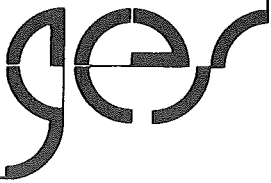
RL-BASIC

für NDR-Klein-Computer

V 2.4

Graf Elektronik Systeme GmbH



Sehr geehrter Kunde,

Sie erhalten mit dieser Diskette RL-BASIC 2.4, das Sie durch Eingabe von BASIC aus JADOS heraus aufrufen koennen. Es wird dann die Datei RL-BASIC.OVL in den Hauptspeicher geladen und gestartet.
In der ausgelieferten Version befindet sich unter RL-BASIC.OVL eine Version, die auf allen Prozessoren der 68Øxx-Baureihe laeuft.
Um die spezielle Version fuer 68Ø2Ø / 68881 starten zu koennen, muessen Sie RL-BASIC.OVL in RL-BASIC.68K und RL-BASIC.FPU in RL-BASIC.OVL umbenennen.

Alle Beispielprogramme sind als ASCII-Dateien auf der Diskette abgelegt. Sie koennen diese aus BASIC heraus durch NEW <cr> MERGE "Datelename.lst"<cr> aufrufen.

Falls Sie im Besitz einer frueheren BASIC-Version (vor 2.4) sein sollten, so koennen alte BASIC-Programme mittels dem Programm KONVERT.LST in eine ASCII-Darstellung ueberfuehrt werden, die dann ebenfalls durch MERGE ".." eingelesen werden kann.

Viel Erfolg und Spass beim Programmieren mit RL-BASIC

Wuenscht Ihnen Ihr

Rolf Lobreyer

P.S.:

Anmerkungen zu diesem Handbuch RL-BASIC Version 2.4

In der Vorbemerkung zu BASIC V 2.4 lesen Sie die neuesten Änderungen gegenüber den vorhergehenden Versionen.

Anschliessend wird der neue Programmeditor vorgestellt.

Die dann beschriebenen BASIC-Befehle und -Funktion sind in der Version 2.4 ausschliesslich neu.

Die Anhänge bis D sind gegenüber dem Handbuch V 2.0 modifiziert, die weiteren sind in dieser Version neu.

Es folgt ein Systemhandbuch und der Teil für Referenzkarten des RL-BASIC V 2.4.

Wir wünschen Ihnen Viel Freude beim Umgang mit RL-BASIC und sind für Anregungen jederzeit dankbar.

Ihr GES Team

Vorbemerkung zur BASIC-Version 2.4

Ein Jahr ist nun wieder ins Land gegangen, das ich genutzt habe RL-BASIC zu verbessern und zu erweitern. Es liegt Ihnen nun eine im Befehlsumfang vergrößerte und auch schnellere Version vor. Die Codegröße hat sich von ca. 24 Kbyte auf nun mehr 36 Kbyte erhöht, dies liegt zum einen daran, daß neue Funktionen und Anweisungen hinzugekommen sind, aber auch daran, daß bereits vorhandene verbessert wurden.

RL-BASIC ist mit allen Prozessorkarten der 680xx-Baureihe einsetzbar. Es werden 2 Varianten des Interpreters angeboten :

1. RL-BASIC für alle Prozessoren (incl. 68020)
2. RL-BASIC für 68020 mit Unterstützung der FPU 68881

RL-BASIC sollte nun auch in der Verarbeitungsgeschwindigkeit einiges zugelegt haben. Ein Grund hierfür dürfte die Verwaltung der Symbole, d.h. aller Programmvariablen und Sprungmarken, darstellen. Bisher wurde intensiver Gebrauch von der im Grundprogramm eingebauten Symboltabelle gemacht. Dies hatte jedoch einige Nachteile für den Bedienungskomfort und die Geschwindigkeit des RL-BASIC's, die nun verbessert wurden :

Variablennamen dürfen beliebig (naja) lang sein und werden in allen Zeichen unterschieden. (früher 5 Zeichen)

Der Zugriff auf den Wert einer Variable hat sich durch Implementierung einer Hashtabelle um ca 30% beschleunigt. Dies wirkt sich zum Beispiel bei der Auswertung von arithmetischen Ausdrücken, in denen viele Variable vorkommen, besonders aus.

RL-BASIC hat nun auch einen weiteren Schritt in Richtung strukturierter Programmierung unternommen. Es gibt nun nicht nur das Schleifenkonstrukt WHILE .. WEND, sondern auch das von PASCAL bekannte REPEAT .. UNTIL. Wer dennoch weiterhin mit GOTO programmieren möchte, kann nun auf den Komfort von Programmarken zurückgreifen. Zeilennummern hinter GOTO sollten vermieden werden, da durch die Verwendung von Programmarken, das BASIC-Programm auch wesentlich leichter zu lesen ist.

Es hat sich gezeigt, daß das BASIC auch viel bei Steuerungs- und Meßaufgaben eingesetzt wird. Um hier den Komfort zu erhöhen, wurden weitere Anweisungen integriert, die es ermöglichen Analog-Digitalwandler- und Digital-Analogwandler Karten direkt aus einem BASIC-Programm heraus anzusprechen. Der Zugriff auf die serielle Schnittstelle wurde auch geschaffen, sodaß nun einfach Daten mit anderen Rechnern oder Geräten ausgetauscht werden können.

In punkto Grapfik hat sich auch einiges getan :

Die COL256-Graphikkarte kann nun auch unter **RL-BASIC** angesprochen werden. Es können so ziemlich alle Programme mit Graphikausgabe auch für die COL256 verwendet werden, mit dem Unterschied, daß hier 256 Farben bzw. Graustufen zur Verfügung stehen. Die einzige Einschränkung liegt in der Verwendung des PAGE-Befehls. Da die COL256-Karte standardmäßig mit 64KByte bestückt ist, kann nur eine Bildschirmseite mit einer Auflösung von 256 * 256 dargestellt werden. **RL-BASIC** ignoriert nun einfach den PAGE-Befehl, sobald er für die COL256 gedacht ist. Die virtuelle Auflösung, die bei **RL-BASIC** schon bei der GDP64K Graphik verwendet wurde, ist auch hier gültig, d.h. alle Koordinatenwerte liegen in einem Bereich von Ø..511 und werden auf die reale Auflösung von 256 * 256 bzw 512 * 256 abgebildet. Ein Problem stellt die Darstellung von Text auf der COL-Karte dar. Um hier mit der GDP64K-Testdarstellung mithalten zu können wäre eine Auflösung von mindestens 512 Punkten in der Horizontalen erforderlich. Es können mit der COL256 in **RL-BASIC** zwar auch 8Ø Zeichen je Zeile dargestellt werden, was aber unleserlich wird. Die Zeichengröße entspricht der Einstellung bei Verwendung der GDP64K-Textausgabe. Es sind aber auch alle anderen Schriftgrößen verfügbar.

Da die COL256-Karte noch einige weitere Möglichkeiten bietet, die über den Möglichkeiten der GDP64K-Karte hinausreichen, wurden noch weitere Anweisungen und Funktionen integriert, die diese unterstützen.

Standardmäßig erfolgt die Graphikausgabe über die GDP64K-Karte. Durch einen BASIC-Befehl kann diese auf die COL256-Karte umgeleitet werden.

Eine weitere umfassende Neuerung liegt in der Möglichkeit, Daten auf Diskette zu speichern oder von dort zu laden. Neue Befehle zur Manipulation von Dateien wurden speziell hierfür in den Befehlssatz mit aufgenommen. Bei allen Anweisungen wurde strikt darauf geachtet, daß ähnliche bereits in anderen BASIC-Dialekten zu den gleichen Ergebnissen führen. So können auch BASIC-Programme anderer Rechner leicht an RL-BASIC adaptiert werden. Als Betriebssystem, das die Dateiverwaltung übernimmt, liegt **JADOS** zu Grunde. Das bedeutet für Sie, daß Sie im Besitz von **JADOS** sein sollten, um diese neuen Befehle auch einsetzen zu können. Auf eine Erweiterung von RL-DOS wurde verzichtet, um ein einheitliches Betriebssystem für den **NDR-Klein-Computer** zu unterstützen.
Unter **JADOS** können nun alle Arten von Daten auf Diskette abgespeichert oder von Diskette geladen werden. Momentan werden allerdings nur sequentielle Dateien unterstüzt, wobei man nicht auf Textdateien beschränkt ist. Es können zum Beispiel auch Programmdateien eingelesen werden, um sie dann per CALL-Anweisung von einem BASIC-Porgramm aufzurufen. Eine weitere Anwendung wäre z.B. das Ablegen von Messwerten, die durch die Analog-Digitalwandlerkarten aufgenommen wurden ...

Sie sehen RL-BASIC kann für die unterschiedlichsten Aufgaben sinnvoll eingesetzt werden.

Alles weitere erfahren Sie in den nachfolgenden Kapiteln.

Triberg im April 1987

Der Programmeditor

Die Eingabe eines Programmes erfordert oftmals das Abändern von Programmzeilen, die fehlerhaft eingegeben wurden.
RL-BASIC enhält hierzu einen leistungsfähigen Bildschirmeditor, der das Ändern von Programmzeilen vereinfacht.
Grundsätzlich ist es nämlich nur möglich einzelne Programmzeilen, insofern diese Fehler enthalten, ganz neu einzugeben.

Bildschirmorientiert heißt hier, daß alles was auf dem Bildschirm steht, so auch als Programm gespeichert ist.
Mit **LIST** kann ein Bereich auf den Bildschirm ausgelistet werden, der dann, mit Hilfe der schon vom Texteditor des Grundprogramms bekannten Cursortasten, verändert werden kann.
Der Programmeditor wurde in der Version 2.4 erheblich überarbeitet. Es können nun leicht auch Programmzeilen mit mehr als 80 Zeichen pro Zeile eingegeben werden. Das Übernehmen kann ab jeder Position innerhalb einer Programmzeile erfolgen. Programmzeilen sind auf 246 Zeichen beschränkt. Zur besseren Übersicht (Struktur) sollte jedoch nur eine Anweisung je Programmzeile verwendet werden, sodaß 246 Zeichen/Zeile auf jeden Fall ausreichen dürften.

Die Cursor-Richtungstasten sind wie im Texteditor in Form eines Achsenkreuzes angeordnet :

```
   E                                          Oben
S    D        steht für die Richtungen  Links      Rechts.
   X                                          Unten
```

Der Cursor wird durch drücken von *CONTROL* und einer dieser Tasten in die gewünschte Richtung bewegt. Hierbei ist zuerst die *CONTROL* dann der nachfolgende Buchstaben gemeinsam niederzudrücken.

Steht der Cursor auf der gewünschten Zeile, so kann mit *CONTROL-D* die fehlerhafte Position angefahren werden.
Wortweises Springen ist mit *CONTROL-F* nach rechts und *CONTROL-A* nach links möglich.

Das Ändern kann folgendermaßen unterteilt werden :

1. Löschen eines / mehrere Buchstaben erfolgt durch *CONTROL-G*.
2. Das Einfügen einer Leerstelle für weitere Eingaben erfolgt durch mehrmaliges Betätigen von *CONTROL-V*

3. Überschreiben eines bereits vorhanden Textes durch Eingabe des neuen Textes anstelle des Fehlerhaften.

Nach Beendigung einer Änderung muß die *RETURN*-Taste gedrückt werden, um die Änderung wirksam zu machen !!

Dies wären eigentlich schon die wichtigsten Funktionen des Editors.
Weitere nützliche Funktionen sind unten aufgelistet :

ESC-A	Wie *CTRL-E* Cursor eine Zeile höher.
ESC-B	Wie *CTRL-X* Cursor eine Zeile tiefer.
ESC-C	Wie *CTRL-D* Cursor eine Position nach rechts.
ESC-D	Wie *CTRL-S* Cursor eine Position nach links.
CONTROL-A	Springen zum Wort links vom Cursor
CONTROL-F	Springen zum Wort rechts vom Cursor
BACKSPACE	Löschen des zuletzt eingegebenen Zeichens bzw. das Zeichen vor dem Cursor wird gelöscht.
DEL	Löschen des Zeichens rechts vom Cursor
CONTROL-G	Löschen des Zeichens rechts vom Cursor Programmzeile rückt nach.
CONTROL-V	Ein Leer-zeichen rechts neben dem Cursor einfügen Die Programmzeile wird nachgeführt.
CONTROL-T	Löscht den Rest der Zeile.
CONTROL-N	Zeile einfügen
CONTROL-Y	Zeile löschen
CONTROL-Z	Löschen des Bildschirms und Cursor links oben positionieren.
ESC-H	Cursor nach links oben, ohne Bildschinrm löschen.
ESC-J	Wie CTRL-Z Bildschirm löschen.
CONTROL-C	Zeile ignorieren, Rest der Zeile wird gelöscht und der Cursor eine Zeile tiefer positioniert.
CONTROL-B	Warmstart !

Diese Cursorsteuerung ist während des **INPUT** - Befehls abgeschaltet,
was durch einen stehenden Cursor angezeigt wird !

C L O S E

Syntax : CLOSE [#<Dateinummer>]

Effekt : Schliesst eine oder alle zuvor geöffnete(n) Datei(en).

Bemerkung : Dieser Befehl beendet den Zugriff auf eine Datei. <Dateinummer> bezeichnet die Nummer unter der die Datei früher geöffnet wurde. Wird das **CLOSE** - Kommando, ohne Angabe der Dateinummmer aufgerufen, so werden alle Dateien geschlossen.

Nachdem eine Datei geschlossen wurde, kann sie unter einer anderen Nummer, oder derselben Nummer wieder geöffnet werden.

Alle Dateien werden automatisch nach einem **END,CLEAR**, oder **NEW**-Befehl geschlossen.
Nach einem Programmabbruch, bleiben die bis dahin geöffneten Dateien geöffnet. Durch ändern einer Programmzeile, werden auch alle Dateien geschlossen.

(Siehe auch **OPEN, INPUT# , PRINT#**)

G E T A D 8

Syntax : GETAD8(<Kanalnummer>)

Effekt : Liest einen digitalisierten Wert von der Baugruppe AD8*16 ein

Ergebniswert: Integer,Real

Bemerkung : Kanalnummer sollte im Bereich von Ø..15 liegen.
Es werden allerdings auch Werte darüber oder darunter akzeptiert, allerdings dann auf eine Kanalnummer zwischen Ø und 15 abgebildet.
Die vom Wandler zurückgelieferten Daten liegen in einem Werte-Bereich von Ø..255.

Befindet sich keine AD8*16 - Karte im System, so 'hängt' das Programm und kann nur noch durch ein RESET abgebrochen werden. Deshalb Vorsicht bei der Verwendung dieses Befehls ohne vorhandene AD8*16 Karte.

Beispiel :

```
5  PAGE 3,3 : CLPG
1Ø FOR X= Ø TO 511
2Ø Y = GETAD8(Ø)
3Ø PSET X,Y*2
4Ø NEXT
```

Zeichnet eine Kurve auf den Bildschirm ,die dem Spannungs-verlauf am Analog-wandler entspricht.

G E T A D 1 Ø

Syntax : GETAD1Ø

Effekt : Liest einen digitalisierten Wert von der Baugruppe AD1Ø*1 ein

Ergebniswert: Integer,Real

Bemerkung : Im Gegensatz zum Befehl **GETAD8** muss keine Kanalnummer angegben werden, da nur ein Eingangskanal zur Verfügung steht.

Die von der Baugruppe AD1Ø*1 gelieferten Werte überstreichen einen Wertebereich von Ø..1Ø23 was einer Genauigkeit von 1Ø-Bit entspricht. D.h. alle eingelesenen Werte liegen in diesem Bereich.

Befindet sich keine AD1Ø*1 - Karte im System, so sind die zurückgelieferten Werte willkürlich. Das Programm 'hängt' sich jedoch nicht auf, wie dies beim **GETAD8**-Befehl möglich ist, falls keine AD1Ø*1-Karte auf den BUS aufgesteckt ist.

Beispiel :

```
5  PAGE 3,3 : CLPG
1Ø FOR X= Ø TO 511
2Ø Y = GETAD1Ø
3Ø PSET X,Y/2 : REM durch 2 Teilen da sonst Ueberlauf auftritt
4Ø NEXT
```

Zeichnet eine Kurve auf den Bildschirm ,die dem Spannungsverlauf am Analog-wandler entspricht.

I N P U T

Syntax : INPUT #<Dateinummer>,<Variable>[,<Variable>] ...

Effekt : Diese Anweisung wird benutzt, um Daten aus einer sequentiellen Datei in Variablen einzulesen.

Bemerkung : Die sequentielle Datei muss, bevor Daten mittels **INPUT #** eingelesen werden können, zuvor mit dem OPEN-Befehl geöffnet werden. Die nach einem **Input #** spezifizierte <Dateinummer> ist diesselbe, wie sie bei der Eingabe des OPEN-Befehl verwendet wurde.
Es können sowohl Text als auch Ganzzahl und Gleitkommazahl-variablen bei einem INPUT # - Befehl eingesetzt werden.

Das Ende einer Eingabe ist durch ein Carriage-Return gekennzeichnet, d.h. wird <CR>-<LF> von einer sequentiellen Datei gelesen, wird der **INPUT #** Befehl beendet. Bei Falscher Eingabe wird das Programm mit einer entsprechenden Fehlermeldung abgebrochen.
Wird z.B. versucht von einer Datei zu lesen, die zum Schreiben geöffnet wurde, so erfolgt ebenso eine Fehlermeldung wie, wenn eine falsche Dateinummer angegeben wurde, zu der keine Datei geöffnet wurde.

Gegenspieler von **INPUT #** ist der Befehl **PRINT #**, mit dem Daten in eine Datei ausgegeben werden können.

Beispiel:
```
10 REM LESE TEXT-DATEI EIN UND GEBE SIE AUF DEM BILDSCHIRM AUS
20 INPUT "DATEINAME :";NAM$
30 OPEN "I",#1,NAM$
40 INPUT #1,A$
50 PRINT A$
60 IF EOF(1) THEN CLOSE #1:END
70 GOTO 40
```

M E R G E

Syntax : MERGE <Dateiname>

Effekt : Bindet ein, als Text abgespeichertes, BASIC-Programm an das im Speicher befindliche Programm an.

Bemerkung : Dieser Befehl ist nur im Zusammenhang mit einem Diskettenbetriebssystem, das sequentielle Dateien verwalten kann, anzuwenden. Unter JADOS ist dies z.B. gewährleistet.
Das BASIC-Programm muß vor Verwendung des **MERGE**-Befehls mit SAVE <Dateiname>,A als Textdatei abgespeichert worden sein.
Achtung ! Zeilenummern, die sowohl im aktuellen BASIC-Programm, als auch in dem dazuzubindenden Programm vorkommen, werden durch das "neue" Programm überschrieben.

So kann der **MERGE**-Befehl als Einfügen von Programmzeilen von Diskette in ein sich im Speicher befindliches Programm angesehen werden.

Merge kann nur im Direktmodus verwendet werden.

Beispiel :
```
SAVE "Program.lst",A        ; Speichert das Programm als Text ab.
NEW

10 PRINT "Start"
1000 PRINT "Ende"
MERGE "Program.lst"         ; Das Programm "Program.lst" wird an
                              das vorhandene angehängt.
```

O P E N

Syntax : OPEN "<Modus>",#<Dateinummer>,<Dateiname>

Effekt : Öffnet eine Sequentielle Datei zum Lesen oder Beschreiben.

Bemerkung : Bei diesem Befehl ist <Modus> ein Zeichenketten-Ausdruck dessen 1. Zeichen eines der folgenden darstellt :
O Spezifiziert eine sequentielle Ausgabe-Datei.
I Spezifiziert eine sequentielle Eingabe-Datei.

Alle Modi sind für Disketten-Dateien gültig, bei Geräte-Dateien ist nicht immer Lesen und Schreiben möglich.
Als Geräte Dateien sind CON: für Konsole
LST: für Drucker
SER: für serielle Schnittstelle
CAS: für Kassetten-Recorder
vordefiniert. "LST:" darf nur beschriben werden.
<Dateinummer> ist eine ganzzahlige Zahl <> 0, unter der die Datei nach dem Öffnen durch Datei-Ein-/Ausgabebefehle angesprochen wird.
<Dateiname> ist ein Name, der den Konventionen des gerade verwendeten Betriebssystems entspricht.

Eine Datei muß vor irgend einem Ein-Ausgabebefehl, der die Datei betrifft, göffnet werden.

Beispiel :
```
10 OPEN "O",#1,"test.txt"
20 INPUT "Name ?";NA$
30 PRINT #1,NA$
40 INPUT "Telefonnummer ?";TEL$
50 PRINT #1,TEL$
60 CLOSE #1
```
Liest einen Namen, eine Telefonnummer ein und schreibt diese dann in die Datei "test.txt"

Beispiel 2:
```
10 OPEN "I",#1,"test.txt"
20 INPUT #1,NA$
30 PRINT NA$
40 INPUT #1,TEL$
50 PRINT TEL$
60 CLOSE #1
```
Öffnet die Datei "test.txt" und gibt den Name sowie die Telefonnummer auf den Bildschirm aus.

O U T

Syntax : OUT #<Dateinummer>,<Wert>

Effekt : OUT gibt ein Zeichen (Byte) im Bereich von ASCII ØØ..$FF auf die Datei mit der Dateinummer .. aus.

Bemerkung : OUT dient in Verbindung mit der **INP()**-Funktion dazu einzelne Zeichen(Bytes) in eine Datei zu schreiben.
Es können alle Werte im Bereich Ø bis 255 auf eine Datei geschrieben werden. Da als Datei auch die Gerätedatei LST: angegeben werden kann, ist es auch möglich Graphiken an einen graphikfähigen Drucker als Byte-Folge zu senden.
Eine weitere Anwendung ist zum Beispiel die Ablage von Messwerten aus den AD-Wandlerkarten auf Datei. Im Gegensatz zu **PRINT #..** und **INPUT #..** werden die Datenbytes nicht in eine lesbare Form konvertiert, sondern werden als Byte-Strom auf die zuvor zu öffnende Datei geschrieben. Auch ASCII $00 kann somit in einer Datei vorkommen !

Beispiel:

```
1Ø REM Nur in Verbindung mit COL256-Karte
2Ø REM Lege ein Bild auf Diskette ab
3Ø GRMODE 1
4Ø OPEN "O",#1,"DEMO.PIC"
5Ø FOR IX = Ø TO 511 STEP 2
6Ø FOR IY = Ø TO 511 STEP 2
7Ø OUT #1,POINT(IX,IY)
8Ø NEXT
9Ø NEXT
1ØØ CLOSE #1
11Ø END
```

Speichert ein Bild auf Diskette ab.

P A I N T

Syntax : PAINT (<X-Koordinate>,<Y-Koordinate>),<Farbe>

Effekt : Füllt einen Bildschirmbereich mit der aktuellen Zeichenfarbe ab Position x,y bis die Grenzfarbe <Farbe> erreicht wird.

Bemerkung : Dieser Befehl ist nur in Verbindung mit der COL256-Baugruppe funktionsfähig. Die Grenze bis zu der ein Bildschirmausschnitt ausgefüllt wird durch <Farbe> markiert.
Die Füllfarbe wird durch den **COLOR**-Befehl vorgeschrieben.
Durch **PAINT** ist es möglich eine beliebig geformte Fläche einzufärben.

Beispiel:

```
10 RANDOMIZE 511
20 REPEAT
30 C = C + 1 : COLOR = C
40 X1 = RND
50 Y1 = RND
60 X2 = RND
70 Y2 = RND
80 GOSUB BOX!
90 PAINT ((X1+X2)/2),((Y1+Y2)/2),C
100 UNTIL INKEY$<>""
110 END
200 BOX!
210 LINE (X1,Y1)-(X2,Y1)-(X2,Y2)-(X1,Y2)-(X1,Y1)
220 RETURN
```

Zeichnet zufällige ausgefüllte Rechtecke

P R I N T

Syntax : PRINT #<Dateinummer>,[<Liste von Ausdrücken>]

Effekt : Dieser Befehl wird zur Ausgbae von Daten in eine sequentielle Datei verwendet.

Bemerkung : <Dateinummer> ist die Nummer unter der die Datei geöffnet wurde. Alle Daten werden in derselben Weise in eine Datei ausgebeben, wie sie auch auf den Bildschirm ausgegeben würden. Deshalb ist Vorsicht geboten, falls die ausgegebenen Daten auch wieder mittels **INPUT #** eingelesen werden sollen. D.h. Numerische Ausdrücke sollten mit "," getrennt werden, oder besser gleich durch ein Zeilenvorschub (<CR>-<LF>).

Beispiel : PRINT #1,<Zeichenkette>
PRINT #1,<Numerischer Ausdruck>;",";<Numerischer Ausdruck>
können durch INPUT #1,A$ bzw. durch INPUT #1,I,J wieder einglesen werden.

R E L A I S

Syntax : RELAIS ON ¦ OFF

Effekt : Schaltet das Relais der Kasetten-Schnittstelle an oder aus.

Bemerkung : Dieser Befehl kann dazu verwendet werden, bei Dateizugriffen auf Kasettenrecorder das Relais für die Motorsteuerung ein oder auszuschalten.

Wird eine Datei 'CAS:' geöffnet, wird das Motor-Relais nicht automatisch eingeschaltet. Dies muß mit diesem Befehl erfolgen.

Beispiel :

```
10 OPEN "O",#1,"CAS:"
20 RELAIS ON
30 FOR I = 1 TO 100
40 OUT #1, I
50 NEXT
60 CLOSE #1
70 RELAIS OFF
```

Gibt 100 Bytes auf die Kasettenschnittstelle aus.

R E P E A T . . U N T I L

Syntax : Repeat
.
.
UNTIL <Bool'scher Ausdruck>

Effekt : Wiederholt die Anweisungen innerhalb des **REPEAT..UNTIL** Blockes sooft, bis <Bool'scher Ausdruck> wahr wird.

Bemerkung : Zur Unterstützung von gut strukturierten Programmen wurde diese Anweisung eingeführt. So ist es im Zusammenhang mit dem **WHILE..WEND** - Schleifenkonstrukt möglich fast ganz ohne GOTO auszukommen. Denn alle Schleifen können in eine Form gebracht werden, sodaß **WHILE .. WEND** oder **REPEAT .. UNTIL** verwendet werden können.
Im Gegensatz zur **WHILE .. WEND** -Schleife wird bei **REPEAT .. UNTIL** nur am Ende der Schleife (bei UNTIL) überprüft ob die Abbruchbedingung erfüllt ist. D.h. die Schleife wird zumindest einmal durchlaufen.

Beispiel:
```
10 REPEAT
20 UNTIL INKEY$="a"
```

wartet solange bis die Taste 'a' gedrückt wurde und setzt dann das Programm fort.

Beispiel 2:
```
10 REPEAT
20 INPUT "Weiter ja / nein [j/n]";A$
30 UNTIL A$="J" OR A$="N" OR A$="j" OR A$="n"
```

So ist eine einfache Möglichkeit gegeben die Eingabe auf Zulässigkeit zu überprüfen.

S A V E

Syntax : SAVE [K,] <Dateiname> oder
SAVE <Dateiname>,A

Effekt : Speichert das, sich im Arbeitsbereich befindliche, Programm auf Diskette oder Kassette ab.

Bemerkung : Um Programme auf Diskette abspeichern zu können, ist ein Diskettenbetriebssystem erforderlich, das aber in der Diskettenversion von RL-BASIC bereits enthalten ist.
Wird RL-Basic ohne JADOS betrieben, ist die einfache Abspeicherung der Programme aus BASIC heraus nicht möglich. Es ist erforderlich das Programm entweder auf Kassette oder durch inspizieren einiger Speicherzellen auch auf Diskette abzuspeichern.
Wird das Diskettenbetriebssystem JADOS benutzt, kann durch Eingabe von **SAVE** <Dateiname>,A das BASIC-Programm als Text auf Diskette abgespeichert. Dies hat den Vorteil, daß das Programm dann mit Merge "Dateiname" an ein vorhandenes Programm angesetzt werden kann. (Siehe hierzu auch **MERGE**)

Abspeichern auf Kassette : Legen Sie eine Kassette in Ihren Recorder ein und drücken die Tasten START und RECORD gleichzeitig nieder.
Geben Sie nun **SAVE K,"Programmname"** ein und drücken die RETURN-Taste.
Nach ca. 5 sek. wird das Programm auf Kassette gespeichert.

Abspeichern auf Diskette : Legen Sie eine Diskette in Ihr Diskettenlaufwerk ein.
Geben Sie nun **SAVE "Programmname"** ein und drücken die RETURN-Taste.

Sollten Sie nicht im Besitz von RL-DOS oder JADOS sein :
Verlassen Sie BASIC mit **SYSTEM**. In den Speicherzellen $1de und $1f2 oberhalb des Grundprogrammes erfahren Sie Start- und Endadresse Ihres Basic-Programmes. Sie können nun diesen dort angegeben Speicherbereich auf Diskette speichern. Das Laden erfolgt dann so : Laden Sie das Basic-Programm ab der Speicherstelle $400 oberhalb Ihres Arbeitsbereiches. Setzen Sie nun die Speicherzellen $1de und $1f2, auf die oben ermittelten Werte. Führen Sie nun einen Warmstart durch.

S E T D A

Syntax : SETDA ‹Wert Kanal1›,‹Wert Kanal2›

Effekt : Gibt zwei Werte auf die Digital-Analog-Wandler Baugruppe aus.

Bemerkung : Dieser Befehl ist nur in Verbindung mit der o.g. Baugruppe sinnvoll. Die Wandlung erfolgt in ca 800ns.

Beispiel :

```
10 W1% = 0
20 W2% = 0
30 FOR I = 0 TO 3
40 FOR ZAEHLER% = 0 TO 30
50 SETDA W1%,W2%
60 NEXT
70 READ A%,B%
80 W1% = W1% + A%
90 W2% = W2% + B%
100 NEXT I
110 RESTORE
120 RUN
130 DATA 1,0,0,1,-1,0,0,-1
```

Gleiches Beispiel wie in Grundprogrammlisting (nicht getestet !)

S E R I N I T

Syntax : SERINIT ‹Kontroll-Byte›,‹Baudrate›

Effekt : Inititialisiert die serielle-Schnittstelle.

Bemerkung : Dieser Befehl ist nur in Verbindung mit der seriellen Schnittstellen-Baugruppe sinnvoll.
Kontroll-Byte und Baudrate werden, wie im Grundprogrammlisting kodiert. D.h. $1E wählt die Baudrate zu 9600 Bits/s.
(Siehe auch Grundprogrammlisting Funktion SERINIT)

Beispiel :

```
10 SERINIT $0B,$1E : REM 8 Datenbits, 1 Stoppbit, 9600 Baud
20 OPEN "I",#1,"TEXT.TXT"
30 OPEN "O",#2,"SER:"
40 WHILE NOT(EOF(1))
50 BYTE% = INP(1)
60 OUT #2, BYTE%
70 WEND
80 CLOSE #1
90 CLOSE #2
100 END
```

Öffnet die Datei TEXT.TXT, falls sie auf Diskette existiert und gibt sie dann über die serielle Schnittstelle mit 9600 Baud Übertragungsgeschwindigkeit aus.

E O F

Syntax : EOF(‹Dateinummer›)

Effekt : Zeigt das Ende einer sequentiellen Datei an.

Ergebnistyp : Boolean

Bemerkung : ‹Dateinummer› ist die Nummer, der vorher geöffneten sequentiellen Datei. Die Funktion liefert WAHR, falls das Dateiende bereits erreicht ist und FALSCH, falls das Dateiende noch nicht erreicht wurde.

EOF ist notwendig, um nicht über das Dateiende mittels des **INPUT #** -Befehls hinwegzulesen.

Beispiel :

```
10 OPEN "O",#1,"Datei.txt"
20 FOR I = 1 TO 20
30 PRINT #1,I
40 NEXT I
50 CLOSE #1
60 OPEN "I",#1,"Datei.txt"
70 INPUT #1,A$
80 PRINT A$;" ";
90 IF EOF(1) THEN GOTO 110
100 GOTO 70
110 CLOSE #1

RUN
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

I N P

Syntax : INP(<Dateinummer>)

Effekt : Liest ein Zeichen (Byte) aus der zum lesen geöffneten Datei

Ergebnistyp : Integer, Real

Bemerkung : Mittels **INP()** ist es möglich, einzelne Bytes aus einer Datei zu lesen. So kann auf einfache Art und Weise z.B. Zeichen von der seriellen Schnittstelle eingelesen werden, oder Datenwerte, die vorher mit **OUT #..** abgespeichert wurden wieder von Datei geladen werden. Eine Anwendung findet sich zum Beispiel beim Laden von Binärdateien. Diese können und dürfen nicht mit **INPUT #** .. eingelesen werden, da dies zum Absturz von RL-BASIC führen könnte. Mit der COL256 Baugruppe können die Farbwerte direkt von Diskette geladen und dann angezeigt werden.

Achtung ! Werden Daten in Form eines Byte-Stromes auf Diskette abgelegt so kann es vorkommen,daß ein Byte den Wert $00 besitzt. In diesem Fall darf das Dateiende nicht durch **EOF()** abgefragt werden, da **EOF** auf den WERT $00 reagiert ! Man sollte also genau wissen wielange eine Datei ist. (Diese Einschränkung gilt momentan für JADOS!)

Beispiel:

```
10 OPEN "I",#1,"DEMO.PIC"
20 FOR IX = 0 TO 511 STEP 2
30 FOR IY = 0 TO 511 STEP 2
40 C = INP(1)
50 COLOR = C
60 PSET IX,IY
70 NEXT IY
80 NEXT IY
90 CLOSE #1
```

Liest aus der Datei DEMO.PIC einzelne Bytes ein und gibt diese auf dem Bildschirm als Graphikpunkte aus.

P O I N T

Syntax : POINT(<x-Kooridnate>,<y-Koordinate>)

Effekt : Liefert den Farbwert an Position x,y zurück

Ergebnistyp : Integer,Real

Bemerkung : Da mit der COL256-Baugruppe ein Rücklesen der Farbinformation möglich ist, wurde diese Funktion integriert, um dies auch aus Basic-Programmen zu tun. **POINT** ist leider mit der GDP64K Graphikkarte nicht sinnvoll einzusetzen, da auf einzelne Bytes des Bildspeichers auf der GDP-Karte nicht per Programm zugegriffen werden kann. Aus Kompatibilitäts-Gründen liefert **POINT** bei der GDP-Karte immer den Wert Ø = Schwarz zurück ! x-Koordinate,y-Koordinate sollte im Bereich von Ø..511 liegen, um ordentliche Werte geliefert zu bekommen.

Eine Anwendung wäre zum Beispiel ein Bildschirmausdruck auf Drucker zu machen, wobei die gelieferten Werte in Graustufen auf dem Drucker umgesetzt werden müssen.

Beispiel: siehe **OUT** #.. Befehl

S Q R

Syntax : SQR(‹numerischer Ausdruck›)

Effekt : Berechnet die 2. Wurzel aus ‹numerischer Ausdruck›

Ergebnistyp: Integer,Real

Bemerkung : SQR dient zur Berechnung der Wurzel einer Zahl.
‹numerischer Ausdruck› muss eine Positive Zahl ›= Ø sein.
Bei negativen Werten, wird eine Fehlermerldung generiert.

Beispiel :
```
5 PRINT " Berechne Hypothenuse eines rechtwinkligen Dreiecks"
1Ø INPUT " Gebe die Laenge der Kathete ein :";LK
2Ø INPUT " Gebe die Laenge der Ankathete ein : ";LAK
3Ø PRINT " Die Laenge der Hypothenuse betraegt ";SQR(LK*LK+LAK*LAK)

run
 Berechne Hypothenuse eines rechtwinkligen Dreiecks
 Gebe die Laenge der Kathete ein :1
 Gebe die Laenge der Ankathete ein :1
 Die Laenge der Hypothenuse betraegt 1.414214
```

G O T O

Syntax : GOTO ‹Programmzeilennummer› ¦ ‹Marke›

Effekt : Führt einen unbedingten Sprung im Programmablauf aus.

Bemerkung : GOTO wird dazu benutzt den Programmablauf an eine bestimmte Zeile des Programmes abzugeben. 'Bad linenumber' erscheint, falls Sie versuchen zu einer Zeile zu springen, die nicht vorhanden ist.
GOTO kann auch dazu genutzt werden, Endlosschleifen zu programmen.

Ab Version 2.3 sind auch Marken als Sprungziele erlaubt. Marken sind durch ein nachfolgendes '!' gekennzeichnet z.B. 1Ø MARKE! definiert eine Marke in Zeile 1Ø. Wird nun **GOTO MARKE!** verwendet, wird das Programm in Zeile 1Ø fortgesetzt.
Das bedeutet, daß Marken an eine Zeilennummer gebunden sind, es ist nicht möglich auf eine Marke innerhalb einer Zeile zu springen.

Beispiel :
```
1Ø MARKE! : READ A,B
2Ø IF A=Ø AND B=Ø THEN END
3Ø PRINT "A=";A,"B=";B
4Ø S = A * B
5Ø PRINT "PRODUKT LAUTET :";S
6Ø GOTO MARKE! : REM ist gleichbedeutend mit GOTO 1Ø
7Ø DATA 12,5,8,3,9,Ø,Ø,Ø
8Ø END

RUN
A=12            B=5
PRODUKT LAUTET :6Ø
A=8             B=3
PRODUKT LAUTET :24
A=9             B=Ø
PRODUKT LAUTET :Ø
```

G R M O D E

Syntax : GRMODE Ø | 1

Effekt : Wählt zwischen GDP64K und COL256-Graphik aus.

Bemerkung : Seit Version 2.3 stehen auch Befehle zur Programmierung der COL256-Baugruppe zur Verfügung. Alle Befehle die bisher für die GDP64K Graphic gültig waren, können nun unverändert auf die COL256-Baugruppe übernommen werden.
Eine Ausnahme besteht bei dem Befehl **WRITE**, mit dem nur Text der Größe $21 auf die COL256-Graphik ausgegeben werden kann und dem Befehl **PAGE x,y** bei dem eine Schreib und Leseseite bei der GDP-Karte ausgewählt werden konnte. Da die COL256 standardmäßig mit 64K-Byte bestückt ist und somit keine weiteren Bildschirmseiten zur Verfügung stehen, wird dieser Befehl einfach ignoriert.
GRMODE Ø = GDP64K Graphik ist Standardwert bei der Initialisierung
GRMODE 1 = COL256 Graphik kann im Programm oder direkt eingegeben werden.

Beispiel:
```
>GRMODE 1
>LOAD "LINES-E"
>RUN
```

läßt das Graphikprogramm aus dem Anhang nun über die COL256 laufen.

R U N

Syntax : RUN [<Zeilennummer>|<Marke>] oder RUN <Dateiname>

Effekt: Startet einen Programmlauf.

Bemerkung : Das erste Format wird benutzt, ein Programm auszuführen, das sich gerade im Arbeitsbereich befindet.
Der Programmlauf startet in der ersten Zeile des Programmes, insofern <Zeilennummer> nicht angegeben wurde.
Wurde <Zeilennummer> spezifiziert, startet das Programm in <Zeilennummer>. (falls vorhanden)

Ab Version 2.4 kann eine Sprungmarke als Zieladresse angegeben werden. Eine Marke wird wie eine normale Variable mit nachfolgendem '!' definiert z.B. : START!
Alle Marken werden beim Programmlauf gesetzt !

Das zweite Format wird benutzt, um Programme zuerst von Diskette in den Arbeitsbereich zu laden und dann auszuführen.
<Dateiname> muß der Syntax des Betriebssystems entsprechen.
Das automatische Starten von Programmen von Kassette ist allerdings nicht vorgesehen.

RUN löscht vor einem Programmlauf erst alle Variablen !

Beispiel : LOAD K,"LINES"
RUN
Lädt das Programm LINES von Kassette und startet es dann.

ODER
RUN "LINES"
Lädt das Programm LINES von Diskette und beginnt direkt mit der Programmausführung.

I F .. T H E N
.
E L S E
.
E N D I F

Syntax : IF ‹Bool'scher Ausdruck› THEN

‹Anweisungen für WAHR-Teil›

ELSE

‹Anweisungen für FALSCH-Teil›

ENDIF

Effekt : Fallunterscheidung , bedingte Anweisung.
Erlaubt auf eine Bedingung hin weitere Anweisungen mit dieser zu verbinden.

Bemerkung : Mehrzeiliges **IF THEN ELSE**
IF.. THEN.. ELSE .. ENDIF wurde eingeführt, um das strukturierte Programmieren noch weiter zu unterstützen, als dies mit einer einzeiligen IF..THEN..ELSE - Anweisung möglich wäre. **IF ..ENDIF** muß auf mehere Programmzeilen verteilt werden. Ein **ELSE** darf nur zu Beginn einer Programmzeile stehen, auch dürfen nach **ELSE** keine weiteren Anweisungen in derselben Zeile folgen. Das ganze Konstrukt kann sich über mehrere Programmzeilen hinwegziehen. Als Ende einer so strukturierten mehrzeiligen **IF..THEN..ELSE** Konstruktion folgt ein **ENDIF**, das wiederum nur zu Beginn einer Zeile stehen darf.
Der **ELSE**-Zweig ist obligatorisch, er kann auch weggelassen werden.

Achtung! Zur Unterscheidung zwischen den beiden **IF** Konstrukten dient der Umstand, daß beim mehrzeiligen **IF..ENDIF** nach **THEN** keine Anweisung mehr in derselben Zeile folgt.

Beispiel :

```
 1 REM  Kleines Beispiel  zur  strukturierten
 2 REM  Programmierung :
 3 REM
 4 REPEAT
 5   INPUT a$
10   IF a$ = "wahr" THEN
20     PRINT " WAHR-Teil der IF Anweisung."
30   ELSE
40     PRINT " FALSCH-Teil der IF Anweisung."
50   END IF
60 UNTIL a$ = "wahr"
```

H A R D C O P Y

Syntax : HARDCOPY

Effekt : Erzeugt einen Aufruf über einen Sprungvektor, sodaß eine eigene Hardcopy-routine eingebunden werden kann.

Bemerkung : HARDCOPY wurde in den Befehlssatz von RL-BASIC aufgenommen, um jedem BASIC-Porgrammierer die Möglichkeit zu geben, einen Bildschirmausdruck vom aktuellen Bildschirm zu machen.
Das Problem hierbei liegt in der Tatsache, daß eine Hardcopy-Maus-Baugruppe notwendig ist, um Bildschirmausdrucke von der GDP64K-Karte zu erzeugen. Ein weiteres Problem liegt in den unterschiedlichen Steuersequenzen der Drucker, um diese zu einer Hardcopy zu bewegen. Diese Probleme wurden dadurch gelöst, daß der HARDCOPY-Befehl nicht selbst einen Bildschirmausdruck erzeugt, sondern nur ein Maschinenunterprogramm aufruft, das dies übernehmen sollte.
Es steht, wie vom Grundprogramm her gebräuchlich, ein Sprungvektor im RAM-Bereich oberhalb des Grundprogramms zur Verfügung, in den ein Sprungbefehl mit Zieladresse eingetragen werden kann. Im Fall des **HARDCOPY**-Befehls : $1C5A oberhalb des Grundprogramms. In dieser und der nachfolgenden Adresse ist der 680xx Befehl RTS-eingetragen. Soll nun eine eigene Hardcopy-Routine aufgerufen werden, so ist das Hardcopy-programm zuvor in einen freien Speicherbereich zu bringen und ab der Adresse $1C5A oberhalb des Grundprogramms ein Sprungbefehl auf dieses Hardcopy-Programm einzutragen.
Ein Beispiel : Sie haben ein Hardcopy-Programm für Ihren Drucker geschrieben. Ihr Hardcopy-Programm stehe zum Beispiel ab Adresse $80000, so müssen Sie nur noch einen Sprungbefehl (JMP $80000) ab Adresse $1C5A oberhalb des Grundprogramms einzutragen, um diese Routine auch von BASIC mittels des Befehls **HARDCOPY** aufrufen zu können.

T R U E / F A L S E

Syntax : TRUE bzw. FALSE

Effekt : liefert die Bool'schen Konstanten *WAHR* bzw. *FALSCh*

Ergebnistyp: Boolean (tritt nicht explizit auf)

Bemerkung : Um Testausgaben oder Endlosschleifen o.ä. zu programmieren dienen die Bool'schen Konstanten **TRUE** und **FALSE**. Sie spiegeln die Wahrheitswerte *WAHR* und *FALSCH* wieder, wie sie auch in Vergleichen innerhalb Bool'scher Ausdrücke vorkommen.

Beispiel :

```
10 WHILE TRUE
20   PRINT i
30   i = i + 1
40 WEND
```

Erzeugt eine Endlosschleife, in der alle positiven Zahlen auf den Bildschirm ausgegben werden.

Beispiel 2 :

```
10 IF TRUE THEN
20   PRINT " Wahr-Teil einer IF-Anweisung"
30 ELSE
40   PRINT " Falsch-Teil einer IF-Anweisung"
50 END  IF
```

Läßt den Wahrteil der If-Anweisung ausführen.

A N D / O R / N O T

Syntax : ‹numerischer Ausdruck 1› AND ‹numerischer Ausdruck 2›
‹numerischer Ausdruck 1› OR ‹numerischer Ausdruck 2›
NOT ‹numerischer Ausdruck›

Effekt : Berechnet ein durch bitweise Verknüpfung entstandenes Ergebnis. Als Verknüpfung stehen zur Verfügung : **AND**, OR, NOT

Ergebnistyp : Integer

Bemerkung : Um Hardwarenah programmieren zu können, sind auch Operationen notwendig, um einzelne Bits in einem Wort ansprechen und verändern zu können. In **RL-BASIC** stehen die Operation bitweises Undieren zweier 32-Bit Zahlen, bitweises Odern zweier 32-Bit Zahlen und das bitweise Negieren einer 32 Bit-Zahl mit **AND**, **OR**, **NOT** zur Verfügung.

Beispiel :
```
10 port%=$FFFFFFE0 * CPU
20 wert% = peek(port%) AND %1011
```

Maskiert den aus $FFFFFFE0 gelesenen Wert mit %1011

C P U

Syntax : CPU

Effekt : liefert den aktuellen CPU-Typ zurück

Ergebnistyp : Integer, Real

Bemerkung : Da **RL-BASIC** mit allen CPU-Varianten der 680xx Baureihe zusammenarbeiten kann, und auch BASIC-Programme auf allen diesen Progzessoren ohne Änderung laufen sollen, dient diese Funktion dazu die einzelnen PORT-Adressen aneinander anzupassen.

CPU liefert :
1 = 68008
2 = 68000
4 = 68020

Beispiel : WAIT $FFFFFF68 * CPU, 32
wartet auf das Drücken der Leertaste.
CPU sollte immer dann eingesetzt werden, falls auf IO-Adressen des NDR-Klein-Computers zugegriffen wird.

M O U S E X , M O U S E Y , M O U S E K

Syntax : MOUSEX bzw. MOUSEY oder MOUSEK

Effekt : Liefert die Mausposition bzw die Tastenwerte der Maus zurück, falls sich eine *HARDCOPY/MAUS*-Baugruppe im System befindet.

Ergebnistyp : Integer, Real

Bemerkung : Ist eine *Hardcopy/Maus*-Baugruppe im System und eine Maus angeschlossen, so können die Mauskoordinaten mit **MOUSEX, MOUSEY** abgefragt werden. Die Maustaste mit **MOUSEK**. **MOUSEK** liefert, je nach verwendeter Maus unterschiedliche Werte, die selbst ermittelt werden müssen !

Beispiel :

```
10 REM Male
20 REPEAT
30   PSET MOUSEX , MOUSEY
40 UNTIL FALSE
```

Zeichnet auf den Bildschirm. Die Punkte folgen dabei der Bewegung der Maus.

M O D

Syntax : ‹numerischer Ausdruck 1› MOD ‹numerischer Ausdruck 2›

Effekt : Berechnet den durch eine ganzzahlige Division entstehenden Rest. ‹numerischer Ausdruck 1› wird durch den ‹numerischer Ausdruck 2› dividiert.

Ergebnistyp : Integer,Real

Bemerkung : MOD dient zur Berechnung des Restes bei ganzzahliger Division.
Beispiel : 10 **MOD** 3 erhält den Wert 1.
Folgende Konventionen liegen der MODULO-Funktion zu Grunde :
Das Ergebnis erhält das Vorzeichen des 1. Operanden.
Die Modulo Operation wird zunächst ohne Betrachtung der Vorzeichen der Operanden durchgeführt.
Die prozessorinterne Modulo Funktion erzeugt dieselben Ergebnisse, wie die hier implementierte.

Beispiel :

```
10 PRINT " Apfelmaennchen in RL-BASIC"
20 INPUT " Gebe den Bereich des Realteils ein [xa,xe]";xa,xe
30 INPUT " Gebe den Bereich des Imaginaerteils ein [ya,ye]";ya,ye
40 INPUT " Anzahl der Iterationen [10..]";itmax%
50 PAGE 3,3:CLPG
55 inx = (xe-xa) / 512
56 iny = (ye-ya) / 512
60 FOR ix = xa TO xe STEP inx
70   FOR iy = ya TO ye STEP iny
80     it% = 0
85     zr = 0
86     zi = 0
90     REPEAT
100      zrq = zr * zr
104      ziq = zi * zi
106      it% = it% + 1
110      zi = 2 * zr * zi + iy
120      zr = zrq - ziq + ix
130    UNTIL zrq+ziq>4 OR it%>itmax%
140    IF it% MOD 2 = 1 AND it% <> itmax% THEN
145      PSET INT((ix-xa)/inx),INT((iy-ya)/iny)
150    END IF
160  NEXT IY
170 NEXT IX
180 END
```

```
RUN
 Apfelmaennchen in RL-BASIC
 Gebe den Bereich der Realteils ein [xa,xe] -2.25,1.8
 Gebe den Bereich des Imaginaerteils ein [ya,ye] -2,2
 Anzahl der Iterationen [10..] 20
```

zeichnet das Apfelmännchen ! (Kann aber etwas dauern ...)

S C R E E N

Syntax : SCREEN ‹Numerischer Ausdruck›,‹Numerischer Ausdruck›

Effekt : Liefert den Code des Zeichens des Textbildschirmes an Position ‹Spalte›,‹Zeile›

Ergebnistyp : Integer, Real

Bemerkung : ‹Spalte› darf Werte zwischen Ø..79 und ‹Zeile› Werte zwischen Ø..23 annehmen. Zurückgeliefert wird der ASCII-Code des Zeichens an Position ‹Spalte›,‹Zeile›. Dieser Befehl findet Anwendung bei der Programmierung von Bildschirmmasken etc.

Beispiel :

```
 10 REM Hardcopy des Textbildschirms
 20 REM Kann z.B. als Unterprogramm verwendet werden, um
 30 REM einen Ausdruck des aktuellen Textbildschirms, zu
 40 REM erreichen
 50 hardc!
 60 FOR zeile% = Ø TO 23
 70   FOR spalte% = Ø TO 79
 80     LPRINT CHR$(SCREEN spalte%,zeile%);
 90   NEXT spalte%
100   LPRINT
110 NEXT zeile%
120 REM hier z.b. RETURN eintragen, falls als Unterprogramm
130 REM benutzt.
```

A N H A N G A

Reservierte Schlüsselwörter

Die unten aufgelisteten Schlüsselwörter dürfen in BASIC-Programmen nicht als Anfangswort in Variablennammen enthalten sein.
Z.B. Führt der Variablenname FORM$ zu einem Laufzeitfehler, da in ihm das Schlüsselwort FOR entdeckt wird, erlaubt wäre z.B. MFOR$

ABS AND ARC ASC ATN AUTO
BIN$
CALL CHAIN CHR CIRCLE CLEAR CLOSE CLPG CLS COLOR CONNECT COS CONT
CPU
DATA DATE DEF DELETE DIM DRAW
END EOF ERASE ERL ERR ERROR EXP
FILES FOR FN FRE
GETAD8 GETAD1Ø GOSUB GOTO GRMODE
HARDCOPY HEX$
IF INKEY INP INPUT INSTR INT
KILL
LEFT LEN LET LINE LIST LLIST LOAD LOCATE LOG LPRINT
MID MOUSEX MOUSEY MOUSEK MOVE MOVETO MOD
NAME NEW NEXT NOT
OFF ON OPEN OR OUT
PAGE PAINT PEEK POINT POKE POS PRESET PRINT PSET
RANDOMIZE READ RELAIS REM REPEAT RESTORE RESUME RETURN RIGHT RND
RUN RENUM
SAVE SETDA SERINIT SGN SIN SOUND SPACE SPC(SQR STOP STR STRING
SWAP SYSTEM SCREEN
TAB TAN TIME TO TRON TROFF TURN
UNTIL USING
VAL VERIFY VARPTR
WAIT WEND WHILE WRITE
+ - / * ^ < > = ?

A N H A N G B

RL-BASIC Fehlermeldungen :

Fehlernummer :	Fehlermeldung :
1	Syntax error
2	Type Mismatch error
3	Out of Range error
4	Divide by Zero
5	NEXT without FOR error
6	WEND without WHILE error
7	RETURN without GOSUB error
8	Memory overflow
9	Undefined function error
10	Out of DATA error
11	STRING TOO LONG
12	ReDIMed array
13	Bad Subscript error
14	Formula too complex error
15	Illegal direct error
16	Cannot continue error
17	Break
18	Stopped
19	Future expansion
20	Bad linenumber or undefined label
21	":" missing
22	Stack overflow
23	Fatal error
24	WHILE without WEND error
25	UNTIL without REPEAT
26	No BASIC Data
27	Verify error
28	Checksum error
29	Label double defined
30	IF without END IF
31	END IF without IF
40	File not found
41	File already exists
42	Drives different
43	Disk full
44	Directory full
45	undefined Filenumber
46	File not open
47	Diskaccess error
48	Illegal Filename
50	Use Filenumber <>0
51	Undefined Filnumber
52	File is still open
53	File does not exist
54	Unable to write File
55	Unable to read File
56	Too many Files open
57	No Fileacces allowed

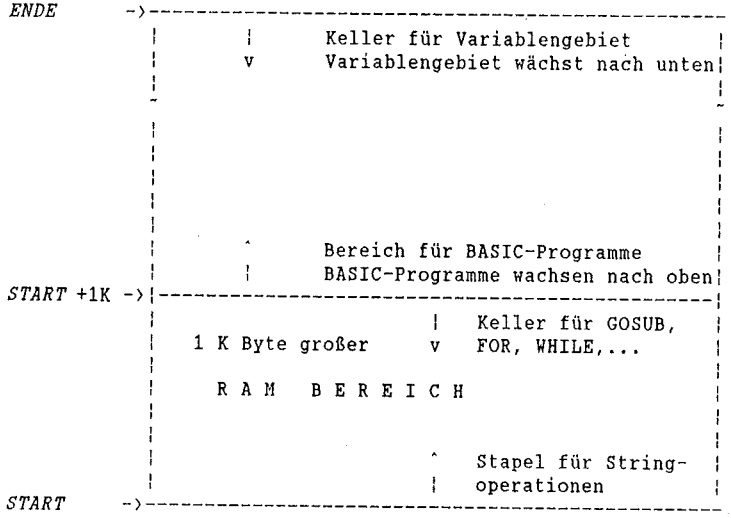
A N H A N G C

Adressbelegung durch RL-BASIC :

RL-Basic setzt eine Mindestgröße von 1-2 KByte RAM-Bereich voraus. Dies ist für die allermeisten Anwendungen zu klein, so daß ein Bereich von mindestens 8 Kbyte vorhanden sein sollte, um kleinere Programme schreiben zu können.

Nachfolgend wird nun aufgelistet, wie **RL-BASIC** den ihm zur Verfügung stehende Speicherplatz aufteilt :

Arbeitsbereich : *START* bis *ENDE*



A N H A N G D

Adressbelegung der Systemvariablen oberhalb des Grundprogramms :

Zum Betrieb von **RL-BASIC** sind einige Puffer und Speicherplätze erforderlich, die alle oberhalb des Grundprogramms angesiedelt wurden, da dieser Bereich schon von Systemvariablen des Grund-Programms benutzt wird.

Dieser Bereich erstreckt sich von Adresse $0000 bis $1FFF, oberhalb des Grundprogramms. Z.B. bei einem System dessen Grundprogramm ab Adresse $000000 liegt, befinden sich die BASIC-Systemvariablen im Bereich von $8000 - $9FFF.(68008 und 68000)
Dieser Speicherbereich steht dann Maschinenprogrammen etc. nicht mehr zur Verfügung !

Effektiv frei verfügbar wäre damit der Bereich $2000 -.... oberhalb des Grundprogramms.
Dies gilt aber nur dann, falls sich nicht **RL-DOS** im Speicher befindet, denn dieses liegt ab Adresse $4000 - $7000 oberhalb des Grundprogramms !

Speicheraufteilung oberhalb des Grundprogramms :

$x0000 - $x1C00 : Symboltabelle der Basicprogrammvariablen

$x1C00 - $x1FFF : Systemvariable des RL-BASIC-Interpreters

$x2000 - $x3FFF : Frei

$x4000 - $x7FFF : **RL-DOS** incl. Inhaltsverzeichnis, falls DOS geladen ansonsten frei (z.B. unter **JADOS** frei)

Für ‹x› ist jeweils die RAM-Startadresse des Grundprogrammrams einzusetzen :
RAM-Startadresse = $008000 => $008000 - $009C00 für die Symboltabelle.

A N H A N G E

Zahlenbereiche , Genauigkeit der Arithmetik

INTEGER :

4 Byte = 32 Bit
Alle Operationen werden 32-Bit breit ausgeführt, sodaß ein Zahlenbereich von +- 2^31 für einfache Arithmetik zur Verfügung steht.

REAL :

8 Byte = 64 Bit

i) Normales BASIC :

Alle Berechnungen werden mit einer 49 Bi Gleit-Kommaarithmetik durchgeführt.
32 Bit - Mantisse
16 Bit - Exponent
1 Bit Vorzeichen
Allerdings werden für Zahlenwandlungen 4 Bit zu Rundungszwecken benötigt. Berechnungen erfolgen aber mit der vollen internen Genauigkeit.
Zahlenbereich : +- 9.999999 E+- 999

ii) BASIC mit Unterstützung der FPU 68881 :

Alle Berechnungen erfolgen mit 80 Bit Genauigkeit. Die Ergebnisse werden aber nur mit 64Bit auch in Variablen abgelegt. Dies ergibt i.a. eine Genauigkeit von ca. 16 Stellen.

Zahlenbereich : +- 9.9999999999999999 E+-999
(für Variablen 2.2 e -308 .. 18 E 307)

Die Operatorrangfolge lautet :

* / MOD

\+ - AND OR

Um diese zu umgehen, können Klammern verwendet werden.
Unäre Operationen wie NOT, INT,... haben die höchste Priorität.

A N H A N G F

Hinweise zur Programmierung von BASIC-Programmen unter RL-BASIC

Mit **RL-BASIC** haben Sie einen leistungsfähigen Interpreter zur Hand, mit dem es umzugehen gilt. Es gibt einige Besonderheiten bzw. Tricks, um das gewünschte Ziel, nämlich ein korrekt funktionierendes Programm, zu erreichen.
Es gilt einige Besonderheiten in **RL-BASIC** zu beachten !
Hier deshalb einige Hinweise, wie man "vernünftig" in **RL-BASIC** programmiert :

1. Springe nie aus Programmschleifen heraus, die durch Konstrukte, wie
   WHILE .. WEND
   REPEAT .. UNTIL oder
   FOR .. NEXT gebildet sind, da durch unsachgemäßes Beenden noch Müll im Rückkehrkeller, der für solche Zwecke eingerichtet ist, verbleibt. So ist es z.B. zu erklären, daß der Interpreter die Rückkehradresse nicht mehr findet, falls in einem Unterprogramm aus einer **FOR .. NEXT** - Schleife gesprungen wird.

2. Versuche, wo möglich, nur einen Befehl je Programmzeile zu verwenden, um ein besser lesbares Programm zu erhalten. Das Programm wird beim Listen optisch besser strukturiert.

3. Vermeide **GOTO**. Falls **GOTO** verwendet werden muß, so sollte auf Programmarken als Sprungziel zurückgegriffen werden. (**RENUM** verändert beim Neunummerieren nur die Zeilenadressen, nicht die Sprungziele nach **GOTO, GOSUB** !)

4. Verwende, wo möglich, Integer-Variablen als Laufvariable in **FOR .. NEXT** - Schleifen, da sie schneller sind.

5. Vermeide **POKE, PEEK** in BASIC-Programmen, da hierdurch ein Programm auf anderen Rechnern nicht übertragbar ist.
   Falls auf Ein-Ausgabe Adressen (IO) zugegriffen werden müssen, ist mit **CPU** zu multiplizieren, damit dasselbe Programm auch auf einem anderen Prozessor der 680xx-Reihe im NDR-Klein-Rechner ablauffähig ist.

6. Verwende in einer einzeiligen **IF**-Anweisung keine Schleifenkonstrukte, wie **FOR .. NEXT, REPEAT .. UNTIL, WHILE .. WEND**, falls sich diese über mehr als eine BASIC-Zeile hinziehen. Das kann dann zu unvorhersehbaren Verhältnissen im Programmablauf führen.
   Benutze besser das Mehrzeilige **IF.. THEN .. END IF**, um eine Struktur besser sichtbar zu machen.

7. Wandle in Funktionen und Anweisungen, die Integer-Werte als Parameter verlangen, Gleitkommaausdrücke als Gesamtes mittels der Funktion **INT()** in einen Integer-Wert.
   (z.B. **PSET** Ø.Ø12 * 234.5,2 ergibt einen Syntax-Fehler
   PSET INT(Ø.Ø12 * 234.5) , 2 ist korrekt
   PSET x * 2 , 2 für x = Ø.5 ergibt dasselbe wie **PSET** Ø,2 !!)

A N H A N G G

Abgeleitete mathematische Funktionen

Logarithmus zur Basis B :	LOG(x) / LOG (B)
Sekans :	1 / COS (x)
Cosekans :	1 / SIN (x)
Cotangens :	1 / TAN (x)
Arcussinus :	ATN (x / SQR (1 - x^2)
Arcuscosinus :	ATN(1) * 2 - ATN(x / SQR (1 - x^2)
Arcussekans :	ATN (SQR (x^2 -1) + SGN(x) * ATN(1) * 4
Sinushyperbolicus :	(EXP (x) - EXP (-x)) / 2
Cosinushyperbolicus :	(EXP (x) + EXP (-x)) / 2
Tangenshyperbolicus :	(EXP (x) - EXP (-x)) / (EXP(x) + EXP(-x))
Area Sinus Hyperbolicus :	LOG (x + SQR(x^2 + 1))
Area Cosinus Hyperbolikus :	LOG (x + SQR(x^2 - 1))

A N H A N G H

Beispielprogramme in RL-BASIC

```
 10 REM Apfelmaennchen in Basic
 20 REM
 30 REM (c) R. Lobreyer 21.04.87
 40 REM
 50 PRINT "Apfelmaennchen in RL-BASIC"
 60 INPUT "Gebe den Bereich des Realteils ein [xa,xe]";xa,xe
 70 INPUT "Gebe den Bereich des Imaginaerteils ein [ya,ye]";ya,ye
 80 INPUT "Anzahl der Iterationen [10..]";itmax%
 90 PAGE 3,3: CLPG
100 inx = (xe - xa) / 512
110 iny = (ye - ya) / 512
120 FOR ix = xa TO xe STEP inx
130   FOR iy = ya TO ye STEP iny * 2
140     it% = 0
150     zr = 0
160     zi = 0
170     REPEAT
180       zrq = zr * zr
190       ziq = zi * zi
200       it% = it% + 1
210       zi = 2 * zr * zi + iy
220       zr = zrq - ziq + ix
230     UNTIL zrq + ziq > 4 OR it% > itmax%
260     IF it% <  > itmax% THEN
265       COLOR = it%
270       PSET  INT ((ix - xa) / inx), INT ((iy - ya) / iny)
280     END  IF
290   NEXT iy
300 NEXT ix
310 END
```

Zeichnet einen Ausschnitt aus der Mandelbrotmenge.
-2.25 , 2.0 für Realteil
-2.0 , 2.0 für Imaginärteil liefern das "Apfelmännchen"

-2.25 , 0.75 für Realteil
-1.5 , 1.5 für Imaginärteil sowie

-1.254024, -1.252861 für Realteil
0.046252, 0.047125 für Imaginärteil liefern schöne Bilder.
Obiges Programm liefert insbesondere in Verbindung mit der COL256-Karte beeindruckende Ergebnisse. (Vor Aufruf GRMODE 1 eingeben)

```
  5 REM Zeichne ein paar willkuerlich liegende Rechtecke
 10 RANDOMIZE 511
 20 PAGE 3,3
 30 CLPG
 40 COLOR  = $cc
 50 FOR i = 1 TO 1000
 60   x1% =  RND
 70   y1% =  RND
 80   x2% =  RND
 90   y2% =  RND
100   CONNECT (x1%,y1%) - (x2%,y1%) - (x2%,y2%) - (x1%,y2%) - (x1%,y1%)
110   COLOR  =  RND
120 NEXT
```

Dieses Programm erzeugt einige (1000) Rechtecke auf dem Bildschirm
Falls die COL256-Karte benutzt wird erscheinen diese in den entsprechenden Farben bzw. Graustufen.

```
 10 REM Freizeichnen mit der Maus
 20 REM
 30 k% = 170      <-- hier einen vorher ermittelten Wert eintragen
 40 REPEAT
 50   REM LOCATE 5,5
 60   REM   PRINT  MOUSEK
 70   x% =  MOUSEX  * 2
 80   y% =  MOUSEY  * 2
 90   IF  MOUSEK  <  > k% THEN
100     PSET  ABS (x%), ABS (y%)
105     FOR w% = 1 TO 100: NEXT : PRESET  ABS (x%), ABS (y%)
110   ELSE
120     FOR w% = 1 TO 10:t% =  MOUSEK : NEXT
130     IF  MOUSEK  = k% THEN
140       CIRCLE (x%,y%),5,f
150     END  IF
160   END  IF
170 UNTIL  FALSE
```

Dieses Programm dient dazu, mit der Maus etwas auf den Bildschirm zu malen. Es kann natürlich nur dann funktionieren, falls sich eine Hardcopy-Maus-Baugruppe im System befindet und eine Maus angeschlossen ist.
In Programmzeile 30 muß ein vorher ermittelter Wert eingetragen werden, der dem Drücken der rechten oder linken Maustaste entspricht.

```
 10 REM Verschicke eine Datei ueber die serielle Schnittstelle
 20 REM Baudrate : 9600
 30 REM Uebertragung : 8 Bit ohne Paritaet, 1 Stop-Bit
 40 REM
 50 REM (c) R. Lobreyer 02.05.87
 60 REM
 70 CLS
 80 PRINT "Uebertrage Dateien per serieller Schnittstelle "
 90 SERINIT $b,$1e
100 FILES
110 INPUT "Dateiname ->";nam$
120 OPEN "I",#1,nam$
130 OPEN "O",#2,"ser:"
140 WHILE  NOT  EOF (1)
150   INPUT #1,a$
160   PRINT #2,a$
170 WEND
180 CLOSE #2
190 CLOSE #1
200 END
```

Dieses Programm kann zur Versendung von Dateien über eine serielle Schnittstelle verwendet werden. Ich habe es selber im Gebrauch und verschicke damit Textdateien an einen anderen Rechner. Wird in Zeile 130 statt SER: einfach CON: eingetragen, so können Sie Textdateien auf dem Bildschirm anzeigen lassen.

```
 10 PAGE 3,3: CLPG
 20 DIM OLDLIN%(50,4)
 30 RANDOMIZE 511
 40 X0% =  RND :X1% =  RND :Y0% =  RND :Y1% =  RND
 50 DX% = 3:DY% = 7:DX1% = 5:DY1% = 3
 60 REPEAT
 70   K% = (K% + 1) MOD 50
 80   COLOR  = 0
 90   LINE (OLDLIN%(K%,0),OLDLIN%(K%,1)) - (OLDLIN%(K%,2),OLDLIN%(K%,3))
100   COLOR  = C%:C% = C% + 1
110   LINE (X0%,Y0%) - (X1%,Y1%)
120   OLDLIN%(K%,0) = X0%
130   OLDLIN%(K%,1) = Y0%
140   OLDLIN%(K%,2) = X1%
150   OLDLIN%(K%,3) = Y1%
160   X0% = X0% + DX%
170   Y0% = Y0% + DY%
180   IF X0% < 10 OR X0% > 500 THEN DX% =  - DX%
190   IF Y0% < 10 OR Y0% > 500 THEN DY% =  - DY%
200   X1% = X1% + DX1%
210   Y1% = Y1% + DY1%
220   IF X1% < 10 OR X1% > 500 THEN DX1% =  - DX1%
230   IF Y1% < 10 OR Y1% > 500 THEN DY1% =  - DY1%
240 UNTIL T = 4
```

Linien-demo : Zeichnet wandernde Linien auf den Bildschirm

```
 10 REM Konvertiere RL-BASIC Programme, die in RL-BASIC Version 1.x und 2.0
 20 REM erstellt wurden.
 30 REM Ein RL-BASIC Porgramm muss zuvor auf eine JADOS Diskette abgespeichert
 40 REM werden : 1. Programm in einen freien Speicherbereich bringen
 50 REM          2. mit DSAVE Porgramm als Daten auf Diskette unter JADOS ab
 60 REM             speichern.
 70 REM
 80 REM Mittels dieses Programmes wird das BASIC-Programm als TEXT-Datei
 90 REM unter dem Dateiname "NAME.LST" auf Diskette abgespeichert.
100 REM
110 REM Es ist dann moeglich, das Programm mit MERGE "NAME.LST" wieder
120 REM in ein BASIC-Programm zu wandeln, das dann wie gewohnt abgespeichert
130 REM werden kann.
140 REM
150 REM (c) R. Lobreyer 23.03.87
160 REM
170 REM
190 FILES
200 INPUT "Bitte Dateiname des BASIC-Programmes angeben :";na$
205 INPUT "Ausgabedatei (Programmtext) :";na1$
210 OPEN "I",#1,na$
280 OPEN "O",#2,na1$
290 base% =  INP (1) +  INP (1) +  INP (1) +  INP (1)
300 REPEAT
310   znr =  INP (1) * 256 +  INP (1)
315   m% = 0
320   PRINT #2,znr;" ";
330   REPEAT
```

```
340     x% =  INP (1)
345     m% = m% + 1
350     IF x% >  = 128 THEN
360       x% = x% - 128
370       IF x% = 127 THEN
380         x% =  INP (1)
385         m% = m% + 1
390       END  IF
391       RESTORE
393       FOR index% = 0 TO x%
394         READ a$
396       NEXT
397       PRINT #2," ";a$;" ";
400     ELSE
410       IF x% <  > 0 THEN
420         PRINT #2, CHR$ (x%);
430       END  IF
436     END  IF
440   UNTIL x% = 0
450   PRINT #2
455   IF m% MOD 2 = 1 THEN
456     base% =  INP (1) +  INP (1) +  INP (1)
460   ELSE
500     base% =  INP (1) +  INP (1) +  INP (1) +  INP (1)
510   END  IF
515 UNTIL base% = 0
520 PRINT #2,"$END"
530 CLOSE #2
540 END
```

```
39900 DATA ""
39910 DATA "END"
40000 DATA "FOR"
40005 DATA "NEXT"
40010 DATA "DATA"
40015 DATA "INPUT"
40020 DATA "DIM"
40025 DATA "READ"
40030 DATA "LET"
40035 DATA "GOTO"
40040 DATA "RUN"
40045 DATA "IF"
40050 DATA "RESTORE"
40055 DATA "GOSUB"
40060 DATA "RETURN"
40065 DATA "REM"
40070 DATA "STOP"
40075 DATA "PRINT"
40080 DATA "CLEAR"
40085 DATA "LIST"
40090 DATA "NEW"
40095 DATA "ON"
40100 DATA "REPEAT"
40105 DATA "WAIT"
40110 DATA "DEF"
40115 DATA "POKE"
40120 DATA "CONT"
40125 DATA "CLPG"
40130 DATA "UNTIL"
40135 DATA "OUT"
40140 DATA "LPRINT"
40145 DATA "LLIST"
40150 DATA ""
40155 DATA ""
40160 DATA "ELSE"
40165 DATA "TRON"
40170 DATA "TROFF"
40175 DATA "SWAP"
40180 DATA "ERASE"
40185 DATA "VERIFY"
40190 DATA "ERROR"
40195 DATA "RESUME"
40200 DATA "DELETE"
40205 DATA "AUTO"
40210 DATA "RENUM"
40215 DATA ""
40220 DATA ""
40225 DATA ""
40230 DATA ""
40235 DATA "LINE"
40240 DATA "LOMEM"
40245 DATA "HIMEM"
40250 DATA "WHILE"
40255 DATA "WEND"
40260 DATA "CALL"
40265 DATA "WRITE"
40270 DATA ""
```

```
40275 DATA "CHAIN"
40280 DATA "ARC"
40285 DATA "RANDOMIZE"
40290 DATA "MOVETO"
40295 DATA "SYSTEM"
40300 DATA "HARDCOPY"
40305 DATA "OPEN"
40310 DATA ""
40315 DATA ""
40320 DATA ""
40325 DATA "CLOSE"
40330 DATA "LOAD"
40335 DATA "MERGE"
40340 DATA "FILES"
40345 DATA "NAME"
40350 DATA "KILL"
40355 DATA ""
40360 DATA ""
40365 DATA "SAVE"
40370 DATA ""
40375 DATA "CLS"
40380 DATA "LOCATE"
40385 DATA ""
40390 DATA "SOUND"
40395 DATA "GRMODE"
40400 DATA "COLOR"
40405 DATA "PSET"
40410 DATA "PRESET"
40415 DATA "CIRCLE"
40420 DATA "PAINT"
40425 DATA "CONNECT"
40430 DATA ""
40435 DATA ""
40440 DATA "KEY"
40445 DATA "PAGE"
40450 DATA "TO"
40455 DATA "THEN"
40460 DATA "TAB("
40465 DATA "STEP"
40470 DATA ""
40475 DATA "FN"
40480 DATA "SPC("
40485 DATA "NOT"
40490 DATA "ERL"
40495 DATA "ERR"
40500 DATA "STRING$"
40505 DATA "USING"
40510 DATA "INSTR"
40515 DATA "TURN"
40520 DATA "VARPTR"
40525 DATA "INKEY$"
40530 DATA "OFF"
40535 DATA "MOVE"
40540 DATA "DRAW"
40545 DATA ">"
40550 DATA "="
40555 DATA "<"
40560 DATA "+"
40565 DATA "-"
40570 DATA "*"
40575 DATA "/"
40580 DATA "^"
40585 DATA "AND"
40590 DATA "OR"
40595 DATA "XOR"
40600 DATA "EQU"
40605 DATA "IMP"
40610 DATA "MOD"
40615 DATA "\"
40620 DATA ""
40625 DATA ""
40630 DATA ""
40635 DATA "LEFT$"
40640 DATA "RIGHT$"
40645 DATA "MID$"
40650 DATA "SGN"
40655 DATA "INT"
40660 DATA "ABS"
40665 DATA "SQR"
40670 DATA "RND"
40675 DATA "SIN"
40680 DATA "LOG"
40685 DATA "EXP"
40690 DATA "COS"
40695 DATA "TAN"
40700 DATA "ATN"
40705 DATA "FRE"
40710 DATA "INP"
40715 DATA "POS"
40720 DATA "LEN"
40725 DATA "STR$"
40730 DATA "VAL"
40735 DATA "ASC"
40740 DATA "CHR$"
40745 DATA "PEEK"
40750 DATA "SPACE$"
40755 DATA "BIN$"
40760 DATA "HEX$"
40765 DATA "LPOS"
40770 DATA ""
40775 DATA ""
40780 DATA ""
40785 DATA ""
40790 DATA ""
40795 DATA ""
40800 DATA ""
40805 DATA ""
40810 DATA ""
40815 DATA ""
40820 DATA ""
40825 DATA ""
40830 DATA ""
40835 DATA "TRUE"
40840 DATA "FALSE"
40845 DATA ""
40850 DATA ""
40855 DATA ""
40860 DATA ""
```

```
40865 DATA "EOF"
40870 DATA ""
40875 DATA "LOF"
40880 DATA ""
40885 DATA ""
40890 DATA ""
40895 DATA "MOUSEX"
40900 DATA "MOUSEY"
40905 DATA "MOUSEK"
40910 DATA ""
40915 DATA ""
40920 DATA ""
40925 DATA ""
40930 DATA ""
40935 DATA ""
40940 DATA ""
40945 DATA ""
40950 DATA ""
40955 DATA ""
40960 DATA ""
40965 DATA ""
40970 DATA ""
40975 DATA ""
40980 DATA ""
40985 DATA ""
40990 DATA ""
40995 DATA ""
41000 DATA ""
41005 DATA ""
41010 DATA ""
41015 DATA "CPU"
41020 DATA "GETAD10"
41025 DATA "GETAD8"
41030 DATA "CSRLIN"
41035 DATA "POINT"
41040 DATA "DAY"
41045 DATA "DATE"
41050 DATA "TIME"
41055 DATA ""
41060 DATA "SCREEN"
41065 DATA "DSKF"
41070 DATA ""
41075 DATA ""
41080 DATA ""
41085 DATA ""
41090 DATA "SERINIT"
41095 DATA "SETDA"
41100 DATA "RELAIS"
```

Konvertierungsprogramm, um BASIC-Programme der Versionen vor 2.4 in eine Form zu bringen, die dann mittels **MERGE "<Dateiname>"** geladen werden können. (Der Dateiname ist jeweils mit Endung .BAS einzugeben !)

```
*
*
* Programm zur Druchfuehrung einer Hardcopy
* Drucker : NEC P5/P6/P7 bzw.  EPSON 24 Nadel-Drucker
*
* (C) R. LOBREYER  21.04.87
* Nach Vorlage von G. Sternberg und Uwe Koch
*
CPU      EQU     4
READY    EQU     $FFFFFF8A*CPU
CLEAR    EQU     $FFFFFF8D*CPU
LOX      EQU     $FFFFFF89*CPU
HIX      EQU     $FFFFFF88*CPU
LOY      EQU     $FFFFFF8B*CPU
HIY      EQU     $FFFFFF8A*CPU
HC_ENTRY EQU     $1C5A
USERCI   EQU     $18
USERCSTS EQU     $1E
IOSTATB EQU      $2B

*-------------------------------------------------------------------------------*
* Hardcopy unter RL-BASIC
* Hardcopy mit CTRL-@ in jedem Anwenderprogramm, das Tastatureingabe fordert
*
START:  MOVE     #!SETA5,D7
        TRAP     #1
        MOVE     #$4EF9,HC_ENTRY(A5)            ; Trage Vektoren ein
        LEA      $2000+HARDC-TAST_CI(A5),A0
        MOVE.L   A0,HC_ENTRY+2(A5)
        LEA      $2000(A5),A0
        MOVE     #$4EF9,USERCI(A5)              ; Eingabe Umlenkung
        MOVE.L   A0,USERCI+2(A5)
        LEA      $2000+TAST_CSTS-TAST_CI(A5),A0
        MOVE     #$4EF9,USERCSTS(A5)
        MOVE.L   A0,USERCSTS+2(A5)              ;

        MOVE.B   #6,IOSTATB(A5)                 ; Und Eingabe umlenken

        LEA      $2000(A5),A0                   ; $2000 oberhalb Grundprogramm
        LEA      TAST_CI(PC),A1                 ; Kopiere Pogramm hinter
        LEA      BUFFER(PC),A2                  ; Programmbereich.
COPYL:  MOVE     (A1)+,(A0)+
        CMPA.L   A2,A1
        BLS.S    COPYL
        RTS
```

```
*-------------------------------------------------------------------------------*
* TAST_CI wird bei jeder Tastatureingabeanforderung angesprungen.
* Folgendes passiert :
* [1] normale CI-Routine aufrufen
* [2] Falls CTRL-@ gedrueckt wurde -> Hardcopy aufrufen.
*-------------------------------------------------------------------------------*
TAST_CI:
* [1]
        MOVEM.L D1-D7/A0-A6,-(A7)       ; Rette verwendete Register
        MOVE    #!SETA5,D7
        TRAP    #1
        CLR.B   IOSTATB(A5)             ; alte CI Routine reaktivieren
        MOVE    #!CI,D7
        TRAP    #1
        MOVE.B  #6,IOSTATB(A5)
        MOVEM.L (A7)+,D1-D7/A0-A6
* [2]
        TST.B   D0                      ; CTRL-@ -> Hardcopy aufrufen
        BEQ     HARDC
        RTS

TAST_CSTS:
        MOVEM.L D1-D7/A0-A6,-(A7)
        MOVE    #!SETA5,D7
        TRAP    #1
        CLR.B   IOSTATB(A5)
        MOVE    #!CSTS,D7
        TRAP    #1
        MOVE.B  #6,IOSTATB(A5)
        MOVEM.L (A7)+,D1-D7/A0-A6
        TST.L   D0
        RTS
*-------------------------------------------------------------------------------*
* HARDC erzeugt eine Hardcopy auf dem angeschlossenen Drucker
* Hardcopy ist leider um 90 Grad gedreht
*
HARDC:  MOVEM.L D0-D7/A0-A6,-(A7)       ; vorsorglich alle Register Retten
        BSR     INIT_PRT                ; Drucker initialisieren
        MOVE    #$FFFF-720,D2           ; Rechts anfangen
        MOVE.B  #67,D3

LOOP:   BSR     GET_LINE                ; Hole Bildschirmzeile
        BSR     INIT_LINE               ; Drucker fuer Zeile vorbereiten
        BSR     PRT_LINE                ; Zeile ausdrucken
        SUBQ.B  #1,D3
        BNE.S   LOOP
        CLR.B   LOX                     ; Fadenkreuz ausschalten
        CLR.B   HIX
        CLR.B   LOY
        CLR.B   HIY
        BSR     PRT_RESET               ; Drucker Ruecksetzen
        MOVEM.L (A7)+,D0-D7/A0-A6
        RTS
```

```
*-------------------------------------------------------------------------------
* Unterprogramme : Initialisiere Drucker
* Zeilenvorschub, ...
INIT_PRT:
        LEA     INIT_TAB(PC),A0
PR_OUT: MOVE.B  (A0)+,D0
        MOVE    #!LO,D7
        TRAP    #1                      ; Zeichen ausgeben
        CMP.B   #$FF,(A0)               ; Ende erreicht ?
        BNE.S   PR_OUT
        RTS
*-------------------------------------------------------------------------------
* initialisere Drucker fuer eine zu druckende Zeile
* Graphikmodus einschalten !
INIT_LINE:
        LEA     INITL_TAB(PC),A0
        BRA     PR_OUT
*-------------------------------------------------------------------------------
* Setze Drucker zurueck
*
PRT_RESET:
        LEA     RESET_TAB(PC),A0
        BRA     PR_OUT
*-------------------------------------------------------------------------------
* Hole Bildschirmzeile und lege diese in Puffer ab
*
GET_LINE:
        MOVEQ   #8,D4           ; Zahl der Spalten pro Druckzeile
GET_LL1:MOVE    #256  ,D5       ; Zahl der Punkte pro Spalte
        LEA     BUFFER(PC),A0
        ADDQ    #1,D2           ; Spaltenzaehler
        MOVE.B  #$FE,LOY        ;
        MOVE.B  #$FF,HIY
        ROR.W   #8,D2
        MOVE.B  D2,HIX
        ROR.W   #8,D2
        MOVE.B  D2,LOX          ; Spalte und Zeile einstellen
        MOVE.B  CLEAR,D6

GET_LL2:BTST.B  #7,READY        ; warte bis Punkt gefunden
        BEQ.S   GET_LL2
        MOVE.B  READY,D6
        LSL.B   #2,D6
        MOVE.B  (A0),D7
        ROXL.B  #1,D7
        MOVE.B  D7,(A0)+
        MOVE.B  CLEAR,D6
        SUBQ    #1,D5
        BNE.S   GET_LL2
        SUBQ    #1,D4
        BNE.S   GET_LL1

        RTS
```

```
*-------------------------------------------------------------------------------
* Gebe Puffer auf den Drucker aus. Werte werden als Graphikuster ausgedruckt.
* Ein Graphikzeichen = 3Byte , wovon nur mittleres Byte verwendet wird, Rest = Ø
*
PRT_LINE:
        LEA     BUFFER(PC),AØ
        MOVE.W  #256,D1
PRT_LL1:MOVE.B  (AØ)+,DØ
        NOT.B   DØ
        MOVE    #!LO,D7
        TRAP    #1
        MOVE    #!LO,D7
        CLR     DØ
        TRAP    #1
        MOVE    #!LO,D7
        CLR     DØ
        TRAP    #1
        SUBQ    #1,D1
        BNE.S   PRT_LL1

        MOVE    #$D,DØ          ; nachfolgend : CR-LF
        MOVE    #!LO,D7
        TRAP    #1
        MOVE    #!LO,D7
        MOVE    #$A,DØ
        TRAP    #1
        RTS
*-------------------------------------------------------------------------------
* Tabellen und Puffer :
INIT_TAB:
; Drucker RESET, Zeilenabstand  8/18Ø"
        DC.B    $1B,'@'
        DC.B    $1B,'3',8
        DC.B    $FF
        DS      Ø
INITL_TAB:
; links etwas Platz lassen, Graphikmodus mit Anzahl auszugebender Bytes.
; Graphikaufloesung : Horizontal 1/9Ø"
;                     Vertikal   1/18Ø"
        DC.B    '           '
        DC.B    $1B,'*',38,Ø,1
        DC.B    $FF
        DS      Ø
RESET_TAB:
        DC.B    $1B,'@',$FF

ds  Ø
OLDTRAP1:DS.L   4

BUFFER: DS.B    256
END
```

Systemhandbuch RL-BASIC 2.4

Programmablaufplan in Struktogrammform

Erläuterungen

Initialisierung bei Aufruf über die Bibliothek

- Stelle Basisadresse des Grundprogramms fest. Register A5 enthält dann den Wert Grundprogrammstart + $8000
- Lege den Arbeitsbereich fest, falls in Speicherzelle SERIAL nicht die Seriennummer steht.
  (Arbeitsbereich := RAMSTART+$2000 .. RAMENDE-$400)
- Gebe Eingangsmenü auf den Bildschirm aus und warte auf eine geeignete Eingabe der Tastatur.

Lese Eingabezeile von EDITOR

- Stelle fest in welcher Bildschirmzeile eine Eingabe erfolgte. Lese sodann diese Zeile in einen Puffer ein.
- Lege Zeilennummer in eine Speicherzelle ab, falls eine Zeilennummer eingegeben wurde, ansonsten lege dort den Wert 65355 ab (Kennung Direktmodus).
  (Puffer enthält nun keine Zeilennummer mehr)
- Wandle alle Klein- in Großbuchstaben. (außer in " ")
- Durchsuche Puffer nach Schlüsselwörtern und ersetze (Diese sind in einer Liste abgelegt, die linear durchsucht wird.) diese durch die entsprechenden TOKEN. (Stellung der Schlüssel in der Liste) Ziffernfolgen werden durch eine Maschinenkonstante mit vorangehendem Token dargestellt.

Füge Eingabezeile in das Programm ein

- Durchsuche das 'tokenisierte' Programm, bis diejenige Zeile gefunden wurde, deren Zeilennummer der in ZNR gespeicherten ist.
- Sind beide Zeilennummern identisch, so wird diese Zeile durch die Eingabezeile ersetzt. Falls der Puffer leer war, wird die angegebene Zeile aus dem Programmtext entfernt.
- Das Einfügen erfolgt nach folgendem Schema : Stelle Länge der Eingabzeile fest, runde hier auf gerade Zahlen auf. Stelle Länge der Programmzeile fest, die ersetzt werden soll (entfällt, falls völlig neue Zeile eingefügt wird) Der Längenunterschied bestimmt in wie weit das Programm ab dieser Programmzeile nach 'oben' bzw. nach 'unten' verschoben werden muß. Nach Verschiebung müssen alle nachfolgenden Zeilenreferenzen neu berechnet werden und in die dafür vorgesehenen Stellen im Programmtext abgelegt werden.

Aufbau der Interpreterschleife

- Teste Einzelschrittflagge, ob Programm in Einzelschritten abgearbeitet werden soll.
- Frage Tastatur ab, ob eine Taste gedrückt wurde.
  Falls ja, Taste mit CTRL-C vergleichen. (CTRL-C führt immer zu einer Programmunterbrechung zwischen zwei BASIC-Befehlen.)
- Hole TOKEN aus dem Programmtext. (TOKEN ist durch einen Wert › $80 gekennzeichnet)
  TOKEN ist gleichzeitig Index auf eine Sprungleiste.
- Lade TOKEN entsprechende Adresse aus der Sprungleiste in den Befehlszähler (= zum Unterprogramm.. springen). Falls kein Token vorhanden ist (Wert ‹$80) => Aufruf des Unterprogrammes LET.
- Führe entsprechendes Unterprogramm aus [1]
- ':' nächstes Zeichen im Programmtext ?
  ja : Beginne wieder von vorne
  nein : hole neue Zeile (setze Programmzeiger auf nächste BASIC-Zeile)
  Falls keine weitere Zeile vorhanden => Programmende.

Programmstart

- Lösche Graphikzeiger für X,Y und Winkel
- Lade Programmzeilenzeiger (Register A2)
- Lade DATAZeilenzeiger
- Lade DATAElementzeiger
- Lade Kellerzeiger (Register A3)
- Lade BASIC-Befehlszeiger (Register A4)
- Lösche Direktmodusflagge
- Rufe Unterprogramm PRE_RUN auf, das alle Sprungmarken und Variablen deklariert.

Programmende

- Schalte Schildkröte aus.
- Lösche Einzelschrittflagge.
- Schliesse noch alle offenen Dateien.
- Schalte den Cursor wieder ein und lege die Blinkfrequenz fest.

[1] Exemplarisch sollen hier noch einige Funktionen beschrieben werden. (CLS, GOTO, INPUT, WRITE)

CLS

Durch Aufruf der Prozedur CLS, soll der Bildschirm gelöscht, auf die Graphikseiten 0 - 1 umgeschaltet und der Cursor eingeblendet werden. Hierzu wird die Funktion sCLRSCREEN des Grundprogramms aufgerufen, die o.g. Eigenschaften erfüllt. Das Unterprogramms CLS wird mit einem Rücksprungbefehl beendet.

GOTO

Sinn des GOTO-Befehls ist es, einen unbedingten Sprung im BASIC-Programm auszuführen, oder das BASIC-Programm selbst aufzurufen, ohne Variablen etc. zu löschen.
Verwirklicht wird dies so :
Zuerst muß festgestellt werden, wohin gesprungen werden soll, d.h. die Zeilennummer muß aus dem Programmtext entnommen werden, hierzu wird eine Funktion aufgerufen, die den Programmtext weiterverarbeitet und die Zeilennummer an das Unterprogramm GOTO zurück liefert. Entstand hierbei ein Fehler, wird die Interpreterschleife mit einer Fehlermeldung verlassen. Sodann wird das Programm nach der angegebenen Zeilennummer durchsucht und der Programmzeiger auf den Beginn dieser Zeile gesetzt; falls die angegebene Zeilennummer nicht vorhanden war, wird die Interpreterschleife ebenfalls mit einer Fehlermeldung abgebrochen. Marken werden hierbei wie Zeilennummern behandelt.
Wurde GOTO aus dem Direktmodus aufgerufen, müssen verschiedene Zeiger, wie bereits oben beschrieben, initialisiert werden. GOTO endet nicht, wie CLS, mit einem Rücksprungbefehl, sondern durch einen direkten Sprung in die Interpreterschleife.

INPUT

verwirklicht die Eingabe während des Programmlaufs in eine durch das Programm bestimmte Variable.
Zunächst muß aber sichergestellt werden, daß wirklich ein Programm abgearbeitet wird und INPUT nicht im Direktmodus aufgerufen wurde.
Daraufhin wird eine eventuelle 'Prompting'-Zeichenkette ausgegeben und eine gesamte Zeile von der Tastatur eingelesen. (ohne Steuerzeichen)
Für die durch das Programm bestimmte Variable wird Platz im Speicher reserviert und die von der Tastatur eingelesene Zeile dieser zugewiesen, d.h. in den reservierten Speicherbereich abgelegt. Hierbei ist zu beachten, daß bei INPUT REAL-, INTEGER-Variablen durch Kommas abgetrennt werden können, STRING-Variablen aber immer als gesamte Zeile zugewiesen werden.
Der INPUT-Befehl kann während seiner Ausführung immer durch Drücken von CONTROL-X abgebrochen werden (Beendet gleichzeitig das Programm).
Bei REAL-,INTEGER-Variablen wird vor der eigentlichen Zuweisung zu einer Variablen das eventuell vorhandene Minuszeichen im Eingabepuffer durch das entsprechende TOKEN ersetzt.

Als letztes wird untersucht, ob im Programmtext weitere Eingaben mit dieser INPUT-Anweisung eingelesen werden sollen, ist dies der Fall wird zunächst untersucht, ob der Eingabepuffer, der die gerade 'abgeschickte' Zeile enthält bereits geleert ist, oder sich noch weitere Zeichen im Puffer befinden, die der nächsten Variablen zugewiesen werden können.
Waren keine Zeichen mehr vorhanden wird eine neue Zeile begonnen und ein Fragezeichen ('?') als Eingabeaufforderung ausgegeben.

WRITE

dient dazu eine Zeichenkette auf einem der 4 Graphikbildschirme in einer bestimmten Zeichengröße auszugeben, ohne auf das durch den Textmodus bestimmtes Raster festgelegt zu sein.
Vorgegangen wird hierbei so :
Teste als erstes für welche Graphikkarte der Befehl gedacht ist, denn ab Vers. 2.4 können alle Graphikfunktionen auch auf der COL256-Karte angesprochen werden. GRMODE liefert hierzu die entsprechende Information.
Werte zunächst die ganzzahligen Ausdrücke für die X,Y-Richtung und die Größe aus. (',' werden als Trennzeichen gefordert) Diese Werte werden zwischengespeichert, um danach die Zeichenkette auszuwerten. (Wie bei fast allen Funktionen/Anweisungen, können statt einer einfachen Zeichenkette ganze Ausdrücke angegeben werden) Die Zeichenkette liegt in einem Puffer vor, der dann mit den entsprechenden Prametern für die WRITE-Funktion des Grundprogramms an der vorbestimmten Position mit der angegebenen Größe ausgegeben wird.

Spezielle Unterprogramme - Funktionen

GETADR

dient dazu diejenige Adresse zu erhalten, ab der der Inhalt einer bestimmten Variable im Speicher abgelegt wurde. Dies ist sicherlich die einfachste Funktion, die GETADR zu erfüllen hat, ist aber nicht die einzigste, denn GETADR muß eine eventuell noch nicht vorhandene Variable, sobald diese das erste mal verwendet wird, in die Symboladresse eintragen, Speicherplatz für sie reservieren und sie mit einem Anfangswert initialisieren. GETADR muß aber auch ein ganzes Feld im Speicher anlegen, falls eine Feldvariable verwendet wurde, die vorher noch nicht dimensioniert wurde. Das Feld wird hierbei mit einer Dimension und Indizes bis 10 versehen, wobei alle Feldelemente auf 'Null' gesetzt werden. Der Aufruf von GETADR erfolgt mit dem Zeiger auf die zu ermittelnde Variable im Register A0.

Nach Beendigung wird die ermittelte Speicheradresse in der Variablen SPVAKT übergeben und der Status des Unterprogrammaufrufs in Register D6 und dem Carry-Flag dem aufrufenden Programm mitgeteilt. A0 zeigt dann auf das nächste zu 'akzeptierende' Zeichen, das nicht mehr verwendet werden konnte.

FWERTS

wertet Stringfunktionen aus, die in einem Stapel abgelegt werden. FWERTS hat die Funktion einen Stringausdruck auszuwerten, also die einzelnen Funktionen, die in einem Stringausdruck auftreten können zu einer Ergebniszeichenkette zusammenzusetzen. Das Einrichten eines Stapels dient nur dazu, die Zeichenkette einfach wieder ausgegeben zu können und dennoch flexibel in der Länge der Zeichenkette zu bleiben. Aufgerufen wird FWERTS mit einem Zeiger auf den auszuwertenden Zeichenkettenausdruck im Register A0. FWERTS liefert nach Beendigung eine Zeichenkette ab der durch STRBOT definierten Speicherstelle zurück. Der Status wird über das Register D6 und das Carry-Flag dem aufrufenden Programm mitgeteilt.

FWERTI bzw. **FWERTR**

werten arithmetische Ausdrücke aus, wobei FWERTI ganzzahl- und FWERTR gleitkommazahl-Ausdrücke auswertet. Aufgerufen werden FWERTI bzw FWERTR mit einem Zeiger auf den auszuwertenden Ausdruck (Register A0). Nach erfolgreicher Berechnung des Ausdruck zeigt das Register A0 hinter den Ausdruck und die Register D0 bzw D0, D1 enthalten das Ergebnis der Berechnung. Der Fehlerstatus wird, wie oben, im Register D6 und dem Carry-Flag gespeichert. Im Fall der FPU-Version wird bei FWERTR natürlich ein GK-Register (FP0) verwendet. Die Auswertung der Ausdrücke erfolgt nach einer vorgegebenen Grammatik für die Ganzzahl- bzw. die Gleitkomma-ausdrücke. Hierbei wird die Methode des rekursiven Abstiegs verwendet. Die Grammatik für die arithmetischen Ausdrücke wird in einem Beiblatt mittels Syntaxdiagramm festgelegt.

FWERTB

kann ähnlich beschrieben werden, wie die o.g. Funktionen FWERTI, FWERTR und FWERTS. FWERTB dient dazu Bool'sche Funktionswerte zu berechnen, wie sie bei IF..THEN und WHILE..WEND Anweisungen eingesetzt werden. Das Aufrufprinzip ist mit dem der o.g. Funktionen identisch, d.h in Register A0 befindet sich ein Zeiger auf den auszuwertenden bool'schen Ausdruck. Als Ergebnis sind hier aber lediglich die beiden Werte WAHR und FALSCH zu erwarten. WAHR wird durch den Wert $FF und FALSCH durch den WERT $00 in Register D0

dargestellt. Eine Besonderheit stellt bei dieser Funktion die Möglichkeit dar, sowohl arithmetische als auch String-Ausdrücke mit ein und derselben Funktion verarbeiten zu können. Dies zwang bei der Implementierung dazu, nur Gleitkomma- und Zeichenketten-Ausdrücke zu berücksichtigen, um eine Trennung von Gleitkomma- und Ganzzahl-Ausdrücken zu unterbinden. Die Statusinformation erhält man wie immer über das Carry-Flag und das Register D6 zurück, in dem sich der Code des evtl. aufgetretenen Fehlers befindet. Für die Auswertung der Bool'schen Ausrücke wurde wieder eine Grammatik herangezogen, die mittels des rekursiven Abstiegs realisiert wurde. (Grammatik ebenfalls auf Beiblatt)

Datenformate

Datenformate bezeichnet die Struktur, in der bestimmte Daten aufgebaut sind.

Unten aufgeführt sind nun alle wichtigen Datenformate, wie z.B der Aufbau einer BASIC-Zeile...

1. Aufbau einer BASIC-Zeile :

Byte:	Bedeutung:
0-3	Adresse der nächsten BASIC-Programmzeile, sie enthält den Wert $0000000, falls keine weitere Programmzeile vorhanden ist. Die Verkettung der Programmzeilen erfolgt durch relative Adressierung.
4-5	Zeilennummer der aktuellen Programmzeile, als Binärzahl kodiert. Zeilennummern dürfen im Bereich von 0 bis 65534 liegen, 65535 ist als Kennzeichen für ein direktes Kommando reserviert.
6 ..	BASIC-Anweisung in 'TOKENisierter' Form. Das Ende einer Programmzeile ist durch $00 gekennzeichnet. Eine neue Zeile beginnt immer auf einer geraden Adresse !

Beispiel :

```
10 PRINT "HALLO"
20 END
```

wird im Speicher in der Form

$.....0 :	00 00 00 0E	Zeiger auf die nächste Programmzeile
$.....4 :	00 0A	Zeilennummer = 10
$.....6 :	91	Token für PRINT
$.....7 :	22 48 41 4C 4C 4F 22	entspricht "HALLO"
$.....E :	00 00 00 16	Zeiger auf die nächste Programmzeile
$....12 :	00 14	Zeilennummer = 20
$....14 :	81	Token für END
$....15 :	00	Ende dieser Programmzeile, $00 wird hier eingefügt, um auf eine gerade Adresse zu kommen. $00 wurde oben nicht eingefügt, da das obere Byte der nächsten Zeile sowieso $00 ist !
$....16 :	00 00 00 00	Kennzeichnet das Programmende abgelegt.

Ein BASIC-Programm wächst im Speicher von Adresse START ab in Richtung der durch die Speicherzelle HIMEM festgelegten Adresse (Stapel) Bei der Wandlung eines BASIC-Programmtextes in seine tokenisierte Form werden ab Version 2.4 alle Konstanten (Integer und Real) in die interne Maschinendarstellung einer Real-Zahl konvertiert, was seinerseits beim Programmablauf einen erheblichen Geschwindigkeitsgewinn mit sich zieht. Allerdings ist hierdurch ein Programm nicht mehr kompatibel zu der entsprechenden Version mit Unterstützung des GK-Prozessors MC 68881, der eine andere Darstellung der GK-Zahlen besitzt. Programme können aber mittels SAVE "‹Programmname›",A und MERGE "‹Programmname›" leicht unter den beiden BASIC-Versionen transferriert werden.

2. Aufbau der Variablen im Speicher :

Es sind grundsätzlich drei Typen von Variablen zu unterscheiden :
1. die einfachen Variablen (keine Felder => keine Indizierung)
2. Feldvariablen.
3. Programmarken

Die Variablen eines BASIC-Programms wachsen von der durch die Speicher zelle HIMEM festgelegten Adresse nach unten in Richtung der durch ENDE festgelegten Adresse (Keller).

Die einzelnen Elemente beanspruchen jeweils einen Speicherplatz von 2,4,8 bzw 128 Bytes für die Typen MARKE,INTEGER, REAL bzw. STRING.

Eine Unterscheidung der Variablennamen erfolgt in allen Stellen eines jeden Variablennamens. In der Symboltabelle des RL-BASICs werden alle Variablennamen mit einem Verweis auf die Speicherzelle, die den Inhalt repräsentiert, abgelegt.
Alle Variablen erhalten ihrem Typ entsprechend einer Kennung im letzten Zeichen. Gegenüber Version 2.0 und vorigen, wurde eine eigene Symboltabellen-Verwaltung integriert, sodaß sich bei Variablenzugriffen ein Geschwindigkeitsgewinn von ca. 30 % ergibt. Im übrigen dürfen Variablennamen beliebig viele Zeichen enthalten, die alle signifikant sind. (Nur durch Zeilenlänge von 246 Zeichen / Zeile beschränkt.)

Die Kennung der einzelnen Variablentypen soll nun angegeben werden :

TYP	Kennung	Beispiel	(Name Name mit Kennung)
Integer	I	IN%	INI
Real	R	RE	RER
String	S	ST$	STS
Marke	L	LOOP!	LOOPL
Integer-Feld	F	INF%(0)	INFF
Real-Feld	G	REF(0)	REFG
String-Feld	Z	STF$(0)	STFZ

A. Aufbau der einfachen Variablen :

i) INTEGER - Variablen werden als 32Bit Zahl abgespeichert, d.h. können einen riesigen Zahlenbereich von 2^31 abdecken.

ii) REAL - Variablen werden als 64Bit Zahl abgespeichert. ab der Version 1.1 mit 32 Bit Mantisse ohne Vorzeichen, 16 Bit Exponenten in 2er Komplementdarstellung + 1 Bit für das Vorzeichen der Gleitkommazahl :

```
6666555555555544444444443333333333222222222211111111100000000000
3210987654321098765432109876543210987654321098765432198765432l0
↑               ↑                       ↑
Vorzeichen      Charakteristik          Mantisse
```

iii) STRING - Variablen (Zeichenketten) benötigen unabhängig von ihrer wahren Größe immer einen Speicherplatz von 128 Bytes. Dies liegt daran, daß hierdurch die Indexrechnung erheblich vereinfacht wird, und eine langsame 'Garbage collection' entfällt! Hierdurch werden aber Zeichenketten auf eine Größe von 126 Zeichen beschränkt, was aber in den meisten Fällen ausreichen dürfte. Das Ende einer Zeichenkette ist durch $00 gekennzeichnet, d.h. $00 kann in einer Zeichenkette nicht explizit auftreten.

B. Aufbau der Feldvariablen :

Feldvariablen unterscheiden sich bezgl. der Indizierung von 'normalen' Variablen. In RL-BASIC ist eine Indizierung für fast beliebige Dimensionen möglich. (z.B. DIM A% (10,10,10,10) = 4 dimensionales Integer-Feld) Der Platzbedarf für ein Feld (Array) setzt sich aus folgenden Punkten zusammen:

1. Platzbedarf für ein einziges Element
2. Platzbedarf für eine Dimension dieses Elements
3. Platzbedarf für jede weitere Dimension = Platzbedarf aller Elemente

Der Gesamtspeicherbedarf errechnet sich nun aus dem Platzbedarf aller Elemente + dem Bedraf für einen Identifikationskopf, der sich wie folgt zusammensetzt :

Byte 0-1 : Anzahl der Dimensionen - 1 (im Beispiel z.B $0003)
Byte 2-5 : Gesamtbedarf aller Elemente in Bytes ! (hier $E4C4)
Byte 6-9 : Bedarf der Elemente in Bytes eine Dimension weniger ($14CC)
Byte A-D : Bedarf der Elemente zwei Dimensionen weniger ($01E4)
.
.
Byte x-x+1 : Bedarf für ein Element in Bytes (hier $0004)

Ein Feld ist auf maximal 65535 Komponenten beschränkt. Bei Integer-Feldern entspricht dies einem Speicherbedarf von maximal 256 KByte, bei Real- Feldern einem Speicherbedarf von max. 512 Kbyte und bei String-Feldern einem maximalen Bedarf von 8 MByte !

3. Aufbau einer DEF FN Benutzerdefinition :

Beispiel :
10 DEF FN F(X) = SIN(X)/X

wird in die Symboltabelle mit FU eingetragen U bezeichnet hierbei die Kennung für benutzerdefinierte Funktionen.
Im Speicher werden 8 Bytes für die Funktionsdefinition reserviert :
Byte 0..3 Zeiger auf die Parametervariable (hier X) absolut.
Byte 4..7 Zeiger auf die Funktion im Programmtext ab '=' (hier SIN(x)/x) absolute Adresse !!
Die Parametervariable wird bei jedem Funktionsaufruf zwischengespeichert, um diese Variable im Programm unabhängig davon verwenden zu können.

Schnittstelle mit dem Diskettenbetriebssystem

RL-BASIC kann prinzipiell alle Diskettenbetriebssysteme unterstützen, die u.a. folgende Routinen zur Verfügung stellen :

1. Inhaltsverzeichnis auf dem Bildschirm ausgeben

2. Binärdatei lesen
3. Binärdatei schreiben
4. Datei verifizieren

5. Datei löschen
6. Datei umbennen

7. Datei öffnen
8. Datei schreiben
9. Datei lesen
10. Datei schließen
11. Länge einer Datei bestimmen
12. Dateiende bestimmen

Die Komunikation mit dem Betriebssystem erfolgt über einen Sprungvektor, der auf einen Verteiler zeigt. Dieser Verteiler muß die o.a. Funktionen realisieren.

Das Protokoll wird nun definert :

a.) Alle Funktionen werden über das Register D7 selektiert
b.) werden Übergabeparameter verlangt, so sind diese in den Registern D0,D1,D2 und A0 zu übergeben. Das Register D6 erhält eine besondere Bedeutung, in ihm wird ein Fehlercode zurückgeliefert, falls ein Fehler während des DOS-Aufrufs aufgetreten ist.

Es folgt nun eine Auflistung wie dieses Protokoll realisiert wird :

Funktion	Register	Eingabe	Ausgabe
Laden eines Progr.	D7.W	$0000	–
	D0.L	Ladeadresse	Endadresse
	A0.L	Zeiger auf Dateiname	–
Vergleichen eines	D7.W	$0001	
Programms	D0.L	Ladeadresse (virtuell)	Endadresse
	A0.L	Zeiger auf Dateiname	–
Inhaltsverzeichnis	D7.W	$0002	–
anzeigen	A0.L	Zeiger auf Dateiname	–
Progr. speichern	D7.W	$0003	–
	D0.L	Startadresse	–
	D1.L	Endadresse	–
	D2.W	Dateikennung [1]	–
	A0.L	zeigt auf Dateiname	–
Datei löschen	D7.W	$0004	–
	A0.L	Dateiname	
Datei umbenennen	D7.W	$0005	–
	A0.L	Zeigt auf Dateinamen [2]	–
Datei öffnen	D7.W	$0006	
	D1.W	Handlenummer [3]	
	A0.L	Dateiname	
Datei schliessen	D7.W	$0007	
	D1.W	Handlenummer [3]	
Datei schreiben	D7.W	$0008	
	D0.L	auszugebendes Zeichen	
	D1.W	Handlenummer [3]	
Datei lesen	D7.W	$0009	eingelesenes Zeichen
	D1.W	Handlenummer [3]	D0.L
Dateiende	D7.W	$000A	Dateiende erreicht ?
	D1.W	Handlenummer [3]	D0.L = $ff
Dateilänge	D7.W	$000B	Dateilänge in Bytes
	A0.L	Dateiname	D0.l
Diskettenplatz	D7.W	$000C	Diskettenplatz in Bytes
ermitteln			D0.l

[1] Dateikennungen : 0 = TXT, 1 = BIN, 2 = BAS
[2] Dateinamen : ‹Dateiname1›,‹Dateiname2›
[3] Handlenummer : Zur Kennung der Datei wird eine Nummer im Bereich 0..2 erwartet

Die Funktionen 0 bis 5 sind in RL-DOS realisiert. Um unter BASIC diese nutzen zu können, muß das DOS einen Sprung- vektor auf den Verteiler richten. Dieser Sprungvektor ist in Adresse $x1C60 angesiedelt. Hier hinein ist ein Sprung auf den Verteiler zu richten. Wird dies versäumt, so erhält $x1C60 durch BASIC den Wert $4E75 (RTS). Die Funktionen sind dann von RL-BASIC nicht ansprechbar.

Gegenüber Version 2.0 sind in der jetzigen einige Änderungen durchgeführt worden: Datei-Kommandos sind hinzugekommen, um Dateien direkt per BASIC zu bearbeiten. Diese Kommandos stehen allerdings nur in dem für JADOS angepassten BASIC zur Verfügung. RL-DOS kann diese nicht unterstützen ! (Funktionen 6 bis $A) Die Funktionen 'Länge einer Datei' und 'Diskettenplatz ermitteln' können auch unter JADOS noch nicht angesprochen werden, bzw. liefern ein ungültiges Ergebnis, da diese Funktionen in JADOS 2.0 noch nicht integriert sind.

Alle von JADOS erkannten Fehler können in BASIC durch eine ON ERROR Routine abgefangen. Eine Anpassung an CP/M 68k soll noch folgen.

Codierung der BASIC-Anweisungen im RL-BASIC

"Token" des RL-BASIC ohne vorangestelltes $FF

80 -	A0 -	C0 - *FIELD*	E0 - *USR*
81 - END	A1 - WIDTH	C1 - *GET*	E1 - FN
82 - FOR	A2 - ELSE	C2 - *PUT*	E2 - SPC(
83 - NEXT	A3 - TRON	C3 - CLOSE	E3 - NOT
84 - DATA	A4 - TROFF	C4 - LOAD	E4 - ERL
85 - INPUT	A5 - SWAP	C5 - MERGE	E5 - ERR
86 - DIM	A6 - ERASE	C6 - FILES	E6 - STRING$
87 - READ	A7 - *EDIT*	C7 - NAME	E7 - USING
88 - LET	A8 - ERROR	C8 - KILL	E8 - INSTR
89 - GOTO	A9 - RESUME	C9 - *LSET*	E9 - TURN
8A - RUN	AA - DELETE	CA - *RSET*	EA - VARPTR
8B - IF	AB - AUTO	CB - SAVE	EB - INKEY$
8C - RESTORE	AC - RENUM	CC - *RESET*	EC - OFF
8D - GOSUB	AD - *DEFSTR*	CD - CLS	ED - MOVE
8E - RETURN	AE - *DEFINT*	CE - LOCATE	EE - DRAW
8F - REM	AF - *DEFSNG*	CF - *BEEP*	EF - ›
90 - STOP	B0 - *DEFDBL*	D0 - SOUND	F0 - =
91 - PRINT	B1 - LINE	D1 - GRMODE	F1 - ‹
92 - CLEAR	B2 - LOMEM	D2 - COLOR	F2 - +
93 - LIST	B3 - HIMEM	D3 - PSET	F3 - -
94 - NEW	B4 - WHILE	D4 - PRESET	F4 - *
95 - ON	B5 - WEND	D5 - CIRCLE	F5 - /
96 - REPEAT	B6 - CALL	D6 - PAINT	F6 - ^
97 - WAIT	B7 - WRITE	D7 - CONNECT	F7 - AND
98 - DEF	B8 - COMMON	D8 - GCURSOR	F8 - OR
99 - POKE	B9 - CHAIN	D9 - *COPY*	F9 - XOR
9A - CONT	BA - *OPTION*	DA -	FA -
9B - CLPG	BB - RANDOMIZE	DB - PAGE	FB -
9C - UNTIL	BC - MOVETO	DC - TO	FC - MOD
9D - OUT	BD - SYSTEM	DD - THEN	FD -
9E - LPRINT	BE - HARDCOPY	DE - TAB(	FE -
9F - LLIST	BF - OPEN	DF - STEP	FF - Funktionen

Tokens der RL-BASIC-Funktionen.
Ihnen ist jeweils ein Byte $FF vorangestellt :

80 -	A0 - CONSTI	C0 -	E0 -
81 - LEFT$	A1 - CONSTR	C1 -	E1 -
82 - RIGHT$	A2 -	C2 -	E2 -
83 - MID$	A3 -	C3 -	E3 -
84 - SGN	A4 -	C4 -	E4 -
85 - INT	A5 -	C5 -	E5 -
86 - ABS	A6 -	C6 -	E6 -
87 - SQR	A7 -	C7 -	E7 -
88 - RND	A8 -	C8 -	E8 -
89 - SIN	A9 - TRUE	C9 -	E9 -
8A - LOG	AA - FALSE	CA -	EA -
8B - EXP	AB - *CVI*	CB -	EB -
8C - COS	AC - *CVS*	CC -	EC -
8D - TAN	AD - *CVD*	CD - CPU	ED -
8E - ATN	AE -	CE - GETAD10	EE -
8F - FRE	AF - EOF	CF - GETAD8	EF -
90 - INP	B0 - LOC	D0 - CSRLIN	F0 -
91 - POS	B1 - LOF	D1 - POINT	F1 -
92 - LEN	B2 - *MKI$*	D2 - DAY	F2 -
93 - STR$	B3 - *MKS$*	D3 - DATE	F3 -
94 - VAL	B4 - *MKD$*	D4 - TIME	F4 -
95 - ASC	B5 - MOUSEX	D5 -	F5 -
96 - CHR$	B6 - MOUSEY	D6 - SCREEN	F6 -
97 - PEEK	B7 - MOUSEK	D7 - DSKF	F7 -
98 - SPACE$	B8 -	D8 -	F8 -
99 - BIN$	B9 -	D9 -	F9 -
9A - HEX$	BA -	DA -	FA -
9B - *LPOS*	BB -	DB -	FB -
9C - *CINT*	BC -	DC - SERINIT	FC -
9D - *CSNG*	BD -	DD - SETDA	FD -
9E - *CDBL*	BE -	DE - RELAIS	FE -
9F - *FIX*	BF -	DF -	FF -

Die *kursiv* markierten Schlüsselworte sind noch nicht integriert, sie stehen nicht zur Verfügung .

Systemadressen :

Systemadressen bezeichnet diejenigen Speicherzellen, die durch RL - BASIC als spezielle Variablenzellen belegt sind.

Bemerkung :

$x.... bezeichnet die Startadresse des Variablengebietes, also bei Grundprogramm ab Adresse $000000 ist für $x.... $8.... einzusetzen, bei verschobenem Grundprogramm z.B. auf Adresse $E0000 ist für $x.... $E8.... einzusetzen, wobei .... eine 16Bit Adresse bezeichnet und immer zur Basisadresse hier z.B. $E8000 hinzuzuaddieren ist. Bei einem 68020 - System z.B. $310000 für $x0000.

Es folgt nun eine Auflistung aller Adressen mit deren Bedeutung :

Symbol:	Adresse:	Bemerkung:
USERCI	$x0018	; Sprungvektor für benutzerdefinierte Eingabe
USERCSTS	$x001E	; Sprungvektor für Test, ob Zeichen vorliegt
USERCO	$x0024	; Sprungvektor für benutzerdefinierte Ausgabe
IOSTAT	$x002A	; Lenke Ausgabe um 1= NIL 2=CRT 3,4=LST 5=USER
IOSTATB	$x002B	; Lenke Eingabe um 6=USER
TURX	$x0046	; Schildkrötenposition X-Koordinate 0..4095
TURY	$x0048	; Schildkrötenposition Y-Koordinate 0..4095
TURPHI	$x004A	; Schildkrötenblickrichtung in Grad 0..359
PCSTAND	$x0052	; enthält die aktuelle Zeilennummer bei Interpreterla
EINBUF	$x0070	; Eingabepuffer für Programmzeilen
AUSBUF	$x00F4	; Ausgabepuffer für Programmzeilen
		; Ein- und Ausgabepuffer werden auch andersweitig
		; benutzt.
LOMEM	$x016C	; Startadresse Arbeitsbereich
HIMEM	$x0170	; Endadresse Arbeitsbereich
START	$x01DE	; RAM-Untergrenze für Programm(1. Programmzeile)
ENDE	$x01F2	; Programmende , ab hier bis HIMEM frei für Var.
ZNR	$x0174	; Zeilennummer aktuelle BASIC-Zeile im EDITOR
ZLEN	$x0176	; Zeilenlänge aktuelle BASIC-Zeile im EDITOR
REGSAVE	$x0178	; Registerzwischenspeicher
FIRST	$x021D	; Kennung Schildkrötengraphik initialisiert
INSFL	$x0221	; Insert-Flag für EDITOR (nicht mehr benutzt)

```
CURX        $x0223   ; aktuelle Cursor-Position in X-Richtung
CURY        $x0224   ; aktuelle Cursor-Position in Y-Richtung

SCREENB     $x0259   ; Start Bildschirmpuffer 80*24
CPU         $x0414   ; Kennung CPU-Typ 1=68008,2=68000,4=68020
TRACEFL     $x1C00   ; Flag für Einzelschrittmodus
DIREKTFL    $x1C02   ; Flag für Direktmodus (<>0 dann Direktmodus)
TOS         $x1C04   ; Top of Stack = Beginn Stack-bereich

AUTOFL      $x1C08   ; Flag für automatische Zeilennummerierung
AUTONR      $x1C0A   ; Zeilennummer der nächsten Zeile bei AUTO
AUTOINC     $x1C0C   ; Schrittweite bei AUTO (i.a. 10)

ONERRFL     $x1C10   ; ON error Anweisung aufgetreten = $FF
ONERRZN     $x1C12   ; Zeilennummer der Fehlerbehandlungsroutine
ONERRS0     $x1C14   ; Aktuelle Position im Programm bei Fehler
ONERRS1     $x1C18   ; Zeilennummer Fehlerhalfter Zeile
ONERRFC     $x1C1C   ; Fehlercode der aufgetreten ist

IFFL        $x1C20   ; If-Anweisung aufgetreten ? (für ELSE)
EOFFL       $x1C22   ; End of File ? (nicht benutzt)

LRPX        $x1C24   ; Last referenz Pointer: Letzte X-Position Graphik
LRPY        $x1C26   ; Last referenz Pointer: Letzte Y-Pos der Graphik
LRPPHI      $x1C28   ; LRP der Schildkröte (Winkel)
AKTCOL      $x1C2A   ; Aktuelle Zeichenfarbe (Weiss oder Schwarz)

BEREICH     $x1C2C   ; Bereich für Zufallszahlengenerator

SPVBOT      $x1C30   ; Zeiger auf Ende Variablenbereich
SPVAKT      $x1C34   ; Zeiger auf aktuelle Adresse im Variablenbereich

HTBEGIN     $x1C38   ; Beginn der Hashtabelle
HTPEGEL     $x1C3C   ; Füllstandsanzeiger der Hashtabelle
STPEGEL     $x1C40   ; Symboltabellenpegel

SERIAL      $x1C44   ; Seriennummer, wird als Aufrufsflag benutzt, um
                     ; festzustellen ob der Arbeitsbereich gesetzt
                     ; werden muss.
DATAZNR     $x1C46   ; Zeile der 1. Datazeile
DATAPTR     $x1C4A   ; Zeiger auf nächstes Datum

SCHREIBS    $x1C4E   ; Aktuelle Schreibseite (0..3)
LESES       $x1C50   ; Aktuelle Leseseite    (0..3)

GR_ENTRY    $x1C52   ; Graphikerweiterung (nicht benutzt)
HC_ENTRY    $x1C5A   ; Sprungvektor zur eigenen Hardcopy - Routine
```

```
DOSENTRY    $x1C60    ; 6 Bytes für Sprungvektor in das DOS
                      ; falls kein JMP-Befehl in Adresse $x1C60 steht
                      ; wird beim Aufruf von BASIC ein RTS eingetragen

TVARFL      $x1C66    ; Trace Flag für auszugebende Variable.
TVARTYP     $x1C68    ; Variablentyp der Variable, die im Einzel-
                      ; schrittmodus angezeigt wird.(0=real,1=int,2=S)
TVARPTR     $x1C6A    ; Zeiger auf Tracevariable

AKKU0       $x1C6E    ; Zwischenspeicher bei arithm. Berechnungen

STRBOT      $x1C78    ; Zeiger auf den Anfang einer Zeichenkette
STRPTR      $x1C7C    ; Zeiger auf aktuelles Ende einer Zeichenkette
                      ; diese beiden Zeiger werden nur bei Zeichen-
                      ; kettenoperationen verwendet.

BUFFER      $x1C80    ; Puffer = Zwischenspeicher (128 Zeichen)
BUFFER2     $x1D00    ; Puffer2 = Zwischenspeicher (128 Zeichen)
BUFFER3     $x1D80    ; Puffer3 = Zwischenspeicher (128 Zeichen)

GDPPTR      $x1E00    ; Basisadresse GDP-I/O-Ports (mit CPU-Kennung Mult.)
TATSPTR     $x1E04    ; Basisadresse Tastaturport

HANDLE_NR   $x1E08    ; Handlenummer beim Öffnen einer Datei
HANDLEINR   $x1E0A    ; Handlenummer beim Lesen aus einer Datei
HANDLEONR   $x1E0C    ; Handlenummer beim Schreiben in eine Datei
RW_FLAG     $x1E0E    ; Kennung, ob Datei zum Lesen oder zum Schreiben
                        geöffnet
IOCB        $x1E20    ; Beginn des IO-Kontrollblocks. Für insgesamt 7
                      ; Dateien wird ein Kontrollblock angelegt. Maximal
                      ; 3 der 7 Dateien können Diskettendateien sein.
NEW_LNR     $x1F40    ; neue Zeilennummer für Zeilenneunummerierung
OLD_LNR     $x1F42    ; Start ab Zeile ..
INCR        $x1F44    ; Schrittweite der Neunummerierung

HVBEGIN     $x1F46    ; Hashtabellenbeginn

CHR_CNT     $x1F4A    ; Zähler für Ausgabe von Zeichen
CHR_WIDTH   $x1F4C    ; Ausgabegerät Breite
PRE_RUNFL   $x1F4E    ; Merker, ob PRE_RUN noch gültig

CPU_P       $x1F50    ; CPU-TYP 1 = 68008,2=68000,4=68020
CRT_P       $x1F52    ; COL256-Karte
CRTD_P      $x1F56    ; IO-Adresse Register 6845
CRTB_P      $x1F5A    ; Port-Adresse fuer Seitenauswahl COL256
COLRAM_P    $x1F5E    ; Start Bildschirmspeicher COL256
GR_MODE     $x1F62    ; Graphikmodus : 0 = GDP   1 = COL256
XORMODE     $x1F64    ; Schreibmodus bei COL256-Graphikbefehlen (n.u.)
```

```
X1          $x1F66   ; X-koordinate 1. Punkt für Linie
Y1          $x1F6A   ; Y-Koordinate 1. Punkt für Linie
X2          $x1F68   ; X-koordinate 2. Punkt für Linie
Y2          $x1F6C   ; Y-Koordinate 2. Punkt für Linie
FLAGOBEN    $x1F6E   ; Merkvar für Füllalgo
FLAGUNTEN   $x1F70   ; Merkvar für Füllalgo
COLPAINT    $x1F72   ; Füllfarbe
LIST_MODE   $x1F74   ; List-Modus : 0 = Normal, 1= mit aut. Einrücken
OLD_ST      $x1F76   ; Schachtelungstiefe letzte Zeile
NEW_ST      $x1F78   ; Schachtelungstiefe aktuelle Zeile
AKT_Z_ST    $x1F7A   ; Zeiger auf neue Zeile, in der Sechachtelungst.
                     ; berechnet werden soll.
GET_STFL    $1F7E    ; Merker, daß GET_STAKT-Routine aufgerufen wurde.
LOX_P       $1F80    ; Zeiger für IO-Port HC-MAUS
HIX_P       $1F84
LOY_P       $1F88
HIY_P       $1F8C
UP_P        $1F90
DOWN_P      $1F94
LEFT_P      $1F98
RIGHT_P     $1F9C
MOUSE_X     $1FA0    ; x-Koordinate der Maus
MOUSE_Y     $1FA2    ; y-Koordinate der Maus
GCURSOR_X   $1FA4    ; x-Koordinate Fadenkreuz
GCURSOR_Y   $1FA6    ; y-Koordinate Fadenkreuz
```

ABS X=ABS (N) FUNKTION

Syntax: X = ABS (<Numerischer Ausdruck>)
Effekt: Gibt den absoluten Wert einer Zahl aus. (Wert immer Ø oder positiv)

AND / OR / NOT WERT% = PEEK(PORT%) AND %1Ø11 FUNKTION

Syntax: <numerischer Ausdruck 1> AND <numerischer Ausdruck 2>
<numerischer Ausdruck 1> OR <numerischer Ausdruck 2>
NOT <numerischer Ausdruck>
Effekt: Berechnet ein durch bitweise Verknuepfung entstandenes Ergebnis.
Als Verknuepfung stehen zur Verfuegung: AND, OR, NOT

ARC ARC (X,Y),RAD,START,END ANWEISUNG

Syntax: ARC (<Mittelpunkt-X>,<mid-Y>),<Radius>,<Startwinkel>,<Endwinkel>
Effekt: Zeichnet einen Kreisbogen

ASC I%=ASC(A$) FUNKTION

Syntax: I% = ASC (<String Ausdruck>)
Effekt: Gibt den ASCII-Wert des ersten Zeichens einer Zeichenkette (String) aus.

AUTO AUTO oder AUTO 5Ø,25 oder AUTO ,2Ø oder AUTO 5Ø BEFEHL

Syntax: AUTO [<erste Zeilennummer>][,<Erhoehung>]
Effekt: Erstellt fuer jedes Betaetigen der Return-Taste eine Zeilennummer.
Mit CTRL-C wird AUTO abgeschaltet. Eine Zeilennummer darf nicht groesser
als 65354 sein.

**BIN$** X$ = BIN$(2Ø) FUNKTION

Syntax: X$ = BIN$ (<numerischer Ausdruck>)
Effekt: Gibt den numerischen Ausdruck (Ø..255) in binaerer Schreibweise aus .

CALL CALL $1ØØØØ ANWEISUNG

Syntax: CALL <Startadresse>
Effekt: Gibt die Programmkontrolle an eine Maschinensprache-Unterroutine ab.

CHAIN CHAIN "NeuProg" ANWEISUNG

Syntax: CHAIN "<Dateiname>"
Effekt: Uebergibt die Kontrolle und uebertraegt Variablen an ein anderes Program

CHR$ A$=CHR$(97) FUNKTION

Syntax: A$= CHR$(<numerischer Ausdruck>)
Effekt: Gibt das Zeichen aus, dessen ASCII-Wert der eingegebenen Dezimalzahl, die zwischen Ø und 255 liegen muss, entspricht.

CIRCLE CIRCLE (X,Y),RADIUS ANWEISUNG

Syntax: CIRCLE [STEP](<horizont. Mittelpunkt, vertik. Mittelpunkt>),<Radius>[,F]
Effekt: Zeichnet einen Kreis mit Radius r an Position x,y.

CLEAR CLEAR ANWEISUNG

Syntax: CLEAR
Effekt: Bereinigt den mit Programmdaten belegten Arbeitsspeicher, ohne dadurch das derzeit im Arbeitsspeicher befindliche Programm zu loeschen.

CLOSE CLOSE #1 Anweisung

Syntax: CLOSE [#<dateinummer>]
Effekt: Schliesst eine oder alle zuvor geoeffnete(n) Datei(en).

CLPG CLPG ANWEISUNG

Syntax: CLPG
Effekt: Loescht die aktuelle Schreibseite.

CLS CLS ANWEISUNG

Syntax: CLS
Effekt: Loescht die Schreibsseiten Ø und 1 zur Textausgabe und postioniert, den Cursor links oben.

COLOR COLOR = Ø oder COLOR = 1 ANWEISUNG

Syntax: COLOR = <Farbe>
Effekt: Stellt die Zeichenfarbe ein. Ø entspricht Schwarz, 1 entspr. Weiss.

CONNECT CONNECT (X1,Y1)-(X2,Y2)-(X3,Y3) ANWEISUNG

Syntax: CONNECT [STEP](<Start X>,<Start Y>)-[STEP](<Ziel X>,<Ziel Y>)...
Effekt: Verbindet Linien von Anfangskoordinate mit Endkoordinaten. Mit STEP werden Linien relativ zur letzten Zeichenposition gezeichnet.

COS X = COS(Y) FUNKTION

Syntax: X = COS (<numerischer Ausdruck>)
Effekt: Berechnet den Cosinus einer Zahl .

CONT CONT BEFEHL

Syntax: CONT
Effekt: Nimmt einen durch BREAK (CTRL-C) oder STOP unterbrochenen Programmlauf wieder auf.

CPU WAIT $FFFFFF68 * CPU, 32 FUNKTION

Syntax: CPU
Effekt: liefert den aktuellen CPU-Typ zurueck.

DATA DATA 25,15,925,"WORT" ANWEISUNG

Syntax: DATA ‹Konstante›[,Konstante]...
Effekt: Definiert eine Liste von Konstanten, die eine READ-Anweisung Variablen zuordnen kann.

**DATE$** DATE$ = "10.12.85" oder PRINT DATE$ FUNKTION

Syntax: DATE$ = "DD.MM.JJ" oder A$ = DATE$
Effekt: Setzen oder Zuruecklesen des Datums. (Bei Uhrenkarte)

DEF FN DEF FNY (X) = X*2 + 5 ANWEISUNG

Syntax: DEF FN‹Funktionsname› (‹Parameter›) = ‹Definition›
Effekt: Definiert anwenderspezifische Funktionen.

DELETE DELETE 20 - 30 BEFEHL

Syntax: DELETE ‹Zeilennummer Liste›
Effekt: Loescht eine oder mehrere Programmzeilen aus dem Arbeitsspeicher.

DIM DIM A$(5) oder DIM X (5,10,4) ANWEISUNG

Syntax: DIM ‹Feldname› (‹Unterdefinition›[,Unterdefinition]..)
Effekt: Definiert die Anzahl der Dimensionierung und die Zahl der Elemente in einem Feld.

DRAW TO DRAW TO 200,0 ANWEISUNG

Syntax: DRAWTO ‹X-Pos›,‹y-Pos›
Effekt: Zeichnet eine Linie von der letzten Zeichenposition zum Koordinatenpunkt ‹x-Pos›,‹y-pos›

END END ANWEISUNG

Syntax: END
Effekt: Beendet einen Programmlauf und kehrt zum Befehls-Level zurueck.

EOF IF EOF (1) THEN GOTO 110 — FUNKTION

Syntax: EOF (<Dateinummer>)
Effekt: Zeigt das Ende einer sequentiellen Datei an.

ERASE ERASE A$ — ANWEISUNG

Syntax: ERASE <Feldname>
Effekt: Setzt ein ganzes Feld auf Ø.

ERL, ERR X = ERL oder X = ERR — FUNKTION

Syntax: X = ERL oder X = ERR
Effekt: Die ERL- und ERR- Variablen sind reservierte Variablen, die in Unterroutinen fuer die Fehlerbehandlung verwendet werden koennen.

ERROR ERROR X — ANWEISUNG

Syntax: ERROR <numerischer Ausdruck>
Effekt: Simuliert einen BASIC Laufzeitfehler und uebergibt die Kontrolle an eine Fehleraufdeckungsroutine (falls Vorhanden).

EXP X = EXP (Y) — FUNKTION

Syntax: X = EXP (<numerischer Ausdruck>)
Effekt: Berechnet die Y-te Potenz der Konstanten e.

FILES FILES — BEFEHL

Syntax: FILES [<Laufwerk>]
Effekt: Gibt das Inhaltsverzeichnis der sich im Laufwerk befindlichen Diskette aus.

FOR FOR I = 1 TO 5 STEP 2 — ANWEISUNG

Syntax: FOR <Zaehl-Variable>=<num. Ausdr.> TO <num. Ausdr.> [STEP <num. Ausdr.>]
Effekt: Erstellt eine Schleife, die so oft ausgefuehrt wird, wie angegeben wurde

FRE X = FRE (Ø) — FUNKTION

Syntax: X = FRE (<Test-Argument>)
Effekt: Gibt die Anzahl nicht verwendeter Bytes im Arbeitsspeicher aus.

GETAD8 Y = GETAD8 (Ø) — FUNKTION

Syntax: GETAD8 (<Kanalnummer>)
Effekt: Liest einen digitalisierten Wert von der Baugruppe AD8*16 ein.

GETAD1Ø Y = GETAD1Ø FUNKTION

Syntax: GETAD1Ø
Effekt: Liest einen digitaliesierten Wert von der Baugruppe AD1Ø*1 ein.

GOSUB GOSUB 25Ø ANWEISUNG

Syntax: GOSUB ‹Zeilennummer›
Effekt: Gibt die Programmkontrolle an eine Unterroutine ab.

GOTO GOTO 5Ø ANWEISUNG

Syntax: GOTO ‹Programmzeilennummer› ¦ ‹Marke›
Effekt: Fuehrt einen unbedingten Sprung im Programmablauf aus.

GRMODE GRMODE 1 ANWEISUNG

Syntax: GRMODE Ø ¦ 1
Effekt: Waehlt zwischen GDP64k und COL256-Graphik aus.

HARDCOPY HARDCOPY ANWEISUNG

Syntax: HARDCOPY
Effekt: Erzeigt einen aufruf ueber einen Sprungvektor, so dass
eine eigene Hardcopy-Routine eingebunden werden kann.

HEX$ A$ = HEX$(Y%) FUNKTION

Syntax: A$ = HEX$ (‹numerischer Ausdruck›)
Effekt: Gibt eine Zeichenkette aus, die dem Hexadezimalwert einer Zahl entspr.

IF IF X=Y THEN PRINT A: GOTO 25Ø ELSE GOTO 3Ø ANWEISUNG

Syntax: IF ‹logischer Ausdruck› THEN ‹Anweisung 1›¦‹Zeilennummer›
[ELSE ‹Anweisung 2›]
Effekt: Stellt Bedingungen auf, die den Programmablauf festlegen.

IF ... END IF Nur im Programm auf mehrere Zeilen verteilt erlaubt ANWEISUNG

Syntax: IF ‹Bool'scher Ausdruck› THEN
‹Anweisungen fuer WAHR-Teil›
ELSE
‹Anweisungen fuer FALSCH-Teil›
END IF
Effekt: Fallunterscheidung, bedingte Anweisung.
Erlaubt auf eine Bedingung hin weitere Anweisungen mit dieser zu
verbinden.

**INKEY$** A$ = INKEY$ FUNKTION

Syntax: A$ = INKEY$
Effekt: Liefert den Tastencode zurueck, falls eine Taste gedrueckt wurde. Wurde bei Abarbeitung dieses Befehles keine Taste gedrueckt, so wird ein Leerstring ("") zurueckgeliefert.

INP C = INP(1) FUNKTION

Syntax: INP (<Dateinummer>)
Effekt: Liest ein Zeichen (Byte) aus der zum Lesen geoeffneten Datei.

INPUT INPUT A$ oder INPUT "Name :";A$ ANWEISUNG

Syntax: INPUT [<Prompt-String>;]<Variable>[,Variable]
Effekt: Ermoeglicht Dateneingabe waehrend des Programmlaufs und ordnet diese Daten den Programm-Variablen zu.

INPUT # INPUT #1,A$ ANWEISUNG

Syntax: INPUT # <Dateinummer>,<Variable>[,<Variable>] ...
Effekt: Diese Anweisung wird benutzt, um Daten aus einer sequentiellen Datei in Variablen einzulesen.

INSTR X = INSTR (3,A$,"DO") oder X= INSTR(A$,B$) FUNKTION

Syntax: X = INSTR ([<Anfangspunkt>,<Zielstring>,<Musterstring>)
Effekt: Sucht eine Zeichenkette innerhalb eines anderen String und gibt deren Position aus.

INT X = INT (Y) FUNKTION

Syntax: X = INT (<numerischer Ausdruck>)
Effekt: Wandelt eine REAL-Zahl in eine Integer-Zahl um.

KILL KILL "DEMO1" ANWEISUNG

Syntax: KILL "<Dateiname>"
Effekt: Loescht eine Disketten Datei.

**LEFT$** X$ = LEFT$ (A$,5) FUNKTION

Syntax: X$ = LEFT$ (<Zielstring>,<Numersicher Ausdruck>)
Effekt: Gibt eine Zeichenkette aus, die die ersten Zeichen, gerechnet von links in einem String enthaelt.

LEN Z = LEN (A$) FUNKTION

Syntax: Z = LEN (<String-Ausdruck>)
Effekt: Gibt die Laenge einer Zeichenkette aus.

LET LET X = Y oder LET X(1) = Y ANWEISUNG

Syntax: LET <Variable> = <Ausdruck>
Effekt: Orndet einen Wert einer Variablen oder Feldvariablen zu.

LINE LINE (X1,Y1)-(X2,Y2) ANWEISUNG

Syntax: LINE [STEP](<Start X>,<Start Y>)-[STEP](<Ziel X>,<Ziel Y>)
Effekt: Zeichnet eine Linie von Anfangskoordinate nach Endkoordinate. Mit STEP wird diese Linie relativ zur letzten Zeichenposition gezeichnet.

LIST LIST oder LIST 10-50 oder LIST 20 ANWEISUNG

Syntax: LIST [<Zeilennummer>][-<Zeilennummer>]
Effekt: Gibt ein Programm-Listing auf den Bildschirm aus.

LLIST LLIST oder LLIST 10-30 oder LLIST 40 ANWEISUNG

Syntax: LLIST [<Zeilennummer>][-<Zeilennummer>]
Effekt: Listet das Programm fuer den Drucker auf.

LOAD LOAD "DEMO" oder LOAD K,"DEMO" ANWEISUNG

Syntax: LOAD [K,]"<Dateiname>"
Effekt: Laedt eine Programmdatei von Diskette oder Kassette in den Arbeis-speicher.

LOCATE LOCATE 10,40 ANWEISUNG

Syntax: LOCATE <y-Pos>,<X-Pos>[,OFF]
Effekt: Positioniert den Cursor an Position <y-Pos>,<x-Pos>. Die Option OFF schaltet dabei den Cursor aus.

LOG X = LOG (N) FUNKTION

Syntax: X = LOG (<numerischer Ausdruck>)
Effekt: Gibt den natuerlichen Logarithmus einer Zahl aus.

LPRINT LPRINT A$;" = ";X oder LPRINT USING F$;A$,x ANWEISUNG

Syntax: LPRINT [<Liste mit Ausdruecken> bzw LPRINT USING <Format String>;<Liste>
Effekt: Leitet die Ausgabe an den Drucker (siehe auch PRINT).

MERGE MERGE "Program.lst" ANWEISUNG

Syntax: MERGE ‹Dateiname›
Effekt: Bindet ein, als Text abgespeichertes, Basic-Programm an das im Speicher befindliche Programm.

**MID$** MID$ (A$,3)= B$ ANWEISUNG

Syntax: MID$ (‹String-Var›,‹Anfangspunkt›)
Effekt: Ersetzt einen Teil einer Zeichenkette durch einen anderen.

**MID$** A$ = MID$(B$,3,2) FUNKTION

Syntax: A$ = MID$(‹String-Ausdruck›,‹Startpos›,‹Laenge›)
Effekt: Gibt ein Segment einer Zeichenkette aus.

MOD IF IT% MOD 2=1 THEN..... FUNKTION

Syntax: ‹numerischer Ausdruck 1› MOD ‹numerischer Ausdruck 2›
Effekt: Berechnet den durch eine ganzzahlige Division entstehenden Rest. ‹numerischer Ausdruck 1› wird durch den ‹numerischer Ausdruck 2› dividiert.

MOUSEX, MOUSEY, MOUSEK PSET MOUSEX , MOUSEY FUNKTION

Syntax: MOUSEX bzw. MOUSEY oder MOUSWEK
Effekt: Liefert die Mausposition bzw. die Tastenwerte der Maus zurueck, falls sich eine HARDCOPY/MAUS-Baugruppe im System befindet.

MOVE MOVE 1Ø ANWEISUNG

Syntax: MOVE ‹Schrittzahl›
Effekt: MOVE ist ein Turtlegraphik-Befehl, der die Schlidkroete um ‹Schrittzahl› in die aktuelle Blickrichtung schreiten laesst.

MOVETO MOVETO 2ØØ,255 ANWEISUNG

Syntax: MOVETO ‹X-Pos›,‹y-Pos›
Effekt: Positioniert die Schildkroete an die Stelle ‹X-Pos›,‹y-Pos›.

NAME NAME "ALTPROGRAMM","NEUPROGRAMM" ANWEISUNG

Syntax: NAME "‹Dateiname alt›","‹Dateiname neu›"
Effekt: Benennt eine Disketten-Datei neu.

NEW NEW BEFEHL

Syntax: NEW
Effekt: Loescht das Programm und die Variablen im Arbeitsspeicher.

NEXT NEXT X ANWEISUNG

Syntax: NEXT [<Zaehlvariable>]
Effekt: Markiert das Ende einer FOR/NEXT-Schleife.

ON ON X GOTO 10,20,30 oder ON X GOSUB 300,400,500 ANWEISUNG

Syntax: ON <Numerischer Ausdruck> GOTO <Zeilenangabe>,<Zeilenangabe>.. oder
ON <Numerischer Ausdruck> GOSUB <Zeilenangabe>,<Zeilenangabe>..
Effekt: Uebergibt die Programmkontrolle an eine Programmzeile in einer Auflistung, abhaengig vom errechneten Ergebnis des numerischen Ausdruckes.

ON ERROR GOTO ON ERROR GOTO 1000 ANWEISUNG

Syntax: ON ERROR GOTO <Zeilennummer>
Effekt: Ermoeglicht die Aufdeckung eines Laufzeit-Fehlers und uebergibt die Kontrolle an eine Zeilennummer, sobald ein Fehler auftritt.

OPEN OPEN "O",#1, "test.txt" ANWEISUNG

Syntax: OPEN "<Modus>",#<Dateinummer>,<Dateiname>
Effekt: Oeffnet eine sequentielle Datei zum Leser oder Beschreiben.

OUT OUT #1,POINT (IX,IY) ANWEISUNG

Syntax: OUT #<Dateinummer>,<Wert>
Effekt: OUT gibt ein Zeichen (Byte) im Bereich von ASII 00..$FF auf die Datei mit der Dateinummer .. aus.

PAGE PAGE 3,3 ANWEISUNG

Syntax: PAGE <Schreibseite>,<Leseseite>
Effekt: Waehlt eine der 4 Seiten als Schreibseite und eine der 4 Seiten als Leseseite aus. Wobei die Schreibsseite nicht unbedingt gleich der Leseseite sein muss.

PAINT PAINT ((X1+X2)/2),((Y1+Y2)/2),C ANWEISUNG

Syntax: PAINT (<X-Kooerdinate>,<Y-Koordinate>),<Farbe>
Effekt: Fuellt einen Bildschirmbereich mit der aktuellen Zeichenfarbe ab Position x,y bis die Grenzfarbe <Farbe> erreicht wird.

PEEK X% = PEEK(Y) FUNKTION

Syntax: X% = PEEK (<Speicheradress>)
Effekt: Gibt den Inhalt einer Speicheradresse aus.

POINT OUT #1,POINT (IX,IY) FUNKTION

Syntax: POINT (<X-Koordinate>,<Y-Koordinate>)
Effekt: Liefert den Farbwert an Position x,y zurueck.

POKE POKE 1565,X ANWEISUNG

Syntax: POKE <Speicheradresse>,<Numerischer Ausdruck>
Effekt: Schreibt ein Daten-Byte in eine Speicher-Adresse

POS X = POS FUNKTION

Syntax: X = POS
Effekt: Gibt die derzeitige Cursorposition am Bildschirm aus.

PRESET PRESET 100,200 ANWEISUNG

Syntax: PRESET [STEP]<X-POS>,<Y-POS>
Effekt: Setzt einen Punkt an Position <x-Pos>,<y-Pos> auf dem Bildschirm zurueck

PRINT PRINT X,Y oder PRINT X;Y oder PRINT A$ ANWEISUNG

Syntax: PRINT [<Ausdruck><, oder ;><Ausdruck>[<, oder ;>]
Effekt: Druckt Daten auf den Bildschirm aus.

PRINT USING PRINT USING FORM$; X,Y,Z oder PRINT#1, USING FORM$,X ANWEISUNG

Syntax: PRINT USING <Format-String>,<Ausdruck>[,<Ausdruck>..]
Effekt: Druckt eine Ausgabe gemaess dem vorgegebenen Format.

PRINT # PRINT #1,A$ ANWEISUNG

Syntax: PRINT #<Dateinummer>,[<Liste von Ausdruecken>]
Effekt: Dieser Befehl wird zur Ausgabe von Daten in eine sequentielle
Datei verwendet.

PSET PSET 100,100 ANWEISUNG

Syntax: PSET [STEP]<X-POS>,<Y-POS>
Effekt: Setzt einen Punkt an Position <x-Pos>,<y-Pos> auf dem Bildschirm.

RANDOMIZE RANDOMIZE X ANWEISUNG

Syntax: RANDOMIZE [<numerischer Ausdruck>]
Effekt: Setzt den Random-Zahlengenerator.

READ READ A$ ANWEISUNG

Syntax: READ <Variable>
Effekt: Ordnet Werte aus einer Data-Anweisung Variablen zu.

RELAIS RELAIS ON oder RELAIS OFF ANWEISUNG

Syntax: RELAIS ON ¦ OFF
Effekt: Schaltet das Relais der Kassetten-Schnittstelle an oder aus.

REM REM "Anmerkung oder REM ANMERKUNG ANWEISUNG

Syntax: REM <Anmerkung>
Effekt: Ermoeglicht Anmerkungen im Programmtext.

REPEAT .. UNTIL REPEAT... UNTIL INKEY$="a" ANWEISUNG

Syntax: REPEAT
.
.

UNTIL <Bool'scher Ausdruck)
Effekt: Wiederholt die Anweisungen innerhalb des REPEAT .. UNTIL Blockes so oft, bis <Bool'scher Ausdruck> wahr wird.

RESTORE RESTORE 200 ANWEISUNG

Syntax: RESTORE [<Zeilennummer>]
Effekt: Setzt Data-Zeiger auf bestimmte Zeilennummer bzw. Liest Data-Anweisungen neu.

RESUME RESUME oder RESUME NEXT oder RESUME 200 ANWEISUNG

Syntax: RESUME oder RESUME NEXT oder RESUME <Zeilennummer>
Effekt: Faehrt nach einem Fehler mit dem Programmlauf fort.

RETURN RETURN ANWEISUNG

Syntax: RETURN
Effekt: Uean die Anweisung, die dem letzten GOSUB folgte.

**RIGTH$** A$= RIGTH$(B$,3) FUNKTION

Syntax: A$ = RIGHT$ (<Zielstring>,<Anzahl Zeichen>)
Effekt: Gibt die letzten Zeichen, gerechnet von rechts, der Zeichenkette aus.

RND X = RND FUNKTION

Syntax: X = RND
Effekt: Generiert eine zufaellig gewaehlte Zahl.

RUN RUN oder RUN 200 oder RUN "DEMO" ANWEISUNG

Syntax: RUN [<Zeilennummer> ¦ <Marke>] oder RUN <Dateiname>
Effekt: Startet einen Programmlauf.

SAVE SAVE "DEMO" oder SAVE K,"DEMO" ANWEISUNG

Syntax: SAVE [K,] <Dateiname> oder SAVE <Dateiname>,A
Effekt: Speichert das, sich im Arbeitsbereich befindliche, Programm
auf Diskette oder Kassette ab.

SCREEN a$ = SCREEN X,Y FUNKTION

Syntax: A$=SCREEN <Spalte>,<Zeile>
Effekt: liefert das Zeichen auf Position <Spalte>,<Zeile> des
Textbildschirmes.

SETDA SETDA W1%,W2% ANWEISUNG

Syntax: SETDA <Wert Kanal1>,<Wert Kanal2>
Effekt: Gibt zwei Werte auf die Digital-Analog-Wandler Baugruppe aus.

SERINIT SERINIT $0B,$1E ANWEISUNG

Syntax: SERINIT <Kontroll-Byte>, <Baudrate>
Effekt: Initialisiert die serielle-Schnittstelle.

SGN X = SGN (Y) FUNKTION

Syntax: X = SGN (<numerischer Ausdruck>)
Effekt: Gibt das Vorzeichen einer Zahl aus (-1 = negativ, 1 = positiv ,0 = 0).

SIN X = SIN (Y) FUNKTION

Syntax: X = SIN (<numerischer Ausdruck>)
Effekt: Berechnet den Sinus eines Argumentes, das in Radian angegeben ist, aus.

SOUND SOUND $DE,1,$DD,1,$BE,0,0,$F8,$10,$10,$10,0,$A,8,0,0 ANWEISUNG

Syntax: SOUND <Parameter1>,....<Parameter 16>
Effekt: Uebergibt die Parameter der Baugruppe SOUND, die zur Ausgabe eines
Geraeusches notwendig sind.

SPACE$ X$ = SPACE$(Y) FUNKTION

Syntax: X$ = SPACE$ (<numerischer Ausdruck>)
Effekt: Gibt einen String mit Leerzeichen aus.

SPC PRINT SPC (10) FUNKTION

Syntax: PRINT SPC (<Numerischer Ausdruck>)
Effekt: Gibt Leerzeichen in einer PRINT-Anweisung aus.

SQR SQR(LK*LK+LAK*LAK) FUNKTION

Syntax: SQR (<numerischer Ausdruck>)
Effekt: Berechnet die 2. Wurzel aus <numerischer Ausdruck>

STOP STOP ANWEISUNG

Syntax: STOP
Effekt: Bricht den Progammlauf ab und kehrt in den Befehls-Level zurueck.

STR$ X$=STR$(Y) FUNKTION

Syntax: X$ = STR$ (<numerischer Ausdruck>)
Effekt: Gibt einen String aus, der das dem Dezimalzeichen entsprechende Argument enthaelt.

STRING$ X$ = STRING$ (10,"A") FUNKTION

Syntax: X$ = STRING$ (<Numerischer Ausdruck>,<String-Ausdruck>)
Effekt: Gibt einen String mit der angegebenen Laenge aus. Die Zeichen werden durch das zweite Argument definiert.

SWAP SWAP X,Y ANWEISUNG

Syntax: SWAP <erste Variable>,<zweite Variable>
Effekt: Vertauscht die Werte zweier Variablen.

SYSTEM SYSTEM BEFEHL

Syntax: SYSTEM
Effekt: Verlaesst BASIC und kehrt ins Grundprogramm / Betriebssystem zurueck.

TAB PRINT TAB (Y) FUNKTION

Syntax: PRINT TAB (<numerischer Ausdruck>)
Effekt: Bewegt den Cursor an eine angegeben Tabulatorposition.

TAN X = TAN (Y) FUNKTION

Syntax: X = TAN (<Winkel in Radian>)
Effekt: Gibt den Tangens eines Winkels aus.

**TIME$** TIME$ = "24:12:40" ANWEISUNG

Syntax: TIME$ = "HH:MM:SS"
Effekt: Setzt die Systemuhr, wenn eine solche als Steckkarte vorhanden ist.

**TIME$** A$ = TIME$ FUNKTION

Syntax: A$ = TIME$
Effekt: Gibt die aktuelle Uhrzeit aus.

TRON TRON oder TRON,S ANWEISUNG

Syntax: TRON [,<Variable>]
Effekt: Verfolgt selektiv den Programmlauf Zeile fuer Zeile und druckt die Zeilennummern und Anweisungen aus, die als naechstes ausgefuehrt werden.

TROFF TROFF ANWEISUNG

Syntax: TROFF
Effekt: Widerruft den Befehl TRON.

TRUE / FALSE WHILE TRUE FUNKTION

Syntax: TRUE bzw. FALSE
Effekt: liefert die Bool'schen Konstanten WAHR bzw. FALSCH.

TURN TURN 30 ANWEISUNG

Syntax: TURN <Winkelgrad>
Effekt: Dreht die Schildkroete um <Winkelgrad> Grad in mathematisch positivem Sinn relativ zu ihrer aktuellen Blickrichtung.

TURN TO TURN TO 90 ANWEISUNG

Syntax: TURN TO <Winkelgrad>
Effekt: Dreht die Schildkroete in die Richtung <Winkelgrad>.

VERIFY VERIFY "DEMO" oder VERIFY,K "DEMO" ANWEISUNG

Syntax: VERIFY "<Dateiname>" oder VERIFY,K "<Dateiname>"
Effekt: Vergleicht die angegebene Datei (Programm) mit dem sich im Arbeitsspeicher befindliche Programm. (K steht fuer Kassette)

VAL X = VAL (A$) FUNKTION

Syntax: X = VAL (<String-Ausdruck>)
Effekt: Wandelt den String in eine Realzahl um.

VARPTR X = VARPTR(Y) FUNKTION

Syntax: X = VARPTR (<Variable>)
Effekt: Gibt die Adresse einer Variablen aus.

WAIT WAIT $FFFFFF68,20 ANWEISUNG

Syntax: WAIT <Speicher-Adresse>,<Wert>
Effekt: Haelt das Programm an und wartet darauf, dass die angegebene Speicheradresse den spezifizierten Wert enthaelt.

WEND WEND ANWEISUNG

Syntax: WEND
Effekt: Signalisiert das Ende einer WHILE/WEND Schleife.

WHILE WHILE A<B ANWEISUNG

Syntax: WHILE <logischer Ausdruck>
Effekt: Stellt eine Bedingung auf, die eine While/Wend-Schleife steuert.

WRITE WRITE X,Y,$33,"Hallo" ANWEISUNG

Syntax: WRITE <X-Koord.>,<Y-Koor.>,<Schriftgroesse>,<Stringausdruck>
Effekt: Gibt den Stringausdruck an der Position X,Y in der Schriftgroesse S auf den Bildschirm aus.

I N H A L T S V E R Z E I C H N I S S