

**BEDIENUNGS-
HANDBUCH
PET 2001**



Dieses Handbuch wurde gescannt, bearbeitet und ins PDF-Format konvertiert von

Rüdiger Schuldes

schuldes@itsm.uni-stuttgart.de

(c) 2003

Inhaltsverzeichnis

EINFÜHRUNG	1
TASTENFELDOPERATIONEN	2
Direktes Rechnen	3
Programmierung	4
Dateneingabe	6
BASIC-PROGRAMMIERUNG	
1. Zeilennummer	8
2. Variablen	8
2.1. Namen	8
2.2. Variablenfelder (Arrays)	9
2.3. String (Zeichenketten-Variablen).....	9
2.4. String-Arrays	9
3. Text	10
4. Eingabe	10
4.1. INPUT	11
4.2. GET	11
4.3. READ	11
5. Syntax	12
6. Ausgabeformat	12
7. Logische Vergleichsoperationen	13
8. Sprünge, Verzweigungen, Subroutinen	14
9. Programmschleifen (Loops)	16
10. Mathematische Funktionen	17
11. Uhr	18
12. Fehlersuche	18
Fehlermeldungen:	
SYNTAX ERROR	18
ILLEGAL QUANTITY ERROR	19
BAD SUBSCRIPT ERROR	19
DIVISION BY ZERO ERROR	19
TYPE MISMATCH ERROR	19
UNDEF'D STATEMENT ERROR	19
UNDEF'D FUNCTION ERROR	19
STRING TOO LONG ERROR	19
NEXT WITHOUT FOR ERROR	19
OUT OF DATA ERROR	19
OUT OF MEMORY ERROR	19
OVERFLOW ERROR	19
RETURN WITHOUT GOSUB ERROR	19
REDIM'D ARRAY ERROR	20
BAD DATA ERROR	20
ILLEGAL DIRECT ERROR	20
CANT CONTINUE ERROR	20
13. Korrektur	20

MASCHINENSPRACHE	22
-------------------------------	----

BASIC-BEFEHLE UND ANWEISUNGEN

Systemkommandos	23
RUN	23
CONT	23
GOTO	23
LIST	23
NEW	23
CLR	23
BASIC-Befehle	24
LET	24
READ	24
DATA	24
RESTORE	24
IF .. THEN	24
FOR .. NEXT	24
STEP	24
GOTO	25
GOSUB	25
RETURN	25
ON .. GOTO	25
ON .. GOSUB	25
PRINT	26
DIM	26
END	26
REM	26
INPUT	26
DEFFN	26
FN	26
GET	26
STOP	26
Input-Output-Anweisungen	27
OPEN	27
CLOSE	27
SAVE	27
LOAD	27
VERIFY	27
PRINT	27
INPUT	27
GET	27
String (Zeichenketten)-Befehle	28
LEN	28
VAL	28
STR\$	28
ASC	28
CHR\$	28
LEFT\$	28
RIGHT\$	28
MID\$	28
+	28
Logische Operatoren: AND, OR und NOT	28

KASSETTENOPERATIONEN	29
Programme	29
Daten	30

PERIPHERIEGERÄTE

IEEE 488-Bus (IEC-Bus)	33
Steuerbefehle für den IEEE-Bus	33
USER PORT	35
Interface für zweite Kassette	36
Speichererweiterungsanschluß	36

Einführung

Der COMMODORE PET 2001 ist ein freiprogrammierbares BASIC-Tischcomputer-System mit hoher Leistungsfähigkeit. Von einem System kann man sprechen, da bereits alle für den Betrieb notwendigen Ein- und Ausgabeeinheiten eingebaut sind:

Tastatur mit 73 Tasten

Bildschirm für 1000 Zeichen

Kassetteneinheit zur Speicherung von Programmen und Daten

Außer den Möglichkeiten zur kompletten Text- und Stringverarbeitung und zum Verkehr mit Peripheriegeräten ist das komplette Interface für den Anschluß sämtlicher IEEE 488-kompatiblen Geräte bereits in der Grundausführung eingebaut. Der 8 k BASIC-Interpreter ist in ROM's untergebracht und beansprucht daher keinen Platz im Arbeitsspeicher.

Bei dem im PET 2001 verwendeten COMMODORE-BASIC handelt es sich um eine leicht programmierbare und äußerst vielseitig anwendbare Programmiersprache. Obwohl COMMODORE-BASIC weitaus umfangreichere Möglichkeiten bietet als die meisten BASIC-Versionen, kann fast jedes vorliegende BASIC-Programm praktisch ohne Änderung eingegeben werden.

Um mit dem PET 2001 arbeiten zu können, genügt es, ihn an eine normale 220-Volt Schuco-Steckdose anzuschließen und den Netzschalter — auf der Rückseite, links — einzuschalten. Nach einigen Sekunden erscheint auf dem Bildschirm:

```
*** COMMODORE BASIC ***  
7167 BYTES FREE  
READY.  
█
```

Das blinkende Quadrat unter READY ist der sogenannte Cursor. Der Cursor zeigt an, daß auf dem Bildschirm geschrieben werden kann und wo geschrieben wird. Wird irgendeine Taste gedrückt, so erscheint das betreffende Zeichen auf dem Bildschirm an der Stelle des Cursors, der Cursor bewegt sich um einen Schritt nach rechts.

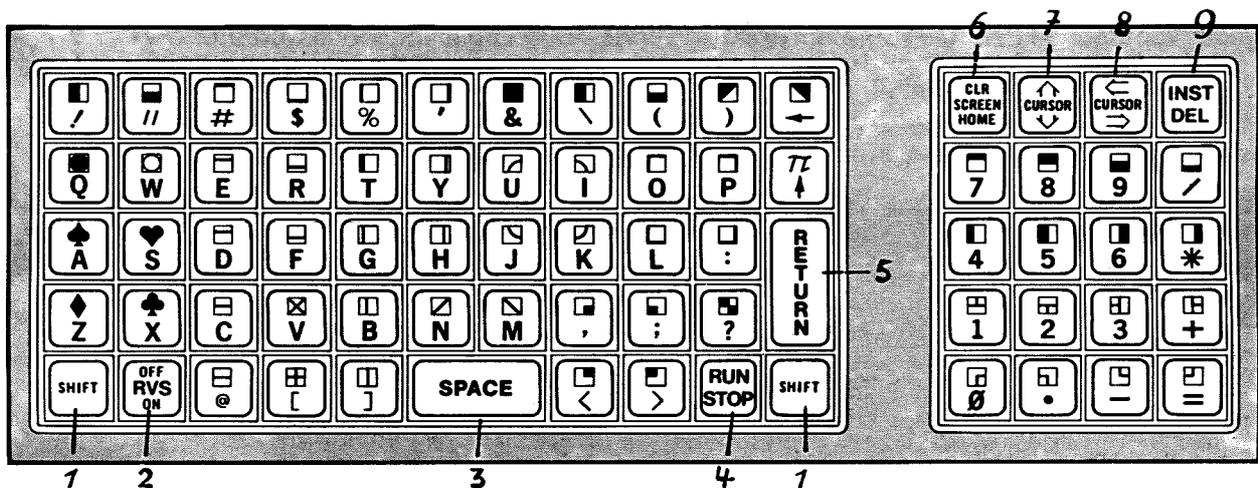
Da auf dem Bildschirm gleichzeitig bis zu 25 Zeilen dargestellt werden können, ist bei Programmierung, Programmausführung und bei allen sonstigen Operationen eine vorzügliche Übersichtlichkeit gewährleistet.

Tastefeldoperationen

Um mit einem Computer arbeiten zu können, muß man ihm Mitteilungen (z.B. Daten, Arbeitsanweisungen,) zukommen lassen, der Computer seinerseits muß der Außenwelt Mitteilungen (Ergebnisse, Fragen,) geben können. Dieser Verkehr mit der Außenwelt kann auf mannigfaltige Weise vonstatten gehen. Die Eingabe kann von Meßgeräten, Kassetten, Telefonleitung, Band, etc. erfolgen. Zur Ausgabe bieten sich Drucker, Plotter, Kassette, Telefonleitung, etc. an. Die wichtigsten und meistgebrauchten Mittel im Verkehr mit dem Computer sind das Tastenfeld zur Eingabe und der Bildschirm zur Ausgabe.

Über das Tastenfeld kann wie mit einer normalen Rechenmaschine gerechnet werden, es können Programme und es können Daten in ein laufendes Programm eingegeben werden. Im Zusammenwirken mit dem Bildschirm kann ein echter Dialogverkehr mit dem Rechner stattfinden.

Das Tastenfeld besteht aus 73 Tasten, die in zwei Felder aufgeteilt sind. Der linke Teil entspricht etwa einer normalen Schreibmaschinentastatur, der rechte Teil der Tastatur einer Rechenmaschine.



Auf beiden Tastenfeldern befinden sich spezielle Tasten, die farblich hervorgehoben sind.

Die beiden Tasten SHIFT (1) bewirken die Umschaltung auf die jeweils oben auf der Taste befindlichen Zeichen.

Die Taste SPACE (3) liefert — wie bei einer Schreibmaschine — einen Zwischenraum.

RVS (2) bewirkt die Negativdarstellung (schwarz auf weiß) der Zeichen auf dem Bildschirm. OFF stellt den Normalzustand wieder her.

STOP (4) beendet ein laufendes Programm. RUN hat nicht die Funktion des Buchstabe für Buchstabe eingegebenen RUN, sondern entspricht dem Befehl LOAD mit anschließendem RUN.

RETURN (5) wird zur Beendigung jeglicher Eingabe gebraucht. Es bewirkt Wagenrücklauf (carriage return), bzw. auf dem Bildschirm das Ende einer Zeile und den Beginn der nächsten. Beim direkten Rechnen wirkt es als Ausführungsbefehl. Fängt eine auf dem Bildschirm geschriebene Zeile mit einer Zahl zwischen 0 und 63999 an, so wird die Zeile unter dieser Nummer in den Programmspeicher geschrieben, wenn RETURN gedrückt wird. Nach einer Dateneingabe im Anschluß an einen INPUT-Befehl bewirkt RETURN die Fortsetzung des Programms. Wird jedoch in einem INPUT-Befehl die Taste RETURN gedrückt, ohne daß Daten eingegeben wurden, so wird das Programm beendet.

HOME (6) bringt den blinkenden Cursor in die linke obere Ecke des Bildschirms (in die Home-Position). CLR löscht den gesamten Bildschirm und bringt ebenfalls den Cursor in Home-Position.

CRSR (7 und 8) sind Cursor-Kontrolltasten. Mit ihrer Hilfe kann der Cursor an jede beliebige Stelle des Bildschirms geführt werden. Werden die Cursor-Tasten während eines Textes (Zeichen zwischen Anführungszeichen) gedrückt, so erfolgt keine Cursorbewegung auf dem Bildschirm,

sondern die Bewegung wird in das Programm geschrieben. Auf diese Weise kann der Cursor voll programmiert werden.

DEL (9) löscht das links vom Cursor stehende Zeichen. INST schafft einen Zwischenraum zum Einfügen eines Zeichens.

Mittels SHIFT (1) lassen sich die auf den Tasten zusätzlich dargestellten graphischen Zeichen erreichen.

Die über den Buchstaben befindlichen graphischen Zeichen können mittels POKE 59468,14 durch entsprechende Kleinbuchstaben ersetzt werden. Der Befehl POKE 59468,12 bewirkt wieder die Grafikdarstellung.

Direktes Rechnen

Wie mit jeder normalen Rechenmaschine können auch mit dem PET 2001 alle üblichen Rechnungen direkt (ohne Programm) durchgeführt werden. Bei der Eingabe ist folgendes zu beachten:

Beginnt eine Eingabe mit einer Zahl, so wird diese Zahl vom PET als Zeilennummer interpretiert und alles was hinter dieser Zahl steht in den Programmspeicher geschrieben. Die vom Taschenrechner her gewohnte Rechnung $2*3=$ führt also zur Speicherung der (sinnlosen) Programmzeile $2*3=$. Die Eingabe muß in diesem Fall lauten $?2*3$ (bzw. PRINT $2*3$). Nach Drücken der RETURN-Taste erscheint das Ergebnis 6 auf dem Bildschirm. Selbstverständlich können auch kompliziertere Rechnungen und auch mehrere Rechnungen, durch Doppelpunkt getrennt, auf einmal eingegeben werden.

Beispiel:

```
?3*4 : ?5/7
  12
  .714285714
READY.
█
```

Für die Durchführung einer Rechnung können maximal 80 Zeichen verwendet werden. Reicht dies nicht aus, so kann das Ergebnis der ersten Zeile in einer Variablen zwischengespeichert und in der nächsten Zeile weiterverwendet werden.

Beispiel:

```
A=3*LOG(5/3)+2 : ?A
3.53247687
READY.
?A*8/7-1.2
2.83711643
READY.
█
```

Außer dem direkten Rechnen über das Tastenfeld, können auch Befehle auf dem Tastenfeld eingegeben werden. Lediglich INPUT, GET und DEFFN werden nur innerhalb eines Programms ausgeführt.

Programmierung

Erst in Verbindung mit einem Programm wird die außerordentliche Leistungsfähigkeit des PET 2001 voll ausgeschöpft. Ein Programm ist die Arbeitsanweisung für den Computer. In ihm stehen sämtliche Operationen, die er mit gegebenen Ausgangswerten auszuführen hat, um möglichst schnell die gewünschten Ergebnisse zu erzielen. Das Programm bestimmt auch die Art der Eingabe und die Form der Ausgabe (Dialogverkehr). Diese oft als nebensächlich betrachteten Teile des Programms bedürfen der meisten Mühe und Aufmerksamkeit, denn von ihnen hängt letztlich die praktische Brauchbarkeit des Programms ab. Eine bequeme Eingabe und eine leichte Korrekturmöglichkeit, wie sie der PET 2001 bietet, sind daher von großer Wichtigkeit. Am Beispiel eines Primitivst-Programmes (Multiplikation zweier Zahlen) soll gezeigt werden, wie ein Programm benutzerfreundlich aufgebaut sein kann.

In seiner einfachsten Form sieht das Programm folgendermaßen aus:
Beispiel:

```
10?2*3
RUN
6
READY.
█
```

Nach Eingabe von RUN und Drücken der RETURN-Taste erscheint das richtige Ergebnis: 6. Dies ist zweifellos ein Programm, jedoch ein kaum brauchbares. Sollen andere Zahlen als 2 und 3 multipliziert werden, so müssen diese Zahlen jedesmal neu ins Programm geschrieben werden. Die beiden Faktoren sollten jeweils ohne Umstände eingegeben werden können. Man kann das Programm daher folgendermaßen abändern:

Beispiel:

```
10 INPUT A
20 INPUT B
30 ?A*B
RUN
? 2
? 3
6
READY.
█
```

Der PET 2001 fragt jetzt jeweils mit einem Fragezeichen nach den Faktoren, so daß wir beliebige Zahlen multiplizieren können. Aus dem Fragezeichen geht jedoch nur hervor, daß PET etwas wissen will, jedoch nicht, was er wissen will. Eine weitere kleine Änderung des Programms macht die Fragestellung deutlich.

Beispiel:

```

10 INPUT"FAKTOR 1";A
20 INPUT"FAKTOR 2";B
30 ?A*B
RUN
FAKTOR 1? 2
FAKTOR 2? 3
6
READY.

```

Es ist jedoch nicht ersichtlich, welche Bewandtnis es mit der Zahl hat, die der Rechner unter die beiden Fragen schreibt. Nur wer das Programm kennt, weiß, daß diese Zahl das Ergebnis darstellt. Mit ein wenig Text in der Zeile 30 kann Abhilfe geleistet werden.
Beispiel:

```

10 INPUT"FAKTOR 1";A
20 INPUT"FAKTOR 2";B
30 PRINT"ERGEBNIS";A*B
READY.
RUN
FAKTOR 1? 2
FAKTOR 2? 3
ERGEBNIS 6
READY.

```

Es ist noch störend, daß nach jeder ausgeführten Rechnung erneut RUN eingegeben werden muß, um die nächste Berechnung vornehmen zu können. Wir befehlen dem Rechner nun, nach Beendigung sofort wieder mit der Frage nach dem Faktor 1 zu beginnen.
Beispiel:

```

10 INPUT"FAKTOR 1";A
20 INPUT"FAKTOR 2";B
30 PRINT"ERGEBNIS";A*B
40 GOTO10
RUN
FAKTOR 1? 2
FAKTOR 2? 3
ERGEBNIS 6
FAKTOR 1? 4
FAKTOR 2? 7
ERGEBNIS 28
FAKTOR 1?

```

Ein letzter Schönheitsfehler besteht noch darin, daß zwischen den einzelnen Rechnungen kein Zwischenraum vorhanden ist. Mit der Zeile 35 wird eine Leerzeile erzeugt.
Beispiel:

```
10 INPUT"FAKTOR 1";A
20 INPUT"FAKTOR 2";B
30 PRINT"ERGEBNIS";A*B
35 PRINT
40 GOTO10
READY.
RUN
FAKTOR 1? 2.23
FAKTOR 2? 6.8
ERGEBNIS 15.164

FAKTOR 1? 9
FAKTOR 2? 3
ERGEBNIS 27

FAKTOR 1? █
```

Erst in dieser Form kann man von einem benutzerfreundlichen Programm sprechen.

Dateneingabe

In dem vorangegangenen Beispiel wurden Werte mittels INPUT-Befehl in das laufende Programm eingegeben. Dies ist die gebräuchlichste Methode der Dateneingabe. INPUT ist nicht nur auf numerische Werte beschränkt, sondern es kann sich auch um Texte, alphanumerische oder graphische Zeichen handeln, wenn sie in einer Stringvariable eingegeben werden. Mit einer INPUT-Anweisung können mehrere Daten (numerische oder alphanumerische, aber auch gemischt) eingegeben werden.

Beispiel:

```
100 INPUT A$,B,C%
120 PRINT A$,B,C%
READY.
RUN
3.57,KELLER,623.85,20147
3.57 KELLER 623.85 20147

READY.
█
```

Text zur Abfrage im Dialogverkehr kann direkt im INPUT-Befehl stehen. Er muß durch ein Semicolon von den Variablen getrennt sein.

Beispiel:

```
40 INPUT"PREIS";P
50 INPUT"NAME";N$
60 PRINTN$,P
READY.
RUN
PREIS? 122.68
NAME? KASSETTENRECORDER
KASSETTENRECORDER 122.68
READY.
█
```

Eine weitere Möglichkeit der Eingabe von Zeichen in ein laufendes Programm bietet der Befehl GET. Im Gegensatz zu INPUT (hier können viele Zeichen in einem Befehl eingegeben werden, das Programm hält bis zum Drücken der RETURN-Taste an) wird mit GET ein einzelnes Zeichen übernommen, das Programm hält dabei nicht an. Wurde jedoch vorher eine Taste gedrückt, so bleibt dieses Zeichen gespeichert und wird später durch GET übernommen. GET wird weniger zur Dateneingabe in ein Programm verwendet, als um beliebige Tasten der Tastatur zu Spezialfunktionstasten zu machen.

Beispiel:

```
10 GETG$
20 IFG$="A" THEN230
30 IFG$="B" THENPRINT"B GEDRUECKT":GOTO2
40 IFG$="X" THENX=π:GOTO400
READY.
█
```

Soll ein Programm beim GET-Befehl anhalten, so läßt sich dies leicht mit einer kleinen „Warteschleife“ verwirklichen. Das Programm wird dann sofort nach Druck auf eine beliebige bzw. eine bestimmte Taste fortgesetzt, ohne daß RETURN gedrückt werden muß.

Beispiel:

```
10 GETX$:IFX$="" THEN10
20 IFX$="J" THEN100
30 IFX$="N" THEN200
40 GOTO10
100 PRINT"JA":GOTO10
200 PRINT"NEIN":GOTO10
READY.
RUN
JA
NEIN
```

Basic-Programmierung

1. Zeilennummer

Ein Programm besteht aus einer Reihe von Operationen, die nacheinander ausgeführt werden. Welche Operation auszuführen ist, wird dem Rechner jeweils durch einen entsprechenden Befehl mitgeteilt. Die Befehle werden in nummerierten Zeilen im Arbeitsspeicher des Rechners gespeichert. Bei den meisten bisherigen BASIC-Versionen konnte nur ein Befehl in eine Zeile geschrieben werden. COMMODORE-BASIC läßt mehrere Befehle innerhalb einer Zeile zu. Dies ist besonders wertvoll bei logischen Vergleichsoperationen (auf Vergleichsoperationen wird später eingegangen).

Zugelassen sind alle Zeilennummern von 0 bis 63999. Die Abarbeitung der Zeilen erfolgt in der Reihenfolge der Zeilennummern. Zweckmäßigerweise beginnt man die Zeilennumerierung nicht bei 0 oder 1 und auch die Wahl der Abstände der Zeilennummern sollte nicht in 1-er Schritten erfolgen. Besser geeignet ist beispielsweise der Start mit Zeilennummer 10 und die Weiterführung in 10-er Abständen (10,20,30,...). Der Grund hierfür ist, bei später eventuell notwendigen Programmkorrekturen oder -erweiterungen ohne Schwierigkeiten neue Zeilen einschieben zu können. Es besteht keine Veranlassung, kleine Zeilennummern zu verwenden, um Speicherplatz zu sparen, denn jede Zeilennummer benötigt unabhängig von ihrer Größe 4 Byte.

Gleichfalls kann es zweckmäßig sein, einzelne Programmteile bereits durch die Zeilennummer unterscheidbar zu machen. Beispielsweise kann ein Teil des Programms die Zeilennummer 10 bis 130 belegen, der zweite Teil die Nummern 1000 bis 1220 für sich beanspruchen und der dritte Teil bei 2000 beginnen. Große Abstände zwischen den Zeilennummern schaden weder in Bezug auf Speicherplatzbedarf noch auf Ausführungsgeschwindigkeit.

2. Variablen

Zahlenwerte — eingegebene oder berechnete — können auf Speicherplätzen, die mit einem Namen bezeichnet werden, gespeichert werden. Um eine Zahl zurückzurufen, genügt es, diesen Namen zu benutzen. Einen derart mit Namen bezeichneten Zahlenwert nennt man eine Variable (eine Variable entspricht also ziemlich genau dem von kleineren Rechnern her bekannten Register).

2.1 Namen

Als Namen können verwendet werden:

- 26 Buchstaben von A - Z (Großbuchstaben)
- ein Buchstabe und eine Zahl z.B. A0, A3, C7, X9
- zwei Buchstaben z.B. AA, AC, DE, FG ... ZZ

Dies sind insgesamt 962 Namen. Zwei dieser Namen können jedoch nicht verwendet werden, und zwar TI und ST. TI ist belegt durch die laufende Zeit in Einheiten von 1/60-stel Sekunden. ST zeigt den Status von Peripherie-Geräten an. Außerdem können IF, TO, FN, ON und OR nicht verwendet werden, da sie vom Rechner als BASIC-Befehle verstanden werden.

Während diese Variablen Zahlenwerte mit der vollen Genauigkeit und dem gesamten Rechenbereich des Rechners speichern können, kann eine andere Variablenart nur ganze Zahlen im Bereich von -32767 bis 32767 aufnehmen. Auch für diese Variablenart können die oben aufgeführten Namen verwendet werden, mit dem einen Unterschied, daß zusätzlich das Prozentzeichen (%) zu verwenden ist.

Z.B.: C%, C7%, LD%

Es können auch längere Namen als Variable verwendet werden

Beispiel: KASSEL

Hiervon werden nur die beiden Anfangsbuchstaben, also KA zur Kennzeichnung benutzt. Dies bedeutet, daß die gleiche Variable auch mit KATZE oder nur mit KA zurückgerufen werden kann.

Außerdem darf in einem solchen Namen an keiner Stelle eine Zeichenfolge enthalten sein, die vom Rechner als Befehl verstanden werden könnte.

Beispiel: UNSINN

Die Variable UNSINN ist unzulässig, da das in diesem Wort enthaltene SIN als Sinus interpretiert wird.

2.2 Variablenfelder (Arrays)

Alle zulässigen Namen lassen sich für Felder verwenden. Unter einem Feld versteht man eine Gruppe von Variablen, die den selben Namen, jedoch verschiedene Indices besitzen. Die Anzahl der verwendeten Indices bezeichnet man auch als Dimension des Feldes. Beispielsweise handelt es sich bei A(2,4,3) um ein Element eines drei-dimensionalen Feldes. Die Indices können Variable oder mathematische Ausdrücke sein — z.B. A(I,J,K) oder C(4,Z+2). Auf diese Weise ist es möglich, diese Variablengruppen indirekt zu adressieren um mit einem einzigen Befehl in einer Schleife viele Variable zuzuordnen oder zurückzurufen. Die Anzahl der Dimensionen und der Elemente eines solchen Feldes ist nur begrenzt durch die verfügbare Speicherkapazität.

Außer diesen Feldern aus Variablen mit voller Genauigkeit können auch Felder aus Ganzzahl-Variablen gebildet werden.

Beispiel: A% (3,2)

Soll ein Feld in irgendeiner Dimension mehr als 11 Elemente (0-10) umfassen, so muß es dimensioniert werden.

Beispiel: DIM B (4,20)

COMMODORE-BASIC erlaubt nicht nur Dimensionierung mit Konstanten, sondern auch mit Variablen oder mathematischen Ausdrücken.

Beispiel: DIMA(M,N), DIMB(I+3,J*2)

Bei kleineren Feldern ist eine Dimensionierung überflüssig. Speziell bei mehrdimensionalen Feldern kann jedoch eine Dimensionierung notwendig sein, um Speicherplatz einzusparen. Wird beispielsweise die Variable A(2,2,2) verwendet, ohne daß sie dimensioniert wurde, so werden automatisch $11 \times 11 \times 11 = 1331$ Variable und damit 6655 Byte des Arbeitsspeichers belegt.

2.3 String (Zeichenketten-Variable)

Während in normale Variablen nur Zahlenwerte eingegeben werden können, erlauben String-Variablen die Eingabe beliebiger Zeichen — Buchstaben, Zahlen, Satzzeichen, graphische Zeichen. Mit den zur Verfügung stehenden String-Operatoren können aus derartigen Zeichenketten beliebige Zeichen oder Gruppen von Zeichen herausgenommen, mit anderen Zeichen verglichen und — falls es sich um Zahlen handelt — in normale Variablen umgewandelt werden. Ganze Wörter oder Sätze, die häufig gebraucht werden, können jeweils mit dem Namen der String-Variablen aufgerufen werden. Ein einzelner String kann bis zu 255 Zeichen umfassen.

Zu beachten:

Wie bereits TI in Kapitel 2.1, darf auch TI\$ nicht verwendet werden. Die Spezialvariable TI\$ ist belegt für die Tageszeit in Stunden, Minuten und Sekunden.

2.4 String-Arrays

Wie bei gewöhnlichen Variablen können auch hier ein- oder mehrdimensionale Felder gebildet werden. Es gelten die gleichen Regeln wie bei den gewöhnlichen Feldern, das heißt, auch hier brauchen nur String-Arrays mit mehr als 11 Elementen in einer Dimension dimensioniert zu werden. Es ist nicht — wie bei manch anderen BASIC-Versionen — notwendig, einzelne Strings zu dimensionieren. Der Befehl DIM A \$(30) bedeutet *nicht* die Dimensionierung eines Strings mit 30 Zeichen, sondern die Dimensionierung eines Arrays mit 30 Strings!

4.1 INPUT

Steht im Programm der Befehl INPUT und dahinter eine oder mehrere Variablen — mehrere Variablen müssen durch Kommata getrennt werden — so bleibt das Programm an dieser Stelle stehen und auf dem Bildschirm erscheint ein Fragezeichen (?) als Aufforderung zur Dateneingabe. Sollen Daten für mehrere Variablen eingegeben werden, so erfolgt die Eingabe ebenfalls mit Trennung durch Kommata. Zum Abschluß jeder Eingabe ist die Taste RETURN zu betätigen. Werden zu wenige Daten eingegeben, so erscheinen am Bildschirm zwei Fragezeichen (??) als Aufforderung, die restlichen Daten einzugeben. Werden jedoch zu viele Daten eingegeben, läuft das Programm ebenfalls weiter, die überzähligen Daten werden ignoriert und es erfolgt die Meldung: ?EXTRA IGNORED

Soll auf dem Bildschirm eine Frage erscheinen, welche Daten jetzt eingegeben werden sollen, so kann diese Frage als Text vor die Variablen gestellt werden (Dialogverkehr, „Prompting“). Text und Variable müssen durch ein Semicolon getrennt werden.

Beispiel:

```
10 INPUT"TAG ";T$
20 INPUT"MONAT";M$
30 INPUT"JAHR ";J$
40 PRINT:PRINT"HEUTE IST DER ";T$;".";M$;"
";J$
50 READ Y.
60 PRINT Y.
70 TAG ? 3
80 MONAT ? 5
90 JAHR ? 1978
100 HEUTE IST DER 3.5.1978
110 READY.
```

Werden falsche Daten eingegeben — alphanumerische Daten, wenn in eine numerische Variable eingegeben werden soll — so erfolgt die Meldung REDO FROM START. Die Eingabe kann dann mit den richtigen Daten wiederholt werden.

4.2. GET

Im Gegensatz zu INPUT bleibt das Programm nicht stehen, wenn es zu einem GET-Befehl kommt. GET übernimmt nur ein einziges Zeichen. Da bis zu 10 Tastenbetätigungen im Rechner gespeichert werden, wird jedes Zeichen übernommen, das während des Programmlaufes vor Erreichen des GET-Befehles gedrückt wurde. Soll bei einem GET-Befehl der Rechner auf Eingabe eines Zeichens warten, so ist dies leicht durch eine kleine Schleife zu erreichen.

Beispiel: 5 GETR\$:IFR\$=""THEN5
6 IFR\$<>"A"THEN5

Zeile 5 und 6 bewirken einen Programmhalt, bis die Taste "A" gedrückt wird.

4.3. READ

READ war bei den ersten BASIC-Versionen die einzige Möglichkeit, Daten im Programm in Variablen einzugeben. Die Daten werden hierzu an beliebiger Stelle des Programms unter DATA ins Programm geschrieben. Der Befehl READ liest die Daten in der vorgegebenen Reihenfolge. Sollen Daten mehrfach gelesen werden, kann dies durch den Befehl RESTORE erreicht werden.

Beispiel:

```

1000 READ A$,B$,C%,X
1001 PRINT A$,A,B$,B,X,C%
1002 END
100 DATA 2.55,MILCH,3.27,EIER,3210,2.718

READY.
MILCH      2.55      EIER      3.27
MILCH      2.718     3210
READY.

```

5. Syntax

Der Rechner versteht nur Befehle, die in seinem Befehlsvorrat enthalten und in einer genau festgelegten Form gegeben sind. Es ist daher wichtig, diese Befehle auch genau in dieser Form ins Programm zu schreiben mit allen jeweils erforderlichen Satzzeichen. Werden Befehle falsch geschrieben, so daß sie vom Rechner nicht verstanden werden, erfolgt die Fehlermeldung: UNDEF'D STATEMENT. Bei der Verwendung falscher Zeichen (z.B. Komma statt Dezimalpunkt) lautet die Fehlermeldung: SYNTAX ERROR. Erscheint eine solche Fehlermeldung, wird gleichzeitig angegeben, in welcher Zeile der Fehler auftrat. In diesem Fall listet man die betreffende Zeile auf und untersucht, ob die darin enthaltenen Befehle mit den in der Befehlsliste angegebenen übereinstimmen.

Mehrere Befehle pro Zeile

Im Gegensatz zu den meisten anderen BASIC-Versionen ist es bei COMMODORE-BASIC nicht notwendig, jedem Befehl eine neue Zeilen-Nummer zuzuordnen. Um einen Befehl ins Programm zu schreiben, genügt es, die Befehle durch einen Doppelpunkt (:) voneinander zu trennen. Bis zu einer Maximallänge von 72 Zeichen pro Zeile können auf diese Weise beliebig viele Befehle in eine Zeile geschrieben werden. Neben einer Ersparnis an Speicherplatz bewirkt dies auch eine schnellere Programmausführung. Besonders vorteilhaft wirkt sich das Schreiben mehrerer Befehle in eine Zeile bei logischen Vergleichen aus.

Werden mehrere Befehle in eine Zeile geschrieben, ist jedoch zu beachten:

1. nach Vergleichen wird, wenn der Vergleich „falsch“ ergibt, der ganze Rest der Zeile übergangen; die Programmausführung wird bei der folgenden Zeile fortgesetzt.
2. Sprünge sind nur auf den Zeilenanfang, nicht zu einem Befehl innerhalb der Zeile möglich.

6. Ausgabeformat

Nach einem PRINT-Befehl erfolgt normalerweise stets ein „CR/LF“ (Wagenrücklauf und Zeilenvorschub), so daß jeder PRINT-Befehl auf einer neuen Zeile ausgeführt werden muß. Steht jedoch hinter dem PRINT-Befehl ein Komma, so wird der nächste PRINT-Befehl auf der nächsten freien vortabulierten Position ausgeführt. Vortabulierte Positionen sind die Kolonnen 0, 10, 20, 30. Wird statt des Kommas ein Semicolon verwendet, so bleiben zwischen aufeinanderfolgenden Zahlenwerten jeweils 2 Leerstellen frei, von denen die zweite für das Vorzeichen reserviert ist. Handelt es sich um Strings, so werden die PRINT-Befehle ohne Zwischenraum aneinandergefügt. Mit SPC kann ein Zwischenraum, mit SPC(n) können n Zwischenräume eingefügt werden. SPC bestimmt immer den Abstand zur vorangegangenen Darstellung, TAB bezieht sich jedoch immer auf den linken Bildschirmrand.

Beispiele:

```

10 PRINT "1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20"
20 INPUT A$
30 INPUT B$
40 IF A$ < B$ THEN PRINT A$ " IST KLEINER ALS " B$
50 IF A$ > B$ THEN PRINT A$ " IST GROESSER ALS " B$
60 IF A$ = B$ THEN PRINT A$ " IST GLEICH " B$
70 GOTO 10
80 PRINT "READY."
90 END

```

Es ist zu beachten, daß bei Verwendung von TAB und SPC die Trennung stets mit einem Semi-colon zu erfolgen hat. Die Verwendung des Kommas führt stets zu vortabulierten Positionen.

7. Logische Vergleichsoperationen

Mit der Befehlsfolge IF .. THEN (wenn .. dann) können Werte verglichen werden. Alle 6 Vergleichsoperationen können durchgeführt werden.

Symbole	Bedeutung
=	gleich
>	größer als
<	kleiner als
=>oder>=	größer oder gleich
=<oder<=	kleiner oder gleich
<>oder<<	ungleich

Verglichen werden können Konstanten, Variablen oder mathematische Ausdrücke. Vergleichsoperationen sind nicht nur zwischen Zahlenwerten möglich, sondern auch zwischen Zeichen bzw. Zeichenketten. Hierbei richtet sich die Größe nach dem ASCII-Code des betreffenden Zeichens. Zeichenketten werden in ihrer ganzen Länge verglichen. Da die Buchstaben im ASCII-Code in alphabetischer Reihenfolge angeordnet sind, ist auf diese Weise ebenfalls eine alphabetische Sortierung möglich.

Beispiel:

```

10 INPUT A$
20 INPUT B$
30 IF A$ < B$ THEN PRINT A$ " IST KLEINER ALS " B$
40 IF A$ > B$ THEN PRINT A$ " IST GROESSER ALS " B$
50 IF A$ = B$ THEN PRINT A$ " IST GLEICH " B$
60 GOTO 10
70 PRINT "READY."
80 END

```

```

10 PRINT "1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20"
20 INPUT A$
30 INPUT B$
40 IF A$ < B$ THEN PRINT A$ " IST KLEINER ALS " B$
50 IF A$ > B$ THEN PRINT A$ " IST GROESSER ALS " B$
60 IF A$ = B$ THEN PRINT A$ " IST GLEICH " B$
70 GOTO 10
80 PRINT "READY."
90 END

```

Die 6 Vergleichsoperatoren können über die logischen Operatoren AND, OR und NOT miteinander verknüpft werden.

Beispiel:

```
10 INPUT A,B,C
20 IFA=BAANDC=3THENPRINT"RICHTIG":GOTO10
30 PRINT"FALSCH":GOTO10
40 END
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
300
310
320
330
340
350
360
370
380
390
400
410
420
430
440
450
460
470
480
490
500
510
520
530
540
550
560
570
580
590
600
610
620
630
640
650
660
670
680
690
700
710
720
730
740
750
760
770
780
790
800
810
820
830
840
850
860
870
880
890
900
910
920
930
940
950
960
970
980
990
1000
```

Ergibt ein Vergleich — oder eine Verknüpfung von Vergleichen — die Aussage „wahr“, so wird der Rest der Zeile nach dem THEN ausgeführt. Bei der Aussage „falsch“ wird die Programmausführung mit der nächsten Zeile fortgesetzt. Soll lediglich ein Sprung zu einer anderen Zeile erfolgen, so genügt es, nach THEN die betreffende Zeilennummer einzugeben.

Bei den logischen Operatoren ist zu beachten, daß die Verknüpfung Bit für Bit erfolgt. Es sollte also nicht geschrieben werden: IF A AND B THEN, wenn eine Operation ausgeführt werden soll, falls beide Werte von Null verschieden sind ($4 \text{ AND } 2 = 0$)! Die Schreibweise sollte in diesem Fall lauten: IF A \llcorner O AND B \llcorner O THEN . . . Der Vorteil dieser bitweisen Operationen ist, daß manche Operationen hierdurch sehr einfach durchgeführt werden können.

8. Sprünge, Verzweigungen, Subroutinen

Nicht in jedem Fall ist es erwünscht oder möglich, ein Programm jeweils mit der nächsten Zeilennummer fortzuführen. Es kann notwendig sein, von einer bestimmten Programmzeile aus stets zu einer an einer anderen Stelle befindlichen Zeile des Programms zu springen. Ebenso kann die Forderung vorliegen, nur unter bestimmten Bedingungen zu dieser anderen Zeile zu gehen. In diesen Fällen spricht man von einem unbedingten bzw. einem bedingten Sprung. Ein solcher Sprung wird durch den Befehl GOTO bewirkt. GOTO kann auch bei einem bedingten Sprung (nach THEN) Verwendung finden.

Beispiel:

9. Programmschleifen (Loops)

Häufig ist es notwendig, daß Teile von Programmen mehrfach durchlaufen werden. Dies ist möglich, indem man mittels eines GOTO Befehles auf eine niedrigere Zeilennummer zurückspringt. Für eine bestimmte Zahl von Schleifendurchläufen kann dann mit einer Variablen ein Zähler aufgebaut werden. Der Zählerstand wird in einer Vergleichsoperation mit der vorgegebenen Anzahl von Durchläufen verglichen.

Beispiel:

```
000100  A=3+A
000200  PRINT A,
000300  I=1+I
000400  IF I<10 THEN 10
000500  PRINT "FERTIG"
000600  GOTO 10
000700  END
000800
000900
001000
001100
001200
001300
001400
001500
001600
001700
001800
001900
002000
002100
002200
002300
002400
002500
002600
002700
002800
002900
003000
003100
003200
003300
003400
003500
003600
003700
003800
003900
004000
004100
004200
004300
004400
004500
004600
004700
004800
004900
005000
005100
005200
005300
005400
005500
005600
005700
005800
005900
006000
006100
006200
006300
006400
006500
006600
006700
006800
006900
007000
007100
007200
007300
007400
007500
007600
007700
007800
007900
008000
008100
008200
008300
008400
008500
008600
008700
008800
008900
009000
009100
009200
009300
009400
009500
009600
009700
009800
009900
010000
READY.
```

Eine weitaus elegantere Methode ist der „FOR-NEXT-LOOP“. Die für eine Schleife notwendigen Operationen — Sprung, Zählen, Vergleich — werden durch die beiden Befehle „FOR“ am Schleifenbeginn und „NEXT“ am Schleifenende ausgeführt.

Beispiel:

```
000100  FOR I=1 TO 10
000200  A=3+A
000300  PRINT A,
000400  NEXT I
000500  PRINT "FERTIG"
000600  END
000700
000800
000900
001000
001100
001200
001300
001400
001500
001600
001700
001800
001900
002000
002100
002200
002300
002400
002500
002600
002700
002800
002900
003000
003100
003200
003300
003400
003500
003600
003700
003800
003900
004000
004100
004200
004300
004400
004500
004600
004700
004800
004900
005000
005100
005200
005300
005400
005500
005600
005700
005800
005900
006000
006100
006200
006300
006400
006500
006600
006700
006800
006900
007000
007100
007200
007300
007400
007500
007600
007700
007800
007900
008000
008100
008200
008300
008400
008500
008600
008700
008800
008900
009000
009100
009200
009300
009400
009500
009600
009700
009800
009900
010000
READY.
```

Die Schrittweite des Zählers kann beliebig gewählt werden. Wird sie nicht vorgeschrieben, so wird vom Rechner die Schrittweite +1 ausgeführt.

Beispiel:

```

FOR I=1 TO 38 STEP 2.5
PRINT I,
NEXT I
PRINT "FERTIG"
READY.
      3.5      6      10.5
      13.5    16      20.5
      20.5    22      26.5
FERTIG
READY.

```

Nach „NEXT“ kann der Variablenname entfallen. Nur in Ausnahmefällen — mehrere verschachtelte Loops — ist es erforderlich, den Variablennamen nach diesem Befehl aufzuführen. Bei mehreren verschachtelten Schleifen genügt es auch, den Befehl NEXT ein einziges Mal zu schreiben und dahinter die Variablen in der richtigen Reihenfolge, durch Kommata getrennt, aufzuführen.

10. Mathematische Funktionen

Außer den 4 arithmetischen Funktionen (+, -, *, :) stehen folgende mathematische Funktionen zur Verfügung:

- ↑ Potenzieren
 - SQR Quadratwurzel
 - LOG natürlicher Logarithmus (zur Basis e). Der dekadische (Brigg'sche Logarithmus) kann erreicht werden durch Division mit LOG (10)
 - EXP Exponentialfunktion zur Basis e
 - SIN Sinus
 - COS Kosinus
 - TAN Tangens
 - ATN Arcus Tangens
- } Winkel in Bogenmaß

Die anderen Arcus-Funktionen (Arcus-Sinus, Arcus-Kosinus) werden mit einer einfachen Routine aus dem Arcus Tangens errechnet.

$$\text{ARCSIN } W = \text{ATN } \frac{W}{\sqrt{1 - W^2}}$$

$$\text{ARCCOS } W = \text{ATN } \frac{\sqrt{1 - W^2}}{W}$$

Winkelangaben in Grad werden ebenfalls mit einer kleinen Hilfsroutine auf das Bogenmaß umgerechnet (Der Wert von π ist fest gespeichert).

- INT ergibt den ganzzahligen Anteil einer Zahl, die Nachkommastellen werden ignoriert. Auch bei negativen Zahlen wird die nächstkleinere ganze Zahl genommen, das heißt, INT (-2.3) ergibt -3.

- ABS Absoluter Wert
Positive Zahlen bleiben unverändert, bei negativen Zahlen wird das Vorzeichen geändert.
- SGN Ermittelt das Vorzeichen einer Zahl und liefert bei positiven Zahlen den Wert 1, bei Null 0, bei negativen Zahlen den Wert -1.

11. Uhr

Der PET 2001 besitzt eine eingebaute Uhr, die durch einen Quarz gesteuert wird. Die Zeit steht in zwei verschiedenen Formaten zur Verfügung:

- TI Die Variable TI enthält eine laufende Zeit in Einheiten von 1/60-stel Sekunden. TI wird vorteilhaft zur Erzeugung genauer Pausen, zur Zeitmessung von Vorgängen und dergleichen verwendet.
Beispiel:

```

10 T=TI
20 FOR I=1 TO 1000 : A=A+1 : NEXT I
30 PRINT (TI-T)/60
40 READY.
50 RUN
4.1
READY.

```

- TI \$ TI \$ enthält die Tageszeit von 0 bis 24 Uhr im Format HHMMSS (Stunden, Minuten, Sekunden). Voraussetzung für die richtige Zeitangabe durch diese Uhr ist — wie bei jeder anderen Uhr — daß sie vorher auf die richtige Zeit gestellt wurde. Dies erfolgt wie bei einer normalen Eingabe in eine String-Variable. Es müssen stets sämtliche 6 Zahlen (auch Nullen) eingegeben werden:

TI \$="HHMMSS"

Die Tageszeit kann jederzeit im Programm abgerufen werden und beispielsweise zu bestimmten Zeiten bestimmte Vorgänge auslösen.

12. Fehlersuche

Es gehört zu den ganz seltenen Ereignissen, wenn ein Programm, speziell ein etwas umfangreiches, sofort fehlerlos arbeitet. Es ist beinahe unvermeidlich, daß beim Programmieren gelegentlich eine Klammer vergessen, ein falsches Satzzeichen verwendet, ein Feld falsch dimensioniert ist,, die Anzahl der möglichen Fehler ist praktisch unbegrenzt. Kann ein Programm wegen eines Fehlers nicht weitergeführt werden, so wird die Ausführung unterbrochen und auf dem Bildschirm erscheint eine Meldung über die Art des Fehlers und die Zeilennummer, in der der Fehler auftrat. Die betreffende Zeile kann dann durch den Befehl LIST mit der Zeilennummer (Beispiel: LIST 360) in den Bildschirm geholt und auf Fehler untersucht werden.

Folgende Fehlermeldungen . . . ERROR IN . . . können auftreten:

SYNTAX ERROR

Diese Fehlermeldung erfolgt stets, wenn die „SYNTAX“ (die für BASIC geltenden Regeln des Satzbaues) nicht eingehalten wurde, das heißt, beispielsweise die Anzahl der geöffneten und der

geschlossenen Klammern stimmt nicht überein, es wurde ein falsches Satzzeichen verwendet, ein nach BASIC-Regeln in Klammer zu setzender Wert wurde ohne Klammer eingegeben und ähnliche Fehler.

ILLEGAL QUANTITY ERROR

Diese Fehlermeldung besagt, daß ein Zahlenwert verwendet wurde, der bei der betreffenden Operation nicht zulässig ist. Beispielsweise können in die Ganzzahl-Variable A% nur Werte zwischen -32768 und 32767 eingegeben werden. Der Versuch, einen außerhalb dieser Grenzen liegenden Zahlenwert einzugeben, führt zu dieser Fehlermeldung.

BAD SUBSCRIPT ERROR

Diese Fehlermeldung bedeutet, daß ein falscher bzw. unzulässiger Index verwendet wurde. Dies bedeutet im einzelnen:

- Wird ein Feld benutzt, das nicht dimensioniert wurde und der Index bzw. einer der Indices ist größer als 10, erfolgt Fehlermeldung.
- Wurde ein Feld dimensioniert und die Anzahl der Dimensionen stimmt bei Dimensionierung und Benutzung nicht überein, erfolgt Fehlermeldung.
- Das selbe Feld wird ohne Dimensionierung mit verschiedenen Dimensionszahlen benutzt.

DIVISION BY ZERO ERROR

Division durch Null (Es ist zu beachten, daß Zahlen kleiner 10^{-38} zu Null gerundet werden).

TYPE MISMATCH ERROR

Einzugebende Daten passen nicht zum Variablentyp (z.B. Zahlenwert in String-Variable oder Text in einfache Variable).

UNDEF'D STATEMENT ERROR

UNDEF'D STATEMENT ERROR wird angezeigt, wenn ein Sprung zu einer nicht vorhandenen Zeilennummer erfolgen soll.

UNDEF'D FUNCTION ERROR

Im Programm wird eine Funktion aufgerufen, die nicht definiert ist. Es ist zu beachten, daß Funktionen vor ihrem ersten Aufruf im Programm definiert sein müssen. Die Definition von Funktionen sollte immer am Programmanfang erfolgen.

STRING TOO LONG ERROR

Die Länge eines Strings ist begrenzt auf 255 Zeichen. Werden Strings zusammengesetzt, so kann es eintreten, daß der zusammengesetzte String mehr als 255 Zeichen umfassen müßte. Es erfolgt dann die Fehlermeldung.

NEXT WITHOUT FOR ERROR

Ein NEXT-Befehl kann vom Programm nur ausgeführt werden, wenn vorher ein FOR durchlaufen wurde. Diese Fehlermeldung erscheint, wenn entweder der FOR-Befehl vergessen wurde oder wenn eine Verzweigung auf eine Zeilennummer innerhalb einer FOR . . . NEXT-Schleife erfolgt (Sprünge innerhalb eines FOR-NEXT-Loops sind möglich).

OUT OF DATA ERROR

- In einer READ-Anweisung werden mehr Daten verlangt als unter DATA vorhanden waren.
- Die vorhandenen Daten sollten noch einmal gelesen werden. Es wurde jedoch keine RESTORE-Anweisung gegeben.

OUT OF MEMORY ERROR

- Die Anzahl der für Programm und Variablen benötigten Bytes übersteigt die Speicherkapazität des Rechners.
- Die Kapazität eines Teils des Arbeitsspeichers wurde überschritten (mehr als 26 GOSUB-Rücksprungadressen).

OVERFLOW ERROR

Diese Fehlermeldung tritt auf, wenn ein Zahlenwert größer als $1,5 \times 10^{38}$ ist.

RETURN WITHOUT GOSUB ERROR

Ein RETURN-Befehl kann vom Programm nur ausgeführt werden, wenn vorher ein GOSUB

durchlaufen wurde. Diese Fehlermeldung erscheint, wenn eine Verzweigung auf eine Zeilennummer innerhalb einer Subroutine erfolgt (Sprünge innerhalb einer Subroutine sind möglich).

REDIM'D ARRAY ERROR

Ein bereits dimensioniertes Feld darf nicht nochmals dimensioniert werden. Eine erneute Dimensionierung kann erst nach dem Befehl CLR (CLR löscht alle Variablen, einschließlich Dimensionierung) erfolgen. RUN schließt CLR mit ein.

BAD DATA ERROR

Diese Fehlermeldung erfolgt, wenn von einem Peripheriegerät falsche Daten (z.B. Alphazeichen statt Zahlen) an den Rechner gegeben werden.

Außerdem gibt es noch zwei Fehlermeldungen, die jedoch nicht im Programm, sondern bei Fehlbedienung auftreten.

ILLEGAL DIRECT ERROR

INPUT und DEFFN können nicht über die Tastatur, sondern nur über das Programm ausgeführt werden.

CAN'T CONTINUE ERROR

Das Programm kann nicht weitergeführt werden, weil

- kein Programm existiert, das zu Ende geführt werden könnte oder
- eine Programmkorrektur vorgenommen wurde oder
- das Programm durch eine ERROR-Meldung gestoppt wurde.

In den meisten Fällen ist ein Fehler bei genauer Untersuchung der angezeigten Fehlerzeile sehr schnell aufzufinden. Es kann jedoch vorkommen, daß in der betreffenden Zeile kein Fehler vorhanden ist. Der Rechner zeigt nur die Zeile an, in der sich ein Fehler bemerkbar macht. Beispielsweise könnte durch einen Rechenfehler in einer Zeile, die lange zuvor ausgeführt wurde, erst in der angezeigten Zeile eine Zahl auftreten, die zum OVERFLOW ERROR führt. In solchen Fällen muß der vorher abgearbeitete Teil des Programmes untersucht werden. Oft ist es zweckmäßig, zur Fehlersuche an verschiedenen Stellen des Programms die Variablen, die zu dem Fehler führen könnten, durch einen eingefügten PRINT-Befehl auf dem Bildschirm darzustellen. Treten hierbei unerwartete Werte auf, läßt sich der Fehler meist sehr schnell einkreisen.

Ein häufiger Fehler ist vergessene Dimensionierung. Da die normalerweise benötigten relativ kleinen Felder nicht dimensioniert werden müssen, kann im Normalfall auf jegliche Dimensionierung verzichtet werden. Dies verleitet dazu, auch in den Fällen keine Dimensionierung vorzunehmen, wo es erforderlich wäre. Die Fehlermeldung BAD SUBSCRIPT ERROR tritt dann in irgendeiner hohen Zeilennummer auf, obwohl diese Zeile fehlerfrei ist. Der eigentliche Fehler ist das Fehlen der Dimensionierung ganz am Anfang des Programms. Dies soll nur ein Beispiel sein, daß Fehlerort und Anzeige, wo Fehler zu suchen ist, nicht übereinstimmen müssen. Dasselbe kann selbstverständlich auch bei vielen anderen Operationen auftreten.

13. Korrektur

Ist ein Fehler im Programm gefunden worden, so muß es entsprechend abgeändert werden. Der PET 2001 bietet folgende Korrekturmöglichkeiten:

- 13.1. Zeilen können gelöscht werden, indem die zu löschende Zeilennummer eingetastet und anschließend die RETURN-Taste gedrückt wird.
- 13.2. Zeilen können überschrieben werden, indem die Zeile mit der betreffenden Zeilennummer neu eingegeben wird.
- 13.3. Zeilen können eingeschoben werden, indem eine Zeile mit einer Zeilennummer zwischen den Zeilennummern der vorhergehenden und der nachfolgenden Zeile eingetastet wird.

Alle diese Operationen können an jeder beliebigen Stelle des Bildschirms durchgeführt werden. Bedingung ist lediglich, daß auf der benutzten Zeile des Bildschirms keine weiteren unerwünschten Zeichen vorhanden sind.

Außer diesen Korrekturmöglichkeiten, die sich auf eine ganze Zeile beziehen, können auch einzelne Zeichen in vorhandenen Zeilen gelöscht, überschrieben oder eingefügt werden. Dies ist besonders wertvoll, wenn sehr lange Zeilen zu korrigieren sind.

Um Korrekturen innerhalb einer Zeile vorzunehmen, wird die Zeile zuerst mittels LIST-Befehl auf dem Bildschirm dargestellt. Anschließend wird der Cursor mit Hilfe der beiden Cursor-Kontrolltasten an die Stelle der Zeile gebracht, an der die Korrektur stattfinden soll. Um ein Zeichen zu ändern, muß der Cursor genau auf diesem Zeichen stehen. Das neue Zeichen wird eingetastet und überschreibt so das bisherige. Um ein Zeichen zu löschen, muß sich der Cursor rechts neben dem zu löschenden Zeichen befinden. Sollen mehrere aufeinanderfolgende Zeichen gelöscht werden, so setzt man den Cursor rechts neben das letzte zu löschende Zeichen. Durch wiederholtes Drücken von DEL wird nun von rechts nach links ein Zeichen nach dem anderen gelöscht. Sollen Zeichen eingeschoben werden, so ist der Cursor über das Zeichen zu bringen, vor dem ein Zwischenraum einzufügen ist. Bei jedem Druck auf INST wird der Teil der Zeile, der mit dem unter dem Cursor befindlichen Zeichen beginnt, um eine Position nach rechts verschoben, während der Cursor an der gleichen Stelle bleibt. Man kann daher unmittelbar nach dem Schaffen des nötigen Zwischenraumes mit dem Schreiben beginnen.

Nach der Korrektur wird RETURN gedrückt. Der Cursor kann sich hierbei an irgend einer beliebigen Stelle in der Zeile befinden.

Etwas schwieriger als die normale Korrektur ist die Programmkorrektur von Text, das heißt, von Zeichen zwischen Anführungszeichen. Wird eine Zeile geschrieben, so kann der Cursor nach einem Anführungszeichen durch die Cursor-Kontrolltasten nicht mehr bewegt werden, sondern die entsprechenden Befehle werden ins Programm geschrieben. Es ist also nicht möglich, ein falsch geschriebenes Zeichen dadurch zu korrigieren, daß man wie sonst üblich den Cursor zurücksetzt und das Zeichen neu schreibt, sondern man muß DEL benutzen. Eine andere Möglichkeit besteht darin, daß man die RETURN-Taste betätigt und den Cursor dann mit den Cursor-Kontrolltasten wieder in die Zeile führt. Jetzt kann der Cursor auch im Text beliebig bewegt werden und es können Zeichen überschrieben werden. Andererseits können jetzt natürlich keine Cursorbewegungen ins Programm geschrieben werden. Sollen also jetzt solche Cursorbewegungen geändert werden, so geht dies nicht durch Überschreiben, sondern nur durch Löschen des alten Zeichens mit DEL und anschließendem INST. Nach INST werden Cursorbewegungen stets programmiert. Die selben Regeln wie für Cursorbewegungen gelten für HOME, CLR (Bildschirm löschen), RVS und OFF.

Der sicherste Weg Text zu korrigieren, ist speziell für Anfänger das Neuschreiben der Zeile. Man sollte sich jedoch die andere Methode der Textkorrektur baldmöglichst aneignen, da sie speziell bei längeren Texten sehr viel weniger Mühe macht.

Maschinensprache

Mittels PEEK und POKE ist der Inhalt jedes einzelnen Bytes des Arbeitsspeichers zugänglich. Mit dem Befehl PEEK wird der Inhalt der angegebenen Adressen gelesen.

Beispiel:

Der Inhalt der Adresse 59468 soll auf dem Bildschirm dargestellt werden. Dies erfolgt mittels

?PEEK (59468)

Nach Bestätigen der RETURN-Taste erscheint auf dem Bildschirm die Zahl 12 (diese Adresse ist für die Umschaltung des Zeichengenerators zuständig; 12 bedeutet, daß auf graphische Zeichen geschaltet ist).

Mit dem Befehl POKE kann das adressierte Byte mit einem anderen Inhalt überschrieben werden. Beispiel:

PET soll nun auf Kleinschreibung umgeschaltet werden. Der hierfür notwendige Befehl lautet

POKE 59468,14

Über einen neuen PEEK-Befehl kann man sich vergewissern, daß der neue Inhalt dieser Adresse 14 ist. Durch Schreiben von Buchstaben bei gedrückter SHIFT-Taste sieht man, daß die Umschaltung erfolgt ist.

Sowohl die Adressen als auch (bei POKE) der zu schreibende Wert können Variablen sein.

Der zulässige Bereich für die Adressen ist 0 — 65535, der für die Inhalte 0 — 255.

Weiteres zur Maschinensprache siehe Anhang.

BASIC-Befehle und Anweisungen

Systemkommandos

Systemkommandos sind Anweisungen an den Rechner, die normalerweise nicht programmiert werden. Zwar lassen sie sich auf dem PET 2001 zum großen Teil programmieren, dies ist jedoch nur in einzelnen Fällen sinnvoll.

RUN	RUN	Führt das Programm aus, beginnend mit der niedrigsten Zeilennummer. RUN ist programmierbar.
	RUN 120	Programmausführung beginnt bei der angegebenen Zeilennummer.
	RUN löscht sämtliche Variablen einschließlich Dimensionierung und bewirkt ein RESTORE. Der Befehl RUN muß mit den Buchstaben R U N eingegeben werden. Er darf nicht mit der Taste "RUN" verwechselt werden!	
CONT	CONT	Setzt die Programmausführung von der momentanen Position aus fort. Man kann ein laufendes Programm durch Betätigen der Taste STOP anhalten, von Hand Variablen auf den Bildschirm bringen, Variablen neu zuordnen, etc. und dann die Programmausführung mittels CONT fortsetzen. Nach Programmkorrekturen oder nach Anhalten des Programms mit einer Fehlermeldung ist CONT nicht möglich. CONT ist nicht programmierbar.
GOTO	GOTO 160	GOTO ist kein eigentliches Systemkommando, sondern steht normalerweise im Programm. Es kann jedoch als Systemkommando verwendet werden. Die Programmausführung beginnt dann bei der angegebenen Zeile. Es werden jedoch keine Variablen gelöscht, wie es bei RUN 160 der Fall wäre.
LIST	LIST	Das gesamte Programm wird aufgelistet. Wird bei längeren Programmen während des Listens die Taste RVS gedrückt, so wird die Geschwindigkeit der Auflistung verringert, sobald der untere Bildschirmrand erreicht ist.
	LIST 40	Es wird die angegebene Zeile dargestellt.
	LIST —120	Das Programm wird ab der niedrigsten Zeilennummer bis einschließlich der angegebenen Zeile aufgelistet.
	LIST 120—	Das Listen erfolgt ab der angegebenen Zeile bis zum Ende des Programms.
	LIST 40—80	Es wird nur der Programmteil von der ersten angegebenen Zeilennummer bis einschließlich der zweiten angegebenen Zeilennummer gelistet.
LIST ist programmierbar. Nach Ausführung dieses Befehles wird die Programmausführung unterbrochen.		
NEW	NEW	Löscht den gesamten Programmspeicher einschließlich der Variablen. NEW ist programmierbar. Das Programm löscht sich dann selbst.
CLR	CLR	Löscht sämtliche Variablen einschließlich Dimensionierung sowie sämtliche Rücksprungadressen. CLR ist programmierbar. Dies ist beispielsweise zweckmäßig, wenn für eine neue Aufgabe ein Feld neu dimensioniert werden muß und nicht mit RUN frisch begonnen werden soll. CLR muß mit den Buchstaben CLR eingegeben werden und darf nicht mit der Taste "CLR" verwechselt werden!

BASIC-Befehle

LET	LET G=4	Bei früheren BASIC-Versionen war LET zur Zuordnung notwendig. Es kann auch beim PET 2001 verwendet werden, so daß vorhandene Programme ohne Änderung übernommen werden können. Im angeführten Beispiel genügt es jedoch G=4 zu schreiben.
READ	READ A,A\$,B,C	Daten, die unter DATA im Programm stehen, werden in die betreffenden Variablen eingelesen. Bei den Daten kann es sich sowohl um Zahlenwerte als auch um alphanumerische Texte handeln. Daten, die in einem DATA-Befehl stehen, können mittels mehrerer READ-Anweisungen zugeordnet werden. Umgekehrt können auch mit Hilfe eines READ-Befehles die Daten mehrerer DATA-Befehle zugeordnet werden.
DATA	DATA3, Text,4,5	Die Daten, die durch READ zugeordnet werden sollen, können in einem oder mehreren DATA-Befehlen an beliebiger Stelle ins Programm geschrieben werden. Die Zuordnung erfolgt in der Reihenfolge, in der sie im Programm stehen. Text braucht nur dann in Anführungszeichen gesetzt zu werden, wenn er Kommata oder Doppelpunkte oder Zwischenräume am Anfang oder Ende enthält.
RESTORE	RESTORE	Bei jeder READ-Anweisung wird jeweils der nächste unter DATA stehende Wert benutzt. Sind sämtliche vorhandenen Daten eingelesen, so erfolgt bei der nächsten READ-Anweisung die Fehlermeldung OUT OF DATA. In vielen Fällen sollen jedoch die vorhandenen Daten mehrfach verwendet werden. Durch den Befehl RESTORE wird erreicht, daß das Lesen wieder beim ersten Wert beginnt.
IF . . THEN	IFA=OTHEN200	Logischer Vergleich. Im Beispiel wird ein Sprung zur Zeile 200 durchgeführt, wenn der Vergleich wahr ist. Im anderen Falle wird das Programm mit der nächsten Zeile fortgesetzt.
	IFB=2THEN?C	Außer Sprüngen können nach einem Vergleich auch andere Befehle verwendet werden. Auch bei mehreren Befehlen, die innerhalb der selben Zeile auf THEN folgen, wird jeweils der gesamte folgende Zeileninhalt bei „wahrem“ Vergleich ausgeführt, bei „falschem“ Vergleich weggelassen.
FOR . . NEXT	FORI=1T020 ... NEXT	Die zwischen FOR und NEXT befindlichen Befehle werden so oft durchlaufen, wie Start- und Endwert angeben (im Beispiel 20mal) (Schleife, FOR . . NEXT—Loop). Die angegebene Variable (im Beispiel I) muß eine gewöhnliche Variable, keine Ganzzahl-Variable sein. Die Angabe dieser Variablen nach NEXT, die bei vielen BASIC-Versionen notwendig ist, kann im PET 2001 entfallen.
STEP	FORK=5TO2STEP —5	Mittels STEP kann eine beliebige Schrittweite (positiv oder negativ) vorgegeben werden. Im Beispiel wird die Schleife nacheinander mit den Werten 5, 4,5, . . . , 2,5,2 durchlaufen.
	FORI=3TO6 FORJ=2TOB ... NEXT J,I	Bei mehreren ineinandergeschachtelten FOR . . NEXT-Loops (mit oder ohne STEP) genügt eine einzige NEXT-Anweisung, wenn die betreffenden Variablen durch Komma getrennt sind. FOR . . NEXT-Loops können selbst mehrfachgeschachtelt in eine Zeile geschrieben werden.

GOTO	GOTO245	Unbedingter Sprung zu der angegebenen Zeile. GOTO kann auch zu bedingten Sprüngen (nach Vergleichen) verwendet werden. Bei mehreren Befehlen pro Zeile muß GOTO der letzte Befehl sein, da Befehle nach GOTO nie ausgeführt werden.
GOSUB	GOSUB320	Sprung zu einer Subroutine. Beim Durchlaufen des GOSUB-Befehles wird eine „Rücksprungadresse“ gesetzt. Wird in der Subroutine der Befehl RETURN erreicht, so erfolgt die Fortsetzung der Programmausführung bei der Rücksprungadresse. Die Rücksprungadresse steht unmittelbar beim GOSUB-Befehl, so daß in einer Zeile mehrere GOSUB-Befehle stehen können, die alle ausgeführt werden. Innerhalb einer Subroutine kann eine weitere Subroutine gerufen werden, von der aus wiederum eine Subroutine gerufen werden kann Die mögliche „Schachtelungstiefe“ beträgt 26 Ebenen. Subroutinen können an beliebiger Stelle des Programmes stehen und beliebig oft, auch von verschiedenen Stellen des Programms aus gerufen werden.
RETURN	RETURN	RETURN ist der letzte Befehl in einer Subroutine und bewirkt den Rücksprung zum letzten durchlaufenen GOSUB Befehl. Wird RETURN erreicht, ohne daß ein entsprechender GOSUB-Befehl durchlaufen wurde, so erscheint die Fehlermeldung RETURN WITHOUT GOSUB ERROR. Dies erfolgt manchmal dadurch, daß anstelle eines GOSUB-Befehles ein GOTO-Befehl verwendet wurde. Meist ist die Ursache jedoch darin zu suchen, daß Subroutinen häufig ans Ende des Programms geschrieben werden und dann vergessen wird, vor deren Beginn ein STOP oder END zu setzen. Nach Abarbeitung des gesamten Programms „fällt“ der Rechner dann in die Subroutine. RETURN muß Buchstabe für Buchstabe eingegeben werden und darf nicht mit der Taste „RETURN“ verwechselt werden!
ON . . GOTO	ONF GOTO50, 60,70	Mittels dieser „berechneten“ Verzweigung kann vom Wert einer Variablen abhängig gemacht werden, an welcher Stelle das Programm fortgesetzt werden soll. Im Beispiel wird das Programm bei F=1 mit der Zeile 50, bei F=2 mit der Zeile 60 und bei F=3 mit der Zeile 70 fortgesetzt. In allen anderen Fällen wird das Programm ohne Sprung fortgesetzt. Statt einer Variablen kann auch ein mathematischer Ausdruck verwendet werden.
ON . . GOSUB	ONJGOSUB55, 65,75	Genau wie bei der berechneten Verzweigung kann bei Subroutinen verfahren werden. Sowohl bei berechneten Verzweigungen als auch bei berechneten Subroutinen ist die Anzahl der „Zweige“ nur begrenzt durch die verfügbare Zeilenlänge. Reicht diese nicht aus, um alle Zeilennummern unterzubringen, so können diese auf zwei oder mehrere Zeilen verteilt werden. In der ersten Zeile, die 8 Zeilennummern enthält, kann dann stehen: ONK . . , auf der nächsten dann: ONK—8. . . Auf diese Weise kann zu beliebig vielen Punkten verzweigt werden.

PRINT	PRINTA,X\$,B	Der PRINT-Befehl bewirkt die Darstellung der angegebenen Variablen auf dem Bildschirm, und zwar in dem Format, das durch die Verwendung von Komma bzw. Semicolon vorgegeben ist (Näheres siehe Kapitel Programmierung — 6. Ausgabeformat). Statt PRINT kann ? benutzt werden. Wird das Programm aufgelistet, sieht man, daß ? durch PRINT ersetzt wurde.
DIM	PRINT DIMA(6,6) DIMC\$(30) DIME(N)	PRINT ohne Variablenangabe bewirkt den Zeilenvorschub. Felder, die in einer Dimension mehr als 11 Elemente (0—10) enthalten, müssen vor ihrer ersten Verwendung dimensioniert werden. Nachdem eine automatische Dimensionierung bei der ersten Verwendung eines Elementes des Feldes erfolgt und eine Dimensionierung nur einmal möglich ist, ist es ratsam, die Dimensionierung ganz am Anfang des Programmes vorzunehmen. Die Dimensionierung kann außer mit Konstanten auch mit Variablen oder mathematischen Ausdrücken vorgenommen werden. Die Anzahl der Dimensionen ist nur durch die verfügbare Speicherkapazität begrenzt.
END	END	Im allgemeinen letzte Anweisung eines Programms. Kann weggelassen werden. END kann auch mehrfach innerhalb eines Programmes verwendet werden. Bewirkt Programmhalt ohne Meldung „BREAK IN . . .“ wie bei STOP. Nach END kann das Programm mittels CONT fortgesetzt werden.
REM	REM TEXT	Erlaubt Bemerkungen in das Programm zu schreiben (keine Wirkung im Programm). Wird REM gemeinsam mit anderen Befehlen in einer Zeile verwendet, so muß REM der letzte Befehl der Zeile sein. Befehle nach REM werden nicht ausgeführt!
INPUT	INPUTA INPUTC,F\$ INPUT"TEXT";A	Erlaubt Eingabe von Werten über die Tastatur (siehe Kapitel Programmierung — 4. Eingabe).
DEFFN	DEFFNY(X)=A*B	Oft gebrauchte Funktionen können definiert und dann vom Programm gerufen werden. Im Beispiel ist Y der Name der Funktion (muß einer der 26 Buchstaben sein), „(X)“ ist eine „Füll-Variable“. DEFFN muß im Programm eine niedrigere Zeilennummer besitzen als der erste Ruf nach dieser Funktion. Maximale Länge der definierten Funktion ist eine Zeile.
FN	?FNY(X)	Funktion wird aufgerufen. Im Beispiel wird A*B ausgegeben.
GET	GETV GETV\$	Eine einzelne Ziffer wird vom Tastenfeld in die Variable V gelesen. Ein einzelnes Zeichen wird vom Tastenfeld in V\$ eingelesen. Nähere Ausführungen zu GET sind in Kapitel Programmierung — 4. Eingabe zu ersehen.
STOP	STOP	Beendet ein laufendes Programm. Bewirkt die Meldung BREAK IN . . . (Zeilennummer). Nach STOP kann das Programm mittels CONT fortgesetzt werden.

Input-Output-Anweisungen

OPEN	OPEN1,1,1	Eröffnung eines logischen File (jedes Peripheriegerät wird als File angesprochen). Der erste Parameter ist die File-Nr. (0 bis 255, max. 10 Files gleichzeitig offen), der zweite die Gerätenummer (0=Tastatur, 1=eingebaute Kasette, 2=Kassette, 3=Bildschirm, 4—15=externe Geräte), der dritte bestimmt, ob Ein- oder Ausgabe (0=Lesen, 1=Schreiben, 2=Schreiben mit zusätzlichem End-of-Tape-Zeichen). Im Beispiel wird File 1 zur Aufzeichnung auf die eingebaute Kasette eröffnet.
	OPEN1	Der Ersatzwert für den 2. Parameter ist 1, für den dritten 0. Beispiel: File 1 wird zum Lesen von der eingebauten Kasette eröffnet.
	OPEN1,1,1, "Daten"	Files können mit Namen bezeichnet und dann gezielt geladen werden. Namen dürfen nicht in anderen Namen enthalten sein: Das File ZINSESZINS wird auch geladen, wenn ZINS gesucht wird!
CLOSE	CLOSE1	Schließen eines File. Eröffnete Files müssen stets wieder geschlossen werden!
SAVE	SAVE	Aufzeichnung eines Programms auf die eingebaute Kasette.
	SAVE"NAME"	Aufzeichnung eines Programms auf die eingebaute Kasette mit Namen.
LOAD	LOAD	Laden des nächsten auf der Kasette befindlichen Programms.
	LOAD"NAME"	Laden des mit Namen bezeichneten Programms von der eingebauten Kasette. LOAD ohne zusätzlichen Namen kann ersetzt werden durch Drücken der Taste RUN (SHIFT und STOP). In diesem Falle wird sofort mit der Programmausführung begonnen, das heißt, der so gegebene LOAD-Befehl beinhaltet den Befehl RUN.
VERIFY	VERIFY	Vergleich des im Rechner gespeicherten Programmes mit dem nächsten auf der Kasette befindlichen Programm.
	VERIFY"NAME"	Vergleich des unter diesem Namen auf der Kasette befindlichen Programms mit dem in der Maschine gespeicherten.
Bei allen Kassettenoperationen mit Namen gilt die selbe Regelung wie bei OPEN: Namen dürfen nicht in anderen Namen enthalten sein.		
PRINT #	PRINT# 1,A,B,C	Die Variablen A, B und C werden im File 1 ausgegeben auf das Gerät, das im OPEN-Befehl dem File 1 zugeordnet wurde. Dieses File muß ein Ausgabefile sein (3. Parameter 1 oder 2)
INPUT#	INPUT# 2,X,Y,Z	Vom File 2 und dem im OPEN-Befehl zugeordneten Gerät werden Daten in X, Y und Z eingelesen. Dieses File muß ein Eingabefile sein (3. Parameter 0). Soll INPUT ausgeführt werden, ohne daß das File eröffnet wurde, erscheint FILE NOT OPEN ERROR.
GET #	GET# 2,A\$	Ein einzelnes Zeichen wird vom File 2 in A\$ eingelesen. Zu beachten: Im Gegensatz zu INPUT # wird GET # durch das vom Band gelesene Zeichen CR (das nach jeder PRINT-Anweisung auf das Band geschrieben wird) <i>nicht</i> beendet! Es muß also gegebenenfalls geprüft werden, ob es sich um das ASCII-Zeichen 13 handelt bzw., ob ST<>0 ist.

String-(Zeichenketten)-Befehle

LEN	A=LEN(A\$)	Die tatsächliche Länge des Strings (Anzahl der Zeichen einschließlich der Leerstellen) wird festgestellt und im Beispiel in A gespeichert.
VAL	H=VAL(B\$)	Beginnt der String mit Zahlen, Leerstellen, +, — oder einem Dezimalpunkt, so wird der Zahlenwert in eine gewöhnliche Variable umgewandelt. Beginnt der String mit nicht-numerischen Zeichen, so ist VAL=0.
STR\$	N\$=STR\$(X)	Eine normale Zahl wird in einen String umgewandelt.
ASC	?ASC(D\$)	Der ASCII-Code des ersten Zeichens im String wird ermittelt und im Beispiel auf dem Bildschirm dargestellt.
CHR\$	A\$=CHR\$(77)	Das zum ASCII-Code 77 gehörende Zeichen (M) wird ermittelt und in A\$ gespeichert.
LEFT\$	L\$=LEFT\$(A\$,3)	Die ersten (hier 3) Zeichen von A\$ werden in L\$ gespeichert. Hat A\$ weniger Zeichen als spezifiziert, so wird der gesamte String dargestellt.
RIGHT\$?RIGHT\$(B\$,5)	Die letzten (hier 5) Zeichen von B\$ werden auf den Bildschirm gebracht. Hat B\$ weniger Zeichen als spezifiziert, so wird der gesamte String dargestellt.
MID\$	M\$=MID\$(C\$,4)	Vom 4. Zeichen ab wird C\$ in M\$ gespeichert.
	M\$=MID\$(C\$,4,2)	2 Zeichen, beginnend mit dem 4. Zeichen, aus C\$, werden in M\$ gespeichert.
+	G\$=A\$+B\$	Aus A\$ und B\$ wird ein neuer String (G\$) erzeugt. A\$ und B\$ bleiben unverändert. Gegebenenfalls ist sicherzustellen, daß der erzeugte String nicht länger wird als 255 Zeichen.

Logische Operatoren: AND, OR und NOT

Mit diesen 3 Operatoren können Boole'sche Operationen durchgeführt werden. Sie wandeln die Argumente zunächst in 16—Bit Binärzahlen um (dezimaler Bereich von —32768 bis 32767). Anschließend werden die jeweiligen Operationen Bit für Bit vorgenommen. Im Falle NOT ist das Ergebnis das binäre Komplement, dezimal bedeutet dies: NOT X=—(X+1). Bei AND und OR ist das Ergebnis jeweils die Zahl, die sich bei AND- bzw. OR-Verknüpfung jeweils gleichwertiger Bits ergibt.

Beispiel:

	binär	dezimal
Argument 1	1 0 1 1 0 0 1	89
Argument 2	1 0 0 1 1 0 1	77
AND	1 0 0 1 0 0 1	73
OR	1 0 1 1 1 0 1	93
Argument	1 0 1 1 0 0 1	89
NOT	1 1 1 1 1 1 1 1 0 1 0 0 1 1 0	-90

Dieser bitweise Vergleich führt einerseits dazu, daß bei einer Entscheidung, die wahr sein soll, wenn sowohl A als auch B von 0 verschieden sind, nicht geschrieben werden darf:

IFAANDBTHEN . . . (2AND4=0!). Die Schreibweise muß sein: IFA«»OANDB«»OTHERN . . . Der Vorteil dieser Methode des Bit-weisen Vergleiches ist jedoch darin zu suchen, daß diese Operatoren nicht nur in logischen Vergleichen verwendet werden können, sondern daß auch Vergleiche oder Operationen mit einzelnen Bits durchgeführt werden können.

Beispiele:

?IFA=4ANDB=5THEN

?IFA=2ORB=4THEN

N=NOTA

Kassettenoperationen

Es ist nicht notwendig, besonders teure HIFI- oder spezielle EDV-Kassetten zu verwenden. Man sollte, um übermäßige Suchzeiten zu vermeiden, C 90- oder C 120-Kassetten nicht benutzen. Es lassen sich auf diesen zwar sehr viel mehr Programme oder Daten speichern, die Praxis zeigt jedoch, daß es viel zweckmäßiger ist, mehrere kurze Bänder statt eines langen zu verwenden. Selbst C 60-Kassetten bieten reichlich Speicherplatz. Um nicht Gefahr zu laufen, daß öfter als notwendig die Meldung READ ERROR auftritt, sollte man vermeiden, die billigsten Kassetten zu verwenden.

Mittels der eingebauten Kassette können Programme und Daten aufgezeichnet und in den Speicher eingelesen werden.

Programme

Um ein Programm auf der eingebauten Kassette aufzuzeichnen, muß die Kassette, auf die das Programm geschrieben werden soll, eingelegt werden. Es muß REW (rewind) gedrückt werden, um die Kassette zurückzuspulen. Danach ist SAVE einzugeben und die RETURN-Taste zu drücken (soll ein Programm mit einem Namen bezeichnet werden, um es später gezielt laden zu können, so muß SAVE "NAME" eingegeben werden). Auf dem Bildschirm erscheint die Aufforderung: PRESS PLAY & RECORD ON TAPE # 1. Nach dem Drücken dieser beiden Tasten bringt der Bildschirm die Mitteilung: OK WRITING (bei einem mit Namen gekennzeichneten Programm ist auf dem Bildschirm zu sehen: OK WRITING NAME). Nach der Beendigung der Aufzeichnung erscheint READY und die Kassette hält an. Zum Laden eines Programmes von der Kassette gibt es zwei Möglichkeiten. Wird die Taste RUN (SHIFT und STOP) gedrückt, so erscheint auf dem Bildschirm: LOAD, PRESS PLAY ON TAPE # 1. Bestätigt man die PLAY-Taste, kann man folgende Mitteilung auf dem Bildschirm lesen: OK SEARCHING. Sobald der Rechner den ersten Programmanfang nach der augenblicklichen Bandposition gefunden hat, zeigt der Bildschirm: FOUND (bzw. FOUND NAME), LOADING. Nach Beendigung des Ladens wird das Programm automatisch gestartet. Soll ein Programm gezielt eingeladen werden, so ist es erforderlich LOAD "NAME" einzugeben, der Rechner antwortet wieder mit PRESS PLAY ON TAPE 1. Anschließend erfolgen die Meldungen: OK, SEARCHING FOR NAME, FOUND NAME und LOADING. Werden auf der Suche nach dem aufgerufenen Programm andere Programme gefunden, so werden diese auf dem Bildschirm registriert, jedoch nicht geladen. Erst wenn das gesuchte Programm erreicht ist, zeigt der Bildschirm LOADING und es wird geladen. In diesem Fall muß das Programm jedoch durch RUN gestartet werden.

Beispiele:

```
LOAD"PLOTV"
PRESS PLAY ON TAPE #1
OK
SEARCHING FOR PLOTV
FOUND PLOTH
FOUND PLOTV
LOADING
READY.
```

Treten beim Ladevorgang Fehler auf (schlechtes oder verschmutztes Band), so erfolgt die Meldung LOAD ERROR. Das Laden muß dann wiederholt werden.

Bei der Namensgebung eines Programmes ist zu beachten, daß ein Name nicht bereits im Namen eines anderen Programmes enthalten sein darf. So wird beispielsweise das Programm ZINSES-ZINS auch geladen, wenn nach dem Programm ZINS gesucht wird.

Soll das Programm auf eine Kassette überspielt werden, die beispielsweise schon zwei Programme enthält, kann man folgendermaßen vorgehen: Kassette rückspulen, VERIFY (es erscheint ?VERIFY ERROR, siehe unten), und nochmals VERIFY. Das Band steht jetzt am Ende des zweiten Programms. Mit SAVE kann jetzt das neue aufgenommen werden.

Mit dem Befehl VERIFY kann geprüft werden, ob das im Rechner vorhandene Programm mit einem auf der Kassette aufgezeichneten identisch ist. Vorzugehen ist hier genau wie bei LOAD. Sind die Programme identisch, so antwortet der Rechner mit OK, im anderen Falle bringt er die Mitteilung: VERIFY ERROR.

Daten

Zur Eingabe von Daten muß ein logisches File mit seiner File-Nummer eröffnet werden. Der Befehl PRINT # hat nun auf dieses File die gleiche Wirkung wie der Befehl PRINT für den Bildschirm. Der Befehl INPUT # entspricht dem Befehl INPUT (der Befehl INPUT # bezieht sich auf das File in der gleichen Weise wie der Befehl INPUT auf das Tastenfeld). Zu beachten ist folgendes:

1. PRINT muß immer Buchstabe für Buchstabe eingegeben werden. PRINT darf in diesem Fall nicht durch ? ersetzt werden. ? # n führt zur Fehlermeldung SYNTAX ERROR.
2. Für PRINT -Operation muß das logische File ein Ausgabefile sein, das heißt, der dritte Parameter muß 1 oder 2 lauten. Entsprechend muß für INPUT -Operationen ein Eingabefile mit 0 als dritten Parameter definiert sein.
3. Die File-Nummer kann eine beliebige Zahl im Bereich von 1 bis 255 sein. Der zweite Parameter muß für die eingebaute Kassette 1 lauten.
4. Wird nur die File-Nummer angegeben, so wird als zweiter Parameter 1 (eingebaute Kassette) und als dritter Parameter 0 angenommen, das heißt, OPEN 4 ist gleichbedeutend mit OPEN 4,1,0.
5. Eröffnete File müssen stets wieder geschlossen werden. Der entsprechende Befehl ist CLOSE n. Der Befehl, das File 4 zu schließen, nachdem es mit dem Befehl OPEN 4,1,0 eröffnet wurde, lautet: CLOSE 4.

Die zur Aufzeichnung von Daten auf die Kassette und das Wiedereinlesen notwendigen Operationen werden am besten durch das folgende Programm erklärt:

```

2000 PRINT "DATEN EINGEBEN"
3000 FOR I=1 TO 5 : INPUT A(I) : NEXT I
4000 PRINT "DATENKASSETTE EINLEGEN UND RUE
C 5000 P
5000 PRINT "DANN /S/ DRUECKEN"
6000 GET A$ : IFA$ <> "S" THEN 60
7000 OPEN 1,1,1
8000 FOR I=1 TO 5 : PRINT #1, A(I) : NEXT I
9000 CLOSE 1
10000 CLR
11000 PRINT "DATENKASSETTE RUECKSPULEN, DA
N 12000 L
13000 GET A$ : IFA$ <> "L" THEN 220
14000 OPEN 3
15000 FOR I=1 TO 5 : INPUT #3, A(I) : NEXT I
16000 CLOSE 3
17000 FOR I=1 TO 5 : PRINT A(I) : NEXT I
18000 READY.

```

Zeile 20 löscht den Bildschirm und fordert zur Dateneingabe auf. In Zeile 30 werden 5 Werte in A(1) bis A(5) eingegeben. Die Zeilen 40 und 50 fordern zum Einlegen der Datenkassette und Rückspulen sowie zum Drücken der Taste „S“ (Schreiben) auf. In Zeile 60 wartet der Rechner bis die Taste „S“ gedrückt wurde. In Zeile 70 wird das File 1 zum Schreiben auf der Kassette 1 eröffnet. In Zeile 80 werden die Werte aus A(1) bis A(5) auf der Kassette aufgezeichnet. Zeile 100 schließt das File 1. Zeile 200 löscht sämtliche Variablen und stellt damit sicher, daß die anschließend gezeigten Werte nur von der Kassette stammen können. Nachdem die in Zeile 210 gegebene Anweisung zum Rückspulen ausgeführt wurde, kann nach Drücken von „L“ (Lesen) die Warteschleife in Zeile 220 verlassen werden und es erfolgt die Eröffnung des Eingabefile 3. Zeile 240 liest die Werte von der Kassette wieder in A(1) bis A(5), die nach Schließen des File 3 in Zeile 260 dann durch die Zeile 270 auf dem Bildschirm dargestellt werden.

Beispiel:

```

DATEN EINGEBEN
.2
.75
.123
.47
.14159
DATENKASSETTE EINLEGEN UND RUECKSPULEN
DANN 'S' DRUECKEN

PRESS PLAY & RECORD ON TAPE #1
OK
DATENKASSETTE RUECKSPULEN, DANN 'L'

PRESS PLAY ON TAPE #1
OK
.2
.75
.123
.47
.14159

READY.

```

Wird statt des normalen Feldes A() das Feld A\$() verwendet, so können beliebige Zeichen aufgezeichnet werden.

Beispiel:

```

20 PRINT"DATEN EINGEBEN"
30 FOR I=1 TO 5: INPUT A$(I): NEXT I
40 PRINT"DATENKASSETTE EINLEGEN UND RUECKSPULEN"
50 PRINT"DANN 'S' DRUECKEN"
60 GET A$: IFA$(C) = "S" THEN 60
70 OPEN 1, 1, 1
80 FOR I=1 TO 5: PRINT#1, A$(I): NEXT I
100 CLOSE 1
200 CLR
210 PRINT"DATENKASSETTE RUECKSPULEN, DANN 'L'"
220 GET A$: IFA$(C) = "L" THEN 220
230 OPEN 3
240 FOR I=1 TO 5: INPUT#3, A$(I): NEXT I
260 CLOSE 3
270 FOR I=1 TO 5: PRINT A$(I): NEXT I
READY.

```

```

DATEN EINGEBEN
MAIER
MUELLER
SCHMIDT
SCHULZE
BAUER
DATENKASSETTE EINLEGEN UND RUECKSPULEN
DANN 'S' DRUECKEN

PRESS PLAY & RECORD ON TAPE #1
OK
DATENKASSETTE RUECKSPULEN, DANN 'L'

PRESS PLAY ON TAPE #1
OK
MAIER
MUELLER
SCHMIDT
SCHULZE
BAUER

READY.

```

Statt INPUT kann GET verwendet werden. Hierbei ist jedoch zu beachten, daß die GET-Anweisung nicht wie INPUT durch ein „carriage return“ (ASCII 13) beendet wird. Um zu verhindern, daß auf der Kassette immer weitergelesen wird — was zur Folge haben kann, daß der Rechner auf keine Taste mehr reagiert und nur durch Aus- und Einschalten wieder betriebsfähig gemacht werden kann — muß jedes gelesene Zeichen überprüft werden, ob es sich um einen CR-Befehl handelt. Das abgeänderte Programm sieht folgendermaßen aus:

```

20 PRINT"DATEN EINGEBEN"
30 FOR I=1 TO 5: INPUT A$(I): NEXT I
40 PRINT"DATENKASSETTE EINLEGEN UND RUE
CKSPULEN"
50 PRINT"DANN 'S' DRUECKEN"
60 GET A$: IF A$(">"S)" THEN 60
70 OPEN 1, 1, 1
80 FOR I=1 TO 5: PRINT#1, A$(I): NEXT I
100 CLOSE 1
200 CLR
210 PRINT"DATENKASSETTE RUECKSPULEN, DA
NN 'L'"
220 GET A$: IF A$(">"L)" THEN 220
230 OPEN 3
240 FOR I=1 TO 5
242 GET#3, Q$: IF ASC(Q$(">"))=13 THEN A$(I)=A$(
I)+Q$: GOTO 242
244 NEXT I
260 CLOSE 3
270 FOR I=1 TO 5: PRINT A$(I): NEXT I
READY.

```

Der FOR . . . NEXT-Loop in Zeile 240 wird in 3 Zeilen aufgeteilt. In der neuen Zeile 242 wird durch GET 3 Zeichen für Zeichen in Q\$ übernommen und — falls es sich nicht um CR (ASCII) handelt — zum String A\$(I) hinzugefügt. Bei CR wird das Programm mit Zeile 244 fortgesetzt und der nächste String A\$(I) begonnen.

Peripheriegeräte

IEEE 488-Bus (IEC-Bus)

Das Interface nach IEEE 488 ist bereits eingebaut. Außer den 8 Datenleitungen DI01 bis DI08 sind 3 Handshake-Leitungen und 5 Steuerleitungen vorhanden. Das im PET eingebaute IEEE 488 Interface weist folgende Besonderheiten auf:

1. REN (remote enable) ist immer auf low.
2. IFC (interface clear) wird stets beim Einschalten ausgelöst (Impuls ca. 100 ms).
3. PET erwartet innerhalb von 60 ms Antwort vom Peripheriegerät. Bei langsamen Peripheriegeräten muß durch den Befehl WAIT der Programmablauf bis zur Rückmeldung angehalten werden. Diese Eigenschaft ermöglicht eine Datenübernahme bzw. -übergabe auch mit Geräten, die der IEEE-Norm nicht voll entsprechen.

Der Anschlußstecker entspricht den elektrischen Spezifikationen nach IEEE 488, aber nicht den mechanischen. Dies ist jedoch kein Nachteil, da sich IEEE-Bus und IEC-Bus bei gleichen elektrischen Spezifikationen im Stecker unterscheiden.

Steckerbelegung:

1. DI01	7. NRFD	A. DI05	H. \perp (NRFD)
2. DI02	8. NDAC	B. DI06	J. \perp (NDAC)
3. DI03	9. IFC	C. DI07	K. \perp (IFC)
4. DI04	10. SRQ	D. DI08	L. \perp (SRQ)
5. EO1	11. ATN	E. REN	M. \perp (ATN)
6. DAV	12. \perp (CHASSIS)	F. \perp (DAV)	N. \perp (DATA)

Steuerbefehle für den IEEE-Bus

Zusammenfassung der für Input-/Output-Operationen notwendigen BASIC-Befehle:

OPEN
CLOSE
INPUT
GET
PRINT
CMD
WAIT

Peripheriegeräte, die am IEEE-Bus angeschlossen sind, werden als logische Files angesprochen. Bei der Eröffnung des Files gilt folgendes:

OPEN A,B,C,"TEXT"

A ist die File-Nummer (logische Adresse) im Bereich von 1 bis 255. Alle INPUT -, GET -, PRINT -, CMD - und CLOSE-Befehle beziehen sich auf diese logische Adresse.

B ist die Geräteadresse (Primäradresse) und liegt im Bereich von \emptyset bis 15. C ist die Sekundäradresse und braucht nur bei Bedarf angegeben werden. Der Bereich der Adresse erstreckt sich von 0—31. Die Sekundäradresse wird nur bei OPEN- und CLOSE-Befehlen ausgegeben. Erfolgt die Ausgabe, so werden die Bits 6 und 7 gesetzt. Wird „TEXT“ eingegeben, so *muß* die Sekundäradresse angegeben werden. In diesem Falle wird sie folgendermaßen ausgegeben: Bei der Ausführung von OPEN und CLOSE wird Bit 8 gesetzt, bei OPEN zusätzlich noch Bit 5.

Der OPEN-Befehl zieht die ATN-Leitung auf low, das heißt, beim OPEN-Befehl werden Kommandos ausgegeben. CLOSE gibt ebenfalls die Sekundäradresse aus. Eröffnete Files müssen stets geschlossen werden!

PRINT hat folgende Wirkung:

1. Das durch die File-Nummer bestimmte Gerät wird als Listener adressiert.
2. Die spezifizierten Daten werden ausgegeben.
3. Es wird unlisten an alle Geräte ausgegeben.

Bei Geräten, die nur Talker sind und daher nur Befehle empfangen können, besteht die Möglichkeit, mit PRINT auch Befehle auszugeben.

INPUT bewirkt analog die Adressierung als Talker, das Einlesen von Daten und die Ausgabe des Untalk-Befehles.

CMD hat beim PET eine andere Bedeutung als bei manchen anderen Rechnern. CMD dient nicht zur Ausgabe von Kommandos — dies geschieht bei PET im OPEN-Befehl — sondern leitet jegliche Ausgabe auf den IEEE-Bus. So bewirkt der Befehl LIST — wenn vorher CMD ausgeführt wurde — daß das Programm nicht mehr auf dem Bildschirm, sondern auf einem beispielsweise angeschlossenen Drucker gelistet wird. Da CMD den IEEE-Bus aktiv hält, erlaubt es auch Mehrfachadressierung.

WAIT A,B,C wirkt folgendermaßen:

Der Inhalt der in A angegebenen Adresse wird mit C durch die Exklusiv-ODER-Funktion verknüpft (Wird C nicht angegeben, so wird \emptyset angenommen), das Resultat hieraus wird mit B durch die UND-Funktion verknüpft. Das Programm wird solange angehalten, bis das Ergebnis dieser Verknüpfung von Null verschieden ist.

Die UND-Verknüpfung gestattet es, einzelne Bits zu überwachen, indem der Wert des betreffenden Bits als Maske unter B angegeben wird.

Beispiel:

Bit 1 (geringstwertiges Bit) der Adresse 59456 ist die Leitung NDAC. WAIT 59456,1 bewirkt nun ein Anhalten des Programmes bis dieses Bit gesetzt wurde. Das Bit 7 der selben Adresse zeigt den Zustand der Leitung NRFD. Mit dem Befehl WAIT 59456,64 wird also NRFD überwacht. Der Parameter C gestattet die Umkehrung der Logik. Wird er zu 1 gesetzt, so wird die Wartezeit beendet, wenn das betreffende Bit 0 wird.

Dies sind die für den IEEE 488-Bus wichtigen Adressen:

Dezimal Adr.	Bits	IEEE-Leitungen
59424	1—8	DIO 1—8 (INPUT)
59426	1—8	DIO 1—8 (OUTPUT)
59425	4	NDAC (OUTPUT)
59427	4	DAV (INPUT)
	8	SRQ (INPUT)
59408	7	EOI (INPUT)
59456	1	NDAC (INPUT)
	2	NRFD (OUTPUT)
	3	ATN (OUTPUT)
	7	NRFD (INPUT)
	8	DAV (OUTPUT)

(Die in der Aufstellung aufgeführten Leitungen müßten eigentlich überstrichen werden, da es sich um Active Low handelt. Nach der neuen Norm ist dies jedoch nicht mehr erforderlich.)

Bei Datenübertragung von langsamen Peripheriegeräten kann statt der Verwendung von WAIT die Statusvariable ST abgefragt und die übernommenen Werte nur dann weiterverarbeitet werden, wenn ST gleich Null ist.

Die Statusvariable ST ist:

- 0 für gültige Daten
- 1 bei überschrittener Zeit
— bei Übernahme der Daten
ohne Handshake —
- 64 für EOI und
- 128 falls Peripheriegeräte überhaupt nicht geantwortet hat
— ergibt den DEVICE NOT PRESENT ERROR.

Im übrigen wird auf die Bedienungsanleitungen der jeweils anzuschließenden Peripheriegeräte und auf den IEEE-Standard 488 hingewiesen.

USER PORT

Dieser Stecker enthält neben den zum Test mit dem Diagnosestecker notwendigen Leitungen und Ausgängen für einen externen Bildschirm-Monitor (Video, Horizontal- und Vertikalsynchronisation) noch einen 8-Bit Parallel-Ein-/Ausgang mit 2 Handshake-Leitungen.

Steckerbelegung:

1. $\bar{\text{E}}$ (DIGITAL)	7. Kassette 2 (W)	A. $\bar{\text{E}}$ (DIGITAL)	H. PA4
2. TV video	8. Kassette Aufn.	B. CA1	J. PA5
3. IEEE SRQ	9. TV vertical	C. PA \emptyset	K. PA6
4. IEEE EOI	10. TV horizontal	D. PA1	L. PA7
5. Diagnostik	11. $\bar{\text{E}}$ (DIGITAL)	E. PA2	M. CB2
6. Kassette 1 (W)	12. $\bar{\text{E}}$ (DIGITAL)	F. PA3	N. $\bar{\text{E}}$ (DIGITAL)

Die Ein- und Ausgangsleitungen können mittels PEEK und POKE einzeln gelesen bzw. gesetzt werden. Die Adresse ist 59471. Die Zuordnung der einzelnen Bits zu den Leitungen sieht folgendermaßen aus:

PIN	C	D	E	F	H	J	K	L
Bit	1	2	3	4	5	6	7	8

CA1 ist die Eingang-, CB2 die Ausgang-Handshake-Leitung. CB2 kann auch als serieller Ein- oder Ausgang Verwendung finden.

Beim Einschalten sind die Datenleitungen als Eingangsleitung programmiert. CA1 reagiert auf negative Übergänge. Das Umprogrammieren von CA1 erfolgt mit dem Befehl POKE 59468, PEEK (59468) OR 1. Hierdurch wird auf der angegebenen Adresse zusätzlich Bit 1 gesetzt und damit die Logikrichtung von CA1 umgestellt. Die Adresse des Flag-Register ist 59469. Mittels des Befehles WAIT 59469,2 kann dieses Register das Programm bis zum Empfang des Flag-Signals stoppen. Anschließend wird der übermittelte Wert durch D=PEEK (59457) aus der Adresse 59457 in eine Variable übernommen. Es kann sich hierbei um eine Zahl zwischen 0 und 255 handeln. Durch dieses PEEK wird außerdem das CA1-Register zurückgesetzt, sodaß der Rechner zur Übernahme des nächsten Zeichens bereit ist.

Die 8 Datenleitungen können durch den Befehl: POKE 59459,255 zu Ausgangsleitungen umprogrammiert werden. Die Ausgangs-Handshake-Leitung CB2 wird durch den Befehl:

POKE 59468, PEEK (59468) OR 244

auf 1 gesetzt, bzw. durch den Befehl:

POKE 59468, PEEK (59468) AND 31 OR 192

auf Null gebracht.

Interface für zweite Kassette

An dem dafür vorgesehenen Stecker kann eine zweite Kassetteneinheit angeschlossen werden. Die eingebaute Kassette wird mit 1 angesprochen, die zweite mit 2.

Steckerbelegung:

1. Kein Anschluß	1. \bar{M} (DIGITAL)
2. „ „	2. +5
3. „ „	3. Motor
4. „ „	4. Wiedergabe
5. „ „	5. Aufnahme
6. „ „	6. Motorstrom-Überprüfung

Speichererweiterungsanschluß

Durch den Anschluß externer RAM's ist eine Speichererweiterung bis zu 32k möglich. Die notwendigen Leitungen stehen an dem seitlichen Stecker zur Verfügung. Die Stromversorgung des Zusatzspeichers muß extern erfolgen.

Steckerbelegung:

2. A0 Adreß-Bit	42. NS6 Adreß select
4. A1 „	44. NS7 „
6. A2 „	46. NS9 „ „
8. A3 „	48. NSA „ „
10. A4 „	50. NSB „ „
12. A5 „	52. kein Anschluß
14. A6 „	54. Reset
16. A7 „	56. IRQ
18. A8 „	58. Clock (\emptyset 2)
20. A9 „	60. Read/Write
22. A10 „	62. kein Anschluß
24. A11 „	64. kein Anschluß
26. kein Anschluß	66. BD0 Daten-Bit
28. kein Anschluß	68. BD1 „ „
30. kein Anschluß	70. BD2 „ „
32. NS1 Adreß select	72. BD3 „ „
34. NS2 „ „	74. BD4 „ „
36. NS3 „ „	76. BD5 „ „
38. NS4 „ „	78. BD6 „ „
40. NS5 „ „	80. BD7 „ „

A N H A N G

Im folgenden finden Sie die bisher herausgegebenen Mitteilungen für PET-Benutzer.

Es ist dies der Grundstock einer Loseblattsammlung mit Software-, Hardware- und anderen Informationen, die laufend ergänzt und erweitert werden wird.

Die Mitglieder des PET-BENUTZER-CLUBS erhalten alle zukünftigen Ausgaben kostenlos.

I N H A L T

=====

A - ALLGEMEINES

A03	010878	Literatur
A04	151078	Klubbedingungen
A05	151278	Programmangebote
A06	151078	Entfällt
A07	151078	Kursor
A08	150978	Rechengeschwindigkeit
A09	150978	Rechengeschwindigkeit u. Speicherbedarf
A10	150978	Speicherbedarf

ANGEBOTE (VERMITTLUNG)

ANGEBOTE 1	010878		ANGEBOTE 5	151278
ANGEBOTE 2	010878		ANGEBOTE 6	151278
ANGEBOTE 3	150978		ANGEBOTE 7	151278
ANGEBOTE 4	011078			

FW - FIRMWARE

FW01	010778	BASIC-Bugs
FW02	010778	BASIC-Bugs
FW03	010878	OPEN (Kassette)
FW04	011078	Speicherung von Programmen
FW05	011078	Interpretercode
FW06	011078	RAM-Zuteilung und Pointer
FW07	011078	Adressenbelegung
FW08	151278	Page 0
FW09	151278	Page 0 und 1
FW10	151278	Page 2 und 3

HW - HARDWARE

HW01	010978	Diverses
HW02	010978	Kurzinfo Drucker
HW03	010878	Recorder (Service)
HW04	010978	Kurzinfo Datassette

P - LIEFERBARE PROGRAMME

PØ1 15Ø978

PØ2 15Ø978

PC - PROGRAMME DES CLUBS

PCØ1 151278

SW - SOFTWARE

SWØ1	Ø1Ø878	DEFFN, Bildschirm, Files
SWØ2	Ø1Ø778	Kassette INPUT-OUTPUT
SWØ3	Ø1Ø778	Kassette INPUT-OUTPUT
SWØ4	Ø1Ø978	Kassette INPUT-OUTPUT
SWØ5	Ø1Ø978	Kassette INPUT-OUTPUT
SWØ6	Ø1Ø978	Maschinensprache
SWØ7	Ø1Ø978	Maschinensprache
SWØ8	Ø1Ø978	Maschinensprache
SWØ9	Ø1Ø978	Maschinensprache
SW1Ø	Ø1Ø978	Abkürzung für BASIC-Befehle
SW11	151278	Overlays
SW12	151278	Overlays
SW13	151278	Overlays. Abschneiden und runden.



? Wo erhalte ich nähere Unterlagen über den Mikroprozessor 6502 und die Peripheriebausteine?

A: Wir empfehlen vorläufig die Dokumentation zum KIM-Mikrocomputer, insbesondere das Programmierhandbuch (Maschinsprache) und das Hardware-Handbuch (Aufbau des Prozessors, Peripheriebausteine).

COMMODORE Büromaschinen GmbH
Abt. M O S
Frankfurter Str. 171 - 175

6078 Neu Isenburg

Te1.: 06102 8003

Preis inkl. MwSt

KIM 1 Handbuch	deutsch	DM 11,75
	englisch	DM 11,75
Hardware-Handbuch	deutsch	DM 14,00
	englisch	DM 14,00
Programmierhandbuch	deutsch	DM 16,25
	englisch	DM 16,25
KIM 2,3,4 User Manual	englisch	DM 5,25
KIM Assembler Manual	englisch	DM 5,70
KIMath Subroutinen	englisch	DM 8,95
KIM Text Editor	englisch	DM 5,00

? Kann ich das deutsche Handbuch erhalten, ohne den PET 2001 zu bestellen?

A: Ja, es kostet DM 15,-- inkl. MwSt.



PET-BENUTZERCLUB

Der PET-Benutzerclub für Deutschland, Österreich und die Schweiz besteht seit September 1978. Die Mitgliedschaft ist für jeden offen und beginnt nach dem Eingang Ihres Schecks mit dem 1. des folgenden Monats. Sie gilt für jeweils 1 Jahr.

Für den Mitgliedsbeitrag von DM 50,-** erhalten Sie u.a.:

- * die Club-Mitteilungen (6 x pro Jahr und bei Bedarf)
- * Zugang zur Programmbibliothek des Clubs. Die Programme des Benutzerclubs sind nur für Mitglieder erhältlich, im Gegensatz zur Standardsoftware von Commodore. Diese wird generell angeboten.
- * 2 x jährlich senden wir Ihnen ein Verzeichnis der vom Club erhältlichen Programme, ebenfalls 2 x im Jahr die neuesten Ergänzungen.
- * sofern wir Mitteilung davon erhalten, werden wir auch ein Verzeichnis größerer Programmierungsvorhaben erstellen. Dies soll Doppelentwicklungen vermeiden helfen.

Der Preis der Programme des Clubs richtet sich nach dem Umfang. Er setzt sich zusammen aus:

Fixkosten	DM 6,-*	(Kassette, Porto, Arbeitsaufwand)
plus programmabhängige Kosten:		
für Programme kleiner 10kB ..	DM 10,-*	
zwischen 10kB und kleiner 25kB	DM 15,-*	
ab 25kB	DM 20,-*	

Ein Programm mit 13kB würde also beispielsweise DM 6,-* plus DM 15,-* kosten.

* Alle Preisangaben auf dieser Seite behalten 12 % MWSt.



Wenn Sie dem Benutzerclub ein von Ihnen entwickeltes Programm zur Verfügung stellen, erhalten Sie dafür beliebige Programme des Clubs entsprechend dem dreifachen Umfang Ihres Programms.

Beachten Sie bitte: Commodore sendet Ihnen die Kasette in kei-
nen Fall zurück. Wir versichern Ihnen jedoch, keinen Gebrauch
von Ihren Unterlagen samt Software zu machen, falls eine Ober-
einkunft nicht zustande kommt.

Sie können beliebig viele Programme auf einer Kasette einsenden.

Fügen Sie bitte folgende Informationen bei:

1. Name des Programmierers;
2. kurze Abschreibung (max. 20 Zeilen) des Programms; hier sol-
len kurz, aber vollständig die Fragen beantwortet werden:
"Was macht dieses Programm?" und
"Was kann man damit tun?";
3. eine Kasette mit Ihrem Programm;
4. eine kurze "Bedienungsanleitung" des Programms. Diese Be-
dienungsanleitung sollte aber möglichst im Programm selbst
stehen;
5. Angabe etwaiger Besonderheiten in der Entwicklung oder An-
wendung dieses speziellen Programms.

Richten Sie bitte Ihre Angebote an:

Commodore Büromaschinen GmbH
PET-BENUTZERCLUB
Frankfurter Str. 171-175
6078 Neu-Isenburg



? Wie kann ich den Befehl PRINT AT X realisieren?

A.: Der Befehl PRINT AT X ist in manchen BASIC-Versionen enthalten. Er bewirkt den Beginn des Ausdrucks an Stelle X.

Der Bildschirm des PET ist unterteilt in 25 Zeilen (Zeile 0 bis 24) und 40 Spalten (Spalte 1 bis 39). Das sind 1000 Plätze, an denen beliebige Zeichen stehen können. Diese 1000 Zeichen sind im Bildschirmspeicher von (dezimal) Adresse 32768 bis 33767 gespeichert.

Im Speicher des PET befinden sich folgende Informationen über die Lage des Cursors:

Adresse 245 enthält die Zeile (0 bis 24).

Adresse 226 enthält die Spalte (0 bis 39).

Adressen 224 und 225 enthalten die der Stellung des Cursors entsprechende Adresse des Bildschirmspeichers (niederwertiges Byte in 224, höherwertiges Byte in 225).

Das untenstehende Beispiel-Programm druckt ein * in Zeile Z, Spalte S des Bildschirms. Der Übersichtlichkeit halber sind die Zeilen von 1 bis 25 und die Spalten von 1 bis 40 numeriert.

```

5 PRINT"@"
10 INPUT" ZEILE";Z
20 INPUT"SPALTE";S
25 IF Z<1 OR Z>25 OR S<1 OR S>40 THEN5
30 GOSUB1000
40 PRINT"@"
50 END
1000 M=32727+40*Z+S
1010 I=INT(M/256)
1020 J=M-I*256
1030 POKE224,J:POKE225,I
1040 RETURN

```



?: Wie kann ich die Rechengeschwindigkeit des PET erhöhen?

A.: 1. Entfernen Sie alle unnötigen Leerstellen und REM-Befehle aus dem Programm. Dies führt zu einer geringen Erhöhung der Rechengeschwindigkeit. Der BASIC-Interpreter spart die Zeit ein, die er sonst zum Überprüfen und Überspringen dieser Zeichen benötigen würde.

2. Verwenden Sie Variablen anstatt Konstanten.

Die Umwandlung einer Konstanten in die Fließkommadarstellung benötigt wesentlich mehr Zeit, als der Zugriff auf den Wert einer einfachen oder Matrix-Variablen.

Dies ist besonders wichtig bei FOR-NEXT-Schleifen oder ähnlichen Befehlsfolgen, die wiederholt ausgeführt werden.

Im folgenden Beispiel erreicht man so eine Erhöhung der Rechengeschwindigkeit fast um den Faktor 10. (Beachten Sie Zeile 40).

<pre> 10 X=PI 30 FORG=1TO1000 40 A=A+3.14 50 NEXT 60 PRINT(XI-X)/50 </pre>	<pre> 10 X=PI 20 PI=3.14 30 FORG=1TO1000 40 A=A+PI 50 NEXT 60 PRINT(XI-X)/50 </pre>
----------------------------------------------------------------------------	-------------------------------------------------------------------------------------

3. Variablen, die während der Programmausführung zuerst angetroffen werden, erhalten einen Speicherplatz am Beginn der Variablen-tabelle zugewiesen.

Die Programmzeile:

```
10 A=0:B=0:C=0
```

bewirkt, daß A als erste, B als zweite und C als letzte Variable plaziert wird. (Falls Zeile 10 die erste Programmzeile ist.)

Wenn BASIC im Lauf der weiteren Programmausführung einen Befehl findet, welcher auf die Variable Bezug nimmt, so wird es einen Suchvorgang benötigen, um A aufzufinden, zwei Suchvorgänge für B und drei für das Auffinden von C.



4. NEXT Befehle ohne die Indexvariable.

NEXT ist etwas schneller als NEXTR, weil die Überprüfung auf Übereinstimmung mit der zugehörigen FOR-Variablen entfällt. Man sollte den so erreichbaren Zeitgewinn abwägen gegenüber dem Verlust an Klarheit und Übersichtlichkeit des Programms.

Beispiel:

```
10 T=TI
20 FORR=1T010000:NEXT
30 PRINT(TI-T)/60
```

```
10 T=TI
20 FORR=1T010000:NEXTR
30 PRINT(TI-T)/60
```

5. Ist der noch freie RAM-Speicher kleiner als etwa 300 Bytes, dann erhöht sich die Rechenzeit dramatisch. (Es wird dann für das Betriebssystem immer schwieriger, etwa einen veränderten String neu zu plazieren.)

?: Wie kann ich Speicherplatz einsparen?

A.: 1. Verwenden Sie mehrere Befehle in einer Zeile. Jede Zeilennummer bewirkt die Belegung von 5 Bytes (außer den für die Befehle benötigten Bytes), siehe FW 04.

2. Entfernen Sie alle unnötigen Leerstellen aus dem Programm. Die Zeile:

```
10 PRINT X, Y, Z
```

benötigt 3 Bytes mehr als:

```
10 PRINTX,Y,Z
```

(Alle Leerstellen zwischen Zeilennummer und erstem Zeichen werden vom Betriebssystem ignoriert.)

3. Entfernen Sie alle unnötigen REM-Befehle aus dem Programm. Jedes REM benötigt mindestens 1 Byte plus Bytes für den Text. Die Zeile:

```
130 REM Bemerkung
```

benötigt (5 + 1 + 10) Bytes.

In der Zeile:

```
140 X = X + Y : REM SUMME
```



verbraucht der REM-Befehl 8 Bytes, einschließlich des trennenden Doppelpunktes.

4. Verwenden Sie Variablen als Konstanten. Wenn Sie beispielsweise die Konstante 1000 in einem Programm oft benötigen, dann fügen Sie den Befehl ein:

10 T=1000

anstatt jedesmal die Konstante auszuschreiben.

5. Der END-Befehl kann bei Commodore-BASIC entfallen.
6. Verwenden Sie Subroutinen für öfter vorkommende Befehlsfolgen (auch für Text!).
7. Verwenden Sie gleiche Variablen für Zwischenergebnisse, Schleifen, usw. Achten Sie aber darauf, nicht etwa die Variable X innerhalb einer FORX= ... TO-Schleife zu verwenden.
8. Benutzen Sie von Feldern auch das Element mit dem Index Null, also A(0), B(0), usw.
- 9.*Jede aktive FOR-NEXT-Schleife belegt 22 Bytes.

*Jede aktive Subroutine (RETURN ist noch nicht ausgeführt) belegt 6 Bytes.

*Jede offene Klammer benötigt 4 Bytes und jedes berechnete Zwischenergebnis 12 Bytes.
10. Verwenden Sie Ganzzahlvariablen, wo möglich. (Solche Variablen liegen zwischen -33767 und +33767 und haben das Zeichen % im Namen. Beispiel: X%, M%(7,2), A8%).



Bei den PET-Computern der ersten Serie sind gewisse Besonderheiten in der Programmierung zu beachten.

In den Festwertspeichern (ROM's) des PET-Computers ist ein Programm gespeichert, das folgende Aufgaben erfüllt:

1. Übersetzen von BASIC in die Maschinensprache (Interpreter)
2. Steuerung und Koordinierung von Bildschirm, Recorder, Peripherie usw. (Betriebssystem)
3. Selbsttest (Testroutinen)

Dieses Programm hat immerhin einen Speicherumfang von 13 kB. Es ist verständlich, daß bei einem Programm dieser Länge manche Fehler (engl. bugs = Käfer) erst nach längerem Betrieb auftreten. In solchen Fällen wird Sie der Benutzer-Club informieren.

In absehbarer Zeit werden wir ein ROM anbieten, das alle nötigen Verbesserungen aufweist.

1. IF F OR I = 10 THEN 250 wird zusammengezogen zu:
IF FOR I = 10 THEN 250 und führt auf einen ?SYNTAX ERROR

Das einzige BASIC-Wort, welches mit Zwischenraum geschrieben werden darf, ist GOTO; es ist auch in der Form GO TO zulässig.

Der Befehl IF F OR I = 10 THEN 250 ist in Zukunft zulässig (Nach dem Einsetzen des neuen ROM's).

2. Die BYTES FREE - Nachricht nach dem Einschalten des PET-Computers ist verschieden von dem Ergebnis, das man durch Eingabe von ?FRE(0) unmittelbar nach dem Einschalten erhält.

Dies ist kein Fehler, sondern kommt dadurch zustande, daß der Befehl PRINT FRE(0) 3 bytes Speicherplatz benötigt.



3. Der Befehl SAVE sollte vom Betriebssystem mit "PRESS REC & PLAY" anstatt "PRESS PLAY & REC" beantwortet werden, da die Taste REC zuerst gedrückt werden muß.

Dies ist tragbar und wird wahrscheinlich nicht geändert.

4. Die Funktion POS wird aus Commodore-BASIC entfernt.
5. SPACE und SPACE bei gedrückter SHIFT-Taste ergeben verschiedene ASCII-Werte.

Dies ist kein Fehler; die obenliegenden Zeichen sind generell anders indiziert als die untenliegenden.

6. Bei Output (=Ausgabe) eines Anführungszeichens (Code 34 oder 98) werden etwaige Cursor-Bewegungen wörtlich aufgefaßt.

B E I S P I E L : 10?CHR\$(34),1,2,3

(Die Cursor-Bewegungen zu den vor-
tabulierten Positionen werden also
geschrieben, anstatt ausgeführt.)

Wird vorläufig nicht geändert.

7. SPC(0) druckt 256 Leerstellen.

Wird geändert.

8. Bei direkter Eingabe werden Zeilen mit Doppelpunkt am Anfang ignoriert.

B E I S P I E L : :?A

Wird geändert.

9. Bei Matrizen (Feldern) sind nur 255 Elemente pro Feld zulässig.

B E I S P I E L : DIM B(25,9) hat zuviele Elemente

Wird geändert.

10. CHR\$ akzeptiert auch Strings als Argument.

Wird geändert.



11. Die Befehle: OPEN n,1,m
OPEN n,2,m

(n bedeutet die Nummer des Files - n ist eine ganze Zahl von 1 bis 255. m = 0 bewirkt Einlesen
m = 1 bewirkt Schreiben und
m = 2 bewirkt Schreiben mit EOT-Zeichen.)

Es hat sich herausgestellt, daß unter bestimmten Bedingungen das BOF-Zeichen (Beginning Of File = Fileanfang) trotz des OPEN-Befehls nicht ordnungsgemäß geschrieben bzw. gelesen wird.

Das korrekte Eröffnen eines Files kann erzwungen werden durch die Befehle:

POKE 243,122:POKE 244,2

vor einem OPEN-Befehl für Recorder #1

Für den Recorder #2 lauten die entsprechenden Befehle:

POKE 243,58:POKE 244,3

vor einem OPEN-Befehl für Recorder #2

Die Anfangsadresse des jeweils aktiven Kassettenpuffers ist in 243 (niederwertiges Byte) und 244 (höherwertiges Byte) abgespeichert. Die Startadresse des Pufferspeichers für Recorder #1 ist \$27A (\$2 = 2 und \$7A = 122). Die Startadresse des Puffers für Recorder #2 ist \$33A (\$3 = 3 und \$3A = 58). Durch die obigen POKE-Befehle wird also eine korrekte Initialisierung sichergestellt.

Diese Befehle können auch im Programm bleiben, nachdem die in Zukunft erhältlichen ROM's mit dem neuen Betriebssystem eingesetzt wurden.



*SPEICHERUNG VON PROGRAMMEN (8k-Version des PET 2001)

Eine Programmzeile besteht aus Zeilennummer und Befehlen. Alle Programmzeilen werden in Reihenfolge der Zeilennummer von 1024 an aufwärts folgendermaßen gespeichert (n ist die Zeilennummer, (n+x) ist die auf n folgende Zeilennummer):

Das erste Byte ist Null. (A_n)

Es folgen 2 Bytes (B_n, C_n). Sie stellen die Verbindung zur nächsten Programmzeile her. Der Wert $B_n + 256 * C_n$ gibt die Adresse an, in der die Größe B_{n+x} steht. Sind B_n und C_n beide Null, dann ist n die letzte Zeilennummer des Programms.

Die nächsten zwei Bytes (D_n, E_n) geben die Zeilennummer an:
 Zeilennummer = $D_n + 256 * E_n$

Es folgen die BASIC-Befehle (codiert in 1 Byte pro Befehl bzw. 1 Byte pro Zeichen).

Beispiel: Das Programm

```
1000 PRINTX
2000 GOTO 1000
```

wird folgendermaßen gespeichert:

0	8	4	232	3	153	88	Inhalt
1024	1025	1026	1027	1028	1029	1030	Speicherstelle
A	B	C	D	E	PRINT	X	Übersetzung

0	18	4	208	7	137	49	48	48	48
1031	1032	1033	1034	1035	1036	1037	1038	1039	1040
A	B	C	D	E	GOTO	1	0	0	0

0	0	0
1041	1042	1043
A	B	C

PET-2001 INTERPRETERCODE



000 ZL.ANFG	064 @	128 END	192 TAN
001 N.V.	065 A	129 FOR	193 ATN
002 N.V.	066 B	130 NEXT	194 PEEK
003 N.V.	067 C	131 DATA	195 LEN
004 N.V.	068 D	132 INPUT#	196 STR#
005 N.V.	069 E	133 INPUT	197 VAL
006 N.V.	070 F	134 DIM	198 ASC
007 N.V.	071 G	135 READ	199 CHR#
008 N.V.	072 H	136 LET	200 LEFT#
009 N.V.	073 I	137 GOTO	201 RIGHT#
010 N.V.	074 J	138 RUN	202 MID#
011 N.V.	075 K	139 IF	203 N.V.
012 N.V.	076 L	140 RESTORE	204 N.V.
013 N.V.	077 M	141 GOSUB	205 N.V.
014 N.V.	078 N	142 RETURN	206 N.V.
015 N.V.	079 O	143 REM	207 N.V.
016 N.V.	080 P	144 STOP	208 N.V.
017 N.V.	081 Q	145 ON	209 N.V.
018 N.V.	082 R	146 WAIT	210 N.V.
019 N.V.	083 S	147 LOAD	211 N.V.
020 N.V.	084 T	148 SAVE	212 N.V.
021 N.V.	085 U	149 VERIFY	213 N.V.
022 N.V.	086 V	150 DEF	214 N.V.
023 N.V.	087 W	151 POKE	215 N.V.
024 N.V.	088 X	152 PRINT#	216 N.V.
025 N.V.	089 Y	153 PRINT	217 N.V.
026 N.V.	090 Z	154 CONT	218 N.V.
027 N.V.	091 [155 LIST	219 N.V.
028 N.V.	092 \	156 CLR	220 N.V.
029 N.V.	093]	157 CMD	221 N.V.
030 N.V.	094 ↑	158 SYS	222 N.V.
031 N.V.	095 +	159 OPEN	223 N.V.
032 SPC	096	160 CLOSE	224 N.V.
033 !	097 !	161 GET	225 N.V.
034 "	098 "	162 NEW	226 N.V.
035 #	099 #	163 TAB(227 N.V.
036 \$	100 \$	164 TO	228 N.V.
037 %	101 %	165 FN	229 N.V.
038 &	102 &	166 SPC(230 N.V.
039 '	103 '	167 THEN	231 N.V.
040 (104 (168 NOT	232 N.V.
041)	105)	169 STEP	233 N.V.
042 *	106 *	170 +	234 N.V.
043 +	107 +	171 -	235 N.V.
044 ,	108 ,	172 *	236 N.V.
045 -	109 -	173 /	237 N.V.
046 .	110 .	174 ↑	238 N.V.
047 /	111 /	175 AND	239 N.V.
048 0	112 0	176 OR	240 N.V.
049 1	113 1	177 >	241 N.V.
050 2	114 2	178 =	242 N.V.
051 3	115 3	179 <	243 N.V.
052 4	116 4	180 SGN	244 N.V.
053 5	117 5	181 INT	245 N.V.
054 6	118 6	182 ABS	246 N.V.
055 7	119 7	183 USR	247 N.V.
056 8	120 8	184 FRE	248 N.V.
057 9	121 9	185 POS	249 N.V.
058 :	122 :	186 SQR	250 N.V.
059 ;	123 ;	187 RND	251 N.V.
060 <	124 <	188 LOG	252 N.V.
061 =	125 =	189 EXP	253 N.V.
062 >	126 >	190 COS	254 N.V.
063 ?	127 ?	191 SIN	255 *

* N.V. = Nicht verwendet

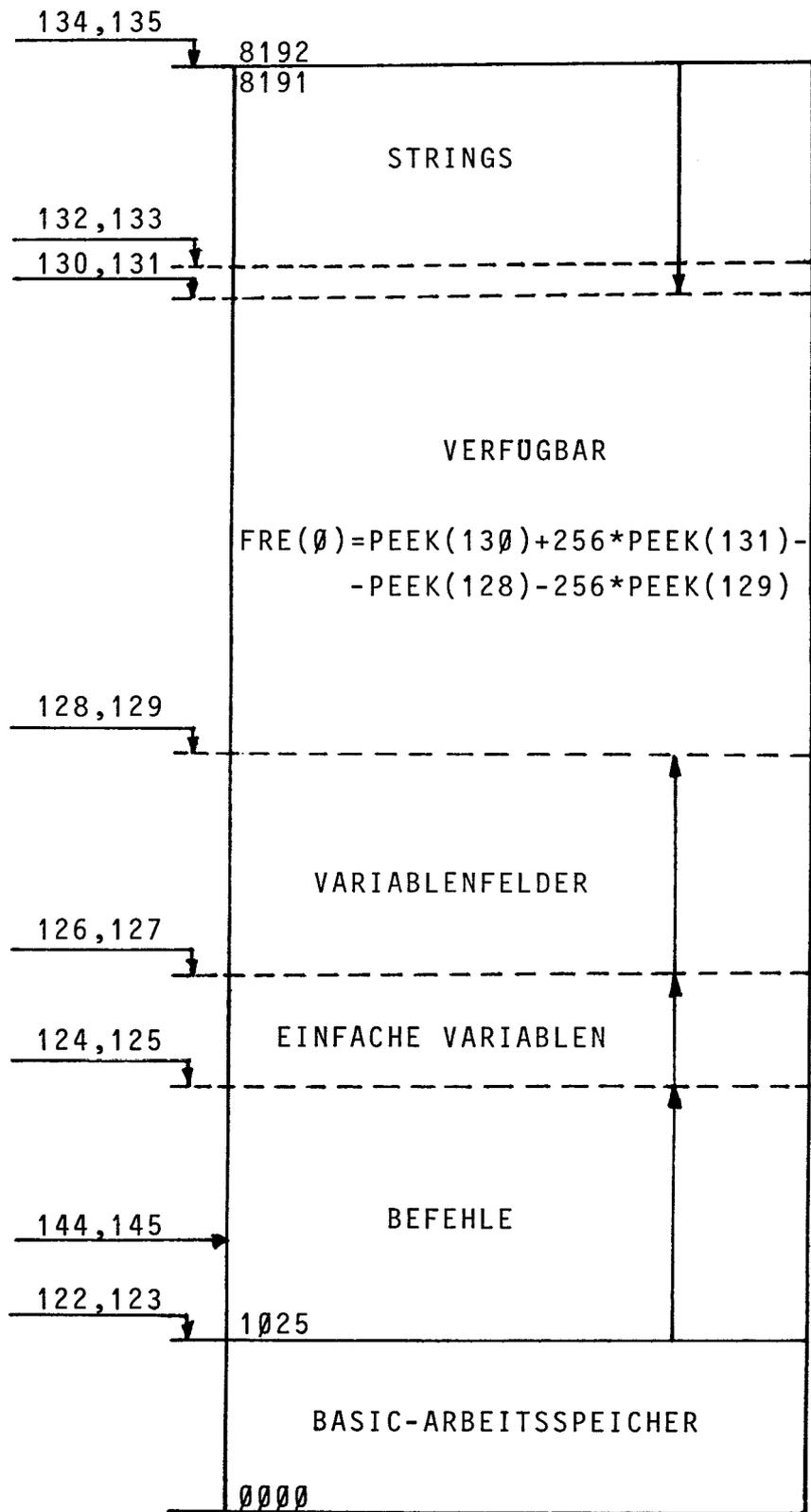


* RAM-ZUTEILUNG UND POINTER (8k-Version des PET 2001)

Obergrenze des
vorhandenen RAM's

Beginn des letzten
Strings
Ende des letzten
Strings

DATA-Befehl



* Dies ist eine vorläufige Spezifikation. Änderungen vorbehalten



65535	H7	ROM 4K
63488		
63487	H4	
61448		
61439	A2	ZEICHENGENERATOR 2K
53392		
59931	H3	ROM 1ØK
57344		
57343	H6	
55296		
55295	H2	
53248		
53247	H5	
51288		
51199	H1	
49152		
49151		15K ROM-ERWEITERUNG ≈
33792		
33791	C3,C4	BILDSCHIRM-RAM 1K
32768		
32767		24K RAM-ERWEITERUNG ≈
8192		
8191	I8 J8	8K RAM
7168		
7167	I7 J7	
6144		
6143	I6 J6	
5128		
5119	I5 J5	
4896		
4895	I4 J4	
3872		
3871	I3 J3	
2848		
2847	I2 J2	
1824		
1823	I1 J1	
ØØØØ		

ADRESSENBELEGUNG DES
PET-2001*

(Erste Spalte = Dezimale
Adresse

Zweite Spalte = Lage
des entsprechenden IC's
auf der Platine)

*Dies ist eine vorläufige Spezifikation, Änderungen vorbehalten.



*** PET 2001 SPEICHERBELEGUNG: PAGE 0 ***

(Nicht aufgefuehrte Adressen werden gebraucht, haben aber keine eindeutige Funktion)

! VORLAEUFIGE SPEZIFIKATION - AENDERUNGEN VORBEHALTEN !

0 - 000 - \$4C (JMP-Befehl)
1 2 001 002 Adresse fuer die USR-Funktion; L,H

Input-Output:

3 - 003 - # des aktiven Ein- Ausgabekanals
5 - 005 - Naechste Druckspalte fuer BASIC
10 09 00A 059 80 Bytes BASIC-Input-Puffer
98 - 05A - Hilfszaehler fuer BASIC
91 - 05B - Konstante \$00 als Trennungsfeld
92 - 05C - Hilfszaehler fuer BASIC

Variablen:

93 - 05D - Flag fuer dimensionierte Variablen
94 - 05E - Flag fuer Variablenart (0=numerisch, 1=String)
95 - 05F - Flag fuer Integers
96 - 060 - Flag fuer ' ' und REM
97 - 061 - Flag fuer Indizierung
98 - 062 - Flag fuer INPUT (=0), GET/GET# (=64) oder READ (=152)
99 - 063 - Flag fuer das Vorzeichen von TAN
100 - 064 - Flag fuer Output (negativ, wenn Ausgabe unterdrueckt)
101 - 065 - zeigt auf naechsten Descriptor
102 103 066 067 Zeiger auf den zuletzt verwendeten String; L,H
104 111 068 06F Tabelle der Doppelbyte-Descriptoren fuer Variablen
112 113 070 071 Indirekter Index #1; L,H
114 115 072 073 Indirekter Index #2; L,H
116 121 074 079 Pseudoregister fuer Operanden von Funktionen

Datenabspeicherung:

122 123 07A 07B Zeiger auf den Beginn der BASIC-Befehle; L,H
124 125 07C 07D Zeiger auf den Beginn der Variablen-tabelle; L,H
126 127 07E 07F Zeiger auf den Beginn der Array-Tabelle; L,H
128 129 080 081 Zeiger auf das Ende der Variablen-tabelle; L,H
130 131 082 083 Zeiger auf Ende des letzten Strings; L,H
132 133 084 085 Zeiger auf Beginn des letzten Strings; L,H
134 135 086 087 Zeiger auf hoechste RAM-Adresse; L,H
136 - 088 - ist 255, wenn ein direkter Befehl ausgefuehrt wird
136 137 088 089 # der gerade ausgefuehrten Zeile; L,H
138 139 08A 08B Zeilen# fuer CONT; L,H
140 141 08C 08D Zeiger auf den naechsten Befehl; L,H
142 143 08E 08F # der DATA-Zeile bei ERROR; L,H
144 145 090 091 Zeiger auf naechstes DATA; L,H



Terme:

146	147	092	093	Herkunft des Input
148	149	094	095	Name der lfd. Variablen (1. und 2. Zeichen)
150	151	096	097	Zeiger auf die lfd. Variable; L,H
152	153	098	099	Zeiger auf die lfd. FOR-NEXT-Variable; L,H
154	155	09A	09B	Zeiger auf den lfd. Operator; L,H
156	-	09C	-	Maske fuer lfd. Operator (> = ()
157	158	09D	09E	Zeiger fuer DEFFN; L,H
159	160	09F	0A0	Zeiger auf den String-Descriptor; L,H
161	-	0A1	-	Laenge des obigen Strings
162	-	0A2	-	Konstante
163	-	0A3	-	42C (JMP-Befehl in Maschinensprache)
164	165	0A4	0A5	Vektor fuer Funktionen; L,H
166	171	0A6	0AB	Fliesskommaakkumulator #3
172	173	0AC	0AD	Zeiger #1 fuer Blocktransfer; L,H
174	175	0AE	0AF	Zeiger #2 fuer Blocktransfer; L,H
176	181	0B0	0B5	Fliesskommaakkumulator #1 (u.a. fuer USR)
182	-	0B6	-	Vorzeichen der Mantisse aus Akku #1
183	-	0B7	-	Zaehlt die zum Normalisieren von Akku #1 noetigen Shifts
184	189	0B8	0BD	Fliesskommaakkumulator #2
190	-	0BE	-	Overflowbyte des Arguments aus Akku #2
191	-	0BF	-	Vorzeichen der Mantisse aus Akku #2
192	193	0C0	0C1	Zeiger auf ASCII-Darstellung des Akku #2; L,H

RAM-Subroutinen:

194	199	0C2	0C7	CHRGOT-Routine; holt naechstes Zeichen des BASIC-Texts
200	-	0C8	-	CHRGOT; enthaelt lfd. Zeichen des BASIC-Texts
201	202	0C9	0CA	Zeiger in den BASIC-Text; L,H
203	223	0CB	0DF	enthaelt naechste Zufallszahl fuer RND

Betriebssystem:

224	225	0E0	0E1	Zeiger auf Cursorstellung im Bildschirm-RAM; L,H
226	-	0E2	-	Cursorposition (Spalte)
227	228	0E3	0E4	Indirekter Zeiger fuer diverse Zwecke; L,H
234	-	0EA	-	Flag fuer "
238	-	0EE	-	Laenge des lfd. Filenamens
239	-	0EF	-	# des lfd. Files
240	-	0F0	-	lfd. Sekundaeradresse
241	-	0F1	-	lfd. Primaeradresse
242	-	0F2	-	Flag fuer einfache (=39) oder doppelte (=79) Zeile
243	244	0F3	0F4	Zeiger auf den lfd. Tape-Puffer; L,H
245	-	0F5	-	Cursorposition (Zeile)
246	-	0F6	-	Zwischenspeicher fuer Input-Output
247	248	0F7	0F8	Zeiger fuer das Operating-System (LOAD, VERIFY); L,H
249	250	0F9	0FA	Zeiger auf den lfd. Filenamens; L,H
255	-	0FF	-	Overflowbyte fuer ASCII-Darstellung des Akku-Inhalts

*** PET 2001 SPEICHERBELEGUNG: PAGE 1 ***

256-317 (dezimal) werden zur Fehlerkorrektur beim Lesen vom Band benutzt, sowie als Puffer fuer Zahlenumwandlungen. Der Rest von Page 1 dient als Speicher im Zusammenhang mit GOSUB und FOR-NEXT, sowie als Hardware-Stack.



* PET 2001 SPEICHERBELEGUNG: PAGE 2,3*

(Nicht aufgefuehrte Adressen werden gebraucht, haben aber keine eindeutige Funktion)

! VORLAEUFIGE SPEZIFIKATION - AENDERUNGEN VORBEHALTEN !

512	514	200	202	24-Stunden-Uhr in 1/60 Sekunden
515	-	203	-	Gedruckte Taste
516	-	204	-	Flas fuer SHIFT
517	518	205	206	Korrekturfaktor fuer die Uhr, L,H
519	520	207	208	Interrupt-Flas fuer Recorder-(#1, #2)-Schalter
521	-	209	-	Flas fuer RVS, SPACE, STOP, . , [, = .
523	-	20B	-	Flas fuer VERIFY und LOAD
524	-	20C	-	Statusbyte fuer I/O
525	-	20D	-	Anzahl Bytes im Tastaturpuffer
526	-	20E	-	Flas fuer RVS
527	536	20F	218	Tastaturpuffer
537	538	219	21A	IRQ-Vektor, L,H (zeigt auf: Tastendruck speichern')
539	540	21B	21C	BRK-Vektor, L,H
547	-	223	-	Tastatur-Input-Code, solange Taste gedrueckt
549	-	225	-	Zaehler fuer die Blinkdauer des Cursors
551	-	227	-	Flas fuer Cursor ein/aus
553	577	229	241	Tabelle der LSB's der Zeilenanfaenge (Bildschirm)
578	587	242	24B	Tabelle der Filenummern
588	597	24C	255	Tabelle der Geraetenummern
598	607	256	25F	Tabelle der Sekundaeradressen
608	-	260	-	Flas fuer Input (Bildschirm oder Tastatur)
610	-	262	-	Anzahl der offenen Files
611	-	263	-	# des Input-Geraetes (0 = Tastatur)
612	-	264	-	# des Output-Geraetes (3 = Bildschirm)
613	-	265	-	Paritaetsberechnung beim Schreiben auf Cassette
616	-	268	-	Zaehler fuer Recorder-Puffer
621	-	26D	-	Zaehler fuer Blocks auf Cassette
624	-	270	-	Synchronisation fuer Schreiben auf Cassette
625	626	271	272	Zeiser auf naechstes Zeichen Recorder-Puffer rein/raus
627	-	273	-	Synchronisation fuer Lesen von Cassette
628	-	274	-	Flas fuer Bit/Byte-Kassette-Fehler
630	631	276	277	Zeiser fuer Korrektur beim Lesen von Cassette
632	-	278	-	Flas fuer Lesen von Cassette
633	-	279	-	Dauer der 'shorts' vor dem Schreiben von Daten
634	825	27A	339	Puffer fuer Recorder #1
826	1017	33A	3F9	Puffer fuer Recorder #2
1017	1023	3FA	3FF	nicht benutzt



? Wann sind Speichererweiterung und Floppy-Disc erhältlich;

A: Nicht vor Jahresende; Daten und Preise werden in einer der nächsten Benutzer-Info's bekanntgegeben.

? Wann ist der Drucker lieferbar?

A: Der Termin steht noch nicht fest. Ein Kurzprospekt liegt bei.

? Wann sind ROM's mit geändertem Betriebssystem lieferbar;

A: Das ROM 6540-019 verhindert ein unkontrolliertes Wegbleiben des Cursors beim Schreiben und Korrigieren von Programmen. Es wird etwa DM 30,-- kosten und Anfang Oktober erhältlich sein.

Das ROM mit den Korrekturen bezüglich Syntax, Matrizen etc. (siehe Seiten FW 01 bis FW 03) ist noch nicht lieferbar. Sobald dies der Fall ist, erhalten Sie Mitteilung.



cbm 3022 / cbm 3023

KURZINFO

Die neuen Drucker für Ihren Commodore Microcomputer

Commodore Business Machines
3023 PRINTER

ZEICHENVORRAT ENTHAELT REVERSE FIELD, SPERRSCHRIFT
UND PROGRAMMIERBARE SONDERZEICHEN WIE Z.B.:

Ⓞ Ⓞ
 !"#%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz+*!%&
 !"#%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz+*!%&
 !"#%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz+*!%&
 !"#%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz+*!%&

DATENFORMATIERUNG UND FEHLERAUSDRUCK SIND MOEGLICH:

ZZZZZ	AAAAAAAAAAAA	####\$.99	999	999	99-	99	9999
00002	CBM-COMPUTER	\$999.99	50	- 1	1-	1	****

AUCH MIT TRAKTORFUEHRUNG (PRINTER 3022)

***** 3023 PRINTER *****

Commodore Business Machines

- Matrixdrucker (7 x 8 bzw. 7 x 16)
- Zeichenvorrat: 160
- 80 Zeichen pro Zeile (40 bei Sperrschrift)
- Voll formatierbare Ausgabe (numerisch und alphanumerisch)
- Vorgesehen als # 4 am IEEE-Bus (modifizierbar zu # 5 bis # 11)
- Normalpapier 8,5" breit. Rolle mit 3,25" ϕ . Bei offenem Papierfach bis zu 4,5" ϕ
- Papiervorschub über Rollen (3023) oder Traktorführung (3022)
- Großschreibung, Kleinschreibung **und** alle Zeichen in einer Zeile darstellbar
- Sperrschrift (doppelte Zeichenbreite)
- Negativdarstellung
- Abschaltbarer Fehlerausdruck
- Automatischer Rücklauf am Zeilenende
- Unterdrückung des Zeilenvorschubs (Überschreiben von Zeilen) möglich
- Paging (erzeugt automatisch oberen und unteren Rand beim Drucken jeder Seite)
- Datenausdruck wie empfangen (z. B. vom Bildschirm), oder formatiert möglich (links- oder rechtsbündige Darstellung, Vorzeichenunterdrückung, Vornullunterdrückung, etc.)
- Auch Strings formatierbar
- 1 Sonderzeichen pro Zeile frei programmierbar



KASSETTENRECORDER ENTMAGNETISIERUNG UND REINIGUNG

Der Tonkopf des Recorders wird bei jedem Schreibzyklus leicht aufmagnetisiert. Commodore empfiehlt daher häufiges

(jeweils nach etwa 3 Stunden SAVE-Betrieb)

Entmagnetisieren des Recorders.

Zusätzlich sollte ungefähr nach jeweils 100 Betriebsstunden des Recorders oder bei Auftreten von Einlesefehlern folgende Prozedur vorgenommen werden:

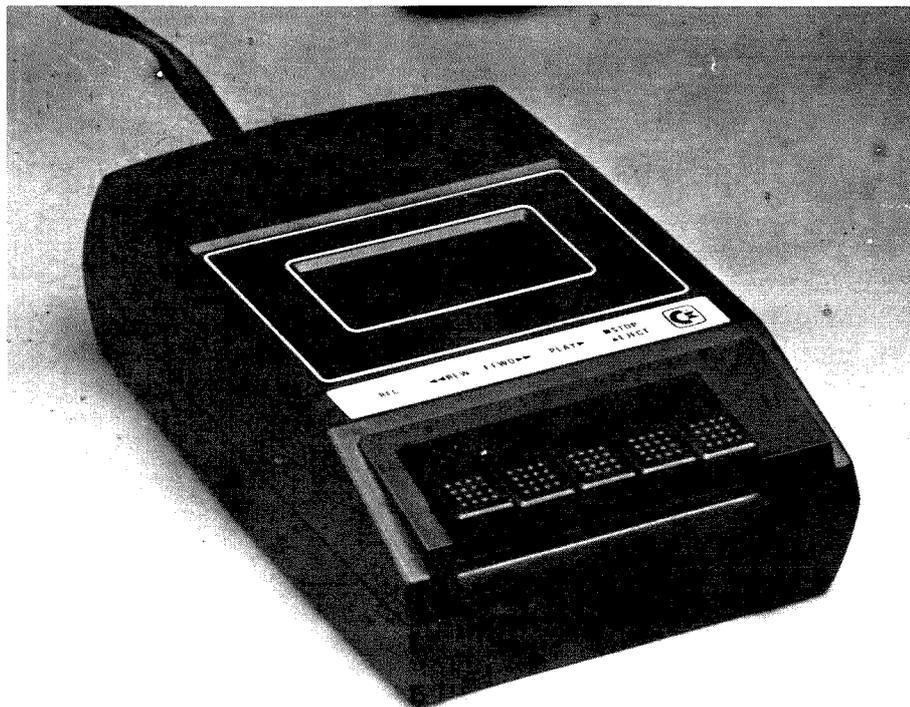
- *1. Legen Sie eine Reinigungskassette ein und lassen Sie diese abspielen.
- *2. Schalten Sie den PET-Computer aus und entfernen Sie alle bespielten Kassetten aus seiner Nähe, damit diese nicht beim Einschalten der Löschdrossel gelöscht werden.
- *3. Öffnen Sie das Kassettenfach des Recorders und drücken Sie die Wiedergabetaste PLAY.
- *4. Schalten Sie die Löschdrossel ein. Die Entfernung zum Tonkopf sollte dabei mindestens einen halben Meter betragen.
- *5. Bewegen Sie die Entmagnetisierungsdrossel langsam (ca. 3 cm/sec oder langsamer) auf den Tonkopf zu und führen Sie diese langsam über alle metallischen Teile, die mit dem Tonband in Berührung kommen, einschließlich Löschkopf. Verwenden Sie keine Drossel mit blanker Oberfläche, um den Tonkopf nicht zu zerkratzen.
- *6. Entfernen Sie die Drossel langsam vom Recorder. In etwa einem Meter Entfernung schalten Sie aus.
- *7. Überprüfen Sie den Aufnahme-Wiedergabekopf auf Verschleiß. Ist er mehr als einige Bänddicken abgeschliffen und treten immer noch Einlesefehler auf, so muß der Tonkopf ersetzt werden. Üblicherweise ist dies erst nach mehr als 3000 Betriebsstunden der Fall.



KURZINFO

DATASSETTE:

Der externe Recorder für Ihren PET 2001



- Elektrisch und mechanisch dem eingebauten Recorder gleichwertig
- Benötigt keine externe Stromversorgung (Anschluß am PET 2001 serienmäßig vorhanden)
- Ermöglicht beispielsweise das Kopieren von Datenfiles unter Programmkontrolle

Die externe Kassette wird als Gerät # 2 am IEEE-Bus angesprochen.

Die entsprechenden Befehle sind:

SAVE " ", 2

VERIFY " ", 2

LOAD " ", 2

bzw.: SAVE "NAME", 2

VERIFY "NAME", 2

LOAD "NAME", 2

Mit dem Befehl OPEN X,2,Y bzw. OPEN X,2,Y, "NAME" wird das logische File X der Datensette (= Recorder # 2) zugeordnet. Die zugehörigen Ein- und Ausgabebefehle lauten dann:

INPUT # 2, ...

GET # 2, ...

PRINT # 2, ...



Folgende Programme (Nr. 1 bis Nr. 5) können wir in der englischen Fassung anbieten:

Nr. 1: Target Pong and Off-The-Wall

Zwei Spiele für bis zu 9 Spieler mit 9 möglichen Schwierigkeitsgraden.

Mit zwei Tasten können Sie den Ball ablenken und zugleich Hindernisse aufbauen. Bei Target Pong versuchen Sie, ein bewegliches Ziel zu treffen. Bei Off-The-Wall ist die Aufgabe, den Ball so zu bewegen, daß er weder auf die Spielfeldbegrenzung trifft, noch auf einen der auf dem Feld verteilten Kakteen.

Spitzenleistung in Programmierertechnik.

DM 24,95 incl. MWSt.
=====

Nr. 2: Mortgage

Ein Finanzprogramm. Sie können folgende Daten eingeben: Grundschuld, jährlicher Zinssatz, Beginn der Rückzahlung, Zahlungsperiode, Zeitpunkt der Anleihe, Datum der nächsten Zahlung.

PET berechnet: Den Betrag jeder Rate (unterteilt in Schuld- u. Zinsanteil) noch offene und bereits bezahlte Schulden und Zinsen sowie die Anzahl der Raten.

Sehr übersichtliche und klare Darstellung auf dem Bildschirm des PET.

DM 37,95 incl. MWSt.
=====



Nr. 3: Galaxy Games

Wieder mal unterwegs im Weltraum und wohlversehen mit Torpedos versuchen Sie, einen amoklaufenden Roboter oder mehrere feste Ziele abzuschießen. (Sie können unter zwei Möglichkeiten wählen, Ihr Raumschiff zu steuern, damit Sie nicht etwa in einem der Sterne verglühen.) Für bis zu 5 Spieler. Schwierigkeitsgrad wählbar von 1 bis 9

DM 24,95 incl. MWSt.
=====

Nr. 4: Space Talk, Space Fight

Space Talk enthält die Bedienungsanleitung für das Raumschiff und das Universum.

In der anschließenden Verfolgungsjagd (Space Fight) versuchen Sie, den Schüssen des Mitspielers und anderen Hindernissen auszuweichen und ihm zuvorzukommen.

Für 2 Spieler.

DM 24,95 incl. MWSt.
=====

Nr. 5: Diet Planner and Biorythm

Diet Planner erstellt einen auf Ihre persönlichen Bedürfnisse zugeschnittenen Kalorienplan und, falls Sie abnehmen wollen, eine Liste der zu erwartenden Fortschritte. Die Ein- und Ausgaben erfolgen wahlweise in englischen oder metrischen Maßen.

Biorythm zeichnet nach Eingabe Ihres Geburtsdatums ein Diagramm der emotinalen, intellektuellen und physischen Zyklen für beliebige 30 Tage.

DM 37,95 incl. MWSt.
=====



Die folgenden Programme sind nur für Mitglieder des PET-Benutzerclubs erhältlich. Die Programme laufen (falls nichts anderes angegeben) auf der 8k-Version des PET und kosten je DM 16,- incl. MWSt. Lieferumfang: Kassette mit Programm.

Die Software ist von uns auf grobe Fehler überprüft, Commodore kann jedoch keinerlei Gewähr übernehmen. Bitte bestellen Sie nur mit beiliegender Bestellkarte.

PC1 : LIFE (Dr. Randow). Spiel.

Eine Gruppe von Individuen vermehrt und vermindert sich nach bestimmten mathematischen Regeln. (Graphisch dargestellt).
1945 Bytes.

PC2 : LOGIK (Dr. Randow). Mathematik.

Ein Programm zur Berechnung von Wahrheitswerten in der Aussagenlogik. Mit dreiseitiger Anleitung.
1156 Bytes.

PC3 : PET-UHR 3 und PET-UHR 4 (K. Knäpper und Commodore).

Das Programm enthält zwei verschiedene Versionen, den PET 2001 als Digitaluhr zu verwenden.
2959 Bytes.

PC4 : PRIMZ und PRIMF (Commodore). Mathematik.

Berechnet Primzahlen ab einer vorgegebenen Anfangszahl oder zerlegt eine ganze Zahl in die Primfaktoren.
1298 Bytes.

PC5 : CODEBRECHER (Rolf Krämer). Spiel.

Der Rechner erzeugt eine vierstellige Zahl, die erraten werden soll, Dazu gibt PET Hinweise auf Anzahl und Position der richtigen Ziffern.
3063 Bytes.

PC6 : DREI SPIELE (H.M. Kath).

Das "U-BOOT" soll in einem 10 x 10-Feld aufgespürt und vernichtet werden.
Der "TURM VON HANOI" soll scheibchenweise umgestellt werden, ohne eine größere Scheibe auf eine kleine zu legen.
In "HARTE NUSS" soll ein Muster in das komplementäre Muster umgewandelt werden.
5725 Bytes.

PC7 : KALENDER (F. Scheuer).

Berechnet den Wochentag zu einem Datum, den Abstand zweier Daten und das Datum nach einer beliebigen Anzahl von Tagen. Kann außerdem als fortlaufender Kalender verwendet werden. Gültig zwischen 1.3.1900 und 28.2.2100.
1831 Bytes.



? Wie wird DEFFN verwendet?

A: B E I S P I E L (B1):

```

0 REM B1
10 INPUT"X";X
20 DEFFNA(X)=2↑X
30 DEFFNB(X)=3↑X
40 Y=2↑X+3↑X
50 DEFFNC(Y)=Y
60 PRINT:PRINT
70 PRINT"2 ↑";X;"=";FNA(X)
80 PRINT"3 ↑";X;"=";FNB(X)
90 PRINT"2 ↑";X;" + 3 ↑";X;"=";FNC(Y)
100 PRINT:PRINT:PRINT:GOTO10

```

Die Variablen A, B und C in Zeilen 20, 30 und 40 sind die Funktionsnamen. Für verschiedene Funktionen im gleichen Programm müssen verschiedene Namen verwendet werden; zulässig sind A, B, C..Z. Sobald das obige Programm einmal gelaufen ist, können die drei Funktionen auch über die Tastatur aufgerufen werden;

B E I S P I E L : ?FNB(7)

ergibt 2187(= 3⁷).

(Beachten Sie, daß FNC hier bei Aufruf über die Tastatur einen anderen Wert ergibt als bei Aufruf vom Programm).

? Kann man erreichen, daß der Bildschirm nicht zeilenweise, sondern auf einmal vollgeschrieben wird?

A: Verwenden Sie den Befehl POKE 59409,52 (Bildschirm dunkel), schreiben Sie den Text und geben Sie dann den Befehl POKE 59409,60 . Die Anzeige erscheint auch dann wieder, wenn der Cursor in Zeile 25 steht und ein Zeilenvorschub erfolgt.

? Wie kann ich zuverlässig Daten auf die Kasette schreiben und einlesen?

A: Im folgenden bedeutet n die Nummer des betreffenden Files. n ist eine ganze Zahl von 1 bis 255.



1. Beim Schreiben des Programms darf für PRINT#n die Abkürzung ?#n nicht verwendet werden. ?#n würde bei der Ausführung des Programms einen ?SYNTAX ERROR zur Folge haben.
2. Nach mindestens 79 Zeichen Output muß ein "carriage return" - CHR\$(13) - ausgegeben werden, falls die Daten später mit INPUT#n eingelesen werden sollen. CHR\$(13) kann auch öfter als nach jeweils 79 Zeichen geschrieben werden.

Sollen die Daten mit GET#n eingelesen werden, so ist CHR\$(13) nicht erforderlich.

3. GET#n liest über EOF (end of file) hinweg, wenn dies nicht verhindert wird. Nach GET#n sollte deshalb generell der Status abgefragt werden. (IF ST<>0 THEN ...).
4. Wurde das File mit einem Namen bezeichnet, dann muß dieser Name auch im OPEN-Befehl zum Einlesen stehen. Beachten Sie Zeile 110 im folgenden Beispiel:

```
10 OPEN 72,1,1"FILE1"  
    :  
60 CLOSE 72  
    :  
110 OPEN 72,1,0,"FILE1"  
    :  
160 CLOSE 72
```

5. PRINT#n schreibt in derselben Weise auf das Band, wie der Befehl PRINT auf den Bildschirm schreiben würde, wenn dieser nur aus einer einzigen, entsprechend langen Zeile bestünde.
6. Werden mit PRINT#n mehrere Daten in eine "Zeile" geschrieben, so müssen sie durch ein Komma ", " voneinander getrennt werden.



7. Anstelle des Befehls INPUT#n,X,Y kann man in Zweifelsfällen auch INPUT n,X\$,Y\$ verwenden. Anschließend wandelt man die String- wieder in Zahlvariablen um.

```
B E I S P I E L (B2):  0 REM B2
                     10 OPEN 1,1,1
                     20 FOR J=1 TO 2.9 STEP 0.1
                     30 PRINT#1,J;" ";J*10
                     35 PRINT#1,CHR$(13)
                     40 NEXTJ
                     50 CLOSE1
                     60 PRINT"BITTE DAS BAND ZURUECKSPULEN
                        UND DANN EINE TASTE DRUECKEN"
                     70 GETA$:IFA$= ""THEN 70
                     80 OPEN1
                     90 FORJ=1TO20
                    100 INPUT#1,X$,Y$
                    105 X=VAL(X$):Y=VAL(Y$)
                    110 PRINT X,Y
                    120 NEXTJ
                    130 CLOSE1
                    140 PRINT"FERTIG!"
```

Zeile 30: Beachten Sie die Trennung der Variablen J und J*10 durch " , " .

Zeile 35: Siehe Punkt 2.

Zeile 60: Die Daten sind jetzt auf das Band geschrieben und sollen im folgenden wieder eingelesen werden.

Zeile 100: Die Daten werden als Strings eingelesen.

Zeile 105: Die Strings werden in Zahlvariablen umgewandelt und in

Zeile 110 auf dem Bildschirm ausgegeben.



8. Der Pufferspeicher für die Kassette #1 reicht von einschließlich 634 bis 825 (dezimale Adressen) bzw. von \$027A bis \$0339 (Hexadezimaladressen).

Der Puffer für die externe Kassette #2 reicht von 826 bis 1017 bzw. von \$033A bis \$03F9.

9. Beim Schreiben von Daten auf die Kassette bewegt der Motor das Band ein kurzes Stück weiter, sobald ein Record geschrieben ist. Ein Record entspricht 191 Bytes (Bufferkapazität minus eins).

Beim Einlesen der Daten kann es geschehen, daß der Abstand zwischen den einzelnen Records zu gering ist, insbesondere bei Verwendung von leichtgängigen Kassetten. Dies führt dann zu Fehlern.

ABHILFE: Verwenden Sie beim Schreiben der Daten eine Routine, die mindestens nach jeweils 191 ausgegebenen Zeichen den Motor für kurze Zeit einschaltet, ohne daß eine Datenaufzeichnung erfolgt.

```

B E I S P I E L (B3): 0 REM B3
                      10 BU=0
                      20 OPEN 2,1,1
                      30 FORA=1 TO 999
                      40 A$=STR$(A)
                      50 LA=LEN(A$)
                      55 BU=BU+LA
                      60 IF BU<192 THEN 120
                      70 POKE 59411,53
                      80 T=TI
                      90 IF(TI-T)<20THEN90
                     100 POKE 59411,61
                     110 BU=BU-192
                     120 PRINT#2,A
                     130 NEXTA
                     140 CLOSE 2
                     150 END
  
```



```
500 J=0
510 OPEN 2
520 FOR I = 1 TO 999
530 INPUT#2,A
540 IF(ST)AND64THEN580
550 PRINTA
560 IF A<>B+1 THENPRINT"FEHLER":J=J+1
570 B=A:NEXTI
580 CLOSE2
590 PRINT:PRINTJ"FEHLER"
```

Zeile 40: Umwandlung der Zahlen (1 bis 999) in Strings.

Zeile 50: Berechnung der Anzahl Bytes, die für die Speicherung des Strings nötig sind.

Zeile 70: Einschalten des Motors von Recorder #1.

Zeile 90: Warten für 20/60 Sekunden.

Zeile 100: Ausschalten des Motors von Recorder#1.

Mit RUN500 kann anschließend die einwandfreie Aufzeichnung der Daten überprüft werden.

Bei Verwendung des externen Recorders kann mit den Befehlen POKE 59456,239 und POKE 59456,255 der Motor des Recorders#2 ein- bzw. ausgeschaltet werden.



? Wie kann ich Programme in Maschinensprache schreiben?

A.: (Für Fragen über die Maschinensprache des Mikroprozessors MOS 6502 empfehlen wir als Literatur die Programmierfibel von MOS-Technology, siehe Seite A 03).

Angenommen, Sie möchten die Inhalte zweier Speicherplätze (839 und 840) addieren und das Ergebnis in einem dritten Speicherplatz (841) ablegen.

Ein Assemblerprogramm könnte folgendermaßen aussehen:

CLC		Clear Carry-Flag
LDA	\$0347	Lade-Akku mit Inhalt von 839
ADC	\$0348	Addiere zum Akku den Inhalt von 840
STA	\$0349	Speichere das Ergebnis in 841
RTS		Return (Rückkehr aus der Subroutine)

Umgesetzt in Maschinensprache erhält man:

18		
AD	71	3
6D	72	3
8D	73	3
60		

(Hexadezimalcode; Adressen im Doppel-byte-Hexcode)

Dieses Maschinenprogramm soll ab Adresse 826 gespeichert werden. 826 bis 1017 ist der Puffer für den Recorder #2 und damit ein "sicherer" Bereich. (Sofern man den Recorder #2 nicht verwendet, wird dieser Bereich von BASIC nicht benutzt.)

Zunächst werden die Befehle nach Dezimal umgewandelt:

24		
173	71	3
109	72	3
141	73	3
96		

(Dezimalcode; Adressen im Doppel-byte-Dezimalcode)



und dann mit einem kurzen BASIC-Programm in den Speicher geschrieben:

```

10 DATA 24,173,71,3,109,72,3,141,73,3,96
20 FORM=826T0836
30 READ A
40 POKEM,A
50 NEXT

```

Zur Verbindung des obigen Maschinenprogramms mit BASIC kann man den Befehl SYS (826) verwenden.

- * SYS(A) bewirkt einen Sprung nach der Dezimaladresse A. Der
- * Inhalt von A wird als Maschinenprogramm verstanden und so-
- * lange ausgeführt, bis der Befehl RTS (§60) erkannt wird.
- * Nach RTS wird die Kontrolle an den auf SYS(A) folgenden
- * Befehl übergeben. RTS bewirkt also die Rückkehr aus dem
- * Maschinenprogramm.

Da der Befehl NEW den Pufferspeicher nicht löscht, kann man jetzt NEW eingeben und etwa folgendes Programm:

```

10 INPUT "1.ZAHL";Z1:POKE839,Z1
20 INPUT "2.ZAHL";Z2:POKE840,Z2
30 SYS(826)
40 PRINTZ1"+"Z2"="PEEK(841)

```

Ein weiteres Beispiel für ein Programm in Assembler:

	LDX #000	Lade Indexregister X mit 0
SCHLEIFE	TXA	Bringe Inhalt von X in den Akkumulator
	STA 08150,X	Speichere Inhalt des Akk. in (08150+X)
	INX	Erhöhe X um 1
	BNE SCHLEIFE	Ist das Ergebnis ungleich 0, dann
		verzweige nach SCHLEIFE, sonst
	RTS	Return (Rückkehr aus der Subroutine)



Es kann mit folgendem BASIC-Programm eingegeben werden:

```
10 DATA 162,0,138,157,80,129,232,208,249,96
20 FOR M=826T0835
30 READ A
40 POKE M,A
50 NEXT
```

und wird dann mit SYS(826) gestartet.

Für einen Vergleich der Ausführungszeiten Maschinenprogramm/
BASIC-Programm ist unten noch ein BASIC-Programm angegeben.
Es bewirkt dasselbe, wie obiges Maschinenprogramm.

```
100 A=0
110 POKE33104+A,A
120 A=A+1
130 IFA<256THEN110
```

Bem.: 32768 bis 33767 sind die Adressen des Bildschirmspeichers.



? Wo anders als im Puffer für Recorder #2 kann ich ein Maschinenprogramm speichern, ohne daß es von BASIC überschrieben wird?

A.: Die Dezimaladressen 134 und 135 im PET stellen einen Pointer dar, der auf das Ende des Schreib-Lese-Speichers +1 zeigt. 134 enthält die 8 niederwertigen und 135 die 8 höherwertigen Bits dieses Binärpointers.

In der 8 k-Version:

	Inhalt (binär)	Inhalt (Doppelbyte-Dez.)
Adresse 134	00000000	0
Adresse 135	00100000	32

Der Befehl

? PEEK(134)+256*PEEK(135)-1

gibt also die Dezimaladresse des obersten Speicherplatzes im RAM an.

Verändert man unmittelbar nach dem Einschalten oder nach einem NEW-Befehl den Inhalt von 134 und 135 beispielsweise auf 6000, so existiert für BASIC der darüberliegende Speicherplatz nicht mehr und kann für andere Zwecke verwendet werden.

Also:

```

NEW
? 6000/256
  23.4375
? 6000-23*256
  112
POKE 134,112
POKE 135,23

```

Die Obergrenze für BASIC ist damit auf 6000 festgelegt. ?FRE(0) zeigt 4972 (=5999 minus 1024 für den Arbeitsspeicher minus 3 für den Befehl ?FRE(0)).

Ab 6000 bis 8191 können Sie jetzt das Maschinenprogramm speichern.



ABKÜRZUNGEN FOR BASIC-BEFEHLE

Viele BASIC-Befehle können abgekürzt werden. Bei der Programmierung ist aber folgendes zu beachten: Es ist möglich, beispielsweise 39 PRINT-Befehle in einer Zeile unterzubringen, indem man nach der Zeilennummer 39 Mal die Abkürzung ?: eingibt. Es ist aber dann nicht mehr möglich, diese Zeile ohne weiteres zu ändern (normalerweise sind nur 80 Zeichen pro Zeile zulässig).

Untenstehende Abkürzungen sind unabhängig von der Betriebsart Groß- oder Kleinschreibung. Lediglich die Darstellung auf dem Bildschirm erscheint verschieden.

BEFEHL	ABK	BEFEHL	ABK
ABS	Ab	OPEN	Op
AND	An	PEEK	Pe
ASC	As	POKE	Po
ATN	At	PRINT	?
CHR\$	Ch	PRINT#	Pr
CLOSE	CLo	READ	Re
CLR	Cl	RESTORE	REs
CMD	Cm	RETURN	REt
CONT	Co	RIGHT\$	Ri
DATA	Da	RND	Rn
DEF	De	RUN	Ru
DIM	Di	SAVE	Sa
END	En	SGN	Sg
EXP	Ex	SIN	Si
FOR	Fo	SPC	Sp
FRE	Fr	SQR	Sq
GET	Ge	STEP	STe
GOSUB	GOs	STOP	St
GOTO	Go	STR\$	STr
INPUT#	In	SYS	Sy
LEFT\$	LEf	TAB	Ta
LET	Le	TAN	TAN
LIST	Li	THEN	Th
LOAD	Lo	USR	Us
LOG	LOG	VAL	Va
MID\$	Mi	VERIFY	Ve
NEXT	Ne	WAIT	Wa
NOT	No		



OVERLAYS

BASIC-Programme werden im PET von Adresse 1024 an aufwärts gespeichert, siehe auch FW04. Die Pointer (Zeiger in die nächste BASIC-Zeile) und Zeilennummern sind in je zwei Bytes (niederwertiges - höherwertiges) codiert. Das höherwertige Byte mal 256 wird zum niederwertigen Byte addiert, um den tatsächlichen Wert zu erhalten.

Wenn ein Programm den LOAD-Befehl enthält, so impliziert dieser weder CLR noch NEW. Das neue Programm wird eingelesen, ab 1024 gespeichert und abgearbeitet. (Bei einigen BASIC-Versionen ersetzt das neue PGM nur die entsprechenden Zeilennummern; beim PET ist dies nicht so.) Alle Befehle des Aufrufprogramms aber, die über dem Platzbedarf des geladenen Programms liegen, bleiben unverändert. Einzig der Zeiger im neuen Programm zum Rest des alten ist nicht vorhanden.

Nennen wir also P1 das aufrufende PGM und P2 das von P1 geladene PGM und sind folgende Bedingungen erfüllt:

- * Die Befehle von P1, die erhalten bleiben sollen, stehen in Zeilen mit hohen Nummern;
- * Die Befehle von P1, die erhalten bleiben sollen, werden durch P2 nicht überschrieben (d.h.: P2 ist kleiner, als der nicht mehr benötigte Teil von P1);
- * Die Befehle von P1, die erhalten bleiben sollen, haben Zeilennummern, die in P2 nicht vorkommen;

Dann kann man folgendermaßen vorgehen:

- * Man suche in P1 den letzten Befehl, der gelöscht werden soll. Dieser enthält einen Zeiger in die nächste Zeile (die erhalten bleiben soll).
- * Man speichere diesen Zeiger und
- * Nach dem Laden von P2 suche man die letzte Zeile von P2 und schreibe den obenerwähnten Zeiger in diese Zeile.



Beispiel: Folgende Programme sollen hintereinander auf Kassette gespeichert werden:

P1:

```
10 ?"HIER IST P1
20 LOAD
5000 ?"DIESE ZEILE SOLL ERHALTEN BLEIBEN"
```

P2:

```
10 ?"HIER IST P2"
20 GOT05000
```

P1 würde zwar ausdrucken:

```
HIER IST P1!
```

und dann das nächste PGM von der Kassette laden. Damit Zeile 5000 aber erhalten bleibt, muß man unmittelbar vor dem LOAD-Befehl einfügen:

```
16 GOSUB4998
18 Z1=PEEK(Z4):Z2=PEEK(Z4+1)
```

und unmittelbar vor der Zeile 5000 (die erhalten bleiben soll):

```
4998 Z4=256*PEEK(202)+PEEK(201)+3
4999 RETURN
```

Z4 enthält die Dezimaladresse des Zeigers aus Zeile 4999, der auf die Zeile 5000 zeigt (siehe auch FW09). $Z1+256*Z2$ ergibt die Dezimaladresse des Beginns von Zeile 5000.

Fügen wir zur Sicherheit noch Zeile 9 hinzu, so erhalten wir:

```
9 REM DIESE ZEILE DIENT NUR ZUR VERGROESSERUNG VON P1
10 PRINT"HIER IST P1!"
16 GOSUB4998
18 Z1=PEEK(Z4):Z2=PEEK(Z4+1)
20 LOAD
4998 Z4=256*PEEK(202)+PEEK(201)+3
4999 RETURN
5000 PRINT"DIESE ZEILE SOLL ERHALTEN BLEIBEN"
```

P1 kann jetzt mit SAVE aufs Band geschrieben werden.

In P2 muß als erstes der Zeiger auf Zeile 5000 gesetzt werden (Z1 und Z2 stammen von P1):

```
1 GOSUB3998
2 POKEZ3,Z1:POKEZ3+1,Z2
```



als letzte Befehle von P2 fügen wir ein:

```
3998 Z3=256*PEEK(202)+PEEK(201)+3
3999 RETURN
```

Z3 gibt die Dezimaladresse des Zeigers aus Zeile 3999 an, der auf das Ende des PGM's zeigt. Dieser Zeiger wird in Zeile 2 so umgestellt, daß er auf die (noch immer vorhandene) Zeile 5000 zeigt. Die Variablen Z1 und Z2 wurden durch das LOAD nicht gelöscht.

Wir haben also:

(P2)

```
1 GOSUB3998
2 POKEZ3,Z1:POKEZ3+1,Z2
10 PRINT"HIER IST P2!"
20 GOT05000
3998 Z3=256*PEEK(202)+PEEK(201)+3
3999 RETURN
```

P2 kann jetzt hinter P1 auf das Band geschrieben werden. Rückspulen, dann LOAD und RUN ergibt, wie gewünscht:

```
HIER IST P1!
HIER IST P2!
DIESE ZEILE SOLL ERHALTEN BLEIBEN
```

ABSCHNEIDEN UND RUNDEN VON DEZIMALZAHLEN

```
10 INPUT"ZAHL,NACHKOMMASTELLEN";Z,N
20 PRINT INT(Z*10↑N)/10↑N :REM ABSCHNEIDEN
30 PRINT INT(Z*10↑N+.5)/10↑N :REM RUNDEN
40 GOT010
```



commodore

Commodore Büromaschinen GmbH
Frankfurter Straße 171-175
Postfach 4 26
6078 Neu-Isenburg
Telefon (0 61 02) * 80 03 · Telex 4185663 como d