sie haben ein Taste, um mit einem NMI-Impuls die Rueckkehr in den ASZMIC erzwingen zu koennen.

Breakpoint

Ihre erste Bastion gegen ein selbstmoerderisches Programm ist der Breakpoint.
Ein Breakpoint ist der Maschinen-Code fuer "RST 0". (Das gilt aber nur hier beim ASZMIC ! Bei einem anderen Programm kann die Methode ganz anders funktionieren !) Er kann ueber den B-Befehl eingefuegt werden. Dabei uebernimmt der ASZMIC die Wiederherstellung des urspruenglichen Codes und er manipuliert den gesicherten PC (Programm-Zaehler), damit Sie mit dem Befehl weitermachen koennen, wann Sie wollen. Sie koennen auch mit dem D-Befehl (DUMP) direkt im Speicher einsetzen. Den originalen Object-Code im RAM koennen Sie natuerlich auch modifizieren. Das erfordert kompliziertes Ueberspringen oder Ersetzen von Befehlen.

Beim normalen Gebrauch des Breakpoints wird das Programm in logische Ablaeufe zerlegt. Beginnen Sie am Anfang des Programms und setzen einen Breakpoint an das Ende der ersten Befehlsfolge. Wenn Sie das Programm ausfuehren und auf den Breakpoint stossen, koennen Sie mit dem D-Befehl und dem O-Befehl die Register und den Speicher ansehen.

Dann versetzen Sie den Breakpoint an das Ende des naechsten logischen Befehlablaufs und machen mit J weiter. Das machen Sie so lange, bis etwas schiefgeht. Der Fehler muss dabei nicht unbedingt in jenem Teil liegen, in dem der Fehler sich auswirkte - er kann auch schon vorher Fallstricke ausgelegt haben, in die Ihr Programm viel spaeter geraet.

Denken Sie daran, dass beim Erreichen des Breakpoints dieser automatisch weggeraeumt wird und der tatsaechliche Object-Code wieder eingesetzt wird ! Wenn Sie also an der gleichen Stelle noch einmal einen Breakpoint haben wollen, muessen Sie mit G B und J durch den Breakpoint, ihn restaurieren und die Befehlsausfuehrung wiederholen.

Wenn Sie einen Breakpoint gesetzt haben und ein neues Programm nachladen, dann wird ein folgendes Versetzen des Breakpoints den Object-Code, also das Byte, das noch vom alten Programm stammt, in Ihr neues Programm einsetzen !

Bedenken Sie, dass Sie beim Single-Stepping (Einzel-Schritt) nicht durch einen Breakpoint koennen, weil Sie in die Routine zur Bearbeitung des Breakpoints geraten.

Der G-Befehl

Wenn Sie das Programmtueck gefunden haben, in dem der Fehler auftritt, koennen Sie es im Detail untersuchen. Dazu nehmen Sie den Single-Step, also das G-Kommando des ASZMIC. Es hat die gleiche Wirkung wie der J-Befehl, unterbricht aber nach jedem ausgefuehrten Object-Code-Befehl. Eine Breakaktion sichert automatisch den Inhalt der Register im Abbild der Register im Speicher, also REGIM. Wenn Sie nicht nur einen, sondern mehrere Schritte ausfuehren lassen wollen, so geben Sie die gewuenschte Zahl an. Dann muessen Sie im G-Kommando aber nicht nur die Zahl angeben, sondern auch die Adresse, die angesprungen werden soll - selbst dann, wenn diese Adresse schon im PC1 gespeichert ist !

Der Schritt-Zahler fuer das Single-Steppen ist sehr nuetzlich. Merken Sie sich, dass trotz des mitlaufenden Zaehlers bei jedem ausgefuehrten Object-Code-Befehl ein Break ablaeuft und damit automatisch REGIM immer auf dem neuesten Stand gehalten wird. Darum ist die Ablaufgeschwindigkeit nicht sehr hoch. Weil der Single-Step den mit dem NMI verknuepften Zaehler des ZX 81 verwendet, werden Breakpoint und die Unterbrechungen beim Single-Step durch die gleiche Routine erledigt. Das laesst auch die Moeglichkeit zu, einen Breakpoint im ROM zu simulieren. Wenn Sie ein selbstgebrautes EPROM haben, das ab :2000 laeuft und Sie beispielsweise bei :2122 unterbrechen wollen, koennen Sie mit

B :2122-1

und

G :2000 32767

die Routine im EPROM anspringen. Der ASZMIC wird dann durch das EPROM single-steppen, bis er die Instruktion vor der Adresse :2122 ausgefuehrt hat. Dann meint er, einen Breakpoint erreicht zu haben und hoert mit dem Single-Steppen auf.

Das G-Kommando benutzt die auf dem NMI basierende innere Uhr des ZX 81 auf eine besondere Weise. Darum ist es nicht ganz brauchbar fuer Routinen, die die Bildschirmanzeige steuern. Das erfordert ein I-Register mit dem Inhalt 14 und einen freigegebenen maskierbaren Interrupt.

Ein wenig Scharfsinn im Umgang mit Breakpoints und dem Single-Stepping ueber unkritische Teile wird Ihnen ueber diese Klippen hinweghelfen.

Der O-Befehl

Dieser Befehl zeigt den Inhalt des Abbilds der Register im Speicher an. Beim Testen ist er sehr wertvoll - er ist Ihr einziges Mittel, die Register zu sehen. Sonst muessten Sie sich extra eine Routine schreiben, die den Inhalt eines Registers in RAM befoerdert, wo Sie es sich dann mit DUMP ansehen koennen.

Die Register erscheinen in 2 Zeilen :

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

Jene Register mit dem ' Strich sind die Hintergrund-Register. Der PC ist der Programm-Zaehler. Er zeigt auf den naechsten auszufuehrenden Befehl. Der SP ist der Stack Pointer und der Rest ist der normale Registersatz.

Weil nicht genug Platz im ROM ist, musste auf eine Ueberschrift ueber der Anzeige verzichtet werden.

Der O-Befehl wird am sinnvollsten in einem SHIFT-Macro untergebracht : Geben Sie "SHIFT T" ein (TOP) und scheiben O in die Kopfzeile. Bei jeder Break-Bedingung, sei es Single-Step oder Breakpoint, wird automatisch der Registerinhalt angezeigt.

Wenn Sie wichtige Variablen haben, die Sie ebenfalls bei jedem Breakpoint anzeigen wollen, koennen Sie Kommandos in der SHIFT-Macro-Zeile zusammenhaengen. Dazu gibt es das

;/

Semikolon mit Schraegstrich dahinter als eine Trennung der einzelnen Befehle. Die Befehle beginnen gleich hinter jedem / Schraegstrich.

Diese Option ist eigentlich fuer das DUMPen gedacht, aber die meisten DEBUG-Befehle werden ebenfalls so funktionieren. Der I-Befehl geraet allerdings in eine Endlosschleife.

Der I-Befehl

Wenn Sie eine Zeile in Assembler eingeben, so uebersetzt der I-Befehl sie und fuehrt sie sofort aus. Vor der Ausfuehrung muss der Prozessor in den Betriebszustand des zu testenden Programms gebracht werden. Dazu wird automatisch eine Art J-Kommando ausgefuehrt und die Register geladen. Nach der Ausfuehrung des Befehls wird mittels eines Breakpoint erzwungen, dass die Register wieder als Kontext des Programms in das Abbild der Register im Speicher gerettet werden. Mit dem I-Befehl wird also automatisch der Kontext Ihres Programms aufgebaut.

Unter der Voraussetzung, dass die originale Symbol-Tabelle noch intakt ist, koennen Sie mit dem I-Befehl Label angeben, so wie auch in den anderen DEBUG-Befehlen. Der einzige Pferdefuss ist, dass der I-Befehl den gespeicherten PC (Programm-Zaehler) aendert. Sie muessen sich also fuer J und fuer G die spezielle Adresse merken, an die Sie springen wollen. Haben Sie naemlich den I-Befehl ausgefuehrt und rufen J oder G ohne Adresse auf, landen Sie naemlich im Stack des ASZMIC und fuehren den als Object-Code aus ...

Der I-Befehl ist ein praktisches Werkzeug, um unwesentliche Unterlassungen zu korrigieren und mit dem Wesentlichen weitermachen zu koennen.

Mit dem I-Befehl weist man normalerweise Registern bestimmte Werte zu. Es kann aber jeder ausfuehrbare Befehl gegeben werden. Wenn Sie kein

Blank haben zwischen dem I und dem Beginn des Assembler-Befehl, wird angenommen, dass Sie das Statement mit einem Label [illegible]onnen haben, das Sie mittels =-Anweisung definieren. Der einzige Haken besteht nun darin, dass die Logik der Definition eines Labels ein <Newline> als Trenner benutzt und so wird das "I" als Teil des Labels angesehen. Sie koennen also jedes Label verwenden - es muss nur mit "I" beginnen.

Die Befehle D, F und C

Diese Befehle arbeiten geradeaus und sie stellen nichts Besonderes dar. Jedes Debug-Programm muss die Inhalte eines Speichers anzeigen und modifizieren koennen. Das Fuellen mit einem bestimmten Bitmuster ist gut, um gewisse Arbeitsbereiche vorzubelegen. Der C-Befehl, dess wesentliche Aufgabe das Verschieben und Kopieren ist, kann auch zum Retten von Daten oder Programmstuecken sein : Es gibt ein "Original", von dem man immer wieder in den eigentlichen Arbeitsbereich kopiert, wenn durch den Programm-Ablauf die Daten oder Programmteile ueberschrieben wurden und nun zum Testen wieder rekonstruiert werden muessen. Der F-Befehl und der C-Befehl koennen den gesamten Speicher zerstoeren. Seien Sie also sehr vorsichtig und achten ganz besonders darauf, Zeilen richtig einzugeben, bevor Sie sie mit <Newline> "abschicken".

KAPITEL IX

Graphik

Der ASZMIC beherrscht hochaufloesende Graphik. Im BASIC-ROM ist ein Zeichen-Generator enthalten, bei dem eine feste Anzahl von Rasterpunkten pro Zeichen vorgegeben ist. Das schraenkt die graphischen Faehigkeiten erheblich ein, es sei denn man umgeht das Problem mit einer Hardware-Erweiterung.

Der ASZMIC hat einen programmierbaren Treiber, bei dem Sie Parameter veraendern koennen. Ausserdem wurden die Graphik-Zeichen des Sinclair-Zeichensatzes ersetzt durch einen von der Hardware-Rasterung unabhaengigen Zeichensatz.

Als Ergebnis koennen Sie aus einem Maschinenprogramm in einer Matrix von 244 x 144 Punkten plotten, ein stehendes Bild auf dem Bildschirm anzeigen (nur beim ZX 81 und beim ZX 80 mit eingebautem Slow-Modus) und dabei immer noch Zeit fuer andere Befehle haben, die der Prozessor ausfuehrt.

KERNEL = Der asynchrone Treiber

In den Anwendungsbeispielen wird eine KERNEL-Routine (kernel = Kern) aufgefuehrt. Sie erzeugt ein leeres Display-File (die Grundstruktur) und zeigt es 50 mal in der Sekunde auf dem Bildschirm. Zusaetzlich hilft sie bei Synchronisierungen und dem Lesen der Tastatur. Solange sie nicht arbeitet, kann ein Anwenderprogramm die Kontrolle uebernehmen.
Vom Standpunkt des Benutzers aus muss sein Programm nur den Inhalt des Display-Files bearbeiten - den Rest macht die KERNEL-Routine.

Bei einem ZX 80 ohne eingebauten Slow-Modus kann die KERNEL-Routine nicht benutzt werden, sondern man muss selbst das Display erzeugen, mit OFRMl ausgeben und das Timing und die Bild-Synchronisierung (Frame-Sync) steuern.

Wenn Sie bei einem ZX 81 die KERNEL-Routine benutzten, darf Ihr Programm so arbeiten, als ob der Mechanismus der Anzeige unsichtbar waere.

Betrachten Sie die KERNEL-Routine nicht als unumstoessliches Heiligtum. Diese Routine und auch die Beispiele fuer Graphik wurden in kurzer Zeit geschrieben, um Ihnen die Moeglichkeiten demonstrieren zu koennen.

Vielleicht koennen Sie selbst etwas Besseres entwickeln. Beginnen Sie doch und variieren NNN und IDLE. Das sind die Raster am Fuss des Rahmens und die Laenge des Sync-Pulses.

PLOT = Das Plotten

Die PLOT-Routine in den Anwendungsbeispielen wird durch einen Unterprogramm-Aufruf mit CALL bedient : Das B-Register muss dazu die X-Koordinate und das C-Register die Y-Koordinate enthalten. Nullpunkt ist die linke untere Ecke des Bildschirms.

Der Inhalt des fuer diesen Punkt zustaendigen Bytes wird ausgewertet, dann der neue Punkt eingefuegt und das Byte wieder zurueckgeschrieben.

UNPLOT hat die gleiche Funktion, beseitigt aber einen Punkt.

Es wird nicht nachgeprueft, ob der Punkt ueberhaupt innerhalb der zulaessigen Grenzen liegt. Das muessen Sie selber einbauen. Zur Berechnung der Linien-Adresse wird geshiftet und subtrahiertund bei dem betreffenden Byte wird (wegen der 4 Punkte = Pixel, die man darin unterbringen kann) das hoechstwertige Bit benutzt.

LINE = Linien ziehen

Zum Ziehen einer Linie zwischen 2 Punkten gibt es auch eine Subroutine. Sie verwendet PLOT. Wenn die Linie zwischen (X,Y) und (X',Y') gezogen werden soll, muss man beim Aufruf von LINE die Register so vorbelegen :

D = Y B = Y'
E = X C = X'

Es wird schrittweise hochgezaehlt, um die benoetigten Punkte zu berechnen. Die Schrittweite wird maximiert, um ueberfluessiges PLOTten zu vermeiden.

LINE funktioniert - Sie koennen aber sicherlich andere Methoden finden, um schoenere Linien zu bekommen. Das gilt vor allem fuer kleine Winkel zwischen der Linie und einer der beiden Achsen. Gepunktete Linien koennen viel besser aussehen, weil das menschliche Gehirn diese Punkte automatisch als feiner feiner gerastert betrachtet und zu einer Linie verbindet. Derartige Programme wurden jedoch weggelassen, weil es als Aufgabe betrachtet wurde, Werkzeuge fuer diese Dinge zu konstruieren, nicht die Anwendung selbst.

ULINE loescht eine Linie.

UPROCS = User PROGram

Hinter diesem seltsamen Namen verbergen sich 2 Anwenderprogramme. Eines, STRUCTURES, malt Punkte und MOIRE nimmt Linien.

Bilder koennen auf dem Schirm verschoben und gedreht werden.

Eine Verschiebung im RAM um weniger als 37 Bytes ergibt eine gerade Bewegung in der X-Richtung, ein Mehrfaches von 36 bewegt in Y-Richtung. Das ist wesentlich einfacher als peinlichstes Loeschen und Neu-Zeichnen.

Sie koennen auch Text und Graphik mischen !

Der Text ist in 8 Linien hohen Zeichen zu schreiben. Daher muessen Sie ein Raster von 8 Linen hohen Baendern konstruieren, um die richtige Schreibzeile fuer den Text zu finden. Innerhalb dieses Bandes sind dann die Zeichen Linie fuer Linie einzutragen.

Beachten Sie, dass eine "1" als Bit 6 die Hardware des Rechners dazu zwingt, so ein Byte als Befehl fuer den Prozessor zu betrachten und dieses Byte ausfuehren will. Das fuehrt dann von minimalen Stoerungen bis zum totalen Absturz.

Andere Loesungen

Die Hoehe eines plotbaren Punktes kann als PIXSIZE im KERNEL geaendert werden. Die Anzahl der Punkte in vertikaler Richtung koennen Sie durch Variieren von RASTERS aendern. Ueber der Anzeige gibt es auf dem Schirm eine Anzahl leerer Linien : dafuer koennen Sie TOPS aendern.

[illegible] RASTERS und TOPS und NNN ausmiert

werden und das Ergebnis um 300 liegt, werden Sie wohl immer noch ein synchronisiertes Bild auf dem Fernseher erhalten. Die billigsten und primitivsten Fernseher sind gegenueber Abweichungen im Zeitraster wesentlich toleranter als teure und hochwertige Geraete.

TOPS	leere Linien am Kopf des Bildschirms
NNN	ueberwacht Rechenzeit pro Bildschirm
RASTERS	kontrolliert die Zahl der aktiven Linien auf dem Display
PIXSIZE	bestimmt die Hoehe eines Punktes

Unterprogramme im ASZMIC

Nach dem Einschalten beginnt der ASZMIC erst mit einer Ueberpruefung des Rechners. Dazu wird Adresse :1000 gelesen. Ist dort ein :C3, also das erste Byte eines Sprungbefehls, so wird mit

CALL :1000

dorthin gesprungen. Wenn Sie ab :1000 ein EPROM einsetzen, so wird es automatisch in den ASZMIC integriert.

Das gilt bei der Version E04. Bei der Version E07 ist diese Aussprungmoeglichkeit auf :2000 geaendert worden !

DADDR	kann in die Bearbeitung eines Befehls eingreifen
PRTJMP	ist im RAM die Adresse der Drucker-Routine - kann durch die Adresse Ihrer eigenen Routine ersetzt werden
INTJMP	uebernimmt die Break-Bedingungen nach dem Retten der Umgebung (das ist das, was der KERNEL tut)
KEYJMP	enthaelt normalerweise die Adresse KEYRET. Hier koennen Sie die Adresse einer eigenen Routine zur Bearbeitung der Tastatur einsetzen und von dieser dann entweder zurueckspringen nach KEYRET oder mit mit RETurn nach LIX zurueckkehren.
SAVMEM	kann benutzt werden, um den ASZMIC neu zu initialisieren und dabei aber den Inhalt des Arbeitsbereiches zu erhalten oder um fuer mehr als 16k RAM zu initialisieren. Das HL-Registerpaar wird mit der Adresse geladen, die als Oberkante des Speichers betrachtet werden soll und dann springt man direkt mit JP nach SAVMEM. Das entspricht der Verwendung von NEW im BASIC, wenn man RAMTOP geaendert hatte.

Wenn Sie Programme debuggen, kann es praktisch sein, eine Taste zum Ausloesen eines NMI (nichtmaskierbarer Interrupt) zu haben. Der J-Befehl unterstuetzt dies, indem er das I-Register mit 1 laedt, so dass ein einzelner NMI-Impuls die Wirkung eines Breakpoint hat. Wegen zeitlicher Schwierigkeiten der Steuerung des Prozessors und des RAMs sollten Sie ein MonoFlop nehmen, damit der Impuls ausreichend kurz ist und steile Flanken hat. Bei einem ZX 80 gab es mit einem "angeschweisstem" 300 pF-Kondensator brauchbare Resultate : ohne Slow-Modus braucht man (wenn der G-Befehl nicht funktioniert) den NMI wirklich.

Wenn die Entprellung nicht in Ordnung ist (schlechte Impulsform oder mehrfache Impulse), kann es Hieroglyphen auf dem Schirm geben, aber der erste Druck auf die Tastatur behebt das. Bei einem ZX 80 ohne Slow-Modus kann ein schlechter Impuls den ASZMIC irritieren und ihn glauben machen, er sei in einem ZX 81 : worauf er auf weitere 270 NMI-Impulse wartet. Sie sollten also doch lieber auf gute Signale achten.

Interne Routines des ASZMIC

Es gibt interne Unterprogramme, die aber auch von ausserhalb aufgerufen werden koennen. Sie muessen ueblicherweise im Kontext des ASZMIC angewandt werden :

```
IY = :4000
I  = 14
IM = 1 (interrupt modus)

22 Bytes Platz fuer den Stack
```

Diese Routinen gehoeren dazu :

GETFLD	decodiert ein Feld
PUTDE	codiert eine Hex-Zahl
WRITA	codiert ein einzelnes Hex-Byte
WSTRNG	schreibt den Inhalt des Druck-Puffers auf den Schirm
NRM2	schreibt ein Zeichen auf den Schirm
OUTFRM	schreibt eine Rahmen auf den Schirm (Synchronisierungsrahmen)
KEYBRD	einfache Dekodierung der Tastatur
KEYINT	uebersetzt die Dekodierung in ein Zeichen
RDCASS	liest ein Zeichen von der Cassette
WRCASS	schreibt ein Zeichen auf Band
PRNT	schreibt eine Zeile auf dem Drucker

Die Gruesse mit auf den Weg

Im englischen Original schliesst das Handbuch mit einer Entschuldigung fuer die teilweise etwas flappsigen Beschreibungen. Wer lange an einem Programm oder einer Schaltung arbeitet und dann mehr und mehr fuehlt, dass ihm die Worte und die Zeit fehlen, all das zu beschreiben, was man damit (hier mit dem ASZMIC) tun kann, wird eben frustiert und entwickelt eine Art Galgenhumor. Fuer dieses neue deutsche Handbuch gilt das auch. Um all die Dinge zu beschreiben, die man den Anwendern gerne mitgeben moechte, muesste man mehrere Buecher fuellen und das ist einfach nicht zu bewaeltigen. Darum :

Wenn Sie Fragen haben : Rufen Sie an oder schreiben (mit Rueckporto).

Es gibt keine Garantie, dass wir Ihnen helfen koennen - aber wir wollen es versuchen.

Aribert Deckers, Stuttgart, Juli 1985

ANHANG

Uebersicht ueber die SHIFT-Funktionen

SHIFT 0 Rubout links

Der Cursor wird um 1 Zeichen nach links gerueckt. Das dort stehende Zeichen wird geloescht und die rechts stehenden Zeichen ruecken nach links nach. Wenn der Cursor am Beginn einer Zeile steht, wird mit "SHIFT 0" das <Newline>-Zeichen der davorstehenden Zeile geloescht : so werden die rechts vom Cursor stehende Zeile und jene davor aneinander gehaengt.

SHIFT 1 Zeile loeschen

Es wird die Zeile gelescht, in der sich der Cursor gerade befindet. Der Cursor wird nun an den Anfang der naechstfolgenden Zeile gestellt. Wenn der Cursor in der Fusszeile der letzten Textseite steht, geschieht nichts : der Befehl wird ignoriert.

SHIFT 2 File loeschen

Von der aktuellen Cursor-Position bis zur naechsten Filemarke (^) wird alles geloescht. Wenn keine Filemarke gefunden wird, wird nichts geloescht. Wird durch das Loeschen gleichzeitig auch die letzte Seite des Texts erreicht, wird der Cursor automatisch in die Fusszeile gestellt.

SHIFT 3 Seite runter

Das Fenster fuer die Bildschirmanzeige wird 27 Zeilen nach unten versetzt. Der Cursor wird in mittlere Zeile des Schirms gestellt. Wenn Sie die unterste Seite erreichen, wird der Cursor in die Fusszeile gestellt.

SHIFT 4 Seite rauf

Das Fenster fuer die Bildschirmanzeige wird 27 Zeilen nach oben versetzt. Der Cursor wird in mittlere Zeile des Schirms gestellt.
Zwingen Sie den ASZMIC nicht, das Fenster ueber die END-OF-DATA-Markierung hinauszugehen.

SHIFT 5 Cursor links

Bewegt den Cursor um 1 Zeichen weiter nach links, aber niemals auf das <Newline>-Zeichen.

SHIFT 6 Cursor runter

Bewegt den Cursor an den Beginn der hoeher liegenden Zeile. Damit der Cursor auf dem Bildschirm bleibt, kann auch automatisch um eine Zeile nach unten gescrolled werden. Stellt den Cursor weder auf oder hinter das END-OF-DATA-Zeichen

SHIFT 7 Cursor rauf

Bewegt den Cursor an den Beginn der tiefer liegenden Zeile. Damit der Cursor auf dem Bildschirm bleibt, kann auch automatisch um eine Zeile nach oben gescrolled werden.

SHIFT 8 Cursor rechts

Bewegt den Cursor um 1 Zeichen weiter nach rechts, aber niemals auf das <Newline>-Zeichen.

SHIFT 9 HOME in den DEBUG-Modus

Schaltet den ASZMIC in den DEBUG-Modus. Behaelt die gegenwaertige Cursor-Position im Gedaechtnis : man kehrt dorthin mit "SHIFT E" zurueck. Bewegt den Cursor in die letzte Zeile der letzten Seite. Das DEBUG-Modus-Flag wird gesetzt und der Cursor blinkt langsam.

SHIFT D Mergen

Dies ist eine Kann-Option des englischen Herstellers. Sie ist nicht unbedingt im ASZMIC eingebaut. Arbeitet wie SHIFT D, aber mit * als Markierung.

SHIFT E EDIT-Modus

Schaltet den ASZMIC um in den EDIT-Modus. Cursor kehrt an jene Position zurueck, die er innehatte, bevor mit SHIFT 9 in DEBUG gewechselt wurde. Damit der Cursor im Fenster sichtbar ist, wird notfalls der Bildinhalt geaendert. Der schnell blinkende Cursor gibt an, dass man sich im EDIT-Modus befindet.

Es koennen unvorhersehbare Effekte eintreten, wenn der Text vom DEBUG-Modus aus modifiziert wurde !

SHIFT F Mergen

Dies ist eine Kann-Option des englischen Herstellers. Sie ist nicht unbedingt im ASZMIC eingebaut. Arbeitet wie SHIFT D, aber mit < als Markierung.

SHIFT G Mergen

Im EDIT-Modus :
Vom Beginn des Textes an wird nach dem Merge-Zeichen ">" gesucht. Der Text zwischen diesem Merge-Zeichen und der naechsten Filemarke "^" wird an der aktuellen Cursor-Position eingefuegt. Wenn Merge-Zeichen und File-Marke fehlen, gibt es eine Katastrophe.

Im DEBUG-Modus :
Wenn im DEBUG-Modus eine Zeile mit einem <Newline>-Zeichen darin kopiert wird, fuehrt der Kommando-Interpreter diese Zeile sofort aus. Damit erfuellt dieser Befehl die Eigenschaft eines Macros fuer Kommandos. SHIFT G hat im DEBUG-Modus dieselbe Wirkung wie der M>-Befehl (siehe auch M-Befehl im Anhang 3).

SHIFT Q Rubout rechts

Das Zeichen, auf dem der Cursor steht, wird geloescht. Die rechts nachfolgenden Zeichen werden um eines nach links geschoben.

Es ist nicht moeglich, das letzte Zeichen in einer Zeile mit SHIFT Q zu eliminieren. Dieses liegt uebrigens zwischen 2 <Newline>-Zeichen !

SHIFT R Macro ausfuehren

Unabhaengig vom Modus, in dem sich der ASZMIC befindet, wird der Inhalt der obersten Zeile dem Kommando-Interpreter zur Ausfuehrung uebergeben. Die Cursor-Position und die angezeigte Bildseite bleiben unveraendert, es sei denn, das Ausfuehren des Kommandos erzwingt so etwas.

SHIFT W Cursor nach rechts an den Rand

Der Cursor wird an den rechten Rand der gegenwaertigen Zeile gestellt.

Einen Befehl fuer die Bewegung an den linken Rand gibt es nicht. Hierzu behilft man sich mit "SHIFT 7" (Cursor rauf) und anschliessendem "SHIFT 6" (Cursor runter).

SHIFT T Cursor an Textanfang

Der Cursor wird ganz nach Vorne an den Anfang des Textes gestellt. Dazu wird das Fenster automatisch auf die erste Textseite geschoben.

Bei Eingabe eines normales Zeichens wird dieses Zeichen an der aktuellen Cursor-Position eingefuegt. Die rechts nachfolgenden Zeichen werden um 1 nach rechts verschoben. Der Zeiger fuer das END-OF-DATA-Zeichen wird um 1 erhoeht.

Die Eingabe eines <Newline>-Zeichens wird ein Leerzeichen ("Blank") an die Zeile angehaengt - wenn das letzte Zeichen davor nicht bereits ein Leerzeichen ist. Das Fenster wird um eine Zeile nach unten geschoben, das heisst : der Text auf dem Bildschirm wird um eine Zeile nach oben gescrolled. Der Zeiger fuer das END-OF-DATA-Zeichen wird um 1 erhoeht.

Wenn der Speicherplatz fuer den Text voll ist, erreicht der Zeiger fuer das END-OF-DATA-Zeichen die Obergrenze TXTLIM (Textlimit). Dann wird kein weiteres Textzeichen zum Einfuegen mehr angenommen. Man kann weiterhin editieren (Loeschen und alle Cursorbewegungen ausfuehren).

ANHANG

Definition der DEBUG-Funktionen

Felder werden vom ASZMIC gleich behandelt. Man kann sie hintereinander schreiben. Dabei muss man sie mit Blanks ODER Kommas voneinander trennen, nicht also durch Blank UND Komma.

Wann immer ein <Newline>-Zeichen in den Textbereich geschrieben wird, so wird vom ASZMIC - wenn er im DEBUG-Modus ist - die so beendete Zeile dem Kommando-Interpreter zur sofortigen Ausfuehrung uebergeben.
Der Kommando-Interpreter nimmt die Zeile und sieht nach, ob das erste Zeichen (ganz links) darin einer der Buchstaben von A bis P ist. Ist das der Fall, so wird die entsprechende Routine zur Abarbeitung des Befehls aufgerufen. Ist die Bedingung aber nicht erfuellt, so wird das Zeichen ignoriert.

Man kann mehrere DEBUG-Kommandos in einer Zeile aneinanderhaengen. Sie werden dabei durch ein ;/ (Semikolon mit einem Schraegstrich dahinter) getrennt. Es gibt keine Blanks nach dem /

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

Der Inhalt der SHIFT-Macro-Zeile wird ebenfalls dem Kommando-Interpreter uebergeben, wenn ein "SHIFT R" eingetippt wird. Diese Zeile wird ebenfalls ausgefuehrt, wannimmer eine Break-Bedingung erfuellt ist :

- Breakpoint
- RST 0
- Single-Step
- externer NMI

A Assemblieren

```
A ^ Filename Option <Newline>
- - -------- ------ ---------
```

Das angegebene File wird gesucht und assembliert - vom Filenamen am Anfang bis zur Filemarke am Ende. Wenn keine Option angegeben wird, ist damit vorgegeben :

- ein vollstaendiges Assembly mit 2 Durchlaeufen
- Erzeugung des Object-Codes
- keine alte Symbol-Tabelle wird gerettet, es wird eine neue erstellt
- es wird kein Listing erzeugt

Das Byte der Optionen wird nach folgendem Muster zerlegt und bewirkt :

```
Optionen : 128  Bit 7    2. Durchlauf erzwingen
           64   Bit 6    Erzeugung des Object-Codes unterdruecken
           32   Bit 5    64 Zeichen pro Druckzeile / keine Laengenbegrenzung
           16   Bit 4    gibt es nicht
            8   Bit 3    gibt es nicht
            4   Bit 2    alte Symbol-Tabelle erhalten und ergaenzen
            2   Bit 1    Listing zum Drucker senden
            1   Bit 0    Listing erzeugen
```

Folglich wird die Erzeugung eines Listings auf dem Drucker bei Unterdrueckung des Object-Codes :43 als Option haben (dezimal 67) oder 64+2+1 (so koennen Sie es schreiben).

Wenn Sie nicht die ORG-Anweisung benutzen, wird der ASZMIC moeglicherweise beim Inhalt von TXTLIM beginnen, aber das haengt von der Version des ASZMIC ab und ist daher zu vermeiden.

Die Zeilen eines Assembler-Programms koennen in der 1. Spalte beginnen mit :

1. Semikolon

   Die Zeile wird als Kommentar betrachtet und erzeugt keinen Object-Code.

2. ein Zeichen, das kein Blank ist

   Das Zeichen wird als Beginn eines Symbols betrachtet (Label), das hiermit definiert wird.

3. Blank

   Dann muss ein Befehl in Assembler folgen.

Befehle in Assembler koennen auch eines oder zwei Argumente haben.

Argumente werden durch ein Komma von einander getrennt.

Eine Zeile darf nach dem eigentlichen Assembler-Befehl auch einen Kommentar enthalten. Es wird dann durch ein davorgestelltes Semikolon gekennzeichnet.

Ein File wird mit einer alleinstehenden Filemarke in einer Zeile beendet.

Beispiel :

```
^BEISPIEL
 ORG :7000
NRM2=:492 ;IM ANHANG NACHPRUEFEN
START LD HL,$+120 ; DIE ADRESSE DER TABELLE

 LD B,TABENDE-TAB
LOOP LD A,(HL)
 PUSH HL
 PUSH BC
 CALL NRM2
 POP BC
 POP HL
 INC HL
 DJNZ LOOP
; UND NUN ZURUECK ZUM ASZMIC
 RST 0
;
 ORG :7000+120
TAB DEFM "TEST"
TABENDE=$
^
A ^BEISPIEL 1
```

Achtung : Wenn die Variable OFFSET ist ungleich 0 ist, wird sie benutzt, um den erzeugten Object-Code im Speicher zu verschieben. Der Object-Code wird so erzeugt, dass er ab ORG lauffaehig ist. OFFSET "laeuft ueber" :

```
Beispiel :    ORG    :8000
              OFFSET :C000
```

wird den erzeugten Object-Code ab :4000 in den Speicher schreiben.

B Breakpoint

B<Newline>
B adresse<Newline>

Wenn eine Adresse angegeben wird, so wird der gegenwaertige Breakpoint beseitigt (das gerettete Original-Byte wird wieder eingesetzt) und die angegebene Adresse wird der neue Breakpoint : Das Byte an der Adresse, wo der Breakpoint wirksam werden soll, wird durch den Code fuer einen RST 0 -Befehl (also :C7) ersetzt.

Erreicht der Rechner beim Progamm-Ablauf das RST 0, so springt der Prozessor an die Adresse 0 und der ASZMIC "fischt ihn dann wieder auf". Wenn ein BREAKPOINT bei normalem Programmablauf erreicht wird, ersetzt der ASZMIC das ausgetauschte Byte durch den urspruenglichen Inhalt.

Wenn ein B-Kommando ohne Adresse eingegeben wird, wird der RST 0 -Code an der gegenwaertigen Breakpoint-Adresse eingefuegt und das Original-Byte gerettet.

C Kopieren

C Adresse des Originals Adresse des Ziels Laenge<Newline>

Von der Adresse des zu kopierenden Originals wird fuer die vorgegebene Laenge ein Bereich kopiert. Die Kopie wird ab der angewiesenen Zieladresse ins RAM geschrieben. Der ASZMIC kontrolliert Ueberlappungen und kopiert so, dass keine Fehler entstehen.

D DUMP

D Adresse Bereich<Newline>		DUMP ueber einen Bereich.
D Adresse <Newline>		DUMP-Modify.
.<Newline>	(Punkt und Newline)	Beenden des DUMPs

Der Rechner nimmt die Zahlen automatisch als hexadezimal an.

Zur Eingabe von dezimalen Zahlen stellt man das "Ø+" vor die Zahl.

Ø + dezimalzahl

Beim formatierten DUMP einer vorgegebenen Zahl von Bytes werden 8 hexadezimale Bytes angezeigt. Waehrend der Aufbereitung langer DUMPS kann der Bildschirm verschwimmen.

<BREAK> unterbricht einen DUMP.

DUMP-Modify : Die Adresse und ihr Inhalt in Hex angezeigt. Der ASZMIC wartet dann auf eine Eingabe. <Newline> belaesst den alten Inhalt. Die Eingabe einer Zahl - abgeschlossen mit <Newline> - bewirkt, dass diese Zahl in die betreffende Adresse geschrieben wird. Der Dump kann mit einem Punkt beendet werden : .<Newline>

Achtung : Wenn Bit 1 von ASSFLG gesetzt ist, um den DUMP zum Drucker zu senden, muss man in jeder Zeile mindestens 8 vorlaufende Blanks eingeben.

E EDIT

E<Newline>
E Symbol<Newline>

Wird im DEBUG-Modus das "E" ohne Symbol eingegeben, so kehren Sie in den EDIT-Modus zurueck (schnell blinkender Cursor).

Wird ein Symbol angegeben, dann durchsucht der ASZMIC den Text (von hinten nach vorn ?) nach diesem Symbol und stellt den Cursor an den Beginn des Symbols (wenn es gefunden wurde). Dazu kann das Fenster verschoben werden. Wenn das Symbol nicht gefunden wird, geschieht nichts.

Jedes Zeichen, das weder

^ . 0 bis 9 noch A bis Z

ist, beendet die Vergleichsoperation fuer das Symbol.

^ ist nur als erstes Zeichen erlaubt.

F Fuellen

F Anfangs-Adresse End-Adresse Inhalt <Newline>

Von der angegebenen Anfangsadresse bis zur Endadresse wird der Speicher mit dem gewuenschten Inhalt gefuellt.

G GO

G<Newline>
G Adresse<Newline>
G Adresse Schrittzaehler<Newline>

Das ist die Single-Step-Funktion des ASZMIC. Wenn keine Argumente angegeben werden, wird der Kontext vom Abbild der Register im Speicher REGIM zurueckgeholt und wieder hergestellt. Dann wird ein Befehl ausgefuehrt an der vom geretteten PC angegebenen Adresse. Am Ende des Befehls wird ein Single-Step-Break ausgeloest und der neue Kontext wird nach REGIM gerettet. (Wenn die Variable INTJMP modifiziert wurde und nun eine andere Adresse enthaelt als INTRET, wird an dieser Stelle ein Sprung zum Inhalt von INTJMP ausgefuehrt.) Dann wird die SHIFT-Macro-Zeile ausgefuehrt, bevor der ASZMIC wieder die Kontrolle uebernimmt.

Wenn eine Adresse angegeben wird, ueberschreibt sie den geretteten PC (PC1) und wird die Adresse jenes Befehls, der ausgefuehrt werden soll.

Wenn beim G-Kommando keine Adresse angegeben wurde, benutzt der ASZMIC jene Adresse, die im Speicher als Abbild der Register (REGIM) abgelegt worden ist.

Wenn ein Schrittzaehler angegeben wurde, wird der Ablauf (so wie oben) weiter fortgesetzt. Aber nach dem Retten des Kontexts und vor der Ausfuehrung des Macros wird der gerettete Schritt-Zaehler heruntergezaehlt - und wenn der Schrittzaehler noch ungleich 0 ist, wird der nachste Befehl ausgefuehrt.

Die Ausfuehrungsgeschwindigkeit ist typisch 1/100 der normalen, so dass man bei einem grossen Schrittzaehler eine Weile braucht, um vorwaerts zu kommen. Maximum sind :7FFF, also 32767 Schritte. Der G-Befehl arbeitet nicht auf einem ZX 80 ohne Slow-Modus.

Anmerkung : Wenn Sie Single-Steppen und dabei auf einen BREAKPOINT stossen, dann werden Sie die Logik (das Programm) zum Ausfuehren eines BREAKPOINT Single-Steppen. Der ASZMIC merkt naemlich nicht, dass durch das Einsetzen eines BREAKPOINT im RAM ein RST 0 eingefuegt wurde : er geht auch den Befehl RST 0 und die nachfolgenden im Single-Step durch.

Achtung : Der Single-Step kann benutzt werden, um in einem ROM oder EPROM einen Breakpoint zu simulieren. Setzen sie den Breakpoint auf

Adresse-1

und benutzen den Befehl

G Adresse 32767

Wird die Stop-Adresse erreicht, so glaubt die Logik zum Abarbeiten eines Breaks, dass ein Breakpoint erreicht wurde und beendet das Single-Steppen.

H HINEIN !

H Startadresse <Newline>

Das ist ein Sprung in ein Programm, geradewegs an jene angegebene Adresse. Das HL-Registerpaar zeigt auf die Kommando-Zeile nach der Adresse; das angesprungene Programm kann im ASZMIC-Kontext arbeiten und mit RET zurueckkehren. Sie koennen hier eigene Routinen in den ASZMIC einfuegen.

I Immediate

I Assemblerzeile <Newline>

Der ASZMIC akzeptiert Befehle in Assembler und fuehrt sie (als eine Ergaenzung Ihres Programms) in der Umgebung (dem Kontext) Ihres Programms direkt aus. Es muss also nicht erst neu assembliert werden.

Die Zeile wird uebersetzt und ausgefuehrt, wobei ein Break angehaengt wird. Die Ausfuehrung erfolgt mit einem internen J-Befehl und benutzt darum den geretten Kontext von REGIM. Nach der Ausfuehrung wird der Kontext wieder nach REGIM gerettet - wie beim normalen Break.

In einem Immediate-Befehl koennen sie jeden Assembler-Befehl geben. Sie koennen sogar Label mit der EQU-Zuweisung festlegen. Relative Sprueage und andere Anweisungen, wie ORG, DEFB, DEFW, DEFM, sind bedeutungslos und koennen das System zum Absturz bringen, wenn man mit dem I-BEfehl ihre Ausfuehrung erzwingen will.

Label sollten nur mit Label-Wert in Form von direkten Anweisungen zugewiesen werden. Anweisungen wie ORG,DEFM, DEFB, DEFW und die Befhel JR und DJNZ sollten vermieden werden.

J JUMP

J<Newline>
J Adresse<Newline>

Das ist wie beim G-Kommando. Aber es gibt bei der Ausfuehrung kein Single-Step-Break, es sei denn, das B-Kommando wurde benutzt, um einen Breakpoint im logischen Ablauf einzufuegen. Die Wirkung eines Breakpoints oder eines extern erzeugten NMI ist aehnlich die eines Single-Step-Interrupts.

K Saven Cassette

K^S "Kennung" ^Filename<Newline>
K^S "Kennung" Startadresse Endadresse<Newline>
K^S "Kennung" Startadresse Endadresse L <Newline>

Der ASZMIC benutzt das gleiche Aufzeichnungsprotokoll wie der ZX 80/81. Man kann Texte (Files) oder Speicherbereiche auf Band saven.

Es ist aeusserst wichtig, dass zwischen dem "S" und dem Gaensefuesschen nur ein Blank steht - sonst kann der ASZMIC die Zeile hinterher nicht als Titel erkennen und darum Ihr File nicht lesen koennen.

Nachdem Sie <Newline> gedrueckt haben, muessen Sie in den folgenden 5 Sekunden den Cassetten-Recorder starten. Die Anzeige verschwindet fuer ca 1/2 Sekunde und die Titel-Zeile wird auf's Band geschrieben. Nun folgt eine Pause von 5 Sekunden, in der das Display stabil ist, danach verschwindet es wieder und das File wird gesaved.

Wenn alles auf Band ist, erscheint die Anzeige wieder. Mit der <BREAK>-Taste kann man das Saven jederzeit abbrechen.

Wenn ein Speicherbereich gefolgt wird von dem

Blank L

so wird fuer das Laden LABEND auf den Wert der Startadressse gesetzt.

L Laden Cassette

L "Kennung" <Newline>
L "Kennung" L <Newline>

Als Antwort auf diesen Befehl wird der Cassetten-Anschluss staendig ueberwacht - und wenn ein gueltiger Titel vorgefunden wird, wird er auf den Bildschirm geschrieben und das Display fuer 5 Sekunden erhalten. Das ergibt einen Katalog der auf dem Band vorhandenen Files.
Wenn "Kennung" und Titel auf dem Band uebereinstimmen, wird geprueft, ob es sich um einen Text oder einen Speicherbereich handelt, der geladen werden soll. Im 2. Fall wird an die beim Save-Befehl angegebene Adresse geladen (dieser Befehl wird mit aufgezeichnet !). Wenn die Variable OFFSET einen von 0 abweichenden Wert hat, wird um eben diesen Wert von OFFSET verschoben geladen.

Gibt es nach der Angabe des Speicherbereichs ein "L", so ist das das Zeichen fuer eine Symbol-Tabelle und LABEND wird auf den Wert der Startadresse (Beginn des Bereichs) gesetzt. Das setzt aber voraus, dass Saven und Laden mit einem gleichgrossen Speicher erfolgen.

Sowohl der K-Befehl als auch der L-Befehl koennen mit <Break> beendet werden, was einer BREAK-Bedingung simuliert.

M MACRO

M Buchstabe <Newline>

Das Zeichen nach dem "M" gibt das Startzeichen fuer den Kommando-Macro an - im Gegensatz zum "SHIFT G", bei dem der Macro immer mit ">" beginnt.

Das erlaubt die Benutzung einer ganzen Reihe von Macros. So kann beispielsweise ein DUMP, der mit dem "=" Gleichheitszeichen angefordert wird, vom Macro mit dem "=" aufgerufen werden und jener DUMP mit dem "+" davor vom Macro mit dem "+" nach dem "M".

Definieren Sie keinen Macro durch M allein und geben Sie acht, dass es das erste Auftreten des Buchstabens ist, welches den Beginn des Macro definiert (ausgenommen die SHIFT-Macro-Zeile).

N NEW N <Newline>

Laed das BC-Registerpaar mit dem Inhalt von TXTLIM und setzt HL auf :03CA (BASIC-Befehl "NEW"). Springt an den Inhalt von TXTLIM. WIrd benutzt, um zwischen ASZMIC und BASIC umzuschalten (External Card).

O Old Register O <Newline>

Die im REGIM (Register Image = Register-Abbild) genannten Bereich im Speicher gelagert werden angezeigt :

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

PC : Programm-Zaehler
SP : Stack-Pointer
Die gestrichenen Register (zum Beispiel BC') sind die Hintergrundregister des Z80.

Es ist sehr nuetzlich, ein "O" in die SHIFT-Macro-Zeile zu schreiben : die Macro-Zeile wird bei jeder BREAK-Bedingung ausgefuehrt. Da man ueblicherweise einen BREAK macht, um die Register anzusehen, wird das hier also automatisch ausgefuehrt.

P PRINT P ^ Filename <Newline>

Das angegebene File wird zum Drucker gesendet, bis eine Filemarke als das erste Element einer Zeile vorgefunden wird oder mit <BREAK> manual abgebrochen wird.

Der Druckvorgang kann jederzeit durch das Druecken der <Break>-taste beendet werden.

Achtung : Die Variable PRTJMP kann von ihrem vorgegebenen Wert PRTRET mit einer Adresse belegt werden, die Sie als Startadresse einer eigenen Drucker-Routine angeben. Damit kann man sich den ASZMIC an andere Drucker anpassen.

ANHANG

Die verschiedenen Versionen der ASZMIC-ROMs

Das ASZMIC-ROM ist die professionelle Arbeit eines erfahrenen Entwicklungsingenieurs. Es war das Ziel, ein extrem preiswertes Entwicklungssystem zu konstruieren auf der Basis des billigsten Computers, des ZX 81.

Der ASZMIC wurde in mehreren Versionen geschrieben, um den Anforderungen in den jeweiligen Laendern mit ihren besonderen Fernsehsystemen und ihren sprachspezifischen Zeichensaetzen etc gerecht zu werden. Im Laufe der Zeit wurden verschiedene Aenderungen und Verbesserungen vorgenommen :

Version	Kommentar
E04	Europa / Diese Version ist fuer den ZX 81 oder einen aufgeruesteten ZX 80 gedacht. Sie kann nur innen im Computer eingesetzt werden.
E05	USA / Diese Version hat die gleichen Eigenschaften wie E04, ist aber an das amerikanische Fernsehsystem angepasst.
E06	Experimental-Version : Wurde nur zu Entwicklungsarbeiten verwendet und kam nicht in den Handel.
E07	Version fuer Europa : 2 Fehler der E04-Version wurden beseitigt : - die Routine MSKINT wurde modifiziert, damit das I-Register mit 1E hex geladen wird anstelle von 0E hex. Dadurch kann das ROM intern und extern (am Busstecker des Computers) gesteckt werden. - Ferner wurde der Zeichengenerator und die Zeichentabelle KEY TAB umgebaut. Das erlaubt einen Betrieb nur noch am ZX 81 und nicht wie beim E04 an einem aufgeruesteten ZX 80. Version fuer USA / Canada : Diese Variante wurde den dortigen Fernsehnormen angepasst (basiert auf 60 Hz Netzfrequenz).

Obwohl laut Hersteller alle Versionen durch die an Adresse 0016 hex abgelegte Variable MKDEF gekennzeichnet sein sollen, gibt es Varianten mit verschiedenem Inhalt, die dennoch die gleiche Kennung an jener Adresse tragen !

Fuer die verschiedenen Laender gibt es 3 Sonderzeichen. Das entspricht 24 Bytes im Zeichengenerator (ab Adresse 0E40 aufwaerts). Die englischen Versionen haben dort die gleichen Zeichen wie das BASIC-ROM des ZX 81. Die deutsche Version des E07 wird mit Umlauten ausgeliefert.

Die Symbol-Tabellen des ASZMIC-ROMs

Wenn man Programme schreibt, so bestehen diese aus allen moeglichen Programmstuecken und Unterprogrammen. Benutzt man den ASZMIC nur, um fuer einen anderen Computer zu assemblieren, muss man alle Unterprogramme und Funktionen selbst erarbeiten.

Will man seine Programme jedoch nur im ZX 81 mit dem ASZMIC-ROM darin benutzen, so kann man durchaus im ASZMIC-ROM oder im BASIC-ROM befindliche Unterprogramme und Funktionen aufrufen.

Es gibt die Moeglichkeit, das ASZMIC-ROM auf einer externen Karte am Bus anzustecken und ASZMIC-ROM und BASIC-ROM gleichzeitig zu benutzen. Das ASZMIC-ROM liegt dann im Adress-Bereich von 0000 bis 0FFF hex. Die obere Haelfte des BASIC-ROMs von 1000 bis 1FFF ist frei zugaenglich - also koennen auch dort liegende Programmteile und der Zeichengenerator verwendet werden !

Die nun folgenden Symboltabellen fuer die internen Adressen des ASZMIC-ROMs muessen Sie sorgfaeltig studieren und sich die interessanten Teile herausuchen.

Hex Addr	Name	Size	Function
4000	MFLAG	1	Flags: bit 7 is edit/debug flag; rest used as NMI counter
4001	ARG1	2	Used by GET2 to decode debug command arguments
4003	ARG2	2	
4005	ASSFLG	1	Assembler options byte described in manual
4006	BOPSAV	1	Breakpoint saved byte
4007	BADDR	1	Breakpoint address
4009	SSCNT	2	Single step counter
400B	OFFSET	2	Offset value used by load & assemble
400D	TEMP	2	Assembler storage
400F	STMEND	2	Latest keyboard decode (see KEYBRD routine)
4011	USAMOD	4	Unused
4015	PRTJMP	2	Address of printer routine
4017	KEYJMP	2	Addres of STMEND analysis routine
4019	INTJMP	2	Address of routine to handle breaks after context save
401B	DADDR	2	Address of debug command interpreter
401D	ECPOSN	2	Cursor address when shift 9 last pressed
401F	CURSOR	2	Cursor address
4021	EOD	2	Address of current end of text
4023	DFILE	2	Address of ;76 before first byte of current display
4025	TXTLIM	2	Address of partition between text & program areas
4027	LABEND	2	Bottom of symbol table address
4029	LABSTK	2	Top of symbol table pointer;usually top of memory
402B	PRBUFF	65	Printer buffer
406C	TEMP2	2	Temporary storage; mostly for assembler
406E	LSTEXP	1	No of frames to delay for keyboard debounce
406F	REPEAT	1	No of frames to delay for key repeat
4070	FRAMES	2	Frame count; used for kevin simulation after a no of frames
4072	ELEM1	2	Result of GETFLD analysis
4074 - 4079		6	GETFLD working variables
407A	STKLOW	32	ASZMIC stack area
409A	REGIM	24	Context save area pc1 hl1 hl2 bc2 de2 af2 af1 bc1 de1 iy1 ix1 spl
40B4	DSPBGN		Start of text area

SYSTEM ADDRESSES

To ensure that the ASZMIC ROM corresponds to the version described here, check the two bytes at MKDEF against the declared variable "VRSION" in the list. They should correspond; i.e. VRSION E04 has MKDEF 04 0E.

ACOM2	0891	CREN1	045B	EDEXIT	05F7	G4IN2	0A69
ACOMIN	08C4	CRGEN1	0457	EDIN	0ACA	G4IN3	0A76
ACOMM	052D	CRHNDL	0C24	EDINX	0AC9	G4IN4	0A79
AF1	40A6	CSTR1	02B3	EDLP1	0775	G4REG	0A57
AF2	40A4	CTAB	0D04	EDLP2	0776	G6INX	0ACE
ARG1	4001	CTAB2	0D4B	ELEM1	4072	G6PA	0AC7
ARG2	4003	CUDRET	048D	EOD	4021	G8NIXY	0AF1
ASEXIT	086D	CUDRTX	0487	EODCHR	0005	G92	0B00
ASSFLG	4005	CURSOR	401F	EODOK	04B8	GAHERE	0B18
ASSMBL	0870	DADDR	401B	EX102	09E1	GANDBR	0B23
AUDUMP	0223	DBTLIM	05B4	EX103	0A11	GARNDX	0B12
AXX	0547	DCLP1	0881	EX104	09F6	GAUNC	0B1B
BADDR	4007	DCOMM	057A	EX105	0A47	GB01	0B3A
BC1	40A8	DCOMX	057F	EX107	0A81	GBCOND	0B37
BC2	40A0	DCONT	0B95	EX1213	09BB	GCENTX	022A
BCOMM	054C	DE1	40AA	EX1NN	09AC	GCHERE	0B48
BFLAG	4075	DE2	40A2	EX1NTA	09BE	GCINX2	0978
BFLGST	0C2E	DECLOK	08A2	EX2DBR	0A13	GCNTSB	0250
BIGMEM	01A7	DEFMNT	0934	EX2IND	0A2A	GCOM2	0973
BLONE	055E	DELAY5	00DC	EX2NAF	09F9	GCOMM	0615
BOPSAV	4006	DFILE	4023	EX2NAG	0A07	GCUNC	0B5D
BRKCHK	030D	DFLIP	032E	EX2NB	09E3	GDLOOP	0B63
BRKOD	00C7	DIGCON	0327	EX3IR	0A3B	GELOOP	0B69
CALLAS	0823	DLLP1	00D6	EX3IRX	0A42	GET2	0059
CCC	0BC3	DLLP5	00D4	EXCF1	09A2	GETCHR	0226
CCOMM	0561	DLN1	03E0	EXCF2	09D0	GETFLD	0010
CDLP1	04B1	DLOOP1	058E	EXCF3	0A31	GFCHAR	0C61
CHRTAB	0D92	DLOOP2	0585	EXTNAC	09CC	GFDCLN	0BE1
CINCRT	0470	DLY05	00D2	EXTNHL	0A27	GFDHEX	0C1A
CKINV	0C9F	DMD2	05ED	FCOMM	0602	GFDLBL	0C89
CLDIR	0577	DMDLP1	05DE	FILCHR	000C	GFDLBX	0C86
CLNLP	0945	DMODIN	05CB	FLDFND	4076	GFDNUM	0C40
CLNUP	0941	DMOUT	05B0	FND2	0C83	GFDREG	0CBD
CLPRLD	0BE8	DMPMOD	05B3	FNDLBL	0C72	GFDTNM	0C38
CMDSUB	0295	DMPREG	0733	FNDLCR	0028	GHXVAL	0BDE
CMPSTR	02A3	DOLLAR	0C1E	FNDRCR	0030	GROUP0	095F
CMPSTX	02B0	DONCHR	0C6E	FRAMES	4070	GROUP1	095F
CMRTX1	0512	DONE	0CB9	FRMSND	00E5	GROUP2	097C
CMSM1	02C3	DOTTIM	0804	FRMSNX	00E8	GROUP3	0A49
COMINC	0C23	DPG1	03FE	G1COMN	096E	GROUP4	0A4D
COMINT	0507	DSPBGN	40B4	G2IN2	0B88	GROUP5	0AAC
COMMND	052D	DSPSET	019E	G2SBCN	0B76	GROUP6	0AB9
COMRTX	0511	DSPTCH	0947	G2SUB	000B	GROUP7	0AD0
COMXTB	051D	ECOMM	05F0	G34TAB	0AA2	GROUP8	0AE3
CONLIN	0023	ECPOSN	401D	G3G4	0A83	GROUP9	0AF3
CRCHAR	0076	EDCODE	0080	G42ER	0A4D	GROUPA	0B03

GROUPB	0B30	LDLP2	06D2	PDOT	0809	SHFT9	042D
GROUPC	0B3C	LDLP3	0703	PDOTX	07F9	SHFTD	0339
GROUPD	0B61	LDLP8	08A4	PRBUFF	4[illegible]8	SHFTE	0398
GROUPE	0B68	LDN2	06BC	PRCLR	0[illegible]1	SHFTF	033D
GROUPF	0B71	LDN6	06B8	PREADY	07D9	SHFTG	0341
HASH	08C7	LINEND	00C1	PRIGET	0924	SHFTQ	036C
HASHDN	08DC	LIX	022D	PRNTER	07CD	SHFTR	03B1
HCOMM	0652	LIX2	0236	PROFF	081E	SHFTT	03B7
HFLIP	0070	LIXIMM	023B	PRTJMP	4015	SHFTW	0394
HL1	409C	LIXSUB	00AD	PRTRET	07D2	SHIFTS	0339
HL2	409E	LODFIL	0716	PSLP	0756	SHOME	03AE
HLRIN	0CD0	LOOPGF	0BF0	PTCON	00CB	SINGLE	01D7
HLRTST	0CD7	LSTEXP	406E	PUTA	0B9D	SINGLX	01F2
HMCHK	03F9	LX	4000	PUTB	0B9C	SNLP1	0698
HOME2	0434	LY	4015	PUTDE	0319	SNLP2	06A8
HOME3	0448	LZ	406E	PUTDEF	0316	SP1	40B0
HOMLP1	044A	MCOMM	0724	PUTENT	0BAE	SRECUR	0679
HRTIN1	0950	MFLAG	4000	PUTNN	0B8E	SSCNT	4009
HRTLP	0953	MIDWAY	0C70	PUTNNX	0B92	START	0000
HSHDN2	08FE	MKDEF	0016	PUTOUT	0BBF	STENT	039F
HSHERR	0913	MRGCHR	0017	QCOMM	0762	STKLOW	407A
HSHLP1	08CB	MRGIN	0352	QUOTE	000B	STMEND	400F
IAENT	061A	MRGLP1	035B	RASCON	07E2	STRBOK	0773
ICOMM	0655	MSKINT	004A	RASPRS	0043	STRCDB	0763
IGNBLK	0020	MSMTCH	02D0	RASTER	07D5	STRSCH	0762
IMTAB	0AB6	NCOMM	0728	RCOMM	0762	TABLE	0082
INHERE	0C9F	NEGEND	0CB0	RCOMP	0D99	TEMP	400D
INICON	017E	NEGFLG	4077	RCOMPX	0DB0	TEMP2	406C
INIT	017C	NEGNOT	0CAF	RDC1	07A2	TIMING	0066
INIT2	0189	NGENT	0CB4	RDC2	07A4	TXTLIM	4025
INIT3	0183	NGHNDL	0C33	RDC3	07B2	UPDG1	0418
INTJMP	4019	NLBLF	0C45	RDC4	07B4	UPDG2	0421
INTRET	01FC	NLONP	0023	RDC5	07C5	USAMOD	4011
IX1	40AC	NOBRK1	0623	RDCASS	07A2	V	0040
IXIY	4074	NOBXRK	061E	RDCX	07B0	VRSION	0E04
IY1	40AE	NOGCNT	061F	REG	4078	WCNORM	08F8
JCOMM	0667	NOLIST	0859	REGCOD	4079	WCRPT	08E1
JGENT	0617	NORMSV	0690	REGIM	409A	WCTAB	0DC7
JMP2	063D	NOTBRK	0215	REPEAT	406F	WCTABX	0DE4
JMPTYP	064D	NOTHL	0CEA	RESTOR	0628	WRC1	0783
JPTAB	0BCE	NOTIXY	096C	RETNR	0C24	WRC2	0788
KBD1	00F8	NRM2	0492	RLB	0B96	WRCASS	0780
KBD2	00FA	NRM4	04A6	RLOAD	0CE5	WREG	073C
KCOMM	066B	NRMCHR	0491	RUBDNO	0380	WREGL	0742
KEYADD	027F	NULIN2	04E6	RUBOK	0378	WRITA	031E
KEYBRD	0145	NULINE	04D9	SAVMEM	018D	WSTR1	02F2
KEYINT	0277	NXTPLN	0818	SAVSTR	001B	WSTRG2	02E2
KEYINX	0275	NXTSTP	087A	SFILE	067E	WSTRLP	02DC
KEYJMP	4017	OCOMM	0733	SHFT0	0426	WSTRNG	02D6
KEYRET	0254	OFFSET	400B	SHFT1	03BE	X1	0010
KYRDLP	014F	OFRM1	016D	SHFT2	03E2		
LABEND	4027	OFRM2	0171	SHFT3	0408		
LABSTK	4029	OUTFRM	0169	SHFT4	0412		
LBINX	08AC	PARSE	0826	SHFT5	0423		
LCOMM	06B7	PC1	409A	SHFT6	0473		
LDF2	0719	PCHAR	07F8	SHFT7	045F		
LDLP1	06C3	PCOMM	0752	SHFT8	0456		

E07

ACOM2	0890	ACOMIN	0BC3	ACOMM	052B	AF1	40A6	AF2	40A4	ARG1	4001
ARG2	4003	ASEXIT	0B6C	ASSFLG	4005	ASSMBL	084F	AUDUMP	0221	AXX	0545
BADDR	4007	BC1	40AB	BC2	40A0	BCOMM	054A	BFLAG	4075	BFLGST	0C2D
BIGMEM	01A5	BLONE	055C	BOPSAV	4006	BRKCHK	030B	BRKOD	00C7	CALLAS	0822
CCC	0BC2	CCOMM	055F	CDLP1	047F	CHRTAB	0D91	CINCRT	046E	CKINV	0C9E
CLDIR	0577	CLNLP	0944	CLNUP	0940	CLPRLD	0BE7	CMDSUB	0293	CMPSTR	02A1
CMPSTX	02AE	CMRTX1	0510	CMSM1	02C1	COMINC	0C22	COMINT	0505	COMMND	052B
COMRTX	050F	COMXTB	051B	CONLIN	0023	CRCHAR	0076	CREN1	0459	CRGEN1	0455
CRHNDL	0C23	CSTR1	02B1	CTAB	0D03	CTAB2	0D4A	CUDRET	04BB	CUDRTX	0485
CURSOR	401F	DADDR	401B	DBTLIM	0584	DCLP1	0B80	DCOMM	057A	DCOMX	057F
DCONT	0B94	DE1	40AA	DE2	40A2	DECLOK	0BA1	DEFMNT	0933	DELAY5	00DA
DFILE	4023	DFLIP	032C	DIGCON	0325	DLLP1	00D4	DLLP5	00D2	DLN1	03DE
DLOOP1	058E	DLOOP2	0585	DLY05	00D0	DMD2	05ED	DMDLP1	05DE	DMODIN	05CB
DMOUT	0580	DMFMOD	05B3	DMPREG	0732	DOLLAR	0C1D	DONCHR	0C6D	DONE	0CB8
DOTTIM	0B03	DPG1	03FC	DSPBGN	40B4	DSPSET	019C	DSPTCH	0946	ECOMM	05F0
ECFOSN	401D	EDCODE	00B0	EDEXIT	05F7	EDIN	0AC9	EDINX	0AC8	EDLP1	0774
EDLP2	0775	ELEM1	4072	EOD	4021	EODCHR	0005	EODOK	04B6	EX102	09E0
EX103	0A10	EX104	09F5	EX105	0A46	EX107	0A80	EX1213	09BA	EX1NN	09AB
EX1NTA	09BD	EX2DBR	0A12	EX2IND	0A29	EX2NAF	09F8	EX2NAG	0A06	EX2NB	09E2
EX3IR	0A3A	EX3IRX	0A41	EXCF1	0941	EXCF2	09CF	EXCF3	0A30	EXTNAC	09CB
EXTNHL	0A26	FCOMM	0602	FILCHR	000C	FLDFND	4076	FND2	0CB2	FNDLBL	0C71
FNDLCR	0028	FNDRCR	0030	FRAMES	4070	FRMSND	00E3	FRMSNX	00E6	GICOMN	096D
G2IN2	0B87	G2SBCN	0B75	G2SUB	0008	G34TAB	0AA1	G3G4	0AB2	G42ER	0A4C
G4IN2	0A68	G4IN3	0A75	G4IN4	0A78	G4REG	0A56	G6INX	0ACD	G6PA	0AC4
G8NIXY	0AF0	G92	0AFF	GAHERE	0B17	GANOBR	0B22	GARNDX	0B11	GAUNC	0B1A
GB01	0B39	GBCOND	0B36	GCENTX	022B	GCHERE	0B47	GCINX2	0977	GCNTBB	024E
GCOM2	0972	GCOMM	0615	GCUNC	0B5C	GDLOOP	0B62	GELOOP	0B68	GET2	0059
GETCHR	0224	GETFLD	0010	GFCHAR	0C60	GFDCLN	0BE0	GFDHEX	0C19	GFDLBL	0CBB
GFDLBX	0C85	GFDNUM	0C3F	GFDREG	0CBC	GFDTNM	0C37	GHXVAL	0BDD	GROUP0	095E
GROUP1	095E	GROUP2	097B	GROUP3	0A4B	GROUP4	0A4C	GROUP5	0AAB	GROUP6	0ABB
GROUP7	0ACF	GROUP8	0AE2	GROUP9	0AF2	GROUPA	0B02	GROUPB	0B2F	GROUPC	0B3B
GROUPD	0B60	GROUPE	0B67	GROUPF	0B70	HASH	08C6	HASHDN	08DB	HCOMM	0653
HFLIP	0070	HL1	409C	HL2	409E	HLRIN	0CCF	HLRTST	0CD6	HMCHK	03F7
HOME2	0432	HOME3	0446	HOMLP1	0448	HRTIN1	094F	HRTLP	0952	HSHDN2	08FD
HSHERR	0912	HSHLP1	08CA	IAENT	061A	ICOMM	0656	IGNBLK	0020	IMTAB	0AB5
INHERE	0C9E	INICON	017C	INIT	017A	INIT2	0187	INIT3	0181	INTJMP	4019
INTRET	01FA	IX1	40AC	IXIY	4074	IY1	40AE	JCOMM	0668	JGENT	0617
JMP2	063E	JMPTYP	064E	JPTAB	0BCD	KBD1	00F6	KBD2	00F8	KCOMM	066C
KEYADD	027D	KEYBRD	0143	KEYINT	0275	KEYINX	0273	KEYJMP	4017	KEYRET	0252
KYRDLP	014D	LABEND	4027	LABSTK	4029	LBINX	08AB	LCOMM	06BB	LDF2	071A
LDLP1	06C4	LDLP2	06D3	LDLP3	0704	LDLP8	08A3	LDN2	06BD	LDN6	06BC
LINEND	00C1	LIX	022B	LIX2	0234	LIXIMM	0239	LIXSUB	00AD	LODFIL	0717
LOOPGF	0BEF	LSTEXP	406E	LX	4000	LY	4015	LZ	406E	MCOMM	0725
MFLAG	4000	MIDWAY	0C6F	MKDEF	0016	MRGCHR	0017	MRGIN	0350	MRGLP1	0359
MSKINT	004A	MSMTCH	02CE	NCOMM	0729	NEGEND	0CAF	NEGFLG	4077	NEGNOT	0CAE
NGENT	0CB3	NGHNDL	0C32	NLBLF	0C44	NLONP	0023	NOBRK1	0623	NOBXRK	061E
NOGCNT	061F	NOLIST	0858	NORMSV	0691	NOTBRK	0213	NOTHL	0CE9	NOTIXY	096B
NRM2	0490	NRM4	04A4	NRMCHR	048F	NULIN2	04E4	NULINE	04D7	NXTPLN	0817
NXTSTP	0879	OCOMM	0732	OFFSET	400B	OFRM1	016B	OFRM2	016F	OUTFRM	0167
PARSE	0825	PC1	409A	PCHAR	07F7	PCOMM	0751	PDOT	0808	PDOTX	07F8
PRBUFF	4028	PRCLR	02FF	PREADY	07D8	PRIGET	0923	PRNTER	07CC	PROFF	081D
PRTJMP	4015	PRTRET	07D1	PSLP	0755	PTCON	00CB	PUTA	0B9C	PUTB	0B9B
PUTDE	0317	PUTDEF	0314	PUTENT	0BAD	PUTNN	0B8D	PUTNNX	0B91	PUTOUT	0BBE
QCOMM	0761	QUOTE	000B	RASCON	07E1	RASPRS	0043	RASTER	07D4	RCOMM	0761
RCOMP	0D98	RCOMPX	0DAF	RDC1	07A1	RDC2	07A3	RDC3	07B1	RDC4	07B3
RDC5	07C4	RDCASS	07A1	RDCX	07AF	REG	4078	REGCOD	4079	REGIM	409A
REPEAT	406F	RESTOR	0628	RETNR	0C23	RLB	0B95	RLOAD	0CE4	RUBDNO	037E
RUBOK	0376	SAVMEM	018B	SAVSTR	001B	SFILE	067F	SHFT0	0424	SHFT1	03BC
SHFT8	0454	SHFT9	042B	SHFTD	0337	SHFTE	0396	SHFTF	033B	SHFTG	033F
SHFTQ	036A	SHFTR	03AF	SHFTT	03B5	SHFTW	0392	SHIFTS	0337	SHOME	03AC
SINGLE	01D5	SINGLX	01F0	SNLP1	0699	SNLP2	06A9	SP1	40B0	SRECUR	067A
SSCNT	4009	START	0000	STENT	039D	STKLOW	407A	STMEND	400F	STRBOK	0772
STRCDB	0762	STRSCH	0761	TABLE	0082	TEMP	400D	TEMP2	406C	TIMING	0066
TXTLIM	4025	UPDG1	0416	UPDG2	041F	USAMOD	4011	V	0040	VRSION	0E07
WCNORM	08F7	WCRPT	08E0	WCTAB	0DC6	WCTABX	0DE3	WRC1	0782	WRC2	078A
WRCASS	077F	WREG	073B	WREGL	0741	WRITA	031C	WSTR1	02F0	WSTRG2	02E0
WSTRLP	02DA	WSTRNG	02D4	X1	0010	X2	0020	X3	0030	ZXJPND	0505
ZXJPTB	04F3										

ANHANG

Routines des ASZMIC

BRKCHK — BREAK CHECK

Funktion : Prueft die <Break>-Taste, simuliert Break-Bedingung, wenn <Break>-Taste gedrueckt wurde.
Aufruf : CALL BRKCHK
Wirkung : Setzt den Sync-Pegel auf 0.
Anwendung : Abbruch-Routinen

CMPSTR — COMPARE STRINGS

Funktion : Vergleicht 2 Zeichenketten
Aufruf : CALL CMPSTR ; (HL) zeigt auf die gefundene Zeichenkette -1 (DE) zeigt auf die originale Zeichenkette
Wirkung : Beendet den Vergleich beim Vorfinden eines <Newline> oder eines Zeichens < .
(Wenn die vorgefundene Zeichenkette nicht mit einem Zeichen beginnt, das "kleiner" ist als . (der Punkt), dann wird der Vergleich fuer ungueltig erklaert. Wenn das erste Zeichen der vorgefundenen (und der originalen) Zeichenkette eine Filemarke ist, muss ein <Newline> vor dem ^ Pfund-Zeichen in der gefundenen Zeichenkette stehen, damit der Vergleich akzeptiert wird.)
Wenn die Zeichenketten gleich sind, wird das Carry-Flag zurueckgesetzt (auf 0), HL zeigt auf den Beginn der gefundenen Zeichenkette, DE zeigt auf den Trenner der originalen Zeichenkette. Wenn die Zeichenketten nicht identisch sind, wird Carry-Flag gesetzt und HL und DE bleiben unversaendert.
Benutzt : A, HL, DE
Anwendung : Identifizierung der Zeichenketten

COMMANDS — COMMANDO INTERPRETATION

Funktion : ^ (Pfund-Zeichen)
Aufruf : Alle Kommandos haben die Form *COMM, wobei * ein DEBUG-Buchstabe ist.
HL zeigt auf das erste Zeichen hinter dem DEBUG-Buchstaben, das kein Blank ist. Rueckkehr ueber RET (nach LIX)
Wirkung : Fuehrt die den Befehlen zugeordneten Funktionen aus.
Anwendung : ^ (Pfund-Zeichen)

DFLIP

Funktion : Setze die Adresse, die angesprungen wird, um das Kommando zu interpretieren.
Aufruf : CALL DFLIP
HL enthaelt die neue Adresse des Kommando-Interpreters
Wirkung : - HL wird in DADDR geladen
- der alte Wert wird auf den Stack gelegt
- dann wird zurueckkehrt mit RET
Benutzt : HL
Anwendung : Der Anwender kann damit das <Newline>-Zeichen bearbeiten,

das heisst : der ASZMIC-Editor kann als Eingabe-Routine eines selbstgeschriebenen Programms benutzt werden.

DELAY5 — DELAY 5 SECONDS, DISPLAY

Funktion : Zeige 5 Sekunden lang an.
Aufruf : CALL DELAY5
Wirkung : Gibt 5 Sekunden lang einen Bildschirminhalt auf den Schirm. FRAMES wird auf 250 gesetzt und FRMSND aufgerufen. Siehe auch FRMSND wegen der Deatils zum RETurn. Es gibt auch eine andere Verzoegerung : DLY05, welche den SYNC auf 0 setzt und dann in einer Schleife fuer eine halbe Sekunde wartet, bevor sie RETUrned (benutzt A und DE).
Anwendung : ^

EDLP1 — EDITOR LOOP 1

Funktion : Einsprungpunkt fuer STRSCH, welches erwartet :
- dass DE auf die originale Zeichenkette zeigt
- dass HL auf das obere Ende der zu durchsuchenden Region zeigt
- dass BC die Laenge der Region angibt in Bytes+1
Aufruf : CALL ELDP1, Register wie oben angegeben
Wirkung : siehe STRSCH
Anwendung : Durchsuchen von Tabellen

FNDLCR — FIND LEFT CARRIAGE RETURN

Funktion : findet das erste <Newline>-Zeichen nach links.
Aufruf : RST 40, HL zeigt auf den Text
Wirkung : Schiebt HL links von dem ersten <Newline>, das links (oberhalb) der aktuellen Position ist
Benutzt : A, HL
Anwendung : Syntax-Analyse, Textbearbeitung

FNDRCR — FIND RIGHT CARRIAGE RETURN

Funktion : findet das erste <Newline>-Zeichen nach rechts.
Aufruf : RST 48, HL zeigt auf die aktuelle Position im Text
Wirkung : Schiebt HL rechts von dem ersten <Newline>, das rechts (unterhalb) der aktuellen Position ist
Benutzt : A, HL
Anwendung : Syntax-Analyse, Textbearbeitung

FRMSND — FRAMES SEND

Funktion : sendet ein Display-File zum Bildschirm, bis FRAMES 0 wird (wenn es positiv war) oder bis eine Taste gedrueckt wird.
Aufruf : CALL FRMSND, DFILE muss die Adresse eines gueltigen Display-File enthalten.
Wirkung : - Sendet Display-File zum Bildschirm
- Erzeugt die Sync-Impulse fuer die Video-Aufbereitung im Fernseher
- liest die Tastatur und bewirkt eine Entprellung der Tasten
- laesst den Cursor blinken
- RETurned mit gesetztem Carry-Flag, wenn eine Taste gedrueckt wurde und mit zurueckgesetzten Carry-Flag, wenn FRAMES Timeout meldet

Benutzt : der Wert von KEYBRD ist in BC und STMEND, nicht HL
Anwendung : Bildschirmausgabe

GETFLD GET FIELD

Funktion : Analysiere ein Feld zu einem 16-Bit Wert
Aufruf : RST 16, HL zeigt auf oder vor den Beginn des Feldes
Wirkung : - HL schreitet vor zum Trenner zur Beendigung des Feldes
- das Feld wird in 2 Bytes umgesetzt in DE und ELEM1
- das Zero-Flag wird zur Rueckkehr gesetzt, wenn kein Feld gefunden wurde
- jedes Argument, das der ASZMIC erkennen kann, darf im Feld enthalten sein.
Benutzt : alle Register, mit Ausnahme von BC, IX, IY und I
Anwendung : unendlich

GET2 GET 2 FIELS

Funktion : Analyse von bis zu 2 Feldern
Aufruf : CALL GET 2, HL zeigt auf oder vor den Beginn der Felder
Wirkung : - Benutzt GETFLD
- laedt ARG1 und BC mit dem Wert des ersten Feldes
- laedt ARG2 und DE mit dem Wert des zweiten Feldes
- Zero-Flag wird zur Rueckkehr gesetzt, wenn weniger als 2 Felder gefunden wurden
Benutzt : I, IX, IY bleiben unveraendert
Anwendung : Syntax-Analyse

IGNBLK IGNORE BLANK

Funktion : bewegt das HL-Registerpaar, damit es auf ein Zeichen zeigt, das kein Blank ist
Aufruf : RST 32, HL zeigt auf auf die Zeichenkette
Wirkung : HL wird witergeschoben, bis es auf ein Zeichen zeigt, das kein Blank ist. A enthaelt dann dieses Zeichen.
Anwendung : Syntax-Analyse

KEYBRD KEYBOARD-READ

Funktion : Scannt die Tastatur-Matrix
Aufruf : CALL KEYBRD
Wirkung : Liest die Tastatur-Matrix in H (D5 bis D1, Shift als D0), dazu die 8 Adress-Leitungs-Bits in L.
Wenn sowohl H, als auch L =:FF sind, wurde keine Taste gedrueckt.
Benutzt : A, BC, DE, HL. HL wird auch in STMEND gespeichert (siehe auch den Schaltplan des ZX 81 wegen des Tastatur-Anschlusses
Anwendung : Tastatur lesen; initiiert den vertikalen Sync-Impuls

KEYINT KEYBOARD INTERPRETATION

Funktion : decodiert eine gedrueckte Taste
Aufruf : CALL KEYINT, BC enthaelt jenes Muster, das HL bei KEYBRD erhalten hat
Wirkung : - sowohl B als auch C muessen mindstens 1 Bit enthalten, das 0 ist
- Carry-Flag wird zur Rueckkehr gesetzt, wenn mehrere Tasten (unerlaubt) gleichzeitig gedrueckt wurden

ansonsten enthaelt HL die absolute Adresse des Bytes und A ist die Differenz zwischen Beginn der Tabelle zum Zeichen : HL=Tabelle-1+(A)
Beachten Sie, dass die geshifteten Tasten A bis G und Q bis T und 1 bis 0 als nicht-gueltiges Zeichen ausgewertet werden.
Anwendung : Auswertung der Tastatur-Eingabe

LIX

Funktion : Rueckkehr-Punkt fuer alle Abarbeitungs-Routinen des ASZMIC
Aufruf : ^
Wirkung : ^
Anwendung : ^

MSKINT MASK INTERRUPT

Funktion : initialisiert einen Teil der Umgebung des ASZMIC
Aufruf : CALL MSKINT
Wirkung : laed I=14, IY=:4000, Bit in MFLAG wird auf 0 zurueckgesetzt setzt Interrupt-Modus auf 1
Anwendung : Programme, in die mit dem J-Kommando gesprungen wird, koennen dies benutzen, wenn sie Routinen des ASZMIC benutzen

OUTFRM OUTPUT FRAME

Funktion : sendet einen Bildrahmen an den Fernseher
Aufruf : CALL OUTFRM, ein gueltiges DFILE wird benoetigt
B sollte die Zahl der zu schreibenden Linien enthalten
C sollte die Zahl der leeren Linien am Kopf des Bildes enthalten
Wirkung : gibt einen Bildrahmen aus
Anwendung : Display

OFRM1 OUTPUT FRAME 1

Funktion : OUTFRM fuer besondere Anwendungen
Aufruf : CALL OFRM1, wie bei OFRM, aber zusatelich muss das D-Register mit der Zahl der Raster pro Zeile enthalten.
Wirkung : ^
Anwendung : Graphik

OFRM2 OUTPUT FRAME 2

Funktion : besonderes Display
Aufruf : CALL OFRM2, wie OFRM, aber zusaetzlich ist der Anwender verantwortlich dafuer, die OUT-Anweisung und den vertikalen Sync-Impuls zu loeschen und den Akkumulator A mit der Zahl der M1-Impulse, bevor der erste horizontale Sync-Impuls benoetigt wird
Wirkung : ^
Anwendung : besondere Graphik

PRCLR PRINTER BUFFER CLEAR

Funktion : loescht den Printer-Puffer, belegt ihn mit 0
Aufruf : CALL PRCLR
Wirkung : loescht Drucker-Puffer

Benutzt : HL, BC, DE
Anwendung : Misbrauch des Drucker-Pufferr durch den Anwender

PRNTER — PRINTER

Funktion : druckt eine Zeile auf dem Sinclair-Drucker
Aufruf : CALL PRNTER
- HL zeigt auf den Beginn der zu druckenden Zeile
- Die Zeile wird gedruckt, bis ein <Newline>-Zeichen vorgefunden wird
- Bei Rueckkehr zeigt HL auf das Zeichen nach dem beendenden <Newline>

Wirkung : ^
Benutzt : AF, BC, DE, HL
Anwendung : Drucken

PUTDE — OUTPUT DE

Funktion : Umwandlung einer Hex-Zahl
Aufruf : CALL PUTDE
- HL zeigt auf den Ausgabe-Bereich
- DE zeigt die Zahl an, die umgesetzt werden soll

Wirkung : - benutzt WRITA, um den Inhalt des DE-Registers als eine 4-stellige Hex-Zahl ab dem Ort abzulegen, wohin der Inhalt von HL zeigt.
- HL wird hochgezaehlt hinter die letzte Stelle
- PUTDEF ist ein Einsprungpunkt, der HL auf PRBUFF+1 vorbelegt

Benutzt : A, HL
Anwendung : Ausgabe-Routinen

RDCASS — READ CASSETTE

Funktion : liest ein Byte von der Cassette
Aufruf : CALL RDCASS
Wirkung : - kehrt zurueck mit einem gelesenen Byte im Akkumulator
- kann mit <Break>-Taste unterbrochen werden
- man hat 800 Mikrosekunden Zeit, um das Byte zu verarbeiten, bevor RDCASS wieder aufgerufen werden muss, um das nachste Byte einzufangen

Benutzt : A, HL, BC, DE
Anwendung : besondere der Cassetten-Schnittstelle

SHIFTS — SHIFT KEYS

Funktion : ^
Aufruf : - alle Shift-Befehle werden aufgerufen mit der Folge CALL SHIFT*, wobei* das geshiftete Zeichen ist
- HL muss beim Aufruf auf Ziel des Cursors zeigen HL=(CURSOR)

Die Aktion des SHIFT wird entsprechend ausgefuehrt und dann wird zum Aufrufer RETurned.

Wirkung : ^
Anwendung : ^

START — START

Funktion : Breakpoint oder - wenn I=0 ist - Neustart
Aufruf : RST 0
Wirkung : - Neustart initialisiert den ASZMIC

- ein Breakpoint bewirkt das Retten des Kontexts und die Rueckkehr zum ASZMIC

Anwendung : Rueckkehr in den Monitor

STRSCH — STRING SEARCH

Funktion : Suche einer Zeichenkette
Aufruf : CALL STRSCH, HL zeigt auf das erste Zeichen der originalen Zeichenkette
Wirkung : Wenn (HL)<DSPBGN+40 ist, dann wird HL erst mit CURSOR geladen (Verwendung des SHIFT-Macro).
Der Textbereich von (HL) bis DSPBGN+50 wird nach der einer Zeichenkette durchsucht, die mit dem Original uebereinstimmt.
CMPSTR wird benutzt, daher ist Carry-Flag gesetzt, wenn
- nicht gefunden
- oder wenn gefunden und HL und DE wie bei CMPSTR, mit der Ausnahme, dass, wenn nicht gefunden, HL auf DSPBGN+39 zeigt.

Benutzt : AF, BC, DE, HL. EDLP1 ist ein Einsprungpunkt, welcher erwartet, dass
- DE auf die Original-Zeichenkette zeigt
- HL auf das obere Ende der zu durchsuchenden Region zeigt
- BC die Zahl der zu durchsuchenden Bytes +1 ist

Anwendung : Syntax-Analyse

WRCASS — WRITE TO CASSETTE

Funktion : schreibt ein Byte auf Band
Aufruf : CALL WRCASS, HL zeigt auf das zu auszugebende Byte
Wirkung : Das auf Band geschriebene Byte entspricht dem Standard des Sinclair-BASIC. Bei der Rueckkehr ist HL hochgezaehlt worden und der Akkumulator enthaelt das gerade ausgegebene Byte.
Benutzt : AF, BC, DE, HL
Anwendung : besondere Operationen mit dem Recorder

WRITA — WRITE ACCUMULATOR

Funktion : wandelt ein Byte um in Hex-Zahl
Aufruf : CALL WRITA
- A enthaelt die Zahl
- HL zeigt auf die zu beschreibende Region

Wirkung : - Zahl wird in 2-stellige Hex-Zahl umgesetzt
- HL erhoeht und zeigt hinter die 2. Stelle

Anwendung : Ausgaben

WSTRNG — WRITE STRING

Funktion : Schreibt den Inhalt des PRBUFF (Drucker-Puffer) auf den Bildschirm oder sendet ihn zum Drucker.
PRBUFF enthaelt mindestens 1 Zeichen, das kein Blank ist

Wirkung : - B wird mit der Laenge des PRBUFF vorbelegt.
- PRBUFF wird von hinter her mit <Newline>-Zeichen aufgefuellt, bis ein Zeichen gefunden wird, das kein Blank ist. Macht weiter bei WSTRNG2.

Benutzt : A, B, HL
Anwendung : Ausgabe

WSTRG2 WRITE STRING 2

Funktion : siehe WSTRNG
Aufruf : CALL WSTRG2 B wird vorbelegt mit der maximalen Zahl der zu schreibenden Zeichen
Wirkung : Wenn Bit 1 von ASSFLG gesetzt ist, wird PRNTER aufgerufen, gefolgt von PRCLR und dan RETurned.
DE stellt auf PRBUFF
Benutzt : macht weiter bei WSTR1
Anwendung :

WSTR1 WRITE STRING 1

Funktion : schreibt eine Zeichenkette auf den Bildschirm
Aufruf : CALL WSTR1
DE zeigt auf den Beginn der Zeichenkette
B ist vorbelegt mit der maximalen Zahl der zu schreibenden Zeichen
Wirkung : - schreibt ab Ziel von DE (also PRBUFF) auf den Schirm
- geht so lange vorwaerts, bis der Zahler in B erschoepft ist oder ein <Newline> geschrieben wurde
- macht weiter bei PRCLR
Benutzt : alle Register, ausgenommen IX, IY und I
Anwendung : Ausgabe

ANHANG

Anwendungsbeispiele

Die nachfolgenden Anwendungsbeispiele wurden vom englischen Original abgeschrieben. Es kann daher nicht ausgeschlossen werden, dass sich Fehler eingeschlichen haben. Bitte sehen Sie sich die Programme aufmerksam an und - wenn Sie einen Fehler finden oder etwas nicht verstehen - wenden sich an Decker & Computer.

Der Aufruf der Anwendungsprogramme erfolgt mit dem H-Kommando. Nehmen Sie die =-Zuweisung (Equate), um die externen Label zu deklarieren (definieren). Die ORG-Anweisung machen Sie der Groesse Ihres RAMs entsprechend.

1 Schreiben eines Zeichens auf den Bildschirm

```
START LD A,"X"
 CALL NRM2
 RET
```

2 Schreiben einer Zeichenkette auf den Bildschirm

```
START LD DE,STRING
 LD B,7
 CALL WSTR1
 RET
STRING DEFM "HALLO"
```

3 Zwei Zahlen addieren und das Ergebnis anzeigen

Mit H START Zahl1 Zahl2 aufrufen.

```
START RST 16 ;GETFLD FELD LESEN
 PUSH DE
 RST 16 ;DIE ZWEITE ZAHL LESEN
 POP HL
 ADD HL,DE
 EX DE,HL
 CALL PUTDEF
 LD (HL),:76
 CALL WSRTNG
 RET
```

4 Palindrome

Mit H START aufrufen. Wichtig, weil es Ihnen zeigt, wie sie Ihr eigenes Programm einbinden (linken) als einen Kommando-Interpreter koennen und wie man den ASZMIC am Ende restauriert. Nach dem H START geben Sie die Zeile ein, die sie umgedreht haben moechten.

```
START LD HL,HANDLE; ADRESSE DES ANWENDER KOMMANDO INTERPRETERS
 CALL DFLIP; AUSTAUSCH DER ADRESSEN DER KOMMANDO-INTERPRETER
 JP LIX; SPRUNG ZUM ASZMIC KONTEXT DER NOCH AUF DEM STACK IST
;
HANDLE RST 48; FNDRCR CARRIAGE RETURN SUCHEN
```

```
 RST 40; PRUEFE CARRIAGE RETURN SUCHEN
 SET 7,(IY); EDIT MODUS EINSCHALTEN
LOOP LD A,(HL)
 PUSH HL
 PUSH AF
 CALL NRM2
 POP AF
 POP HL; ZEICHEN GESCHRIEBEN
 DEC HL
 CP :76; WURDE EIN <NEWLINE> GESCHRIEBEN /
 JR NZ,LOOP; WENN NICHT DANN WEITERMACHEN
 POP AF; LIX RETURN AUF DEM STACK LOESCHEN
 POP HL  ;ADRESSE DES ASZMIC-KOMMANDOINTERPRETERS DURCH  DFLIP
;GESPEICHERT
 LD (DADDR),HL; ADRESSE DES ASZMIC-KOMMANDO-INTERPRETERS WIEDER
;HERSTELLEN
 RES 7,(IY); NUR WENN SIE IN DEN DEBUG-MODUS ZURUECKKEHREN WOLLEN
 RET
```

5 Drucken einer Zahl von 0 bis 6

Einsprung mit H START

```
START XOR A
 LD HL,PRBUFF
LOOP LD B,A ; A RETTEN
 CALL WRITA
 LD A,B; AKKU WIEDER HOLEN
 INC HL;
 INC A
 CP 7; IST ES 7 ?
 JR NZ,LOOP
 CALL WSTRNG
 RET
```

6 Drucken einer Zeile

```
START LD HL,TEXT
 CALL PRNTER
 RET
TEXT DEFM "HURRA ES GEHT"
 DEFB :76; <NEWLINE>-ZEICHEN
```

7 Erzeugen eines BASIC-Programms mit einer REM-Zeile darin

Den Inhalt der REM-Zeile koennen Sie selbst schreiben. Zur Beachtung : Die nachfolgenden Zeilen haben in Spalte 1 ein Blank - es sind ja Anweisungen und keine Label !

```
 ORG :4000
 DEFM "12345678"
 DEFB "9"+128; LETZTES ZEICHEN INVERTIERT WIE AUCH BEIM ZX 81
 DEFB 0
 DEFW 1
 DEFW DFILE
 DEFW DFILE+1
 DEFW VARS
 DEFW 0
 DEFW VARS+1
 ORG $+9
 DEFW MEMBOT
 DEFW 512
```



```
 DEFW 0
 DEFW :FDBF
 DEFW :37FF
 DEFW DFILE
 ORG $+9
 DEFW -1
 DEFW 0
 DEFW :218C
 DEFW :4018
 ORG $+32
 DEFB :76
MEMBOT ORG $+32
 DEFW 256
 DEFW DFILE-PROG
PROG DEFB :EA; REM-STATEMENT
;
;
; FUEGEN SIE HIER DEN OBJECT-CODE EIN
; DER NACH DEM REM STEHEN SOLL
;
;
DFILE=$+1
 DEFW :7676 ;<NEWLINE>-ZEICHEN
 DEFW :7676
 DEFW :7676
 DEFW :7676
 DEFW :7676
 DEFW :7676
 DEFW :7676
 DEFW :7676
 DEFW :7676
 DEFW :7676
 DEFW :7676
 DEFW :7676
 DEFW :7676
; 13 MAL 2 <NEWLINES> ERGIBT 26 ZEILEN !
VARS DEFB 128
```

8 Programme zur Erzeugung von Graphik

Einsprung mit H :7000

```
; DIE DEFINITIONEN HIER SIND FUER DIE VERSION E04
; SIE MUESSEN DIE FUER IHR ASZMIC-ROM GUELTIGEN
; ADRESSEN EINGEBEN
;
 ORG :7000
INTJMP=:4019
DFILE=:4023
PCONE=:409A
TXTLIM=:4025
FRAMES=:4070
OFRM1=:16D
MFLAG=:4000
RESTOR=:628
KEYBRD=:145
STMEND=:400F
SAVMEM=:18D
NNN=46
TOPS=1
PIXSIZE=1
IDLE=30
RASTERS=255
DISPEND=:6501
```

```
DSTART=:4100
;
; KERNEL ROUTINE
;
KERNEL LD HL,ONEP
 LD (INTJMP),HL ; DIE BEARBEITUNG EINES BREAK DURCH EIGENEN
; INTERRUPT-HANDLER BEARBEITEN
 CALL CLEAR; DISPLAY AUFBEREITEN
 LD HL,UPROG
 LD (PCONE),HL ;ERSTES RETTEN DES KONTEXTES UM ZUM
; UPROG RETURNEN ZU KOENNEN
;
ONEP CALL KEYBRD; TASTATUR LESEN (Zeile ?)
 LD HL,(FRAMES)
 INC HL
 LD (FRAMES),HL; FRAME ZAEHLER MANIPULIEREN
 LD B,IDLE
 DJNZ $ ; WARTESCHLEIFE
 LD D,PIXSIZE
 LD B,RASTERS
 LD C,TOPS
 CALL OFRM1; SCHREIBT EINEN TEILWEISEN BILDINHALT
 LD A,1
 LD (MFLAG),A; DEM HANDLER FUER DEN NMI MITTEILEN
; DASS ER BEI 0 EINEN BREAK AUSFUEHREN SOLL
 LD C,NNN+NNN+1; C FUER DEN G-BEFEHL VORBEREITEN
 EXX
 JP RESTOR; GEHE IN DIE MITTE DES HANDLERS FUER DEN G-BEFEHL
;
;AUFBEREITUNG DES DISPLAY-FILE > KANN VERBESSERT WERDEN
;
CLEAR LD HL,DSTART
 LD DE,DSTART+2
 LD BC,:2400
 LD (HL),:76; <NEWLINE>
 LD (DFILE),HL
 INC HL
 LD (HL),0
 LDIR
 RET
;
; PLOT UND UNPLOT SUBROUTINES Y-> B  X-> C
;
PLOT LD D,15
 JR $+4
UNPLOT LD D,0
 LD A,3; MASKIEREN FUER DIE NUMMER DES PIXELS IM BYTE
 AND C
 SRL C
 SRL C
 INC A; C IST NUN X BYTE, NICHT PIXEL
 LD E,16; BIT MASKE
LOOP1 SRL E
 DEC A
 JR NZ,LOOP1 ;SCHLEIFEN BIS 0-3 UMGEWANDELT IN 8,4,2,1 IN E
 CP D; UNPLOT
 JR NZ,ENT2; SPRUNG WENN NICHT
 LD A,E
 CPL
 LD D,A; BYTE MASKIEREN
ENT2 PUSH DE
 LD HL,DISPEND; ZUR BERECHNUNG DER Y-ADRESSE VORBEREITEN
 LD DE,:1200
 LD A,B
 LD B,8
```



```
 JR NC,NOSUBT
 OR A
 SBC HL,DE; SUBTRAHIEREN NUR
; WENN 2-ER POTENZ IN Y ENTHALTEN IST
NOSUBT SRL D
 RR E; SHIFT SUBTRAHIERER
 DJNZ LOOP2
 ADD HL,BC; ADDIEREN AUF X UM ZIEL-BYTE ZU BEKOMMEN
; NUN DAS BYTE BEARBEITEN, IN DAS DAS PIXEL SOLL
 POP DE
 LD A,(HL)
 BIT 7,A; DIE TRICKSEREI MIT DEM INVERSEN UMGEHEN
 JR Z,$+3
 CPL; CONVERTIERT 8,2,1,0 IN 3,2,1,0
 AND 15
;
 OR E
 AND D; PIXEL NUN DRIN
; WIEDER CODIEREN
 CP 8
 JR C,NOTINV
; WIR BRAUCHEN KEINE INVERTIERUNG
 CPL
 AND :87
NOTINV LD (HL),A
;
; DER FOLGENDE CODE BEWIRKT, DASS DIE SUBROUTINE IN
; EINER SCHLEIFE LAEUFT BIS EIN BILDRAHMEN
; GESENDET WORDEN IST
; ES IST NICHT WICHTIG, VERLANGSAMT ABER DAS PLOTTEN
; EIN WENIG
;
NLINE LD DE,FRAMES
 LD A,(DE)
 LD B,A
LOOPX LD A,(DE)
 CP B
 JR Z,LOOPX
; ENDE DER VERZOEGERUNG
 RET
;
; LINE ODER UNLINE VON XY BIS X'Y'
;
; X-> E Y->D X'-> C Y'-> B
;
LINE LD A,15
 JR $+3
UNLINE XOR A
 LD (DORDEL),A; DO OR DELETE -> MERKER FUER PLOT ODER UNPLOT
 LD HL,XMID
; INITIALISIEREN DER ZELLEN
 XOR A
 LD (HL),A
 INC HL
 LD (HL),E
 INC HL
 LD (HL),A
 INC HL
 LD (HL),D
 INC HL
 LD (HL),C
 INC HL
 LD (HL),B
;
```

```
; NUN DIE SCHRITTE AUSRECHNEN
;
 LD L,C
 LD D,A
 LD H,A
 SBC HL,DE
 PUSH HL; X'-X
 LD L,B
 LD H,A
 LD A,(YMID+1); MID SOLL WOHL MIDDLE HEISSEN -> ALSO MITTE
 LD E,A
 SBC HL,DE; HL IST NUN Y'-Y
 POP DE; LINKSBUENDIGE SCHRITTE BIS EINER >= 128/256
JUSTIFY LD C,L; JUSTIFY = BUENDIG AUSRICHTEN
 LD A,H
 RLCA
 RL L
 ADC A,0
 EX AF,AF'
 LD B,E
 LD A,D
 RLCA
 RL E
 ADC A,0
 JR NZ,JSTDUN ; JSTDUN = JUST DONE = GERADE FERTIG
 EX AF,AF'
 JR Z,JUSTIFY
JSTDUN LD E,B
 LD B,H
;
; DE IST X-INKREMENT BC IST Y-INKREMENT
; INKREMENT IST EINE SCHRITTWEITE
;
LLOOP LD HL,(XMID)
 ADD HL,DE; SCHRITTWEITE
 LD (XMID),HL
 LD HL,(YMID)
 ADD HL,BC
 LD (YMID),HL
 EXX; DIE SCHRITTE RETTEN
LPLOT LD A,(XMID+1)
 LD C,A; X
 LD A,(YMID+1)
 LD B,A; Y
 LD A,(DORDEL)
 LD D,A
 CALL UNPLOT+2; PLOTTEN ODER UNPLOTTEN DES NEUEN PUNKTES
 LD HL,XPRIM
 LD A,(XMID+1)
 CP (HL)
 INC HL
 EX AF,AF'
 LD A,(YMID+1)
 CP (HL)
 EXX
 JR NZ,NYLIM; DAS Y-LIMIT WURDE NOCH NICHT ERREICHT
 LD BC,0; DIE Y-SCHRITTWEITE AUF 0 SETZEN, WENN Y AM LIMIT
NYLIM EX AF,AF'
 JR NZ,NXLIM
 LD DE,0; DIE X-SCHRITTWEITE AUF 0 SETZEN, WENN X AM LIMIT
NXLIM LD A,D; X-SCHRITT UND Y-SCHRITT BEIDE 0
; ALS ENDE-DER-ZEILE-TEST
 OR E
 OR B
 OR C
```



```
 JR NZ,LLOOP
 RET
;
; DATENBEREICH
;
XMID DEFW 0; DIE REIHENFOLGE IST WICHTIG !
YMID DEFW 0
XPRIM DEFB 0
YPRIM DEFB 0
DORDEL DEFB 0
;
```

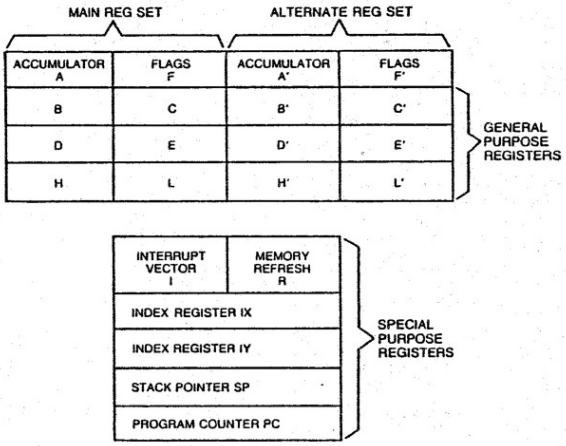
M O I R E E

```
UPROG LD SP,:7F00
MAILOOP LD DE,:4078
 LD BC,:101
MLOOP PUSH DE
 PUSH BC
 CALL LINE
 POP BC
 POP DE
 INC D; D VERDOPPELN
 INC D
 INC B; B MAL 4 NEHMEN
 INC B
 INC B
 INC B
 LD A,250
 CP B
 JR C,OUTCOD
 LD A,(STMEND)
 AND 1
 JR NZ,MLOOP
;ZXCV HIER GEDRUECKT - WAS IMMER DAS SEIN MAG
 LD HL,:8000; ERSTER NICHT-BESCHREIBBARER PLATZ IM SPEICHER
 JP SAVMEM; OBERKANTE FESTLEGEN
OUTCOD CALL CLEAR
 JP MAINLOOP
```

S T R U C T U R E S

```
UPROG LD SP,:7F00
 JR OUTCOD
MAINLOOP LD IX,XVAR
 CALL PROSUB
 LD IX,YVAR
 CALL PROSUB
 LD B,(IX+VAR)
 LD A,(XVAR)
 LD C,A
 CALL PLOT
 LD A,(STMEND)
 AND 1
 JR Z,OUTCOD
 JR MAINLOOP
;
OUTCOD LD HL,DUMMY
 LD DE,XVAR
 LD BC,12
 LDIR; INITIALISIEREN DES VARIABLEN-BEREICHS
 CALL CLEAR
```

```
 LD HL,(FRAMES)
 LD A,(HL)
 AND 15
 INC A
 LD (XVAR+1),A
 INC HL
 LD A,(HL)
 AND 7
 INC A
 LD (YVAR+1),A
 JR MAINLOOP
;
PROSUB LD A,(IX+VAR)
 ADD A,(IX+DIR)
 LD (IX+VAR),A
 CP (IX+MAX)
 JR NC,OUTCOD
 CP (IX+LOW)
 JR NC,POSCHECK ; POSITIVE-CHECK
 LD A,(IX+INC)
 CPL
 ADD A,(IX+LOW)
 LD (IX+LOW),A
 CALL DIDLIM
POSCHECK LA A,(IX+VAR)
 CP (IX+HIGH)
 RET C
 LD A,(IX+INC)
 ADD A,(IX+HIGH)
 LD (IX+HIGH),A
;
DIDLIM LD A,(IX+DIR)
 NEG
 LD (IX+DIR),A
 LD A,(IX+INC)
;
 DEC A
 JR NZ,$+3
 INC A
;
 LD (IX+INC),A
 RET
;
; DATENBEREICH
;
VAR=0
INC=1
DIR=2
LOW=3
HIGH=4
MAX=5
;
XVAR ORG $+6
YVAR ORG $+6
DUMMY DEFB 75
 DEFB 8
 DEFB -1
 DEFB 70
 DEFB 90
 DEFB 135
 DEFB 110
 DEFB 12
 DEFB 1
 DEFB 90
 DEFB 130
 DEFB 240
 DEFB 0
```

Z80-CPU REGISTER CONFIGURATION

MOSTEK.
1215 W. Crosby Rd. • Carrollton, Texas 75006 • 214/323-6000
In Europe, Contact MOSTEK Brussels
270-272 Avenue de Tervuren (BTE21), B-1150 Brussels, Belgium.
Telephone 762 18 80

PRINTED IN USA February 1978
Publication No. MK78518

SUMMARY OF FLAG OPERATION

Instruction	D7 S	Z		H		P/V	N	D0 C	Comments
ADD A, s; ADC A, s	↕	↕	X	↕	X	V	0	↕	8-bit add or add with carry
SUB s; SBC A, s; CP s; NEG	↕	↕	X	↕	X	V	1	↕	8-bit subtract, subtract with carry, compare and negate accumulator
AND s	↕	↕	X	1	X	P	0	0	Logical operations
OR s; XOR s	↕	↕	X	0	X	P	0	0	
INC s	↕	↕	X	↕	X	V	0	•	8-bit increment
DEC s	↕	↕	X	↕	X	V	1	•	8-bit decrement
ADD DD, SS	•	•	X	X	X	•	0	↕	16 bit add
ADC HL, SS	↕	↕	X	X	X	V	0	↕	16-bit add with carry
SBC HL, SS	↕	↕	X	X	X	V	1	↕	16-bit subtract with carry
RLA; RLCA; RRA; RRCA	•	•	X	0	X	•	0	↕	Rotate accumulator
RL s; RLC s; RR s; RRC s; SLA s; SRA s; SRL s	↕	↕	X	0	X	P	0	↕	Rotate and shift locations
RLD; RRD	↕	↕	X	0	X	P	0	•	Rotate digit left and right
DAA	↕	↕	X	↕	X	P	•	↕	Decimal adjust accumulator
CPL	•	•	X	1	X	•	1	•	Complement accumulator
SCF	•	•	X	0	X	•	0	1	Set carry
CCF	•	•	X	X	X	•	0	↕	Complement carry
IN r, (C)	↕	↕	X	0	X	P	0	•	Input register indirect
INI; IND; OUTI; OUTD	X	↕	X	X	X	X	1	•	Block input and output
INIR; INDR; OTIR; OTDR	X	1	X	X	X	X	1	•	Z = 0 if B ≠ 0 otherwise Z = 1
LDI; LDD	X	X	X	0	X	↕	0	•	Block transfer instructions
LDIR; LDDR	X	X	X	0	X	0	0	•	P/V = 1 if BC ≠ 0, otherwise P/V = 0
CPI; CPIR; CPD; CPDR	X	↕	X	X	X	↕	1	•	Block search instructions Z = 1 if A = (HL), otherwise Z = 0 P/V = 1 if BC ≠ 0, otherwise P/V = 0
LD A, I; LD A, R	↕	↕	X	0	X	IFF	0	•	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag
BIT b, s	X	↕	X	1	X	X	0	•	The state of bit b of location s is copied into the Z flag

The following notation is used in this table:

Symbol	Operation
C	Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result.
Z	Zero flag. Z=1 if the result of the operation is zero.
S	Sign flag. S=1 if the MSB of the result is one.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V=1 if the result of the operation is even, P/V=0 if result is odd. If P/V holds overflow, P/V=1 if the result of the operation produced an overflow.
H	Half-carry flag. H=1 if the add or subtract operation produced a carry into or borrow from bit 4 of the accumulator.
N	Add/Subtract flag. N=1 if the previous operation was a subtract.
	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
↕	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care".
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction.
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range <0, 255>
nn	16-bit value in range <0, 65535>

8-BIT LOAD GROUP
'LD'

DESTINATION		SOURCE: IMPLIED I	IMPLIED R	REGISTER A	B	C	D	E	H	L	REG INDIRECT (HL)	(BC)	(DE)	INDEXED (IX+d)	(IY+d)	EXT. ADDR. (nn)	IMME. n
REGISTER	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
REG INDIRECT	(HL)			77	70	71	72	73	74	75							36 n
	(BC)			02													
	(DE)			12													
INDEXED	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
EXT.ADDR.	(nn)			32 n n													
IMPLIED	I			ED 47													
	R			ED 4F													

8-BIT LOAD GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD r, s	r ← s	•	•	X	•	X	•	•	•	01 r s		1	1	4	r, s Reg.
LD r, n	r ← n	•	•	X	•	X	•	•	•	00 r 110 ← n →		2	2	7	000 B 001 C
LD r, (HL)	r ← (HL)	•	•	X	•	X	•	•	•	01 r 110		1	2	7	010 D
LD r, (IX+d)	r ← (IX+d)	•	•	X	•	X	•	•	•	11 011 101 01 r 110 ← d →	DD	3	5	19	011 E 100 H 101 L
LD r, (IY+d)	r ← (IY+d)	•	•	X	•	X	•	•	•	11 111 101 01 r 110 ← d →	FD	3	5	19	111 A
LD (HL), r	(HL) ← r	•	•	X	•	X	•	•	•	01 110 r		1	2	7	
LD (IX+d), r	(IX+d) ← r	•	•	X	•	X	•	•	•	11 011 101 01 110 r ← d →	DD	3	5	19	
LD (IY+d), r	(IY+d) ← r	•	•	X	•	X	•	•	•	11 111 101 01 110 r ← d →	FD	3	5	19	
LD (HL), n	(HL) ← n	•	•	X	•	X	•	•	•	00 110 110 ← n →	36	2	3	10	
LD (IX+d), n	(IX+d) ← n	•	•	X	•	X	•	•	•	11 011 101 00 110 110 ← d → ← n →	DD 36	4	5	19	
LD (IY+d), n	(IY+d) ← n	•	•	X	•	X	•	•	•	11 111 101 00 110 110 ← d → ← n →	FD 36	4	5	19	
LD A, (BC)	A ← (BC)	•	•	X	•	X	•	•	•	00 001 010	0A	1	2	7	
LD A, (DE)	A ← (DE)	•	•	X	•	X	•	•	•	00 011 010	1A	1	2	7	
LD A, (nn)	A ← (nn)	•	•	X	•	X	•	•	•	00 111 010 ← n → ← n →	3A	3	4	13	
LD (BC), A	(BC) ← A	•	•	X	•	X	•	•	•	00 000 010	02	1	2	7	
LD (DE), A	(DE) ← A	•	•	X	•	X	•	•	•	00 010 010	12	1	2	7	
LD (nn), A	(nn) ← A	•	•	X	•	X	•	•	•	00 110 010 ← n → ← n →	32	3	4	13	
LD A, I	A ← I	↕	↕	X	0	X	IFF	0	•	11 101 101 01 010 111	ED 57	2	2	9	
LD A, R	A ← R	↕	↕	X	0	X	IFF	0	•	11 101 101 01 011 111	ED 5F	2	2	9	
LD I, A	I ← A	•	•	X	•	X	•	•	•	11 101 101 01 000 111	ED 47	2	2	9	
LD R, A	R ← A	•	•	X	•	X	•	•	•	11 101 101 01 001 111	ED 4F	2	2	9	

Notes: r, s means any of the registers A, B, C, D, E, H, L
IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation: • = flag not affected, 0 = flag reset, ↕ = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

16-BIT LOAD GROUP
'LD'
'PUSH' AND 'POP'

DESTINATION REGISTER	SOURCE REGISTER: AF	BC	DE	HL	SP	IX	IY	IMM. EXT. nn	EXT. ADDR. (nn)	REG. INDIR. (SP)
AF										F1
BC								01 n n	ED 4B n n	C1
DE								11 n n	ED 5B n n	D1
HL								21 n n	2A n n	E1
SP				F9		DD F9	FD F9	31 n n	ED 78 n n	
IX								DD 21 n n	DD 2A n n	DD E1
IY								FD 21 n n	FD 2A n n	FD E1
EXT. ADDR. (nn)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			
REG. IND. (SP)	F5	C5	D5	E5		DD E5	FD E5			

PUSH INSTRUCTIONS →

NOTE: The Push & Pop Instructions adjust the SP after every execution.

↑ POP INSTRUCTIONS

16-BIT LOAD GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD dd, nn	dd ← nn	•	•	X	•	X	•	•	•	00 dd0 001 ← n → ← n →		3	3	10	dd Pair 00 BC 01 DE 10 HL 11 SP
LD IX, nn	IX ← nn	•	•	X	•	X	•	•	•	11 011 101 00 100 001 ← n → ← n →	DD 21	4	4	14	
LD IY, nn	IY ← nn	•	•	X	•	X	•	•	•	11 111 101 00 100 001 ← n → ← n →	FD 21	4	4	14	
LD HL, (nn)	H ← (nn+1) L ← (nn)	•	•	X	•	X	•	•	•	00 101 010 ← n → ← n →	2A	3	5	16	
LD dd, (nn)	dd_H ← (nn+1) dd_L ← (nn)	•	•	X	•	X	•	•	•	11 101 101 01 ddl 011 ← n → ← n →	ED	4	6	20	
LD IX, (nn)	IX_H ← (nn+1) IX_L ← (nn)	•	•	X	•	X	•	•	•	11 011 101 00 101 010 ← n → ← n →	DD 2A	4	6	20	
LD IY, (nn)	IY_H ← (nn+1) IY_L ← (nn)	•	•	X	•	X	•	•	•	11 111 101 00 101 010 ← n → ← n →	FD 2A	4	6	20	
LD (nn), HL	(nn+1) ← H (nn) ← L	•	•	X	•	X	•	•	•	00 100 010 ← n → ← n →	22	3	5	16	
LD (nn), dd	(nn+1) ← dd_H (nn) ← dd_L	•	•	X	•	X	•	•	•	11 101 101 01 ddR 011 ← n → ← n →	ED	4	6	20	
LD (nn), IX	(nn+1) ← IX_H (nn) ← IX_L	•	•	X	•	X	•	•	•	11 011 101 00 100 010 ← n → ← n →	DD 22	4	6	20	
LD (nn), IY	(nn+1) ← IY_H (nn) ← IY_L	•	•	X	•	X	•	•	•	11 111 101 00 100 010 ← n → ← n →	FD 22	4	6	20	
LD SP, HL	SP ← HL	•	•	X	•	X	•	•	•	11 111 001	F9	1	1	6	
LD SP, IX	SP ← IX	•	•	X	•	X	•	•	•	11 011 101 11 111 001	DD F9	2	2	10	
LD SP, IY	SP ← IY	•	•	X	•	X	•	•	•	11 111 101 11 111 001	FD F9	2	2	10	
PUSH qq	(SP-2) ← qq_L (SP-1) ← qq_H	•	•	X	•	X	•	•	•	11 qq0 101		1	3	11	qq Pair 00 BC 01 DE 10 HL 11 AF
PUSH IX	(SP-2) ← IX_L (SP-1) ← IX_H	•	•	X	•	X	•	•	•	11 011 101 11 100 101	DD E5	2	4	15	
PUSH IY	(SP-2) ← IY_L (SP-1) ← IY_H	•	•	X	•	X	•	•	•	11 111 101 11 100 101	FD E5	2	4	15	
POP qq	qq_H ← (SP+1) qq_L ← (SP)	•	•	X	•	X	•	•	•	11 qq0 001		1	3	10	
POP IX	IX_H ← (SP+1) IX_L ← (SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 001	DD E1	2	4	14	
POP IY	IY_H ← (SP+1) IY_L ← (SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 001	FD E1	2	4	14	

Notes: dd is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs AF, BC, DE, HL
$(PAIR)_H$, $(PAIR)_L$ refer to high order and low order eight bits of the register pair respectively.
e.g. BC_L = C, AF_H = A

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ flag is affected according to the result of the operation.

EXCHANGES 'EX' AND 'EXX'

		IMPLIED ADDRESSING				
		AF'	BC', DE' & HL'	HL	IX	IY
IMPLIED	AF	08				
	BC, DE & HL		D9			
	DE			EB		
REG. INDIR.	(SP)			E3	DD E3	FD E3

BLOCK TRANSFER GROUP

DESTINATION			SOURCE: REG. INDIR.	SOURCE: (HL)
REG. INDIR.	(DE)	ED A0	'LDI' – Load (DE) ← (HL) Inc HL & DE, Dec BC	
		ED B0	'LDIR' – Load (DE) ← (HL) Inc HL & DE, Dec BC, Repeat until BC = 0	
		ED A8	'LDD' – Load (DE) ← (HL) Dec HL & DE, Dec BC	
		ED B8	'LDDR' – Load (DE) ← (HL) Dec HL & DE, Dec BC, Repeat until BC = 0	

HL points to source
DE points to destination
BC is byte counter

BLOCK SEARCH GROUP

SEARCH LOCATION: REG. INDIR.	SEARCH LOCATION: (HL)
ED A1	'CPI' Inc HL, Dec BC
ED B1	'CPIR' – Inc HL, Dec BC repeat until BC = 0 or find match
ED A9	'CPD' – Dec HL & BC
ED B9	'CPDR' – Dec HL & BC Repeat until BC = 0 or find match

HL points to location in memory to be compared with accumulator contents
BC is byte counter

EXCHANGE GROUP AND BLOCK TRANSFER AND SEARCH GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE←→HL	•	•	X	•	X	•	•	•	11 101 011	EB	1	1	4	
EX AF, AF'	AF←→AF'	•	•	X	•	X	•	•	•	00 001 000	08	1	1	4	
EXX	(BC←→BC' DE←→DE' HL←→HL')	•	•	X	•	X	•	•	•	11 011 001	D9	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ←→(SP+1) L ←→(SP)	•	•	X	•	X	•	•	•	11 100 011	E3	1	5	19	
EX (SP), IX	IX_H←→(SP+1) IX_L←→(SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 011	DD E3	2	6	23	
EX (SP), IY	IY_H←→(SP+1) IY_L←→(SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 011	FD E3	2	6	23	
LDI	(DE)←(HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	X	0	X	① ↕	0	•	11 101 101 10 100 000	ED A0	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE)←(HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 110 000	ED B0	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE)←(HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	X	0	X	① ↕	0	•	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE)←(HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A – (HL) HL ← HL+1 BC ← BC-1	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A – (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A – (HL) HL ← HL-1 BC ← BC-1	↕	② ↕	X	↕	X	(1) ↕	1	•	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A – (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	↕	(2) ↕	X	↕	X	(1) ↕	1	•	11 101 101 10 111 001	ED B9	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

Notes: ① P/V flag is 0 if the result of BC-1 = 0, otherwise P/V = 1
② Z flag is 1 if A = (HL), otherwise Z = 0.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown, ↕ = flag is affected according to the result of the operation.

8-BIT ARITHMETIC AND LOGIC

SOURCE

	REGISTER ADDRESSING A	B	C	D	E	H	L	REG. INDIR. (HL)	INDEXED (IX+d)	(IY+d)	IMMED. n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADD w CARRY 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUBTRACT 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SUB w CARRY 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARE 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

8-BIT ARITHMETIC AND LOGICAL GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD A, r	A ← A + r	↕	↕	X	↕		X	V	0	↕	10 000 r		1	1	4	r Reg.
ADD A, n	A ← A + n	↕	↕	X	↕		X	V	0	↕	11 000 110		2	2	7	000 B
											← n →					001 C
																010 D
ADD A, (HL)	A ← A+(HL)	↕	↕	X	↕		X	V	0	↕	10 000 110		1	2	7	011 E
ADD A, (IX+d)	A ← A+(IX+d)	↕	↕	X	↕		X	V	0	↕	11 011 101	DD	3	5	19	100 H
											10 000 110					101 L
											← d →					111 A
ADD A, (IY+d)	A ← A+(IY+d)	↕	↕	X	↕		X	V	0	↕	11 111 101	FD	3	5	19	
											10 000 110					
											← d →					
ADC A, s	A ← A+s+CY	↕	↕	X	↕		X	V	0	↕	001					s is any of r, n, (HL), (IX+d), (IY+d) as shown for ADD instruction. The indicated bits replace the 000 in the ADD set above.
SUB s	A ← A - s	↕	↕	X	↕		X	V	1	↕	010					
SBC A, s	A ← A - s - CY	↕	↕	X	↕		X	V	1	↕	011					
AND s	A ← A ∧ s	↕	↕	X	1		X	P	0	0	100					
OR s	A ← A ∨ s	↕	↕	X	0		X	P	0	0	110					
XOR s	A ← A ⊕ s	↕	↕	X	0		X	P	0	0	101					
CP s	A - s	↕	↕	X	↕		X	V	1	↕	111					
INC r	r ← r + 1	↕	↕	X	↕		X	V	0	•	00 r 100		1	1	4	
INC (HL)	(HL) ← (HL)+1	↕	↕	X	↕		X	V	0	•	00 110 100		1	3	11	
INC (IX+d)	(IX+d) ← (IX+d)+1	↕	↕	X	↕		X	V	0	•	11 011 101	DD	3	6	23	
											00 110 100					
											← d →					
INC (IY+d)	(IY+d) ← (IY+d)+1	↕	↕	X	↕		X	V	0	•	11 111 101	FD	3	6	23	
											00 110 100					
											← d →					
DEC s	s ← s - 1	↕	↕	X	↕		X	V	1	•	101					s is any of r, (HL), (IX+d), (IY+d) as shown for INC. DEC same format and states as INC. Replace 100 with 101 in OP Code.

Notes: The V symbol in the P/V flag column indicates that the P/V flag contains the overflow of the result of the operation. Similarly the P symbol indicates parity. V = 1 means overflow, V = 0 means not overflow. P = 1 means parity of the result is even, P = 0 means parity of the result is odd.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

GENERAL PURPOSE AF OPERATIONS

Decimal Adjust Acc, 'DAA'	27
Complement Acc, 'CPL'	2F
Negate Acc, 'NEG' (2's complement)	ED 44
Complement Carry Flag, 'CCF'	3F
Set Carry Flag, 'SCF'	37

MISCELLANEOUS CPU CONTROL

'NOP'	00	
'HALT'	76	
DISABLE INT '(DI)'	F3	
ENABLE INT '(EI)'	FB	
SET INT MODE 0 'IM 0'	ED 46	8080A MODE
SET INT MODE 1 'IM 1'	ED 56	RESTART TO LOCATION 0038_H
SET INT MODE 2 'IM 2'	ED 5E	INDIRECT CALL USING REGISTER I AND 8 BITS FROM INTERRUPTING DEVICE AS A POINTER.

GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands	↕	↕	X	↕	X	P	*	↕	00 100 111	27	1	1	4	Decimal adjust accumulator
CPL	$A \leftarrow \overline{A}$	*	*	X	1	X	*	1	*	00 101 111	2F	1	1	4	Complement accumulator (One's complement)
NEG	$A \leftarrow \overline{A} + 1$	↕	↕	X	↕	X	V	1	↕	11 101 101 01 000 100	ED 44	2	2	8	Negate acc. (two's complement)
CCF	$CY \leftarrow \overline{CY}$	*	*	X	X	X	*	0	↕	00 111 111	3F	1	1	4	Complement carry flag
SCF	CY ← 1	*	*	X	0	X	*	0	1	00 110 111	37	1	1	4	Set carry flag
NOP	No operation	*	*	X	*	X	*	*	*	00 000 000	00	1	1	4	
HALT	CPU halted	*	*	X	*	X	*	*	*	01 110 110	76	1	1	4	
DI	IFF ← 0	*	*	X	*	X	*	*	*	11 110 011	F3	1	1	4	
EI	IFF ← 1	*	*	X	*	X	*	*	*	11 111 011	FB	1	1	4	
IM 0	Set interrupt mode 0	*	*	X	*	X	*	*	*	11 101 101 01 000 110	ED 46	2	2	8	
IM 1	Set interrupt mode 1	*	*	X	*	X	*	*	*	11 101 101 01 010 110	ED 56	2	2	8	
IM 2	Set interrupt mode 2	*	*	X	*	X	*	*	*	11 101 101 01 011 110	ED 5E	2	2	8	

Notes: IFF indicates the interrupt enable flip-flop
CY indicates the carry flip-flop.

Flag Notation: * = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.
* = Interrupts are not sampled at the end of EI or DI

16-BIT ARITHMETIC

DESTINATION		SOURCE	BC	DE	HL	SP	IX	IY
'ADD'		HL	09	19	29	39		
		IX	DD 09	DD 19		DD 39	DD 29	
		IY	FD 09	FD 19		FD 39		FD 29
ADD WITH CARRY AND SET FLAGS 'ADC'		HL	ED 4A	ED 5A	ED 6A	ED 7A		
SUB WITH CARRY AND SET FLAGS 'SBC'		HL	ED 42	ED 52	ED 62	ED 72		
INCREMENT 'INC'			03	13	23	33	DD 23	FD 23
DECREMENT 'DEC'			0B	1B	2B	3B	DD 2B	FD 2B

16-BIT ARITHMETIC GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD HL, ss	HL ← HL+ss	•	•	X	X	X	•	0	↕	00 ss1 001		1	3	11	ss Reg.
															00 BC
ADC HL, ss	HL ← HL+ss+CY	↕	↕	X	X	X	V	0	↕	11 101 101	ED	2	4	15	01 DE
										01 ss1 010					10 HL
															11 SP
SBC HL, ss	HL ← HL-ss-CY	↕	↕	X	X	X	V	1	↕	11 101 101	ED	2	4	15	
										01 ss0 010					
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	↕	11 011 101	DD	2	4	15	pp Reg.
										00 pp1 001					00 BC
															01 DE
															10 IX
															11 SP
ADD IY, rr	IY ← IY + rr	•	•	X	X	X	•	0	↕	11 111 101	FD	2	4	15	rr Reg.
										00 rr1 001					00 BC
															01 DE
															10 IY
															11 SP
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00 ss0 011		1	1	6	
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10	
										00 100 011	23				
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10	
										00 100 011	23				
DEC ss	ss ← ss - 1	•	•	X	•	X	•	•	•	00 ss1 011		1	1	6	
DEC IX	IX ← IX - 1	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10	
										00 101 011	2B				
DEC IY	IY ← IY - 1	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10	
										00 101 011	2B				

Notes: ss is any of the register pairs BC, DE, HL, SP
pp is any of the register pairs BC, DE, IX, SP
rr is any of the register pairs BC, DE, IY, SP.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

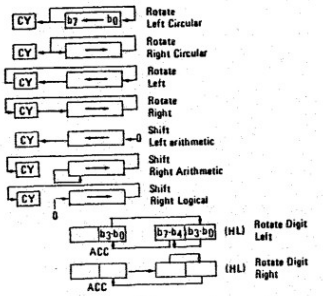
ROTATES AND SHIFTS

Source and Destination

TYPE OF ROTATE OR SHIFT

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
'RLC'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
'RLD'								ED 6F		
'RRD'								ED 67		

	A
'RLCA'	07
'RRCA'	0F
'RLA'	17
'RRA'	1F



ROTATE AND SHIFT GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLCA		•	•	X	0	X	•	0	‡	00 000 111	07	1	1	4	Rotate left circular accumulator
RLA		•	•	X	0	X	•	0	‡	00 010 111	17	1	1	4	Rotate left accumulator
RRCA		•	•	X	0	X	•	0	‡	00 001 111	0F	1	1	4	Rotate right circular accumulator
RRA		•	•	X	0	X	•	0	‡	00 011 111	1F	1	1	4	Rotate right accumulator
RLC r		‡	‡	X	0	X	P	0	‡	11 001 011 00 000 r	CB	2	2	8	Rotate left circular register r r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (HL)		‡	‡	X	0	X	P	0	‡	11 001 011 00 000 110	CB	2	4	15	
RLC (IX+d)	 r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	11 011 101 11 001 011 – d – 00 000 110	DD CB	4	6	23	
RLC (IY+d)		‡	‡	X	0	X	P	0	‡	11 111 101 11 001 011 – d – 00 000 110	FD CB	4	6	23	
RL s	 s≡r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	010					Instruction format and states are as shown for RLC's. To form new Op-Code replace 000 of RLC's with shown code
RRC s	 s≡r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	001					
RR s	 s≡r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	011					
SLA s	 s≡r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	100					
SRA s	s≡r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	101					
SRL s	s≡r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	111					
RLD		‡	‡	X	0	X	P	0	•	11 101 101 01 101 111	ED 6F	2	5	18	Rotate digit left and right between the accumulator and location (HL). The content of the upper half of the accumulator is unaffected
RRD		‡	‡	X	0	X	P	0	•	11 101 101 01 100 111	ED 67	2	5	18	

Flag Notation: • = flag not effected, 0 = flag reset, 1 = flag set, X = flag is unknown, ‡ = flag is affected according to the result of the operation.

BIT MANIPULATION GROUP

	BIT	REGISTER ADDRESSING A	B	C	D	E	H	L	REG. INDIR. (HL)	INDEXED (IX+d)	(IY+d)
TEST 'BIT'	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

BIT SET, RESET AND TEST GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
BIT b, r	$Z \leftarrow \overline{r_b}$	X	‡	X	1	X	X	0	•	11 001 011 01 b r	CB	2	2	8	r: 000 B, 001 C, 010 D, 011 E, 100 H, 101 L, 111 A
BIT b, (HL)	$Z \leftarrow \overline{(HL)_b}$	X	‡	X	1	X	X	0	•	11 001 011 01 b 110	CB	2	3	12	b: 000 Bit 0, 001 1, 010 2, 011 3, 100 4, 101 5, 110 6, 111 7
BIT b, (IX+d)$_b$	$Z \leftarrow \overline{(IX+d)_b}$	X	‡	X	1	X	X	0	•	11 011 101 11 001 011 – d – 01 b 110	DD CB	4	5	20	
BIT b, (IY+d)$_b$	$Z \leftarrow \overline{(IY+d)_b}$	X	‡	X	1	X	X	0	•	11 111 101 11 001 011 – d – 01 b 110	FD CB	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011 11 b r	CB	2	2	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011 11 b 110	CB	2	4	15	
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 011 101 11 001 011 – d – 11 b 110	DD CB	4	6	23	
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 111 101 11 001 011 – d – 11 b 110	FD CB	4	6	23	
RES b, s	$s_b \leftarrow 0$ s ≡ r, (HL), (IX+d), (IY+d)	•	•	X	•	X	•	•	•	10					To form new OP-Code replace 11 of SET b, s with 10. Flags and time states for SET instruction

Notes: The notation s_b indicates bit b (0 to 7) of location s.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown, ‡ = flag is affected according to the result of the operation.

JUMP GROUP

			CONDITION: UNCOND	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	DEC B, JNZ
JUMP 'JP'	IMMED. EXT.	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVE	PC ← e	18 e 2	38 e 2	30 e 2	28 e 2	20 e 2					
JUMP 'JP'	REG. INDIR.	(HL)	E9									
JUMP 'JP'		(IX)	DD E9									
JUMP 'JP'		(IY)	FD E9									
DECREMENT B, JUMP IF NON ZERO 'DJNZ'	RELATIVE	PC ← e										10 e 2

JUMP GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	X	•	X	•	•	•	11 000 011 ← n → ← n →	C3	3	3	10	
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	X	•	X	•	•	•	11 cc 010 ← n → ← n →		3	3	10	
JR e	PC ← PC+e	•	•	X	•	X	•	•	•	00 011 000 ← e-2 →	18	2	3	12	
JR C, e	If C = 0, continue	•	•	X	•	X	•	•	•	00 111 000 ← e-2 →	38	2	2	7	If condition not met
	If C = 1, PC ← PC+e											2	3	12	If condition is met
JR NC, e	If C = 1, continue	•	•	X	•	X	•	•	•	00 110 000 ← e-2 →	30	2	2	7	If condition not met
	If C = 0, PC ← PC+e											2	3	12	If condition is met
JR Z, e	If Z = 0 continue	•	•	X	•	X	•	•	•	00 101 000 ← e-2 →	28	2	2	7	If condition not met
	If Z = 1, PC ← PC+e											2	3	12	If condition is met
JR NZ, e	If Z = 1, continue	•	•	X	•	X	•	•	•	00 100 000 ← e-2 →	20	2	2	7	If condition not met
	If Z = 0, PC ← PC+e											2	3	12	If condition is met
JP (HL)	PC ← HL	•	•	X	•	X	•	•	•	11 101 001	E9	1	1	4	
JP (IX)	PC ← IX	•	•	X	•	X	•	•	•	11 011 101 11 101 001	DD E9	2	2	8	
JP (IY)	PC ← IY	•	•	X	•	X	•	•	•	11 111 101 11 101 001	FD E9	2	2	8	
DJNZ, e	B ← B-1 If B = 0, continue	•	•	X	•	X	•	•	•	00 010 000 ← e-2 →	10	2	2	8	If B = 0
	If B ≠ 0, PC ← PC+e											2	3	13	If B ≠ 0

cc	Condition
000	NZ non zero
001	Z zero
010	NC non carry
011	C carry
100	PO parity odd
101	PE parity even
110	P sign positive
111	M sign negative

Notes: e represents the extension in the relative addressing mode.

e is a signed two's complement number in the range <-126, 129>

e-2 in the op-code provides an effective address of pc+e as PC is incremented by 2 prior to the addition of e.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown, ‡ = flag is affected according to the result of the operation.

CALL AND RETURN GROUP

			CONDITION									
			UN-COND.	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG.	SIGN POS.	REG. B ≠ 0
'CALL'	IMMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
RETURN 'RET'	REGISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETURN FROM INT 'RETI'	REGISTER INDIR.	(SP) (SP+1)	ED 4D									
RETURN FROM NON MASKABLE INT 'RETN'	REGISTER INDIR.	(SP) (SP+1)	ED 45									

NOTE -- CERTAIN FLAGS HAVE MORE THAN ONE PURPOSE. REFER TO Z80-CPU TECHNICAL MANUAL FOR DETAILS.

RESTART GROUP

		OP CODE	
CALL ADDRESS	0000_H	C7	'RST 0'
	0008_H	CF	'RST 8'
	0010_H	D7	'RST 16'
	0018_H	DF	'RST 24'
	0020_H	E7	'RST 32'
	0028_H	EF	'RST 40'
	0030_H	F7	'RST 48'
	0038_H	FF	'RST 56'

CALL AND RETURN GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $PC \leftarrow nn$	•	•	X	•	X	•	•	•	11 001 101 – n – – n –	CD	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	X	•	X	•	•	•	11 cc 100 – n – – n –		3	3	10	If cc is false
												3	5	17	If cc is true
RET	$PC_L \leftarrow (SP)$ $PC_H \leftarrow (SP+1)$	•	•	X	•	X	•	•	•	11 001 001	C9	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	X	•	X	•	•	•	11 cc 000		1	1	5	If cc is false
												1	3	11	If cc is true
RETI	Return from interrupt	•	•	X	•	X	•	•	•	11 101 101 01 001 101	ED 4D	2	4	14	
RETN[1]	Return from non maskable interrupt	•	•	X	•	X	•	•	•	11 101 101 01 000 101	ED 45	2	4	14	
RST p	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $PC_H \leftarrow 0$ $PC_L \leftarrow p$	•	•	X	•	X	•	•	•	11 t 111		1	3	11	

cc	Condition	
000	NZ	non zero
001	Z	zero
010	NC	non carry
011	C	carry
100	PO	parity odd
101	PE	parity even
110	P	sign positive
111	M	sign negative

t	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

[1] RETN loads $IFF_2 \rightarrow IFF_1$

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

INPUT GROUP

INPUT DESTINATION			PORT ADDRESS IMMED. n	PORT ADDRESS REG. INDIR. (C)
INPUT 'IN'	REG. ADDRESSING	A	DB n	ED 78
		B		ED 40
		C		ED 48
		D		ED 50
		E		ED 58
		H		ED 60
		L		ED 68
'INI' – INPUT & Inc HL, Dec B	REG. INDIR.	(HL)		ED A2
'INIR' – INP, Inc HL, Dec B, REPEAT IF B≠0				ED B2
'IND' – INPUT & Dec HL, Dec B				ED AA
'INDR'–INPUT, Dec HL Dec B, REPEAT IF B≠0				ED BA

BLOCK INPUT COMMANDS

OUTPUT GROUP

			SOURCE REGISTER A	B	C	D	E	H	L	REG. IND. (HL)
'OUT'	IMMED.	n	03 n							
	REG. IND.	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69	
'OUTI' – OUTPUT Inc HL, Dec b	REG. IND.	(C)								ED A3
'OTIR' – OUTPUT, Inc HL, Dec B, REPEAT IF B≠0	REG. IND.	(C)								ED B3
'OUTD' – OUTPUT Dec HL Dec B	REG. IND.	(C)								ED AB
'OTDR'–OUTPUT, Dec HL Dec B, REPEAT IF B ≠ 0	REG. IND.	(C)								ED BB

PORT DESTINATION ADDRESS

BLOCK OUTPUT COMMANDS

INPUT AND OUTPUT GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
IN A, (n)	A ← (n)	•	•	X	•	X	•	•	•	11 011 011 – n –	DB	2	3	11	n to A_0 ~ A_7 Acc to A_8 ~ A_{15}
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	↕	↕	X	↕	X	P	0	•	11 101 101 01 r 000	ED	2	3	12	C to A_0 ~ A_7 B to A_8 ~ A_{15}
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	X	↕ ①	X	X	X	X	1	•	11 101 101 10 100 010	ED A2	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	•	11 101 101 10 110 010	ED B2	2 2	5 4 (If B ≠ 0)	21 16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	X	↕ ①	X	X	X	X	1	•	11 101 101 10 101 010	ED AA	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	•	11 101 101 10 111 010	ED BA	2 2	5 4 (If B ≠ 0)	21 16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OUT (n), A	(n) ← A	•	•	X	•	X	•	•	•	11 010 011 – n –	D3	2	3	11	n to A_0 ~ A_7 Acc to A_8 ~ A_{15}
OUT (C), r	(C) ← r	•	•	X	•	X	•	•	•	11 101 101 01 r 001	ED	2	3	12	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	X	↕ ①	X	X	X	X	1	•	11 101 101 10 100 011	ED A3	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OTIR	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	•	11 101 101 10 110 011	ED B3	2 2	5 4 (If B ≠ 0)	21 16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	X	↕ ①	X	X	X	X	1	•	11 101 101 10 101 011	ED AB	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	•	11 101 101 10 111 011	ED BB	2 2	5 4 (If B ≠ 0)	21 16	C to A_0 ~ A_7 B to A_8 ~ A_{15}

Notes: ① If the result of B - 1 is zero the Z flag is set, otherwise it is reset.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown, ↕ = flag is affected according to the result of the operation.

Z80 – CPU INTERRUPT STRUCTURE

MASKABLE ($\overline{INT}$)

Mode 0

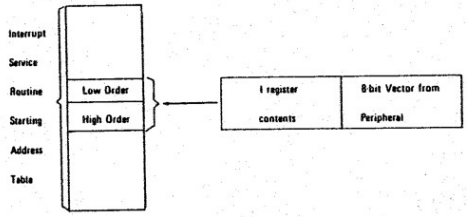
Place instruction onto Data Bus during $\overline{INTA}$ = $\overline{M1}$ • $\overline{IORQ}$ like 8080A

Mode 1

Restart to 38_H or 56_{10} ('RST 56')

Mode 2

Used by Z80 Peripherals



NON MASKABLE ($\overline{NMI}$)

Restart to 66_H or 102_{10}

INTERRUPT ENABLE/DISABLE FLIP-FLOPS

Action	IFF_1	IFF_2	
CPU Reset	0	0	
DI	0	0	
EI	1	1	
LD A, I	•	•	IFF_2 ← Parity flag
LD A, R	•	•	IFF_2 ← Parity flag
Accept $\overline{NMI}$	0	•	
RETN	IFF_2	•	IFF_2 ← IFF_1
Accept $\overline{INT}$	0	0	
RETI	•	•	

"•" indicates no change

PIO PROGRAMMING SUMMARY

REGISTER SELECTION

SELECT LINES C/D	B/A	REGISTER SELECTED
0	0	A Data
0	1	B Data
1	0	A Control
1	1	B Control

LOAD INTERRUPT VECTOR

D7							D0	
V7	V6	V5	V4	V3	V2	V1	0	Control Register

SET OPERATING MODE

D7							D0	
M1	M0	X	X	1	1	1	1	Control Register

Mode Number	M1	M0	Mode
0	0	0	Output
1	0	1	Input
2	1	0	Bidirectional
3	1	1	Bit Control

If Mode 3 selected, the next control word to the PIO is

D7							D0	
I/O_7	I/O_6	I/O_5	I/O_4	I/O_3	I/O_2	I/O_1	I/O_0	Control Register

I/O = 1 Sets bit to Input

I/O = 0 Sets bit to Output

SET INTERRUPT CONTROL

D7							D0	
Int Enable	AND/ OR	High/ Low	Mask Follows	0	1	1	1	Control Register

In Mode 3 if Mask follows = 1, the next control word to the PIO is

D7							D0	
MB_7	MB_6	MB_5	MB_4	MB_3	MB_2	MB_1	MB_0	Control Register

MB = 0 Monitor the bit

MB = 1 Mask the bit

ENABLE / DISABLE INTERRUPTS

D7							D0	
Int Enable	X	X	X	0	0	1	1	Control Register