
MAXAM
Z 80 Entwicklungs
System
Assembler
Monitor
Text Editor
CPC 464/664/6128

MAXAM ASSEMBLER

Inhaltsuebersicht:

Nehmen Sie Kontakt mit ARNOR auf um Einzelheiten zu erfahren, wie man ROM - oder Diskversionen aufruesten kann. Sie koennen Ihre jetzige Version an ARNOR zurueckschicken,jede Korrespondenz die sich auf ARNOR - Produkte bezieht ist willkommen. Bei speziellen Anfragen bitte die Referenznummer angeben. Assembler u. Editor haben jeweils eine Versionnummer welche nach dem Laden erscheint, wenn Sie Help eintippen.

Diese Uebersetzung erfolgte mit freundlicher Genehmigung der ARNOR L T D durch RALF PROBST EDV-SERVICE.

Duisburg, Oktober 1985

Publiziert durch Ralf Probst EDV-Service, Friedrich Ebert Str. 14
4100 Duisburg 17

All that is gold does not glitter,
Not all those who wander are lost;
The old that is strong does not wither,
Deep roots are not reached by the frost.

From the ashes a fire shall be woken,
A light from the shadows shall spring;
Renewed shall be blade that was broken,
The crownless again shall be king.

J.R.R. Tolkien

Anmerkung: Wir halten es nicht fuer sinnvoll, diese Verse durch eine Uebersetzung zu verstuemmeln.
Vielleicht gelingt es Ihnen aber mit etwas Muehe diese Zeilen zu interpretieren.

2. Die ersten Dinge zuerst

ROM Version

Die ROM Version wird in einem 16K EPROM mit Adapter geliefert. Der Adapter wird an der Kantenverbindung, mit 'FLOPPY DISC' markiert, auf der Rueckseite des Computers befestigt,und besitzt einen Anschluss fuer ein Diskettenlaufwerk.Bevor Sie versuchen den Adapter einzubauen,lesen Sie bitte die Einbauinstruktionen.

Wenn MAXAM installiert ist,ist er sofort Arbeitsbereit fuer den Benutzer,da automatisch initialisiert ist durch Basic.MAXAM benutzt gerade 256 bytes vom RAM (Speicherplatz fuer Hintergrund-ROM).Sie werden merken ,dass einige Programme nicht laufen werden,da sie diesen Teil des Speichers benoetigen.Wenn dies passiert tippen Sie das Kommando 'öMAXOFF' ein.Dies bewirkt,dass ein Reset erfolgt und der Rechner so arbeitet als waere MAXAM nicht installiert.

Disketten Version

Fuehren Sie zuerst einen Reset durch indem Sie CTRL-SHIFT-ESC eingeben.Dies ist wichtig.Legen Sie die MAXAM Diskette ein und tippen Sie 'RUN "MAXAM" '.
Dann haben Sie die Wahl

(a) das komplette Programm
(b) nur den Editor
(c) alles ausser dem Editor zu laden.

Wenn geladen,so wird das Programm in der hoechstmoeglichen Speicheradresse abgelegt.Es wird nur soviel Speicherplatz belegt wie erforderlich ist, wenn man z.B. nur den Editor laedt,so bleibt mehr Speicherplatz frei fuer den Text.
Schliesslich werden Sie gefragt:"Wieviele bytes moechten Sie reservieren fuer M-Code?"
Als Antwort koennen Sie einfach 'ENTER' druecken,in welchem Fall nichts weiter geschieht. Gibt man eine Zahl ein, so Wird 'HIMEM' um diesen Wert heruntergesetzt um Speicherplatz fuer M-Code zu reservieren (dies wird erklaert in Sektion 5).

Cassetten Version

Fuehren Sie einen Reset durch,indem Sie CTRL-SHIFT-ESC eingeben.Dies ist wichtig.Zwei Kopien werden geliefert,eine auf jeder Seite des Bandes. Um das Programm zu laden,tippen Sie bitte:

'RUN "MAXAM"
oder 'RUN "

Nun werden Sie gefragt: "Moechten Sie den Editor laden ?"
Wenn Sie den Editor nicht benoetigen so antworten Sie mit 'N'.Dadurch haben Sie mehr freien Speicherplatz zur Verfuegung. Schliesslich werden Sie gefragt: "Wieviele bytes moechten Sie reservieren fuer M-CODE ?" Als Antwort koennen Sie einfach 'ENTER' eingeben,in welchem Fall nichts weiter geschieht.Gibt man eine Zahl ein, so wird 'HIMEM' um diesen Wert heruntergesetzt,um Speicherplatz fuer M-Code zu reservieren (dies wird erklaert in Sektion 5).

MAXAM ist nun installiert, egal welche Version des Programms Sie haben.Der verbleibende Teil dieses Handbuches wird zu gleichen Teilen auf die ROM Version und die Disketten/Cassetten Version angewandt.

3. Wie man Maxam gebraucht

Wenn Maxam installiert ist, kommen eine Anzahl von Befehlen zum System hinzu. Diese sind als externe Kommandos bekannt, und werden von Basic abgerufen, indem man vor dem Befehl einen vertikalen Strich davorsetzt (Shift + Klammeraffe). Der vollstaendigkeithalber sind alle diese Befehle hier gelistet, obgleich sie noch mal ausfuehrlich in der relevanten Sektion behandelt werden.

Kommandos, welche in allen Versionen verfuegbar sind

MAXAM	bringt Sie im das Hauptmenue, von wo aus der Texteditor und die Utility Kommandos verfuegbar sind.
M	das Gleiche wie MAXAM
MAXAM,2	Hauptmenue in MODE 2
ASSAMBLE	assembliert Z80 Code in Basic Programmen
ASSEM	wie ASSAMBLE, alle Meldungen werden unterdrueckt
CAT	katalogisiert Files in der gleichen Weise wie der BASIC-Befehl
CLEAR	Loeschen des Editor Textes.
FIND	sucht nach einem String im Assembler Teil eines BASIC-Programms
MODE	SCHALTET ZWISCHEN MODE 1 UND MODE 2
SPEED	legt die Schreibgeschwindigkeit des Datenrecorders fest

Kommandos, die nur auf ROM Versionen verfuegbar sind

HELP	listet alle ROMs mit den Versions Nummern
HELP,n	listet die Befehle mit der ROM Nummer n
MAXOFF	schaltet MAXAM ab
MSL	legt den Bildschirmspeicherbereich tiefer
MSH	legt den Bildschirmspeicherbereich hoeher
ROMOFF	schaltet die ausgewaehlten ROMs aus

Beachte: Man kann vor Clear, FIND und HELP ein 'M' stellen (fuer MAXAM). Das ist fuer den Fall, dass zukuenftige ROMs diese Befehle benutzen und den Befehl vor MAXAM abfangen.
Wir hoffen jedenfalls, dass die Schreiber von ROM-Software diese Befehle vermeiden werden.

Einige allgemeine Punkte

1. Durch das ganze System von MAXAM funktioniert die ESC Taste in der gleichen Weise wie im Basic, das heisst, dass jede Operation gestoppt werden kann, indem man einmal ESC drueckt. Wenn Sie ESC nocheinmal druecken, wird die Operation abgebrochen, jede andere Taste wird verursachen, dass die Funktion wieder aufgenommen wird. Wenn MAXAM versucht ein Zeichen zum Drucker zu senden, und der Drucker ist nicht betriebsbereit, wird der Cursor eingeschaltet. Entweder stellen Sie den Drucker auf ON LINE oder druecken die ESC Taste. (Moeglicherweise muessen Sie ESC fuer eine halbe Sekunde druecken)
2. Die DEL Taste arbeitet in gleicher Weise wie in Basic, d. h. loescht sie den letzten getippten Buchstaben.
3. Der Cursor ist nicht funktionsfaehig, wenn irgendeine Operation durchgefuehrt wird. Immer wenn der Cursor sichtbar ist, wartet der Computer auf eine Eingabe. Wenn kein 'prompt' angezeigt wird, wartet der Computer auf die Eingabe irgendeiner Taste um fortzufahren oder auf ESC, um die Operation abzubrechen. Dieses wiederum ist gleich wie in Basic.
4. MAXAM besitzt im 40 Zeichen MODE eine farbige Darstellung. Sie haben die Moeglichkeit schlecht sichtbare Farben (z.B. rot bei Gruenmonitor) zu aendern. So koennen Sie z. B. rot in hellgruen durch die Eingabe von 'INK 3,18' von BASIC aus aendern.

4. Lernen Sie den Assembler kennen

Das Ziel dieses Kapitels ist es eine leichte Einfuehrung zur Benutzung des Assemblers zu geben. Es gibt zwei komplette kleine Programme mit Erklaerungen, was sie tun. Es ist empfehlenswert, diese Programme einzugeben, besonders, wenn Sie vorher noch nie einen Assembler benutzt haben. Ein oder zwei Einzelheiten sind hier nur angedeutet, werden aber in Kapitel 5 genauer erlaeutert.

Es gibt ein paar Punkte, die man sich merken muss, wenn man diese Programme eingibt.

(i) Alles kann entweder in Gross- oder Kleinschrift eingegeben werden - es ist kein Unterschied.

(ii) Achten Sie darauf, dass das Apostroph nach der Zeilennummer erscheint. Dieses Zeichen erhalten Sie, wenn Sie Shift und 7 druecken.

Programm 1 stellt den ASCII Zeichensatz dar. Das aequivalente Basic Programm ist gegeben.

Programm 1 (a) M-code

```
10    MEMORY HIMEM-11
20    start=HIMEM+1
30    GOSUB 1000
40    CALL start
50    END
1000  |ASSEMBLE
1010 'LD A,32          ; erster ASCII Code in Akkumulator
1020 '.loop            ;definieren der Schleife
1030 'CALL &BB5A       ;CALL txt output,Firmwareausgabe-Routine
1040 'INC A            ;gehe zum naechsten Zeichen
1050 'CP 128           ;haben wir alle Zeichen?
1060 'JR C,loop          ;nein, springe zurueck fuer ein weiteres
1070 'RET                ;ja-return
1080 'END              Assembler abbrechen
1090 'RETURN
```

Programm 1 (b): BASIC

```
10 FOR I%=32 TO 127
20 PRINT CHR$(I%);
30 NEXT
```

Zeilenweise Erklaerung des Programms 1(a)

10 Dies reserviert Speicherplatz fuer die Maschinen-Code Routine, in diesem Fall werden 11 Bytes benoetigt.

20 Die Basic Variable 'Start' enthaelt die Startadresse der Maschinen Code Routine.

30 Die Unterroutine assembliert den Code. Es ist nuetzlich,die Assembleranweisungen in einer separaten Subroutine unterzubringen.

40 Ruft das (nun assemblierte) Maschinen Code Programm ab.

1000 Ein externer Befehl, der den Assembler aufruft und welcher die Assembler-Befehle in den folgenden Zeilen assembliert.

1010 Das Maschinen Code Programm. Die quotierten Zeichen teilen Basic mit, dass diese Zeilen nur Bemerkungen sind, deshalb werden sie ignoriert. Die Semikolons teilen dem Assembler mit, dass das Folgende eine Bemerkung ist.

1090 Der Assembler uebergibt die Kontrollfunktion wieder dem Basic, welches aus der Unterroutine zurueckkehrt.

Wenn Sie dieses Programm benutzen, werden Sie 2 Dinge bemerken. Erstens, wenn der Assembler aufgerufen wird,stellt er sich selbst vor, indem es eine Titelzeile mit der Assembler Versions-Nummer anzeigt. Zweitens, nach Beendigung des Assemblierens wird die Anzahl der Fehler und Warnungen angezeigt.
Eine Warnung ist kein Fehler, sondern nur ein Hinweis des Assemblers, dass Sie sich verschrieben haben koennten. Alle Fehler die gemacht wurden, werden in einfachem Englisch erklaert. Sie muessen nie einen Fehlercode nachschlagen.

Programm 2 liest ein Zeichen von der gegenwaertigen Cursor-Position auf dem Bildschirm, und gibt seinen ASCII-Code in einer Integer-Variablen zurueck. Die Koordinaten, die erforderlich sind, sind dieselben, die vom dem Basic LOCATE Befehl gebraucht werden. Wenn kein erkennbares Zeichen gegeben wird, wird null ausgegeben.

Programm 2

```
  10 MEMORY HIMEM-50
  20 GOSUB 1000
  30 rdchar=HIMEM+1
  40 char%=0
  50 INPUT "Eingabe Bildschirmposition:",x,y
  60 oldx=POS (#0):oldy=VPOS(#0)
  70 LOCATE oldx,oldy
  80 CALL rdchar,@char%
  90 LOCATE oldx,oldy
 100 PRINT
 110   PRINT   "Der   ASCII    des    Zeichens    ist";char%;"dieser
       ist";CHR$(char%)
 120 END
1000 |ASSEMBLE
1010 'DECA:RET NZ;                  return wenn kein Parameter
1020 'ld l,(ix);                    get low byte von @char%
1030 'ld h,(ix+1);                  get high byte von @char%
1040 'call &bb60 ;                  Firmwareroutine TXT RDCHAR
1050 'ld (hl),a  ;                  setze Zeichen in char% ein
1060 'inc hl:ld (hl),0              ;zero in high byte von char%
1070 'ret
1080 'end
1090 'RETURN
```

Der Assembler - im Detail

Programm - Format

Das Programm, das zusammenzustellen ist, besteht aus einer Reihe von Angaben. Jede Angabe hat das Format:
‹Kennzeichnungsfeld› ‹Instruktionsfeld› ‹Erlaeterungsfeld›
Eine Zeile kann verschiedene Angaben enthalten, die durch Kommas getrennt werden, genauso wie in Basic. Eine Angabe besteht aus drei Teilen, wie oben beschrieben, jedes Teil kann aber leer sein. Das Erlaeuterungsfeld, falls vorhanden, muss mit einem Semikolon anfangen - der Effekt eines Semikolons ist, dass der Assembler alle Zeichen ignorieren wird bis zum naechsten Komma oder bis zum Ende der Zeile. Um Kommas setzen zu koennen, muessen Sie vor und hinter dem Komma Semikolons setzen. Alle Zeichen bis zum Ende dieser Zeile werden dann ignoriert.

Instruktionen und Namen, die im Quellcode gebraucht werden, koennen in Gross- oder Kleinschreibweise eingegeben werden. Deshalb werden 'START', 'Start' und 'start' gleichbehandelt. Vor dem Kennzeichen kann ein Punkt gesetzt werden. Dies sagt dem Assembler, dass das folgende ein Kennzeichen ist.
wenn kein Punkt gesetzt wird, wird der Assembler versuchen eine Z80 Instruktion oder eine Assembler-Order zu erkennen. Sollte dies fehlschlagen, wird er das erste Zeichen als ein Kennzeichen behandeln. Kennzeichen muessen mit einem Buchstaben beginnen, und koennen von jeder Laenge sein.
Aus diesem Grund muss auch vor eine Z80 Instruktion,die als Kennzeichen genutzt werden soll, unbedingt ein Punkt gesetzt werden, dies bedeutet '.halt'.

Eine Warnung wird gegeben wenn ein Kennzeichen, dem kein Punkt vorgestellt ist mit dem Namen eines 'mnenomics' oder einer Order anfaengt. Beispielsweise wenn Sie durch einen Fehler ein Leerzeichen auslassen (z.B. bei der Eingabe von 'inca' statt 'inc a) wird der Assembler Sie warnen. Durch die Eingabe eines Punktes wird die Warnung verhindert, wenn Sie beabsichtigen ein Kennzeichen anzusagen.

Wie man den Assembler von Basic aus nutzt

MAXAM kann zweifach angewandt werden; entweder zur Eingabe von Z80 Quellcode als Teil eines Basic-Programms, oder indem man den Full Screen Editor benutzt. Der Editor wird in Kapitel 8 behandelt.
Durch das Kommando 'ASSEMBLE' kann innerhalb eines Basic-Programms der Assembler aufgerufen werden. Der Assembler liesst, an dieser Stelle beginnend, die Z80 Instruktionen des Basic-Programms. Diese Zeilen koennen nicht direkt eingegeben werden, da Basic versuchen wuerde diese Zeile in Tokens umzuwandeln. Aus diesem Grund muss jede Zeile des Z80 Code mit einem Hochkomma beginnen.
z.B. 100 'LD A,10
Der Quellcode kann in der gleichen Zeile eingegeben werden wie öASSEMBLE: z.B. öASSEMBLE:'org &b55d:rst 1,&87F2
Das kann auch im Direkt Mode ausgefuehrt werden.
Das Hochkomma weist Basic an, diese Zeile als Kommentar aufzufassen und zu ignorieren. Am Ende eines Maschinen-Code Abschnitts - welches automatisch vom Assembler entdeckt wird wenn er eine Zeile ohne Hochkomma vorfindet - nimmt Basic an der Zeile, die öASSEMBLE folgt, automatisch die Ausfuehrung wieder auf. Der Assembler-Code wird nun uebergangen, und Basiczeilen weden abgearbeitet.
Ein Programm kann beliebig viele Abschnitte mit Assembler-Code beinhalten, wobei jeder mit öASSEMBLE beginnen muss.
Es ist ratsam jeden Assembler-Abschnitt in einer separaten Subroutine abzulegen, wie auch in den Beispielprogrammen des Handbuches.

Das Korrigieren von Fehlern

Das Programm kann in der gleichen Art und Weise korrigiert werden, wie jedes BASIC-Programm. Sollten Sie z.B. in einem Assembler-Abschnitt ein Hochkomma vergessen, so wird der Assembler abbrechen, und BASIC wird versuchen die Zeile abzuarbeiten. Dies wird natuerlich fast immer einen SYNTAX ERROR verursachen; BASIC wird auf dem Zeileneditor erscheinen und erlaubt eine einfache Korrektur dieser Zeile.

Der !FIND Befehl

Der Find Befehl wurde in das Programm aufgenommen, um das Editieren eines Programmes zu beschleunigen. Wenn Sie '!FIND' eingeben, werden Sie nach dem String gefragt, nach welchem Sie suchen moechten. Alles was diesen STRING betrifft und sich in einer Zeile befindet die mit einem Hochkomma beginnt, wird nun gefunden und die zugehoerige Zeilennummer angezeigt.
Dieser Befehl wurde geschaffen, um bestimmte Teile eines Assemblercodes zu finden; er funktioniert nicht in Basic, wegen der ungewoehnlichen Art wie BASIC seine Zeilen in TOKENS umwandelt.
Die groesste Zeichenkettenlaenge besteht aus 17 Zeichen.
Find gibt es nicht in der Cassetten Version.

Debugging und Breakpoints

Das Debugging von Maschinenprogrammen wird durch die Benutzung von MAXAM sehr stark vereinfacht, es kann von BASIC aus geschehen - es ist nicht erforderlich,dass Sie eine Vielzahl komplizierter Befehle erlernen.
Dies wird erreicht, indem es moeglich ist, ueberall im Programm Breakpoints zu setzen. ein Breakpoint ist eine spezielle Anweisung, die einen Sprung zu einer Unterroutine verursacht, welche die Werte des Z80 Registers anzeigt und die naechste durchzufuehrende Instruktion zerlegt. Indem man die Werte ueberprueft die sich in den Registern befinden, kann man sehen ob das Programm einwandfrei arbeitet oder nicht. Falls nicht, so drueckt man ESC um ins BASIC zurueckzukehren, prueft das Programm, macht die notwendigen Korrekturen, assembliert und macht einen erneuten Versuch.
Wenn die Register in Ordnung sind druecken Sie eine Taste, und das Programm wird fortfahren bis zum naechsten Breakpoint oder der abschliessenden RET Anweisung.
Das Mnenomic fuer einen Breakpoint ist 'BRK'.
Tatsaechlich ist BRK das gleiche wie RST 6, was bedeutet, dass wenn Sie aus irgendeinem Grund mit RST 6 unterbrechen, eine BRK Anweisung anstelle der MAXAM Breakpoint-Routine Ihre Routine zum Fortfahren veranlasst.Machen Sie einen Reset, um die MAXAM Routine wiederherzustellen.
Schauen Sie sich nocheinmal Programm 1(a) an und fuegen die folgende Zeile ein:

```
1025  '  BRK
```

Jetzt werden,wenn Sie das Programm laufen lassen,bei jedem Durchlauf der Schleife die Register angezeigt.

Hybrid BASIC/Maschinencode Programm

Wird waehrend des Assemblierens ESC gedrueckt, wird Basic (egal ob Sie das Assemblieren durch nochmaliges druecken von ESC unterbrechen oder nicht) bis zur Vervollstaendigung des Assemblierens warten, als sei ESC nur einmal gedrueckt worden.
Demnach muessen Sie um das Programm zu unterbrechen, 3 mal ESC druecken.

Der !ASSEM Befehl

Wenn ein Proramm voellig fehlerfrei gemacht wurde, koennte es wuenschenswert sein den Code zu assemblieren, ohne das Meldungen vom Assembler erfolgen. Dies kann man erreichen, indem man !ASSEMBLE gegen !ASSEM austauscht. Der Effekt ist, dass alles ausser den Fehlermeldungen unterdrueckt wird.

Assemmbler Errormeldungen

Immer wenn ein Fehler beim Assemblieren auftritt, wird die fehlerhafte Zeile gelistet, Sie hoeren einen Ton und eine selbsterklaerende Fehlermeldung wird angezeigt.
Es gibt 3 Grade von Fehlern, die beim Assembler auftreten koennen. Der folgenschwerste ist der 'fatal error', welcher dazu fuehrt,dass der Assembler sofort abbricht. Es gibt 10 Arten eines 'fatal error', welche alle in der Referenzliste aufgefuehrt sind.
Mit einer Ausnahme (code limit exceeded) erfolgen 'fatal errors' beim ersten Durchlauf. Alle anderen Fehler treten beim zweiten Durchlauf auf, und fuehren nicht zum Abbruch des Assemblers.

Am wenigsten folgenschwer ist 'warning'. Eine Warnung wird ausgegeben, wenn der Code zwar assembliert werden kann, der Programmierer aber hoechstwahrscheinlich etwas anderes gemeint hat. Das passiert in den folgenden Faellen:

1. Ein'Label', welches keinen vorgestellten Punkt besitzt, beginnt mit dem Namen eines Z80 Mnenomics, einer Assembler-Directive oder Kommandos,z.B INPUT.

2.Ein Ausdruck besitzt mehr als 8 bits, wenn ein 8 bit Wert gefordert ist, z.B. LD A ,300. Achtung: Eine Warnung wird nicht gegeben, wenn die 8 bits alle 1 sind, so ist z.B. LD C,-3 erlaubt.

3.'Unechter Text wird gefunden, nachdem ein Statement korrekt assembliert wurde.Dies kann das Ergebnis eines vergessenen Hochkommas oder Semikolons sein.

Alle anderen Fehler sind mit 'error' gekennzeichnet. Sollte sich ein Fehler ereignen, so sollten Sie vor dem Aufruf des Maschinen-Codes unbedingt 'reassemblieren'. Wenn sich irgendwelche Fehler ereignen oder Warnungen erfolgen, wird das Basic-Programm unterbrochen. So wird es ausgeschlossen, eine fehlerhaft assemblierte Routine aufzurufen.

Wo man den Objekt Code speichert

Wenn man Maschinencodeprogramme schreibt, muss man grosse Sorgfalt dafuer verwenden, den Code in einer Sektion des Speichers zu lagern, die fuer nichts anderes benutzt wird. MAXAM verschafft Ihnen einen bequemen und nuetzlichen Weg dies zu tun. Betrachten Sie dazu die Speicherplatzbelegung des RAM:

Adresse	
0	Firmware - Arbeitsbereich
40	Arbeitsbereich des Hintergrund-ROM
(40)	BASIC Input Buffer
(170)	BASIC Programmbereich
	Basic - Variablen
	*** freier Speicherbereich (vermindert sich,wenn das Programm waechst ***
	BASIC - Strings
HIMEM+1	*** freier Speicherbereich (reserviert durch das Veraendern von HIMEM ***
	Arbeitsbereich von MAXAM und Code (Disketten und Cassettenversion
(A578)	Durch den Anwender definierte Zeichen
(A5f8)	Arbeitsbereich des Hintergrund-ROM, MAXAM und AMSDOS beinhaltend
AC00	BASIC - Arbeitsbereich
B100	Firmware - Arbeitsbereich
C000	Bildschirmspeicher

Adressen, die in Klammern angegeben wurden, sind fuer einen Rechner mit AMSDOS und MAXAM - ROM, aber fuer keine anderen ROMs. Diese sollten nicht in Programmmen verwendet werden.
Die nuetzlichste Adresse auf dieser Karte ist HIMEM. Basic wird nur Speicher unter HIMEM zur Lagerung von Programmen und Variablen benutzen. Der beste Platz, wo man Maschinencodeprogramme lagern kann ist ueber HIMEM. Um dies tun zu koennen verkleinert man HIMEM:
MEMORY HIMEM - 100
Das wird 100 Bytes reservieren, welche nicht vom Basic beruehrt werden.

Anmerkungen:
(i) Es ist oft nuetzlich, die erforderliche Speicherlaenge innerhalb eines Programms zu reservieren. Natuerlich kann die Speicheraussage innerhalb eines Programms enthalten sein, was aber den Nachteil hat, dass bei jedem Programmlauf HIMEM um 100 Bytes reduziert wird. Eine moegliche Loesung:

```
10 Goto 30
20 MEMORY HIMEM - 100
30 REM Rest des Programms
```

Benutzen Sie das erste Mal, wenn das Programm gestartet wird: RUN 20, danach nur RUN.

(ii) Wenn Sie die Symbol After Anweisung benoetigen, muss sie gebraucht werden bevor Sie HIMEM setzen.
Abgesehen von dieser Einschraenkung kann HIMEM jeder Zeit gesetzt werden

Diese Methode der Reservierung von Speicherfuer Objektcode wird sehr empfohlen. Zwei Sachen sind in den Assembler eingebaut, welche den obigen Arbeitsablauf sicher und leicht benutzbar machen. Dies sind :
1.Beim Fehlen einer Anweisung sagt der Assembler wo der Code gespeichert werden soll, und wird sofort automatisch ueber HIMEM gelagert.
2. Der Assembler wird diesen Objektcode ueberpruefen, dass er nicht den Bereich fuer anwenderdefinierte Zeichen ueberschreibt, den Arbeitsbereich des Hintergrund ROM oder (im Fall von Disc- oder Cassettenversion) den MAXAM CODE.
Wenn nicht genug Speicherbereich zur Verfuegung steht das gesamte Maschinencodeprogramm zu lagern, wird die Fehlermeldung 'Code limit exceeded' gegeben. Falls dieses passiert, reduzieren Sie HIMEM und versuchen es noch einmal.

Der obige Mechanismus ist ideal um praktisch alle Programme zu testen, aber machmal muss ein Programm in einem anderen Teil des Speichers ausgefuehrt werden.
Die ORG Anweisung sagt dem Assembler welche Adresse gebraucht wird. Es gibt 2 Formen:

Syntax: 1. ORG ‹Ausdruck›
2. ORG ‹Ausdruck1› , ‹Ausdruck2›

In der ersten Form wird der Assembler den Ausdruck auswerten und benutzt dies als den 'Ursprung'des Code (d.h. die Adresse, womit der Code gestartet wird). Der assemblierte Code wird am Anfang dieser Adresse abgelegt.
Oft ist es nicht moeglich den Code an der Adresse abzulegen, wo er ausgefuehrt werden soll, weil der Raum von etwas anderem benoetigt wird (z.B. MAXAM oder BASIC). In diesem Fall benutzen Sie die zweite Form.Der Assembler wird beiden Anordnungen folgen, die Startadresse an der ersten, aber den Code an der zweiten Adresse (Speicheradresse) ablegen. Dies ist normalerweise nur noetig, wenn ein Programm voellig fehlerfrei ist, und auf bzw. von Diskette oder Cassette abgespeichert oder geladen werden soll.

Anmerkungen
(i) Alle ORG Anweisungen koennen benutzt werden.
(ii) Die Anweisungen duerfen keine undefinierten Zeichen enthalten.

Die LIMIT Anweisung

Wird eine ORG Anweisung benutzt, prueft der Assembler, dass der Code die vom Benutzer definierten Zeichen nicht ueberschreibt. Dies geschieht durch eine interne Variable, die 'LIMIT' genannt wird. Diese wird auf den hoechstmoeglichen Wert des Speicherbereichs gesetzt, in welchem Objekt Code abgelegt werden kann. Zu Anfang wird diese Adresse auf das Byte, welches unterhalb von Basic's Benutzer definierten Zeichensatz liegt, gesetzt. Dies kann aber auch auf jeden anderen Wert gesetzt werden:

Syntax: LIMIT ‹Ausdruck›

Drei Anwendungen der LIMIT Anweisung:

1. Um Speicherbereich gegen Ueberschreiben zu schuetzen
2. Wenn man ein Programm mit festgelegter maximaler Groesse schreibt (z.B. die Groesse eines EPROM)
3. Viele Programme werden die Firmware Routinen unterbrechen (Einzelheiten, wie man so etwas macht, finden Sie in der Amsoft Dokumentation). Die Firmware Jumpblocks liegen im Speicherbereich hoeher als das nicht gesetzte LIMIT,deshalb ist es notwendig die LIMIT Anweisung zu setzen, bevor es moeglich ist direkt im Firmware Jumpblock zu assemblieren. 'LIMIT &FFFF' erlaubt das Assemblieren in jedem Teil des Speichers.

Bemerkungen: LIMIT betrifft nur das Ablegen von Code im Speicher, nicht den Speicherbereich (wenn dies verschieden ist).

Das Pruefen geschieht nur auf Pass 2, da Code auch nur auf Pass 2 gespeichert wird.

Nocode und Code

Syntax: Nocode
Syntax: Code

Gelegentlich ist es nuetzlich ein Programm zusammenzustellen ohne ueberhaupt irgendeinen Code abzulagern - vielleicht nur um zu pruefen,ob es korrekt assembliert ist,oder um eine kleine Routine zu assemblieren, die in Hexadezimal geschrieben wird. (Vielleicht auch auf einem anderen Computer.) Die Anweisung Nocode fuehrt dieses aus. Die Anweisung Code loescht den Effekt vom Nocode und verursacht,dass das Speichern vom Objektcode wieder aufgenommen wird.

Die Anweisung End

Syntax: End

Die End Anweisung sagt dem Assembler einfach, er soll anhalten. Es kann ausgelassen werden, aber es hat zwei Bedeutungen:

1. Um zu vermeiden, dass das ganze Programm assembliert wird - setzt man zeitweise eine End-Anweisung ein.
2. End verursacht, dass der Speicherplatz in dem Listing erscheint. Ein nuetzliches Mittel ist "LIST:END" als letzte Zeile des Quellcodes einzusetzen, so koennen Sie sehen, wo das Ende des Programms ist.

Ausdruecke

Arithmetische Ausdruecke koennen ueberall im Assembler benutzt werden, wo immer eine Zahl benoetigt wird. Dies schliesst Zeichensaetze von Z80 Instruktionen und Assembler Anweisungen ein. Der Ausdrucksbewerter arbeitet von links nach rechts und erlaubt das Folgende:

Zahlen:

1. Ganze Dezimalzahlen, z.B. 132
2. Ganze Hexadezimalzahlen, z.B. &BB5A oder #2A, entweder & oder # koennen zur Kompatibilitaet mit Basic und der Firmware Dokumentation benutzt werden.
3. Ganze Binaerzahlen, z.B. %1011101.
4. Schriftzeichen, z.B. 'A', "3", '"'. Entweder einzelne oder doppelte Anfuehrungszeichen duerfen benutzt werden - um ein Anfuehrungszeichen zu klassifizieren schliesst man es in ein andere Art von Anfuehrungszeichen ein. Der Wert eines Buchstaben ist der ASCII Code Zeichensatz, so ist "3" das gleiche wie #33. Ein Null-Zeichen ,"", hat den Wert 0.
6. Eins der zwei Symbole:
   $ Stellt den gegenwaertigen Codestandort dar (Programmzaehler).
   @ Stellt den gegenwaertigen Speicherbereich dar.

Zeichensaetze

1. Arithmetische Oeratoren +,-,/,MOD.
2. Bitweise logische Operatoren AND, OR, XOR.

Der Assembler enthaelt eine Symboltafel, jedem ist ein Wert von 16 bit zugewiesen. Ein Symbol ist einer Basicvariablen aehnlich. Der Assembler macht zwei Arbeitsgaenge; beim ersten stellt er die Symboltafel auf, beim zweiten erschafft er den Objektcode, indem er die Symboltafelbenutzt um Sprungadresse usw.zu kalkulieren.Im ersten Arbeitsgang, wenn ihm ein Symbol zugewiesen wird, das noch nicht definiert worden ist, wird es auf die Symboltafel gesetzt. Der Wert wird eingegeben, wenn das Symbol definiert worden ist. Diese spaeteren Zuweisungen muessen alle im ersten Arbeitsgang absolviert werden;
Fehlermeldungen werden alle Symbole anzeigen, die nicht definiert worden sind. Keinem Symbol duerfen verschiedene Werte waehrend der zwei Arbeitsgaenge zugewiesen werden. Wenn dies passiert kann der Assembler viele Fehler erzeugen.
Es gibt einige Assembler Anweisungen, welche keine spaeteren Zuweisungen erlauben, weil der Ausdruckswert im ersten Arbeitsgang bekannt sein muss. Diese schliessen ORG ein - der Code-Ursprung muss gut definiert sein, denn anderweitig waere es unmoeglich fuer den Assembler eine korrekte Symboltafel zu erstellen. Die volle Liste dieser Anweisungen wird in dem Nachschlageabschnitt gegeben.

Ein 'Identifier' ist der Name eines Symbols. Gueltige 'Identifier' muessen die folgenden Regeln beachten:

1. Das erste Zeichen muss ein Buchstabe sein.
2. Die anderen Zeichen koennen sein wie folgt: Buchstabe, eine Ziffer unter 10, Fragezeichen (?), Punkt (.), Unterstreichungslinie (_).
Es gibt keine Laengenbegrenzungen, noch gibt es irgendwelche reservierte Worte.

4 Wege um ein Symbol zu definieren

1. Als ein Kennzeichen. Dies ist ein 'Identifier' am Beginn einer Angabe, moeglicherweise mit einem Punkt vorausgehend. Dem Symbol wird der Wert der gegenwaertigen Code-Location zugewiesen.

2. Durch die EQU (Gleichungs-) Anweisung
(Indentifier) EQU (Ausdruck)
Das Symbol ist definiert und ihm wird der Wert des Ausdrucks zugewiesen, welcher gut definiert sein muss (d.h. es darf keine spaeteren Referenzen beinhalten). Wenn ein Symbol schon definiert worden ist, wird eine Fehlermeldung gegeben (aber nicht wenn der neue und der alte Wert identisch ist). Mit anderen Worten, EQU darf nicht benutzt werden un ein Symbol zu redefinieren.

3. Die LET Anweisung.
LET ‹Identifier› = ‹Ausdruck›
Dies hat die gleiche Wirkung wie EQU, ausser dass LET Redefinition von Symbolen erlaubt. Achtung: Zur Vertraeglichkeit mit anderen Assemblern kann dies folgendermassen geschrieben werden: ‹Indentifier› DEFL ‹Ausdruck›

4. Die GET Anweisung. Siehe Kapitel ueber GET und PUT.

Wie man Daten in den Objektcode uebergibt

Die 3 Anweisungen die in diesem Abschnitt beschrieben werden bewirken , dass die Daten an der gegenwaertigen Codeposition assembliert werden.

BYTE ‹Liste von Ausdruecken und Zeichenketten›
TEXT ‹Liste von Ausdruecken und Zeichenketten›

BYTE und TEXT sind verschiedene Namen fuer die gleiche Sache. Sie nehmen eine Reihe von Parametern auf, wovon jedes ein arithmethischer Ausdruck oder ein Text-String sein kann. Jeder Ausdruck wird ausgewertet, und das Ergebnis in den Objektcode uebertragen. jeder String wird direkt Zeichen fuer Zeichen in den Objektcode uebertragen. Strings koennen entweder in Anfuehrungszeichen gesetzt werden, oder durch Hochkommas eingeschlossen werden. Wird das abschliessende Zeichen weggelassen, so wird davon ausgegangen, dass der String den Rest der Zeile bildet.

Achtung: von jedem String der aus *einem* Zeichen besteht, wird angenommen, dass es sich um eine numerische Konstante handelt. Daher sind Ausdruecke wie "A"+&80 erlaubt.

Beispiele:
```
BYTE 1,3,count*3+1,"q" or 128
Text "A string ending with cr-lf",13,10
```

WORD ‹Liste von Ausdruecken›

Jeder Ausdruck wird bewertet, und das aus 2 Bytes bestehende Ergebnis wird in den Objektcode uebertragen; das niederwertige Byte zuerst.

Beispiel:
```
WORD &C000,Adresse
```

RMEM ‹Ausdruck›

RMEM bewirkt, dass der Assembler die spezifizierte Anzahl von Bytes reserviert. Der Bereich des Objektcodes und der Speicherbereich wachsen beide jeweils um den Wert des Ausdrucks. Der so reservierte Raum wird mit Nullen aufgefuellt. Der Ausdruck darf keine spaeteren Referenzen enthalten.

Beispiele:
```
.buffer 256     RMEM  256
.word           RMEM  2
```

Compatibilitaet mit anderen Assemblern

Die folgenden alternativen Befehlsausdruecke sind erlaubt:

```
DEFB, DB, DEFM ... gleich mit BYTE, TEXT
DEFW, DW       ... gleich mit WORD
DEFS,DS        ... gleich mit RMEM
```

GET und PUT

GET und PUT sind die beiden Anweisungen, die benutzt werden um Parameter zwischen dem BASIC-Programm und dem Assembler waehrend des Assemblierens zu uebergeben. Anwendungen hierzu sind:

1. Uebergabe der Adresse wo der Code gespeichert werden soll
2. Uebergabe von Variablen
3. Rueckgabe von 'Entry Point' Adressen an BASIC

Um diese Anweisungen zu nutzen, werden eine Reihe von Parameter an das !ASSEMBLE Kommando angehaengt, welche durch Kommas getrennt werden. z.B. !ASSEMBLE,start,x,@start1

Jeder Parameter muss entweder eine numerische BASIC-Variable sein, oder die Adresse einer numerischen (integer od. realen) BASIC-Variablen (z.B. @start1).

Die GET Anweisung wird innerhalb des Assemblerprogramms benutzt, die Werte dieser Parameter zu lesen.

Syntax: GET ‹Liste der 'Identifier'›

So kann eine GET Anweisung,die mit den oben aufgefuehrten Parametern arbeitet, folgendermassen aussehen:

```
GET  start, x, start_adr
```

Die Wirkung hiervon ist, dass die 2-Byte-Werte den Zeichen innerhalb der GET Anweisung zugewiesen werden.
Die gewaehlten Namen stehen nicht in Zusammenhang mit den Namen der BASIC-Variablen,der Uebersichtlichkeithalber ist es aber von Vorteil gleiche Bezeichnungen zu waehlen. Es ist nicht erforderlich saemtliche Werte mit einer einzigen GET Anweisung zu lesen, deshalb ist folgendes genauso gut moeglich:

```
GET  start
GET  x,start,start_adr
```

Eine ERROR-Meldung erfolgt, wenn versucht wird Parameter zu uebernehmen, und keine Daten mehr vorhanden sind.

Beispiele fuer die Benutzung von GET

1. Um eine Adresse an den Assembler zu uebergeben

```
   10 GOSUB 1000
   20 CALL start
   30 END
 1000 start=HIMEM+1
 1010 !ASSEMBLE,start
 1020 'GET start
 1030 'ORG start
 1040 ';hier folgt der Quellcode
      '...
 1999 RETURN
```

Der Wert von HIMEM+1 wird der BASIC-Variablen 'start' zugewiesen und an den Assembler uebergeben, welcher den Code-Ursprung auf diesen Wert setzt.Spaeter fuehrt BASIC die Anweisung 'call start' aus, um die Maschinencode Routine aufzurufen.

Achtung: In diesem Beispiel ist GET nicht notwendig, gibt aber Klarheit zum Programm. Da der Assembler bei nicht einsetzen den Code-Ursprung auf Himem+1 setzt,ist das obige Programm gleichzusetzen mit folgendem:

```
   10 !ASSEMBLE
   20 ';hier folgt der Quellcode
      '...
  900 CALL HIMEM+1
```

2. Zur Uebergabe von Variablen zur bedingten Assemblierung
   Siehe Kapitel 'Bedingtes Assemblieren'
3. Zur Rueckgabe von 'Entry Point' Addressen an BASIC.

Oft hat ein Maschinencodeprogramm mehr als einen 'Entry Point'.Der einfache Gebrauch von GET ermoeglicht es nur den Anfang des Codes abzurufen, aber auch beim Gebrauch von PUT, kann jede Anzahl von Adressen an BASIC zurueckgegeben werden.
Syntax: PUT <Ausdruck> , <Ausdruck>

Die Wirkung von PUT ist, einen Wert einer BASIC Variablen zuzuweisen (integer oder real). Da Namen von BASIC Variablen keine Bedeutung fuer den Assembler haben , wird eine kompliziertere Methode gebraucht um dies auszufuehren. Dies ist das Verfahren, dass benutzt werden muss:

1. Schaffen der Basic Variable durch, z.B.
   entry=0
2. Uebergabe der Variablen-Adresse an den Assembler:
   !ASSEMBLE,@entry
3. Einfuegen der Adresse in eine Assembler Variable:
   GET entry_adr
4. (Freigestellt, aber zur Klarheit empfohlen). Am 'Entry Point' vom Code wird ein Label definiert:
   .entry
5. Eingabe der 'entry address' in die Basic Variable:
   PUT entry_adr,$

oder PUT entry_adr,entry

PUT ist dem Basic Kommando Poke insofern aehnlich als das es einen Wert hat und es in einer spezifizierten Memoryadresse speichert. Der Unterschied ist der, das POKE einen einzelnen byte Wert hat, waehrend PUT einen zwei byte Wert und verwendet wenn noetig einen schwebenden Punkt.
Warnung: PUT kann nur mit einer variablen Zuweisung als ersten Parameter benutzt werden.
Bemerkung: GET ist ein assemblierungsweise Einrichtung. Parameter, die jederzeit die aufgerufene Routine aendern, sollten mit dem Basic 'CALL' Angabe verschwinden, wie im Programm 3 illustriert wird.

Programm 3: Darstellung des Gebrauchs von PUT

```
  10 GOTO 30    'zuerst RUN 20 um speicher zu reservieren
  20 MEMORY HIMEM-100
  30 GOSUB 1000
  40 CALL hexout2,7
  50 PRINT
  60 call hexout4,&1E2B
 999 END
1000 REM Maschinencode Routine zur Ausgabe von Hex-Zahlen
1010 hexout2=0:hexout4=0  'Variablen welche den 'Entry Point'
                                                  enthalten
1020 |ASSEMBLE,@hexout2,@hexout4
1030 'GET hexout2_ref,hexout4_ref
1040 'LET txtoutput=&BB5A
1050 'hexout4    PUT hexout4_ref,$
1060 '           LD  A,(IX+1)      ; get high byte of parameter
1070 '           CALL hexout2      ; output in hex
1080 '           LD A,(IX)         ; get low byte
1090 '           CALL hexout2      ; output it
1100 '           RET
1110 '           PUT hexout2_ref,$
1120 '           LD A (IX)         ; get (low) byte
1130 'hexout2    PUSH AF           ; save A
1140 '           RRCA:RRCA:RRCA:RRCA
1150 '           CALL hexout1      ;output high order hex digit
1160 '           POP AF            ;restore A and output 2nd digit
1170 'hexout1    CALL binasc       ;convert binary to ASCII
1180 '           CALL txtoutput    ;output ASCII character
1190 'RET
1200 'binasc     AND &F            ;mask out top 4 bits
1210 '           ADD A,&30         ;convert decimal digits to ASCII
1220 '           CP &3A            ;ist es eine Dezimahlzahl?
1230 '           RET C             ;ja, dann sind wir fertig
1240 '           ADD A,7           ;nein, dann ist es ei HEX-Wert
                                    zwischen A und F

1250 '           RET
1260 RETURN
```

Bemerkung: Die CALL Angabe stellt die Register wie folgt auf:
A = die Anzahl der Parameter
IX = die Adresse der Parameter
Alle anderen Register sind undefiniert.

Die Parameter sind in umgekehrter Anordnung aufgelistet, das niederwertige Byte zuerst.

Bedingte Assemblierung

Bedingte Assemblierung wird benutzt, wenn zwei oder mehr Versionen eines Programmes benoetigt werden. (z.B. Cassetten- und Diskettenversion). Dieses Merkmal ermoeglicht es, dass eine Anzahl von verschiedenen Versionen vom selben Quellcode assembliert werden kann.
Dies wird getan, indem man die Bloecke vom Quellcode, die assembliert werden sollen definiert, vorausgesetzt es werden einige Bedingungen eingehalten. Die Formate von IF Bloecken sind:

1. IF ‹Ausdruck›
   ‹Code wird assembliert, wenn der Ausdruck richtig ist.›
   ENDIF
2. IF ‹Ausdruck›
   ‹Code wird assembliert, wenn der Ausdruck richtig ist.)
   ELSE
   ‹Code wird assembliert, wenn der Ausdruck falsch ist.)
   ENDIF

Der Ausdruck kann ein arithmetischer Ausdruck sein. In diesem Zusammenhang wird der Wert der Ausdruecke fuer eine gekennzeichnete 16 bit Zahl gehalten, mit 'true' fuer irgendeine positive Zahl (z.B. zwischen 1 und 32767) und 'false' fuer Null oder irgendeine negative Zahl.
Der empfehlenswerte Gebrauch ist, eine Variable, die den Wert 1 fuer richtig und den Wert 0 fuer falsch beinhaltet, zu definieren.
Beispiel: Nehmen Sie an, ein Programm kaeme in zwei Versionen, fuer Kassette und Diskette, und da waeren ein paar Unterschiede zwischen den beiden. Definieren Sie eine Variable am Anfang des Quellcodes:

```
LET cassette=1 ; um die Cassettenversion zu assemblieren
LET cassette=0 ; um die Diskettenversion zu assemblieren
```

Dann schliesst man jeden Abschnitt, in dem der Code abweicht, in einen IF Block, wie folgt, ein:

```
IF Cassette
‹Code fuer Cassettenversion›
ELSE

‹Code fuer Diskettenversion›
ENDIF
```

Testen von Ungleichheiten

Oft muss ein Programm in eine vorgegebene Anzahl von Bytes passen. Die IF Anweisung erlaubt dem Assembler die gegenwaertige Codelage gegenueber der hoechstmoeglichen Lage und folglichen Handlung zu pruefen. (Bemerkung: Die LIMIT Anweisung ueberprueft nur die Speicherlage).
Beispiel: Anweisungen 'start' und 'end' sind erklaert. Die Laenge des Programms darf die Maximallaenge (maxlength),welche als Parameter gilt, nicht ueberschreiten.

```
1000 !ASSEMBLE, maxlength
1010 'GET maxlength
1020 '.start
     ‹hier Code›
1900 '.end
1910 IF end-start-maxlength  ; wahr if end-start  ›  maxlength
1920 'PRINT "Code zu lang!"
1930 'ENDIF
1940 RETURN
```

Logische Operatoren

AND, OR und XOR koennen mit Vorsicht in If Anweisungen benutzt werden. Diese sind bitweise logische Operatoren und arbeiten wie erwartet, wenn wahr mit einer 1, und unwahr nur mit 0 repraesentiert werden. So werden Variablen, die nur immer die Werte 1 oder 0 erhalten, in den ueblichen Resultaten erfasst (1 OR 0 ist wahr, 1 AND 0 ist unwahr, 1 XOR 1 ist unwahr, etc.).

Beispiel: IF cassette.version AND english.version
Warnung : Obwohl 1 und 2 beide wahr darstellen, wird der Ausdrueck 1 AND 2 als 0 berechnet (d.h. unwahr).

IFNOT

Der Bequemlichkeithalber kann IFNOT anstatt IF benutzt werden.Es kann einfach die Logik der If Anweisung reversieren.

```
IFNOT ‹Ausdruck›
‹zu assemblierender Code, wenn der Ausdruck unwahr ist›
Else ‹Ausdruck›
‹zu assemblierender Code, wenn Ausdruck wahr ist›
ENDIF
```

Verschachteln von IF Bloecken

IF Bloecke lassen sich bis zu einer Tiefe von 10 verschachteln. Es ist allerdings ungewoehnlich, dass man eine Verschachtelung benoetigt, die ueber den Umfang von 2 hinausgeht.

Beispiel:

```
IF rom.version
‹ROM code›
ELSE: IF Diskettenversion
      ‹cassette code›
      ENDIF
ENDIF
```

IF1, IF2

Diese speziellen Formen der IF Anweisung uebergeben den Wert 'wahr' beim ersten und zweiten pass der Assemblierung.
Sie koennen nuetzlich sein, um verschiedene Meldungen bei pass 1 und 2 anzuzeigen, Z80 Befehle und Direktiven sollten allerdings nicht in einem IF1 oder IF2 Block plaziert werden.

Wie man den Quellcode von einem File liest

Syntax: READ<Filename>
Wenn der Assembler eine READ Anweisung findet, wird er den spezifizierten File eroeffnen, den Inhalt des Files assemblieren, und der READ Anweisung folgen.
Das File kann ein Basic File sein (tokenised oder ASCII Format) oder ein Text File (erstellt mit dem MAXAM Editor oder einem anderen Wordprozessor). Das erste nichtleere Zeichen hinter dem Wort "READ" wird als Filename Abgrenzer benutzt.
Es kann jeweils nur ein File eroeffnet werden. Deshalb muss die READ Anweisung selbst im Speicher sein. Wenn das ganze Programm sich in einem File befindet,kann das Programm im Speicher so einfach sein wie folgendes:

```
| ASSEMBLE:'READ"FILE"
```

Wie man READ mit Cassette benutzt

Das File wird bei jedem Durchgang des Assemblers gelesen, deshalb wird es noetig sein den Datenrekorder fuer jeden Durch- in der korrekten Position zu haben. (Es koennte leichter sein, das Quellcode File zweimal auf der Cassette aufzunehmen um ein unnoetiges zurueckspulen zu vermeiden). Wenn mehrere Files in Gebrauch sind, kann der PRINT Befehl nuetzlich sein, um anzuzeigen, welches File benoetigt wird. Laesst man den Filename aus, wird das erste File benutzt, dass auf dem Band gefunden wird.

Ein nuetzlicher Rat

Wenn das Programm zwischen mehreren Textfiles geteilt ist, hilft es den Namen jedes Files auszudrucken, den der Assembler liest. Das ist leicht gemacht, wenn die erste Zeile jedes Files wie folgt aussieht:

```
1 PRINT "<Name des Files>  <Datum>"
```

Dies ist auch nuetzlich beim Editieren; ohne einen Namen im oberen Teil des Files kann leicht vergessen werden, welches File man gerade editiert.
Die Ziffer 1 verursacht, dass der Assembler den Zeilenzaehler zurueck auf 1 setzt. Dies bedeutet, dass Fehlermeldungen die korrekte physikalische Zeilennummer innerhalb der Files angeben, wo der Fehler eintrat. Der Editor erhaelt den Befehl zu einer bestimmten Zeile zu gehen. Diese Besonderheiten gemeinsam genutzt, erlauben ein wesentlich schnelleres Debugging.

Wie man Objektcode in ein File schreibt

Syntax: WRITE <Filename>
Die WRITE Direktive teilt dem Assembler mit, einen neuen Binaer File zu schaffen und alle dementsprechenden Objektcodes in dem File zu speichern (ausser wenn dies mit NOCODE verhindert wird). Nur fuer Diskettenbetrieb: Wenn der Filename ein '.COM' als Suffix beinhaltet, wird ein CP/M Objektfile gebildet, welcher direkt unter CP/M ausgefuehrt werden kann. Der Code Ursprung muss durch 'ORG &100 definiert werden. Ein solches File hat kein Binaerformat, und kann somit nicht unter AMSDOS benutzt werden.

Syntax: CLOSE
Die CLOSE Anweisung teilt dem Assembler mit, den gegenwaertig offenen Outputfile zu schliessen, und das Speichern des Code wieder aufzunehmen.
CLOSE kann normalerweise ausgelassen werden, da ein File automatisch durch eine WRITE Anweisung oder am Ende der Assemblierung geschlossen wird.
Beispiel: Wie man Sourcecode von einem File liesst und auf ein anderes File schreibt

```
WRITE "objekt"
READ "source"
```

Assembler Anweisungen

Anweisungen kontrollieren das Listing und den Output, welcher vom Assembler produziert wird. Sie selbst erscheinen nicht auf dem Assembler-Listing, es sei denn, ein Kennzeichen ist mit einer Anweisung verbunden oder es ist ein Fehler im Befehl.

LIST
NOLIST

LIST bewirkt, dass der Assembler mit dem Auflisten beginnt. Dies ist der Anfangszustand.
NOLIST bewirkt, dass der Assembler mit dem Auflisten aufhoert.

PRINT ‹String›

Der String wird auf dem Bildschirm angezeigt, sogar wenn das Auflisten abgestellt ist. Es wird angenommen, dass das erste nicht leere Zeichen nach dem Wort 'PRINT' der String-Delimiter ist.

PAUSE

Der Assembler wird warten bis eine Taste gedrueckt wird. PAUSE funktioniert nur wenn man auflisten kann. Es erlaubt, dass ein Teil der Auflistung geprueft werden kann. PAUSE koennte nuetzlich sein, sofort nach einem PRINT Befehl.

DUMP

Wenn ein DUMP Befehl im Progamm erscheint, und das Listen ermoeglicht ist, wird eine komplette Liste von allen definierten Symbolen mit ihren Werten in Hexadezimalzahlen produziert, nachdem das Assemblieren beendet ist. Die Folge ist nicht streng alphabetisch, aber alle Symbole, die mit dem gleichen Buchstaben beginnen, sind zusammen aufgelistet. Dies ist eine Konsequenz des Weges, wie der Assembler die Symbole speichert - eine Methode, die zwecks Schnelligkeit und Sparsamkeit, fuer die Belegung des Speichers gewaehlt wurde.

Ein Beispiel-Listing

```
ARNOR Z80 ASSEMBLER Version 1.02                          Page 001

00001  0000          ; zur Unterdrueckung von Line Feeds an den Drucker
00002  0160  (0160)            org     &160
00003  0160  (FFFF)            limit   &FFFF
00004  0160  (BD2B)    mcprintchar
                               equ     &BD2B
00005  0160  FE 0A     patch   cp      10
00005  0162  37                scf
00005  0163  C8                ret     z
00005  0164  CF 1B 88           rst     1,&881B       ;    nur 6128
00006                ;464 - rst 1,&87f2   664 - rst 1,&880B
00007  BD2B  (BD2B)            org mcprintchar
00008  BD2B  C3 60 01          jp patch
00009  BD2E  (BD2E)            end

Symboltabelle:

B55D MCPRINTCHAR          ABF0 PATCH

ERRORS: 00000     Warnings:00000
```

Die 7 Teile des Listings

1. Zeilennummer.

Wenn der Quellcode sich innerhalb eines BASIC-Programmes befindet, so wird die BASIC-Zeilennummer ausgedruckt . Dies erlaubt leichtes orten von Fehlern, weil wenn ein Fehler eintritt, die Zeile,die den Fehler enthaelt, zusammen mit der Zeilennummer aufgelistet wird. Wenn der Quellcode im ASCII Format ist, wird der Assembler nach einer Zeilennummer am Beginn der Zeile sehen (in Dezimal), sollte er eine finden,so wird er sie benutzen. Ansonsten wird er die Zeilen zaehlen.
Bemerkung: Die Zeilennummer beziehen sich auf physikalische Zeilen; Doppelpunkttrenner werden nicht die Zeilennummer aendern.

2. Codeortung

3. Objektcode

Bis zu 4 Bytes pro Zeile. Verschiedene Anweisungen koennen bewirken,dass mehr als 4 Bytes zu assemblieren sind. In diesem Fall wird der Objektcode auf mehr als einer Zeile gelistet. 4 Bytes in einer Zeile.

4. Kennzeichenfeld

5. Instruktionsfeld

6. Operandfeld

7. Kommentarfeld

Dies bezieht sich auf das Listing im 80 Zeichen Mode. Im 40 Zeichen Mode wird die Zeilennummer ausgelassen, und die Informationen werden in komprimierter Form gegeben. Ein Farbcode-Schema dient dabei als Hilfe.
Mehrere Anweisungen bewirken, dass eine Zahl in Klammern hinter der Adresse ausgegeben wird.

Die Anweisungen und die Bedeutung der Zahlen sind wie folgt:

END	:	die Speicherstelle
EQU	:	der zugewiesene Wert
GET	:	der Wert des letzten gelesenen Parameters
IF	:	der Wert des bedingten Ausdrucks
LET	:	der zugewiesene Wert
LIMIT	:	das festgelegte Limit
ORG	:	die Speicherstelle
PUT	:	der zugewiesene Wert
RMEM	:	die Anzahl der reservierten Bytes

Listing mit dem Drucker

LIST P
schaltet das 'Listen' mit dem Drucker ein
Beispiele:
(i) zum 'Listen' auf Bildschirm und Drucker
LIST:LIST P
(ii) Auflistung nur auf dem Drucker
NOLIST:LIST P

PLEN ‹Ausdruck›
Ohne das PLEN-Kommando ist die Auflistung kontinuirlich ohne Seitenumbruch. PLEN definiert die Anzahl der Zeilen pro Seite. Dazu muss sich der Druckkopf am Anfang der Seite befinden (genau dort, wo die erste Zeile gedruckt werden soll). Stellen Sie PlEN auf die genaue Anzahl der Zeilen pro Seite ein. Dies ist nicht die Anzahl der Zeilen die gedruckt werden soll-einige Leerzeilen befinden sich automatisch am unteren Ende der Seite.
Der Wert des Ausdrucks kann entweder 0 sein oder zwischen 40 und 255 liegen.
PLEN 0 weisst den Assembler zum kontinuirlichen 'listen' an.

Beispiele: PLEN 66 fuer 11" Papier
PLEN 72 fuer 12" Papier

PAGE (‹Ausdruck›)
Das Kommando PAGE verursacht, dass eine Seite ausgeworfen wird. Die Seitenlaenge wird dazu benutzt die Anzahl der Leerzeilen auszurechnen, wodurch die neue Seite an der richtigen Position auf dem Papier beginnen kann.
Der Ausdruck ist optional. Bei Angabe eines Wertes wird dieser als die neue Seitennummer verwendet. Dies darf 255 nicht ueberschreiten. Gibt man keinen Wert an, so ist die Seitennummer um 1 hoeher als die vorherige.

TITLE ‹String›
Dies bestimmt einen Titel, der jeweils am Kopf der Seite gedruckt werden soll. Um dies auf der ersten Seite drucken zu koennen, muss das TITLE Kommando vor der ersten Direktive oder dem ersten Mnemonic erscheinen. Der Titel wird in der ersten Spalte beginnend gedruckt. Um den Titel zu zentrieren, geben Sie bitte die noetige Anzahl von Leerstellen in den String ein.
TITLE ohne einen String bricht die TITLE Option ab, waehrend TITLE"" eine Leerzeile ausgeben wird.
Das erste nicht leere Zeichen hinter dem Wort 'TITLE' wird als String-Abgrenzer angenommen.

WIDTH ‹Ausdruck›
Dies legt die Anzahl der Zeichen pro Zeile in einem Listing fest. Ohne Angabe entspricht dies dem gegenwaertigen Bildschirmmode (40 oder 80 Zeichen). Der Ausdruck kann im Bereich zwischen 40 und 255 liegen.
WIDTH 0 bewirkt ein ruecksetzen auf den urspruenlichen Mode.

Beispiel: WIDTH 132

Programm 4

```
  10 REM M-Code Routine um Bloecke im Speicher zu verschieben
  20 REM
  30 REM RUN 50 als Erstes, um Speicher zu reservieren
  40 GOTO 60
  50 MEMORY HIMEM-100
  60 mov=HIMEM+1
  70 GOSUB 1000
  80 REM ein einfaches Beispiel,welches den Bildschirm-
     speicher verschiebt
  90 first=&C000  'Beginn des zu verschiebenden Blocks
 100 length=&3ff0 'Laenge des Blocks
 110 dest=&c010   'Adresse an welche der Block verschoben wird
 120 call mov,first,length,dest
 130 END
1000 |ASSEMBLE,mov
1010 ' get move                          ;get Startadresse
1020 'txt_output equ &bb5a               ;Firmware Entrypoint
1030 '
1040 'org move
1050 'cp3                                ;pruefe Parameterwerte
1060 'jr nz,error                        ;falscher Wert
1070 'ld e, (ix)                         ;dest low byte
1080 'ld d, (ix+1)                       ;dest high byte
1090 'ld c, (ix+2)                       ;Laenge, low byte
1100 'ld b, (ix+3)                       ;Laenge, high byte
1110 'ld l, (ix+4)                       ;first, low byte
1120 'ld h, (ix+5)                       ;first, high byte
1130 'push hl                            ;sichern
1140 '
1150 'or a
1160 'sbc hl,de                          ;first mit dest vergleichen
1170 'pop hl                             ;restore first
1180 'jr c,moveb                         ;jump wenn dest >=first
1190
1200 'ldir                               ;Block verschieben
1210 'ret                                ;
1220 '
1230 'moveb                              ;rueckwaerts verschieben
1240 'ex de,hl                           ;dest in hl
1250 'add hl,bc                          ;Laenge hizufuegen
1260 'ex de,hl                           ;
1270 'dec de                             ;de momentan=Endadresse
1280 'add hl,bc
1290 'dec hl                             ;
1300 'lddr                               ;Block verschieben
1310 'ret
1320 '                                   ;
1330 'error
1340 'call txtout                        ;print Fehlermeldung
1350 'text "Falsche Parameterwerte",13,10,0
1360 'ret
1370 '
1380 'txtout
1390 'pop hl                             ;'get'Adresse der Meldung
1400 'txtl
1410 'ld a, (hl)                         ;'get' Zeichen
1420 'call txt_output                    ;Textausgabe
1430 'inc hl                             ;Sprung zum Text
1440 'or a                               ;Test Akkumulator
1450 'jr nz, txtl                        ;fortfuehren bis 0
1460 'jp hl                              ;return
1470 '
1480 'end
1490 RETURN
```

Programm 5(a)

Dies ist ein BASIC Programm um ein Dreieck auf dem Bildschirm zu fuellen. Eine direkte Uebersetzung zu dem Maschinen Code ist darunter gegeben.
Versuchen Sie sie beide, und vergleichen Sie die Geschwindigkeiten!

```
10  REM  triangle fill - BASIC
20  INPUT "Enter triangle coordinates: ",x1,y1,x2,y2,x3,y3
30  CLS
40  IF y1>y2 THEN t=y1:y1=y2:y2=t:t=x1:x1=x2:x2=t
50  IF y1>y3 THEN t=y1:y1=y3:y3=t:t=x1:x1=x3:x3=t
60  IF y2>y3 THEN t=y2:y2=y3:y3=t:t=x2:x2=x3:x3=t
70  FOR h=0 TO y2-y1
80  MOVE x1+h*(x2-x1)/(y2-y1),y1+h
90  DRAW x1+h*(x3-x1)/(y3-y1),y1+h
100 NEXT
110 FOR h=0TO y3-y2
120 MOVE x2+h*(x3-x2)/(y3-y2),y2+h
130 DRAW x1+(h+y2-y1)*(x3-x1)/(y3-y1),y2+h
140 NEXT
```

Programm 5(b)

```
10 REM triangle fill - machine code
20 GOTO 40  'RUN 30 the first time
30 MEMORY HIMEM-600
40 GOSUB 1000
50 INPUT "Enter triangle coordimates: ",x1,y1,x2,y2,x3,y3
60 y1=y1/2:y2=y2/2:y3=y3/2  '  convert coordinates
70 CLS:CALL tfill,x1,y1,x2,y2,x3,y3:END
1000 tfill=HIMEM+1
1010 öASSEMBLE
1020  '          scr_horizontal  equ   &bc5f;Firmwareroutine  um
Linien                                      Linien zu zeichnen
1030 '         txt_get_pen equ &bb93       ;Firmwareroutine fuer Pen-
                                            farben
1040 '         scr_ink_encode equ &bc2c
1050 '         cp 6:ret nz                ;   Return,   falls   falsche
                                            Parameterwerte
1060 '         push ix:pop h1             ; Adressen der  Parameter  in
                                            hl bringen
1070  '        ld de,y3:ld bc,12:ldir     ; in den Arbeitsspeicher kop-
                                            pieren
1080                                       ; erster Scheitelpunkt der Y-
                                            Coordinate
```

```
1090      ld hl,(y2):ld de,(y1)
1100 '    call cphlde:jr nc,noswap1
1110 '    push hl:ld hl,(x1):ld bc,(x2)
1120 '    ld (x1),bc:ld (x2),hl:pop hl:ex de,hl
1130 '    noswap1
1140 '    ld (y1),de:ld (y2),hl
1150 '
1160 '    ld hl,(y3):ld de,(y1)
1170 '    call cphlde:jr nc,noswap2
1180 '    push hl:ld hl,(x1):ld bc,(x3)
1190 '    ld (x1),bc:ld (x3),hl:pop hl:ex de,hl
1200 '    noswap2
1210 '    ld (y1),de:ld (y3),hl
1220 '
1230 '    ld hl,(y3):ld de,(y2)
1240 '    call cphlde:jr nc,noswap3
1250 '    push hl:ld hl,(x2):ld bc,(x3)
1260 '    ld (x2),bc:ld (x3),hl:pop hl:ex de,hl
1270 '    noswap3
1280 '    ld (y2),de:ld (y3),hl ;
1290 '
1300 '
1310 '    ld hl,0:ld (count),hl
1320 '    loop1
1330 '    ld hl,(x2):ld de,(x1)
1340 '    or a:sbc hl,de:call testneg
1350 '    ld de,(count):call multhlde
1360 '    push hl:ld hl,(y2):ld de,(y1)
1370 '    or a:sbc hl,de
1380 '    ex de,hl:pop hl
1390 '    call divhlde
1400 '    ld a,(flag):or a:call nz,neghl
1410 '    ld de,(x1):add hl,de:push hl          ;Start der X-
                                                    Koordinate
1420 '
1430 '    ld hl,(x3):ld de,(x1)
1440 '    or a:sbc hl,de:call testneg
1450 '    ld de,(count):call multhlde
1460 '    push hl:ld hl,(y3):ld de,(y1)
1470 '    or a:sbc hl,de
1480 '    ex de,hl:pop hl
1490 '    call divhlde
1500 '    1d a,(flag):or a:call nz,negh1
1510 '    ld de,(x1):add  hl,de:push  hl        ;Ende der x Ko-
1520 '                                                ordinate
1530 '    ld hl,(y1):ld de,(count)
1540 '    add hl,de                             ;y Koordinate
1550 '    pop bc:pop de
1560 '    call drawline
1570 '
1580 '
1590 '    ld de,(count):inc de:ld (count),de
1600 '    ld bc,(y1):ld hl,(y2)
1610 '    or a:sbc hl,bc
1620 '    call cphlde:jp nc,loop1
1630 '
1635 '    ld hl,0:ld (count),hl
1640 '    loop2
1650 '    ld hl,(x3):ld de,(x2)
1660 '    or a:sbc hl,de:call testneg
1670 '    ld de,(count):call multhlde
```

```
1680 ' push hl:ld hl(y3):ld de,(y2)
1690 ' or a:sbc hl,de
1700 ' ex de,hl:pop hl
1710 ' divhlde '
1720 ' ld a,(flag):or a:call nz,neghl
1730 ' ld de,(x2):add hl,de:push hl            ;Start X-Koordinate
1740 '
1750 ' ld hl,(x3):ld de,(x1)
1760 ' or a:sbc hl,de:call testneg
1770 ' push hl:ld hl,(count)
1780 ' ld de,(y2):ld bc,(y1)
1790 ' add hl,de:or a:sbc hl,bc
1800 ' ex de,hl:pop hl
1810 ' call multhlde
1820 ' push hl:ld hl,(y3):ld de,(y1)
1830 ' or a:sbc hl,de
1840 ' ex de,hl:pop hl
1850 ' CALL DIVHLDE
1860 ' ld a,(flag):or a:call nz,neghl
1870 ' ld de,(x1):add hl,de:push hl            ;Ende X-Koordinate
1880 '
1890 ' ld hl,(y2):ld de (count)
1900 ' add hl,de                               ;Y-Koordinate
1910 ' pop bc:pop de
1930 ' call drawline
1940 '
1950 ' ld de,(count):inc de:ld (count),de
1960 ' ld bc,(y2):ld hl,(y3)
1970 ' or a:sbc hl,bc
1980 ' call cphlde: jp nc,loop2
1990 ' ret
2000 '
2005 ' drawline call txt_get_pen:call scr_ink_encode ;ink in A
2006 ' push bc:ex hl,(sp):call cphlde:jr nc,dline1:ex de,hl
2007 ' dline1 ex hl,(sp):pop bc:call scr_horizontal:ret
2008 '
2010 ' multhlde              ;subroutine to do hl:hl*de
2020 ' push bc:ld c,h:ld a,l:ld b,16:ld hl,0
2030 ' mult1 srl c:rra:jr nc,mult2:add hl,de
2040 ' mult2 ex de,hl:add hl,hl:ex de,hl:djnz mult1:pop bc:ret
2060 ' divhlde              ;subroutine to do:hl/de
2070 ' push bc:ld c,l:ld a,h:ld hl,0:ld b,16
2080 ' div1 rl c:rla:adc hl,hl:sbc hl,de:jr nc,div2:add hl,de
2090 ' div2 ccf:djnz div1:rl c:rla:ld h,a:ld l,c:pop bc:ret
2110 ' cphlde               ;subroutine to set carry if hl<de
2120 ' push hl:or a:sbc hl,de:pop hl:ret
2130 ' testneg ld a,0:call c,neghl:ld (flag),a:ret
2140 ' neghl push de:ld de,0:ex de,hl:or a:sbc hl,de:pop de:lda,&ff
2150 ' ret
2160 ' y3 word 0:x3 word 0          ;workspace
2170 ' y2 word 0:x2 word 0y1 word 0:x1 word 0
2190 ' count word 0:flag byte 0
2210 ' RETURN
```

7. MENUE betriebene Utilities

Der Befehl '|MAXAM' bringt Sie in das Haupt-Menue von MAXAM. Das Menue listet alle zur Verfuegung stehenden Befehle auf - jedes ist durch einen ein- oder Zwei-Buchstabencode selektiert.Um einen Befehl zu gebrauchen, tippen Sie den Code ein und druecken Sie ENTER. Hilfen auf dem Bildschirm werden Ihnen sagen, welche Eingabe fuer jeden Befehl noetig ist. Sie werden sehen, dass der Wert von HIMEM auf der obersten Zeile zu sehen ist, zusammen mit der ROM Auswahl (welche unten erklaert werden). Durch ESC gelangen Sie immer ins Menue zurueck.

Beachte:alle Adressen werden in Hexadezimal angegeben. Dieser Abschnitt erklaert im folgenden alle Befehle.

T bringt Sie in den Text Editor, welcher im naechsten Kapitel abgehandelt wird.
B laesst Sie ins BASIC zurueckkehren, wobei der Editor-Text und das BASIC-Programm erhalten bleiben.
D Disassemble
DP Disassemble to Printer

Start- und Endadresse sind erforderlich und muessen in Hexadezimal eingegeben werden. Werden sie nicht eingegeben, so wird von 0 und &FFFF ausgegangen. Der Speicherinhalt wird in Z80 mnenomonics disassembliert; die HEX- und ASCII-Werte werden ebenfalls aufgelistet.Es werden die Standard Zilog Mnemonics angezeigt (siehe Referenzsektion fuer eine komplette Auflistung). RST Instruktionen werden mit den passenden Parametern disassembliert (siehe Referenzsektion).

L List Memory
LP List Memory to printer

Dies funktioniert in der gleichen Weise wie D, listet jedoch nur Hex und ASCII; 8 oder 16 Bytes in einer Zeile. Die ASCII-Anzeige ignoriert bit 7 und zeigt einen Punkt als Kontrollcode an. Dies erlaubt ein einfaches Erkennen des Textes, Strings mit dem oberen bit des letzten Zeichensatzes eingeschlossen.
Der Memory Editor (unten beschrieben) stellt den kompletten Zeichensatz des CPC dar.

S Select upper ROM

Der angewaehlte ROM wird durch alle Komandos des Hauptmenues benutzt ,welche Adressen zulassen die im Bereich zwischen &C000 und &FFF liegen. (Die ROM-Nummern werden durch das HELP-Kommando angezeigt - siehe unten). So z.B. um ROM 7 zu disassemblieren:

S ‹ENTER› 7 ‹ENTER›
D ‹ENTER› C000 ‹ENTER› ‹ENTER›

Das gegenwaertig selektierte ROM wird oberhalb des Menues angezeigt. als erstes wird ROM 0 selektiert, welches das BASIC ROM ist.

O Lower ROM on/off

Dieses Kommando waehlt entweder den unteren RAM oder ROM, welcher durch Kommandos genutzt werden kann, deren Adressen im Bereich zwischen 0 und &3FFF liegen. Die gegenwaertige Wahl wird in der obersten Zeile angezeigt. Die anfaengliche Stellung ist lower ROM= off.
Zum listen des unteren ROM:

O ‹ENTER›
L ‹ENTER›‹ENTER›‹ENTER›

E Speicher loeschen

Der Memory Editor erlaubt Ihnen den Inhalt des Speichers direkt zu aendern, indem man einfach ueberschreibt, was auf dem Bildschirm zu sehen ist. Sie werden nach der Adresse gefragt, welche Sie editieren moechten, und es wird der Speicherinhalt angezeigt, in einer aehnlichen Weise, wie Sie es vom L Kommando kennen.
Die Adresse, die Sie waehlen, wird in der Mitte des Bildschirms sein, und der Cursor wird auf dem Byte sein, das diese Adresse belegt.
Um den Inhalt einesjeden Bytes zu wechseln, tippt man einfach den neuen Wert ein. Es ist nicht noetig, ENTER nach der Nummer zu druecken, geben Sie einfach die zwei Hex Zahlen unter 10 ein. Eine einzelne einstellige Zahl kann durch ENTER oder irgendeine Cursortaste begrenzt werden. DEL wird eine einstellige Zahl aufheben.
Die Cursor Tasten koennen benutzt werden um auf dem Schirm herumzuwandern. Vier andere Funktionen werden gegeben:

SHIFT- ↑ or CTRL- ↕ : einen Bildschirm zurueckbewegen
SHIFT- ↓ or CTRL- ↕ : einen Bildschirm vorwaerts bewegen
SHIFT- ← or CTRL- ← : Cursor zur oberen linken Ecke
SHIFT→ or CTRL→ : Cursor zur unteren rechten Ecke

Der Editor kann auch benutzt werden, um ASCII Zeichen direkt in den Speicher zu geben. Druecken Sie TAB, und der Cursor wird an der Stelle positioniert, die den ASCII des aktuellen Bytes darstellt. Der Editor funktioniert in der gleichen Weise, ausser dass ASCII Zeichen eingegeben werden, anstelle von Hex-Zahlen.
Druecken Sie TAB noch einmal, kehren Sie zum Hex Editing zurueck. Druecken Sie ESC um den Editor zu verlassen und zum Menue zurueckzukehren.

F Finde String
FP Finde String und Drucke

Diese Befehle suchen in einem Speicherblock nach einem String. Sie koennen zwischen ASCII und HEX waehlen. Im ersten Fall kann der String bis zu 20 Zeichen beinhalten, im zweiten Fall bis zu 8 Bytes. ENTER wird benutzt um den Eingang von Strings zu beenden, und ESC wird den Befehl zu jeder Zeit abbrechen. Jedesmal wenn ein String gefunden wird, wird ein einzeiliges Listing des Speichers, beginnend mit dem ersten Byte des Strings, angezeigt.
FP uebergibt die Ausgabe an den Drucker.
Wildcards duerfen in einem String benutzt werden. Ein Wildcard ist ein Zeichen, das zu jedem Byte passt. Um ein Wildcard einzusetzen, tippen Sie '?'.
Beispiel fuer die Benutzung: Geben Sie Hex String CD,?,B9 ein. Dies wird alle uneingeschraenkten Unterroutinen Calls an Adressen zwischen &B900 und &B9FF auffinden.

M Move block
Sie werden nach 3 Eingaben gefragt; Anfangs- und Endadresse um einen Memoryblock zu definieren, und die Adresse zu der Sie den Block kopieren wollen. Die obere und untere ROM Auswahl wird beruecksichtigt, auf diese Art koennen Bloecke von jedem ROM kopiert werden. Ueberlappende Bloecke sind erlaubt.

C Bloecke vergleichen
CP Vergleichen und Drucken

Die zwei spezifizierten Bloecke werden Byte fuer Byte verglichen. Irgendwelche Unterschiede werden aufgelistet, indem die zwei verschiedenen Werte angezeigt werden. Die ROM Auswahl wird beruecksichtigt, dadurch kann RAM mit ROM vergleichen werden.
CP sendet den Output zum Drucker.

Relocate block

Es gibt zwei Wege um den Relocator zu benutzen,einen einfachen und einen komplizierten. Um den einfachen zu benutzen, geben Sie als Antwort 'Y' zur Aufforderung 'Simple (Y/N)?'. Sie werden dann nach einer Anfangs-, Endadresse und einer Adresse gefragt, zu der Sie veschieben wollen. Das funktioniert auf die gleiche Art wie 'Move Block', ausser dass zur gleichen Zeit der Code versetzt wird um unter der neuen Adresse zu laufen.

Das ist in vielen Faellen gut, wird aber nicht immer arbeiten. Obgleich Sie wahrscheinlich nie das generelle Relocate Kommando benutzen muessen (wie unten beschrieben), ist es gut sich darueber im Klaren zu sein, dass Relocate fehlerhaft arbeiten kann. Der Grund dafuer ist, dass der Relocator den Unterschied zwischen Code und Data nicht erkennt und deshalb versucht Data zu versetzen, als sei es Programm.Der zu versetzende Memoryblock, sollte ausschliesslich reines Programm enthalten.

Die zweite Form des Kommandos fragt Sie nach fuenf Eingaben. Die Bedeutung von jedem ist wie folgt:

Start und Endadresse- definieren Sie einen Memoryblock, der den zu verschiebenden Code aufnehmen soll.

Erste und Letzte - definieren Sie den Bereich der Adressen,zu welchen die Referenzen geaendert werden sollen.

Beim einfachen Relocate sind diese die gleichen wie Start und End.

Offset - der Wert der allen Adressen im Bereich zuzufuegen ist (erste und letzte).

Mit dieser Form des Kommandos wird der Platz des Codes versetzt, und nicht bewegt. Das erlaubt viele Variationen: Sie koennen Code versetzen (relocate), welcher in den ROM eingehen soll, Sie koennen aber auch Code als Erstes bewegen (move) und anschliessend durch relocate verschieben um ihn an jeder beliebigen Stelle ablaufen zu lassen.

Warnung: Etwas vor dem man sich bei der Verwendung von Relocate in acht nehmen muss, ist eine Instruktion wie 'LD HL,& 3090', wo der Operand als Buchstabe gemeint ist. Der Relocator wird annehmen es sei eine Adresse und wird sie dann moeglicherweise aendern. Wenn der Code versetzt werden soll, so ist die Instruktion durch 'LD H,&30:LD,&90' zu ersetzen.

I Blockinitialisierung

Ein Memoryblock kann mit jedem Byte-Wert aufgefuellt werden. Dies ist sehr hilfreich,wenn mann einen Block mit Nullen auffuellen will.

X Externe Kommandos

Wenn Sie 'X' anwaehlen, wird ein '*' als Prompt angezeigt. Jedes externe Kommando kann nun mit seinen Parametern eingegeben werden. Dies vermeidet die umstaendliche Syntax um externe Kommandos von BASIC aus zu benutzen.

Beides, Strings und numerische Parameter koennen eingegeben werden. Die Einzelheiten eines jeden Kommandos muessen in der zugehoerigen Dokumentation geprueft werden, um zu sehen welche Parameter erforderlich sind. Strings koennen durch Anfuehrungszeichen eingeschlossen werden, jedoch bleibt dies freigestellt. Beginnt jedoch der String mit einem dezimalen Wert, so sind die Anfuehrungszeichen erforderlich, da andernfalls davon ausgegangen wird, dass es sich um numerische Parameter handelt. Parameter lassen sich durch Leerzeichen, Kommas oder gleichwertige Zeichen trennen. Hexzahlen muessen genau wie in BASIC durch ein vorgestelltes '&' gekennzeichnet werden. Zum Beispiel um ein File mit dem Namen JUNK.BAS umzubenennen in PROG.BAS geben Sie eines des folgenden ein:

```
REN PROG.BAS=JUNK.BAS
REN "PROG.BAS","JUNK.BAS"
REN 'PROG.BAS'junk.bas
```

Nach der Durchfuehrung eines jeden Kommandos erscheint wieder das '*' Prompt. Entweder geben Sie ein weiteres Kommando ein oder kehren durch Druecken von ESC in das Menue zurueck.

Verschiedene externe Befehle

MODE schaltet den Bildschirm zwischen MODE 1 und MODE 2 (40/80 Spalten)

HELP Das Eintippen von 'HELP' im externen Befehlsmodus, oder '|HELP' von BASIC wird eine Liste von allen ROMs mit ihren Versionsnummern produzieren.

HELP,n Hier ist n eine ROM Nummer, wie aufgelistet von HELP. Die externen Befehle, die von ROM n zur Verfuegung gestellt werden, werden aufgelistet.

ROMOFF
Es wird ein Reset durchgefuehrt, welcher den Speicherinhalt loescht. Es werden alle Hintergrund-ROMs abgeschaltet. Es erfolgt ein Einsprung zum BASIC, wobei aber keine anderen ROMs initialisiert sind. Dies kann hilfreich sein um zusaetzlichen Speicherplatz zu erhalten, vergessen Sie aber nicht, dass jedes Programm im Speicher geloescht wird.

ROMOFF (Liste der Rom Nummern)
Es erfolgt ein Reset, der Inhalt des Speichers wird geloescht und die spezifizierten ROMs werden nicht initialisiert.

MSL (move screen low) Dieser Befehl bewirkt, dass der Bildschirmspeicher bei &4000 anstatt bei &C000 beginnt. HIMEM muss vor &4000 gesetzt werden, bevor man MSL benutzt. Der Schirm ist geloescht.

MSH Bewege Bildschirm nach oben. Der Schirm ist wieder hergestellt bis &C000. Der Bildschirm ist geloescht.

Warum moechte jemand den Bildschirmspeicher bewegen?

Das Verschieben des Bildschirmspeichers nach &4000 reduziert den RAM auf beinahe 27k, dies ist keine gute Idee! Der Gebrauch von MSL ermoeglicht es Sideways RAM zu editieren.Sideways RAMs werden in &C000 plaziert, dies ist jedoch die normale Position des Bildschirmspeichers.
Der Sideways RAM kann zwar editiert werden, jedoch wuerden Sie gleichzeitig den Bildschirminhalt zerstoeren. Aus diesem Grund geben Sie vorher MSL und zum Abschluss MSH ein.

Der Text Editor

Um in den Text Editor zu gelangen geben Sie bitte folgendes ein :
↑MAXAM
T
Das Editor Menue wird dann angezeigt.Waehlen Sie Punkt 'E' um in den Editier Modus zugelangen.Durch druecken von ESC koennen Sie jeder Zeit ins Menue zurueckkehren.Wenn Sie in den Editier Modus gelangen ist der Bildschirm leer falls Sie noch kein Textfile geladen haben . Nur die oberste Zeile des Bildschirmes ist mit einigen Hinweisen belegt . Nun die Bedeutung der Hinweise von Links nach Rechts :

1. Cursor Position (Zeile,Spalte)
2. Freier Speicher den Sie zur Verfuegung von Textfiles haben.
3. Einfuegen / Ueberschreiben.Der Editor kann in diesen beiden Modis arbeiten .
   Um den Modus zu waechseln druecken Sie CTRL-TAB . Danach wird der neue Modus angezeigt und Sie koennen nun in ihm arbeiten .
4. Caps lock / shift lock status. Druecken Sie CAPS LOCK oder CTRL-CAPS LOCK und Sie koennen sehen das sich die Anzeige veraendert und nun den momentanen Status anzeigt .

In dieser Zeile werden auch Fehler und Meldungen angezeigt .

Die Text Eingabe ist sehr einfach . Sie brauchen nur den gewuenschten Text einzugeben . Um den Cursor zu bewegen brauchen Sie nur die Cursor Tasten zu benutzen . Durch druecken von SHIFT oder CTRL und eine Cursor Taste haben Sie noch nuetzliche Funktionen . Diese entnehmen Sie bitte aus der nachfolgenden Tabelle .
Um Zeichen zu loeschen koennen Sie die Tasten DEL und CLR benutzen . Sie haben die gleiche Wirkung wie im BASIC .
Wenn Sie ein Assembler Programm eingeben benoetigen sie keine Zeilennummern . Sie brauchen auch nicht den Befehl ↑ASSEMBLE .

Einfuegungs Modus

Wenn Sie Zeichen eingeben wird der Text der sich in dieser Zeile schon befindet um eine Stelle nach rechts gerueckt . Wenn Sie ENTER druecken wird der Text der rechts vom Cursor steht in die naechste Zeile gesetzt . Wenn Sie den Cursor am Anfang einer Zeile stellen und dann DEL druecken wird diese Zeile an die Zeile die sich ueber ihr befindet angehangen .

Ueberschreibungs Modus

In diesem Modus koennen Sie Zeichen die sich auf dem Bildschirm befinden einfach ueberschreiben . In diesem Modus wird durch den Druck der ENTER Taste kein Zeilenvorschub durchgefuehrt, sondern der Cursor geht auf den Beginn der naechsten Zeile . Um Zeilen einzufuegen oder zu loeschen druecken Sie einfach SHIFT-TAB b.z.w. SHIFT-DEL .

Horizontales Scrolling

Textzeilen koennen groesser sein als die Bildschirmbreite. Wenn eine Zeile laenger ist als die Bildschirmbreite, springt der Qursor automatisch zur naechsten Zeile und der Text wird dann ausgerichtet. Der Editor kontrolliert die Zeilenlaenge nicht. Eine Zeile darf aber nicht laenger als 255 Zeichen sein, weil es sonst beim Assembieren zu einer Fehlermeldung kommt .

Sprung zu einer Zeile

Durch das Druecken von CTRL-G erscheint in der obersten Bildschirm Zeile die Frage in welcher Zeile der Cursor gesetzt werden soll.Geben Sie nun die gewuenschte Zeilennummer ein und druecken Sie ENTER.Danach erscheint der Cursor in der gewuenschten Zeile .

Blockmarkierungen

Sie haben die Moeglichkeit Teile ihres Programms als Block zu markieren. Diese Bloecke koennen Sie kann verschieben oder an andere stellen des Programms kopieren . Block Anfang und Block Ende werden durch das Druecken von SHIFT-COPY markiert. Um eine Markierung zu entfernen druecken Sie SHIFT-CLR . Ein markierter Block wird invers dargestellt (ausser die Markierung am Blockende, sie bleibt normal). Falls Sie versuchen einen Block der nicht makiert ist zu verschieben ertoent ein heller Piep Ton, und in der ersten Bildschirmzeile erscheint eine Fehlermeldung. Nach einer Fehlermeldung muessen Sie ESC druecken um weiter machen zu koennen.

Block Befehle

CTRL-CLR verschiebe Block
Der Block wird auf die aktuelle Cursor Position verschoben wobei der alte Block geloescht wird. Nach dieser Verschiebung sind die Block Markierungen geloescht. Ein Block kann nur verschoben werden wenn an der Zielposition genuegend freier Speicher ist.

CTRL-COPY kopiere Block
Diese Funktion hat den gleichen Effekt wie CTRL-CLR, nur dass der alte Block nicht geloescht wird, sondern erhalten bleibt .

CTRL-DEL loesche Block
Diese Funktion loescht den markierten Block .

Die Tastenbelegung im Editier Modus

Taste	Normal	Mit Shift	Mit Ctrl
	Cursor links	Cursor zum Zeilen Anfang	Cursor zum Anfang des Textes
	Cursor rechts	Cursor zum Zeilen Ende	Cursor zum Ende des Textes
	Cursor hoch	Eine Zeile zurueck scrollen	Eine Seite zurueck scrollen
	Cursor runter	Eine Zeile weiter scrollen	Eine Seite weiter scrollen
COPY	suche weiter	setze/loesche Markierung	kopiere Block
CLR	loesche Zeichen auf Cursor Position	Loesche Markierung	verschiebe Block
DEL	loesche Zeichen vor dem Cursor	loesche Zeile	loesche Block
TAB	Cursor auf naechste Tab Position	fuege Zeile ein	Wechsel Einfuegungs / Ueberschreib Modus
ESC	Zurueck zum Menue	Zurueck zum Menue	Zurueck zum Menue

Das Editor Menue

L Lade File
Sie werden nach dem Filenamen gefragt . Beim laden von Kassette reicht es wenn Sie ENTER druecken um das naechste File von der Kassette zu laden .

LB Lade Block
Mit diesem Befehl koennen Sie ein Textfile mit einem anderem verbinden . Gehen Sie in den Editier Modus und stellen Sie den Cursor auf die Position, wo das zu ladende Textfile abgelegt werden soll.Danach gehen Sie wieder ins Menue zurueck und laden mit dem Befehl LB das anzuhaengende Textfile ein.

S Save File
Saven des gesamten Textfiles.

SB Save Block
Nur der Makierte Block wird gesaved . Wie man einen Block makiert koennen Sie auf Seite 35 lesen .

P Ausdrucken des Textfiles
Das gesamte Textfile wird auf dem Drucker ausgegeben .

PB Nur der Makierte Block wird auf dem Drucker ausgegeben .

Erstellen von Basic Kompatiblen Files

M Modifiziere Text
MB Modifiziere Block

Modifiziere ist ein spezieller Befehl der es ihnen ermoeglicht gemischte Basic und Maschinen Code Files zu erstellen .
Mit diesem Befehl haben Sie folgende Moeglichkeiten :

1. Das Textfile mit Zeilennummern zu versehen falls noch keine vorhanden sind .
2. Einfuegen von Systemdaten zwischen Zeilennummer und dem Text .
3. Alle Zeilennummern zu loeschen .
4. Alle Systemdaten zu loeschen falls welche vorhanden sind .

Um ein gemischtes File zu erstellen gehen Sie bitte folgendermassen vor :

(a)

1. Geben Sie im Editier Modus das Basic Programm ohne Zeilennummern und Systemdaten ein .
2. Makieren Sie den Assembler Teil als einen Block.
3. Gehen Sie ins Menue und waehlen Sie den Punkt 'M' an . Nun werden die Zeilennummern und die Systemdaten eingefuegt . Sie koennen den Zeilennummer Abstand selbstbestimmen.
   Bemerkung : Danach sind keine Block Makierungen mehr vorhanden .
4. Wiederholen Sie Punkt 2 und 3 fuer jeden weiteren Assembler Code .
5. Waehlen Sie wieder Punkt 'M' um nun das gesamte Programm durchzunummerrieren .
6. Saven Sie das Textfile

(b)

Wenn Sie in Punkt 'M' nach dem Zeilennummer Abstand gefragt werden koennen Sie einfach ENTER druecken und das Programm wird in einer Schritten durchnummerriert .

F Finde Ausdruck

Diese Funktion sucht nach einem von ihnen bestimmten Ausdruck im Textfile. Die Suche beginnt an der aktuellen Cursorposition. Wenn der Ausdruck gefunden wurde, steht der Cursor am Ende des Ausdrucks. Um nach dem gleichen Ausdruck weiter zu suchen, druecken Sie COPY. Falls der Ausdruck nicht gefunden wird, erscheint eine Meldung in der obersten Bildschirmzeile. Nach dieser Meldung muessen Sie ESC druecken um weiter machen zu koennen.

R Finde und aendere Ausdruck

Diese Funktion sucht nach einem Ausdruck und aendert ihn. Zuerst werden Sie nach zwei Ausdruecken gefragt, wobei der erste der alte Ausdruck und der zweite der neue Ausdruck ist. Sobald der zu suchende Ausdruck zum erstenmal gefunden wird, erscheint die Frage ob er geaendert werden soll . Es ist aber zu beachten, dass bei mehrfachem Vorhandensein des zu suchenden Ausdrucks diese automatisch mit geaendert werden .

T Setzen und loeschen von Tabulatoren

Sie koennen bis zu 8 Tabulatoren setzten. Nachdem Sie die Tabulatoren gesetzt haben, koennen Sie im Editier Modus durch das Druecken von TAB den Cursor auf die naechste Tabulator-Position setzten. Im Einfuegungsmodus wird in den Text rechts vom Cursor eine bestimmte Anzahl von Leerzeichen eingefuegt. Vom Programm aus sind schon 3 Tabulatoren defininiert (position 9,17,27).
Der Befehl 'T' hat die folgenden 3 Funktionen :
1. Auflistung aller Tabulator-Positionen.
2. Setzten der Tabulator-Positionen.
3. Loeschen von Tabulator-Positionen.
Um die Eingabe in Punkt 2 oder 3 abzubrechen, druecken Sie ESC.

Assemblieren und Testen von Maschinencodeprogrammen

A Assembliere

Der momentane Text wird assembliert. Um nach dem Assemblieren wieder ins Menue zu gelangen, druecken Sie eine Taste.

J Starte Maschinenprogramm

Sie werden zuerst nach der Startadresse gefragt. Sobald das Programm beendet ist, springt der Computer wieder zurueck ins Menue. Falls Sie von BASIC aus schon einen MEMORY Befehl gesetzt haben, und ihr Programm eine Speicherstelle hinter HIMEM liegt, brauchen Sie keine Startadresse anzugeben .
Dadurch koennen die meisten Codes durch einfaches Eingeben von J ‹ENTER› ‹ENTER› ausgefuehrt.

Wichtiger Hinweis: Speicherplatz ist durch den MEMORY-Befehl in BASIC zu reservieren.

Fehlersuche

Um Fehler in einem Programm zu suchen ist der BRK Befehl sehr nuetzlich . Wenn Sie ein Programm geschrieben haben doch dieses Fehlerhaft sein sollte gehen Sie am besten folgendermassen vor .

1. Editieren Sie das Textfile und fuegen Sie an wichtigen Punkten BRK Befehle ein.
2. Assemblieren Sie das Programm nun wieder .
3. Starten Sie das Programm nun mit dem 'J' Kommando .
4. Untersuchen Sie die Register nun wenn sie Angezeigt werden . Falls Sie dort einen Fehler finden druecken Sie ESC und beginnen Sie wieder bei Punkt 1 .
5. Wenn die Register korrekt sind druecken Sie eine Taste um weiter zu machen .
6. Durch druecken von CTRL gelangen Sie wieder zurueck ins Menue .

Vielseitige Befehle

G Editiere Zeile
Mit diesem Kommando koennen Sie nach dem Assemblieren einzelne Zeilen Editieren .

X RSX Befehl ausfuehren
Mit diesem Kommando koennen Sie vom MAXAM aus direkt RSX Befehle ausfuehren . Es gibt zwei Befehle die im Editor sehr nuetzlich sind :
|CAT erzeugt einen Katalog von Cassette oder Disc .
|SPEED setzen der Schreibgeschwindigkeit (0 oder 1) .
Unter AMSDOS sind diese Befehle sehr nuetzlich :
|ERA um ein File zu loeschen .
|REN um ein File umzubennenen .

Selbstdefinierte Zeichen

Die Zeichen mit dem ASCII Wert 32-127 koennen vom Benutzer selbstdefieniert werden . Der Editor benutzt diese Zeichen aber nur wenn man im Basic einen SYMBOL AFTER 0 Befehl gemacht hat .

Der Text Editor und Basic

Der Text Editor ist kompatibel zum BASIC. Editor Text und BASIC koennen zur gleichen Zeit im Speicher sein . BASIC Variablen werden nicht beinflusst solange man nicht mit dem Text Editor arbeitet . Diese kompatibilitaet hat nur den Nachteil, dass wenn man ein BASIC Programm saved der Editor Text mit abgespeichert wird . Dieses kann man aber verhindern, indem man den Editor Text mit dem Befehl !CLEAR loescht bevor man das BASIC Programm speichert .
Man kann aber auch das Basic Programm als ASCII FILE abspeichern , dadurch wird der Editor Text nicht mit gespeichert . Dieses ASCII FILE kann man dann auch direkt mit dem Editor aendern . Diese Moeglichkeit hat nur den Nachteil das dieses ASCII FILE mehr Speicher verbraucht und sich dadurch auch die Ladezeit erhoeht .

Befehle um Programm oder Text zu loeschen:

1. Um das Text File und das Basic Programm zu loeschen geben Sie den Befehl NEW ein.
2. Um nur das Basic Programm zu loeschen benutzen Sie den Befehl DELETE.
3. Um nur das Text File zu loeschen benutzen Sie den Befehl !CLEAR.

Das Laden von Basic Programmen

Durch das Laden von Basic Programmen mit dem LOAD Befehl wird der Editor Text geloescht . Dieses kann man aber umgehen wenn man das BASIC Programm ueber MERGE einlaedt.

Um ein BASIC Programm mit dem Text Editor zu editieren, gehen Sie bitte folgendermassen vor :

1. Speichern Sie das BASIC Programm im ASCII Format:'SAVE "NAME",A '
2. Gehen Sie in den Text Editor .
3. Laden Sie das File in den Editor ein. : L <ENTER> NAME
4. Editieren Sie das File .
5. Saven Sie das File nun wieder ab . : S <ENTER> NAME
6. Gehen Sie zurueck ins BASIC .
7. Laden Sie das File wieder ein . : LOAD "NAME"

9. Anhang

(a) Bibliographie

1. "The concise Firmware specification",(Amsoft)
2. "The concise Basic specification",(Amsoft)
3. "Disc drive Firmware specification",(Amsoft)
4. "Programming the Z80",Rodnay Zaks (Sybex)
5. "Amstrad Machine language for the absolute beginner",(Melbourne)
6. "The Lord of the Rings",J.R.R. Tolkein (George Allen Unwin)

(b) Assembler Befehle

Name	Bedeutung	Seite
BYTE	fuege String in Objekt Code	16
CLOSE	schliesse Objekt File	22
CODE	streiche NOCODE	14
DB	wie BYTE	16
DEFB	wie BYTE	16
DEFM	wie BYTE	16
DEFS	wie RMEM	16
DEFW	wie WORD	16
DS	wie RMEM	16
DW	wie WORD	16
ELSE	sonst assembliere	20
END	Ende der Assemblierung	14
ENDIF	Ende des IF Blocks	20
EQU	setze gleich	15
GET	hole Parameter	17
IF	assembliere wenn	20
IFNOT	assembliere wenn nicht	21
IF1	assembliere wenn 1.Durchlauf	21
IF2	assembliere wenn 2.Durchlauf	21
LET	definiere Symbol	15
LIMIT	setze obere Speichergrenze fuer Ob.Code	13
NOCODE	assembliere aber speicher Ob.Code nicht	14
ORG	setze Startadresse fuer Objekt Code	13
PUT	gebe Ausdruck ans BASIC zurueck	18
READ	definiere Eingabe File	22
RMEM	definiere einen Speicherbereich	16
TEXT	wie BYTE	16
WORD	fuege 2 byte Zahl in den Ob.Code	16
WRITE	definiere Objekt File	22

(c) Assembler Kommandos

Name	Bedeutung	Seite
DUMP	Auflisten aller definierten Symbole	23
LIST (P)	Ausgabe des Listings auf Drucker/Monitor	23(25)
NOLIST	Listing nicht auf Monitor ausgeben	23
PAGE	beginne neue Seite	25
PAUSE	warte auf Tastendruck	23
PLEN	definiere Seitenlaenge fuer Drucker	25
PRINT	drucke String auf Bildschirm aus	23
TITLE	definiere Ueberschrift	25
WIDTH	setze Druckbreite fuer Drucker	25

(d) Assembler Fehler

1. ORG Anweisung mit ungueltigem Ausdruck
2. EQU Anweisung mit ungueltigem Ausdruck
3. RMEM Anweisung mit ungueltigem Ausdruck
4. IF oder IFNOT Anweisung mit ungueltigem Ausdruck
5. PUT Anweisung mit undefinierter Variablen Adresse
6. Ein falsch verschachtelter IF Block .
7. Eine Zeile die laenger als 255 Zeichen ist
8. Der Assembler hat keinen Speicher mehr fuer die Symbol Tabelle
9. Bei einer READ Anweisung wurde ein falscher Wert angegeben z.B. eine Binaer Zahl
10. Es wurde versucht Read Anweisungen zu verschachteln
11. Disc Fehler z.B. 'disc full' oder 'file not found'
12. Es wurde versucht den Code an einer Adresse abzulegen die durch eine Limit Anweisung begrenzt wurde

(e) RSX Komandos

Name	Bedeutung	Seite
ASSEM	assembliere text	10
ASSEMBLE	assembliere vom BASIC aus	9
CAT	Katalog	39
CLEAR	loesche Editor Text	40
FIND	suche String im Text File	10
HELP	listet alle ROMs auf	33
MAXAM / M	starte MAXAM	30,6
MAXOFF	schalte MAXAM aus	5
MCLEAR	wie CLEAR	6
MFIND	wie FIND	6
MHELP	wie HELP	6
MODE	schaltet Bildschirm Modus um	33
MSH	lege den Bildschirm in hohen Speicherbereich	33
MSL	lege den Bildschirm in unterem Speicherbereich	33
ROMOFF	schaltet ROMs aus	33
SPEED	setze Schreibgeschwindigkeit	39

(f) Wichtige Basic Befehle

Name	Bedeutung	Seite
CALL	ruft ein Maschinensprache Programm auf	19
MEMORY	ermoeglicht den HIMEM zu aendern	12

(g) Hauptmenue Kommandos

Name	Bedeutung	Seite
T	gehe in den TEXT EDITOR	34
D	disassembliere auf dem Bildschirm	30
DP	disassembliere auf dem Drucker	30
L	liste Speicher auf dem Bildschirm aus	30
LP	liste Speicher auf dem Drucker aus	30
S	schalte oberes ROM ein	30
O	schalte unteres RAM ein bzw. aus	30
E	Speicher Editor	31
F	suche Ausdruck	31
FP	suche Ausdruck und gebe ihn auf dem Drucker aus	31
M	verschiebe Speicher Block	31
R	verschiebe Speicher Block	32
I	fuelle Speicher Block	32
C	vergleiche Speicher Bloecke	31
CP	vergleiche Speicher Bloecke und gebe auf Drucker aus	31
X	RSX Kommando	32
B	kehre zurueck ins BASIC	30

(h) Editor Menue

Name	Bedeutung	Seite
E	gehe in den Editier Modus	34
L	lade File	36
LB	lade File und fuege es an die aktuelle Cursor Position an	36
S	speicher Text	37
SB	speicher Block	37
P	gebe Text File auf Drucker aus	37
PB	gebe Block auf Drucker aus	37
M	modifiziere Text	37
MB	modifiziere Block	37
G	editiere Zeile	39
F	suche Ausdruck	38
R	suche Ausdruck und ersetze ihn	38
T	setze und loesche Tabulatoren	38
A	assembliere Text	38
J	starte Maschinen Sprache Programm	38
X	RSX Kommando	39
Q	verlasse Editor	39

(i) Z80 Befehle

Mnemonic	Name	Format		
ADC	Addition mit Carry	A,n	A,r	HL,rh
ADD	Addition	A,n	A,r	HL,rh
		IX,rx	IY,ry	
AND	Und mit A	A,n	A,r	
BIT	Teste Bit	b,r		
BRK	MAXAM Breakpoint	-		
CALL	Rufe Unterroutine auf	nn	cc,nn	
CCF	Complementiere Carry Flag	-		
CP	Vergleiche mit A	A,r		
CPD	Vergleiche und erniedrige	-		
CPDR	Vergleiche Block und erniedrige	-		
CPI	Vergleiche und erhoehe	-		
CPIR	Vergleiche Block und erhoehe	-		
CPL	Vergleiche A	-		
DAA	Dezimale Anpassung von A	-		
DEC	Erniedrige	r	rr	
DI	Sperre Interrupt	-		
DJNZ	Erniedrige B und Springe wenn nicht Zero	e		
EI	Erlaube Interrupt	-		
EX	Tausche Register aus	AF,AF'	DE,HL	(SP),ra
EXX	Tausche alternativ Register aus	-		
HALT	Halte CPU an	-		
IN	Eingabe	r,(C)		
INC	Erhoehe	r	rr	
JP	Springe	nn	cc,nn	(ra)
JR	Springe Relativ	e	c,e	
LD	Lade	r,n	r,s	s,r
		A,(nn)	A,(BC)	A,(DE)
		(nn),A	(BC),A	(DE),A
		rr,nn	rr,(nn)	(nn),rr
		A,I	A,R	
		I,A	R,A	
LDD	Lade und erniedrige	-		
LDDR	Lade Block und erniedrige	-		
LDI	Lade und erhoehe	-		
LDIR	Lade Block und erhoehe	-		
NEG	Negiere A	-		
NOP	Tue nichts	-		
OR	Oder mit A	A,r		
OUT	Ausgabe	(C),r		
POP	Bringe Register auf Stapel	rp		
PUSH	Hole Register vom Stapel	rp		
RES	Setze Bit auf 0	b,r		

Mnemonic	Name	Format
RET	Return	-
RETI	Return aus Interrupt	-
RL	Rotiere links	r
RLA	Rotiere A links	-
RLC	Rotiere links mit Carry	r
RLCA	Rotiere A links mit Carry	-
RLD	Rotiere links aber Dezimal	-
RR	Rotiere rechts	r
RRA	Rotiere A rechts	-
RRC	Rotiere rechts mit Carry	r
RRCA	Rotiere A rechts mit Carry	-
RRD	Rotiere rechts aber Dezimal	-
RST	Restart	Siehe naechste Seite
SBC	Subtraktion mit Carry	A,r HL,rh
SCF	Setze Carry Flag	-
SET	Setze Bit	b,r
SLA	Geshiftete linke Arithmetik	r
SRA	Geshiftete rechte Arithmettik	r
SRL	Geshiftete rechte Logische Arithmetik	r
SUB	Subtraktion von A	A,r
XOR	Exklusiv Oder mit A	A,r

Schluesselwoerter :

r Bedeutet eines der Register von A,B,C,D,E,H,L,(HL),(IX+D),(IY+D)
s Bedeutet eines der Register von A,B,C,D,E,H,L
d Bedeutet eine Integer Zahl von -128 bis +127
rr Bedeutet eines der Register von BC,DE,HL,SP,IX,IY
rh Bedeutet eines der Register von BC,DE,HL,SP
rx Bedeutet eines der Register von BC,DE,IX,SP
ry Bedeutet eines der Register von BC,DE,IY,SP
rp Bedeutet eines der Register von BC,DE,HL,AF,IX,IY
ra Bedeutet eines der Register von HL,IX,IY
n Bedeutet ein einzelnes Byte
nn Bedeutet zwei Bytes oder eine Adresse die sich aus zwei Bytes zusammen setzt
b Bedeutet eine Bit Nummer zwischen 0 und 7
e Bedeutet eine Adresse im Bereich von $-126 und $+129 wobei $ die Adresse des Momentanen Befehles ist
cc Bedeutet eine der Bedingungen von C,NC,Z,NZ,M,P,PE,PO
c Bedeutet eine der Bedingungen von C,NC,Z,NZ
- Bedeutet das kein Operand noetig ist

Bemerkung :
(i) Bei den folgenden Befehlen kann das erste Parameter weggelassen werden :
A:ADC ADD AND CP OR SBC SUB XOR
Zum Beispiel sind 'OR B' und 'OR A,B' gleich .
(ii) Die folgenden Befehle sind bei AMSTRAD weggelassen worden :
IM,IND,INDR,INI,INIR,OTDR,OTIR,OUTD,OUTI,RETN

Der RST Befehl

Der AMSTRAD CPC 464/664/6128 benutzt den RST Befehl um Befehlssatz zu erweitern. Einige der RST Befehle benoetigen Parameter . Der Assembler erlaubt aber auch RST Befehle ohne Parameter Angabe , dieser RST Befehl verbraucht dann auch nur ein Byte .
Um genaueres ueber den RST Befehl zu erfahren sehen Sie bitte im 'Firmware Handbuch' von Amsoft nach .

```
RST 0          Zuruecksetzen des Rechners (Reset)
RST 1,nn       Low jump
RST 2,nn       Side call
RST 3,nn       Far call
RST 4          LD A,(HL) mit allen verfuegbaren ROMs
RST 5,nn       Firm jump
RST 6          BRK-MAXAM BREAKPOINT (ruft die Register Anzeige auf)
RST 7          Interrupt
```

(j) Disassembierung als ASCII File

Starten Sie das folgende Basic/Assembler Programm

```
10 MEMORY HIMEM-11
20 |ASSEMBLE
30 'patch push ix: push hl: call &BC95; CAS OUT CHAR
40 'pop hl: pop ix: scf: ret
50 'limit &FFFF: org &BD2B: jp patch ; redirect MC PRINT CHAR
60 OPENOUT "FILE"
70 |MAXAM
```

Dieses Programm leitet die Druckerausgabe um , das heisst das nun alle MAXAM Kommandos die normalerweise die Daten auf dem Drucker ausgeben die Daten nun auf Kassette ausgeben bis die Ausgabe durch CLOSEOUT geschlossen wird . Brechen Sie diese Ausgabe nicht durch ESC ab .

Das folgende Programm listet alle Zeichen vor dem Disassemblierten Feld auf .

```
100 OPENIN "file"
110 OPENOUT "newfile"
120 WHILE NOT EOF
130 LINE INPUT #9,A$:PRINT #9,MID$(A$,26)
140 WEND
150 CLOSEIN:CLOSEOUT
```

(k) Erklaerung der Ausdruecke

ADRESS
Eine Zahl die eine Position im Speicher angibt.

ARNOR
'Das Land der Koenige'. Im dritten Zeitalter war Arnor bekannt als das 'Verlorene Koenigreich des Nordens'. Das Koenigreich wurde von Elessar nach dem 'Krieg um den Ring' wiedergegruendet.

ASCII (American Standard Code for Information Interchange)
1. Der ASCII Code wird vom Computer zur Darstellung von Zahlen, Buchstaben usw. benutzt.
2. Er dient zur Darstellung von Programmen ohne dabei spezielle 'TOKENS' zu benutzen.

ASSEMBLER
1. Ein Programm das Mnemonics in binaeren Maschinencode uebersetzt.
2. Ein anderer Name fuer 'ASSEMBLY LANGUAGE'.

ASSEMBLY LANGUAGE
Der Mnemonics Befehlssatz der es dem Programierer erlaubt sich mit dem Z80 zu verstaendigen.

BINARY
Das Zahlensystem das nur mit den Ziffern 1 und 0 arbeitet.

BIT
Ein Teil einer binaeren Zahl (0 oder 1).

BREAKPOINT
Eine Hilfe zur Fehlersuche. Ein Programm unterbricht seinen Ablauf an einem BREAKPOINT um den User zu ermoeglichen einen Fehler zu finden.

BYTE
Ein Byte besteht aus 8 Bits. Byte ist die gebreuchliche Einheit der Datenuebertragung.

CODE ORIGIN
Die Adresse wo ein Maschinen Programm anfaengt.

CODE LOCATION
Die Adresse des naechsten abzulegenden Bytes waehrend der Assemblierung.

KOMMANDO
1. Ein Befehl der den Assembler in irgend einer Weise beeinflusst.
2. Ein Befehl der BASIC oder MAXAM zu irgend etwas auffordert.

DELIMITER

Ein Zeichen das dem Assembler mitteilt wo ein String beginnt und endet .

DIRECTIVE

Ein Befehl der den Assembler dazu veranlasst den Opcode in irgend einer Weise zu beeinflussen .

DISASSEMBLE

Setze binaercode in Mnemonics um .

ENTRY POINT (EINSPRUG PUNKT)

Die Adresse ab wo ein Maschinen Programm gestartet werden soll .

EPROM (Erasable Programmable Read Only Memory)

Ein EPROM ist ein Elektrisch programmierbarer Lese Speicher .

EXTERNAL COMMAND (RSX Befehl)

Ein Softwaremaessig erzeugter Befehl .

FIRMWARE

Fest gespeicherte Programme im ROM .

HEXADEZIMAL (HEX)

Zahlen System mit 16 Ziffern . Die Ziffern 10-15 werden durch die die Buchstaben A-F ersetzt .

IDENTIFIER

Ein String der den Namen einer SYMBOL Variable angibt .

INSTRUCTION

Ein Befehl, Kommando oder eine Anweisung im Assembler File .

LABEL

Ein oder mehrere Zeichen die eine Position im Assembler File angeben .

LISTING

Die Schriftliche Ausgabe eines Programmes auf Bildschirm oder Drucker .

LOWER ROM

Das untere ROM das bei Adresse 0 beginnt und bei &3FFF endet .

MASCHINE CODE

Eine Zusammenstellung von Binaerzahlen die der Computer als Programm anerkennt .

MARKER

Im Text Editor, eine markierung im Text File die zu verschiedenen Zwecken dient .

MNEMONIC

Eine Zeichen Kette die der Assembler in Maschinen Code umsetzt .

OBJEKT CODE

Ein Maschinen Programm das durch einen Assembler erzeugt und in den Speicher gelegt wurde .

OPCODE

Eine binaere Zahl die einen Z-80 Befehl darstellt .

OPERAND

Meist eine Zahl die einen Befehl auf eine Adresse hinweist .

OPERATING SYSTEM

Ein Maschinen Programm das die Hardware direkt anspricht und durch einen Aufruf in einer Sprungtabele ausgefuehrt wird .

RAM (Random Access Memory) / (Schreib, Lese Speicher)

Dort werden die Programme gespeichert .

REGISTER

Eine 1 oder 2 byte lange 'Variable' in der dem Z-80 Nachrichten uebergeben werden . Die meisten Befehle benoetigen Register .

RELOCATE

Verschieben und aendern der Sprung Adressen , so das es auch an einer anderen Speicherstelle lauffaehig ist .

ROM (Read Only Memory) / (Lese Speicher)

Ein Speicher aus dem man nur lesen kann . Der AMSTRAD hat ROM von 32K .

SIDEWAYS RAM

Ein Erweiterungs RAM das im Bereich von &C000 bis &FFFF liegen kann .

SIDEWAYS ROM

Ein Erweiterungs ROM wie MAXAM, AMSDOS, PROTEXT, UTOPIA .

SOURCE CODE

Ein Assembler Programm `, bestehend aus mnemonics, Befehlen, Kommandos .

STORAGE LOCATION

Die Speicherstelle an der das naechste zu Assemblierende Byte abgelegt wird (wie Code Location) .

STRING

Ein Ausdruck der aus mehreren Zeichen besteht .

SYMBOL

Eine vom Assembler benutze Variable .

SYMBOL TABLE

Die Liste der vom Assembler benutzen Variablen .

TOKENISATION

Eine Zahl die der BASIC INTERPRETER in einen BASIC BEFEHL umsetzt . Dieses spart Speicher und Rechenzeit .

UPPER ROM (Hoeheres Rom)

Ein Rom das zwischen &C000 und &FFFF liegt wie z.B. :
BASIC, AMSDOS, MAXAM, PROTEXT, UTOPIA

Z-80 (CPU)

Die Zentrale Rechen Einheit des AMSTRAD CPC 464/664/6128 .

NACHWORT :
Wir danken Georges Allen und Urwin das wir Auszuege aus dem Buch 'Herr der Ringe' benutzen duerfen .