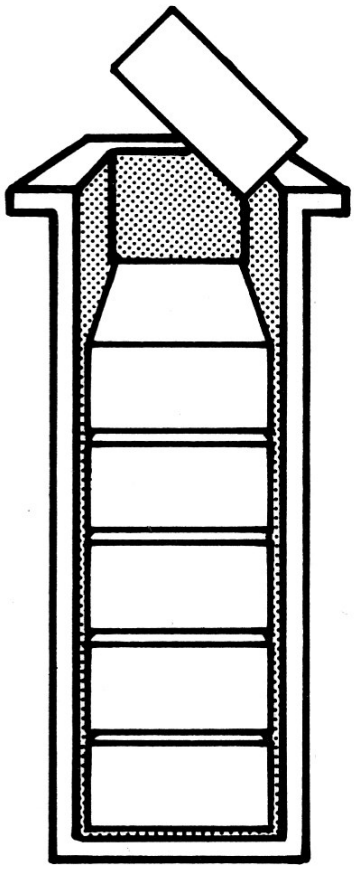
FORTH

Das DATA BECKER FORTH

für Commodore 64



EIN DATA BECKER PROGRAMM

Wichtiger Hinweis

Das vorliegende Handbuch und das dazugehörige Programm wurden vom Autor mit größter Sorgfalt erarbeitet und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf Programmfehler oder fehlerhafte Angaben im Handbuch zurückgehen, übernommen werden kann. Für die schriftliche Mitteilung eventueller Fehler sind wir jederzeit dankbar.

DATA BECKER A
Merowingerstr. 30
4000 Düsseldorf

DATA BECKER INFORMIERT :

---

1) LADESCHWIERIGKEITEN

Bei der Vervielfältigung unserer Software verwenden wir ausschließlich qualitativ hochwertige Disketten, die während des Dupliziervorgangs auf 100%ige Lesefähigkeit geprüft werden. Schon bei der kleinsten Unstimmigkeit weist unser Dupliziercomputer den Datenträger zurück. Eine derart überprüfte Programmdiskette müßte von jedem 1541-Laufwerk von Commodore problemlos gelesen werden können. Haben Sie trotz alledem Leseschwierigkeiten bei DATA BECKER Software, so ist der Fehler in den meisten Fällen bei Ihrer Diskettenstation zu suchen. Eine Floppy kann schon nach wenigen Bestriebsstunden "dejustiert" sein, d.h. der Schreib-Lesekopf wird nicht mehr 100% richtig positioniert. Bei herkömmlicher Software (z.B. eigenen Programmen) tritt kein Lesefehler auf, da diese Programme ja auch mit einem schlecht justierten Laufwerk gespeichert wurden. Unsere Programmdisketten stellen jedoch höhere Ansprüche an die Justage der Diskettenstation.
Überprüfen Sie deshalb bitte, ob diese Diskette sich bei einem Freund oder Ihrem DATA BECKER Händler in den C-64 laden läßt. Ist dies der Fall, so müßten Sie Ihre Floppy justieren lassen. Ergeben sich auf anderen Floppys dieselben Probleme, so kann es sein, daß die Diskette z.B. auf dem Postwege (durch magnetische Aussonderungsverfahren) Schaden genommen hat. In diesem Fall wird Ihre Diskette natürlich problem- und kostenlos umgetauscht.

2) UMTAUSCH DEFEKTE DISKETTEN

Sollte aus irgendeinem anderen Grund Ihre Diskette beschädigt oder gar zerstört werden (z.B. mechanische Schäden durch Anwenderverschulden), so senden Sie uns bitte Ihre Originaldiskette mit einem Verrechnungsscheck über DM 10.- zurück. Sie erhalten dann umgehend eine neue Programmdiskette.

3) KOPIERVERSUCHE KÖNNEN DIE DISKETTE ZERSTÖREN

Unsere Programmdisketten sind ausnahmslos gegen Kopieren geschützt ! Natürlich ist kein Kopierschutz absolut sicher, aber wir werden mit allen uns zur Verfügung stehenden rechtlichen Mitteln versuchen, den Schwarzkopierern das Handwerk zu legen.

1. EINLEITUNG

"FORTH soll alles können, was andere Programmiersprachen auch können - nur eleganter".

Das war der Anspruch, den Charles H. Moore sich bei der Entwicklung der von ihm erfundenen Sprache FORTH stellte.

In der Tat wird die ständig wachsende FORTH-Jüngerschaft nicht müde, Beispiele dafür zu liefern, wie Programme aus anderen Programmiersprachen in FORTH schneller und kürzer (und oft schöner) realisiert werden können. Beispielsweise wurden Compiler in FORTH geschrieben und auch die berühmte TURTLE-Graphik der Sprache LOGO wurde in FORTH bereits implementiert - schnell und elegant. Besonders gern wird FORTH auch zur Realisierung von Echtzeitproblemen verwendet, da FORTH-Programme schnell, kompakt und effizient sind.

FORTH ist aber nicht nur Programmiersprache, sondern gleichzeitig auch Betriebssystem. Mit FORTH steht Ihnen ein Gesamtsystem zur Verfügung, mit dem Sie programmieren, assemblieren und editieren können, mit dem Sie (im speziellen Fall von FORTH) Graphik und Musik erzeugen können, mit dem Speicherdumps und Programmpatches gemacht werden können, und mit dem Sie beliebige Erweiterungen definieren können, die dann sofort im gesamten System resident werden.

FORTH besteht aus vielen hundert Wörtern, die alle ein eigenes Programm bezeichnen, das bei Aufruf sofort ausgeführt wird. Allerdings ist FORTH eine nicht ganz leicht zu erlernende Sprache, da sie sich von allen höheren Programmiersprachen gänzlich unterscheidet. FORTH benutzt im Normalfall nur den Stack als Schnittstelle zur Parameterübergabe und führt u.a. keine spezielle Parameterprüfung durch.

Außerdem verwendet FORTH die unübliche UPN- oder Postfix-Notation, die sehr gewöhnungsbedürftig ist. Trotzdem werden Sie sich am Anfang, wenn Sie merken, daß in FORTH schon ein Wort genügt, um allerhand bewirken zu können, und später, wenn Sie eine gewisse Durststrecke beim Lernen dieser Sprache hinter sich gebracht haben, an der Effizienz und Eleganz dieser Sprache erfreuen.

I N H A L T S V E R Z E I C H N I S

1. EINLEITUNG.....1

2. FORTH.....3
   2.1. ALLGEMEINES.....3
   2.2. ERSTES ARBEITEN MIT FORTH.....4
   2.3. EINGABE VOM TERMINAL.....5
   2.4. DAS SCREENFILE ´SCR´.....6

3. DER FORTH-EDITOR.....9
   3.1. ALLGEMEINES.....9
   3.2. EINFACHES EDITIEREN.....10
   3.3. ÜBERSICHT DER EDITORPUFFER.....12
   3.4. BESCHREIBUNG ALLER EDITOR-BEFEHLE.....13
      3.4.1. SCREEN-BEFEHLE.....13
      3.4.2. ZEILENBEFEHLE.....14
      3.4.3. ZEICHENBEFEHLE.....15
      3.4.4. BEENDEN DES EDITIERVORGANGS.....17
   3.5. FUNKTIONSTASTEN DES EDITORS.....17
   3.6. ANMERKUNG.....18

4. PROGRAMMIEREN IN FORTH.....19
   4.1. ALLGEMEINES.....19
   4.2. DER STACK.....20
   4.3. BEISPIELE.....21

5. MUSIK MIT DEM SOUND-VOKABULAR.....25
   5.1. BESCHREIBUNG ALLER SOUND-BEFEHLE.....25
   5.2. EINE ERSTE KOMPOSITION MIT ´SOUND´.....26

6. GRAPHIK MIT DEM GRAPHIC-VOKABULAR.....29
   6.1. LO-RES.....29
   6.2. HI-RES.....30
   6.3. BESCHREIBUNG ALLER GRAPHIC-BEFEHLE.....34

7. DER FORTH-ASSEMBLER.....37
   7.1. ALLGEMEINES.....37
   7.2. DAS ASSEMBLER-VOKABULAR.....38
   7.3. EIN KLEINES ASSEMBLER-PROGRAMM.....39

8. SONSTIGES.....41

ANHANG 1.....43

ANHANG 2.....45

2. FORTH

2.1. ALLGEMEINES

FORTH entstand aus dem, von der in San Carlos, USA, ansässigen FORTH INTEREST GROUP (FIG) hervorgebrachten, Standard FIG-FORTH.

Natürlich werden heutige FORTH-Versionen mit erheblich mehr Wörtern angeboten, und so ist auch in diesem FORTH der Grundwortschatz gegenüber dem Standard etwa verdreifacht worden.

Zunächst sind nahezu alle Vokabeln des zweiten FORTH-Standards FORTH79 in dieses FORTH eingebracht worden. Weiterhin wurden einige Wörter der jüngsten FORTH-Generation FORTH83 in FORTH aufgenommen. Leider existieren einige Wörter in FORTH83 mit einem, gegenüber früheren FORTHs, veränderten Programm, einige weitere Wörter wurden umbenannt und sind in der deutschsprachigen FORTH-Literatur noch nicht zu finden. Bei solchen Doppeldeutigkeiten (z.B. haben die Wörter PICK und ROLL eine leicht veränderte Funktion, und das alte Wort ´0=´ wurde in FORTH83 durch NOT ersetzt) wird in FORTH die alte Syntax und Semantik verwendet.

Unabhängig vom FORTH-Standard enthält FORTH eine Anzahl weiterer, sehr nützlicher Wörter, mit denen z.B. SID (Sound Interface Device) und VIC (Video Interface Controller) angesprochen, mit denen also Musik und Graphik gemacht werden kann. FORTH enthält selbstverständlich den speziellen FORTH-Assembler, mit dem Sie (in der FORTH-eigenen Notation) beliebige Assemblerprogramme erstellen können, und FORTH bietet einen komfortablen Editor. Vergessen werden dürfen auch nicht die Wörter, mit denen Sie externe Geräte ansprechen (Floppy) und Zeichenketten (ähnlich BASIC) verarbeiten können, und die vielen Hilfsmittel wie DUMP, HELP und der TRACE-Befehl.

Sollte Ihnen für Ihre speziellen Anwendungen trotzdem noch das eine oder andere Wort fehlen, können Sie es sich jederzeit selbst definieren, und es steht Ihnen - wie jedes andere FORTH Wort auch - sofort zur Verfügung.

An dieser Stelle soll nur noch auf die sehr komfortable Möglichkeit hingewiesen werden, Funktions- oder beliebige andere Tasten zu belegen, d.h. FORTH-Wörtern zuzuordnen.

FORTH macht selbst davon Gebrauch und hat den Funktionstasten sowohl im Editor als auch im Standardmodus häufig benutzte Funktionen zugeordnet. Das Kommando HELP zeigt die jeweilige Tastenbelegung an, auch wenn die Zuordnung von Ihnen verändert worden ist.

Alle im Handbuch aufgeführten Beispiele werden Ihnen als Quellprogramme auf der Floppy mitgeliefert. Die Programme sind nützlich und können sofort geladen, ausprobiert und gegebenenfalls verändert werden.

2.2. ERSTES ARBEITEN MIT FORTH

Die FORTH-Diskette enthält drei Files:

-GFORTH	(das Ladeprogramm)
-FIG-FORTH	(das FORTH-Programm)
-SCR	(ein angelegtes Screenfile)

Geladen wird FORTH mit: LOAD "GFORTH",8,1 .

FORTH startet sich selbst, d.h. Sie brauchen nach dem Laden kein RUN oder ein ähnliches Kommando einzugeben.

FORTH meldet sich mit der üblichen Anfangsmeldung aus der Sie Version, Erstellungsdatum, und Programmnamen entnehmen können.

Die ersten Kommandos, die Sie nun geben können, sind beispielsweise VLIST, HELP oder auch .S .

Mit VLIST erhalten Sie sämtliche Wörter, des gerade aktuellen CONTEXT- und des FORTH-Vokabulars. Als CONTEXT-Wörterbuch kommt ASSEMBLER, EDITOR, GRAPHIC, SOUND, oder FORTH selbst in Frage. Nach dem Laden ist es FORTH.

Dauert Ihnen die Ausgabe der vielen Wörter, die vom Kommando VLIST ausgelistet werden, zu lange, können sie diese mit Hilfe der STOP-Taste abbrechen. Das ist im übrigen auch bei den meisten anderen, länger andauernden Befehlen möglich.

Das Kommando ´.S´ liefert Ihnen den Zustand des Stacks, der nach dem Laden von FORTH leer sein sollte. Wenn Sie aber eine Zahl eingeben (z.B. 3 CR), und sich anschließend den Stack ansehen, werden Sie diese Zahl solange dort wiederfinden, bis ein anderes FORTH-Wort diese Zahl entfernt hat, z.B. Kommando CLEAR CR oder ´. CR´. (CR bedeutet, daß Sie die RETURN-Taste betätigen müssen.)

Geben Sie noch das Kommando HELP ein. Es zeigt Ihnen die aktuelle Tastenbelegung an. Dabei wird eine Zuordnung von ASCII-Code zu FORTH-Wort hergestellt. Sie können beliebige Tasten beliebigen Wörtern zuordnen, allerdings mit der Einschränkung, daß das zugeordnete Wort möglichst keine Werte vom Stack benötigen sollte.

Vielleicht möchten Sie nun einige kleine Veränderungen an Ihrem System vornehmen. Die Zusammenstellung der Farben auf Ihrem Bildschirm gefällt Ihnen möglicherweise nicht.

Geben Sie einfach folgenden Satz ein:

```
BLUE BORDER  CYAN SCREEN  BROWN PEN CR
```

und schon sieht der Bildschirm ganz anders aus.

Vergessen Sie BASIC, vergessen sie die POKEREI von Werten an Adressen, die sich kein Mensch merken kann!

In FORTH sind den Farbcodes 0 bis 9 Konstanten - also Wörter - zugeordnet. Schreiben Sie BLACK, wenn Sie BLACK meinen und nicht 0! Sollten Sie mit der englischen Sprache auf Kriegsfuß stehen, können Sie sich selbstverständlich auch deutsche Wörter definieren:

```
0 CONSTANT SCHWARZ CR
: RAND BORDER ; CR
```

Dann können Sie sich auch einen schwarzen Rand mit den Kommando

```
SCHWARZ RAND CR
```

beschaffen. Zugegeben, kein makelloses Deutsch, aber immerhin verständlich.

Sie können auch ein neues Wort für Ihre liebste Gesamteinstellung definieren:

```
: FCN BLACK BORDER RED SCREEN WHITE PEN ; CR
```

wenn dies Ihre Lieblingsfarben sind.

2.3. EINGABE VOM TERMINAL

In FORTH war es ursprünglich vorgesehen, die Eingabe im KEY-Modus ablaufen zu lassen, d.h. jedes Zeichen anzunehmen und zu verarbeiten. Damit würde man FORTH aber völlig unnötigerweise eine Arbeit aufbürden, die das Betriebssystem automatisch erledigt, wenn Sie die entsprechende Routine aufrufen. Daher wurde in FORTH nicht so verfahren. Die Übernahme von Zeichen erfolgt in FORTH genauso wie in BASIC, also erst, wenn Sie eine editierte Zeile mit CR abschicken.

Eine kleine Modifikation an dieser Eingabefunktion wurde jedoch (zu Ihrem Vorteil) durchgeführt:

Beim ersten Zeichen, das Sie nach dem FORTH-üblichen "OK" eingeben, wird geprüft, ob es in einer sogenannten Zuordnungstabelle enthalten ist, die eine Verbindung von Eingabezeichen und FORTH-Funktionen herstellt. Ist dies der Fall, wird diese Funktion ausgeführt, und anschließend wieder eine Eingabe erwartet.

Dies wird solange fortgeführt, bis eine Taste betätigt wird, die nicht in dieser Tabelle enthalten ist. In diesem Fall wird die normale Zeileneingaberoutine abgearbeitet, wobei das erste eingegebene Zeichen selbstverständlich nicht verloren geht.

Ein Beispiel für eine eigene Belegung finden Sie in Kapitel 5.

2.4. DAS SCREENFILE 'SCR'

Natürlich wird es Ihnen eines Tages keinen Spaß mehr machen, Ihre langen Programme immer wieder im Dialog einzugeben. Sie können diese mit Hilfe eines Editors in ein File schreiben, um dieses dann bei Bedarf nachzuladen. FORTH kennt nur ein einziges File. Dieses heißt bei FORTH 'SCR'. 'SCR' ist ein Relativfile, das in jeder beliebigen Größe, am besten auf einer eigenen Diskette, eingerichtet werden kann. FORTH verwaltet dieses FILE virtuell, d.h. Sie können beliebige Stellen dieses Files ansprechen, vorwärts und rückwärts blättern, und brauchen sich um den Platz, den Sie dazu im Rechner zur Verfügung haben, keine Gedanken zu machen. Den Ein- und Aus-Transfer besorgt FORTH selbständig. Lediglich das Ende einer Editierung müssen Sie dem Rechner mitteilen, damit dieser eventuell im Speicher aktualisierte Zeilen auf Floppy sichert. Dies geschieht mit dem Befehl DONE.

Das Screenfile ist in FORTH in einzelne logische Seiten (Screens) unterteilt. Eine Seite enthält 16 Zeilen zu je 64 Zeichen. Eine Seite ist auch die zum Editieren vorgelegte logische Einheit. Sie können in FORTH jederzeit jede Seite anwählen, ansehen und bearbeiten. Zwei Seiten (Screens) sind von FORTH standardmäßig mit Fehlermeldungen belegt. Es handelt sich um die Seiten 4 und 5. Vergessen Sie beim Einrichten eines neuen Screenfiles nicht, diese Seiten ebenfalls zu kopieren. Sie würden sonst im Falle eines Fehlers keine Fehlermeldung erhalten, was die Beseitigung nicht gerade erleichtert.

Bevor Sie eigene Programme editieren, legen Sie sich sicherheitshalber eine eigene Datenfloppy an. Sie lernen dabei gleich einige neue Wörter von FORTH kennen:

- Originaldiskette einlegen
- EMPTY-BUFFERS 4 LIST 5 LIST CLOSE-SCREEN CR
  (Fehlermeldungen einlesen)
- neue Floppy ins Laufwerk legen
- " N:floppyname,id" DOS CR (Achtung: die Leerzeichen hinter " sind relevant! Das Kommando formatiert Ihre Floppy; alle auf der Floppy gespeicherten Daten werden zerstört!)
- CREATE-SCREEN CR (Es wird ein Screenfile mit 2200 Sätzen angelegt und die Seiten 4 und 5, die sich im Diskettenpuffer befinden, werden auf Floppy kopiert. Das Anlegen dieses Files dauert einige Minuten.)

Sie haben nun eine vollwertige FORTH-Datendiskette, mit der Sie sofort arbeiten können. Selbstverständlich können Sie auf diese Weise beliebig viele Datenfloppies erstellen.

3. DER FORTH-EDITOR

3.1. ALLGEMEINES

Eine der Eigenheiten von FORTH besteht darin, daß Sie Ihre Texte und Programme nicht auf beliebig viele Files verteilen, sondern in ein einziges virtuelles File - es heißt in FORTH "SCR" - schreiben. Dieser virtuelle FORTH-Speicher ist in sogenannte Screens (logische Seiten) unterteilt. Sie können beliebige Seiten dieses Files anwählen und bearbeiten.

So wie beliebige Seiten 'random' zum Editieren angewählt werden können, so können diese Seiten auch in beliebiger Reihenfolge übersetzt werden, entweder durch Verweise am Ende jeder Seite oder über eine Kommandoseite.

Bearbeitungseinheit ist jedoch immer eine solche Seite!

Das Format einer FORTH-Seite ist genormt. Sie besteht aus 16 Zeilen zu je 64 Zeichen. Die Maximalanzahl der Seiten hängt von dem auf der Floppy zur Verfügung stehenden Speicherplatz ab.

Eine weitere Vereinbarung besteht darin, daß die Seiten 4 und 5 Fehlermeldungen enthalten. Diese Seiten sollten nicht mit Programmen überschrieben werden, da sonst bei fehlerhaften FORTH-Anweisungen keine sinnvollen Fehlertexte mehr erscheinen.

Wenn Sie sich diese beiden Seiten anschauen (4 LIST 5 LIST CR), erkennen Sie, daß einige Zeilen nicht beschrieben sind. (CR bedeutet, daß Sie die RETURN-Taste betätigen müssen.) Diese Zeilen können Sie mit eigenen Fehlermeldungen beschreiben. Zeile 15 auf Seite 4 hat schließlich noch eine besondere Bedeutung. Sie soll einen kurzen Text zur Identifikation der Diskette enthalten, z.B. Angaben über deren Inhalt und Erstellungsdatum. Sie wird von den der Dokumentation dienenden Wörtern TRIAD und INDEX ausgewertet.

Sie können ihre 'message' mit dem Befehl 15 MESSAGE CR empfangen.

Einen Einblick, wie in FORTH editiert wird, erhalten Sie anhand des Beispiels im nächsten Kapitel, in dem Sie die 'Disketten-Identifizierungszeile' verändern können. Über die vielen Möglichkeiten, die hier nicht an Beispielen erklärt werden, verschaffen Sie sich am besten mit Hilfe der Beschreibung des EDITOR-Vokabulars in Abschnitt 3.4 einen Überblick.

3.2. EINFACHES EDITIEREN

Zum Editieren von Seite 4 geben Sie folgendes Kommando:

```
4 EDIT CR
```

Nun wird Ihnen die Seite 4 mit den entsprechenden Fehlermeldungen vorgelegt:

```
SCR # 4
 0 P ( ERROR MESSAGES )
 1 P EMPTY STACK !
 2 P DICTIONARY FULL !
 3 P HAS INCORRECT ADDRESS MODE
 4 P ISN'T UNIQUE
 5 P
 6 P DISC RANGE ?
 7 P FULL STACK !
 8 P DISC ERROR !
 9 P
10 P
11 P
12 P
13 P
14 P
15 P  MASTERDISKETTE FORTH
```

Die Zeilennummern sind von 0 bis 15 fest vergeben und nicht, wie beispielsweise in BASIC, frei wählbar. Das P hinter der Zeilennummer bedeutet PUT oder PLACE und bewirkt, daß der dahinter (durch Blank getrennte) Text in die entsprechende Zeile geschrieben wird.

Sie werden vielleicht bemerkt haben, daß das FORTH-übliche 'OK' nicht ausgegeben wird. Das hat seinen Sinn, denn nun können Sie den Cursor an die zu ändernde Stelle setzen und die entsprechenden Änderungen vornehmen. Beim Abschicken des P-Befehls mittels CR wird Ihnen auf diese Weise nicht der Anfang der nächsten Zeile zerstört. Sie können also wie gewohnt weiter editieren.

Wenn Sie den Cursor nicht über den Bildschirm bewegen wollen, können Sie ihre Zeilen auch direkt eingeben, ohne die Vorlage zu benutzen. Wenn Sie Screens haben, die mehr als 7 Zeilen mit über 35 Zeichen enthalten, rollen Ihnen die obersten Zeilen ohnehin davon, und Sie müssen die ersten Zeilen auf diese Weise editieren.

Außer den Zeilen 0 (enthält Screeninfo) und 15 (da stört es nicht mehr) sollten Sie keine Zeilen 'doppelzeilig' eingeben. FORTH-Programme werden ansonsten sehr unübersichtlich, und unübersichtliche FORTH-Programme sind selbst vom Entwickler nach einiger Zeit kaum mehr zu verstehen.

Wenn Sie die Seite 4 nach Ihrem Geschmack verändert haben (Sie dürfen die Texte z.B. auch ins Deutsche übersetzen), speichern Sie die Veränderungen am besten mit dem Kommando SSAVE (F7) auf Floppy.

Mit den Kommandos +EDIT und -EDIT (F1,F2) gelangen Sie zur nächsten bzw. vorigen Seite. Wenn Sie nicht mehr editieren wollen, geben Sie das Kommando DONE (F8) ein. Es impliziert u.a. SSAVE, und speichert sogar alle Veränderungen im Diskettenpuffer auf Floppy ab, falls Sie SSAVE am Ende einer editierten Seite einmal vergessen haben sollten. Auch die OK-Meldung erscheint, und Sie befinden sich wieder im normalen FORTH-Modus.

Zur Kontrolle, ob Ihre Veränderung etwas bewirkt hat, sehen Sie sich bitte das Inhaltsverzeichnis der Seiten 4 bis 5 mit dem Kommando INDEX an:

```
4 5 INDEX CR
```

Auf dem Bildschirm erscheinen die Kommentarzeilen der Seiten 4 und 5 und die Diskettenidentifizierungszeile. Letztere müßte Ihren neuen Text enthalten.

Als weitere Übung könnten Sie sich Ihre bevorzugte Farbvoreinstellung für Bildschirm und Rand auf einer Seite definieren und diese Seite nachladen, nachdem Sie FORTH geladen haben. Das Laden einer Seite geschieht mit dem Kommando LOAD, z.B.:

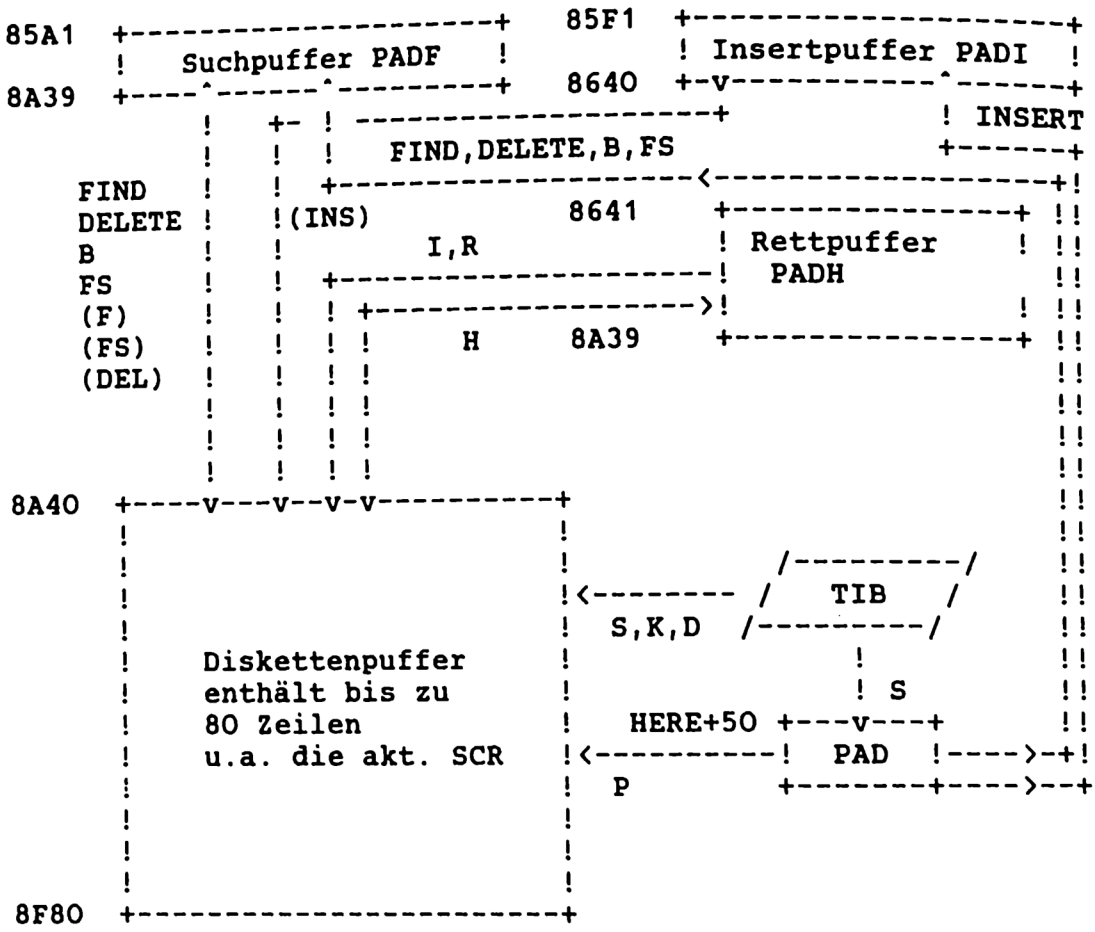
```
10 LOAD CR
```

wenn Ihr Programm auf Seite 10 steht.

Wenn Seite 10 die zuletzt editierte Seite war, bzw. wenn in der Variablen SCR der Wert 10 steht, brauchen Sie nur die RUN-Taste Ihres Rechners zu betätigen, die mit der Funktion ELOAD belegt ist.

Falls FORTH beim Laden einer Seite einen Fehler entdeckt, z.B. ein Wort nicht erkennt, brauchen Sie nur das Kommando WHERE zu geben, und die fragliche Seite wird Ihnen mit einer Marke an der fehlerhaften Stelle zum Editieren vorgelegt. Verbessern Sie die Stelle! Mit DONE und 10 LOAD (kürzer F8 und RUN-Taste) können Sie es noch einmal versuchen.

3.3. ÜBERSICHT DER EDITORPUFFER



Alle Puffer bis auf PAD werden nur vom Editor benutzt und können sonst anderweitig verwendet werden. Sie sind in diesem Fall allerdings vor Benutzung des Editors mit EMPTY-BUFFERS zu löschen.

3.4. BESCHREIBUNG ALLER EDITOR-BEFEHLE

3.4.1. SCREEN-BEFEHLE

Im folgenden werden die Befehle beschrieben, die auf eine oder mehrere Seiten (SCREENS) wirken:

EDIT n --

Anwahl der n-ten Seite zum Editieren.

OK wird ausgeschaltet, interne Cursorposition wird an den Beginn der Zeile 0 gesetzt (Variable R#), die Seite wird vorgelegt, der Cursor auf die erste Zeile positioniert. Vorgelegte Seite kann editiert werden.

E --

die aktuelle Seite (Seitennummer in SCR) wird wieder (wie bei EDIT) vorgelegt.

+EDIT --

die nächste Seite wird zum Editieren vorgelegt.

-EDIT --

die vorherige Seite wird zum Editieren vorgelegt.

WIPE --

die aktuelle Seite wird gelöscht (nur im Diskettenpuffer). Nach dem Kommando E wird Ihnen eine leere Seite vorgelegt.

SSAVE --

die aktuelle Seite wird auf Floppy gesichert, vorausgesetzt, daß sie verändert wurde.

SCRATCH --

Die Veränderungen in der editierten Seite werden im Diskettenpuffer ungültig gemacht. Der zuletzt gespeicherte Stand kann mit dem Kommando E weiter editiert werden.

3.4.2. ZEILENBEFEHLE

Befehle die eine oder mehrere ganze Zeilen manipulieren:

P i --

plaziert den folgenden Text in Zeile i der aktuellen Seite. Die interne Cursorpositionierung wird auf die folgende Seite vorgeschaltet (wichtig beim Suchen).

S i n --

spreizt die aktuelle Seite ab Zeile i um n Zeilen. Die letzten n Zeilen gehen verloren.

R i --

ersetzt die aktuelle und die folgenden Zeilen durch den Inhalt des Rettpuffers. Dieser kann durch die Kommandos H oder XH gefüllt worden sein.

I i --

fügt den Inhalt des Rettpuffers ab Zeile i ein. Die letzten Zeilen der Seite fallen heraus.

H i n --

kopiert n Zeilen ab Zeile i der aktuellen Seite in den Rettpuffer.

K i n --

löscht n Zeilen ab Zeile i in der aktuellen Seite.

D i n --

fügt n Leerzeilen ab Zeile i in der aktuellen Seite ein.

XH s i n --

"External HOLD". Kopiert n Zeilen ab Zeile i des SCREEN s in den Rettpuffer. Damit können auch Auszüge von anderen Seiten in die aktuell editierte gebracht werden, ohne jene extra anwählen zu müssen.

3.4.3. ZEICHENBEFEHLE

Folgende Befehle wirken auf einzelne Zeichen einer Zeile bzw. auf den internen Cursorzeiger R#. Such- und Löschbefehle wirken ab der in R# gespeicherten Cursorposition. Such- und Insertpuffer werden beim Aufruf des Editors nicht initialisiert, um den Inhalt derselben auch über mehrere Editieraufrufe erhalten zu können. Beim ersten Such- bzw. Insert-Aufruf sind die entsprechenden Parameter also unbedingt mitzugeben!

TOP --

setzt den internen Cursorzeiger auf 0 (Zeile 0, Spalte 0).

L i --

setzt den internen Cursorzeiger auf den Anfang der Zeile i (Zeile 1, Spalte 0).

!R# n --

Cursorzeiger wird auf n gesetzt.

+!R# n --

Cursorzeiger wird um n Positionen weitergezählt.

V --

schreibt die Zeile, in die der Cursorzeiger zeigt, auf den Bildschirm. Die Cursorposition wird durch inverse Darstellung des betreffenden Zeichens angezeigt.

X --

Rest der Zeile wird ab Cursorposition gelöscht.

FIND --

Der folgende, durch Trennzeichen POUND eingeschlossene Text wird ab der aktuellen Cursorposition gesucht. Der Text wird im Suchpuffer aufgehoben. Das Trennzeichen befindet sich in der Variablen DELI und kann bei Bedarf verändert werden. Wird ein leerer String eingegeben, (z.B. FIND && CR), verändert sich der Suchpuffer nicht. Es wird dann nach dem zuletzt eingegebenen Suchbegriff gesucht. Wird kein Trennzeichen (POUND-Zeichen) in der folgenden Zeichenkette gefunden, so wird der gesamte Rest der Eingabezeile als Suchbegriff verwendet.

(INSERT) --

wie INSERT, jedoch wird Text aus dem Insertpuffer eingefügt. Es gilt sinngemäß dasselbe wie bei (FIND).

(FS) --

wie FS, jedoch Texte aus Insertpuffer und Suchpuffer.

(DELETE) --

wie DELETE, jedoch Text aus Suchpuffer.

3.4.4. BEENDEN DES EDITIERVORGANGS

DONE --

Es erfolgt Umschaltung auf das FORTH-Vokabular. Der Editiermodus wird verlassen. Vorher werden sämtliche aktualisierten Zeilen (sofern nicht schon geschehen) auf Floppy geschrieben. OK-Meldung wird reaktiviert und Editorfunktionstasten werden inaktiviert.

3.5. FUNKTIONSTASTEN DES EDITORS

Im FORTH-Editor haben die Funktionstasten eigene Zuordnungen erhalten. Sie sind über das Kommando HELP zu erfragen.

F1:	+EDIT	editiere nächste Seite
F2:	-EDIT	editiere vorherige Seite
F3:	(DELETE)	aktuellen Suchbegriff noch einmal löschen
F4:	WIPE	lösche aktuelle Seite
F5:	(INSERT)	aktuellen INSERT-Puffer noch einmal einfügen
F6:	(FIND)	aktuellen Suchbegriff noch einmal suchen
F7:	SSAVE	editierte Seite abspeichern
F8:	DONE	editieren beenden
SHIFT RUN:	ELOAD	zuletzt editierte Seite laden
↑ :	E	aktuelle Seite erneut vorlegen

Die Funktion FS kann dann durch F3 F5 beliebig oft wiederholt werden. Such- und Insertpuffer werden auch bei Seitenwechsel nicht gelöscht, so daß beispielsweise eine Substitution durch die Funktionstasten F1 F3 F5 und - bei Beendigung der Editierung - F7 beliebig wiederholt werden kann.

3.6. ANMERKUNG

Wenn Sie editieren wollen, FORTH aber die eingegebenen Edit-Kommandos nicht annimmt, haben Sie den Editor zwischenzeitlich verlassen! Geben Sie (erneut?) das Kommando i EDIT CR bzw. E CR.

4. PROGRAMMIEREN IN FORTH

Dieses FORTH-Handbuch ist kein Lehrbuch zur FORTH-Programmierung. Wenn Sie FORTH erlernen und anspruchsvollere FORTH-Programme schreiben wollen, benötigen Sie unbedingt ein entsprechendes Lehrbuch (z.B. das "Trainingsbuch zu FORTH" von *DATA BECKER*).

In diesem Kapitel sollen nur einige Grundzüge von FORTH erklärt werden, ohne die Sie die folgenden Kapitel kaum verstehen können. Es werden kleine Beispielprogramme beschrieben, die im Gegensatz zu den Programmen der folgenden Kapitel nicht auf der Programmdiskette enthalten sind. Sie sollten sich anhand dieser Beispiele mit dem Editor von FORTH einerseits und mit der Sprache FORTH andererseits vertraut machen.

4.1. ALLGEMEINES

Die Sprache FORTH besteht wie jede andere Sprache auch aus Wörtern. Jedes dieser Wörter entspricht einem Befehl an das Programm etwas auszuführen. Im Anhang dieses Handbuchs sind alle zum Grundwortschatz von FORTH gehörenden Wörter beschrieben. Es gibt davon viele hundert, mit denen Sie sich nach und nach vertraut machen können. Einige (z.B. HELP oder VLIST) sind schon in früheren Kapiteln beschrieben worden.

Die meisten Wörter erhalten jedoch erst mit weiteren Informationen eine Bedeutung. Genauso wie Sie Niemandem den Auftrag ´Schreibe!´ geben können, ohne ihm zu sagen, was er schreiben soll, ist es sinnlos FORTH das entsprechende Kommando ´. CR´ zu geben, ohne anzugeben, was auf dem Bildschirm ausgegeben werden soll.

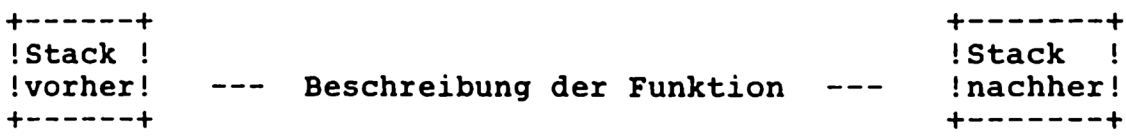
Die Anweisung ´5 . CR´ hingegen kann verarbeitet werden, FORTH gibt die Zahl 5 auf dem Bildschirm aus. Zu beachten ist, daß Sie FORTH zuerst den (die) Operanden und dann den Operator nennen müssen, also zuerst die Zahl 5 und dann die Anweisung ´.´ .

Ebenso verhält es sich bei arithmetischen Ausdrücken. Die Summe zweier Zahlen 7 und 12 berechnen Sie in FORTH, indem Sie den Ausdruck ´7 12 + CR´ statt wie üblich ´7 + 12 =´ eingeben.

Jetzt muß nur noch geklärt werden, wie FORTH-Wörter eine Bedeutung erhalten bzw. woher diese wissen, was sie verarbeiten sollen. Wir beschreiben daher jetzt den sogenannten Stack (deutsch: Stapel).

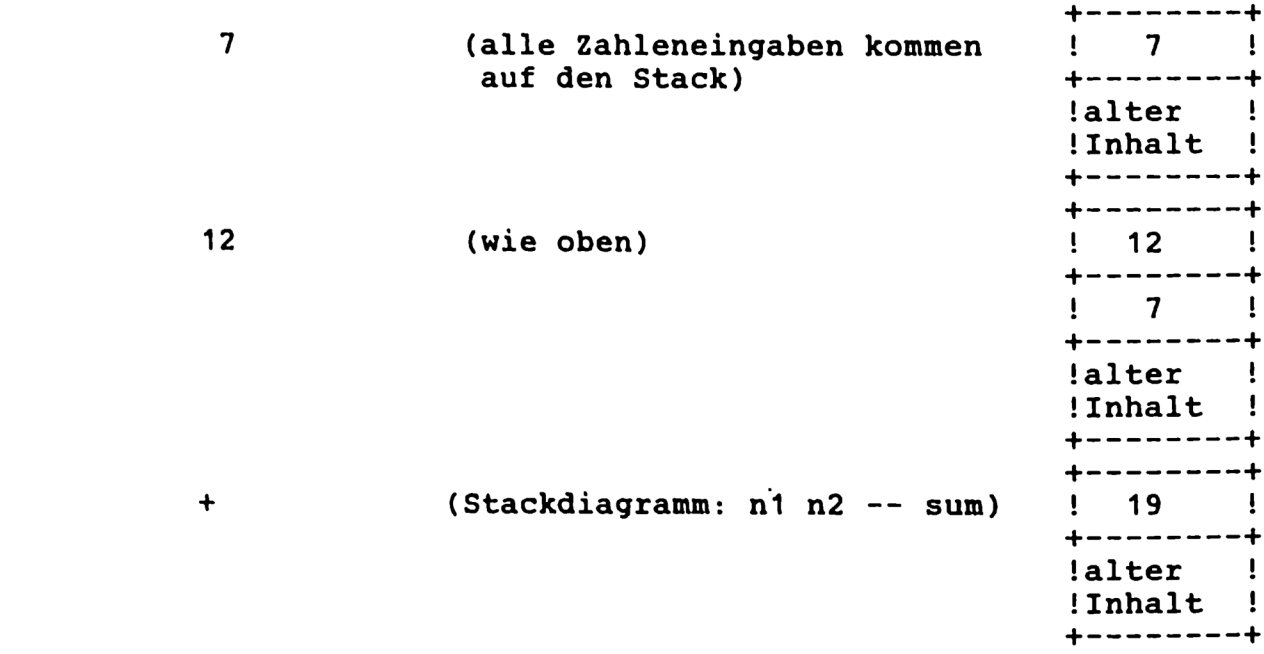
4.2. DER STACK

In FORTH kommunizieren alle Programme über eine einzige zentrale Nahtstelle - den Stack. Jedes Unterprogramm in FORTH, das zu seinem Ablauf mit Parametern versorgt sein muß, holt sich diese Parameter vom Stack. Andererseits müssen Sie vor dem Aufrufen dieses Wortes dafür Sorge tragen, daß die entsprechenden Parameter auf dem Stack vorliegen. Ansonsten holt sich das aufgerufene Wort irgendeine unsinnige Information, was in ungünstigen Fällen zum Systemabsturz führen kann. Ein FORTH-Wort beschreibt man daher am besten so, daß man den logischen Ablauf der Funktion darstellt, und zusätzlich die jeweiligen Eingangs- und Ausgangsparameter erklärt:

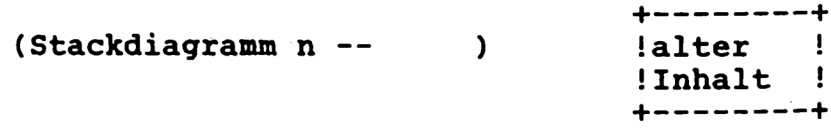


Genauso sind die Beschreibungen der FORTH-Wörter im Anhang auch zu verstehen.

Wir sehen uns einmal an, was beim Addieren zweier Zahlen ´7 12 + CR´ auf dem Stack abläuft.



Zur Ausgabe des Ergebnisses schreibt man noch:



Der Stack ist nun wieder im ursprünglichen Zustand.

4.3. BEISPIELE

Anhand von einigen kleinen Beispielen soll nun gezeigt werden, wie man in FORTH programmiert. Dabei bietet sich als erstes ein Sprachübersetzer bzw. der Aufbau eines beispielsweise deutsch-englischen Lexikons an. Wir benötigen dazu nur die beiden zusammengehörenden Wörter '."' und '"', sowie die zur Definition jedes Wortes gehörenden Definitionsbegrenzer ':' und ';'. Wir definieren:

```
: LIEBE    ." LOVE " ;
: ICH      ." I " ;
: KOENIGIN ." QUEEN " ;
: MARIA ." MARY " ;
```

ACHTUNG: Das Leerzeichen (Blank) ist in FORTH der einzige Begrenzer. Zwischen zwei FORTH-Wörtern muß daher (mindestens) ein Leerzeichen stehen (z.B. auch hinter ").

Sie können nun von jedem gelernten Wort die Übersetzung anfordern. Schreiben Sie KOENIGIN CR, und FORTH antwortet mit QUEEN. Sie können auch ganze Sätze übersetzen (allerdings grammatikalisch oft falsch):

```
ICH LIEBE KOENIGIN MARIA CR
```

Die Bezeichnung Lexikon für die Liste aller FORTH-Wörter bekommt in diesem Fall einen ganz wörtlichen Sinn.

Als nächstes werden wir ein Wort 2SORT definieren, das zwei Zahlen der Größe nach sortiert. Das Stackdiagramm dazu lautet:

```
n1 n2  ---  min(n1,n2) max(n1,n2)
```

Oben auf dem Stack soll am Ende von 2SORT der größere der beiden Werte stehen. Das dazugehörige Programm lautet:

```
: 2SORT                          STACK:  n1 n2
        2DUP                             n1 n2 n1 n2
        >                                n1 n2 f
        IF                               n1 n2
           SWAP                                 n2 n1
        ENDIF
;                                        n1 n2  oder n2 n1
```

Im Programm müssen die beiden Werte n1 und n2 erst einmal dupliziert werden, weil der Vergleichsoperator '>' zwei Werte vom Stack entfernt, dafür aber eine Kennung 'wahr' (=1) oder 'falsch' (=0) absetzt, je nachdem ob n1 > n2 wahr ist oder nicht. Das Wort IF entfernt die Kennung vom Stack. Der Zweig zwischen IF und ENDIF wird nur bei Kennung 'wahr' durchlaufen. In ihm werden die obersten beiden Zahlen des Stacks vertauscht.

Wir werden dieses Wort bei unserer nächsten Definition (Berechnung des größten gemeinsamen Teilers GGT zweier Zahlen) verwenden.

Der Euklidische Algorithmus zur Bestimmung des GGT zweier positiver ganzer Zahlen lautet etwa folgendermaßen:

- subtrahiere die kleinere der beiden Zahlen (z.B. n2) solange von der größeren n1, bis das Ergebnis n3 der Subtraktion kleiner als n2 ist.

- anschließend subtrahiere das Ergebnis n3 solange von n2, bis das neue Ergebnis n4 kleiner als n3 ist, usw.

- beende das Verfahren, wenn sich als Differenz 0 ergibt. Die zuletzt abgezogenen Zahl ist der GGT.

Das Stackdiagramm für das Wort GGT:

```
n1 n2 ---  ggt
```

Das zugehörige Programm kann so aussehen:

```
: GGT               n1 n2
    BEGIN           Die Schleife soll solange wiederholt
                    werden, bis kein Rest bei Division
                    auftritt
       2SORT        wir sorgen für die richtige Ordnung
       OVER         Stack: n2 n1 n2
                           klein groß klein
       MOD          Stack: n2 n3
       -DUP         wir duplizieren n3 nur, wenn n3 > 0
                    Fall A: n3>0, dann Stack: n2 n3 n3
                    sonst :                   n2 0
       0=
                    Fall A: n2 n3 0
                    sonst : n2 1
    UNTIL           Fall A: n2 n3   und zurück zu repeat
                    sonst : n2
;
```

Die Schleife wird bei UNTIL verlassen, wenn dort nicht 0 steht.

Das Programm ist schnell (ca. zehnmal schneller als ein analoges BASIC-Programm) und kann allgemein verwendet werden.

Zum Schluß dieses Kapitels soll noch ein Programm aufgeführt werden, das zwei Brüche addiert und das Ergebnis auch noch kürzt. An diesem Beispiel werden auch die Möglichkeiten und Gefahren in der Anwendung von FORTH deutlich.

Das Stackdiagramm für das Additionsprogramm von Brüchen lautet:

```
z1 n1 z2 n2 -- zsum nsum
```

wobei 'z' und 'n' die Abkürzungen für Zähler und Nenner sind.

Das FORTH-Programm:

```
: BRUCHADDITION       ( Z1 N1 Z2 N2 ----- ZSUM NSUM)
     DUP                Z1 N1 Z2 N2 N2
     5 ROLL             N1 Z2 N2 N2 Z1
     *                  N1 Z2 N2 P1
     3 ROLL             N1 N2 P1 AA
     4 PICK             N1 N2 P1 Z2 N1
     *                  N1 N2 P1 P2
     +                  N1 N2 ZSUM
     -ROT               ZSUM N1 N2
     *                  ZSUM NSUM
     2DUP               ZSUM NSUM ZSUM NSUM
     GGT                ZSUM NSUM GGT
     DUP                ZSUM NSUM GGT GGT
     -ROT               ZSUM GGT NSUM GGT
     /                  ZSUM GGT (GEK. NENNER)
     -ROT               (GEK. NENNER) ZSUM GGT
     /                  NENNER ZÄHLER
     SWAP
;
```

Das Programm Bruchaddition funktioniert, ist aber ziemlich unübersichtlich. So sollten FORTH-Programme besser nicht geschrieben werden. Lesbarer wird das Programm, wenn man sich Hilfsfunktionen schafft (Übungsaufgabe!), z.B folgende:

```
Funktion        Stackdiagramm
ZÄHLER          z1 n1 z2 n2 --- Zähler
                Zähler=z1*n2+z2*n1
KÜRZEN          z n teiler  --- zählergekürzt nennergekürzt
```

Die Addition von Brüchen lässt sich dann folgendermaßen formulieren:

```
: ADDBRUCH
     3 PICK          z1 n1 z2 n2 n1
     OVER            z1 n1 z2 n2 n1 n2
     *               z1 n1 z2 n2 nenner
     >R              z1 n1 z2 n2 (Nenner wird auf Return-
                                  Stack gerettet)
     ZÄHLER          zähler
     R>              zähler nenner
     2DUP            zähler nenner zähler nenner
     GGT             zähler nenner ggt
     KÜRZEN          zählergekürzt nennergekürzt
;
```

5. MUSIK MIT DEM SOUND-VOKABULAR

5.1. BESCHREIBUNG ALLER SOUND-BEFEHLE

FORTH enthält ein kleines Vokabular, mit dessen Hilfe Musik oder zumindest Töne erzeugt werden können. Das Vokabular heißt SOUND und enthält folgende Wörter:

VOICE VOLUME ENVELOPE NOISE PULSE SAWTOOTH TRIANGLE und SID.

Bevor Sie eines dieser Wörter aufrufen können, müssen Sie zuerst das Kommando SOUND geben.

ENVELOPE n1 n2 n3 n4 n5 ---

Definition der Hüllkurve der Stimme n1

n1	-	Stimme	(1.. 3)
n2	-	Anschlag	(1..15)
n3	-	Abschwellen	(1..15)
n4	-	Halten	(1..15)
n5	-	Ausklingen	(1..15)

VOICE n1 n2 n3 n4 n5 n6 ---

Erzeugung eines Tones in der Stimme n1

n1	-	Stimme	(1.. 3)
n2	-	Frequenz	(0..65535)
n3	-	Wellentyp	(17,33,65,129)
n4	-	Tastverhältnis	low (0..255)
n5	-	Tastverhältnis	high (0..15)

Das Tastverhältnis ist nur beim Wellentyp 65=PULSE relevant.

n6	-	Dauer	(-32768..32767)

Klangdauer des zu erzeugenden Tons. Pro Einheit wird eine verzögerte Dummy-Schleife einmal durchlaufen.

VOLUME n ---

Festlegung der Lautstärke (0..15)

NOISE --- 129

Die Konstante 129 wird zur Erzeugung eines Geräusches auf dem Stack abgelegt.

PULSE --- 65

Die Konstante 65 wird zur Erzeugung eines Tones mit Rechteckcharakteristik auf dem Stack abgelegt.

SAWTOOTH --- 33

Die Konstante 33 wird zur Erzeugung eines Tones mit Sägezahncharakteristik auf dem Stack abgelegt.

TRIANGLE --- 17

Die Konstante 17 wird zur Erzeugung eines Tones mit Triangelcharakteristik auf dem Stack abgelegt.

SID --- D400

Die Adresse des SID wird abgelegt (Dezimal 54272).

Das Kommando VOICE ist nach den Einstellungen für Lautstärke und Hüllkurve zu geben.

5.2. EINE ERSTE KOMPOSITION MIT 'SOUND'

Mit den oben angegebenen Wörtern können Sie Ihrem C64 gleich einmal ein paar Töne entlocken. Geben Sie folgende Kommandos ein:

15 VOLUME CR	(Volle Lautstärke für die neue Musik)
1 9 3 0 8 ENVELOPE CR	(Dies ist eine vom C64-Handbuch empfohlene Einstellung für eine Klavierstimme)

und schließlich

1 7493 PULSE 255 1 2000 VOICE CR (Erzeugung des def. Tones)

Sie hören jetzt den berühmten Kammerton A!

Wenn Sie die angegebenen Einstellungen für ENVELOPE und VOICE ein wenig ändern, werden sie jedesmal (nach VOICE) eine mehr oder weniger deutliche Klangveränderung gegenüber dem zuletzt erzeugten Ton bemerken..

Wenn Sie eine Kombination entdeckt haben, die Sie sich erhalten wollen, sollten Sie diese als eigenes Wort definieren.

Sie können anhand dieses Beispiels erkennen, wie schön es sich mit FORTH experimentieren lässt, und auf welch spielerische Weise in FORTH Programme geschrieben werden können.

Stellen Sie ähnliche Versuche mit den beiden Hüllkurven für die zweite und dritte Stimme an:

```
2 0 15 240 0 ENVELOPE CR
3 96 10 0 10 ENVELOPE CR
```

Im Commodore-Handbuch wird die zweite Einstellung in Verbindung mit der Wellenform TRIANGLE und die dritte Einstellung in Verbindung mit der Wellenform SAWTOOTH als günstige Parametrierung für Orgel- und Trompetenklänge angegeben. Eigene Experimente bringen Sie diesem Ideal sicher noch ein wenig näher!

Sie können folgendes Programm zum Ausgangspunkt einer Komposition für Klavier, Orgel und Trompete machen, einer heutzutage selten gewordenen kammermusikalischen Instrumentierung:

```
SCR # 15
 0 P ( CHAMBER MUSIC  #1,  PIANO-,ORGAN-,TRUMPET-DEFINITIONS
     10.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P                ( ENVELOPE DEFINITIONS)
 3 P : PIANO        (  -- WAVE )
 4 P    SOUND 1 9 3 50 8 ENVELOPE
 5 P ;
 6 P
 7 P : ORGAN        (  -- WAVE )
 8 P    SOUND 2 0 15 240 0 ENVELOPE
 9 P ;
10 P
11 P : TRUMPET      (  -- WAVE )
12 P    SOUND 3 96 0 0 5 ENVELOPE
13 P ;
14 P
15 P    -->     (DEFINITION AUF NAECHSTER SEITE FORTSETZTEN!)

SCR # 16
 0 P ( CHAMBER MUSIC #2, PLAY        10.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P : PLAY              (MACH MUSIK!)
 3 P    15 VOLUME        (VOLLE LAUTSTÄRKE)
 4 P    PIANO ORGAN TRUMPET (ENVELOPES)
 5 P    BEGIN
 6 P       3 RND 1+      (DIE STIMME BESTIMMT DER ZUFALL)
 7 P       CASE
 8 P         1 OF 1 PULSE    ENDOF
 9 P         2 OF 2 TRIANGLE ENDOF
10 P         3 OF 3 SAWTOOTH ENDOF
11 P       ENDCASE
12 P       5000 RND 5000 + SWAP (FREQUENZ)
13 P       240 5                (TASTVERHÄLTNIS)
14 P       16 RND 4 - 1000 * VOICE
15 P     ?TERMINAL UNTIL ;  (BIS STOP GEDRÜCKT)
```

MUSIK MIT DEM SOUND-VOKABULAR

Laden Sie das Programm mit 15 LOAD CR und lassen Sie es mit dem Kommando PLAY musizieren. Es spielt solange, bis Sie die STOP-Taste betätigen (vermutlich sehr bald).

Die Vertreibung der Räuber durch die Bremer Stadtmusikanten hätte zweifellos auch mit dem C64 Erfolg haben können.

6. GRAPHIK MIT DEM GRAPHIC-VOKABULAR

6.1. LO-RES

Sie können mit FORTH nicht nur Musik machen, sie können auch Bilder und Graphiken erzeugen. Dabei helfen Ihnen im HI-RES-Modus das GRAPHIC-Vokabular, im LO-RES-Modus einige Wörter des Grundvokabulars FORTH.

Einige Wörter zur Ausgabe auf den Bildschirm haben Sie bereits in Kapitel 2 kennengelernt: die Farben und die Wörter BORDER, SCREEN und PEN.

Wenn Sie an eine beliebige Stellen des Bildschirms ein Zeichen schreiben wollen, oder wissen wollen, welches Zeichen sich dort befindet, benötigen Sie die Wörter 'S!' und 'S@'. Beide Wörter benötigen die Bildschirmkoordinaten in der Form (Zeile,Spalte) auf dem Stack:

S@ z s -- c

liefert den Code des Zeichens in Zeile z, Spalte s

S! c z s --

schreibt ("poked") das Zeichen c in Zeile z, Spalte s auf den Bildschirm.

Ein kleines Programm, das den Zufallszahlengenerator testet, lautet folgendermaßen:

```
SCR # 20
 0 P ( ZUFALLSZAHLENTESTER ?RND   10.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P : ?RND              (  --  )
 3 P       (ERST BILDSCHIRM INITIALISIEREN)
 4 P       1024 1000 ASCII 0 FILL
 5 P       BEGIN
 6 P          1000 RND    (RANDOM 0.999)
 7 P          40 /MOD     (SPALTE,ZEILE)
 8 P          SWAP        (VERTAUSCHEN )
 9 P          2DUP S@     (CHARACTER   )
10 P          1+ -ROT     (ADDIERE 1   )
11 P          S!          (ABSPEICHERN )
12 P       ?TERMINAL UNTIL  (BIS STOP GEDRÜCKT)
13 P ;
14 P    ;S
15 P
```

Sie laden das Programm mit 20 LOAD und lassen es mit ?RND CR ausführen.

6.2. HI-RES

Feinere und schönere Bilder lassen sich im HI-RES-Modus erzeugen. Die Wörter, die Sie dazu benötigen, finden Sie im GRAPHIC-Vokabular, das im nächsten Abschnitt beschrieben wird. Schreiben Sie vor der Eingabe eines GRAPHIC-Kommandos das Wort GRAPHIC CR, sonst versteht FORTH die nachfolgenden Befehle nicht.

Wie der hochauflösende Bildschirm aussieht, bevor Sie etwas gemalt haben, sehen Sie, wenn Sie das Kommando &HI-RES oder einfacher F3 geben. Das Bild, das Sie nun sehen, besteht aus vielen, kleinen verschiedenfarbigen Kästchen. Löschen Sie das Bild mit dem Kommando &CLEAR.

Dieses Kommando müssen Sie, wenn Sie nicht wieder in den Textmodus zurückgeschaltet haben (&LO-RES oder F4), blind schreiben, d.h. Sie können die eingegebenen Zeichen nicht auf dem Bildschirm sehen. Trotzdem wird Ihre Eingabe angenommen und verarbeitet.

Auch im HI-RES-Modus können Sie Bildschirm- und Schreibfarbe mit Hilfe der GRAPHIC-Wörter &PAPER und &INK (unabhängig vom Textbildschirm) festlegen. Schreiben Sie:

```
GRAPHIC   YELLOW &PAPER   BLACK &INK   &CLEAR   CR
```

Sie können sich nun den Graphikbildschirm erneut mit F3 ansehen.

Wenn Sie jetzt noch einen winzigen Punkt ganz nach unten links in die Ecke setzen wollen, geben Sie folgende Anweisung:

```
1 0 0 &S!  CR
```

(Falls FORTH Ihnen voller Unverständnis mit der Meldung "&S! ?" antwortet, geben Sie an, daß das Wort ´&S!´ im GRAPHIC-Wortschatz zu finden ist (Kommando GRAPHIC CR), denn sonst sucht FORTH am Ende noch im SOUND-Vokabular nach Graphikwörtern.)

Beachten Sie auch, daß im hochauflösenden Modus der Ursprung (0,0) links unten liegt, und daß die Koordinaten in x-Richtung nach rechts bis 319, und in y-Richtung nach oben bis 199 wachsen dürfen.

Ein weiteres GRAPHIC-Wort, das Sie sicher oft verwenden werden, lautet &LINE. Es zeichnet eine Linie zwischen zwei angegebenen Punkten bzw. löscht diese, wenn Sie als ersten Parameter eine Null eingeben.

Beispiel: &HI-RES 1 10 10 50 50 &LINE CR

GRAPHIK MIT DEM GRAPHIC-VOKABULAR

Ihre aktuelle Graphik können Sie mit Kommando &SAVE auf Floppy speichern, z.B.:

```
" 1.BILD" 8  &SAVE   CR  .
```

Eine gespeicherte Graphik können Sie mit &LOAD lesen.

Wie Sie mit Hilfe von Cursor- und Funktionstasten Bilder malen können, lernen Sie anhand des folgenden Programmes:

```
SCR # 25
 0 P ( BILD MALEN #1, PS, AMOVE   11.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P
 3 P 1 VARIABLE PS        (PENSTATUS -1/0/1    )
 4 P                      (PEN ABSOLUT MOVEN   )
 5 P                      (KOORDIN. AUS XO,YO  )
 6 P : AMOVE         (  -- )
 7 P      GRAPHIC PS @
 8 P      0< 0= IF
 9 P               PS @
10 P               (XO) @
11 P               (YO) @
12 P               &S!
13 P            ENDIF
14 P ;  -->
15 P

SCR # 26
 0 P ( BILD MALEN #2, RMOVE       11.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P                      (PEN UM DX,DY BEWEGEN)
 3 P : RMOVE         ( DX,DY -- )
 4 P       GRAPHIC
 5 P       2 ?ENOUGH
 6 P       (YO) +! (XO) +!
 7 P       AMOVE
 8 P ;
 9 P    -->
10 P
11 P
12 P
13 P
14 P
15 P
```

```
SCR # 27
 0 P ( BILD MALEN #3, CURSOR MOVEMENTS    11.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P
 3 P
 4 P : CRIGHT      1  0  RMOVE ;
 5 P : CLEFT      -1  0  RMOVE ;
 6 P : CUP         0  1  RMOVE ;
 7 P : CDOWN       0 -1  RMOVE ;
 8 P
 9 P     -->
10 P
11 P     MAN KÖNNTE HIER AUCH DIAGONALE
12 P     BEWEGUNGEN DEFINIEREN, Z.B.
13 P : UPRIGHT     1  1  RMOVE ;
14 P     USW.
15 P

SCR # 28
 0 P ( BILD MALEN #4, CHG-PS,TOGGLEPIX      11.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P
 3 P                        ( PENSTATUS VERÄNDERN )
 4 P : CHG-PS      ( -- )
 5 P       PS @
 6 P       0=               ( ÄQUIVALENT ZU NOT    )
 7 P       PS !
 8 P ;
 9 P              ( PIXEL LÖSCHEN/SETZEN)
10 P : TOGGLEPIX
11 P       GRAPHIC (XO) @  (YO) @
12 P       2DUP  &S@
13 P       0= -ROT  &S!
14 P ;
15 P     -->

SCR # 29
 0 P ( BILD MALEN #5, PEN-UP      11.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P
 3 P          ( PEN HOCHSTELLEN  ).
 4 P : PEN-UP
 5 P       -1 PS !
 6 P ;
 7 P
 8 P
 9 P
10 P
11 P
12 P
13 P
14 P
15 P
```

Wenn Sie nun die vorstehenden Seiten übersetzt haben (25 LOAD übersetzt alles bis zur Seite 29), können Sie per Kommando ihre ersten Bilder malen. Setzen Sie beispielsweise die Anfangsposition Ihres gedachten Füllers mit 160 (XO) ! 100 (YO) ! CR in die Mitte des Graphikbildschirms, dann gelangen Sie mit CRIGHT um eine Position nach rechts, mit CUP um eine Position nach oben usw.

Je nachdem welchen Wert die Variable PS gerade hat, wird an der neuen Stelle ein Punkt gesetzt oder gelöscht. Hat PS den Wert 0, handelt es sich um einen Radiergummi, bei +1 ist es ein schreibender Füller und bei (-1) liegt der radierende Füller nicht auf dem Papier - es passiert also nichts.

Natürlich ist das Zeichnen mit diesen relativ langen Wörtern mühsam. Wir werden jetzt mit Hilfe der in Kapitel 2 angedeuteten Tastenbelegungsmethode mit Cursor- und Funktionstasten zeichnen.

Dazu müssen wir eine Tabelle (wir nennen sie PTABLE) erzeugen:

```
SCR # 30
 0 P ( BILD MALEN #6, PRETURN,PTABLE 11.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P : PRETURN     (SETZT LO-RES UND STANDARDKEYTABELLE)
 3 P   GRAPHIC
 4 P   &LO-RES FFTABLE TO.NKEY
 5 P ;
 6 P 0 VARIABLE PTABLE    -2 ALLOT
 7 P 29  CAP CRIGHT  (ZUORDUNG CURSOR-RECHTS)
 8 P 157 CAP CLEFT   (ZUORDUNG CURSOR-LINKS )
 9 P 145 CAP CUP
10 P 17  CAP CDOWN
11 P 133 CAP CHG-PS  (CHANGE PS AUF F1)
12 P 134 CAP PEN-UP  (STIFT HOCH      )
13 P 135 CAP TOGGLEPIX
14 P 136 CAP PRETURN
15 P 0 C,        -->    (OBLIG. ABSCHLUSS EINER KEYTABELLE)
```

und diese aktivieren:

```
SCR # 31
 0 P ( BILD MALEN #7 OF 7, PSTART  11.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P : PSTART      (STARTET PAINT)
 3 P   GRAPHIC
 4 P   160 (XO) !
 5 P   100 (YO) !
 6 P   PTABLE TO.NKEY
 7 P   &HI-RES
 8 P ;   ;S
 9 P
10 P
11 P   FERTIG!
12 P
13 P
14 P
15 P
```

Übersetzen Sie (30 LOAD CR), geben Sie das Kommando PSTART CR und zeichnen Sie mit Hilfe der 4 Cursortasten.

ACHTUNG: Wenn Sie versehentlich eine nicht zugeordnete Taste betätigt haben, wirkt die Tastenbelegeung erst wieder nach der Betätigung von CR (s. Kapitel 2).

6.3. BESCHREIBUNG ALLER GRAPHIC-BEFEHLE

&CLEAR ---

Graphikbildschirm löschen

&HI-RES --

Umschalten in den HI-RES-Modus

&INK n ---

Schreibfarbe festlegen

&LINE f x1 y1 x2 y2 ---
Zeichnet/löscht eine Linie zwischen x1,y1 und x2,y2, je nach Zustand des Flags f.

&LO-RES --
Umschalten in den LO-RES-Modus

&LOAD st dev ---

HI-RES-Graphik von Gerät dev (Floppy=8), File st laden

&PAINT addr --

initialisiert bei Adresse a im Graphikbereich ein Byte mit Papier- und Schreibfarbe

&PAPER b ---

Hintergrundfarbe festlegen

&SAVE st dev ---

HI-RES-Graphik auf Gerät dev (Floppy=8), File st laden
Beispiel: " BILD1" 8 &SAV CR .

&S§ x y --- f

Liefert den Wert des Pixels bei x,y.

&S! f x y ---

Einen Punkt bei x,y setzen (f=TRUE) oder löschen (f=FALSE)

(&ADD) x y --- addr

liefert relative Pixeladresse im Graphikbereich für die Koordinaten x und y

(&ECM) ---

Extended-Color-Modus einschalten

(&CADD) x y --- addr

liefert relative Adresse im Farbbereich zu den Koordinaten x und y.

(&MASK) x --- m

liefert Maske zum Bit x eines Bytes im Graphikbereich

(&MCM) ---

Multi-Color-Modus einschalten

(&SBM) ---

Standard Bitmap setzen

(&SCM) ---

Single-Color-Modus einschalten

(&TM) ---

Textmodus einstellen

(DX) --- addr

temporäre Variable

(DY) --- addr

temporäre Variable

(XO) --- addr

temporäre Variable, für x-Koordinate

(YO) --- addr

temporäre Variable, für y-Koordinate

SET-AREA addr1 addr2 ----

Adressbereiche für Graphikmodus im VIC setzen, addr1=Farbbereich, addr2=Pixelbereich.

7. DER FORTH-ASSEMBLER

7.1. ALLGEMEINES

FORTH enthält einen vollständigen ASSEMBLER. Damit ist es möglich, auch besonders zeitkritische Funktionen zu programmieren und diese wie jedes andere FORTH-Wort aufzurufen.

Für Sie ist es nicht erkennbar (höchstens an der Geschwindigkeit, mit der die betreffende Funktion abläuft), ob ein Wort in FORTH oder in ASSEMBLER geschrieben ist. Auch für den Assembler gilt, daß dieser nicht gesondert geladen werden muß - er ist immer vorhanden.

Der FORTH-ASSEMBLER enthält selbstverständlich alle dem Prozessor 6502 bekannten Befehle. Darüberhinaus ermöglicht er sogar eine strukturierte Programmierung mit Hilfe der Wörter BEGIN, AGAIN, UNTIL, IF, ELSE und ENDIF. Die Wörter, die die Bedingungen für die entsprechenden Abfragen liefern, heißen VS, >=, 0<, 0=, CS und NOT und prüfen die jeweiligen Bits des Statusbytes.

Es existieren allerdings einige Abweichungen zur gewohnten Assemblerprogrammierung. Beispielsweise werden alle Mnemonics mit einem Komma abgeschlossen, um Namenskonflikte mit anderen FORTH-Wörtern zu vermeiden. Weiterhin bleibt auch im Assembler die FORTH-eigene UPN-Notation bestehen, d.h. Sie geben zuerst den Operanden und dann den Operator ein. Zwischen beiden darf noch ein die Operation modifizierendes Glied (zur unmittelbaren, indizierten oder indirekten Adressierung) stehen.

Einfache Befehle sind z.B.:

```
4 # LDA,                          LDA #4
XSAVE STX,        statt           STX XSAVE
50 ) JMP,                         JMP (0050)
```

Wenn Assemblerprogramme als FORTH-Wörter aufgerufen werden sollen, müssen noch einige weitere Besonderheiten beachtet werden.

- FORTH verwendet das X-Register als Stackpointer. Wenn Sie das X-Register anderweitig verwenden möchten, müssen Sie den Wert dieses Registers vorher speichern. Dafür gibt es die Variable XSAVE. Natürlich muß er am Ende des Assemblerprogramms wieder zurückgestellt werden.

- Da in FORTH der Stack als Schnittstelle zur Übergabe von Parametern dient, gibt es für deren Verarbeitung kleine Hilfen, die Wörter BOT und SEC für die Übernahme der Parameter, und die Aussprunglabels PUSH, PUSHOA, PUT, PUTOA und NEXT.

Die Anweisungen BOT LDA bzw. BOT 1+ LDA liefern die obersten beiden Bytes des Stacks. Analog erhalten Sie das zweite Element des Stacks, indem Sie statt mit BOT mit SEC indizieren.

- Assemblerprogramme beginnen mit dem Wort CODE und enden mit dem Wort END-CODE. Durch CODE wird automatisch das ASSEMBLER-Vokabular aktiviert. Die Klammerung CODE ... END-CODE entspricht der FORTH Klammerung : ... ; .

- Assemblerprogramme können auch mit dem Wort LABEL beginnen. Sie sind dann aber nicht direkt von FORTH aus ansprechbar, sondern können nur als Unterprogramme (Ansprungstellen) von anderen Assemblerprogrammen verwendet werden.

- Schließlich gibt es noch feste Rücksprunglabels zu FORTH, von denen eines am Ende über den Befehl JMP, angesprungen werden muß. Wenn der Stack nicht verändert werden soll, verwendet man die symbolische Adresse NEXT, ansonsten je nach Bedarf POP, PUSH, PUSHOA oder PUT.

7.2. DAS ASSEMBLER-VOKABULAR

In dieser Aufzählung werden die eigentlichen Assemblermnemonics nicht beschrieben. Ihre genaue Bedeutung entnehmen Sie bitte einem Buch zur Assemblerprogrammierung des 6502. In FORTH existieren alle diese Kommandos, allerdings ist allen jeweils ein Komma anzuhängen, z.B. JMP, INX, ...

Ebenfalls erklärt wurden die strukturierenden Wörter BEGIN, AGAIN, UNTIL, IF, ELSE, und ENDIF. Ihre Bedeutung deckt sich mit den entsprechenden Wörtern im FORTH-Vokabular.

Es bleiben also noch die FORTH-spezifischen Adressmodifikatoren, sowie einige globale Assemblerlabels:

IP	:	Adresse des I(nterpretative) P(ointer)
W	:	Adresse des Code Feld Zeigers
N	:	Adresse eines 8 Byte langen Scratchbereichs
R	:	Zeiger auf den Returnstack
XSAVE	:	Adresse des Rettregisters für X
UP	:	Adresse eines 8 Byte langen Bereichs für Userparameter
.A	:	bezeichnet den Akkumulator Adressmodus
#	:	unmittelbare Adressierung
)	:	indirekte Adressierung
,X ,Y	:	indizierte Adressierung
X))Y	:	indirekt indizierte Adressierung
BOT	:	Adresse des LOW-Bytes einer 16-Bit Größe im ,X-Modus. Das X-Register zeigt auf den aktuellen Datenstack in der Zero-Page

BOT 1+:	Adresse des HIGH-Bytes einer 16-Bit Größe im ,X-Modus. Das X-Register zeigt auf den aktuellen Datenstack in der Zero-Page
SEC	Adresse des LOW-Bytes der zweiten auf dem Datenstack befindlichen Größe.
SEC 1+:	Adresse des HIGH-Bytes der zweiten auf dem Datenstack befindlichen Größe.
PUT	Adresse einer Routine, die das HIGH-Byte des Datenstacks durch den Inhalt des Akkumulators und das LOW-Byte durch das oberste auf dem Maschinenstack(!) befindliche Byte ersetzt; wird bei NEXT fortgesetzt.
PUSH	wie PUT, jedoch wird das Wort auf den Stack gepusht.
SETUP :	Adresse einer Routine, die n 16-Bit Wörter vom Stack in den Hilfsbereich N transferiert (popt). Die Anzahl n wird im Akkumulator erwartet.
BINARY:	Adresse einer Routine, die das oberste Wort des Datenstacks entfernt und anschließend PUT ausführt.
POP	Adresse einer Routine, die ein Wort vom Datenstack entfernt.
POPTWO:	wie POP, nur werden zwei Worte entfernt.
PUSHOA:	wie PUSH, jedoch wird das HIGH-Byte nicht vom Maschinenstack genommen, sondern 0 gesetzt.
NEXT	Adresse des inneren Interpreters. Alle Routinen müssen NEXT am Ende indirekt (z.B. PUSH, PUT ..) oder direkt anspringen: NEXT JMP,.

7.3. EIN KLEINES ASSEMBLER-PROGRAMM

Sie kennen sicher die Möglichkeit, Ausgaben mit Hilfe des Kommandos CMD vom Bildschirm auf ein anderes Gerät umzuleiten:

```
OPEN 4,4
CMD 4
```

Eine Assemblerroutine in FORTH, die solches bewirkt, könnte etwa folgendermaßen aussehen:

```
SCR # 40
 0 P ( BILDSCHIRMAUSGABE AUF DRUCKER : CMD  11.84/WP)
 1 P FORTH DEFINITIONS HEX
 2 P CODE CMD    ( BEGINN EINES ASSEMBLERPROGRAMMS)
 3 P    XSAVE STX,    ( STACKPOINTER RETTEN  )
 4 P    4 # LDA, TAX, 0 # LDY,
 5 P    FFBA  JSR,          ( SETPAR    )
 6 P    0 # LDA, FFBD JSR, FFC0 JSR,
 7 P    4 # LDX, FFC9 JSR,  ( CKOUT    )
 8 P    XSAVE LDX,
 9 P    NEXT JMP,
10 P END-CODE
11 P
12 P CODE RESET  ( BS RÜCKSETZEN)
13 P    XSAVE STX, FFCC JSR, XSAVE LDX,
14 P NEXT JMP,
15 P END-CODE DECIMAL
```

Sie können Ihre Ausgaben über das Kommando CMD auf den Drucker umleiten. Mit dem Kommando RESET setzen Sie wieder alles zurück.

Eine Routine, die Ihre Seiten zuerst auf dem Bildschirm und anschließend auf dem Drucker ausgibt, sieht so aus:

```
SCR # 41
 0 P ( PRINT-ROUTINE: PRINT , ANALOG ZU LIST  11.84/WP)
 1 P FORTH DEFINITIONS DECIMAL
 2 P : PRINT     ( SCR -- )
 3 P    RESET
 4 P    DUP LIST   ( ERST AUF BS )
 5 P    CMD LIST   ( DANN DRUCKER)
 6 P    RESET
 7 P ;
 8 P    ;S
 9 P
10 P
11 P
12 P
13 P
14 P
15 P
```

Wenn Sie das Wort PRINT geladen haben, können Sie die oben angegebene FORTH-Seite mit dem Kommando 41 PRINT CR ausdrucken lassen.

8. SONSTIGES

An dieser Stelle finden Sie noch einige Hinweise zur Programmierung in FORTH:

- Schreiben Sie kurze Wörter!

- Benutzen Sie die Wörter ´!CSP´ und ´?CSP´.

- Fragen Sie am Anfang eines Wortes die Anzahl der Mindestelemente ab, die auf dem Stack vorhanden sein müssen (Wort ?ENOUGH). Die meisten Systemabstürze entstehen durch Über- oder Unterlaufen des Stacks.

- Aktualisieren Sie die BOOT-Parameter, wenn Sie Ihre neuen Wörter getestet und für gut befunden haben. Sollte sich Ihr System später einmal verabschieden, können Sie sich Ihr geprüftes System mit RUN/STOP + RESTORE und etwas Glück retten.
  BOOT-Parameter werden so gesetzt:

```
HERE FENCE !
HERE 28 +ORIGIN !        ( FENCE  )
HERE 30 +ORIGIN !        ( DP     )
LATEST 12 +ORIGIN !      ( TOP NFA)
```

  Wenn Sie ein neues Vokabular eingerichtet haben, müssen Sie auch den entsprechenden Verweis aktualisieren:

```
´ vokabularname  6 + 32 +ORIGIN ! ´
```

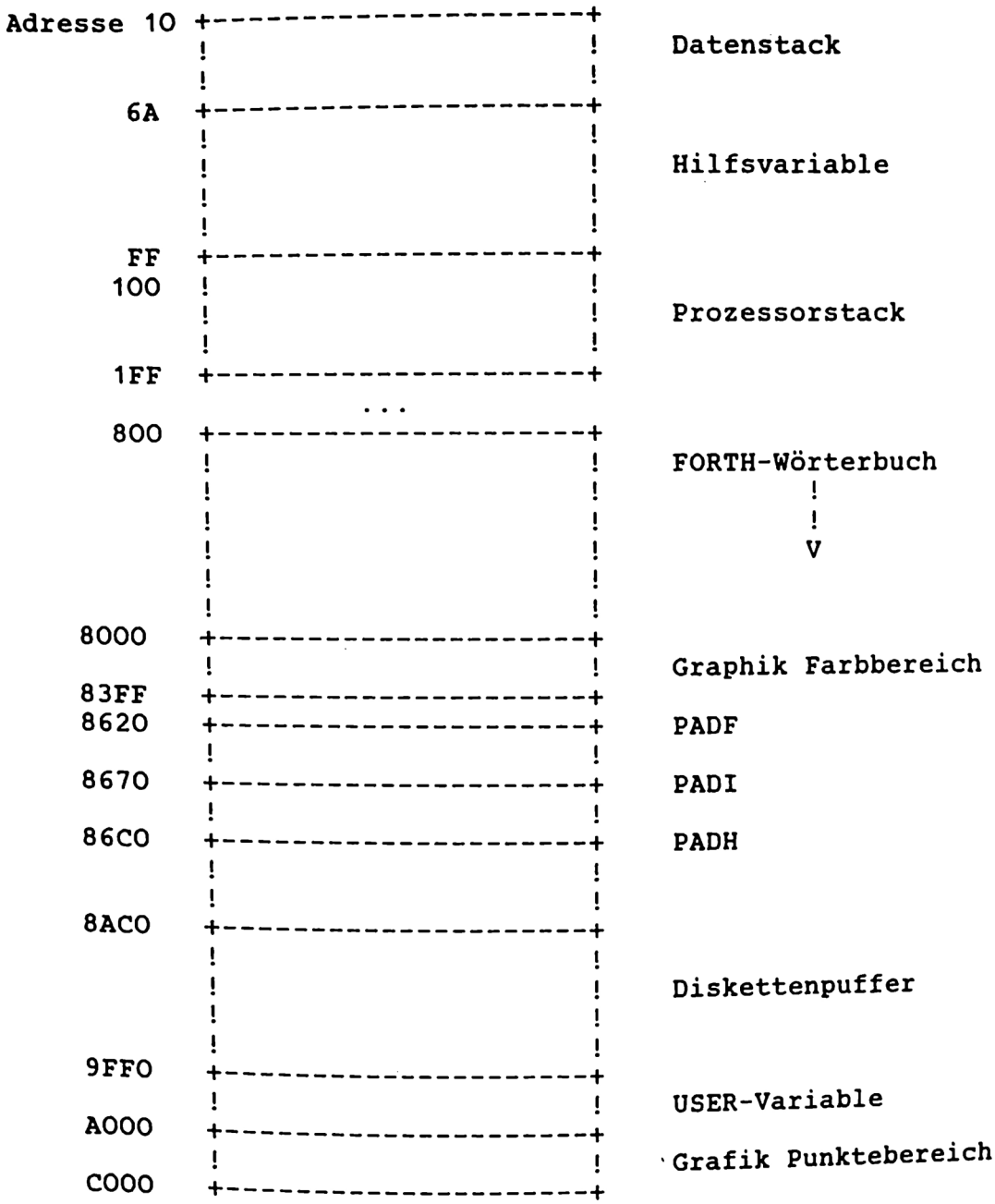
- Die Fehlermeldung NO CHANNEL vom Floppylaufwerk kann u.a. durch das Kommando CLOSE-SCREEN behoben werden. Wenn dies nicht funktioniert, müssen belegte Kanäle mit selektivem CLOSE freigegeben werden.

- FORTH-Wörter dürfen keinesfalls Zeichen mit einem ASCII-Code >127 enthalten, also insbesondere keine Großbuchstaben im Darstellungsmodus 2 des C64 (Klein- / Großbuchstaben). FORTH verschwindet sonst im Nichts!

  Hingegen dürfen Texte, z.B. ´." das ist Text, das geht!"´ solche Zeichen enthalten.

ANHANG 1

SPEICHERBEREICHSVERTEILUNG



Alle Betriebssystemroutinen sind uneingeschränkt verfügbar, sofern sie nicht auf den Datenstack in der Zero-Page zugreifen.

ANHANG 2

DAS FORTH-VOKABULAR

Im folgenden werden nur die Wörter des Wörterbuchs 'FORTH' erklärt. Die Wörter der anderen Vokabulare EDITOR, SOUND, GRAPHIC und ASSEMBLER wurden bereits in eigenen Kapiteln beschrieben.

BESCHREIBUNGSSYMBOLE

addr	Speicheradresse
b	8 bit Byte (linkes Byte=0)
c	7 bit ASCII Character (rechtsbündig, links 0)
d	32 bit (doppeltgenaue) ganze Zahl
f	Logische Kennung 0=falsch, sonst wahr
ff	Logische Kennung 'falsch' (=0)
n	Ganze Zahl (16 bit)
tf	Logische Kennung 'wahr' (#0)
u	16 bit Betragszahl
st	Stringvariable (entspricht n,b für Adresse und Länge eines Strings)

An manchen Stellen, erscheinen statt der oben angegebenen Symbole auch Zahlen im Stackdiagramm. Es handelt sich dann immer um Konstanten, die synonym zu Systemadressen des C64 zu verwenden sind. Sie sind in hexadezimaler Form angegeben.

Große Buchstaben auf der rechten Seite weisen auf bestimmte Eigenschaften der jeweiligen Definition hin:

P	Precedencebit gesetzt (Definition ist 'Immediate')
U	User-Variable

ANHANG 2

`!` n addr ---
Speichert ein Wort n bei addr. (sprich 'store')

`!CSP` Stack-Zeiger von CSP retten. (korrespondiert mit ?CSP)

`"` --- st

legt den folgenden, in Anführungszeichen stehenden Text im Hilfspuffer PAD als Stringkonstante ab

`#` d1 --- d2

erzeugt von einer doppeltlangen Integer das nächste Zeichen für einen Ausgabestring. Das Ergebnis d2 ist der Quotient nach der Division durch BASE. Wird zwischen <# und #> verwendet. (s. auch #S).

`#>` d --- addr count

beendet numerische die Konversion für die Ausgabe. Das Ergebnis auf dem Stack ist die Adresse und Länge (in Bytes) des auszugebenden Strings. Kann direkt von TYPE übernommen werden.

`#S` d1 --- d2

erzeugt ASCII-Text im Text-Ausgabepuffer durch Anwendung von #, bis das Doppelwort d2 den Wert Null erreicht. Wird zwischen <# und #> verwendet.

`$!` st1 st2 --

abspeichern eines Strings st1 in die Stringvariable st2

`$+` st1 st2 -- st

Addition zweier Strings; Ergebnis in PAD

`$2C+` st n -- st

verlängert den String st um das Wort n (2 Zeichen)

`$C+` st b -- st

verlängert den String st um das Zeichen b.

`$LEFT` n st -- st

liefert die ersten n Zeichen der Zeichenkette st in PAD

ANHANG 2

$LEN st -- b

liefert die aktuelle Länge eines Strings

$MID n1 n2 st -- st

liefert die Zeichen n1 bis n2 der Zeichenkette st (im PAD)

$MLEN st -- n

liefert die maximale Zeichenlänge der Stringvariablen st

$RIGHT n st -- st

liefert alle Zeichen ab dem n-ten der Zeichenkette st (im PAD)

$STR d -- st

konvertiert d in einen String (abgelegt in PAD)

$VAL st -- d

konvertiert String in ganze D-Zahl

$VARIABLE b --

Definitionswort, wird in der Form b $VARIABLE cccc verwendet. Es wird eine Stringvariable mit der Maximallänge b definiert. Bei Aufruf von cccc werden Adresse und Länge des aktuellen Strings auf dem Stack hinterlegt.

' --- pfa p

wird verwendet in der Form: ' nnnn

Die Parameterfeldadresse des Wortes nnnn wird auf dem Stack abgelegt. In einer ':'-Definition wird die Adresse als Literal kompiliert.

(p

wird in der Form:

(cccc)

zum Schreiben eines Kommentars benutzt, der in derselben Zeile mit) abgeschlossen wird. Hinter (muß ein Leerzeichen stehen.

ANHANG 2

(.")

die run-time-Prozedur, die den folgenden in-line-Text an das Ausgabegerät ausgibt, wird von ." aufgerufen.

(+LOOP) n ---

die run-time-Prozedur, die durch die +LOOP Anweisung kompiliert wird, erhöht den Schleifenzähler um n und prüft die Aussprungbedingung. (s. +LOOP)

(ABORT)

wird nach einer Fehlermeldung ausgeführt, falls WARNING den Wert -1 enthält.

(DO)

die run-time-Prozedur, die durch die DO-Anweisung kompiliert wurde, überträgt den Schleifenzähler auf den Return-Stack (s. DO)

(FIND) addr1 addr2 --- pfa b tf ;falls gefunden
addr1 addr2 --- ff ;falls nicht gefunden

Das Wörterbuch wird ab der Namensfeldadresse addr2 abgesucht und mit dem String ab addr1 verglichen. Wenn der String gefunden wurde, werden die Parameterfeldadresse, das Längenbyte des Namenfeldes, und die Kennung 'wahr' auf dem Stack abgelegt, sonst wird nur die Kennung 'falsch' abgelegt. Das Wort wird von -FIND aufgerufen, und wird normalerweise nur vom System verwendet.

(KEY) --- c

Laufzeitroutine von KEY. Liefert den Code der nächsten betätigten Taste (ohne auf CR zu warten).

(LINE) n1 n2 --- addr count

liefert Länge und interne Pufferadresse zur Zeile n1 auf Screen n2

(LOOP) n ---

die run-time-Prozedur, die durch die LOOP-Anweisung kompiliert wurde, erhöht den Schleifenzähler um 1 und prüft die Aussprungbedingung (Systemroutine)

ANHANG 2

(NUMBER) d1 addr1 --- d2 addr2

konvertiert den ASCII-Text ab der Adresse addr1+1 in die aktuelle Basis. Der neue Wert wird in d1 aufgebaut und in d2 abgelegt. addr2 ist die Adresse des ersten nicht konvertierten Zeichens. Wird von NUMBER verwendet (Systemroutine).

(SETFPA) chann dev --

dient als LABEL für ASSEMBLER-Routinen, um Fileparameter zu setzen (Systemroutine)

(SETFNA) st ---

dient als LABEL für ASSEMBLER-Routinen, um dem Betriebssystem einen neuen Filenamen mitzuteilen

* n1 n2 --- prod

das Ergebnis ist das Produkt der beiden Eingabewerte

*/ n1 n2 n3 --- n4

n4=n1*n2/n3. Das 31 Bit Zwischenergebnis n1*n2 wird durch n3 geteilt. Die Genauigkeit ist also größer als durch die Folge n1 n2 * n3 /

*/MOD n1 n2 n3 --- n4 n5

n5 ist der Quotient, n4 der Rest der Operation n1*n2/n3. Ein 31 bit Zwischenprodukt gewährleistet eine hohe Genauigkeit.

\+ n1 n2 --- sum

n1 und n2 werden addiert.

+! n addr ---

n wird zu dem Wert in addr addiert. Das Ergebnis wird in addr gespeichert. (Sprich 'Plus Store')

+- n1 n2 -- n3

n3=SGN(n2)*ABS(n1)

+BUF addr1 --- addr2 f

liefert aus der (Disketten-)Pufferadresse addr1 die darauffolgende Pufferadresse addr2. Die Kennung f hat den Wert 'falsch', wenn addr2 der Puffer ist, auf den die Variable PREV verweist.

+LOOP n1 ---

wird in einer ':'-Definition benutzt in der Form:

DO ... n1 +LOOP

Der Schleifenzähler wird um n1 erhöht bzw. verringert. Die Schleife wird verlassen, wenn der Schleifenzähler den Endwert erreicht oder überschritten hat. Bei negativem n1 erfolgt der Aussprung aus der Schleife, wenn der Schleifenzähler erreicht oder unterschritten wurde.

+ORIGIN n --- addr

übergibt die Adresse ORIGIN+n. Bei ORIGIN beginnen die Parameter für den FORTH-Bootstrap.

, n ---

speichert n in das nächste Wort des Wörterbuchs. Der Dictionary-Pointer wird erhöht.

'KEY --- n U

eine USER-Variable. Sie soll die Adresse der Routine (KEY) enthalten.

- n1 n2 .--- diff

diff=n1-n2

--> P

Interpretation der Eingabe von Floppy wird mit dem nächsten Screen fortgesetzt.

-DUP n1 --- n1 ;falls n1=0
n1 --- n1 n1 ;falls n1#0

n1 wird dupliziert, wenn sie ungleich 0 ist. Die übliche Anwendung ist ein -DUP direkt vor einem IF, damit keine DROP-Anweisung im ELSE-Teil notwendig wird.

-ROT n1 n2 n3 --- n3 n1 n2

läßt die drei obersten Elemente des Stacks in die ROT entgegengesetzte Richtung rotieren.

-FIND --- pfa b tf ;falls gefunden p
--- ff ;falls nicht gefunden

der nächste durch Blanks begrenzte String der Eingabe wird nach HERE übertragen. Das CONTEXT-Vokabular und, falls nicht gefunden, das CURRENT-Vokabular werden nach einem identischen Namensheader abgesucht. Falls gefunden, werden die Parameterfeld-Adresse, das count-Byte und die Kennung 'wahr' auf dem Stack abgelegt, ansonsten lediglich die Kennung 'falsch'.

-TRAILING addr n1 --- addr n2

die Zeichenanzahl eines Textstrings der Länge n1 wird um die Anzahl der am Schluß befindlicher Leerzeichen vermindert.

n ---

gibt den Wert n in der Konversionsbasis BASE aus. Nach der Ausgabe wird ein BLANK ausgegeben (sprich 'Dot')

." p

wird benutzt in der Form: ." cccc"

Kompiliert die folgende Zeichenkette bis zum begrenzenden ". Zur Laufzeit wird dieser String dann ausgegeben.

.LINE line scr ---

die Zeile line des Screens scr wird auf den Terminal ausgegeben. Blanks am Textende werden unterdrückt.

.R n1 n2 ---

gibt die Nummer n1 rechtsbündig in ein Feld der Länge n2 aus.

.S ---

schreibt den gesamten Stackinhalt (zerstörungsfrei) auf das Ausgabegerät.

/ n1 n2 --- quot

quot=n1/n2

/MOD n1 n2 --- rem quot

ganzzahlige Division mit Rest. Der Rest hat das Vorzeichen des Dividenden.

0 1 2 3 --- n

diese häufig verwendeten kleinen Zahlen werden als Konstanten im Wörterbuch geführt.

0< n --- f

setzt die Kennung 'wahr', falls n<0, sonst 'falsch'

0= n --- f

setzt die Kennung 'wahr', falls n=0, sonst 'falsch'

1+ n1 --- n2

n1 um 1 erhöhen

10* n1 --- n2

n1 mit 10 multiplizieren (schnelle Assembler-Multiplikation)

2! d addr ---

Doppelwort ab Adresse addr speichern

2+ n1 --- n2

n1 um 2 erhöhen

2@ addr --- d

Doppelwort von Adresse addr auf den Stack kopieren

2CONSTANT d --

Definitionswort; wird in der Form d 2CONSTANT cccc definiert. Wenn cccc aufgerufen wird, wird die doppeltgenaue Konstante d auf den Stack gebracht.

2VARIABLE d --

Definitionswort; wird in der Form d 2VARIABLE cccc definiert. Wenn cccc aufgerufen wird, wird die Adresse der Variablen, die das Doppelwort enthält, auf dem Stack abgelegt.

2DROP d ---

zwei Wörter vom Stack entfernen

2DUP d --- d d

ein Doppelwort auf den Stack kopieren

2SWAP d1 d2 -- d2 d1

zwei Doppelworte auf dem Stack vertauschen

P

wird in einer ':'-Definition benutzt in der Form:
: cccc ... ;
Definiert ein neues Wort im Wörterbuch, das zu den durch ... dargestellten Anweisungen äquivalent ist. Die Definition wird durch ; beendet. Das CONTEXT-Vokabular wird zum CURRENT-Vokabular erklärt. Worte, deren Precedencebit gesetzt ist, werden ausgeführt statt kompiliert.

; P

die ':'-Definition und auch der Compile-Modus werden beendet.

;S P

wird verwendet, um die Interpretation eines Screens an beliebiger Stelle zu beenden.

< n1 n2 --- f

setzt die Kennung 'wahr', falls n1 < n2, sonst falsch

<# d1 --- d2

Anfangsanweisung einer formatierten numerischen Ausgabe bis zur Verwendung der Anweisungen:
<# # #S SIGN #>
Das Doppelwort auf dem Stack wird konvertiert, der erzeugte Text wird im PAD abgelegt.

<BUILDS C

wird in einer ':'-Definition in der Form verwendet:
: cccc <BUILDS ... DOES> ... ;
Wenn cccc aufgerufen wird, wird ein neues High-Level-Wort definiert. Der Aufruf in der Form:
cccc nnnn
bringt die Anweisungen hinter <BUILDS zur Ausführung. Ein neues Wort mit dem Namen nnnn wird definiert. Durch die <BUILDS .... DOES> Konstruktion können run-time-Prozeduren in High-Level geschrieben werden.

<SHIFT n b --

Linksshift des Wortes n um b Bytes. Kann evtl. zur schnellen Multiplikation mit 2 verwendet werden.

= n1 n2 --- f

setzt die Kennung 'wahr', falls n1 = n2, sonst 'falsch'

> n1 n2 --- f

setzt die Kennung 'wahr', falls n1 > n2, sonst 'falsch'

>CMOVE n1 n2 n3

Aufwärtsschieben von n3 Bytes ab Adresse n1 nach Adresse n2. Bereiche dürfen sich überlappen, im Gegensatz zu CMOVE.

>R n ---

das oberste Element des Datenstacks (TOS) Stack wird vom Stack geholt und auf den Return-Stack übertragen. Eine R>-Anweisung sollte in derselben Definition vorhanden sein.

>SHIFT n b --

Rechtsshift des Wortes n um b Bytes.

? addr ---

gibt den Wert in addr in der aktuellen Basis formatfrei aus

?COMP

gibt eine Fehlermeldung aus, falls sich das System nicht im Compile-Modus befindet.

?CSP

gibt eine Fehlermeldung aus, falls der Stack-Zeiger nicht mit dem in CSP gespeicherten Wert übereinstimmt (s. !CSP).

?DISC

Zustand des Diskettenlaufwerks abfragen und auf den Bildschirm ausgeben. Wird bei blinkendem Kontrollämpchen am Floppylaufwerk aufgerufen.

?ENOUGH n ---

es wird geprüft, ob mindestens n Parameter auf dem Stack sind. Soll das System vor dem Absturz bewahren, wenn Wörter mit zu wenig Parametern aufgerufen werden.

?ERROR f n ---

gibt die Fehlermeldung n aus, falls die Kennung f 'wahr' ist. Es wird Zeile n von SCR#4 ausgegeben. n darf auch größer als 15 sein, dann werden Zeilen der folgenden Screens geschrieben.

?EXEC

gibt eine Fehlermeldung aus, falls sich das System nicht im Execute-Modus befindet.

?LOADING

gibt eine Fehlermeldung aus, falls momentan nicht geladen wird.

?PAIRS n1 n2 ---

gibt eine Fehlermeldung aus, falls n1 # n2. Die Anweisung prüft, ob zwei strukturierte Sprachelemente zusammengehören oder nicht.

?STACK

gibt eine Fehlermeldung aus, falls ein Stacküberlauf erfolgt.

?TERMINAL --- f

es wird getestet, ob die RUN/STOP-Taste betätigt wurde. Die Kennung 'falsch' bedeutet, daß keine Betätigung erfolgte.

@ addr --- n

der Inhalt von Adresse addr wird auf dem Stack abgelegt.

ABORT

initialisiert die beiden Stacks und setzt den Execute-Modus. Der Rechner arbeitet mit Standard Ein- und Ausgabe weiter und gibt als erstes die Anfangsmeldung aus.

ABS n --- u

es wird der Absolutbetrag von n gebildet.

AGAIN P

wird in einer ':'-Definition in der Form benutzt:
BEGIN ... AGAIN
Bewirkt den Rücksprung zum entsprechenden BEGIN. Der Stack bleibt unverändert. Die Schleife kann nur durch die Folge R> DROP verlassen werden.

ALLOT n ---

die ganze Zahl n wird zum Dictionary-Pointer DP addiert. Diese Anweisung wird benutzt, um Platz im Wörterbuch zu reservieren, oder Speicherplatz zu verlagern (re-origin).

AND n1 n2 --- n3

Bitweise logisches UND

ASCII --- n

liefert den ASCII-Wert des 1. Zeichens des folgenden Wortes.
Beispiel: ASCII A legt 65 auf dem Stack ab.

B/BUF --- n

die Anzahl der Bytes pro Diskettenpuffer (=64) wird auf dem Stack abgelegt.

B/SCR --- n

die Anzahl der Diskettenpuffer (=16) pro Screen wird auf dem Stack abgelegt. Vereinbarungsgemäß entspricht ein Screen 1024 Bytes. Er wird in 16 Zeilen zu je 64 Zeichen unterteilt.

BACK addr ---

der Offset von HERE zu addr für einen Rückwaertssprung wird errechnet und in den nächsten freien Platz des Wörterbuchs kompiliert.

BASE --- addr U

User-Variable, die die aktuelle Konversionsbasis für Input/Output enthält

BEGIN --- P

wird in einer ':'-Definition in der Form benutzt:

```
BEGIN   ...   UNTIL
BEGIN   ...   AGAIN
BEGIN   ...   WHILE  ...  REPEAT
```

BEGIN markiert den Anfang einer Befehlsreihe, die wiederholt werden kann. Dieser Punkt dient als Rücksprungmarke für das korrespondierende UNTIL, AGAIN, oder REPEAT.

BL --- c

das ASCII-Zeichen BLANK wird auf dem Stack abgelegt.

BLACK --- 0

liefert Konstante 0 als Farbwert für schwarz.

BLANKS addr count ---

besetzt ab addr, count Bytes mit Leerzeichen vor.

BLK --- addr

User-Variable. Enthält die Blocknummer des Blocks, der gerade interpretiert wird. Null bedeutet Input von Terminal.

BLOAD vaddr st n2 n3 ---

binäres Laden eines Files von Diskette.
vaddr ist die Anfangsadresse, ab der geladen werden soll.
st bezeichnet den Filenamen.
n2 und n3 bezeichnen Kanal (0 oder 1) und Gerätenummer (=8) des zu ladenden Files. Achtung: Bei Verwendung von Kanal 1 wird der File ab der Originaladresse abgelegt. Die Angabe von vaddr ist dann irrelevant, darf aber nicht ausgelassen werden.

BLOCK n --- addr

legt die Speicheradresse addr des Blockpuffers, der den Block n enthält, auf dem Stack ab. Wenn der Block momentan nicht im Speicher liegt, wird er in den 'ältesten' Puffer geschrieben. Falls der Inhalt dieses Puffers als 'UPDATE' markiert ist, wird sein Inhalt vorher auf Diskette austransferiert (s. BUFFER, R/W, UPDATE FLUSH)

BLUE --- 6

liefert Konstante 6 als Farbwert für blau.

BMOVE addr ---

verschiebt einen string von PAD nach n.

BORDER b ---

färbt den Rand des Bildschirms mit Farbe b ein.

BROWN --- 9

liefert Konstante 9 als Farbwert für braun.

BSAVE addr1 addr2 st n2 n3 ---

binaeres Speichern eines Files auf Diskette.
addr1 und addr2 bezeichnen den Bereich (von-bis), der geschrieben werden soll.
st bezeichnet den Filenamen.
n2 und n3 bezeichnen Kanal (0 oder 1) und Gerätenummer (=8) des zu speichernden Files.

BUFFER n --- addr

ermittelt den nächsten Blockdiskettenpuffer und weist ihm den Block n zu. Wenn der aktuelle Inhalt des Puffer als 'updated' markiert ist, wird er vor dem Einlesen des nächsten Blocks auf Diskette geschrieben. Der Block wird nicht von der Diskette gelesen. Die abgelegte Adresse ist die Anfangsadresse der einzulesenden Daten.

C! b addr ---

8 Bits in addr speichern.

C, b ---

speichert das Byte b in den nächsten freien Platz des Wörterbuchs. Der Dictionary-Pointer wird um 1 erhöht.

CAP b ---

schreibt einen drei Bytes langen Eintrag der Form b (1 Byte) n (das ist die CFA des folgenden Wortes) ins Wörterbuch. Wird zur Tastenzuordnung benötigt, z.B. 133 CAP ?DISC .

CASE n -- n

CASE und END-CASE bilden zusammmen mit OF und ENDOF die Möglichkeit, in FORTH-Programmen eine gegliederte Fallunterscheidung durchzuführen. Diese Wörter werden wie in folgendem Beispiel verwendet:

```
CASE
n1   OF Anweisungen für Alternative 1 ENDOF
n2   OF Anweisungen für Alternative 2 ENDOF
...
nk   OF Anweisungen für Alternative k ENDOF
ENDCASE
```

Dabei wird die Alternative i dann ausgeführt, wenn ni mit dem Wert übereinstimmt, der vor CASE auf den Stack gebracht wurde. n wird durch END-CASE vom Stack entfernt.

C/L --- n

liefert die Anzahl Zeichen pro Zeile (=64).

C@ addr --- b

8 bits von addr auf dem Stack ablegen (s. @).

CFA pfa --- cfa

konvertiert die Parameter-Feldadresse in ihre Code-Feldadresse

CIA1 --- DC00

liefert Adresse des Bausteins CIA1 des C64.

CIA2 --- DD00

liefert Adresse des Bausteins CIA2 des C64.

CLEAR ---

löscht den Stack.

CLIT --- b

wie LIT, nur Bearbeitung eines Bytes statt Wortes. CLIT läuft schneller als LIT

CLOSE n1 n2 ---

das mit OPEN über Kanal n1 angesprochene Gerät n2 wird geschlossen

CLOSE-SCREEN ---

schließt File SCR (s. OPEN-SCREEN)

CMOVE addr1 addr2 n ----

Übertraegt n Bytes von Adresse addr1 nach Adresse addr2. Die Übertragung erfolgt von der niedrigsten Adresse an aufwärts, d.h. zuerst Adresse addr1, dann Adresse addr1+1, usw.

COLD

Die Kaltstart-Prozedur, um das System zu initialisieren. COLD kann vom Terminal aktiviert werden, um Anwenderprogramme zu löschen und das System zu reinitialisieren.

COLORAREA --- n

Variable; sie enthält den Beginn des Farbbereichs für Graphik

COMBINE b1 b2 -- n

setzt aus den Bytes b1(low) und b2 ein Wort n zusammen.

COMPILE

kompiliert die CFA (Codefeldadresse) des folgenden Wortes an den nächsten freien Platz im Wörterbuch.

CONSTANT n ---

dient zur Definition von Konstanten. Wird in der Form

n CONSTANT cccc

verwendet, um ein Wort cccc mit dem Wert n zu definieren. cccc ist dann aequivalent zu n.

CONTEXT --- addr U

User-Variable. Enthält den Zeiger auf das Vokabular, in dem Suchläufe beginnen.

CONTROL --- n

liefert den CONTROL-Code des folgenden Zeichens. Dieser entspricht dem um 64 verminderten ASCII-Code. Beispiel: CONTROL C legt 3 auf dem Stack ab.

COUNT addr1 --- addr2 n

legt Adresse addr2 und Anzahl Bytes n des Texts, der bei addr1 anfängt, auf dem Stack ab. Der Text muß in der Weise abgelegt sein, daß das erste Byte bei addr1 die Länge des Textes enthält, und daß der Text im zweiten Byte anfängt. In der typischen Anwendung folgt TYPE auf COUNT

CR

bewirkt Zeilenvorschub am Terminal

CREATE

ein Wort, mit dem man neue FORTH-Woerter aufbauen kann, z.B.
CREATE cccc
Es dient dazu, einen Header für das zu definierende Wort cccc abzusetzen. Die Aktionen, die von diesem Wort durchgeführt werden sollen, müssen beispielsweise mit COMPILE oder , in das Wort kompiliert werden. Das Smudge-Bit wird gesetzt. Das Wort ist erst dann von außen ansprechbar, wenn das Smudge-Bit wieder gelöscht wurde (Kommando SMUDGE).

CREATE-SCREEN

richtet ein File SCR in einer Länge von 2200 Sätzen (137 Screens) ein. Eine etwaige Meldung RECORD NOT PRESENT kann an dieser Stelle ignoriert werden. Außerdem werden die Screens 4 und 5 (Fehlermeldungen), die im Diskettenpuffer stehen sollten, kopiert.

CSP --- addr U

User-Variable. Enthält den Stack-Pointer.

CURRENT --- addr

User Variable. Enthält einen indirekten Verweis auf die Namensfeldadresse der Definition, die gerade definiert wird (s. CONTEXT).

CYAN --- 3

liefert Konstante 3 als Farbwert für türkis.

D+ d1 d2 --- dsum

addiert die Doppelworte d1+d2

D+- d1 n --- d2

das Doppelwort d1 erhält das Vorzeichen von n.

D. d ---

gibt eine Doppelwort auf dem Ausgabegerät aus (vorzeichengerecht) (sprich 'D-Dot')

D.R d n ---

gibt das Doppelwort d als eine Zahl mit Vorzeichen in ein n Zeichen langes Feld aus.

DABS d --- u

ersetzt das Doppelwort d durch seinen Betrag

DECIMAL

setzt die Konversionsbasis für Input/Output auf 10.

DELI --- n

USER-Variable, die den Delimiter (Trennzeichen) bei FIND und INSERT-Routinen des Editors enthält. Normalerweise ist dies das POUND-Zeichen.

DEPTH --- b

liefert die "Tiefe" (Anzahl der Elemente) des Stacks.

DEFINITIONS

wird benutzt in der Form:
cccc DEFINITIONS
Setzt das CURRENT-Vokabular auf das CONTEXT-Vokabular. Mit diesem Aufruf wird cccc zum CONTEXT-Vokabular

DIGIT c n1 --- n2 tf ; bei zulässiger Konversion
c n1 --- ff ; bei unzulässiger Konversion

konvertiert den ASCII-Wert von c in seinen Binarwert n2. Die Konversion wird durch eine Kennung ergänzt. Ist die Konversion nicht zulässig, wird nur die Kennung gesetzt.

DLITERAL d --- d ;run-time P

während der Kompilierung wird d als ein Literal kompiliert. Wenn die Definition später durchgeführt wird, wird der Wert auf dem Stack abgelegt. Zur Laufzeit passiert nichts.

DMINUS d1 --- d2

Vorzeichenumkehr.

DO n1 n2 --- ;run-time

wird in einer ':'-Definition in der Form benutzt:
DO ... LOOP
DO ... +LOOP
Zur Laufzeit werden die Schleifengrenze, n1 und der Schleifenindex n2 vom Datenstack entfernt und auf dem Return-Stack abgelegt. Bei jedem Durchlauf wird n2 erhöht (bei LOOP um +1; bei +LOOP um eine beliebige Zahl). Beim Verlassen der Schleife werden Grenze und Index vom Return-Stack entfernt. n1 und n2 werden zur Laufzeit ermittelt. Innerhalb einer Schleife legt die I-Anweisung den aktuellen Wert des Indexes auf dem Stack ab (s. I, LOOP, +LOOP, LEAVE).

DOES>

definiert den run-time Ablauf eines durch <BUILDS ... DOES> definierten Wortes.

DONE ---

schließt einen Editiervorgang ab, schreibt alle veränderten Seiten auf Diskette und sorgt für die Standardbelegung der Funktionstasten.

DOS st ---

schickt einen Diskettenbefehl an Geraet 8, z.B. "N:DISKETTE1,01" DOS. Ihre Diskette im Geraet 8 wird formatiert (Vorsicht!).

DP --- addr U

User-Variable. Enthält die Adresse des Zeigers, der auf den nächsten freien Platzes des Wörterbuches verweist. Mit HERE kann der Wert gelesen, mit ALLOT kann er geändert werden.

DPL U

User-Variable. Enthält die Anzahl der Ziffern, die bei der Eingabe von Doppelwortzahlen rechts vom Dezimalpunkt stehen. Wird auch verwendet, um den Dezimalpunkt für formatierte Ausgabe zu definieren. Der Default-Wert bei Einwortzahlen ist -1.

DROP n ---

n wird vom Stack entfernt

DUMP addr1 addr2 ---

Speicherbereich von addr1 bis addr2 wird gedumpt.

ANHANG 2

DUP n --- n n

dupliziert den obersten Stack-Wert.

E ---

die zuletzt editierte (Nr. steht in SCR) Seite erneut vorlegen

EDIT n ---

Seite n soll zum Editieren vorgelegt werden; dabei wird n in SCR eingetragen. Es erfolgt Umschaltung in den Edit-Modus. Die Funktionstasten werden neu belegt.

ELOAD ---

die zuletzt editierte Seite (Nr. steht in SCR) wird geladen.

ELSE P

wird in einer ':'-Definition in der Form benutzt:
IF ... ELSE ... ENDIF

EMIT c ---

gibt das ASCII-Zeichen c auf das Ausgabegerät aus.

EMPTY-BUFFERS

löscht den gesamten Diskettenpufferbereich

ENCLOSE addr1 c --- addr1 n1 n2 n3

wird von WORD verwendet, um einen Text zu separieren. Der Text beginnt ab addr1. c ist der Delimiter. Die übergebenen Parameter sind:
1) addr1 (wie Eingabe)
2) n1 -- der Byte-Offset zum ersten vom Delimiter verschiedenen Zeichen
3) n2 -- der Byte-Offset zum ersten Delimiter hinter dem Text
4) n3 -- der Byte-Offset zum ersten vom Delimiter verschiedenen Zeichen hinter dem Text-Ende Delimiter.

Ein ASCII-Null im Text wird wie ein unbedingter Delimiter behandelt.

END Synonym für UNTIL P

ENDCASE n -- P

wird als Abschluß eines CASE-Statements verwendet. Holt die zu prüfende Größe vom Stack (s. CASE!)

ENDIF P

wird in einer ':'-Definition in der Form benutzt:
IF ... ENDIF
IF ... ELSE ... ENDIF
ENDIF ist das Ziel eines Vorwärtssprungs von IF oder ELSE.
THEN ist ein Synonym für ENDIF (s. IF, ELSE)

ENDOF P

wird als Abschluß einer Variante in (s.) CASE-Statements verwendet.

ERASE addr n ---

setzt n Hauptspeicherworte ab Adresse addr auf Null

ERROR line --- in blk

gibt Fehlermeldungen aus und bewirkt einen Systemrestart. Die genaue Fehlerbehandlung hängt von der Variablen WARNING ab:
+1: Es wird die Zeile n relativ zu Screen 4 ausgegeben. Die durch n festgelegte Zeile darf außerhalb Screen 4 liegen.
0: Es wird der Wert n als Nummer ausgegeben.
-1: Es wird ABORT aufgerufen.

EXECUTE addr ---

addr ist die Codefeld-Adresse des Wortes, das durch EXECUTE zur Ausführung gelangt. Im Anschluß an die Durchführung eines mit EXECUTE aufgerufenen Wortes wird mit den auf EXECUTE folgenden Worten kontinuierlich fortgefahren.

EXP2 b --- n

schnelle Routine zur Berechnung von 2 hoch b (b ist positiv).

EXPECT addr count ---

überträgt count Zeichen vom Terminal zur Adresse addr. Ein Carriage-Return bricht die Übertragung vorzeitig ab. Am Stringend werden Nullen angehängt.

FENCE --- addr

User-Variable. Enthält die Adresse, ab der die FORGET-Anweisung wirkungslos ist. Wenn FORGET unterhalb dieser Grenze wieder wirksam werden soll, muß der Inhalt von FENCE geändert werden.

FFTABLE --- n

Adresse der Tabelle für die FORTH-Funktionstastenbelegung

FILL addr quan b ---

besetzt quan Bytes ab Adresse addr mit Byte b vor

FIRST --- n

Konstante. Liefert die Adresse des 1. Disketten-Blockpuffers

FLD --- addr U

User-Variable. Enthält die Feldbreite für formatierte numerische Ausgabe.

FTABLE --- n

Verweis auf die aktuelle Tabelle der Funktionstastenbelegung

FLIP n --- n

vertauscht die beiden Bytes des Wortes n.

FLUSH ---

kann am Ende der Editierung verwendet werden, um sicherzustellen, daß alle Textänderungen auf die Diskette übertragen wurden. Besser und systemkonform ist DONE.

FORGET E

wird in der Form benutzt:
FORGET cccc
Die Definition cccc wird mit allen folgenden Definitionen aus dem Wörterbuch gelöscht.

FORTH P

der Name des FORTH-Sprachkernes. Die Anweisung setzt FORTH als CONTEXT-Vokabular.

GTEXT n ---

folgenden Text, der durch den in DELI definierten Delimiter begrenzt ist, ab n abspeichern.

GREEN --- 5

Konstante 5 für den Farbwert grün.

HELP ---

liefert Funktionstastenbelegung

HERE --- addr

die nächste freie Adresse im Wörterbuch wird auf dem Stack abgelegt.

HEX

setzt die Konversionsbasis auf 16 (hexadezimal).

HLD --- addr

User-Variable, die das letzte Zeichen eines Textes einer numerischen Ausgabekonversion enthält.

HOLD c ---

wird zwischen <# und #> benutzt. Überträgt das abgelegte ASCII-Zeichen in den Text einer numerischen Ausgabekonversion.

I --- n

wird innerhalb eines DO ... LOOP verwendet, um den Schleifenzähler auf dem Stack abzulegen.

ID. addr ---

druckt den Namen eines Wortes mit NFA addr aus.

IF f --- ;run-time

wird in einer ':'-Definition in der Form benutzt:

```
IF(tp)  ...  ENDIF
IF(tp)  ...  ELSE(fp)  ...  ENDIF
```

Wenn f wahr ist, werden die Befehle unmittelbar hinter IF durchgeführt; wenn f falsch ist, erfolgt ein Sprung zu ELSE (falls vorhanden) oder ENDIF.

IMMEDIATE

markiert das letzte definierte Wort als 'immediate'. Dadurch wird erreicht, daß das Wort im Compiliermodus ausgeführt statt übersetzt wird. Sie können die Übersetzung einer Immediate-Definition durch eine vorangehende [COMPILE]-Anweisung erzwingen.

IN --- addr

User-Variable. Enthält den Byte-Offset im aktuellen Inputtextpuffer, von dem der nächste Text geholt wird. Die Anweisung WORD verwendet und ändert den Inhalt von IN.

INDEX n1 n2 ---

liefert sozusagen ein Inhaltsverzeichnis der Seiten n1 bis n2. Schreibt die jeweils erste Zeile der genannten Seiten auf den Bildschirm. Zusätzlich wird die Disketteninformationszeile Zeile 15 von Screen 4 mit ausgegeben.

INKCOLOR --- n

Variable, die die Schreibfarbe für Graphik enthält.

INTERPRET

Der äußere Text-Interpreter, der Eingaben (von Terminal oder Disk) ausführt oder kompiliert. Das CONTEXT- und evtl. das CURRENT-Vokabular wird nach dem Namen durchsucht. Wenn der Name nicht gefunden wird, wird der Wert in die aktuelle Basis konvertiert. Tritt ein Konversionsfehler auf, wird der Name zusammen mit einem "?" ausgegeben.

LATEST --- addr

die Namensfeldadresse des zuletzt definierten Wortes im CURRENT-Vokabular wird auf dem Stack abgelegt.

LEAVE

bewirkt das Verlassen einer DO-Schleife beim nächsten LOOP oder +LOOP Befehl.

LFA pfa --- lfa

konvertiert die Parameterfeldadresse einer Definition in ihre Linkfeldadresse.

LIMIT --- n

Konstante; legt die Adresse, die unmittelbar über der letzten Diskettenpufferadresse liegt, auf dem Stack ab.

LIST n ---

Screen n (logische Seite n) wird auf dem Bildschirm gelistet. Die Variable SCR wird nicht verändert!

LIT --- n

LIT wird beim Kompilieren vor jedem 16-Bit Literal abgesetzt. Wenn LIT zur Laufzeit durchgeführt wird, wird der Wert auf dem Stack abgelegt.

LITERAL n --- p

wird im Modus Kompilieren verwendet, um einen errechneten Wert in das Wörterbuch zu übersetzen.
Die vorgesehende Anwendung ist:
: xxx [Berechnung] LITERAL .
Aufheben des Kompilier-Modus, Durchführung einer Berechnung, Kompilieren des errechneten Wertes mittels LITERAL.

LOAD n ---

Screen n wird geladen (interpretiert). Das Ende des Ladens kann mit der Anweisung ;S erzwungen werden.

LOOP addr n --- p

wird in einer ':'-Definition in der Form benutzt:
DO ... LOOP
und markiert den Endpunkt einer Schleife. Der Schleifenzähler wird um 1 erhöht und mit der Schleifengrenze n verglichen. Die Schleife wird verlassen, wenn der Schleifenzähler die Obergrenze n erreicht hat.

M* n1 n2 --- d

das Ergebnis d von n1*n2 ist ein Doppelwort

M/ d n1 --- n2 n3

n2 ist der Rest, n3 der Quotient der Division d/n1. Der Rest n2 hat das Vorzeichen des Dividenden

M/MOD ud1 u2 --- u3 ud4

Betragsoperation; der Dividend ud1 und der Quotient ud4 sind Doppelworte. Der Divisor u2 und Rest u3 und sind 16 bit Worte

MAX n1 n2 --- max

das Ergebnis ist das Maximum von n1 und n2

MESSAGE n ---

Textzeile n relativ zu Screen 4, wird auf dem Ausgabegerät ausgegeben. Die Zeile darf außerhalb Screen 4 liegen. Hat WARNING den Wert Null, so wird lediglich der Wert n ausgegeben.

MIN n1 n2 --- min

das Ergebnis ist das Minimum von n1 und n2

MINUS n1 --- n2

n1 wird negiert.

MOD n1 n2 --- n3

n3 =n1 Modulo n2. n3 hat dasselbe Vorzeichen wie n1.

NFA pfa --- nfa

konvertiert die Parameterfeldadresse einer Definition in ihre Namensfeldadresse.

NKEY --- n

alternative KEY-Funktion, die die Funktionstabelle abfragt und nur dann einen ASCII-Wert auf dem Stack liefert, wenn die betätigte Taste nicht in der Funktionstabelle enthalten ist.

NOK ---

schaltet FORTH´s OK-Meldung aus.

NOP ---

Routine, die nichts ausführt (Dummyroutine)

NOTABLE --- addr

Variable, die den Wert 0 enthalten sollte. Ihre Adresse wird in der Variablen FTABLE gespeichert, wenn keine Funktionstasten belegt werden sollen.

NUMBER addr --- d

die Zeichenkette, die in Adresse addr anfängt und deren Länge unmittelbar vor der Kette steht, wird gemäß der aktuellen Konversionsbasis in ein Doppelwort konvertiert. Die Stellung eines gegebenenfalls vorhandenen Gleitpunkts wird in DPL gespeichert. Sonst hat der Gleitpunkt keine Wirkung. Eine Fehlermeldung wird bei einem Konversionsfehler ausgegeben.

OK ---

schaltet FORTH's OK-Meldung ein

OPEN st n2 n3 ---

es wird ein File mit dem Filenamen st auf Gerät n2 über Kanal n3 geöffnet

OPEN-SCREEN ---

eröffnet File SCR (enthält Screens). Braucht von Ihnen normalerweise nicht gegeben zu werden, da FORTH dieses File selbst verwaltet.

OR n1 n2 --- or

das Ergebnis ist die logische OR-Verknüpfung von n1 und n2

ORANGE --- 8

liefert die Konstante 8 als Farbwert für orange.

OUT --- addr U

User-Variable; enthält einen Wert, der von EMIT verwendet und geändert wird. Sie können den Wert abfragen oder ändern, um eine formatierte Ausgabe zu erstellen.

OVER n1 n2 --- n1 n2 n1

das zweite Wort des Stacks wird auf den Stack kopiert

PAD --- addr

liefert die Adresse des Text-Ausgabepuffers (Lage variiert)

PAPERCOLOR --- n

Variable, die den aktuellen Farbwert des Papiers für Graphik enthält.

PEN b ---

setzt die (Text-) Schreibfarbe auf b.

PENCOLOR --- n

Variable, die die aktuelle Schreibfarbe enthält

PFA nfa --- pfa

konvertiert eine Namensfeldadresse einer Definition in ihre Parameterfeldadresse.

PICK b --- n

kopiert das Element b des Stacks auf den Stack, z.B. 1 PICK entspricht DUP, 2 PICK entspricht OVER etc.

PIXELAREA --- n

Variable, die den Beginn des Graphikbereichs enthält

PREV --- addr

Variable, die die Adresse des zuletzt angesprochenen Diskettenblock-Puffers enthält. Die UPDATE-Anweisung bezieht sich auf diesen Puffer.

PURPLE --- 4

liefert die Konstante 4 als Farbwert für violett

QUERY

liest 80 Zeichen bzw. bis zum Carriage Return vom Bildschirm. Der Text wird in das Feld, dessen Adresse in TIB steht, eingetragen. IN wird auf Null gesetzt.

QUIT

initialisiert den Return-Stack, beendet Compile-Modus, und schaltet das System auf Terminal-Eingabe

R --- n

kopiert das oberste Wort des Return-Stacks auf den Datenstack, ohne ersteren zu verändern. (In manchen Versionen R@).

R# --- addr U

User-Variable; wird vom Editor zur Positionsangabe des Cursors verwendet.

RO --- addr U

Variable, mit der der Returnstack initialisiert wird

R/W addr blk f ---

Input/Output Routine für einen Satz. addr ist die Adresse des zu transferierenden Block-Puffers. blk ist die Blocknummer des Blocks; f ist eine Kennung, f=0 für Output, f=1 für Input.

R> --- n

legt das oberste Wort des Return-Stacks auf dem Datenstack ab.

READ addr count n1 n2 --- f

count Bytes von Gerät n2 über Kanal n1 ab Adresse addr einlesen; Fehlerflag 0=OK

RED --- 2

liefert die Konstante 2 als Farbwert für rot.

REPEAT P,C2

wird in einer ':'-Definition in der Form benutzt:
BEGIN ... WHILE ... REPEAT
Zur Laufzeit bewirkt es einen unbedingten Sprung zum entsprechenden BEGIN.

RND n1 --- n2

liefert eine Zufallszahl im Bereich 0 bis n1-1

RNDNR --- n

Variable, die zur Zufallszahlenerzeugung mittels RND benutzt wird

ROLL b ---

läßt die oberen b Elemente des Stacks um eine Position nach unten rotieren, und holt das bisherige Element b nach oben. ROLL ist eine Verallgemeinerung von ROT (- entpricht 3 ROLL).

ROT n1 n2 n3 --- n2 n3 n1

läßt die drei oben liegenden Worte des Stacks rotieren

RP!

initialisiert den Returnstackpointer

S! b1 b2 b3 --

schreibt den ASCII-Wert b1 in Zeile b2, Spalte b3 des Textrams (Adresse 1024 ff.)

S->D n --- d

das oberste Wort des Stacks wird vorzeichengerecht in eine doppeltlange Variable konvertiert

SO --- addr U

Variable, mit der der Datenstack initialisiert wird

S@ b1 b2 -- b3

liefert den ASCII-Wert im Textram bei Zeile b1, Spalte b2.

SCOPY n1 n2 ---

kopiert SCR n1 (liegt im Diskettenpuffer) nach SCR n2 auf Diskette. Der zu kopierende Screen sollte vollständig im Diskettenpuffer vorliegen. Dieses Wort dient u.a. zum Kopieren einzelner Screens auf eine andere Diskette, wobei beim Wechseln der Disketten gegebenenfalls die Anweisungen CLOSE-SCREEN und " I" DOS zum Initialisieren der Diskette zu geben sind.

SCR --- addr U

User-Variable; enthält die aktuelle Screennummer

SCREEN b ---

färbt den Bildschirm mit der Farbe b ein

SIGN n d --- d

Überträgt ein ASCII '-' in den Text einer numerischen Ausgabekonversion, wenn n negativ ist. n wird vom Stack entfernt. Darf nur zwischen <# und #> verwendet werden.

SMUDGE

wird in einer Definition benutzt, um in deren Namensfeld das 'Smudge-Bit' zu komplementieren (logisches NOT). Dies verhindert, daß eine unvollständige Definition vor einer fehlerfreien Kompilation gefunden wird.

SP! --- addr

initialisiert den Datenstackpointer

SP@ --- addr

der Stack-Pointer wird auf dem Stack abgelegt. Beispielsweise würde die Befehlsfolge:

```
1 2 SP@ @ . . .
```

die Ausgabe

```
2 2 1 OK
```

bewirken.

SPACE

überträgt ein ASCII-BLANK auf das Ausgabegerät

SPACES n ---

überträgt n BLANKs auf das Ausgabegerät

SPLIT n --- b1 b2

splittet n in LO- (b1) und HI-Byte

SREAD n ---

liest Screen n in den Diskettenpuffer ein

STATE --- addr U

User-Variable; enthält eine Kennung, ob sich das Programm im Zustand Kompilation oder Laufzeit befindet.

SWAP n1 n2 --- n2 n1

vertauscht die beiden obersten Worte des Stacks

TEXT b ---

liest Text bis zum nächsten Delimiter b (notfalls eine ganze Zeile) nach PAD

THEN P

Synonym für ENDIF

TIB --- addr U

User-Variable; enthält die Adresse des Terminal-Input-Puffers.

TOGGLE addr b ---

die gesetzten Bits von Byte b werden in der Zelle addr komplementiert.

TO.NKEY b ---

speichert die Adresse der Tastenzuordnungstabelle b in FTABLE und aktiviert die alternative KEY-Funktion NKEY.

TRACEON ---

Einschalten eines sehr ausführlichen TRACE-Protokolls. Achtung: Alle weiteren Befehle (auch TRACEOFF) können nur mehr zeichenweise mit anschließendem CR eingegeben werden.

TRACEOFF

schaltet den Tracemodus aus

TRAVERSE addr1 n --- addr2

diese Anweisung tastet sich von einem Ende eines Namenfelds zum anderen. addr1 ist entweder die Adresse des Längen-Bytes eines Namensfeldes oder die Adresse des letzten Zeichens des Namens. Ist n=1, so erfolgt das Tasten in Richtung steigender Adressen, bei n=-1 in Richtung fallender Adressen. addr2 ist die Adresse am entgegengesetzten Ende des Namensfelds; d.h. entweder die Adresse des Längen-Bytes oder die Adresse des letzten Zeichens.

TRIAD n ---

listet auf dem Ausgabegerät Screen n und zwei weitere Seiten. Diese Ausgabe eignet sich zur Dokumentation und beinhaltet die Ausgabe von Zeile 15, Screen 4 als Standard-Kommentarzeile.

TYPE addr count ---

Gibt count Zeichen ab addr auf das Ausgabegerät aus

U* u1 u2 --- ud

Betragsrechnung. Das Ergebnis ist das Doppelwort der Multiplikation u1*u2

U/ ud u1 --- u2 u3

Betragsrechnung. u2 ist der Rest, u3 der Quotient der Division ud/u1. Der Dividend ud ist ein Doppelwort.

U< u1 u2 --- f

setzt die Kennung 'wahr', wenn die Betragszahl u1 kleiner als die Betragszahl u2 ist, ansonsten 'falsch'.

U. u ---

Ausgabe einer Betragszahl an das Ausgabegerät

UNTIL f --- (run-time)

wird in einer ':'-Definition in der Form benutzt:
  BEGIN ... UNTIL
Die durch BEGIN ... UNTIL definierte Schleife wird verlassen, wenn f=TRUE

UPDATE

Markiert den zuletzt angesprochenen Block (auf den PREV verweist) als geändert. Dieser Block wird automatisch auf Diskette transferiert, wenn sein Puffer von einem anderen Block benötigt wird.

USE --- addr

Variable, die die Adresse des als nächstes zu verwendenden Block-Puffers enthält. Es ist die Adresse des Puffers, in den zuletzt geschrieben wurde.

USER n ---

Definitionswort der Form
  n USER cccc
Diese Anweisung definiert eine Variable cccc im USER-Bereich (s. Anhang 1). n ist der Offset zum Anfang des Bereichs. Sie können die Uservariable mit einem Offset von 64 bis 127 definieren. Die Variable wird mit 0 initialisiert.

VARIABLE E

definierendes Wort der Form:
  n VARIABLE cccc
Wenn VARIABLE durchgeführt wird, wird cccc definiert und mit n vorbesetzt. Wenn cccc durchgeführt wird, wird die Variablenadresse auf dem Stack abgelegt. Der Wert der Variablen kann mit cccc @ ermittelt werden.

VIC --- D000

Konstante; liefert die Adresse des VIC im C64

VICCR1 --- D011

Konstante; liefert die Adresse des 1. Kontrollregisters im VIC.

VICCR2 --- D016

Konstante; liefert die Adresse des 2. Kontrollregisters im VIC.

VIC-ADDPTR --- D018

Konstante; liefert die Adresse des Adressweichenregisters im VIC.

VIC-BORDER --- D014

Konstante; liefert die Adresse des VIC-Registers, das die Randfarbe des Bildschirms enthält.

VIC-BGO --- D015

Konstante; liefert die Adresse des VIC-Registers, das die (normale) Hintergrundfarbe enthält.

VIDEOAREA --- n

Variable, die die Adresse des Videobereichs (HEXA 400) enthält.

VOC-LINK --- addr U

User-Variable; enthält die Adresse des zuletzt verwendeten Vokabulars. Alle Vokabulare sind durch diese Felder verkettet. Ein FORGET ist also für mehrere Vokabulare wirksam.

VOCABULARY E,L

Definitionswort; wird in der Form benutzt:
VOCABULARY cccc IMMEDIATE
um ein neues Vokabular cccc zu definieren. Ein späterer Aufruf von cccc macht es zum CONTEXT-Vokabular, das als erstes bei einer INTERPRET-Anweisung abgesucht wird. Die Anweisungfolge:
cccc DEFINITIONS
macht cccc zum CURRENT-Vokabular, in dem neue Definitionen abgelegt werden. Vokabelnamen werden vereinbarungsgemäß 'immediate' deklariert.

VLIST

gibt alle definierten Namen des Wörterbuchs aus, beginnend mit dem CONTEXT-Vokabular.

WARNING --- addr U

User-Variable; enthält einen Wert zur Steuerung von Systemmeldungen, bei
1: werden Fehlermeldungen von Diskette ab Screen 4 ausgegeben
0: werden nur Fehlernummern ausgegeben
-1: wird (ABORT) durchgeführt

WHERE ---

wenn beim Laden von Screens ein Fehler auftritt, wird dieser mit WHERE lokalisiert. Die entsprechende Seite wird automatisch zum Editieren vorgelegt.

WHILE f ---

wird in einer ':'-Definition in der Form benutzt:
BEGIN ... WHILE(tp) ... REPEAT
Zur Laufzeit verwaltet WHILE den Ablauf des Programms. Wenn f 'falsch' ist, wird die Schleife hinter REPEAT fortgesetzt, ansonsten wird sie weiter durchlaufen.

WHITE --- 1

Farbkonstante mit dem Wert 1

WIDTH --- addr U

User-Variable; enthält den Wert 31 für die Maximallänge eines FORTH-Namens.

WORD c ---

liest den nächsten Text vom Input, bis zum Delimiter c. Der Text wird ab Adresse HERE in der Form abgelegt:
erstes Byte: Anzahl der Zeichen im Text
ab zweitem Byte: der Text
anschließend: zwei Blanks
Führende Zeichen c werden ignoriert.

WRITE addr n1 n2 n3 --- f

n1 Bytes ab Adresse addr werden über Kanal n2 auf Gerät n3 ausgeben; (f=0 bedeutet OK).

XOR n1 n2 --- xor

exklusives OR von n1 und n2

YELLOW --- 7

Farbkonstante mit dem Wert 7

[P

wird in einer ':'-Definition in der Form benutzt:

```
: xxx [ words ] words ;
```

Die Anweisungen hinter ' [' werden ausgeführt statt compiliert. Die ' ['-Anweisung erlaubt das Errechnen oder die Behandlung spezieller Fälle, bevor die Kompilation durch eine '] '-Anweisung fortgesetzt wird (s. LITERAL).

[COMPILE] P

wird in einer ':'-Definition z.B. in der Form benutzt:

```
: xxx [COMPILE] FORTH
```

[COMPILE] bewirkt die Kompilation einer Immediate-Definition, die ansonsten durchgeführt würde. In diesem Beispiel wird das Vokabular FORTH während der Ausführung der Anweisung statt während der Kompilation aufgerufen.

]

die durch eine ' ['-Anweisung suspendierte Kompilation wird wieder fortgesetzt. (s. [ ).

Deutschlands großer Verlag für Computerbücher und Programme präsentiert:

DAS TRAININGSBUCH ZU TURBO-PASCAL

Jeder, der Turbo-Pascal wirklich beherrschen will, sollte mit diesem leichtverständlichen Trainingsbuch arbeiten. Aus dem Inhalt: Grundlagen und Struktur von Pascal, Anweisungen, Prozeduren und Funktionen, Datentypen, Typkonstanten, Zeigertyp, Compileroptionen und Direktiven, Abweichungen von Standard-Pascal, Graphik PC-Dos/MS-DOS und natürlich Hinweise und Tips zur Installation und zur Bedienung des Editors. Viele Übungsaufgaben und Beispiele helfen das Gelernte zu vertiefen. Mit zahlreichen übersichtlichen Abbildungen und Struktogrammen. Schneller können Sie Turbo-Pascal nicht lernen!
Das Trainingsbuch zu Turbo-Pascal, 1984, 269 Seiten, DM 39,–

Alles über PASCAL

Dieses Trainingsbuch bietet eine leichtverständliche Einführung sowohl in UCSD-Pascal wie auch in PASCAL 64, wobei allerdings EDV- und BASIC-Grundkenntnisse vorausgesetzt werden. Themen des Buches: Programmstrukturen, Ein/Ausgabe, Arithmetik und Funktionen, Prozeduren und Rekursionen, Sets, Files und Records. Ein Anhang mit PASCAL-Schlüsselwörtern macht das Buch zusätzlich zum Nachschlagewerk.
Das Trainingsbuch zu Pascal, 1984, 256 Seiten, DM 39,–

Eine Fülle interessanter Programme und Anregungen für jeden PASCAL 64 Anwender: Definition neuer Datenstrukturen: Liste, Stack, Schlange. Funktionsprogramme: Selbstlernende Programme (Spiele), Zahlenumwandlung, CAD/CAM, 3-D-GRAFIK, Koordinatensysteme. Musik: Prinzip eines Synthesizers. TEXTOMAT als Editor für PASCAL, Disk-Dump und viele weitere Superprogramme.
PASCAL 64 Tips & Tricks, 1984, über 200 Seiten, DM 39,–
Diskette zum Buch: PASCAL 64 Tips & Tricks, DM 39,–

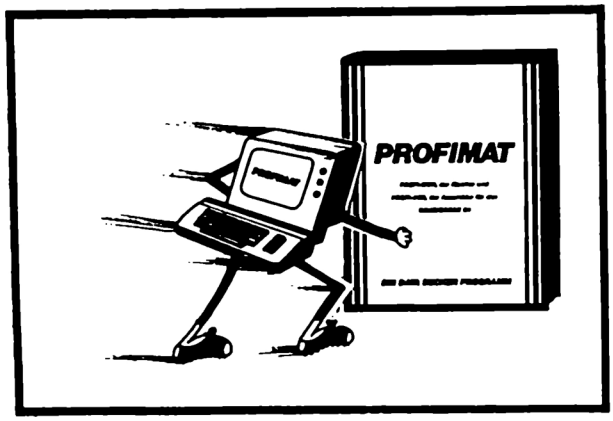
PASCAL 64 ist ein PASCAL Compiler für den C-64: sehr umfangreicher Befehlsvorrat, erlaubt Interruptprogrammierung und erzeugt sehr schnelle Programme in reinem Maschinencode, unterstützt relative Dateiverwaltung, Graphik und Sound, Datentypen: REAL, CHAR, BOOLEAN, INTEGER, Aufzähltypen, Pointer. Datenstrukturen: RECORD, SET, ARRAY und PACKED ARRAY. Mit ausführlichem Handbuch.
PASCAL 64, DM 99,–

DATA BECKER

Merowingerstr. 30 · 4000 Düsseldorf · Tel. (0211) 310010

Der C64 spricht nicht nur Basic...

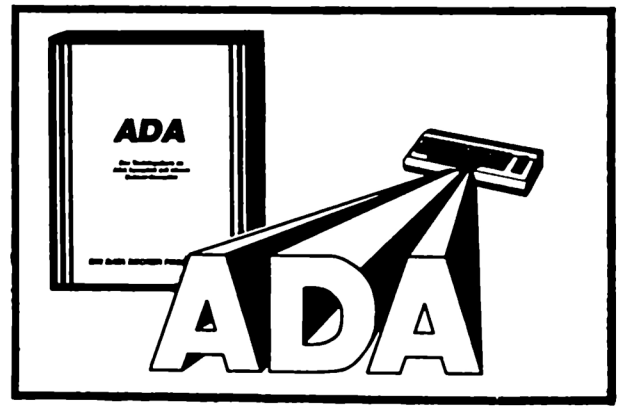
PROFIMAT



Wer sich tiefer in die Innereien des Computers begeben will, kommt ohne besonderes Werkzeug nicht aus. Einerseits muß der volle Einblick in alle Speicherbereiche möglich sein, andererseits soll der Umgang mit Maschinenprogrammen so komfortabel wie möglich gestaltet sein.
PROFIMAT hat Lösungen für beide Probleme: Der Maschinensprache-Monitor PROFI-MON bietet alle Hilfsmittel zum Umgang mit Maschinenprogrammen; PROFI-ASS ist ein Macro-Assembler, der das Schreiben von Maschinenprogrammen fast so einfach macht wie das Programmieren in BASIC.

PROFIMAT in Stichworten:
Registerinhalte und Flags anzeigen – Speicherinhalte anzeigen – Maschinenprogramme laden, ausführen und speichern – Speicherbereiche durchsuchen, vergleichen, füllen und verschieben – echter Einzelschrittmodus – Setzen von Unterbrechungspunkten – schneller Trace-Modus – Rückkehr zu BASIC – formatfreie Eingabe – Verkettung beliebig vieler Quellprogramme – erzeugter Objektcode kann in Speicher oder auf Diskette gehen – formatiertes Assemblerlisting – ladbare Symboltabellen – redefinierbare Symbole – Operatoren – Unterstützung der Fließkommaarithmetik – bedingte Assemblierung – Assemblerschleifen – MACROS mit beliebigen Parametern.
DM 99,– *

*unverbindliche Preisempfehlung



ADA

Diese Programmiersprache der Zukunft, die das Pentagon in Auftrag gegeben hat, wird jetzt durch DATA BECKER auch dem C-64 Anwender zugänglich gemacht durch den TRAININGSKURS zu ADA, der eine sehr gute Einführung in diese Supersprache bietet. Der dazu gelieferte Compiler liefert ein umfangreiches Subset der Sprache.

ADA in Stichworten:
blockstrukturierte Programme – modularer Aufbau der Programme – ermöglicht die Behandlung von Ausnahmezuständen – Fehlerüberprüfung beim Übersetzen und zur Laufzeit – ermöglicht das einfache Einbinden von Maschinenprogrammen – sehr leichtes Arbeiten mit Programmbibliotheken – Programmdiskette enthält Editor, Übersetzer, Assembler und Disassembler – umfangreiches deutsches Handbuch.
DM 198,– *

*unverbindliche Preisempfehlung

BASIC 64

Leistungsdaten, die überzeugen

BASIC 64 ist ein optimierender BASIC-Compiler für den Commodore 64, der Ihre in BASIC geschriebenen Programme schneller und leistungsfähiger macht. Dabei können Sie wählen, ob BASIC 64 Ihre Programme in einen P-Code oder direkt in Maschinensprache übersetzen soll. Das übersetzte Programm wird dadurch bis zu 6mal schneller als das Original-Programm.

In vielen Fällen, besonders beim Einsatz von Integer-Variablen, wird durch den in BASIC 64 enthaltenen Programmoptimierer Ihr BASIC-Programm sogar 6–10mal schneller als das Original. Dies gilt ebenfalls für die Verarbeitung von Zeichenketten, außerdem ist die gefürchtete Garbage Collection bei compilierten Programmen wesentlich schneller als die des Interpreters (max. 1 Sekunde statt mehrere Minuten), was die Anwendung von BASIC für anspruchsvollere Programme erst möglich macht.

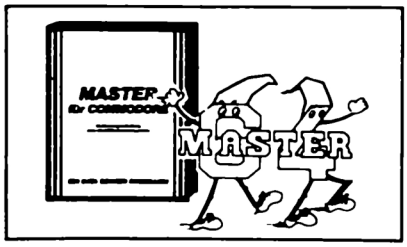
Obwohl zu jedem compilierten BASIC-Programm noch ein Runtimemodul mit einer Länge von 5 K hinzukommt, werden besonders größere BASIC-Programme durch das Compilieren kürzer (um ca. 25%). Außerdem stehen dem compilierten Programm 62 K für Programm und Daten zur Verfügung und nicht nur 38 K, wie bei Verwendung des BASIC-Interpreters.

BASIC 64 kann aus Ihren BASIC-Programmen aber auch direkt Maschinenspracheprogramme erzeugen, welche zwar ca. doppelt so lang sind wie das Original, aber nochmals um einiges schneller als ein P-Code-Programm. Mit BASIC 64 compilierte Programme werden dadurch in vielen Fällen 10–20fach schneller als ein BASIC-Programm. Ein Mischen von kurzem P-Code und extrem schneller Maschinensprache ist ebenfalls möglich.

Selbstverständlich compiliert BASIC 64 Programme beliebiger Größe und auch Overlay-Pakete, wobei die Compiliergeschwindigkeit ca. 1 K Byte pro Minute beträgt. Ebenfalls unterstützt werden die meisten Befehle folgender BASIC-Erweiterungen: Supergrafik 64, Simon's BASIC, Exbasic Level II, BASIC 4.0 (enthalten in Diskomat und Master 64) und andere.

BASIC 64 bietet Ihnen viele weitere Möglichkeiten, wie z.B.: zwei Optimierungsstufen, variabler Codestart, variable Speicherbenutzung, Umdefinieren von Datentypen von Variablen, Berechnung von konstanten Ausdrücken und Zeichenketten während der Compilierung, Optimieren und Umstellen von Formeln, Syntaxprüfung von Programmen, Erstellen einer Zeilenadressenliste, Start von Unterprogrammen bei Laufzeitenfehlern, IF...THEN...ELSE, Datenschnittstelle zum Assembler Profi-Ass, etc.

BASIC 64 ist kompatibel zum eingebauten BASIC-Interpreter des Commodore 64 und bildet zusammen mit diesem ein ideales Programmentwicklungssystem, mit dem schnelle und leistungsfähige Programme geschrieben werden können, für die bisher das Programmieren in Maschinensprache nötig war. Der Preis: nur DM 99,– einschließlich ausführlichem Handbuch.



MASTER 64 ist ein professionelles Programmentwicklungssystem für den C-64, das es Ihnen ermöglicht, die Programmentwicklungszeit auf einen Bruchteil der sonst üblichen Zeit zu reduzieren. MASTER 64 bietet einen Programmkomfort, den Sie nutzen sollten.

MASTER 64 in Stichworten:
70 zusätzliche Befehle – Bildschirmmaskengenerator – definieren von Bildschirmzonen – Eingabe aus Zonen – formatierte Ausgabe – Abspeicherung von Bildschirminhalten – Arbeiten mit mehreren Bildschirmmasken – ISAM Dateiverwaltung, in der Datensätze über einen Zugriffschlüssel angesprochen werden können – Datensätze bis zu 254 Zeichen – Schlüssellänge bis zu 30 Zeichen – Dateigröße nur von Diskettenkapazität abhängig – Zugriff über Schlüssel und Auswahlmasken – Bildschirm- und Druckmaskengenerator – Erstellung beliebiger Formulare und Ausgabemasken – BASIC-Erweiterungen – Toolkitfunktionen – Mehrfachgenaue Arithmetik (Rechnen mit 22 Stellen Genauigkeit). **DM 198,–***

*unverbindliche Preisempfehlung

...einfach besser programmieren

Bedienerfreundlich und erfolgreich in BASIC programmieren ist keine Sache nur für Profis. Wie man's macht verraten die Software-Autoren aus dem Hause DATA BECKER: Menuesteuerung, Maskenaufbau, Parameterisierung und Dokumentation sind die Stichworte. Dazu die leistungsfähige Datenverwaltung QUISAM mit lauffertigen Beispielprogrammen.

64 FÜR PROFIS, 2. Auflage, 1984, ca. 300 Seiten, DM 49,–

Viele weitere interessante DATA BECKER Bücher und Programme finden Sie im großen DATA BECKER KATALOG, den Sie kostenlos bei Ihrem Händler oder gegen DM 1,10 in Briefmarken direkt von uns erhalten.

DATA BECKER

Merowingerstr. 30 · 4000 Düsseldorf · Tel. (0211) 310010

Das TRAININGSBUCH ZU FORTH gibt nicht nur eine leicht verständliche Einführung, sondern bietet auch viele tiefergehende Informationen und wichtige Hinweise über den internen Aufbau dieser Programmiersprache.
Das Trainingsbuch zu FORTH, 1984, 300 Seiten, DM 39,–

Compiler gehören zu den wichtigsten Arbeitsmitteln eines Programmierers. Hier werden Grundlagen, Funktionsweise und richtiger Einsatz von Compilern gezeigt, ebenso die Entwicklung eines Compilers am Beispiel einer eigenen Sprache. Eine Pflichtlektüre für jeden ernsthaften Programmierer.
COMPILER VERSTEHEN – ANWENDEN – ENTWICKELN, 1984, 336 Seiten, DM 49,–