N.A.S. Elektronische Halbleiter GmbH.

N.A.S.-ELEKTRONIK · BRIENNERSTR. 56 · D-8000 MÜNCHEN 2 · TELEFON: 089/523 153/4 · TELEX 0522061

# N A S C O M

## 8K B A S I C

## Programmierhandbuch

Inhaltsverzeichnis

## 7. Praktische Arbeit mit BASIC

Anhang

## 1. Einführung

NASCOM 8K BASIC basiert auf dem Microsoft BASIC, das inzwischen zu einem Industriestandard geworden ist. Damit besteht ein hoher Grad an Kompatibilität zwischen den NASCOM-BASIC-Programmen und den Programmen, die in Büchern und Fachzeitschriften veröffentlicht werden.

Der BASIC-Interpreter bietet 7-stellige Fließkommazahlen im Bereich zwischen 1.70141E38 bis 2.9387E-38. Außerdem verfügt er über die trigonometrischen Funktionen und erlaubt auch String-Handling und PIO-Steuerung. Der Benutzer kann eigene Funktionen in BASIC oder Maschinensprache definieren und sie ins Programm einbauen, um somit sehr flexibel arbeiten zu können.

Außerdem bietet der Interpreter folgende Eigenschaften:

- arbeitet mit NASBUG T2, T4 oder mit dem Nas-sys.
- mit Nas-sys stehen ausgefeilte Editiermöglichkeiten für Daten- und Programmeingaben zur Verfügung.
- Befehle zum Löschen des Bildschirmes und zur Cursorpositionierung. Damit ist eine beliebige Formatierung bei der Ein/Ausgabe mit BASIC möglich.
- verbesserter LIST-Befehl für den Gebrauch mit dem NASCOM Display.
- Drucker oder externe Datensichtgeräte können über Serienschnittstelle angesteuert werden.
- verbesserte Handhabung des Cassettenspeichers. Programme und Daten können einfach identifiziert werden.Fehlerprüfung bei Programm-und Datenspeicherung. (Bei vielen anderen Systemen können Daten fehlerhaft abgespeichert werden, ohne daß es der Rechner bemerkt.)
- kann die Graphikoptionen (Zusatzkarte) des NASCOM bedienen. Befehle dazu: SET, RESET und POINT .

### 1.1 Einführung in dieses Handbuch

Dieses Handbuch beschreibt die Eigenschaften des NASCOM 8K BASIC. Es ist nicht als Einführung in das Programmieren mit BASIC gedacht.Das soll anderen Büchern vorbehalten bleiben. (Siehe Anhang H).

a. Vereinbarungen bzgl. Schreibweise

Zur Vereinfachung wollen wir für unsere Schreibweise einige Vereinbarungen treffen, wenn wir das BASIC-Handbuch durcharbeiten:

1. Worte mit Großbuchstaben müssen immer genauso geschrieben werden wie gezeigt. Es sind meist Namen von Befehlen oder Kommandos.
2. Begriffe, die in Dreiecksklammern eingeschlossen sind, (<>) müssen in der Weise, wie im Text erläutert, eingefügt werden. Begriffe, die in eckigen Klammern eingeschlossen sind ([ ]), müssen nur bei Bedarf eingefügt werden. Begriffe, die in beiden Klammertypen gleichzeitig eingeschlossen sind, z.B. [< W>] , müssen nur eingesetzt werden, wenn eine zusätzliche Eigenschaft des BASIC genutzt werden soll. Begriffe, die von (...) gefolgt sind, können beliebig oft wiederholt werden, falls erforderlich.

3. Shift / oder Control /, gefolgt von einem Buchstaben, bedeutet, daß das Zeichen geschrieben wird, indem man zunächst die Shift- oder Control-Taste betätigt und gedrückt läßt. Anschließend betätigt man die angegebene Zeichen-Taste.
4. Alle angegebenen Satzzeichen müssen geschrieben werden.

b. Definitionen

Einige Dinge sind wichtig zu wissen:

Alphanumerische Zeichen: Alle Buchstaben und Zahlen zusammengenommen bezeichnet man als alphanumerische Zeichen.

Enter, Newline, Carriage Return: (Enter, Neue Zeile, Wagenrücklauf) Beziehen sich immer auf die Taste, die das Terminal veranlaßt, den Druckkopf oder den Cursor auf dem Anfang der nächsten Zeile zu positionieren bei gleichzeitigem Wagenrücklauf. Es ist dies die Taste, die veranlaßt, daß eine gerade geschriebene Zeile in den Arbeitsspeicher übernommen wird.

Kommandoebene: Die Kommandoebene beim BASIC-Interpreter ist immer dann eingeschaltet, wenn BASIC "OK" geschrieben hat. Nun kann der Interpreter Kommandos annehmen.

Kommandos und Befehle: Anweisungen im NASCOM BASIC werden grob in zwei Klassen eingeteilt. Wir nennen Sie Kommandos und Befehle. Kommandos sind Anweisungen, die man normalerweise nur im direkten Betrieb verwendet, d.h. also: nicht in einem BASIC-Programm. (Siehe dazu auch 1.2 Betriebsarten). Einige Kommandos, z.B. CONT können nur im direkten Betrieb verwendet werden, da sie in einem BASIC-Programm gar keinen Sinn ergeben würden. Einige Anweisungen werden normalerweise nicht als Programmbefehle verwendet, da Ihre Verwendung eine Rückkehr auf die Kommandoebene verursachen würde. Die meisten Kommandos können jedoch auch in einem Programm sinnvoll verwendet werden. Befehle sind Anweisungen, die normalerweise im indirekten Betrieb (also im Programm) verwendet werden. Einige Befehle, wie z.B. DEF können nur im indirekten Betrieb verwendet werden.

Editieren: Das Löschen, Hinzufügen und Ändern von Zeilen eines Programmes sowie das Vorbereiten von Daten für die Ausgabe gemäß einem bestimmten Format, bezeichnet man als Editieren. Die jeweilige Bedeutung des Begriffes wird aus dem Zusammenhang klar werden. (Anm. d. Übers.: Im deutschen Sprachgebrauch spricht man bei der formatierten Ausgabe von Datensätzen nicht von Editieren, sondern von "Formatieren". Wir halten uns jedoch hier an den engl. Sprachgebrauch).

Integer-Ausdruck: Ein Ausdruck, der ganzzahlig ist. Die Einzelkomponenten eines Ausdruckes, der einen ganzzahligen Wert ergibt (oder gerundet wird) brauchen nicht vom Typ Integer zu sein.

Reservierte Worte: Einige Worte sind in BASIC für Kommandos und Befehle reserviert. Diese Worte dürfen nicht für Variablen-oder Funktionsbezeichnungen verwendet werden.

String-Literal: Eine Zeichenkette (String) - sie ist dadurch gekennzeichnet, daß sie in Hochkommata eingeschlossen ist (") - wird genauso eingegeben und ausgegeben, wie sie geschrieben wird. Die Anführungszeichen sind nicht im String enthalten. Anführungszeichen dürfen in einem String nicht auftreten. Konstruktionen wie ""Das Ergebnis ist:"" sind unzulässig.

## 1.2 Betriebsarten

Das NASCOM BASIC arbeitet mit zwei verschiedenen Betriebsarten. Im direkten Betrieb wird jede Anweisung ausgeführt, sobald sie in den Rechner eingegeben worden ist. Die Ergebnisse der arithmetischen und logischen Operationen werden angezeigt und Variablenwerte für spätere Verwendung gespeichert. Der Befehl selbst wird jedoch nicht gespeichert. Diese Betriebsart dient zur Fehlersuche oder für schnelle kleine Berechnungen, die den Aufwand eines Programmes nicht rechtfertigen.

Im indirekten Betrieb führt der Rechner die Anweisungen aus, die ein im Speicher abgelegtes Programm liefert. Jede Programmzeile muß bei der Eingabe mit einer Zeilennummer versehen werden. Eine Programmzeile beginnt mit der Zeilennummer. Ein Programm wird mit dem Kommando RUN gestartet.

## 1.3 Formate

a. Zeilen

Die Zeile ist die grundsätzliche Anweisungsform beim NASCOM BASIC. Eine NASCOM BASIC Zeile hat folgendes Aussehen:

nnnnn ⟨BASIC Anweisung⟩[:⟨BASIC Anweisung⟩ ...]

Jede NASCOM BASIC Zeile beginnt mit einer Nummer. Diese Nummer bezeichnet die Adresse der Zeile im Speicher und zeigt an, in welcher Reihenfolge die Zeilen des Programmes abgearbeitet werden sollen. Die Zeilennummer wird auch für Sprünge benötigt und zum Editieren. Die Zeilennummern müssen zwischen Ø und 65529 liegen. Es ist sehr zweckmäßig von Zeile zu Zeile immer fünf oder 10 Nummern unbenutzt zu lassen, damit man später Ergänzungen einfügen kann.

Auf eine Zeilennummer folgen eine oder mehrere BASIC-Anweisungen. Das erste Wort einer Anweisung legt den Typ der Operation fest. Die folgende Liste von Argumenten, die auf das Operations-Wort folgen, können den verschiedensten Zwecken dienen. Ein Argument (man nennt es auch "Operand") kann sich auf Daten beziehen (oder auf den Namen einer Variablen, die einen bestimmten Wert hat). Bei einigen wichtigen Anweisungen hängt die Operation davon ab, welche Bedingungen (z.B.: "Ergebnis=O") oder welche Optionen in der Operandenliste spezifiziert sind.

Jeder Anweisungstyp wird in den Abschnitten 2,3 und 4 ausführlich vorgestellt.

Will man mehr als eine Anweisung in eine Zeile schreiben, so kann man die Anweisungen durch Doppelpunkte (:) trennen. Jede Anzahl von Anweisungen kann so verbunden werden, wenn die Liste der Anweisungen nicht länger als 72 Zeichen ist. Wenn der Nas-sys-Monitor im Mode "normal" verwendet wird, sind nicht mehr als 48 Zeichen pro Zeile zulässig. 72 Zeichen können auch bei Nas-sys verwendet werden, wenn man den"monitor mode" verwendet. Man schreibt dann einen XØ-Befehl und kehrt mit Z in den BASIC-Interpreter zurück.

b. Bemerkungen (REM, engl.:"remarks")

In den meisten Fällen wird ein reichlich kommentiertes Programm viel leichter verständlich sein, als ein schlecht kommentiertes

Programm. NASCOM Basic bietet die Möglichkeit Bemerkungen einzubauen, ohne daß der Programmablauf beeinflußt wird. Dazu dient der REM (Remark ≙ Bemerkung)-Befehl. Das Format dieses Befehles ist:

REM <Bemerkungen>

Ein REM-Befehl wird vom Basic Interpreter nicht ausgeführt, doch kann eine Verzweigung (ein Sprung) zu einer Zeilennummer führen, die vor einem REM-Befehl steht. Sprünge zu REM-Anweisungen sind erlaubt. REM-Befehle werden mit einem NEWLINE abgeschlossen oder mit einem "end of the line", niemals aber mit einem Doppelpunkt.

Beispiel: 1ØØ REM FÜHRE DIESE SCHLEIFE AUS: FOR E = 1 TO 1Ø
1Ø1 FOR E =1 TO 1Ø: REM FÜHRE DIESE SCHLEIFE AUS

In der Zeile 1ØØ wird die FOR-Schleife nicht ausgeführt, weil sie hinter dem REM-Befehl steht!! In der Zeile 1Ø1 wird die FOR-Schleife ausgeführt. Der REM-Befehl steht erst hinter der FOR-Schleife.

c. Fehler:

Wenn der Basic Interpreter einen Fehler findet, führt dies zu einer Programmunterbrechnung und es wird eine Fehlermeldung ausgegeben. Die Fehlermeldungen haben beim NASCOM Basic folgendes Format:

Direkte Betriebsart: ?XX ERROR
Indirekte Betriebsart: ?XX ERROR IN nnnnn

XX ist der Code der Fehlermeldung (siehe Abschnitt 6.5: Liste der Fehlercodes und Fehlermeldungen) und nnnnn ist die Nummer der Zeile, in der der Fehler aufgetreten ist. Zu jedem Befehl gibt es einige spezielle Möglichkeiten für Fehlermeldungen. Ebenso gibt es einige Fehlermeldungen (Syntax), die für alle Befehle gleich aussehen. Die Fehlermeldungen die nur bei bestimmten Fehlern vorkommen, werden bei der Beschreibung dieser Befehle mit diskutiert.

## 1.4 Editieren - Einfache Möglichkeiten

Mit Hilfe der Editiermöglichkeiten des Basic können Fehler korrigiert und Programmteile angefügt werden, ohne daß dadurch der Rest des Programmes beeinträchtigt würde. Falls erforderlich kann das gesamte Programm gelöscht werden.

Folgende Möglichkeiten bietet der Basic Interpreter mit NASBUG T2 oder T4 oder NAS-SYS in XØ-Modus (d.h. mit externem Datensichtgerät):

a. Korrigieren eines einzelnen Zeichens. Wenn ein fehlerhaftes Zeichen geschrieben wurde, kann es sofort wieder mit BACKSPACE gelöscht werden. Jedes Mal, wenn BACKSPACE gedrückt wird, wird ein Zeichen auf dem Bildschirm gelöscht. Wenn ein Zeichen keinen Vorgänger in der gleichen Zeile hat und BACKSPACE getätigt wird, wird ein Wagenrücklauf (CR) ausgeführt und eine neue Zeile begonnen. Wenn alle unerwünschten Zeichen beseitigt sind kann der Rest der Zeile geschrieben werden.

Wenn RUBOUT (CTRL Z) gedrückt wurde, wird das letzte Zeichen gelöscht und nochmals auf dem Bild ausgegeben. Mit jedem RUBOUT wird das nächste Zeichen gelöscht. Wenn ein neues Zeichen

eingegeben wurde, erscheint es auf dem Bildschirm.

Beispiel:

```
100 X==X Y=10
```

Mit RUBOUT werden nacheinander "=" und "X" gelöscht, die durch "Y=" ersetzt wurden.

b. Korrektur einer Zeile: Eine gerade geschriebene Zeile kann gelöscht werden, wenn man statt eines NEWLINE ein @-Zeichen eingibt. Ein Wagenrücklauf wird automatisch eingefügt, wenn die Zeile gelöscht wurde. Die Eingabe CONTROL/U hat den gleichen Effekt.

c. Korrigieren eines ganzen Programmes: Mit dem NEW-Kommando wird der gesamte Speicherbereich und alle Variablen gelöscht. Immer dann, wenn man ein neues Programm beginnen will, schreibt man ein NEW-Kommando.

## 1.5 Eingabe von Programmen mit Nas-sys

Wenn der Nas-sys-Monitor verwendet wird, steht die ganze Palette von Möglichkeiten zur Verfügung, die der Nas-sys-Editor bietet. Zusätzlich zu den Editiermöglichkeiten für eine Programmzeile kann ein Programm aufgelistet werden, Zeilen einzeln auf dem Bildschirm editiert und die geänderten Zeilen wieder eingegeben werden. Dateneingabe erfolgt normalerweise genauso. (siehe dazu auch 5).

## 2. Befehle und Ausdrücke

Die einfachsten BASIC-Ausdrücke sind Konstanten, Variablen und Funktionsaufrufe.

a. Konstanten NASCOM BASIC kennt Integerzahlen, Fließkommazahlen und Strings als Konstanten. Siehe dazu 4.1. Einige Beispiele für Konstanten:

```
123
3.141
0.0436
1.25E+05
```

Bei der Eingabe numerischer Konstanten vom Terminal aus oder beim Angeben von Konstanten im Programm, dürfen diese Konstanten eine beliebige Anzahl von Ziffern haben - bis zur Länge einer Zeile. Siehe dazu auch 1.3. Es sind jedoch immer nur die ersten 7 Ziffern einer Zahl signifikant. Die siebte Ziffer wird aufgerundet. Der Befehl:

```
PRINT 1.234567890123
```

erzeugt also die folgende Ausgabe:

```
1.23457
OK
```

Das Ausgabeformat für eine Zahl ergibt sich nach den folgenden Regeln:

1. Wenn die Zahl negativ ist, wird links vor der Zahl ein Minuszeichen (-) gedruckt. Wenn die Zahl positiv ist, wird ein Zwischenraum ausgegeben.
2. Wenn der Wert einer Zahl ganzzahlig ist und zwischen Ø und 999999 liegt, wird er als Integerwert ausgegeben.
3. Wenn der Absolutwert einer Zahl größer oder gleich 0.01 oder kleiner oder gleich 999999 ist, wird die Zahl mit Festkomma und ohne Exponent ausgegeben.
4. Wenn die Zahl nicht in die Kategorie 2 oder 3 fällt, wird die wissenschaftliche Notation benutzt:

   ```
   SX.XXXXXESTT
   ```

   S steht für das Vorzeichen bei Mantisse und Exponent (diese Vorzeichen können sich natürlich unterscheiden). X steht für die Ziffern der Mantisse und T für die Ziffern des Exponenten. E und D haben die Bedeutung: " ...mal zehn hoch....". Nicht-signifikante Nullen in der Mantisse werden unterdrückt, im Exponenten werden sie jedoch immer angegeben Die Vorzeichenkonvention aus Regel 1 gilt bei wissenschaftlicher Notation für die Mantisse. Der Exponent muß im Wertebereich -38 bis +38 liegen. Die größte darstellbare Zahl in BASIC ist 1.70141E38, die kleinste positive Zahl ist 2.9387E-38. Die folgenden Beispiele zeigen, wie der NASCOM errechnete Werte ausgibt:

| Errechneter Wert | NASCOM BASIC Ausgabe |
|---|---|
| +1 | 1 |
| -1 | -1 |
| 6523 | 6523 |
| 1E20 | 1E20 |
| -12.34567E-10 | -1.23456E-09 |
| 1.234567E-7 | 1.23457E-07 |
| 1000000 | 1E+06 |
| .1 | .1 |
| .01 | .01 |
| .000123 | 1.23E-04 |
| -25.460 | -25.46 |

Bei allen Ausgabeformaten wird ein Zwischenraum eingefügt, nachdem eine Zahl ausgegeben wurde. BASIC überprüft, ob die auszugebende Zahl in die aktuelle Zeile paßt. Falls nicht, wird ein Wagenrücklauf ausgeführt und die ganze Zahl in die nächste Zeile geschrieben.

b. Variablen Einer Variablen können beliebige Werte zugewiesen werden, die im Rechenbereich von BASIC liegen. Der Wert, der einer Variablen zugewiesen wird, kann entweder zu irgendeinem Zeitpunkt explizit festgelegt werden, oder er wird im Ablauf des Programmes errechnet. Bevor einer Variablen ein Wert zugewiesen wird, ist ihr Wert Ø. In NASCOM BASIC sind beliebig lange Variablennamen zugelassen. Es werden jedoch immer nur die ersten zwei Zeichen des Namens verarbeitet. Das erste Zeichen muß immer ein Buchstabe sein. Folgende Beispiele zeigen zulässige und unzulässige Variablennamen:

| Zulässig | Unzulässig | |
|---|---|---|
| A | %A | (Erstes Zeichen muß ein |
| Z1 | | Buchstabe sein !) |
| TP | TO | (Variablenname darf kein reser- |
| PSTG$ | | viertes Wort sein). |
| COUNT | RGOTO | (Variablenname darf kein reser- |
| | | viertes Wort enthalten.) |

Ein Variablenname kann auch einen String darstellen. Wie man diese Möglichkeit verwenden kann, zeigt Abschnitt 4.

Intern stellt BASIC alle Variablen binär dar. Daher werden auch einige einfachgenaue 8-stellige Zahlen richtig (ohne Rundungsfehler) verarbeitet.

c. Feldvariablen; Variablenfelder Es ist oftmals von Vorteil, wenn man mehrere Variablen unter einem Namen aufrufen kann. Zum Beispiel bei Matrizenoperationen wird jedes Element der Matrize separat bearbeitet, doch ist es bei der Programmierung zweckmäßig, die ganze Matrize unter einem Namen aufzurufen. Zu diesem Zweck bietet NASCOM BASIC die Möglichkeit, Variablen zu indizieren. Alle Variablen gleichen Namens (alle mit verschiedenem Index) bilden ein Feld. Die Form einer Feldvariablen ist folgende:

VV(<Index>[,<Index>....])

VV ist der Variablenname und die Indizes sind Integerzahlen. Indizes können in runde oder eckige Klammern eingeschlossen werden. Eine Arrayvariable kann soviele Indizes haben, wie in eine Zeile passen und der Speicher aufnehmen kann. Der kleinste Index ist Ø.

Beispiel: A(5) Ist das sechste Element des Feldes A. A(Ø) ist das erste Element des FEldes A.

ARRAY(I,2*J) Die Adresse dieses Elementes in einem zweidimensionalen Feld wird genau dann errechnet (mit den gerade aktuellen Werten für I, J), wenn das Programm diesen Aufruf erreicht. I und J werden immer zu Integerzahlen gerundet. Wenn I=3 und J=2.4, dann wird das Feldelement ARRAY (3,4) aufgerufen.

Die DIM-Anweisung reserviert Speicherplatz für Feldvariablen und setzt alle Feldvariablen =Ø. Die Form der DIM-Anweisung ist:

DIM VV (<Index>[,<Index>...])

Dabei ist VV ein zulässiger Variablenname. Die Indizes sind Integerzahlen, die festlegen, welches die größten Indizes sein dürfen, mit denen VV aufgerufen werden kann. Mit einer DIM-Anweisung können auch mehrere Felder definiert werden. Dazu ein Beispiel:

```
113 DIM A(3),D$(2,2,2)
114 DIM R2%(4),B(10)
115 DIM Q1(N),Z$(2+I)
```

Felder können auch während des Programmablaufes mit errechneten Indizes vereinbart werden. (Siehe Zeile 115).

Sobald die DIM-Anweisung erreicht wird, werden die aktuellen Werte (hier für N und I) eingesetzt und auf Integerformat gebracht.
Falls eine Feldvariable aufgerufen wird, für die keine DIM-Anweisung vorangegangen ist, so nimmt BASIC an, daß das zugehörige Feld maximal 11 Elemente (0..10) für jede Dimension, die verwendet wurde, haben soll. Eine Fehlermeldung BS oder SUBSCRIPT OUT OF RANGE wird immer dann ausgegeben, wenn ein Feldelement aufgerufen wird, das nicht innerhalb der definierten Grenzen eines Feldes liegt. Ein solcher Fehler kann z.B. auch auftreten, wenn die aufgerufene Variable und DIM-Anweisung des Feldes unterschiedliche Anzahlen Indizes haben:
Z.B.:

```
30 LET A(1,2,3)=X gibt Fehler, wenn A definiert war:
10 DIM A(2,2)
```

Eine Fehlermeldung DD oder REDIMENSIONED ARRAY wird ausgegeben, wenn eine DIM-Anweisung auftritt, nachdem schon vorher für das gleiche Feld eine DIM-Anweisung durchlaufen wurde.Dieser Fehler tritt häufig dann auf, wenn einem Feld bereits die Dimension 10 zugeordnet wurde (einfach dadurch, daß eine Variable des Feldes aufgerufen wurde, bevor das DIM-Statement erreicht wurde) und dann erst eine DIM-Anweisung folgt.

d. Operatoren und Reihenfolge der Abarbeitung NASCOM BASIC bietet alle üblichen arithmetischen und logischen Operatoren. Wenn mehrere Operatoren in einem Ausdruck auftreten, ist die Reihenfolge der Abarbeitung so geregelt, wie es untenstehende Tabelle zeigt. Die Reihenfolge der Abarbeitung kann nach den üblichen Regeln der Algebra durch Klammern festgelegt werden.

Tabelle: Reihenfolge der Abarbeitung von Operationen:
Operatoren sind in fallender Priorität der Abarbeitung aufgelistet. Operatoren gleicher Priorität stehen in der Tabelle unter gleicher Zeilennummer. Ein Ausdruck wird immer von links nach rechts abgearbeitet.

1. Ausdrücke, die in Klammern () eingeschlossen sind.
2. ↑ Exponent ( $Y^x$). Jede Zahl Y mit x=0 ergibt 1; bei negativem x wird ein /Ø oder DIVIDE BY ZERO-Fehler ausgegeben
3. - Negatives Vorzeichen einfügen
4. *,/ Multiplikation, Division
5. +,- Addition, Subtraktion
6. Vergleichsoperatoren

| | | | |
|---|---|---|---|
| = gleich | < kleiner als | <=; =< | kleiner oder gleich |
| <> ungleich | > größer als | >=; => | größer oder gleich |

7. NOT logische Negation (bitweise)
8. AND logisches UND (bitweise)
9. OR logisches ODER (bitweise).

Vergleichsoperatoren können in jedem Ausdruck verwendet werden. Vergleichsoperatoren liefern entweder den Wert logisch wahr (-1) oder logisch falsch (Ø).

e. Logische Operationen: Logische Operatoren verwendet man für Bit-Manipulationen und für Boolesche algebraische Funktionen. Die Operationen AND,OR,NOT wandeln ihre Ergebnisse in 16Bit-Integerzahlen um, die dann im Bereich -32768 bis +32767 liegen. Die Eingaben für logische Operationen haben in gleicher Weise zu erfolgen. Falls ein Argument für eine logische Operation nicht in diesem Bereich liegt, wird eine Fehlermeldung FC oder ILLEGAL FUNCTION CALL ausgegeben und die Abarbeitung des Programmes unterbrochen. Wahrheitstabellen für die logischen Operationen zeigen die nachfolgenden Beispiele. Die Operationen werden bitweise durchgeführt und die Ergebnisse ebenso bitweise ermittelt. Bei binären Operationen ist Bit 7 das MSB ("most significant bit") und Bit Ø das LSB ("least significant bit").

logisches UND:

| X | Y | X AND Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

logisches Oder:

| X | Y | X OR Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

logisches Nicht:

| X | NOT X |
|---|---|
| 1 | 0 |
| 0 | 1 |

Einige Beispiele sollen verdeutlichen, wie die logischen Operationen funktionieren:

63 AND 16 = 16 63=binär 111111 und 16=binär 10000 also 63 AND 16=16

15 AND 14 = 14 15=binär 1111 und 14=binär 1110; also ist 15 AND 14 = 14

-1 AND 8 = 8 -1 ist binär 1111111111111111 und 8 ist binär 1000 Also ist -1 AND 8 = 8

4 OR 2 = 6 4 ist binär 100 und 2 ist binär 10. Also ist 4 OR 2 = 6.

10 OR 10 = 10 10 ist binär 1010; Oderiert mit sich selbst ändert sich keine Bitposition, also Ergebnis 10.

-1 OR -2 = -1 -1 ist binär 1111111111111111 und -2 ist binär 1111111111111110. So ist -1 OR -2 = -1

NOT Ø=-1 Das Stellenkomplement von sechszehn Ø sind sechszehn 1; das ist die Darstellung von -1.

Typische logische Operationen sind z.B. Maskierungen. Man verwendet Masken, wenn man feststellen will, ob bestimmte Bits in einem Wort gesetzt sind. Solche Eingabewerte könnten z.B. von einem Port des Rechners geliefert werden und stellen Zustände irgendeines externen Gerätes dar. Weitere Anwendungen logischer Operationen werden wir bei der IF-Anweisung besprechen.

f. LET-Anweisung Die LET-Anweisung wird verwendet, um einer Variablen einen Wert zuzuweisen. Die LET-Anweisung hat die Form:

LET <VV>=<Ausdruck>

VV ist der Variablenname und der Ausdruck ist ein arithmetischer, logischer oder String-Ausdruck. Beispiele dazu:

```
1000   LET V=X
110    LET I=I+1     ; "=" bedeutet: "ersetze durch"
```

Das Wort LET in der LET-Anweisung kann weggelassen werden. Zuweisungen der Form 120 V=.5/(X*2) sind zulässig. Eine Fehlermeldung SN oder SYNTAX ERROR wird immer dann ausgegeben, wenn ein Befehl eine unzulässige Form hat, unzulässige Zeichen verwendet wurden, fehlerhafte Satzzeichen eingefügt wurden oder Klammern fehlen. Eine Fehlermeldung OV oder OVERFLOW ERROR tritt immer dann auf, wenn ein Wert zu groß ist, um im NASCOM BASIC-Zahlenformat dargestellt zu werden. Alle Zahlen müssen im Bereich 1E-38 bis 1.70141E38 bzw. -1E-38 bis -1.70141E38 liegen. Beim Versuch durch Ø zu dividieren, wird ein /Ø-Fehler bzw. DIVIDE BY ZERO gemeldet.

Eine Diskussion von Strings, String Variablen und String Operationen erfolgt im Abschnitt 4.

## 2.2 Verzweigungen, Schleifen und Unterprogramme

a. Verzweigungen Normalerweise wird in einem Programm eine Zeile nach der anderen abgearbeitet. Verzweigungen ändern die Reihenfolge der Abarbeitung. Verzweigungen sind die Grundlage aller Entscheidungsvorgänge in einem Programm. Die Anweisungen, die zu Verzweigungen führen, heißen beim NASCOM BASIC: GOTO, IF ... THEN und ON ... GOTO.

1. GOTO ist ein unbedingter Sprung. Seine Form ist:

   GOTO <mmmmm>

   Nachdem die GOTO-Anweisung ausgeführt wurde, wird das Programm bei der Zeilennummer mmmmm fortgesetzt.

2. IF ... THEN ist ein bedingter Sprung. Seine Form ist:

   IF <Ausdruck> THEN <mmmmm>

   Der Ausdruck ist eine zulässige arithmetische Operation, Vergleichsoperation oder logische Operation. mmmmm ist die Zeilennummer, zu der gesprungen wird. Falls der Ausdruck nicht =Ø ist, führt BASIC das Programm weiter ab Zeilennummer mmmmm aus. Falls das Ergebnis =Ø ist, wird die nächstfolgende Programmzeile abgearbeitet. Eine alternative Form der IF ... THEN - Anweisung ist die folgende:

   IF <Ausdruck> THEN <Anweisung>

Beispiele:

```
10  IF A=10  THEN  40   ; Wenn der Ausdruck A=10 logisch wahr
                          ist, springt BASIC zur Zeile Nr. 40.
                          Andernfalls wird die nächste Zeile
                          ausgeführt.
15  IF A<B+C OR X THEN 100  ; Wenn der Ausdruck ein Ergebnis
                              liefert, das nicht =Ø ist, dann
                          springt BASIC zur Zeile 100. Ansonsten
                          wird die nächste Zeile ausgeführt.
20  IF X THEN 25        ; Wenn X nicht =Ø springt BASIC zu Zeile 2
30  IF X=Y THEN PRINT X; Wenn X=Y ist, wird die PRINT-Anweisung
                          ausgeführt. Andernfalls wird keine
                          Ausgabe durchgeführt.
```

```
35  X=Y+3 GOTO 39 ; Entsprechende Form zur IF ... THEN-
                    Anweisung. GOTO muß allerdings eine
                    Zeilennummer folgen, nie aber ein anderer
                    Befehl.
```

3. ON ... GOTO bietet die Möglichkeit für einen anderen Typ einer bedingten Verzweigung. Seine Form ist die folgende:

ON ⟨Ausdruck⟩ GOTO ⟨Liste von Zeilennummern⟩

Nachdem der Wert des Ausdruckes in einen Integerwert umgeformt worden ist, nennen wir diesen Wert I, springt BASIC zur I-ten Zeilennummer der "Liste von Zeilennummern". Der Anweisung dürfen soviele Zeilennummern folgen, wie in eine Zeile passen. Wenn I=Ø ist oder größer als die Anzahl Zeilennummern in der Liste, wird das Programm bei der nächsten Zeile nach der ON...GOTO-Anweisung fortgesetzt. Wenn I kleiner als Ø oder größer als 255 ist, wird die Fehlermeldung FC oder ILLEGAL FUNCTION CALL ERROR ausgegeben.

b. Schleifen  Oftmals erscheint es wünschenswert, Programmteile mehrfach, z.B. mit verschiedenen Datensätzen, zu durchlaufen. Zu diesem Zweck bietet BASIC die Befehle FOR und NEXT. Die Form des FOR-Befehles ist folgende:

FOR ⟨Variable⟩=⟨X⟩ TO ⟨Y⟩[STEP ⟨Z⟩]

Dabei sind X,Y und Z Ausdrücke. Wenn die FOR-Anweisung erstmalig erreicht wird, werden diese Ausdrücke errechnet. Die Variable wird auf den Anfangswert X gesetzt. BASIC führt dann die Anweisungen, die der FOR-Anweisung folgen, wie normal aus. Wenn eine NEXT-Anweisung aufgefunden wird, wird die Schrittweite Z zur Variablen addiert und mit dem Endwert Y vergleichen. Wenn Z, die Schrittweite, positiv ist und die Variable kleiner oder gleich dem Endwert ist, oder wenn die Schrittweite negativ ist und die Variable größer oder gleich dem Endwert, springt BASIC zurück zu dem Befehl, der unmittelbar auf die FOR-Anweisung folgt.Andernfalls wird das Programm beim ersten Befehl hinter dem NEXT-Statement fortgesetzt. Wenn keine Schrittweite explizit angegeben wird, nimmt BASIC den Wert 1 an.

Beispiele:

```
10  FOR I=2 TO 11 ; Die Schleife wird 10 mal ausgeführt,
                    wobei I nacheinander die Werte 2...11
                    annimmt.

20  FOR V=1 TO 9.3; Diese Schleife wird 9 mal ausge-
                    führt, bis V größer als 9.3 ist.

30  FOR V=10*N TO 3.4/Z  STEP SQR(R) ; Anfangswert, End-
                    wert und Schrittweite brauchen keine
                    Integerwerte zu sein. Sie werden
                    einmal errechnet, wenn die Schleife
                    beginnt.

40  FOR V=9 TO 1 STEP -1 ; Diese Schleife wird 9 mal
                           ausgeführt.
```

FOR...NEXT-Schleifen können ineinander geschachtelt werden. Innerhalb von FOR...NEXT-Schleifen können wieder FOR...NEXT-Schleifen auftreten. Hier ein Beispiel für zwei verschachtelte Schleifen:

```
100  FOR I=1 TO 10
120  FOR J=1 TO I
130  PRINT A(I,J)
140  NEXT J
150  NEXT I
```

In Zeile 130 wird je ein Element für I=1, I=2 usw. ausgedruckt. Wenn Schleifen verschachtelt werden, müssen sie verschiedene Variablennamen haben. Die NEXT-Anweisung für die innere Schleife (in unserem Beispiel J), muß erscheinen, bevor das NEXT-Statement für die äußere Schleife (Variable I) erreicht wird. Es können beliebig viele FOR...NEXT-Schleifen verschachtelt werden. Eine Begrenzung tritt nur durch die verfügbare Speicherkapazität auf.

Die NEXT-Anweisung hat die Form:

NEXT [<Variable>[,<Variable>...]]

Jeder Variablennahme ist der Variablenname einer FOR-Schleife deren Ende die NEXT-Anweisung bildet. Wenn bei einem NEXT-Statement kein Name angegeben ist, wird die Anweisung als Abschluß der letzten FOR-Schleife interpretiert. Bei verschachtelten Schleifen, die alle den selben Endpunkt haben, genügt eine NEXT-Anweisung für alle Schleifen. Der erste Variablenname der Liste bezieht sich immer auf die innerste Schleife, der nächste auf die nächst äußere Schleife usw. Falls BASIC eine NEXT-Anweisung auffindet, ohne daß eine FOR-Anweisung voranging, wird die Fehlermeldung NF oder NEXT WITHOUT FOR ERROR ausgegeben und die Programmausführung gestoppt.

c. Unterprogramme Oftmals kommt es vor, daß eine Reihe von Operationen in gleicher Weise an mehreren Stellen eines Programmes ausgeführt werden soll. Um Speicherplatz zu sparen und übersichtlich programmieren zu können, ist es sinnvoll, solche Befehlssequenzen zu Unterprogrammen zusammenzufassen. Ein Sprung zu einem Unterprogramm wird mit einem GOSUB-Befehl bezeichnet. Das Ende des Unterprogrammes wird mit einem RETURN-Befehl gekennzeichnet. Nachdem BASIC den RETURN-Befehl erkannt hat, führt es den Befehl aus, der auf die GOSUB-Anweisung folgt. Der GOSUB-Befehl hat folgendes Format:

GOSUB <Zeilennummer>

wobei die Zeilennummer die erste Zeilennummer des BASIC-Unterprogrammes ist. Ein Unterprogramm kann von jeder beliebigen Stelle des Programmes aus aufgerufen werden. Ebenso kann ein Unterprogramm andere Unterprogramme aufrufen. Dieses Verschachteln von Unterprogrammen ist nur durch die Größe des verfügbaren Speicherplatzes begrenzt. Bedingte Sprünge zu Unterprogrammen erlaubt der ON ... GOSUB - Befehl:

ON <Ausdruck> GOSUB < Liste von Zeilennummern>

Der Befehl wird genauso abgearbeitet, wie die ON ... GOTO-Anweisung, nur daß in der Liste der Zeilennummern die Anfangszeilen der jeweiligen Unterprogramme angegeben werden. Nachdem das aufgerufene Unterprogramm abgearbeitet ist, führt BASIC die Programmzeilen aus, die auf die ON...GOSUB-Anweisung folgen.

d. OUT OF MEMORY-Fehlermeldungen Die mögliche Anzahl Schleifen, Unterprogramme und Sprünge ist nicht durch BASIC eingeschränkt. Lediglich der vorhandene Speicherplatz schränkt die Möglichkeiten der Programmierung ein. Die Fehlermeldung OM oder OUT OF MEMORY zeigt an, daß ein Programm mehr Speicherplatz benötigt, als vorhanden ist. Anhang B gibt Auskunft darüber, wieviel Speicherplatz erforderlich ist, um ein Programm laufen zu lassen.

## 2.3 Eingabe und Ausgabe

a. INPUT Mit dem INPUT-Befehl werden Daten für das Programm angefordert. Diese Daten muß der Benutzer eingeben. Die INPUT-Anweisung hat die Form:

INPUT ⟨Liste von Variablen⟩

Der INPUT-Befehl bewirkt, daß die auf der Tastatur eingegebenen Werte den einzelnen Variablen der Liste zugeordnet werden. Wenn ein INPUT-Befehl ausgeführt wird, schreibt der Rechner ein Fragezeichen (?) auf den Bildschirm. Damit zeigt er an, daß eine Eingabe gewünscht wird. Der Bediener gibt die angeforderten Werte oder Strings(Zeichenketten) durch Kommata getrennt ein und betätigt die NEWLINE-Taste. Wenn nicht die erwarteten Eingaben erfolgen (z.B. Strings wenn Daten kommen sollten), dann schreibt BASIC "REDO FROM START?" und wartet darauf, daß die gewünschten Eingaben erfolgen. Wenn der INPUT-Befehl mehr Daten verlangt, als eingegeben wurde, schreibt BASIC ?? und wartet auf die restlichen Daten. Wenn mehr Daten eingegeben wurden, als der Interpreter erwartet, wird die Meldung "EXTRA IGNORED" ausgegeben und das Programm weiter abgearbeitet. Nachdem alle Daten eingegeben sind, wird das Programm bei der Programmzeile fortgesetzt, die auf den INPUT-Befehl folgt. Es besteht auch die Möglichkeit, beim INPUT-Befehl einen String anzugeben, der ausgegeben wird, wenn BASIC den INPUT-Befehl erreicht:

INPUT [⟨"string"⟩ ;]⟨Variablenliste⟩

Der String muß in Anführungszeichen (") eingeschlossen und von der Variablenliste durch ein Semikolon (;) getrennt sein. Beispiel:

```
100 INPUT "EINGABE FUER WERT X,Y";X,Y erzeugt die Ausgabe:

EINGABE FUER WERT X,Y?
```

Die angeforderten Wert für X und Y werden hinter dem Fragezeichen eingegeben. Wenn statt der angeforderten Werte ein NEWLINE eingegeben wird, werden die aktuellen Werte von X und Y nicht verändert.

b. PRINT Mit dem PRINT-Befehl können Daten oder Strings auf dem Bildschirm des NASCOM ausgegeben werden. Die einfachste PRINT-Anweisung ist:

PRINT

Sie führt einen Zeilenvorschub durch. Häufiger wird die Anweisung in der Form gebraucht:

PRINT ⟨Liste von Ausdrücken⟩

Der PRINT-Befehl veranlaßt, daß die Liste von Ausdrücken auf dem Bildschirm ausgegeben wird. Strings können ausgegeben werden, wenn man sie in Hochkommata einschließt (").
Das Ausgabeformat wird durch die Satzzeichen bestimmt, die die Ausdrücke der Liste voneinander trennen. Der NASCOM teilt eine Zeile in Abschnitte von je 14 Zeichen ein. Wenn zwei Ausdrücke durch Komma getrennt sind, beginnt die Ausgabe im nächsten 14-Stellen-Feld. Wenn die Ausdrücke durch einen Doppelpunkt (:) getrennt sind, wird garkein Zwischenraum eingefügt. Wenn die Liste von Ausdrücken mit einem Komma oder einem Semikolon (;) abgeschlossen wird, schreibt die nächste PRINT-Anweisung in der gleichen Zeile weiter. Andernfalls wird ein Wagenrücklauf und Zeilenvorschub ausgeführt.

c. DATA,READ,RESTORE

1. Die DATA-Anweisung Numerische Daten oder Strings können in einem Befehl des Programmes selbst auftreten, über Tastatur oder von einem anderen Peripheriegerät aus eingegeben werden oder von einer DATA-Anweisung erzeugt werden. Die DATA-Anweisung hat folgende Form:

DATA ⟨Liste⟩

Eingaben in die Liste sind entweder Zahlen oder Strings, die durch Kommata getrennt sind. Die DATA-Anweisung bewirkt, daß die in der Liste eingetragenen Zahlen oder Strings in codierter Form im Speicher abgelegt werden, sodaß man darauf mit einer READ-Anweisung Zugriff hat.
Beispiele:

```
10  DATA 1,2,-1E3,.04
20  DATA "ARR",NASCOM
```

Führende und nachfolgende Zwischenräume werden gelöscht, wenn ein String nicht in Hochkommata eingeschlossen ist.

2. Die READ-Anweisung Die Daten, die mit der DATA-Anweisung abgespeichert wurden, werden mittels READ-Anweisung gelesen:

READ ⟨Liste von Variablen⟩

wobei die Variablennamen durch Kommata getrennt sind. Durch die READ-Anweisung werden den Variablen in der READ-Liste nacheinander (von links nach rechts aus dem DATA-Datenfeld) die Daten zugewiesen, die mit dem DATA-Befehl abgespeichert wurden. Dies ist solange möglich, bis keine Daten aus der mit DATA erzeugten Liste zur Verfügung stehen. Falls mehr Daten mit READ gelesen werden, als zur Verfügung stehen, wird die Fehlermeldung OD oder OUT OF DATA ausgegeben. Wenn mit der DATA-Anweisung mehr Daten abgelegt wurden, als mit READ gelesen wurden, dann liest die nächste READ-Anweisung dort weiter, wo die vorhergehende Anweisung aufgehört hatte. Eine einzige READ-Anweisung kann auf mehr Daten zugreifen, als mit einer einzigen DATA-Anweisung erzeugt wurden. Ebenso können mehrere READ-Anweisungen nacheinander auf die Daten zugreifen, die mit nur einer DATA-Anweisung abgelegt wurden.

Oftmals treten SN oder SYNTAX-Fehler auf, wenn die DATA-Liste nicht in richtigem Format geschrieben wurde. Die Zeilennummer der Fehlermeldung zeigt an, in welcher DATA-Anweisung der Fehler aufgetreten ist.

3. Die RESTORE-Anweisung Nachdem die RESTORE-Anweisung ausgeführt wurde, wird das nächste Datum, das von einer READ-Anweisung gelesen wird, das erste Datum sein, das überhaupt von einer DATA-Anweisung generiert wurde. D.h.: Die mit DATA abgelegten Tabellen werden wieder von Beginn an abgearbeitet. Somit können die Daten nochmals neu gelesen werden.

d. CSAVE und CLOAD Zahlenfelder können auf Cassette übertragen oder von Cassette gelesen werden. Dazu werden die Befehle CSAVE* und CLOAD* verwendet. Die entsprechenden Formate sind:

CSAVE* ⟨Feldname⟩

und

CLOAD* ⟨Feldname⟩

Das Feld wird auf Cassette geschrieben, wobei vier oktale Bytes 210 vorangehen, um den Anfang des Datenfeldes zu markieren.

Diese Bytes werden später beim Einlesen mit CLOAD aufgesucht. Die Anzahl Bytes, die auf Cassette übertragen werden ist: 4+ 4*<Anzahl der Elemente des Feldes>.
Wenn ein Feld auf Cassette übertragen wird, wird angezeigt, welches Element gerade geschrieben wird. Dabei läuft der ganz links stehende Index am schnellsten durch. Mit

```
DIM A(10)
CSAVE* A
```

wird geschrieben:

```
A(Ø), A(1).....A(10)
```

Mit

```
DIM A(10,10)
CSAVE* A
```

wird geschrieben:

```
A(0,0), A(1,0)....A(10,0),A(10,1)...A(10,10).
```

Indem man diese Methode der Speicherung ausnutzt, kann man auch ein zweidimensionales Feld als eindimensionales Feld wieder einlesen.
NASCOM BASIC erzeugt außerdem zu jedem übertragenen Datensatz eine Prüfsumme. Wenn ein Prüfsummenfehler beim Lesen auftritt, wird die Fehlermeldung "Bad" ausgegeben und man kann nun das Programm neu starten und versuchen, die Daten nochmals zu lesen.

## e. Weitere Möglichkeiten zur Eingabe/Ausgabe

1. WAIT Der Zustand der Eingabeports kann mit dem WAIT-Befehl überwacht werden. Er hat folgende Form:

```
WAIT<I,J>[,<K>]
```

Dabei ist I die Adresse des Port, der überwacht werden soll und J und K sind Integer-Ausdrücke. Der Port-Zustand wird mit einer Exclusiv Oder-Funktion mit K verknüpft, das Ergebnis über eine Und-Verknüpfung mit J. Das Programm wartet solange, bis dabei der Wert Ø auftritt.J bezeichnet die Bits des Port I, die überprüft werden sollen, wobei das Programm wartet, bis das gleiche Wort wie bei K auftritt. Dann wird das Programm bei dem auf WAIT folgenden Befehl fortgesetzt. Falls K nicht angegeben wird, wird es als Ø angenommen. I,J und K müssen zwischen 0 und 255 liegen.

Beispiele:

| | |
|---|---|
| `WAIT 20,6` | Das Programm wartet solange, bis entweder Bit 1 oder Bit 2 von Port 20 gleich 1 sind. (Bit Ø ist LSB, Bit 7 MSB). |
| `WAIT 10,255,7` | Das Programm wartet solange, bis irgendeines der 5"most significant bits" des Port 10 gleich 1 ist oder irgendeines der3"least significant bits" gleich Ø ist. |

2. POKE, PEEK Mit diesen Befehlen können Werte direkt in den RAM-Speicher eingegeben oder aus dem Speicher gelesen werden. Der POKE-Befehl hat folgendes Format:

```
POKE <I,J>
```

wobei I und J Integer-Zahlen sind. POKE speichert das Byte J in der Speicherzelle, die durch I spezifiziert ist. I muss kleiner als 32768 sein. J muss im Bereich 0 bis 255 liegen. Daten können auch in den Bereich über 32768 gespeichert werden, wenn man I negativ macht. In diesem Falle wird I errechnet durch Adresse-65536

Um Daten in die Speicherzelle 45000 zu schreiben ist der Wert von I= 45000-65536=-20536. Man muß bei POKE-Befehlen gut aufpassen, daß man nicht die Bereiche verändert, in denen der BASIC-Interpreter Daten oder Programmtext abgelegt hat. Wenn BASIC nicht in ROM vorliegt muß man zusätzlich noch darauf achten, daß man nicht Teile des Interpreters überschreibt.
Um Daten aus dem RAM-Speicher zu lesen, benutzt man die PEEK - Anweisung. Das Format ist:

```
PEEK (I)
```

wobei I eine Integerzahl ist, die die Adresse angibt, von der ein Byte gelesen werden soll. I wird genauso gewählt, wie beim POKE-Befehl. Der erzeugte Wert ist eine Integerzahl zwischen 0 und 255. Eine häufige Anwendung für PEEK und POKE ist die Datenübergabe zwischen einem BASIC-Programm und einem Maschinenprogramm.

3. DOKE, DEEK Diese Befehle entsprechen POKE und PEEK, allerdings mit dem Unterschied, daß 16Bit-Zahlen in den Speicher übertragen und aus ihm gelesen werden. Der Wertebereich ist dabei: +32767 bis -32768 für J. I wird genauso errechnet wie bei PEEK und POKE. Das "least significant byte" wird in der Adresse I gespeichert, das "most significant byte" in der Adresse I+1. So können sehr einfach 16Bit-Zahlen an Maschinenprogramme übergeben werden.

4. OUT, INP Das Format des OUT-Befehles ist:

```
OUT <I,J>
```

wobei I und J Integerzahlen oder Integerausdrücke sind. OUT sendet ein Byte J an den Ausgabeport I. I und J müssen im Bereich 0 ... 255 liegen.

Die INP-Funktion hat folgende Syntax:

```
INP (I)
```

INP liest ein Byte vom Port I, wobei I eine Integerzahl im Bereich 0 ... 255 ist.

Beispiel:

```
20  IF INP(J) = 16 THEN PRINT "EINGESCHALTET"
```

## 3. Funktionen

NASCOM BASIC ermöglicht Funktionsaufrufe in mathematischer Notation. Das Format eines Funktionsaufrufes ist:

<Name> <Argument>

Dabei ist "Name" der Name einer bereits definierten Funktion und das Argument ein mathematisch/logischer Ausdruck. Es ist nur ein einziges Argument zulässig. Funktionsaufrufe können Teile von Ausdrücken sein, wie z.B.:

```
10  LET T=(F*SIN(T))/P
20  C=SQR(A↑2+B↑2+2*A*B*COS(T))
```

### 3.1. Standardfunktionen

NASCOM BASIC hat eine große Anzahl häufig benötigter Funktionen, die in Abschnitt 6-3 vorgestellt werden.

### 3.2 Benutzer-definierte Funktionen

#### a. Die DEF-Anweisung

Der Programmierer kann Funktionen vereinbaren, die nicht in der Tabelle unter 6.3 angegeben sind. Dies ist mithilfe der DEF-Anweisung möglich. Die DEF-Anweisung hat folgende Form:

DEF <Funktionsname>(<Variablenname>) = <Ausdruck>

wobei der Funktionsname mit "FN" beginnen muß, gefolgt von einem zulässigen Variablennamen und einem "Dummy"-Variablennamen. Der "Dummy"-Name steht für die Variable oder den Wert des Argumentes. Für eine Benutzer-definierte Funktion ist nur ein Variablenname zulässig. Auf der rechten Seite der Gleichung darf jeder zulässige Ausdruck auftreten, soweit er nicht länger als eine Zeile ist. Benutzer-definierte String-Funktionen sind nicht zulässig. Zwei Beispiele für zulässige Funktionsvereinbarungen:

```
10 DEF FNAVE(V,W) = (V+W)/2
12 DEF FNRAD (DEG) = 3.14159/180*DEG ; liefert zu einem
                     Winkel den zugehörigen Wert im Bogenmaß.
```

Eine Funktion kann im Programmablauf mehrfach umdefiniert werden. Bevor eine Funktion erstmalig aufgerufen wird, muß sie definiert sein und das Programm die entsprechende DEF-Anweisung abgearbeitet haben.

#### b. USR

Der USR-Funktion erlaubt den Aufruf von Maschinensprachprogrammen. Siehe dazu Anhang D.

### 3.3. Fehler

Ein FC-Fehler oder ILLEGAL FUNCTION CALL ERROR wird immer dann gemeldet, wenn eine Funktion nicht richtig aufgerufen wird (z.B.: war noch nicht definiert) oder die Funktion einen Fehler enthält. Einige Fehlermöglichkeiten:

1. Negativer Index eines Feldes; Z.B.: LET A(-1)= Ø
2. Ein Feldindex ist zu groß (größer 32767)
3. LOG-Argument =0 oder kleiner 0.
4. Negatives Argument für SQR
5. A↑B, wobei A negativ oder B nicht Integer (ganzzahlig).
6. Aufruf mit USR wobei keine Adresse angegeben wurde, die im Bereich der Maschinenprogramme liegt.

7. Fehlerhafte Argumente beim Aufruf der Funktionen MID$, LEFT$,RIGHT$, INP,OUT,WAIT,PEEK,POKE,TAB, SPC,INSTR,STRING$,SPACE$, oder ON ... GOTO

Beim Versuch, eine Funktion aufzurufen, die noch nicht definiert wurde, wird ein UF-ERROR oder UNDEFINED USER FUNCTION ERROR gemeldet.

Eine Fehlermeldung TM ERROR oder TYPE MISMATCH ERROR wird ausgegeben, wenn eine Funktion, die einen String verarbeitet mit einer Zahl aufgerufen wurde oder umgekehrt.

## 4. Zeichenketten ("STRINGS" )

In NASCOM BASIC kann ein Ausdruck entweder numerische Variablen oder Zeichenketten enthalten. BASIC bietet eine Anzahl Möglichkeiten, mit Strings zu arbeiten. Einige der Anweisungen wurden im Prinzip schon behandelt, sodaß hier nur kurz skizziert werden soll, wie diese Funktionen auf Zeichenketten angewendet werden können.

### 4.1 String-Daten

Ein String ist eine Zeichenkette von alphanumerischen Zeichen, die 0 .. 255 Zeichen lang sein kann. Strings können explizit geschrieben werden, oder es kann ein String einer Stringvariablen zugewiesen werden. String-Konstanten werden durch Hochkomma am Anfang und Ende eingegrenzt. Eine Stringvariable wird durch ein Dollarzeichen ($) am Ende des Variablennamens kenntlich gemacht:

```
A$ = "ABCD"  Der Stringvariablen A$ wird "ABCD" zugewiesen.
B9$= "14A/56" Der Stringvariablen B9$ wird die 6 Zeichen
              lange Zeichenkette : "14A/56" zugewiesen.
FOOFOO$=E$   Der Variablen FOOFOO$ wird der String "E$"
             zugewiesen.
```

Bei einer Eingabe, die durch eine INPUT-Anweisung verlangt wird, werden bei einem String die Hochkommata nicht angegeben.

String-Felder können wie andere Felder auch dimensioniert werden, indem man die DIM-Anweisung benutzt. Jedes Element eines Feldes ist ein String und kann maximal 255 Zeichen lang sein. Die gesamte Anzahl von Stringzeichen ist durch die Größe des vorhandenen Speicherplatzes begrenzt. Falls die Strings zuviel Platz beanspruchen, wird eine Fehlermeldung OS ERROR oder OUT OF STRING SPACE ERROR gemeldet. Nach der CLEAR-Anweisung (siehe 6.2) wird den Strings eines Programmes ihr Platz zugewiesen.

### 4.2 String-Operationen

#### a. Vergleichsoperationen

Auch für Strings sind Vergleichsoperationen möglich. Es sind die gleichen Anweisungen, wie für Zahlen:

```
=   gleich          <>  ungleich
>   größer als          kleiner als
=>, >=  größer oder <=, =<  kleiner oder gleich
        gleich
```

Die untersuchten Strings werden zeichenweise verglichen und zwar werden ihre ASCII-Codes verglichen. Falls sich beim Vergleich ergibt, daß ein String kürzer ist, so wird er behandelt, als wäre er eine kleinere Zahl. ASCII-Codes siehe Anhang A.

```
A<Z    A ist ASCII 065, Z ist ASCII 090
1<A    ASCII 1 ist 049
```

#### b. String-Ausdrücke

String-Ausdrücke setzen sich aus String-Literalen, String-Variablen und String-Funktionsaufrufen zusammen.Sie werden durch + oder einen Verknüpfungsoperator verbunden. Mit einem Verknüpfungsoperator wird der rechts vom Operator stehende Teil des String mit dem links von ihm stehenden String verbunden. Ergeben sich dabei mehr als 255 Zeichen, so erfolgt die Fehlermeldung LS ERROR oder STRING TOO LONG ERROR.

c. Eingabe/Ausgabe

Für die Eingabe und Ausgabe von Strings werden dieselben Befehle verwendet, wie für die Ein/Ausgabe von Zahlenwerten:

1. INPUT,PRINT Mit INPUT werden Strings vom Datensichtgerät gelesen, mit PRINT auf den Bildschirm geschrieben. Strings brauchen nicht in Hochkommata eingeschlossen zu werden. Läßt man die Hochkommata weg, so werden führende oder auf den String folgende Leerzeichen nicht mit ausgegeben. Außerdem wird der String nur bis zum ersten Komma oder Doppelpunkt verarbeitet. Beispiele:

```
10 INPUT ZOO$,FOO$          Liest zwei Strings ein.
20 INPUT X$                 Weist X$ den eingeg. String zu.
30 PRINT X$," HALLO, "      Gibt zwei Strings aus. Beim
                            zweiten String wird auch das
                            Leerzeichen und das Komma mit
                            ausgegeben.
```

2. DATA, READ Mit DATA und READ werden Strings genauso wie Zahlen behandelt. Es gelten die gleichen Formatkonventionen, wie oben erläutert.

## 4.3 String-Funktionen

Das Format der Standardfunktionen sieht für numerische und für String-Aufrufe gleich aus. Eine Liste der Funktionen findet man in Abschnitt 6.3. Die Namen von String-Funktionen müssen mit einem Dollarzeichen enden ($).

## 5. Zusätzliche Kommandos

NASCOM 8k BASIC verfügt über etliche Befehle, die man in einem 8k BASIC normalerweise nicht findet. Zusätzlich zu den unter 2.3 beschriebenen Funktionen DEEK und DOKE gibt es die Möglichkeit, den Monitor aufzurufen, auf beliebige Bildschirmstellen zu schreiben, zusätzliche Terminals und Drucker zu bedienen oder ein Graphik-Interface anzusteuern.

a. MONITOR Mit diesem Kommando wird die Steuerung dem Betriebssystem zurückgegeben. Die Kontrolle kann bei NAS-SYS mit dem J-Befehl an BASIC zurückgegeben werden oder mit Z, wenn auch alle Programme erhalten bleiben sollen. Falls Sie Breakpoints gesetzt haben, um Fehler in einem Maschinenprogramm aufzufinden, sollten Sie mit EFFFA oder EFFFD in den BASIC-Interpreter zurückspringen. Das Monitor-Kommando ist oft nützlich, wenn man mit einem Befehl XØ einen Drucker anschalten will.

b. WIDTH N Normalerweise wird angenommen, daß die Zeilen bei Eingabe und Ausgabe maximal 48 Zeichen lang sind. Dabei können sich allerdings Probleme ergeben, z.B. wenn man einen Drucker einsetzen will. WIDTH (N) ändert die vorgegebene Zeilenlänge auf den Wert N, der zwischen 1 - 255 liegen darf.

Bei der Ausgabe über den seriellen Port wird ein Zeilenvorschub nach N Zeichen automatisch erzeugt. Auf dem Bildschirm des NASCOM wird immer nach 48 Zeichen und zusätzlich nach N Zeichen ein Vorschub erzeugt.

Bei der Eingabe nimmt der Eingabepuffer N Zeichen oder bis zu 72 Zeichen auf, je nachdem, welcher Wert größer ist. Wenn Daten oder Programme mit NASBUG T2, T4 oder NAS-SYS im XO-Mode eingegeben werden, wird nach 48 Zeichen ein interner NEWLINE erzeugt. Wenn Daten oder Programme mit NAS-SYS im "normal mode" (Standardbetriebsart) eingegeben werden, dann ist die Zeilenlänge immer 48 Zeichen, unabhängig, welcher Wert für N angegeben wurde.

c. CLS Löscht den Bildschirm, unabhängig davon, welches Betriebssystem verwendet wird.

d. SCREEN X,Y setzt die Position des Cursors in die Spalte X, Zeile Y. X kann die Wert 1...48, Y die Werte 1...16 annehmen. Beachten Sie bitte, daß Zeile 16 die oberste Zeile ist, die nicht gescrollt wird. Zeile 1 ist die nächste Zeile darunter. Zeile 15 ist die unterste Zeile.
Die Befehle:

```
SCREEN  24,8
PRINT  "HALLO"
```

bewirken, daß "HALLO" in die Zeile 8 geschrieben wird, beginnend bei Spalte 24.

Beachten Sie bitte, daß mit NAS-SYS immer nur ein Zeichen bei einem Druckbefehl in die Zeile 16 eingetragen werden kann, weil beim Weiterrücken des Cursor, dieser immer wieder in die unterste Zeile gesetzt wird. Das in Anhang I beschriebene kurze Programm löst dieses Problem.

e. LINES N  Normalerweise werden bei einem LIST-Kommando immer 5 Zeilen ausgegeben. Dann wartet der Interpreter bis irgendeine Taste gedrückt wurde und gibt nochmal 5 Zeilen aus usw. Mit dem LIST-Kommando wird festgelegt, wieviele Zeilen auf einmal ausgegeben werden. Dies ist auch wichtig, um Programmausdrucke auf einen Drucker bringen zu können.

N muß im Bereich +32767 bis -32768 liegen. Negative Zahlen werden behandelt wie 65536 + N.

Die folgenden drei Kommandos werden für die einfachen Graphikoptionen des NASCOM 1 oder NASCOM 2 verwendet, wobei beim NASCOM 2 der Graphik-Zeichengenerator eingesetzt wird. Es können Punkte in einem Feld von 96 x 48 Punkten weiß oder schwarz geschaltet werden, wobei O,O die obere linke Ecke des Feldes ist.

| | | |
|---|---|---|
| f. | SET (X,Y) | Setze Punkt X,Y (weiß) |
| g. | RESET (X,Y) | Lösche Punkt X,Y (schwarz) |
| h. | POINT (X,Y) | Stelle fest, ob Punkt X,Y gesetzt ist.<br>1 heißt: Punkt gesetzt<br>O heißt: Punkt nicht gesetzt |

X und Y müssen Integerzahlen im Bereich O...95 bzw. O...47 sein. Punkte mit dem Y-Wert 45-47 erscheinen in der obersten Zeile (wird nicht gescrollt). Wenn Y im Bereich O...44 liegt, erscheint der entsprechende Punkt im gescrollten Bereich. Die Zeile, in der ein Punkt auftritt, wird folgendermaßen berechnet:

| | |
|---|---|
| L=INT(Y+3) + 1 | Zeile |
| C=INT(X+2) + 1 | Spalte |

Alle Zeichen, die nicht die Graphiksonderzeichen COH...FFH sein, werden mit dem SET-Befehl überschrieben,von RESET und POINT allerdings ignoriert.

1. Dateneingabe mit NAS-SYS  In der normalen Betriebsart von NAS-SYS wird nach dem Fragezeichen (?) ein Zeilenvorschub durchgeführt. Dann kann der Benutzer eine Zeile unter Zuhilfenahme der Editiermöglichkeiten mit NAS-SYS erstellen. Für einige Anwendungen, die eine komplexe Bildschirmaufteilung erfordern, ist dies nicht sinnvoll. Dann kann die Betriebsart gewechselt werden mit:

```
DOKE 4175,-6670
```

womit der Zeilenvorschub unterdrückt wird. Die Eingabezeile umfaßt dann die gesamte Zeile, also auch das Fragezeichen und alle anderen Zeichen, die in der Zeile stehen. Durch eine Stringmanipulation kann man diese störenden Zeichen abschneiden:

```
DOKE 4175,-6649
```

Mit dieser DOKE-Anweisung wird wie im normalen XO-Betrieb des NAS-SYS bei jedem Tastendruck ein Zeichen eingegeben und verarbeitet. Die NAS-SYS-Editiermöglichkeiten stehen in diesem Falle nicht zur Verfügung. Es wird kein Zeilenvorschub nach dem Fragezeichen ausgegeben.

Mit dem Befehl:

```
DOKE 4175,-25
```

wird wieder die normale Betriebsart hergestellt. Beachten Sie: Falls Sie bei den vorstehenden DOKE-Befehlen einen Fehler machen, kann dies einen "memory crash" bei der Eingabe provozieren. Die angegebenen Werte können sich bei späteren erweiterten BASIC-Versionen noch ändern.

j. DRUCKERANSCHLUSS Zusätzlich zu der Möglichkeit, Drucker oder zusätzliche Terminals an die Serienschnittstelle anzuschließen, gibt es die Möglichkeit, sie an die Parallelschnittstelle PIO anzuschließen. Programme, um solche Geräte anzuschließen, können mit DOKE und POKE-Anweisungen erstellt werden (siehe zugehöriges Betriebssystem-Handbuch).

k. PROGRAMME ANHALTEN ESCAPE (Shift und Newline gleichzeitig) dient dazu, laufende Programme anzuhalten (siehe 6.4). Wenn man einen nichtmaskierbaren Interrupt erzeugt, hat dies einen entsprechenden Effekt. Es ist jedoch nicht möglich, eine Cassetten-Schreiboperation oder Leseoperation auf diese Weise abzubrechen. In der Normalbetriebsart mit NAS-SYS ist es nicht möglich, die Zeileneingabe so zu unterbrechen, aber es ist möglich, RESET zu betätigen und dann den "Z"-Befehl einzugeben.

# 6. Befehlslisten

## 6.1 Kommandos

Kommandos dienen beim NASCOM BASIC in der Hauptsache für "Verwaltungsarbeiten". Dazu gehören Speicherorganisation, Organisation der Ein/Ausgabe, Listen und Editieren von Programmen und Daten usw. Nachdem NASCOM BASIC sein OK auf den Bildschirm geschrieben hat, befindet es sich auf der Kommandoebene und kann Kommandos annehmen. Die untenstehende Tabelle gibt eine Auflistung der Kommandos in alphabetischer Reihenfolge.

Kommando:

CLEAR Setzt alle Variablen auf Null. CLEAR [<Ausdruck>] Wenn man bei CLEAR einen Ausdruck angibt, (siehe 4.1) wird mit dem Wert des Ausdrucks festgelegt, wieviele Speicherplätze für Strings reserviert werden sollen. Wenn kein Ausdruck angegeben wird, bleibt die Größe des String-Speichers unverändert.

CLOAD <Stringausdruck> Bewirkt, daß das Programm auf der Cassette, dessen erstes Zeichen mit dem Stringausdruck übereinstimmt, in den Speicher geladen wird. Bevor das Programm geladen wird, wird automatisch ein NEW-Kommando ausgeführt.

CLOAD?<Stringausdruck> Verifiziert, daß das Programm fehlerfrei ist und ladbar.

CLOAD*<Feldname> Lädt das angegebene Feld von der Cassette. Dieses Kommando kann als Programmbefehl verwendet werden.

CONT Mit diesem Kommando wird ein Programm, das mit ESCAPE abgebrochen oder mit STOP oder END angehalten wurde, fortgeführt. Das Programm läuft an der Stelle weiter, wo es mit ESCAPE unterbrochen wurde. Wenn eine Eingabe unterbrochen wurde, wird das Programm dort fortgesetzt. CONT ist ein sehr wertvolles Hilfsmittel, um Programme zu unterbrechen, bei denen man vermuten muß, daß sie in einer Endlosschleife stecken. Wenn man ein Programm angehalten hat, kann man in der direkten Betriebsart z.B. Variablenwerte ausgeben lassen, um festzustellen, ob sie im zulässigen Bereich liegen etc. Dann kann das Programm nach dem CONT-Kommando weiterlaufen. Man kann es auch mit einem GOTO-Kommando wieder anlaufen lassen. Mit GOTO beginnt man die Programmausführung an einer Stelle, die man im GOTO-Kommando als Argument angibt. Falls nach dem Anhalten des Programmes in der direkten Betriebsart (Kommandoebene) ein Fehler gemacht wurde, kann das Programm nicht mit CONT fortgeführt werden. Die Ausführung kann auch nicht fortgeführt werden, wenn das Programm zwischendurch verändert wurde.

CSAVE<Stringausdruck> Das gerade im Speicher befindliche Programm wird auf Cassette unter dem Namen abgespeichert, den der erste Buchstabe des Stringausdruckes festlegt. (Der erste Buchstabe ist dieser Name).

CSAVE*<Feldname> Das Feld, das unter diesem Namen auf Cassette abgespeichert wurde, wird in den Speicher geladen. Dieses Kommando kann als Programmbefehl verwendet werden.

LIST Das gerade im Speicher stehende Programm wird, beginnend bei der niedrigsten Zeilennummer, auf dem Bildschirm ausgegeben. Listen wird mit ESCAPE abgebrochen.

LIST [<Zeilennummer>] Das Programm, das sich gerade im Speicher befindet, wird - beginnend bei der angegebenen Zeilennummer - aufgelistet. Es werden immer 5 Zeilen auf einmal ausgegeben (oder die mit dem LINES-Kommando festgelegte Anzahl) und dann wartet BASIC darauf, daß irgendeine Taste gedrückt wird. Sobald eine Taste gedrückt wird, werden nochmals 5 Zeilen ausgegeben usw. Mit ESCAPE kehrt BASIC in die Kommandobetriebsart zurück.

NEW Löscht das gerade im Speicher befindliche Programm und sämtliche Variablen. Wird verwendet, bevor man ein neues Programm eingibt.

NULL <Integerausdruck> Legt die Anzahl Dummyzeichen fest, die am Ende einer Zeile ausgegeben werden sollen. Für einen Lochstreifenstanzer, der 10 Zeichen pro Sekunde verarbeitet, sollte der Integerausdruck größer oder gleich 3 sein. Für einen Stanzer mit 30 Zeichen pro Sekunde Stanzgeschwindigkeit sollte er den gleichen Wert haben. Wenn keine Lochstreifen gestanzt werden, sollte der Wert 0 oder 1 für einen 8-Kanal-Fernschreiber sein, 2 oder 3 für einen Drucker mit 30 Zeichen pro Sekunde Druckgeschwindigkeit. Der Wert, der ohne NULL-Kommando für den Integerausdruck festgelegt ist, ist Ø. Die verschiedenen NASCOM Monitorsysteme ignorieren normalerweise alle Dummyzeichen (ØFFH) bei der Ausgabe. Es kann also nur eine Zeitverzögerung erreicht werden, wenn man diese Dummyzeichen (255D≙ ØFFH) einfügt. Die Dummyzeichen werden über die Benutzerschnittstellen mit ausgegeben.

RUN [<Zeilennummer>] Startet das gerade im Speicher befindliche Programm bei der angegebenen Zeilennummer. Wenn keine Zeilennummer angegeben wurde, wird das Programm bei der niedrigsten Zeilennummer gestartet.

## 6.2. Befehle

Die folgende Tabelle der Befehle ist in alphabetischer Reihenfolge angeordnet. In der Tabelle bezeichnen X und Y jeden Ausdruck, der in dem gegebenen Zusammenhang zulässig ist. I und J stehen für Ausdrücke, deren Werte Integerformat haben, oder als Integerzahlen behandelt oder in Integerzahlen umgeformt wurden. V und W stehen für die Namen irgendwelcher Variablen. Eine BASIC-Zeile in NASCOM BASIC hat folgendes Format:

<nnnnn><Befehl>[:<Befehl>....]

wobei nnnnn die Zeilennummer ist.

DATA DATA<Liste> Spezifiziert die Zahlen oder Zeichen, die später von einer READ-Anweisung gelesen werden sollen. Elemente der Liste können entweder Zahlen oder Strings sein. Die Listenelemente werden durch Kommata getrennt.

DEF DEF FNV(<W>)=<X> Mit dieser Anweisung wird eine Benutzer-definierte Funktion vereinbart. Der Name der Funktion ist FN, gefolgt von einer zulässigen Variablenbezeichnung (V). Die Definition darf nicht länger sein als eine Zeile (72 Zeichen).

DOKE DOKE<I>,<J> Speichert den Wert J in den Speicherzellen I und I+1. Wenn I oder J negative Werte annehmen, werden sie als 65536+I bzw. 65536+J interpretiert. DOKE ist somit ein 16-Bit-POKE-Befehl, mit dem ein Direktzugriff auf den Arbeitsspeicher möglich ist.

DIM DIM<V>(<I>)[,J...][....] Reserviert Speicherplatz für Feldvariablen. Es können soviele Variablen mit einer DIM-Anweisung

vereinbart werden, wie eine Zeile fassen kann. Der Wert jedes Ausdrucks gibt an, welches der größte zulässige Index für eine Feldvariable in einer Dimension ist. Der kleinstmögliche Index ist Ø. Wenn keine Dimensionierung vereinbart wurde, wird angenommen, daß die Feldvariable in jeder Dimension, die angegeben wird, maximal einen Index 10 haben darf. Für A(I,J) wird z.B. angenommen, daß A maximal 121 Elemente haben kann, von A(Ø,Ø) bis A(10,10), wenn A nicht explizit dimensioniert worden ist.

END  END  Schließt ein Programm ab.

FOR  FOR<V>=<X> TO <Y>[STEP <Z>]

Gibt die Möglichkeit, eine Gruppe von Befehlen mehrfach zu wiederholen. Bei der ersten Ausführung wird V=X gesetzt. Dann wird das Programm weiter ausgeführt, bis eine NEXT-Anweisung gefunden wird. Z wird zu V addiert. Wenn Z kleiner Null ist und V größer Y oder Z größer Null und V kleiner Y, dann springt BASIC zu dem nächsten Befehl, der hinter FOR steht und die Schleife wird nochmals abgearbeitet. Anderenfalls wird das Programm bei dem Befehl fortgeführt, der auf die NEXT-Anweisung folgt.

GOTO  GOTO <nnnnn>  Unbedingter Sprung zur Zeile nnnnn.

GOSUB  GOSUB<nnnnn>  Unterprogrammaufruf des Programmes, das bei Zeile nnnnn beginnt.

IF .. GOTO  IF <X> GOTO <nnnnn>  Genauso wie IF ... THEN, jedoch kann nur eine Zeilennummer angegeben werden, nicht aber ein Befehl, der ausgeführt werden soll, wenn die Bedingung erfüllt ist.

IF .. THEN  IF<X> THEN<X>  oder
IF<X>THEN <Befehl>[:<Befehl>...]

Wenn der Wert von X ungleich Ø ist, wird das Programm bei der angegebenen Zeilennummer weitergeführt. Bei der zweiten Variante wird/werden der/die angegebene/n Befehl/e ausgeführt. Falls die Bedingung nicht erfüllt ist, wird das Programm linear weitergeführt.

INPUT  INPUT <V>[,<W>]  Wenn BASIC diesen Befehl erreicht, erwartet es Eingaben vom Terminal. Die eingegebenen Werte werden den Variablen der Liste nacheinander zugewiesen.

LET  LET <V> =<X>  Weist der Variablen V den Wert des Ausdruckes X zu. Das Wörtchen LET kann weggelassen werden.

LINES  LINES <A>  Legt fest, wieviele Zeilen bei einem LIST-Befehl ausgegeben werden, bevor BASIC darauf wartet, daß irgendeine Taste gedrückt wird.

NEXT  NEXT[<V>,<W>...]  Letzter Befehl einer FOR-Schleife. V ist die Variable der innersten Schleife, W die der nächst äußeren Schleife usw. NEXT ohne Variablenangabe bezieht sich immer auf die nächstliegende Schleife.

ON...GOTO  ON <I> GOTO <Liste von Zeilennummern>

Verzweigt zu der Zeilennummer, die als I-te in der Liste angegeben ist. Listelemente werden durch Kommata abgetrennt. Wenn I=0 oder größer der Anzahl Elemente der Liste ist, wird das Programm bei nächsten Befehl fortgeführt. Wenn I kleiner als Null oder größer 255 ist, wird eine Fehlermeldung ausgegeben.

ON...GOSUB  ON <I>GOSUB <Liste>  Genauso wie ON...GOTO, jedoch werden die Anfangszeilennummer der anzuspringenden Unterprogramme angegeben.

OUT OUT<I>,<J> Schickt das Byte J zum Port I. I muß größer oder gleich Null sein, J kleiner gleich 255.

POKE POKE<I>,<J> Speichert das Byte J in der Speicherzelle, die I angibt. J muß zwischen Ø und 255 liegen, I zwischen -32768 und +65536. Wenn I negativ ist, wird die Adresse berechnet als 65536+I, wenn I positiv ist, ist Adresse=I.

PRINT PRINT<X>[,<Y>...] Veranlaßt die Ausgabe der Variablen, die in der Liste angegeben werden. Die Formataufteilung erfolgt mit diesen Zeichen:

, nächstes Zeichen wird zu Beginn eines neuen 14-Zeichen-Feldes ausgegeben.

; kein Zwischenraum zum vorhergehenden Zeichen.

Wenn keine Formatzeichen angegeben wurden, beginnt die nächste Ausgabe in einer neuen Zeile.
String-Literale werden in Hochkommata eingeschlossen (").

READ READ <V>[,<W>...] Weist den Variablen der Variablenliste die in der DATA-Anweisung vereinbarten Werte zu. Dabei werden nacheinander die vereinbarten Werte den angegebenen Variablen zugewiesen, unabhängig davon, wo DATA und READ-Anweisungen im Programm stehen. Ausschließlich ausschlaggebend ist die Reihenfolge der DATA-Anweisungen.

REM REM[<Bemerkung>] Mit der REM-Anweisung werden Bemerkungen in ein Programm eingeflochten. REM-Anweisungen können angesprungen werden, werden jedoch nicht ausgeführt, sondern übersprungen.

RESTORE RESTORE Nach der RESTORE-Anweisung können die mit den DATA-Anweisungen vereinbarten Werte neu gelesen werden. Die nächste auf eine RESTORE-Anweisung folgende READ-Anweisung beginnt mit dem Auslesen des ersten Zeichens der ersten DATA-Anweisung.

RETURN RETURN Abschluss eines Unterprogrammes. BASIC arbeitet bei dem nächsten Befehl nach der GOSUB-Anweisung weiter, die die Subroutine aufgerufen hat.

SCREEN SCREEN <X>,<Y> Setzt den Cursor in Spalte X der Zeile Y des Bildschirmes.

STOP STOP Hält ein BASIC-Programm an. BASIC befindet sich nach dem STOP in der Kommandobetriebsart und gibt aus: BREAK IN LINE nnnnn.

WAIT WAIT <I>,<J>[,<K>] Der Zustand des Port I wird Exclusiv-ODER-verknüpft mit K und UND-verknüpft mit J. Wenn das Ergebnis der Verknüpfung nicht Null ist, wird das Programm fortgesetzt. Der Wert von K ist ohne gesonderte Angabe immer Null. Die Werte für I,J,K müssen zwischen Ø und 255 liegen.

WIDTH WIDTH <n> Legt die Länge des Eingabepuffers auf den Wert n fest. Siehe auch 5b.

## 6.3. Standardfunktionen

NASCOM BASIC bietet eine Reihe von Standardfunktionen für algebraische Zwecke und für String-Funktionen. Diese Standardfunktionen können ohne weitere Definition aufgerufen werden. In der folgenden Liste sind die Standardfunktionen angegeben. Normalerweise setzt man für X und Y einen Ausdruck ein, für I und J einen Integerausdruck und für X$ und Y$ einen Stringausdruck.

ABS ABS (X) Liefert den Absolutwert von X. ABS(X)=X wenn X>=Ø; ABS(X)=-X, wenn X<Ø

ASC ASC (X$) Der ASCII-CODE des ersten Zeichens im String X$ wird ermittelt. ASCII-Codes sind in Anhang A zusammengestellt.

ATN ATN(X) Liefert den Wert des Arcustangens von X. Das Resultat liegt im Bogenmaß zwischen - π/2 und + π/2

CHR$ CHR$(I) Dem String wird das ASCII-Zeichen zugewiesen, das dem Wert von I entspricht. ASCII-Codes siehe Anhang A.

COS COS(X) Liefert den Wert des Cosinus von X. X wird im Bogenmaß angegeben.

EXP EXP(X) Liefert den Wert $e^X$. X <= 87,3365

FRE FRE(O) Gibt die Anzahl Bytes an, die nicht von BASIC verwendet werden. Wenn das Argument ein String ist, wird die Anzahl freier Bytes im String-Speicherbereich angegeben.

INP INP(I) Liest ein Byte vom Port mit der Bezeichnung I ein.

INT INT(X) Gibt den Wert der größten Integerzahl <=X an. Es wird also X gerundet.

LEFT$ LEFT$(X$,I) Diese Funktion liefert die von links nach rechts gezählt ersten I Zeichen des String X$.

LEN LEN(X$) Gibt die Länge des String X$ an. Sonderzeichen und Zwischenräume werden mitgezählt.

LOG LOG(X) Liefert den natürlichen Logarithmus von X. Es muß X>Ø sein.

MID$ MID$(X$,I[,J]) Wenn man keinen Wert für J angibt, liefert diese Funktion die Zeichen ab dem I-ten Zeichen, von links nach rechts gezählt. Wenn I>LEN(X$), dann liefert MID$ einen inhaltslosen String. I muß zwischen Ø und 255 liegen. Wenn drei Argumente angegeben werden, beginnt MID$ bei dem I-ten Zeichen zu zählen und liefert J Zeichen, beginnend bei dem I-ten Zeichen von X$. Wenn J größer ist als die Anzahl Zeichen, die der String von I ab gezählt aufweist, dann liefert MID$ alle Zeichen bis Stringende. J liegt ebenfalls zwischen Ø und 255.

POS POS(I) Gibt an, wo der Druckkopf eines extern angeschlossenen Druckers gerade steht. Position links außen=Ø.

RND RND(X) Liefert eine Zufallszahl zwischen O und 1. Wenn X kleiner Ø ist, wird eine neue Folge von Zufallszahlen begonnen. Wenn X=Ø wird der Wert der letzten Zufallszahl nochmal angegeben. Wenn X größer O ist, wird die nächste Zufallszahl einer Zufallszahlenreihe ausgegeben. Die Funktion erzeugt für die gleichen negativen Zahlen auch immer die gleichen Reihen von Zufallszahlen.

RIGHT$ RIGHT$(X$,I) Liefert die ersten I Zeichen des String X$, gezählt von rechts nach links. Wenn die Länge des Strings gleich X$ ist, wird X$ angegeben.

SGN SGN(X) Signum-Funktion. Wenn X größer Ø, dann liefert die Funktion 1, wenn X kleiner Ø,dann liefert sie -1. Beispiel: ON SGN(X)+2 GOTO 100,200,300 verzweigt zu 100, wenn X negativ ist, zu 200 wenn X=Ø ist und zu 300 wenn X positiv ist.

SIN SIN(X) Liefert den Wert des Sinus, wobei X im Bogenmaß angegeben wird. COS(X)=SIN(X+3.14159/2).

SPC SPC(I) Schreibt eine Anzahl von I Zwischenräumen auf das Terminal. I muß zwischen Ø und 255 liegen.

SQR SQR(X) Gibt den Wert der Quadratwurzel von X an. X muß größer oder gleich Ø sein.

STR$ STR$(X) Formt den Wert von X in einen String um.

TAB TAB(I) Tabulatorfunktion. Führt soviele Zwischenräume aus, daß der Druckkopf (Cursor) auf Position I steht. Ø ist ganz links, 71 ganz rechts. Wenn der Cursor schon hinter Position I steht, hat die Funktion keine Wirkung. I muß zwischen Ø und 255 liegen. TAB nur in PRINT-Anweisungen verwenden !!

TAN TAN(X) Liefert den Wert des Tangens von X.
X wird im Bogenmaß angegeben.
USR USR(X) Ruft ein Maschinenprogramm des Benutzers
mit dem Argument X auf.
VAL VAL(X$) Liefert den numerischen Wert des String X$.
Wenn das erste Zeichen in X$ nicht +,-,& oder eine Zahl ist,
dann ist VAL(X$)=Ø.

## 6.4. Sonderzeichen

NASCOM BASIC erkennt eine Anzahl Sonderzeichen aus dem ASCII-Zeichensatz, die Sonderfunktionen haben. Dazu gehören z.B. Cursorsteuerung, Editieren und Programmunterbrechungen. Zeichen wie CONTROL C , CONTROL S etc. werden erzeugt, indem man zunächst die CONTROL-Taste betätigt und anschließend den zugehörigen Buchstaben eingibt. Diese Sonderzeichen werden weiter unten aufgeführt. Die Zeichen, die mit einem Sternchen markiert sind, können in der normalen Betriebsart des NAS-SYS nicht eingegeben werden.

*ESCAPE oder SHIFT+NEWLINE bei NAS-SYS (@)
Löscht das letzte eingegebene Zeichen. Wenn kein Zeichen eingegeben wurde, führt es einen CR durch.

*_ (wie Unterstreichung oder Cursorzeichen)
Selbe Funktion wie BACKSPACE.

NEWLINE (oder ENTER)
Cursor wird eine Zeile vorgeschoben und auf den Anfang der Zeile positioniert.

ESCAPE (Shift+NEWLINE)
Unterbricht das gerade laufende Programm oder das LIST-Kommando. Das Programm wird nach dem gerade in Abarbeitung befindlichen Befehl unterbrochen. BASIC kehrt in den Kommandomodus zurück und gibt OK aus. Mit CONT wird das Programm fortgesetzt. Siehe dazu auch 6.1.

Beachten Sie bitte: Diese Tasten müssen immer auf der Tastatur des NASCOM betätigt werden. Die Eingabe eines ESC auf einem externen Terminal hat keine Wirkung.

:
Trennt die Befehle, die in einer Zeile stehen.

CONTROL O
Unterdrückt alle Ausgaben, bis eine Eingabeanweisung (INPUT) erreicht oder nochmals CONTROL O gedrückt wird. Die Ausgabe wird auch wieder eingeschaltet, wenn ein Fehler auftritt oder BASIC auf die Kommandoebene zurückkehrt.

?
Entspricht dem PRINT-Befehl

*RUBOUT (CONTROL Z)
Löscht das vorhergehende Zeichen einer Eingabezeile. Das erste Rubout löscht das letzte Zeichen, das nächste Rubout das nächste Zeichen usw. Die gelöschten Zeilen erscheinen also in der umgekehrten Reihenfolge, die sie in dem gelöschten Wort hatten. Wenn man eine Taste drückt, wird das zugehörige Zeichen an der entsprechenden Stelle in den Text eingefügt.

*CONTROL R
Gibt einen CR aus und gibt die eingegebene Zeile nochmals aus.

CONTROL U
Entspricht (a) (Löschen der gerade eingegebenen Zeile)

Eingabe von Kleinbuchstaben
Kleinbuchstaben werden bei der Eingabe auch immer als Kleinbuchstaben dargestellt. LIST und PRINT wandeln Kleinbuchstaben wieder in Großbuchstaben um, wenn sie nicht Teil eines Stringliterals oder einer REM oder (')-geführten Anweisung sind.

## 6.5. Fehlermeldungen

Nachdem ein Fehler aufgetreten ist, kehrt BASIC auf die Kommandoebene zurück und gibt OK aus. Werte von Variablen und der Programmtext bleiben erhalten, aber das Programm kann nicht mit dem CONT-Befehl zum Weiterlaufen gebracht werden. Der Zusammenhang aller GOSUB- und FOR-Anweisungen wird erst beim neuen Programmstart wieder hergestellt. Das Programm kann aber mithilfe einer GOTO-Anweisung in der direkten Betriebsart fortgeführt werden. Wenn in der direkten Betriebsart (Kommandoebene) ein Fehler auftritt, wird keine Zeilennummer mit angezeigt. Die Fehlermeldung hat folgendes Format:

Direkte Betriebsart: ?XX ERROR
Indirekte Betriebsart: ?XX ERROR IN YYYYY

wobei XX der Fehlercode ist und YYYYY die Zeilennummer der Zeile, in der der Fehler aufgetreten ist. Die folgenden Beispiele zeigen mögliche Fehlermeldungen mit ihren Codes und deren Bedeutung:

| Fehlercode | Fehlermeldung | Zahl |
|---|---|---|
| BS | SUBSCRIPT OUT OF RANGE (Index zu groß) | 9 |

In diesem Falle wurde ein Feldelement aufgerufen, das außerhalb des dimensionierten Bereiches liegt. Ein solcher Fehler kann z.B. auftreten, wenn eine falsche Anzahl Dimensionen in einem Feldaufruf angegeben werden. Z.B.:
LET A(1,1,1)=Z
wobei A schon weiter oben als A(10,10) dimensioniert wurde.

| | | |
|---|---|---|
| DD | REDIMENSIONED ARRAY (Feld mehrfach dimensioniert) | 10 |

Nachdem ein Feld bereits dimensioniert worden ist, wurde nochmals eine DIM-Anweisung für das gleich Feld gefunden. Dieser Fehler tritt häufig dann auf, wenn ein Feld bereits angesprochen wurde (dann wird automatisch für jede Dimension 10 eingesetzt) und dann das Feld anschließend dimensioniert wurde.

| | | |
|---|---|---|
| FC | ILLEGAL FUNCTION CALL (Unzulässiger Funktionsaufruf) | 5 |

Der Parameter, der einer math. oder einer Stringfunktion übergeben wurde, war außerhalb des erlaubten Bereiches. FC-Fehlermeldung kann auftreten wenn:

1. Ein negativer Index in einem Feldaufruf auftritt: A(-1)=Ø.
2. Ein Feldindex größer 32767 auftritt.
3. LOG mit negativem Argument oder Argument=Ø auftritt.
4. SQR mit negativem Argument auftritt.
5. Aufruf eines Unterprogrammes, bevor die Adresse der Maschinenroutine festgelegt wurde (nur bei USR).
6. Aufrufe der Funktionen MID$,LEFT$,RIGHT$,INP,OUT, SCREEN,WIDTH,SET,RESET,POINT,DEEK,DOKE,PEEK,POKE, TAB, SPC, STRING$,SPACE$,INSTR oder ON...GOTO mit fehlerhaftem Argument.

| | | |
|---|---|---|
| ID | ILLEGAL DIRECT (Fehlerhafte Eingabe im Direktbetrieb) | 12 |

INPUT und DEF sind im Direktbetrieb unzulässig.

| | | |
|---|---|---|
| NF | NEXT WITHOUT FOR (NEXT und FOR passen nicht zusammen) | 1 |

Die Variable in einer NEXT-Anweisung passt nicht zu einer schon begonnenen FOR-Anweisung.

OD OUT OF DATA 4
Es wurde eine READ-Anweisung erreicht, doch sind nicht mehr genug Daten angelegt worden, um die READ-Anweisung ausführen zu können.

OM OUT OF MEMORY 7
Entweder ist das Programm zu lang, es hat zuviele Variablen, zuviele FOR-Schleifen, zuviele GOSUBs oder zu komplexe Ausdrücke, zu deren Abarbeitung der Speicherplatz nicht ausreicht. Siehe dazu Anhang C.

OV OVERFLOW 6
Das Ergebnis einer Berechnung war zu groß, um im Zahlenformat von BASIC dargestellt zu werden. Wenn die Zahl zu klein wird, weist BASIC ihr Ø als Ergebnis zu und das Programm wird fortgeführt ohne daß eine Fehlermeldung auftritt.

SN SYNTAX ERROR 3
Fehlende Klammern, unzulässige Zeichen in einer Zeile, fehlerhafte Zeichensetzung ( syntaktischer Fehler ).

RG RETURN WITHOUT GOSUB 3
Es wurde eine RETURN-Anweisung durchlaufen, bevor eine GOSUB-Anweisung erreicht wurde.

UL UNDEFINED LINE 8
Bei einer GOTO, GOSUB,IF...THEN...ELSE oder DELETE-Anweisung wurde eine nicht existente Zeilennummer angegeben.

/O DIVISION BY ZERO 11
Kann bei einer Integerdivision und bei MOD genauso auftreten wie bei Fließkommadivision. $0^x$ mit negativem x gibt auch einen /O-Fehler.

CN CAN'T CONTINUE 17
Wenn man ein Programm startet, obwohl überhaupt kein Programm im Speicher ist, dann wird diese Fehlermeldung ausgegeben. Wird auch ausgegeben, wenn ein Fehler auftrat oder ein Programm verändert wurde, das mit CONT fortgesetzt werden soll.

LS STRING TOO LONG 15
Wird ausgegeben, wenn der Versuch unternommen wurde, einen String zu erzeugen, der mehr als 255 Zeichen hat.

OS OUT OF STRING SPACE 14
Der Speicherplatz, der für die Strings angelegt wurde, ist kleiner als der Bereich, den das Programm belegen möchte. Wenn Sie eine CLEAR-Anweisung verwenden, um String-Speicherplatz zu gewinnen oder weniger Stringvariablen und kürzere Strings verwenden, kann das Problem behoben werden.

ST STRING FORMULA TOO COMPLEX 16
Ein Stringausdruck war zu lang oder zu komplex. Teilen Sie ihn in zwei kleinere Ausdrücke auf.

TM TYPE MISMATCH 13
Auf der einen Seite einer Gleichung stand ein String, auf der anderen Seite eine Variable. Vielleicht wurde auch einer Funktion, die einen String erwartet eine Zahl übergeben oder umgekehrt.

UF UNDEFINED USER FUNCTION 18
Es wurde eine Funktion aufgerufen, die der Benutzer noch garnicht definiert hatte.

## 6.6. Reservierte Worte

Es gibt eine Anzahl von Worten in BASIC, die reserviert sind. Sie dürfen nicht für die Namen von Variablen oder Funktionen verwendet werden. Untenstehend sind diese reservierten Worte aufgelistet. Zusätzlich dazu sind die Namen der Standard-funktionen ebenfalls reserviert.

| | | | |
|---|---|---|---|
| CLEAR | NEW | AND | OUT |
| DATA | NEXT | CONT | POINT |
| DIM | PRINT | DEF | RESET |
| END | READ | DOKE | POKE |
| FOR | REM | FN | SCREEN |
| GOSUB | RETURN | LINES | SET |
| GOTO | RUN | NOT | SPC |
| IF | STOP | NULL | WAIT |
| INPUT | TO | ON | WIDTH |
| LET | TAB | OR | |
| LIST | THEN | | |
| | USR | | |

## 7. Praktische Arbeit mit dem BASIC

BASIC ist erhältlich als 8K Byte ROM oder in 8 EPROMs, die 1k x 8 organisiert sind. Die EPROMs liegen im Adreßbereich EØØØH - FFFFH

Die folgenden Einsprungadresse sind bei BASIC von Wichtigkeit:

a) EØØØH Start des Interpreters (erstmaliger Start), mit dem Befehl EEØØØ. Bei NAS-SYS wird mit diesem Befehl gleichzeitig der Monitor zurückgesetzt.

b) FFFAH Normaler "Kaltstart". Bei NAS-SYS wird mit dem Befehl J zu FFFA gesprungen, bei NASBUG mit EFFFA, wobei kein Monitor-Reset erfolgt.

c) FFFD "Warmstart" (Entspricht Z bei NAS-SYS). Wenn schon Programme im Speicher sind und diese erhalten bleiben sollen, dann verwenden Sie diesen Einsprung-punkt. BASIC muß aber schon vorher mit a) oder b) gestartet worden sein.

Wenn BASIC gestartet wurde, gibt der Interpreter aus:

Memory Size?

Dann können Sie eingeben:

a) Ein NEWLINE oder ENTER-Zeichen, wonach BASIC allen Speicherplatz oberhalb von 1OOOH belegen kann, oder

b) die dezimal dekodierte Adresse, die BASIC noch mit-verwenden darf. Auf diese Weise können Sie sich Speicher-plätze für Ihre Maschinenprogramme freihalten.

Wenn BASIC erfolgreich gestartet wurde, wird ausgegeben:

```
NASCOM ROM BASIC Ver 4.7
Copyright (c) 1978 by Microsoft
(n)  Bytes free
```

wobei (n) die Anzahl der Speicherplätze ist, die noch nicht von Programm und/oder Daten belegt sind. BASIC steht nun auf Kommandoebene und erwartet Ihre Eingaben.

## Anhang A ASCII-Zeichentabelle

| Dezimal | Zeichen | Dezimal | Zeichen | Dezimal | Zeichen |
|---|---|---|---|---|---|
| 000 | NUL | 043 | + | 086 | V |
| 001 | SOH | 044 | , | 087 | W |
| 002 | STX | 045 | - | 088 | X |
| 003 | ETX | 046 | . | 089 | Y |
| 004 | EOT | 047 | / | 090 | Z |
| 005 | ENQ | 048 | 0 | 091 | [ |
| 006 | ACK | 049 | 1 | 092 | \ |
| 007 | BEL | 050 | 2 | 093 | ] |
| 008 | BS | 051 | 3 | 094 | ^ |
| 009 | HT | 052 | 4 | 095 | < |
| 010 | LF | 053 | 5 | 096 | ' |
| 011 | VT | 054 | 6 | 097 | a |
| 012 | FF | 055 | 7 | 098 | b |
| 013 | CR | 056 | 8 | 099 | c |
| 014 | SO | 057 | 9 | 100 | d |
| 015 | SI | 058 | : | 101 | e |
| 016 | DLE | 059 | ; | 102 | f |
| 017 | DC1 | 060 | < | 103 | g |
| 018 | DC2 | 061 | = | 104 | h |
| 019 | DC3 | 062 | > | 105 | i |
| 020 | DC4 | 063 | ? | 106 | j |
| 021 | NAK | 064 | @ | 107 | k |
| 022 | SYN | 065 | A | 108 | l |
| 023 | ETB | 066 | B | 109 | m |
| 024 | CAN | 067 | C | 110 | n |
| 025 | EM | 068 | D | 111 | o |
| 026 | SUB | 069 | E | 112 | p |
| 027 | ESCAPE | 070 | F | 113 | q |
| 028 | FS | 071 | G | 114 | r |
| 029 | GS | 072 | H | 115 | s |
| 030 | RS | 073 | I | 116 | t |
| 031 | US | 074 | J | 117 | u |
| 032 | SPACE | 075 | K | 118 | v |
| 033 | ! | 076 | L | 119 | w |
| 034 | " | 077 | M | 120 | x |
| 035 | = | 078 | N | 121 | y |
| 036 | $ | 079 | O | 122 | z |
| 037 | % | 080 | P | 123 | { |
| 038 | & | 081 | Q | 124 | \| |
| 039 | ' | 082 | R | 125 | } |
| 040 | ( | 083 | S | 126 | ~ |
| 041 | ) | 084 | T | 127 | DEL |
| 042 | * | 085 | U | | |

LF=Line Feed FF=Form Feed CR=Carriage Return DEL=Rubout

Zeilenvorschub Formularvorschub Wagenrücklauf Zeichen löschen

Verwendung der CHR$-Funktion.
CHR$(X) liefert einen String, in dem das Zeichen steht, das dem ASCII-Code X entspricht. ASC(X$) wandelt das erste Zeichen des String X$ in den zugeordneten Dezimalwert um.

Eine der gebräuchlichsten Anwendungen für CHR$ ist es, ein Sonderzeichen an das Benutzer-Terminal zu senden. Häufig ist z.B. das Zeichen BEL (Klingel: ASCII 007). Wenn man dieses Zeichen aussendet, wird bei einigen Fernschreibern eine Klingel angeschlagen oder ein Terminal erzeugt einen Piepton. Beispiel:

```
PRINT CHR$(7).
```

Man kann diese Möglichkeit z.B. ausnutzen, um Fehlermeldungen akustisch anzuzeigen.

Eine Anwendung ist auch das Positionieren des Cursors auf dem Bildschirm bei Bildschirmgeräten, die diese Möglichkeit bieten. Auch für Steuerfunktionen ("Drucker ein") ist die Funktion recht nützlich.

Einige Terminals können mithilfe von Sonderzeichen Graphik erstellen. Auch diese Möglichkeiten kann man dank der CHR$-Funktion nutzen.

## Anhang B Hinweise zu Speicherplatzaufteilung und Verarbeitungsgeschwindigkeit

### A. Speicherplatzaufteilung

Wieviel Speicherplatz ein Programm benötigt, hängt von einer ganzen Anzahl verschiedenster Faktoren ab. Die folgende Tabelle vermittelt einen Eindruck davon, wieviele Byte wofür gebraucht werden:

| Element | Benötigter Speicherplatz |
|---|---|
| numerische Variable | 6 Byte |
| Feld; String oder Fließkommaelemente im Feld. | (Anzahl der Feldelemente)*6 + 5 + (Anzahl der Dimensionen)*2 Byte |
| Funktionen: | |
| Standardfunktion | 1 Byte für den Aufruf |
| Benutzer-definiert | 6 Byte für die Definition |
| Reservierte Worte | 1 Byte |
| Andere Zeichen | 1 Byte |
| Stack-Platzbedarf: | |
| Aktive FOR-Schleife | 16 Byte |
| Aktives GOSUB | 5 Byte |
| Klammern | 6 Byte pro Klammernpaar |
| Ergebnis | 10 Byte |

BASIC selbst belegt 8k ROM.

### B. Hinweise zum Einsparen von Speicherplatz

Wenn einige einfache Regeln bei der Programmerstellung berücksichtigt werden, kann man einigen Speicherplatz sparen, ohne daß Funktion oder Komfort des Programmes beeinträchtigt werden:

1. Bringen Sie nach Möglichkeit mehrere Befehle in einer Zeile unter. Für jede Zeile belegt die Zeilennummer 5 Byte. Je weniger Zeilen ein Programm hat, desto weniger Speicherplatz braucht es.
2. Lassen Sie unnötige Zwischenräume weg. So können Sie statt
   ```
   10 PRINT X,Y,Z
   ```
   genauso gut schreiben:
   ```
   10 PRINTX,Y,Z
   ```
3. Lassen Sie REM-Anweisungen weg, wenn nicht unbedingt nötig. Jede REM-Anweisung braucht ein Byte für "REM" und ein Byte pro Zeichen.
4. Verwenden Sie Variablen anstatt Konstanten, insbesondere wenn ein Wert mehrfach gebraucht wird. Es ist viel günstiger, statt 3.14159 zehn Mal in ein Programm zu schreiben, eine Variable P zu definieren: P=3.14159 und diese zehn Mal zu verwenden.
5. END ist zum Abschluss eines Programmes nicht nötig und braucht ein Byte extra.
6. Wenn Sie eine Variable nicht mehr brauchen, verwenden Sie den gleichen Variablennamen für einen anderen Zweck weiter.

7. Verwenden Sie Unterprogramme, anstatt Programmteile zu wiederholen.
8. Verwenden Sie auch die mit Ø indizierten Elemente eines Feldes. Bedenken Sie, daß ein Feld , das mit 100 DIM A(10) dimensioniert wurde, insgesamt 11 Elemente hat,nämlich A(Ø)...A(10)

C. Beschleunigung der Verarbeitung

1. Lassen Sie unnötige Zwischenräume weg, ebenso REM-Anweisungen.
2. Sobald eine Variable in einem Programm verwendet wird, wird sie in eine Tabelle eingetragen. Später sucht BASIC diese Tabelle ab, um die Variable zu finden und ihr einen Wert zuweisen zu können. Variablen, die oben in der Tabelle stehen, werden viel schneller gefunden, als Variablen, die unten in der Tabelle stehen. Daher: Verwenden Sie Variablennamen mehrfach, soweit das Programm dies zuläßt und versuchen Sie, mit möglichst wenigen Variablen auszukommen.
3. Verwenden Sie NEXT ohne Indexvariable.
4. Verwenden Sie Variablen anstatt Konstanten, insbesondere in Schleifen, die häufig durchlaufen werden müssen.
5. Stringvariablen definieren einen "Descriptor", der angibt, wie lange ein String ist und wo das erste Zeichen des String im Speicher steht. Wenn mit Strings gearbeitet wird, füllt sich der String-Speicherbereich mit Zwischenergebnissen und anderen Informationen, ebenso wie mit dem gewünschten Ergebnis. Wenn dieser Fall auftritt, läuft ein Programm an, das den bei einer Operation entstandenen "Schrott" aufsammelt und vernichtet. Die häufige Referenz von Stringbereichen provoziert diese Operation. Die Zeit, die zum „Reinigen" des Speicherbereiches gebraucht wird, ist proportional zum Quadrat der Anzahl von Stringvariablen. Um die Zeit für das „Reinigen" des Speicherbereiches zu minimieren, sollte man den String-Speicherbereich so groß wie möglich und die Anzahl der Stringvariablen zu klein wie möglich wählen.

## Anhang C Mathematische Funktionen

### 1. Abgeleitete Funktionen

Folgende Funktionen sind keine NASCOM BASIC-Standardfunktionen, können aber sehr leicht definiert werden:

```
SECANT                              SEC(X) = 1/COS(X)
COSECANT                            CSC(X) = 1/SIN(X)
COTANGENT                           COT(X) = 1/TAN(X)
INVERSE SINE                        ARCSIN(X) = ATN(X/SQR(-X*X+1))
INVERSE COSINE                      ARCCOS(X) = -ATN X(X/SQR(-X*X+1))
                                        +1.5708
INVERSE SECANT                      ARCSEC(X) = ATN(XSQR(X*X-1))
                                        +SGN(SGN(X)-1)*1.5708
INVERSE COSECANT                    ARCCSC(X) = ATN(1/SQR(X*X-1))
                                        +(SGN(X)-1)*1.5708
INVERSE COTANGENT                   ARCCOT(X) = ATN(X)+1.5708
HYPERBOLIC SINE                     SINH(X) = (EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE                   COSH(X) = (EXP(X)+EXP(-X))/2
HYPERBOLIC TANGENT                  TANH(X) = EXP(-X)/EXP(X)+EXP(-X))
                                        *2+1
HYPERBOLIC SECANT                   SECH(X) = 2/(EXP(X)+EXP(=X))
HYPERBOLIC COSECANT                 CSCH(X) = 2/(EXP(X)-EXP(-X))
HYPERBOLIC COTANGENT.               COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))
                                        *2+1
INVERSE HYPERBOLIC SINE             ARCSINH(X) = LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE           ARCCOSH(X) = LOG(X+SQR(X*X+-1))
INVERSE HYPERBOLIC TANGENT          ARCTANH(X) = LOG((1+X)/(1-X))/2
INVERSE HYPERBOLIC SECANT           ARCSECH(X) = LOG((SQR(-X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT         ARCCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X
INVERSE HYPERBOLIC COTANGENT        ARCCOTH(X) = LOG((X+1)/(X-1))/2
```

## Anhang D BASIC und ASSEMBLER-Sprache

NASCOM BASIC bietet die Möglichkeit Assemblerprogramme (Maschinenprogramme) von einem BASIC-Programm aus mit USR genauso wie ein BASIC-Unterprogramm aufzurufen.

Als ersten Schritt beim Koppeln von BASIC und Maschinenprogrammen ist es wichtig, für das Maschinenprogramm Speicherplatz freizuhalten. Wenn BASIC fragt "MEMORY SIZE?", dann sollte man berücksichtigen, wieviel Speicherplatz man für das Maschinenprogramm braucht und entsprechend weniger Speicherplatz angeben. BASIC verwendet ansonsten ab 4096 alle verfügbaren Speicherplätze, die es finden kann. Die Speicherzellen C5Ø-1000, die der Monitor nicht verwendet, können ebenfalls für Maschinenprogramme verwendet werden. Wenn die Antwort auf die Frage MEMORY SIZE? nicht befriedigend ausfällt (zu wenig Speicher), dann fragt BASIC solange, bis es den erforderlichen Speicherplatz zugewiesen bekommt.

Das Maschinenprogramm kann wie gewohnt in den Speicher geladen werden, oder von BASIC aus mit POKE-Befehlen.

Die Startadresse des Maschinenprogrammes wird in die Speicherzelle USRLOC eingetragen, deren MSB die Speicherzelle 1005H und LSB die Speicherzelle 1004 belegt. Die Funktion USR ruft das Maschinenprogramm auf, dessen Adresse in USRLOC steht. Anfänglich steht in USRLOC die Adresse von ILLFUN, einem Unterprogramm, das einen FC-ERROR (ILLEGAL FUNCTION CALL ERROR)meldet. Wenn USR angewandt wird, ohne daß vorher die Adresse des Maschinenprogrammes in 1004H/1005H eingetragen wurde, dann wird der FC-ERROR gemeldet.

Wenn ein USR-Aufruf erfolgt, wird der Stapelzeiger 16 Byte zurückgesetzt, um Platz für das Benutzerstack zu haben. Falls mehr Speicherplatz für den Stapelzeiger benötigt wird, kann man den BASIC-Stack sichern und einen eigenen Stack für das Maschinenprogramm aufbauen. Bevor man aus der Maschinenroutine zurückspringt, muß der BASIC-Stack wieder restauriert werden.

Alle Register und Speicherzelleninhalte des Maschinenprogramm-Speicherbereiches können von dem BASIC-Programm verändert werden. Wichtig ist, daß keine Bytes des BASIC-Interpreters, kein Programmtext und keine Variablenfelder überschrieben werden. Wichtig ist auch, daß genausoviele Bytes auf dem Stack abgelegt werden, wie man wieder herausholt.

USR ist ein Aufruf mit einem Argument. Das Maschinenprogramm kann dieses Argument verwenden, indem es eine andere Maschinenroutine aufruft, deren Adresse in den Speicherzellen E008H und E009H steht. Das LSB der Adresse steht in E008H, das MSB in E009H. Dieses Programm (DEINT) speichert das Argument im Registerpaar (DE).

Das Argument erhält einen Integerwert beim Aufruf. Sein Wert muß im Bereich -32768 bis +32767 liegen, ansonsten wird ein FC ERROR gemeldet.

Um Ergebnisse wieder an BASIC zu übergeben, speichere man das Resultat in den Registern A,B. Dieser Wert wird als vorzeichenbehaftete Integerzahl wie oben definiert betrachtet.

Rufen Sie dann das Maschinenprogramm auf, dessen Adresse in den Speicherzellen EOOAH und EOOBH angegeben ist. Wenn dieses Unterprogramm nicht aufgerufen wird, dann liefert der Aufruf USR(X) den Wert X. Sonst wird der Wert aus A,B als Ergebnis übergeben. Um zum BASIC-Programm zurückzuspringen muß eine RET-Anweisung durchlaufen werden.

Wenn irgendwelche Interruptserviceroutinen abgearbeitet werden, sollten Stack, Register A-L und das PSW gesichert werden. Außerdem sollten die Interrupts wieder eingeschaltet werden (EI), bevor der Rücksprung erfolgt, weil bei einem Interrupt automatisch alle weiteren Interrupts abgeschaltet werden (DI), bis wieder ein Interrupt-Enable (EI) gegeben wird.

Es gibt nur diese eine Möglichkeit, ein Maschinenprogramm und ein BASIC-Programm zu verknüpfen. Dennoch sind die Möglichkeiten, mehrere Maschinenprogramme zu haben, nicht begrenzt. Da die Adresse, zu der bei USR gesprungen wird, jederzeit mit PEEK oder DEEK gelesen bzw. mit POKE und DOKE geändert werden kann, steht die Möglichkeit offen, beliebig viele Maschinenunterprogramme zu verwenden.

## Anhang E Verwendung des Cassetten-Interface

Programme können mit dem CSAVE-Befehle auf Cassette abgespeichert werden. CSAVE kann im Programmbetrieb oder im Kommandobetrieb verwendet werden:

CSAVE <Stringausdruck>

Das gerade im Speicher befindliche BASIC-Programm wird unter dem durch den ersten Buchstaben des Stringausdruckes spezifizierten Namens auf der Cassette abgelegt. Beachten Sie bitte, daß ein Programm, das A genannt werden soll, mit dem Befehl CSAVE "A" abgelegt wird. Nachdem die Speicherung abgeschlossen ist, kehrt BASIC auf die Kommandoebene zurück. Die Programme werden in einer internen Darstellung geschrieben, die nur BASIC verwendet. Variablenwerte werden nicht mit gespeichert, sodaß CSAVE auch nicht die Variablenwerte des augenblicklich gespeicherten Programmes verändert.Die Anzahl Dummyzeichen (siehe NULL-Befehl) hat keinen Einfluß auf die Wirkungsweise von CSAVE. Bevor Sie CSAVE aufrufen, schalten Sie Ihren Cassettenrecorder auf Aufnahme und lassen Sie ihn anlaufen.

CSAVE schreibt zunächst einen Block, der den ersten Buchstaben des Stringausdruckes enthält und ruft dann ein Maschinenprogramm auf, das den Programmtext ausgibt.

Programme können mit dem CLOAD-Befehl von Cassette gelesen werden. CLOAD führt zu Beginn einen NEW-Befehl aus und löscht damit alle alten Programme und Speicherzellen. Anschließend wird das neue Programm in den Speicher übertragen. Nach dem Einlesen befindet sich BASIC wieder auf Kommandoebene. Der Lesevorgang beginnt, wenn das CLOAD-Programm drei aufeinanderfolgende ØØ erkannt hat. BASIC kehrt nicht auf die Kommandoebene zurück , wenn bei CLOAD ein Lesefehler auftritt oder das Programm nicht gefunden werden kann. Dann sucht der Rechner solange, bis er das File findet. Wenn er es nicht findet, muß man RESET betätigen und BASIC neu starten. Wenn das Programm gefunden wird, gibt BASIC die Meldung aus:

File program identifier found

Wie das Programm, das geladen wird, während des Ladevorganges auf dem Bildschirm angezeigt wird, hängt davon ab, welchen Monitor Sie verwenden.

Datenfelder können mit CSAVE* gespeichert und mit CLOAD* in den Speicher zurück geladen werden. Es werden folgende Formate verwendet:

CSAVE* <Feldname>
CLOAD* <Feldname>

Siehe auch Abschnitt 2.4d.

Bei Eingabe und Ausgabe wird eine Prüfsumme erzeugt. Falls ein Prüfsummenfehler festgestellt wird, wird die Meldung "bad" (schlecht) ausgegeben und BASIC springt auf die Kommandoebene zurück. Fehlerhafte Daten werden jedoch im Speicher abgelegt.

Im Falle eines Lesefehlers kann mit CLOAD oder einem direkten GOTO abermaliges Lesen veranlaßt werden.

Bei Verwendung von NAS-SYS wird mit dem Befehl CLOAD?⟨Stringausdr.⟩ das Programm von Cassette gelesen, aber nicht in den Speicher übertragen. So kann man feststellen, ob ein Programm richtig abgespeichert wurde.

Man kann mit BASIC auch über LIST oder PRINT-Befehle Programme oder Daten mithilfe des XØ-Kommandos bei NASBUG4 und NAS-SYS auf Cassette abspeichern und mit INPUT wieder lesen. Dies ist ein nützliches Hilfsmittel, wenn man Unterprogrammbibliotheken abspeichern will, die für mehrere Programme verwendet werden sollen.

## Anhang F Umwandlung von Fremdprogrammen in NASCOM BASIC

Die meisten BASIC-Versionen haben nur kleine Abweichungen gegenüber NASCOM 8K BASIC. Es ist nützlich zu wissen, wo diese Abweichungen liegen können.

### 1. Strings

Verschiedene BASIC-Versionen verlangen eine Angabe darüber, wie lange ein String sein darf. Alle DIM-Anweisungen, die diesem Zweck dienen, sollen entfernt werden. In einigen BASIC-Versionen wird mit einer Vereinbarung wie A$(I,J) ein Stringfeld von J Elementen vereinbart, bei denen jedes maximal die Länge I haben darf. Wandeln Sie diese Anweisungen in äquivalente NASCOM BASIC-Anweisungen um: DIM A$(J). NASCOM BASIC verwendet "+" zum Verketten von Strings, nicht " , " oder " & ". NASCOM BASIC verwendet LEFT$,RIGHT$ und MID$, um Unterstrings aus einem String abzutrennen.Andere BASIC-Versionen verwenden gelegentlich A$(I),um das I-te Zeichen eines String zu bekommen und A$(I,J), um einen Unterstring von der I-ten bis zur J-ten Zeichenposition des String A$ zu isolieren. Setzen Sie dies so um:

| Fremdversion | NASCOM |
|---|---|
| A$(I) | MID$(A$,I,1) |
| A$(I,J) | MID$(A$,I,J-I+1) |

Hierbei wird davon ausgegangen, daß der Index von A$ ein Ausdruck ist oder auf der rechten Seite einer Zuweisung steht. Wenn der Bezug auf A$ auf der linken Seite einer Zuweisung steht und X$ der Stringausdruck ist, der Zeichen in A$ ersetzen soll, dann erfolgt die Umwandlung so:

| Fremdversion | NASCOM |
|---|---|
| A$(I)=X$ | A$=LEFT$(A$,I-1)+X$+MID$(A$,I+1) |
| A$(I,J)=X$ | A$=LEFT$(A$,I-1)+X$+MID$(A$,J+1) |

### 2. Mehrfache Wertzuweisungen

Manche BASIC-Versionen erlauben Anweisungen der Form:

```
500 LET B=C=Ø,
```

wobei dieser Befehl B und C auf Ø setzt. In NASCOM BASIC hat diese Anweisung einen völlig anderen Effekt. Alle "="-Zeichen werden, wenn sie rechts vom ersten Gleichheitszeichen stehen, als Vergleichoperatoren angesehen. Das würde dazu führen, daß B auf den Wert -1 gesetzt würde, wenn C=Ø. Wenn C≠Ø, dann würde B auf Ø gesetzt werden. Die einfachste Methode, um solche Befehle umzusetzen ist:

```
500 C=Ø:B=Ø
```

3. Einige BASIC-Versionen verwenden "\" statt ":", um Befehle, die in einer Zeile stehen, gegeneinander abzugrenzen.

4. Programme, die die MAT-Funktionen (Matrizenoperationen) verwenden, die einige BASIC-Versionen anbieten, müssen als Benutzer-definierte Funktion geschrieben werden.

## Anhang G Speicherplatzbedarf

NASCOM BASIC läßt die Speicherzellen unter 1000H für den Monitor und Benutzer-Maschinenprogramme frei. Es verwendet die Speicherzellen 1000H bis 3E11H als Arbeitsspeicher und ist selbst in E000H bis FFFFH untergebracht. Die Speicherzellen 3E12H bis E000H sind für BASIC-Programme und Datenfelder frei verfügbar.

## Anhang H Literaturhinweise

1. Haase, Stucky,
BASIC - Programmieren für Anfänger
BI-Taschenbuch.

2. Pub. People's Computer Company
"What To Do After You Hit Return

## Anhang 1 Nützliche Hilfsprogramme

### 1. Mit NAS-Sys in Zeile 16 schreiben (nicht gescrollt)

```
1  REM THIS ROUTINE WRITES TO LINE 16
2  REM USING NAS-SYS
10 CLS
20 SCREEN 1,15
25 REM THAT PUTS IT ON BOTTOM LINE
30 PRINT "HEADER";
35 REM NOW WE ARE GOING TO COPY IT TO TOP
40 FOR C=2954 to 3000 STEP 2
50 DOKE C+64,DEEK(C)
60 NEXT C
65 REM CHR$(27) GENERATES ESC=LINE DELETE
70 PRINT CHR$(27);
80 REM REST OF PROGRAM CAN START HERE
```

### 2. Umwandlung von Hex-Zahlen in Dezimalzahlen

```
4  CLS
5  PRINT
10 INPUT"ENTER HEX No.";H$
20 T=0:D=1
30 FOR P=LEN(H$)-1 TO 0 STEP-1
40 C=ASC(MID$(H$,D,1))
50 D=D+1
60 IF (C>=48)*(C<=57) THEN C=C-48:GOTO 100
70 IF (C>=65)*(C<=70) THEN C=C-55: GOTO 100
80 PRINT "+ve Hex with no D.P. please":GOTO 5
100 T=T+C*16↑P
110 NEXT
120 PRINT "Hex ";H$;" in Decimal is";T
130 GOTO 5
```

# C A S S E T T E N - T I P S

Von einigen NASCOM 2 - Anwendern haben wir gehoert, das es gelegentlich Schwierigkeiten mit dem Einlesen von Cassetten gibt, wenn man die hohe Uebertragungsrate (1200 Baud) benutzt. Deshalb einige Tips, die zum Teil auf eigenen Erfahrungen beruhen und zum Teil der englischen Nascom - Clubzeitung INMC NEWS 1/80 entstammen.

1.) Reinigen Sie von Zeit zu Zeit den Tonkopf und den Antrieb Ihres Cassettenrecorders mit einem Wattestaebchen und etwas Alkohol. Benutzen Sie aber den Recorder erst wieder, wenn die Fluessigkeit vollstaendig verflogen ist.

2.) Stellen Sie sicher, dass es keine Masseschleifen gibt. Wenn der Recorder und der NASCOM mit dem Schutzleiter verbunden sind und eine Masseverbindung zwischen beiden besteht, koennen solche Probleme auftauchen. Unterbrechen Sie dann am besten die Masseverbindung zwischen NASCOM und Recorder am Ausgang des Recorders.

3.) Die Eingangsspannung fuer das Cassetten- Interface und der Aufnahmepegel sind gerade bei 1200 Baud recht kritisch. Die besten Pegel muessen Sie experimentell bestimmen. Der beste Aufnahmepegel ist erfahrungsgemaess in der Naehe der Vollaussteuerung. Der Eingang des Cassetteninterfaces sollte mit dem Lautsprecherausgang des Recorders verbunden werden. Dann koennen Sie den Pegel leicht mit dem Lautstaerkeregler einstellen. In diesem Fall wird das Signal allerdings auch von dem evtl. vorhandenen Klangregler beeinflusst. Beste Resultate lassen sich erzielen, wenn die Hoehen weder beschnitten noch angehoben werden.

4.) VR1 des NASCOM 2 laesst sich am einfachsten mit einem Oszilloskop abgleichen. Nehmen sie dazu einfach eine gewisse Zeit den Ton des Cassetten- Interfaces auf, ohne dass Daten uebertragen werden. Beim Abspielen dieses Tones beobachten Sie das Signal an TP 19 mit dem Oszilloskop. Gleichen Sie VR1 so ab, dass ein sauberes Rechtecksignal erscheint.

5.) Benutzen Sie keine Cassetten minderer Qualitaet. Die Cassetten duerfen natuerlich nicht 'eiern' oder Dropouts verursachen. Ausserdem ist die Hoehenaussteuerbarkeit sehr wichtig. Wir haben mit unseren Datencassetten C6, die Sie bei uns sehr preiswert beziehen koennen, beste Erfahrungen gemacht.

6.) Die Leitungen zum Recorder muessen unbedingt abgeschirmt sein (normales Uebergabekabel), damit Brummeinstreuungen verhindert werden.

7.) Wenn der Lautsprecheraussang benutzt wird (s. 3.) und der Lautsprecher damit abgeschaltet ist, kann es suenstig sein, wenn der Aussang zusaetzlich mit einem Widerstand von 8 Ohm belastet wird.

Wir haben mit einem Billigstrecorder (ca. 50DM) nach Beherzigung obiger Tips recht brauchbare Ergebnisse erzielen koennen. Probleme ergaben sich lediglich dadurch, dass die Aussteuerungsautomatik nicht abschaltbar ist, und deshalb die Aufnahmen stark uebersteuert wurden. Das Einlesen war jedoch problemlos moeglich.

Mit dem NASCOM 2 koennen Sie sehr leicht die Uebertragungsrate des Cassetteninterfaces auf 2400 Baud (!) steigern. Zunaechst muss dazu TP20 mit TP4 und TP21 mit TP5 verbunden werden. Die Schalter LSW2 muessen wie folgt eingestellt werden:

| Schalter - | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Stellung - | on | up | up | on | up | up | down |

Wir haben dies ausprobiert und recht hohe Zuverlaessigkeit festgestellen koennen.

Zum Schluss noch ein Hinweis zum ROM - BASIC: Wie Sie aus dem Handbuch ersehen koennen, ist es moeglich, numerische Felder mit dem Befehl 'CSAVE*N' auf Cassette zu speichern und mit dem Befehl CLOAD*N wieder einzulesen. Was der Dokumentation nicht so ohne Weiteres zu entnehmen ist: Es koennen nur solche Variablenfelder eingelesen werden, die vorher mit dem Befehl 'DIM N(n)' dimensioniert worden sind, obwohl Variablenfelder u.U. auch ohne Dimensionierung benutzt werden koennen.

Wir hoffen, Ihnen einige nuetzliche Tips gegeben zu haben

Ihre

N.A.S. elktronische Halbleiter GmbH