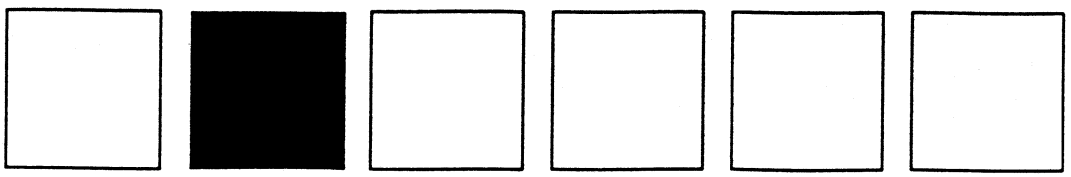
ABBEMBLER

HANDBUCH

HP-75



Autor: Michael Hartmann

Einleitung

Durch dieses vor Ihnen liegende Handbuch über die Maschinensprache des HP-75 wird die Programmierung Ihres Rechners nach Ihren Anforderungen erheblich vereinfacht. Denn es ist nun möglich, LEX-Files, mit denen der Befehlssatz des Rechners erweitert werden kann, mit Hilfe von Mnemonics so einzugeben, daß man sie bequem bearbeiten, listen, kopieren,.und noch vieles mehr machen kann.
Das Handbuch gliedert sich in 3 Hauptteile:

- Im ersten Teil wird der logische Aufbau des Rechners erklärt. Dazu gehören die Erklärung von Arithmetik, sowie Registerverwaltung, und noch einiger zum Verstehen der logischen Zusammenhänge wichtiger Einheiten.
- Im zweiten Teil erhält der Leser nun schrittweise die Fähigkeit, sich selbst LEX-Files herzustellen.
- Im dritten Teil schließlich wird das wichtigste Werkzeug dieser neuen Programmierung ausführlich besprochen: der Assembler.

Das Ganze wird zum Schluß noch abgerundet durch den Anhang, in dem noch viele weitere Überraschungen versteckt sind.
Doch nun möchte ich nicht zuviel verraten, sondern sie auffordern, diesen Exkurs ins Reich der Nullen und Einsen von Beginn an über sich ergehen zu lassen.

St.Wendel, den 4.11.1984 Der Verfasser

INHALT

1. Kapitel

Das Innenleben des HP-75

1. Die Zentraleinheit (kurz: CPU)
   Der Prozessor des HP-75 ist der gleiche, der schon im HP-85 Verwendung gefunden hat. Der Unterschied besteht nur darin, daß dem Prozessor des HP-75 nun CMOS-Speicherbausteine unterstellt sind und nicht herkömmliche Speicher-ICs wie beim HP-85. Durch diese CMOS-Technik wurde auch die Größe des Rechners erheblich vermindert.
   Dieser Prozessor steuert den 8-bit Systembus, der sämtliche Daten, Befehle und Adressen zu dem jeweilig zuständigen IC leitet. Die Frequenz, dh. den Takt, für diesen Bus liefert der Clock-Teil der CPU. Über diese Takteinehit wird jedoch noch unter Punkt 4 genauer berichtet.
   Nun jedoch, wie arbeitet diese CPU ?
   Die Central Processing Unit (=CPU) besteht aus 64 8-bit-Registern, einem Pointer für das Adreßregister (=ARP), einem Pointer für das Register, von dem Daten entnommen werden (=DRP), der Arithmetikeinheit, in der sämtliche Grundoperationen ausgeführt werden (=Addition, Subtraktion, Verschiebung,..), einem Shift-Register und einer Reihe von Statusanzeigen (=Flags).
   Die 64 8-bit-Register sind aufgeteilt in 2 Gruppen:
   Die ersten 32 Register (Nummern 00-40 im Oktalsystem) sind jeweils 2-Byte lang und werden ausnahmslos von der CPU zur Speicherung von Adressen genutzt. Sie erfüllen auch bestimmte Zwecke zur Kommunikation mit anderen Teilen (zB. Stackpointer, ...). Zu diesen Registern hat die CPU direkten Zugang.
   Die nächsten 32 Register (Nummern 41-77 im Oktalsystem) sind voneinander nur schwer zu trennen. So werden beispielsweise bis zu 8 Register zu einer Gruppe (dh. zu einem Multibyte) zusammengeschlossen. Um diese abgeschlossene Einheit von anderen Single- oder Multibytes abzugrenzen, benutzt der Rechner eine interne Grenze zwischen jeweils 8 Bytes. Nur in diesen Registern können Fließkommazahlen gespeichert werden.
   Irgendein Register der CPU kann als Akkumulator benutzt werden, wenn eine Operation ausgeführt werden soll. Auf dieses Register zeigt dann der DRP. In diesem Register liegt auch nach einer Operation das Ergebnis.
   Die CPU enthält 8 Flags und ein 4-bit Register für den Programmstatus. Dieser Status kann auch von der CPU beeinflußt werden.
   Instruktionen der CPU:
   - Arithmetik:
     Addition, Subtraktion, Rechts- und Linksverschiebung.
   - Nicht-Arithmetik:
     Laden, Speichern, Vertauschen, logische Operationen (AND, OR,..), Löschen, Testen und Shiften.

   Flags:
   - DCM (Dezimal-Mode-Flag): Dieses Flag gibt den momentanen Rechenmodus der CPU an: 0 für binär, 1 für BCD.
   - CY (Carry-Flag): Dieses Flag ersetzt bei einer Addition zweier Zahlen das 9.Bit.
   - OV (Overflow): Eine negative Zahl wird durch eine 1 im höchsten BIt des Registers dargestellt. Diese 1 kann aber auch bei einer Addition von zwei großen positiven Zahlen vorkommen. Deshalb gibt es ein Flag, das angibt, welcher Fall nun vorliegt: 1 falls positiv
     o falls negativ.
   - OD (letztes bestimmendes Bit): OD=0 für gerade Zahl, OD=1 für ungerade Zahlen.
   - NG (erstes bestimmendes Bit): NG=1 für negative Zahlen, NG=0 für positive Zahlen.
   - ZERO (Nullregister): Dieses Flag hat den Wert 1, falls das Datenregister leer ist, sonst 0. Wichtig bei Vergleichen und Sprüngen.
   - LDZ (linkes Byte gelöscht): Sind die ersten 4 Bit eines Registers 0000, dann hat LDZ den Wert 1, sonst 0.
   - RDZ (rechtes Byte gelöscht): Sind die letzten 4 Bit eines Registerblocks (dh. Byte oder Multibyte) leer (dh. den Wert 0000), hat RDZ den Wert 1, sonst den Wert 0.

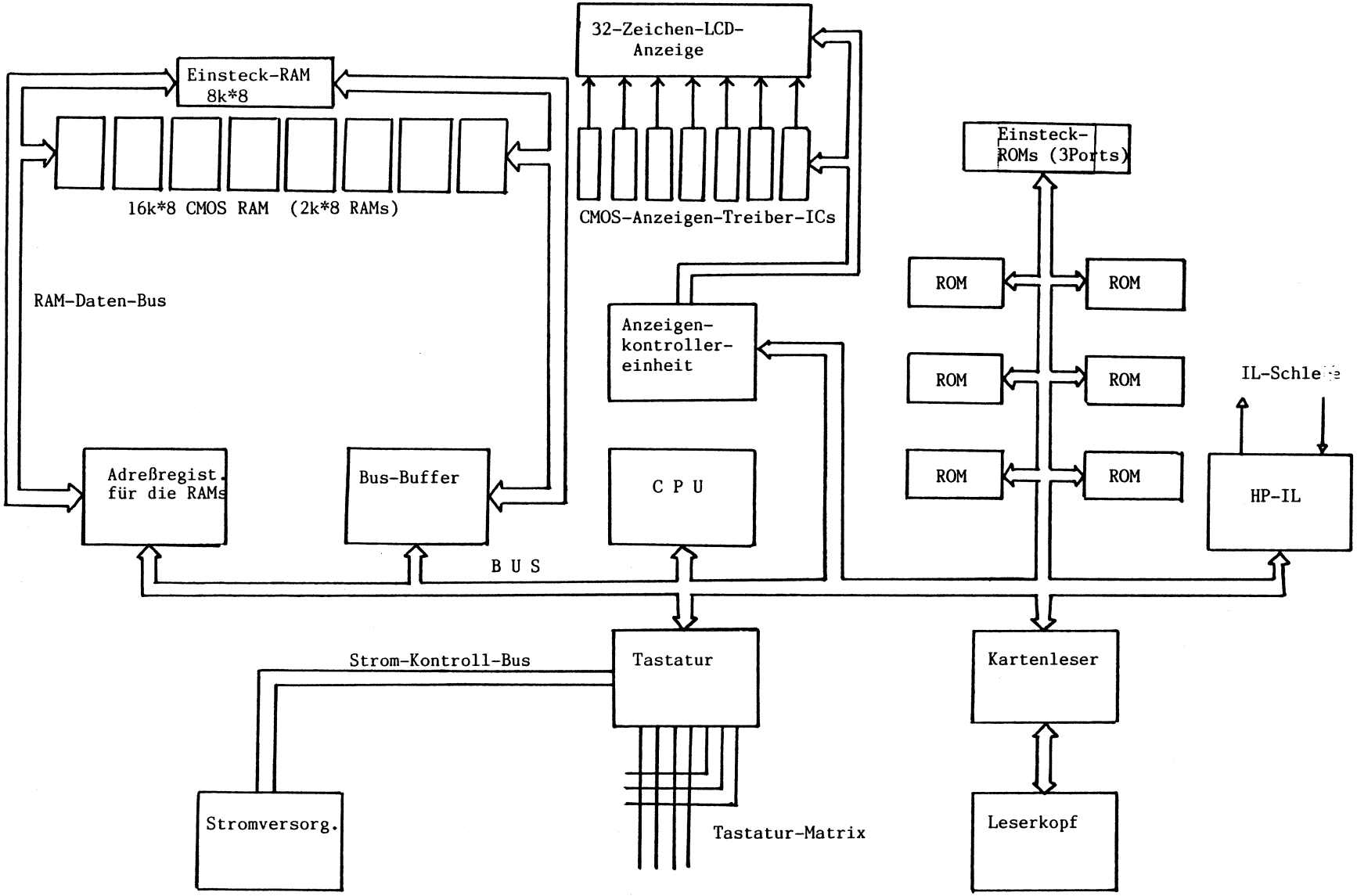


Abb. 1: Blockschaltbild des HP-75C

- E (Zusatzregister): Dieses Register arbeitet im BCD-Modus. Es kann durch einfache Assemblerbefehle vergrößert (Increment), verkleinert (Decrement) und gelöscht werden. In ihm werden die angezeigten Zeichen nach einer Verschiebung aufbewahrt.

Als nächstes ein Wort zu den festgelegten Registern:

R00,R01 Registerbankzähler: Diese beiden Register geben den Rest der CPU-Registerbank an.

R02,R03 Diese beiden Register werden bei der Berechnung von Adressen benötigt wenn mit Indices adressiert wird.

R04,R05 In diesen beiden Registern steht die absolute Adresse des nächsten Befehls.

R06,R07 In diesen beiden Registern wird bei einem Sprung in ein Unterprogramm die Adresse des Befehls festgehalten, bei dem dieser Sprung erfolgte. Das Programm fährt dann mit dem folgenden Befehl fort.

R08 Enthält den Programm-Counter bei der Ausführung von BASIC-Programmen.

R09 Dieses Register enthält im Parse-Modus (=Zerlegungsmodus) die Adresse des Zerlegungsausgabestacks.

R10,R11 Im RUN-Modus nicht softwaregesteuert. Im PARSE-Modus zeigen diese beiden Register auf das nächste Zeichen eines ASCII-Strings.

R12,R13 Operations-Stack: Parameter und Ergebnisse werden in das Register geleitet, das durch diese beiden Register festgelegt ist.

R14 Beim Zerlegen enthält R14 das aktuelle Zeichen, das verändert werden soll.

R16 Dieses Register enthält den Code, der den laufenden Modus einer Operation festlegt.

R17 Dieses Register enthält den Status für eine externe Kommunikation.

Alle anderen Register können durch den Benutzer frei belegt werden.

Nun noch etwas über die Speicherung von Zahlen in den Speichern R40-R77:

Die Zahlenspeicherung in der CPU erfolgt im Oktalsystem. Eine Zahl wird in insgesamt 8 Registern gespeichert. Man unterscheidet dabei zwischen der Speicherung von mit REAL festgelegten Zahlen und den übrigen.

- REAL-Zahlen: Diese Zahlen werden ohne Vorzeichen jedoch mit Exponent gespeichert. zB. die Zahl 4678.912345 entspricht 4.678912345 * 10^3 und wird wie folgt gespeichert:

4 6	7 8	9 1	2 3	4 5	0 0	+ 0	0 3
0100 0110	0111 1000	1001 0001	0010 0011	0100 0101	0000 0000	0000 0000	0000 0011

- Anders sieht die Sache bei normalen Zahlen aus. Hier wird das Vorzeichen zusätzlich mitgespeichert.

zB. Die Zahl -0.048 entspricht -4.8 * 10^-2

- 4	8 0	0 0	- 2
0011 0100	1000 0000	0000 0000	0011 0010

SHORT-Zahlen

zB. Die Zahl 5001

+ 0	5 0	0 1
0000 0000	0101 0000	0000 0001

INTEGER-Zahlen

Wenn man nun im Handbuch des HP-75 nachschlägt, sieht man, daß grundsätzlich
REAL-Zahlen 12 Digits mit einem dreistelligen Exponenten haben(siehe oben),
SHORT-Zahlen 5 Digits mit einem zweistelligen Exponenten haben (siehe oben),
INTEGER-Zahlen 5 Digits ohne Exponenten haben.
Damit ist die oben aufgestellte These richtig.

2. Die Stromversorgung:

Da ohne Strom nichts läuft, gehört auch diese Einheit zu den Grundpfeilern eines Computers. Der HP-75 besitzt als Portable-Computer natürlich einen Akku, der durch eine Spannung von 8V und eine Stromstärke von 375mA aufgeladen wird. Dieser Akku versorgt zwei Stromkreise mit dem nötigen "Saft". Im ersten Stromkreis befinden sich die RAMs, deren Inhalt ja auch nach dem Ausschalten erhalten bleibt (siehe Continuos Memory), und die interne Zeiteinheit, die ja zu jeder Zeit die richtige Zeitangabe angeben soll. Der andere Stromkreis läßt sich ein- und ausschalten durch die ATTN-Taste. Die vom Akku ankommende Versorgungsspannung wird durch einen Converter in eine Spannung von 3.5-5.5V (je nach Ladezustand des Akku) und eine Stromstärke von 50-100mA (auch abhängig vom Zustand des Akku) umgewandelt. Dieser DC-DC-Converter taktet sich selbst durch einen eingebauten Oszillator (= interne Spule).

3. Die Anzeige:
Beim Betrachten des HP-75C fällt als erste Ausgabeeinheit die 32-stellige Anzeige über der Tastatur auf. Jedes Zeichen in der Anzeige wird durch eine 5x8-Matrix dargestellt, wobei die achte, dh. die unterste Reihe für das Unterstreichen reserviert ist. Zu der Anzeige gehören auch noch 4 Indikatoren:
a) BAT erscheint, wenn die Spannung im Akku zu gering ist, um ohne Fehler arbeiten zu können.
b) ERROR erscheint, wenn ein Fehler in einem Programm oder in einem Ablauf direkt über die Tastatur aufgetreten ist.
c) APPT erscheint, wenn ein Alarm angesprochen wird oder wurde, dh. sobald Zeit der Systemuhr (TIME-Modus) die Alarmzeit (APPT) überschreitet.
d) PRGM erscheint bei der Programmausführung, dh. bei der RUN-Phase.
Im Gegensatz zur Anzeige mit 32 Zeichen speichert der Anzeigespeicher insgesamt 96 Zeichen. Dies bedeutet, daß immer nur ein Ausschnitt aus dem Anzeigespeicher angezeigt wird.
Die LCD-Anzeige wird von 8 ICs aus gesteuert und kontrolliert. Eines der ICs, das Anzeigekontroll-IC, hat eine Sonderstellung. Es steuert die Stromzufuhr für die Einzelpunkte der Anzeige, dh. es erzeugt die Punktmatrix, die für jedes Zeichenmuster verschieden ist. Dieses IC steht bei der Anzeige ständig in Verbindung mit der CPU, so daß die CPU immer weiß, wo welches Zeichen steht. In diesem IC wird in einem speziellen RAM jedes Zeichen durch zwei Bytes gespeichert. Das erste Byte gibt an, wo das Zeichen im Anzeigenspeicher steht, das zweite Byte gibt den ASCII-Code des Zeichens an.
Jedes Zeichen besteht aus 5 (für eigentliches Zeichen) + 2 (für Zwischenraum), also insgesamt aus 7 Spalten. Für jede Spalte eines Zeichens steht 1 IC zur Verfügung, das diese Spalte steuert. Haben die 7 ICs ein Zeichen erzeugt, so gehen sie weiter zum nächsten Zeichen, bis die Zeile beschrieben ist.
4. Die interne Uhr:
Der Hauptteil der Uhreinheit ist ein Zähler, der mit SET auf die gewünschte Zeit gesetzt wird. Mit Hilfe eines 32kHz-Quarzes erreicht er eine Abweichung von 3 Minuten im Monat. Diese Genauigkeit läßt sich jedoch noch verbessern: Man wählt mit EXACT einen Zeitpunkt, von dem an bis zum nächsten EXACT-Zeitpunkt der Rechner die Abweichung feststellt und schon gleich als Ausgleich zur Zeit dazugezählt wird. Damit läßt sich eine Abweichung von nun mehr 15 Sekunden im Monat erreichen. Dies wird dadurch möglich, daß die CPU die Zeit manipulieren kann. Sie kann eine Sekunde je nach den Erfordernissen bis auf 1.25 Sekunden verlängern, so daß nun 4 Sekunden auf 5 Sekunden verlängert sind. Das IC, das diese Vorgänge steuert, wird auch noch für eine andere Zeitsteuerung im Rechner gebraucht. Wie in jedem Computer benötigt auch die CPU des HP-75 eine Taktfrequenz, mit deren Hilfe sie Daten und Befehle durch das Bus-System durchdrückt. Die Frequenz, mit der dabei gearbeitet wird, ist abhängig vom Schaltzustand des Rechners. Man unterscheidet dabei 3 Zustände:
a) Der Tiefschlafmodus: Dieser ist dann vorhanden, wenn der Rechner ausgeschaltet ist, jedoch trotzdem intern für die Uhr, für die Alarmkontrolle und den Speicherinhalt mit Strom und Spannung versorgt wird. Dafür wird mit einer Taktfrequenz von wenigen Kiloherz gearbeitet.
b) Der Normalmodus: Dieser ist dann vorhanden, wenn der Rechner zwar eingeschaltet ist, aber nicht im RUN-Modus arbeitet. Nur die Anzeige ist aktiv. Es wurde keine Taste gedrückt. In diesem Modus wird mit einer Taktfrequenz von 4.68kHz gearbeitet.
c) Der Arbeitsmodus: Dieser ist dann vorhanden, wenn eine Taste gedrückt wird, oder etwas in die Anzeige gebracht wird, dh. zusammenfassend, wenn Daten oder Befehle durch das Bus-System, vom User gesteuert, durchgedrückt werden.
Das Umschalten vom Tiefschlafmodus in den Normalmodus bewirkt die ATTN-Taste. Das weitere Umschalten dann in den Arbeitsmodus bewirkt das Drücken irgendeiner beliebigen Taste. Ist eine Operation abgeschlossen, schaltet die CPU automatisch wieder zurück in den Normalmodus. Beim Kommando BYE oder dem gleichzeitigen Drücken von ATTN- und SHIFT-Taste wird in den Tiefschlafmodus zurückgeschaltet, dh. Display und Tastatur, bis auf die ATTN-Taste, werden inaktiv.
Auch für die Interface-Loop (abgekürzt: IL) muß eine Taktfrequenz zur Verfügung gestellt werden, durch die Daten und Kommandos zu den einzelnen Peripherieeinheiten transportiert werden. Diese Arbeit übernimmt auch der og. Integrierte Schaltkreis.

5. Die Tastatur:
   Die QWERTY-Tastaur des HP-75 besteht aus einer 8-Spalten-mal-10-Reihen-Matrix. Wird eine Taste gedrückt, wird dieses von der CPU als Signal aus dieser Matrix empfangen. Mit Hilfe der Spalten und Reihen, kann die CPU bestimmen, welche Taste gedrückt worden ist. Jede Position in dieser Matrix besitzt einen entsprechenden Code. Dieser Code wird dann von der CPU weiter verwertet.
   Ist der Rechner angeschaltet, sind auch die Tasten in einem energiereicheren Zustand. Sie erwarten kontinuierlich Informationen von Benutzer. Ist der Rechner jedoch ausgeschaltet, befinden sich alle Tasten außer der ATTN-Taste im Zustand 0. Deshalb läßt sich der Rechner auch nur durch die ATTN-Taste einschalten. Außer den obengenannten Eigenschaften, besitzt die Tastatur noch die Eigenschaft des Prellens, wenn die Taste längere Zeit gedrückt bleibt.
6. Die HP-IL:
   Im HP-75 bereits eingebaut ist ein HP-IL-Modul. Mit dessen Hilfe läßt sich die HP-IL-Schleife direkt ansteueren. Auch im Betriebssystem des HP-75 ist im Bereich von $FF10-$FF17 die Steuerung des HP-IL-ICs schon vorhanden. Durch Verändern dieser Statusbytes, lassen sich neue IL-Befehle kreieren. Auch die LEX-Karte HPILCMDS aus dem I/O-Programmpaket, greift auf diese Statusbytes zurück und ermöglicht dem HP-75 damit, jede IL-Schleife individuell zu steuern.
7. Der Kartenleser:
   Der im HP-75C eingebaute Magnetkartenleser, mit dem man Daten auf Karten schreiben kann oder Daten von Karten lesen kann, hat eine Speicherkapazität von 1.3KBytes. Die Karten werden von Hand durch die Lesevorrichtung gezogen, wenn der Kartenleser auf Datenempfang geschaltet wurde. Die Karten sind 25.4 cm lang und 1 cm breit.
   Der Vorgang der Verarbeitung von Magnetkarten:
   Der Benutzer initialisiert eine Kartenleser-Operation mit den zur Verfügung stehenden Befehlen. Danach schiebt er die Karte bis zur schwarzen Markierung vor, macht den Rechner durch Drücken der RTN-Taste bereit, Daten zu empfangen, und zieht schließlich die ganze Karte mit gleichbleibender Geschwindigkeit durch.
   Aufbau der Karte:
   Es gibt vier Bereiche auf jeder Karte: 2 Daten-Bereiche und zwei Timing-Bereiche. In den Timing-Bereichen sind die Informationen für den Kartenleser festgehalten über die Geschwindigkeit, mit der die Karte durchgezogen werden muß. Die Daten-Bereiche enthalten 4 Felder von Informationen.
   Jedes Feld beginnt mit einem Führungsbyte, das bestimmt, ob und welche Daten abgespeichert werden. Desweiteren gibt es noch einen Header (dieser enthält Informationen über die Größe und das Format der Karte), ein Schutzbyte (dieses Flag zeigt an, ob die Karte vor dem Überschreiben geschützt ist oder nicht), dem File-Header (dieser Teil enthält Informationen über Filename, Größe, Filetyp, Daten, Bereichnummer und Anzahl der insgesamt benötigten Bereiche). Danach folgen endlich die Daten (1.3KByte pro Karte).
   Die Ansteuerung:
   Der Kartenleser wird durch den HP-75C mit Hilfe von BASIC-Befehlen angesteuert. Diese Befehle erlauben das Kopieren von Daten von Karte in den Hauptspeicher oder zurück in den Optionen privat, nicht privat, mit oder ohne Paßwortschutz; außerdem das Katalogisieren und Auflisten von Inhalten einer Karte.
   Wird die Karte zu schnell oder zu langsam durch den Leserschlitz gezogen, stellt dies ein Sensor fest und fordert den Benutzer auf, es noch einmal zu versuchen.Gleichfalls, wenn ein Fehler beim Lesen oder Schreiben der Karte auftritt, wird dieser direkt an den Benutzer weitergeleitet, und es wird zu einem weitern Versuch des Lesens oder Schreibens dieser Karte aufgefordert.
   Die Befehle, durch die der Kartenleser angesteuert wird, sind folgende:
   COPY 'filename' TO (CARD/'filename :(CARD/PCRD)(/passwort)')
   COPY (CARD/'filename:(CARD/PCRD)(/passwort)') TO 'filename
   CAT (CARD/':(CARD/PCRD)')
   PROTECT
   UNPROTECT

Ist der File zugleich workfile, braucht der Filename nicht angegeben zu werden. Alle Operationen des Kartenlesers können durch Drücken der ATTN-Taste gelöscht werden. Bereits eingelesene Fileteile werden wieder gelöscht. Bevor Informationen eingelesen werden, prüft die Steuerungssoftware des Kartenlesers das Schutz-Flag und teilt dem Benutzer mit, wenn das File geschützt ist.
Das Interface zwischen Rechner und Kartenleser:
Dieses Interface benutzt eine 2-Byte-I/O-Adresse. Das erste Byte enthält Informationen und Daten (vom oder zum System, Angaben, ob Lese- oder Schreibmodus). Das zweite Byte enthält den Status des Kartenlesers. Dieses Byte enthält Bits, die dem Benutzer erlauben, den Kartenleser an- oder auszuschalten, den Kartenleser in den Lese- oder Schreibmodus zu bringen, einen RESET durchzuführen, die Hardware des Kartenlesers zu prüfen.
Der Prozeß des Datenaustauschs ist einfach: Zuerst wird die Hardware angesprochen und geprüft, ob sie READY ist. Danach beginnt eine Schleife, in der abwechselnd Daten gesendet werden und die Hardware auf READY (=Betriebsbereitschaft) überprüft wird. Diese Schleife endet, wenn sämtliche Daten eingelesen sind. Danach wird auch wieder der Kartenleser ausgeschaltet. Dieser Vorgang wiederholt sich für jedes Feld der Karte. Das besondere an diesem Kartenleser ist die Eigenschaft, daß er liest und schreibt mit den gleichen Codierungen. Während die Software wartet bis die Hardware bereit ist, wird im Timing-Register der Inhalt rückwärts herabgezählt. Nach jedem Byte wird das Register zurückgesetzt. Wenn der Inhalt 0 erreicht hat, muß die Oeration neu gestartet werden.

8. Memory-Organisation:
Die CPU des HP-75 hat die Fähigkeit, 64kByte des Speichers direkt zu adressieren. Dieser Speicher beinhaltet Platz für verschiedene Teile:
Im Memory des Rechners unterscheidet man mehrere Teile:
   a) Sechs System-ROMs zu je 8192 Bytes. Von diesen ROMs belegen zunächst 4 die Adressen 0000-7FFF, ein ROM die Adressen E000-FFFF. Im Bereich 6000-7FFF können unter der gleichen Adresse auch noch mehr ROMs gesteuert werden. In diesem Bereich liegt in der Grundversion noch das 6.ROM (System-ROM 180), später Zusatz-ROMs.
   b) Zwei RAMs zu je 8192 Bytes in der Grundversion mit den Adressen 8000-BFFF. Bei der Erweiterung mit dem 8k-Speichererweiterungsmodul belegt dieses den Adreßbereich C000-DFFF.
   c) Memory durch CPU adressierbar (Adreßbereich 0000-FFFF)
   d) Memory durch den EMC adressierbar 24k-208k.
   e) Der Block FF00-FFFF dient zum Abspeichern der I/O-Adressen, wenn der Bereich direkt von der CPU gesteuert wird, ansonsten zum Abspeichern von RAM-Daten.

Dezimale Adresse	Hexadezimale Adresse	
0	0000	IRL-Vektoren
17 Byte	0011	ROM
8 k	2000	ROM
16 k	4000	ROM
24 k	6000	ROM
32 k	8000	

Dezimale Adresse	Hexadezimale Adresse	
32 k	8000	RAM
40 k	A000	RAM
48 k	C000	RAM
56 k	E000	ROM
63 k	FF00	I/O-Adressen
64 k	FFFF	
256 k	FFFFFF	RAM

Das Operationssystem im HP-75 steuert über die 48k ROM den RAM-Bereich. Bringt man den Rechner dazu, dei Steuerung an externe Stellen abzugeben, ist auch ein höherer RAM-Bereich möglich. Über die I/O-Adressen steuert der Rechner direkt sämtliche INPUT- und OUTPUT-Glieder (Tastatur, Kartenleser und die IL-Schleife). Über diese I/O-Adressen werden auch Daten übermittelt und zwar in Strings von 8 Bytes Länge.
Einer der Vorteile des HP-75 ist die Möglichkeit im RAM-Bereich auch Daten festzuhalten, wenn der Rechner ausgeschaltet ist (Continous Memory).
Betrachten wir uns den RAM-Kreis, so fällt noch ein Register zwischen RAM-Bank und der CPU des Rechners auf. Dieses Register übernimmt die Adressierung der CPU. Will die CPU auf Daten im RAM zurückgreifen, so spricht sie über dieses Register Adressen im RAM-Bereich des Rechners an.
Anders verläuft dieses bei der Adreßierung von ROMs. Hier besitzt jedes ROM einen eigenen Kreis und ein eigenes Register, das direkt die Daten, die benötigt werden, zur Verfügung stellt. Hier werden also die ROMs direkt von der CPU angesprochen.

9. Ausblick:
Das Interessanteste im Betriebssystem und im internen Aufbau des Rechners ist der zur Verfügung stehende RAM-Bereich. Durch richtiges Anpassen des Betriebssystem durch die Adreßpointer EMC auf eigene Zwecke ist eine Erweiterung des RAM-Bereichs ähnlich der ROM-Bank möglich. Dies ist die dringendste Aufgabe jedes HP-75-Benutzers !!!!!!!!

2.Kapitel

Der Aufbau von Files

1. BASIC-Files:

1.1. Darstellung interner Zahlen:

a) absolute Adresse:
Die Darstellung erfolgt Hexadezimal. Eine absolute Adresse belegt 2 Bytes. Um die Adresse richtig zu lesen, muß erst das 2.Byte und dann das 1.Byte aneinandergeschrieben werden - dies ergibt die richtige Adresse (so sind auch alle anderen Zahlen, die 2 Bytes belegen in umgekehrter Reihenfolge dargestellt).
Bsp: Die absolute Adresse lautet C9 02, dann lautet das 1.Byte Hex 02 und das 2.Byte Hex C9.

b) Programmlänge:
Ebenso wie bei der absoluten Adresse erfolgt auch hier die Darstellung Hexadezimal in umgekehrter Reihenfolge.
Bsp: Programmlänge 2483 Bytes ergibt binär 0000100110110011 und Hexadezimal 09 B3. Dann lautet das 1.Byte B3 und das 2.Byte lautet 09.

c) Zeilennummern:
Hier erfolgt die Darstellung im BCD-Code in umgekehrter Reihenfolge. (BCD-Code = Binary Coded Decimal). Durch diese Darstellung läßt sich die Zeilennummer in der Hexadezimaldarstellung der Bytes im Klartext lesen. Die Hunderter- und Tausenderstelle wird im 2.Byte dargestellt und die Einer- und Zehnerstelle im 1.Byte. Diese Form der Darstellung erklärt auch zugleich, warum die höchste Zeilennummer 9999 ist.
Bsp: Bei einer gewünschten Zeilennummer von 2497 lautet das 1.Byte Hex 97, das 2.Byte Hex 24.

d) Längenangaben:
Die Angabe von Längen (Zeilenlänge, Textlänge etc.) erfolgt stets im Hex-Code. Werden 2 Bytes für die Längenangabe benötigt, so sind diese wieder in umgekehrter Reihenfolge dargestellt. (siehe Programmlänge)

e) Pointerangaben (siehe Längenangaben)

1.2. Allgemeiner Aufbau von BASIC-Files:
Man unterscheidet 2 Teile eines BASIC-Files: den Programmkopf, auch Header genannt und den eigentlichen Programmhauptteil, die auf Kassette direkt aufeinanderfolgen und im RAM des Rechners getrennt aufgeführt sind (siehe Aufbau des Memory).

a) Header:

aa) Angabe der absoluten Adresse (Byte 1 und Byte 2):
Hierbei ist zu bemerken, daß die RAM-Adressen von 8000 bis E000 gehen und in drei Bausteine aufgeteilt sind: RAM 1 8000-9FFF
RAM 2 A000-BFFF
RAM 3 C000-DFFF (entspricht 8k RAM-Erweit.)
Näheres über die RAM-Bausteine unter dem Stichwort "Memory-Aufbau".
Darstellung der Adresse ist in 1.1.a) beschrieben.

ab) Angabe der Programmlänge (Byte 3 und Byte 4):
Darstellung wie unter 1.1.b) beschrieben.

ac) PRIVATE-Sicherung (Byte 5):
Dieses Byte hat den Dezimalwert 254 bei non-privaten BASIC-Files
202 bei privaten BASIC-Files

ad) Angabe der Fileart (Byte 6):
Dieses Byte hat für BASIC-Files den Hex-Wert 42 (ASCII-Code für B)

ae) Speicherung des Datums und der Zeit (Byte 07 bis Byte 10):
Grunddatum für die Errechnung des Datums ist der 1.1.1900 um 0.00 Uhr. Diese 4 Bytes geben nun in codierter Form die Anzahl der Sekunden, die von diesem Grunddatum bis zum Datum der Abspeicherung des Programms verflossen sind. Als Grundwert des verwendeten Zahlensystems dient die Zahl 256.
Die Erklärung der Codierung läßt sich am besten an einem Beispiel zeigen: Byte 07 bis Byte 10 haben die Werte Hex 60 56 AD 9B, dem dezimal entspricht: 96 86 173 155. Die Anzahl der Sekunden seit dem Grunddatum errechnet sich wie folgend: Byte 07 + Byte 08*256 + Byte 09*256^2 + Byte 10*256^3. Dies ergibt für unser Beispiel 2.611.828.320 Sekunden. Eine Division durch 86400 ergibt 30229 Tage ergibt (also 82 Jahre und 278 Tage). Den Rest rechnet man nun in Minuten und Sekunden um (Ergebnis: 11 Stunden, 52 Minuten).

Als endgültiges Datum erhält man nun den 5.10.1982 um 11.52 Uhr.

af) Angabe des Filenamens (Byte 11 bis Byte 18):
Diese Bytes geben als ASCII-Zeichen den genauen Namen (8 Zeichen) des Files an. Der Name wird wenn nötig mit Blanks aufgefüllt.

b) Hauptteil des Programmes, wenn es nicht initialisiert ist.

ba) Pointer:
Diese Pointer haben nur bei LEX-Files eine besondere Bedeutung (dazu später mehr). Sie legen dabei besondere Zielpunkte innerhalb eines Files fest.
Bei BASIC-Files haben sie den Wert 00 (also bei 5 Pointern 00 00 00 00 00).

bb) Programmzeilen:

bba) Angabe der Zeilennummer:
Diese erfolgt wie in 1.1.c) beschreiben in den ersten beiden Bytes einer Programmzeile.

bbb) Angabe der Zeilenlänge:
In diesem 3.Byte einer Programmzeile wird dezimal die Länge einer Zeile angegeben, dh. die Anzahl der Bytes von einschließlich dem 4.Byte einer Zeile bis zum letzten Byte der Zeile (siehe unter Punkt bbd)).

bbc) Angabe der Befehle:
4.Byte bis zum vorletzten Byte einer Zeile.

bbd) Angabe des Zeilenendes:
Dieses Byte hat konstant den Wert dezimal 14 (Hex OE).

bc) Ende des Programms:
Die letzen 5 Bytes eines Programmes haben immer denselben Aufbau:
Byte 1 = Hex 99
Byte 2 = Hex A9
Byte 3 = Hex 02 (Zeilenlänge 2)
Byte 4 = Hex 8A (Codierung für END)
Byte 5 = Hex OE (Zeilenende, hier speziell Programmende)
Nachfolgende Befehle werden nicht mehr mit diesem Programm in Verbindung gebracht.

1.3. Aufbau der BASIC-Befehle, wenn ein Programm noch nicht initialisiert ist:
Da man in eine BASIC-Zeile mehrere BASIC-Befehle schreiben kann, muß sich der Rechner eines Trennbytes bedienen. Er benutzt dafür den Klammeraffen, ein Byte mit dem Wert dezimal 64.

a) Rechnergrößen:

aa) Die Darstellung von Konstanten:

aaa) Textkonstanten:
Byte 1 = dezimal 150, wenn die Konstante in einfachen Hochkommas (') steht.
= dezimal 5 , wenn die Konstnate in Doppelhochkommas (") steht.
= dezimal 6 , wenn die Konstante nicht eingeschlossen ist, wie bei Befehlen wie DATA und IMAGE bzw. bei Kommentaren.
Byte 2 = Angabe der Länge des Strings
Bei einem String der Länge 0 natürlich 00, wobei dann die folgenden Bytes 3ff. entfallen.
Byte 3 ff. = Codierung des ALPHA-Strings im ASCII-Code
Die Anzahl dieser Bytes entspricht der im Byte 2 angegebenen Länge.

aab) numerische Konstanten: 1. INTEGER-Konstanten:
Byte 1 = dezimal 26
Byte 2 bis 4 = Darstellung der Konstanten im BCD-Code
1.Halbbyte = Zehnerstelle
2.Halbbyte = Einerstelle
3.Halbbyte = Tausenderstelle
4.Halbbyte = Hunderterstelle
5.Halbbyte = Hex 0, da INTEGER-Zahlen keine Hunderttausenderstellen besitzen (maxint = 99999)
6.Halbbyte = Zehntausenderstelle
Bei negativen Konstanten kommt jetzt als 5.Byte ein Byte mit dem Wert dez. 56.

2. REAL-Konstanten
Byte 1 = dezimal 4

Byte 2 bis 3 = Darstellung des Exponenten
positiver Exponent: BCD-Code in umgekehrter Reihenfolge
Bsp: E 259 = Hex 59 02
negativer Exponent: Exponent+1000 im BCD-Code in umgekehrter Reihenfolge
Bsp: E-159 = Hex 41 08, da -159+1000=841

Zu beachten ist, daß vom 3.Byte in Abweichung zu anderen Zahlensystemen nur das 2.Halbbyte benutzt wird und das 1.Halbbyte stets 0 ist.

Byte 4 bis 9 = Darstellung der Mantisse im BCD-Code in umgekehrter Reihenfolge:

1.Halbbyte 11.Stelle
2.Halbbyte 12.Stelle
3.Halbbyte 9.Stelle
4.Halbbyte 10.Stelle
.
.
.
9.Halbbyte 3.Stelle
10.Halbbyte 4.Stelle
11.Halbbyte 1.Stelle
12.Halbbyte 2.Stelle

Bei negativen Konstanten kommt jetzt als 10.Byte ein Byte mit dem Wert Dez. 56.

c) Darstellung von Variablen:

ca) Variablenname:

Byte 1 des Variablennamens:
bei einstelligen Variablen Dezimal 32 (Blank)
bei zweistelligen Variablen die Zahl im ASCII-Code

Byte 2 des Variablennamens:
der Buchstabe im ASCII-Code

Ausnahmen zu dieser Codierung gibt es bei der Definition von Funktionen, dies wird jedoch im Teil Funktionendefinition erläutert.

cb) Jedem Variablennamen ist ein Byte vorangestellt, aus dem hervorgeht,

1. ob es sich um eine Variable handelt, der etwas zugewiesen wird,oder ob diese Variable nur angesprochen wird (zB. bei Ausgabebefehlen oder arithmetischen Ausdrücken).
2. um welchen Typ von Variable es sich handelt.

somit gibt Dezimal 1 an: numerische Variable nur angesprochen
2 : Array-Variable nur angesprochen
3 : ALPHA-Variable nur angesprochen
17 : numerische Variable, der etwas zugewiesen wird
18 : Array-Variable, der etwas zugewiesen wird
19 : ALPHA-Variable, der etwas zugewiesen wird

cc) Array-Variablen und ALPHA-Variablen benötigen in den meisten Fällen noch weitere Angaben, die dann dem Variablennamen angehängt werden. Bei Array-Variablen müssen die Indices und bei ALPHA-Variablen die Länge des Strings angegeben werden.
So wird nach den expliciten Angaben (siehe oben) zu den Indices folgendes angegeben:

Dezimal 9 bei Ende Array-Variable mit einem Index, der etwas zugewiesen wird
10 bei Ende Array-Variable mit zwei Indices, der etwas zugewiesen wird
11 bei Ende Array-Variable mit einem Index, die nur angesprochen wird
12 bei Ende Array-Variable mit zwei Indices, die nur angesprochen wird

cd) Besonderheiten beim READ- und PRINT=-Befehl:
Hier ist es möglich ganze Arrays anzusprechen. Dabei wird Dezimal 181 dem Variablenamen vorangestellt bei Arrays mit einem Index und Dezimal 182 bei Arrays mit zwei Indices.
Solche Arrays werden nach dem Variablennamen mit Dezimal 36 beim READ-Befehl und mit Dezimal 34 beim PRINT#-Befehl abgeschlossen (näheres bei der Beschreibung der Befehle).

d) Die verschiedenen Rechnerkonstanten:
Für verschiedene Rechnerkonstanten existieren einzelne Bytes, diese belegen dann keinen Textstring.

Dezimal 151 = TEXT
152 = BASIC
153 = LIF1
155 = INTO
168 = ON (bei Befehlen, bei denen das ON dahintersteht wie zB:
ALARM ON, DEFAULT ON usw.)
169 = OFF (ALARM OFF. DEFAULT OFF usw.)
187 = APPT
212 = CARD
214 = keys
219 = TO (nur bei FOR-NEXT-Schleife nicht)

e) Eigentlicher Aufbau der BASIC-Befehle:
Nachdem nun die Konstanten und Rechnervariablen erläutert sind, ist es möglich, auf den Aufbau eines Befehls einzugehen.
Grundsätzlich löst der Rechner die Befehle intern so auf, daß er fortlaufend alles abarbeiten kann. Dabei scheint der Rechner zur Ausführung der Befehle wohl intern einen Stack zu belegen, in dem die zur Ausführung notwendigen Befehle zwischengespeichert werden.
Allgemein gibt es Funktionen, die gar keine Werte benötigen; solche, die einen Wert zur Ausführung benötigen und solche, die zwei Werte zur Ausführung benötigen.
Zum anderen ist es aber auch möglich, durch Angabe von Klammerungen, die Anzahl der zwischengespeicherten Werte zu beeinflussen. Da der Rechner die Werte ohne die Klammern abspeichert und zwar in der Reihenfolge, wie sie auszurechnen sind, wobei je nach Anzahl der Klammerebenen der Stack mehrere Zahlen abspeichern muß, verschwinden zB. nach der Eingabe einer Befehlszeile mit überflüssigen Klammern, diese Klammern.
Wir geben also einen Befehl als Text in eine Zeile ein. Der Rechner interpretiert diese Zeile, löst sie in einen internen Code auf und, wenn diese Zeile wieder angezeigt wird, wird umgekehrt der interne Code wieder interpretiert angezeigt. Damit ist auch das Verschwinden von überflüssigen Klammern klar, da durch die Stackverwaltung keine Bytes für Klammern benötigt werden.

ea) Befehle, die keine Werte benötigen (konstante Funktionen):
Dezimal 171 = EPS
178 = MEM
184 = KEY$
192 = VER$
199 = INF
201 = PI
210 = ERRN
211 = ERRL
154 = RES

eb) Befehle, die einen Wert zur Ausführung benötigen:
Hier steht der Wert, der zur Ausführung des Befehles notwendig ist, vor dem Byte, das den Befehl angibt. Hierbei ist es egal, ob der Wert durch eine andere Funktion erzeugt wird, ob er nur eine Konstante ist, oder durch eine Kombination von Funktionen und Konstanten entsteht. Wichtig ist, daß wenn das eine Byte, das den Befehl angibt, im internen Code erscheint, ein Wert im Stack steht, der für diese Funktion paßt, egal, wie dieser Wert entstanden ist.

Bsp: SIN(2) ist intern codiert als dezimal 26 2 0 0 216,
wobei 26 2 0 0 die Konstante 2 ist und 216 das Byte für die Funktion Sinus.

Bsp: SIN(EPS) ist intern codiert als dezimal 171 216,
wobei 171 EPS ist und 216 Sinus

Bsp: COS(2+1) ist intern codiert als 26 2 0 0 26 1 0 0 43 217
(hier mache ich einen Vorgriff auf Funktionen mit zwei Variablen)
Das Prinzip ist erkennbar: Zunächst werden die beiden Werte für die Funktion + (Dezimal 43 = ASCII-Code des Zeichens +) bereitgestellt, die Funktion + ausgeführt und im Stack steht dann die Summe 3 für die Funktion Cosinus (Dezimal 217) zur Verfügung.

Nun also die Liste der Bytes für diese Funktionen:
Einige Funktionen werden hier vermißt werden, da für ihre Codierung mehr

als ein Byte benötigt wird, dazu in einem späteren Punkt.
Dezimal 56 = Vorzeichenwechsel, also die Angabe eines Minuszeichens vor einer Funktion oder einer Konstanten.

Dezimal 170 = IP
172 = FP
173 = CEIL
176 = SQR
179 = ABS
183 = SGN
185 = COT
186 = CSC
188 = EXP
189 = INT
190 = LOG10
191 = LOG
193 = SEC
194 = CHR$
195 = STR$
196 = LEN
197 = NUM
198 = VAL
202 = UPRC$
216 = SIN
217 = COS
218 = TAN
226 = DEG
227 = RAD
228 = FLOOR

Zum Abschluß dieses Punktes noch ein Beispiel für eine Stringfunktion:
Bsp: UPRC$('abc') ist intern codiert als dezimal 150 3 97 98 99 202, wobei 150 Textkonstante heißt, 3 die Länge des strings, 97 98 99 abc und 202 die Funktion UPRC$ ist.

ec) Befehle, die zwei Werte zur Ausführung benötigen:
Hier müssen im Stack zwei Werte zur Verfügung stehen, damit die Funktion ausgeführt werden kann, folglich stehen vor dem Byte, das die Funktion angibt, auch mindestens zwei Konstanten und/oder Funktionen.
Hier ein etwas komplexeres Beispiel:
LOG(1+2)*SIN(A+B) ist intern codiert als dezimal
26 1 0 0 26 2 0 0 43 191 1 32 65 1 32 66 43 216 42, wobei
26 1 0 0 die Konstante 1 ist
26 2 0 0 die Konstante 2 ist
43 die Funktion + ist
191 die Funktion LOG ist
1 32 65 die numerische Variable A ist, die nur angesprochen wird
1 32 66 die numerische Variable B ist, die nur angesprochen wird
43 wieder die Funktion + ist
216 die Funktion Sinus ist
42 die Funktion * ist

Als neue Funktionen erscheinen hier + und *.
Aufgrund des Beispiels wird die interne Codierung allmählich klar.
Es ist immer dasselbe. Es erscheinen zunächst die zur Ausführung notwendigen Werte und dann die Codierung für die Funktion. Analog erfolgt die Codierung bei Stringfunktionen.
Hier die Bytecodierung der Funktionen, die zwei Werte zur Ausführung benötigen:

Dezimal 38 = & (Verkettung von zwei ALPHA-Strings)
42 = *
43 = +
45 = -
47 = /
48 = ^

Dezimal 174 = MAX
177 = MIN
209 = DIV
224 =
225 = POS

Hier noch ein Beispiel für die Funktion POS:

Bsp: POS(F1$,"a'b") ist intern codiert als dezimal
3 49 70 5 3 97 39 98 225, wobei
3 49 70 die ALPHA-Variable F1$ ist, die nur angesprochen wird
(49 steht für die 1 und 70 für F)
5 3 97 39 98 die Stringvariable a'b ist
(5 steht für Textkonstante, die mit " eingeschlossen ist, 3 steht für die Länge der der Konstanten, 97 39 98 ist der ASCII-Code für a'b)
225 die Funktion POS ist.

ed) Beispiele für Klammerungen:

(4+5)*3 ist intern codiert als dezimal
26 4 0 0 26 5 0 0 43 26 3 0 0 42, wobei
26 4 0 0 die Konstante 4 ist
26 5 0 0 die Konstante 5 ist
43 die Funktion + ist
26 3 0 0 die Konstante 3 ist
42 die Funktion * ist

SIN((26-A)*(C1*(D+E))) ist intern codiert als dezimal
26 38 0 0 1 32 65 45 1 32 68 1 32 69 43 1 49 67 42 42 216, wobei
26 38 0 0 die Konstante 26 ist (Dez 38 ist Hex 26)
(man beachte hier: 26 ist der Anzeiger, daß eine numerische Integerkonstante vorliegt)
1 32 65 die numerische Variable A ist, die nur angesprochen wird
45 die Funktion - ist
1 32 68 die numerische Variable D, die nur angesprochen wird
1 32 69 die numerische Variable E, die nur angesprochen wird
43 die Funktion +-
1 49 67 die numerische Variable C1, die nur angesprochen wird
42 die Funktion *
42 die Funktion *
216 die Funktion SIN

Mit diesem Beispiel dürfte auch die Stackbildung klar werden, denn man sieht, daß hier Werte im Laufe der Rechnung für die einzelnen Funktionen zwischengespeichert werden müssen.

ee) Wertzuweisungen:

Nun ist es an der Zeit zu zeigen, wie eigentlich den einzelnen Variablen Werte zugewiesen werden.
Dies geschieht folgendermaßen:
Zunächst erscheint im internen Code die Variable mit dem entsprechend vorangestellten Byte wie oben erläutert, dann erfolgt die Ausführung der Funktionen und/oder Konstanten, die der Variablen zugewiesen werden sollen, wobei die Codierung genauso erfolgt, wie oben beschreiben ist.
Die Zuweisung wird abgeschlossen durch das sogenannte Zuweisungsbyte.
Hier nun die Codierung des Zuweisungsbytes:

Dezimal 7 = Zuweisung zu einer ALPHA-Variablen
8 = Zuweisung zu einer numerischen Variablen, wobei hier numerische Variable eine normale Variable oder ein Array sein kann.

Liegen Mehrfachzuweisungen vor, so erscheinen am Ende der Zuweisung ebensoviele Zuweisungsbytes, wie Variablenzuweisungen vorliegen, hier werden dann aber nicht die Bytes mit den Dezimalwerten 7 oder 8 benutzt, sondern

Dezimal 20 = Zuweisung zu mehreren numerischen Variablen
21 = Zuweisung zu mehreren ALPHA-Variablen

Wird bei einer Zuweisung das Wort LET hinzugefügt, so wird vor die Variable, der etwas zugewiesen wird ein Byte mit dem Wert dezimal 98 vorangestellt. (Auf selbstdefinierte Funktionen gehe ich später ein).
Zu den Wertzuweisungen nun ein etwas komplexeres Beispiel:
A1(C2/3,SQR(25))=A(1)*B(1,2) ist intern codiert als
18 49 65 1 50 67 26 3 0 0 47 26 37 0 0 176 10 2 32 65 26 1 0 0 11 2 32 66 1 0 0 26 2 0 0 12 42 8, wobei
18 49 65 die Array-Variable A1 ist, der etwas zugewiesen wird (deshalb beginnt der Code mit 18)
1 50 67 die numerische Variable C2 ist, die nur angesprochen wird (deshalb der Beginn mit 1)
26 3 0 0 die INTEGER-Konstante 3 ist
47 die Funktion / ist
26 37 0 0die INTEGER-Konstante 25 ist (dezimal 37 entspricht Hex 25)
176 die Funktion SQR ist
10 bedeutet das Ende der Indices zu einer Array-Variablen, die 2 Indices hat und der etwas zugewiesen wird.
2 32 65 ist die Array-Variable A, die nur angesprochen wird (dehalb zu Beginn die 2)
26 1 0 0 ist die INTEGER-Konstante 1
11 ist das Ende der Indices zu einer Array-Variablen, die nur einen Index hat und auch nur angesprochen wird.
2 32 66 ist die Array-Variable B, die nur angesprochen wird
26 1 0 0 ist die INTEGER-Konstante 1
26 2 0 0 ist die INTEGER-Konstante 2
12 ist das Ende der Indices zu einer Array-Variablen, die zwei Indices hat und nur angesprochen wird
42 ist die Funktion *
8 ist das Zuweisungsbyte für einen numerischen Wert

An diesem Beispiel ist die Codierung der Array-Variablen gut zu sehen. Erst erfolgt die Angabe der Variablen mit Namen und dann erfolgt die Angabe der dazugehörigen Indices. genauso sieht es bei der Codierung einer Stringvariablen aus, wo nur ein Teil ausgewählt wird.
Auch hierzu ein kleines Beispiel:
A$ 2,5 = C$&B$ 1 ist codiert als
19 32 65 26 2 0 0 26 5 0 0 30 3 32 67 3 32 66 26 1 0 0 29 38 7, wobei
19 32 65 die Stringvariable A$ bedeutet, der etwas zugewiesen wird
26 2 0 0 die Integerkonstante 2 ist
26 5 0 0 die Integerkonstante 5 ist
30 ist das Ende der Indices einer Stringvariablen, bei der zwei Grenzen angegeben worden sind. (Bei Stringvariablen gibt es keine Unterschiede, ob der Variablen etwas zugewiesen wird oder nicht, im Gegensatz zu den Array-Variablen)
3 32 67 ist die Stringvariable C$
3 32 66 ist die Stringvariable B$
26 1 0 0 ist die Integerkonstante 1
29 ist das Ende der Indices einer Stringvariablen, bei der nur der Anfangsindex gegeben wurde)
38 ist Funktion &
7 ist Zuweisungsbyte für einen Stringwert

ef) Befehle, die zur Programmsteuerung notwendig sind:

1. Der IF-Befehl:

Im Zusammenhang mit dem IF-Befehl tauchen eine Reihe von Funktionen auf, die auch im Zusammenhang mit Wertzuweisungen zu numerischen Variablen hätten erläutert werden können.
Zunächst die Codierung dieser Funktionen:
Die Codierung dieser Funktionen "kleiner", "größer", "gleich" usw. und die dazugehörigen Werte werden auf die gleiche Weise codiert, wie auch numerische Funktionen mit zwei Variablen, zB. die Funktion *
Ein kleines Beispiel zeigt dies:
A AND B ist codiert als dezimal 1 32 65 1 32 66 213
A * B ist codiert als dezimal 1 32 65 1 32 66 42
Der Unterschied in der Codierung liegt also nur in der Funktion.
Einmal dezimal 213 für AND und dann dezimal 42 für *
Hier nun die einzelnen Codierungen:
Zunächst die Funktion, die nur einen Wert zur Ausführung benötigt:
Dezimal 208 = NOT
Dann die Funktionen, die zwei Werte zur Ausführung benötigen:

```
Dezimal  55 = "kleiner" bei Stringvergleichen
         58 = "kleiner gleich" bei numerischen Vergleichen
         59 = "größer gleich" bei numerischen Vergleichen
         60 = "ungleich" (weder größer noch kleiner) bei num. Vergleichen
         49 = "ungleich" (nicht gleich) bei Stringvergleichen
         53 = "gleich" bei Stringvergleichen
         61 = "gleich" bei numerischen Vergleichen
         62 = "größer" bei numerischen Vergleichen
         63 = "kleiner" bei numerischen Vergleichen
         50 = "kleiner gleich" bei Stringvergleichen
         51 = "größer gleich" bei Stringvergleichen
         52 = "ungleich" (weder größer noch kleiner) bei Stringvergleichen
         54 = "größer" bei Stringvergleichen
         57 = "ungleich" (nicht gleich) bei numerischen Vergleichen
        158 = OR
        207 = EXOR
        213 = AND
```

Nach diesem notwendigen Rüstzeug für Vergleiche, kann ich nun auf den Aufbau des IF-Befehls näher eingehen:
Für das Wort IF gibt es keine Codierung, dieses erscheint in der Anzeige nur als Interpretation des sonstigen Codes, der dazugehört (THEN und ELSE).
Im internen Code erscheint zunächst die Codierung für die Abfragen,
dann die Codierung für THEN,
dann die Codierung für die Befehle nach dem THEN,
dann falls vorhanden, die Codierung für das ELSE,
zuletzt die Befehlscodierung für die Befehle nach dem ELSE, falls ELSE vorhanden ist.
Für THEN und ELSE gibt es zwei Formate:

1. Format THEN (bzw. ELSE) plus Angabe der Zeilennummer ohne GOTO
   1.Byte dezimal 24 bei THEN und dezimal 31 bei ELSE
   2. bis 3.Byte Codierung der Zeilennummer
   Bsp: THEN 210 ist codiert als dezimal 24 16 02
   hexadezimal 18 10 02
2. Format THEN (bzw. ELSE) plus Befehle
   1.Byte dezimal 27 bei THEN und dezimal 28 bei ELSE
   2.Byte a)gibt es nur THEN, dann gibt dieses Byte die Länge bis zum Zeilenende an (=dezimal 14).
   b)gibt es THEN und ELSE, dann gibt dieses Byte bei THEN die Länge bis zum Befehlsanfang nach ELSE an, bei ELSE die Länge bis zum Zeilenende (=dezimal 14) an.
   3.Byte ist stets 0
   4.Byte und folgende geben die Befehlsfolge in der gewohnten Codierung an.

2. Die Sprungbefehle:
   1.Byte dezimal 90 bei GOTO und dezimal 91 bei GOSUB.
   2.Byte bis 3.Byte Codierung der Zeilennummer

3. Die bedingten Sprungbefehle:
 1.Byte dezimal 102 für ON
 2.Byte und folgende für die Codierung des Wertes, von dem der Sprungbefehl abhängt.
 nach diesen Bytes folgen die Sprungbefehle und zwar in der unter 2. angegebenen Codierung. Die Sprungbefehle stehen unmittelbar hintereinander.
4. andere Programmsteuerungen:
 Dezimal 138 = END
 113 = RETURN
 103 = BYE
 83 = POP
 117 = STOP
5. FOR-NEXT-Schleife:
 1.Byte dezimal 140
 2. bis 4.Byte Codierung des Schleifenindex (Schleifenvariable)
 Wert des 2.Byte ist dezimal 17, da der Variablen etwas zugewiesen wird.
 5.Byte und folgende Codierung für den Anfangswert
 das darauf folgende Byte enthält dezimal 8 für numerische Zuweisung
 die nächsten Bytes enthalten die Codierung für den Endwert
 das folgende Byte enthält den Wert dezimal 159 für TO
 Ist noch ein STEP-Wert angegeben, so enthalten die nachfolgenden Bytes die Codierung für diesen STEP-Wert,
 abgeschlossen wird diese Reihe dann durch das Byte mit dem Wert dezimal 206 für STEP.
 Nach der Codierung der Befehle innerhalb der Schleife, erfolgt die Codierung für NEXT:
 1. bis 3.Byte enthält den Schleifenindex (Schleifenvariable), wobei auch hier das 1.Byte den Wert dezimal 17 hat, da der Variablen etwas zugewiesen wird.
 4.Byte enthält den wert dezimal 143 für NEXT
 Bsp: 10 FOR I=1 TO B STEP 2
 20 A=A+I
 30 NEXT I
 ist intern codiert als dezimal

```
16 0 19 140 17 32 73 26 1 0 0 8 1 32 66 159 26 2 0 0 206 14  für Zeile 10
32 0 12 17 32 65 1 32 65 1 32 73 43 8                   14  für Zeile 20
48 0 5 17 32 73 143                                     14  für Zeile 30
```

 16 0 bzw. 32 0 bzw. 48 0 sin die Codierungen für die Zeilennummern.
 Danach die Codierung für die Befehle:
 Zeile 10:

```
     19 gibt die Länge der Zeile an
        140 ist die Codierung der FOR-NEXT-Schleife
            17 32 73 Variable I, der etwas zugewiesen wird
                     26 1 0 0 Integer-Zahl 1
                              8 Codierung für numerische Zuweisung
                                1 32 66 Variable B, die angesprochen wird
                                        159 Codierung für TO
                                            26 2 0 0 Integer-Zahl 2
                                                     206 Codierung für STEP
                                                         14 Ende der Zeile
```

 Zeile 20:

```
     12 gibt die Länge der Zeile an
        17 32 65 Variable A, der etwas zugewiesen wird
                 1 32 65 Variable A, die angesprochen wird
                         1 32 73 Variable I, die nur angesprochen wird
                                 43 Funktion +
                                    8 Codierung für numerische Zuweisung
                                                      14 Ende der Zeile
```

 Zeile 30:

```
     5 gibt die Länge der Zeile an
       17 32 73 Variable I, der etwas zugewiesen wird (nämlich I=I+2)
                143 Codierung für NEXT
                                                      14 Ende der Zeile
```

6. ON ERROR:
1.Byte dezimal 65 für ON ERROR
2.Byte dezimal 156 für Beginn des Funktionsteils
es folgen nun die Befehle in codierter Form, die unter der Bedingung eines Fehlers ausgeführt werden sollen.
Nach diesen Befehlscodierungen folgt das Abschlußbyte mit dem dezimalen Wert 16.

7. ON TIMER:
1.Byte und ff. gibt die Codierung für den Wert der Nummer des Timers an
Danach folgen die Bytes, die in codierter Form die Dauer angeben
Es folgen die Bytes mit den dezimalen Werten 100 zur Kennzeichnung des Befehls und 157 für den Beginn des Funktionsteiles.
Danach folgen wieder die Codierungen für die Befehle, die, wenn die Bedingung erfüllt ist, ausgeführt werden.
Schließlich folgt nach diesen Befehlscodierungen wieder das Abschlußbyte mit dem Wert dezimal 16.

8. OFF ERROR:
Dieser Befehl benötigt nur ein Byte mit dem Wert dezimal 66.

9. OFF TIMER:
Hier erfolgt zuerst die Codierung des Wertes für die Timernummer.
Dann folgt das Byte mit dem Wert dezimal 101, das den Befehl bezeichnet.

10. RUN, CALL und CONT:
Beim RUN-Befehl gibt es mehrere Möglichkeiten. Sie werden aber alle auf die gleiche Art und Weise codiert. Zunächst werden alle notwendigen Wertzuweisungen in den Stack gebracht, um den Befehl RUN ausführen zu können.
Danach folgt das Byte mit dem Wert dezimal 84 für RUN.
Bsp: RUN 'PGM',10 ist codiert als
150 3 80 71 77 26 16 0 0 84
150 Textkonstante, die in eifachen Hochkommas eingeschlossen ist.
3 Länge des Strings
80 71 77 Codierung für PGM
26 16 0 0 Integer-Konstante 10 (Dez.16 = Hex.10)
84 Codierung des Befehls RUN
Wichtig ist hier, daß die Zeilennummern nicht in der üblichen Zeilennummerncodierung codiert sind, sondern als Integer-Konstante. Dies gilt auch bei den Befehlen MERGE, AUTO, RENUMBER usw.
Beim CALL-Befehl erfolgt zunächst die Codierung für den Stringwert, dann das Byte mit dem Wert 79 für CALL.
Beim CONT-Befehl steht zuerst der Integer-Konstantenwert, dann das Byte mit dem Wert dezimal 148.

11. Editierbefehle:
Die unter 10. angegebene Codierung des RUN-Befehls steht beispielhaft für die Codierung der nun folgenden Befehle. Alle Werte werden dem eigentlichen Byte für den Befehl vorangestellt. Hier also nochmals die Codierungen für diese Befehle:
Dezimal 69 = AUTO
75 = LIST (nicht LIST IO)
78 = MERGE
87 = FETCH (nicht FETCH KEY)
108 = PLIST
129 = DELETE
137 = RENUMBER
Bsp: PLIST 10,20 ist codiert als
26 16 0 0 26 32 0 0 108
26 16 0 0 Integer-Konstante 10
26 32 0 0 Integer-Konstante 20
108 Codierung für PLIST

MERGE 'AB' ist codiert als
150 2 65 66 78
150 Textkonstante, die in einfachen Hochkommas eingeschlossen ist
2 Länge des Textes
65 66 Codierung für AB
78 Codierung für den Befehl MERGE

LIST keys ist codiert als

214 75

214 Konstante für keys

75 Codierung für LIST

Nun die Editierbefehle, die nichts mit Zeilennummern zu tun haben. Hier stehen ähnlich wie bei den vorhergehenden Befehlen die notwendigen Strings an Stelle der Zahlen im Stack. Dabei können anstatt der Strings auch Konstanten wie keys, appt usw. stehen.

Nun folgen die Codierungen für diese Befehle:

NAME: erst der Stringwert, dann das Byte mit dem Wert 76

RENAME: erst ein oder zwei Stringwerte, je nachdem ob der File workfile war oder nicht.
Danach ein Byte mit dem Wert dezimal 219 für TO
Schließlich das Byte mit dem Wert dezimal 126 für RENAME

EDIT: Zuerst kommt, wenn vorhanden, der Name des editierten Programmes nach der üblichen Codierung.
Dann folgt falls ein Filetyp angegeben ist ein Byte, das diesen Filetyp spezifiziert (151 für TEXT, 152 für BASIC)
Zum Schluß kommt das Byte, das den Befehl EDIT festlegt. Es hat den Wert dezimal 115.
Bsp: EDIT 'A',TEXT ist codiert als
150 1 65 151 115
150 Textkonstante, die in einfachen Hochkommas eingeschlossen ist
1 Länge des Strings
65 Codierung für A
151 Codierung für den Filetypen TEXT
115 Codierung für den Befehl EDIT

COPY: Die Codierung dieses Befehles erfolgt wie bei RENAME, nur steht hier anstelle des Bytes mit dem Wert 126 für RENAME das Byte mit dem Wert 124 für COPY.

PURGE: erst die Angabe des Stringwertes, falls vorhanden bei nicht workfiles, dann folgt das Byte mit dem Wert dezimal 125 für den Befehl PURGE.

CAT ALL: Codierung durch das Byte mit dem Wert dezimal 70

CAT: Filename oder appt bzw. keys durch Angabe der Codierung (Stringwert) in den Stack laden, dann zum Schluß das Byte mit dem Wert dezimal 74 für CAT

CAT$: ist eine Stringfunktion. Zuerst eine Integer-Konstante in den Stack laden. Dann zum Abschluß das Byte mit dem Wert dezimal 72 für CAT$.

FETCH KEY: Zunächst Codierung des Stringwertes im Stack, dann das Byte mit dem Wert dezimal 81 für FETCH KEY.

12. Befehle, die bestimmte Systemvariablen setzen:

BEEP: Zuerst Frequenz und Tonlänge im Stack laden, dann dezimal 144

DEFAULT: Dezimal 89 (ON und OFF wie bei ALARM)

DELAY: Wert im Stack, dann dezimal 77

ENDLINE: Dezimal 122

MARGIN: Wert im Stack, dann Byte mit Wert dezimal 93

PWIDTH: Printgröße codiert im Stack, danach Byte mit dem Wert dezimal 88

STANDBY: Dezimal 99 (ON und OFF wie oben)

WIDTH: Druckgröße codiert im Stack, dann Byte mit dem Wert dezimal 82

TRACE FLOW: Dezimal 119

TRACE OFF: Dezimal 120

TRACE VARS: Dezimal 121

13. Kommentare:

REM: 1.Byte dezimal 131 für REM bzw. dezimal 139 für !
2.Byte ist dezimal 6, da der nachfolgende String nicht in Hochkommas eingeschlossen ist.
3.Byte gibt die Länge des folgenden Kommentars an
4.Byte und folgende: Kommentar in ASCII-Codierung

14. IMAGE-Anweisung:

1.Byte dezimal 142 für IMAGE
2.Byte ist 6, da der nachfolgende Gesamtstring nicht in Hochkommas eingeschlossen ist.

3.Byte gibt die Länge des gesamten Strings der IMAGE-Anweisung an.
Hochkommas, die im String der IMAGE-Anweisung erscheinen, werden mit ihrem ASCII-Code dargestellt und zählen mit zum String dazu.

4.Byte und folgende enthalten die eigentliche Anweisung in ASCII-Codierung.

15. DATA-Anweisung:

1.Byte ist ein Byte mit dem Wert dezimal 134 für DATA.
ab dem 2.Byte werden ohne irgendwelche Trennbytes, wie sie zum Beispiel bei PRINT-Anweisungen auftreten, die Konstanten hintereinander dargestellt mit ihren entsprechenden Anfangsbytes:

4 für REAL-Konstante
5 für String in Doppelhochkommas
6 für String ohne Hochkomma
26 für INTEGER-Konstante
150 für String in einfachen Hochkommas

eine Längenangabe hinter DATA erfolgt nicht.

16. Deklaration von Variablen:

1.Byte: dezimal 85 für REAL
127 für INTEGER
128 für SHORT
136 für DIM

2.Byte und folgende:
Hier werden die Variablen in den üblichen Codierkonventionen ohne Trennbytes hintereinanderaufgeführt. Die vorangestellten Bytes und auch die nachgestellten Bytes bei Arrays entsprechen denen, die verwandt werden, wenn eine Variable nur angesprochen wird.

17. Aufbau der PRINT-Befehle:

Man unterscheidet zwei Arten von PRINT-Befehlen:

a) Der PRINT-Befehl:
1.Byte ist immer dezimal 107 für PRINT
-ohne USING geht es sofort mit der Codierung der auszugebenden Werte
-mit den dazugehörigen Trennbytes weiter -siehe unten-.
mit USING gibt es zwei Möglichkeiten:
1. USING Zeilennummer wird codiert
160 für USING und nachfolgend 2 Bytes für die Zeilennummer in der üblichen Zeilennummerncodierung.
2. USING mit Stringanweisungen wird codiert
zunächst folgen die Codierungen für die String-Anweisungen und anschließend wird das Byte mit dem Wert dezimal 203 für USING codiert.
nach diesen unterschiedlichen Codierungen für USING folgt dann aber einheitlich innerhalb von den USING-Befehlen das Byte dezimal 39 für ;

Das folgende ist wieder innerhalb der PRINT-Befehle gleich:
Es folgen die Druckanweisungen, dabei werden die Anweisungen in der normalen Codierung durchgeführt, lediglich die Trennbytes und die Endbytes unterliegen besonderen Regeln:
Byte 163 repräsentiert ein ; nach einer ALPHA-Druckanweisung
Byte 164 " , "
Byte 231 " ; nach einer numerischen Druckanweisung
Byte 232 " , "
Am Ende steht dann das Byte 162, wenn Carriage Return und Line Feed erzeugt werden sollen (oder was die ENDLINE-Anweisung vorgibt)
-Diese ganze Syntax gilt auch für den DISP-Befehl, nur hat dieser im 1.Byte immer den Wert 86 für DISP stehen anstatt der 107.

b) Der PRINT#-Befehl:
Zuerst erfolgt die Codierung für den Wert der Filenummer und der Zeilennummer.
Dann steht das Byte mit dem Wert dezimal 92 für PRINT#.
Danach steht, wenn vorhanden, Byte 39 für ;
Anschließend kommen die Codierungen für die Variablen.
Das vorangestellte Byte ist hier immer das Byte, das angibt, daß die Variable nur angesprochen wird. Ein ganzes Array ist hier wieder eine Ausnahme. In diesem Fall steht 181 oder 182 voran. (181 bedeutet, daß ein ganzes Array mit einem Index angesprochen wird, 182 bedeutet, daß ein ganzes Array mit zwei Indices angesprochen wird) Hinter den nun folgenden Arraynamen steht das Byte mit dem Wert 34.

Beim PRINT-Befehl steht zwischen mehreren Variablen immer das Byte mit dem Wert 166. Am Ende der Variablenliste stehen immer zwei Bytes, nämlich 165 und 167.
Wird der PRINT#-Befehl ohne Variablen codiert, so steht nach dem Byte 92 für PRINT#-Befehl das Byte 167.
Bsp: PRINT# 2;A,B(),C ist codiert als
26 2 0 0 92 39 1 32 65 166 181 32 66 34 166 1 32 67 165 167
26 Integerkonstante
2 0 0 Integerzahl 2
92 PRINT#-Befehl
39 Codierung für ;
1 Variable, die angesprochen wird
32 65 Variable A
166 Trennbyte
181 Array mit einem Index wird angesprochen
32 66 34 Array B
166 Trennbyte
1 32 67 Variable C, die nur angesprochen wird
165 167 Ende der Variablenliste

18. Der Aufbau der READ-Befehle:
Es gibt zwei READ-Befehle, entsprechend gibt es auch leichte Unterschiede in der Codierung.
a) Der READ-Befehl:
Das 1.Byte ist hier stets 110 für READ.
es folgen die Codierungen für die Variablen, denen etwas zugewiesen werden soll.
Hier tauchen die gleichen Zuweisungsbytes auf wie beim PRINT#-Befehl.
Zuweisungsbytes 181 bedeutet, daß ein ganzes Array mit einem Index angesprochen wird.
182 bedeutet, daß ein ganzes Array mit zwei Indices angesprochen wird
Bei diesen beiden Bytes gibt es keinen Unterschied, ob das Feld nur angesprochen wird oder ob im Feld etwas zugewiesen wird, denn beim PRINT#-Befehl stehen die gleichen Bytes vor einem ganzen Feld.
Wird ein ganzes Array angesprochen, so steht unmittelbar hinter dem Namen der Variablen ein Byte, das angibt, ob das ganze Array nur angesprochen wird (Wert 36 dahinter) oder ob dem Array etwas zugewiesen wird (Byte 34 hinter den restlichen Bytes dieses Befehls).
Beim READ-Befehl steht vor den Variablen das Byte, das angibt, daß etwas zugewiesen wird. (Ausnahme ganzes Array - siehe oben). Hinter einer numerischen Variablen wird immer Byte 200 codiert, hinter einer ALPHA-Variablen steht immer Byte 161. Hinter einem Array steht immer Byte 36.
Bsp: READ A,A$,A(,) ist codiert als
110 17 32 65 200 19 32 65 161 182 32 65 36
110 Codierung des READ-Befehls
17 32 65 Variable A, der etwas zugewiesen wird
200 Abschluß der numerischen Variablen A
19 32 65 Variable A$, der etwas zugewiesen wird
161 Abschluß der ALPHA-Variablen A$
182 32 65 Array A (ganz) mit 2 Indices
36 Abschluß des Arrays A

b) Der READ#-Befehl:
Der READ#-Befehl beginnt mit der Codierung für die Filenummer, anschließend steht die Codierung für die Zeilennummer (die auch fehlen kann).
Dann steht das Byte mit dem Wert dezimal 80 für READ#.
Wird der Befehl fortgesetzt, so steht dann stets das Byte mit dem Wert 39 für ;,anschließend steht die Codierung für die Variablen.
Der Aufbau ist hier genau wie unter a).
Bsp: READ# 2,3;A ist codiert als
26 2 0 0 26 3 0 0 80 39 17 32 65 200

```
26 2 0 0 ist die Konstante 2
         26 3 0 0 ist die Konstante 3
                  80 ist die Codierung für READ#
                     39 ist die Codierung für ;
                        17 32 65 ist die Variable A, der etwas zugewiesen
                                 wird
                                 200 Abschluß der numerischen Variable A
```

19. Der INPUT-Befehl:

Je nachdem in welcher Syntax der INPUT-Befehl vorliegt, werden zunächst der String für den Anforderungstext und der String für den Defaultwert der Variablen codiert. Letztere oder beide können auch wegfallen. Dann, und nur dann, steht das Byte mit dem Wert dezimal 95 für den INPUT-Befehl. Anschließend steht die Codierung für die Variablen. Dabei steht hier das erste Byte in der Form von Variablen, denen etwas zugewiesen wird. Hinter der Codierung für eine Variable steht bei einer ALPHA-Variablen stets das Byte 229 und bei einer numerischen Variablen das Byte 221. Am Ende eines INPUT-Befehls steht immer Byte 25.

Bsp: INPUT 'A=','1';A ist codiert als

```
150 2 65 61 150 1 49 17 32 65 221 25
150 Textkonstante, die in einfachen Hochkommas eingeschlossen ist
    2 Länge des Textes
      65 61 Codierung für A=
            150 Text in einfachen Hochkommas
                1 Länge der Textkonstante
                  49 Codierung für 1
                     17 32 65 Variable A, der etwas zugewiesen wird
                              221 Zeichen für numerische Variable
                                  25 Codierung für INPUT-Befehl
```

20. Selbstdefinierte Funktionen, wenn sie noch nicht im Programm initialisiert sind:

Dies ist wohl das komplizierteste, was es bei den BASIC-Files gibt. Hat man einen BASIC-File ganz neu editiert, so zeigen bei selbstdefinierten Funktionen etliche Bytes den Wert Null. Wenn das BASIC-File initialisiert ist, werden diese Bytes verändert (davon mehr unter 1.4.).
Man unterscheidet zwei Formen von DEF FN.:

a) DEF FN.. in einer Zeile, dh. ohne END DEF

1. Byte ist hier stets 135
2. bis 3.Byte gibt den Namen der Funktion im üblichen Code an.
4. bis 5.Byte haben die Werte 0 0
6. Byte gibt die Anzahl der Variablen verschlüsselt wieder.
   Es gilt hierbei folgende Regel:
   Die Anzahl der Variablen wird mit 2 multipliziert. Handelt es sich um eine ALPHA-Funktion, dann wird noch 1 auf den Wert dazuaddiert. Die Codierung erfolgt im Hex-Code.

ab dem 7.Byte steht die Codierung für die Variablennamen. Diese weicht hier erheblich von der sonst üblichen Codierung ab. Es gelten folgende Regeln:

1. Byte gibt die Ziffer des Variablennamens (oder deren Fehlen) verschlüsselt an.
   Zunächst werden die Ziffern von 1 bis 9 mit ihren Werten 1 bis 9 bewertet Fehlt die Ziffer, wird dafür der Wert 10 genommen. Handelt es sich um eine numerische Variable, dann wird folgende Codierung wirksam:
   a) REAL-Variablen:
      Byte 1 besitzt den Wert der Ziffer in Hexcode
   b) INTEGER-Variablen:
      Byte 1 besitzt den Wert der Ziffer plus 16 in Hexcode
   c) SHORT-Variablen:
      Byte 1 besitzt den Wert der Ziffer plus 32 in Hexcode
   Handelt es sich um eine ALPHA-Variable, wird zu dem Wert der Ziffer 128 dazuaddiert (zB ist der Wert von Byte 1 bei A$ 10+128=138)

Handelt es sich um Arrays, sieht die Codierung wie folgt aus:
a) REAL-Array:
   Wert der Ziffer plus 64
b) INTEGER-Array:
   Wert der Ziffer plus 80
c) SHORT-Array:
   Wert der Ziffer plus 96

2.Byte gibt den Buchstaben verschlüsselt wieder:
Regel: A=1, B=2, C=3, ..., Z=26

Bei Alphavariablen kommen nun noch zwei Bytes, die die Länge der Stringvariablen angeben. Hat man nichts angegeben, so steht hier 32 0, ansonsten ist der Wert in der üblichen Längencodierung wie oben.

Bei Arrays kommen nach den 2 Bytes für die Codierung der Variablennamen noch insgesamt 8 Bytes.
3.Byte - 4. Byte dimensionierte Länge des Array
5.-6. Byte Codierung des 1.Index
7.-8. Byte Codierung des 2.Index
9.-10. Byte haben die Werte 0 0
wobei bei OPTION BASE 0 im Vergleich zu OPTION BASE 1 128 zur Länge des Array in Byte 3-4 addiert ist.

Bei numerischen Variablen reichen die 2 Bytes für die Codierung des Variablennamens aus.
Für jede Varaible, gleich welchen Typs, kommen nun zwei Bytes mit den Werten 0 0, solange das Programm noch nicht initialisiert ist.
Wenn nun nach diesm Schema alle Variablen aufgeführt sind, werden diese mit zwei Nullbytes abgeschlossen.
Danach kommt die Codierung für die Befehle, es gibt kein Zuweisungsbyte.
Nach den Befehlen erfolgt der Abschluß mit dem Byte 175 und zwei Nullbytes.
Bsp: DEF FNA$(A,B$ 12000 ,C) = B$(A,C) ist codiert als
135 32 65 0 0 7 10 1 0 0 138 2 224 46 0 0 10 3 0 0 0 0 3 32 66 1 32 65 1 32 67 30 175 0 0

```
135 Codierung für DEF FN
    32 65 Variable A
          0 0 werden beim Initialisieren ersetzt (siehe 1.4.)
              7 bedeutet 3 Variablen, 1 ALPHA-Funktion
                10 1 Funktionsvariable A
                     0 0 siehe oben
                         138 2 Funktionsvariable B$
                               224 46 Längenangabe von B$
                                      12000 ist hex 2E E0 (=46 224)
                                      0 0 siehe oben
                                          10 3 Funktionsvariable C
                                               0 0 0 0 siehe oben
                                                       3 ALPHA-Variable, die
                                                         nur angesprochen wir
                                                         32 66 Variable B$
                                                               1 32 65 num.
                                                               Variable A
1 32 67 numerische Variable C, die nur angesprochen wird
        30 Abschluß der Indices
           175 Ende der Funktionendefinition
               0 0 siehe oben
```

b) DEF FN.. über mehrere Zeilen:
1. Byte ist auch hier 135
die ganze Syntax für die Varaiblendefinition gilt auch hier.
Bsp: DEF FNA$ (A,B$ 12000 ,C)
     FNA$ = B$(A,C) ist codiert als
     END DEF

Bei der nun folgenden Codierung ist nur die Codierung der Funktionen wichtig:

```
135 32 65 0 0 7 10 1 0 0 138 2 224 46 0 0 10 3 0 0 0 0 für die 1.Zeile
135 Codierung für DEF FN
    32 65 Variable A
          0 0 siehe oben
              7 3 Variablen, 1 Alphafunktion
                10 1 Funktionsvariable A
                     0 0 siehe oben
                         138 2 224 46 Funktionsvariable B$ 12000
                                      0 0 siehe oben
                                          10 3 Funktionsvariable C
                                               0 0 0 0 siehe oben
68 3 32 65 3 32 66 1 32 65 1 32 67 30 7                  für die 2.Zeile
68 Codierung für Funktionszuweisung
   3 Alphavariable, die nur angesprochen wird
     32 65 Alphavariable A$
           3 32 66 Alphavariable B$, die nur angesprochen wird
                   1 32 65 numerische Variable A, die angesprochen wird
                           1 32 67 numerische Variable C, die angespr. wird
                                   30 Abschluß der Indices
                                      7 Zuweisungsbyte für ALPHA-Zuweisung
133 0 0                                                  für die 3.Zeile
133 Codierung für END DEF
    0 0 siehe oben
```

Anhand dieses Beispieles wird die Codierung klar.
Da es noch die Möglichkeit gibt, LET FN..= anzugeben, steht bei solcher Art von Codierung für 68 das Byte 97.
Neben der Definition wird die Funktion aber auch im Programm aufgerufen. Auch hier gibt es eine besondere Codierung.
Zunächst erfolgt die Codierung für die Variablen innerhalb der Klammern mit dem Byte am Anfang, das für angesprochene Variablen gilt. Dann steht bei numerischen Funktionen das Byte 22, bei Alphafunktionen das Byte 23. Dann folgt die Anzahl der Variablen in der Klammer. Danach folgt eine Liste von Bytes und zwar für jede Variable in der Klammer ein Byte, 128 für numerische Variablen, 129 für Alphavariablen.
Bsp: R$=FNA$(5,R$,9) ist codiert als

```
19 32 82  26 5 0 0 3 32 82  26 9 0 0 23 3 128 129 128 7
19 32 82 Variable R$, der etwas zugewiesen wird
         26 5 0 0 Integerkonstante 5
                  3 32 82 Variable R$, die angesprochen wird
                          26 9 0 0 numerische Konstante 9
                                   23 Alphafunktion
                                      3 Anzahl der Variablen
                                        128 129 128 Bytes für die Var.
                                                    7 Zuweisungsbyte
```

21. Andere Funktionen und Befehle:
Bei der großen Menge von Befehlen und Variationen fehlen natürlich einige Funktionen, die nicht einzuordnen sind.

RESTORE — Hier werden, wenn Werte angegeben sind, zuerst die Werte in der Form der Integerkonstanten angegeben, dann folgt Byte 112 für RESTORE

RESTORE# — Zuerst erfolgt die Codierung für den Wert der Filenummer, dann, wenn angegeben, der Wert für die Zeilennummer, dann das Byte 94 für RESTORE#.

ASSIGN# TO filename ist folgendermaßen codiert:
Zuerst die Codierung für die Filenummer
dann die Codierung für den Filenamen
dann Byte 219 für TO und Byte 96 für ASSIGN#

ASSIGN# TO * Zuerst die Codierung für die Filenummer, dann 6 1 42 219 96

ASSIGN# TO ''Zuerst die Codierung für die Filenummer, dann 150 0 219 96

DEF KEY — Hier folgt zuerst die Codierung für den String, der einer Taste normal zugeordnet ist, dann folgt der String für die neue Codierung der Taste, dann das Byte 67 für DEF KEY
Ist am Ende ein Semikolon codiert, dann erscheint als letztes Byte 39 für ;

TAB — Hier erscheint zunächst die Codierung für die Positionierung des Cursors, dann folgt das Byte 205 für TAB.

PROTECT — Byte dezimal 105

UNPROTECT — Byte dezimal 114

PUT — Hier steht zuerst die Codierung für den String, der in den INPUT-Buffer gestellt werden soll, dann das Byte 118 für PUT.

WAIT — Hier steht zuerst der codierte Zahlenwert, dann Byte 104 für WAIT.

CLEAR VARS — Byte dezimal 123

RANDOMIZE — Hier steht zuerst die Codierung für den Wert, der der Funktion mit übergeben werden soll - dieser kann auch fehlen -, dann das Byte 109 für RANDOMIZE.

22. Befehle, die die Peripherie ansprechen:
Eine Vorbemerkung zur Codierung: CAT ':tp' wird genauso codiert wie CAT'abc'. In dem einen Stringwert steht halt :tp statt abc. Befehle, die sich nur in den unterschiedlichen Strings unterscheiden, ansonsten aber gleich sind, werden hier nicht aufgeführt, wie zB. COPY.

ASSIGNIO — Byte dezimal 146

LISTIO — Byte dezimal 71

PRINTER IS* ist codiert als 6 1 42 106
6 Textkonstante, die in keinen Hochkommas steht
1 Länge des Textes
42 Codierung für *
106 Codierung für PRINTER IS

PRINTER IS''ist codiert als 150 0 106
150 String in einfachen Hochkommas
0 Länge des Strings
106 Codierung für PRINTER IS

DISPLAY IS* ist codiert als 6 1 42 73
6 Textkonstante, die in keinen Hochkommas steht
1 Länge des Textes
42 Codierung für *
73 Codierung für DISPLAY IS

DISPLAY IS''ist codiert als 150 0 73
150 String in einfachen Hochkommas
0 Länge des Strings
73 Codierung für DISPLAY IS

OFFIO — Dezimal 116

RESTOREIO — Dezimal 111

CLEAR LOOP — Dezimal 147

CLEAR (devices) Dezimal 149

23. Befehle, die nicht unmittelbar im System-ROM enthalten sind:
Zum Schluß fehlen nur noch die Befehle, die nicht im eigentlichen Betriebssystem stehen, sondern in den 8k Zusatz-ROM (siehe Innenleben des HP-75C). Zu diesen Befehlen gehören auch die Befehle der ROM-Bank, dh. von Zusatz-ROMs,und die Befehle aus LEX-Files. Die Eigenart dieser Befehle ist, daß ihnen das Byte 180 vorgeschoben ist.

a) Befehle im ROM-Bereich, die mit 180 1 0 beginnen:
180 1 0 1 ALARM (ON und OFF sind hier als Konstanten anzusehen und werden mit 168 bzw. 169 vor dem Befehl codiert)
180 1 0 2 LOCK
180 1 0 3 OPTION ANGLE DEGREES
180 1 0 4 OPTION ANGLE RADIANS

b) Befehle im ROM-Bereich, die mit 180 2 0 beginnen:
180 2 0 1 TRANSFORM (erst die Angabe über den (die) Stringwert(e), dann ein Byte mit dem Wert für die Fileart, dann Byte 155 für INTO, danach die Codierung des Befehls 180 2 0 1)
180 2 0 2 PACK
180 2 0 3 INITIALIZE
180 2 0 4 TIME$
180 2 0 5 DATE$
180 2 0 6 TIME
180 2 0 7 DATE

180 2 0 8 ANGLE
180 2 0 9 ACOS
180 2 0 10 ATN
180 2 0 11 ASIN
180 2 0 12 RMD
180 2 0 13 MOD
180 2 0 14 RND

c) LEX-File MCOPY

180 1 80 1 MCOPY

Syntax: MCOPY ':master' TO ':copy1 (,:copy2,..) '
'N:master' 'ALL'
'P:master'

Durch MCOPY werden festgelgte Files (N=non-private, P=private) auf angegebene Massenspeichermedien kopiert (entweder Angabe der device-Codes oder Kopien an alle Einheiten (ALL)).

d) LEX-File INSTALL

180 2 80 1 INSTALL

Syntax: INSTALL 'filename:device code'

Mit Hilfe dieses Befehles werden mit Paßwort auf Massenspeichereinheiten gespeicherte Files in den Rechner als privat geschützte Files geladen und zwar ohne Kenntnis des Paßwortes.

e) LEX-File PMSINSTR

Dieses LEX-File ist hilfreich bei der Arbeit mit PMS-Systemen.

180 3 80 1 BUILD
Syntax: BUILD 'bankname',Bankgröße,'filename'
Mit diesem Befehl wird im PMS-Speicher eine festgelegte Bank für ein File reserviert.

180 3 80 2 PRIVATE
Syntax:
Privatisierung eines Files im PMS-ROM

180 3 80 3 CHECKSUM
Syntax:
Dieser Befehl bestimmt eine Prüfsumme,um die Größe einer Bank zu bestimmen.

180 3 80 4 ROMAVAIL
Syntax:
Berechnung des ungenutzten, noch zur Verfügung stehenden Bereiches

180 3 80 5 ROMID
Syntax:
Zuweisung einer Identität an eine PMS-Bank

180 3 80 6 ROMSIZE
Syntax: ROMSIZE 'filename'
Berechnung der Anzahl von Bytes, die für ein File zur Abspeicherung im PMS benötigt werden.

180 3 80 7 PMSREV
Syntax:

f) LEX-File AUTOSTRT:

180 35 80 1 AUTOSTRT
Bei Drücken der ATTN-Taste, um den Rechner einzuschalten, wird der im Text-File angesprochene Befehl ausgeführt.

g) LEX-File RIOWIO

180 19 64 1 WIO
Syntax: WIO (IL-Register),Daten
Dieser Befehl dient zum Schreiben von eigenen IL-Rahmen in den IL-Registern (0-7)

180 19 64 2 RIO
Syntax: RIO (IL-Register)
Dieser Befehl dient zum Lesen des Inhaltes von IL-Registern

h) LEX-File PEKEPOOK

180 20 64 1 POOK
Syntax: POOK (Adresse),Inhalt
Durch diesen Befehl wird in den Bereich 8000-FFFF des Betriebssystems ein bestimmter Inhalt an die Adresse geschrieben.

180 20 64 2 PEKE
Syntax: PEKE (Adresse dezimal)
Lesen des Inhaltes aus einer Adresse (8000-FFFF)

i) LEX-File PEEKPOKE

180 21 64 1 POKE
Syntax: POKE (dezimale Adresse), Inhalt
wie POOK, jedoch für Bereich 0000-7FFF

180 21 64 2 PEEK
Syntax: PEEK (dezimale Adresse)
Lesen des Inhaltes aus einer Adresse (0000-7FFF)

j) LEX-FIle HPILCMDS

180 25 64 1 SENDIO
Syntax: SENDIO 'Device-Codes','Kommandos','Daten'
Dieser Befehl dient zum Senden von Statusmeldungen und Daten in die IL-Schleife und an einzelne IL-Einheiten

180 25 64 2 ENTIO$
Syntax: ENTIO$('Device-Codes','Kommandos')
Mit Hilfe dieses Befehles werden Daten und Statusmeldungen aus der IL-Schleife und von einzelnen IL-Einheiten empfangen.

180 25 64 3 SEND?
Syntax: SEND?
Dieser Befehl sendet noch restliche Datenteile, wenn der Empfang aus der IL-Schleife durch Fehlermeldungen an einzelnen Einheiten unterbrochen war.

k) LEX-File MUSIC

180 9 128 1 MUSIC
Syntax: MUSIC
Bei Ausführung dieses Befehles wird das Tastenfeld umgewandelt in eine Tastatur, mit der man harmonische Töne spielen kann, und zwar wird der Ton so lange gehalten, wie die Taste gedrückt wird. Durch Drücken der ATTN-Taste kann der alte Status wieder hergestellt werden.

1) LEX-File AUTOLOOP

180 8 64 1 AUTOLOOP OFF
Syntax: AUTOLOOP OFF
Dieser Befehl löscht die automatische Benennung der IL-Einheiten und ihre Zuweisung.

180 8 64 2 AUTOLOOP ON
Syntax: AUTOLOOP ON
Dieser Befehl bewirkt, daß beim Einschalten des Rechners die IL-Schleife neu adressiert wird.

180 8 64 3 AUTOLOOP
Syntax: AUTOLOOP
Mit diesem Befehl wird eine automatische Adressierung durchgeführt.

180 8 64 4 LISTIO$
Syntax: LISTIO$
Mit Hilfe dieses Befehls kann einer ALPHA-Variablen eine Liste der IL-Einheiten zugewiesen werden. Die Namen der Einheiten sind jeweils durch Doppelpunkt voneinander getrennt.

Nun noch die Codierungen für die Funktionen des MATHE-ROMs. Die Erklärung der Befehle muß im Handbuch des MATHE-ROMs nachgeschlagen werden.
Man unterscheidet zwei Gruppen von Befehlen:
- Befehle mit der Systembasis 180 3 32
- Befehle mit der Systembasis 180 23 32

m) Befehle mit der Systembasis 180 3 32

1. 180 3 32 1 REDIM (nachfolgend die Codierung der Arrays, abschließend 180 3 32 139)
2. Real Scalar Functions:
   180 3 32 6 LOG2
   180 3 32 7 LOGA (die Codierung der Variablen/Konstanten erfolgt normal)
   180 3 32 14 ROUND
   180 3 32 15 TRUNCATE
   180 3 32 16 FACT

3. Base Conversions:
   180 3 32 17 BVAL
   180 3 32 18 BSTR$
4. nochmals Real Scalar Functions:
   180 3 32 8 SINH
   180 3 32 9 COSH
   180 3 32 10 TANH
   180 3 32 11 ASINH
   180 3 32 12 ACOSH
   180 3 32 13 ATANH

   Die Codierung der Variablen muß vor der Codierung für die Funktion stehen.

5. Array-Input and Output:
   Die folgenden Befehle sind Zuweisungsbefehle, ihre Codierung beginnt immer mit MAT Array =
   Dies ist codiert als 180 3 32 4.
   Dann folgt die Zuweisung und zwar erst die Codierung für das Array, dann der interne Doppelbefehl.
   Ausnahme: MAT A=B ist codiert als
   180 3 32 4 2 32 65 2 32 66 180 3 32 127, wobei
   180 3 32 4 Codierung für MAT
   2 32 65 Arrayvariable A, die nur angesprochen wird
   2 32 66 Array-Variable B, die nur angesprochen wird.
   180 3 32 127 MAT-Zuweisung ohne Doppelbefehl

   Bei den Befehlen, bei denen eine zusätzliche Angabe einer Redimensionierung möglich ist, gibt es immer 3 Codierungen für einen Befehl, je nachdem, ob ohne subscript, mit 1 oder 2 subscripts.
   So ist CON codiert als
   180 3 32 116 ohne subscript
   180 3 32 117 mit 1 subscript
   180 3 32 118 mit 2 subscripts
   IDN ist codiert als
   180 3 32 122 ohne subscript
   180 3 32 123 mit 1 subscript
   180 3 32 124 mit 2 subscripts
   ZER ist codiert als
   180 3 32 119 ohne subscript
   180 3 32 120 mit 1 subscript
   180 3 32 121 mit 2 subscripts
   Bsp: MAT A=CON(B) ist codiert als
   180 3 32 4 2 32 65 2 32 66 180 3 32 117 180 3 32 126, wobei
   180 3 32 4 Codierung für MAT
   2 32 65 Array-Variable A, die nur angesprochen wird
   2 32 66 Array B, die nur angesprochen wird
   180 3 32 117 Codierung für CON (1 subscript)
   180 3 32 126 MAT-Zuweisung mit Doppelbefehl

   Bei der Zuweisung mit einem Numeric Expression erfolgt die Codierung
   180 3 32 134 180 3 32 141
   Bsp: MAT A=(X) ist codiert als
   180 3 32 4 2 32 65 2 32 88 180 3 32 134 180 3 32 141, wobei
   180 3 32 4 Codierung für MAT
   2 32 65 Array A, die nur angesprochen wird
   2 32 88 Array X, die nur angesprochen wird
   180 3 32 134 Codierung für num. Ausdruck
   180 3 32 141 Zuweisung mit num. Ausdruck

   Der Array-INPUT ist wie folgt codiert:
   Es beginnt mit 180 3 32 2 für MAT INPUT, dann die Codierung für ein Array oder mehrere Arrays, die dann immer mit 180 3 32 138 abgeschlossen sind.
   Bsp: MAT INPUT A,B ist codiert als
   180 3 32 2 2 32 65 180 3 32 138 2 32 66 180 3 32 138
   (Erklärung siehe oben)

Der Array READ ist wie INPUT codiert, nur steht statt 180 3 32 2 bei READ 180 3 32 3.
Bsp: MAT READ A,B ist codiert als
180 3 32 3 2 32 65 180 3 32 138 2 32 66 180 3 32 138
(Erklärung siehe oben)
Der Array OUTPUT beginnt immer mit 180 3 32 4 (für MAT), es folgt die Codierung des entsprechnden Befehls (PRINT, PRINT USING, DISP und DISP USING), wie im normalen Betriebssystem codiert. Der normalen Arraycodierung nach dem Befehl folgen immer die "Trennbytes"
180 3 32 136 bei Trennung mit Komma
180 3 32 137 bei Trennung mit Semikolon
Der Abschluß erfolgt wie im normalen Betriebssystem.
Bsp: MAT PRINT A;B ist codiert als
180 3 32 4 107 2 32 65 180 3 32 137 2 32 66 162, wobei
180 3 32 4 Codierung für MAT
107 Codierung für PRINT
2 32 65 Array A, die nur angesprochen wird.
180 3 32 137 Trennbytes für Trennung mit ;
2 32 66 Array B, die nur angesprochen wird
162 Endbyte für CR und LF

6. Matrix-Algebra:
Hier handelt es sich wieder um Doppelbefehle.
Vor der Zuweisung steht 180 3 32 4 für MAT.
Nach der Zuweisug folgt bei den meisten Befehlen abschließend 180 3 32 126.
MAT A=-B ist charakterisiert durch 180 3 32 130 für -
Bsp: MAT A=-B ist codiert als
180 3 32 4 2 32 65 2 32 66 180 3 32 130 180 3 32 126, wobei
180 3 32 4 Codierung für MAT
2 32 65 Array A, die nur angesprochen wird
2 32 66 Array B, die nur angesprochen wird
180 3 32 130 Codierung für -
180 3 32 126 Zuweisung bei Doppelbefehlen
Bei der Arithmetik wird nun die Bytekombination 180 3 32 130 für - ersetzt, je nach der gewünschten Funktion.
Addition: 180 3 32 129
Bsp: MAT A=B+C ist codiert als
180 3 32 4 2 32 65 2 32 66 2 32 67 180 3 32 129 180 3 32 126

Subtraktion: 180 3 32 131
Bsp: MAT A=B-C ist codiert als
180 3 32 4 2 32 65 2 32 66 2 32 67 180 3 32 131 180 3 32 126

Multiplikation: 180 3 32 128
Bsp: MAT A=B*C ist codiert als
180 3 32 4 2 32 65 2 32 66 2 32 67 180 3 32 128 180 3 32 126

Skalarmultiplikation: 180 3 32 132
Bsp: MAT A=(X)*B ist codiert als
180 3 32 4 2 32 65 2 32 88 180 3 32 133 180 3 32 141 2 32 66 180 3 32 132

Nun noch ein paar Funktionen, die nach folgendem Schema codiert sind:
180 3 32 4, Codierung aller Arrays, Codierung der Funktion, 180 3 32 126

Funktionen:
180 3 32 97 INV
180 3 32 99 TRN
180 3 32 101 RSUM
180 3 32 102 CSUM
180 3 32 111 CROSS
Bsp: MAT A=CSUM(B) ist codiert als
180 3 32 4 2 32 65 2 32 66 180 3 32 102 180 3 32 126

7. Real-Valued Matrix Functions:
   Da hier keine MAT-Zuweisung erfolgt, ist hier immer nur eine Befehlskombination anzutreffen. Die Codierung sieht wie folgt aus:
   Zuerst werden die Werte zur Ausführung zur Verfügung gestellt, danach folgt die Codierung der Funktion (vgl. mit Funktionen wie SIN, COS, MAX, MIN (siehe 1.3. eb) und ec))).
   Funktionen:
   180 3 32 19 FNORM
   180 3 32 20 CNORM
   180 3 32 21 RNORM
   180 3 32 22 DOT
   180 3 32 23 DETL
   180 3 32 24 DET
   180 3 32 25 LBND
   180 3 32 26 UBND
   180 3 32 27 SUM
   180 3 32 28 ABSUM
   180 3 32 29 MAXAB
   180 3 32 30 AMAX
   180 3 32 31 MINAB
   180 3 32 32 AMIN
   Bsp: DET(A) ist codiert als
   1 32 65 180 3 32 24, wobei
   1 32 65 die num. Variable A, die nur angesprochen wird
   180 3 32 24 Codierung für DET
   Bsp: A=DOT(X,Y) ist codiert als
   17 32 65 1 32 88 1 32 89 180 3 32 22 8, wobei
   17 32 65 num. Variable A, der etwas zugewiesen wird
   1 32 88 num. Variable X, die nur angesprochen wird
   1 32 89 num. Variable Y, die nur angesprochen wird
   180 3 32 22 Codierung für DOT
   8 Zuweisungsbyte für num. Variable
8. LU Decomposition:
   Diese Funktion ist auch nach folgendem Schema codiert:
   180 3 32 4, Codierung der Arrays, 180 3 32 103, 180 3 32 126
9. Solving a System of Equations
   180 3 32 4, Codierung der Arrays, 180 3 32 112, 180 3 32 126
10. Complex Variables:
    Es gilt bei allen Funktionen folgendes Codierungsschema:
    180 3 32 4, Codierung der Arrays, Codierung der Funktion, 180 3 32 126
    Codierungen der Funktionen:
    180 3 32 77 CRTOP
    180 3 32 78 CPTOR
    180 3 32 91 CONJ
    180 3 32 92 CRECP
    180 3 32 106 CADD
    180 3 32 107 CSUB
    180 3 32 108 CDIV
    180 3 32 109 CMULT
    Bsp: MAT Z=CSUB(W,U) ist codiert als
    180 3 32 4 2 32 90 2 32 87 2 32 85 180 3 32 107 180 3 32 126
11. Complex Functions
    Schema:
    180 3 32 4, Codierung der Arrays, Codierung der Funktion, 180 3 32 126
    Codierung der Funktionen:
    180 3 32 79 CACOSH
    180 3 32 80 CACOS
    180 3 32 81 CASINH
    180 3 32 82 CASIN
    180 3 32 83 CATNH
    180 3 32 84 CATN
    180 3 32 85 CCOSH
    180 3 32 86 CCOS
    180 3 32 88 CEXP

180 3 32 90 CLOG
180 3 32 93 CSINH
180 3 32 94 CSIN
180 3 32 95 CTANH
180 3 32 96 CTAN
180 3 32 104 CSQR
180 3 32 110 CPOWER
180 3 32 115 CROOT
Bsp: MAT Z=CASINH(W) ist codiert als
180 3 32 4 2 32 90 2 32 87 180 3 32 81 180 3 32 126

12. Complex Matrix Operations:
Schema:
180 3 32 4, Codierung der Arrays, Codierung der Funktion, 180 3 32 126
Codierung der Funktionen:
180 3 32 87 CDET
180 3 32 89 CINV
180 3 32 100 CTRN
180 3 32 112 CSYS
180 3 32 114 CMMULT
180 3 32 125 CIDN
Bsp: MAT Z=CDET(A)ist codiert als
180 3 32 4 2 32 90 2 32 65 180 3 32 87 180 3 32 126

13. Finding Roots of Polynomials:
Schema der Funktion PROOT:
180 3 32 4, Codierung der Arrays, 180 3 32 98 180 3 32 126
Bsp: MAT R=PROOT(P) ist codiert als
180 3 32 4 2 32 82 2 32 80 180 3 32 98 180 3 32 126

14. Die Finite Fourier Transform:
Schema der Funktion FOUR:
180 3 32 4, Codierung der Arrays, 180 3 32 105 180 3 32 126
Bsp: MAT W=FOUR(Z) ist codiert als
180 3 32 4 2 32 87 2 32 90 180 3 32 105 180 3 32 126

n) Befehle mit der Systembasis 180 23 32

1. Solving f(x)=0
Bei diesem und den folgenden Befehlen, bei denen eine user-definierte Funktion auftritt, weicht die Reihenfolge der Codierung etwas ab.
FNROOT (A,B,user-definierte Funktion(X))
Zunächst erfolgt die Codierung der Variablen A und B (Arrays),
dann folgen die Bytes 180 23 32 1,
dann erfolgt die Codierung für den Funktionsaufruf wie im normalen Betriebssystem, abschließend folgen die Bytes 180 23 32 6
Bsp: FNROOT(A,B,FNY(X)) ist codiert als
2 32 65 2 32 66 180 23 32 1 1 32 88 22 1 128 180 23 32 6, wobei
2 32 65 Array-Variable A, die nur angesprochen wird
2 32 66 Array-Variable B, die nur angesprochen wird
180 23 32 1 Codierung für FNROOT
1 32 88 num. Var. X, die angesprochen wird
22 num. Funktion
1 Anzahl der Variablen
128 Bytes für die Variablen
180 23 32 6 Cod. für FNROOT
Anstelle der Funktionscodierung 180 23 32 1 für FNROOT tritt bei FNGUESS die Codierung 180 23 32 2 auf.

2. Numerical Integration:
Es gilt das unter 1. gesagte:
Hier folgen nach der Codierung der Arrayvar. die Bytes 180 23 32 1 und nach der Codierung für die Funktion 180 23 32 5
Bsp: INTEGRAL (A,B,E,FNY(X)) ist codiert als
2 32 65 2 32 66 2 32 69 180 23 32 1 1 32 88 22 1 128 180 23 32 5, wobei
2 32 65 Array-Variable A, die nur angesprochen wird
2 32 66 Array-Variable B, die nur angesprochen wird
2 32 69 Array-Variable E, die nur angesprochen wird
180 23 32 1 Codierung für INTEGRAL
1 32 88 num. Var. X, die angesprochen w
22 num. Funktion

1 Anzahl der Variablen
128 Bytes für die Var.
180 23 32 5 Abschluß

Anstelle der Funktionscodierung 180 23 32 1 für INTEGRAL treten bei den übrigen Funktionen folgende Codierungen auf:
180 23 32 3 IVALUE
180 23 32 4 IBOUND

1.4. Änderung der Codierung der Befehle bei Initialisierung des Programms:

a) Angabe von aufgelösten Adressen:
Diese erfolgt wie alle Längenangaben (siehe 1.1.d)), und zwar wird hier stets die relative Anzahl von Bytes, vom Programmanfang gezählt, angegeben. Das erste Byte des Programms hat dann 0, das zweite 1, das dritte 2 usw..

b) Pointer am Programmanfang:
Es gibt insgesamt 5 Pointer am Programmanfang, sie sind stets 2 Bytes lang, da sie Längenangaben enthalten:
1.Pointer entspricht der Programmlänge des nicht initialisierten Programms.
2.Pointer enthält die Anzahl der Bytes, die hinter das Programm geschrieben werden bei der Initialisierung
3.Pointer ist Null.
4.Pointer enthält die Angabe der Länge aller Variablen.
5.Pointer ist Null.

c) Alle Line-Nummern bei GOSUB und GOTO werden durch die Anzahl der Bytes relativ vom Programmanfang bis zu der entsprechenden Line-Nummer ersetzt.
Für alle Variablen und selbstdefinierten Funktionen werden Bytes hinter das Programm angehängt, aus denen hervorgeht, wie die Variable heißt, wo sie zu finden ist usw. (siehe unten). Im Programm selbst werden die jeweiligen 2 Bytes, die den Programmnamen angeben, durch die Anzahl der Bytes relativ zum Programmanfang ersetzt, die angeben, wo die Definition der entsprechenden Variablen hinter dem initialisierten Programm steht.

d) Hier nun die Codierung der Variablendefinition nach dem Programmende:

da) Codierung der Namen:
Diese erfolgt in den ersten beiden Bytes der Definition. Das 1.Byte gibt die Art der Variablen an und die Zahl oder das Blank (zB. A1 und A) vom Variablennamen.
Das zweite Byte gibt den Buchstaben der Variablen an. Aus diesem Byte ergibt sich dann auch, ob hier eine Variable definiert wird, die von einer selbstdefinierten Funktion ausgegeben wird.
Beim ersetn Byte ist zunächst die Zahl des Variablennamens mit ihrem Wert zu nehmen, ein Blank wird hier mit dem Wert 10 angesetzt. Zu diesem Grundwert wird eine Konstante addiertm, aus der hervorgeht, um was für einen Variablentyp es sich handelt.
Die Konstanten im einzelnen:
0 numerische REAL-Variable
16 numerische INTEGER-Variable
32 numerische SHORT-Variable
64 REAL-Array-Variable
80 INTEGER-Array-Variable
96 SHORT-Array-Variable
128 ALPHA-Variable
Beim zweiten Byte wird der Buchstabe A mit dem Wert 1, B mit 2 usw. bis zum Z mit dem Wert 26. Handelt es sich um eine selbstdefinierte Funktion, so werden zu diesem Wert noch 32 dazuaddiert.
Bsp: -Die numerische REAL-Variable A besitzt die ersten beiden Bytes 10 1
-Die INTEGER-Array-Variable Z9 als selbstdef. Funktion besitzt die ersten beiden Bytes 89 58.

db) Beschreibung der folgenden Bytes:
Bei ALPHA-Variablen wird hier die Länge der Variablen angegeben (2 Bytes).
Bei selbstdefinierten Funktionen steht hier immer die Längge 32, selbst wenn die Länge, die ausgegeben wird, größer als 32 ist.
Dann folgen 2 Bytes, die eine relative Adresse angeben, wo die Variable im Speicher abgelegt wird (bei numerischen Variablen fallen die 2 Bytes für die Länge weg und diese Bytes für die Wdresse kommen dann direkt nach dem Namen).

Hierbei ist folgendes zu beachten:
Die erste Adresse lautet immer 00 1E (hexadezimal), also 1.Byte 1E, 2.Byte 00.
Die folgenden Adressen enthalten dann immer den Wert der vorhergehenden Adresse, zu der die Länge der vorhergehenden Variable addiert wird, mit einer Ausbahme: Für die Adresse der selbstdefinierten Funktionen sind noch zusätzlich 9 Bytes dazuzurechnen, da der Rechner sich vor der eigentlichen Variablen immer 9 Bytes für irgendwelche Rechnungen reserviert.
Bei Array-Variablen steht die Adresse allerdings erst in den Bytes 9 und 10. die Bytes 3 bis 8 sind für andere Zwecke reserviert (siehe unten).
Um von der Adresse, die bei einer selbstdefinierten Funktion steht, auf die nächste Adresse zu kommen, muß man bei einer numerischen Funktion 8 Bytes, bei einer ALPHA-Funktion 32 Bytes plus die Länge aller lokalen Variablen rechnen.
Bei einer selbstdefinierten Funktion folgen nun noch 2 Bytes, die die Länge, angegeben relativ zum Programmanfang bis zur Definition der lokalen Variablen, die hier ja mitten im Programm stehen und mit der verschlüsselten Variable beginnt.
Die Bytes 3 bis 8 bei Array-Variablen enthalten die zu erwartenden Angaben, wie Länge und Indices:
Bytes 3 und 4 enthalten die Gesamtlänge des Arrays in verschlüsselter Form;
liegt OPTION BASE 1 vor, ist die Länge normal verschlüsselt (siehe 1.1.b)),
liegt OPTION BASE 0 vor, so enthält Byte 4 einen um 128 höheren Wert.
Bytes 5 und 6 enthalten die Zahlenangabe des ersten Index, wie sie bei der Dimensionierung angegeben wurden (hexadezimal).
Bytes 7 und 8 enthalten die Angaben für den 2.Index. Hat das Array nur einen Index, so ist der Wert von Byte 7 und 8 hexadezimal FF.

dc) Nachtrag zur Verschlüsselung von selbstdefinierten Funktionen bei DEF FN.. und beim Ende der Funktion:
Bei Initialisierung des Programms werden die Bytes 2 und 3 durch die Längenangabe ersetzt, die angibt, wo die Variable definiert ist, die ausgegeben wird. Die gleiche Angabe wird auch am Ende der Funktion angegeben.
Die Bytes 4 und 5 geben die Länge an bis zum Ende der selbstdefinierten Funktion relativ zum Programmanfang. Dieses hilft wohl beim Überspringen des Codes der Funktion. Die nachfolgenden Bytes enthalten die Variablendefinition der lokalen Variablen, beginnend mit der verschlüsselten Anzahl, die Definition der Variablen erfolgt dann wie o. a. Im Anschluß an die Variablendefinition folgen noch zwei Bytes, die die Länge relativ zum Programmanfang bis zum eigentlichen Beginn der Definition der lokalen Variablen angeben (also nach der verschlüsselten Anzahl).

1.5. Darstellung der Variablen im Rechner:

a) ALPHA-Variablen:
ALPHA-Variablen beginnen mit 2 Bytes, die die aktuelle Länge der Variablen angeben, gefolgt von eigentlichen Bytes der Variablen, wobei die noch nicht belegten Bytes am Ende der Variablen mit dezimal 32 belegt sind.

b) numerische Variablen:

ba) INTEGER-Variablen:
Diese belegen 3 Bytes. Die Codierung erfolgt ähnlich wie bei den INTEGER-Konstanten im Programm.
Eine INTEGER-Variable kann 5 Ziffern aufnehmen. 3 Bytes können maximal 6 Ziffern aufnehmen. Diese 6.Stelle (entspricht dem linken Halbbyte des 3.Bytes) wird für das Vorzeichen benutzt. 0 bedeutet positiv, 9 negativ. Positive Zahlen sind wie die INTEGER-Konstanten codiert. Bei negativen Zahlen ist das Komplement zu 100000 abgespeichert.

bb) SHORT-Variablen:
Diese belegen 4 Bytes.
Das 1.Byte enthält den Absolutwert des Exponenten im BCD-Code. Die Bytes 2 bis 4 sind ähnlich codiert wie eine INTEGER-Konstante. Die Stelle für das Vorzeichen ist nun zwangsläufig anders codiert:
0: positive Zahl, positiver Exponent
1: positive Zahl, negativer Exponent
2: negative Zahl, positiver Exponent
3: negative Zahl, negativer Exponent

bc) REAL-Variablen:
Diese belegen 8 Bytes und sind ähnlich codiert wie die REAL-Konstanten. Hier dient das linke Halbbyte des 2.Bytes zur Codierung des Vorzeichens der Zahl: 0 positiv, 9 negativ. Der Exponent und die Mantisse sind identisch codiert wie eine REAL-Konstante.
Besonderheit: Solange im Programm einer REAL-Variablen nur INTEGER-Werte zugewiesen werden, unterteilt der Rechner die Variable in zwei Teile: die ersten 4 Bytes enthalten Zwischenwerte von INTEGER-Rechnungen mit dieser Variablen. Byte 5 enthält dezimal 255 und die Bytes 6-8 enthalten den eigentlichen Wert. Man kann deshalb schließen, daß dieser Bereich auch für Nebenrechnungen mitbenutzt wird.

bd) Array-Variablen:
Hier trifft das unter ba) bis bc) gesagte auch zu. Es werden im Rechner halt soviel Variablen des angegebenen Typs angelegt, wie bei der Dimensionierung angegeben wurde. Naturgemäß sind bei Arrays nicht alle Variablen mit Werten belegt, wie auch andere Vars noch nicht mit Werten belegt sein können (die Meldung NO VALUE sagt dies ja überdeutlich). Der Rechner erkennt dies an der Codierung der Variablen, die nun unter be) folgen.

be) Nicht belegte Variablen:
Bei der Programminitialisierung legt der Rechner sich alle Variablen an, die aber zunächst einen undefinierten Zustand besitzen, solange ihnen kein Wert zugewiesen wird. Dies erkennt der Rechner an dem 1.Byte bei num. Variablen und an den Bytes 1 und 2 bei ALPHA-Variablen. Die restlichen Bytes werden mit hex 0 bei numerischen und mit hex 20 bei ALPHA-Variablen aufgefüllt. Die Kennungsbytes enthalten den dezimalen Wert 255. Eine ALPHA-Variable fängt dann also mit 255 255 an.

2. Private BASIC-Files

2.1. Sicherung von Files auf Massenspeichern:
Man unterscheidet hier zwei Arten (PROTECT und UNPROTECT werden nicht beachtet, da sie im allgemeinen keinen Schutz vor fremdem Zugriff bieten: UNPROTECT hebt PROTECT wieder auf).

a) Sicherung durch Paßwörter:
Das Paßwort kann aus einem bis zu vier Buchstaben oder Ziffern bestehen. Wird ein Programm ohne Paßwort abgespeichert, so stehen an der Stelle der Buchstaben oder Ziffern vier Blanks (Dez.-Wert 32). Der Paßwort-Eintrag erfolgt nicht im eigentlichen Programm, sondern nur im Katalog des Programmes. Wenn man das Programm, das mit Paßwort geschützt ist, von einer Massenspeichereinheit einlesen will, muß das Paßwort im Filespezifikator mit angegeben werden; andernfalls erhält man eine Fehlermeldung - INVALID PASSWORD -, und das Programm kann nicht kopiert werden. Damit läßt sich der Programmgebrauch auf die Besitzer des Paßwortes beschränken. Ist das Programm aber wieder im Rechner, ist der Schutz weg. Möchte man ein Programm einlesen, dessen Paßwort nicht bekannt ist, kann man dies mit der von Hewlett-Packard erhältlichen LEX-Karte INSTALL tun, durch die ein mit Paßwort geschütztes Programm ohne Kenntnis des Paßwortes in den Rechner gespeichert wird, und zwar als ein Programm, das im Rechner durch ein Statusbyte geschützt wird. Auf Kassette läßt sich dieser Schutz auch anders entfernen, nämlich indem man die letzten 4 Zeichen eines Katalogeintrages (ab Record 2, Hilfsmittel 'PRREC'), der für jedes Programm 32 Zeichen lang ist, auf die Werte 32 setzt, dann läßt sich das Programm wie gewohnt in den Rechner laden.

b) Sicherung durch Statusbytes:
Dieses Statsubyte befindet sich im Header eines BASIC-Files als 5. Byte. Dieses Byte kann zwei Werte annehmen:
dezimal 254 bei non-privaten BASIC-Files
dezimal 202 bei privaten BASIC-Files
Hierbei findet im Katalog-Eintrag auf Band keine Veränderung statt. Durch Ändern dieses Statusbytes kann man selbst BASIC-Files schützen oder den Schutz aufheben.

2.2. Sicherung der Files im Rechner:
Grundsätzlich läßt sich im Rechner nur ein BASIC-File schützen (PB im Katalog). Da der Rechner ein Programm nur mit Header und Hauptteil speichert, also ohne den in der Massenspeichereinheit benutzten Zusatzkatalog, kann ein Programm nur durch das Statusbyte geschützt werden. Der Rechner geht bei der Editierung, Abspeicherung usw. eines Programms so vor, daß er erst den Header, der ja vom eigentlichen Hauptteil getrennt ist, nach dem Wert dieses Statsubytes überprüft. Dann entscheidet er je nach Wert dieses Statusbytes, ob dieser Befehl ausgeführt werden kann oder nicht.

3. Text-Files:

3.1. Aufbau von normalen Textfiles

3.1.1. Einzelbestandteile:

a) Header:

aa) Angabe der absoluten Adresse (Byte 1 und Byte 2):
siehe unter BASIC-Files

ab) Angabe der Programmlänge (Byte 3 und Byte 4)

ac) Wert des 5.Bytes ist immer dezimal 190

ad) Angabe der Fileart (Byte 6):
Dieses Byte hat für Text-Files den Hex-Wert 54 (ASCII-Code für T)

ae) Speicherung des Datums und der Zeit (Byte 7 bis 10):
siehe unter BASIC-Files

af) Angabe des Filenamens (Byte 11 bis 18)

b) Hauptteil des Text-Files:

ba) Pointer:
Auch hier haben die 5 Pointer zu je 2 Bytes den Wert 0 0.

bb) Textzeilen:

bba) Angabe der Zeilennummer:
Codierung der Zeilennummern ist gleich mit der Codierung in BASIC-Files.

bbb) Längenangabe des Textes:
Dieses Byte gibt auch Auskunft über die Länge einer Zeile, da keine eigenen Trennbytes zwischen den Zeilen vorhanden sind wie zB. das Byte mit dem Wert dezimal 14 bei BASIC-Files.

bbc) Codierung des Textes:
Der Text wird durch die Hex-Codes seiner ASCII-Zeichen repräsentiert.

3.1.2. Zusammenfügen der Einzelbestandteile zu der Gesamtheit des Text-Files:
Header und Hauptteil werden ohne Unterbrechung oder Zwischenbytes aneinandergefügt. Dies bedeutet,daß die Textzeilen auch direkt aufeinanderfolgen. Für das Ende des Text-Files ist auch hier eine eigene Schlußzeile vorgesehen mit folgendem Aufbau: 5 Bytes mit folgenden Hex-Werten 99 A9 02 8A OE. Was danach kommt, bringt der Rechner nicht mehr mit diesem Text-File in Verbindung.

3.2. Aufbau von keys-Textfiles

3.2.1. Einzelbestandteile:

a) Header:
Codierung wie unter 3.1.1.a) beschrieben

b) Hauptteil des keys-Text-Files:

ba) Pointer

bb) Textzeilen:

bba) Angabe des Dezimalwertes des Zeichens, dessen Taste neu belegt werden soll. Dies erfolgt in der gleichen Codierung wie die Codierung von Zeilennummern.

bbb) Angabe der Länge der Zuweisung, dh. der Länge des Textes, der für die Zuweisung benötigt wird.

bbc) Zeichen für Anhängen:
Dieses Byte hat immer den Wert dezimal 59.

bbd) Neubelegung der Taste:
Auch hier wird die Neubelegung durch die Hexcodes des ASCII-Textes repräsentiert.

3.2.2. Zusammenfügen der Einzelbestandteile:
Auch hier werden die Einzelzuweisungen direkt hintereinandergeschrieben, wie unter 3.1.2. Ende des Files, siehe unter 3.1.2.

4. LIF1-Files:

4.1. Einzelbestandteile:

Austauschfiles oder LIF1-Files dienen zum Austausch von Informationen zwischen dem HP-75 und anderen Computern. Jeder Textfile und nichtprivate BASIC-File im Speicher kann mit Hilfe des Befehles TRANSFORM in einen LIF1-File umgewandelt werden und dann auf Massenspeichern abgelegt werden. Das Ganze kann auch umgekehrt vollzogen werden. So können Informationen zwischen den Rechnern ausgetauscht werden. Das Besondere bei diesen Files ist, daß sie nicht editiert, gelistet und mit RUN oder CALL aufgerufen werden können.

a) Header:
LIF1-Files sind die einzigen Files, die keinen Header besitzen auf Massenspeichereinheiten. Im RAM des Rechners sieht der Header aus wie folgt:

aa) Angabe der absoluten Adresse (Byte 1 und 2):

ab) Angabe der Programmlänge (Byte 3 und Byte 4)

ac) Wert des 5.Bytes ist immer dezimal 140

ad) Angabe der Fileart:
Dieses Byte hat für LIF1-Files den Hex-Wert 49 (ASCII-Code für I)

ae) Speicherung des Datums und der Zeit (Byte 7 bis 10)

af) Angabe des Filenamens (Byte 11 bis 18)

b) Hauptteil des LIF1-Files:

ba) Pointer:
5 Pointer zu je 2 Bytes mit den Werten 0 0.

bb) Zeilen:

bba) Byte mit dem Wert dezimal 0

bbb) Längenangabe der LIF1-Zeile, wie unter 3.1.1.bbb)

bbc) Zeilennummer:
Die Zeilennummer sind anders als bisher codiert. Sie sind vierstellig und sie werden durch die 4 Hex-Codes ihrer ASCII-Zeilennummern dargestellt.
Bsp: Eine Zeilennummer von 10 ist codiert durch dezimal
48 48 49 48 (in ASCII-Form 0010)

bbd) Codierung des Textes:
Der Text wird durch die Hex-Codes seiner ASCII-Zeichen repräsentiert. Wird zB. im LIF1-File ein BASIC-Programm dargestellt, so stehen die BASIC-Befehle im Wortlaut (Bsp: DISP wird codiert durch Hex 44 49 53 50 (ASCII: DISP)).

4.2. Zusammenfügen der Einzelbestandteile:

Das Byte mit dem Wert 0 gibt gleichsam das Ende der einen und den Anfang der nächsten Zeile an. Danach werden die Einzelzeilen nur durch 0 0 getrennt hintereinander codiert. Hinter der letzten Zeile stehen zur Markierung des Endes des Files zwei Bytes mit den Werten 255 255. Was danach kommt, ist für den Rechner dann uninteressant.

5. APPT-Files:

5.1. Normale Appointments (N):

Bei normalen Appointments sieht das Display wie folgt aus:

```
Day Mo/Dy/Yr Hr:Mn AM #1N !Note
Day ist der Wochentag (MON, TUE, WED, THU, FRI, SAT, SUN)
    Mo ist der Monat (1..12)
       Dy ist der Tag (1..31)
          Yr ist das Jahr
             Hr ist die Stunde
                Mn ist die Minute
                   AM ist eine Anzeigeart (AM, PM, **)
                      # Alarmfeld
                       1 Alarmtyp 1 (1..9)
                        N normaler Alarm
                          ! steht für einfache Nachricht, > steht für Kommando
                           Note. Es kann eine Nachricht folgen
```

Wird nun ein Appointment-File initialisiert, so wird dieses File wie folgt im Memory des Rechners oder auf Magnetband gespeichert:

Byte 1 gibt die Länge des folgenden Appointments an

Byte 2-5 enthalten die Anzahl der Sekunden, wenn der Alarm fällig ist
von dieser Zahl muß das Systemdatum 0A375400 subtrahiert werden.

Byte 6 hat den Wert 14, wenn die interne Uhr nach einem Memory Lost auf eine Zeit gesetzt wurde,
hat den Wert 0, wenn die interne Uhr nach einem Memory Lost mit RTN ohne Eingabe von Datum und Zeit gesetzt wurde.

Byte 7 enthält den Wert für den Alarmtyp, falls der Alarm noch nicht vorbei ist
enthält den Wert für den Alarmtyp + 32, falls der Alarm schon vorbei ist

Byte 8 enthält den Wert 0, wenn keine Notiz oder Kommandos folgen.
enthält den Wert dezimal 33, falls eine Notiz folgt
enthält den Wert dezimal 62, falls ein Kommando folgt

Wenn Notizen oder Kommandos folgen, beginnen diese jetzt mit dem Byte 9, ansonsten endet hier der Eintrag.

5.2. Appointments für Wiederholungsalarm ohne Bestätigung (R):

Bei diesen Appointments sieht das Display so aus:

```
Day Mo/Dy/Yr Hr:Mn AM #1R !Note
                       R Wiederholungsalarm ohne Bestätigung
```

Nach Drücken von RTN ergibt sich folgende Anzeige:

```
Rept=Mo+Dy+Hr+Mn   I  DOW
Rept Wiederholungsalarm
     Mo Monat
        Dy Tag
           Hr Stunde
              Mn Minute
                      DOW In diesem Feld wird der Wochentag eingegeben,
                          an dem der Alarm wiederholt werden soll.
```

Wird nun dieser APPT-File initialisiert, so wird er wie folgt abgespeichert:

Byte 1 gibt die Länge des folgenden Appointments an

Byte 2-5 enthalten die Anzahl der Sekunden, die dann seit dem Systemdatum bis zum Fälligkeitszeitpunkt des Alarms vergangen sind. Von dieser Zahl wird deshalb das Systemdatum 0A375400 subtrahiert.

Byte 6 hat den Wert 14, wenn die interne Uhr nach einem Memory Lost auf eine Zeit gesetzt wurde,
hat den Wert 0, wenn die interne Uhr nach einem Memory Lost mit RTN gestartet wurde.

Byte 7 enthält den Wert für den Alarmtyp, zu dem 128 addiert wird.

Im Anschluß daran folgen zusätzlich 5 Bytes, die folgendermaßen codiert sind:

Byte 1 gibt den Tag an, der im Repeat-Feld angegeben wurde:

```
0 kein Tag wurde angegeben
16 SA   32 SU    48 MO   64 TU    80 WE  96 TH   112 FR
17 SA1  33 SU1   49 MO1  65 TU1   81 WE1 97 TH1  113 FR1
18 SA2  34 SU2   50 MO2  66 TU2   82 WE2 98 TH2  114 FR2
19 SA3  35 SU3   51 MO3  67 TU3   83 WE3 99 TH3  115 FR3
```

20	SA4	36	SU4	52	MO4	68	TU4	84	WE4	100	TH4	116	FR4
21	SA5	37	SU5	53	MO5	69	TU5	85	WE5	101	TH5	117	FR5
24	SA+	40	SU+	56	MO+	72	TU+	88	WE+	104	TH+	120	FR+
25	SA-	41	SU-	57	MO-	73	TU-	89	WE-	105	TH-	121	FR-

Byte 2 gibt die Anzahl der Monate an (BCD-Code)
Byte 3 bis 5 geben die Anzahl der Sekunden aller restlichen Felder aufaddiert im Hex-Code codiert an (ähnlich wie Datum und Uhrzeit).
Dann folgen die eingegebenen Textbytes, sofern eine Notiz angegeben wurde.
Das 1. Byte ist dann entweder das Ausrufezeichen oder das Größerzeichen, wenn ein Kommando ausgeführt werden soll. Die folgenden Bytes enthalten dann die Nachricht.

5.3. Appointments für Wiederholungsalarme mit Bestätigung (A):
Bei diesen Appointments sieht das Display so aus:

```
Day Mo/Dy/Yr Hr:Mn AM #1A !Note
                          A Wiederholungsalarm mit Bestätigung
```

Nach Drücken von RTN ergibt sich folgende Anzeige:

```
Rept=Mo+Dy+Hr+Mn    I  DOW
```

Wird nun dieser APPT-File initialisiert, so wird er wie folgt abgespeichert:
Byte 1 gibt die Länge des folgenden Appointments an.
Byte 2-5 enthalten die Anzahl der Sekunden, die dann seit dem Systemdatum bis zum Fälligkeitszeitpunkt des Alarms vergangen sind. Von dieser Zahl wird deshalb das Systemdatum 0A375400 subtrahiert.
Byte 6 hat den Wert 14, wenn die interne Uhr nach einem Memory Lost auf eine Zeit gesetzt wurde,
hat den Wert 0, wenn die interne Uhr nach einem Memory Lost mit RTN gestartet wurde.
Byte 7 setzt sich aus der Addition verschiedener Zahlen zusammen:
Alarmtyp + 196 für Wiederholungsalarm, sofern dieser Alarm noch nie fällig war.
Alarmtyp + 212 für Wiederholungsalarm, der schon einmal fällig war, aber im Moment nicht fällig ist
Es folgen wieder die 5 Bytes wie beim Wiederholungsalarm ohne Bestätigung.
Die Codierung dieser 5 Bytes ist wieder gleich.

Zum Abschluß noch eine Bemerkung zum Ende eines APPT-Files.
Diese Fileart besitzt nicht wie die anderen Files eine Endeanzeige, wie zB. bei BASIC-Files die Bytes 99 A9 02 8A 0E.

6. LEX-Files:

6.1. Aufbau von LEX-Files:

Ein LEX-File ist grundsätzlich in einer Assembler-Sprache geschrieben. Damit das System wichtige Stellen des Files direkt schon vor der Ausführung findet, benutzt man zur Festlegung dieser Stellen Markierungen, auf die Pointers in einer bestimmten Reihenfolge zeigen. Mit Hilfe dieser Pointers werden solche Teile wie Angabe der Schlüsselworte, Angabe der Fehlermeldungen, Angabe der Phase, in der die Schlüsselworte zerlegt werden, etc.
Man unterscheidet grob folgende Teile eines LEX-Files:

- Der Kontrollblock des Programms
- Definitionen von Markierungen, mit deren Hilfe die durch den LEX-File festgelegten Befehle in eine binäre Codierung gebracht und dann ausgeführt werden:
  1. Definition der RUNTIME-Phase
  2. Definition der NAMES-Phase
  3. Definition der PARSE-Phase
  4. Definition der ERRORS-Phase
  5. Definition der RUN-Phase
  6. Definition der MAIN-Phase(n)
  7. Definition der RUNAGAIN-Phase
- In diesen Definitionen der einzelnen Phasen werden die Adressen angegeben von den Stellen im File, von denen an die definierte Phase beginnt.
  + die PARSE-Phase teilt dem System mit, wie es das Schlüsselwort auf richtige Syntax und richtige Anzahl von Parametern etc. überprüft und diese dann in binäre Codierungen überführt.
  + In der RUNTIME-Phase werden die Schlüsselworte dann in Maschinensprache übersetzt.
  + Die Angabe der durch den LEX-File festgelegten Befehle und der dazugehörigen Schlüsselworte erfolgt in der NAMES-Phase.
  + In der ERROR-Phase werden die ERROR-Meldungen für diese LEX-Karte festgelegt.
  + In der RUN-Phase werden die Statusbedingungen für dieses LEX-File hergestellt.
  + In der MAIN-Phase beginnt die Hauptphase des Files. Hier sind die Befehle in Assembler codiert, die beim Aufruf eines Befehls abgearbeitet werden, damit am Ende das richtige Ergebnis rauskommt.
  + In der RUNAGAIN-Phase werden wieder die Bedingungen hergestellt, die vor dem Aufruf des Befehls vorhanden waren.
  + Am Ende dieser Pointertabelle stehen zwei Bytes mit den Werten 255 255. Sie veranlassen den Rechner, daß er beim ersten Aufruf des Files für jeden Befehl eine Codierung in einem System-ROM zur Verfügung stellt.
- Nach dieser Pointertabelle kommen die Routinen, die zur Ausführung des Befehls benötigt werden.
  Der Rechner verläßt sich bei der Abarbeitung nur auf die richtige Syntax dieser Pointertabelle. Liegt in dieser ein Fehler, so stürzt der Rechner ab und läßt sich nur durch Entfernen des Akku und des Netzteils wieder abfangen.
  Außerdem muß jede Phase mit einem RTN beendet werden.

a) Der Kontrollblock:

Der Kontrollblock ist 20 Bytes lang. In ihm sind folgende Informationen der Reihe nach enthalten:

- Die ersten 2 Bytes geben in umgekehrter Reihenfolge den RAM-Bereich an, in dem das Programm steht.
- Die nächsten beiden Bytes geben die Länge des Files an und zwar nach der Codierung der Programmlänge wie unter 1.1.b)
- Die Bytes 5 und 6 haben die Hex-Werte 8D und 4C(= ASCII L)
- In den Bytes 7 bis 10 ist die Zeit codiert. Sie geben die Anzahl der Sekunden an vom 1.1.1900 0.00 Uhr bis zum vorliegenden Datum.
- Der Name des Files, der benötigt wird, wenn man den File kopieren will, steht in den Bytes 11 bis 18. Überflüssige Bytes werden mit Blanks aufgefüllt.
- Die Bytes 19 und 20 spielen eine entscheidende Rolle bei der Codierung der Befehle in einem BASIC-File. So wird der 1.Befehl des LEX-Files wie folgt codiert: 180,Byte 19(dez.),Byte 20(dez.),1(=Hinweis, daß es der erste Befehl des LEX-Files ist).
  Bsp: Bytes 19 und 20 haben beim PEEKPOKE-LEX-File die Werte dezimal 21 64
  Damit hat der Befehl PEEK als 2.Befehl die Codierung 180 21 64 2
  der Befehl POKE als 1.Befehl die Codierung 180 21 64 1

b) Die Systemtabelle:
Mit Hilfe dieser Tabelle interpretiert der Rechner die Befehle und übersetzt sie in den Maschinencode. Für jede Phase im File benutzt der Rechner Pointer. Jeder Pointer besteht aus 2 Bytes. Das 2.Byte muß mit 256 multipliziert und zum Wert des 1.Byte dazugezählt werden. Dann erhält man die Anzahl der Zeichen, die der Rechner vom 19.Byte, dh. von der Basis-Adresse, bis zur Markierung dieser Phase überspringen muß.
Auf diese Art sind die einzelnen Adressen für die Phasen in der Systemtabelle definiert. Bei meheren Befehlen müssen auch mehrere MAIN-Phasen, dh. auch mehrere Definitionen von MAIN-Phasen, vorliegen. In diesem Fall werden die Pointer für die MAIN-Phasen durch zwei Bytepaare eingeschlossen, die vor und hinter der Definition der MAIN-Phase gleiche Werte haben.

6.2. Attribute:
Attribute definieren den Typ des Schlüsselwortes. Der Rechner benötigt Attribute auch dazu, um den Befehl später im PARSE-Teil richtig zerlegen zu können. Man unterscheidet zwei Arten von Attributen: primäre und sekundäre Attribute. Alle Schlüsselworte besitzen primäre Attribute, jedoch nur Funktionen besitzen auch sekundäre Attribute. Attribute müssen direkt vor der MAIN-Phase eines Schlüsselwortes plaziert werden.

a) primäre Attribute:
Primäre Attribute bestehen aus einem Byte, das Informationen über den Typ sowie die Klasse von Schlüsselworten enthält.

Bits:	7	6	5	4	3	2	1	0
	Typ			Klasse				

aa) Typ:

Bits: 7	6	Typ
1	1	BASIC-Statement, nicht erlaubt hinter THEN
1	0	BASIC-Statement, erlaubt hinter THEN
0	1	Systembefehle, nur über Tastatur auszuführen
0	0	Funktionen und andere Arten

ab) Klasse:

Bits: 5	4	3	2	1	0	Klasse
1	1	1	1	1	1	
1	1	1	1	1	0	
1	1	1	1	0	1	
1	1	1	1	0	0	
1	1	1	0	1	1	
1	1	1	0	1	0	
1	1	1	0	0	1	
1	1	1	0	0	0	
1	1	0	1	1	1	
1	1	0	1	1	0	
1	1	0	1	0	1	
1	1	0	1	0	0	
1	1	0	0	1	1	
1	1	0	0	1	0	
1	1	0	0	0	1	
1	1	0	0	0	0	
1	0	1	1	1	1	
1	0	1	1	1	0	String-Funktiion (wie CHR$, VAL$)
1	0	1	1	0	1	numerische Funktionen (wie SIN, COS, IP)
1	0	1	1	0	0	
1	0	1	0	1	1	
1	0	1	0	1	0	
1	0	1	0	0	1	
1	0	1	0	0	0	
1	0	0	1	1	1	

Bits: 5	4	3	2	1	0	Klasse
1	0	0	1	1	0	
1	0	0	1	0	1	
1	0	0	1	0	0	
1	0	0	0	1	1	
1	0	0	0	1	0	
1	0	0	0	0	1	Reservierte Schlüsselworte
1	0	0	0	0	0	
0	1	1	1	1	1	
0	1	1	1	1	0	
0	1	1	1	0	1	
0	1	1	1	0	0	
0	1	1	0	1	1	
0	1	1	0	1	0	
0	1	1	0	0	1	
0	1	1	0	0	0	
0	1	0	1	1	1	
0	1	0	1	1	0	
0	1	0	1	0	1	
0	1	0	1	0	0	
0	1	0	0	1	1	
0	1	0	0	1	0	
0	1	0	0	0	1	
0	1	0	0	0	0	
0	0	1	1	1	1	
0	0	1	1	1	0	
0	0	1	1	0	1	
0	0	1	1	0	0	
0	0	1	0	1	1	
0	0	1	0	1	0	
0	0	1	0	0	1	
0	0	1	0	0	0	
0	0	0	1	1	1	
0	0	0	1	1	0	
0	0	0	1	0	1	
0	0	0	1	0	0	
0	0	0	0	1	1	
0	0	0	0	1	0	
0	0	0	0	0	1	
0	0	0	0	0	0	

b) sekundäre Attribute:
Beim Zerlegen eines Schlüsselwortes in der PARSE-Phase muß, wenn das Schlüsselwort eine Funktion ist, auch der Typ dieser Funktion festgelegt werden. Dies geschieht durch die sekundären Attribute. Aus wieviel Bytes dieses Attribut besteht hängt von der Anzahl der Parameter ab, die für die Ausführung der Funktion benötigt werden. In den ersten 4 Bits des ersten Bytes steht die Anzahl der benötigten Parameter. Danach stehen für jeden Parameter 2 Bits zur Verfügung, in denen festgelegt wird, von welchem Typ der Parameter ist.

Parameter-Typs

Bits: 1	2	Typ
1	1	Fremde Art
1	0	String
0	1	Numerisches Array
0	0	Numerisch

Bsp: POKE(m),n ist definiert als (nur sekundäres Attribut):

0 0 1 0	0 1	0 0
2 Parameter	numerisches Array	numerischer Parameter

6.3. Assembler-Befehle:

Da man beim HP-75C über kein eigenes Assembler-ROM verfügt, müssen die Befehle mit ihrem Hex-Code über das Programm 'LEXIN' über das Laufwerk in den Rechner gebracht werden. Zunächst einige Vorbemerkungen:

6.3.1. Systemgrößen:

a) Operationscode:
Der Operationscode ist eine hexadezimale Wiedergabe eines Assemblerbefehles.

b) Literal:
Diese Größe ist ein Byte lang. Sie dient bei den Operationen dazu, eine absolute Zahl zu repräsentieren. So gibt sie zB. in Verbindung mit einem Sprungbefehl an, wieviele Bytes absolut übersprungen werden müssen.

c) Label:
Diese Systemgröße gibt immer eine Adresse im Memory des Rechners an. Die Codierung erfolgt in umgekehrter Reihenfolge.
Bsp: Die Bytes 10 FF geben die Adresse FF10 an.

d) AR: Position des Adressregisters
Diese Größe ist ein Byte lang und gibt an, auf welches Register der ARP zeigt.

e) DR: Position des Datenregisters
Auch diese Größe ist ein Byte lang. Sie gibt die Position des Registers an, auf das der DRP zeigt.

6.3.2. Vorbemerkungen zur Codierung eines Befehles:

In den Codierungen der Befehle, besonders bei den Bezeichnungen der Register arbeitet der Rechner im Oktalsystem.

a) Codierung der AR:
Um zum Hex-Code zu kommen, der die AR repräsentiert, wandelt man die Registernummer vom Oktalsystem in das Hexadezimalsystem um.
Bsp: R 20 als AR ist codiert als
oktal 20 ist 010 000, was 0001 0000 also 10 im Hex-System entspricht

b) Codierung der DR:
Um zu diesem Hex-Code zu kommen, wandelt man die Registernummer vom Oktalsystem ins Dezimalsystem um, addiert 64, und wandelt dann diese Zahl um ins Hexadezimalsystem.
Bsp: R 20 als DR ist codiert als
oktal 20 ist 010 000, also 16 im Dezimalsystem.
16 + 64 = 80, was im Hex-Code 0101 0000, also 50 entspricht.

c) Zusatzbemerkungen zu a) und b):
AR oder DR müssen bei einem Befehl nur angegeben werden, sofern sie zur Ausführung des Befehls verändert werden müssen. Ansonsten können sie zur Byteeinsparung entfallen. Der Rechner benutzt dann die aktuelle bis jetzt gültige DR oder AR.
In einem Befehl mit AR und/oder DR muß folgende Reihenfolge eingehalten werden:
- Hex-Code für DR oder XR
- Hex-Code für AR oder XAR (XAR besitzt die gleiche Codierung wie AR)
- Hex-Code für den Befehl = Operationscode

d) Sprung mit Literal nach vorne (dh. positiver Sprung):
Der Hex-Code gibt umgewandelt ins Dezimalsystem die Anzahl der Zeichen an, die übersprungen werden müssen.
Bsp: F7 3D entspricht JZR 3D
3D im Hex-Code ist 61 im Dezimasytem. Dies bedeutet nun, daß bei der Ausführung dieses Befehls 61 Zeichen übersprungen werden, und daß mit dem 62.Byte die Ausführung des Files fortgesetzt wird.

e) Sprung mit Literal nach hinten (dh. negativer Sprung):
Der Hex-Code, umgewandelt ins Dezimalsystem, und von 255 abgezogen, gibt die Anzahl der Zeichen an, die rückwärts gesprungen werden.
Bsp: F6 F7 entspricht JNZ F7
F7 im Hex-Code ist 247 im Dezimalsystem. Dies bedeutet nun, daß der Rechner 255-247=8 Bytes nach hinten gesprungen werden, und daß mit dem 8.Byte,nach hinten gesprungen, die Ausführung des Files fortgesetzt wird.

f) Multibytes:
Im Gegensatz zu den Single-Bytes, die aus nur einem Byte bestehen, können Multibytes aus bis zu 8 Bytes bestehen. Da ja wie bekannt die Registernummer im Oktalsystem codiert ist, bilden 8 Bytes eine Einheit. Das letzte Byte dieser Einheit ist das Byte, dessen 3 Abschlußbytes im 2.Halbbyte den Wert 1 haben (also Registernummern 07, 17, 27, 37, 47, 57, 67, 77)
Auch für das letzte Byte eines Multibytes gilt diese Regel. Ist nun in einem Assemblerbefehl eine Registernummer für das Multibyte angegeben, so beginnt dieses Multibyte mit diesem Register und endet mit dem letzten Register dieser Einheit.
Bsp: Multibyte R40 beinhaltet die Register 40-47 (8 Single-Bytes)
Multibyte R26 beinhaltet die Register 26-27 (2 Single-Bytes)
Multibyte R14 beinhaltet die Register 14-17 (4 Single-Bytes)

6.3.3. LOAD/STORE-Befehle:

a) LDB DR, AR:
Codierung: A0
Der Inhalt von AR wird nach DR geladen
Bsp: LDB R36, R32 mit der Hex-Codierung 5E 1A A0 bedeutet, daß der Inhalt von Register 32 ins Register 36 geladen wird.
LDM DR, AR:
Codierung: A1
Der Inhalt von dem Multibyte, beginnend bei AR, wird in das Multibyte, beginnend mit DR, geladen.
Bsp: LDM R40, R50 mit der Hex-Codierung 60 28 A1 bedeutet, daß die Inhalte von Register 50 bis 57 in die Register 40 bis 47 übertragen werden.

b) STB DR, AR:
Codierung: A2
Der Inhalt von DR wird nach AR kopiert
Bsp: STB R36, R32 mit der Hex-Codierung 5E 1A A2 bedeutet, daß der Inhalt von Register 36 nach Register 32 kopiert wird.
STM DR, AR:
Codierung: A3
Der Inhalt des Multibytes, beginnend bei DR, wird in das Multibyte, beginnend bei AR, geladen.
Bsp: STM R40, R50 mit der Hex-Codierung 60 28 A3 bedeutet, daß die Inhalte der Register 40 bis 47 in die Register 50 bis 57 geladen werden.

c) LDBD DR, AR:
Codierung: A4
Das Register DR wird mit dem Inhalt gefüllt, der an der Stelle des Memory des Rechners zu finden ist, auf die AR zeigt.
Bsp: LDBD R36, R32 mit der Hex-Codierung 5E 1A A4 bedeutet, daß Register 36 mit dem Inhalt gefüllt wird, der an der Stelle des Memory zu finden ist, dessen Adresse in den Registern 32 bis 33 steht. Hat Register 32 den Inhalt 4E und Register 33 den Inhalt A1, dann wird nach Register 36 der Inhalt gebracht, der im Memory auf der Adresse A1 4E zu finden ist.
LDMD DR, AR:
Codierung: A5
Das Multibyte, beginnend mit DR, wird mit dem Inhalt geladen, der, beginnend bei der Adresse AR bis AR+1, im Memory des Rechners steht.
Bsp: LDMD R40, R50 mit der Hex-Codierung 60 28 A5 bedeutet, daß die Register 40 bis 47 mit den Inhalten geladen werden, die im Memory des Rechners bei der Adresse zu finden sind, auf die die Register 50 bis 51 zeigen.
Überträgt man die Zahlenwerte von oben, sieht das dann so aus:

R40 enthält den Inhalt des Memory bei der Adresse		A1 4E
R41	"	A1 4F
R42	"	A1 50
R43	"	A1 51
R44	"	A1 52
R45	"	A1 53
R46	"	A1 54
R47	"	A1 55

d) STBD DR, AR:
Codierung: A7
Der Inhalt von DR wird an die Stelle des Memory geschrieben, dessen Adresse in den Registern AR bis AR+1 steht.
Bsp: STBD R36, R32 mit der Hex-Codierung 5E 1A A6 bedeutet, daß der Inhalt von Register 36 an die Stelle im Memory geschrieben wird, dessen Adresse in den Registern 32 bis 33 steht. Steht in Register 32 4E und in Register 33 der Inhalt A1, dann wird der Inhalt von Register 36 im Memory an der Adresse A1 4E gespeichert.
STMD DR, AR:
Codierung: A8
Der Inhalt des Multibytes, das bei DR beginnt, wird an die Stelle des Memory geschrieben, dessen Adresse mit der Adresse beginnt, die in den Registern AR bis AR+1 steht.
Bsp: STMD R40, R50 mit der Hex-Codierung 60 28 A7 bedeutet, daß die Inhalte der Register R40 bsi R47 an die Stelle des Memory geschreiben werden, dessen Adresse in den Registern R50 bis 51 steht. Steht in Register 50 4E und in Register 51 A1, so werden die Register 40 bis 47 in die Memory-Adressen A1 4E bis A1 55 geschrieben.

e) LDB DR, =literal:
Codierung: A8
Das Register DR wird mit den im literal-Feld stehenden Informationen geladen.
Bsp: LDB R36, =124 mit der Hex-Codierung 5E A8 7C bedeutet, daß R36 den Inhalt dezimal 124 erhält.
LDM DR, =literal:
Codierung: A9
Das Multibyte, beginnend bei Register DR, wird mit den im literal-Feld stehenden Informationen geladen.
Bsp: LDM R40, =0,0,0,0,0,0,0,5 mit der Hex-Codierung 60 A9 5 0 0 0 0 0 0 0 bedeutet, daß die Register 40-47 mit den Inhalten 0 0 0 0 0 0 0 und 5 geladen werden.

f) STB DR, =literal:
Codierung: AA
Der Inhalt des Registers DR wird an die Stelle des Memory geschrieben, auf die der Programmcounter, vertreten durch das literal-Feld, zeigt.
Bsp: STB R36, =124 mit der Hex-Codierung 5E AA 7C bedeutet, daß der Inhalt von Register 36 in den Memory-Bereich 00 7C geschrieben wird.
STM DR, =literal:
Codierung: AB
Der Inhalt des Multibytes, beginnend bei Register DR, wird in den Memory-Bereich geschrieben, auf die der Programmcounter (literal-Feld) zeigt.
Bsp: STM R32, =KEYBOARD mit der Hex-Codierung 5A AB 02 FF bedeutet, daß das Multibyte R32-33 in den Memory-Bereich geschrieben wird, auf die der Programmcounter hochgeschoben wird.
Anmerkung: Im Programmcounter steht nach der Operation.der Inhalt des Registers DR.

g) LDBI DR, AR:
Codierung: AC
Der Inhalt, auf den das Register zeigt, dessen Nummer sich im AR befindet, wird in das Register DR geladen.
Bsp: LDBI R36, R32 mit der Hex-Codierung 5E 1A AC bedeutet, daß, wenn Register 50-51 FF00 enthält, und an der Adresse FF00 A167 gespeichert steht, der Inhalt A167 in R36 geladen wird.
LDMI DR, AR:
Codierung: AD
Die Inhalte, auf die das Register zeigt, das sich im AR befindet, werden in das Multibyte, beginnend bei DR, geladen.

Bsp: LDMI R40, R50 mit der Hex-Codierung 60 28 AD bedeutet, daß, wenn Register 50 FF00 enthält, und an der Adresse FF00 A167 gespeichert ist, der Inhalt A167 in R40, der Inhalt A168 in R41, der Inhalt A169 in R42, der Inhalt A16A in R43, ..., der Inhalt A16E in R47 gespeichert wird.

h) STBI DR, AR:
Codierung: AE
Der Inhalt von DR wird an die Stelle des Memory gespeichert, auf die das Register zeigt, dessen Registernummer in der Memory-Lokation liegt, auf das AR zeigt.
Bsp: STBI R36, R32 mit der Hex-Codierung 5E 1A AE bedeutet, daß, wenn Register 32 FF00 enthält, und an der Stelle FF00 A167 steht, der Inhalt von Register 36 im Memory bei der Adresse A167 gespeichert wird.
STMI DR, AR:
Codierung: AF
Der Inhalt der Register, beginnend bei DR, wird an die Stellen des Memory gespeichert, die mit dem Register beginnen, dessen Registernummer in der Memory-Lokation liegt, auf das AR zeigt.
Bsp: STMI R40, R50 mit der Hex-Codierung 60 28 AF bedeutet, daß, wenn Register 50 FF00 enthält, und an der Stelle FF00 A167 steht, das Multibyte R40-47 im Memory bei den Adressen A167-A16E gespeichert wird.

i) LDBD DR, =label:
Codierung: B0
Das Register DR wird mit dem Inhalt geladen, der unter dem angegebenen label steht.
Bsp: LDBD R34, =KEYBOARD mit der Hex-Codierung 5C B0 02 FF bedeutet, daß der Inhalt bei der Memory-Adresse FF02 ins Register 34 geladen wird.
LDMD DR, =label:
Codierung: B1
Das Multibyte, beginnend mit dem Register DR, wird mit dem Inhalt geladen, der im Memory unter den Adressen steht, der mit dem angegebenen label beginnt.
Bsp: LDMD R34, =KEYBOARD mit der Hex-Codierung 5C B1 02 FF bedeutet, daß der Inhalt der Adresse FF02 in R34, Inhalt bei FF03 in R35, der Inhalt bei FF04 in R36 und der Inhalt bei FF05 in R37 geladen wird.

j) STBD DR, =label:
Codierung: B2
Der Inhalt des Registers DR wird in den Memory-Bereich unter der Adresse abgespeichert, die in label angegeben ist.
Bsp: STBD R34, =KEYBOARD mit der Hex-Codierung 5C B2 02 FF bedeutet, daß der Inhalt des Registers 34 im Memory-Bereich unter der Adresse FF02 abgespeichert wird.
STMD DR, =label:
Codierung: B3
Der Inhalt des Multibytes, beginnend mit Register DR, wird in den Memory-Bereich gespeichert, beginnend mit der Adresse, die in label angegeben ist.
Bsp: STMD R34, =KEYBOARD mit der Hex-Codierung 5C B3 02 FF bedeutet, daß die Inhalte der Register 34-37 im Memory-Bereich unter den Adressen FF02-FF05 abgespeichert werden.

k) LDBD DR, XAR, label:
Codierung: B4
Das Register DR erhält als Inhalt die Summe aus der Adresse, unter der das label abgespeichert ist, und dem Wert von XAR.
Bsp: LDBD R36, X32, KEYBOARD mit der Hex-Codierung 5E 1A B4 02 FF bedeutet, daß, wenn R32-33 den Inhalt 0003 haben, in Register 36 die Summe FF02+0003=FF05 abgespeichert wird.
LDMD DR, XAR, label:
Codierung: B5

Das Multibyte, beginnend mit DR, erhält als Inhalt die Summe aus der Adresse, unter der das label abgespeichert ist, und dem Wert von XAR. Die Inhalte der auf DR folgenden Register des Multibytes erhält man, wenn man zu der og. Summe immer hex 01 addiert.

Bsp: LDMD R34, X32, KEYBOARD mit der Hex-Codierung 5C 1A B5 02 FF bedeutet, daß, wenn R32-33 den Inhalt 0003 haben, in Register 34-35 die Summe FF02+0003=FF05 und in Register 36-37 FF06 abgespeichert wird.

1) STBD DR, XAR, label:

Codierung: B6

Der Inhalt des Registers DR-DR+1 wird unter der Adresse abgespeichert, die sich ergibt, wenn man die Summe aus der Adresse, unter der das label aufgeführt ist, und dem Wert von XAR bildet.

Bsp: STBD R36, X32, KEYBOARD mit der Hex-Codierung 5E 1A B6 02 FF bedeutet, daß, wenn R32-33 den Inhalt 0003 haben, der Wert von Register 36 unter der Memory-Adresse FF02+0003=FF05 abgespeichert wird.

STMD DR, XAR, label:

Codierung: B7

Der Inhalt des Multibytes, beginnend mit DR, wird abgespeichert beginnend unter der Memory-Adresse, die sich ergibt, wenn man die Summe aus der Adresse, unetr der das label aufgeführt ist, und dem Wert von XAR bildet.

Bsp: STMD R34, X32, KEYBOARD mit der Hex-Codierung 5C 1A B7 02 FF bdeutet, daß, wenn R32-33 den Inhalt 0003 haben, der Wert von R34-35 unter der Adresse FF02+0003=FF05, der Wert von R36-37 unter der Adresse FF06 abgespeichert wird.

m) LDBI DR, =label:

Codierung: B8

Der Inhalt, der sich unter der Memory-Adresse befindet, die durch den Inhalt der Memory-Adresse festgelegt wird, die sich bei label befindet, wird nach DR geladen.

Bsp: LDBI R34, =KEYBOARD mit der Hex-Codierung 5C B8 02 FF bedeutet, daß, wenn sich unter der Adresse FF02-FF03 der Inhalt 1A67 befindet, der Inhalt, der sich unter der Adresse 1A67 befindet, ins Register 34 geladen wird.

LDMI DR, =label:

Codierung: B9

Beginnend mit dem Inhalt, der sich unter der Memory-Adresse befindet, die durch den Inhalt der Memory-Adresse festgelegt wird, die sich bei label befindet, wird das Multibyte, beginnend mit DR, geladen.

Bsp: LDMI R36, =KEYBOARD mit der Hex-Codierung 5E B9 02 FF bedeutet, daß, wenn sich unter der Adresse FF02-FF03 der Inhalt 1A67 befindet, der Inhalt, der sich unter der Adresse 1A67 befindet, ins Register 36 , der Inhalt der Adresse 1A68 ins Register 37 geladen wird.

n) STBI DR, =label:

Codierung: BA

Der Inhalt des Registers DR wird unter der Memory-Adresse geladen, die durch den Inhalt der Memory-Adresse festgelegt wird, die sich bei label befindet.

Bsp: STBI R34, =KEYBOARD mit der Hex-Codierung 5C BA 02 FF bedeutet, daß, wenn sich unter der Adresse FF02-FF03 der Inhalt 1A67 befindet, der Inhalt von Register 34 unter der Adresse 1A67 gespeichert wird.

STMI DR, =label:

Codierung: BB

Der Inhalt des Multibytes, beginnend bei DR, wird beginnend unter der Memory-Adresse geladen, die durch den Inhalt der Memory-Adresse festgelegt wird, die sich bei label befindet.

Bsp: STMI R36, =KEYBOARD mit der Hex-Codierung 5E BB 02 FF bedeutet, daß, wenn sich unter der Adresse FF02-FF03 der Inhalt 1A67 befindet, der Inhalt von Register 36 unter der Adresse 1A67, der Inhalt von Register 37 unter der Adresse 1A68 gespeichert wird.

o) LDBI DR, XAR, label:
Codierung: BC
Der Inhalt, der sich unter der Memory-Adresse befindet, die durch den Inhalt der Memory-Adresse festgelegt wird, auf die die Summe von XAR und der Adresse von label zeigt, wird nach DR geladen.
Bsp: LDBI R34, X32, KEYBOARD mit der Hex-Codierung 5C 1A BC 02 FF bedeutet, daß, wenn R32-33 den Inhalt 0003 haben und wenn sich unter der Adresse FF05-FF06 der Inhalt E5F0 befindet, der Inhalt, der sich unter der Adresse E5F0 befindet, ins Register 34 geladen wird.
LDMI DR, XAR, label:
Codierung: BD
Beginnend mit dem Inhalt, der sich unter der Memory-Adresse befindet, die durch den Inhalt der Memory-Adresse festgelegt wird, auf die die Summe von XAR und der Adresse von label zeigt, wird das Multibyte, beginnend mit DR, geladen.
Bsp: LDMI R36, X32, KEYBOARD mit der Hex-Codierung 5E 1A BD 02 FF bedeutet, daß, wenn R32-33 den Inhalt 0003 haben und wenn sich unter der Adresse FF05-FF06 der Inhalt E5F0 befindet, der Inhalt, der sich unter der Adresse E5F0 befindet, ins Register 36, der Inhalt, der sich unter der Adresse E5F1 befindet, ins Register 37 geladen wird.

p) STBI DR, XAR, label:
Codierung: BE
Der Inhalt des Registers DR wird unter der Memory-Adresse geladen, die durch den Inhalt der Memory-Adresse festgelegt wird, auf die die Summe aus XAR und der Adresse von label zeigt.
Bsp: STBI R34, X32, KEYBOARD mit der Hex-Codierung 5C 1A BE 02 FF bedeutet, daß, wenn R32-33 den Inhalt 0003 haben und wenn sich unter der Adresse FF05-FF06 der Inhalt E5F0 befindet, der Wert des Registers 34 unter der Memory-Adresse E5F0 abgespeichert wird.
STMI DR, XAR, label:
Codierung: BF
Der Inhalt des Multibytes, beginnend bei DR, wird, beginnend unter der Memory-Adresse, geladen, die durch den Inhalt der Memory-Adresse festgelegt wird, auf die die Summe aus XAR und der Adresse von label zeigt.
Bsp: STMI R36, X32, KEYBOARD mit der Hex-Codierung 5E 1A BF 02 FF bedeutet, daß, wenn R32-33 den Inhalt 0003 haben und wenn sich unter der Adresse FF05-FF06 der Inhalt E5F0 befindet, der Wert des Registers 36 unter der Memory-Adresse E5F0, der Wert des Registers 37 unter der Memroy-Adresse E5F1 abgespeichert wird.

6.3.4. Arithmetische und logische Funktionen:

a) Additionsbefehle:
ADB DR, AR:
Codierung: C2
Zum Inhalt von DR wird der Inhalt von AR addiert. Die Summe steht nach der Operation in DR.
Bsp: ADB R36, R32 mit der Hex-Codierung 5E 1A C2 bedeutet, daß der Inhalt von Register 32 zum Register 36 addiert wird. Die Summe R36+R32 steht nach der Operation in Register 36.
Kurzform: DR=AR+DR
ADB DR, =literal:
Codierung: CA
Zum Inhalt von DR wird die im literal-Feld stehende absolute Zahl addiert. Die Summe steht nach der Operation in DR.
Bsp: ADB R36, =124 mit der Hex-Codierung 5E CA 7C bedeutet, daß, wenn Register 36 als Inhalt 16 hat, die Operation 7C+16 ausgeführt wird, und die Summe 92 ins Register 36 geschrieben wird.
ADM DR, AR:
Codierung: C3

Zum Inhalt des Multibytes, beginnend bei DR, wird das Multibyte, beginnend bei AR, addiert. Die Summe der beiden Multibytes, befindet sich nach der Operation im Multibyte, beginnend bei DR.

Bsp: ADM R36, R46 mit der Hex-Codierung 5E 26 C3 bedeutet, daß zum Register 36 das Register 46, zum Register 37 das Register 47 addiert wird. Die Summe aus R36 und R46 steht nach der Operation in R36, die Summe aus R37 und R47 in R37.

ADM DR, =literal:

Codierung: CB

Zum Inhalt des Multibytes, beginnend bei DR, werden die im literal-Feld angegebenen ganzen Zahlen addiert und die Summe im Multibyte, beginnend bei DR, gespeichert.

Bsp: ADM R36, =0,124 mit der Hex-Codierung 5E CB 00 7C bedeutet, daß zu Register 36 die 0, zum Register 37 die Zahl 124 addiert wird. Die Summe aus dem Inhalt des Registers 36 und der Zahl 0 wird nach der Operation im Register 36 gespeichert, die Summe aus dem Inhalt des Registers 37 und der ganzen Zahl 124 wird nach der Operation im Register 37 gespeichert.

ADBD DR, AR:

Codierung: DA

Zum Inhalt des Registers DR wird der Inhalt addiert, der an der Stelle des Memory zu finden ist, auf dessen Lokation das Register AR zeigt.

Bsp: ADBD R36, R46 mit der Hex-Codierung 5E 26 DA bedeutet, daß, wenn Register 46 A1 und Register 47 EF enthält, und an der Stelle EFA1 im Memory sich der Inhalt 29 befindet, dieses hexadezimale 29 zum Inhalt von Register 36 addiert werden.

ADBD DR, =label:

Codierung: D2

Zum Inhalt von Register DR wird der Inhalt addiert und in DR gespeichert, der sich an der Stelle des Memory befindet, die durch das label festgelegt ist.

Bsp: ADBD R36, =KEYBOARD mit der Hex-Codierung 5E D2 02 FF bedeutet, daß wenn sich unter der Adresse FF02 ein Inhalt befindet, dieser Inhalt zum Inhalt des Registers 36 addiert und in Register 36 gespeichert wird.

ADMD DR, AR:

Codierung: DB

Zum Inhalt des Multibytes, beginnend bei DR, wird der Inhalt addiert, der sich beginnend mit den Stellen im Memorybereich befindet, auf die die Register AR und AR+1 zeigen.

Bsp: ADMD R36, R46 mit der Hex-Codierung 5E 26 DB bedeutet, daß, wenn Register 46 A1 und Register 47 EF enthält, der Inhalt , der sich bei der Memory-Adresse EFA1 befindet, zum Register 36, der Inhalt, der sich bei der Memory-Adresse EFA2 befindet, zum Register 37 addiert wird. Die Summen befinden sich im Register 36 bzw. 37 (Verfahren wie oben).

ADMD DR, =label:

Codierung: D3

Zum Inhalt des Multibytes, beginnend bei DR, wird der Inhalt addiert, der sich beginnend mit den Stellen im Memory-Bereich befindet, auf die die Adresse des labels zeigt.

Bsp: ADMD R36, =KEYBOARD mit der Hex-Codierung 5E D3 02 FF bedeutet, daß der Inhalt, der sich unter der Memory-Adresse FF02 befindet, zum Inhalt des Registers 36 addiert und in Register 36 abgespeichert wird, der Inhalt, der sich unter der Adresse FF03 befindet, zum Inhalt des Registers 37 addiert und in Register 37 abgespeichert wird.

b) Logische Operationen (AND):

ANM DR, AR:

Codierung: C7

Zwischen den Registerpaaren DR-DR+1 und AR-AR+1 wird die logische Operation AND durchgeführt, dh. es bleibt die 1 nur enthalten, wenn die 1 in beiden Paaren an der gleichen Stelle vorhanden ist.

Bsp: ANM R36, R46 mit der Hex-Codierung 5E 26 C7 bedeutet, daß bei folgender Registerbelegung im Dualsystem folgendes Resultat entsteht:

R36:	10101100	R37:	11101000
R46:	01011101	R47:	01110010

Resultat:

R36:	00001100	R37:	01100000

ANM DR, =literal:

Codierung: CF

Zwischen den Registerpaaren DR-DR+1 und den im literal-Feld stehenden ganzen Zahlen findet eine AND-Verknüpfung statt.

Bsp: ANM R36, =34, 217 mit der Hex-Codierung 5E CF 22 D9 bedeutet, daß folgendes Resultat entsteht:

R36:	10101100	R37:	11101000
=34:	00100010	=217:	11011001

Resultat:

R36:	00100000	R37:	11001000

ANMD DR, AR:

Codierung: DF

Zwischen den Inhalten der Register DR-DR+1 und dem Register, auf das beginnend mit AR gezeigt wird, findet eine AND-Verknüpfung statt.

Bsp: ANMD R36, R46 mit der Hex-Codierung 5E 26 DF bedeutet, daß, wenn Register 46-47 A156 enthalten und an der Stelle A156 01011101, an der Stelle A157 01110010 als Inhalt stehen, folgendes Resultat gebildet wird:

R36:	10101100	R37:	11101000
A156:	01011101	A157:	01110010

Resultat:

R36:	00001100	R37:	01100000

ANMD DR, =label:

Codierung: D7

Zwischen den Inhalten der Register DR-DR+1 und dem Inhalt, der dort zu finden ist, wo, beginnend mit der label-Adresse, der Pointer hinzeigt, findet eine AND-Verknüpfung statt.

Bsp: ANMD R36, =KEYBOARD mit der Hex-Codierung 5E D7 02 FF bedeutet, daß, wenn an der Adresse FF02 der Inhalt 01011101 und an der Adresse FF03 der Inhalt 01110010 zu finden ist, folgendes Resultat entsteht:

R36:	10101100	R37:	11101000
FF02:	01011101	FF03:	01110010

Resultat:

R36:	00001100	R37:	01100000

c) Logische Operationen (OR):

ORB DR, AR:

Codierung: 94

Zwischen dem Register DR und AR findet eine OR-Verknüpfung statt. Das Ergebnis wird nach DR übertragen.
(OR=das Ergebnis wird dann 1, wenn wenigstens eine der beiden Ausgangszustände 1 war)

Bsp: ORB R36, R46 mit der Hex-Codierung 5E 26 94 bedeutet, daß sich folgende Ergebnisentwicklung ergibt:

R36:	10101100
R46:	01011101

Resultat:

R36:	11111101

ORM DR, AR:

Codierung: 95

Zwischen dem Multibyte, beginnend mit DR, und dem Multibyte, beginnend mit AR, findet eine OR-Verknüpfung statt. Das Ergebnis wird ins Multibyte, beginnend mit DR, übertragen.

Bsp: ORM R36, R46 mit der Hex-Codierung 5E 26 95 bedeutet, daß sich folgende Ergebnisentwicklung ergibt:

R36: 10101100 R37: 11101000
R46: 01011101 R47: 01110010
Resultat:
R36: 11111101 R37: 11111010

d) Logische Operationen (EXOR):
XRB DR, AR:
Codierung: 96
Zwischen dem Register DR und dem Register AR findet eine EXOR-Verknüpfung statt. Das Ergebnis wird nach DR übertragen.
(EXOR=das Ergebnis ist nur dann 1, wenn beide Ausgangsbits verschieden voneinander sind)
Bsp: XRB R36, R46 mit der Hex-Codierung 5E 26 96 bedeutet, daß sich folgende Ergebnisentwicklung ergibt:
R36: 10101100
R46: 01011101
Resultat:
R36: 11110001

XRM DR, AR:
Codierung: 97
Zwischen dem Multibyte, beginnend mit DR, und dem Multibyte, beginnend mit AR, findet eine EXOR-Verknüpfung statt. Das Ergebnis wird ins Multibyte, beginnend mit DR, übertragen.
Bsp: XRM R36, R46 mit der Hex-Codierung 5E 26 97 bedeutet, daß sich folgende Ergebnisentwicklung ergibt:
R36: 10101100 R37: 11101000
R46: 01011101 R47: 01110010
Resultat:
R36: 11110001 R37: 10011010

e) Vergleichsoperationen (Compare):
Vergleiche gliedern sich in zwei Teile:
- der eigentliche Vergleich, nach dessen Ergebnis der Zustand von 2 Statusbytes gesetzt wird.
- der Sprung bzw. die Auswertung dieses Vergleichs, die direkt auf den Status der beiden og. Bytes zurückgreift.

Man unterscheidet vier Arten von Vergleichen:
- Ist DR kleiner als AR, wird das Flag CY auf 0 gesetzt.
- Ist AR kleiner oder gleich DR, wird das Flag CY auf 1 gesetzt.
- Ist AR gleich DR, wird das Flag ZR auf 1 gesetzt.
- Ist AR ungleich DR, wird das Flag ZR auf 0 gesetzt.

In den eigentlichen Assemblerbefehlen kann man jetzt nur noch die Werte angeben, die verglichen werden. Wie die Flags gesetzt werden, ergibt sich dann von selbst. Die Regeln für das Setzen der Flags sind auf jeden Fall bei allen Assemblervergleichen gleich.
CMB DR, AR:
Codierung: C0
Es findet ein Vergleich zwischen den Bytes DR und AR statt.
Bsp: CMB R36, R46 mit der Hex-Codierung 5E 26 C0 bedeutet, daß sich folgender Vergleich ergibt:
R36: 10101100 R46: 01011101
Da Register 36 (=DR) größer ist als R46 (=AR) wird CY auf 1, ZR auf 0 gesetzt.

CMB DR, =literal:
Codierung: C8
Es findet ein Vergleich zwischen dem Byte DR und der ganzen Zahl im literal-Feld statt.
Bsp: CMB R36, =217 mit der Hex-Codierung 5E C8 D9 bedeutet, daß sich folgender Vergleich ergibt:
R36: 10101100 =217: 11011001
Da Register 36 (=DR) kleiner ist als die ganze Zahl 217 (=imaginärer Inhalt von AR), wird CY und ZR auf 0 gesetzt.

CMM DR, AR:
Codierung: C1
Es findet ein Vergleich zwischen den Multibytes, beginnend bei DR und beginnend bei AR, statt.

Bsp: CMM R36, R46 mit der Hex-Codierung 5E 26 C1 bedeutet, daß sich folgender Vergleich ergibt:

```
R36:   10101100          R37:  11101000
R46:   01011101          R47:  01110010
R36&R37:   1010110011101000
R46&R47:   0101110101110010
```

Da Register R36&R37 größer ist als R46&R47, wird CY auf 1, ZR auf 0 gesetzt.

CMM DR, =literal:

Codierung: C9

Es findet ein Vergleich zwischen dem Multibyte, beginnend bei DR, und den ganzen Zahlen im literal-Feld statt.

Bsp: CMM R36, =217, 124 mit der Hex-Codierung 5E C9 D9 7C bedeutet, daß sich folgender Vergleich ergibt:

```
R36:   10101100          R37:   11101000
=217:  11011001          =124:  01111100
R36&R37:   1010110011101000
=217&=124: 1101100101111100
```

Da die ganze Zahl 217&124 größer ist als das Multibyte R36&R37, werden CY und ZR auf 0 gesetzt.

CMBD DR, AR:

Codierung: D8

Es findet ein Vergleich statt zwischen dem Byte DR und dem Byte im Memory des Rechners, auf das AR zeigt.

Bsp: CMBD R36, R46 mit der Hex-Codierung 5E 26 D8 bedeutet, daß, wenn R46-R47 die Adresse A167 enthalten und an der Adresse A167 sich der Inhalt E8 befindet, sich folgender Vergleich ergibt:

```
R36:    10101100
$A167:  11101000
```

Da Register 36 kleiner ist als der Inhalt unter der Adresse A167, wird CY und ZR auf 0 gesetzt.

CMBD DR, =label:

Codierung: D0

Es findet ein Vergleich statt zwischen dem Byte DR und dem Byte, das unter der im label-Feld stehenden Adresse sich befindet.

Bsp: CMBD R36, =KEYBOARD mit der Hex-Codierung 5E D0 02 FF bedeutet, daß, wenn sich unter der Adresse FF02 das Byte mit dem Wert AC befindet, sich folgender Vergleich ergibt:

```
R36:   10101100
$FF02: 10101100
```

Da Register 36 gleich dem Inhalt unter der Adresse FF02 ist, werden ZR und CY auf 1 gesetzt.

CMMD DR, AR:

Codierung: D9

Es findet ein Vergleich statt zwischen dem Multibyte, beginnend mit DR und den Bytes, die unter der Adresse beginnen, auf die AR zeigt.

Bsp: CMMD R36, R46 mit der Hex-Codierung 5E 26 D9 bedeutet, daß, wenn Register 46-47 A167 enthalten und unter der Adresse A167 sich der Inhalt E8, unter A168 der Inhalt AC befindet, sich folgender Vergleich ergibt:

```
R36:   10101100                  R37:    11101000
$A167: 11101000                  $A168:  10101100
R36&R37:     1010110011101000
$A167&$A168  1110100010101100
```

Da das Multibyte R36&R37 kleiner ist als das Mulitbyte $A167&$A168, werden CY und ZR auf 0 gesetzt.

CMMD DR, =label:

Codierung: D1

Es findet ein Vergleich statt zwischen dem Multibyte, beginnend mit DR und den Bytes, die unter der im label-Feld angegebenen Adresse beginnen.

Bsp: CMMD R36, =KEYBOARD mit der Hex-Codierung 5E D1 02 FF bedeutet, daß, wenn sich unter der Adresse FF02 das Byte mit dem Wert AC, unter der Adresse FF03 das Byte mit dem Wert E8 befindet, sich folgender Vergleich ergibt:

R36: 10101100 R37: 11101000
$FF02: 10101100 $FF03: 11101000
R36&R37: 1010110011101000
$FF02&$FF03: 1010110011101000
Da das Multibyte R36&R37 gleich dem Multibyte $FF02&$FF03 ist, werden CY und ZR auf 1 gesetzt.

f) Inkrementierung (IC):
ICB DR:
Codierung: 88
Bei Ausführen dieses Befehls wird das Register DR um 1 erhöht.
Bsp: ICB R36 mit der Hex-Codierung 5E 88 bedeutet, daß zum Inhalt von Register 36 1 addiert wird.
ICM DR:
Codierung: 89
Bei Ausführung dieses Befehls wird zu dem Multibyte DR-DR+1 die Zahl 1 addiert, und das Ergebnis in dem Multibyte DR-DR+1 abgelegt.
Bsp: ICM R36 mit der Hex-Codierung 5E 89 bedeutet, daß zum Multibyte R36-R37 1 addiert wird. Das Ergebnis steht nach der Operation in den Registern R36-R37.

g) Decrementierung (DC):
DCB DR:
Codierung: 8A
Bei Ausführung dieses Befehls wird das Register DR um 1 erniedrigt.
Bsp: DCB R36 mit der Hex-Codierung 5E 8A bedeutet, daß der Inhalt des Registers 36 um 1 erniedrigt wird.
DCM DR:
Codierung: 8B
Bei Ausführung dieses Befehls wird der Inhalt des Multibytes DR-DR+1 um 1 erniedrigt.
Bsp: DCM R36 mit der Hex-Codierung 5E 8B bedeutet, daß der Inhalt des Multibytes R36-R37 um 1 erniedrigt wird.

h) Zweier-Komplement:
TCB DR:
Codierung: 8C
Bei Ausführen dieses Befehls werden alle Einsen im Register DR durch Nullen, alle Nullen in DR durch Einsen ersetzt.
Bsp: ICB R36 mit der Hex-Codierung 5E 8C bedeutet, daß, wenn Register 36 den Inhalt DB enthält, sich folgende Umwandlung ergibt:
R36: 1101 1011
0010 0100
Nach der Umwandlung enthält Register 36 also den Inhalt 24.
TCM DR:
Codierung: 8D
Bei Ausführen dieses Befehls werden alle Einsen im Multibyte DR-DR+1 durch Nullen, alle Nullen durch Einsen ersetzt.
Bsp: ICM R36 mit der Hex-Codierung 5E 8D bedeutet, daß, wenn das Register 36 den Inhalt DB, das Register 37 den Inhalt 5E besitzt, sich folgender Verlauf ergibt:
R36: 1101 1011 R37: 0101 1110
0010 0100 1010 0001
Nach der Umwandlung besitzt Register 36 also den Inhalt 24, Register 37 den Inhalt A1.

i) Neuner-Komplement:
1. Im BIN-Modus:
Im binären Modus verhalten sich die beiden folgenden Operationen wie die äquivalente Operation im Zweier-Komplement.
2. Im BCD-Modus:
NCB DR:
Codierung: 8E
Man erhält den Wert der beiden Halbbytes des Registers DR, wenn man den ursprünglichen Wert der Halbbytes jeweils von hexadezimal 9 subtrahiert.
Bsp: NCB R36 mit der Hex-Codierung 5E 8E bedeutet, daß sich folgender Verlauf ergibt:

R36: 1001 0110 entspricht hexadezimal 96
Zieht man 96 von 99 ab, so ergibt sich mit 03 der neue Inhalt des Registers 36.

NCM DR:
Codierung: 8F
Man erhält die Werte der beiden Bytes DR-DR+1, indem man die ursprünglichen Werte von jeweils 99 subtrahiert.
Bsp: NCM R36 mit der Hex-Codierung 5E 8F bedeutet, daß sich folgender Verlauf ergibt:

R36:	1001 0110	R37:	0101 0111
	99-96=03		99-57=42

Mit 03 für das Register 36 und 42 für das Register 37 stehen die neuen Inhalte fest.

j) Subtraktionsbefehle:
Man unterscheidet zwei Modi, in denen diese Befehle wirken:
1. Im BIN-Modus:
   In diesem Fall wird von der Zahl, die subtrahiert wird, das Zweierkomplement gebildet (Umwandlung siehe oben unter Punkt f).
2. Im BCD-Modus:
   In diesem Fall wird von der Zahl, die subtrahiert wird, das Neunerkomplement gebildet (Umwandlung siehe oben unter Punkt g).

Auf die Unterscheidung zwischen diesen beiden Fällen verzichte ich bei der näheren Erklärung der Assemblerbefehle. Man könnte die Subtraktion auch als Addition zwischen der einen Zahl und dem Komplement der anderen Zahl auslegen.

SBB DR, AR:
Codierung: C4
Bei dieser Operation wird zum Inhalt des Registers DR das Komplement des Registers AR addiert.
Bsp: SBB R36, R32 mit der Hex-Codierung 5E 1A C4 bedeutet, daß der Inhalt des Registers 36 und das Komplement des Registers 32 addiert und in Register 36 abgelegt werden.

SBB DR, =literal:
Codierung: CC
Bei dieser Operation wird zum Inhalt des Registers DR das Komplement der ganzen Zahl, die im literal-Feld zu finden ist, addiert.
Bsp: SBB R36, =124 mit der Hex-Codierung 5E CC 7C bedeutet, daß zum Inhalt von Register 36 das Komplement von 124 (hex 83 im BIN-Modus, hex 4B im BCD-Modus) addiert wird.

SBBD DR, AR:
Codierung: DC
Zum Inhalt des Registers DR wird das Komplement des Inhaltes addiert, der an der Stelle des Memory zu finden ist, auf dessen Lokation das Register AR zeigt.
Bsp: SBBD R36, R46 mit der Hex-Codierung 5E 26 DC bedeutet, daß, wenn Register 46 A1 und das Register 47 EF enthält, und an der Stelle EFA1 im Memory sich der Inhalt 29 befindet, das Komplement von 29 (hex D6 im BIN-Modus, hex 70 im BCD-Modus) zum Inhalt von Register 36 dazuaddiert wird.

SBBD DR, =label: Codierung: D4
Zum Inhalt des Registers DR wird das Komplement des Inhaltes addiert, der an der Stelle des Memory zu finden ist, die durch das label festgelegt ist.
Bsp: SBBD R36, =KEYBOARD mit der Hex-Codierung 5E D4 02 FF bedeutet, daß, wenn sich unter der Adresse FF02 ein Inhalt befindet, das Komplement dieses Inhaltes zum Inhalt des Registers 36 dazuaddiert wird.

SBM DR, AR:
Codierung: C5
Bei dieser Operation wird zum Inhalt des Multibytes, beginnend mit Register DR, das Komplement des Multibytes, beginnend bei AR, addiert.
Bsp: SBM R36, R46 mit der Hex-Codierung 5E 26 C5 bedeutet, daß zum Register 36 das Komplement des Registers 46, zum Register 37 das Komplement des Registers 47 addiert wird.

SBM DR, =literal:
Codierung: CD
Zum Inhalt des Multibytes, beginnend bei DR, werden die Komplemente der im literal-Feld angegebenen ganzen Zahlen addiert.
SBMD DR, AR:
Codierung: DD
Zum Inhalt des Multibytes, beginnend bei DR, wird das Komplement des Inhaltes addiert, der sich, beginnend mit den Stellen im Memorybereich befindet, auf die die register AR bis AR+1 zeigen.
SBMD DR, =label:
Codierung: D5
Zum Inhalt des Multibytes, beginnend bei DR, wird das Komplement des Inhaltes addiert, der sich beginnend mit den Stellen im Memory-Bereich befindet, auf die die Adresse des labels zeigt.

6.3.5. Modus-Funktionen:
Nachdem nun in den letzten Befehlen die Rede von den zwei Modi BCD und BIN war, muß nun geklärt werden, wie man in diese beiden Modi kommt.
BIN:
Codierung: 98
Mit Hilfe dieses Befehls wird der Rechner in den binären Modus gesetzt. Dies bedeutet, daß er die folgenden Assemblerbefehle nur im binären Modus abarbeitet.
BCD:
Codierung: 99
Mit Hilfe dieses Befehls wird der Rechner in den BCD-Modus umgeschaltet. Dies bedeutet, daß der Rechner die folgenden Assemblerbefehle nur im BCD-Modus abarbeitet.

6.3.6. Clear-Befehle:
CLB DR:
Codierung: 92
Mit Hilfe dieses Befehls wird das Register DR gelöscht.
Bsp: CLB R36 mit der Hex-Codierung 5E 92 bedeutet, daß das Register 36 gelöscht wird. Dies bedeutet, daß dem Register 36 der Inhalt 0 zugewiesen wird.
CLM DR:
Codierung: 93
Mit Hilfe dieses Befehls wird das Multibyte, beginnend mit Register DR, gelöscht.
Bsp: CLM R36 mit der Hex-Codierung 5E 93 bedeutet, daß die Register 36 und 37 gelöscht werden, dh. den Wert 0 erhalten.

6.3.7. Befehle mit dem E-Register (extend-Register):
CLE:
Codierung: 9D
Die ersten 4 Bits des extend-Registers erhalten den Wert 0000.
ICE:
Codierung: 9C
Der Inhalt des extend-Registers wird um 1 erhöht.
DCE:
Codierung: 9B
Der Inhalt des extend-Registers wird um 1 erniedrigt.
Weitere Einzelheiten zu diesem Zusatzregister im Kapitel 1 unter Punkt 1.

6.3.8. Pointerbefehle:
DRP DR:
Codierung: zwischen 40 und 7F
Direkt in die Befehlscodierung aufgenommen ist die Codierung für das Register, und zwar in der Weise:
die ersten beiden Bits der Codierung sind immer 01, die folgenden Bits enthalten den Wert des Registers DR in der gewohnten Weise.
Bsp: DRP R36
oktal 36 entspricht binär: 0001 1110
Bit 6 und Bit 7 erhalten den Wert 01,
damit ergibt sich für die Codierung des Befehls DRP R36:
0101 1110, also hexadezimal 5E

ARP AR:
Codierung: zwischen 00 und 3F
Direkt in die Befehlscodierung aufgenommen ist die Codierung für das AR-register, und zwar folgendermaßen:
Die ersten beiden Bits der Codierung haben den Wert 00, die folgenden Bits enthalten den Wert des Registers AR in der gewohnten Weise.
Bsp: ARP R36
oktal 36 entspricht binär: 0001 1110
Bit 6 und Bit 7 enthalten den Wert 00,
damit ergibt sich für die Codierung des Befehls ARP R36:
0001 1110, also hexadezimal 1E.

6.3.9. Sprungbefehle:
JMP, literal:
Codierung: F0
Dieser Sprung ist unabhängig. Immer wenn dieser Befehl erfolgt, wird ein Unterprogramm aufgerufen.
JNO, literal:
Codierung: F1
Ein Sprung erfolgt, wenn bei einer Operation ein Übertrag erfolgte. (wenn ein Übertrag erfolgt, wird das Flag OVF (Overflow) auf 1 gesetzt.
JPS, literal:
Codierung: F5
Der Sprung erfolgt, wenn die Operation EXOR zwischen den Flags OVF (Beschreibung siehe oben) und NG (NG ist 1, wenn eine Zahl bzw. ein Resultat, Ergebnis einer Operation negativ ist) den Wert 1 hat. Ansonsten erfolgt kein Sprung.
JNG, literal:
Codierung: F4
Der Sprung erfolgt, wenn die Operation EXOR zwischen den Falgs OVF und NG den Wert 0 hat. Ansonsten erfolgt kein Sprung.
Bsp: Im Anzeigeregister steht die Zahl 1 1001 0010, was bedeutet, daß die Zahl negativ ist (1 in Bit 8 von rechts), und daß ein Übertrag stattgefunden hat (1 in Bit 9 von rechts). Nach der Operation EXOR mit 2 Einsen erhält man als Ergebnis die 0.
Folgt der Operation, mit der die negative Zahl ins Anzeigeregister kam, der Sprungbefehl JNG, so erfolgt auch der Sprung nach der angegebenen Adresse.
JRZ, literal:
Codierung: FE
Ein Sprung erfolgt, wenn das äußerst rechte Digit den Wert 0 hat. Ansonsten erfolgt kein Sprung.
JRN, literal:
Codierung: FF
Ein Sprung erfolgt, wenn das äußerst rechte Digit einen Wert ungleich 0 hat.
JLZ, literal:
Codierung: FC
Ein Sprung erfolgt, wenn das äußerst linke Digit im Anzeigeregister den Wert 0 hat.
JLN, literal:
Codierung: FD
Ein Sprung erfolgt, wenn das äußerst linke Digit in der Anzeige einen Wert ungleich 0 hat.
JEZ, literal: Codierung: F9
Ein Sprung erfolgt, wenn das E-Register (extend-Register) den Wert 0 besitzt.
JEN, literal:
Codierung: F8
Ein Sprung erfolgt, wenn das E-Register einen Wert ungliech 0 als Inhalt hat.
JOD, literal:
Codierung: F2

Ein Sprung erfolgt, wenn die Zahl im Anzeigeregister ungerade ist.
JEV, literal:
Codierung: F3
Ein Sprung erfolgt, wenn die Zahl im Anzeigeregister gerade ist.
JCY, literal:
Codierung: FB
Ein Sprung erfolgt, wenn das Flag CY den Wert 1 hat. CY nimmt diesen Wert bei Vergleichen an, wobei der Operand im Datenregister größer oder gleich dem Operanden im Adreßregister ist. Nähere Einzelheiten bei den Vergleichsbefehlen (Compare).
JNC, literal:
Codierung: FA
Ein Sprung erfolgt, wenn das Flag CY den Wert 0 hat. CY nimmt den Wert 0 an, wenn bei Vergleichen der Operand im Datenregister kleiner ist als der Operand im Adreßregister. Nähere Einzelheiten bei den Vergleichsbefehlen (Compare).
JSB, label:
Codierung: CE
Es erfolgt der Aufruf eines Unterprogrammes, das sich im Betriebssystem oder im RAM-Bereich des Rechners befindet. Diese Unterprogramme enden alle mit dem Befehl RTN, was bedeutet, daß ein Rücksprung zum Befehl erfolgt, der sich hinter dem JSB-Befehl befindet. Im JSB-Befehl gibt das label die Adresse an, unter der das gewünschte Unterprogramm zu finden ist.
JSB XR, label:
Codierung: C6
Es erfolgt, wie beim JSB-Befehl, der Aufruf eines Unterprogrammes, mit der Ausnahme, daß der Rechner sich bei diesem Befehl noch die Adresse, unter der das Unterprogramm zu finden ist, selbst errechnen muß, dh. die Adresse ist variabel. Die Adresse, unter der das Unterprogramm steht, wird errechnet, indem zu dem Wert des labels noch den Wert von XR-XR+1 dazu addiert.
JNZ, literal:
Codierung: F6
Dieser und der folgende Befehl dienen zum Simulieren der IF-THEN und der FOR-TO-Anweisung.
a) Simulieren der IF-THEN-Anweisung:
   Vor dem Sprung-Befehl steht dabei ein Vergleichsbefehl. Sind die Operanden beim Vergleich gleich, so wird das Flag ZR auf 1 gesetzt, ansonsten auf 0. Der nun auf den Vergleichsbefehl folgende JNZ-Befehl führt nur dann zu einem Sprung, wenn das Flag ZR den Wert 0 hat.
b) Simulieren einer FOR-TO-Anweisung:
   Vor dem Sprungbefehl steht hierbei ein Befehl, bei dem ein Byte oder Multibyte decrementiert oder inkrementiert wird. Hat das Byte, das in diesem Befehl erhöht oder erniedrigt wurde, den Wert 0 erreicht, wird das Flag ZR auf 1 gesetzt. Ist sein Wert ungleich 0, so hat ZR den Wert 0. Der nun folgende JNZ-Befehl führt nur dann zu einem Sprung, wenn das Falg ZR den Wert 0 hat.
JZR, literal:
Codierung: F7
Wie oben erwähnt dient auch dieser Befehl zum Simulieren der IF-THEN- sowie der FOR-TO-Anweisung. Die Simulation erfolgt wie beim JNZ-Befehl erwähnt:
a) Simulieren einer IF-THEN-Anweisung:
   Vor dem Sprungbefehl steht hierbei ein Vergleichsbefehl. Je nachdem wie das Ergebnis des Vergleiches aussieht, wird das Flag ZR gesetzt (Sind die Operanden gleich wird ZR=1, sonst ist ZR=0). Ein Sprung mit dem JZR-Befehl erfolgt, wenn das Flag ZR den Wert 1 hat.
b) Simulieren einer FOR-TO-Anweisung:
   Vor dem Sprungbefehl steht hierbei ein DC- oder ein IC-Befehl. Hat der Operand in diesem Befehl den Wert 0 erreicht, wird ZR auf 1 gesetzt, ansonsten hat es den Wert 0. Der Sprung mit dem JZR-Befehl erfolgt, wenn der Operand den Wert 0, dh. ZR den Wert 1 hat.

6.3.10. Stackbefehle:

RTN:

Codierung: 9E

Dieser Befehl steht hinter jedem Unterprogramm und zum Abschluß der Hauptteile eines Assemblerprogrammes. Bei Ausführung dieses Befehls wird der return-stack-Pointer um 2 erniedrigt, und die return-Adresse in den Programmzähler geschrieben.

SAD:

Codierung: 9A

Bei Ausführung dieses Befehls werden 3 Bytes in den Stack gebracht, um den Status des Rechners zu speichern. Dabei enthalten die einzelnen Bytes folgende Informationen:

```
Byte 1: Bit 0-5  Wert von ARP
        Bit 6    Wert des CY-Flags
        Bit 7    Wert des OVF-Flags
Byte 2: Bit 0-5  Wert von DRP
        Bit 6    Wert des DCM-Flags (näheres siehe in Kapitel 1)
        Bit 7    Wert des OVF-Flags
Byte 3: Bit 0    Wert von LSB
        Bit 1    Wert des RDZ-Flags
        Bit 2    Wert des ZR-Flags
        Bit 3-5  Werte 0
        Bit 6    Wert des LDZ-Flags
        Bit 7    Wert von MSB
```

Der Stack-Pointer wird bei Ausführung dieser Funktion um 1 erhöht, insgesamt 3-mal. Der Status selbst wird durch diese Funktion nicht verändert.

PAD:

Codierung: 9F

Bei Ausführung dieser Funktion wird ein Status hergestellt, über den Informationen in 3 Stackregistern enthalten sind.

```
ARP:        Byte 1, Bit 0-5
DRP:        Byte 2, Bit 0-5
OVF:        Byte 1, Bit 7          oder Byte 2, Bit 7
CY :        Byte 1, Bit 6
DCM:        Byte 2, Bit 6
LSB:        Byte 3, Bit 0
RDZ:        Byte 3, Bit 1
ZR :        Byte 3, Bit 2
LDZ:        Byte 3, Bit 6
MSB:        Byte 3, Bit 7
```

Der Stack-Pointer wird bei Ausführung dieser Funktion um 1 erhöht, insgesamt 3-mal.

TSB DR:

Codierung: 90

Der Status des Inhaltes eines Registers DR wird getestet. Je nachdem, wie der Test ausfällt, werden Statusindikatoren gesetzt:

```
DCM: wird gesetzt, wenn der Inhalt im BCD-Modus vorliegt, sonst Wert 0.
E  : keine Wirkung
CY : wird in jedem Fall gelöscht (Wert 0)
OVF: wird in jedem Fall gelöscht (Wert 0)
OD : erhält den Wert des Bits 0 innerhalb des Registers DR
NG : erhält den Wert des Bits 7 innerhalb des Registers DR
ZR : erhält den Wert 1, wenn das Register DR leer ist, sonst den Wert 0
LDZ: haben die Bits 4-7 den Wert 0000, so hat LDZ den Wert 1, sonst 0.
RDZ: haben die Bits 0-3 den Wert 0000, so hat RDZ den Wert 1, sonst 0.
```

Bsp: TSB R36 mit der Codierung 5E 90 bedeutet, daß sich folgender Test ergibt, wenn Register 36 den Inhalt 0000 1011 hat.

```
DCM          Wert 1(BCD-Modus), Wert 0(BIN-Modus)
E            keine Wirkung
CY           Wert 0
OVF          Wert 0
OD           Wert 1
NG           Wert 0
ZR           Wert 0
```

LDZ Wert 1
RDZ Wert 0

TSM DR:
Codierung: 91
Der Status des Multibytes DR-DR+1 wird getestet. Die Statusindikatoren werden gesetzt, wie bei TSB DR.
Unterschiede:
OD : erhält den Wert des Bits 0 innerhalb des Registers DR+1
ZR : erhält den Wert 1, wenn die Register DR und DR+1 leer sind, sonst 0
LDZ: haben die Bits 4-7 des Registers DR den Wert 0000, so hat LDZ den Wert 1, sonst 0.
RDZ: haben die Bits 0-3 des Registers DR+1 den Wert 0000, so hat RDZ den Wert 1, sonst 0.
Bsp: TSB R36 mit der Hex-Codierung 5E 91 bedeutet, daß sich folgender Test ergibt, wenn Register 36 den Inhalt 0100 0011 und Register 37 den Inhalt 0000 1011 hat:

DCM	Wert 1 (BCD-Modus), Wert 0 (BIN-Modus)
E	keine Wirkung
CY	Wert 0
OVF	Wert 0
OD	Wert 1
NG	Wert 0
ZR	Wert 0
LDZ	Wert 0
RDZ	Wert 0

6.3.11. Verschiebebefehle:

a) Logische Verschiebung:

LLB DR:
Codierung: 84
Der Inhalt des Registers DR wird um 1 Bit nach links verschoben. Dabei finden folgende Verschiebungen statt.
7.Bit des Registers DR kommt nach CY
6.Bit des Registers DR kommt ins 7.Bit des Registers DR
5.Bit " " " " " 6.Bit " " "
4.Bit " " " " " 5.Bit " " "
3.Bit " " " " " 4.Bit " " "
2.Bit " " " " " 3.Bit " " "
1.Bit " " " " " 2.Bit " " "
0.Bit " " " " " 1.Bit " " "
Danach wird das 0.Bit des neuen Registers DR gelöscht.
Bsp: LLB R36 mit der Hex-Codierung 5E 84 bedeutet enn Register 36 den Inhalt 1100 0111 hat, findet folgende Verschiebung statt:

CY	R36
1	1000 1110

LLM DR:
Codierung: 85
Der Inhalt der Register DR-DR+1 wird um 1 Bit nach links verschoben.
Dabei finden folgende Verschiebungen statt.
CY erhält den Wert des 7.Bits des Registers DR+1
Das 7.Bit des Registers DR+1 erhält den Wert des 6.Bits des Registers DR+1
" 6.Bit " " " " " " " 5.Bits " " "
" 5.Bit " " " " " " " 4.Bits " " "
" 4.Bit " " " " " " " 3.Bits " " "
" 3.Bit " " " " " " " 2.Bits " " "
" 2.Bit " " " " " " " 1.Bits " " "
" 1.Bit " " " " " " " 0.Bits " " "
" 0.Bit " " " " " " " 7.Bits des Registers DR
Das 7.Bit des Registers DR erhält den Wert des 6.Bits des Registers DR
" 6.Bit " " " " " " " 5.Bits " " "
" 5.Bit " " " " " " " 4.Bits " " "
" 4.Bit " " " " " " " 3.Bits " " "
" 3.Bit " " " " " " " 2.Bits " " "

Das 2.Bit des Registers DR erhält den Wert des 1.Bits des Registers DR
" 1.Bit " " " " " " " 0.Bits " " "
" 0.Bit " " " " " " 0.

LRB DR:
Codierung: 86
Der Inhalt des Registers DR wird um 1 Bit nach rechts verschoben. Dabei finden folgende Verschiebungen statt.
CY erhält den Wert des 0.Bits des Registers DR
Das 0.Bit des Registers DR erhält den Wert des 1.Bits des Registers DR
" 1.Bit " " " " " " " 2.Bits " " "
" 2.Bit " " " " " " " 3.Bits " " "
" 3.Bit " " " " " " " 4.Bits " " "
" 4.Bit " " " " " " " 5.Bits " " "
" 5.Bit " " " " " " " 6.Bits " " "
" 6.Bit " " " " " " " 7.Bits " " "
" 7.Bit " " " " " " 0.

Bsp: LRB R36 mit der Hex-Codierung 5E 86 bedeutet: Wenn Register 36 den Inhalt 1100 0111 hat, findet folgende Verschiebung statt:
CY R36

[1] [0110 0011] im BIN-Modus

LRM DR:
Codierung: 87
Der Inhalt der Register DR bis DR+1 wird um 1 Bit nach rechts verschoben. Dabei finden folgende Verschiebungen statt.
CY erhält den Wert des 0.Bits des Registers DR
Das 0.Bit des Registers DR erhält den Wert des 1.Bits des Registers DR
" 1.Bit " " " " " " " 2.Bits " " "
" 2.Bit " " " " " " " 3.Bits " " "
" 3.Bit " " " " " " " 4.Bits " " "
" 4.Bit " " " " " " " 5.Bits " " "
" 5.Bit " " " " " " " 6.Bits " " "
" 6.Bit " " " " " " " 7.Bits " " "
" 7.Bit " " " " " " " 0.Bits des Registers DR+1
Das 0.Bit des Registers DR+1 erhält den Wert des 1.Bits des Registers DR+1
" 1.Bit " " " " " " " 2.Bits " " "
" 2.Bit " " " " " " " 3.Bits " " "
" 3.Bit " " " " " " " 4.Bits " " "
" 4.Bit " " " " " " " 5.Bits " " "
" 5.Bit " " " " " " " 6.Bits " " "
" 6.Bit " " " " " " " 7.Bits " " "
" 7.bit " " " " " " 0.

b) Erweiterte Verschiebung:
ELB DR:
Codierung: 80
Der Inhalt des Registers DR wird um 4 Bit nach links verschoben. Dabei findet folgende Verschiebung statt (im BCD-Modus, Erklärungen später).
E erhält den Wert der Bits 4-7 des Registers DR.
Die Bits 4-7 des Registers DR erhalten den Inhalt der Bits 0-3 von DR
Die Bits 0-3 des Registers DR erhalten den ursprünglichen Inhalt von E.
Bsp: ELB R36 mit der Hex-Codierung 5E 80 bedeutet: Wenn Register 36 den Inhalt 1000 0000 und das Register E den Inhalt 0001 besitzt, so findet folgende Verschiebung statt:
R36 E

[0000 0001] [1000] im BCD-Modus

ELM DR:
Codierung: 81
Der Unterschied zu dem Befehl LLM DR besteht darin, daß das 0.Bit des Registers DR den ursprünglichen Wert von CY erhält.
ERB DR:
Codierung: 82

Der Unterschied zu dem Befehl LRB DR besteht darin, daß bei ERB DR das 7.Bit des Registers DR den ursprünglichen Wert von CY erhält.
ERM DR:
Codierung: 83
Der Unterschied zu dem Befehl LRM DR besteht darin, daß bei ERM DR das 7.Bit des Registers DR+1 den ursprünglichen Wert von CY erhält.

c) Zusatzbemerkungen:
Man unterscheidet bei den Verschiebebefehlen zwei Arten, je nachdem in welchem Modus die Verschiebung passiert.
- Verschiebung im BIN-Modus:
  Die Verschiebung findet zwischen dem Byte oder Multibyte, beginnend mit DR, und dem Inhalt des Flags CY um 1 Bit statt (siehe Beschreibung LRB DR, LRM DR, ELM DR, ERB DR, ERM DR, LLB DR und LLM DR).
- Verschiebung im BCD-Modus:
  Die Verschiebung findet zwischen dem Byte oder Multibyte, beginnend mit DR, und dem Register E um 4 Bit statt (siehe Beschreibung von ELB DR).

6.3.12. Stack-Adressierung:

a) Stack-Instruktionen:
In der Stack-Adressierung, dient ein Registerpaar als Pointer auf den Stack im Memory des Rechners. Am Kopf des Stacks findet ein Laden oder Speichern statt, und das Registerpaar wird vergrößert oder verkleinert, bis der neue Kopf des Stacks nach der Stackverschiebung erreicht ist. Man unterscheidet Befehle, die Daten in den Stack im Hauptspeicher laden, und Befehle, die Daten aus dem Stack im Hauptspeicher herauslesen. Diese Stacks können indirekt oder direkt adressiert werden.

b) Stack-Adressierung:
Man kann einen Stack adressieren von einem beliebigen CPU-Registerpaar aus. Die Register 6 und 7 sind durch die Hardware vorbestimmt, daß sie immer auf den Stack zeigen, in dem die Rücksprungadresse bei einem Unterprogrammaufruf gespeichert ist. Dieser Stack hat eine Größe von 512 Bytes. Wird ein Unterprogramm aufgerufen, wird automatisch eine Adresse in den Stack geladen, und zwar die Adresse, auf der der Rechner vor dem Aufruf des Unterprogrammes gerade stand. Beim Rücksprung wird der Programmzähler wieder automatisch mit der Adresse aus dem Stack geladen, die an der Spitze des Stacks steht, dh. von der der letzte Aufruf ausgegangen war. Dieses Laden der Rücksprungadresse in den Programmzähler bewirkt, daß das Programm mit dem Befehl hinter dem Unterprogrammaufruf wieder fortgesetzt wird. Der Stack R6 wird auch durch die SAD- und PAD-Instruktion beeinflußt, da in ihm auch eines der drei Bytes verarbeitet wird.
Es gibt noch einen weiteren Stack, der die Systemroutinen eines Programmes leicht beeinflussen kann: der Operationsstack in den Registern 12 und 13. Dieser Stack wird dazu benutzt, Parameter zwischen den Systemroutinen eines Programms hin- und herzuschieben.
Stacks können vergrößert oder verkleinert werden. Ein wachsender Stack ist ein Stack, der gefüllt wird in Richtung zu höheren Memoryadressen hin, und der Daten abgibt in Richtung zu kleineren Memoryadressen.
Ein fallender Stack ist ein Stack, der gefüllt wird in Richtung zu niedrigeren Memoryadressen hin, und der Daten abgibt in Richtung zu höheren Memoryadressen hin. Um Verwirrungen sich zu ersparen, sollte man für wachsende und fallende Stacks jeweils getrennt nur eine Art von Befehlen benutzen, dh. Überschneidungen zwischen wachsendem und fallendem Stack zu vermeiden.
Bei Stackadressierungen ist der Stackzeiger im Register AR enthalten. Sein Vorzeichen gibt an, ob wachsender (+) oder fallender (-) Stack. Ein Stack wird aktiviert, indem man den Pointer ARP auf die Position des Stackpointers setzt.
Bei einem wachsendem Stack muß das Register AR auf die Position im Stack gesetzt werden, die unmittelbar auf die augenblickliche Position folgt. Bei einem fallendem Stack muß das Register AR auf die momentane Position des Stack-Pointers zeigen.

c) Adressierungsformen der Stackadressierung:

1. direkte Adressierung:
   In dieser Adressierungsform wird der Stack veranlaßt, Daten zu enthalten. Speichern in den Stack bedeutet zugleich Füllen des Stacks, Laden aus dem Stack bedeutet zugleich Leeren des Stacks.
   Zum Speichern in einen wachsenden Stack zeigt AR auf die Stelle, wo Daten abgespeichert werden müssen. Während des Speichervorgangs wird AR bei jedem Byte, das eingelesen wird, um 1 erhöht.
   Zum Laden aus einem wachsenden Stack wird AR zuerst um die Anzahl der zu lesenden Bytes erniedrigt. Dann zeigt bei jedem Lesevorgang AR auf die Position, von der Bytes geladen werden.
   Zum Speichern in einen fallenden Stack, wird AR zuerst um die Anzahl der zu speichernden Bytes erniedrigt. Danach werden die Daten in die Stackpositionen geladen. Nach dem Laden wird AR wider um 1 erniedrigt. Dies geht solange bis alle Bytes geladen sind.
   Zum Laden aus einem fallenden Stack zeigt AR auf die Stelle, von der an Daten gelesen werden. Während des Lesevorgangs wird AR immer um 1 erhöht, bis sämtliche Daten eingelesen sind.
2. indirekte Adressierung:
   In dieser Adressierungsform wird der Stack veranlaßt, eine Liste von Adressen zu enthalten. Diese Adressen wiederum zeigen auf die Stellen im Memory des Rechners, von dem Daten gelesen oder in den Daten gespeichert werden sollen.
   Zum Speichern in einen wachsenden Stack, zeigt AR zur effektiven Adresse. Nach dem Speichern, wird AR immer um 2 erhöht.
   Zum Laden aus einem wachsenden Stack wird AR zuerst immer um 2 erniedrigt, damit es auf die effektive Adresse zeigt. Danach wird die effektive Adresse in das CPU-Register geladen, das durch den Wert im DRP innerhalb des Assemblerbefehls festgelegt wird.

d) Die Befehle in Zusammenhang mit einem wachsendem Stack:

PUBD DR, +AR:
Codierung: E4
Dieser Befehl dient zum direkten Speichern in den Stackbereich. Dabei wird AR nach jedem Byte um 1 erhöht.

PUMD DR, +AR:
Codierung: E5
Dieser Befehl dient zum Speichern eines Multibytes direkt in den Stackbereich. Dabei wird AR um 1 erhöht.

PUBI DR, +AR:
Codierung: EC
Dieser Befehl dient zum indirekten Speichern einer Adresse in den Stackbereich. Dabei wird AR nach jeder Adresse um 2 erhöht.

PUMI DR, +AR:
Codierung: ED
Dieser Befehl dient zum indirekten Speichern eines Multibytes in den Stackbereich. Dabei wird AR nach jeder Adresse um 2 erhöht.

POBD DR, -AR:
Codierung: E2
Dieser Befehl dient zum direkten Laden eines Bytes aus dem Stackbereich. Dabei wird AR vor jedem Ladevorgang um 1 erniedrigt.

POMD DR, -AR:
Codierung: E3
Dieser Befehl dient zum direkten Laden eines Multibytes aus dem Stackbereich. Dabei wird AR vor jedem Ladevorgang um 1 erniedrigt.

POBI DR, -AR:
Codierung: EA
Dieser Befehl dient zum indirekten Laden einer Adresse aus dem Stackbereich. Dabei wird AR vor jedem Ladevorgang um 2 erniedrigt.

POMI DR, -AR:
Codierung: EB
Dieser Befehl dient zum indirekten Laden eines Multibytes aus dem Stackbereich. Dabei wird AR vor jedem Ladevorgang um 2 erniedrigt.

e) Die Befehle in Zusammenhang mit einem fallenden Stack:
PUBD DR, -AR:
Codierung: E6
Dieser Befehl dient zum direkten Speichern eines Bytes in den Stackbereich. Dabei wird AR vor jedem Speichervorgang um 1 erniedrigt.
PUMD DR, -AR:
Codierung: E7
Dieser Befehl dient zum direkten Speichern eines Multibytes in den Stackbereich. Dabei wird AR vor jedem Speichervorgang um 1 erniedrigt.
PUBI DR, -AR:
Codierung: EE
Dieser Befehl dient zum indirekten Speichern einer Adresse in den Stackbereich. Dabei wird AR vor jedem Speichervorgang um 2 erniedrigt.
PUMI DR, -AR:
Codierung: EF
Dieser Befehl dient zum indirekten Speichern eines Multibytes in den Stackbereich. Dabei wird AR vor jedem Speichervorgang um 2 erniedrigt.
POBD DR, +AR:
Codierung: E0
Dieser Befehl dient zum direkten Laden eines Bytes aus dem Stackbereich. Dabei wird AR nach jedem Byte um 1 erhöht.
POMD DR, +AR:
Codierung: E1
Dieser Befehl dient zum direkten Laden eines Multibytes aus dem Stackbereich. Dabei wird AR nach jedem Byte um 1 erhöht.
POBI DR, +AR:
Codierung: E8
Dieser Befehl dient zum indirekten Laden einer Adresse aus dem Stackbereich. Dabei wird AR nach jeder Adresse um 2 erhöht.
POMI DR, +AR:
Codierung: E9
Dieser Befehl dient zum indirekten Laden eines Multibytes aus dem Stackbereich. Dabei wird AR nach jeder Adresse um 2 erhöht.

6.4. Entwicklung eines LEX-Files:

Im folgenden möchte ich erklären, wie man von einer Idee über die Entwicklungsarbeit zum Ergebnis, dem eigentlichen LEX-File kommt.

6.4.1. Die Idee:

Durch die Escape-Codes in Verbindung mit dem DISP-Befehl kann man Cursor und Zeichen zu einer beliebigen Stelle in der Anzeige des Rechners bewegen. Dies ist jedoch noch sehr aufwendig, vor allen Dingen fehlt noch die Möglichkeit, innerhalb eines Programms aus der Anzeige leicht den Inhalt zu lesen.
Um diese Lücke zu beseitigen, sollen nun zwei Befehle kreiert werden, mit deren Hilfe ich die Anzeige manipulieren kann. Diese beiden Befehle möchte ich nun kurz vorstellen:

RLCD (Wert von 0-100) Mit diesem Befehl soll der Inhalt aus der Anzeige bzw der Zustand der LCD-Inikatoren abgefragt werden.

WLCD (Wert von 0-100), zu schreibender Inhalt
Mit diesem Befehl soll man Bytes in die Anzeige bringen bzw die LCD-Inikatoren verändern können.

6.4.2. Einführung in die Beschreibung der LCD-Anzeige:

Die LCD-Anzeige des HP-75C beruht auf einem genau abgestimmten Zusammenspiel von Soft- und Hardware. Die Hardwareangaben zur Anzeige finden Sie im 1.Kapitel. An dieser Stelle möchte ich nur auf die Software der Anzeige, die ja wichtig ist für die Entwicklung der Befehle RLCD und WLCD, näher eingehen.

a) LCDOFF: Dieser Pointer befindet sich an der Adresse $82E0.
Hat dieser Pointer den Wert 0, so ist die Anzeige aktiv.
Hat der Pointer den Wert 255, so ist die Anzeige inaktiv.
Demnach kann dieser Pointer nur zwei verschiedene Werte (0 und 255) annehmen.

b) DEAD: Dieses Flag befindet sich an der Adresse $82E1.
Es hat den Wert 0, wenn der Inhalt des Anzeigebuffers in der Anzeige steht.
Es hat den Wert 255, wenn der Inhalt des Anzeigebuffers nicht in der Anzeige steht.

c) LCDPTR: Dieser Pointer belegt die Adressen $82E2 und $82E3. Er gibt in diesen beiden Bytes die Position des Cursors an.
Bsp: Hat die Adresse $82E2 den Inhalt E6 und die Adresse $82E3 den Inhalt 82, so befindet sich der Cursor bei der Adresse $82E6, was der 1. Position in der Anzeige entspricht (unter e) näheres).

d) LCDWIN: Dieser Pointer belegt die Adressen $82E4 und $82E5. Er gibt in diesen beiden Bytes die Position des linken Zeichens, das in der 32-stelligen Anzeige sichtbar ist, innerhalb des 96-stelligen Anzeigebuffers an. Die Adresse wird festgelegt wie bei LCDPTR.

e) Beginn von LCDBUFFER: Die folgenden 96 Bytes repräsentieren die 96-stellige Anzeige des HP-75C. In diesem Buffer werden die Zeichen dezimal gespeichert.

6.4.3. Umsetzung der LCD-Software auf die gewünschten Befehle:
Bei der Umsetzung wird von der gewünschten Adresse die Adresse $82E0 subtrahiert (natürlich intern). Das führt dazu, daß man mit dem Befehl RLCD(0) den Status von LCDOFF, mit RLCD(6) den Wert des ersten Zeichens in der Anzeige abfragen kann.

6.4.4. Entwicklung des LEX-Files:
Zur Entwicklung des LEX-Files benutze ich die Formblätter. Zunächst muß ich jedoch noch auf einzelne Adressen eingehen, die im LEX-File vorkommen:

$ 3E8B: An dieser Adresse befindet sich die Subroutine 'ONEB'. Sie wandelt eine im Befehl vorkommende Integer-Zahl um in binäre Form und speichert sie in den Registern 46-47.

$ 466D: An dieser Stelle befindet sich die Subroutine 'SYSJSB'. Mit Hilfe dieser Routine erfolgt ein Sprung ins System-ROM. Dort wird der eingegebene LEX-Befehl vom System aufgerufen.

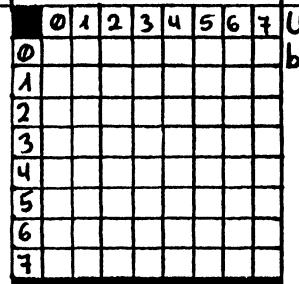
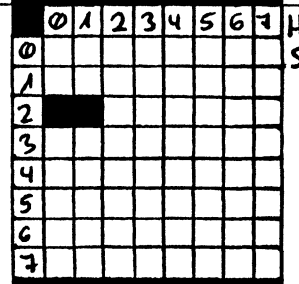
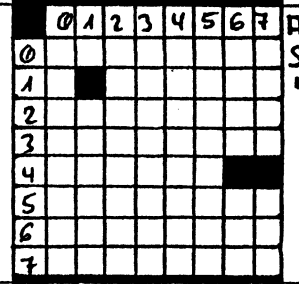
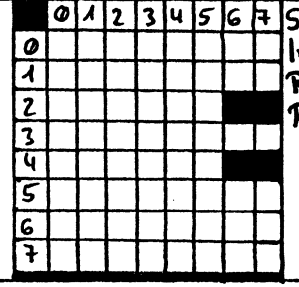
$ 4C99: An dieser Adresse befindet sich die Subroutine 'ERROR+'. Sie regelt die Fehlerfeststellung. Nach Aufruf dieser Subroutine erhält das Register E den Wert 0, wenn kein Fehler, den Wert 1, wenn ein Fehler vorliegt.

$ FCE4: An dieser Adresse befindet sich die Subroutine 'PUINTG'. Mit ihrer Hilfe wird eine binäre Zahl in den Registern 36-37 in eine reelle Zahl in den Registern 40-47 umgewandelt.

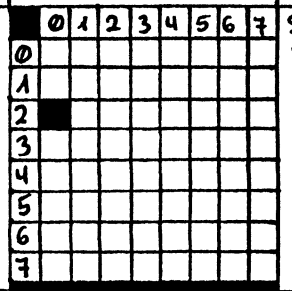
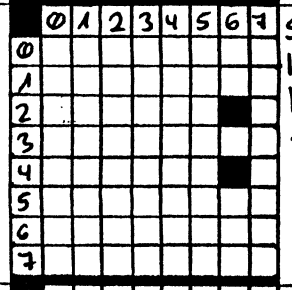
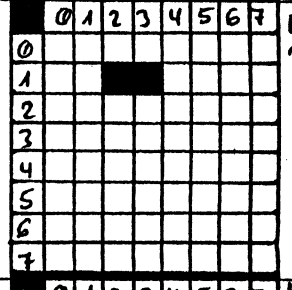
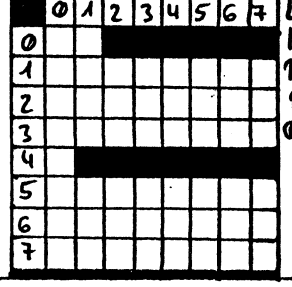
Als Basisadresse für das gewünschte LEX-File wird die Adresse 40 16 festgelegt.

Sonstige Vereinbarungen werden nicht getroffen.

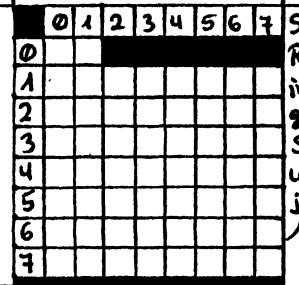
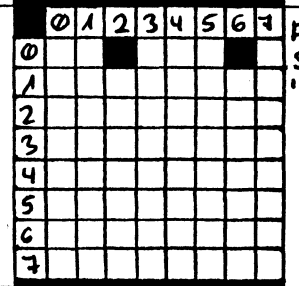
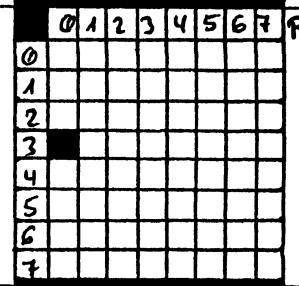
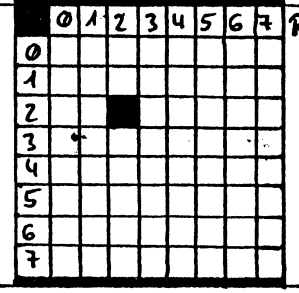
LCD

$	Befehl	PC	DR	AR	OV	CY	NG	LZ	ZR	RZ	OD	DC	E	BKP1	BKP2	PTR1	PTR2	Register	Kommentar
871B	BIN	871C										0							Umschalten in binären Modus
871C	SBM R20, $60F1	8720	20									0							Herstellen einer Sprungadresse
8720	JSB $3E8B	3E8B	46									0	#0						Aufruf der Subroutine 'ONEB'
8723	STM R46, R26	8726	46	26								0							Speichern des Inhaltes von R46-47 in den Registern 26-27

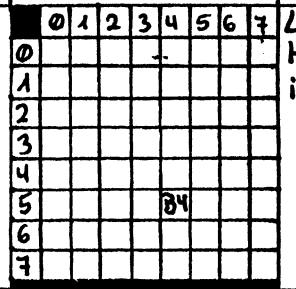
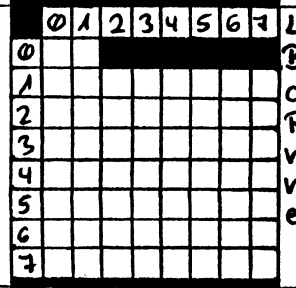
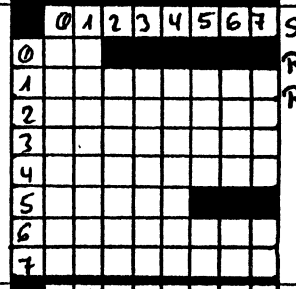
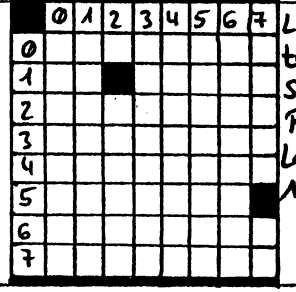
LCD

$	Befehl	PC	DR	AR	OV	CY	NG	LZ	ZR	RZ	OD	DC	E	BKP1	BKP2	PTR1	PTR2	Register	Kommentar
8726	JSB XR 20, $6133	*	46	20								0							Sprung in die Vektoradressen
8728	STBD R26, XR46, $82E0	8727										0							Speichern des Inhaltes des Wertes + Adresse $82E0.
872A	RTN											0							Ende der MAIN Routine
872C	LDM R02, R41	8733	02	41								0							Laden des Inhaltes von R41-47 in die Register R02-07.

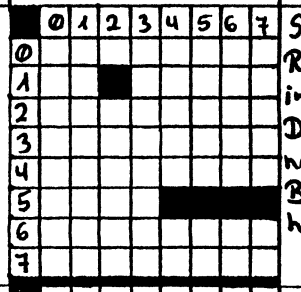
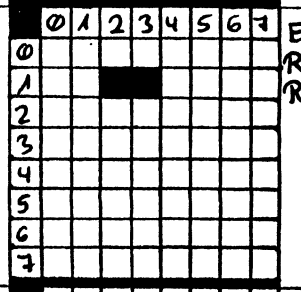
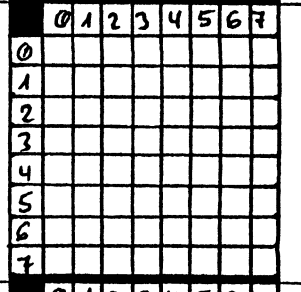
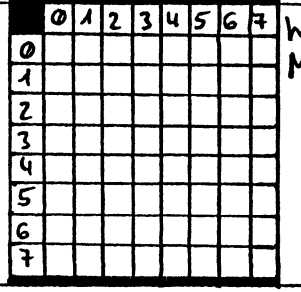
LCD

$	Befehl	PC	DR	AR	OV	CY	NG	LZ	ZR	RZ	OD	DC	E	BKP1	BKP2	PTR1	PTR2	Register	Kommentar
8733	PUMD R02, +R06	8735	02	06								0							Speichern der Register R02-07 im Stack. R06 gibt dabei den Stackpointer an, und wird nach jedem Byte um 1 erhöht.
8735	JSB $466D	466D	02	06								0							Aufruf der Subroutine 'SVSJSB'
8738	DRP R30	8739	30									0							R30 wird DR
8739	ARP R22	873A	30	22															R22 wird AR

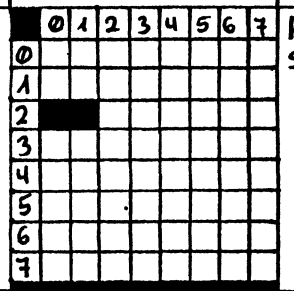
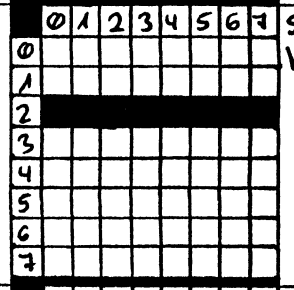
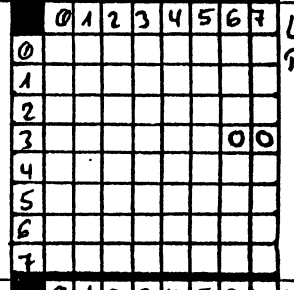
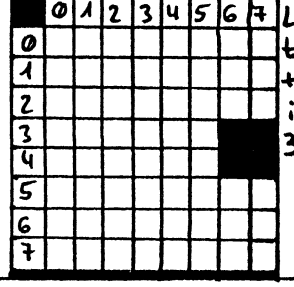
LCD

$	Befehl	PC	DR	AR	OV	CY	NG	LZ	ZR	RZ	OD	DC	E	BKP1	BKP2	PTR1	PTR2	Register	Kommentar
873A	LDB R54, =B4	873D	54	22			1				0	0							Laden der Hex-Zahl B4 in R 54
873D	POMD R02, -R06	8740	02	06								0							Laden der Bytes in R02-07 aus dem Stack. R06 wird dabei vor jedem Ladevorgang um 1 erniedrigt.
8740	STM R02, R55	8742	02	55								0							Speichern von R02-04 in die Register R55-57
8742	POBD R57, -R12	8745	57	12								0							Laden des Registers 57 aus dem Stack. Dabei wird R12 vor jedem Ladevorgang um 1 erniedrigt.

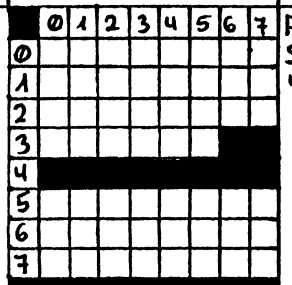
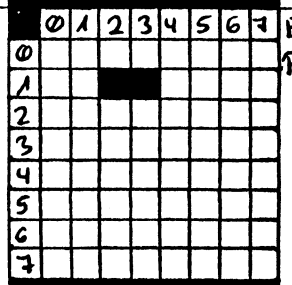
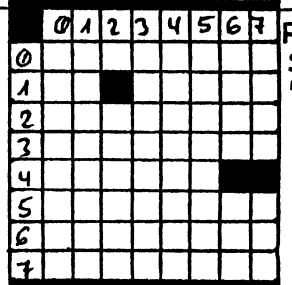
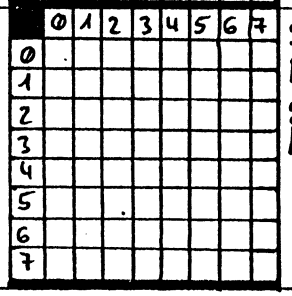
LCD

$	Befehl	PC	DR	AR	OV	CY	NG	LZ	ZR	RZ	OD	DC	E	BKP1	BKP2	PTR1	PTR2	Register	Kommentar
874E	PUMD R54, +R12	8747	54	12								0							Speichern der Register R54-57 in den Stack. Dabei wird R12 nach jeden Byte um 1 erhöht.
874F	RTN											0							Ende der RUNAGAIN-Routine
8748	Attribute	874A																	
8749	BIN	874B										0							Wahl des bin. Modus

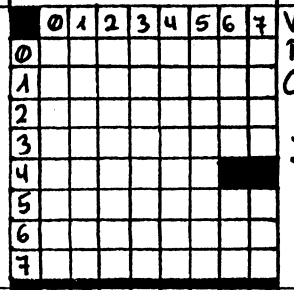
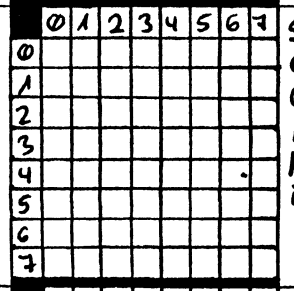
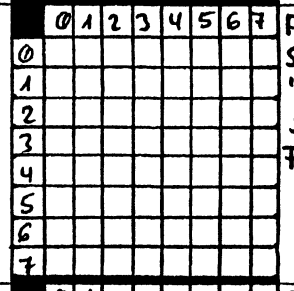
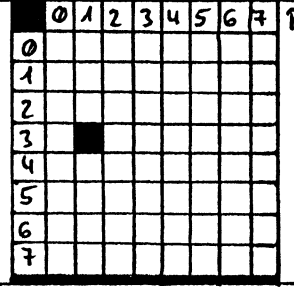
LCD

$	Befehl	PC	DR	AR	OV	CY	NG	LZ	ZR	RZ	OD	DC	E	BKP1	BKP2	PTR1	PTR2	Register	Kommentar
874B	SBM R20, $6120	8747	20									0							Herstellen einer Sprungadresse
8747	JSB XR20, $6133	*	20	20								0							Sprung in die Vektor-Adressen
8753	CLM R36	8755	36	20								0							Löschen von R36-R37
8755	LDBD R36, XR46, $82E0	8759	36	46								0							Laden des Inhaltes des Wertes + Adresse 82E0 in den Registern 36-37

LCD

$	Befehl	PC	DR	AR	OV	CY	NG	LZ	ZR	RZ	OD	DC	E	BKP1	BKP2	PTR1	PTR2	Register	Kommentar
8759	JSB $FCE4	FCE4	36	46															Aufruf der Subroutine 'PLINTG'
875C	RTN																		Ende der MAINZ Routine
875D	JSB $3E8B	3E8B																	Aufruf der Subroutine 'ONEB'
8760	JNG =06	8762			x	x	x												Sprung, wenn NG und OV gleiche Werte haben.

LCD

$	Befehl	PC	DR	AR	OV	CY	NG	LZ	ZR	RZ	OD	DC	E	BKP1	BKP2	PTR1	PTR2	Register	Kommentar
9762	CMM R46, $0067	8766	46									0							Vergleich der Register 46-47 (Inhalt bei WLCD) mit der Zahl 0067
9768	JNC =F4	8768	46			x						0							Sprung, wenn CY den Wert 0 hat (dh. das Multibyte 46-47 kleiner als 0067 ist).
9768	JSB $4C99	4C99											x						Aufruf der Subroutine 'ERROR+' zur Steuerung der Fehlermeldungen
9768	DRP R31	----	31																R31 wird DRP

```
0000 ID   4016
0002 DEF  RUNTIME
0004 DEF  NAMES
0006 DEF  PARSE
0008 DEF  ERRORS
000A DEF  RUN
000C LABEL RUNTIME
000C ZD   6105
000E DEF  MAIN1
0010 DEF  MAIN2
0012 LABEL PARSE
0012 ZD   6105
0014 DEF  RUNAGAIN
0016 ENDE POINTERS
0018 LABEL NAMES
0018 ASC  "WLCD"
001C ASC  "RLCD"
0020 ENDE NAMES
0021 LABEL ERRORS
0021 ENDE ERRORS
0022 DRP  R01
0023 ARP  R27
0024 LABEL RUN
0024 RTN
0025 BYT  A1
0026 LABEL MAIN1
0026 BIN
0027 DRP  R20
0028 SBM  R20,=F1,60
002B JSB  $3E8B
002E DRP  R46
002F ARP  R26
0030 STM  R46,R26
0031 ARP  R20
0032 JSB  X20,$6133
0035 DRP  R26
0036 ARP  R46
0037 STBD R26,R46,$82E0
003A RTN
003B LABEL RUNAGAIN
003B DRP  R02
003C ARP  R41
003D LDM  R02,R41
003E ARP  R06
003F PUMD R02,+R06
0040 JSB  $466D
0043 BYT  58
0044 BYT  12
0045 DRP  R54
0046 LDB  R54,=B4
0048 DRP  R02
0049 ARP  R06
004A POMD R02,-R06
004B ARP  R55
004C STM  R02,R55
004D DRP  R57
004E ARP  R12
004F POBD R57,-R12
0050 DRP  R54
0051 PUMD R54,-R12
0052 RTN
0053 BYT  10
0054 BYT  2D
0055 LABEL MAIN2
0055 BIN
0056 DRP  R20
```

```
0057 SBM  R20,=20,61
005A ARP  R20
005B JSB  X20,$6133
005E DRP  R36
005F CLM  R36
0060 ARP  R46
0061 LDBD R36,X46,$82E0
0064 JSB  $FCE4
0067 RTN
0068 JSB  $3E8B
006B JNG  =06
006D DRP  R46
006E CMM  R46,=66,00
0071 JNC  =F4
0073 JSB  $4C99
0076 BYT  59
0077 END
```

6.4.5. Bearbeiten des LEX-Files 'LCD' mit Hilfe des Assemblers:
Zunächst editiert man ein Text-File mit dem Namen 'LCD'.
Dafür führt man folgende Befehlsfolge aus:

```
EDIT LCD,TEXT
AUTO 10,10
```

Nun gibt man die einzelnen Zeilen ein:

```
ID4016
DEF RUNTIME
DEF NAMES
DEF PARSE
DEF ERRORS
DEF RUN
LABEL RUNTIME
ZD6105
DEF MAIN1
DEF MAIN2
LABEL PARSE
ZD6105
DEF RUNAGAIN
ENDE POINTERS
LABEL NAMES
ASC "WLCD"
ASC "RLCD"
ENDE NAMES
LABEL ERRORS
ENDE ERRORS
BYT 41
BYT 17
LABEL RUN
BYT A1
LABEL MAIN1
BIN
SBM R20,=F1,60
JSB $3E8B
STM R46,R26
JSB X20,$6133
STBD R26,X46,$82E0
RTN
LABEL RUNAGAIN
LDM R02,R41
PUMD R02,+R06
JSB $466D
DRP R30
ARP R22
LDB R54,=B4
POMD R02,-R06
STM R02,R55
POBD R57,-R12
PUMD R54,+R12
RTN
BYT 10
BYT 2D
LABEL MAIN2
BIN
SBM R20,=20,61
JSB X20,$6133
CLM R36
LDBD R36,X46,$82E0
JSB $FCE4
RTN
JSB $3E8B
JNG =06
CMM R46, =67,00
JNC =F4
```

```
JSB $4C99
DRP R31
END
```

Nun lädt man das Programm 'COMPILE' und kompiliert das Programm.

6.4.6. Beispiele zur Anwendung des neuen LEX-Files:

a) Beispiele zur Anwendung von 'RLCD':

```
10 ! ** LCD1 **
20 DIM A(96),A$[0]
30 ON ERROR OFF ERROR
40 DISP 'Text'
50 INPUT A$
60 FOR I=6 TO 100
70 A(I-6)=RLCD(I)
80 NEXT I
90 STOP
```

Gibt man nun einen Text ein, so ist dieser schon als numerischer Code in den Registern A(0)-A(96) gespeichert, Will man nun die Länge des eingegebenen Textes bestimmen, fügt man folgende Zeilen an:

```
90 FOR I=96 TO 0 STEP -1
100 IF A(I)#32 THEN 120
110 NEXT I
120 DISP 'Laenge: ';I
130 STOP
```

Der Vorteil bei dieser Speichermethode ist, daß weniger Speicherplatz benötigt wird, da die Dimensionierung bei numerischen Arrays weniger Platz benötigt als die Dimensionierung von ALPHA-Ketten.

b) Beispiele zur Anwendung von 'WLCD':

```
10 ! ** LCD2 **
20 RESTORE
30 FOR A=19 TO 24 @ READ J
50 WLCD A,J @ WLCD 1,0
70 NEXT A
80 STOP
100 DATA 72,80,45,55,53,67
```

So kann man leicht ohne DISP-Befehl den Ausdruck 'HP-75C' in die Mitte der Anzeige bringen.
Nachteil ist noch, daß nur dann eine Anzeige erfolgt, wenn das Programm steht.
Eine Laufschrift sähe dann so aus:
Nach jdem Schritt muß man (RTN) drücken.

```
10 ! ** LCD3 **
20 ON ERROR OFF ERROR @ RESTORE
25 DIM A$[0]
30 FOR A=19 TO 24 @ READ J
40 FOR I=35 TO A STEP -1
50 WLCD I,J @ WLCD I+1,32 @ WLCD 1,0 @ INPUT A$
60 NEXT I
70 NEXT A
80 STOP
100 DATA 72,80,45,55,53,67
```

3. Kapitel

Der Assembler

1. Definition:
   Ein Assembler ist ein Programm zur Umwandlung von Programmen, die in Assemblersprache geschrieben sind, in Maschinensprache.
   Eine Assemblersprache enthält drei Arten von Anweisungen: Assemblerbefehle, Assemblerinstruktionen und Makrobefehle.

1.1. Assemblerbefehle:
   Assemblerbefehle sind Maschinenbefehle, deren Operations- und Adressenteil nicht numerisch geschrieben werden dürfen. Anstelle des Maschinencodes für den operationsteil eines Befehls darf ein Symbol geschrieben werden. Um das Programmieren zu erleichtern, verwendet man in Assemblerbefehlen mnemonische Bezeichnungen für den Operationsteil, dh. Zeichen, die eine Gedächtnisstütze bezüglich der Wirkung der damit bezeichneten Operation darstellen (zB. ADB für die Operation "Addiere Bytes"). Für den Adreßteil darf der Programmierer ein von ihm gewünschtes Symbol bzw. eine alphanumerische Bezeichnung setzen, die entweder eine Variable (einen Operanden, Register) oder eine Marke (ein Label, Sprungziel) repräsentiert. Befehle mit Symbolen anstelle von Operationsteil und Adressenteil heißen symbolische Befehle oder Pseudobefehle, ihr Adreßteil symbolische Adresse. Eine Assemblersprache ist eine maschinenorientierte Programmiersprache, da die Assemblerbefehle in ihrem Aufbau den Maschinenbefehlen gleich oder sehr ähnlich sind. Daher sind Assemblersprachen im Gegensatz zu problemorientierten Programmiersprachen von Anlage zu Anlage verschieden.

1.2. Assemblerinstruktionen:
   Assemblerinstruktionen geben organisatorische Anweisungen, die das Festlegen von Konstanten und Speicherbereichen, das Zuweisen von Speicheradressen, die Steuerung von Ein- und Ausgabe, die Steuerung der Übersetzungsarbeit und das Unterteilen und Verknüpfen von Programmen betreffen.

1.3. Makrobefehle:
   Makrobefehle dienen zum Einfügen von Unterprogrammen, die durch Parameterangaben modifiziert werden können. Die Umwandlung des Assemblerprogrammes in ein Maschinenprogramm, das Assemblieren, umfaßte ursprünglich sowohl die Übersetzung in die Maschinensprache als auch das Zusammenfügen von Teilen eines Programmes, einschließlich des Einfügens von Bibliotheksprogrammen und der Herstellung der notwendigen Programmverbindungen. Anlagen mit einem hochorganisierten Betriebssystem führen das Verknüpfen von assemblierten Programmteilen mit Hilfe eines Programmverbinders gesondert durch und reduzieren das Assemblieren im wesentlichen auf das Übersetzen. Die Assemblersprachen haben gegenüber Maschinensprachen den Vorteil, daß die Programmierung durch die Verwendung von symbolischen Adressen und Makroaufrufen erleichtert ist. Außerdem lassen sich die Programme durch Ausnutzung der Ähnlichkeit zur Maschinensprache sowohl speicherplatz- als auch laufzeitoptimal gestalten. Gegenüber höheren Programmiersprachen treten als Nachteile die größere Anzahl der zu programmierenden Anweisungen auf (folglich längere Programmierzeiten) sowie die höhere Anzahl und schwierige Ortung von Programmierfehlern sowohl syntaktischer als auch logischer Natur (folglich höherer Testzeitbedarf zur Beseitigung der Fehler).

2. Vorgang des Assemblierens:
   Nachdem man sich nun mit Hilfe der Formblätter ein Assemblerprogramm zusammengestellt hat, gibt man die einzelnen Zeilen in ein Text-File ein mit Hilfe des im HP-75 eingebauten Text-Editors.
   Dieses Text-File wird dann vom Compiler 'COMPILE' interpretiert und danach in ein LEX-File umgewandelt.

3. Mögliche Assemblerbefehle:

3.1. Vorbemerkungen:

Die Eingabe der Assemblerbefehle ist variabel.
Wieviele Balnks miteingegeben werden, ist unwichtig, da vor Bearbeitung der Rechner selbst alle Blanks löscht und die richtige Syntax herstellt.

4.2. Steuerbefehle:

a) ID (Identifier)
   Syntax: ID 4-Byte-Adresse
   Bsp: ID 4016

b) ZD (Zwischenbyte)
   Syntax: ZD 4-Byte-Adresse
   Bsp: ZD6005

c) LABEL:
   Syntax: LABEL Labelname
   Bsp: LABEL RUNTIME

d) DEF (Labeldefinition)
   Syntax: DEF Labelname
   Bsp: DEF RUNTIME

e) ENDE
   Ein Programmteil wird beendet
   3 mögliche Programmteile: POINTERS, NAMES, ERRORS
   Syntax: ENDE Programmteil
   Bsp: ENDE NAMES

f) END
   Programmende
   Syntax: END
   Bsp: END

g) ASC (ASCII-Commands)
   Mit diesem Befehl wird ein Befehlsmnemonic definiert.
   Syntax: ASC "Befehlsmnemonic"
   Bsp: ASC "PEEK"

h) ASE (ASCII-Errors)
   Mit diesem Befehl wird eine Errormeldung definiert.
   Syntax: ASE "Fehlermeldung"
   Bsp: ASE "device sent NRD"

i) Kommentare (wie bei BASIC)
   Syntax: ! Kommentar
   Bsp: ! Programmkopf

4.3. 1-Byte-Befehle:

Zu dieser Befehlsgruppe gehören die Befehle:

a) BIN
   BCD
   SAD
   DCE
   ICE
   CLE
   RTN
   PAD
   Sie werden eingegeben, wie sie geschrieben werden.
   Bsp: BCD

b) BYT + 2 Bytes
   Bsp: BYT F8
   DRP R Register
   Bsp: DRP R40
   ARP R Register
   Bsp: ARP R40

3.4. 2-Byte-Befehle:

a) Registerbefehle:

Zu dieser Befehlsgruppe gehören die Befehle:

ELB DR
ELM DR
ERB DR
ERM DR
LLB DR
LLM DR
LRB DR
LRM DR
ICB DR
ICM DR
DCB DR
DCM DR
TCB DR
TCM DR
NCB DR
NCM DR
TSB DR
TSM DR
CLB DR
CLM DR

Bei diesen Befehlen wird DR mit R Register eingegeben.

Bsp: Löschen des Registers 40

```
CLB R40
```

b) Befehle mit Literals:

Zu dieser Befehlsgruppe gehören die Befehle:

JMP =literal
JNO =literal
JOD =literal
JEV =literal
JNG =literal
JPS =literal
JNZ =literal
JZR =literal
JEN =literal
JEZ =literal
JNC =literal
JCY =literal
JLZ =literal
JLN =literal
JRZ =literal
JRN =literal

Der eingegebene literal-Wert besteht aus einem Byte, das eine ganze Zahl und damit die Sprunglänge definiert (in Hex-Form)

Bsp:

```
JMP =80
```

3.5. 3-Byte-Befehle:

a) Befehle mit 2 Registern:

Zu dieser Befehlsgruppe gehören die Befehle:

ORB DR,AR
ORM DR,AR
XRB DR,AR
XRM DR,AR
LDB DR,AR
LDM DR,AR
STB DR,AR
STM DR,AR
LDBD DR,AR
LDMD DR,AR
STBD DR,AR
STMD DR,AR
LDBI DR,AR
LDMI DR,AR

STBI DR,AR
STMI DR,AR
CMB DR,AR
CMM DR,AR
ADB DR,AR
ADM DR,AR
SBB DR,AR
SBM DR,AR
ANM DR,AR
CMBD DR,AR
CMMD DR,AR
ADBD DR,AR
ADMD DR,AR
SBBD DR,AR
SBMD DR,AR
ANMD DR,AR
Bei diesen Befehlen wird der Registerteil mit R DR,R AE eingegeben.
Bsp: LDBD R40,R30

b) Befehle mit einem Register und einem literal:
Zu dieser Befehlsgruppe gehören die Befehle:
LDB DR,=literal
LDM DR,=literal
STB DR,=literal
STM DR,=literal
CMB DR,=literal
CMM DR,=literal
ADB DR,=literal
ADM DR,=literal
SBB DR,=literal
SBM DR,=literal
Bsp: LDM R45,=80,7F,7E

c) Schiebebefehle:
Zu dieser Befehlsgruppe gehören die Befehle:
POBD DR,+AR
POMD DR,+AR
POBD DR,-AR
POMD DR,-AR
PUBD DR,+AR
PUMD DR,+AR
PUBD DR,-AR
PUMD DR,-AR
POBI DR,+AR
POMI DR,+AR
POBI DR,-AR
POMI DR,-AR
PUBI DR,+AR
PUMI DR,+AR
PUBI DR,-AR
PUMI DR,-AR
Die Codierung dieser Befehle ist wie bei den Befehlen mit 2 Registern, nur muß vor das 2.Register noch ein Vorzeichen gesetzt werden.
Bsp: POBD R40,+R02

d) Befehle mit Labels:
Zu dieser Befehlsgruppe gehört nur ein Befehl:
JSB $Adresse
oder JSB 'Unterprogrammname'
Bsp: JSB $FF10
JSB 'ROMJSB'

3.6. 4-Byte-Befehle:

a) Befehle mit einem Register und einem Label:
Zu dieser Befehlsgruppe gehören die Befehle:
LDBD DR,$label
LDMD DR,$label

STBD DR,$label
STMD DR,$label
LDBI DR,$label
LDMI DR,$label
STBI DR,$label
STMI DR,$label
CMBD DR,$label
CMMD DR,$label
ADBD DR,$label
ADMD DR,$label
SBBD DR,$label
SBMD DR,$label
ANMD DR,$label

Anstelle von $labeladresse kann auch 'Unterprogrammname' stehen.

Bsp: LDBD R40, $FF10
LDBD R40,'ROMJSB'

b) Der Befehl JSB XR,$label:
Die Handhabung dieses Befehl ist wie unter a) beschrieben.

Bsp: JSB XR20,$FF10
JSB XR20,'ROMJSB'

3.7. 5-Byte-Befehle:
Zu dieser Befehlsgruppe gehören die Befehle:
LDBD DR,XR,$label
LDMD DR,XR,$label
STBD DR,XR,$label
STMD DR,XR,$label
LDBI DR,XR,$label
LDMI DR,XR,$label
STBI DR,XR,$label
STMI DR,XR,$label

Bsp: LDBD R20,XR30,$FF10
LDBD R20,XR30,'ROMJSB'

3.8. Zusammenfassung:
Diese Assemblerbefehle werden mit der unter 3.2.-3.7. beschriebenen Schreibweise eingegeben.

4. Das Programm 'COMPILE'

4.1. Vorgänge beim Starten des Programmes:
Nach Ausführung von RUN 'COMPILE' erscheint in der Anzeige die Aufforderung 'Filename:', worauf man den Namen des Textfiles eingibt, das man vorher mit dem eingebauten Text-Editor des Rechners editiert hat, und das man nun in ein LEX-File verwandeln will.

4.2. Programmablauf:
Nun erscheint nach zwei Kataloganzeigen die Anzeige 'Compilation . . . .', die nach Beendigung des Compilierens von der Anzeige 'READY . . . .' abgelöst wird. Wurde ein Fehler gemacht, so stürzt der Rechner nun meistens ab. Dieses Verhalten läßt sich leider nicht verhindern. Deshalb kopiert man das Text-File vorher am besten auf Kassette.
Liegt kein Fehler vor, so läßt sich nun das neue LEX-File benutzen.
Der genaue Ablauf des Programmes (=Programmteile) kann aus den Kommentaren im Listing ersehen werden.

5. Beispiel:
Eingabe des LEX-Files 'PEEKPOKE'
Als Grundlage für die Eingabe dient folgendes Listing:

```
0000 ID   4015
0002 DEF  RUNTIME
0004 DEF  NAMES
0006 DEF  PARSE
0008 DEF  ERRORS
000A DEF  RUN
000C LABEL RUNTIME
000C ZD   6105
000E DEF  MAIN1
0010 DEF  MAIN2
0012 LABEL PARSE
0012 ZD   6105
0014 DEF  RUNAGAIN
0016 ENDE POINTERS
0018 LABEL NAMES
0018 ASC  "POKE"
001C ASC  "PEEK"
0020 ENDE NAMES
0021 LABEL ERRORS
0021 ENDE ERRORS
0022 DRP  R01
0023 ARP  R27
0024 LABEL RUN
0024 RTN
0025 BYT  A1
0026 LABEL MAIN1
0026 BIN
0027 DRP  R20
0028 SBM  R20,=F1,60
002B JSB  $3E8B
002E DRP  R46
002F ARP  R26
0030 STM  R46,R26
0031 ARP  R20
0032 JSB  X20,$6133
0035 DRP  R26
0036 ARP  R46
0037 STBD R26,R46,$0000
003A RTN
003B LABEL RUNAGAIN
003B DRP  R02
003C ARP  R41
003D LDM  R02,R41
003E ARP  R06
003F PUMD R02,+R06
0040 JSB  $466D
0043 BYT  58
0044 BYT  12
0045 DRP  R54
0046 LDB  R54,=B4
0048 DRP  R02
0049 ARP  R06
004A POMD R02,-R06
004B ARP  R55
004C STM  R02,R55
004D DRP  R57
004E ARP  R12
004F POBD R57,-R12
0050 DRP  R54
0051 PUMD R54,-R12
0052 RTN
0053 BYT  10
0054 BYT  2D
0055 LABEL MAIN2
0055 BIN
0056 DRP  R20
```

```
0057 SBM  R20,=20,61
005A ARP  R20
005B JSB  X20,$6133
005E DRP  R36
005F CLM  R36
0060 ARP  R46
0061 LDBD R36,X46,$0000
0064 JSB  $FCE4
0067 RTN
0068 JSB  $3E8B
006B JNG  =06
006D DRP  R46
006E CMM  R46,=FF,FF
0071 JNC  =F4
0073 JSB  $4C99
0076 BYT  59
0077 END
```

Zur Eingabe dieses Text-Files geht man wie folgt vor:

1. EDIT 'PEEKPOKE',TEXT
2. Es erscheint in der 0.Spalte der : zur Kennzeichnung der Text-File-Umgebung. Zur Erleichterung der Eingabe gibt man folgenden Befehl ein AUTO 10,10
3. Im folgenden möchte ich auf die Korrektur von Eingabefehlern nicht eingehen, da die Text-Editor-Commands im Handbuch des HP-75 zur Genüge erläutert sind.

```
 10 ID4015
 20 DEF RUNTIME
 30 DEF NAMES
 40 DEF PARSE
 50 DEF ERRORS
 60 DEF RUN
 70 LABEL RUNTIME
 80 ZD6105
 90 DEF MAIN1
100 DEF MAIN2
110 LABEL PARSE
120 ZD6105
130 DEF RUNAGAIN
140 ENDE POINTERS
150 LABEL NAMES
160 ASC "POKE"
170 ASC "PEEK"
180 ENDE NAMES
190 LABEL ERRORS
200 ENDE ERRORS
210 DRP R01
220 ARP R27
230 LABEL RUN
240 RTN
250 BYT A1
260 LABEL MAIN1
270 BIN
280 DRP R20
290 SBM R20,=F1,60
300 JSB $3E8B
310 DRP R46
320 ARP R26
330 STM R46,R26
340 ARP R20
```

```
350 JSB X20,$6133
360 DRP R26
370 ARP R46
380 STBD R26,R46,$0000
390 RTN
400 LABEL RUNAGAIN
410 DRP R02
420 ARP R41
430 LDM R02,R41
440 ARP R06
450 PUMD R02,+R06
460 JSB $466D
470 BYT 58
480 BYT 12
490 DRP R54
500 LDB R54,=B4
510 DRP R02
520 ARP R06
530 POMD R02,-R06
540 ARP R55
550 STM R02,R55
560 DRP R57
570 ARP R12
580 POBD R57,-R12
590 DRP R54
600 PUMD R54,+R12
610 RTN
620 BYT 10
630 BYT 2D
640 LABEL MAIN2
650 BIN
660 DRP R20
670 SBM R20,=20,61
680 ARP R20
690 JSB X20,$6133
700 DRP R36
710 CLM R36
720 ARP R46
730 LDBD R36,X46,$0000
740 JSB $FCE4
750 RTN
760 JSB $3E8B
770 JNG =06
780 DRP R46
790 CMM R46,=FF,FF
800 JNC =F4
810 JSB $4C99
820 BYT 59
830 END
```

4. Nachdem man nun das Listing eingegeben hat, kopiert man das Text-File auf Kassette. Nun beginnt das Kompilieren. Dazu lädt man die Files 'COMPILE' und 'PEKEPOOK' von Kassette in den Rechner. Danach startet man das Programm 'COMPILE' mit RUN 'COMPILE'. In der Anzeige erscheint die Aufforderung 'Filename:'. Man gibt darauf PEEKPOKE, den Namen des zu kompilierenden Text-Files, ein. In der Anzeige erscheint nach dem RTN folgende Anzeigefolge: Compilation . . . .
   READY . . . .
   Das LEX-File ist nun geladen und kann mit seinen Funktionen verwendet werden.

7. Das Text-File 'ASSEM2':

7.1. Einführung:

In dieses Text-File können Einsprungpunkte mit ihren Adressen eingegeben werden. Befindet sich dieses File im Rechner, so genügt es, anstelle der Adresse die Kurzbezeichnung des Einsprungpunktes in Befehlen mit Adressen anzugeben.

Bsp: JSB $018A entspricht JSB 'CONBIN'

In dem auf der Kassette gespeicherten Text-File befindet sich eine Auswahl von 10 Einsprungpunkten. Die ganze Palette, die nun mit den Adressen unter 7.2. aufgelistet wird, sollte nur in Ausschnitten verwandt werden, da dies auf Kosten des Speicherplatzes geht. Ich empfehle maximal 20 Punkte zu verwenden. Die Funktion der Einsprungpunkte ist zur Erklärung in diesem Buch zu umfangreich. Als Nachschlagewerk empfehle ich dafür das Buch

75 NOMAS VOL I
606 Seiten (englisch)
Preis: PPC-Mitglieder $ 28.50
Nicht-Mitglieder $ 35.63
erhältlich bei:
PPC
P.O: BOX 9599
Fountain Valley, CA
92728-9599 USA

7.2. Liste der Einsprungpunkte:

Mit * markierte Adressen befinden sich im auf der Kassette abgespeicherten Text-File 'ASSEM2'. Ihre Funktion wird noch näher unter 7.3. erläutert.

A.ZERO	$EAC7	ABORT	$FBCD	ABS5	$3A43
ABS5-	$3A46	ABTHRD	$F8EA	ACMND	$35F2
ADD15	$380A	ACREAT	$7210	ADD20	$380F
ADD9	$3804	ADDR	$F042	ADDROI	$37D4
ADJUST	$233F	AINCHK	$67B0	ALARM.	$7430
ALBEEP	$7233	ALCALL	$1F93	ALFA	$0E02
ALFA#	$0E03	ALINI	$2E39	ALLOC	$1F9F
ALMCHK	$6CD7	ALOSNV	$27F9	ALVENT	$28E8
ALWAYS	$E9D1	AND.	$4E6A	ANN.A+	$440A
ANN.A-	$4415	ANN.B+	$43E4	ANN.B-	$43E9
ANN.E+	$43EE	ANN.E-	$43F3	ANN.P+	$4400
ANN.P-	$4405	ANNUNS	$43E0	ANOTE	$3607
ANYER?	$483A	AOPEN	$7220	AOPEN	$7224
APDEL'	$6401	APEXIT	$668A	APFILO	$09EF
APFILE	$09F7	APFND'	$644E	APINFO	$6361
APMSKE	$6537	APMSKY	$653C	APPROC	$72BC
APPT.	$4993	APPTMD	$65E4	APPTRS	$7229
APPTTM	$6552	APSTAT	$7300	APTACK	$7318
APTCHK	$6864	APTDEL	$63F2	APTDSP	$641B
APTERR	$642C	APTFND	$644A	APTGET	$646C
APTINS	$6482	APTMRG	$735D	APTR+	$64A4
APTR-	$64B2	APTRIG	$73B3	ARDON	$F3FD
ARITH	$F3A9	ARRAY	$105B	ASCIIS	$6069
ASCNAM	$2E20	ASCNM1	$2E29	ASINIT	$664C
ASNPAR	$49CF	ASPACK	$64D7	ASSIGN	$1303
ASSIN.	$E48F	ASTBCD	$742D	ATMPLT	$650D
ATN2.	$74F4	ATN2.+	$7501	ATSIGN	$E2ED
ATTN?	$084B	AVADR1	$5666	AVADR2	$566B
AVCALL	$2803	AVVAL1	$567E	AVVAL2	$5683

```
AWAKE!   $09C9     B.INIT   $064C     BACK10   $0E2B
BADDEV   $6ADB     BASEND   $209A     BASIC.   $5245
BASLOC   $E002     BATS     $358C     BCDINT   $2E77
BCKSTO   $43BD     BEEP     $FEB4     BEEP.    $FED1
BEEP.'   $7D6C     BEEPER   $FED7     BEEPR'   $7DBB
BININT   $2EDA     BLANKS   $09FB     BLEBUF   $4BCA
BLIMP    $4100     BLKFIL   $5FAE     BOUND    $0862
BRT10    $7549     BRT5     $7541     BRTS35   $069F
BRTS40   $06A1     BTAB.R   $E022     BYE.     $0889
C.INIT   $EAAB     CALCSZ   $2EFF     CALEN?   $540F
CALL     $50EE     CALL.    $514F     CALNAM   $0044
CARD.    $49BB     CAT$     $7E80     CAT$.    $593B
CAT.     $58DA     CATAL.   $590D     CATALL   $7EEB
CATBUF   $7F62     CATHED   $7F88     CATLIN   $58EB
CATTMP   $7F5E     CDPC?    $1B08     CEIL10   $37A8
CH2LCD   $5AFD     CHANGE   $5C02     CHECK    $087A
CHEDIT   $7AE4     CHKEND   $5E8D     CHKEOL   $42F4
CHR$.    $FC7A     CHSROI   $37B0     CKTRIG   $6772
CL.ACT   $EBFE     CLDEV.   $5E4F     CLIF     $7186
CLNCAL   $2C57     CLOOP.   $5E46     CLRCOD   $6A93
CLRENV   $509E     CLRERR   $4BC2     CLRINT   $7A05
CLRLCD   $EFA4     CLROFF   $5E16     CLRTMT   $5E38
CLRVA.   $003F     CMDREP   $EC1F     CMDSND   $EB15
CMPCHK   $052C     CMPDSA   $04F7     CMPENA   $04EA
CMPENT   $0501     CMPLD    $05BF     CMPSET   $05B5
CMPSRV   $040A     CMPSTR   $4DAA     CNAJST   $04B9
CNTRIG   $7448     COMA#.   $E7B1   * CONBIN   $018A
CONCA-   $FC3D     CONCA.   $FC32     CONFIG   $5BD7
* CONINT  $1D37     CONTI.   $51CC     COPY     $22E7
COPY.    $1A43     COS10    $39E5     COT10    $39D5
CRCOM?   $107F     CRDCPY   $F6CA     CRDEXM   $F696
CRDUPR   $F664     CRDWPR   $F66C     CREOL?   $455D
CRTFLE   $FA2C     CRTMDT   $7528     CRUNCH   $009D
CSEC10   $39BA     CSEC11   $39D1     CSTRIG   $676E
CURSE    $EFBF     CURSE+   $06BF     CURSE-   $06C4
CURSE?   $EFB9     CVNUM    $F35A     CYCLE    $4848
DALLAL   $2C93     DALLOC   $2CDC     DATA     $159C
DATCHK   $6D08     DATCK'   $6D27     DATE$.   $7C74
DATE.    $7CB3     DATREP   $EC24     DATRP+   $67B7
DDTSD-   $700A     DATSND   $7008     DAYCHK   $6DAD
DAYOK    $6E23     DAYSEC   $0A56     DAYTAB   $6E0E
DCCLOK   $6930     DCDAY    $6E3F     DCLIN#   $32A8
DCON1    $3E03     DDLREP   $6FFC     DDLRP+   $67B2
DDTREP   $7001     DECOM    $2F4D     DECOPR   $2F46
DECURS   $EFE8     DEF      $13E1     DEFKEY   $4E8E
DEFKY.   $4ECB     DEG.     $79FB     DEG10    $3ACA
DEGCNV   $3AD8     DEL      $7D41     DELET.   $1C01
DELETE   $1FB3     DELLIN   $21F3     DELLNS   $222D
DEQUE    $07A3     DEVOK    $7830     DFNAME   $5260
DIGIT    $0E1C     DIM      $1358     DISP.    $E38A
DIV10    $376E     DIV14    $3771     DIV2     $376B
DIV20    $3786     DIV77    $3744     DMNDCR   $1365
DNENT    $5919     DOCMD    $0121     DOSEE    $F01C
DOSTEP   $516A     DSPIS.   $5BAE     DSPSET   $45BF
DTAB-1   $0A4B     DTABO    $0A4C     DTECNV   $7547
DUPCHK   $6E7B     EDIT     $49CA     EDIT.    $525A
EDITFI   $74AF     EDITIT   $5293     EDLIN#   $4482
EDLIN-   $4486     EDPURG   $1986     EFORM    $597A
EJMP#    $EC55     ENCLOK   $6A2A     END.     $5204
END.ON   $E30E     ENDEN?   $5038     ENDIT    $00CC
ENTSIZ   $2749     ENVALO   $2AAB     ENVERR   $2BE7
EOL      $0C7C     EOL.     $EA90     EOL14?   $455C
EOLND    $0835     EPS10    $3ABE     EQ$.     $4DDF
EQ.      $4DEF     ER16+    $7472     ER43     $4340
ERMSG    $787D     ERNUM.   $FD46     EROUT-   $4177
ERR.R2   $4C45     ERR1     $4CA4     ERR1+    $4CAC
```

	ERR89	$1089		ERRL.	$FCAB		ERRMSG	$4D70
*	ERROR	$4C91	*	ERROR+	$4C99		ERRORR	$4C8C
	ERRORX	$E2A0		ERRREP	$4B41		EVIL	$5EF0
	EXEC	$008A		EXNXT	$EE1D		EXOR.	$4E56
	EXP17	$3D68		EXP20	$3D53		EXP5	$386B
	EXTRJM	$3576		FCOPY	$2104		FCOPYG	$2114
	FCRALO	$2011		FCREAT	$206D		FCRNUL	$2078
	FDELLN	$221F		FEMPT?	$20A5		FETAV	$1DAD
	FETAVA	$1E43		FETCH.	$4459		FETCHA	$1DC6
	FETIND	$1DB8		FETK.	$4F3B		FETNUM	$1DB5
	FETRVA	$1DD0		FETSET	$1DC3		FETST	$1EDF
	FETSV	$1DB2		FETSVA	$1DA6		FILDRV	$6016
	FILHAN	$19B9		FILNM!	$4A3C		FILNM+	$4A22
	FILNM?	$4A29		FINDTD	$6AF8		FINMSG	$FBA5
	FIX65	$6B6D		FLCAF+	$6814		FLCAT	$60DF
	FLCHR?	$4B31		FLCOPY	$622F		FLDCHK	$6EC2
	FLFIN+	$67E8		FLFIND	$67EB		FLFTOF	$63FB
	FLGOF+	$67E3		FLGOFE	$6809		FLGER1	$6819
	FLGTFN	$6823		FLINIT	$1F1B		FLNEW	$68A8
	FLODTH	$19BF		FLOPEN	$2212		FLORD	$4A45
	FLORDT	$4A47		FLPUR!	$6392		FLPUR	$6225
	FLPURG	$620C		FLR36	$63EF		FLRENA	$64B3
	FLSAM?	$6A09		FLSBOW	$6886		FLSTAK	$683A
	FLSWCH	$63EC		FLTOFL	$49FE		FLOTTOT	$6385
	FLVFO?	$685F		FNCAL$	$56CC		FNCAL.	$56CC
	FNDLED	$214C		FNDLIN	$214C		FNDLPR	$2140
	FNDLRN	$2146		FNDRTN	$ED0F		FNDTD	$6B2B
	FNDVFA	$2A9C		FNEND	$146F		FNLET	$1696
	FNLET.	$5839		FNRET.	$5891		FNRTN.	$585B
	FNUM	$1DF8		FOPAC?	$247F		FOPEN	$24A0
	FOPEN!	$2492		FOPENG	$24A4		FOR	$1760
	FOR.	$ED64		FORMAR	$161B		FORMN	$F36B
	FORMN+	$F363		FP5	$3AA1		FPURGE	$1FD6
	FRENAM	$20C4		FREPLN	$2259		FREPLS	$224C
	FRET	$26A1		FRMNA#	$2EE3		FRMNAM	$2EE0
	FSEEK	$2154		FSREPL	$21A4		FTADR	$56AC
	FTR12	$3B07		FTR13	$3B0D		FTR53	$3BE4
	FTR88	$3C63		FTSTL	$54B1		FTSTLS	$54BA
	FTSVL	$549E		FUN1	$0FC1		FXALARM	$6ED8
	FXAPPT	$6C49		FXDATE	$6EF3		FXDAY	$6F27
	FXTIMW	$6F37		FXYEAR	$6F6F		G$!012	$12D0
	G$012N	$12C6		G01$	$1250		G01$/*	$5B87
	G012N	$12DC		G01N	$12D7		G01N$	$12B9
	GOTO4N	$12E1		G1$OR*	$5B7F		G1OR2N	$12F3
	GCHAR	$0DEB		GCHAR#	$0DED		GEQ$.	$4DCD
	GEQ.	$4E1C		GET#1	$5D94		GET#2+	$5D9A
	GET.IN	$416D		GET1$	$128E		GET1N	$1261
	GET2N	$1258		GETAD#	$1F12		GETAD+	$1F14
	GETADR	$1F11		GETALN	$00E6		GETCHR	$0750
	GETCLK	$745B		GETCM?	$106B		GETCMA	$1066
	GETFC	$EDCC		GETFST	$4AF8		GETINT	$2E6F
	GETLIN	$15DD		GETLN	$06C9		GETLN+	$06E9
	GETLNX	$64C7		GETNAM	$4A77		GETNM	$3133
	GETNXT	$2E85		GETPW?	$125D		GETPAD	$5D70
	GETPAR	$1264		GETREP	$4158		GETRID	$24F8
	GETRM2	$475E		GETROM	$4761		GETRUN	$18AA
	GETSTP	$1296		GETTD	$746C		GETTEM	$4110
	GETTER	$412E		GF2LN#	$1BBB		GOCARD	$F868
	GOPEN	$24CC		GOPEN!	$2497		GORTN	$568E
	GOTOPR	$17BB		GOTOSU	$17BE		GR$.	$4DE6
	GR.	$4E25		GRANGE	$1BD8		GTNXT1	$2EA5
	GXX3	$1289		HALT?	$1889		HANDI	$5419
	HANDIO	$544C		HANDIR	$541C		HANG	$016A
	HLFLIN	$07F1		HLFOUT	$07D6		I#PUSH	$57DA
	I/ROFF	$7CE6		ICONST	$54CE		ICOS.	$7535
	IDYSND	$EB0D		IF	$16D5		ILET	$1644

INENV?	$53EE	INF10	$39CB	INFR2	$3A7C
INFR3	$3A4E	INFR4	$3A51	INFR9	$3A90
INIPR-	$0B06	INIPRS	$0B02	INISIZ	$675D
INIT.	$666B	INITGL	$00F5	INPU$.	$4373
INPUN.	$4309	INPUT	$1557	INPUT.	$4276
INSERT	$1FFC	INSRTM	$23CE	INT5	$3A68
INTCON	$57D7	INTDIV	$3A6F	INTEGR	$0D16
INTMUL	$394E	INTORL	$3F16	INTRP	$00BE
INVPOP	$ED23	INVRTN	$ED04	INWIND	$EFFB
IO.OFF	$F015	IO.ON	$F00C	IO?	$E99F
IOFILE	$09E7	IOOPEN	$218E	IP5	$3AE8
ISCMA!	$1075	ISETUP	$43D5	ISIN.	$752D
ITAIL.	$43DF	ITAN.	$753D	ITHEN	$16DF
IZOPEN	$2199	JMPLN#	$EC54	JMPSUB	$EC44
JSBCRT	$702D	JTRUE#	$EC35	KEY$.	$FC60
KEY?	$0854	KEYDSP	$419A	KEYNAM	$4EAD
KEYS.	$499F	KEYSRV	$0277	KEYTRM	$4195
KOPY	$0A5F	KYADDR	$E457	KYOPEN	$4EA7
LA	$7DDF	LADREP	$EC18	LBL4	$7260
LCDOUT	$59BB	LCDRDY	$F027	LCDRST	$F037
LEAPYR	$6F94	LEN.	$FCBE	LENCAL	$E939
LEQ$.	$4DC4	LEQ.	$4E0B	LET	$1649
LETGO	$079A	LIF1.	$524D	LINEDR	$44EF
LINEND	$FE56	LINHED	$0C15	LINLEN	$21E5
LIST.	$1B48	LIT?	$06B3	LN20	$362A
LN30	$3668	LN38	$3654	LN5	$3732
LN77	$3676	LOAD	$F7CC	LOCK.	$79B6
LOCK?	$79CB	LOGCOM	$399A	LOGT5	$379A
LOOKUP	$0A26	LPOFF	$5E20	LPON	$5E28
LSTIO.	$5DAB	LT$.	$4DD6	LT.	$4E3C
MAKEIT	$06A8	MALLOC	$74DC	MARGN.	$7F67
MAX10	$3D34	MAXDAT	$6AFB	MBLIM-	$7446
MBLIMP	$7440	MELJSB	$4661	MEM.	$FCC6
MERGE.	$5339	MIN10	$3D25	MINDD	$6FA8
MINHH	$6FAE	MINMM	$6FA8	MINMN	$6FB0
MINYY	$6FA6	MKEPRV	$F6F0	MOD10.	$7705
MODE?	$E4AB	MODEKY	$E472	MOVE	$2270
MOVEIT	$5EA6	MOVENV	$2C16	MOVEUP	$2218
MPY28	$38E2	MPY30	$38E5	MPYR'	$7F9A
MPYROI	$393D	MRGIN.	$FC01	MSGOUT	$07E3
MULT60	$7473	MVBYTS	$3F43	MVWRD	$30A0
MYGETN	$7418	MYSET2	$74E8	MYSTPR	$7412
NAME.	$5253	NEWLIN	$EC55	NEXT	$1785
NEXT.	$EDD7	NFR	$3AAA	NOCLC!	$17FC
NOT.	$4E61	NOTRSH	$71ED	NULERT	$0A03
NULOLD	$4108	NUM.	$FCD5	NUMBER	$0D40
NUMCHK	$0A69	NUMOUT	$F9B5	NUMREP	$10B6
NUMVA+	$0E6D	* NUMVAL	$0E70	NUNPCK	$6FB5
NXTAPT	$747D	NXTKWF	$4783	NXTLIN	$2EB2
NXTROM	$47AC	OFALARM	$7444	OFFIO+	$5E11
OFFIO.	$5E06	OFFTMR	$3F50	OFFUNL	$5E09
OFTOK.	$5FFC	ON	$1796	ON.	$ECCD
ON/OFF	$5FD7	ON?I	$E2B3	ON?R	$E2B7
ONE7+B	$7DEE	ONE7PR	$E375	* ONEB	$3E8B
* ONEI	$3EAC	* ONER	$3ECD	ONER-	$3ED0
ONERI	$3EE1	ONEROI	$3EDE	ONERRO	$1340
ONOFF2	$5FDD	ONR12?	$1F3A	ONTMR	$3F57
ONTOK.	$5FF6	OPTION	$131F	OR.	$4E4B
OTHER	$014D	OUT1CH	$081D	OUTC40	$0814
OUTCHR	$080C	OUTEOL	$082C	OUTERB	$494A
OUTESC	$0819	OUTLIN	$022B	OUTNUM	$5F6C
OUTSTR	$07E4	P#ARAY	$E87E	P1ANC!	$122C
PACK.	$64E6	PAR1	$09D7	PARSE!	$0BAA
PARSER	$0B0E	PARSIT	$0BA7	PCR.	$7465
PFNDPR	$2164	PGMANN	$43FB	PI10	$39F6
PILOF!	$EB03	PILON!	$EAE8	PILREP	$45E5

```
PLIST.    $1B41
POS.      $FD4F
PR#END    $E737
PREFND    $216A
PRINT     $18E5
PRNOTE    $7492
PRSE!!    $1343
PSETUP    $E9CB
PU36SC    $1248
PURGE.    $197E
PUSH32    $1240
PUTADR    $1F07
PUTCHR    $7451
PUTWIN    $5AD6
QMARK     $34AA
RA        $7E0D
RDTRY     $F7F3
READ      $15F3
READ.     $E593
RECON!    $7483
RELJMP    $26A1
REM10.    76FF'
RENAM.    $19A9
REPLAC    $23EB
REPORT    $4B4A
REPRT-    $4D1D
RESFIL    $5FA6
REST.     $5DD9
RESTN.    $E586
RESUL.    $FCED
REVBYT    $7033
RMERR.    $471C
RNDIZ     $775E
ROM:GO    $4721
ROMCPY    $277D
ROMSET    $47ED
ROOM!     $1F8A
ROUND     $5941
RPTPRS    $7105
RSMEM-    $241F
RSTREG    $0E51
RUNIT     $50E7
S.OFF     $EAC5
SALT      $180E
SAVENV    $4FC4
SBSCR     $1087
SCANE1    $1232
SCNST     $304F
SEC10     $39AA
SEP15     $3CDF
SEQNO     $17D8
SETCL#    $522A
SETI/R    $7CE8
SETLIN    $0704
SETPSB    $09CD
SETUP$    $4D8A
SGN5      $397C
SHFET     $7F6F
SHRONF    $3AA4
SIN10     $39DD
SKIP#     $35E2
SKIPT     $FEDE
SKIPR     $FEE4
SKPCON    $2ECA
SKPLOP    $EE95
SNDFRM    $EB1B
POP.      $ED1E
POSRUN    $FD57
PR#VAL    $E7B1
PRIN#.    $E54E
PRINT.    $E398
PRNTCH    $4575
PRSEM.    $E763
PTALO?    $2207
PU=       $33D0
PUSH1A    $122F
PUSH45    $1237
PUTBIN    $FE89
PUTKEY    $07C1
PWROK?    $062B
QUIP?     $1807
RAD.      $79FF
RDYSD+    $67BC
READ#     $15E5
READCR    $F90E
RECOVR    $5005
RELMEM    $2453
REMDAC    $51F1
RENUM.    $1C2E
REPLIN    $21A7
REPRP#    $0262
REPTTM    $65BE
RESPUT    $FC25
RESTO.    $E580
RESTOR    $15D5
RETRN.    $ED0A
REVPSH    $703D
RND10.    $77F4
RNDIZ.    $3CB2
ROMCL-    $257B
*ROMJSB   $4694
ROMVPA    $2787
ROOM?     $1F6D
RPTADJ    $6FE8
RSETEN    $1F2D
RSMEM=    $2420
RTOIN     $1CE4
RUNKEE    $50B9
S.ON      $EAD1
SAVBUF    $64E2
SAVREG    $0E33
SCAN      $0CA3
SCANLLX   $F193
SCONST    $54D5
SEMI#.    $E7B1
SEP20     $3CC8
SET       $FEA0
SETCOM    $4D96
SETIT     $5E30
SETOFF    $5E11
SETR12    $00E1
SETWDB    $7E55
SHELD?    $7ECF
SHLA      $7F48
SIGNF-    $24FE
SISFUN    $3425
SKIPD     $FEEA
SKIPIT    $FEED
SKIPS     $FEE1
SKPLN     $2185
SKPVR     $27C2
SPY       $4617
POPENV    $5040
PQ010     $3D16
PR.EOL    $4596
PRINS.    $5BB7
PRNFMT    $FDE5
PRNUM.    $5F66
PRSTR.    $5F1F
PTRALO    $2583
PUINTG    $FCE4
PUSH1F    $4A1B
PUT.      $07B0
PUTCAL    $2C14
PUTREL    $1F06
PWRSRV    $05EE
R#ARAY    $E8D9
RAD10.    $3A28
RDYSND    $EB18
READ#.    $E55A
READN.    $E6C9
REFNUM    $16BC
REM       $11AE
REMGSB    $ED54
REP.1     $4D07
REPORP    $0261
REPRT+    $4B54
RESCON    $2418
REST#.    $E542
RESTEN    $4FF6
RESTRT    $43C6
RETURN    $09E6
RINCHK    $67F7
RNDINI    $3CA9
RNLST?    $491F
ROMCLA    $2574
ROMRTN    $481D
RONF      $3AA7
ROPEN     $24D1
RPTINP    $70F2
RSETUP    $E9C2
RSTBUF    $64D9
RUN.      $50AC
RUNROM    $4754
SAFE!     $539D
SAVEME    $4741
SAVRG+    $0E99
SCAN+     $0CA0
SCANN     $F048
SEARCH    $0A05
SEP10     $3CE3
SEQ#      $17E7
SETCAL    $0067
SETED     $5308
SETL30    $745B
SETPR     $1F5C
SETRN     $1F47
SFSCAN    $0C9A
SHF10     $3789
SHRA      $7F5A
SIGNIF    $24FB
SKIP!     $FEE4
SKIPEM    $FEE7
SKIPL     $2DD4
SKPCHK    $5E8A
SKPLN#    $2187
SMLINT    $10A3
SQR30     $38A5
```

SQR5	$3890	STALL	$51C3	STALL+	$51C8
STALLS	$51BA	STALRM	$7400	STAND+	$EADF
STAND-	$EAC2	STAND.	$EABF	STAR?	$5BC3
START	$7A20	START?	$740E	STBEEP	$FECA
STBEP'	$7D54	STDATE	$6811	STEP.	$EE56
STMMSK	$6545	STOP.	$51B9	STOP?	$0626
STORE	$F708	STOST	$54E5	STOSTM	$54E5
STOSV	$5598	STOSVM	$5597	STR1CH	$FC7D
STR?*?	$5B99	STRCN+	$1193	STREX!	$1299
STREX+	$10FA	STREXP	$10FD	STRIND	$1EF8
STRREF	$1156	STRTME	$7901	STRX-!	$129C
SUBROI	$37CA	SUBSCR	$339A	SUBST1	$57E2
SUBST2	$5821	SUBSTF	$EC64	SUMIT	$FAF6
SUMIT+	$FAF9	SVADCK	$580A	SVADR	$54AA
SVADR+	$54A9	SVCCHK	$462B	SVCCLR	$063F
SVCSET	$0634	SVVAL3	$54A4	SYCALL	$50EC
SYSINT	$7A2D	*SYSJSB	$466D	TAB	$5BDB
TAB.	$FD91	TADREP	$EC0F	TAN10	$39ED
TBL3	$3E53	TBL3B	$3E8B	TCKTGL	$75D0
TENRIT	$7047	TEXT.	$523E	THERE!	$5E7D
THERE1	$734C	THERE?	$5E74	TICK	$6844
TIMCHK	$71A7	TIMDIV	$64ED	TIME$.	$7C6D
TIME.	$7CFA	TIMEMD	$673A	TIMMSK	$654A
TINCHK	$67F7	TM#GET	$3FDC	TMECMD	$757A
TMEFAC	$3F38	TMESET	$76B2	TMRCLR	$409E
TMRDNE	$4020	TMRGO	$3FF2	TMRKIL	$407F
TMROF.	$4049	TMRON.	$3F7B	TMROPN	$402F
TMRRLV	$3FFE	TMRSS	$408E	TMTRIG	$40A3
TO.	$EE34	TO?I	$E2A4	TOASC2	$0A91
TOBCD2	$0A9E	TOBCD8	$0AB8	TOBIG?	$E6B0
TOBIN2	$0ACC	TOBIN8	$0ADB	TOLCD	$F02E
TOPROM	$E000	TRAPTX	$EA6F	TRC01	$3C9F
TRC7	$3C95	TRCEP	$48E5	TRCHED	$4957
TRCLP	$48DD	TRCVBL	$489C	TRFLO.	$4887
TRFRM.	$7312	TRJUMP	$4926	TRKEY	$02A6
TRNSLP	$7056	TROFF.	$488C	TRVAR.	$4890
TRY1N.	$12FC	TSTALO	$2207	TSTEND	$2925
TSTRM	$2919	TTMPLT	$6507	TWOB	$3E8E
TWOR	$3D02	TWOROI	$3EE9	TYPOK?	$2478
TYPSTM	$13C7	ULIN#.	$E413	UNAUTO	$01F3
UNEQ$.	$4D8D	UNEQ.	$4DFA	UNLREP	$EBF6
UNLRP?	$E8ED	UNLSND	$EBE5	UNQUOT	$1212
UNROM	$47E2	UNSEE	$F021	UNSTAK	$32D4
UNTREP	$EC04	UNTSND	$EB12	UNTUNL	$7026
UPC$.	$FDCC	UPDISP	$67FF	UPENT	$5913
UPRC*	$1B35	UPRCSE	$7989	UPSET	$005C
UROOM!	$1F85	USING.	$E3F9	UT210	$3D0B
VAL$.	$FC8B	VAL.	$FCF5	VCCALL	$5101
VCREST	$4FFC	VER.	$FBFA	VERB	$6B47
VERIFY	$F829	VERMNO	$069C	VERMN1	$358A
VERMN2	$5417	VERMN3	$720E	VERMN4	$F5C5
VERMN5	$6F44	VERSUB	$FC20	VF1TO2	$6FD8
VFADCL	$6CE8	VFADDR	$6B80	VFBSY	$6D06
VFBSY+	$67C1	VFBYE	$6DAD	VFCD46	$6D8C
VFCDC0	$6D97	VFCDEP	$6D86	VFCLCH	$6CED
VFDDL2	$6CBE	VFDECL	$6DDF	VFDIR	$6DBC
VFDIR+	$67AD	VFDUDE	$6EEE	VFEOD?	$6DF7
VFERR	$6D48	VFEXCH	$6E7A	VFGET	$6A22
VFGLOC	$6E19	VFHI	$6A07	VFHI+	$67A8
VFLAD	$EC18	VFLAD+	$67A3	VFLED?	$6E09
VFLIF?	$6EB3	VFLTBY	$6CFF	VFLTY+	$67CB
VFMFP?	$6E25	VFMM?	$6A9A	VFMOVE	$6E38
VFMSG	$6ED9	VFNXD-	$6DD8	VFNXDE	$6DCD
VFNXE+	$67C6	VFPED?	$6DFC	VFRCEX	$6D76
VFRDE	$6F68	VFRENA	$6E2B	VFRLF?	$6EBA
VFROO?	$6E90	VFRREC	$6C80	VFRVDE	$6EC3

VFRWO+	$67D0	VFRWK+	$679E	VFRWRD	$6BF3
VFRWSO	$6BB2	VFRWSB	$6BE0	VFRWSK	$6BA8
VFRWUO	$6D7F	VFRWWR	$6C92	VFSECT	$6FE3
VFSKFL	$6EE3	VFSTAT	$6D0F	VFTAD	$EC0F
VFTAD+	$67D5	VFTERM	$6DA2	VFTIME	$6F46
VFTRNL	$6F0F	VFUTL+	$67DA	VFWAC2	$6C3D
VFWACH	$6C2F	VFWBUO	$6E84	VFWOOP	$6C8A
VFWR	$6CA8	VFWRBK	$6CC3	VFWRCL	$6CF8
VFWRD-	$6DF0	VFWRD1	$6DED	VFWRDE	$6DE8
VFWREC	$6CB6	VSRCH	$27A0	WAIT.	$E366
WAITCK	$7E64	WAITKY	$0755	WAKEUP	$7666
WARN	$4BD3	WARN.R	$4C34	WCKJSB	$E36F
WRITCR	$F913	YEARTM	$659F	YINCHK	$67F7
YTX	$3974	YTX5'	$778B	Z36.56	$2A67
ZROM1-	$2A79	ZROMEM	$2A6C	ZZZZZZ	$0899

6.3. Beschreibung besonderer Adressen:

a) CONBIN:

Diese Subroutine dient zum Umwandeln einer binären Zahl in eine Zahl mit Fließkomma.

INPUT: Binärzahl in R36/37
OUTPUT: Fließkommazahl in R40/47

b) CONINT:

Diese Subroutine dient zum Umwandeln einer Zahl mit Fließkomma in eine binäre Zahl.

INPUT: Fließkommazahl in R60/67
OUTPUT: Binärzahl in R76/77

c) ERROR:

Mit Hilfe dieser Subroutine wird ein Vorgang auf Fehler untersucht.

OUTPUT: E=1 bei Fehlern
E=0 bei Nicht-Fehler

d) ERROR+:

Mit Hilfe dieser Subroutine wird wie bei ERROR ein Vorgang auf Fehler untersucht.

OUTPUT: E=1 bei Fehlern
E=0 bei Nicht-Fehlern

e) NUMVAL:

Diese Subroutine dient zum Suchen nach einem numerischen Wert innerhalb des auszuführenden Schlüsselwortes.

OUTPUT: E#0 wenn ein numerischer Wert gefunden wird
E=0 wenn kein numerischer Wert gefunden wird

f) ONEB:

Diese Subroutine verwertet wie die noch beschriebenen Routinen ONEI und ONER die Parameter in den Schlüsselworten und schiebt diese in den Stack der CPU.

OUTPUT: Resultat in binärer Form in R46/47 und R76/77.

g) ONEI:

Wie ONEB.

OUTPUT: Resultat in INTEGER-Form in R46/47

h) ONER:

Wie ONEB.

OUTPUT: Resultat in reeller Form in R40/47

i) ROMJSB:

Es wird eine Routine in einem speziellen LEX-File aufgerufen.

INPUT: Die zur Ausführung des LEX-Files benötigten Werte
OUTPUT: Ergebnis der aufgerufenen Routine

j) SYSJSB:

Es wird eine Routine im System-ROM aufgerufen.

INPUT: Adresse der Unterroutine
OUTPUT: Ergebnis der aufgerufenen Routine

7. Programmlistings

7.1. Das Programm 'COMPILE'

```
100 ! ** HP-75-Compiler **
110 DIM C$[25110 DIM C$[256],B$[30],R$[256],F$[30],H9$[16],P$[128]
120 ON ERROR OFF ERROR
130 PURGE 'ASSEM1'
140 DELAY 0 @ H9$='0123456789ABCDEF'
150 FOR I=1 TO 6 @ A(I)=0 @ NEXT I
160 DISP '  *  * A S S E M B L E R *  *  ' @ WAIT 2
170 C$='' @ Z5,A,Y,B=0 @ A9,D9=-1 @ P,F=1
180 ! - Verarbeitung des Textfiles -
190 INPUT 'Filename: ';A$
200 ASSIGN # 1 TO A$,TEXT @ ASSIGN # 2 TO 'ASSEM1',TEXT
210 EDIT A$ @ RENUMBER 10,10,1
220 EDIT 'COMPILE' @ RESTORE # 1 @ DISP 'Compilation . . . . .' @ GOSUB
9000 @ P=1
230 GOSUB 9970 @ Z5=Z5+1 @ READ # 1,Z5*10 ; B$ @ L=LEN(B$) @ E$='' @ 
-Y+10
240 I=6 @ IF B$[1,2]='ID' OR B$[1,2]='ZD' THEN C$=C$&B$[8,9]&B$[6,7] @
 A=A+2 @ GOTO 500
250 IF B$[1,3]='DEF' THEN C$=C$&'0000' @ A=A+2 @ GOTO 500
260 IF B$[1,3]='LAB' THEN 1900
270 IF B$[1,3]='END' AND LEN(B$)#3 THEN 1800
280 IF B$[1,2]='AS' THEN 1700
290 IF B$[1,3]='END' THEN C$=C$&'FF' @ GOTO 600
300 IF L=3 THEN D$=B$[1,3] @ GOTO 400 ELSE D$=B$[1,4]
310 IF B$[1,3]='BYT' THEN C$=C$&B$[6,7] @ A=A+1 @ GOTO 500
320 IF B$[1,1]='!' THEN 230
330 IF B$[1,I]="'" THEN 1500
340 IF B$[I,I]="R" THEN 1000
350 IF B$[I,I]="X" THEN 1100
360 IF B$[I,I]="#" THEN 1200
370 IF B$[I,I]="=" THEN 1400
380 IF I=10 AND POS('-+',B$[I,I]) THEN 1300
390 DISP 'ERR L'&STR$(Y) @ GOTO 230
400 ! - Befehlscodierung -
410 RESTORE @ Q=128
420 READ R$ @ IF R$=D$ THEN 440
430 Q=Q+1 @ IF Q=256 THEN DISP 'ERR L'&STR$(I) @ GOTO 230 ELSE 420
440 A=A+1 @ C$=C$&FNY$(Q)&E$
500 ! - Weitergabe des Hex-Codes -
520 GOTO 230
600 ! - Vervollstaendigen der Hex-Codes -
610 PRINT # 2,P ; ' '&C$ @ F=3
620 Z9=2 @ C$=''
625 READ # 1,Z9*10 ; B$ @ IF B$[1,3]#'DEF' THEN 630
630 IF B$[6,8]='MAI' THEN C=VAL(B$[10,LEN(B$)]) @ C$=C$&FNY$(B(C)) @ 
GTO 630
635 IF B$[6,8]#'PAR' THEN C$=C$&FNY$(A(LEN(B$)-7)) @ GOTO 650
640 IF LEN(B$)=10 THEN C$=C$&FNY$(A(2)) @ GOTO 650
645 C=VAL(B$[12,LEN(B$)]) @ C$=C$&FNY$(C(C))
650 Z9=Z9+1 @ GOTO 625
655 IF B$[1,2]#'ZD' THEN 662
660 C$=C$&B$[8,9]&B$[6,7] @ Z9=Z9+1 @ GOTO 625
662 IF B$[1,5]='LABEL' THEN Z9=Z9+1 @ GOTO 625
665 READ # 2,1 ; R$ @ R$=R$[2] @ IF P>2 THEN READ # 2,2 ; P$ @ P$=P$[C
] ELSE P$=''
670 R$=R$&P$
675 R$[5,LEN(C$)+4]=C$
680 PRINT # 2,1 ; ' '&R$[1,64]
685 IF P>2 THEN PRINT # 2,2 ; ' '&P$[65,128]
690 READ # 2,P ; C$
700 ! - Umwandlung Hex-Codes in LEX-File -
710 I=1354 @ P=P-1
730 R$=''
```

```
740 FOR J=I TO I+255 @ R$=R$&CHR$(PEEK(J)) @ NEXT J
750 U=POS(R$,'ASSEM1') @ IF U=0 THEN I=I+256 @ GOTO 740
760 P1=U-9+I @ U=NUM(R$[U-9,U-9])*256+NUM(R$[U-10,U-10])-32768
770 W=P*32+LEN(C$)/2-1 @ V=U+14
780 FOR J=1 TO P
790 FOR I=V TO V+64 STEP 2
800 R$=CHR$(PEEK(I))&CHR$(PEEK(I+1))
810 T=POS(H9$,R$[1,1])*16+POS(H9$,R$[2,2])-17
820 POKE U,ABS(T) @ U=U+1 @ NEXT I @ V=V+68 @ U=U-1 @ NEXT J
830 IF LEN(C$)=0 THEN 870
840 FOR I=2 TO LEN(C$) STEP 2
850 T=POS(H9$,C$[I,I])*16+POS(H9$,C$[I+1,I+1])-17
860 POKE U,T @ U=U+1 @ NEXT I
870 Q=W\256 @ Y=W-256*Q @ POKE P1,Y @ POKE P1+1,Q
880 POKE P1+2,141 @ POKE P1+3,76
890 DISP 'READY . . . .' @ BEEP @ BEEP @ BEEP
900 STOP
1000 ! - Unterroutine fuer Registercodierung -
1010 IF B$[2,3]='RP' THEN 1150
1020 E=VAL(B$[I+1,I+1])*8+VAL(B$[I+2,I+2])+64*(I=6)
1030 IF I=6 THEN Z=D9 ELSE Z=A9
1040 IF E#Z THEN Z=E @ C$=C$&FNY$(E) @ A=A+1
1050 IF I=6 THEN D9=Z ELSE A9=Z
1060 D$=D$&'R'
1070 I=I+4 @ IF I>L THEN 400 ELSE 330
1100 ! - Unterroutine fuer ind. Variablen -
1110 E=VAL(B$[I+1,I+1])*8+VAL(B$[I+2,I+2])
1120 IF E#A9 THEN A9=E @ C$=C$&FNY$(E) @ A=A+1
1130 DISP D$&D$&'X' @ I=I+4 @ IF I>L THEN 400 ELSE 330
1150 ! - Unterroutine fuer Registerpointer -
1160 E=VAL(B$[7,7])*8+VAL(B$[8,8])+64*(B$[1,1]='D')
1170 IF B$[1,1]='D' THEN D9=E ELSE A9=E
1180 A=A+1 @ C$=C$&FNY$(E) @ GOTO 500
1200 ! - Unterroutine fuer Adressencodierung -
1210 E$=E$&B$[I+3,I+4]&B$[I+1,I+2]
1220 A=A+2 @ I=I+6 @ D$=D$&'$' @ IF I>L THEN 400 ELSE 330
1300 ! - Unterroutine fuer Stacks -
1310 E=VAL(B$[12,12])*8+VAL(B$[13,13])
1320 IF E#A9 THEN A9=E @ C$=C$&FNY$(E) @ A=A+1
1330 D$=D$&B$[10,10] @ GOTO 400
1400 ! - Unterroutine fuer Literal -
1410 E$=E$&B$[I+1,I+2] @ A=A+1 @ D$=D$&'='
1420 IF L=8 OR L=12 AND I=10 THEN 400
1430 FOR I=14 TO L-1 STEP 3
1440 E$=E$&B$[I,I+1] @ A=A+1 @ NEXT I
1450 GOTO 400
1500 ! - Unterroutine fuer Labels -
1510 ON ERROR DISP 'ERR L'&STR$(Y) @ GOTO 230
1520 COPY ':CA' TO 'ASSEM2'
1530 ASSIGN # 3 TO 'ASSEM2'
1540 FOR I=1 TO 100
1550 READ # 3,I ; F$ @ IF F$[6,LEN(F$)]=B$[I+1,L-1] THEN 1570
1560 NEXT I @ DISP 'ERR L'&STR$(Y) @ GOTO 230
1570 E$=E$&B$[3,4]&B$[1,2] @ A=A+2 @ D$=D$&'$' @ GOTO 400
1700 ! - Unterroutine fuer Mnemonics -
1710 FOR X=7 TO L-2 @ C$=C$&FNY$(NUM(B$[X,X])) @ A=A+1 @ NEXT X
1720 C$=C$&FNY$(NUM(B$[L-1,L-1])+128) @ A=A+1 @ GOTO 500
1800 ! - Unterroutine fuer Ende Label -
1810 IF B$[6,6]='P' THEN A=A+2 @ C$=C$&'FFFF' @ GOTO 500
1820 C$=C$&'FF' @ A=A+1 @ GOTO 500
1900 ! - Unterroutine fuer Labels -
1910 IF B$[7,9]='MAI' THEN C=VAL(B$[11,L]) @ B(C)=A @ GOTO 230
1920 IF B$[7,9]#'PAR' THEN A(L-8)=A @ GOTO 230
1930 IF LEN(B$)=11 THEN A(2)=A @ GOTO 230
1940 C=VAL(B$[13,L]) @ C(C)=A @ B=MAX(B,C) @ GOTO 230
2000 ! - Befehlstabelle -
```

```
2010 DATA ELB R,ELM R,ERB R,ERM R,LLB R,LLM R,LRB R,LRM R,ICB R,ICM R,
DCB R,DCM R,TCB R,TCM R
2020 DATA NCB R,NCM R
2030 DATA TSB R,TSM R,CLB R,CLM R,ORB RR,ORM RR,XRB RR,XRM RR,BIN,BCD,
SAD,DCE,ICE,CLE,RTN,PAD
2040 DATA LDB RR,LDM RR,STB RR,STM RR,LDBDRR,LDMDRR,STBDRR,STMDRR,LDB
R=,LDM R=,STB R=,STM R=
2050 DATA LDBIRR,LDMIRR,STBIRR,STMIRR
2060 DATA LDBDR$,LDMDR$,STBDR$,STMDR$,LDBDRX$,LDMDRX$,STBDRX$,STMDRX$,
LDBIR$,LDMIR$,STBIR$
2070 DATA STMIR$,LDBIRX$,LDMIRX$,STBIRX$,STMIRX$
2080 DATA CMB RR,CMM RR,ADB RR,ADM RR,SBB RR,SBM RR,JSB X$,ANM RR,CMB
R=,CMM R=,ADB R=,ADM R=
2090 DATA SBB R=,SBM R=,JSB $,ANM R=
2100 DATA CMBDR$,CMMDR$,ADBDR$,ADMDR$,SBBDR$,SBMDR$,,ANMDR$,CMBDRR,CMM
DRR,ADBDRR,ADMDRR
2110 DATA SBBDRR,SBMDRR,,ANMDRR
2120 DATA POBDR+,POMDR+,POBDR-,POMDR-,PUBDR+,PUMDR+,PUBDR-,PUMDR-,POBI
R+,POMIR+,POBIR-,POMIR-
2130 DATA PUBIR+,PUMIR+,PUBIR-,PUMIR-
2140 DATA JMP =,JNO =,JOD =,JEV =,JNG =,JPS =,JNZ =,JZR =,JEN =,JEI =,
JNC =,JCY =
2150 DATA JLZ =,JLN =,JRZ =,JRN =
9000 ! - Unterroutine zur Herstellung der Syntax -
9010 READ # 1,P*10 ; B$ @ J=1 @ FOR I=1 TO LEN(B$)
9020 IF B$[I,I]#' ' THEN B$[J,J]=B$[I,I] @ J=J+1
9030 NEXT I @ B$=B$[1,J-1] @ IF B$='END' THEN RETURN
9040 IF B$[1,1]='!' THEN B$='! '&B$[2,LEN(B$)] @ P=P+1 @ GOTO 9120
9050 IF LEN(B$)=3 THEN 9120
9060 IF POS(H9$[1,10],B$[3,3])#0 THEN B$=B$[1,2]&'    '&B$[3,LEN(B$)] @
 GOTO 9120
9070 IF B$[1,5]='LABEL' THEN B$=B$[1,5]&' '&B$[6,LEN(B$)] @ GOTO 9120
9080 IF B$[1,4]='ENDE' THEN B$=B$[1,4]&' '&B$[5,LEN(B$)] @ GOTO 9120
9090 IF B$[1,3]='BYT' THEN B$=B$[1,3]&'  '&B$[4,LEN(B$)] @ GOTO 9120
9100 IF POS('DI',B$[4,4])#0 THEN B$=B$[1,4]&' '&B$[5,LEN(B$)] @ GOTO 9
120
9110 B$=B$[1,3]&'  '&B$[4,LEN(B$)]
9120 PRINT # 1,P*10 ; B$ @ P=P+1 @ GOTO 9010
9900 DEF FNY$(X)
9910 H1$='' @ H2=ABS(X) @ H9$='0123456789ABCDEF'
9920 H1=1+MOD(H2,16) @ H1$=H9$[H1,H1]&H1$ @ H2=H2\16 @ IF H2 THEN 9920
9930 H1$='000'&H1$ @ H1$=H1$[LEN(H1$)-F,LEN(H1$)]
9940 IF F=3 THEN H1$=H1$[3,4]&H1$[1,2]
9950 FNY$=H1$
9960 END DEF
9970 IF LEN(C$)>64 THEN PRINT # 2,P ; ' '&C$[1,64] @ C$=C$[65,LEN(C$)]
 @ P=P+1
9980 RETURN
```

7.2. Das Text-File 'ASSEM2'

```
 018ACONBIN
 1D37CONINT
 4C91ERROR
 4C99ERROR+
 0E7CNUMVAL
 3E8BONEB
 3EACONEI
 3ECDONER
 4694ROMJSB
 466DSYSJSB
```

ANHANG A

Die Speicherung von Informationen auf dem Magnetband

Durch das Digitalkassettenlaufwerk HP 82161A wird der HP-75 um beachtliche Massenspeicherfähigkeiten erweitert. Mit Hilfe dieser Peripherieeinheit können bis zu 512 Records (131072 Bytes) an Informationen auf einer Mini-Datenkassette aufgezeichnet werden. Das Laufwerk wird über die Hewlett-Packard-Interface-Loop (HP-IL) an den HP-75 angeschlossen.
Jedes Magnetband muß vor der ersten Benutzung initialisiert werden. Dabei werden auf zwei Spuren 0 und 1 die 512 Records zu je 256 Bytes eingerichtet; dh. es wird auf dem Band Anfang und Ende eines jeden Records festgelegt. Dabei werden die Records auf jeder Spur von 0 bis 255 durchnumeriert und jedes Byte auf den Wert 255 gesetzt.

1. Die Records 0 und 1 auf der Spur 0:
   Auf diesen beiden Records sind Bandkennung und sämtliche Identifikationsinformationen untergebracht.
   Record 0:

Bytes		Werte
0-1	LIF ID	128 0
2-7	Kennungstext (Volume Label)	je nach Text
8-11	Anfangsrecord Verzeichnis	0 0 0 2 (= 2 Spuren)
12-13	System 3000	16 0
14-15	keine Verwendung	0 0
16-19	Länge des Verzeichnisses	0 0 x x
20-21	Version	0 1
22-23	keine Verwendung	0 0
24-27	Spuren pro Oberfläche	0 0 0 2 (= 2 Spuren)
28-31	Anzahl der Oberflächen	0 0 0 1
32-35	Records pro Spur	0 0 1 0 (= 256 Records)
36-41	Datum und Uhrzeit der Initialisierung	x x x x x x
42-255	Verwaltung	0 0 0 . . . 0

   Record 1:

Bytes		
0-255	System 3000	0 0 0 . . . 0

2. Das Directory:
   Das Directory ist das Inhaltsverzeichnis des Bandes. Das Directory beginnt mit dem Record 2 eines Bandes. In jedem Record sind 8 Einträge zu je 32 Bytes zusammengefaßt. Für jedes File auf dem Band gibt es demnach einen Eintrag. Dabei wird in einem Eintrag folgendes abgelegt:

Bytes		Werte	
0-9	Flename	x x x . . . x	
10-11	Filetyp	0 1	LIF1
		224 82	TEXT
		224 83	APPT
		224 136	BASIC
		224 137	LEX
12-15	Anfangsrecord des Files	0 0 x x	
16-19	Filegröße	0 0 x x	
20-25	Datum und Uhrzeit der Erzeugung	x x x x x x	
26-27	Datenträger-Flag/Nummer	128 1	
28	Implementation	x	
29	Implementation	x	
30	Implementation	x	
31	Implementation	x	

   Diese Bytes 28-31 werden beim HP-75 zur Speicherung des Paßwortes, mit dem das Programm auf Kassette geschützt werden kann, benutzt.
   Die Länge des Directory-Teils wird bestimmt durch das Byte 19 in Record 0. Dieser Wert, um 1 vergrößert, gibt an, bis zu welchem Record das Directory geht.

3. Der Teil des Bandes, in dem die Files stehen:
   Dieser teil beginnt direkt hinter dem für das Directory reservierten Platz. Die einzelnen Files sind dabei so codiert wie in Kapitel 2 besprochen. Der Filebeginn eines jeden Files wird durch den Eintrag im Directory für dieses File bestimmt. Und zwar ist der Filebeginn durch die Bytes 14-15 festgelegt. Diese bestimmen, bei welcher Spur- und Recordnummer das File beginnt.

ANHANG B

Beschreibung der benötigten LEX-Karten

1. PEEKPOKE:

PEEK	(dezimale Adresse im Memory-Bereich 0000-7FFF)
POKE	(dezimale Adresse im Memory-Bereich 0000-7FFF), Inhalt für diese Adresse

a) Kopieren des Files 'PEEKPOKE' von Magnetkarte durch das Kommando:

COPY CARD TO 'PEEKPOKE'

b) Der Befehl 'PEEK':
Beim Ausführen dieses Kommandos innerhalb oder außerhalb eines Programmes steht im Anzeigeregister der dezimale Wert des Bytes, das durch die Adresse festgelegt ist.
Bei PEEK(N) muß N im Bereich von 0-32767 liegen.

c) Der Befehl 'POKE':
Beim Ausführen dieses Kommandos wird das Byte mit der festgelegten Adresse auf den festgelegten Inhalt gesetzt.
POKE (N),M bedeutet, daß unter der Adresse N das Byte auf den Wert M gesetzt wird. N liegt im Bereich 0-32767, M im Bereich 0-255.
Auch dieser Befehl kann innerhalb und außerhalb eines Programmes ausgeführt werden.
Beachte: Dieser Befehl läßt sich nur auf ganz bestimmte Bytes im Betriebssystem anwenden. Ob der Wert M übernommen wurde, kann man jedoch leicht durch Ausführen des Kommandos 'PEEK' überprüfen.

d) Man benötigt zur Ausführung dieses Kommandos keine weitere Peripherie, diese LEX-Karte ist ohne Begrenzung immer ausführbar.

ACHTUNG ! ! !

Beim POKEn im Betriebssystem kann es bei nicht sachgemäßer Behandlung passieren, daß der Rechner in einen deep-sleep-mode verfällt. Er läßt sich daraus nur durch Ausschalten und Entfernen der Stromzufuhr wieder abfangen. Dabei wird der Inhalt des Rechners gelöscht.
Deshalb ist es besser, erst durch den Befehl 'PEEK' das System zu erkunden, und dann erst mit 'POKE' zu manipulieren.

```
0000 ID   4015
0002 DEF  RUNTIME
0004 DEF  NAMES
0006 DEF  PARSE
0008 DEF  ERRORS
000A DEF  RUN
000C LABEL RUNTIME
000C ZD   6105
000E DEF  MAIN1
0010 DEF  MAIN2
0012 LABEL PARSE
0012 ZD   6105
0014 DEF  RUNAGAIN
0016 ENDE POINTERS
0018 LABEL NAMES
0018 ASC  "POKE"
001C ASC  "PEEK"
0020 ENDE NAMES
0021 LABEL ERRORS
0021 ENDE ERRORS
0022 DRP  R01
0023 ARP  R27
0024 LABEL RUN
0024 RTN
0025 BYT  A1
0026 LABEL MAIN1
0026 BIN
0027 DRP  R20
0028 SBM  R20,=F1,60
002B JSB  $3E8B
002E DRP  R46
002F ARP  R26
0030 STM  R46,R26
0031 ARP  R20
0032 JSB  X20,$6133
0035 DRP  R26
0036 ARP  R46
0037 STBD R26,R46,$0000
003A RTN
003B LABEL RUNAGAIN
003B DRP  R02
003C ARP  R41
003D LDM  R02,R41
003E ARP  R06
003F PUMD R02,+R06
0040 JSB  $466D
0043 BYT  58
0044 BYT  12
0045 DRP  R54
0046 LDB  R54,=B4
0048 DRP  R02
0049 ARP  R06
004A POMD R02,-R06
004B ARP  R55
004C STM  R02,R55
004D DRP  R57
004E ARP  R12
004F POBD R57,-R12
0050 DRP  R54
0051 PUMD R54,-R12
0052 RTN
0053 BYT  10
0054 BYT  2D
0055 LABEL MAIN2
0055 BIN
0056 DRP  R20
```

```
0057 SBM  R20,=20,61
005A ARP  R20
005B JSB  X20,$6133
005E DRP  R36
005F CLM  R36
0060 ARP  R46
0061 LDBD R36,X46,$0000
0064 JSB  $FCE4
0067 RTN
0068 JSB  $3E8B
006B JNG  =06
006D DRP  R46
006E CMM  R46,=FF,FF
0071 JNC  =F4
0073 JSB  $4C99
0076 BYT  59
0077 END
```

2. PEKEPOOK:

PEKE	(dezimale Adresse im Memory-Bereich 8000-FFFF)
POOK	(dezimale Adresse im Memory-Bereich 8000-FFFF), Inhalt für diese Adresse

a) Kopieren des Files 'PEKEPOOK' von Magnetkarte durch das Kommando:

COPY CARD TO 'PEKEPOOK'

b) Der Befehl 'PEKE':
Beim Ausführen dieses Kommandos innerhalb oder außerhalb eines Programms steht im Anzeigeregister der dezimale Wert des Bytes, das durch die Adresse festgelegt ist.
Die Werte für die Adresse liegen im Bereich 32768-65535. Zur Ausführung des PEKE-Befehls muß von dieser Adresse 32768 subtrahiert werden, so daß bei PEKE(N) die Werte von N wieder im Bereich von 0-32767 liegen.
Bsp: Um den Inhalt der Adresse 55748 zu lesen, muß man N=55748-32768=22980 bilden. Der Befehl für das Lesen des Inhalts der Adresse lautet PEKE(22980).

c) Der Befehl 'POOK':
Beim Ausführen dieses Kommandos wird das Byte mit der festgelegten Adresse auf den festgelegten Inhalt gesetzt.
POOK (N),M bedeutet, daß unter der Adresse N das Byte auf den Wert M gesetzt wird. Für N gilt das unter b) gesagte, M liegt wieder im Bereich 0-255.
Auch dieser Befehl läßt sich inner- und außerhalb eines Programmes anwenden.
Beachte: Dieser Befehl läßt sich nur auf ganz bestimmte Bytes im Betriebssystem anwenden. Ob der Wert M übernommen wurde, kann man jedoch leicht durch Ausführen des Kommandos 'PEKE' überprüfen.

d) Man benötigt zur Ausführung dieser Kommandos keine Peripherie, diese LEX-Karte ist universell einsetzbar.

ACHTUNG ! ! !

Beim POOKen im Betriebssystem kann es bei nicht sachgemäßer Behandlung passieren, daß der Rechner in einen deep-sleep-mode verfällt. Er läßt sich daraus nur durch Ausschalten und Entfernen der Stromzufuhr wieder abfangen. Dabei wird der Inhalt des Rechners gelöscht.
Deshalb ist es besser, erst durch den Befehl 'PEKE' das System zu erkunden, und dann erst mit 'POOK' zu manipulieren.

```
0000 ID   4014
0002 DEF  RUNTIME
0004 DEF  NAMES
0006 DEF  PARSE
0008 DEF  ERRORS
000A DEF  RUN
000C LABEL RUNTIME
000C ZD   6105
000E DEF  MAIN1
0010 DEF  MAIN2
0012 LABEL PARSE
0012 ZD   6105
0014 DEF  RUNAGAIN
0016 ENDE POINTERS
0018 LABEL NAMES
0018 ASC  "POOK"
001C ASC  "PEKE"
0020 ENDE NAMES
0021 LABEL ERRORS
0021 ENDE ERRORS
0022 DRP  R01
0023 ARP  R27
0024 LABEL RUN
0024 RTN
0025 BYT  A1
0026 LABEL MAIN1
0026 BIN
0027 DRP  R20
0028 SBM  R20,=F1,60
002B JSB  $3E8B
002E DRP  R46
002F ARP  R26
0030 STM  R46,R26
0031 ARP  R20
0032 JSB  X20,$6133
0035 DRP  R26
0036 ARP  R46
0037 STBD R26,R46,$8000
003A RTN
003B LABEL RUNAGAIN
003B DRP  R02
003C ARP  R41
003D LDM  R02,R41
003E ARP  R06
003F PUMD R02,+R06
0040 JSB  $466D
0043 BYT  58
0044 BYT  12
0045 DRP  R54
0046 LDB  R54,=B4
0048 DRP  R02
0049 ARP  R06
004A POMD R02,-R06
004B ARP  R55
004C STM  R02,R55
004D DRP  R57
004E ARP  R12
004F POBD R57,-R12
0050 DRP  R54
0051 PUMD R54,-R12
0052 RTN
0053 BYT  10
0054 BYT  2D
0055 LABEL MAIN2
0055 BIN
0056 DRP  R20
```

```
0057 SBM  R20,=20,61
005A ARP  R20
005B JSB  X20,$6133
005E DRP  R36
005F CLM  R36
0060 ARP  R46
0061 LDBD R36,X46,$8000
0064 JSB  $FCE4
0067 RTN
0068 JSB  $3E8B
006B JNG  =06
006D DRP  R46
006E CMM  R46,=FF,FF
0071 JNC  =F4
0073 JSB  $4C99
0076 BYT  59
0077 END
```

3. IOUTIL oder HPILCMDS: (© Copyright für dieses Produkt liegt allein bei Hewlett-Packard GmbH, Warenzeichen)

```
SENDIO ':Code der Einheit','Liste der Kommandos','Liste der Daten'
ENTIO$(':Code der Einheit','Liste der Kommandos')
SEND?
```

a) Kopieren des Files in den rechner durch Eingabe des Assemblerslistings mit Hilfe des Assemblers in den Rechner.

b) Der Befehl 'SENDIO':
Der SENDIO-Befehl wird gebraucht, um Kommandos und Daten an eine HP-IL-Einheit zu senden. SENDIO kann sowohl direkt vom Tastenfeld als auch in einem BASIC-Programm aufgerufen und ausgeführt werden.
Erklärung der Bestandteile:

':Code der Einheit'	Der Name, der der Einheit durch den ASSIGN IO-Befehl gegeben wurde. Ist der Code der Einheit nicht bekannt, steht dann der Leerstring ''.
':Liste der Kommandos'	Eine Liste von HP-IL-Kommandos, durch Kommata getrennt. Nähere Auskünfte über diese Kommandos in IL-Handbüchern.
':Liste der Daten'	Ein Charakter-String, der die Daten übermittelt

Mit Hilfe des SENDIO-Kommandos kann eine Einheit aktiviert oder in einen Status (Listener, Talker, Controller) versetzt werden.

c) Der ENTIO$-Befehl:
Der ENTIO$-Befehl wird benötigt, um Daten von einer Einheit zu empfangen. Man erhält einen String, durch den die Daten codiert sind.
Die angesprochene Einheit muß im Talker-Modus sein, um antworten zu können.
Mit Hilfe von ENTIO$ kann man Einheiten einer Schleife genau einordnen, da verschiedene Einheiten verschiedene Identifikationscodes senden. Auch hier kann, wenn der Code der Einheit nicht bekannt ist, folgendermaßen abgekürzt werden: ENTI$('','Liste der Kommandos')

d) Der SEND?-Befehl:
Der SEND?-Befehl benötigt keine Parameter. Nach seiner Ausführung erhält man eine ganze Zahl. Ihr Wert ist die Position in der Datenliste des Strings , der infolge einer Fehlermeldung nicht vollständig mit dem letzten SENDIO-Befehl übermittelt wurde. Normalerweise erhält man nach Ausführung des SEND?-Befehls den Wert 0.

e) Zur Ausführung der Karte muß eine IL-Schleife mit mindestens einer Einheit vorhanden sein.
Ansonsten ist diese Karte universell für jede Schleife einsetzbar.

```
0000 ID   4019
0002 DEF  RUNTIME
0004 DEF  NAMES
0006 DEF  PARSE
0008 DEF  ERRORS
000A DEF  RUN
000C LABEL RUNTIME
000C ZD   6115
000E DEF  MAIN1
0010 DEF  MAIN2
0012 DEF  MAIN3
0014 LABEL PARSE
0014 ZD   6115
0016 DEF  RUNAGAIN
0018 ENDE POINTERS
001A LABEL NAMES
001A ASC  "SENDIO"
0020 ASC  "ENTIO*"
0026 ASC  "SEND?"
002B ENDE NAMES
002C LABEL ERRORS
002C BYT  96
002D ASE  "device sent NRD"
003C ENDE ERRORS
003D JSB  *0E03
0040 DRP  R36
0041 SBB  R36,=30
0043 JNG  =10
0045 CMB  R36,=0A
0047 JNC  =06
0049 CMB  R36,=11
004B JNC  =08
004D SBB  R36,=07
004F ARP  R24
0050 CMB  R36,R24
0051 JCY  =02
0053 CLE
0054 RTN
0055 JSB  *4CAC
0058 DRP  R30
0059 DRP  R56
005A LDM  R56,=00,01
005D JSB  *241F
0060 RTN
0061 DRP  R01
0062 ARP  R27
0063 LABEL RUN
0063 RTN
0064 BYT  2A
0065 BYT  2E
0066 LABEL MAIN2
0066 BIN
0067 SBM  R1,=27,61
006A ARP  R14
006B STM  R1,R14
006C CLB  R1
006D JSB  X14,*611A
0070 JEN  =F1
0072 ARP  R14
0073 JSB  X14,*61E2
0076 DRP  R26
0077 ARP  R06
0078 PUMD R26,+R6
```

```
0079 DRP  R75
007A LDB  R75,=40
007C ARP  R14
007D JSB  X14,$6391
0080 DRP  R26
0081 ARP  R06
0082 POMD R26,-R6
0083 JEN  =1E
0085 DRP  R24
0086 CLM  R24
0087 DRP  R56
0088 ANM  R56,=E0,FF
008B DRP  R02
008C LDMD R2,$8467
008F CMM  R2,=A0,64,FB,0F,C9,A0
0096 DRP  R02
0097 JNC  =0A
0099 ARP  R56
009A CMM  R2,R56
009B JNZ  =0B
009D JMP  =04
009F JSB  $4C91
00A2 BYT  39
00A3 DRP  R54
00A4 CLM  R54
00A5 ARP  R12
00A6 PUMD R54,+R12
00A7 RTN
00A8 DRP  R56
00A9 ARP  R03
00AA STB  R56,R3
00AB CMM  R56,=A0,40
00AE JZR  =62
00B0 CMM  R56,=A0,41
00B3 JZR  =EA
00B5 DRP  R57
00B6 ARP  R14
00B7 JSB  X14,$61AA
00BA DRP  R03
00BB ANM  R3,=40,C8
00BE DRP  R00
00BF JNZ  =12
00C1 LDBD R0,$846A
00C4 JZR  =0D
00C6 LDB  R0,=0D
00C8 ARP  R14
00C9 JSB  X14,$61AA
00CC DRP  R03
00CD LDB  R3,=0A
00CF ARP  R14
00D0 JSB  X14,$61AA
00D3 DRP  R77
00D4 LDBD R77,$846C
00D7 JZR  =0A
00D9 ARP  R57
00DA CMB  R77,R57
00DB JNZ  =06
00DD ARP  R14
00DE JSB  X14,$61AA
00E1 JMP  =19
00E3 ARP  R14
00E4 JSB  X14,$61DC
00E7 JMP  =BF
00E9 SAD
00EA DRP  R24
00EB ICM  R24
```

```
00EC CMM  R24,=00,01,FB,04
00F1 PAD
00F2 ARP  R26
00F3 PUBD R24,+R26
00F4 RTN
00F5 PAD
00F6 DRP  R02
00F7 ARP  R06
00F8 POMD R2,-R6
00F9 DRP  R57
00FA ARP  R26
00FB PUBD R57,+R26
00FC DRP  R56
00FD ARP  R06
00FE PUMD R56,+R6
00FF LDM  R56,=A0,42
0102 JSB  $EB18
0105 DRP  R56
0106 ARP  R06
0107 POMD R56,-R6
0108 ARP  R14
0109 JSB  X14,$61DC
010C DRP  R56
010D CMM  R56,=A0,40
0110 JNZ  =8D
0112 DRP  R24
0113 ARP  R12
0114 PUMD R24,+R12
0115 DRP  R26
0116 ARP  R24
0117 SBM  R26,R24
0118 ARP  R12
0119 PUMD R26,+R12
011A RTN
011B JSB  $4661
011E BYT  0A
011F DRP  R60
0120 RTN
0121 DRP  R02
0122 CLM  R2
0123 STMD R2,$8469
0126 STBD R2,$846C
0129 ARP  R06
012A POMD R2,-R6
012B DRP  R00
012C LDM  R0,=03,62,0C,C3,06,E5,42,E5
0135 LDBD R0,$83B3
0138 STBD R0,$8468
013B DRP  R03
013C LDB  R3,=FF
013E JSB  $EADF
0141 RTN
0142 DRP  R02
0143 LDBD R2,$83B3
0146 DRP  R03
0147 LDBD R3,$846B
014A JSB  $EADF
014D DRP  R02
014E LDBD R2,$8469
0151 JZR  =0C
0153 DRP  R03
0154 LDBD R3,$82BD
0157 ANM  R3,=14,F7
015A PUBD R3,-R6
015B JSB  $EC0A
015E RTN
```

```
015F JSB $4661
0162 LDMI R3,R6
0163 DRP  R55
0164 RTN
0165 DRP  R24
0166 LDB  R24,=0A
0168 DRP  R36
0169 TSM  R36
016A JZR  =10
016C ARP  R34
016D POBD R36,+R34
016E ARP  R14
016F JSB  X14,$60FE
0172 JEN  =F0
0174 DRP  R36
0175 LLB  R36
0176 ARP  R37
0177 STB  R36,R37
0178 LLB  R36
0179 LLB  R36
017A ADB  R36,R37
017B STB  R36,R37
017C ARP  R34
017D POBD R36,+R34
017E ARP  R14
017F JSB  X14,$60FE
0182 JEN  =E0
0184 DRP  R36
0185 ARP  R37
0186 ADB  R36,R37
0187 CMB  R36,=20
0189 JNC  =D9
018B JSB $4CAC
018E BYT  59
018F BYT  44
0190 BYT  43
0191 BYT  4C
0192 BYT  14
0193 BYT  45
0194 BYT  44
0195 BYT  4E
0196 BYT  0F
0197 BYT  49
0198 DRP  R06
0199 DRP  R03
019A TSB  R3
019B BYT  4E
019C BYT  4F
019D BYT  50
019E BYT  10
019F BYT  52
01A0 DRP  R05
01A1 DRP  R16
01A2 CLB  R16
01A3 BYT  53
01A4 BYT  44
01A5 BYT  43
01A6 BYT  04
01A7 BYT  55
01A8 BYT  4E
01A9 BYT  4C
01AA BYT  3F
01AB BYT  55
01AC BYT  4E
01AD BYT  54
01AE BYT  5F
```

```
01AF BYT  47
01B0 BYT  45
01B1 BYT  54
01B2 BYT  08
01B3 BYT  47
01B4 BYT  54
01B5 BYT  4C
01B6 BYT  01
01B7 BYT  4C
01B8 BYT  4C
01B9 BYT  4F
01BA BYT  11
01BB BYT  4C
01BC DRP  R20
01BD DRP  R04
01BE DCE
01BF BYT  41
01C0 DRP  R01
01C1 DRP  R25
01C2 SAD
01C3 BYT  4E
01C4 DRP  R22
01C5 DRP  R05
01C6 CLM  R5
01C7 BYT  50
01C8 BYT  50
01C9 BYT  44
01CA BYT  05
01CB BYT  50
01CC BYT  50
01CD BYT  55
01CE BYT  15
01CF BYT  00
01D0 BYT  00
01D1 BYT  00
01D2 BYT  00
01D3 BYT  44
01D4 DRP  R04
01D5 DRP  R14
01D6 LDB  R14,R37
01D7 BYT  44
01D8 DRP  R04
01D9 DRP  R24
01DA CMB  R24,R37
01DB BYT  50
01DC DRP  R20
01DD DRP  R05
01DE ELB  R5
01DF BYT  53
01E0 BYT  41
01E1 BYT  44
01E2 BYT  60
01E3 BYT  00
01E4 BYT  00
01E5 BYT  00
01E6 BYT  00
01E7 BYT  4C
01E8 BYT  41
01E9 BYT  44
01EA BYT  20
01EB BYT  54
01EC BYT  41
01ED BYT  44
01EE BYT  40
01EF BYT  00
01F0 BYT  00
```

```
01F1 BYT  00
01F2 BYT  00
01F3 BYT  4E
01F4 BYT  52
01F5 BYT  44
01F6 BYT  42
01F7 BYT  53
01F8 BYT  44
01F9 BYT  41
01FA BYT  60
01FB BYT  53
01FC BYT  53
01FD BYT  54
01FE BYT  61
01FF BYT  53
0200 BYT  44
0201 BYT  49
0202 BYT  62
0203 BYT  53
0204 BYT  41
0205 BYT  49
0206 BYT  63
0207 BYT  54
0208 BYT  43
0209 BYT  54
020A BYT  64
020B BYT  49
020C DRP  R05
020D DRP  R20
020E STMI R20,X37,#4549
0211 DRP  R23
0212 ADMD R23,R37
0213 BYT  49
0214 DRP  R01
0215 DRP  R01
0216 PAD
0217 BYT  49
0218 DRP  R15
0219 DRP  R20
021A JRN  =5A
021C DRP  R05
021D DRP  R23
021E CMB  R23,R37
021F BYT  00
0220 BYT  00
0221 BYT  00
0222 BYT  00
0223 BYT  52
0224 DRP  R04
0225 ARP  R72
0226 LDB  R4,R72
0227 BYT  43
0228 DRP  R04
0229 ARP  R72
022A ELB  R4
022B BYT  49
022C DRP  R04
022D ARP  R72
022E CMB  R4,R72
022F BYT  44
0230 BYT  41
0231 BYT  3A
0232 BYT  00
0233 BYT  45
0234 BYT  4E
0235 BYT  3A
```

```
0236 BYT  40
0237 BYT  49
0238 DRP  R23
0239 ARP  R72
023A POBD R23,+R72
023B BYT  44
023C BYT  53
023D BYT  3A
023E BYT  20
023F BYT  45
0240 BYT  53
0241 BYT  3A
0242 BYT  60
0243 BYT  54
0244 BYT  52
0245 BYT  3A
0246 BYT  00
0247 BYT  00
0248 BYT  00
0249 BYT  00
024A BYT  00
024B BYT  54
024C BYT  4C
024D BYT  2B
024E BYT  01
024F BYT  43
0250 BYT  4C
0251 BYT  2B
0252 BYT  02
0253 BYT  00
0254 BYT  00
0255 BYT  00
0256 BYT  00
0257 BYT  41
0258 DRP  R01
0259 DRP  R04
025A ELB  R4
025B BYT  41
025C DRP  R05
025D DRP  R20
025E LDB  R20,R72
025F BYT  41
0260 DRP  R05
0261 DRP  R23
0262 CMB  R23,R72
0263 BYT  41
0264 DRP  R15
0265 DRP  R20
0266 POBD R20,+R72
0267 BYT  00
0268 BYT  00
0269 BYT  00
026A BYT  00
026B BYT  A1
026C LABEL MAIN1
026C BIN
026D SBM  R20,=2D,53
0270 ARP  R14
0271 STM  R20,R14
0272 JSB  X14,$61E2
0275 DRP  R20
0276 LDB  R20,=01
0278 DRP  R75
0279 LDB  R75,=20
027B ARP  R14
027C JSB  X14,$6391
```

```
027F DRP  R34
0280 CLM  R34
0281 STMD R34,$829E
0284 ARP  R06
0285 POMD R34,-R6
0286 ARP  R36
0287 STM  R34,R36
0288 DRP  R32
0289 ARP  R06
028A POMD R32,-R6
028B JEN  =42
028D JZR  =40
028F DRP  R57
0290 ARP  R34
0291 POBD R57,+R34
0292 JSB $EC24
0295 JEZ  =2D
0297 DCE
0298 JEN  =35
029A DRP  R56
029B ANM  R56,=E0,FF
029E CMM  R56,=A0,42
02A1 JNZ  =2C
02A3 JSB $EB18
02A6 JEN  =27
02A8 DRP  R56
02A9 ANM  R56,=80,FF
02AC DRP  R02
02AD CLM  R2
02AE ARP  R34
02AF POBD R2,-R34
02B0 ARP  R56
02B1 CMM  R2,R56
02B2 JNZ  =1B
02B4 DRP  R34
02B5 ARP  R36
02B6 SBM  R34,R36
02B7 ICM  R34
02B8 STMD R34,$829E
02BB JSB $4BC2
02BE JSB $4BD3
02C1 XRB  R34,R36
02C2 JMP  =04
02C4 DRP  R32
02C5 DCM  R32
02C6 JNZ  =C7
02C8 DRP  R56
02C9 LDM  R56,=A0,40
02CC JSB $EB18
02CF RTN
02D0 DRP  R76
02D1 ARP  R06
02D2 POMD R76,-R6
02D3 DRP  R75
02D4 STMD R75,$8464
02D7 JSB $1F11
02DA DRP  R32
02DB ARP  R12
02DC POMD R32,-R12
02DD DRP  R44
02DE ARP  R32
02DF LDM  R44,R32
02E0 ARP  R06
02E1 PUMD R44,+R6
02E2 DRP  R20
02E3 DCB  R20
```

```
02E4 JPS   =F1
02E6 CLE
02E7 ARP   R12
02E8 POMD  R20,-R12
02E9 POMD  R20,-R12
02EA JZR   =37
02EC DRP   R12
02ED ADM   R12,=04,00,CE,61,46,3A
02F4 DRP   R50
02F5 JEN   =28
02F7 DRP   R32
02F8 LDM   R32,=02,00,76,C8,20,F7
02FF BYT   12
0300 DRP   R32
0301 ICM   R32
0302 DRP   R67
0303 CMB   R67,=20
0305 JZR   =02
0307 DRP   R32
0308 ICM   R32
0309 DRP   R34
030A LDM   R34,=0D,82,74,1C
030F STMD  R34,R12
0310 JMP   =02
0312 ARP   R64
0313 LDM   R34,R64
0314 DRP   R14
0315 ARP   R06
0316 PUMD  R14,+R6
0317 JSB  $5D70
031A DRP   R14
031B ARP   R06
031C POMD  R14,-R6
031D JEZ   =04
031F JSB  $4CA4
0322 BYT   3F
0323 DRP   R26
0324 ARP   R06
0325 POMD  R26,-R6.
0326 DRP   R22
0327 POMD  R22,-R6
0328 JEN   =5D
032A JNZ   =2C
032C DRP   R57
032D LDBD  R57,$8464
0330 DRP   R20
0331 TSM   R20
0332 JNZ   =08
0334 DRP   R57
0335 CMB   R57,=20
0337 JZR   =04
0339 JSB  $4CA4
033C ARP   R77
033D JMP   =17
033F DRP   R36
0340 ARP   R20
0341 LDB   R36,R20
0342 ARP   R14
0343 JSB   X14,$64D9
0346 JEN   =3F
0348 DRP   R57
0349 LDBD  R57,$8464
034C CMB   R57,=40
034E JNZ   =06
0350 LDB   R57,=60
0352 ARP   R14
```

```
0353 JSB  X14,$649D
0356 JMP  =2F
0358 DRP  R20
0359 PUBD R20,+R14
035A DRP  R45
035B ARP  R26
035C POMD R45,+R26
035D DRP  R00
035E LDM  R0,=25,FD,CE,35,18,0C,C6,5C
0367 DRP  R44
0368 JEN  =17
036A DRP  R22
036B SBM  R22,=03,00
036E JZR  =11
0370 DRP  R20
0371 ARP  R26
0372 POBD R20,+R26
0373 CMB  R20,=2C
0375 JNZ  =06
0377 DRP  R22
0378 DCM  R22
0379 JZR  =02
037B JPS  =DD
037D JSB  $4CA4
0380 BYT  59
0381 DRP  R20
0382 ARP  R06
0383 POBD R20,-R6
0384 JSB  $45E5
0387 DRP  R02
0388 LDMD R2,$8465
038B ARP  R06
038C PUMD R2,+R6
038D RTN
038E CLE
038F DRP  R54
0390 ARP  R30
0391 POMD R54,+R30
0393 JZR  =F9
0394 DRP  R45
0395 ARP  R54
0396 CMM  R45,R54
0397 JNZ  =F6
0399 ICE
039A RTN
039B DRP  R30
039C LDM  R30,=50,62,0C,C3,C6,4F,64,F8
03A5 DRP  R65
03A6 ARP  R14
03A7 JSB  X14,$644F
03AA JEN  =66
03AC ARP  R14
03AD JSB  X14,$644F
03B0 JEN  =43
03B2 ARP  R14
03B3 JSB  X14,$644F
03B6 JEN  =24
03B8 ARP  R14
03B9 JSB  X14,$644F
03BC JEN  =6B
03BE ARP  R14
03BF JSB  X14,$644F
03C2 JEN  =22
03C4 ARP  R14
03C5 JSB  X14,$644F
03C8 JEZ  =5B
```

```
03CA ARP  R14
03CB JSB  X14,$6523
03CE JEN  =CA
03D0 DRP  R57
03D1 ARP  R36
03D2 ADB  R57,R36
03D3 ARP  R14
03D4 JSB  X14,$649D
03D7 DCE
03D8 JEZ  =C0
03DA ICE
03DB RTN
03DC DRP  R56
03DD LDB  R56,=A0
03DF STMD R56,$8467
03E2 JSB $EB18
03E5 RTN
03E6 DRP  R57
03E7 ARP  R56
03E8 STB  R57,R56
03E9 CLB  R57
03EA DRP  R56
03EB ADM  R56,=68,84
03EE STBD R56,R56
03EF CLE
03F0 DRP  R56
03F1 LDMD R56,$8467
03F4 RTN
03F5 DRP  R20
03F6 ARP  R26
03F7 POBD R20,+R26
03F8 CMB  R20,=23
03FA JNZ  =14
03FC DRP  R22
03FD DCM  R22
03FE DRP  R20
03FF ARP  R06
0400 POMD R20,-R6
0401 DRP  R36
0402 POBD R36,-R6
0403 PUBD R36,+R6
0404 DRP  R20
0405 PUMD R20,+R6
0406 DRP  R36
0407 TSB  R36
0408 JNZ  =04
040A JSB $4CAC
040D ARP  R77
040E JMP  =08
0410 DRP  R26
0411 DCM  R26
0412 ARP  R14
0413 JSB  X14,$6523
0416 JEN  =DC
0418 DRP  R57
0419 ARP  R36
041A ADB  R57,R36
041B DRP  R56
041C LDB  R56,=80
041E STMD R56,$8467
0421 JSB $EC1F
0424 RTN
0425 JSB $4CAC
0428 BYT  58
0429 DRP  R20
042A ARP  R26
```

```
042B POMD R20,+R26
042C DRP  R22
042D DCM  R22
042E DCM  R22
042F JNG  =F4
0431 ARP  R14
0432 JSB  X14,$654A
0435 JEN  =ED
0437 DRP  R57
0438 ARP  R56
0439 STB  R57,R56
043A JNZ  =06
043C DRP  R20
043D STBD R20,$846C
0440 JMP  =AD
0442 ARP  R03
0443 STB  R20,R3
0444 ARP  R20
0445 LDB  R20,R20
0446 DRP  R03
0447 ANM  R3,=C0,C8
044A ELB  R3
044D JNZ  =05
044F DRP  R55
0450 LDB  R55,=50
0452 JMP  =03
0454 DRP  R55
0455 LDB  R55,=60
0457 DRP  R56
0458 STMD R56,$9467
045B CMM  R56,=80,18
045E JZR  =C6
0460 JSB  $EB1B
0463 RTN
0464 DRP  R36
0465 CLM  R36
0466 DRP  R34
0467 ARP  R26
0468 LDM  R34,R26
0469 DRP  R20
046A POBD R20,+R26
046B JSB  $0E1C
046E JEZ  =87
0470 DRP  R22
0471 DCM  R22
0472 JNG  =B3
0474 DRP  R20
0475 ARP  R26
0476 POBD R20,+R26
0477 JSB  $0E1C
047A JEN  =04
047C DRP  R26
047D DCM  R26
047E JMP  =06
0480 DRP  R22
0481 DCM  R22
0482 JNG  =A3
0484 DRP  R36
0485 ICM  R36
0486 ARP  R14
0487 JSB  X14,$6226
048A RTN
048B DRP  R24
048C LDB  R24,=10
048E DRP  R36
048F ARP  R20
```

```
0490 LDB   R36,R20
0491 ARP   R14
0492 JSB   X14,$60FE
0495 JEN   =F3
0497 DRP   R36
0498 LLB   R36
0499 LLB   R36
049A LLB   R36
049B LLB   R36
049C ARP   R20
049D STB   R36,R20
049E ARP   R21
049F LDB   R36,R21
04A0 ARP   R14
04A1 JSB   X14,$60FE
04A4 DRP   R20
04A5 ARP   R36
04A6 LABEL RUNAGAIN
04A6 ADB   R20,R36
04A7 RTN
04A8 DRP   R02
04A9 LDB   R2,=B4
04AB ARP   R06
04AC PUBD  R2,+R6
04AD ARP   R41
04AE LDM   R2,R41
04AF ARP   R06
04B0 PUMD  R2,+R6
04B1 DRP   R14
04B2 PUBD  R14,+R6
04B3 JSB  $466D
04B6 XRB   R14,R6
04B7 ARP   R22
04B8 JZR   =07
04BA DRP   R44
04BB ARP   R06
04BC POMD  R44,-R6
04BD JSB  $4CAC
04C0 BYT   51
04C1 DRP   R14
04C2 ARP   R36
04C3 LDB   R14,R36
04C4 JSB  $466D
04C7 XRB   R14,R36
04C8 ARP   R22
04C9 JNZ   =EF
04CB DRP   R14
04CC ARP   R36
04CD LDB   R14,R36
04CE JSB  $466D
04D1 NCB   R14
04D2 BYT   12
04D3 DRP   R54
04D4 ARP   R12
04D5 POBD  R54,-R12
04D6 DRP   R54
04D7 ARP   R06
04D8 POMD  R54,-R6
04D9 ARP   R12
04DA PUMD  R54,+R12
04DB RTN
04DC LABEL MAIN3
04DC BYT   00
04DD BYT   2D
04DE DRP   R36
04DF LDMD  R36,$829E
```

```
04E2 JSB $FCE4
04E5 END
```

ANHANG C

Vorstellung und Beschreibung der Hilfsprogramme:

1. PRDIR:
   Das Programm 'PRDIR' druckt alle Angaben (ohne Paßwort) auf dem Matrixdrucker FX 80 von EPSON aus. Zusätzlich wird auch die Anzahl der verbleibenden Einträge und der freien Records ausgedruckt. Dazu muß die LEX-Karte 'HPILCMDS' aus dem Programmpaket I/O-Utilities vorhanden sein und der Drucker (':pr') sowie das Cassetten-Laufwerk (':ca) als IL-Geräte zugeordnet sein.

   Programmbeschreibung:
   1. Das Programm sowie die LEX-Karte 'HPILCMDS' einlesen.
   2. Das Programm starten
   3. Eine bis zu 15stellige Bandkennzeichnung eingeben (sollte auch auf dem Kassettenaufkleber stehen)
   4. Ausdruck erfolgt automatisch, nach Abschluß Form Feed.
   5. Beispiel:

```
ASSEMBLER1          128 Files    10.12.83

Nr.    Rec     Name       Len  Typ   Datum
001    0-018   MUSIC      002   L   28.11.82
002    0-020   PEEK       001   L   12.01.84
003    0-021   ADVIO      006   L   08.11.83
004    0-027   AUTOLOOP   003   L   14.01.84
005    0-030   FORMAT75   031   B   26.11.83
006    0-061   F75HELP    002   T   04.08.82
007    0-063   F75KEYS    001   T   15.11.82
008    0-064   F75PRMPT   002   T   15.11.82
009    0-066   PEEKPOKE   001   L   10.12.93
010    0-067   PRDIR      007   B   15.10.83
011    0-074   IOUTIL     005   L   27.01.00
012    0-079   PEKEPOOK   001   L   11.12.83
013    0-080   PRMEM      004   B   01.01.00
014    0-084   LEXIN      009   B   05.11.83
015    0-093   PRREC      009   B   04.11.83
016    0-102   LEXAN      009   B   04.11.83
017    0-111   PRPRO      007   B   01.01.00
018    0-118   PRIVAT     007   B   21.11.83

110 File(s) frei        387 Records frei
```

   Bedeutung der einzelnen Spalten:

Nr.	Fortlaufende Nummer der Einträge (Files)
Rec.	Spur- und Recordnummer des Fileanfangs zB beginnt das File 'PRDIR' auf Spur 0 und Record 67
Name.	Name, unter dem das File abgespeichert ist
Len.	Länge des Files in ganzen Records (1 Record = 256 Bytes)
Typ.	B BASIC T Text A Appt L LEX I LIF1
Datum.	Datum, an dem der File eingerichtet wurde

```
100 ! ** Directory ausdrucken **
110 INTEGER F,L,P,R,R1,R2,T,S,N,Y
120 DIM D$[8],E$[1],N$[8],R$[256],T$[1]
130 E$=CHR$(27)
140 ! - Rec 0 lesen -
150 SENDIO ':CA','unl,unt,lad#,ddl4',CHR$(0)&CHR$(0) @ GOSUB 510
160 F=NUM(R$[20,20]) @ R=510-F @ F=F*8
170 INPUT 'Bandkennzeichnung: '; B$ @ B$=UPRC$(B$)
180 IF LEN(B$)>15 THEN B$=B$[1,15]
190 ! - Drucker einstellen -
200 PWIDTH 75 @ PRINT E$&'!'&CHR$(8)
210 IMAGE 3x,17a,3d,' Files',3x,8a,/
220 ! - Ueberschrift drucken -
230 PRINT USING 210 ; B$,F,FND$(R$[37,39])
240 IMAGE 3x,6a,6a,9a,4a,5a,5a
250 PRINT USING 240 ; 'Nr.','Rec','Name','Len','Typ','Datum'
260 ! - Band auf Rec 2 positionieren -
270 SENDIO ':CA','lad#,ddl4',CHR$(0)&CHR$(2)
280 N,L=0
290 ! - Rec lesen -
300 IF MOD(L,8)=0 THEN GOSUB 510
310 ! - Rec in 8 Files aufteilen -
320 P=MOD(L,8)*32+1 @ F$=R$[P,P+31] @ L=L+1
330 ! - File zerlegen -
340 N$=F$[1,8] @ IF N$[1,1]=CHR$(255) THEN 460
350 T=NUM(F$[12,12]) @ IF T=0 THEN 300
360 S=NUM(F$[15,15])
370 R1=NUM(F$[16,16])
380 R2=NUM(F$[20,20])
390 D$=FND$(F$[21,23])
400 N=N+1 @ R=R-R2
410 ! - Fileausdruck -
420 PRINT E$&'!'&CHR$(2);
430 IMAGE 3x, 3z,2x,d,'-',3z,2x,9a,3z,2x,3a,8a
440 PRINT USING 430 ; N,S,R1,N$,R2,FNT$(T),D$
450 IF F#L THEN 300
460 ! - Schlusszeile -
470 PRINT E$&'!'&CHR$(8)
480 IMAGE 3x,3d,x,17a,3d,x,12a
490 PRINT USING 480 ; F-N,'File(s) frei',R,'Records frei
500 END
510 ! - Record holen -
520 R$=ENTIO$(':CA','unl,tad#,ddt2,ddt4,ddt1,sda')
530 RETURN
540 ! - Datum/Zeichenumwandlung(dez,hex)
550 DEF FND$(A$) = FNH$(A$[3,3])&'.'&FNH$(A$[2,2])&'.'&FNH$(A$[1,1])
560 DEF FNH$(Z$) = STR$(NUM(Z$)\16)&STR$(MOD(NUM(Z$),16))
570 ! - Filetyp -
580 DEF FNT$(Y)
590 FNT$='?'
600 IF Y=1 THEN FNT$='I'
610 IF Y=82 THEN FNT$='T'
620 IF Y=83 THEN FNT$='A'
630 IF Y=136 THEN FNT$='B'
640 IF Y=137 THEN FNT$='L'
650 END DEF
```

2. PRREC:
Das Programm 'PRREC' druckt Records und Files vom Band aus, und zwar wählbar in dez., hexadez. und/oder ASCII-Codierung. Es werden jeweils 32 Zeichen in eine Zeile gedruckt; damit umfaßt ein Record 8 Zeilen, jeweils in einer der oben genannten Codierungen. Auf diese art können Bandinhalte oder Fileinhalte direkt sichtbar gemacht werden. Dazu muß die LEX-Karte des Programmpaketes I/O-Utilities und der EPSON-Drucker FX80 vorhanden sein. Der Drucker (':pr') und das Cassetten-Laufwerk (':ca') müssen als IL-Geräte zugewiesen sein.

Programmbeschreibung:
1. Das Programm sowie die LEX-Karte 'HPILCMDS' einlesen.
2. Das Programm starten.
3.a) Ausdruck von Records:
Spur- und Recordnummer eingeben, von der ab ausgedruckt werden soll.
Anzahl der auszudruckenden Records eingeben.
b) Ausdruck von Files:
Bei der Abfrage 'Start bei Spur' RTN drücken
Bei der Abfrage 'Start bei Record' Filename eingeben
4. Bei der Abfrage 'Wahl: Dez,Hex,ASCII:' die Anfangsbuchstaben für die gewünschten Ausdruckformen eingeben.
5. Ausdruck erfolgt automatisch
6. Beispiel:
Ausdruck des Records 79 (Start bei Spur 0, Record 79, Anzahl 1, Wahl: HA)

Spur 0 Record 79

F7	86	77	00	8D	4C	E4	F0	D0	9B	50	45	4B	45	50	4F	4F	4B	14	40	0C	00	18	00	12	00	21	00	24	00	05	61
		w			L					P	E	K	E	P	O	O	K		@						!		$			a
26	00	55	00	05	61	3B	00	FF	FF	50	4F	4F	CB	50	45	4B	C5	FF	FF	41	17	9E	A1	98	50	CD	F1	60	CE	8B	3E
&		U			a	;				P	O	O		P	E	K				A					P			`			>
66	16	A3	10	C6	33	61	56	26	B6	00	80	9E	42	21	A1	06	E5	CE	6D	46	58	12	6C	A8	B4	42	06	E3	2D	A3	6F
f					3	a	V	&					B	!					m	F	X		l			B			–		o
0A	E2	6C	E5	9E	10	2D	98	50	CD	20	61	10	C6	33	61	5E	93	26	B4	00	80	CE	E4	FC	9E	CE	8B	3E	F4	06	66
		l				–		P			a			3	a	^		&										>			f
C9	FF	FF	FA	F4	CE	99	4C	59	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
							L	Y																							
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
 100 ! ** Records vom Band ausdrucken **
 110 INTEGER R,A,D,I,J,K,L,S,Z,M,N,P,Q
 120 DIM R$[256],E$[1],H$[128],P$[3],S$[4],D$[128],A$[64],R1$[16],F$[3
2]
 130 E$=CHR$(27) @ S$='     '
 140 ! - Eingaben -
 150 INPUT 'Start bei Spur : ',S$; S$ @ S=VAL(S$) @ S$='    '
 155 INPUT 'Start bei Record : ';R1$ @ R1$=UPRC$(R1$) @ FOR M=1 TO LEN
(R1$)-1
 156 IF NUM(R1$[M,M+1])<48 OR NUM(R1$[M,M+1])>57 THEN 570
 157 NEXT M @ R=VAL(R1$)
 160 INPUT 'Anzahl der Rec.: ';A
 170 ! - Drucker einstellen -
 180 PWIDTH 130 @ PRINT E$&'!'&CHR$(22);
 190 ! - Band positionieren -
 200 SENDIO ':ca','unl,lad#,ddl4',CHR$(S)&CHR$(R)
 210 ! - Beginn 1.Rec -
 215 DISP 'Wahl: '&CHR$(196)&'ez,'&CHR$(200)&'ex,'&CHR$(193)&'SCII: ';
 216 INPUT ''; Q$ @ Q$=UPRC$(Q$) @ D1,D2,D3=0
 217 IF POS(Q$,'D')#0 THEN D1=1
 218 IF POS(Q$,'H')#0 THEN D2=1
 219 IF POS(Q$,'A')#0 THEN D3=1
 220 FOR K=1 TO A
 230 ! - Spur- und Rec.Nr. lesen/drucken -
 240 P$=ENTIO$(':ca','unl,tad#,ddt3,sda')
 250 S=NUM(P$[1,1]) @ R=NUM(P$[2,2])
 260 PRINT 'Spur';S;' Record';R @ PRINT
 270 ! - Rec lesen -
 280 R$=ENTIO$(':ca','unl,tad#,ddt2,ddt4,ddt1,sda')
 290 ! - Rec aufspalten -
 300 FOR J=1 TO 256 STEP 32
 310 ! - Druckzeilen (dez,hex,ASCII) aufbauen -
 320 D$,H$,A$=''
 330 FOR I=J TO J+31
 340 D=NUM(R$[I,I])
 350 D$=D$&FNF$(STR$(D))
 360 H$=H$&FNF$(FNH1$(D))
 370 A$=A$&FNA$(D)&' '
 380 NEXT I
 390 ! - 3 Zeilen drucken -
 400 IF D1=1 THEN PRINT D$
 401 IF D2=1 THEN PRINT H$
 402 IF D3=1 THEN PRINT E$&'!'&CHR$(36)&' '&A$
 410 PRINT E$&'!'&CHR$(22)
 420 NEXT J
 430 PRINT
 440 NEXT K
 450 END
 460 ! - Zeichenumwandlung dez/hex -
 470 DEF FNH1$(D) = FNHO$(D\16)&FNHO$(MOD(D,16))
 480 DEF FNHO$(Z) = CHR$(Z+48+7*(Z>9))
```

```
490 ! - Ausdruck formatieren -
500 DEF FNF$(Z$)
510 L=4-LEN(Z$) @ FNF$=510 @ L=4-LEN(Z$) @ FNF$=S$[1,L]&Z$
520 END DEF
530 ! - ASCII-Zeichen -
540 DEF FNA$(D)
550 IF D>31 AND D<127 THEN FNA$=CHR$(D) ELSE FNA$=' '
560 END DEF
570 R1$=UPRC$(R1$)&'          '
580 R1$=R1$[1,8] @ SENDIO ':ca','unl,lad#,ddl4',CHR$(0)&CHR$(2)
590 N,Q=0
600 IF MOD(Q,8)=0 THEN R$=ENTIO$(':ca','unl,tad#,ddt2,ddt4,ddt1,sda')
610 P=MOD(Q,8)*32+1 @ F$=R$[P,P+31] @ Q=Q+1
620 IF F$[1,1]=CHR$(255) THEN 680
630 N$=F$[1,8] @ A=NUM(F$[20,20])
640 IF NUM(F$[12,12])=0 THEN 600
650 S=NUM(F$[15,15]) @ R=NUM(F$[16,16])
660 IF N$=R1$ THEN 180
670 N=N+1 @ IF A#Q THEN 600 ELSE Q=Q+1 @ GOTO 600
680 DISP 'File nicht gefunden' @ END
```

3. LEXAN:
   Das Programm 'LEXAN' analysiert ein LEX-File, das auf Magnetband gespeichert ist. Es werden Informationen, wie Länge, Befehle, Codierungen der Befehle und Fehlermeldungen ermittelt und ausgedruckt. Für die Ausführung dieses Programmes muß die LEX-Karte 'HPILCMDS' des Programmpaketes I/O-Utilities und der EPSON-Drucker FX80 vorhanden sein. Der Drucker (':pr') und das Cassettenlaufwerk (':ca') müssen als IL-Geräte zugeweisen sein.

   Programmbeschreibung:
   1. Das Programm sowei die LEX-Karte 'HPILCMDS' einlesen.
   2. Das Programm starten.
   3. Filename eingeben.
   4. Ausdruck der Informationen erfolgt automatisch.
   5. Beispiel:
      LEX-File 'PEEKPOKE'

```
PEEKPOKE              119Bytes     10.12.93


Befehle:
 180  21  64  1  POKE
 180  21  64  2  PEEK
Fehlermeldungen:
keine
```

LEX-File 'PEKEPOOK'

```
PEKEPOOK              119Bytes     11.12.83


Befehle:
 180  20  64  1  POOK
 180  20  64  2  PEKE
Fehlermeldungen:
keine
```

LEX-File 'IOUTIL' (gleich mit der LEX-Karte 'HPILCMDS')

```
IOUTIL                1252Bytes    27.01.00


Befehle:
 180  25  64  1  SENDIO
 180  25  64  2  ENTIO$
 180  25  64  3  SEND?
Fehlermeldungen:
 device:saat NBD
```

```
100 ! ** Analyse von LEX-Files **
110 INTEGER N,L,P,F,T,S,R,K,A,J,I
120 DIM E$[1],H$[200],S$[4],A$[8],A1$[200],F$[32],R$[256],N$[8]
130 E$=CHR$(27)
135 SENDIO ':ca','unl,lad#,ddl7','' @ R$=ENTIO$(':ca','unl,tad#,ddt2,
ddt4,ddt1,sda')
140 ! - Positionieren des Bandes auf Directory -
150 INPUT 'Programmname: ';A1$ @ A1$=UPRC$(A1$)&'          '
160 A$=A$[1,8] @ SENDIO ':ca','lad#,ddl4',CHR$(0)&CHR$(2)
170 N,L=0
180 ! - Rec lesen -
190 IF MOD(L,8)=0 THEN R$=ENTIO$(':ca','unl,tad#,ddt2,ddt4,ddt1,sda')
200 P=MOD(L,8)*32+1 @ F$=R$[P,P+31] @ L=L+1
210 IF F$[1,1]=CHR$(255) THEN 270
220 N$=F$[1,8] @ F=NUM(F$[20,20])
230 T=NUM(F$[12,12]) @ IF T=0 THEN 190
240 S=NUM(F$[15,15]) @ R=NUM(F$[16,16]) @ A=F
250 IF N$=A$ THEN 280
260 N=N+1 @ IF F#L THEN 190 ELSE L=L+1 @ GOTO 190
270 DISP 'File nicht gefunden' @ END
280 A1$=A$ @ A$=FNH1$(NUM(F$[23,23]))&'.'&FNH1$(NUM(F$[22,22]))&'.'
290 A$=A$&FNH1$(NUM(F$[21,21]))
300 ! - Band auf Programm positionieren -
310 SENDIO ':ca','unl,lad#,ddl4',CHR$(S)&CHR$(R)
320 ! - Rec lesen -
330 R$=ENTIO$(':ca','unl,tad#,ddt2,ddt4,ddt1,sda')
340 F=NUM(R$[4,4])*256+NUM(R$[3,3])
350 PWIDTH 75 @ PRINT E$&'!'&CHR$(8)
360 IMAGE 3x,17a,5d,'Bytes ',3x,8a,/
370 PRINT USING 360 ; A1$,F,A$
380 PRINT @ PRINT
390 PWIDTH 130 @ PRINT E$&'!'&CHR$(22);
400 PRINT 'Befehle:'
410 M=NUM(R$[30,30])*256+NUM(R$[29,29])+19
420 A2=NUM(R$[20,20]) @ A1=NUM(R$[19,19])
430 K=NUM(R$[24,24])*256+NUM(R$[23,23])+19
440 J=NUM(R$[28,28])*256+NUM(R$[27,27])+19
450 IF R$[K,K]=CHR$(255) THEN PRINT 'keine' @ GOTO 580
460 H$=R$[K,J] @ D=0
470 FOR I=1 TO POS(H$,CHR$(255))
480 IF NUM(H$[I,I])>128 THEN D=D+1
490 NEXT I
500 FOR I=1 TO D-1
510 PRINT 180;A1;A2;I;' ';
520 FOR B=1 TO POS(H$,CHR$(255))
530 IF NUM(H$[B,B])>127 THEN 550
540 NEXT B @ DISP 'Falsche Codierung !!' @ STOP
550 A1$=H$[1,B] @ A1$[B,B]=CHR$(NUM(A1$[B,B])-128)
560 PRINT A1$ @ H$=H$[B+1,LEN(H$)]
570 NEXT I
580 PRINT 'Fehlermeldungen:'
590 IF R$[J,J]=CHR$(255) THEN PRINT 'keine' @ STOP
600 H$=R$[J+1,M] @ D=0
605 IF NUM(H$[1,1])>127 THEN H$=H$[2,LEN(H$)]
607 H$=H$[1,POS(H$,CHR$(255))]
610 FOR I=1 TO LEN(H$)
620 IF NUM(H$[I,I])>128 THEN D=D+1
630 NEXT I
640 FOR I=1 TO D-1
650 FOR B=1 TO LEN(H$)
660 IF NUM(H$[B,B])>127 THEN 680
670 NEXT B @ DISP 'Falsche Codierung !!' @ STOP
680 A1$=H$[1,B] @ A1$[B,B]=CHR$(NUM(A1$[B,B])-128)
690 PRINT A1$ @ H$=H$[B,LEN(H$)]
700 NEXT I
710 PRINT @ PRINT
720 STOP
730 DEF FNH1$(D) = FNH0$(D\16)&FNH0$(MOD(D,16))
740 DEF FNH0$(Z) = CHR$(Z+48+7*(Z>9))
```

4. LEXIN:
Das Programm 'LEXIN' dient zum byteweise Abspeichern eines LEX-Files auf Magnetband. Dazu werden jeweils die Hex-Codes einer Zeile (=32 Hex-Codes) eingegeben. Für die Asuführung dieses Programmes muß die LEX-Karte 'HPILCMDS' des Programmpaketes I/O-Utilities und der Drucker FX80 von EPSON vorhanden sein. Der Drucker (':pr') und das Cassettenlaufwerk (':ca') müssen als IL-Geräte zugewiesen sein.

Programmbeschreibung:
1. Das Programm sowie die LEX-Karte 'HPILCMDS' einlesen.
2. Das programm starten.
3. Zeilen eingeben.
   Nach dem letzten Code als Hex-Code 'GG' eingeben.
4. LEX-File wird uaf Band gespeichert und kann dann in den Rechner geladen werden.
5. Beispiel:

   Eingabe des durch 'PRREC' ermittelten Listings:
   Eingabe der Zeilen:
   1: F78677008D4CE4F0D09B50454B45504F4F4B14400C00180012002100240005 61

   2: 2600550005613B00FFFF504F4FCB50454BC5FFFF41179EA19850CDF160CE8B3E

   3: 6616A310C633615626B600809E4221A106E5CE6D4658126CA8B44206E32DA36F

   4: 0AE26CE59E102D9850CD206110C633615E9326B40080CEE4FC9ECE8B3EF40666

   5: C9FFFFFAF4CE994C59FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

   6: GG

   Im Katalog des Laufwerks steht dann das File mit

```
 Name    Type Len    Time  Date
PEKEPOOK    B  117 19:51 04 11 83
```

   Mit COPY ':ca' TO 'PEKEPOOK' läßt sich das File in den Rechner laden.

```
100 ! ** LEX-Files vom Listing eingeben **
110 INTEGER Z9,N,L,P,P1,P2,P3,P5,S,P4,R9,I,J
120 DIM A$[32],R$[256],H$[8],C$[10],D$[70]
130 ! - Band positionieren -
140 Z9=0 @ R$=ENTIO$(':ca','unl,tad#,ddt2,ddt4,ddt1,sda')
150 SENDIO ':ca','unl,lad#,ddl4',CHR$(0)&CHR$(Z9)
160 N,L=0 @ A$=CHR$(255)&CHR$(255)
170 IF MOD(L,8)=0 THEN R$=ENTIO$(':ca','unl,tad#,ddt2,ddt4,ddt1,sda')
180 P=POS(R$,A$)
190 ! - Ermitteln des Programmrecords -
200 IF P=0 THEN Z9=Z9+1 @ GOTO 150
210 IF MOD(P,32)#0 THEN P=IP(P/32)+1 @ P=P*32
211 P1=P-48 @ IF P1>0 THEN 220 ELSE SENDIO ':ca','unl,lad#,ddl4',CHR$
(0)&CHR$(Z9-1)
212 R$=ENTIO$(':ca','unl,tad#,ddt2,ddt4,ddt1,sda') @ P1=P1+256
220 P2=NUM(R$[P1,P1])+NUM(R$[P1+4,P1+4])
230 S=NUM(R$[P1-1,P1-1]) @ R$='' @ P5=0 @ P3,J=1
240 SENDIO ':ca','lad#,ddl4',CHR$(S)&CHR$(P2)
250 ! - Eingabe des Listings -
260 DISP STR$(J)&':'; @ INPUT ''; D$ @ D$=UPRC$(D$)
270 IF LEN(D$)>64 THEN 260
275 FOR I=1 TO 63 STEP 2 @ H$=D$[I,I+1]
277 IF NUM(H$[1,1])>70 OR NUM(H$[2,2])>70 THEN P3=P3+1 @ GOTO 330
280 P5=P5+1 @ P4=(POS('0123456789ABCDEF',H$[1,1])-1)*16+POS('01234567
89ABCDEF',H$[2,2])-1
290 R$=R$&CHR$(P4)
300 IF P5>=256 THEN P3=P3+1 @ GOTO 305
301 NEXT I @ J=J+1 @ GOTO 260
305 SENDIO ':ca','lad#,ddl4',CHR$(S)&CHR$(P2+P3-2)
310 SENDIO ':ca','lad#,ddl2',R$
320 P5=0 @ R$='' @ J=J+1 @ GOTO 260
330 IF LEN(R$)=256 THEN 350
340 FOR R9=LEN(R$) TO 255 @ R$=R$&CHR$(255) @ NEXT R9
345 SENDIO ':ca','lad#,ddl4',CHR$(S)&CHR$(P2+P3-2)
350 SENDIO ':ca','lad#,ddl2',R$
360 SENDIO ':ca','lad#,ddl4',CHR$(S)&CHR$(P2)
370 R$=ENTIO$(':ca','unl,tad#,ddt2,ddt4,ddt1,sda')
380 C$=R$[11,18]&'  '
390 ! - Eintrag in Directory -
400 SENDIO ':ca','lad#,ddl4',CHR$(0)&CHR$(Z9)
410 R$=ENTIO$(':ca','unl,tad#,ddt2,ddt4,ddt1,sda')
420 A$=C$&CHR$(224)&CHR$(137)&CHR$(0)&CHR$(0)
430 A$=A$&CHR$(S)&CHR$(P2)&CHR$(0)&CHR$(0)&CHR$(0)&CHR$(P3)
440 H$=DATE$ @ P1=VAL(H$[1,1]*16+VAL(H$[2,2])
450 P2=VAL(H$[4,4])*16+VAL(H$[5,5])
460 P3=VAL(H$[7,7])*16+VAL(H$[8,8])
470 A$=A$&CHR$(P1)&CHR$(P2)&CHR$(P3)
480 H$=TIME$ @ P1=VAL(H$[1,1])*16+VAL(H$[2,2])
490 P2=VAL(H$[4,4])*16+VAL(H$[5,5])
500 P3=VAL(H$[7,7])*16+VAL(H$[8,8])
510 A$=A$&CHR$(P1)&CHR$(P2)&CHR$(P3)
520 A$=A$&CHR$(128)&CHR$(1)&'  '
530 R$[P-31,P]=A$
540 SENDIO ':ca','unl,lad#,ddl4',CHR$(0)&CHR$(Z9)
550 SENDIO ':ca','unl,lad#,ddl2',R$
560 STOP
```

5. DEVICES:
Das Programm 'DEVICES' dient dazu, die Anzahl von Peripherieeinheiten bestimmter Einheitsgruppen zu finden. Für die Ausführung dieses Programms muß die LEX-Karte 'HPILCMDS' des Programmpaketes I/O-Utilities vorhanden sein. Außerdem muß das Peripheriesystem in seiner Kennung bestimmte Richtlinien erfüllen.
Das 1.Zeichen in der ASSIGNIO-Zuweisung für jede Einheit gibt die Art der Einheit, das 2.Zeichen gibt an, die wievielte Einheit diese Einheit für die entsprehende Gruppe ist.

Controler	C
Massenspeicher	M
Printer	P
Display	D
Interface	I
Elektr. Instrument	E
Graphikeinheit	G

Programmbeschreibung:
1. Das Programm sowie die LEX-Karte 'HPILCMDS' einlesen.
2. Das Programm starten.
3. Es erscheint die Aufforderung 'ART: '
   Nun gibt man den Buchstaben ein:
   C für Controler
   M für Massenspeicher
   P für Printer
   D für Display
   I für Interface
   E für elektr. Instrumente
   G für Graphikeinheiten
   A für ALL
   Der Rechner bestimmt nun die Anzahl der spezifizierten Einheiten.
4. Beispiel:
   In der Schleife befinden sich 2 Druckereinheiten.
   Nach dem Starten des Programmes gibt man P für Printer ein.
   In der Anzeige steht nun '2 P-Einheiten'

   Nach einem erneuten Starten des Programmes gibt man A für ALL ein.
   In der Anzeige steht nun '2 A-Einheiten'

```
100 ! ** Bestimmung der Zahl von IL-Einheiten **
110 DIM R$[1],A$[30]
120 INTEGER R,A,I
190 ! - Eingabe der Einheitsart -
200 INPUT 'Art: ';A$ @ P=POS('CMPDIEGA',UPRC$(A$))
210 ! - Kontrolle der Zubehoerskennung -
220 I=1 @ A=0 @ IF P=0 THEN DISP 'falsche Art' @ BEEP @ GOTO 200
230 R$=ENTIO$('','UNL,UNT,TAD'&STR$(I)&',SAI')
240 R=NUM(R$) @ IF R=0 THEN 300
250 IF P=8 THEN A=A+1
260 IF R>=16*(P-1) AND R<=16*P-1 THEN A=A+1
270 I=I+1 @ GOTO 230
300 ! - Herstellung der Zustandsmeldung -
310 RESTORE @ FOR I=1 TO P @ READ A$ @ NEXT I
320 DISP 'IL-Schleife: '&STR$(A)&' '&A$
330 STOP
400 DATA Controller,Massenspeicher,Drucker,Display,Interface,E-Instru
mente
410 DATA Graphikeinheiten,Einheiten
```

6. PRMEM:

Das Programm 'PRMEM' dient dazu, das Betriebssystem des HP-75 auszudrucken zu lassen , um es zu analysieren. Der Adreßbereich, der ausgedruckt werden soll, läßt sich in den FOR..TO-Schleifen in den Zeilen 220 und 270 einstellen. Im Listing wird das gesamte Betriebssystem ausgedruckt. Zur Ausführung dieses Programmes muß die LEX-Karte 'PEEKPOKE' und der EPSON-Drucker FX80 vorhanden sein. Der Drucker (':pr') muß in der IL-Schleide vorhanden und deklariert sein.

Programmbeschreibung:

1. Das Programm und die LEX-Karte 'PEEKPOKE' einlesen. LEX-Karte 'PEKEPOOK' einlesen
2. Das Programm starten.
3. Adreßbereich in Zeile 220 auf volle Vielfache von 256 ändern.
   Adreßberich in Zeile 270 auf gewünschte Adresse ändern.
4. Bei der Abfrage 'Wahl: Dez,Hex,ASCII: ' die Anfangsbuchstaben für die gewünschten Ausdruckformen eingeben.
5. Der Ausdruck erfolgt automatisch.
6. Beispiel:
   Ausdruck von 0 bis 192 (Adresse 0000-00BF)
   Zeile 220: FOR K=0 TO 256
   Zeile 270: FOR J=1 TO 192 STEP 32
   Wahl: DHA

```
$0000
  18   0  37 161 119   2  32 132  10   4  40 132 238   5  48 132  56 132 152  88 169   0  32 179 252 255 178  71 255 178  72 255
  12  00  25  A1  77  02  20  84  0A  04  28  84  EE  05  30  84  38  84  98  58  A9  00  20  B3  FC  FF  B2  47  FF  B2  48  FF
      %       w                       (           0       8       X                           G           H
$0020
 177   0  96 201 227  28 246  12 177  22  96 137 246   6 177  32  96 139   4 163 178  73 255 178  70 255  68 169  13 122 161 206
  B1  00  60  C9  E3  1C  F6  0C  B1  16  60  89  F6  06  B1  20  60  8B  04  A3  B2  49  FF  B2  46  FF  44  A9  0D  7A  A1  CE
      `               `               `                   I       F       D           z
$0040
 147  44  96 169  99  97 108  99 112 114 111 103 206 214  31  74 177  79 130 206  45  31 206 103   0 206 186  81  64 177  97 130
  93  2C  60  A9  63  61  6C  63  70  72  6F  67  CE  D6  1F  4A  B1  4F  82  CE  2D  1F  CE  67  00  CE  BA  51  40  B1  61  82
      ,   `       c   a   l   c   p   r   o   g           J       O           -           g               Q   @       a
$0060
 179  95 130 179  91 130 158  96 177  68   0  80 169 226   0 206 120  32 126 169  64   2 179 190 131 146 178 189 131 147  77 146
  B3  5F  82  B3  5B  82  9E  60  B1  44  00  50  A9  E2  00  CE  78  20  7E  A9  40  02  B3  BE  83  92  B2  BD  83  93  4D  92
      _       [       `       D       P               x       ~       @                                       M
$0080
 206 236  80  80 147  28 183   2   0 158 206   0  65  78 147 206   4   1 248 245  78 144 247   3 206 157   0 240 240  79 144 244
  CE  EC  50  50  93  1C  B7  02  00  9E  CE  00  41  4E  93  CE  04  01  F8  F5  4E  90  F7  03  CE  9D  00  F0  F0  4F  90  F4
      P   P                           A   N                           N                           O
$00A0
  46 206  76  84  18 206  65  71 206 246  79 240  17  80 147   8 224 178  84 132 152 133  16 181  34 224 153 198   0   0  79 144
  2E  CE  4C  54  12  CE  41  47  CE  F6  4F  F0  11  50  93  08  E0  B2  54  84  98  85  10  B5  22  E0  99  C6  00  00  4F  90
   .      L   T          A   G          O           P               T                   "                           O
```

```
 100 ! ** Memory des Rechners ausdrucken **
 110 INTEGER R,A,D,I,J,K,L,S,Z,M,N,P,Q
 120 DIM R$[256],E$[1],H$[128],P$[3],S$[4],D$[128],A$[64],R1$[16],F$[3
2]
 130 E$=CHR$(27) @ S$='     '
 150 S$='    '
 170 ! - Drucker einstellen -
 180 PWIDTH 130 @ PRINT E$&'!'&CHR$(6);
 210 ! - Beginn 1.Rec -
 215 DISP 'Wahl: '&CHR$(196)&'ez,'&CHR$(200)&'ex,'&CHR$(193)&'SCII: ';
 216 INPUT ''; Q$ @ Q$=UPRC$(Q$) @ D1,D2,D3=0
 217 IF POS(Q$,'D')#0 THEN D1=1
 218 IF POS(Q$,'H')#0 THEN D2=1
 219 IF POS(Q$,'A')#0 THEN D3=1
 220 FOR K=0 TO 65535 STEP 256
 230 R$=''
 240 FOR J=K TO K+255
 250 IF J<32767 THEN R$=R$&CHR$(PEKE(J-32768)) ELSE R$=R$&CHR$(PEEK(J)
)
 260 NEXT J
 270 FOR J=1 TO 256 STEP 32
 280 S=K+J-1 @ P$='0123456789ABCDEF' @ B$=''
 290 R=1+MOD(S,16) @ B$=P$[R,R]&B$ @ S=S\16 @ IF S THEN 290
 300 FOR S=LEN(B$) TO 5 @ B$='0'&B$ @ NEXT S @ B$=B$[3,6] @ PRINT '$'&
B$
 310 ! - Druckzeilen (dez,hex,ASCII) aufbauen -
 320 D$,H$,A$=''
 330 FOR I=J TO J+31
 340 D=NUM(R$[I,I])
 350 D$=D$&FNF$(STR$(D))
 360 H$=H$&FNF$(FNH1$(D))
 370 A$=A$&FNA$(D)&' '
 380 NEXT I
 390 ! - 3 Zeilen drucken -
 400 IF D1=1 THEN PRINT D$
 401 IF D2=1 THEN PRINT H$
 402 IF D3=1 THEN PRINT E$&'!'&CHR$(36)&' '&A$
 410 PRINT E$&'!'&CHR$(6);
 420 NEXT J
 430 PRINT
 440 NEXT K
 450 END
 460 ! - Zeichenumwandlung dez/hex -
 470 DEF FNH1$(D) = FNH0$(D\16)&FNH0$(MOD(D,16))
 480 DEF FNH0$(Z) = CHR$(Z+48+7*(Z>9))
 490 ! - Ausdruck formatieren -
 500 DEF FNF$(Z$)
 510 L=4-LEN(Z$) @ FNF$=S$[1,L]&Z$
 520 END DEF
 530 ! - ASCII-Zeichen -
 540 DEF FNA$(D)
 550 IF D>31 AND D<127 THEN FNA$=CHR$(D) ELSE FNA$='.'
 560 END DEF
```

7. PRPRO:

Das Programm 'PRPRO' dient dazu, Programme und Files aus dem Memory des HP-75 auszudrucken, um deren Aufbau zu studieren. Es können sämtliche Filearten, sowie die Rechner-Programme 'iofile', 'devfile' und 'calcprogram' ausgedruckt werden. Zur Ausführung dieses Programms muß die LEX-Karte 'PEKEPOOK' und der EPSON-Drucker FX80 vorhanden sein. Der Drucker (':pr') muß in der IL-Schleife vorhanden und deklariert sein.

Programmbeschreibung:

1. Das Programm und die LEX-Karte 'PEKEPOOK' eingeben.
2. Das Programm starten.
3. Bei der Abfrage 'Programmname: ' den gewünschten Namen eingeben.
4. Bei der Abfrage 'Wahl: Dez,Hex,ASCII: ' die Anfangsbuchstaben für die gewünschten Ausdruckformen eingeben.
5. Der Ausdruck erfolgt automatisch.
6. Beispiel:
   Ausdruck des Files 'calcprog'
   Wahl: DHA

calcprog:

27	0	6	0	0	0	34	0	0	0	0	0	9	150	5	80	82	80	82	79	115	14	153	169	2	138	14	138	18	32	0	30
1B	00	06	00	00	00	22	00	00	00	00	00	09	96	05	50	52	50	52	4F	73	0E	99	A9	02	8A	0E	8A	12	20	00	1E
						"									P	R	P	R	O	s												

0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
00	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Ausdruck des Files 'devfile'
Wahl: DHA

devfile:

0	0	0	0	0	0	0	0	0	0	1	0	109	67	65	0	1	82	2	0	0	0	16	0	0	0	135	130	0	255	1	0
00	00	00	00	00	00	00	00	00	00	01	00	6D	43	41	00	01	52	02	00	00	00	10	00	00	00	87	82	00	FF	01	00
												m	C	A			R														

18	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
12	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

255	255	96	4	0	0	0	0	0	0	0	0	0	99	97	108	99	112	114	111	103	13	0	10	0	1	0	0	0	0	0	0
FF	FF	60	04	00	00	00	00	00	00	00	00	00	63	61	6C	63	70	72	6F	67	0D	00	0A	00	01	00	00	00	00	00	00
		`											c	a	l	c	p	r	o	g												

0	0	0	0	0	2	0	0	0	1	0	0	0	0	1	0	0	80	82	79	32	32	32	0	0	0	2	0	4	80	82	128
00	00	00	00	00	02	00	00	00	01	00	00	00	00	01	00	00	50	52	4F	20	20	20	00	00	00	02	00	04	50	52	80
																	P	R	O										P	R	

2	153	169	2	138	14	99	255	255	255	255
02	99	A9	02	8A	0E	63	FF	FF	FF	FF

```
100 ! ** Programm aus dem Memory ausdrucken **
110 INTEGER R,A,D,I,J,K,L,S,Z,M,N,P,Q
120 DIM R$[256],E$[1],H$[128],P$[3],S$[4],D$[128],A$[64],R1$[16],F$[32
]
130 E$=CHR$(27) @ S$='    '
140 ! - Eingaben -
150 INPUT 'Programmname: '; R1$ @ R1$=UPRC$(R1$) @ IF R1$='DEVFILE' OR
 R1$='IOFILE' THEN 570
152 IF R1$='CALCPROG' OR R1$='KEYS' OR R1$='APPT' THEN 570
155 I=1354 @ S$='    ' @ R$=R1$&S$&S$ @ R1$=R$[1,8]
156 IF PEEK(I)=0 THEN I=I+1 @ GOTO 156
157 R$=''
160 FOR J=I TO I+255 @ R$=R$&CHR$(PEEK(J)) @ NEXT J
170 P=POS(R$,R1$) @ IF P=0 THEN I=I+256 @ GOTO 160
175 A=NUM(R$[P-7,P-7])*256+NUM(R$[P-8,P-8])
180 P=NUM(R$[P-9,P-9])*256+NUM(R$[P-10,P-10])
190 ! - Drucker einstellen -
200 PWIDTH 130 @ PRINT E$&'!'&CHR$(6);
205 PRINT E$&'!'&CHR$(36)&R1$&';'&E$&'!'&CHR$(6)
210 ! - Beginn 1.Rec -
215 DISP 'Wahl: '&CHR$(196)&'ez,'&CHR$(200)&'ex,'&CHR$(193)&'SCII: ';
216 INPUT ''; Q$ @ Q$=UPRC$(Q$) @ D1,D2,D3=0
217 IF POS(Q$,'D')#0 THEN D1=1
218 IF POS(Q$,'H')#0 THEN D2=1
219 IF POS(Q$,'A')#0 THEN D3=1
220 FOR K=P TO P+A STEP 32 @ R$=''
230 FOR J=K TO K+31
240 IF J>P+A THEN R$=R$&CHR$(255) @ GOTO 260
250 R$=R$&CHR$(PEEK(J-32768))
260 NEXT J
310 ! - Druckzeilen (dez,hex,ASCII) aufbauen -
320 D$,H$,A$=''
330 FOR I=1 TO 32
340 D=NUM(R$[I,I])
350 D$=D$&FNF$(STR$(D))
360 H$=H$&FNF$(FNH1$(D))
370 A$=A$&FNA$(D)&' '
380 NEXT I
390 ! - 3 Zeilen drucken -
400 IF D1=1 THEN PRINT D$
401 IF D2=1 THEN PRINT H$
402 IF D3=1 THEN PRINT E$&'!'&CHR$(36);' ';A$
410 PRINT E$&'!'&CHR$(6);
440 NEXT K
450 END
460 ! - Zeichenumwandlung dez/hex -
470 DEF FNH1$(D) = FNH0$(D\16)&FNH0$(MOD(D,16))
480 DEF FNH0$(Z) = CHR$(Z+48+7*(Z>9))
490 ! - Ausdruck formatieren -
500 DEF FNF$(Z$)
510 L=4-LEN(Z$) @ FNF$=S$[1,L]&Z$
520 END DEF
530 ! - ASCII-Zeichen -
540 DEF FNA$(Z)
550 IF D>31 AND D<127 THEN FNA$=CHR$(D) ELSE FNA$='
560 END DEF
570 FOR J=1 TO LEN(R1$) @ R1$[J,J]=CHR$(NUM(R1$[J,J])+32) @ NEXT J
580 GOTO 153
```

8. SETTING:
   Das Programm 'SETTING' dient dazu, das System des HP-75 auf den Zustand bestimmter Statusgrößen hin zu überprüfen. Für die Ausführung des Programmes muß die LEX-Karte 'PEKEPOOK' vorhanden sein.

   Programmbeschreibung:
   1. Das Programm sowie die LEX-Karte 'PEKEPOOK' einlesen.
   2. Das Programm starten.
   3. Auf die Abfrage 'Groesse: ' eine der vier möglichen Größen eingeben:
      WIDTH
      DELAY
      PWIDTH
      MARGIN
   4. Der Rechner gibt daraufhin den Zustand der betreffenden Größe aus.
   5. Beispiel:
      Testen von PWIDTH.
      Zuerst setzt man den Zustand von PWIDTH auf 132 durch Ausführung von
      PWIDTH 132.
      Nach dem Starten des Programmes 'SETTING' gibt man auf die Abfrage 'Groesse:' hin PWIDTH ein.
      In der Anzeige erscheint nun 'PWIDTH=132'

```
100 ! ** Abfrage von Statusgroessen **
110 DIM A$[7]
120 SHORT I
130 INTEGER J
200 ! - Eingabe der Groesse -
210 INPUT 'Groesse: ';A$ @ A$=UPRC$(A$)
220 IF A$='PWIDTH' THEN 260
230 IF A$='WIDTH' THEN 280
240 IF A$='MARGIN' THEN 300
250 IF A$='DELAY' THEN 320 ELSE DISP 'falsche Groesse' @ BEEP @ WAIT
2 @ GOTO 200
260 ! - Unterprogramm fuer PWIDTH -
270 I=PEEK(716)-(PEEK(716)=0) @ GOTO 400
280 ! - Unterprogramm fuer WIDTH -
290 I=PEEK(715)-(PEEK(715)=0) @ GOTO 400
300 ! - Unterprogramm fuer MARGIN -
310 I=PEEK(944) @ GOTO 400
320 ! - Unterprogramm fuer DELAY -
330 I=0 @ FOR J=0 TO 3 @ I=I+PEEK(J+648)*256^J @ NEXT J
340 I=I/64
400 DISP A$; @ IF I<0 THEN DISP ' inf' ELSE DISP '='&STR$(I) @ STOP
```

9. ANLEX:
   Das Programm 'ANLEX' analysiert ein LEX-File, das sich im Memory des Rechners befindet. Es werden Informationen, wie Länge, Befehle, Codierungen der Befehle und Fehlermeldungen ermittelt und ausgedruckt. Für die Ausführung dieses Progrmmes muß die LEX-Karte 'PEKEPOOK' und der EPSON-Drucker FX80 vorhanden sein. Der Drucker (':pr') muß als IL-Gerät zugewiesen sein.

   Programmbeschreibung:
   1. Das Programm sowie die LEX-Karte 'PEKEPOOK' einlesen.
   2. Das Programm starten.
   3. Filename eingeben.
   4. Ausdruck der Informationen erfolgt automatisch.
   5. Beispiel:
      LEX-File 'PEKEPOOK'

```
PEKEPOOK              119Bytes     11.12.83


Befehle:
 180  20  64  1  POOK
 180  20  64  2  PEKE
Fehlermeldungen:
keine
```

LEX-File 'IOUTIL' (gleich mit der LEX-Karte 'HPILCMDS')

```
IOUTIL                1252Bytes    27.01.00


Befehle:
 180  25  64  1  SENDIO
 180  25  64  2  ENTIO$
 180  25  64  3  SEND?
Fehlermeldungen:
device sent NRD
```

LEX-File 'PEEKPOKE'

```
PEEKPOKE              119Bytes     10.12.93


Befehle:
 180  21  64  1  POKE
 180  21  64  2  PEEK
Fehlermeldungen:
keine
```

```
100 ! ** Analyse von LEX-Files im Memory des Rechners **
110 INTEGER N,L,P,F,T,S,R,K,A,J,I
120 DIM E$[1],A1$[200],A$[8],R$[256],F$[32],N$[8],H$[200],S$[4]
130 E$=CHR$(27)
140 ! - Positionieren des Bandes auf Directory -
150 INPUT 'Programmname: '; A1$ @ A1$=UPRC$(A1$)&'
160 A$=A1$[1,8]
170 I=1354 @ S$='    '
180 IF PEKE(I)=0 THEN I=I+1 @ GOTO 180
190 R$=''
200 FOR J=I TO I+255 @ R$=R$&CHR$(PEKE(J)) @ NEXT J
210 P=POS(R$,A$) @ IF P=0 THEN I=I+256 @ GOTO 190
220 A=NUM(R$[P-7,P-7])*256+NUM(R$[P-8,P-8])
230 P=NUM(R$[P-9,P-9])*256+NUM(R$[P-10,P-10])
240 R$=''
250 FOR J=P TO P+255 @ R$=R$&CHR$(PEKE(J-32768)) @ NEXT J
260 I=1
270 A1$=CAT$(I) @ IF POS(A1$,A$)=0 THEN I=I+1 @ GOTO 270
280 IF A1$[12,12]#'L' THEN DISP 'kein LEX-File' @ STOP
290 F=VAL(A1$[14,17])
300 A$=A1$[25,26]&'.'&A1$[28,29]&'.'&A1$[31,32]
310 A1$=A1$[1,8]
330 PWIDTH 75 @ PRINT E$&'!'&CHR$(8)
360 IMAGE 3x,17a,5d,'Bytes ',3x,8a,/
370 PRINT USING 360 ; A1$,F,A$
380 PRINT @ PRINT
390 PWIDTH 130 @ PRINT E$&'!'&CHR$(22);
400 PRINT 'Befehle:'
410 M=NUM(R$[12,12])*256+NUM(R$[11,11])+1
420 A2=NUM(R$[2,2]) @ A1=NUM(R$[1,1])
430 K=NUM(R$[6,6])*256+NUM(R$[5,5])+1
440 J=NUM(R$[10,10])*256+NUM(R$[9,9])+1
450 IF R$[K,K]=CHR$(255) THEN PRINT 'keine' @ GOTO 530
460 H$=R$[K,J] @ D=0
470 FOR I=1 TO POS(H$,CHR$(255))
480 IF NUM(H$[I,I])>128 THEN D=D+1
490 NEXT I
500 FOR I=1 TO D-1
510 PRINT 100;A1;A2;I;' ';
520 FOR B=1 TO POS(H$,CHR$(255))
530 IF NUM(H$[B,B])>127 THEN 550
540 NEXT B @ DISP 'Falsche Codierung !!' @ STOP
550 A1$=H$[1,B] @ A1$[B,B]=CHR$(NUM(A1$[B,B])-128)
560 PRINT A1$ @ H$=H$[B+1,LEN(H$)]
570 NEXT I
```

```
580 PRINT 'Fehlermeldungen:'
590 IF R$[J,J]=CHR$(255) THEN PRINT 'keine' @ STOP
600 H$=R$[J+1,M] @ D=0
605 IF NUM(H$[1,1])>127 THEN H$=H$[2,LEN(H$)]
607 H$=H$[1,POS(H$,CHR$(255))]
610 FOR I=1 TO LEN(H$)
620 IF NUM(H$[I,I])>128 THEN D=D+1
630 NEXT I
640 FOR I=1 TO D-1
650 FOR B=1 TO LEN(H$)
660 IF NUM(H$[B,B])>127 THEN 680
670 NEXT B @ DISP 'Falsche Codierung !!' @ STOP
680 A1$=H$[1,B] @ A1$[B,B]=CHR$(NUM(A1$[B,B])-128)
690 PRINT A1$ @ H$=H$[B,LEN(H$)]
700 NEXT I
710 PRINT @ PRINT
720 STOP
730 DEF FNH1$(D) = FNH0$(D\16)&FNH0$(MOD(D,16))
740 DEF FNH0$(Z) = CHR$(Z+48+7*(Z>9))
```

10. CONVERS:
    Das Programm 'CONVERS' dient dazu, Zahlen von einer Basis in eine andere zu verwandeln. Dieses Programm ist sehr nützlich bei der Entwicklung von eigenen LEX-Files, da hierbei dauernd zwischen Oktal- und Hexadezimalsystem gewechselt wird. Zur Ausführung dieses Programmes sind keine Hilfsmittel notwendig.

    Programmbeschreibung:
    1. Das Programm einlesen.
    2. Das Programm starten.
    3. Bei der Abfrage 'Basis 1, Basis 2, Zahl: ' Ausgangsbasis, Endbasis und Zahl eingeben.
    4. Der Rechner gibt automatisch die Zahl in Ausgangs- und Endbasis aus.
    5. a) Bei Drücken von RTN wird das Programm erneut gestartet.
       b) Bei Drücken von ATTN wird das Menue verlassen.
       Andere Tasten sind nicht wirksam.
    6. Beispiel:
       Umwandlung der Hexadezimalzahl OA in eine Oktalzahl:
       Eingabe bei der Abfrage: Ausgangsbasis 16
                                Endbasis   8
                                Zahl   OA
       Ausgabe: OA(16)=12(8)

       Umwandlung der Dualzahl 1000001 in eine Dezimalzahl:
       Eingabe bei der Abfrage: Ausgangsbasis 2
                                Endbasis   10
                                Zahl   1000001
       Ausgabe: 1000001(2)=65(10)

       Umwandlung der Dezimalzahl 64 in eine Oktalzahl:
       Eingabe bei der Abfrage: Ausgangsbasis 10
                                Endbasis   8
                                Zahl   64
       Ausgabe: 64(10)=100(8)

```
10 ! ** Zahlen-Basis-Wandler **
20 DIM A$,A0$[24]
30 INPUT 'Basis 1, Basis 2, Zahl: '; A,A0,A$ @ A$=UPRC$(A$) @ A1=0
40 FOR I=1 TO LEN(A$)
50 A2=NUM(A$[I,I])-64
60 IF A2<=0 THEN A2=A2+7
70 A2=A2+9 @ A1=A1*A+A2
80 NEXT I
90 A0$='' @ A2,A3=A1
100 A3=A3\A0 @ A2=RMD(A2,A0)-9
110 IF A2<=0 THEN A2=A2-7
120 A2=A2+64 @ A0$=CHR$(A2)&A0$
130 IF A3 THEN A2=A3 @ GOTO 100
140 DISP A$;'(';STR$(A);')=';A0$;'(';STR$(A0);')'
150 K$=UPRC$(KEY$)
160 IF K$='' THEN 150 ELSE K=NUM(K$)
170 IF K#13 THEN 150 ELSE 30
```

Fehlermeldungen

1. System-Fehler:
   - 18 ROM missing: Das benötigte LEX-File ist nicht vorhanden
   - 56 no loop response: Es fehlt die Initialisierung der IL-Schleife (ASSIGNIO oder RESTORE IO)
   - 93 mass mem error: Das Laufwerk war nicht bereit, Daten zu empfangen oder zu senden.
   - 94 no medium: Die Kassette war nicht initialisiert
   - 96 invalid medium: siehe Fehler 94

2. Fehler innerhalb der Programme:
   File nicht gefunden: Das angegeben File ist nicht vorhanden im Memory des Rechners

   Falsche Codierung !!: Befehle und Fehlermeldungen eines LEX-Files sind intern falsch codiert

   kein BASIC-File: Ein speziell auf BASIC-Files zugeschnittenes Programm wurde versucht, auf einen anderen Typ anzuwenden

ASCII Tabelle

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0									BS		LF			CR		
1						Ä	ä	Ö	ö	Ü	ü	ESC				
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z		\		^	
6		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z		|		~	
8																
9																
A																
B																
C																
D																
E																
F																

ANHANG D

Übersicht über die Codierungen der BASIC-Befehle

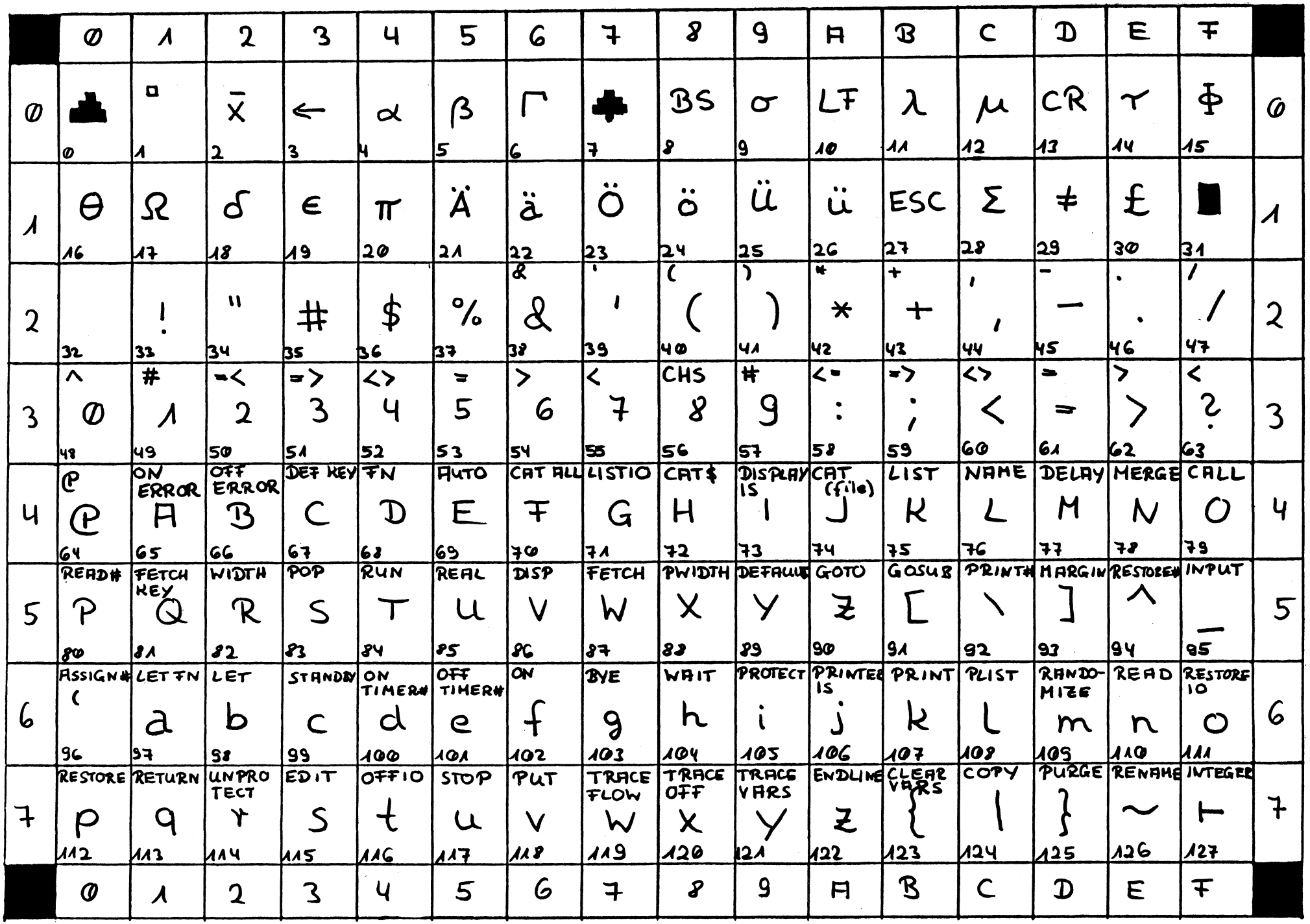
Hex-Code:	Dual-Code:	Befehl:
00	0000 0000	
01	0000 0001	angesprochene numerische Variable
02	0000 0010	angesprochenes Array
03	0000 0011	angesprochene ALPHA-Variable
04	0000 0100	REAL-Konstante
05	0000 0101	Text, der mit " eingeschlossen ist
06	0000 0110	Zweites Byte bei Kommentar und IMAGE bedeutet auch löschen bei DEF KEY
07	0000 0111	Zuweisung zu einer ALPHA-Variablen
08	0000 1000	Zuweisung zu einer numerischen Variablen
09	0000 1001	Ende Array-Variable mit 1 Index, der etwas zugewiesen wird
0A	0000 1010	Ende der Zuweisung einer Array-Variable mit 2 Indices
0B	0000 1011	Ende Array-Variable mit einem Index, die nur angesprochen wird
0C	0000 1100	Ende Array-Variable mit 2 Indices, die nur angesprochen wird
0D	0000 1101	
0E	0000 1110	Zeilenende
0F	0000 1111	
10	0001 0000	Ende bei ON ERROR und ON TIME
11	0001 0001	numerische Variable, der etwas zugewiesen wird
12	0001 0010	Array-Variable, der etwas zugewiesen wird
13	0001 0011	ALPHA-Variable, der etwas zugewiesen wird
14	0001 0100	Zuweisung zu mehreren numerischen Variablen
15	0001 0101	Zuweisung zu mehreren ALPHA-Variablen
16	0001 0110	Bei numerischen Funktionen in selbstdef. Funkt.
17	0001 0111	ALPHA-Funktionen in selbstdef. Funktionen
18	0001 1000	THEN + Sprungadresse
19	0001 1001	Ende von INPUT
1A	0001 1010	INTEGER-Konstante
1B	0001 1011	THEN + 2 Bytes
1C	0001 1100	ELSE + 2 Bytes
1D	0001 1101	Ende der Indices einer ALPHA-Variablen, bei der nur ein Anfangsindex gegeben wurde
1E	0001 1110	Ende der Indices einer ALPHA-Variablen, bei der zwei Anfangsindices gegeben wurden.
1F	0001 1111	ELSE + Sprungadresse
20	0010 0000	
21	0010 0001	
22	0010 0010	Abschluß von Zuweisungen ganzer Arrays bei PRINT#
23	0010 0011	
24	0010 0100	Abschluß von Zuweisungen ganzer Arrays bei READ
25	0010 0101	
26	0010 0110	&-Verknüpfung
27	0010 0111	; bei READ#, PRINT und DEF KEY
28	0010 1000	(
29	0010 1001	)
2A	0010 1010	*
2B	0010 1011	+
2C	0010 1100	,
2D	0010 1101	-
2E	0010 1110	.
2F	0010 1111	/

Hex-Code:	Dual-Code:	Befehl:
30	0011 0000	^
31	0011 0001	# bei ALPHA-Variablen
32	0011 0010	<= bei ALPHA-Variablen
33	0011 0011	>= bei ALPHA-Variablen
34	0011 0100	<> bei ALPHA-Variablen
35	0011 0101	= bei ALPHA-Variablen
36	0011 0110	> bei ALPHA-Variablen
37	0011 0111	< bei ALPHA-Variablen
38	0011 1000	CHS
39	0011 1001	# bei numerischen Variablen
3A	0011 1010	<= bei numerischen Variablen
3B	0011 1011	>= bei numerischen Variablen
3C	0011 1100	<> bei numerischen Variablen
3D	0011 1101	= bei numerischen Variablen
3E	0011 1110	> bei numerischen Variablen
3F	0011 1111	< bei numerischen Variablen
40	0100 0000	@
41	0100 0001	ON ERROR
42	0100 0010	OFF ERROR
43	0100 0011	DEF KEY
44	0100 0100	FN (Zuweisung bei einer Funktion)
45	0100 0101	AUTO
46	0100 0110	CAT ALL
47	0100 0111	LIST IO
48	0100 1000	CAT $
49	0100 1001	DISPLAY IS
4A	0100 1010	CAT (file)
4B	0100 1011	LIST
4C	0100 1100	NAME
4D	0100 1101	DELAY
4E	0100 1110	MERGE
4F	0100 1111	CALL
50	0101 0000	READ#
51	0101 0001	FETCH KEY
52	0101 0010	WIDTH
53	0101 0011	POP
54	0101 0100	RUN
55	0101 0101	REAL
56	0101 0110	DISP
57	0101 0111	FETCH
58	0101 1000	PWIDTH
59	0101 1001	DEFAULT
5A	0101 1010	GOTO
5B	0101 1011	GOSUB
5C	0101 1100	PRINT#
5D	0101 1101	MARGIN
5E	0101 1110	RESTORE#
5F	0101 1111	INPUT
60	0110 0000	ASSIGN#
61	0110 0001	LET FN
62	0110 0010	LET
63	0110 0011	STANDBY
64	0110 0100	ON TIMER#
65	0110 0101	OFF TIMER#
66	0110 0110	ON (goto, gosub)
67	0110 0111	BYE
68	0110 1000	WAIT
69	0110 1001	PROTECT
6A	0110 1010	PRINTER IS
6B	0110 1011	PRINT
6C	0110 1100	PLIST
6D	0110 1101	RANDOMIZE

Hex-Code:	Dual-Code:	Befehl:
6E	0110 1110	READ
6F	0110 1111	RESTORE IO
70	0111 0000	RESTORE
71	0111 0001	RETURN
72	0111 0010	UNPROTECT
73	0111 0011	EDIT
74	0111 0100	OFF IO
75	0111 0101	STOP
76	0111 0110	PUT
77	0111 0111	TRACE FLOW
78	0111 1000	TRACE OFF
79	0111 1001	TRACE VARS
7A	0111 1010	ENDLINE
7B	0111 1011	CLEAR VARS
7C	0111 1100	COPY
7D	0111 1101	PURGE
7E	0111 1110	RENAME
7F	0111 1111	INTEGER
80	1000 0000	SHORT
81	1000 0001	DELETE
82	1000 0010	ROM missing
83	1000 0011	REM
84	1000 0100	OPTION BASE
85	1000 0101	END DEF
86	1000 0110	DATA
87	1000 0111	DEF FN
88	1000 1000	DIM
89	1000 1001	RENUMBER
8A	1000 1010	END
8B	1000 1011	!
8C	1000 1100	FOR
8D	1000 1101	IF
8E	1000 1110	IMAGE
8F	1000 1111	NEXT
90	1001 0000	BEEP
91	1001 0001	
92	1001 0010	ASSIGN IO
93	1001 0011	CLEAR LOOP
94	1001 0100	CONT
95	1001 0101	CLEAR
96	1001 0110	Text folgt + 1 Byte Länge
97	1001 0111	TEXT
98	1001 1000	BASIC
99	1001 1001	LIF1
9A	1001 1010	RES
9B	1001 1011	INTO
9C	1001 1100	Beginn Funktionsteil von ERROR
9D	1001 1101	Beginn Funktionsteil von TIMER
9E	1001 1110	OR
9F	1001 1111	TO
A0	1010 0000	USING mit Reference-Nr.
A1	1010 0001	Schluß von READ bei einer ALPHA-Variablen
A2	1010 0010	Schluß von PRINT, erzeugt CR und LF
A3	1010 0011	; von DISP, PRINT nach Text
A4	1010 0100	, von DISP, PRINT bei ALPHA
A5	1010 0101	Nach letzter Variable bei PRINT#
A6	1010 0110	Zwischen Variablen bei PRINT#
A7	1010 0111	Schluß von PRINT#
A8	1010 1000	ON bei Befehlen
A9	1010 1001	OFF bei Befehlen
AA	1010 1010	IP
AB	1010 1011	EPS

Hex-Code:	Dual-Code:	Befehl:
AC	1010 1100	FP
AD	1010 1101	CEIL
AE	1010 1110	MAX
AF	1010 1111	Abschluß bei einzeiligen Funktionen
B0	1011 0000	SQR
B1	1011 0001	MIN
B2	1011 0010	MEM
B3	1011 0011	ABS
B4	1011 0100	externe BASIC-Befehle (System-ROM)
B5	1011 0101	Zuweisung eines ganzen ARRAYs bei Feld mit 1 In.
B6	1011 0110	Zuweisung eines ganzen ARRAYs bei Feld mit 2 In.
B7	1011 0111	SGN
B8	1011 1000	KEY$
B9	1011 1001	COT
BA	1011 1010	CSC
BB	1011 1011	APPT
BC	1011 1100	EXP
BD	1011 1101	INT
BE	1011 1110	LOG10
BF	1011 1111	LOG
C0	1100 0000	VER$
C1	1100 0001	SEC
C2	1100 0010	CHR$
C3	1100 0011	STR$
C4	1100 0100	LEN
C5	1100 0101	NUM
C6	1100 0110	VAL
C7	1100 0111	INF
C8	1100 1000	Schluß von READ bei numerischen Variablen
C9	1100 1001	PI
CA	1100 1010	UPRC$
CB	1100 1011	USING
CC	1100 1100	THEN
CD	1100 1101	TAB
CE	1100 1110	STEP
CF	1100 1111	EXOR
D0	1101 0000	NOT
D1	1101 0001	DIV
D2	1101 0010	ERRN
D3	1101 0011	ERRL
D4	1101 0100	CARD
D5	1101 0101	AND
D6	1101 0110	KEYS
D7	1101 0111	ELSE
D8	1101 1000	SIN
D9	1101 1001	COS
DA	1101 1010	TAN
DB	1101 1011	TO
DC	1101 1100	
DD	1101 1101	, bei der Zuweisung zu numerischen Variablen in einer INPUT-Anweisung
DE	1101 1110	[
DF	1101 1111	]
E0	1110 0000	\
E1	1110 0001	POS
E2	1110 0010	DEG
E3	1110 0011	RAD
E4	1110 0100	FLOOR
E5	1110 0101	, bei der Zuweisung zu ALPHA-Variablen in eir INPUT-Anweisung
E6	1110 0110	

Hex-Code:	Dual-Code:	Befehl:
E7	1110 0111	; zwischen numerischen Variablen bei PRINT und bei DISP
E8	1110 1000	, bei numerischen Variablen bei PRINT und DISP
E9	1110 1001	
EA	1110 1010	
EB	1110 1011	
EC	1110 1100	
ED	1110 1101	
EE	1110 1110	
EF	1110 1111	
F0	1111 0000	
F1	1111 0001	
F2	1111 0010	
F3	1111 0011	
F4	1111 0100	
F5	1111 0101	
F6	1111 0110	
F7	1111 0111	
F8	1111 1000	
F9	1111 1001	
FA	1111 1010	
FB	1111 1011	
FC	1111 1100	
FD	1111 1101	
FE	1111 1110	
FF	1111 1111	


0 1 2 3 4 5 6 7 8 9 A B C D E F
0 ■ □ X̄ ← α β Γ ♣ BS σ LF λ μ CR τ Φ 0
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
1 Θ Ω δ ∈ π Ä ä Ö ö Ü ü ESC Σ ≠ £ ■ 1
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
2 ! " # $ % & ' ( ) * + , — . / 2
48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
3 ^ 0 # 1 =< 2 => 3 <> 4 = 5 > 6 < 7 CHS 8 # 9 <= : => ; <> < = > > ? < 3
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
4 @ @ ON ERROR A OFF ERROR B DEF KEY C FN D AUTO E CAT ALL F LISTIO G CAT$ H DISPLAY IS I CAT (file) J LIST K NAME L DELAY M MERGE N CALL O 4
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
5 READ# P FETCH KEY Q WIDTH R POP S RUN T REAL U DISP V FETCH W PWIDTH X DEFAULT Y GOTO Z GOSUB [PRINT# \ MARGIN] RESTORE# ^ INPUT _ 5
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
6 ASSIGN# (LET FN a LET b STANDBY c ON TIMER# d OFF TIMER# e ON f BYE g WAIT h PROTECT i PRINTER IS j PRINT k PLIST l RANDOMIZE m READ n RESTORE IO o 6
112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
7 RESTORE p RETURN q UNPROTECT r EDIT s OFF IO t STOP u PUT v TRACE FLOW w TRACE OFF x TRACE VARS y ENDLINE z CLEAR VARS { COPY | PURGE } RENAME ~ INTEGER ⊢ 7
0 1 2 3 4 5 6 7 8 9 A B C D E F

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
8	SHORT  128	DELETE □ 129	ROM missing X 130	REM ← 131	OPTION BASE α 132	END DEF β 133	DATA Γ 134	DEF FN  135	DIM ≃ 136	RENUMBER σ 137	END ⊢ 138	! λ 139	FOR μ 140	IF  141	IMAGE Υ 142	NEXT Φ 143	8
9	BEEP Θ 144	Ω 145	ASSIGN IO δ 146	CLEAR LOOP ∈ 147	CONT π 148	CLEAR Ä 149	ä 150	TEXT Ö 151	BASIC ö 152	LIFA Ü 153	RES ü 154	INTO E. 155	Σ 156	≠ 157	OR £ 158	TO  159	9
A	USING _ 160	! 161	" 162	# 163	$ 164	% 165	& 166	' 167	ON (168	OFF) 169	IP * 170	EPS + 171	FP , 172	CEIL - 173	MAX . 174	/ 175	A
B	SQR Ø 176	MIN 1 177	MEM 2 178	ABS 3 179	externe Befehle 4 180	5 181	6 182	SGN 7 183	KEY$ 8 184	COT 9 185	CSC : 186	APPT ; 187	EXP < 188	INT = 189	LOG10 > 190	LOG ? 191	B
C	VER$ P 192	SEC A 193	CHR$ B 194	STR$ C 195	LEN D 196	NUM E 197	VAL F 198	INT G 199	H 200	PI I 201	UPRC$ J 202	USING K 203	THEN L 204	TAB M 205	STEP N 206	EXOR O 207	C
D	NOT P 208	DIV Q 209	ERRN R 210	ERRL S 211	CARD T 212	AND U 213	KEYS V 214	ELSE W 215	SIN X 216	COS Y 217	TAN Z 218	TO [219	\ 220	] 221	^ 222	_ 223	D
E	` c 224	POS a 225	DEG b 226	RAD c 227	FLOOR d 228	e 229	f 230	g 231	h 232	i 233	j 234	k 235	l 236	m 237	n 238	o 239	E
F	p 240	q 241	r 242	s 243	t 244	u 245	v 246	w 247	x 248	y 249	z 250	{ 251	| 252	} 253	~ 254	⊢ 255	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	 0	ALARM □ 1	LOCK $\overline{x}$ 2	DEG ← 3	RAD α 4	β 5	Γ 6	 7	BS 8	σ 9	LF 10	λ 11	μ 12	CR 13	Υ 14	Φ 15	0
1	Θ 16	Ω 17	δ 18	∈ 19	π 20	Ä 21	ä 22	Ö 23	ö 24	Ü 25	ü 26	ESC 27	Σ 28	≠ 29	£ 30	 31	1
2	32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	− 45	. 46	/ 47	2
3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63	3
4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79	4
5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95	5
6	‘ 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111	6
7	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	← 127	7
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

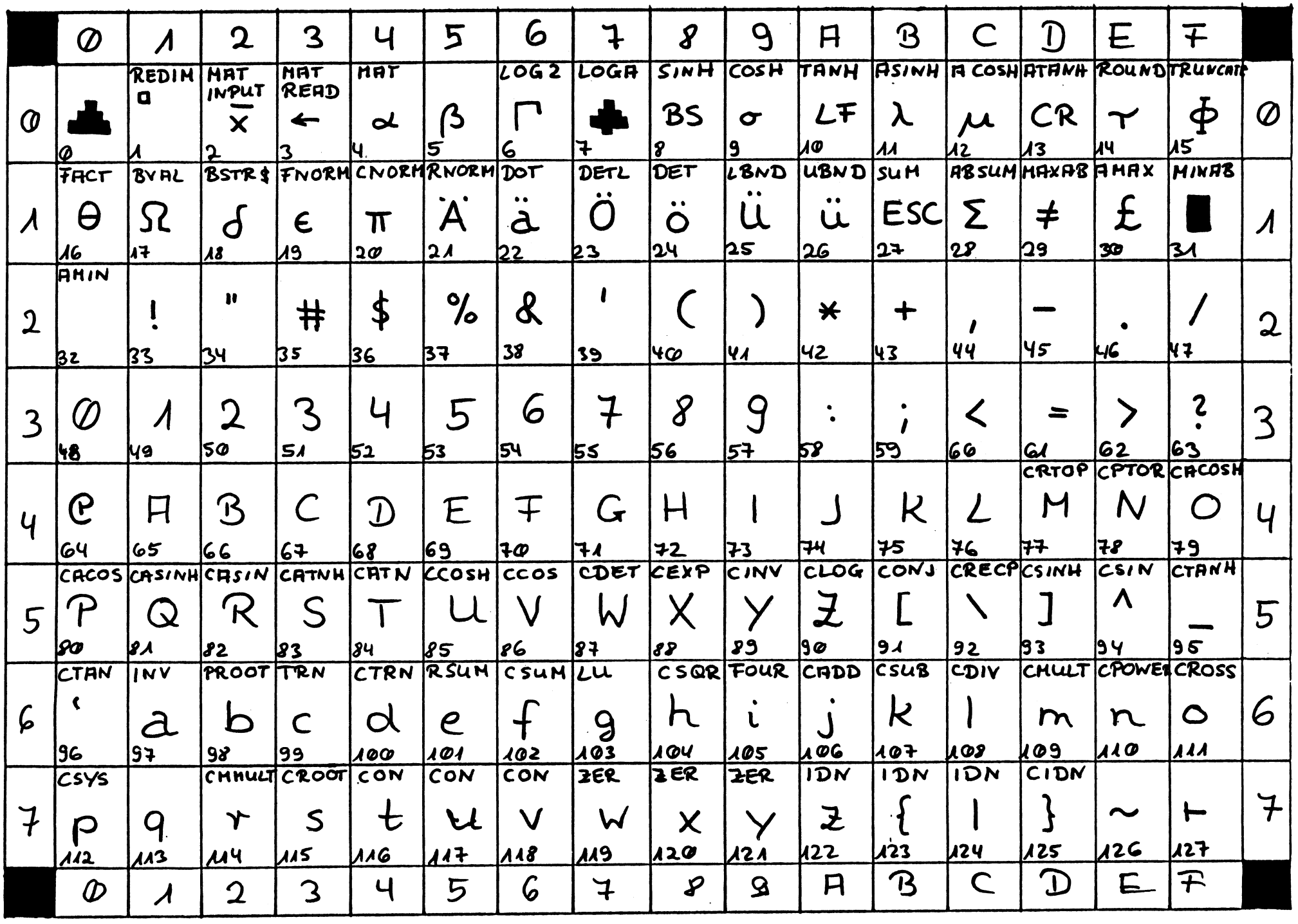
ALTROM

ID = 01 00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	 0	TRANSFORM 1	PACK 2	INITIALISE 3	TIME$ 4	DATE$ 5	TIME 6	DATE 7	ANGLE 8	ACOS 9	ATN 10	ASIN 11	RND 12	MOD 13	RND 14	15	0
1	Θ 16	Ω 17	δ 18	∈ 19	π 20	Ä 21	ä 22	Ö 23	ö 24	Ü 25	ü 26	ESC 27	Σ 28	≠ 29	£ 30	31	1
2	32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	− 45	. 46	/ 47	2
3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63	3
4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79	4
5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95	5
6	‘ 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111	6
7	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	⊢ 127	7
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

MEL ROM

ID = 02 00


MATHROM 1
ID = 03 20

MATHROM 1

ID= 0320

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
8	MAT ×  128	MAT +  129	MAT CHS  130	MAT −  131	α 132	MAT × (Skalar) β 133	Γ 134	135	~ 136	σ 137	⊢ 138	λ 139	μ 140	141	⊤ 142	Φ 143	8
9	Θ 144	Ω 145	δ 146	∈ 147	π 148	Ä 149	ä 150	Ö 151	ö 152	Ü 153	ü 154	E꜀ 155	Σ 156	≠ 157	£ 158	159	9
A	160	! 161	" 162	# 163	$ 164	% 165	& 166	' 167	(168	) 169	* 170	+ 171	, 172	= 173	. 174	/ 175	A
B	0 176	1 177	2 178	3 179	4 180	5 181	6 182	7 183	8 184	9 185	: 186	; 187	< 188	= 189	> 190	? 191	B
C	@ 192	A 193	B 194	C 195	D 196	E 197	F 198	G 199	H 200	I 201	J 202	K 203	L 204	M 205	N 206	O 207	C
D	P 208	Q 209	R 210	S 211	T 212	U 213	V 214	W 215	X 216	Y 217	Z 218	[219	¥ 220	] 221	^ 222	_ 223	D
E	‘ 224	a 225	b 226	c 227	d 228	e 229	f 230	g 231	h 232	i 233	j 234	k 235	l 236	m 237	n 238	o 239	E
F	p 240	q 241	r 242	s 243	t 244	u 245	v 246	w 247	x 248	y 249	z 250	{ 251	| 252	} 253	~ 254	⊢ 255	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

MATHROM 2

ID = 17 20

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		□ 1	FNGUESS IVALUE IBOUND INTEGR FNROOT !													
1	Θ 16	Ω 17	¡ 18	← 3	α 4	ß 5	Γ 6		BS 8	σ 9	LF 10	λ 11	μ 12	CR 13	↑ 14	Φ 15
2		! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	' 44	- 45	. 46	/ 47
3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	↑ 94	← 95
6	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
7	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	⊥ 127
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

ANHANG E

Übersicht über die Assemblerbefehle:

Hex-Code:	Dual-Code:	Befehl:
	00(AR)	ARP AR
01	0000 0001	ARP *
	01(DR)	DRP DR
41	0100 0001	DRP *
80	1000 0000	ELB DR
81	1000 0001	ELM DR
82	1000 0010	ERB DR
83	1000 0011	ERM DR
84	1000 0100	LLB DR
85	1000 0101	LLM DR
86	1000 0110	LRB DR
87	1000 0111	LRM DR
88	1000 1000	ICB DR
89	1000 1001	ICM DR
8A	1000 1010	DCB DR
8B	1000 1011	DCM DR
8C	1000 1100	TCB DR
8D	1000 1101	TCM DR
8E	1000 1110	NCB DR
8F	1000 1111	NCM DR
90	1001 0000	TSB DR
91	1001 0001	TSM DR
92	1001 0010	CLB DR
93	1001 0011	CLM DR
94	1001 0100	ORB DR, AR
95	1001 0101	ORM DR, AR
96	1001 0110	XRB DR, AR
97	1001 0111	XRM DR, AR
98	1001 1000	BIN
99	1001 1001	BCD
9A	1001 1010	SAD
9B	1001 1011	DCE
9C	1001 1100	ICE
9D	1001 1101	CLE
9E	1001 1110	RTN
9F	1001 1111	PAD
A0	1010 0000	LDB DR, AR
A1	1010 0001	LDM DR, AR
A2	1010 0010	STB DR, AR
A3	1010 0011	STM DR, AR
A4	1010 0100	LDBD DR, AR
A5	1010 0101	LDMD DR, AR
A6	1010 0110	STBD DR, AR
A7	1010 0111	STMD DR, AR
A8	1010 1000	LDB DR, =literal
A9	1010 1001	LDM DR, =literal
AA	1010 1010	STB DR, =literal
AB	1010 1011	STM DR, =literal
AC	1010 1100	LDBI DR, AR
AD	1010 1101	LDMI DR, AR
AE	1010 1110	STBI DR, AR
AF	1010 1111	STMI DR, AR

Hex-Code:	Dual-Code:	Befehl:
B0	1011 0000	LDBD DR, =label
B1	1011 0001	LDMD DR, =label
B2	1011 0010	STBD DR, =label
B3	1011 0011	STMD DR, =label
B4	1011 0100	LDBD DR, XAR, label
B5	1011 0101	LDMD DR, XAR, label
B6	1011 0110	STBD DR, XAR, label
B7	1011 0111	STMD DR, XAR, label
B8	1011 1000	LDBI DR, =label
B9	1011 1001	LDMI DR, =label
BA	1011 1010	STBI DR, =label
BB	1011 1011	STMI DR, =label
BC	1011 1100	LDBI DR, XAR, label
BD	1011 1101	LDMI DR, XAR, label
BE	1011 1110	STBI DR, XAR, label
BF	1011 1111	STMI DR, XAR, label
C0	1100 0000	CMB DR, AR
C1	1100 0001	CMM DR, AR
C2	1100 0010	ADB DR, AR
C3	1100 0011	ADM DR, AR
C4	1100 0100	SBB DR, AR
C5	1100 0101	SBM DR, AR
C6	1100 0110	JSB XR, label
C7	1100 0111	ANM DR, AR
C8	1100 1000	CMB DR, =literal
C9	1100 1001	CMM DR, =literal
CA	1100 1010	ADB DR, =literal
CB	1100 1011	ADM DR, =literal
CC	1100 1100	SBB DR, =literal
CD	1100 1101	SBM DR, =literal
CE	1100 1110	JSB =label
CF	1100 1111	ANM DR, =literal
D0	1101 0000	CMBD DR, =label
D1	1101 0001	CMMD DR, =label
D2	1101 0010	ADBD DR, =label
D3	1101 0011	ADMD DR, =label
D4	1101 0100	SBBD DR, =label
B5	1101 0101	SBMD DR, =label
D6	1101 0110	
D7	1101 0111	ANMD DR, =label
D8	1101 1000	CMBD DR, AR
D9	1101 1001	CMMD DR, AR
DA	1101 1010	ADBD DR, AR
DB	1101 1011	ADMD DR, AR
DC	1101 1100	SBBD DR, AR
DD	1101 1101	SBMD DR, AR
DE	1101 1110	
DF	1101 1111	ANMD DR, AR

Hex-Code:	Dual-Code:	Befehl:
E0	1110 0000	POBD DR, +AR
E1	1110 0001	POMD DR, +AR
E2	1110 0010	POBD DR, -AR
E3	1110 0011	POMD DR, -AR
E4	1110 0100	PUBD DR, +AR
E5	1110 0101	PUMD DR, +AR
E6	1110 0110	PUBD DR, -AR
E7	1110 0111	PUMD DR, -AR
E8	1110 1000	POBI DR, +AR
E9	1110 1001	POMI DR, +AR
EA	1110 1010	POBI DR, -AR
EB	1110 1011	POMI DR, -AR
EC	1110 1100	PUBI DR, +AR
ED	1110 1101	PUMI DR, +AR
EE	1110 1110	PUBI DR, -AR
EF	1110 1111	PUMI DR, -AR
F0	1111 0000	JMP label
F1	1111 0001	JNO label
F2	1111 0010	JOD label
F3	1111 0011	JEV label
F4	1111 0100	JNG label
F5	1111 0101	JPS label
F6	1111 0110	JNZ label
F7	1111 0111	JZR label
F8	1111 1000	JEN label
F9	1111 1001	JEZ label
FA	1111 1010	JNC label
FB	1111 1011	JCY label
FC	1111 1100	JLZ label
FD	1111 1101	JLN label
FE	1111 1110	JRZ label
FF	1111 1111	JRN label

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
8	ELB DR  128	ELM DR  129	ERB DR 130	ERM DR  131	LLB DR  132	LLM DR  133	LRB DR 134	LRM DR 135	ICB DR 136	ICM DR  137	DCB DR  138	DCM DR 139	TCB DR 140	TCM DR  141	NCB DR  142	NCM DR  143	8
9	TSB DR  144	TSM DR  145	CLB DR  146	CLM DR  147	ORB DR, AR  148	ORM DR, AR  149	XRB DR, AR 150	XRM DR, AR  151	BIN 152	BCD  153	SHD  154	DCE 155	ICE 156	CLE  157	RTN 158	PAD  159	9
A	LDB DR, AR 160	LDM DR, AR  161	STB DR, AR  162	STM DR, AR  163	LDBD DR, AR  164	LDMD DR, AR  165	STBD DR, AR  166	STMD DR, AR  167	LDB DR, =literal 168	LDM DR, =literal 169	STB DR, =literal 170	STM DR, =literal 171	LDBI DR, AR 172	LDMI DR, AR 173	STBI DR, AR 174	STMI DR, AR 175	A
B	LDBD DR, $ label 176	LDMD DR, $ label 177	STBD DR, $ label 178	STMD DR, $ label 179	LDBD DR, XR, $label 180	LDMD DR, XR, $label 181	STBD DR, XR, $label 182	STMD DR, XR, $label 183	LDBI DR, $ label 184	LDMI DR, $ label 185	STBI DR, $ label 186	STMI DR, $ label 187	LDBI DR, XR, $label 188	LDMI DR, XR, $label 189	STBI DR, XR, $label 190	STMI DR, XR, $label 191	B
C	CMB DR, AR 192	CMM DR, AR 193	ADB DR, AR 194	ADM DR, AR 195	SBB DR, AR 196	SBM DR, AR 197	JSB XR, $label 198	ANM DR, AR 199	CMB DR, =literal 200	CMM DR, =literal 201	ADB DR, =literal 202	ADM DR, =literal 203	SBB DR, =literal 204	SBM DR =literal 205	JSB $label 206	ANM DR, =literal 207	C
D	CMBD DR, $label 208	CMMD DR, $label 209	ADBD DR, $label 210	ADMD DR, $label 211	SBBD DR, $label 212	SBMD DR, $label 213	 214	ANMD DR, $label 215	CMBD DR, AR 216	CMMD DR, AR 217	ADBD DR, AR 218	ADMD DR, AR 219	SBBD DR, AR 220	SBMD DR, AR 221	 222	ANMD DR, AR 223	D
E	POBD DR, +AR 224	POMD DR, +AR 225	POBD DR, -AR 226	POMD DR, -AR 227	PUBD DR, +AR 228	PUMD DR, +AR 229	PUBD DR, -AR 230	PUMD DR, -AR 231	POBI DR, +AR 232	POMI DR, +AR 233	POBI DR, -AR 234	POMI DR, -AR 235	PUBI DR, +AR 236	PUMI DR, +AR 237	PUBI DR, -AR 238	PUMI DR, -AR 239	E
F	JMP =literal 240	JNO =literal 241	JOD =literal 242	JEV =literal 243	JNG =literal 244	JPS =literal 245	JNZ =literal 246	JZR =literal 247	JEN =literal 248	JEZ =literal 249	JNC =literal 250	JCY =literal 251	JLZ =literal 252	JLN =literal 253	JRZ =literal 254	JRN =literal 255	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

$	Befehl	PC	DR	AR	OV	CY	NG	LZ	ZR	RZ	OD	DC	E	BKP1	BKP2	PTR1	PTR2	Register	Kommentar

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								