RL-BASIC

für NDR-Klein-Computer

V 2.0

Graf Elektronik Systeme GmbH

I N H A L T S Ü B E R S I C H T

Vorbemerkung

Es ist nun mittlerweile ein dreiviertel Jahr seit Beginn der Programmierarbeiten für RL-BASIC vorüber. Sie erhalten mit RL-BASIC eine sehr leistungsfähige BASIC-Variante für den NDR-Klein-Computer mit 680xx CPU.

Es stehen über 100 verschiedene Befehle/Anweisungen/Funktionen zur Verfügung. Darunter Graphik-Befehle, Anweisungen zum Stellen und Lesen der Uhr (wenn Uhrenkarte vorhanden ist), nützliche Befehle zur Ausgabe-Formatierung wie PRINT USING usw.

Bei der Konzeption dieses BASIC wurde strikt auf Kompatibilität mit schon vorhandenen BASIC-Interpretern von Microsoft und Digital-Research geachtet, sodaß es ein Leichtes sein dürfte, Programme an RL-BASIC anzupassen.
Wo es möglich war, wurden Befehle implementiert, die die spezielle Hardware des NDR-Klein-Computers unterstützen. (Z.B. die meisten Graphikbefehle)

Von der Realisierung von Dateibefehlen für Sequentielle und Random-Dateien wurde vorläufig noch abgesehen, da bis jetzt kein einheitliches Betriebssystem existiert, welche diese Befehle unterstützt. Ein Schritt hierzu würde CP/M 68K lauten, welches aber, wohl wegen seines hohen Preises, sicherlich noch keine allzu große Anhängerschaft gefunden hat.
Mit RL-BASIC auf Diskette erhalten Sie zwar ein kleines Disketten-Betriebssystem, welches aber momentan nur das Öffnen und Bearbeiten einer einzigen Datei zuließe.
Programme können aber sowohl auf Diskette als auch auf Kasette zuverlässig und schnell abgespeichert werden.

Nachfolgend erhalten Sie eine Übersicht der für den Betrieb von RL-BASIC notwendigen Voraussetzungen.

Triberg im Dezember 1985.

A l l g e m e i n e s

RL-Basic enthält einen Bildschimrorientierten Programmeditor, wie er auch bei Rechnern wie dem Commodore C64 eingesetzt wird.
Dies bringt gegenüber einfachen Zeileneditoren, wie sie bei Microsoft-Basic eingesetzt werden, einige Vorteile beim Editieren fehlerhafter Programmzeilen.
Die Funktionsweise sowie die Kommandos dieses Programmeditors werden in einer der nachfolgenden Kapitel noch behandelt.

Im Gegensatz zu den meisten anderen BASIC-Interpretern sind bei RL-BASIC nicht nur die vordersten zwei Zeichen einer Variablen signifikant, d.h. werden unterschieden, sondern die ersten fünf Zeichen. Hierdurch können Variablen treffender bezeichnet werden, als dies bei nur zwei signifikanten Stellen möglich wäre.

Bei der Eingabe von Befehlen oder Variablen ist es nicht notwendig auf Klein- oder Grossschreibung zu achten, da alle Befehle und Variablen in Grossbuchstaben konvertiert werden. So ist es möglich Befehle in Klein- bzw. Grossbuchstabe oder sogar gemischt einzugeben. Doch keine Sorge, Zeichenketten werden in ihrer Form belassen.

RL-BASIC unterstützt BASIC-Porgramme jeder Größe. Die Größe eines Programms und seiner Variablen wird nur durch die Größe des, für den Arbeitsbereich zur Verfügung stehenden, RAM-Bereichs beschränkt.

RL-BASIC als auch RL-BASIC-Programme sind in ihrer Lage vollständig unabhängig von ihrer aktuellen Position im Adressraum des NDR-68008-Computers. So ist es möglich alle Programme weiterzuverwenden, auch wenn das Grundprogramm und/ oder BASIC verschoben wurde !

Gegenüber Version 1.x ist in der nun vorliegenden Version 2.0 die Funktion ATN () = Arcustangens hinzugekommen.

Restriktion : In der aktuellen Version 2.0 werden noch keine Sequentielle- und Randomdateien unterstützt.
Das Abspeichern ganzer Speicherbereiche ist jedoch sowohl vom Grundprogramm als auch von RL-DOS möglich.

Zeichenketten sind in ihrer Länge auf 80 Zeichen beschränkt.

K o n f i g u r a t i o n

Als Rechnerkonfiguration wird hier die Ausbaustufe des NDR-Klein-Computer bezeichnet. Es sind hierzu zwei grundlegende Bemerkungen zu machen :

1. BASIC für NDR-Computer ohne Floppy Disk.
2. BASIC für NDR-Computer mit Floppy-Disk-Laufwerk

1. RL-BASIC für NDR-Computer ohne Floppy Disk

Die Mindesstattung für reibungslosen Betrieb :

i) CPU68K Karte.
ii) GDP64K Graphikkarte.
iii) KEY Karte für Tastatur.
iv) ROA Karte mit Grundprogramm ab Version 4.0.
v) mindestens 8K-Byte RAM auf ROA Karte.
vi) Weitere ROA Karte für die Aufnahme von RL-BASIC oder zusätzliches RAM
vii) CAS Karte zum Anschluß eines Kassettenrecorders für Laden und Speichern von Programmen.

2. RL-BASIC für NDR-Computer mit Floppy Disk Laufwerk

Die Mindestausstattung für einen reibungslosen Betrieb :

i) CPU68K Karte.
ii) GDP64K Graphikkarte.
iii) KEY Karte für Tastaturanschluß.
iv) ROA Karte mit Grundprogramm ab Vers. 4.0
v) Mindestens 32K oberhalb des Grundprogramms.
vi) Zusätzliche ROA oder andere RAM-Karten für RL-BASIC.
vii) FLO2 Disketten-CONTROLLER.
viii) Diskettenlaufwerk 2* 80 Spuren (1 MByte unformatiert) 3,5" oder 5,25" Laufwerke werden unterstützt.

K o n f i g u r a t i o n

Folgende Baugruppen werden von RL-Basic zusätzlich unterstützt :

1) Uhren-Karte mit Batteriegepufferter Echtzeituhr.
2) IOE-Karte mit CENTRONICS - Erweiterung, zum Anschluß eines Druckers.
3) SOUND-Karte zur Ausgabe von Geräuschen.
4) RAM - Karten jeglicher Größe.

Inbetriebnahme von RL-BASIC

1. RL-BASIC auf Eproms :

i) Rechner ausschalten und eine ROA 64 Karte mit mindestens drei freien Steckplätzen, die zudem aufeinanderfolgen müssen bereitlegen. (Bei 68000er System: 2 ROA Karten mit mindestens jeweils zwei aufeinanderfolgenden Steckplätzen und demselben Adressbereich bereitlegen !)

ii) EPROMS der Reihe nach (von links nach rechts) in die drei aufeinanderfolgende Steckplätze der ROA 64 Karte einstecken. Auf die Polarität (Nasen nach oben) ist zu achten, da ansonsten die Eproms zerstört würden. (Bei 68000er System : gerade und ungerade Adressen beachten !)

iii) ROA 64 Karte(n) auf den BUS stecken und dann den Rechner wieder einschalten.

iv) Das Grundprogramm sollte sich nun wie üblich melden. Wählen Sie das Menü BIBLIOTHEK aus. Folgende Meldung müßte auf Ihrem Bildschirm erscheinen:

B I B L I O T H E K

```
Name    : B A S I C 6 8 K
Start   : x x x x x x  <- Startadresse
Laenge  : 0 0 5 F F F
```

S t a r t e n J = J A

c r = w e i t e r , M = M e n u e

2. RL-BASIC auf Diskette :

i) Rechner einschalten und Menue Floppy Start anwählen.

ii) Diskette mit RL-DOS einlegen. (Wird bei Bestellung von RL-BASIC auf derselben Diskette mitgeliefert)

iii) Es sollte ein Kurzinfo erscheinen wie : Diskettenbetriebsystem wird geladen.

iv) Daraufhin meldet sich RL-DOS mit folgender Bildschirmausgabe :

```
-------------------------
Disketten Betriebssystem
 fuer NDR-Klein-Computer
--   Vers. 1.2  4/86   --
-- (c) R. Lobreyer     --
-------------------------

A>_
```

v) Es ist möglich das BASIC irgendwo in den Adressraum des NDR-Klein-Computer zu laden. Wählen Sie also einen mindestens 24K-Byte grossen Speicherbereich aus, wohin Sie das BASIC laden wollen.
Sei die Anfangsadresse des Bereichs mit ADRESSE bezeichnet.
Geben Sie nun folgenden Befehl ein :
LOAD "BASIC",ADRESSE
Für ADRESSE setzen Sie die oben bezeichnete Anfangsadresse ein.
Ein Beispiel :
Ab Adresse Sedezimal $10000 ist genügend RAM vorhanden, um Basic zu laden, so ist der Befehl LOAD "BASIC",$10000 einzugeben und Return zu drücken.
Das Laufwerk setzt sich nun in Bewegung und lädt RL-Basic in den Hauptspeicher ihres NDR-Klein-Computer.

vi) Verlassen Sie RL-DOS mit dem Befehl EXIT.

vii) Wählen Sie das Menu BIBLIOTHEK an. (Siehe BASIC auf EPROM iv))

Anmerkung : RL-DOS funktioniert mit allen CPUs der 680xx Serie ohne entsprechende Anpassung !

Start von RL-BASIC

Haben Sie oben aufgeführte Schritte durchgeführt, so rufen Sie nun RL-BASIC aus dem BIBLIOTHEKS-Menu mit 'J' auf.

RL-BASIC ist, wie vom Grundprogramm her bekannt, Menügesteuert. Mit folgendem MENÜ können Basic-Parameter geändert werden und der Basic-Interpreter gestartet werden :

```
R L  -  B A S I C 6 8 K   (c)  1 9 8 5


        1 = W a r m s t a r t

        2 = K a l t s t a r t

        3 = S e t z e  A r b e i t s b e r e i c h

        E = E n d e


        --
```

Hier nun eine Erläuterung der einzelnen Menüpunkte :

ad 1: WARMSTART soll nur dann angewählt werden, falls der BASIC-Interpreter bereits mindestens einmal mit Kaltstart aufgerufen worden ist. Dieser KALTSTART ist notwendig um Systemparameter richtig zu setzen.
Der WARMSTART dient nun dazu ein einmal abgebrochenes Programm, z.B. wenn zwischendurch ein EPROM programmiert werden soll, wieder aufnehmen zu können.
Ein Warmstart lässt das Programm und alle Variablen intakt.

ad 2: KALTSTART muß aufgrufen werden, wenn RL-BASIC das erste mal nach dem Rechnereinschalten aufgerufen wird.
Es werden wichtige Systemparameter eingestellt, ohne die RL-BASIC nicht einwandfrei funktionieren kann.

Ein KALTSTART löscht ein Programm und Variablen im Speicher, falls bereits ein Basic-Programm eingegeben wurde.
Es stehen nun der maximale Arbeitsbereich - 1KByte für die Programmierung und Variablen zur Verfügung.

ad 3: SETZE ARBEITSBEREICH dient zum Festlegen eines Speicherbereichs, der ausschließlich von BASIC-Programmen und deren Variablen benutzt werden kann.
Eine Mindestgrösse von 2KByte sollte gewährleistet sein, um mit RL-BASIC sinnvoll arbeiten zu können.
Wird 'Setze Arbeitsbereich' aufgerufen, wird die Eingabe eines durch Anfangs- und Endadresse festgelegten Speicherbereichs gefordert.
Die Eingabe einer 0 läßt den bereits festgelegten Speicherbereich unverändert.
Falsche Eingaben führen zu einer neuen Eingabeaufforderung.
Der aktuelle Arbeitsbereich wird am unteren Bildschirmrand angezeigt.

!!! Achtung !!!
'Arbeitsbereich Setzen' sollte noch vor einem Kaltstart aufgerufen werden, da ansonsten der erste durchgehende Speicherbereich im Adressraum des 680xx ausgewählt wird !
Auf Konflikte mit dem Grundprogrammsbereich und mit dem Keller-Zeiger A7 ist beim Setzen des Arbeitsbereichs zu achten.

ad E. Durch Eingabe von 'E' wird das BASIC-Menü verlassen.
Es kann durch Auswahl des Menüpunktes BIBLIOTHEK aber jederzeit wieder gestartet werden.

Wird der BASIC-Interpreter durch den Befehl SYSTEM verlassen, so wird nicht in dieses Menu zurückgekehrt, sondern mit M=Menue gelangt man in das Hauptmenu des Grundprogramms zurück !

Starten von RL-BASIC

Wählen Sie den Menüpunkt 3 = Arbeitsbereich setzen aus, um einen geeigneten Speicherbereich für die Arbeit mit dem BASIC-Interpreter auszuwählen. Liegt Ihr Grundprogramm ab Adresse $000000, so sollte der Bereich ab ca. $10000 bis RAMende - $400 angegeben werden. So sind Ihre BASIC-Programme vor dem Überschreiben mit Systemvariablen und Ihre Daten (BASIC-Variablen...) vor dem Überschreiben durch den Kellerzeiger A7 sicher.
Liegt Ihr Grundprogramm ab Adresse $e00000, so kann ein Bereich von ca. $1000 bis RAMende - $400 angegeben werden.

Starten Sie nun RL-BASIC mit dem Menüpunkt 2 = KALTSTART.
Folgendes Bildschirmbild sollte sich jetzt einstellen :

```
**************************
*      RL-Basic 2.0      *
* 68008/68000 Version für *
*   NDR-Klein Computer    *
*  (c) R. Lobreyer 6/1985 *
**************************
xxxxxx Bytes free


>_
```

RL-Basic ist nun bereit Eingaben entgegenzunehmen und auszuführen.
Befehle dürfen in Klein- bzw Grossbuchstaben oder auch gemischt eingegeben werden.
Im Anhang finden Sie einige BASIC-Programme, die Sie zur Übung mit dem System eingeben können.
Die Beschreibung des Programmeditors erfolgt im nächsten Kapitel.

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

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.
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 CTRL-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 !!

Der Programmeditor

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

CTRL-Z	Löschen des Bildschirms und Cursor links oben positionieren.
Backspace	Löschen des zuletzt eingegebenen Zeichens bzw. das Zeichen vor dem Cursor wird gelöscht.
CTRL-T	Löscht den Rest der Zeile.
ESC-H	Cursor nach links oben, ohne Bildschinrm löschen.
ESC-J	Wie CTRL-Z Bildschirm löschen.
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.
CTRL-C	Zeile ignorieren, Rest der Zeile wird gelöscht und der Cursor eine Zeile tiefer positioniert.
CTRL-B	Warmstart !

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

Start eines BASIC-Programms

Nachdem das Programm in den Hauptspeicher eingegeben oder von Kasette bzw. von Diskette geladen wurde, steht einem Programmlauf nichts mehr im Wege.
Sie sollten sicherheitshalber, bevor das Programm gestartet wird und Sie das Programm noch nicht abgespeichert haben, dieses abspeichern.
In der EPROM-Version geschieht dies folgendermaßen :

i) Einlegen einer Kassette in den Kasettenrecorder.
ii) START und RECORD Taste drücken.
iii) Eingabe des Befehls SAVE K,"Programm" <RETURN>

In der Disketten-Version so :

i) Diskette ins Laufwerk einlegen.
ii) Eingabe des Befehls SAVE "Programm" <RETURN>

Nachdem Sie das Programm abgespeichert haben, kann es endlich ausgeführt werden.

Die Programmausführung wird mit RUN gestartet. Es wird hierbei der erste Befehl in der ersten Zeile des Programmes interpretiert und ausgeführt.
Eine Programmunterbrechung ist jederzeit durch Drücken der Tasten CONTROL und C gleichzeitig möglich.

Soweit die Einführung !

Es folgt nun eine Auflistung aller in RL-BASIC enthaltenen Befehle / Anweisungen / Funktionen.

In den Anhängen finden Sie die von BASIC reservierten Schlüsselwörter, die Sie für Variablennamen nicht benutzen dürfen.
Des weiteren einige kleine Programmbeispiele, BASIC-Fehlermeldungen sowie die Adressbelegung durch RL-BASIC.

A U T O

Syntax : AUTO [<Zeilennummer>][,<Schrittweite>]

Effekt : Dieser Befehl wird im Direktmodus eingegeben, um die automatische Zeilennumerierung einzuschalten.

Bemerkung : Die Zeilengenerierung startet bei Zeile <Zeilennummer> und wird in Schritten von <Schrittweite> erhöht.
Für beide Werte gilt, daß sie mit 10 voreingestellt sind.
Wird jediglich <Schrittweite> spezifiziert, so beginnt die Numerierung bei 0 und wird in Schritten von <Schrittweite> erhöht.
Sinn dieses Befehles ist es, die Eingabe fortlaufender Programmzeilennummern einzusparen.

AUTO wird durch Drücken von <CTRL-C> abgeschaltet, so daß die Zeilennummern wieder von Hand eingegeben werden müssen.

Beispiel 1: AUTO
Erzeugt Zeilennummer ab 10 mit einer Schrittweite von 10.
(10,20,30,40,50...)

Beispiel 2: AUTO 100,50
Erzeugt Zeilennummern ab 100 in Schritten von 50.
(100,150,200,250...)

Beispiel 3: AUTO ,20
Erzeugt Zeilennummern ab 0 in Schritten von 20.
(0,20,40,60...)

A R C

Syntax : ARC (<x-Pos>,<Y-Pos>),<Radius>,<Startwinkel>,<Endwinkel>

Effekt : ARC zeichnet einen Kreisbogen um <X-Pos>,<Y-Pos> mit Radius <Radius> von Winkel <Startwinkel> bis <Endwinkel>.

Bemerkung : Als Argumente (<Radius>,<Startwinkel>,<X-pos>...) sind nur Ganzzahlige Werte zugelassen. D.h. wenn mit Gleitkommazahlen gerechnet wird, müssen diese zuerst in einen ganzzahligen Wert überführt werden. (Siehe hierzu Funktion INT) Als <Startwinkel> und <Endwinkel> gelten Winkel von 0..359, die in Grad angegeben werden müssen.

RADIUS, X-Pos, Y-Pos können in einem Bereich von 0..2047 liegen. Jediglich Für Werte innerhalb des Bereichs von 0..511, liegen diese im sichtbaren Bildfenster.

Beispiel: ARC (100,100),50,0,30
Zeichnet einen Kreisausschnitt von 30 grad, ab der Stelle 150,100. (100,100 bezeichnet den Mittelpunkt des virtuellen Kreises, 50 den Radius)

C A L L

Syntax : CALL <Startadresse>

Effekt : Startet ein Maschinensprache-Unterprogramm.

Bemerkung : Der CALL-Befehl ist eine Methode, um den Programmablauf einem Maschinenspracheprogramm zu übertragen. Die Ausführung des Maschinenprogramms beginnt ab Adresse <Startadresse>. Der Benutzer hat dabei sicherzustellen, daß sich ab dieser Adresse ein ausführbares Maschinenprogramm befindet, da ansonsten BASIC-Daten zerstört werden können.
Das Maschinenspracheprogramm muß mit dem Befehl RTS ($4e75) abgeschlossen werden.

Eine weitere Möglichkeit Maschinenprogramme aufzurufen ist der Befehl PRINT chr$(1) ...

Beispiel : CALL $9000
ruft ein Maschinenprogramm auf, dessen Startadresse Sedezimal 9000 ist.

C H A I N

Syntax : CHAIN <Dateiname>

Effekt : Dieser Befehl startet das BASIC-Programm <Dateiname>, ohne die vorhandenen Variablen zu löschen.

Bemerkung : CHAIN ist nur in der Diskettenversion lauffähig !
Es ist bei <Dateiname> zu beachten, daß wirklich nur eine Datei angegeben wird, dessen Typ im Direktory mit BAS gekennzeichnet ist.
Das aufgerufene Programm ersetzt das laufende sich im Arbeitsspeicher befindliche Programm. Die Ausführung des Programmes <Dateiname> beginnt mit dem ersten Befehl in der ersten Programmzeile.

C I R C L E

Syntax : CIRCLE [STEP] (<X-Pos>,<Y-Pos>),<Radius>[,F]

Effekt : Circle zeichnet einen Kreis mit Mittelpunkt <X-Pos>, <Y-Pos> und Radius <Radius>.

Bemerkung : Wird STEP spezifiziert, dann beziehen sich die Mittelpunktskoordinaten relativ auf den zuletzt gezeichneten Punkt. Bei Angabe der Option F wird der Kreis mit der Zeichenfarbe ausgefüllt. Nach Ausführung des Kreisbefehls, wird die Zeichenposition auf <X-Pos>, <Y-Pos> aktualisiert.

Beispiel : CIRCLE (100,100), 50
Zeichnet einen Kreis an Stelle 100,100 als Mittelpunkt mit dem Radius 50.

Beispiel 2: CIRCLE STEP (10,0), 50, F
Zeichnet einen ausgefüllten Kreis 10 Koordinatenpunkte in x-Richtung von der letzten Zeichenposition gerechnet.

C L E A R

Syntax : CLEAR

Effekt : Löscht den von Variablen belegten Arbeitsspeicher.

Bemerkung : Das CLEAR-Kommando löscht alle bis dahin initialisierten Variablen und Funktionsdefinitionen, ohne daß dadurch das Programm angehalten oder gelöscht wird.

C L P G

Syntax : CLPG

Effekt : Löscht die aktuelle Schreibseite.

Bemerkung : Dieser Befehl wird zum Löschen einer der vier Graphikseiten benutzt. Zum Löschen von Text sollte dieser Befehl nicht benutzt werden, da sich der Text meist auf zwei Graphikseiten gleichzeitig befindet.

C L S

Syntax : CLS

Effekt : Löscht die Schreibseiten 0 und 1 zur Textausgabe und positioniert den Cursor links oben.

Bemerkung : Dieser Befehl dient zum Löschen des Bildschirms, wenn Text ausgegeben werden soll. Es wird dabei automatisch auf die Graphikseiten 0 und 1 umgeschaltet. Der Cursor wird durch abwechselndes Hin- und Herschalten zwischen Seite 0 und Seite 1 des Graphikbildschirms dargestellt.

Beispiel :
```
10 CLS
20 PRINT "Text"
```

Löscht den Bildschirm und gibt 'Text' links oben aus.

C O L O R

Syntax : COLOR = <Farbe>

Effekt : Color setzt die Zeichenfarbe fest.

Bemerkung : Da die GDP64K Baugruppe jediglich für Schwarz-Weiss Darstellung ausgelegt ist, kann die Zeichenfarbe nur zwischen Schwarz (Farbe = 0) und Weiss (Farbe = 1) gewählt werden.
Die Farbe Weiss (<FARBE>=1) ist zu Beginn eines Programmes voreingestellt.

Beispiel :
```
10 CLPG
20 COLOR = 1
30 CIRCLE (299,100),70,F
```

Zeichnet einen weiss ausgefüllten Kreis.

C L S

Syntax : CLS

Effekt : Löscht die Schreibseiten 0 und 1 zur Textausgabe und positioniert den Cursor links oben.

Bemerkung : Dieser Befehl dient zum Löschen des Bildschirms, wenn Text ausgegeben werden soll. Es wird dabei automatisch auf die Graphikseiten 0 und 1 umgeschaltet. Der Cursor wird durch abwechselndes Hin- und Herschalten zwischen Seite 0 und Seite 1 des Graphikbildschirms dargestellt.

Beispiel :
```
10 CLS
20 PRINT "Text"
```

Löscht den Bildschirm und gibt 'Text' links oben aus.

C O L O R

Syntax : COLOR = <Farbe>

Effekt : Color setzt die Zeichenfarbe fest.

Bemerkung : Da die GDP64K Baugruppe jediglich für Schwarz-Weiss Darstellung ausgelegt ist, kann die Zeichenfarbe nur zwischen Schwarz (Farbe = 0) und Weiss (Farbe = 1) gewählt werden.
Die Farbe Weiss (<FARBE>=1) ist zu Beginn eines Programmes voreingestellt.

Beispiel :
```
10 CLPG
20 COLOR = 1
30 CIRCLE (299,100),70,F
```

Zeichnet einen weiss ausgefüllten Kreis.

C O N N E C T

Syntax : CONNECT [STEP] (<X1>,<Y1>)-(<X2>,<Y2>)-(<X3>,<Y3>)...

Effekt : Connect verbindet mehrere Punkte miteinander.

Bemerkung : Von Koordinate <X1>,<Y1> beginnend werden die Punkte durch Linien miteinander verbunden, sodaß dadurch ein Polygonzug entsteht. Die Anzahl der Koordinatenpaare muß mindestens zwei betragen und ist nur durch die maximale Zeilenlänge begrenzt. Durch Angabe von [STEP] kann relativ zur letzten Zeichenposition gezeichnet werden. Dieser Wert wird durch das letzte Zeichnen eines Kreises, Setzen eines Punktes oder durch das Zeichnen einer Verbindungslinie ständig aktualisiert.

Als Argumente für die einzelnen Koordinaten sind, wie bei allen Graphikfunktionen, nur Integerwerte zugelassen.

Beispiel :
```
10 CLPG
20 CONNECT (100,100)-(200,100)-(150,200)-(100,100)
```

Zeichnet ein gleichschenkliges Dreieck auf den Bildschirm.

C O N T

Syntax : CONT

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

Bemerkung : CONT kann nur im Direktmodus eingegeben werden, um den Programmlauf mit dem als nächstes auszuführenden Befehl wieder fortzusetzen.
Kann das Programm nicht ordnungsgemäß fortgesetzt werden, so erfolgt die Fehlermeldung 'Cannot continue error'.
Eine mit <CTRL-C> unterbrochene Schleife im Direktmodus wird durch CONT ordnungsgemäß fortgeführt.

D A T A

Syntax : DATA <Konstante>[,<Konstante>]...

Effekt : Definiert eine Liste von Konstanten, die eine READ-Anweisung Variablen zuweisen kann.

Bemerkung : Als Konstanten sind sowohl Integer, Real als auch Stringkonstanten zugelassen.
Eine DATA Anweisung sollte zu Beginn einer Zeile stehen, kann aber irgendwo im Programm eingefügt werden.

Wenn mehr als eine DATA Anweisung im Programm vorhanden sein sollte, so ist die Reihenfolge in der sie von einem READ gelesen werden, die der Programmzeilen. Man kann sich die in DATA-Zeilen spezifizierten Konstanten als eine fortlaufende Liste vorstellen.
Die Konstantentypen müssen natürlich mit der bei einem READ notwendigen Variablen übereinstimmen.

Wurde die Liste der Konstanten einer DATA-Zeile gelesen, so können auf diese bis zum nächsten RESTORE nicht mehr zugegriffen werden.

Beispiel : 10 DATA 10,15,1.2332,"Text"

Definiert die Konstanten 10,15 als Integer/REAL, 1.2332 als REAL-konstante und "Text" als Zeichenkettenkonstante.

D A T E $

Syntax : DATE$ = <Datumstring>

Effekt : Aktualisiert das Datum auf der Uhrenkarte.

Bemerkung : <Datumstring> gehorcht folgender Syntax: "DD.MM.JJ".
Das Datum wird nur dann gesetzt, wenn sich die Uhrenkarte auch auf dem Systembus befindet.

Beispiel : DATE$ ="12.12.85"
setzt das Datum der Uhrenkarte auf den 12.12.85.

D E F F N

Syntax : DEF FN <Bezeichner> (<Bezeichner_V>) = <Arithm. Ausdruck>

Effekt : Definiert anwenderspezifische Funktionen.

Bemerkung : Eine benutzerdefinierte Funktion ist ein numerischer Ausdruck welcher den Befehlssatz von RL-Basic erweitert. Wird eine solche Funktion aufgerufen, so wird die Variable <Bezeichner_V> durch das aufrufende Argument ersetzt und die Funktion mit diesem Wert berechnet. Die Variable <Bezeichner_V> kann im Programm unabhängig von dieser Funktion verwendet werden.

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

Definiert eine Benutzerfunktion namens F mit dem Funktions-Parameter X.

D E L E T E

Syntax : DELETE <Zeilennummer>[-<Zeilennummer_1>]

Effekt : Löscht eine oder mehrere Programmzeilen aus dem Arbeitsspeicher.

Bemerkung : Bei Angabe einer nicht vorhandenen Programmzeile, werden alle Zeilen bis zur nächst höheren Zeilennummer gelöscht. Das Löschen beginnt bei Zeile <Zeilennummer> und endet mit Zeile <Zeilennummer_1> einschließlich.
Das Löschen nur einer Programmzeile ist auch durch Eingabe der Zeilennummer und anschliessendem <RETURN> möglich, es muß also nicht immer DELETE .. eingegeben werden.

Beispiel : DELETE 10
löscht Programmzeile 10, insofern diese überhaupt vorhanden sein sollte.

Beispiel 2: DELETE 11-25
Löscht die Programmzeilen 11 bis einschließlich 25.

D I M

Syntax : DIM <Variablenbezeichner> (<num. Ausdruck>[,<num. Ausdruck>]..)

Effekt : Der DIM - Befehl wird benutzt, um den maximalen Bereich für Feldelemente zu definieren und Speicherplatz für die Feldvariablen zu schaffen.

Bemerkung: Als <Variablenbezeichner> sind sowohl Integer-, Real- als auch Stringbezeichner zugelassen. Das Feld A$ (3) unterscheidet sich von der Variablen A$.
Der numerische Integer-Ausdruck definiert die maximale Zahl der Komponenten, wobei der Index von 0 bis Maxindex reicht. Z. B. definiert DIM A (20,50) ein zweidimensionales Feld, dessen individuellen Variablen mit A(N1,N2) bezeichnet werden. Wobei N1 maximal 20 und N2 maximal 50 sein darf. Das Feld besitzt demnach 21*51 Komponenten. Wird versucht ein Element zu adressieren, welches ausserhalb des angegebenen Bereichs liegt, erscheint die Fehlermeldung 'Bad subscript error'.
Wurde ein Feld nicht dimensioniert, so ist der größte für Indizes verfügbare Bereich 10.
Nachdem einmal ein Feld dimensioniert wurde, ist es nicht mehr möglich es neu zu dimensionieren; nur noch nach einem CLEAR - Befehl, der alle Variablen löscht !

Beispiel 1: 10 DIM A$ (20,15)
Definiert ein zweidimensionales Feld und spezifiert dabei 20 und 15 als die jeweils maximalen Indizes.

Beispiel 2: 20 DIM A% (100)
Definiert ein eindimensionales Integerfeld mit 101 Elementen.

D R A W T O

Syntax : DRAW TO <Integer_Ausdruck>,<Integer_Ausdruck>

Effekt : DRAW TO zeichnet eine Linie von der letzten Zeichenposition zur Position <Int_Ausdr.>,<Int_Ausdr.>.

Bemerkung : Die letzte Zeichenposition wird durch verschiedene Graphikbefehle aktualisiert. Z.B. CIRCLE, PSET, LINE .... <Integer_Ausdruck> sollte im Bereich von 0..511 liegen, um sichtbar zu sein.
Die Auflösung der GDP64K Karte wird virtuell auf 511x511 Bildpunkte erhöht, um ein quadratisches Bildfenster zu erreichen und um mit der Turtlegraphik kompatibel zu sein. Möglich ist ein Bereich von 0..4095x0..4095, der allerdings nicht darstellbar ist.

Die Zeichenseite und Farbe wird durch andere Befehle gesetzt.

E N D

Syntax : END

Effekt : Beendet ein BASIC-Programm.

Bemerkung : Stößt der Interpreter auf END, so wird sofort in den Textmodus geschaltet und das System ist wieder bereit Eingaben entgegenzunehmen.
Existieren nach END noch weitere Anweisungen, so werden diese nicht mehr ausgeführt.

E R A S E

Syntax : ERASE <Feldvariablenbezeichner>

Effekt : Setzt ein ganzes Feld auf den Wert 0, falls es sich um ein Integer oder Real-Feld handelt, und auf "", falls es sich um ein String-Feld handelt.

Bemerkung : Dieser Befehl wirkt wie eine Neuinitialisierung des Feldes, ohne jedoch die Dimension des Feldes zu beteffen. Es ist so einfach möglich riesige Felder schnell auf "null" zu setzen.

Beispiel :

```
10 A$ (0) = "Hallo"
20 A$ (1) = "NDR-Computer"
30 A$ (3) = "Text"
40 FOR I = 0 TO 3 : PRINT A$ (I)
50 ERASE A$
60 NEXT
```

Gibt den Text 'Hallo' auf den Bildschirm aus.
Die Feldelemente A$(1) und A$(3) werden nicht mehr gedruckt, da sie bereits wieder gelöscht wurden.

E R R O R

Syntax : ERROR = <Integer-Ausdruck>

Effekt : ERROR macht es möglich das Auftreten eines RL-Basic Laufzeitfehler zu simulieren.

Bemerkung : Wird dieser Befehl abgearbeitet, und ist keine Fehlerbehandlungsroutine vorhanden, so wird die Fehlermeldung, dessen Fehlercode dem Wert des Integer-Ausdruckes entspricht, ausgegeben. Eine Liste aller Fehlermeldungen befindet sich am Ende dieses Handbuches.

Fehlerbehandlungsroutinen können im Zusammenhang mit dem ERROR-Befehl benutzt werden, um Benutzereingaben Fehlercodes zuzuordnen. (Siehe Beispiel)

Beispiel :

```
10 INPUT A,B
20 X = A * B
30 IF X > 50 THEN ERROR 3
40 GOTO 10

RUN
? 4,5
? 7,8
Range error in 30
```

F I L E S

Syntax : FILES

Effekt : Gibt den Direktoryinhalt auf den Bildschirm aus.

Bemerkung : Dieser Befehl funktioniert nur dann, wenn RL-Basic
auf Diskette geliefert wurde, oder RL-DOS vor
Start des Basic-Interpreters geladen wurde.
Alle nachfolgenden Befehle, die in derselben Programmzeile
stehen, werden ignoriert !

Beispiel : FILES
Diskette No. 1 12.12.85

Name	Art	Status	Ladeadr.	Sektoren
BASIC	BIN	R/W	$008000	24K
DEMO	BAS	R/W	$004400	1K

760 Sektoren frei
40 Sektoren belegt

F O R . . N E X T

Syntax : FOR <Laufvar.> = <Ausdruck 1> TO <Ausdruck 2> [STEP <Ausdr. 3>]
.
.
NEXT

Effekt : Die FOR..NEXT Anweisung macht es möglich die Liste von Anweisungen, die zwischen FOR und NEXT steht, eine spezifizierte Anzahl mal zu wiederholen.

Bemerkung : Mit Hilfe dieses Befehls ist es möglich eine kontrollierte Programmschleife zu ermöglichen.
Die Anzahl der Schleifendurchläufe wird durch die Werte der numerischen Ausdrücke <Ausdruck 1>,<Ausdruck 2> und <Ausdruck 3> bestimmt. <Ausdruck 1> bestimmt den Startwert, <Ausdruck 2> den Endwert und <Ausdruck 3> die Schrittweite, in der vom Startwert zum Endwert gezählt wird.
Die Variable <Laufvariable> stellt dabei die aktuelle Zählposition dar.

Eine Schrittweite von 1 wird angenommen, falls die Option STEP nicht angegeben wurde.
Ist der Endwert kleiner als der Startwert, wurde ausserdem keine negative Schrittweite gewählt, so wird die Schleife "unendlich" lange ausgeführt.
FOR..NEXT Schleifen können geschachtelt werden, es muß dann für jede Schleife eine unterschiedliche Laufvariable verwendet werden. Die NEXT Anweisung für die innere Schleife muß vor dem Next der sie umgebenden Schleife stehen.
Wird ein NEXT vor dem entsprechenden FOR entdeckt, so erscheint die Fehlermeldung "NEXT without FOR error".

Achtung ! FOR .. NEXT Schleifen sind in einem Unterprogramm immer abzuschliessen, bevor ein RETURN erfolgen kann.

Beispiel 1:
```
10  PRINT "X","X^2","X^3"
20  FOR X = 0 TO 10
30  PRINT X,X^2,X^3
40  NEXT
```

```
RUN
X             X^2           X^3
0             0             0
1             1             1
2             4             8
3             9             27
4             16            64
5             25            125
6             36            216
7             49            343
8             64            512
9             81            729
10            100           1000
```

Besipiel 2:
```
10  FOR I = 0 TO 200 STEP 20
20  CONNECT (255-I,255-I)-(255+I,255-I)-(255+I,255+I)-(255-I,
    255+I)-(255-I,255-I)
30  NEXT
```

Zeichnet größerwerdende Quadrate auf den Bildschirm.

G O S U B . . R E T U R N

Syntax : GOSUB <Zeilennummer>
.
.
RETURN

Effekt : Gibt die Programmkontrolle an ein BASIC-Unterprogramm ab.

Bemerkung : GOSUB gibt die Programmausführung an die in <Zeilennummer> angegebene Programmzeile ab; das Programm wird nach Beendigung des Unterprogramms an der Stelle fortgesetzt, die dem aufrufenden GOSUB folgt. Also der GOSUB nachfolgenden Anweisung oder Programmzeile.
Es ist möglich mehrere Unterprogrammebenen zu schachteln, Die maximale Schachtelungstiefe beträgt ca. 100.
Befindet sich im Programmtext ein RETURN, bevor eine Subroutine aufgerufen wurde, erscheint die Fehlermeldung 'RETURN without GOSUB error'.
Wurde versucht mehr Unterprogramme aufzurufen, als dies die Kellergröße zulässt, so erscheint die Fehlermeldung 'Stack overflow error'. (Der Keller ist mit 1KByte für die allermeisten Anwendungen ausreichend dimensioniert).

Beispiel :
```
10 FOR I% = 0 TO 100 STEP 5
20 GOSUB 100
30 NEXT
40 END
100 CONNECT (255-I%,100)-(255+I%,100)-(255,255)-(255-I%,100)
110 RETURN
```

Gibt größerwerdende Dreiecke auf dem Bildschirm aus.

G O T O

Syntax : GOTO <Programmzeilennummer>

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.

Beispiel :

```
10 READ A,B
20 IF A=0 AND B=0 THEN 80
30 PRINT "A=";A,"B=";B
40 S = A * B
50 PRINT "PRODUKT LAUTET :";S
60 GOTO 10
70 DATA 12,5,8,3,9,0,0,0
80 END

RUN
A=12            B=5
PRODUKT LAUTET : 60
A=8             B=3
PRODUKT LAUTET : 24
A=9             B=0
PRODUKT LAUTET : 0
```

I F ... T H E N [... E L S E]

Syntax : IF <Logischer Ausdruck> THEN <Anweisung_1>|<Zeilennummer>
[ELSE <Anweisung_2>]

Effekt : Stellt Bedingungen auf, die den Programmablauf festlegen.

Bemerkung : Bei Verwendung einer IF Anweisung wird der logische Ausdruck ausgewertet. Dieser kann die Werte WAHR oder FALSCH annehmen. War der Ausdruck WAHR, so wird die dem THEN folgende Anweisung ausgeführt. Eine Zeilennummer hinter THEN ist gleichbedeutend zu einem GOTO <Zeilennummer>; es ist auch möglich THEN <Zeilennummer> durch ein GOTO <Zeilennummer> zu ersetzen.

<Logischer Ausdruck> = WAHR :
Die dem THEN folgende <Anweisung_1> wird ausgeführt und die nächste Zeile bearbeitet.

<Logischer Ausdruck> = FALSCH :
- ELSE <Anweisung_2> vorhanden -> <Anweisung_2> wird ausgeführt.
- ELSE <Anweisung_2> fehlt -> nächste Zeile.

<Anweisung_1>,<Anweisung_2> stehen für jeweils mindestens eine Anweisung.

Beispiel :
```
10 INPUT "Mögen Sie dieses BASIC ?";A$
20 IF A$ = "JA" OR A$ ="ja" THEN PRINT "TOLL": END
30 IF A$ = "NEIN" OR A$ ="nein" THEN PRINT "MIST": ELSE
   PRINT "Falsche Eingabe, nochmal ":GOTO 10
```

I N P U T

Syntax : INPUT ["<Text>";] <Variable>[,<Variable>] ...

Effekt : Ermöglicht Dateneingabe während des Programmlaufs und ordnet diese Daten Programmvariablen zu.

Bemerkung : Die Programmausführung hält an und wartet auf eine Eingabe. Dies wird durch einen stehenden CURSOR angezeigt. Wird kein <Text> angegeben, erscheint im Programmlauf ein Fragezeichen.

Die Eingabe wird der (den) angegebenen Variablen zugeordnet. Bei falscher Eingabe wird das Programm mit einer Fehlermeldung beendet, falls keine Fehlerbehandlungsroutine (Siehe ON ERROR GOTO) vorhanden ist.
Werden mehere Eingaben gefordert, so können diese durch Kommata oder <RETURN> getrennt werden. Wird die Eingabe durch <RETURN> getrennt, erscheint ein Fragezeichen in der nächsten Zeile.
Es ist unzulässig mehrere String-Variablen mit einer INPUT-anweisung einzulesen, da das Komma hierbei nicht als Trennzeichen anerkannt wird.
Die Eingabe kann durch drücken der Backspace-Taste innerhalb der Eingabezeile korrigiert werden. (Backspace besitzt den ASCII-Code 08, ist also auch durch CTRL-H ansprechbar)

Beispiel 1:
```
10 INPUT " WIE HEISSEN SIE ?";A$
```
Liest eine Zeichenkette in die Variable A$ ein.

Beispiel 2:
```
10 FOR I = 1 TO 3
20 INPUT M (I)
30 NEXT
40 FOR I=1 TO 3 : PRINT M(I) : NEXT
```

Liest drei Feldelemente des Typs Real ein und gibt diese dann auf den Bildschirm aus.

K I L L

Syntax : KILL <Dateiname>

Effekt : Löscht eine Datei aus dem Inhaltsverzeichnis der Diskette.

Bemerkung : KILL ist nur in der Diskettenversion verfügbar.
Das KILL Kommando kann dazu benutzt werden, um jeden Typ von Diskettendateien zu löschen. Also sehr vorsichtig damit umgehen, um ein versehentliches Löschen wichtiger Daten zu vermeiden.

Beispiel : KILL "A:TEST"
Löscht die Datei TEST auf der sich im Laufwerk A befindlichen Diskette.

L E T

Syntax : [LET] <Variable> = <Ausdruck>

Effekt : Weist einer Variablen einen Wert zu.

Bemerkung : LET kann optional eingesetzt werden. Der Typ der Variablen und der des Ausdrucks <Ausdruck> müssen übereinstimmen.

Beispiel 1:
```
10 LET PI = 3.141593
20 PRINT COS (PI)
```

gibt den Wert -1 aus.

Beispiel 2:
```
10 PI = 3.1415923
15 X = 2 * PI
20 PRINT COS (X)
```

gibt den Wert 1 aus und weist den Variablen PI den Wert 3.141593 der Variablen X den Wert 6.2831846.

L I N E

Syntax : LINE [STEP] (<x1>,<y1>)-[STEP](<X2>,<Y2>)

Effekt : Zeichnet eine Gerade zwischen den angegebenen Koordinatenpunkten.

Bemerkung : Relative Koordinaten können für jede der beiden Koordinatenpaare spezifiziert werden, es ist dann die Otion STEP vor jedes Koordinatenpaar zu setzen.
Die Gerade wird in der aktuellen Zeichenfarbe gezeichnet.
Nach Beendigung des Befehls wird die Zeichenposition auf den Endpunkt der Gerade gesetzt.
LINE zeichnet immer auf der durch den Befehl PAGE festgelegten Schreibseite !

Beispiel :

```
10 PAGE 3,3: CLPG
20 RANDOMIZE 511
30 FOR I = 1 TO 200
40 LINE (RND,RND)-(RND,RND)
50 NEXT
60 FOR I = 1 TO 10000: NEXT
70 GOTO 10
```

Zeichnet zufällige Geraden auf die Graphseite 3.

L I S T L L I S T

Syntax : LIST ¦ LLIST [<Zeilennummer>][-<Zeilennummer>]

Effekt : Bringt einen Programmausdruck auf den Bildschirm oder Drucker.

Bemerkung : LIST listet das gesamte Programm auf dem Bildschirm aus. Es ist dabei möglich die Ausgabe durch <CTRL-S> anzuhalten durch <CTRL-Q> wieder zu starten oder durch <CTRL-C> abzubrechen.
LLIST hat die gleichen Parameter wie LIST, hat aber den Effekt, daß der Programmausdruck über die Centronicsschnittstelle ausgegeben wird.
Wird eine Zeilennummer angegeben so wird lediglich diese ausgelistet.
Einen Bereich kann durch Angabe von LIST <Zeile_1>-<Zeile_2> ausgegeben werden.

Beispiel : LIST
listet das gesamte sich im Arbeitsspeicher befindliche Programm.

Beispiel 2: LIST 10
Gibt den Inhalt der Zeile 10 auf den Bildschirm aus.

Beispiel 3: LLIST 10-50
Listet den Inhalt der Zeilen 10 bis 50 einschließlich auf den Drucker.

Beispiel 4: LIST 20-
druckt eine Programmliste von Zeile 20 bis zum Programmende.

L O A D

Syntax : LOAD [K,] <Dateiname>

Effekt : Lädt eine Programmdatei in den Arbeitsspeicher.

Bemerkung : Die Option 'K,' ist dem Laden von Programmen von Kassette vorbehalten. Soll ein Programm von Diskette geladen werden, so ist jediglich LOAD <Dateiname> einzugeben.
Es ist hierbei auch wieder darauf zu achten, daß die auf Diskette existierende Datei vom TYP 'BAS' ist !
Beim einlagern eines Programmes in den Hauptspeicher durch Laden von Kasette wird eine automatische Prüfung vorgenommen.
Bei LOAD K,<Dateiname> wird eine Namensüberprüfung des auf der Kasette befindlichen Programms und des angegebenen Namen durchgeführt, wird eine Unstimmigkeit zwischen den Namen festgestellt, wird mit der Fehlermeldung 'Found '<Name> abgebrochen.

Achtung ! Diskettenfehler können nicht durch den ON ERROR Befehl abgefangen werden !

Beispiel 1: LOAD K, "*"
Lädt das erste sich auf der Kasette befindliche BASIC-Programm, ohne eine Namensüberprüfung durchzuführen.

Beispiel 2: LOAD K, "TEST"
Lädt das Programm TEST von Kasette, wenn es entdeckt wurde.

Beispiel 3: LOAD "DEMO"
Lädt das Programm DEMO von Diskette ein, befindet sich diese Datei nicht auf der Diskette, so wird eine Fehlermeldung ausgegeben.

L O C A T E

Syntax : LOCATE <X-Koordinate>,<Y-Koordinate> [,OFF]

Effekt : Positioniert den CURSOR absolut an die Stelle X,Y.

Bemerkung : <X-Koordinate> darf im Bereich von 0 bis 79,
<Y-Koordinate> im Bereich von 0 bis 23 liegen.
Hierbei liegt die Position 0,0 ganz links oben.
Durch die Option ',OFF' kann der CURSOR ausgeschaltet werden.
LOCATE ist nur im Textmodus anwendbar.

Beispiel :

```
10 CLS
20 PRINT : PRINT "Anrede       :"
30 PRINT : PRINT "Vor- Zuname  :"
30 PRINT : PRINT "Strasse      :"
40 PRINT : PRINT "PLZ Wohnort  :"
50 LOCATE 14,1 :INPUT AN$
60 LOCATE 14,3 :INPUT NAME$
70 LOCATE 14,5 :INPUT S$
80 LOCATE 14,7 :INPUT WO$
```

gibt eine Maske auf den Bildschirm aus und fragt nach
der jeweiligen Eingabe.

L P R I N T . . . L P R I N T U S I N G

Syntax : LPRINT <Ausdruck>[; <Ausdruck>] ...
LPRINT USING <Formatstring>; <Ausdruck> [;<Ausdruck>]

Effekt : Leitet eine Ausgabe an den Drucker.

Bemerkung : Es sind dieselben Optionen möglich, wie auch bei PRINT.
Sehen Sie deshalb auch unter PRINT nach.
Es ist nicht möglich LPRINT durch Eingabe von L? abzukürzen !
<Formatstring> hat folgende Formen :
1. Bei einem Zeichenkettenausdruck -> "!" oder "/ /"
2. Bei einem REAL-Ausdruck -> "#####.####"

"!" Liefert als Ausgabe jeweils nur den Anfangsbuchstaben der Zeichenkette.
"/ /" liefert genau die Anzahl Zeichen, die durch / / festgelegt sind. (Minimal 2, da // zur Länge gerechnet wird.)

Beispiel : FOR I = 0 TO 10 : LPRINT I; " "; : NEXT
Gibt auf den Drucker die Zahlenfolge 1 bis 10 mit Zwischenraum aus.

M I D $

Syntax : MID$ (<Stringvariable>,<Startposition>) = <Zeichenkette>

Effekt : Ersetzt einen Teil der in <Vairiable> enthaltenen Zeichenkette durch <Zeichenkette>

Bemerkung : Es ist sicherzustellen, daß <Startposition> innerhalb der in <Stringvariable> enthaltenen Zeichenkette liegt, da MID$ ansonsten keinen Effekt zeigen würde.
Die in <Stringvariable> enthaltene Zeichenkette wird ab der Stelle <Startposition> durch <Zeichenkette> ersetzt.
Ist <Zeichenkette> länger als das zu ersetzende Stück in <Stringvariable>, so wird diese in vollem Umfang fortgesetzt.

Beispiel :
```
10 A$ = "HALLO IHR DA"
20 MID$ (A$,7) = "WIE GEHT ES DIR ?"
30 PRINT A$
```

Gibt 'HALLO WIE GEHT ES DIR ?" auf den Bildschirm aus.

M O V E

Syntax : MOVE <Schrittweite>

Effekt : Schreitet <Schrittweite> Schritte in Blickrichtung der Schildkröte.

Bemerkung : Es wurde auch die vom Grundprogramm bekannte TURTLEGRAPHIK implementiert. MOVE Schreitet von der letzten Schildkrötenposition um <Schrittweite> in Blickrichtung.
Bei allen Graphikbefehlen gilt, daß diese auf allen 4 Graphikseiten der GDP64K Platine anwendbar sind.
Die vom Grundprogramm her bekannte Schildkröte wird hierbei in keinem Falle angezeigt.

Beispiel : MOVE 100: TURN 60 : MOVE 100 :TURN 60 MOVE 100
zeichnet ein gleichseitiges Dreieck auf die aktuelle Schreibseite.

M O V E T O

Syntax : MOVETO <X-Koordinate>,<Y-Koordinate>

Effekt : Positioniert die Zeichenposition auf die Stelle <X-Koordinate>,<Y-Koordinate> (Turtlegraphikbefehl)

Bemerkung : Mit MOVETO ist es möglich die Schildkröte auf eine gewünschte Postion des Graphikbildschirms zu bringen, ohne eine Spur zu hinterlassen.
Außerdem wird die Zeichenposition auf die Koordinaten <X-Koordinate>,<Y-Koordinate> aktualisiert, was ein relatives Zeichnen von Kreisen Linien etc. möglich macht.

Beispiel :

```
10  PAGE 3,3 : CLPG
20  MOVETO 100,100
30  FOR I% = 1 TO 10
40  CIRCLE STEP (20,20),I%*10
50  NEXT
```

Zeichnet 10 Kreise, die in ihrer Position und Größe verändert werden.

N A M E

Syntax : NAME <Alter Dateiname> , <Neuer Dateiname>

Effekt : Benennt eine Disketten-Datei um.

Bemerkung : Dieser Befehl ist nur in der Diskettenversion verfügbar.
Eine Umbenennung jeglicher Diskettendateien ist möglich.
Es wird jediglich der Name im Inhaltsverzeichnis der Diskette geändert.
Die Dateinamen sind bei RL-DOS auf 16 Zeichen beschränkt.

Beispiel : NAME "TEST" , "Graphik-Demo"

benennt die Datei 'Test' in 'Graphik-Demo' um.

N E W

Syntax : NEW

Effekt : Löscht den, durch ein Programm und seine Daten belegten, Arbeitsbereich.

Bemerkung : Ein durch NEW gelöschtes Programm kann nicht mehr gestartet werden. Alle Variablen werden dem Befehl CLEAR entsprechend gelöscht. DIMensionierung zurückgesetzt.
Nach NEW kann mit der Eingabe eines neuen Programmes begonnen werden.

ON ... GOSUB / ON ... GOTO

Syntax : ON <Integer-Ausdruck> GOTO <Liste von Zeilennummern>
ON <Integer-Ausdruck> GOSUB <Liste von Zeilennummern>

Effekt : Übergibt die Programmkontrolle an eine Programmzeile in einer Auflistung, abhängig vom errechneten Ergebnis des numerischen Ausdrucks.

Bemerkung : Durch ON .. GOTO , ON .. GOSUB ist es möglich eine mehrstellige Fallunterscheidung zu realisieren.
Der Wert des Integer-Ausdrucks bestimmt welche Zeilennummer aus der Liste zum Sprung ausgewählt wird.
Ist der Wert des Ausdrucks 1, so wird das Programm bei der ersten Zeilennummer der Liste fortgeführt.
Ist er 2, wird das Programm bei der zweiten Zeilennummer der Liste fortgeführt usw...
Falls der Wert des Ausdrucks 0 oder größer als die Anzahl der in der Liste vorhandenen Zeilennummern ist, wird das Programm mit der nächsten Anweisung fortgesetzt.

Bei ON..GOSUB muß jede Zeilennummer der Liste die erste Zeile eines Unterprogrammes darstellen.
Nach Beendigung des Unterprogrammes wird die nach ON..GOSUB folgende Anweisung ausgeführt.

Beispiel :
```
10 INPUT "AUSWAHL (1..3)" ; I%
20 IF I%<1 OR I%>3 THEN 10
30 ON I%  GOTO 100,200,300 :END
100 REM PROGRAMMTEIL 1

200 REM PROGRAMMTEIL 2

300 REM PROGRAMMTEIL 3
```

ON ERROR GOTO

Syntax : ON ERROR GOTO <Zeilennummer>

Effekt : Übergibt die Kontrolle an <Zeilennummer>, falls ein Fehler im Programmablauf entdeckt wird.

Bemerkung : Nachdem im Programm einmal ON ERROR GOTO .. ausgeführt wurde, wird der Programmablauf zur spezifizierten Zeile umgelenkt, falls ein Fehler im Programm entdeckt wurde. <Zeilennummer> sollte die erste Zeile einer Fehlerbehandlungsroutine sein, die den aufgetreten Fehler entsprechend behandelt.
ON ERROR GOTO sollte nicht während der Programmentwicklung eingesetzt werden, da ansonsten Syntax-Fehler nicht entdeckt würden.

Beispiel :

```
0 ON ERROR GOTO 1000
10 INPUT I%
20 PRINT I%
30 GOTO 10
1000 PRINT "SIE HABEN EINEN EINGABEFEHLER GEMACHT";
1010 PRINT " BITTE WIEDERHOLEN"
1020 GOTO 10
```

in Zeile 0 wird die Fehlerroutine dem Programm bekannt gemacht. Falls nun bei der Eingabe ein Fehler gemacht wird, würde das Programm abgebrochen. Dieser Fehler wird aber durch die Fehlerbehandlungsroutine abgefangen !

P A G E

Syntax : PAGE <Schreibseite>,<Leseseite>

Effekt : Wählt eine Graphikseite zum Beschreiben und eine zum Lesen der Information aus.

Bemerkung : Die GDP64K - Baugruppe ermöglicht es auf 4 unabhängigen Graphikseiten zu arbeiten. Dabei kann natürlich immer nur eine gleichtzeitig angezeigt werden. Durch PAGE wird nun ausgewählt, welche Seite angezeigt und welche mit Information beschrieben wird. Das heißt Schreib- und Leseseite müssen nicht unbedingt übereinstimmen.
So können Graphiken im Hintergrund aufgebaut werden, während eine zweite Graphik angezeigt wird.
<Schreibseite>,<Leseseite> muß im Bereich von 0 bis 3 liegen.

Beispiel :
```
10 PAGE 2,0
20 CLPG
30 FOR I% = 20 TO 200 STEP 20
40 CIRCLE (255,I%+50),I%
50 NEXT
60 PAGE 2,2
70 FOR J= 0 TO 10000 :NEXT
```

Zeichnet Kreise auf Graphikseite 2 während noch Seite 0 angezeigt wird.
Nachdem alle Kreise gezeichnet sind, wird auf Seite 2 umgeschaltet.

P O K E

Syntax : POKE <Adresse>,<Byte-Wert>

Effekt : Schreibt in Speicheradresse <Adresse> den Wert <Byte-Wert> ein.

Bemerkung : Als Speicheradresse ist jede Adresse gemeint, die durch den 68008 Prozessor ansprechbar ist, also auch die Adressen des Input-Output-Bereichs ab $FFFFFF00.
Da es sich bei <Adresse> und <Byte-Wert> um ganzzahlige Argumente handelt, dürfen auch sedezimale oder binaere Eingaben gemacht werden.
Auf einen OUT-Befehl wurde deshalb verzichtet, da durch POKE derselbe Effekt erreicht wird.
<Byte-Wert> liegt im Bereich von 0 bis 255, eine Bereichsüberschreitung wird mit 'Range overflow error' quittiert.

Beispiel : POKE $E802A,4 :PRINT "DRUCKER FUNKTIONIERT"
In Speicherstelle $e802a wird ein Wert von 4 eingeschrieben. Ist das Grundprogramm nach $e0000 verschoben, so ist dies die Adresse IOSTAT für Ausgabeumlenkung, d. h. Die Ausgabe "DRUCKER FUNKTIONIERT" erscheint nun auf dem Drucker, anstatt auf dem Bildschirm. Vorsicht ! Dieses Beispiel nur mit angeschlossenem Drucker ausprobieren.

P R E S E T

Syntax : PRESET [STEP] <X-KOORDINATE>,<Y-KOORDINATE>

Effekt : Setzt einen Punkt auf dem Graphikbildschirm unabhängig von der voreingestellten Zeichenfarbe zurück.

Bemerkung : <x-Koordinate>,<Y-Koordinate> ist im Bereich von 0 bis 511 zu wählen, um einen Effekt zu bemerken. Das Bildschirmfenster wird auch hier als quadratisch angesehen. (512x512 Bildpunkte) Mit STEP ist eine relative Positionierung zur letzten Zeichenposition zu erreichen.
Da die GDP64K Karte bisher nur eine Schwarz- Weis-Darstellung zuläßt, bewirkt ein PRESET das zurücksetzen eines Bildschirmpunktes auf Schwarz.

Beispiel:

```
10 PAGE 3,3 : CLPG
20 PSET 255,255 : FOR I = 1 TO 10:NEXT
30 PRESET 255,255 : FOR I = 1 TO 10 : NEXT
40 GOTO 20
```

Lässt einen Bildschirmpunkt blinken. Das kleine Programm ist nur durch CTRL-C abzubrechen.

P R I N T

Syntax : PRINT [<Ausdruck>[,¦;<Ausdruck>...] [,¦;]

Effekt : Berechnet den (die) Ausdruck (Ausdrücke) und gibt das (die) Ergebniss(e) auf den Bildschirm aus.

Bemerkung : Ausführen von PRINT ohne <Ausdruck> spezifiziert zu haben, bewegt den Cursor eine Zeile tiefer, ohne irdendetwas auszugeben.

<Ausdruck> darf entweder eine Stringkonstante, Stringvariablen, numerische Konstanten oder numerische Variablen beinhalten.

Unterschiedliche Ausdruckstypen dürfen in derselben PRINT Anweisung nebeneinander benutzt werden. Stringkonstanten müßen in Hochkomma eingebunden werden.

Jeder im PRINT-Befehl angeführte Ausdruck muß entweder durch ein Komma oder durch ein Strichpunkt abgetrennt werden.
Wird ein Strichpunkt als Trennzeichen eingesetzt, schließt der nächste Ausdruck unmittelbar an den vorangegangen an.
Werden Ausdrücke durch Komma abgetrennt, werden die Werte der Ausdrücke zu Beginn einer Tabulatorposition ausgegeben.
(Jede Tabulatorzone entspricht 16 Zeichen)
Andere Ausgabeformate können durch TAB, SPC oder SPACE$ erreicht werden.

Sollte ein Komma oder ein Semikolon am Ende der Liste stehen, bleibt der Cursor auf der gegenwärtigen Zeile. Weiter Ausgaben folgen dann in derselben Zeile.
Ohne Komma oder Semikolon wird der Cursor zur nächsten Zeile fortgeschaltet.

Bei der Eingabe kann PRINT durch ein Fragezeichen (?) ersetzt werden.

Beispiel 1:
```
10 INPUT "GEBE ZWEI ZAHLEN EIN (A,B) ";A,B
20 PRINT "A=";A,"B=";B
30 PRINT
40 PRINT "A+B=";A+B

RUN
GEBE ZWEI ZAHLEN EIN (A,B) 40,60
A=40          B=60

A+B=100
```

Beispiel 2:
```
10 PRINT 123,456,789
20 PRINT 123;456;789
30 PRINT "123";"456";"789"
40 PRINT "ABC";
50 PRINT "DEF"
60 PRINT "ABC","DEF"

RUN
123          456          789
123456789
133456789
ABCDEF
ABC          DEF
```

Beispiel 3:
```
10 FOR I=1 TO 40
20 PRINT CHR$(1);"E @SCHREITE ";-I
30 PRINT CHR$(1);"E @DREHE 90"
40 END
```
Verdeutlicht den Aufruf von Assemblerroutinen durch BASIC, wie dies auch in PASCAL/S möglich ist.

P R I N T U S I N G

Syntax : PRINT USING <Format String>;<Liste von Ausdrücken>

Effekt: Druckt Ausgaben gemaeß angegebenem Format. Nützlich zur Ausgabe formatierter Listen.

Bemerkung : <Format String> beinhaltet spezielle Zeichen, die das Format für die Ausgabe von <Liste von Ausdrücken> festlegen. <Liste von Ausdrücken> dürfen sowohl Zeichenketten, als auch arithmetische Ausdrücke sein. Jeder Ausdruck in der Liste muß vom nachfolgenden durch ein Semikolon getrennt sein. Die Zeichen in <Format String> unterscheiden sich, je nach dem die Ausdrücke der Liste Zeichenketten oder numerische Ausdrücke darstellen. Diese Zeichen und deren Funktion werden nachfolgend behandelt.

Formatzeichen für Zeichenketten :

! Ein Ausrufungszeichen als <Format String>, führt dazu, daß von jedem Ausdruck der Liste nur das 1. Zeichen angezeigt wird.

Beispiel:
```
10 PRINT USING "!";"AAAAA";"BCDE";"CDEFGH"
RUN
ABC
```

\n Leerzeichen\ Zwei Schrägstriche mit n Leerzeichen dazwischen, legt ein Ausgabeformat fest, das 2+n Zeichen umfasst. Die ersten 2+n Zeichen einer jeden Zeichekette der Liste werden ausgegeben. Zwei Zeichen werden angezeigt, falls kein Leerzeichen spezifiziert war. Weitere Zeichen, die über das Format hinausgehen, werden abgeschnitten; sind zu wenige Zeichen vorhanden, werden rechts Leerzeichen aufgefüllt.

Beispiel :
```
10 A$ = "1234567"
20 B$ = "ABCDEFG"
30 PRINT USING "\    \";A$;B$
40 PRINT USING "\         \";A$;B$

RUN
12345ABCDE
1234567    ABCDEFG
```

Format Zeichen für arithmetische Ausdrücke :

Die Position in welcher die Ziffern der Zahlen angezeigt werden, ist durch Angabe von # Zeichen festgelegt, wobei jedes # einer Ziffer entspricht. Ist die Zahl der Ziffern kleiner als die Zahl der # im Format-String, wird die Zahl rechtsbündig justiert. Ist die Zahl hingegen größer, wird ein Prozentzeichen (%) und ide gesamte Ziffernfolge ausgegeben. Minus-Zeichen werden vorne mit einbezogen, Plus-Zeichen hingegen werden nicht ausgegeben.

```
Beispiel :  10 PRINT USING "####";1
            20 PRINT USING "####";0.12
            30 PRINT USING "####";12.6
            40 PRINT USING "####";12345
            RUN
               1
               0
              13
            %12345
```

```
Beispiel 2:  10 PRINT USING "###.#";1
             20 PRINT USING "###.#";0.12
             30 PRINT USING "###.#";12.6
             40 PRINT USING "###.#";123.45
             RUN
               1.0
               0.1
              12.6
             123.5
```

P S E T

Syntax : PSET [STEP] <X-Koordinate>,<Y-Koordinate>

Effekt : Setzt einen Punkt auf Position <X-Koordinate>,<Y-Koordinate> des Graphikbildschirms.

Bemerkung : <x-Koordinate>,<Y-Koordinate> ist im Bereich von 0 bis 511 zu wählen, um einen Effekt zu erzielen. Das Bildschirmfenster wird auch hier als quadratisch angesehen. (512x512 Bildpunkte) Mit STEP ist eine relative Positionierung zur letzten Zeichenposition zu erreichen.
Da die GDP64K Karte bisher nur eine Schwarz- Weis-Darstellung zuläßt, bewirkt ein PSET das Setzen eines Bildschirmpunktes auf Weiss unabhähgig von der Zeichenfarbe.

BEISPIEL :
```
10  RANDOMIZE 511
20  PAGE 3,3 : CLPG
30  FOR I = 1 TO 4000
40  PSET RND,RND
50  NEXT
```

Setzt zufällig gewählte Punkte auf Graphik-Seite 3.

R A N D O M I Z E

Syntax : RANDOMIZE <Integer-Ausdruck>

Effekt : Setzt den Bereich für den Zufallszahlengenerator auf <Integer-Audruck>

Bemerkung : Mit RANDOMIZE wird der Bereich für den Zufallszahlen-generator gesetzt. Die Funktion RND liefert bei ihrem Aufruf eine Zufallszahl, die dann immer im Bereich von <Integer-Ausdruck> liegt.
RND liefert immer eine ganze Zufallszahl, Kommazahlen sind entsprechend durch Division/Multiplikation zu erzielen.

Beispiel :

```
10 RANDOMIZE 511
20 PAGE 2,2 : CLPG
30 FOR I = 1 TO 1000
40 GOSUB 1000 : REM QUADRATE ZEICHNEN
50 NEXT
60 GOTO 20
1000 MOVETO RND,RND : FOR J = 1 TO 4
1010 MOVE 10:TURN 90
1020 NEXT:RETURN
```

Zeichnet zufällig gewählte Quadrate auf den Bildschirm.

R E A D

Syntax : READ <Variable>

Effekt : Ordnet Werte aus einer Dataanweisung einer Variablen zu.

Bemerkung : Werte die durch READ in Variablen eingelesen werden, müssen irgendwo im Programm per DATA Anweisung definiert worden sein. READ liest aus einer Liste von Konstanten immer nur ein Element und weist dieses <Variable> zu. Ein scheinbarer Zeiger wird dabei auf die nächste Konstante einer DATA-Anweisung fortgeschaltet, die dann bei einem weiteren READ gelesen werden kann.
Der Typ von <Variable> und derjenige aus <Konstanten-Liste> der DATA-Anweisung müssen übereinstimmen.

Der Zeiger, der auf die Konstanten der DATA-Liste weist, kann durch ein RESTORE zurückgesetzt werden.

Wird versucht mehr Konstanten zu lesen, als dies vorhanden sind, erhält man die Fehlermeldung 'Out of DATA error'.

Beispiel :
```
10 FOR I=1 TO 4 : READ A$
20 PRINT A$ : NEXT
30 DATA "Dies ist ein Probetext,"
40 DATA "der die READ - Anweisung verdeutlichen soll."
50 DATA "Hierzu wird ein Probetext in eine Variable"
60 DATA "eingelesen und dann ausgegeben."
RUN
Dies ist ein Probetext,
der die READ - Anweisung verdeutlichen soll.
Hierzu wird ein Probetext in eine Variable
eingelesen und dann ausgegeben.
```

R E S T O R E

Syntax : RESTORE [<Zeilennummer>]

Effekt : Setzt den DATA-Zeiger die bestimmte Zeilennummer, bzw. setzt den DATA-Zeiger zurück, um die Daten neu lesen zu können.

Bemerkung : Wird <Zeilennummer> angegeben, greift ein erneutes READ auf das erste Datum, das in dieser Zeile folgt, zu. Wird <Zeilennummer> bei der Ausführung von RESTORE nicht spezifiziert, greift die nächste READ-Anweisung auf das erste Datum des ersten DATA - Zeile zu.

Beispiel:

```
10 FOR I=1 TO 4
20 READ A$(I):READ B$(I)
30 NEXT I
40 FOR I=1 TO 4
50 PRINT A$(I),B$(I)
60 NEXT I
65 PRINT
70 RESTORE 150
80 FOR I=1 TO 2
90 READ D$(I) : READ E$(I)
100 NEXT I
110 FOR I=1 TO 2
120 PRINT D$(I),E$(I)
130 NEXT I
140 DATA "AAAA","BBBB","CCCC","DDDD","EEEE","FFFF"
150 DATA "GGGG","HHHH","IIII","JJJJ","KKKK"
RUN
AAAA          BBBB
CCCC          DDDD
EEEE          FFFF
GGGG          HHHH

GGGG          HHHH
IIII          JJJJ
```

R E S U M E

Syntax : RESUME <Zeilennummer>

Effekt : Fährt nach einem Fehler mit dem Programmlauf fort.

Bemerkung : RESUME soll immer in einer Fehlerbehandlungsroutine eingesetzt werden.
RESUME ermöglicht es den Programmlauf ab einer bestimmten Zeile, Anweisung nach einem Fehler wieder aufzunehmen.
Die Stelle an der die Programmausführung ansetzt ist durch das Format der RESUME-Anweisung bestimmt.
RESUME wiederholt die Anweisung, bei der ein Fehler auftrat.
RESUME <Zeilennummer> setzt die Programmausführung in Zeile <Zeilennummer> fort.

RESUME <Zeilennummer> ist auf jedem Fall einem RESUME vorzuziehen, da es bei RESUME ohne Zeilennummer des öfteren zu weiteren Fehlern kommen kann.

Beispiel :
```
10 ON ERROR GOT 70
20 CLS
30 INPUT "Geben Sie eine Zahl zwischen 1 und 9 ein"; A
40 IF A<1 OR A>9 THEN ERROR 3
50 PRINT A
60 END
70 IF ERR = 3 THEN PRINT "Falsche Eingabe ": RESUME 30

RUN
Geben Sie eine Zahl zwischen 1 und 9 ein 0
Falsche Eingabe
Geben Sie eine Zahl zwischen 1 und 9 ein 10
Falsche Eingabe
Geben Sie eine Zahl zwischen 1 und 9 ein 5
5
```

R U N

Syntax : RUN [<Zeilennummer>] 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)

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.

S A V E

Syntax : SAVE [K,] <Dateiname>

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 RL-DOS 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.

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 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 O U N D

Syntax : SOUND <Parameter 1>,<Parameter 2>, ... ,<Parameter 16>

Effekt : Gibt ein Geräusch über die SOUND-Karte auf einen Lautsprecher aus.

Bemerkung : Dieser Befehl zeigt nur in Verbindung mit der SOUND-Karte eine Wirkung. Die 16 Parameter sind dann wie im Grundprogramm beschrieben auszuwählen. Experimente mit der Soundkarte konnten bisher nicht durchgeführt werden, sodaß hier auf die Herstellerunterlagen verwiesen werden muß.
Bei der SOUND-Programmierung ist die READ-DATA Anweisung eine gute Einrichtung. Es kann die Programmierung hierdurch enorm erleichtert werden.

S T O P

Syntax : STOP

Effekt : Unterbricht den Programmlauf.

Bemerkung : STOP wird gewöhnlich dazu benutzt Fehler während der Programmentwicklung schneller zu entdecken. Variablenwerte können so einfach abgefragt oder geändert werden, um dann das Programm mit CONT fortzusetzen.

Folgende Meldung wird angezeigt, wenn das Programm durch eine STOP-Anweisung abgebrochen wurde :
Stopped in xxxx
Wobei xxxx die Zeile bezeichnet, in der die Programmausführung auf die Anweisung STOP getroffen ist.

S W A P

Syntax : SWAP <Variable 1>,<Variable 2>

Effekt : Vertauscht den Inhalt zweier Variablen.

Bemerkung : SWAP kann dazu benutzt werden, den Inhalt zweier Variablen jeglichen Types zu vertauschen. Die Typen der beiden Variablen müssen allerdings übereinstimmen.
SWAP ersetzt einen Dreieckstausch, wie er notwendig ist, wenn zwei Variableninhalte vertauscht werden sollen.

Dieser Befehl wird insbesondere beim Vertauschen in Sortieralgorithmen Anwendung finden, wo es auf schnelles Vertauschen ankommt.

Beispiel :

```
10 FOR I = 1 TO 4
20 INPUT "Name eingeben :"; A$(I)
30 NEXT I
40 FOR I=2 TO 4
50 IF A$(I-1)>A$(I) THEN SWAP A$(I-1),A$(I) : I=1
60 NEXT I
70 FOR I=1 TO 4
80 PRINT A$(I)
90 NEXT

RUN
Name eingeben :Schmitt
Name eingeben :Maier
Name eingeben :Schulze
Name eingeben :Huber
Huber
Maier
Schmitt
Schulze
```

S Y S T E M

Syntax : SYSTEM

Effekt : Verlässt RL-BASIC und kehrt ins Grundprogramm oder Betriebssystem zurück.

Bemerkung : SYSTEM ist die einzige Möglichkeit ins Grundprogramm zurückzukehren. Das Programm sowie alle Variablen bleiben für einen WARMSTART erhalten. Wird SYSTEM in Verbindung mit RL-DOS verwendet, so wird die Kontrolle an RL-DOS übergeben. (Nur dann, wenn RL-BASIC aus dem Betriebssystem aufgerufen wurde.)

T I M E $

Syntax : TIME$ = "<HH>:<MM>:<SS>"

Effekt : TIME$ setzt die auf der Baugruppe UHR vorhandene Systemuhr.

Bemerkung : TIME$ zeigt nur dann einen Effekt, falls die Baugruppe UHR eingesetzt ist.
Als Anweisung verwendet, setzt TIME die Systemuhr. <HH> bezeichnet eine zweistellige Zahl zwischen 00 und 23, was die Stundenzahl repräsentiert.
<MM>,<SS> bezeichnet eine zweistellige Zahl die im Bereich von 00 bis 59 die Minuten bzw. Sekunden angibt.

Als Variable liefert TIME$ die momentane Uhrzeit wieder.

Beispiel :

```
10 PRINT "Laufende Uhrzeit :"; TIME$
20 INPUT "Neue Uhrzeit eingeben ( HH: MM: SS) "; A$
30 TIME$ = A$
40 PRINT TIME$

RUN
Laufende Uhrzeit : 18: 57: 34
Neue Uhrzeit eingeben ( HH: MM: SS) 16: 44: 40
16: 44: 40
```

T R O N / T R O F F

Syntax : TRON [,<Variable>]
TROFF

Effekt : Diese Befehle werden zur Abarbeitung des Programmes in Einzelschritten verwendet.

Bemerkung : Bei der Fehlersuche ist es durchaus nützlich das Programm in Einzelschritten ausführen zu können.
TRON startet den Einzelschritt-Modus. TROFF schaltet diesen wieder ab.
Die nächste aktuelle Anweisung wird am unteren Bildschirmrand angezeigt.
TRON ,<Variable> startet den Einzelschrittmodus, und gibt zusätzlich den Inhalt der spezifizierten Variable aus.
TRON,<Variable> funktioniert nur dann richtig, wenn TRON als Anweisung im Programm enthalten ist, bzw. wenn das Programm mit GOTO statt RUN gestartet wird.
Beim Programmende wird ein TRON automatisch wieder abgeschaltet.
Die Fortschaltung zum nächsten Befehl erfolgt durch Drücken der RETURN oder LEERTASTE. Soll das Programm nicht in Einzelschritten verfolgt werden, ist eine andere Taste zu drücken. In den Einzelschrittmodus gelangen Sie wieder durch LEERTASTE oder RETURN

Beispiel :
```
10 TRON,S : FOR I = 1 TO 10
20 S =S + I
30 NEXT
40 PRINT S
```

TRON verfolgt den Programmlauf ab der FOR-Anweisung.
Alle Werte die S druchläuft wird in spitzen Klammern angezeigt.

T U R N / T U R N T O

Syntax : TURN [TO] <Winkelgrad>

Effekt : Dreht die Schildkröte um einen bestimmten Winkel.

Bemerkung : TURN <Winkelgrad> dreht die Schildkröte in mathematisch positivem Sinn um <Winkelgrad> von ihrer aktuellen Blickrichtung nach links.
TURN TO <Winkelgrad> setzt die Blickrichtung der Schildkröte absolut auf <Winkelgrad>.
<Winkelgrad> darf im Bereich von 0 bis 359 liegen.

Da die Schildkröte nicht sichtbar ist, zeigt TURN nur dann einen Effekt, wenn es in Zusammenhang mit MOVE verwendet wird.

Beispiel: MOVE 10 : TURN 90 : MOVE 10 : TURN 90 : MOVE 10 : TURN 90: MOVE 10
zeichnet ein Quadrat ab der aktuellen Zeichenposition.

V E R I F Y

Syntax : VERIFY [K,] <Dateiname>

Effekt : Vergleicht den Inhalt des Programmspeichers mit dem Inhalt des Programmes auf Diskette oder Kassette.

Bemerkung : VERIFY (Vergleich) sollte auf jeden Fall nach dem Speichern eines Programmes auf Kassette erfolgen, da hier die Fehlerwahrscheinlichkeit höher als beim Speichern auf Diskette liegt.
VERIFY vergleicht byteweise das sich im Arbeitsbereich befindliche Programm mit dem abgespeicherten.
<Dateiname> setzt sich aus maximal 16 Zeichen als Dateiname + 2 Anführungszeichen zusammen.

VERIFY K,<Dateiname> vergleicht das Programm auf Kassette mit dem im Speicher.

VERIFY <Dateiname>
dem im Speicher.

Beispiel : VERIFY K,"LINES"
Vergleicht das Programm im Speicher mit dem Programm LINES auf Kassette.

W A I T

Syntax : WAIT <Speicheradresse> , <Byte-Wert>

Effekt : Wartet solange mit der Programmfortführung, bis in <Speicheradresse> der Wert <Byte-Wert> erscheint.

Bemerkung : Durch WAIT ist es möglich das Programm solange zu unterbrechen, bis ein bestimmtes Ereignis eintritt.
Z.B Warten bis eine Taste gedrückt wurde, oder ein Eingabeport einen bestimmten Wert innehat.

Beispiel :

```
.
.
100 WAIT $FFFFFF68,32
.
.
```

Das Programm wird solange nicht fortgeführt, bis die Leertaste gedrückt wird.

WHILE ... WEND

Syntax : WHILE <Bool'scher Ausdruck>

.
.

[<Schleifenrumpf>]
.
.

WEND

Effekt : Die WHILE ... WEND Anweisung wird dazu verwendet, eine Reihe von Anweisunge, die zwischen WHILE und WEND stehen, solange auszuführen, wie <Bool'scher Ausdruck> den Wert WAHR besitzt.

Bemerkung : Diese beiden Anweisungen werden in derselben Art verwendet, wie die FOR .. NEXT Anweisung, mit der Ausnahme, daß die Schleife solange ausgeführt wird, bis <Bool'sche Ausdruck> nicht länger erfüllt ist.

Wie bei FOR..NEXT-Schleifen, können WHILE..WEND-Schleifen ineinander verschachtelt werden. (Sie können auch in FOR..NEXT Schleifen enthalten sein oder umgekehrt)
Wenn Schleifen ineinander geschachtelt werden, gehört das erste WEND zur innersten WHILE-Anweisung, das zweite WEND zur nächst inneren WHILE und so weiter.

Eine "WHILE without WEND" Fehlermeldung erscheint, falls auf ein WHILE ohne zugehöriges WEND getroffen wird.
Eine "WEND without WHILE" Fehlermeldung erscheint, falls auf ein WEND ohne vorangegangenes WHILE getroffen wurde.

W R I T E

Syntax : WRITE <X-Position>,<Y-Position>,<Schriftgröße>,<Zeichenkette>

Effekt : Gibt eine Zeichenkette auf dem Graphikbildschirm mit <Schriftgröße> aus.

Bemerkung : WRITE gibt eine Zeichenkette auf einer Seite des Graphikbildschirms ab Position <X-Position>,<Y-Position> in der vorgewählten Schriftgröße aus.
Die Schreibseite muß zuvor durch PAGE ausgewählt werden.

<X-Position>,<Y-Position> kann im Bereich von 0 bis 511 liegen.

<Schriftgröße> wird in derselben Weise gesetzt, wie dies im Grundprogramm gemacht wird. ($11 ist 80 Zeichen/Zeile, $21 bedeutet 40 Zeichen/Zeile ...)

<Zeichenkette> ist ein String-Ausdruck, wie er auch in einer PRINT-Anweisung Verwendung finden würde.
Numerische Werte müssen zuvor noch in eine Zeichenkette gewandelt werden. (Durch STR$...)

Beispiel :
```
10 PAGE 2,2
20 CLPG
30 WRITE 100,100,$33,"H A L L O"
40 FOR I =1 TO 10000 : NEXT
```

Gibt auf Graphikseite 2 den Text "HALLO" in großer Schriftgröße aus.

A B S

Syntax : ABS (<numerischer Ausdruck>)

Effekt : Liefert den Absolut-Betrag einer Zahl zurück.

Ergebnistyp : Integer, Real

Bemerkung : Diese Funktion liefert den Absolutbetrag eines Ausdruckes zurück.
Als <numerischer Ausdruck> gilt jeder berechenbare Integer- (Ganzzahl) oder REAL-Ausdruck (Gleitkomma)
Alle Zahlen besitzen ein Vorzeichen, welches durch ABS in eine vorzeichenlose Zahl konvertiert wird, d. h. immer als positiv oder Null angegeben wird.

Beispiel :
```
10  PRINT ABS (10)
20  PRINT ABS (0)
30  PRINT ABS (-30)
RUN
10
0
30
```

A S C

Syntax : ASC (<Zeichen>)

Effekt : Liefert den ASCII - Code des angegebenen Zeichens zurück

Ergebnistyp : Integer

Bemerkung : <Zeichen> kann sowohl ein einzelnes Zeichen als auch eine Zeichenkette sein, wobei aber nur das erste Zeichen ausgewertet wird.
Jedem Zeichen wird durch einen genormten Code eine Zahl zwischen 0 und 255 zugeordnet.
Die Codes 0 bis 31 enthalten nicht druckbare Zeichen.

Beispiel :
```
10 A$ = INKEY$ : IF A$ = "" THEN 10
20 PRINT ASC(A$)
30 GOTO 10
```
Liefert den ASCII-Code der Taste zurück, die niedergedrückt wird.

A T N

Syntax : ATN (<numerischer Ausdruck>)

Effekt : Berechnet den Arcustangens von <numerischer Ausdruck>

Ergebnistyp : REAL

Bemerkung : <Numerischer Ausdruck> darf dem gesamten Zahlenbereich entstammen.
Als Ergebnis erhaelt man einen Wert zwischen -pi/2 .. +pi/2.

Beispiel :

```
10 PAGE 3,3 : CLPG
20 FOR W = -256 TO 256
40 X% = INT (W+256)
50 Y% = INT (ATN (W)* 200+ 256)
60 PSET X%,Y%
70 NEXT
80 FOR I = 1 TO 10000 : NEXT
```

Zeichnet eine ARCUSTANGENS-Kurve auf den Bildschirm

B I N $

Syntax : BIN$ (<ganzzahliger Ausdruck>)

Effekt : Liefert eine Zeichenkette, die dem binären Äquivalent des ganzzahligen Ausdrucks entspricht.

Ergebnistyp : String

Bemerkung : <ganzzahliger Ausdruck> darf nur im Bereich von 0 bis 255 liegen. Ein vorangestelltes Prozentzeichen (%) wird nicht mit ausgegeben, dies muß gegebenenfalls hinzugefügt werdem.

Beispiel :

```
10 INPUT "Zahl zwischen 0 und 255 ";B%
20 IF B%<0 OR B%>255 THEN 10
30 PRINT B%;" entspricht %";BIN$(B%);" binär"

RUN
Zahl zwischen 0 und 255 15
15 entspricht %00001111 binär
```

C H R $

Syntax : CHR$ (<ganzzahliger Ausdruck>)

Effekt : Liefert ein Zeichen zurück, das den ganzzahligen Wert im ASCII-Code repräsentiert.

Ergebnistyp : String

Bemerkung : CHR$ ist die zu ASC duale Funktion.
<ganzzahliger Ausdruck> kann nur im Bereich von 0 bis 255 liegen.

Beispiel :
```
10 FOR I = 0 TO 255
20 PRINT CHR$(I);
30 NEXT
```

gibt den gesamten Zeichensatz auf dem Bildschirm aus.

C O S

Syntax : COS (<Numerischer Ausdruck>)

Effekt : Berechnet den Cosinus eines numerischen Ausdrucks.

Ergebnistyp : Real

Bemerkung : <Numerischer Ausdruck> stellt einen Winkel, der in Radiant angegeben werden muß, dar.
Die Umrechnung von Winkelgrad in Radiant erfolgt so :
x = W*Pi/180, wobei Pi die Kreiszahl 3.1415926 darstellt.

Beispiel :

```
10 PAGE 3,3 : CLPG
20 FOR W = -6.28 TO 6.28 STEP 0.02
40 X% = INT (W*30+255)
50 Y% = INT (COS (W)*200)
60 PSET X%,Y%
70 NEXT
80 FOR I = 1 TO 10000 : NEXT
```

Zeichnet eine Cosinus-Kurve auf den Bildschirm

C S R L I N

Syntax : CSRLIN

Effekt : Liefert den aktuellen Y-Wert der Cursorposition zurück.

Ergebnistyp : Integer, Real

Bemerkung : Das beschreibbare Textfenster der Seiten 0 und 1 ist genau 80 * 24 Zeichen groß.
Die Schreibmarke (Cursor) befindet sich , wenn manchmal auch nicht sichtbar, immer auf einer dieser Positionen.
Die Y-Koordinate der Cursorposition ist durch CSRLIN festzustellen. CSRLIN liefert eine ganze Zahl im Bereich von 0 bis 23, wobei 0 der obersten und 23 der untersten Textzeile entspricht.

C O S

Syntax : COS (<Numerischer Ausdruck>)

Effekt : Berechnet den Cosinus eines numerischen Ausdrucks.

Ergebnistyp : Real

Bemerkung : <Numerischer Ausdruck> stellt einen Winkel, der in Radiant angegeben werden muß, dar.
Die Umrechnung von Winkelgrad in Radiant erfolgt so :
x = W*Pi/180, wobei Pi die Kreiszahl 3.1415926 darstellt.

Beispiel :

```
10 PAGE 3,3 : CLPG
20 FOR W = -6.28 TO 6.28 STEP 0.02
40 X% = INT (W*30+255)
50 Y% = INT (COS (W)*200)
60 PSET X%,Y%
70 NEXT
80 FOR I = 1 TO 10000 : NEXT
```

Zeichnet eine Cosinus-Kurve auf den Bildschirm

C S R L I N

Syntax : CSRLIN

Effekt : Liefert den aktuellen Y-Wert der Cursorposition zurück.

Ergebnistyp : Integer, Real

Bemerkung : Das beschreibbare Textfenster der Seiten 0 und 1 ist genau 80 * 24 Zeichen groß.
Die Schreibmarke (Cursor) befindet sich , wenn manchmal auch nicht sichtbar, immer auf einer dieser Positionen.
Die Y-Koordinate der Cursorposition ist durch CSRLIN festzustellen. CSRLIN liefert eine ganze Zahl im Bereich von 0 bis 23, wobei 0 der obersten und 23 der untersten Textzeile entspricht.

D A T E $

Syntax : DATE$

Effekt : Gibt das aktuelle Datum aus.

Ergebnistyp : String

Bemerkung : DATE$ liefert nur dann einen Vernünftigen Wert, falls die Uhrenkarte vorhanden ist und das Datum bereits durch DATE$=... gesetzt wurde.
Das Ergebnis von DATE$ wird als Zeichenkette mit dem Format dd.mm.jj zurückgeliefert.
(dd = Tag, mm = Monat , jj = Jahr)

Beispiel : PRINT DATE$
24.12.85

E R L / E R R

Syntax : ERL bzw. ERR

Effekt : Liefert entweder die Zeilennummer oder den Fehlercode, der gerade aufgetreten ist.

Ergebnistyp : Integer, Real

Bemerkung : ERL, bzw ERR können in einer Fehlerbehandlungsroutine dazu verwendet werden, Fehler gezielt herauszufiltern. Eine Übersicht über alle Fehlertypen mit ihren Codes finden Sie im Anhang B.

Beispiel :

```
10 ON ERROR GOTO 80
20 CLS
30 INPUT "Geben Sie eine Zahl zwischen 1 und 9 ein "; A
40 IF A<1 OR A>9 THEN ERROR 200
50 PRINT A
60 ERROR 210
70 END
80 IF ERR = 200 THEN PRINT "Nochmal": RESUME 30
90 IF ERR = 210 THEN PRINT "Fehlercode "; ERR; " in Zeile "; ERL
100 END
RUN
Geben Sie eine Zahl zwischen 1 und 9 ein 0
Nochmal
Geben Sie eine Zahl zwischen 1 und 9 ein 5
5
Fehlercode 210 in Zeile 60
```

E X P

Syntax : EXP (<Numerischer Ausdruck>)

Effekt : Berechnet eine Potenz der eulerschen Zahl e.

Ergebnistyp : Real

Bemerkung : <Numerischer Ausdruck> wird ausgewertet und dann versucht e mit diesem Wert zu potenzieren. (e = 2.71828...)

Beispiel :

```
PRINT EXP (0)
1

PRINT EXP (1)
2.718282

PRINT EXP (2)
7.389056
```

F R E

Syntax : FRE (0)

Effekt : Gibt an wieviel Speicherplatz noch zur Verfügung steht.

Ergebnistyp : Integer

Bemerkung : Durch Setzen des Arbeitsbereiches wird der für BASIC verfügbare Speicherbereich eingegrenzt.
Wird nun ein Programm eingegeben, so kann nur dieser Bereich ausgenutzt werden.
FRE (0) wird dafür benutzt, um den für das Programm oder Variablen noch freien Speicher zu erfahren.
Die Einheit ist Bytes.

Um den maximal verfügbaren Speicherplatz zu erfahren, lösche man mit CLEAR alle Variablen und gebe dann print fre (0) ein.

H E X $

Syntax : HEX$ (<numerischer Ausdruck>)

Effekt : Gibt den durch <numerischer Ausdruck> repräsentierten Wert als sedezimale Zahl aus.

Ergebnistyp : String

Bemerkung : <numerischer Ausdruck> sollte ein Ganzzahliger Ausdruck sein, der noch in das Format einer Integerzahl passt.
Führende Nullen werden mit ausgegeben, das Dollarzeichen ($) hingegen nicht.
Durch HEX$ ist eine einfache Wandlung zwischen den beiden Zahlensystemen (dezimal , sedezimal) möglich.

Beispiel : print hex$ (100)
00000064

I N K E Y $

Syntax : INKEY$

Effekt : Liefert ein Zeichen von der Tastatur, falls eine Taste während der Abarbeitung dieses Befehls gedrückt wurde.

Ergebnistyp : String

Bemerkung : INKEY$ gibt einen Leerstring ("") zurück, falls der Tastaturpuffer (bei KEY genau ein Zeichen) leer ist. Wurde irgendeine Taste gedrückt, dessen Code in der ASCII-Tabelle enthalten ist, liest INKEY$ diese vom Tastaturpuffer ein und übergibt den Wert in Form eines Zeichens.
Zeichen, die durch INKEY$ von der Tastatur eingelesen wurden, werden auf dem Bildschirm nicht angezeigt.
(Ein Cursor erscheint nicht !)

I N S T R

Syntax : INSTR ([<Startpos>,]<Suchstring>,<Musterstring>)

Effekt : Liefert die Position zurück ab der <Musterstring> in <Suchstring> gefunden wurde, 0 sonst.

Ergebnistyp : Integer, Real

Bemerkung : Mit <Startpos> kann die Startposition definiert werden, ab der die Suche beginnen soll.
Als <Suchstring>,<Musterstring> sind alle Stringausdrücke erlaubt.
Es wird nach dem ersten Auftreten von <Musterstring> in <Suchstring> gefahndet.
Wird <Startpos> nicht spezifiziert, beginnt die suche ab Position 1 des Suchstrings.
<Startpos> darf nicht ausserhalb der Länge des Suchstrings liegen.

Beispiel :

```
10 X$ = "ABC.EIBIDKJKLNJDSKLNN"
20 Y$ = "KL"
30 P1% = INSTR (X$,Y$)
40 P2% = INSTR (13,X$.Y$)
50 PRINT P1%,P2%

RUN
12          18
```

```
10 X$ = " 1.11 "
20 X$ = #0000" + X$
30 Y$ = "."
40 P2% = INSTR (4,X$,Y$)
50 P2% = P2% - 4
60 PRINT P2%

RUN

2
```

'INSTR' zeigt unter Startposition -4- nichts [illegible] -0- an

I N T

Syntax : INT (<Numerischer Ausdruck>)

Effekt : Gibt die größte ganze Zahl, die größer oder gleich <Numerischer Ausdruck> ist, zurück.

Ergebnistyp : Integer, Real

Bemerkung : Alle numerischen Ausdrücke können für <Numerischer Ausdruck> eingesetzt werden.

Beispiel :

```
10 PRINT "I","INT(I)"
20 FOR I =-4 TO 4 STEP 0.5
30 PRINT I,INT(I)
40 NEXT I

RUN
I             INT(I)
-4            -4
-3.5          -4
-3            -3
-2.5          -3
-2            -2
-1.5          -2
-1            -1
-0.5          -1
0             0
0.5           0
1             1
1.5           1
2             2
2.5           2
3             3
3.5           3
4             4
```

L E F T $

Syntax : LEFT$ (<Zeichenkette>,<Position>)

Effekt : Wählt den linken Teil von <Zeichenkette> bis zu <Pos.> aus.

Ergebnistyp : String

Bemerkung : <Zeichenkette> bezeichnet einen beliebigen String-Ausdruck.
<Position> darf nur im Bereich von 1 bis zur maximalen
Länge der Zeichenkette liegen.

Beispiel :

```
10 FOR I = 1 TO 4
20 INPUT "Zeichenkette :"; A$
30 PRINT LEFT$( A$, 1)
40 NEXT

RUN
ABCDEF
A
CELL
C
HALLO
H
MAERCHEN
M
```

L E N

Syntax : LEN (<Zeichenkette>)

Effekt : Liefert die Länge von <Zeichenkette> zurück.

Ergebnistyp : Integer, Real

Bemerkung : <Zeichenkette> darf alle Stringfunktionen enthalten. Man beachte aber die in Vers. 1.0 auf 80 Zeichen beschränkte größe von Zeichenketten.

Beispiel:

```
10 INPUT "Zeichenkette eingeben "; A$
20 IF A$="" THEN END
30 PRINT "enthält "; LEN( A$); " Zeichen"
40 GOTO 10

RUN
Zeihenkette eingeben HALLO
enthält 5 Zeichen
Zeichenkette eingeben DFJDDJFHSDK
enthält 11 Zeichen
Zeichenkette eingeben
```

L O G

Syntax : LOG (<Numerischer Ausdruck>)

Effekt : Berechnet den natürlichen Logarithmus einer Zahl.

Ergebnistyp : Real

Bemerkung : LOG berechnet den natürlichen Logarithmus aus <Numerischer Ausdruck>. Negative Zahlen oder die Null führen zu einem Wertebereichsfehler.
Mit LOG lässt sich auch jeder weitere Logarithmus einer beliebigen Basis berechnen :

$$\log_{10}(X) = Y \Leftrightarrow \text{LOG}(X) / \text{LOG}(10)$$

LOG (X)/LOG(10) berechnet den Logarithmus von X zur Basis 10.

Beispiel :

```
10 INPUT "ZAHL >0 :";X
20 IF X<=0 THEN 10
30 PRINT "LOGARITHMUS =";LOG(X)

RUN
ZAHL >0 :10
LOGARITHMUS =2.302585
```

M I D $

Syntax : MID$ (<Zeichenkette>,<Start>,<Länge>)

Effekt : Wählt ab <Start> ein <Länge> langes Stück aus <Zeichenkette> aus.

Ergebnistyp : String

Bemerkung : Es ist bei dieser String-Funktion wieder auf die Länge der Zeichenkette zu achten. Des weiteren darf <Start> natürlich nicht über die Länge von <Zeichenkette> hinausragen !
<Länge> kann dabei auch über die tatsächliche Länge der Zeichenkette spezifiziert werden.

Beispiel :

```
10 X$ = "NDR mit dem Klein-Computer"
20 Y$ = MID$(X$,1,3)+"-"
30 Y$ = Y$+MID$ (X$,13,14)
40 Z$ = " und RL-BASIC"
50 PRINT Y$;Z$

RUN
NDR-Klein-Computer und RL-BASIC
```

P E E K

Syntax : PEEK (<Speicheradresse>)

Effekt : Liefert den Wert den <Speicheradresse> enthält

Ergebnistyp : Integer

Bemerkung : Mit PEEK können einzelne Speicherzellen des NDR-Klein-Computers angesehen werden. Da bei der 68008 CPU auch Ein- Ausgabeadressen als Speicherzellen angesehen werden, können natürlich auch diese durch PEEK inspiziert werden. PEEK gibt immer eine Zahl zwischen 0 und 255 aus, d.h. es wird immer nur eine Speicheradresse (Byte) ausgelesen.

Beispiel :
```
PRINT PEEK ($FFFFFF68)
141

(ist 128 + 13 für RETURN)
```

P O S

Syntax : POS

Effekt : Gibt die horizontale Position des Cursors aus.

Ergebnistyp : Integer

Bemerkung : Das beschreibbare Textfenster der Seiten 0 und 1 ist genau 80 * 24 Zeichen groß.
Die Schreibmarke (Cursor) befindet sich , wenn manchmal auch nicht sichtbar, immer auf einer dieser Positionen. Die X-Koordinate der Cursorposition ist durch POS festzustellen. POS liefert eine ganze Zahl im Bereich von 0 bis 79, wobei 0 dem linken und 79 dem rechten Bildrand entspricht.

R N D

Syntax : RND

Effekt : Wählt eine Zufallszahl.

Ergebnistyp : Integer, Real

Bemerkung : Der Bereich indem Zufallszahlen ausgewählt werden, wird durch RANDOMIZE festgelegt.
RND erzeugt immer eine ganze Zahl !

Beispiel :

```
10 PAGE 3,3 : CLPG
20 RANDOMIZE 511
30 FOR I= 1 TO 1000
40 X%=RND: Y%=RND
50 GOSUB 1000
60 NEXT
70 FOR I=1 TO 10000 ; NEXT
80 END
1000 CONNECT (X%,Y%)-(X%+10,Y%)-(X%+5,Y%+5)-(X%,Y%)
1010 RETURN
```

Zeichnet Dreiecke an zufällige Positionen

S G N

Syntax : SGN (<Numerischer Ausdruck>)

Effekt : Gibt das Vorzeichen des numerischen Ausdruckes aus

Ergebnistyp : Integer, Real

Bemerkung : SGN berechnet sich folgendermaßen :

$$\text{sgn}(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{falls } x = 0 \\ -1 & \text{falls } x < 0 \end{cases}$$

Beispiel :

```
10 INPUT X
20 PRINT SGN (X)

RUN
?10
1

RUN
?1
1

RUN
?-3
-1

RUN
?0
0
```

S I N

Syntax : SIN (<numerischer Ausdruck>)

Effekt : Berechnet den Sinus von <numerischer Ausdruck>

Ergebnistyp : REAL

Bemerkung : <Numerischer Ausdruck> stellt einen Winkel, der in Radiant angegeben werden muß, dar.
Die Umrechnung von Winkelgrad in Radiant erfolgt so :
x = W*Pi/180, wobei Pi die Kreiszahl 3.1415926 darstellt.

Beispiel :

```
10 PAGE 3,3 : CLPG
20 FOR W = -6.28 TO 6.28 STEP 0.02
40 X% = INT (W*30+255)
50 Y% = INT (SIN (W)* 200)

70 NEXT
80 FOR I = 1 TO 10000 : NEXT
```

Zeichnet eine Sinus-Kurve auf den Bildschirm

S P A C E $

Syntax : SPACE$ (<numerischer Ausdruck>)

Effekt : Liefert eine aus Leerzeichen bestehende Zeichenkette zurück.

Ergebnistyp : STRING

Bemerkung : <numerischer Ausdruck> legt die Länge der Zeichenkette fest und sollte einen ganzzahligen Wert der im Bereich von 1 bis 255 liegt berechnen.

Beispiel :

```
10  FOR I = 1 TO 10
20  PRINT "*";SPACE$(I*2);"*"
30  NEXT

RUN
*  *
*    *
*      *
*        *
*          *
*            *
*              *
*                *
*                  *
*                    *
```

S P C

Syntax : SPC (<numerischer Ausdruck>)

Effekt : Liefert eine Leerzeichenkette für Ausgabe auf den Drucker oder Bildschirm.

Ergebnistyp : STRING (nur in PRINT-Anweisungen anzuwenden)

Bemerkung : Im Gegensatz zur SPACE$ Funktion, die auch bei einer Zuweisung benutzt werden kann, darf SPC nur bei Ausgaben mittels PRINT oder LPRINT eingesetzt werden. Der Wert <numerischer Ausdruck> muss im Bereich von 1 bis 255 liegen.

Beispiel :

```
10 CLS
20 FOR I = 4 TO 20 STEP 4
30 LOCATE 20-I/2,CSRLIN
40 PRINT "*";SPC(I);"*"
50 NEXT
60 FOR I = 20 TO 4 STEP -4
70 LOCATE 20-I/2,CSRLIN
80 PRINT "*";SPC(I);"*"
90 NEXT
RUN
```

S T R $

Syntax : STR$ (<numerischer Ausdruck>)

Effekt : Wandelt <numerischer Ausdruck> in eine Zeichenkette um

Ergebnistyp : STRING

Bemerkung : <numerischer Ausdruck> kann ein zusammengesetzter Ausdruck sein, der vor der Umwandlung in eine Zeichenkette berechnet wird. Eventuell auftretende Fehler bei Berechnung des arthmetischen Ausdrucks werden nicht ignoriert !

Beispiel :

```
10  X$ = STR$ (3)
20  Y$ = X$ + X$
30  Z$= STR$(3*11)
40  PRINT X$,Y$,Z$

RUN
3               33              33
```

S T R I N G $

Syntax : STRING$ (<numerischer Ausdruck>,<Zeichenkette>)

Effekt : Produziert eine Zeichenkette der Länge <numerischer Ausdruck> mit dem Zeichen <Zeichenkette>

Ergebnistyp : STRING

Bemerkung : <Numerischer Ausdruck> darf im Bereich von 0 bis 255 liegen. <Zeichenkette> sollte nur aus einem einzigen Zeichen bestehen, ist <Zeichenkette> länger, so wird das erste Zeichen zur Bildung der Ergebniszeichenkette herangezogen.

Beispiel :

```
10 FOR I=1 TO 5
20 A$ = STRING$(I,"A")
30 PRINT A$
40 NEXT I
50 END

RUN
A
AA
AAA
AAAA
AAAAA
```

T A B

Syntax : TAB (<numerischer Ausdruck>)

Effekt : Positioniert die Schreibmarke relativ zu ihrer momentanen Stelle um <numerischer Ausdruck>.

Ergebnistyp : STRING (nur in PRINT Anweisung erlaubt)

Bemerkung : Die Schreibposition am linken Bildschirmrand ist auf 0 und der linke Bildschirmrand ist auf 79 festgelegt. <numerischer Ausdruck> muss im Bereich von 0 bis 255 liegen.
TAB positioniert nun den CURSOR (Schreibmarke) auf eine dieser Positionen zwischen linkem und rechtem Rand, ohne die bis dahin ausgegebenen Zeichen zu löschen.
Diese neue Schreibposition liegt dann genau um <numerischer Ausdruck> Zeichen von der letzten Position entfernt.

TAB arbeitet nur mit einer PRINT, LPRINT Anweisung.

Beispiel :
```
10 PRINT 1;2;3;4;5;6;7
20 PRINT 1;TAB(4);2;TAB(4);3;TAB(4);4;TAB(4);5;TAB(4);6

RUN
1234567
1   2   3   4   5   6
```

T A N

Syntax : TAN (<numerischer Ausdruck>)

Effekt : Berechnet den Tangens von <numerischer Ausdruck>

Ergebnistyp : REAL

Bemerkung : <Numerischer Ausdruck> stellt einen Winkel, der in Radiant angegeben werden muß, dar.
Die Umrechnung von Winkelgrad in Radiant erfolgt so :
x = W*Pi/180, wobei Pi die Kreiszahl 3.1415926 darstellt.

Beispiel :

```
10 PAGE 3,3 : CLPG
20 FOR W = -1.57 TO 1.57 STEP 0.02
40 X% = INT (W*60+255)
50 Y% = INT (TAN (W)*100) + 255
55 IF Y%<0 OR Y%> 512 THEN 70
60 PSET X%,Y%
70 NEXT
80 FOR I = 1 TO 10000 : NEXT
```

Zeichnet die Tangensfunktion auf den Bildschirm.

T I M E $

Syntax : TIME$

Effekt : Liefert die aktuelle Uhrzeit zurück.

Ergebnistyp : STRING

Bemerkung : TIME$ zeigt nur dann einen Effekt, falls die Uhrenbaugruppe vorhanden ist.
Die aktuelle Uhrzeit wird in der Form HH:MM:SS zurückgeliefert.
Zum Setzen der Uhr wird TIME$ = "<HH>:<MM>:<SS>" verwendet.

Beispiel : 10 LOCATE 60,1 : PRINT TIME$: GOTO 10

Zeigt die aktuelle Uhrzeit auf dem Bildschirm links oben an.

V A L

Syntax : VAL (<Zeichenkette>)

Effekt : Wertet <Zeichenkette> als Zahl aus.

Ergebnistyp : REAL

Bemerkung : <Zeichenkette> stellt eine Ziffernfolge dar, die in eine Zahl im REAL-Format gewandelt wird. Es ist sicherzustellen, daß <Zeichenkette> der Syntax einer REAL-Zahl entspricht, da ansonsten eine Fehlermeldung ausgegeben wird.

Beispiel :

```
10 A$ = "3"
20 B$ = "4"
40 C$ = A$ + B$ + "5"
50 PRINT C$, 2*VAL( C$)

RUN
345          690
```

V A R P T R

Syntax : VARPTR (<Variable>)

Effekt : Ermittelt die Speicherzelle ab der <Variable> im Speicher abgelegt ist.

Ergebnistyp : INTEGER

Bemerkung : Zum Teil ist es notwendig, den Inhalt einer Variable mit Maschinenprogrammen etc. zu verändern. VARPTR stellt fest, ab welcher Position sich die Variable <Variable> im Speicher befindet.
<Variable> darf eine Variable jeden Typs sein, auch eine Feldvariable !

Wichtig : INTEGER - Variablen belegen 4 Bytes
REAL - Variablen belegen 8 Bytes
und STRING - Variablen belegen 82 Bytes

Beispiel :
```
10 X% = 0
20 Z$ = HEX$( VARPTR( X%))
30 PRINT Z$

RUN
00005FFC
```

(Dieser Wert kann von Fall zu Fall unterschiedlich sein !)

A N H A N G A

Reservierte Schlüsselwörter

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

ABS AND ARC ASC ATN AUTO
BIN
CALL CHAIN CHR CIRCLE CLEAR CLPG CLS COLOR CONNECT COS CONT
DATA DATE DEF DELETE DIM DRAW
END ERASE ERL ERR ERROR EXP
FILES FOR FN FRE
GOSUB GOTO
HEX
IF INKEY INPUT INSTR INT
KILL
LEFT LEN LET LINE LIST LLIST LOAD LOCATE LOG LPRINT
MID MOVE MOVETO MOD
NAME NEW NEXT
ON OR
PAGE PEEK POKE POS PRESET PRINT PSET
RANDOMIZE READ REM RESTORE RESUME RETURN RIGHT RND RUN
SAVE SGN SIN SOUND SPACE SPC STOP STR STRING SWAP SYSTEM
TAB TAN TIME TO TRON TROFF TURN
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
21	":" missing
22	Stack overflow
23	Fatal error
24	WHILE without WEND error
25	No BASIC Data
26	Verify error
27	Checksum error

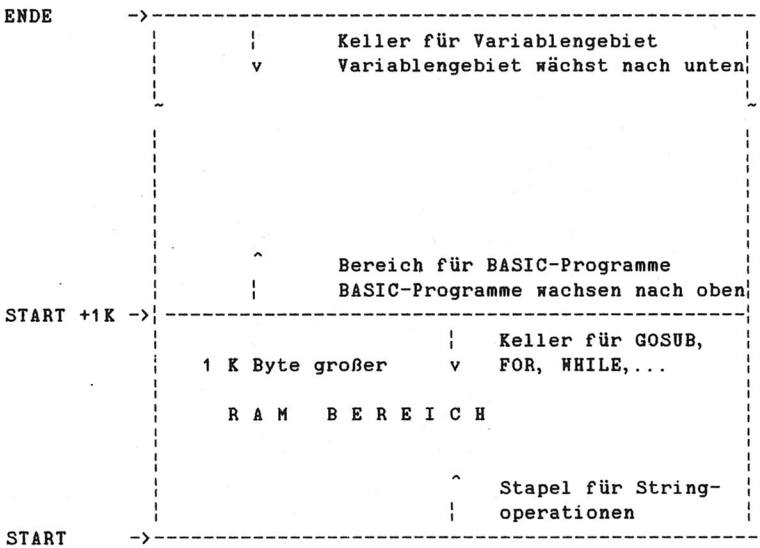
A N H A N G C

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 $1C00 bis $1E00, oberhalb des Grundprogramms. Z. B. bei einem System dessen Grundprogramm ab Adresse $000000 liegt, befinden sich die BASIC-Systemvariablen im Bereich von $9C00 - $9E00.
Dieser Speicherbereich steht dann Maschinenprogrammen etc. nicht mehr zur Verfügung !

Des weiteren sollte auch der Bereich von $1000 bis $1C00 oberhalb des Grundprogramms nicht benutzt werden, da hierin die Symboltabelle der BASIC-Programmvariablen gespeichert ist.

Effektiv frei verfügbar wäre damit der Bereich $1E00 -.... oberhalb des Grundprogramms.
Dies gilt aber nur dann, falls sich nicht das DOS im Speicher befindet, denn dieses liegt ab Adresse $4000 - $7000 oberhalb des Grundprogramms !

Speicheraufteilung oberhalb des Grundprogramms :

$x9000 - $x9C00 : Symboltabelle der Basicprogrammvariablen

$x9C00 - $x9E00 : Systemvariable des RL-BASIC-Interpreters

$x9E00 - $xC000 : Frei

$xC000 - $xF000 : RL-DOS incl. Inhaltsverzeichnis, falls DOS geladen

Für <x> ist jeweils die Startadresse des Grundprogramms einzusetzen :
Startadresse = $000000 => $009000 - $009C00 für die Symboltabelle.

```
0   REM     KLEINES SORTIERPROGRAMM MIT NICHT OPTIMIERTEM BUBBLESORT
1   REM
2   REM
3   REM      EINGABEENDE DURCH DRUECKEN DER RETURN TASTE
4   REM
10   CLS
11   PRINT "Kleines Sortierprogramm"
12   PRINT "-----------------------"
13   PRINT : PRINT "BUBBLESORT sortiert ihre Namen (Ende mit Leereingabe=<CR>)"
14   PRINT : PRINT "Maximal 20 Namen eingeben !": PRINT
15   DIM A$(20):I = 1
20   INPUT "WIE lautet ihr Name ? ";A$(I)
35   IF A$(I) <  > "" THEN I = I + 1: GOSUB 50: GOTO 20
36   PRINT : PRINT " Die sortierten Namen :"
40   FOR II = 1 TO I - 1: PRINT A$(II)
45   NEXT
48   END
50   IF I = 2 THEN  RETURN : REM  NICHTS ZU TUN !
55   FOR R = 1 TO I - 2
60   IF A$(R) > A$(R + 1) THEN  SWAP A$(R),A$(R + 1):V = 1
70   NEXT
80   IF V = 0 THEN 90 ELSE V = 0: GOTO 55
90   RETURN
```

```
0   REM  GRAPHIK-DEMO
1   REM  AUS QUADRATEN ZUSAMMENGESETZTE DREHGRAPHIKEN
2   REM
3   REM   "(c)  R. Lobreyer   1985
4   REM
6   REM  ABBRUCH DRUCH DRUECKEN DER SPACE-TASTE
7   REM  JEDER TASTENDRUCK ERZEUGT EINE ANDERE GRAPHIK
8   REM
10  A$ =  CHR$ (2)
20   PAGE 3,3: CLPG
25   TURN  TO 0
30   FOR I = 10 TO 200
40   MOVE I: TURN 90: MOVE I: TURN 90: MOVE I: TURN 90: MOVE I
45   TURN  ASC (A$)
50   NEXT
60  A$ =  INKEY$ : IF A$ = "" THEN  GOTO 60
70   IF A$ =  CHR$ (13) OR A$ = " " THEN  END  ELSE  GOTO 20
80   GOTO 10
```

```
0   REM  GRAPHIK-DEMO 2
1   REM  NETTE LINIEN
2   REM
3   REM  "(c) R. Lobreyer 1985
4   REM  NUR DURCH CTRL-C ABBRECHBAR
5   REM
6   REM
8   RANDOMIZE 8
10   PAGE 3,3: CLPG
13  E =  RND
15  S =  RND  *  - 1
16   IF E = 0 AND S = 0 THEN 13
20   FOR I = S TO E STEP 0.075
40  X = 255 +  SIN (I) * 255:Y = 255 +  COS (I) * 255
50   LINE ( INT (I * 30 + 255),X) - ( INT (255 - I * 30),Y)
60   NEXT
65   FOR I = 1 TO 1000: NEXT : GOTO 10
```

```
0   REM  GRAPHIK-DEMO 2
1   REM   NETTE LINIEN  2
2   REM
3   REM  "(c) R. Lobreyer 1985
4   REM  NUR DURCH CTRL-C ABBRECHBAR
5   REM
6   REM
8   RANDOMIZE 8
10   PAGE 3,3: CLPG
13  E =  RND
15  S =  RND  *  - 1
16   IF E = 0 AND S = 0 THEN 13
20   FOR I = S TO E STEP 0.075
40  X = 255 +  SIN (I) * 255: Y = 255 +  COS (I) * 255
50   LINE ( INT (I * 30) + 255, X) - (Y, 255 -  INT (I * 30))
60   NEXT
65   FOR I = 1 TO 1000: NEXT : GOTO 10
```

```
REM  GRAPHIK - DEMO 3
REM  KREISE UND LINIEN
REM
REM  (C) R. LOBREYER 1985
REM
REM  DURCH JEDE TASTE ABBRECHBAR
REM
 PAGE 3,3: CLPG
 FOR I = 50 TO 450 STEP 50
 FOR J = 450 TO 50 STEP  - 50
 CIRCLE (I,J),50: LINE (I,J) - (J,I)
 NEXT : NEXT
 CLPG
A$ =  INKEY$ : IF A$ = "" THEN 90 ELSE  END
 FOR J = 1 TO 0 STEP  - 1
 COLOR  = J
  FOR I = 30 TO 180 STEP 15
  CIRCLE (25 + I,255),I: CIRCLE (480 - I,255),I
  NEXT
  NEXT
 A$ =  INKEY$ : IF A$ = "" THEN 10 ELSE  END
```

R L - D O S

Das Diskettenbetriebssystem RL-DOS :

Mit dem Kauf eines BASIC-Interpreter haben Sie zusaetzlich dieses Produkt erworben, das das Speicher und Laden von Programmen / Daten auf Diskette erlaubt.

BASIC waere unvollstaendig, falls man nicht Programme laden oder speichern koennte. Diese Broschuere soll nun die Funktionen von RL-DOS aufzeigen.

Bei RL-DOS handelt sich um ein sehr rudimentaeres Diskettenbetriebssytem, aehnlich wie JOGI-DOS, jedoch nicht zu vergleichen mit Produkten wie etwa CP/M 68 K von Digital Research.

Triberg im Dezember 1985

Voraussetzungen fuer den Betrieb von RL-DOS :

1) Grundprogramm ab Version 4.0 (auch verschoben nach $E0000)
2) Mindestens 32K-Byte RAM direkt oberhalb des Grundprogramms.
3) FLO2 Diskettencontroller fuer den Betrieb mit RL-DOS
4) Diskettenlaufwerk 2*80 Spur 1 MB unformatiert 3,5" oder 5,25".
5) CPU68K mit den Zusatzkarten GDP64K , KEY und ROA fuer das Grundprogramm

RL-DOS unterstuetzt jediglich ein (mehrere) Laufwerk(e) die obengenannten Bedingungen genuegen. Eine Formatierung im NDR-80 Standardformat, d.h. 5 Sektoren a 1KByte pro Spur = 800KByte Nettokapazitaet, mu fuer den Betrieb bereits durchgefuehrt worden sein.

Starten von RL-DOS

Schalten Sie den Rechner ein und waehlen den Menuepunkt FLOPPY START an. Legen Sie erst jetzt eine Diskette ins Laufwerk ein.
Nach erscheinen einer Kurzmitteilung "Diskettenbetriebssystem wird geladen" erhalten sie folgende Bildschirmausgabe :

```
------------------------
Disketten-Betriebssytem
fuer NDR-Klein-Computer
-- Vers. 1.2  4/86    --
--   (c) R. Lobreyer  --
------------------------

A>_
```

RL-DOS ist nun bereit Eingaben anzunehmen und auszufuehren.

Nachfolgend werden alle Befehle einzeln erlaeutert, doch zuvor einige allgemeinen Bemerkungen :

Fuer Datenverluste, die aufgrund von RL-DOS zustande kommen, kann keine Haftung uebernommen werden !

RL-DOS ist nur fuer den privaten Gebrauch bestimmt, eine gewerbliche Nutzung unterliegt einer einmaligen Gebuehr.

Befehlstabelle RL-DOS :

RL-DOS bietet die Moeglichkeit Daten/Programme von Diskette oder auf Diskette zu speichern. RL-DOS ist nicht nur im Zusammenhang mit RL-BASIC einsetzbar, sondern kann auch zum Speichern von Texten oder Maschinenprogrammen herangezogen werden.

Zur Befehlssyntax:

Alle Befehle koennen in Klein- Grossbuchstaben oder gemischt eingeben werden.
Angaben in eckigen Klammern sind optional einzusetzen.
Angaben in spitzen Klammern sind Platzhalter fuer Adressen, die sowohl dezimal, sedezimal als auch als Variablen eingegeben werden koennen.

Eine Ausgabeumleitung fuer die meisten Befehle mit Bildschirmausgabe ist moeglich. Hier kann die Ausgabe auf eines der Geraete (unten als DEV bezeichnet) NIL:, CON:, LST:, LST2: oder USR: geleitet werden.
NIL: bezeichnet die Ausgabe ins Nichts. (Fuer Assembler wichtig)
CON: bezeichnet die Console (= GDP64K Karte), die Standardmaessig angesprochen wird.
LST:, LST2: ist der Centronics-Port, wobei LST2: dem LST: entspricht, ohne Ausgabe eines Zeilenvorschubes bei jedem <CR>

USR: ist die benutzerdefinierte Ausgabe ueber den Sprungvektor des Grundprogramms

Bis zu fuenf Laufwerke werden unterstuetzt, darunter eine RAM-Disk als Laufwerk E:
d: bezeichnet bei Dateinamen das anzusprechende Laufwerk und ist immer dann optional einzusetzen, falls ein Dateiname anzugeben ist.
<Dateiname> bezeichnet einen durch den Anwender vergebbaren Dateinamen der in seiner Laenge auf 16 Zeichen beschraenkt ist. Die Typerweiterungen wie BAS, BIN, TXT usw. werden durch RL-DOS selbstständig vergeben und haben keine unterscheidende Wirkung bei der Namensvergabe von Dateien !

Befehle in alphabetischer Reihenfolge :

ASSEM ASSEM [S] [d:"<Dateiname>"] [,dev]

Effekt: Ruft den Assembler des Grundprogramms auf. Es kann sowohl der im aktuellen Textfenster befindliche Text, als auch ein Text von Diskette assembliert werden.
Soll ein sich im Textfenster befindliches Assemblerprogramm uebersetzt werden, so ist einfach ASSEM einzugeben.
ASSEM "Assembler-text" uebersetzt einen Text von Diskette, ohne den gesamten Text in den Speicher zu laden.
Die Option S gibt an, ob die Symboltabelle vor dem Assemblerstart nicht zu loeschen ist. (Wird standardmaessig geloescht)
,dev dient der Ausgabeumlenkung. (lst: fuer Drucker, NIL: fuer Nichts)

DIR DIR [d:] [,dev]

Effekt: Gibt das Inhaltsverzeichnis der angewaehlten Diskette aus.
Mit dev kann die Ausgabe zum Beispiel auf Drucker umgeleitet werden.

DUMP DUMP [<Anfangsadresse>,<Endadresse>] [,dev]

Effekt: Macht einen Speicherauszug des Speicherbereichs von <Anfangsadresse> bis <Endadresse>. DUMP ohne Argumente gibt die naechsten 128 Bytes seit dem letzten DUMP-Befehl aus. Die Darstellung erfolgt sowohl Sedezimal, als auch als ASCII-Wert, wobei nicht darstellbare Zeichen als '.' gekennzeichnet sind.

EDIT EDIT [d:"<Dateiname>"[,<Ladeadr.>]] oder EDIT [<Startadresse>]

Effekt: Ruft den Grundprogrammtexteditor auf. Wird <Dateiname> angegeben, wird die Textdatei in den Speicher geladen, dann der Editor aufgerufen. Eingabe von EDIT <Startadresse> aendert das Textfenster auf <Startadresse> ab und ruft den EDITOR auf.

ERA ERA d:"<Dateiname>"

Effekt: Loescht eine Datei aus dem Inhaltsverzeichnis der Diskette.

EXIT EXIT

Effekt: Durch Eingabe von EXIT wird die Befehlsebene von RL-DOS verlassen und in das RDK-Grundprogramm zurueckgekehrt.

HELP HELP [,dev]

Effekt: Help gibt eine Liste aller Befehle des RL-DOS auf den Bildschirm samt seiner Syntax aus. Was die einzelnen Befehle bewirken wird allerdings nicht erlaeutert.

INIT INIT d:"<Diskettenname>" [,<Startadresse>,<Endadresse>]

Effekt: INIT kopiert das DOS auf eine im NDR-80-Format formatierte Diskette. Bei Eingabe von e:"<Diskettenname> muss weiterhin die Start- als auch die Endadresse eines Speicherbereichs angegeben werden, der als RAM-Disk benutzt wird (Mindestgroesse 6K)

Belassen Sie die Diskette mit dem original-DOS zuerst im Laufwerk, erst nach der Meldung 'wirklich initialisieren (j/n)' diese durch eine formatierte Diskette ersetzen und 'j' eintippen !!

LBIB LBIB [,dev]

Effekt: Gibt alle Bibliothekseintraege samt Startadresse und Laenge aus.

LOAD LOAD d:"<Dateiname>" [,<Ladeadresse>]

Effekt: Laedt eine Diskettendatei (Text, Binaerdaten ..) in den Hauptspeicher. Es kann die Ladeadresse angegeben werden, ab welcher Adresse RL-DOS die Informationen in den Speicher ablegt. Wird <Ladeadresse> nicht spezifiziert, wird die Datei an die durch LADEADRESSE im Direktory verzeichnete Stelle geladen.

LOCK LOCK d:"<Dateiname>"

Effekt: Setzt einen softwaremaessigen Schreibschutz auf die angegebene Datei. Es sollte nun nicht mehr moeglich sein, die Datei zu loeschen oder zu ueberschreiben. Der Schreibschutz wird im Inhaltsverzeichnis durch R/O angezeigt.

RUN RUN [d:"<Dateiname>"][,<Ladeadresse>] oder RUN <Startadresse>

Effekt: Fuehrt ein Maschinenprogramm aus und kehrt dann nach RL-DOS zurueck. Bei RUN "<Dateiname>" wird die Datei in den Arbeitsspeicher geladen und dann ab der Ladeadresse ausgefuehrt. (1. Befehl im Programm !!!) RUN <Startadresse> fuehrt ein sich im speicher befindliches Programm ab Adresse <Startadresse> aus.

```
RENAME          RENAME d:"<Alter Name>","<Neuer Name>"
-------------------------------------------------------------------------------
Effekt: Benennt eine schon vorhandene Diskettendatei um.
        Es wird dabei jediglich der Name im Direktroy geaendert.


SAVE            SAVE  d:"<Dateiname>" [,<Startadresse>[,<Endadresse>]]
-------------------------------------------------------------------------------
Effekt: Speichert eine Datei/Text/Programm auf Diskette.
        Speichern eines Textes:
         1) Text liegt im Textfenster :      SAVE d:"<Dateiname>"
         2) Text befindet sich im Speicher:  SAVE d:"<Dateiname>",<Textstart>
        Speichern eines Maschinenprogramms bzw einer Datei :
         SAVE d:"<Dateiname>",<Startadresse>,<Endadresse> oder
         SAVE d:"<Dateiname>",<Startadresse>,L<Laenge>


UNLOCK          UNLOCK  d:  "<Dateiname>"
-------------------------------------------------------------------------------
Effekt: Hebt den Schreibschutz einer durch LOCK schreibgeschuetzte Datei  auf.


VERIFY          VERIFY  d:"<Dateiname>"
-------------------------------------------------------------------------------
Effekt: Vergleicht den Inhalt einer Diskettendatei mit der sich noch im Speicher
        befindlichen Datei. Bei einem Vergleichsfehler wird die Adresse aus-
        gegeben, bei der sich ein Unterschied zwischen beiden ergeben hat.


TYPE            TYPE [d:"<Dateiname>"] [,dev]
-------------------------------------------------------------------------------
Effekt: Gibt den Inhalt einer Datei auf den Bildschirm aus.
        TYPE ohne Argumente gibt den aktuellen Inhalt des Textfensters aus.
        Eine Umlenkung der Ausgabe auf Drucker etc. ist moeglich.

-------------------------------------------------------------------------------
```

Telefonservice
08 31- 62 11
jeden Mittwochabend
bis 20.00 Uhr

Graf Elektronik Systeme GmbH
Magnusstraße 13 · Postfach 1610
8960 Kempten (Allgäu)
Telefon: (08 31) 62 11
Teletex: 831804 = GRAF
Telex: 17 831804 = GRAF
Datentelefon: (08 31) 6 93 30

Verkauf:
Computervilla
Ludwigstraße 18 b
(bei Möbel-Krügel)
8960 Kempten-Sankt Mang
Telefon: 08 31 / 6 93 00

Geschäftszeiten: GES GmbH + Verkauf
Mo. - Do. 8.00 - 12.00 Uhr, 13.00 - 17.00 Uhr
Freitag 8.00 - 12.00 Uhr
Telefonservice

Filiale Hamburg
Ehrenbergstraße 56
2000 Hamburg 50
Telefon: (0 40) 38 81 51

Filiale München:
Georgenstraße 61
8000 München 40
Telefon: (0 89) 2 71 58 58

Öffnungszeiten der Filialen:
Montag – Freitag
10.00 – 12.00 Uhr, 13.00 – 18.00 Uhr
Samstag 10.00 – 14.00 Uhr

ges