Hobby Computer, Sonderheft der ELO,Funkschau,Elektronik, S.72-79, 1978(?)

Rolf-Dieter Klein

RDK: *27 Apr 1999: mein Tiny Basic ist ein Derivat von einer Quelle die noch aelter ist. Sie koennen auch gerne meine Quelle ins Netz stellen, leider habe ich die Sourcen nicht mehr im Zugang. gruss rdk*

BASIC für 8080-Systeme

Der Neuling auf dem Mikrocomputergebiet kommt am schnellsten zu einem lauffähigen Programm, wenn er eine höhere Programmiersprache zur Verfügung hat. Mit dem beschriebenen Interpreter haben Besitzer von 8080- und Z-80-Systemen die Möglichkeit, in BASIC, einer der leichtesten Programmiersprachen überhaupt, zu arbeiten. Der Interpreter ist ein Übersetzungsprogramm, das aus den BASIC-Befehlen den Maschinencode des Mikroprozessors erzeugt. Er braucht etwa 3 KByte an Speicherplatz und kann zur Not noch von Hand eingetippt werden. Zusätzlich muß noch ungefähr 1 KByte RAM für das Benutzerprogramm zur Verfügung stehen. Weitere Voraussetzungen sind: alphanumerische Eingabe- und Ausgabemöglichkeit.

1 Arbeitsweise des BASIC-Interpreters

Grundsatzlich gibt es bei der Realisierung einer höheren Programmiersprache auf einem Mikrocomputer zwei verschiedene Möglichkeiten: Da wären zunächst einmal die sogenannten Compiler. Sie übersetzen ein Programm direkt in Maschinensprache und fuhren es dann aus. Das heißt, für einen BASIC-Befehl werden zunächst mehrere Befehle aus der Assemblersprache des betreffenden Prozessors erzeugt, und es entsteht aus dem ursprunglichen BASIC-Programm ein Programm in Assembler. Dieses wird dann in die entsprechenden Operationscodes übersetzt, und es entsteht ein für den Prozessor verstandliches Programm, das in den Arbeitsspeicher geladen und gestartet werden kann. Nun erst läuft das Programm, und eventuelle Fehler werden erkennbar, wenn sie nicht schon vom Übersetzer erkannt wurden.

Ein Interpreter arbeitet etwas anders. Der vorliegende holt sich z. B. eine Zeile aus dem Programm, analysiert sie und führt die entsprechenden Befehle aus. Nach Ausführung der Zeile holt er die nächste und so fort. Ist eine Schleife vorhanden, das heißt, wird ein Programmteil öfters wiederholt, so muß der Interpreter diesen Teil auch genauso oft übersetzen. Hier liegt auch schon der Nachteil eines Interpreters gegenüber einem Compiler: Compiler-Programme sind in der Regel erheblich schneller. Es gibt noch eine Mischform, ebenfalls Interpreter, die das Quellprogramm in einen Zwischencode übersetzen und dann erst ausführen. Dabei werden Befehle wie PRINT einfach durch einen Code, z. B. 85, ersetzt. Der Vorteil dieser Interpreter liegt darin, daß Programme wenigerSpeicherplatz benötigen und daß die Ausfuhrungszeit etwas besser liegt. Der beschriebene Interpreter arbeitet aber nicht so, sondern übersetzt jeden Befehl neu. Er besitzt dazu eine Tabelle, in der sämtliche vorhandenen Befehle stehen. Hinter jedem Befehl steht dann noch eine Adresse. Sie gibt an, an welcher Stelle im Programm sich das entsprechende Unterprogramm für die Abarbeitung des jeweiligen Befehls befindet.

2 Laden, Starten und Modifizieren

2.1 Laden des Interpreters

Um den Interpreter in das System eingeben zu können, sind zwei Möglichkeiten der Eingabe vorhanden Zunächst ist eine sedezimale Liste (Bild 1) vorhanden. Der Interpreter startet auf der Adresse 1000. Die Codes können nun einfach entsprechend der Liste in den eigenen Computer eingegeben werden. Am Schluß sollte dabei jedes Byte mit dem aufgelisteten Byte verglichen werden, um Tippfehler, die das Programm zerstören konnten, zu vermeiden. Ferner sollte das Programm dann auf ein externes Speichermedium gebracht werden (z. B. Lochstreifen, Kassette o. ä.).

Es gibt aber noch eine zweite Möglichkeit. Dafür ist ein Listing im sogenannten „relocating“-Format beigefügt (Bild 2), das es gestattet, den Interpreter auf eine beliebige Stelle im Speicher (allerdings < 8000sed) zu binden. Dies ist möglich, da in diesem besonderen Format die Information enthalten ist, an welchen Stellen Adressen im Programm vorhanden sind.

Um von diesem Format Gebrauch machen zu können, ist es aber nötig, einen sogenannten Lader zu schreiben, der diese Berechnung (addieren einer Konstanten auf alle Adressen) durchfuhrt.

Dazu das Aufzeichnungsformat:

Zeichen 0...1: CR (Wagenrücklauf), LF (Zeilenvorschub), um Blöcke voneinander zu trennen.

Zeichen 2: Strichpunkt, kennzeichnet den Anfang eines relokalisierbaren Blockes.

Zeichen 3...4: Dieses Byte (durch zwei ASCII-Zeichen, ISO-7-bit-Code nach DIN 66003, 0…9, A…F dargestellt) gibt die Anzahl der Daten-Bytes an. Z. B. 19 an dieser Stelle bedeutet, es sind 25 Daten-Bytes in diesem Block.

Zeichen 5...8: In den zwei Bytes steckt die relative Anfangsadresse des Blocks. Sie wird beginnend mit dem höherwertigen Byte der Adresse angegeben.

Zeichen 9...10: Dieses Byte enthalt die Relokalisierinformation. Ist dieses Byte 0, so wird die angegebene Anfangsadresse absolut verwendet. Ist dieses Byte 1, so wird die Adresse durch den Relokalisierfaktor modifiziert.

Zeichen 11…12: Dieses Byte enthält die Relokalisierinformation für die nächsten 8 Daten-Bytes. Dabei entspricht jedes Bit dieses Bytes einem Byte der nächsten 8 Daten-Bytes. Bit 7 ist für das erste Byte zuständig und Bit 0 für das letzte. Ist das betreffende Bit 0, dann heißt dies, das Byte wird unverändert geladen. Ist ein Byte auf 1 gefolgt von einem auf 0, dann sind die beiden dazugehörigen Bytes eine relokalisierbare Adresse, und der Relokalisierfaktor muß aufaddiert werden. Ein Bit auf 1 gesetzt, von einem gefolgt, das ebenfalls auf 1 ist, stellt eine Information für den Linking Loader dar, der aber hier nicht benötigt wird.

Zeichen 13…28: Hierin sind die eigentlichen Daten-Bytes enthalten. Die Gesamtzahl der Daten-Bytes ist durch die Information am Anfang bestimmt. Nach jedem achten wird aber wieder die Relokalisierinformation eingefügt, dann folgen wieder Daten-Bytes.

Zeichen N...(N+1): Dieses Byte enthält die Blockprüfsumme, die dem Zweierkomplement der Summe über alle vorhergehenden Bytes in diesem Block entspricht, so daß nun die Summe aller Bytes 0 ergeben muß. Damit wird eine Blocksicherung erreicht.

Vom Benutzer muß ein Lader geschrieben werden, der speziell auf sein System abgestimmt ist und alle diese Informationen auswertet. Der Vorteil des relokalisierbaren Programms liegt neben der frei wählbaren Anfangsadresse bei der Überprüfbarkeit auf Eingabefehler durch die Prüfsumme.

2.2 Starten und Modifizieren des Interpreters

Um den Interpreter starten zu können, ist es nötig, zunächst die Ein/Ausgabe-Vektoren zu ändern. Standardmäßig sind zwei Vektoren vorhanden, der eine führt auf die Adresse F003 und bedeutet einen Sprung auf die Eingabe eines Zeichens vom Benutzer. Das Zeichen muß dabei über das Register A an den Interpreter weitergegeben werden. Die anderen Register dürfen nicht verändert werden. Ferner gibt es noch einen Sprung auf die Adresse F009, der die Ausgabe eines Zeichens bewirken soll. Dabei ist das auszugebende Zeichen im Register C vorhanden. Am Schluß dieser Routine muß das Zeichen im Register A stehen und C soll unverändert bleiben, wie auch die restlichen Register.

Hat der Benutzer an diesen beiden Adressen keinen Speicher, um entsprechende Sprungbefehle auf seine eigenen E/A-Routinen durchzuführen, so muß er im BASIC-Interpreter die Vektoren ändern.

Dazu die Adressen: Auf Adresse 71DH (H bedeute; HEX also sedezimal) steht die Befehlsfolge CD 03 F0, sie ist in die entsprechende Folge zum Aufruf des eigenen Eingabe-Unterprogramms umzuwandeln. Auf Adresse 70DH und 715H steht die Befehlsfolge CD 09 F0, hier ist der entsprechende Aufruf für die Ausgaberoutine einzutragen.

Nun gibt es noch die Möglichkeit, das Anwenderprogramm durch die Betätigung der Tastenfolge CTRL C zu stoppen, genauso wie den Ausdruck langer Programme. Dazu hat der Benutzer eine Routine zu schreiben, die seinen Status-Port der Eingabetastatur abfragt, ob ein Zeichen vorhanden ist und wenn, ob es sich um CTRL C (Code 3) handelt. Ist das der Fall, muß das Programm auf die Adresse 00C6H springen (RESTART-Adresse des Interpreters), anderenfalls muß es mit einem Return-Befehl zurückkehren. Diese Routine wird von zwei Stellen des BASIC-Interpreters aus angesprungen. Der Aufruf muß an die betreffenden Stellen eingetragen werden.

Dazu die Adresse: Auf AlCH steht der Befehl C9 (Return), dieser muß durch einen JMP-Befehl auf die eigene Routine ersetzt werden. Falls der Platz ausreicht, kann die Routine auch an diese Stelle geschrieben werden. Es stehen dazu 10 Byte zur Verfügung. Die Adresse AlCH wird von dem Teil, der die Programme ausführt, und von dem Teil, der Programme auflistet, bei Ausführung jedesmal mit dem Beginn einer neuen Zeile angesprungen.

Im beschriebenen BASIC gibt es einen Befehl BYE, der das System verläßt und dem Monitor die Steuerung übergibt. Dies geschieht durch die Ausführung des Maschinenbefehls RST 7. Der Benutzer kann einen direkten Sprung zu seinem Monitor einbauen, indem er an die Stelle 8DFH anstelle des RST 7-Befehls einen Sprung zu seinem Monitor schreibt. Das BASIC-Programm kann dann vom Monitor aus gestartet werden, indem ein Sprung auf den Speicherplatz C6 ausgeführt wird.

Alle angegebenen Adressen sind um den Relokalisierfaktor zu erhöhen, den der Anwender beim Laden des Interpreters verwendet hat.

Bild 1. Programmliste

Bild 2. Programmliste für verschiebbares (relocating) Format

3 BASIC-Befehle

3.1 Steuerbefehle

Diese Befehlsgruppe dient der Steuerung des Interpreters. Es kann damit der Ablauf eines Programms bestimmt werden (starten, löschen, ausdrucken).

LIST

Mit Hilfe dieses Befehls kann ein zuvor eingegebenes Programm ausgedruckt werden. Dabei wird, wenn der Befehl LIST CR (CR steht für carriage return, also Wagenrücklauf) eingegeben wird, das gesamte Programm ausgedruckt. Es kann auch eine zusätzliche Zeilennummer angegeben werden, dann wird das Programm beginnend mit dieser Zeilennummer ausgegeben. Soll der Ausdruck eines Programms abgebrochen werden, weil auf dem Datensichtgerät nicht das ganze Programm dargestellt werden kann, so kann dies durch Eingeben des Zeichens CTRL C (Code 3) geschehen. LIST 200 zum Beispiel gibt das Programm beginnend mit der Zeilennummer 200 aus. Falls die Zeilennummer 200 nicht vorhanden ist, wird nach der nächstgrößeren gesucht und von da an gelistet.

RUN

Nach Eingabe von RUN CR wird das Programm beginnend bei der niedrigsten Zeilennummer gestartet.

NEW

Damit kann ein Programm gelöscht werden. Anschließend ist es möglich, ein neues Programm einzugeben.

BYE

Bei der Ausführung dieses Befehls kehrt der Prozessor zum Monitorprogramm zurück und verläßt das BASIC-System.

END

In der ursprünglichen Bedeutung steht dieser Befehl am Ende eines Programms. Dies ist bei dem vorliegenden Interpreter nicht notwendig, somit konnte dieser Befehl mit einer anderen Bedeutung versehen werden. Von Haus aus wird nach Laden des BASIC-Interpreters ein minimaler Speicherbedarf von 700 Byte für den Anwender definiert. Bei der Überschreitung dieses Arbeitsraumes durch Eingabe langer BASIC-Programme wird eine Fehlermeldung vom Interpreter (SORRY) ausgegeben, die darauf hinweisen soll, daß kein Platz mehr für das Programm vorhanden ist. Nun kann es aber sein, daß ein größerer Speicher vorhanden ist. In diesem Fall kann der Benutzer mit Hilfe des Befehls END den zunächst auf 700 Byte vorbesetzten Platz dynamisch erhöhen. Dazu erhält der Befehl END einen zusätzlichen Parameter, der die absolute Adresse der gewünschten neuen höchsten Adresse darstellt.

Zum Beispiel bedeutet die Anweisung END 8000, daß dem BASIC ab sofort Speicherplatz bis zur Adresse 8000 (dezimal) zur Verfügung steht.

Dabei muß berücksichtigt werden, daß der Interpreter darüber hinaus einen Platz für den Textpuffer benötigt, der über dem angegebenen Platz vorhanden sein muß. Er kann mit etwa 140 Byte

veranschlagt werden.

3.2 Programmierbare Befehle

Alle nun folgenden Befehle können im Gegensatz zu den Steuerbefehlen programmiert werden. Die meisten können auch im sogenannten „Direct Mode„ verwendet werden, das heißt, einfach durch Eingabe ohne vorangestellte Zeilennummer. Sie werden dann unmittelbar nach Eingabe von CR ausgeführt.

LET

LET weist einer Variablen einen Wert zu.

Beispiel:

```
 10 LET A=10
 20 LET B=2*(3-9)*6/2
 30 LET A=C
```

Dabei kann der Befehl LET auch weggelassen werden. Er dient eigentlich nur der besseren Lesbarkeit.

FOR TO NEXT

Damit ist es möglich, Schleifen aufzubauen, das heißt, eine bestimmte Befehlsfolge n-mal zu durchlaufen. Beispiel:

```
10 FOR A = 1 TO 10 STEP 2
20 ...
30 ...
40 NEXT A
```

Der Bereich zwischen 10 und 40 wird dabei 5mal durchlaufen. Innerhalb der Schleife kann der aktuelle Wert von A verwendet werden, er sollte jedoch nicht verändert werden. Die Angabe STEP legt die Schrittweite fest Sie kann auch negativ sein, es müssen dann allerdings auch die Variablen nach FOR und TO entsprechend gewählt werden.

Beispiel:

```
10 FOR A = 10 TO 1 STEP -2
20 ...
30 ...
40 NEXT A
```

Dieses Programm bewirkt genau das gleiche wie das erste Beispiel, nur daß hier die Variable zunächst den Wert 10 erhalt, dann 8 dann 6 usw. Wird die Angabe STEP weggelassen, so wird eine Schrittweite von l angenommen.

GOTO

Der Befehl GOTO bewirkt die Ausführung eines Sprungbefehls. Dabei wird hinter dem Befehl die

Zeilennummer angegeben, die angesprungen werden soll. Diese Zellennummer kann auch berechnet werden, indem eine Variable oder ein arithmetischer Ausdruck an diese Stelle geschrieben wird.

Beispiel

```
10 GOTO 209
```

Dieser Befehl bewirkt einen Sprung zur Zeile 209.

Beispiel

```
20 GOTO 100*2+9
```

Hier wird ebenfalls zur Zeile 209 gesprungen.

Es ist auch möglich den Befehl im Direkt-Modus zu verwenden Es wird dann zu der angegebenen Zeile gesprungen und von da an das Programm ausgeführt.

GOSUB

Mit dem GOSUB-Befehl ist es möglich, einen Unterprogramm-Aufruf durchzufuhren Dabei wird ähnlich wie beim GOTO-Befehl die Zellennummer angegeben, die auch hier berechnet werden kann

RETURN

Dieser Befehl stellt das Gegenstuck zum GOSUB-Befehl dar. Nach Aufruf eines Unterprogramms, das mit dem RETURN-Befehl enden muß, kehrt das Programm wieder an die Stelle zurück, von der aus das Unterprogramm aufgerufen wurde.

IF

Mit diesem Befehl kann eine Entscheidung getroffen werden. IF wird von einem arithmetischen Ausdruck gefolgt. Ist der Wert ungleich 0 so wird der nachfolgende Befehl ausgeführt, andernfalls die nächste Zeile.

```
20 IF A = 2 GOTO 10
```

Wenn A den Wert 2 besitzt wird zur Zeile 10 gesprungen. (THEN darf nicht verwendet werden).

REM

Die Anweisung REM ermöglicht es, Kommentare in das Programm einzubauen Dabei wird der Text, der hinter einer REM-Anweisung steht, bis zum Zellenende vom Interpreter ignoriert.

INPUT

Einen der wichtigsten BASIC-Befehle stellt der INPUT-Befehl dar. Er ermöglicht es, Daten in das Programm im Dialogverfahren einzugeben. Will man zum Beispiel in einem Programm der Variablen C einen Wert zuweisen, den der Benutzer eingeben soll, so lautet der Befehl folgendermaßen

```
10 INPUT C
```

Bei der Ausführung des Programms druckt dann der Interpreter

```
C:
```

Nun muß der Benutzer eine Zahl eingeben (oder einen arithmetischen Ausdruck, der dann noch berechnet wird). Will man erreichen, daß ein bestimmter Text anstatt des Vanablennamens ausgedruckt wird, so gibt man diesen Text vor der Variablen in Anführungszeichen gesetzt an.

Beispiel:

```
10 INPUT "Geben Sie eine Zahl ein" C
```

Bei Ausführung dieses Programms wird dann der angegebene Text gefolgt von einem Doppelpunkt ausgedruckt.

Es ist auch möglich, mehrere Variable einzugeben. Dazu werden sie mit Kommas getrennt.

Beispiel

```
10 INPUT A,C,'Zahl'F,K
```

PRINT

Mit Hilfe des PRINT-Befehls ist es möglich, Daten und Texte auszugeben. Dazu werden die verschiedenen Variablen, Zahlen und Texte mit Kommas getrennt angegeben.

Beispiel:

```
10 PRINT 2 B 'Text',7
```

Dieses Programm bewirkt den Ausdruck der Zahl 2, dann des Inhalts der Variablen B, dann wird der Text ausgegeben und die Zahl 7. Zahlen werden mit sechs Stellen ausgegeben. Dies kann aber geändert werden. Dazu dient die sogenannte Formatanweisung. Sie kann auch mehrmals in der PRINT-Anweisung angegeben werden und bleibt bis zur nächsten Formatanweisung innerhalb einer PRINT-Anweisung wirksam. Bei der Ausführung des nächsten PRINT-Befehls in einem Programm ist wieder der Wert 6 voreingestellt. Die Formatanweisung wird mit dem Zeichen # eingeleitet und hat als Parameter eine Zahl oder einen arithmetischen Ausdruck.

Beispiel

```
10 PRINT 1,#10,1,1
```

Die erste 1 wird hier mit insgesamt sechs Stellen ausgedruckt, die anderen beiden mit zehn Stellen.

Wird bei der PRINT-Anweisung an die letzte Stelle ein Komma gesetzt, so wird der Zellenvorschub unterdrückt. Der nächste Ausdruck wird dann an der letzten Position fortgesetzt.

STOP

Der STOP-Befehl beendet den Programmablauf.

CALL

Mit dem Befehl CALL ist es möglich, Unterprogramme in Maschinensprache aufzurufen. Dafür ist ein Parameter anzugeben, der die absolute Adresse des Unterprogramms angibt.

Beispiel

```
10 CALL HEX(54FF)
```

Dieses Programm bewirkt, daß das Maschinenprogramm auf Adresse 54FFH ausgeführt wird. Mit einem RET-Befehl (Code C9) kann wieder in das BASIC-System zurückgekehrt werden.

OUTCHAR

Mit diesem Befehl werden Einzelzeichen ausgegeben, die auch Sonderzeichen und nicht darstellbare Zeichen sein können. Dem Befehl wird als Parameter der dezimale Wert gegeben.

Beispiel:

```
10 OUTCHAR(64)
```

Bei der Ausführung dieses Programms wird das Zeichen 3 gedruckt.

OUT

Mit OUT wird einem 8080-Port direkt ein Wert zugewiesen. OUT wird dabei ähnlich wie eine Variable verwendet. Will man zum Beispiel dem PORT mit der Adresse 18H den Wert 2 zuweisen, so sieht der Befehl wie folgt aus:

```
10 OUT(HEX(18))=2
```

Mit der Funktion HEX wird hier wieder erreicht, daß der sedezimale Wert 18 in einen dezimalen Wert umgerechnet wird und dann dem OUT-Befehl zugeführt werden kann.

O$

Hierbei handelt es sich um einen speziellen Befehl, der eingeführt wurde, um auch schon in diesem kleinen BASIC-System Stringverarbeitung durchführen zu können. Der O$-Befehl ermöglicht die Ausgabe eines Textes, der auf einer beliebigen Adresse stehen kann und von 0 abgeschlossen wird (siehe auch I$, PEEK, POKE). Dazu erhält der Befehl einen zusatzlichen Parameter, nämlich die Adresse.

Beispiel:

```
10 O$ TOP
```

Hier wird ein Text ausgedruckt, der auf der ersten freien Adresse liegt und natürlich zuvor eingegeben werden mußte, z. B. mit I$ oder mit POKE.

I$

Dies ist das Gegenstück zum O$-Befehl. Dieser Befehl erhält eine Adresse als Parameter und legt dann einen Text, der eingegeben wird (ahnlich wie bei INPUT für Zahlen), auf die angegebene Adresse ab. Die Eingabe des Textes wird durch ein CR beendet. Die Länge ist über den LEN-Befehl feststellbar.

Beispiel:

```
10 I$ TOP
```

Dieses Programm legt einen Text, beginnend auf der ersten freien Adresse, ab.

POKE

POKE ist ein Befehl, mit dem Direktspeicherzugriff durchgeführt werden kann, wobei ein automatischer Schreibschutz für ein abgelegtes BASIC-Programm besteht. POKE besitzt zwei Parameter: Der erste gibt die Adresse (absolut) an, der zweite bestimmt den Wert, der auf dieser Adresse abgelegt werden soll.

Beispiel:

```
10 POKE TOP + 1,5
20 POKE 16000,2*5
30 POKE TOP, 'T'
```

Bei Zeile 10 wird der Wert 5 (es wird nur ein Byte gespeichert, falls größere Zahlen als 255 eingegeben werden) auf die zweite freie Speicherzelle gelegt. Bei 20 wird auf die Adresse 16000 (dezimal) der Wert 10 und bei 30 auf die erste freie Adresse der ASCII-Code für den Buchstaben T abgelegt.

TAB

Mit TAB kann die aktuelle Schreibposition verändert werden. Dabei wird im Gegensatz zum Standard-BASIC der TAB-Befehl nicht in eine PRINT-Anweisung geschrieben.

Beispiel:

```
10 TAB(20)
```

Die Schreibposition wird um 20 Zeichen vorgerückt.

RND

Die Funktion RND liefert einen Zufallswert, dabei kann noch angegeben werden, in welchem Bereich dieser Wert liegen soll.

Beispiel:

```
10 A = RND(1000)
```

Die Variable A erhält einen Wert zwischen l und 1000.

ABS

ABS bildet den Absolutbetrag einer Zahl. ABS(-2) entspricht dem Wert 2.

SIZE

Mit SIZE kann der Speicherfreiraum ermittelt werden, der für eigene Programme noch vorhanden ist.

PEEK

Mit PEEK kann ein Direktspeicherzugriff durchgeführt werden. Dazu gibt man die Absolutadresse mit an.

Beispiel:

```
10 A=PEEK(HEX(2000))
```

A erhält den Wert des Bytes, das an der Adresse 2000 (sedezimal) steht. Im Gegensatz zu POKE wird bei PEEK ein Byte geholt.

INCHAR

Mit INCHAR kann ein Zeichen von der Konsole geholt werden Dabei wird dieses Zeichen nicht ausgegeben. Dies ermöglicht es, Zeichen umzudefinieren oder Steuertasten zu definieren.

Beispiel:

```
10 B=INCHAR
```

HEX

Dem Befehl HEX wird in Klammern ein sedezimaler Wert gegeben, der dann in den dezimalen Wert umgerechnet wird.

IN

Mit IN kann der Wert eines 8080-Ports gelesen werden.

Beispiel:

```
10 A=IN(HEX(18))
```

Hier wird der Variablen A der Wert des Ports 18H zugewiesen.

TOP

TOP ist eine Pseudovariable. Mit dieser Funktion erhalt man die Adresse des ersten freien Speicherplatzes (dezimal). Vor diesem Speicher steht das BASIC-Anwenderprogramm.

LEN

LEN ist ebenso eine Pseudovariable Ihr Wert gibt die Länge des zuletzt mit I$ eingegebenen Textes an.

4 Weitere Eigenschaften

4.1 Variable

Als Variable stehen A bis Z zur Verfügung. Als Array (Dimension 1) wird das Zeichen @ verwendet, z. B.

```
10 @(10)=6
```

Die maximale Größe des Arrays hängt dabei von der Länge des Anwenderprogramms ab.

4.2 Arithmetik

Zahlenbereich von -32767 bis + 32767; vier Grundrechenarten **+ * - /**; Klammern können beliebig gesetzt und verschachtelt werden.

4.3 Logische Operatoren

< > ⇐ >= # = liefern den Wert 1, falls die Aussage wahr ist, und 0, falls sie falsch ist. Die Operatoren können beliebig mit * + … verknüpft werden.

4.4 Textoperator

Mit **' '** kann der äquivalente Sedezimalwert eines ASCII-Zeichens berechnet werden. 'A' liefert beispielsweise den Sedezimalcode 41.

4.5 Steuerzeichen

Mit verschiedenen Steuerzeichen kann man Fehler verbessern, die bei der Eingabe entstehen. **CTRL C** (Code 03, d. h die Tasten „CTRL“ und „C“ werden gleichzeitig gedrückt) z. B. unterbricht die Ausführung eines Programms oder eines Listings. Mit **CTRL A** wird das zuletzt eingegebene Zeichen gelöscht (Code 01). Mit **ESC** (Code 1B) wird die gerade eingegebene Zeile gelöscht, wenn CR noch nicht gegeben wurde. **CTRL B** besitzt eine besondere Bedeutung (Code 02): Wird CTRL B ausgeführt, so gibt der Interpreter keine Zeichen mehr aus, aber empfängt noch alle Zeichen. Damit ist es möglich, Programme vom Kassettenrecorder aus einzulesen. Die Programme werden mit Hilfe des LIST-Befehls ausgegeben, während ein Kassettenrecorder mitläuft, der simultan über ein Modem an der Serienschnittstelle zum Datensichtgerät hängt. Bei der Wiedergabe wird CTRL B betätigt (zuvor NEW eingeben), dann wird der Ausgang des Kassettenmodems mit der Serienschnittstelle des Datensichtgeräteausgangs parallel geschaltet. Das Programm kann dann eingelesen werden. Am Schluß wird wieder CTRL B betätigt, so daß mit dem Interpreter wieder normal gearbeitet werden kann. Diese Unterdrückung der Ausgabe ist nötig, da sonst durch die Ausgabevorgänge eine Verlangsamung des Eingabevorgangs die Folge wäre und das System außer Tritt käme.

Programmbeispiele

```
10 PRINT 'GEBEN SIE EINEN STRING EIN'
20 I$ T0P
30 FOR I=1 TO LEN
40 IF PEEK(TOP+I)=' ' POKE TOP+I,'-'
50 NEXT I
55 O$ TOP
>RUN
GEBEN SIE EINEN STRING EIN
DIES IST EIN TESTSTRING
DIES-IST-EIN-TESTSTRING

10 FOR I=1 TO 10
20 FOR J=1 TO 10
30 @(I)=@(I)+RND(100)
40 NEXT J
50 NEXT I
60 FOR I=1 TO 10
70 PRINT @(I),
80 NEXT I
>RUN
   479   6l2   269   685   379   496   514   477   338   422
READY
>RUN
   872  1848   917  1143   983   952  1857   834   899   684
READY
>RUN
  1468  1578  1358  1711  1588  1472  1664  1358  1332  1147
READY

>PRINT A,B,A<B,A>B,A=B,A<==
    32    13     0     1     0     0     0

READY

>PRINT A,B,A<B,A>B,A=B,A#B,A<=B,A>=B
    32    13     0     1     0     1     0     1

READY

>PRINT (1=1)*(A<B)*100+A+A*('1'='1')
    64

>PRINT SIZE
   602
READY

>END HEX(3F00)
READY
```

```
>PRINT SIZE
  8995

>10 REM AUSDRUCKEN VON ZUFALLSZAHLEN ZWISCHEN 0 UND 9
>20 FOR I=1 TO 5
>30 PRINT RND(10)-1
>40 NEXT I
>RUN
     9
     7
     9
     2
     5

>10 INPUT 'GEBEN SIE ZAHL EIN'A,B
>20 PRINT A*A,B*B,A+B
>30 PRINT 'MIT ANDEREM FORMAT'
>40 PRINT #10,A+A,B*B,A+B
>RUN
    GEBEN SIE ZAHL EIN:12+1
    B:23-5
       169   324    31
    MIT ANDEREM FORMAT
            26       324         31

10 REM ANWENDUNG VON INCHAR
20 A=INCHAR
30 IF A=' ' STOP
40 IF A='0' B='*'
50 IF A='1' B=' '
60 IF A='2' B=HEX(D)
70 OUTCHAR(B)
80 GOTO 20
>RUN
****   * *   * *
 *  ** **

10 REM GAUSSCHE VERTEILUNG
20 FOR I=1 TO 300
30 FOR J=1 TO 10
40 @(I)=@(I)+RND(60)
50 NEXT J
60 NEXT I
70 FOR I=1 TO 300
80 @(I)=@(I)/10
90 NEXT I
100 A=0
105 I=1
110 IF @(I)<@(I+1) H=@(I+1);@(I)+1=@(I);@(I)=H;A=1
120 I=I+1
130 IF I<300 GOTO 110
```

```
140 IF A=1 GOTO 100
150 FOR I=1 TO 300
160 IF @(I)=@(I+1) PRINT " ",
170 IF @(I)>@(I+1) PHINT "*"
180 NEXT I
>RUN
 *
  *
   *
         *
       *
       *
         *
           *
           *
              *
              *
              *
                      *
                   *
                     *
                  *
               *
                 *
               *
                    *
                         *
              *
    *
   *
   *
 *
 *
*
*
*
READY
REM ZEITBEDARF ETWA 45 MIN

>REM DRUCKEN IN EINER ZEILE
READY
>10 FOR I=1 TO 5
>20 PRINT RND(10)-1,
>30 NEXT I
>RUN
     7     4     4     7     5
READY
```

From:
https://hc-ddr.hucki.net/wiki/ - **Homecomputer DDR**

Permanent link:
https://hc-ddr.hucki.net/wiki/doku.php/z1013/software/tinybasic/rdk

Last update: **2012/12/03 08:13**