robotron Z1013
TINY BASIC 3.01

---

angepasst an Z9001 und modifiziert
V. Pohlers 2022

Inhaltsverzeichnis

Inhalt

Inhaltsverzeichnis 1
5.3. BASIC 3
5.3.1. Programmiersprache BASIC 3
5.3.2. Der BASIC-Interpreter 3
5.3.3. Laden des BASIC-Interpreters 4
5.3.4. Arbeit mit dem BASIC-Interpreter 4
5.3.5. Kommandos des BASIC-Interpreters 7
LIST 7
EDIT n 7
RUN 8
NEW 8
BYE 8
END 8
CSAVE 9
CLOAD 9
DIR 9
LET 9
IF 9
GOTO 10
FOR...TO...STEP...NEXT 10
GOSUB...RETURN 11
REM 11
INPUT 12
PRINT 12
STOP 13
CALL 13
PEEK 14
POKE 14
BYTE 15
WORD 15
HEX 15
' ' (QUOTE) 15

OUTCHAR 16
INCHAR 16
OUT 16
IN 17
I$ 17
O$ 17
LEN 17
ABS 18
TAB 18
TOP 18
SIZE 19
5.3.7. Dateiformat 19
Anlage 14: Basic-Beispielprogramme 19
1. Basic-Programm zum Speichern von Adressen und Telefonnummern 19
2. Basic-Programm "Begriffe raten" 20
3. Basic-Programm zum Raten einer durch die RND-Funktion ermittelten Zahl 20
4. Mathe-Übungsprogramm in Basic 21
5. Basic-Programm zur Darstellung von Größen im Balkendiagramm 21
6. Basic-Programm "Turm von Hanoi" 21
7. Basic-Programm "Mastermind" 21
8. Basic-Programm "23 Streichhölzer" 21

Auszug aus Z1013-Dokumention
Blau: Anpassung an Z9001

5.3. BASIC

5.3.1. Programmiersprache BASIC

BASIC ist eine sogenannte höhere Programmiersprache, eine Sprache also, die sich nicht direkt auf die Maschinensprache des Rechners bezieht. Diese Sprache wurde um 1965 von John G. Kemeny und Thomas E. Kurtz im Dartmouth College in den USA entwickelt. Sie hatten dabei die Entwicklung einer Computersprache vor Augen, die einerseits vom Anfänger leicht zu erlernen ist und andererseits viele Möglichkeiten bietet. Man sollte mit dieser Sprache leicht numerische (Zahlen-) Probleme angehen können, aber ebenso sollten Verwaltungsaufgaben, bei denen die Textverarbeitung eine große Rolle spielt, in der Sprache zu behandeln sein. Deshalb ersann man für diese Sprache die Bezeichnung BASIC, ein Wort, das konstruiert wurde aus den Anfangsbuchstaben von: Beginner's All purpose Symbolic Instruction Code (etwa: Anfänger-Allzweck-Symbolik-Instruktions-Code).

Neben den Wörtern "Beginner's" und "All purpose" (Allzweck), deren Bedeutung ohne weiteres einleuchtet, können die Wörter "Symbolic Instruction Code" vielleicht ein wenig verwirren. Damit wird nur gesagt, dass man mit einem sogenannten symbolischen Befehlskode arbeitet. Das sind Schlüsselwörter, mit denen bestimmte Verarbeitungsbefehle mit Hilfe von Symbolen angegeben werden, z. B. das Addieren oder Subtrahieren. Also nur wenige Wörter, deren Bedeutung bei der weiteren Beschäftigung mit der Sprache ohne weiteres einleuchten wird.

Kemeny und Kurtz haben mit ihrem einfachen Aufbau genau ins Schwarze getroffen. Die Erwartungen haben sich erfüllt: BASIC wurde eine sehr populäre Programmiersprache. Sie hat sich im Unterricht bewährt, und auch in Laboratorien und in Betrieben gibt es zahllose Computer, die den Befehlen in BASIC gehorchen.

5.3.2. Der BASIC-Interpreter

Die eingegebenen Programmzeilen müssen ihrem Inhalt entsprechend bestimmte Verarbeitungsleistungen aufrufen. Diese Aufgabe übernimmt der BASIC-Interpreter. Dieser Interpreter arbeitet Anweisung für Anweisung interpretativ ab, d. h. jedem entschlüsselten Befehl oder Kommando wird ein entsprechendes Maschinenprogramm zugeordnet und mit den in der BASIC-Anweisung angegebenen Zahlenwerten abgearbeitet. Diese interpretative Abarbeitung nutzt ein Maschinenprogramm für alle im gesamten

Programm vorkommenden gleichen Kommandos bzw. Befehle. Damit sind genauso viele verschiedene Maschinenprogramme notwendig, wie es verschiedene Kommandos und Befehle gibt. Damit ist die Länge des BASIC-Interpreter festgelegt, unabhängig von der Länge der abzuarbeitenden Programme. Als Nachteil ist die geringere Rechengeschwindigkeit gegenüber übersetzten Programmen anzusehen.

Ein Vorteil ist die einfache Dialogfähigkeit, BASIC-Zeilen erscheinen so wieder auf dem Bildschirm, wie sie eingegeben wurden. Außerdem können einzelne Anweisungen sofort ausgeführt werden. Auf diese Weise kann der Computer an Stelle eines Tischrechners verwendet werden. Auf diese Betriebsart wird noch genauer eingegangen.

An dieser Stelle sei auch auf einen Nachteil des Interpreters hingewiesen. Wie spätere Beispiele zeigen werden, kann ein Teil des BASIC-Programmes häufiger als nur einmal ausgeführt werden, z. B. bei der Unterprogrammarbeit oder in Programmschleifen. Vom Interpreter wird aber jedesmal, wenn der entsprechende Abschnitt an der Reihe ist, Anweisung um Anweisung neu interpretiert. Das hat zur Folge, dass solche Programme längere Verarbeitungszeiten gegenüber gleichen Programmlösungen in Maschinensprache erfordern. Im Extremfall kann das dazu führen, dass besonders zeitkritische Probleme nur in Maschinensprache lösbar sein werden. Diese Maschinenprogramme können dann innerhalb einer BASIC-Anweisung aufgerufen werden.

5.3.3. Laden des BASIC-Interpreters

Da der BASIC-Interpreter selbst ein Maschinenprogramm darstellt, benötigt er im Speicher einen Speicherplatz von ca. 2.75 KByte. Dazu wird noch weiterer Speicherraum zum Ablegen der BASIC-Programme gebraucht.

Der BASIC-Interpreter wird im OS des Z9001 mit dem Kommando „TB“ von Kassette geladen. Nach dem einmaligen Laden wird er vom OS aus mit „TB“ neu gestartet. Mit dem Kommando „WTB“ erfolgt ein Warmstart von Tiny-Basic.

5.3.4. Arbeit mit dem BASIC-Interpreter

Nach dem Start des BASIC-Interpreters erscheint auf dem vorher gelöschten Bildschirm die Meldung 'robotron Z 1013 BASIC 3.01', in der nächsten Zeile 'READY' und am Beginn der folgenden Zeile das Zeichen '>' (größer als) als Promptsymbol.

Immer wenn das Zeichen '>' auf dem Bildschirm auftaucht, befindet sich der Interpreter in einer Eingabeschleife, d. h., dass der

Interpreter zur Eingabe von Befehlen, Kommandos oder Programmzeilen bereit ist. Eine solche Programmzeile hat folgenden Aufbau: [Zlnr] anweisung 1 [; anweisung 2, ...]. Die eckige Klammer bedeutet, dass diese Eingaben nicht unbedingt getätigt werden müssen.

Die Länge einer Programmzeile darf 64 Zeichen nicht überschreiten. Jede Programmzeile und jede Kommandoeingabe ist mit der ENTER-Taste abzuschließen.

Beginnt die eingegebene Zeile mit einer Zeilennummer (Zlnr), so wird diese Zeile als Programmzeile interpretiert und abgespeichert. Diese Zeilennummern sind ganze Zahlen im Bereich zwischen 1 und 32767. Bei der Nummerierung der Programmzeilen geht man sinnvollerweise in Zehnerschritten vor. Dadurch ergibt sich die Möglichkeit, noch genügend Einfügungen in bereits bestehende Programme vorzunehmen. Die Abarbeitung des BASIC-Programmes erfolgt in der Reihenfolge der Zeilennummern.

Alle anderen Eingaben ohne Zeilennummern werden als Befehl zur sofortigen Ausführung angesehen. Sind sie zulässig werden sie ausgeführt, sonst erfolgt eine Fehlermeldung. Danach kann die nächste Eingabe erfolgen.

Die Arbeit ohne Zeilennummer nennt man auch Tischrechnermodus.

```
>A=66-20; PRINT A
   46
READY
```

Innerhalb einer einzugebenden Zeile kann beliebig oft mit den Cursortasten "Cursor links '<-'" oder "Cursor rechts '->'" korrigiert werden. Der Cursor wird unter das fehlerhafte Zeichen bewegt und durch Eingabe des richtigen Zeichens der Fehler behoben. Danach muss der Cursor wieder an das Zeilenende gebracht werden (auf die erste freie Zeichenstelle) und die Zeile kann mit der ENTER-Taste abgeschlossen werden.

Soll eine bereits im Programmspeicher abgespeicherte Programmzeile geändert werden, wird diese einfach neu eingegeben (mit der gleichen Zeilennummer). Soll eine Zeile gelöscht werden, so muss nur ihre Zeilennummer angegeben werden.

Alle Befehle und Kommandos können mit einem Punkt nach dem ersten oder nach weiteren Buchstaben abgekürzt werden. In der im Anhang befindlichen Befehlsliste sind die möglichen Abkürzungen zu finden. Es sind auch kürzere Varianten der Befehle möglich, aber dadurch kommt es im Interpreter unter Umständen zu Verwechslungen mit anderen Befehlen und zu fehlerhaften Abarbeitungen.

Auch Leerzeichen zwischen den einzelnen Elementen der Programmzeilen können weggelassen werden. Diese beiden Möglichkeiten der Programmverkürzung erlauben es, den Programmspeicher besser auszunutzen, um mehr Programmzeilen unterzubringen. Dadurch geht aber die Übersichtlichkeit der Programme verloren.

Es dürfen mehrere Anweisungen in einer Programmzeile untergebracht werden. Diese Anweisungen sind durch ein Semikolon voneinander zu trennen.

Der BASIC-Interpreter realisiert eine einfache Ganzzahlarithmetik in den vier Grundrechenarten Addition, Subtraktion, Multiplikation und Division (+ - * /) im Zahlenbereich von -32768 bis +32767. Aus diesem Grund besitzt er nur einen Datentyp. Zu beachten ist, dass auch die Division nur ganzzahlig ausgeführt wird, d. h., dass z. B. 9/4=2 ist. Der Teil des Resultates hinter dem Komma wird einfach weggelassen.

Den meisten der Schlüsselworte folgt ein weiterer Ausdruck. Das können Zahlen, Variable oder arithmetische Konstruktionen mit diesen sein. Erforderlichenfalls sind derartige Konstruktionen mit Klammern aufzubauen, um diese Konstruktionen mathematisch eindeutig zu machen. Der Interpreter arbeitet nach der bekannten Rechenregel: Punktrechnung geht vor Strichrechnung. Mehrere Klammern können dabei beliebig geschachtelt werden. Innerhalb eines Befehls können weitere Befehle oder Funktionen verwendet werden.

Als Variable sind alle Buchstaben des Alphabets von A bis Z erlaubt. Eine Variable darf aber nur aus einem einzelnen Buchstaben bestehen. Es können nur Großbuchstaben verwendet werden.

Mit dem Symbol '@' ist die Nutzung eines eindimensionalen Feldes (Vektor) möglich. Die Teilanweisung @(A) stellt dabei das A-te Element des Feldes dar. Anstelle von A kann sowohl ein anderer Buchstabe des Alphabetes als auch eine Zahl oder eine arithmetische Konstruktion stehen, d. h. ein Ausdruck wie oben bereits beschrieben.

Werden bei der Arbeit mit dem BASIC-Interpreter Syntaxfehler gemacht, so werden diese Fehler erkannt und angezeigt. Logische Fehler im Programm kann der Interpreter nicht finden, das bleibt dem Geschick des Programmierers überlassen. Der BASIC-Interpreter kennt drei verschiedene Fehlermeldungen:

WHAT? - Das Schlüsselwort bzw. der Ausdruck sind nicht erlaubt bzw. fehlerhaft, d. h. der Interpreter versteht die Anweisung nicht.

HOW? - Die Ausführung der Anweisung ist im Rahmen der Möglichkeiten dieses Interpreters nicht möglich (z. B. bei einer Zahlenbereichsüberschreitung).

SORRY - Die Ausführung der Anweisung ist zwar möglich, aber nicht unter den aktuellen Voraussetzungen (z.B. der Programmspeicher ist erschöpft).

Tritt eine Fehlermeldung bei der Abarbeitung eines Programmes auf, so wird zur Fehlermeldung auch die Zeile ausgegeben, in der der Fehler aufgetreten ist. An der fehlerhaften Stelle wird vom BASIC-Interpreter ein Fragezeichen eingefügt. Das erleichtert die Fehlersuche.

5.3.5. Kommandos des BASIC-Interpreters

Im nachfolgenden sind alle Befehle und Kommandos, die der BASIC-Interpreter verstehen und ausführen kann, aufgeführt und erläutert. Zusammen mit dem im vorhergehenden Abschnitt gesagten ergeben sich daraus alle Möglichkeiten der im MRB Z 1013 realisierten Programmiersprache BASIC. Die im Anhang befindliche Befehlsliste beinhaltet auch die möglichen Kurzformen.

LIST

Dieses Kommando bewirkt das Auflisten des im Speicher stehen den BASIC-Programmes auf dem Bildschirm. Dadurch lassen sich Programme leicht kontrollieren.

>LIST Auflisten des gesamten Programmes in aufsteigender Reihenfolge der Zeilennummern.

>LIST 10 Auflisten des BASIC-Programmes ab der Zeile 10. ~~Es werden genau 20 Zeilen aufgelistet.~~ Nach jeweils 16 Zeilen stoppt das Listen, der Computer wartet auf einen Tastendruck. Mit Strg-C / Stop wird das Listen beendet, jede andere Taste listet die nächsten 16 Zeilen.

Strg-S (Pause) lässt die Ausgabe langsamer laufen.

EDIT n

Mit EDIT gefolgt von einer Zeilennummer wird die Zeile zum Editieren bereitgestellt. Das funktioniert wie bei der Ersteingabe einer Zeile. Bei Druck auf ENTER wird alles übernommen, was vor dem Cursor steht.

RUN

Mit dem Kommando 'RUN' wird ein BASIC-Programm gestartet. Bei der Abarbeitung wird mit der niedrigsten Zeilennummer begonnen. Ist die letzte Zeile abgearbeitet oder eine 'STOP'- Anweisung erreicht, kehrt der Interpreter in die Eingabeschleife zurück und meldet sich mit 'READY' und auf der nächsten Zeile mit den Aufforderungszeichen '>'. Er erwartet jetzt weitere Kommando- oder Befehlseingaben.

NEW

Das im Speicher vorhandene BASIC-Programm wird scheinbar gelöscht. Tatsächlich ist es noch im Programmspeicher enthalten, wird aber bei Neueingabe eines Programmes überschrieben.

BYE

Mit 'BYE' wird die Arbeit mit dem BASIC-Interpreter beendet und ins Monitorprogramm zurückgekehrt. Das zuletzt eingegebene Programm befindet sich noch im Speicher. Wird an dessen Inhalt nichts geändert, kann mit dem Monitorkommando 'J 0103' wieder der BASIC-Interpreter aktiviert werden (Restart). Jetzt erscheint nur ein 'READY' und in der nächsten Zeile das Aufforderungszeichen '>' auf dem Bildschirm. Die Arbeit mit dem BASIC-Interpreter kann fortgesetzt und das vorher eingegebene Programm wieder gestartet werden.

END

Dieser Befehl wird zum Vergrößern des vom Interpreter genutzten Programmspeichers verwendet. Mit 'END' kann also das Programmspeicherende neu gesetzt werden, z. B. bei Speichererweiterung. Lässt der augenblickliche Ausstattungsgrad des MRB Z1013 diesen Speicherbedarf nicht zu, weil er real nicht vorhanden ist, wird die Fehlermeldung 'SORRY' ausgegeben. Dabei ist zu beachten, dass ein Bereich von 140 Byte hinter den Programmspeicher frei bleibt, der vom BASIC-Interpreter intern verwaltet wird.

Beispiel:

```
>END HEX(3FFF)-140
```

Die oben aufgeführten Kommandos dürfen in keinem Programm auftauchen, sie dienen nur zur Arbeit mit dem Interpreter und werden prinzipiell sofort ausgeführt.

CSAVE

Mit dem Kommando CSAVE "name" wird ein BASIC-Programm unter dem angegebenen Namen auf Magnetband abgespeichert. Dabei kann der Name maximal 16 Zeichen umfassen und ~~muss~~ kann von Anführungszeichen eingeschlossen sein.

CLOAD

Mit diesem Kommando CLOAD "name" wird ein durch CSAVE abgespeichertes BASIC-Programm wieder von Magnetband eingegeben. ~~Zur Kontrolle wird der im CSAVE-Kommando verwendete Name auf dem Bildschirm ausgegeben.~~

DIR

Listet alle Tiny-BASIC-Programme auf.

5.3.6. Programmierbare Befehle bzw. Anweisungen

LET

Definiert den Anfang einer Ergibtanweisung, d. h. einer Wertzuweisung. 'LET' muss nicht vorangestellt werden, es dient lediglich der besseren Übersichtlichkeit. Die im folgenden angegebenen Beispiele stellen nicht in jedem Fall Programme dar, sondern können auch nur Varianten eines Anweisungstyps verdeutlichen.

```
>10 LET A=1
>20 A=50;B=30
>30 LET C=A-B+3
>40 LET X=3+A+(B-3)/C
>50 LET @(3)=24
```

IF

Mit der 'IF'-Anweisung werden Bedingungen für die Ausführung einer der Anweisung folgenden Anweisung festgelegt. Im Zusammenhang mit der GOTO-Anweisung lassen sich damit Programmverzweigungen realisieren.

```
>100 IF B=10 GOTO 200
>108 IF C=0 PRINT 'FERTIG'
>115 IF A+B<100 A=A+1;GOTO 100
```

Die 'IF'-Anweisung selbst darf keine Folgeanweisung in einer Programmzeile sein, muss also immer am Zeilenanfang stehen. Die 'IF'-Anweisung ist damit einer Vergleichsoperation gleichzusetzen.
Nachfolgend sind alle erlaubten Vergleichsoperatoren aufgeführt:

```
>=   größer gleich
#    ungleich
>    größer
=    gleich
<    kleiner
<=   kleiner gleich
```

Ist die in der IF-Anweisung angegebene Vergleichsoperation wahr, wird die in der gleichen Programmzeile folgende Anweisung ausgeführt, sonst die nachfolgende Programmzeile abgearbeitet.

GOTO

Unbedingter Sprung zu einer Zeile, deren Zeilennummer direkt angegeben wird, oder sich aus dem angegebenen Ausdruck berechnet. 'GOTO zlnr' als Direktanweisung (ohne vorangestellte Programmzeilennummer) startet das Programm ab der angegebenen Zeilennummer. In Verbindung mit 'IF' kann 'GOTO' zur Konstruktion von bedingten Sprunganweisungen verwendet werden (siehe auch Beispiele der 'IF'-Anweisung).

```
>100 GOTO 120
>GOTO 120
>110 GOTO 120+B
>120 GOTO A
>120 IF A>0 GOTO 100
```

FOR...TO...STEP...NEXT

Damit lassen sich leicht Programmschleifen aufbauen, die mit einer freibestimmbaren Anzahl von Schleifendurchläufen abgearbeitet werden sollen. Jede mit 'FOR' eröffnete Schleife muss mit einer NEXT-Anweisung, die die gleiche Zählvariable wie die FOR-Anweisung beinhaltet, abgeschlossen werden.

```
>100 N=10
>110 FOR I=0 to N
>120 LET A=I*10;B=I*I
>121 PRINT A,B, A*B
>150 NEXT I
>160 ...
```

Der Programmabschnitt von Zeile 110 bis Zeile 150 wird N-mal durchlaufen, wobei 'I' mit dem Wert '0' beginnend in jedem Durchlauf um '1' erhöht wird, bis der Wert N überschritten wurde. Danach wird die Schleife verlassen.

Eine Schrittweite wird mit dem Schlüsselwort STEP gekennzeichnet. Wird STEP weggelassen, wird der Standardwert 1 genommen.

In der 'FOR..NEXT'-Anweisung können die Anfangs- und Endwerte sowie die Schrittweite der Schleifendurchläufe für die Zählvariable (im Beispiel das 'I') beliebige arithmetische Konstruktionen bzw. Ausdrücke sein. Bei jeden Schleifendurchlauf wird die Zählvariable um den Wert der Schrittweite verändert, bis der Endwert überschritten wird. Bei negativer Schrittweite ist auf die richtige Angabe der Anfangs- und Endwerte zu achten.

```
>110 FOR X=A TO N+B STEP C
>120 ...
...
>150 NEXT X
```

Eine 'FOR-NEXT'-Schleife kann zu jedem beliebigen Zeitpunkt durch eine 'IF..GOTO'-Anweisung verlassen werden. Es darf aber nicht in eine 'FOR...NEXT'-Schleife von außerhalb der Schleife hineingesprungen werden.

GOSUB...RETURN

Mit GOSUB erfolgt der Aufruf eines in BASIC geschriebenen Unterprogramms, welches mit 'RETURN' beendet werden muss. Nach dem Befehl 'RETURN' wird mit dem, dem Unterprogrammruf folgenden Befehl im BASIC-Programm fortgesetzt. Unterprogramme sind dort sinnvoll, wo gleiche Programmteile an mehreren Stellen benötigt werden. Dadurch wird Speicherplatz eingespart. (ähnliche Problematik der MC-Programmierung, siehe Abschnitt 4.) Innerhalb eines Unterprogrammes sind die Variablen des rufenden Programmes ebenfalls gültig und werden zur Parameter- und Ergebnis-übermittlung genutzt.

```
>120 C=25;GOSUB 180;PRINT 'ZEIT',
>125 C=60;GOSUB 180;PRINT 'SCHLEIFE'
...
>170 STOP
>180 REM ZEITPROGRAMM
>190 IF C#0 C=C-1; GOTO 190
>210 RETURN
```

REM

Dadurch werden in einen BASIC-Programm; Kommentarzeilen gekennzeichnet. Sie dienen der besseren Übersichtlichkeit der Programme und werden bei der Abarbeitung durch den Interpreter

überlesen. Die Programmzeile belegt aber entsprechend ihrer Länge Speicherplatz in RAM-Bereich des Rechners.

```
>110 REM LET C=1024
>180 REM ZEITPROGRAMM
```

Diese Zeilen werden nicht mit abgearbeitet.

INPUT

Dadurch wird die Eingabe von numerischen Werten mittels der Tastatur ermöglicht. Ein eingegebener Wert wird dabei einer Variablen zugeordnet. Alle eingegebenen Zeichen werden wieder auf dem Bildschirm ausgegeben. Korrekturen der Eingabe sind mit der Taste 'Cursor links' bzw. 'Cursor rechts' möglich. Die gesamte Eingabe ist mit der ENTER-Taste abzuschließen (nach Korrekturen muss der Cursor auf die erste freie Position in der Eingabezeile gestellt werden!).

Nach der 'INPUT'-Anweisung kann ein in Hochkomma eingeschlossener Text angegeben werden, welcher bei der Ausführung der Anweisung auf dem Bildschirm mit ausgegeben wird. Mittels einer 'INPUT'-Anweisung können mehrere Eingaben ausgeführt werden. Anstelle einer Zahl kann auch ein Ausdruck eingegeben werden.

```
>10 INPUT X
>20 INPUT 'SPRUNGWEITE' S
>30 INPUT 'WERTEPAAR A'A,B
>RUN
X: eingabe
SPRUNGWEITE: eingabe
WERTEPAAR A: eingabe B: eingabe
```

Das Wort 'eingabe' erscheint nicht auf dem Bildschirm, es wurde hier nur verwendet, um deutlich zu machen, dass an dieser Stelle eine Eingabe mittels der Tastatur erfolgt.

```
>20 INPUT ' 'S
>RUN
: eingabe
```

PRINT

Damit wird die Ausgabe von numerischen Werten und von Textketten ermöglicht. Texte sind dabei in Hochkomma einzuschließen. Mehrere Ausgabeparameter innerhalb einer 'PRINT'-Anweisung sind mit Komma voneinander zu trennen. Zahlen werden bei fehlender Formatangabe sechsstellig mit unterdrückten Vornullen

rechtsbündig ausgegeben (d. h.: eine einstellige Ziffer beansprucht sechs Zeichenpositionen auf dem Bildschirm, wobei die ersten 5 leer bleiben und die auszugebende Ziffer die sechste Position einnimmt).

Durch ein Doppelkreuz, gefolgt von einer Zahl 1...6), kann diese Formatisierung geändert werden. Die Zahl gibt die maximal auszugebende Stellenzahl an. Die Formatisierung bleibt bis zur nächsten 'PRINT'-Ausgabe bestehen. Wird die 'PRINT' Anweisung mit einem Komma beendet, so beginnt die Ausgabe der nächsten 'PRINT'-Anweisung in der gleichen Zeile nach der vorangegangenen Ausgabe.

```
>10 X=5;Y=50;Z=500
>20 PRINT 'ZAHL X=',X
>30 PRINT 'ZAHL X=',#2,X
>40 PRINT X,Y,
>50 PRINT Z
>RUN
ZAHL X=      5
ZAHL X= 5
     5    50   500
```

STOP

Diese Anweisung beendet die Abarbeitung des BASIC-Programmes. Der Interpreter kehrt in die Eingabeschleife zurück. Eine 'STOP'-Anweisung muss nicht unbedingt als letzte Anweisung eines BASIC-Programmes stehen. Gegebenenfalls kann diese Anweisung bei der Abarbeitung durch bedingte Sprünge ('IF...GOTO') oder Unterprogrammrufe übersprungen werden, ehe sie dann im weiteren Programmlauf erreicht wird.

CALL

Mit 'CALL' und einer nachfolgenden, als Hexadezimalausdruck gekennzeichneten MC-Adresse wird ein in Maschinensprache geschriebenes Unterprogramm vom BASIC-Interpreter aus gestartet. Soll nach Abarbeitung des MC-Unterprogrammes die Arbeit des Interpreters mit der folgenden BASIC-Anweisung fortgesetzt werden, so ist das MC-Unterprogramm mit einen 'RETURN'-Befehl (bedingt oder unbedingt, z. b. 0C9H, siehe auch Abschnitt 4, Unterprogrammtechnik) abzuschließen. Zur Übermittlung der Parameter werden die PEEK- und POKE-Befehle verwendet.

Beispiel:

```
>200 CALL (HEX (3000))
...
```

```
MC-Unterprogramme:
3000      3A FF 31       LD A,(31FFH)
3003      3C             INC A
3004      27             DAA
3005      32 FF 31       LD (31FFH),A
3008      C9
```

Das durch die CALL-Anweisung in Zeile 200 aufgerufene MC-Unterprogramm (von Adresse 3000H bis 3009H) zählt den Inhalt des Speicherplatzes 31FFH dezimal um '1' weiter. Auf diesem Speicherplatz könnte dann mit einer PEEK-Anweisung zugegriffen werden.

Mit diesem Befehl können also auch die Routinen des Monitors aufgerufen werden. Das wird z. B. im Anwendungsprogramm 'Telefonverzeichnis' beim Retten von Dateien praktiziert.

PEEK

Die PEEK-Anweisung ermöglicht den direkten Speicherzugriff zum RAM bzw. ROM des Rechners. Die Anweisung enthält eine Adresse als Parameter. Vom Speicherplatz, der durch die Adresse ausgewählt wurde, wird ein Byte als Wert einer Variablen dezimal zugewiesen.

```
>10 X=PEEK (1023)
Speicherplatz 1023 wird adressiert (dezimale Adresse)
>20 X=PEEK (HEX (3FF))
Speicherplatz 1023 wird adressiert (hexadezimale Adresse)
```

POKE

Mit Hilfe von POKE kann ein Speicherplatz beschrieben werden (wobei der Speicherplatz im RAM-Bereich liegen muss). Der erste Parameter bestimmt die Adresse des Speicherplatzes, der zweite gibt den Datenwert an, der abgespeichert werden soll.

```
>10 POKE HEX (ED08),65
```

Der Wert 65 (dezimal) wird in den Speicherplatz mit der Adresse ED08H (BWS) als 41H abgespeichert. 41H entspricht dem ASCII-Kode für den Großbuchstaben A.

```
>10 FOR I=0 TO HEX (3FF)
>20 POKE HEX (3000) + I,PEEK (HEX (F000) + I) ;NEXT I
ODER
>10 A=HEX (3000); B=HEX (F000)
>20 FOR I=0 TO 1023; POKE A + I, PEEK (B + I);NEXT I
```

Es wird der Monitor ab der Adresse F000 in der Länge von 1K Byte nach Adresse 3000H umgespeichert.

Dieses Programm entspricht dem Monitorkommando: T F000 3000 3FF. Man beachte die unterschiedlichen Ausführungszeiten.

BYTE

Damit wird der Wert des nachfolgenden Ausdrucks als Hexadezimalausdruck auf dem Bildschirm ausgegeben. Es erfolgt nur die Ausgabe von dezimalen Werten bis 255 als zweistellige Hexadezimalzahl.

```
>BYTE (16)
     10
```

WORD

Diese Anweisung wirkt ähnlich wie Byte. Es werden aber hier 4 Stellen hexadezimal ausgegeben.

```
>10 N=1023
>20 WORD (N)
>RUN
  03FF
```

HEX

Mittels der HEX-Anweisung wird eine angegebene Hexadezimalzahl in eine Dezimalzahl umgewandelt.

```
>10 X=HEX (1000)
>20 PRINT X
>RUN
  4096
```

' ' (QUOTE)

Der durch ' '(Hochkomma) dargestellte Befehl QUOTE realisiert die Einzelzeichenumwandlung eines ASCII-Zeichens in einen Dezimalwert. Das ASCII-Zeichen ist zwischen dem Hochkomma anzugeben.

```
>10 X='B'
>20 PRINT X
>RUN
    66
```

OUTCHAR

Der der OUTCHAR-Anweisung folgende dezimale Ausdruck wird als entsprechendes ASCII-Zeichen auf dem Bildschirm ausgegeben. Bestimmte Sonderzeichen werden sofort ausgeführt (siehe OUTCH-Funktion des Monitors).

```
>10 OUTCHAR 65
>20 X=66
>30 OUTCHAR X
>40 X='C'
>50 OUTCHAR X
>60 PRINT X
>RUN
ABC
    67
```

OUTCHAR 12 löscht z. B. den gesamten Bildschirm.

INCHAR

Die INCHAR-Anweisung ermöglicht die Eingabe eines einzelnen Zeichens mittels der Tastatur. Bei der Eingabe dieses Zeichens wird es nicht auf dem Bildschirm ausgegeben. Die ENTER-Taste muss nach dar Zeicheneingabe nicht betätigt werden. Der Wert des ASCII-Zeichens wird der in der Anweisung mit angegebenen Variablen zugewiesen.

```
>10 PRINT INCHAR; GOTO 10
oder
>10 X=INCHAR; PRINT X; GOTO 10
```

Mit dieser Programmzeile kann die gesamte Tastatur getestet werden. Es werden alle Tasten mit allen SHIFT-Tasten kombiniert betätigt und dadurch der dezimale Zahlenwert auf dem Bildschirm angezeigt. Ein Verlassen dieser Programmschleife ist nur mit Ctrl-C (STOP) möglich.

OUT

Der in der OUT-Anweisung angegebene Wert des Ausdruckes wird an die in der Anweisung zugewiesene E/A-Adresse des MRB Z1013 ausgegeben. Der Wert darf 255 nicht überschreiten (255 ist bekanntlich die größte Dezimalzahl, die mit 8 Bit darstellbar ist.).

```
>10 OUT (0)=10
```

Das Bitmuster für 10 (00001010B) wird an die E/A-Adresse 0 ausgegeben. In der Grundvariante des MRB Z1013 ist 0H die mögliche E/A-Adresse des PIO-Ports A, die die freie Verwendung durch den Anwender ermöglicht.

Bei Verwendung der PIO als Port darf die entsprechende Initialisierung nicht vergessen werden, z. B.:

```
>10 OUT (1)=HEX (CF)
>20 OUT (1)=0 (Ausgabe) bzw.
>20 OUT (1)=255 (Eingabe)
```

IN

Diese Anweisung ermöglicht die Eingabe von Werten von einer E/A-Adresse des MRB. (Adresse der Grundvariante ist 0H.) Der Wert, der an der E/A-Adresse anliegt, wird einer Variablen zugeordnet.

```
>10 X=IN (0)
```

I$

```
>10 I$ (TOP)
```

Eingabe einer Zeichenkette über Tastatur auf den ersten freien Speicherplatz nach einem BASIC-Programm.

O$

```
>20 O$ (TOP)
```

Ausgabe einer Zeichenkette, die ab dem ersten freien Speicherplatz nach dem BASIC-Programm gespeichert ist, auf dem Bildschirm.

LEN

Stellt die Länge der zuletzt mit einer I$-Anweisung eingegebenen Zeichenkette zur Verfügung.

```
>10 I$ (TOP)
>20 FOR I=0 TO LEN
>30 IF PEEK (TOP + I) = 'A' PRINT (TOP + I)
>40 NEXT I
```

Folgende Funktionen werden außerdem durch den BASIC-Interpreter realisiert:

RND

Die RND-Funktion weist einer Variablen einen zufälligen Wert zwischen 1 und dem in der Anweisung festgelegten Endwert zu.

```
>10 X=RND (2000)
>20 PRINT X
>RUN
  1576
```

ABS

Es wird der Absolutbetrag des folgenden Ausdrucks gebildet und einer Variablen zugewiesen.

```
>10 A=-120
>20 A=ABS (A)
>30 PRINT A
>RUN
   120
```

TAB

Die TAB-Funktion stellt eine sogenannte Tabulatoranweisung dar. Die sich aus dem nachfolgenden Ausdruck ergebende Anzahl von Leerzeichen wird auf dem Bildschirm ausgegeben. Die nachfolgende Ausgabe von Zeichen beginnt nach diesen Leerzeichen.

```
>10 PRINT 'ANWEISUNG:',
>20 X=5
>30 TAB (X)
>40 PRINT 'TAB',
>50 TAB (6)
>60 PRINT '(',#1,X,')'

ANWEISUNG:     TAB      (5)
```

Durch die TAB-Funktion wird die Darstellung von Kurven möglich.

```
>10 FOR I=-5 TO 5; TAB (I*I); PRINT '*'; NEXT I
```

TOP

Die TOP-Funktion ermittelt den zum Zeitpunkt der TOP-Funktion aktuellen ersten freien Speicherplatz hinter dem soeben eingegebenen BASIC-Programm (dezimal).

```
>PRINT TOP
  3561
```

SIZE

Damit wird der aktuelle freie RAM-Speicherbereich ermittelt, der für ein BASIC-Programm noch zur Verfügung steht.

```
>10 PRINT SIZE, 'FREIE BYTE'
>RUN
  260 FREIE BYTE
```

Die ermittelte Anzahl freier Speicherplätze wird SIZE zugewiesen und kann im BASIC-Programm weiterverwendet werden.

5.3.7. Dateiformat

Mit CSAVE werden die BASIC-Programme auf Diskette oder USB gespeichert, mit CLOAD geladen. Das entsprechende Treiberprogramm muss dazu vorab geladen werden (USB-OS, Disk-OS, OS-Erweiterung f. Kassette). Fehlen die Treiberfunktionen, gibt es eine Fehlermeldung „bos error: os".

Das Basic-Programm wird in einem komprimierten Format gespeichert: 2 Byte Zeilennummer, Zeilentext, 1 Byte Zeilenende 0Dh. Als Textende folgt zuletzt ein Byte 1Ah.

Auf USB bzw. Diskette wird kein Kopfblock geschrieben.

Anlage 14: Basic-Beispielprogramme

Die hier angegebenen Programme in der Programmiersprache Basic erfordern 16k RAM und das Vorhandensein des "3k-Basic von rer". eine Erläuterung der Programme erfolgte nur dort, wo es für erforderlich gehalten wurde. ansonsten reicht die enthaltene Bedienerführung für das Verständnis der Programme aus.

1. Basic-Programm zum Speichern von Adressen und Telefonnummern

mit diesem Basic-Programm ist es möglich, ein Adressverzeichnis aufzubauen, anzuzeigen und zu korrigieren. Man kann aber auch nach einem bestimmten Namen suchen, alle Adressen nach den Anfangsbuchstaben sortieren und ein Verzeichnis auf dem Bildschirm ausgeben lassen, in dem nur Telefonnummern und Name erscheinen.

Wenn vor Programmstart die RAM-Grenze mit der Anweisung

END HEX(3FFF)-64

in Basic erweitert wird, sind maximal 80 Adressen speicherbar. bei höheren Forderungen müssen die Daten auf Magnetbandkassette abgespeichert werden oder es ist ein anderes Satzformat zu wählen (im Programm ist die Satzlänge auf 100 Bytes festgelegt), der Dateianfang wurde in Zeile 40 mit E=8600 festgelegt. Vor Programmstart mit RUN ist mit PRINT TOP,SIZE zu kontrollieren, dass der Wert für top kleiner als 8600 ist, ansonsten müsste der Anfangswert für E erhöht werden.

2. Basic-Programm "Begriffe raten"

Mit diesem BASIC-Programm können Begriffe, Sprichwörter oder Sätze abgespeichert werden. Die maximale Satzlänge sollte ein Vielfaches von 32 sein. Die maximale Anzahl der Sätze sollte nicht über 120 liegen. Entsprechend dem Menü kann der Spielmeister die Begriffe aufbauen, zur Kontrolle anzeigen, korrigieren, testen und auf Magnetbandkassette abspeichern.

Die Spielidee besteht darin, dass in einer Spielrunde vom Spielmeister Begriff für Begriff abgerufen werden kann. Dabei wird jeder Buchstabe des Begriffes durch einen Strich angezeigt.

Die Mitspieler haben nun die Möglichkeit, den Begriff zu erraten bzw. durch Eingabe eines Buchstabens diesen im Begriff an allen vorkommenden Stellen aufzublenden. Damit wird der Begriff immer vollständiger.

Wurde der Begriff von einem Mitspieler richtig erraten, so kann dieser durch Betätigen der ENTER-Taste zur Anzeige gebracht werden.

3. Basic-Programm zum Raten einer durch die RND-Funktion ermittelten Zahl

In dem folgenden BASIC-Programm wird die Anwendung der Fensterfunktion des Monitors gezeigt. In dem Unterprogramm ab Zeile 600 werden auf ARG1(1BH) und ARG2(1DH) die Bildschirmadressen (EC00H bis F000H) mit der POKE-Anweisung gebracht (z.B.:volles Fenster). Mit der CALL-Anweisung auf Zeile 630 direkt in das Monitorprogramm (Adresse F6D1H) werden die neuen Adressen des Rollfensters übernommen. Ein anderes Rollfenster wird ab Zeile 350 eingestellt (ED40H bis F000H). Zu beachten ist, dass man vor Verlassen des Programms wieder auf volles Rollfenster stellen muss (Zeile 500).

Bei diesem Zahlenratespiel besteht die Möglichkeit, den Zahlenbereich, in welchem die Zufallszahl ermittelt werden soll, durch die Eingabe in Zeile 400 festzulegen. Mit dem Test in Zeile 405 werden ab 1 alle positiven Zahlen bis 32767 akzeptiert.

Mit der Variablen V wird die Anzahl der Versuche mitgezählt. In den Zeilen 570 und 575 wird eine Warteschleife abgearbeitet, damit die vorherige Ausschrift "AUF WIEDERSEH'N" gelesen werden kann.

4. Mathe-Übungsprogramm in Basic

Rechenaufgaben Grundrechenarten in 4 Schwierigkeitsstufen.

5. Basic-Programm zur Darstellung von Größen im Balkendiagramm

Es werden 3 informative Textzeilen abgefragt (Eingaben ohne weitere Verwendung), danach werden 5 Werte erfasst, die als Balkendiagramm gezeichnet werden.

6. Basic-Programm "Turm von Hanoi"

Die Scheiben müssen von Turm 1 nach 3 umgestapelt werden. Dabei darf immer nur eine Scheibe bewegt werden, und es darf immer nur eine kleinere auf einer größeren Scheibe oder auf einem leeren Stapel abgelegt werden.

7. Basic-Programm "Mastermind"

8. Basic-Programm "23 Streichhölzer"

Auf dem Tisch liegen 23 Streichhölzer. Der Spieler nimmt stets 1,2 oder 3 Hölzer weg. Darauf nimmt der Computer 1,2 ODER 3.
Wer das letzte Streichholz nimmt, hat verloren.