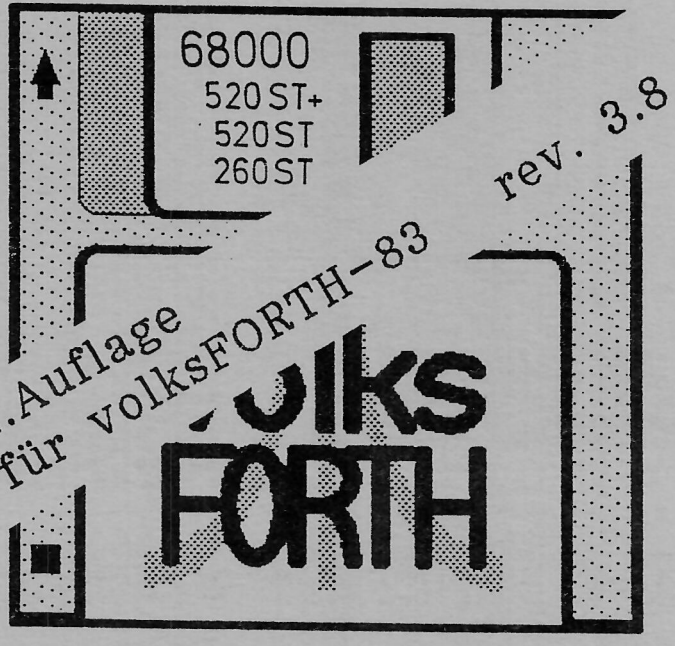


HANDBUCH

für

ATARI

260ST, 520ST, 520ST+

Handbuch zum **volksFORTH83 rev 3.8**
2. Auflage 18.12.1986

Die Autoren haben sich in diesem Handbuch um eine vollständige und akkurate Darstellung bemüht. Die in diesem Handbuch enthaltenen Informationen dienen jedoch allein der Produktbeschreibung und sind nicht als zugesicherte Eigenschaften im Rechtssinne aufzufassen. Ewaige Schadensersatzansprüche gegen die Autoren - gleich aus welchem Rechtsgrund - sind ausgeschlossen, soweit die Autoren nicht Vorsatz oder grobe Fahrlässigkeit trifft. Es wird keine Gewähr übernommen, daß die angegebenen Verfahren frei von Schutzrechten Dritter sind.

Unser Dank gilt der gesamten FORTH-Gemeinschaft, insbesondere Mike Perry, Charles Moore und Henry Laxen.

INHALTSVERZEICHNIS

Teil 3 - Glossare

Teil 4 - Definition der verwendeten Begriffe

Anhang

Prolog : über das volksFORTH83

volksFORTH83 ist eine Sprache, die in verschiedener Hinsicht ungewöhnlich ist. Einen ersten Eindruck vom volksFORTH83 und von unserem Stolz darüber soll dieser Prolog vermitteln.

volksFORTH83 braucht nicht geknackt oder geklaut zu werden. Im Gegenteil, wir hoffen, daß viele Leute das volksFORTH83 möglichst schnell bekommen und ihrerseits weitergeben.

Warum stellen wir dieses System als "public domain" zur Verfügung ?

Die Verbreitung, die die Sprache FORTH gefunden hat, war wesentlich an die Existenz von figFORTH geknüpft. Auch figFORTH ist ein public domain Programm, d.h. es darf weitergegeben und kopiert werden. Trotzdem haben sich bedauerlicherweise verschiedene Anbieter die einfache Adaption des figFORTH an verschiedene Rechner sehr teuer bezahlen lassen. Das im Jahr 1979 erschienene figFORTH ist heute nicht mehr so aktuell, weil mit der weiteren Verbreitung von Forth eine Fülle von eleganten Konzepten entstanden sind, die z.T. im Forth-Standard von 1983 Eingang gefunden haben. Daraufhin wurde von Laxen und Perry das F83 geschrieben und als Public Domain verbreitet.
Dieses freie 83-Standard-Forth mit zahlreichen Utilities ist recht komplex, es wird auch nicht mit Handbuch geliefert.
Insbesondere gibt es keine Version für C64- und Apple-Computer. Der C64 spielt jedoch in Deutschland eine große Rolle.

Wir haben ein neues Forth für verschiedene Rechner entwickelt. Das Ergebnis ist das volksFORTH83, eines der besten Forth-Systeme, die es gibt. Es wurde zunächst für den C64 geschrieben. Nach Erscheinen der Rechner der Atari ST-Serie entschlossen wir uns, auch für sie ein volksFORTH83 zu entwickeln. Die erste ausgelieferte Version 3.7 war, was Editor und Massenspeicher betraf, noch stark an den C64 angelehnt. Sie enthielt jedoch schon einen verbesserten Tracer, die GEM-Bibliothek und die anderen Tools für den ST. Der nächste Schritt bestand in der Einbindung der Betriebssystem-Files. Nun konnten Quelltexte auch vom Desktop und mit anderen Utilities verarbeitet werden. Mit der jetzt verteilten Version 3.8 ist das volksFORTH vollständig: Der GEM-unterstützte Editor erleichert auch Anfängern das Programmieren, und der Code ist relokatibel geworden, sodaß Applikationen sehr einfach zu realisieren sind. Die Probleme der früheren Versionen mit Accessories etc. entfallen ebenfalls. Schließlich wurde die Dokumentation erheblich erweitert, und für die wichtigsten Files liegen Shadowscreens vor. Der Editor dürfte mit seinen fast 50 kByte Kommentar das am besten dokumentierte umfangreiche GEM-Programm sein, das frei erhältlich ist. Gleichzeitig führt er exemplarisch vor, wie man GEM in Forth programmiert.

Warum soll man in volksFORTH83 programmieren ?

Das volksFORTH83 ist ein ausgesprochen leistungsfähiges und kompaktes Werkzeug. Durch residente Runtime-library, Compiler, Editor und Debugger sind die ermüdenden ECLG-Zyklen ("Edit, Compile, Link and Go") überflüssig; der Code wird Modul für

Modul entwickelt, kompiliert und getestet. Der integrierte Debugger ist die perfekte Testumgebung für Worte; es gibt keine riesigen Hexdumps oder Assemblerlistings, die kaum Ähnlichkeit mit dem Quelltext haben.
Ein anderer wichtiger Aspekt ist das Multitasking. So wie man ein Programm in einzelne, unabhängige Module oder Worte aufteilt, so sollte man es auch in einzelne, unabhängige Prozesse aufteilen können. Das ist in den meisten Sprachen nicht möglich, auch mit den sog. Desk-Accessories nicht. Das volksFORTH83 besitzt einen einfachen, aber leistungsfähigen Multitasker, der auch für Aufgaben eingesetzt werden kann, die über einen Druckerspooler hinausgehen.
Schließlich besitzt das volksFORTH83 noch eine Fülle von Details, die andere Forthsysteme nicht haben. Es benutzt an vielen Stellen Vektoren und sog. deferred Worte, die eine einfache Umgestaltung des Systems für verschiedene Gerätekonfigurationen ermöglichen. Es besitzt einen Heap (für "namenlose" Worte oder für Code, der nur zeitweilig benötigt wird). Der Blockmechanismus ist so schnell, daß er auch sinnvoll für die Bearbeitung großer Datenmengen, die in Files vorliegen, eingesetzt werden kann.
Die mit dem System gelieferten Disketten umfassen Tracer, Decompiler, Multitasker, Assembler, Editor, Disassembler, GEM-Bibliothek, Graphikdemos, Printerinterface ...

Das volksFORTH83 erzeugt, verglichen mit anderen Forthsystemen, relativ schnellen Code, der aber langsamer als der anderer Compilersprachen ist. Alle anderen Interpreter hängt es aber mühelos ab, auch sogenannte "schnelle" BASICs.

Kurz: Das System stellt in einer Vielzahl von Details einen Fortschritt dar!

Noch einmal : Ihr dürft und sollt diese Disks an eure Freunde weitergeben.
Aber wenn sich jemand erdreistet, damit einen Riesenreibach zu machen, dann werden wir ihn bis an das Ende der Welt und seiner Tage verfolgen !

Wir behalten uns die kommerzielle Verwertung des volksFORTH83 vor!

Mit diesem Handbuch soll die Unterstützung des volksFORTH83 noch nicht zuende sein. Die Forth Gesellschaft e.v., ein gemeinütziger Verein, bietet dafür die Plattform. Sie gibt die "VIERTE DIMENSION - Forth Magazin" heraus und betreibt den FORTH-Tree, einen ungewöhnlichen, aber sehr leistungsfähigen "Mailbox"-ähnlichen Rechner.

Forth Gesellschaft e.V.
Friedensalle 92
2000 HAMBURG 50
040 - 390 42 04 (Dienstags 18-20 Uhr mündlich, sonst Tree !)

Schickt uns bitte auch eure Meinung zum volksFORTH, Programme, die fertig oder halbfertig sind, Ideen zum volksFORTH, Artikel, die in der Presse erschienen sind (schreibt selbst welche !) , kurz: Schickt uns alles, was entfernt mit dem volksFORTH zu tun hat. Und natürlich : Anregungen, Ergänzungen und Hinweise auf evtl. Fehler im Handbuch und volksFORTH83.

Wir sind nämlich, trotz der vielen verteilten Systeme, des Lobes und der Kritik, die uns erreichte, etwas enttäuscht über die geringe "forthige" Resonanz !

Wenn euch das volksFORTH gefällt, so erwarten wir eine Spende von ca DM 20,-, denn die Entwicklung des volksFORTH sowie des Handbuchs war teuer und der Preis, den wir verlangen, deckt nur die Unkosten.

Für die Autoren des volksFORTH :

Bernd Pennemann
Steilshooper Str. 46
2000 Hamburg 60

Über dieses Handbuch

Dieses Handbuch ist kein Lehrbuch. Für Anfänger sei auf das immer noch beste Lehrbuch "Starting Forth" von Leo Brodie verwiesen. Das Buch ist auf Deutsch unter dem Titel "Programmieren in Forth" im Hanser Verlag erschienen. Dieses Handbuch enthält auch eine Liste der Abweichungen des volksFORTH83 von dem im Brodie besprochenen Forth. Diese Liste ist erforderlich, weil Brodie sein Buch vor 1983 geschrieben hat, als es den 83-Standard noch nicht gab.
Es wird vorausgesetzt, daß der Leser dieses Handbuchs "Starting Forth" oder ein ähnliches Lehrbuch kennt. Darüber hinausgehende Kenntnisse sind jedoch nicht unbedingt erforderlich.

Dieses Handbuch dokumentiert die benutzten Konzepte und Worte.

Der erste Teil führt den Leser in die Benutzung des volksFORTH83 auf dem Atari ST ein.
Der zweite Teil erklärt wie dieses Forth-System aufgebaut ist und wie es funktioniert. Dieser Teil soll dem Anfänger die Möglichkeit geben, sich in diese umfangreiche Sprache vollständig einzuarbeiten, also auch zu wissen, wie sie funktioniert. Dem Fortgeschrittenen soll er die Übertragung auf andere Rechner oder Prozessoren erleichtern.

Der dritte Teil enthält, nach Sachgruppen geordnet, die Worte des volksFORTH83. Soweit sie durch den 83er-Standard festgelegt sind, wurden die Definitionen aus dem Standard übersetzt.

Der vierte Teil enthält Definitionen der benutzten Begriffe. Sie entstammen direkt dem Standard und wurden angepaßt.

Schließlich befinden sich im Anhang Erläuterungen zu Editor, Assembler, GEM, Tools, Fehlermeldungen und Abweichungen von anderen Forth-Systemen.

Namen von Forthworten werden im Text groß geschrieben.

Teil 1 Getting started ...

Um volksFORTH83 vollständig nutzen zu können, sollten Sie dieses Handbuch sorgfältig studieren. Damit Sie möglichst schnell einsteigen können, werden wir in diesem Kapitel beschreiben,

- wie man mit den Disketten umgeht
- wie man das System startet
- wie man eine fertige Applikation erstellt.
- wie man ein eigenes Arbeitssystem zusammenstellt

Umgang mit den Disketten

Zu Ihrem Handbuch haben Sie drei Disketten erhalten. Fertigen Sie auf jeden Fall Sicherheitskopien von diesen Disketten an. Die Gefahr eines Datenverlustes ist groß, da FORTH Ihnen in jeder Hinsicht freie Hand läßt - auch beim versehentlichen Löschen Ihrer Systemdisketten !! Auf der einen Diskette befinden sich die Programme 4TH.PRG, und FORTHKER.PRG sowie ein Ordner mit Textdokumenten und das Demoprogramm 'Super copy'. Diese Diskette wird im Folgenden Systemdiskette genannt.

4TH.PRG ist das normale Arbeitssystem, FORTHKER.PRG eine Minimalversion, die nur den Sprachkern enthält. Damit können Sie eigene FORTH-Versionen z.B. mit einem veränderten Editor oder anderer Graphic zusammenstellen und mit **SAVESYSTEM** als fertiges System abspeichern. In der gleichen Art können Sie auch fertige Applikationen herstellen, denen man ihre 'FORTH-Abstammung' nicht mehr ansieht (ein Beispiel dazu ist SUPER COPY auf Ihrer Systemdiskette.) 4TH.PRG ist ein komplettes Arbeitsssystem mit residentem Fileinterface, Editor, Assembler, Tools, usw.

Die beiden anderen Disketten enthalten alle weiteren Quelltexte des Systems. Ein Verzeichnis erhalten Sie mit **FILES** (analog zu **WORDS**). Bei dem File FORTH_83.SCR handelt es sich um den Quelltext des Sprachkerns. Eben dieser Quelltext ist mit einem Target-Compiler kompiliert worden und entspricht exakt dem FORTHKER.PRG . Sie können sich also den Compiler ansehen, wenn Sie wissen wollen, wie das volksFORTH83 funktioniert. Im File STARTUP.SCR gibt es einen Loadscreen, der alle Teile kompiliert, die zu 4TH.PRG gehören. Mit diesem Loadscreen ist aus FORTHKER.PRG das File 4TH.PRG zusammengestellt worden.

Erster Einstieg

Legen Sie die Systemdiskette in Laufwerk B, die Quelltextdiskette Files 2 in Laufwerk A. (Beim Arbeiten mit einem Laufwerk werden Sie an den entsprechenden Stellen zum Wechseln der Diskette aufgefordert.) Laden Sie 4TH.PRG wie üblich durch Anklicken mit der Maus. volksFORTH83 meldet sich nun mit einer Einschaltmeldung, die die Versionsnummer enthält.

Geben Sie jetzt ein:

```
use tutorial.scr    <Return>
1 l                 <Return>
```

Auf die Aufforderung 'Enter your ID :' antworten Sie zunächst mit <Return>. Sie befinden sich jetzt im Editor und können sich diesen und die folgenden Screens ansehen und dabei den ersten Umgang mit dem Editor erlernen. Der Editor läuft komplett unter einer GEM-Umgebung, sodaß sich Anleitungen erübrigen. Trotzdem finden Sie einige Erklärungen im Anhang. Probieren Sie ein wenig herum und verlassen Sie dann den Editor, z.B. durch Drücken von <Esc>.

Sie können sich jetzt z.B. die Graphik-Demos ansehen. Sie befinden sich auf derselben Diskette. Zusätzlich wird allerdings noch der Assembler von Diskette Files 1 gebraucht. Legen Sie also diese anstelle der Systemdiskette in Laufwerk B. Geben Sie ein

```
include demo.scr  <Return>
```

Das Laufwerk läuft an, Sie sehen zuerst, welche Screens gerade kompiliert werden und dann einige Graphic-Beispiele. Nach jedem Bild können Sie mit einer beliebigen Taste weitermachen oder mit <ESC> abbrechen.

Jetzt gehts aber richtig los ....

Wir wollen jetzt einige eigene Worte kompilieren und dazu ein neues Quelltextfile anlegen. Legen Sie dazu eine neue Diskette in Laufwerk A ein. Geben Sie dann ein:

```
makefile first.scr   2 more   <Return>
```

Sie erzeugen damit ein File namens FIRST.SCR mit einer Länge von 2 Blöcken (2048 Byte), bestehend aus Screen 0 und 1. Um einige Definitionen auf Screen 1 zu schreiben, rufen Sie den Editor auf mit

```
1 l   <Return>
```

In der obersten Zeile eines Screens sollte eine Kurzbeschreibung dessen, was der Screen enthält, stehen. (Mit dem Wort INDEX erhält man so ein Inhaltsverzeichnis des Files.) Schreiben Sie also, beginnend in der linken oberen Ecke :

```
\ Mein erstes FORTH-Programm
```

Der \ (Backslash) sorgt dafür, daß diese Zeile nicht kompiliert wird, sondern als Kommentar aufgefaßt wird. Vielleicht ist Ihnen auch bereits aufgefallen, daß bei unseren Quelltexten in der oberen rechten Ecke Datum und Autor der letzten Änderung vermerkt sind. Diese Kennung erzeugt der Editor automatisch, wenn beim ersten Aufruf eine ID angegeben wurde. Sie können Ihre ID jedoch auch nachträglich setzen, wenn Sie die Tastenkombination CTRL-G drücken oder den entsprechenden Menüpunkt "GET-ID" anwählen.

Die zweite Zeile des Screens bleibt frei, in der dritten definieren wir:

```
: test      ." Das ist mein erstes 'Programm'." ;
```

Verlassen Sie nun den Editor mit CTRL-S. Dabei wird der - inzwischen ja veränderte - Screen auf Diskette zurückgeschrieben. Nun rufen wir den Compiler auf mit

```
1 load   <Return>
```

Mit für 'C'- oder Pascal-Programmierer unvorstellbarer Geschwindigkeit wird unser Mini-Programm kompiliert und steht jetzt zur Ausführung bereit. Geben Sie einmal WORDS ein, dann werden Sie feststellen, daß Ihr neues Wort TEST ganz oben im Dictionary steht (drücken Sie die <ESC>-Taste, um die Ausgabe von WORDS abzubrechen oder irgendeine andere Taste, um sie anzuhalten). Um das Ergebnis unseres ersten Programmierversuchs zu überprüfen, geben wir nun ein:

```
test    <Return>
```

und siehe da, es tut sich etwas. Schöner wäre es allerdings, wenn der Satz auf einer neuen Zeile beginnen würde. Um dies zu ändern, werfen wir erst einmal das alte Wort TEST weg.

```
forget test
```

Nun rufen wir erneut den Editor auf mit V . Vor dem String fügen wir ein CR ein. Das Wort sieht dann so aus :

```
: test   cr ." Das ist mein erstes 'Programm'." ;
```

Dann kompilieren wir wie gehabt. Unsere Änderung erweist sich als erfolgreich, und Sie haben gelernt, wie einfach in FORTH das Schreiben, Austesten und Ändern von Programmteilen ist.

Herstellung einer Applikation

Wir wollen unser 'Programm' nun als eigenständige Applikation abspeichern. Dazu erweitern wir es zunächst ein klein wenig (Editor mit V aufrufen.) FÜgen Sie nun noch folgende Definition in einer neuen Zeile hinzu:

```
: runalone      test  key drop  bye ;
```

RUNALONE führt zuerst TEST aus, wartet dann auf eine Taste und kehrt zum Desktop zurück. Kompilieren Sie nun erneut, führen Sie RUNALONE aber nicht aus, sonst würden wir ja FORTH verlassen. Das Problem besteht vielmehr darin, das System so abzuspeichern, daß es gleich nach dem Laden RUNALONE ausführt und sonst gar nichts.

volksFORTH83 ist an zwei Stellen für solche Zwecke vorbereitet. In den Worten COLD und RESTART befinden sich zwei 'deferred words' namens 'COLD bzw. 'RESTART , die im Normalfall nichts tun, vom Anwender aber nachträglich verändert werden können. Wir benutzen hier 'COLD, um auch schon die Startmeldung zu unterbinden.

Geben Sie also ein

```
' runalone Is 'cold     <Return>
```

und speichern Sie das Ganze mit

```
savesystem myprog.prg
```

auf Diskette zurück. Laden Sie dann MYPROG.PRG durch das übliche anklicken. Sie haben Ihre erste Applikation erstellt !

Etwas enttäuschend ist es aber schon. Das angeblich so kompakte FORTH benötigt über 40 kByte, um einen lächerlichen String auszugeben ?? Da stimmt doch etwas nicht. Natürlich, wir haben ja eine Reihe Systemteile mit abgespeichert, vom Fileinterface über den Assembler, die halbe GEM-Bibliothek, den Editor usw., die für unser Programm überhaupt nicht benötigt werden.

Um dieses und ähnliche Probleme zu lösen, gibt es das File FORTHKER.PRG. Dieses Programm enthält nur den Systemkern und das Fileinterface. Laden Sie also FORTHKER.PRG und kompilieren Sie Ihre Applikation mit

```
include first.scr
```

Dann wie gehabt, RUNALONE in 'COLD patchen und das System auf Diskette zurückspeichern. Sie haben jetzt eine verhältnismäßig kompakte Version vorliegen. Natürlich ließe sich auch diese noch erheblich kürzen, aber dafür bräuchten Sie einen Target-Compiler, mit dem Sie nur noch die wirklich benötigten Systemteile selektiv aus dem Quelltext zusammenstellen könnten. Mit der beschriebenen Methode lassen sich aber auch größere Programme kompilieren und als Stand-alone-Applikationen abspeichern.

Für Fortgeschrittene

Man kann allerdings im obigen Beispiel ohne Target-Compiler auskommen, wenn man auf das File-Interface, das ca. 4 KByte belegt, verzichten kann. Dazu muß man zunächst eine FORTH-Version ohne Fileinterface herstellen. Laden Sie dazu FORTHKER.PRG und geben Sie ein:

```
' blk@   Is makeview     <Return>
' noop   Is custom-remove     <Return>
' STr/w  Is r/w     <Return>
  direct     <Return>
```

Damit werden die 'deferred words', die das Fileinterface umgestellt hat, wieder auf Direktmodus umgeschaltet und das Fileinterface selbst abgeschaltet.

Das erste Wort im Fileinterface oberhalb von SAVESYSTEM ist DOS. Dieses Wort kann man aber nicht mit FORGET vergessen, da es sich im 'geschützten Bereich' befindet. Probieren Sie es ruhig aus; Sie erhalten die Meldung: 'DOS is protected'. Für solche Fälle gibt es das Wort (FORGET .

```
' Dos >name 4- (forget    save     <Return>
```

Damit haben Sie ein System 'ohne alles'. Speichern Sie es mit

```
savesystem minimal.prg     <Return>
```

auf Diskette zurück.

Natürlich können wir unser Testfile nicht mehr mit INCLUDE laden, denn es gibt ja kein Fileinterface mehr. Stattdessen suchen wir uns die entsprechenden Blocks im Direktzugriff. Laden Sie also wieder 4TH.PRG, schalten Sie das Fileinterface mit

```
direct
```

ab, und geben Sie ein:

```
1 500 index
```

(500 ist zwar sicher mehr als reichlich, aber INDEX stoppt sowieso automatisch beim letzten Disketten- oder Fileblock.) Unser Testscreen dürfte ziemlich weit hinten stehen, aber wir bekommen so auch gleich einen Überblick über den Diskettenin-halt. Merken Sie sich die Nummer n des Testscreens, und verlassen Sie FORTH. Laden Sie dann MINIMAL.PRG und kompilieren Sie Ihren Screen ohne Angabe eines Filenamens mit <n> LOAD.
Ganz trickreiche Programmierer sind sicherlich auch in der Lage, mit diesem System das Fileinterface auf den Heap zu laden, ähnlich wie das beim Assembler vorgeführt wird. Dann kann man es zum Zusammenstellen der Applikation benutzen, SAVESYSTEM speichert es aber nicht mit ab.

Herstellen eines eigenen Systems

Das File 4TH.PRG ist als Arbeitsversion gedacht. Es enthält alle wichtigen Systemteile wie Editor, Printer-Interface, Tools, GEM-Graphic, Decompiler, Tracer usw. Sollte Ihnen die Zusammenstellung nicht gefallen, können Sie sich jederzeit ein Ihren speziellen Wünschen angepaßtes System zusammenstellen. Schlüssel dazu ist der Loadscreen im File STARTUP.SCR der Diskette Files 1. Sie können dort Systemteile, die Sie nicht benötigen, mit dem Backslash '\' wegkommentieren oder die entsprechenden Zeilen ganz löschen. Ebenso können Sie natürlich dem Loadscreen eigene Files hinzufügen.

Entspricht der Loadscreen Ihren Wünschen, speichern Sie ihn mit CTRL-S zurück, und verlassen Sie das System mit BYE. Laden Sie nun File FORTHKER.PRG. Geben Sie dann ein:

```
include startup.scr    <Return>
```

Ist das System fertig kompiliert, legen Sie die Systemdiskette ein und schreiben:

```
savesystem 4th.prg   <Return>
```

Damit wird Ihr altes File überschrieben (Sicherheitskopie !!!), sodaß Sie beim nächsten Laden von 4TH.PRG Ihr eigenes System erhalten. Natürlich können Sie 'Ihr' System auch unter einem anderen Namen mit SAVESYSTEM abspeichern.

Ebenso können Sie Systemvoreinstellungen ändern. Unsere Arbeitsversion arbeitet - voreingestellt - neuerdings im dezimalen Zahlensystem. Natürlich können Sie mit

```
hex    <Return>
```

auf Hexadezimalsystem umstellen; wir halten das für sehr viel sinnvoller, weil vor allem Speicheradressen im Dezimalsystem kaum etwas aussagen (oder wissen Sie, ob Speicherstelle 978584 im Bildschirmspeicher liegt oder nicht ?) . Wollen Sie bereits unmittelbar nach dem Laden im Hexadezimalsystem arbeiten, können Sie sich dies mit SAVESYSTEM, wie oben beschrieben, abspeichern.

Im Übrigen empfehlen wir bei allen Zahlen über 9 dringend die Benutzung der sogenannten Präfixe $ für Hexadezimal-, & für Dezimal- und % für Binärzahlen. Man vermeidet so, daß irgendwelche Files nicht - oder noch schlimmer, falsch - kompiliert werden, weil man gerade im anderen Zahlensystem ist. Außerdem ist es möglich, hexadezimale und dezimale Zahlen beliebig zu kombinieren, je nachdem, was gerade sinnvoller ist. In unseren Quelltexten finden Sie genug entsprechende Beispiele.

Wenn Sie nur ein Laufwerk zur Verfügung haben, sollten Sie den voreingestellten Suchpfad - es wird erst Laufwerk A, dann Laufwerk B durchsucht - ändern. Wenn nur auf Laufwerk A gesucht werden soll, geben Sie folgende Sequenz ein

```
path ;  a:
```

und speichern das System mit SAVESYSTEM auf Diskette zurück.

Umkopieren des Systems auf doppelseitige Disketten

Wenn Sie über doppelseitige Laufwerke verfügen, können Sie das System auf doppelseitige Disketten umkopieren. Der Vorteil liegt darin, daß die VIEW-Funktion natürlich nur dann funktioniert, wenn die entsprechenden Quelltextfiles verfügbar sind. Und auf ein oder zwei doppelseitigen Disketten läßt sich schon eine Reihe Files zur Verfügung halten.

Ebenso ist es natürlich möglich, alle Teile des Systems auf einer Festplatte abzulegen. Sie sollten dafür einen eigenen Ordner einrichten und PATH und DIR entsprechend einstellen.

Auch die Arbeit mit einer RAM-Disk ist prinzipiell möglich, allerdings nicht sehr zu empfehlen. FORTH ist sehr maschinennah und Systemabstürze daher vor allem zu Anfang nicht so ganz auszuschließen. Wir empfehlen stattdessen die Verwendung des Files RAMDISK.SCR . Damit werden sehr viele Blockbuffer im RAM - außerhalb des FORTH-Systems - eingerichtet, was die Kompilationszeiten erheblich beschleunigt. Trotzdem ist große Datensicherheit vorhanden, weil alle geänderten Blocks immer auf Diskette zurückgeschrieben werden, wenn man den Editor mit CTRL-S verläßt. Die Gefahr eines Datenverlustes bei Systemabsturz ist so nahezu ausgeschlossen.

Ausdrucken der Quelltexte

Sicher ist es sinnvoll, die Quelltexte des Systems auszudrucken, um Beispiele für den Umgang mit volksFORTH83 zu sehen. Inzwischen ist ein großer Teil der Quelltexte mit Kommentarscreens versehen, was - wie wir hoffen - gerade für den Einsteiger eine große Hilfe darstellt. Welche Files sich im Einzelnen auf Ihren Disketten befinden und ob sie Kommentarscreens enthalten, steht im File README.DOC.

Zunächst müssen Sie das Printerinterface hinzuladen, falls es nicht schon vorhanden ist. Reagiert Ihr volksFORTH83 auf die Eingabe von PRINTER mit einem kräftigen Haeh? , so ist das Printerinterface nicht vorhanden. Legen Sie die entsprechende Diskette ein und laden Sie es mit :

```
include printer.scr
```

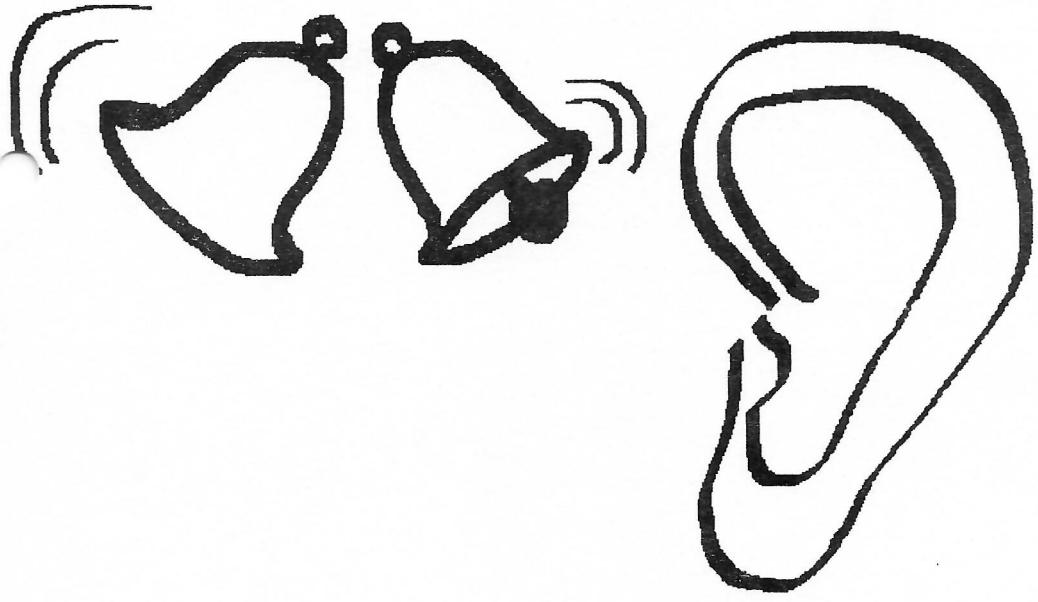
Zum Ausdrucken von Files mit Shadowscreens gibt man ein :

```
USE <filename>    LISTING
```

Für Files ohne Shadowscreens benutzt man :

```
USE <filename>    PRINTALL
```

Näheres zum Drucker-Interface finden Sie im Kapitel 4 des zweiten Teils.



Erläuterungen

Teil 2

1) Dictionarystruktur des volksFORTH83

Das Forthsystem besteht aus einem Dictionary von Worten. Die Struktur der Worte und des Dictionaries soll im folgenden erläutert werden.

1.1) Struktur der Worte

Die Forthworte sind in Listen angeordnet (s.a. Struktur der Vokabulare). Die vom Benutzer definierten Worte werden ebenfalls in diese Listen eingetragen . Jedes Wort besteht aus sechs Teilen.
Es sind dies:

a) "block"
Der Nummer des Blocks, in dem das Wort definiert wurde (siehe auch VIEW im Glossar).

b) "link"
Einer Adresse (Zeiger) , die auf das "Linkfeld" des nächsten Wortes zeigt.

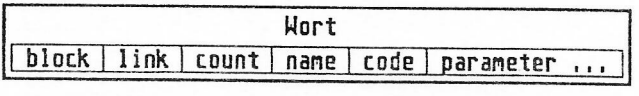
c) "count"
Die Länge des Namens dieses Wortes und drei Markierungsbits.

d) "name"
Der Name selbst.

e) "code"
Einer Adresse (Zeiger) , die auf den Maschinencode zeigt, der bei Aufruf dieses Wortes ausgeführt wird. Die Adresse dieses Feldes heißt Kompilationsadresse.

f) "parameter"
Das Parameterfeld.

Ein Wort sieht dann so aus :



1.1.1) Countfeld

Das count-Feld enthält die Länge des Namens (1..31 Zeichen) und drei Markierungsbits :

restrict	immediate	indirect	Länge
Bit: 7	6	5	4..0

Ist das immediate-Bit gesetzt, so wird das entsprechende Wort im kompilierenden Zustand unmittelbar ausgeführt, und nicht ins Dictionary kompiliert (siehe auch IMMEDIATE im Glossar).

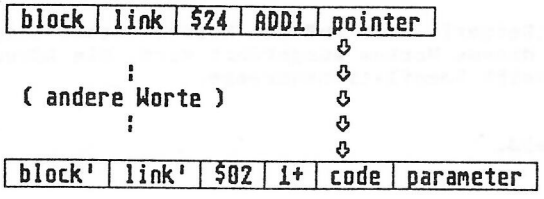
Ist das restrict-Bit gesetzt, so kann das Wort nicht durch Eingabe von der Tastatur ausgeführt, sondern nur in anderen Worten kompiliert werden. Gibt man es dennoch im interpretierenden Zustand ein, so erscheint die Fehlermeldung "compile only" (siehe auch RESTRICT im Glossar).

Ist das indirect-Bit gesetzt, so folgt auf den Namen kein Codefeld, sondern ein Zeiger darauf. Damit kann der Name vom Rumpf (Code- und Parameterfeld) getrennt werden. Die Trennung geschieht z.B. bei Verwendung der Worte ' oder ALIAS.

Beispiel:

```
' 1+ Alias add1
```

ergibt folgende Struktur im Speicher (Dictionary) :



(Bei allen folgenden Bildern werden die Felder block link count nicht mehr extra dargestellt.)

1.1.2) Name

Der Name besteht normalerweise aus ASCII-Zeichen. Bei der Eingabe werden Klein- in Großbuchstaben umgewandelt. Daher druckt WORDS auch nur groß geschriebene Namen. Da Namen sowohl groß als auch klein geschrieben eingegeben werden können, haben wir eine Konvention erarbeitet, die die Schreibweise von Namen festlegt:

Bei Kontrollstrukturen wie DO LOOP etc. werden alle Buchstaben groß geschrieben.

Bei Namen von Vokabularen, immediate Worten und definierenden Worten, die CREATE ausführen, wird nur der erste Buchstabe groß geschrieben. Beispiele sind: Is Forth Constant

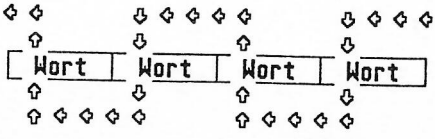
Alle anderen Worte werden klein geschrieben. Beispiele sind: dup cold base

Bestimmte Worte, die von immediate Worten kompiliert werden, beginnen mit der öffnenden Klammer "(" , gefolgt vom Namen des immediate Wortes. Ein Beispiel: DO kompiliert (do .

Diese Schreibweise ist nicht zwingend; Sie sollten sich aber daran halten, um die Lesbarkeit Ihrer Quelltexte zu erhöhen. Das volksFORTH unterscheidet nicht zwischen Groß- und Kleinschreibung, sodaß Sie die Worte beliebig eingeben können.

1.1.3) Link

Über das link-Feld sind die Worte eines Vokabulars zu einer Liste verkettet. Jedes link-Feld enthält die Adresse des vorherigen link-Feldes. Jedes Wort zeigt also auf seinen Vorgänger. Das unterste Wort der Liste enthält im link-Feld eine Null. Die Null zeigt das Ende der Liste an.



1.1.4) Block

Das block-Feld enthält in codierter Form die Nummer des Blocks und den Namen des Files, in dem das Wort definiert wurde. Wurde es von der Tastatur aus eingegeben, so enthält das Feld Null.

1.1.5) Code

Jedes Wort weist auf ein Stück Maschinencode. Die Adresse dieses Code-Stücks ist im Code-Feld enthalten. Gleiche Worttypen weisen auf den gleichen Code. Es gibt verschiedene Worttypen, z.B. :-Definitionen , Variablen , Konstanten , Vokabulare usw. Sie haben jeweils ihren eigenen charakteristischen Code gemeinsam.
Die Adresse des Code-Feldes heißt Kompilationsadresse.

1.1.6) Parameter

Das Parameterfeld enthält Daten, die vom Typ des Wortes abhängen.
Beispiele :

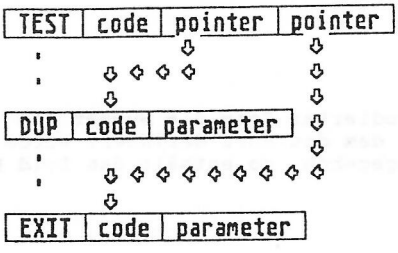
a) Typ "Constant"
Hier enthält das Parameterfeld des Wortes den Wert der Konstanten. Der dem Wort zugeordnete Code liest den Inhalt des Parameterfeldes aus und legt ihn auf den Stack.

b) Typ "Variable"
Das Parameterfeld enthält den Wert der Variablen, der zugeordnete Code liest jedoch nicht das Parameterfeld aus, sondern legt dessen Adresse auf den Stack. Der Benutzer kann dann mit dem Wort @ den Wert holen und mit dem Wort ! überschreiben.

c) Typ ":-definition"
Das ist ein mit : und ; gebildetes Wort. In diesem Fall enthält das Parameterfeld hintereinander die Kompilationsadressen der Worte, die diese Definition bilden. Der zugeordnete Code sorgt dann dafür, daß diese Worte der Reihe nach ausgeführt werden.

Beispiel : `: test dup ;`

ergibt:



Das Wort : hat den Namen TEST erzeugt.
EXIT wurde durch das Wort ; erzeugt

d) Typ "Code"
Worte vom Typ "Code" werden mit dem Assembler erzeugt. Hier zeigt das Codefeld in der Regel auf das Parameterfeld. Dorthin wurde der Maschinencode assembliert. Codeworte im volksFORTH können leicht "umgepatcht" werden, da lediglich die Adresse im Codefeld auf eine neue (andere) Maschinencodesequenz gesetzt werden muß.

1.2) Vokabular-Struktur

Eine Liste von Worten ist ein Vokabular. Ein Forth-System besteht im allgemeinen aus mehreren Vokabularen, die nebeneinander existieren. Neue Vokabulare werden durch das definierende Wort VOCABULARY erzeugt und haben ihrerseits einen Namen, der in einer Liste enthalten ist.
Gewöhnlich kann von mehreren Worten mit gleichem Namen nur das zuletzt definierte erreicht werden. Befinden sich jedoch die einzelnen Worte in verschiedenen Vokabularen, so bleiben sie einzeln erreichbar.

1.2.1) Die Suchreihenfolge

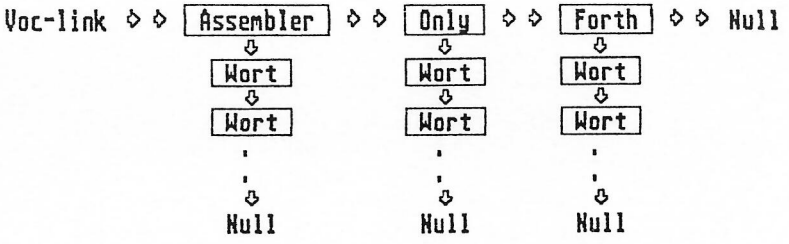
Die Suchreihenfolge gibt an, in welcher Reihenfolge die verschiedenen Vokabulare nach einem Wort durchsucht werden.
Sie besteht aus zwei Teilen, dem auswechselbaren und dem festen Teil. Der auswechselbare Teil enthält genau ein Vokabular. Dies wird zuerst durchsucht. Wird ein Vokabular durch Eingeben seines Namens ausgeführt, so trägt es sich in den auswechselbaren Teil ein. Dabei wird der alte Inhalt überschrieben. Einige andere Worte ändern ebenfalls den auswechselbaren Teil. Soll ein Vokabular immer durchsucht werden, so muß es in den festen Teil befördert werden. Dieser enthält null bis sechs Vokabulare und wird nur vom Benutzer bzw. seinen Worten verändert. Zur Manipulation stehen u.a. die Worte ONLY ALSO TOSS zur Verfügung. Das Vokabular, in das neue Worte einzutragen sind, wird durch das Wort DEFINITIONS angegeben. Die Suchreihenfolge kann man sich mit ORDER ansehen.

Beispiele :

Eingabe :	ORDER ergibt dann :
Onlyforth	FORTH FORTH ONLY FORTH
Editor also	EDITOR EDITOR FORTH ONLY FORTH
Assembler	ASSEMBLER EDITOR FORTH ONLY FORTH
definitions Forth	FORTH EDITOR FORTH ONLY ASSEMBLER
: test ;	ASSEMBLER EDITOR FORTH ONLY ASSEMBLER

1.2.2) Struktur des Dictionaries

Der Inhalt eines Vokabulars besteht aus einer Liste von Worten, die durch ihre link-Felder miteinander verbunden sind. Es gibt also genauso viele Listen wie Vokabulare. Alle Vokabulare sind selbst noch einmal über eine Liste verbunden, deren Anfang in VOC-LINK steht. Diese Verkettung ist nötig, um ein komfortables FORGET zu ermöglichen.
Man bekommt beispielsweise folgendes Bild:



2) Die Ausführung von Forth-Worten

Der geringe Platzbedarf übersetzter Forth-Worte rührt wesentlich von der Existenz des Adressinterpreters her.
Wie aus dem Kapitel 1.1.6 Absatz c) ersichtlich, besteht eine :-Definition aus dem Codefeld und dem Parameterfeld. Im Parameterfeld steht eine Folge von Adressen. Ein Wort wird kompiliert, indem seine Kompilationsadresse dem Parameterfeld der :-Definition angefügt wird. Eine Ausnahme bilden die Immediate Worte. Da sie während der Kompilation ausgeführt werden, können sie dem Parameterfeld der :-Definition alles mögliche hinzufügen.
Daraus wird klar, daß die meisten Worte innerhalb der :-Definition nur eine Adresse, also in einem 16-Bit-System genau 2 Bytes an Platz verbrauchen.
Wird die :-Definition nun aufgerufen, so sollen alle Worte, deren Kompilationsadresse im Parameterfeld stehen, ausgeführt werden. Das besorgt der Adressinterpreter.

2.1) Aufbau des Adressinterpreters beim volksFORTH83

Der Adressinterpreter benutzt einige Register der CPU, die im Kapitel über den Assembler aufgeführt werden. Es gibt aber mindestens die folgenden Register :

IP W SP RP

a) IP ist der Instruktionszeiger (englisch : Instruction Pointer). Er zeigt auf die nächste auszuführende Instruktion. Das ist beim volksFORTH83 die Speicherzelle, die die Kompilationsadresse des nächsten auszuführenden Wortes enthält.

b) W ist das Wortregister. Es zeigt auf die Kompilationsadresse des Wortes, das gerade ausgeführt wird.

c) SP ist der (Daten-) Stackpointer. Er zeigt auf das oberste Element des Stacks.

d) RP ist der Returnstackpointer. Er zeigt auf das oberste Element des Returnstacks.

2.2) Die Funktion des Adressinterpreters

NEXT ist die Anfangsadresse der Routine, die die Instruktion ausführt, auf die IP gerade zeigt.
Die Routine NEXT ist ein Teil des Adressinterpreters. Zur Verdeutlichung der Arbeitsweise schreiben wir hier diesen Teil in High Level:

```
Variable IP
Variable W
: Next   IP @  @  W !
         2 IP +!
         W perform  ;
```

Tatsächlich ist NEXT jedoch eine Maschinencoderoutine, weil dadurch die Ausführungszeit von Forth-Worten erheblich kürzer wird. NEXT ist somit die Einsprungadresse einer Routine, die diejenige Instruktion ausführt, auf die das Register IP zeigt. Ein Wort wird ausgeführt, indem der Code, auf den die Kompilationsadresse zeigt, als Maschinencode angesprungen wird.

Der Code kann z.B. den alten Inhalt des IP auf den Returnstack bringen, die Adresse des Parameterfeldes im IP speichern und dann NEXT anspringen.

Diese Routine gibt es wirklich, sie heißt "docol" und ihre Adresse steht im Codefeld jeder :-Definition. Das Gegenstück zu dieser Routine ist ein Forth-Wort mit dem Namen EXIT . Dabei handelt es sich um ein Wort, das das oberste Element des Returnstacks in den IP bringt und anschließend NEXT anspringt. Damit ist dann die Ausführung der Colon-definition beendet.

Beipiel :

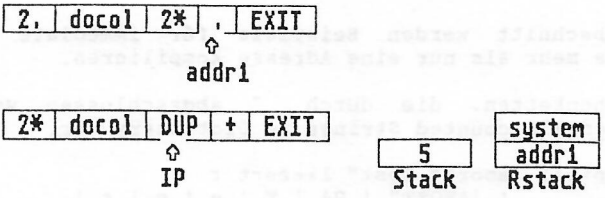
```
: 2*   dup + ;
: 2.   2* . ;
```

Ein Aufruf von
```
    5 2.
```
von der Tastatur aus führt zu folgenden Situationen :

a)

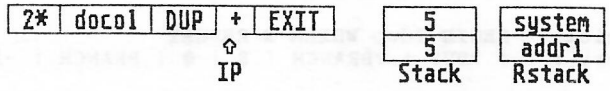
Nach der ersten Ausführung von NEXT bekommt man :

b)



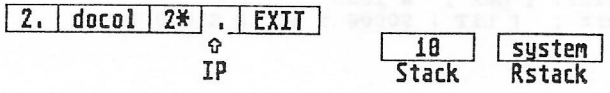
Nochmalige Ausführung von NEXT ergibt :
(DUP ist ein Wort vom Typ "Code")

c)



Nach der nächsten Ausführung von NEXT zeigt der IP auf EXIT und nach dem darauf folgenden NEXT wird addr1 wieder in den IP geladen :

d)



e) Die Ausführung von . erfolgt analog zu den Schritten b,c und d. Anschließend zeigt IP auf EXIT in 2. . Nach Ausführung von NEXT kehrt das System wieder in den Textinterpreter zurück. Dessen Rückkehradresse wird durch system angedeutet. Damit ist die Ausführung von 2. beendet.

2.3) Verschiedene Immediate Worte

In diesem Abschnitt werden Beispiele für immediate Worte angegeben, die mehr als nur eine Adresse kompilieren.

a) Zeichenketten, die durch " abgeschlossen werden, liegen als counted Strings im Dictionary vor.

   Beispiel: **abort" Test"** liefert :
   | (ABORT" | 04 | T | e | s | t |

b) Einige Kontrollstrukturen kompilieren ?BRANCH oder BRANCH , denen ein Offset mit 16 Bit Länge folgt.

   Beispiel: **0< IF swap THEN -** liefert :
   | 0< | ?BRANCH | 4 | SWAP | - |

   Beispiel: **BEGIN ?dup WHILE @ REPEAT**
   liefert : **| ?DUP | ?BRANCH | 8 | @ | BRANCH | -10 |**

c) Ebenso fügen DO und ?DO einen Offset hinter das von ihnen kompilierte Wort (DO bzw. (?DO . Mit dem Dekompiler können Sie sich eigene Beispiele anschauen.

d) Zahlen werden in das Wort LIT , gefolgt von einem 16-Bit-Wert, kompiliert.

   Beispiel: **[hex] 8 1000**
   liefert : **| LIT | $0008 | LIT | $1000 | .**

3) Die Does>-Struktur

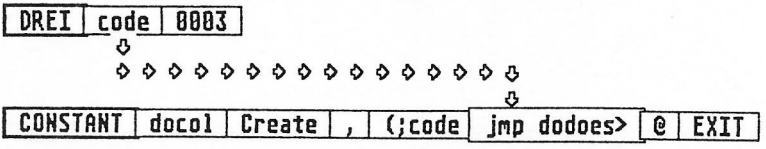
Die Struktur von Worten, die mit CREATE .. DOES> erzeugt wurden, soll anhand eines Beispiels erläutert werden.

Das Beispielprogramm lautet :

```
: Constant     Create , Does> @ ;
3 Constant drei
```

Der erzeugte Code sieht folgendermaßen aus:

Der jump-Befehl ist symbolisch gemeint. In Wirklichkeit wird eine andere Adressierungsart benutzt !



Das Wort (;CODE wurde durch DOES> erzeugt. Es setzt das Codefeld des durch CREATE erzeugten Wortes DREI und beendet dann die Ausführung von CONSTANT . Das Codefeld von DREI zeigt anschließend auf jmp dodoes>. Wird DREI später aufgerufen , so wird der zugeordnete Code ausgeführt. Das ist in diesem Fall jmp dodoes>. dodoes> legt nun die Adresse des Parameterfeldes von DREI auf den Stack und führt die auf jmp dodoes> folgende Sequenz von Worten aus. (Man beachte die Ähnlichkeit zu :-Definitionen).
Diese Sequenz besteht aus @ und EXIT . Der Inhalt des Parameterfeldes von DREI wird damit auf den Stack gebracht. Der Aufruf von DREI liefert also tatsächlich 0003. Statt des jmp dodoes> und der Sequenz von Forthworten kann sich hinter (;CODE auch ausschließlich Maschinencode befinden. Das ist der Fall, wenn wir das Wort ;CODE statt DOES> benutzt hätten. Wird diese Maschinencodesequenz später ausgeführt, so zeigt das W-Register des Adressinterpreters auf das Codefeld des Wortes, dem dieser Code zugeordnet ist. Erhöht man also das W-Register um 2 , so zeigt es auf das Parameterfeld des gerade auszuführenden Wortes.

4) Vektoren und Deferred Worte

Das volksFORTH83 besitzt eine Reihe von Strukturen, die dazu dienen, das Verhalten des Systems zu ändern.

4.1) deferred Worte

Sie werden durch das Wort DEFER erzeugt. Im System sind bereits folgende deferred Worte vorhanden:
R/W 'COLD 'RESTART 'ABORT 'QUIT NOTFOUND .STATUS DISKERR MAKEVIEW und CUSTOM-REMOVE.

Um sie zu ändern, benutzt man die Phrase:

```
' <name> Is <namex>
```

Hierbei ist <namex> ein durch DEFER erzeugtes Wort und <name> der Name des Wortes, das in Zukunft bei Aufruf von <namex> ausgeführt wird.

Wird <namex> ausgeführt bevor es mit IS initialisiert wurde, so erscheint die Meldung "Crash" auf dem Bildschirm.

Anwendungsmöglichkeiten von deferred Worten :

Durch Ändern von R/W kann man andere Floppies oder eine RAM-Disk betreiben. Es ist auch leicht möglich, Teile einer Disk gegen überschreiben zu schützen.
Durch Ändern von NOTFOUND kann man z.B. Worte, die nicht im Dictionary gefunden wurden, anschließend in einer Disk-Directory suchen lassen oder automatisch als Vorwärtsreferenz vermerken.
Ändert man 'COLD , so kann man die Einschaltmeldung des volksFORTH83 unterdrücken und stattdessen ein Anwenderprogramm starten, ohne daß Eingaben von der Tastatur aus erforderlich sind (siehe z.B. Teil 1, "Erstellen einer Applikation"). Ähnliches gilt für 'RESTART.
'ABORT ist z.B. dafür gedacht, eigene Stacks, z.B. für Fließkommazahlen, im Fehlerfall zu löschen. Die Verwendung dieses Wortes erfordert aber schon eine gewisse Systemkenntnis. Das gilt auch für das Wort 'QUIT.
'QUIT wird dazu benutzt, eigene Quitloops in den Compiler einzubetten. Wer sich für diese Materie interessiert, sollte sich den Quelltext des Tracer anschauen. Dort wird vorgeführt, wie man das macht.
.STATUS schließlich wird vor Laden eines Blocks ausgeführt. Man kann sich damit anzeigen lassen, welchen Block einer längeren Sequenz das System gerade lädt und z.B. wieviel freier Speicher noch zur Verfügung steht.
CUSTOM-REMOVE kann vom fortgeschrittenen Programmierer dazu benutzt werden, eigene Datenstrukturen, die miteinander durch Zeiger verkettet sind, zu vergessen. Ein Beispiel dafür sind die File Control Blöcke des Fileinterfaces.

4.2) >interpret

Dieses Wort ist ein spezielles deferred Wort, das für die Umschaltung des Textinterpreters in den Kompilationszustand und zurück benutzt wird.

4.3) Variablen

Es gibt im System die Uservariable ERRORHANDLER . Dabei handelt es sich um eine normale Variable, die als Inhalt die Kompilationsadresse eines Wortes hat. Der Inhalt der Variablen wird auf folgende Weise ausgeführt:

```
ERRORHANDLER PERFORM
```

Zuweisen und Auslesen dieser Variablen geschieht mit @ und ! . Der Inhalt von ERRORHANDLER wird ausgeführt, wenn das System ABORT" oder ERROR" ausführt und das von diesen Worten verbrauchte Flag wahr ist. Die Adresse des Textes mit der Fehlermeldung befindet sich auf dem Stack. Siehe z.B. (ERROR .

4.4) Vektoren

Das volksFORTH83 benutzt die indirekten Vektoren INPUT und OUTPUT. Die Funktionsweise der sich daraus ergebenden Strukturen soll am Beispiel von INPUT verdeutlicht werden.
INPUT ist eine Uservariable, die auf einen Vektor zeigt, in dem wiederum vier Kompilationsadressen abgelegt sind. Jedes der vier Inputworte KEY KEY? DECODE und EXPECT führt eine der Kompilationsadressen aus. Kompiliert wird solch ein Vektor in der folgenden Form:

```
Input: vector
       <name1> <name2> <name3> <name4> ;
```

Wird VECTOR ausgeführt, so schreibt er seine Parameterfeldadresse in die Uservariable INPUT . Von nun an führen die Inputworte die ihnen so zugewiesenen Worte aus. KEY führt also <NAME1> aus usw...

Das Beispiel KEY? soll dieses Prinzip verdeutlichen:

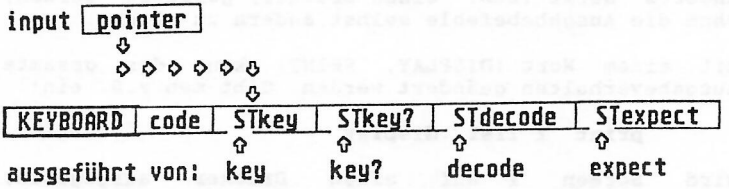
```
: key?   ( -- c )
     input @  2+  perform ;
```

(Tatsächlich wurde KEY? im System ebenso wie die anderen Inputworte durch das nicht mehr sichtbare definierende Wort IN: erzeugt.)

Ein Beispiel für einen Inputvektor, der die Eingabe von der Tastatur holt :

```
Input: keyboard
       STkey STkey? STdecode STexpect ;
keyboard
```

ergibt :



Analog verhält es sich mit OUTPUT und den Outputworten EMIT CR TYPE DEL PAGE AT und AT? . Outputvektoren werden mit OUTPUT: genauso wie die Inputvektoren erzeugt.
Mit der Input/Output-Vektorisierung kann man z.B. mit einem Schlag die Eingabe von der Tastatur auf ein Modem umschalten.

Bitte beachten Sie, daß immer alle Worte in der richtigen Reihenfolge aufgeführt werden müssen ! Soll nur ein Wort geändert werden, so müssen sie trotzdem die anderen mit hinschreiben.

Die Druckeranpassung

Alle Ausgabeworte (EMIT, TYPE, SPACE etc.) sind im volksFORTH-83 vektorisiert, d.h. bei ihrem Aufruf wird die Codefeldadresse des zugehörigen Befehls aus einer Tabelle entnommen und ausgeführt. Im System enthalten ist eine Tabelle mit Namen DISPLAY, die für die Ausgabe auf das Bildschirmterminal sorgt. Dieses Verfahren bietet entscheidende Vorteile:

- Mit einer neuen Tabelle können alle Ausgaben auf ein anderes Gerät (z.B. einen Drucker) geleitet werden, ohne die Ausgabebefehle selbst ändern zu müssen.

- Mit einem Wort (DISPLAY, PRINT) kann das gesamte Ausgabeverhalten geändert werden. Gibt man z.B. ein:

  ```
  print 1 list display
  ```

  wird Screen 1 auf einen Drucker ausgegeben, anschließend wieder auf den Bildschirm zurückgeschaltet. Man braucht also kein neues Wort, etwa PRINTERLIST, zu definieren.

Eine neue Tabelle wird mit dem Wort OUTPUT: erzeugt. (Die Definition können Sie mit VIEW OUTPUT: nachsehen.) OUTPUT: erwartet eine Liste von Ausgabeworten, die mit ; abgeschlossen werden muß. Beispiel:

```
Output: >printer
   pemit pcr ptype pdel ppage pat pat? ;
```

Damit wird eine neue Tabelle mit dem Namen >PRINTER angelegt. Beim späteren Aufruf von >PRINTER wird die Adresse dieser Tabelle in die Uservariable OUTPUT geschrieben. Ab sofort führt EMIT ein PEMIT aus, TYPE ein PTYPE usw. Die Reihenfolge der Worte nach OUTPUT: userEMIT userCR userTYPE userDEL userPAGE userAT userAT? muß unbedingt eingehalten werden.

Der folgende Quelltext enthält die Anpassung für einen Epson-Drucker RX80/FX80 oder kompatible am Atari ST. Zusätzlich zu den reinen Ausgaberoutinen (Screen 7) sind eine Reihe nützlicher Worte enthalten, mit denen die Druckersteuerung sehr komfortabel vorgenommen werden kann.

Arbeiten Sie mit einem Drucker, der andere Steuercodes als Epson verwendet, müssen Sie die Screens 2 und 4 bis 6 anpassen. Bei IBM-kompatiblen Druckern ist es meist damit getan, daß die Umlautwandlung (Screen 6) weggelassen wird. Dies erreicht man, indem man in Zeile 1 einen doppelten Backslash (\\) setzt.

Im Arbeitssystem ist das Printerinterface bereits enthalten. Müssen Sie Änderungen vornehmen, können Sie entweder jedesmal Ihr geändertes Printerinterface mit

```
include printer.scr
```

dazuladen oder sich ein neues Arbeitssystem zusammenstellen wie im Kapitel 'Getting started ...' beschrieben. Sie können natürlich auch den Loadscreen in seiner jetzigen Fassung benutzen und das Printer-Interface 'von Hand' nachladen.

Im zweiten Teil des Printer-Interface (Screen 9 ff.) sind einige Worte zur Ausgabe eines formatierten Listings enthalten. PTHRU druckt einen Bereich von Screens, PRINTALL ein ganzes File, jeweils 6 Screens auf einer DIN A4 Seite in komprimierter Schrift. Ganz ähnlich arbeiten die Worte DOCUMENT und LISTING , jedoch wird bei diesen Worten neben einen Quelltextscreen der zugehörige Shadowscreen gedruckt. (Mehr über das Shadowscreen-Konzept erfahren Sie im Kapitel über den Editor.) Man erhält so ein übersichtliches Listing eines Files mit ausführlichen Kommentaren. Es empfiehlt sich daher, alle Files, die Shadowscreens enthalten, einmal mit LISTING auszudrucken.

An dieser Stelle folgt das Listing des Files PRINTER.SCR.

```
  PRINTER.SCR Scr 0 Dr 0
 0 \\                  *** Printer-Interface ***            10oct86we
 1
 2 Dieses File enthält das Printer-Interface. Die Definitionen für
 3 die Druckersteuerung müssen ggf. an Ihren Drucker angepaßt wer-
 4 den.
 5
 6 PRINT  lenkt alle Ausgabeworte auf den Drucker um, mit  DISPLAY
 7 wird wieder auf dem Bildschirm ausgegeben.
 8
 9 Zum Ausdrucken der Quelltexte gibt es die Worte
10
11    pthru      ( from to -- )   druckt Screen from bis to
12    document   ( from to -- )  wie pthru, aber mit Shadow-Screens
13    printall   ( -- )    wie pthru, aber druckt das ganze File
14    listing    ( -- )    wie document, aber für das ganze File
15
```

```
  PRINTER.SCR Scr 1 Dr 0
 0 \ Printer Interface Epson RX80\FX80                    21oct86we
 1
 2 Onlyforth
 3
 4 \needs file?             ' noop  ¦ Alias file?
 5 \needs capacity          ' blk/drv Alias capacity
 6
 7 Vocabulary Printer   Printer definitions also
 8
 9  1 &13 +thru
10
11 Onlyforth  \ clear
12
13
14
15
```

```
  PRINTER.SCR Scr 2 Dr 0
 0 \ Printer  p! and controls                             18nov86we
 1
 2 ' bcostat ¦ Alias ready?   ' 0 ¦ Alias printer
 3
 4 : p!  ( n -- )
 5   BEGIN  pause  printer ready?  UNTIL  printer bconout ;
 6
 7
 8 ¦ : ctrl:  ( 8b -- )   Create c,   does>  ( -- )   c@ p! ;
 9
10  07   ctrl: BEL      $7F ¦ ctrl: DEL       $0D ¦ ctrl: RET
11 $1B ¦ ctrl: ESC      $0A   ctrl: LF        $0C   ctrl: FF
12
13
14
15
```

```
  PRINTER.SCR Scr 15 Dr 0
 0 \\                  *** Printer-Interface ***          13oct86we
 1
 2 Eingestellt ist das Druckerinterface auf Epson und kompatible
 3  Drucker. Die Steuersequenzen auf den Screens 2, 4 und 5 müssen
 4  gegebenenfalls auf Ihren Drucker angepaßt werden. Bei uns gab
 5  es mit verschiedenen Druckern allerdings keine Probleme, da
 6  sich inzwischen die meisten Druckerhersteller an die Epson-
 7  Steuercodes halten.
 8
 9 Arbeiten Sie mit einem IBM-kompatiblen Drucker, muß die Umlaut-
10  wandlung auf Screen 6 wegkommentiert werden.
11
12 Zusätzliche 'exotische' Steuersequenzen können nach dem Muster
13  auf den Screens 4 und 5 jederzeit eingebaut werden.
14
15
```

```
  PRINTER.SCR Scr 16 Dr 0
 0 \ Printer Interface Epson RX80                          13oct86we
 1
 2 setzt order auf  FORTH FORTH ONLY   FORTH
 3
 4 falls das Fileinterface nicht im System ist, werden die ent-
 5  sprechenden Worte ersetzt.
 6
 7 Printer-Worte erhalten ein eigenes Vocabulary.
 8
 9
10
11
12
13
14
15
```

```
  PRINTER.SCR Scr 17 Dr 0
 0 \ Printer  p! and controls                              10oct86we
 1
 2 nur aus stilistischen Gründen. Das Folgende liest sich besser.
 3
 4 Hauptausgabewort; gibt ein Zeichen auf den Drucker aus. Es wird
 5  gewartet, bis der Drucker bereit ist. (PAUSE für Multitasking)
 6
 7
 8 gibt Steuerzeichen an Drucker
 9
10 Steuerzeichen für Drucker. Gegebenenfalls anpassen!
11
12
13
14
15
```

```
  PRINTER.SCR Scr 3 Dr 0
 0 \ Printer controls                                        09sep86re
 1
 2 | : esc:  ( 8b -- )   Create c,   does>  ( -- )   ESC c@ p! ;
 3
 4 | : esc2  ( 8b0 8b1 -- )   ESC p! p! ;
 5
 6 | : on:  ( 8b -- )  Create c,  does>  ( -- )  ESC c@ p!  1 p! ;
 7
 8 | : off: ( 8b -- )  Create c,  does>  ( -- )  ESC c@ p!  0 p! ;
 9
10
11
12
13
14
15
```

```
  PRINTER.SCR Scr 4 Dr 0
 0 \ Printer Escapes Epson RX-80/FX-80                       12sep86re
 1
 2 $0F | ctrl: (+17cpi              $12 | ctrl: (-17cpi
 3
 4 Ascii P | esc: (+10cpi           Ascii M | esc: (+12cpi
 5 Ascii 0   esc: 1/8"              Ascii 1   esc: 1/10"
 6 Ascii 2   esc: 1/6"              Ascii T   esc: suoff
 7 Ascii N   esc: +jump             Ascii O   esc: -jump
 8 Ascii G   esc: +dark             Ascii H   esc: -dark
 9 \ Ascii 4   esc: +cursive          Ascii 5   esc: -cursive
10
11 Ascii W   on:  +wide             Ascii W   off: -wide
12 Ascii -   on:  +under            Ascii -   off: -under
13 Ascii S   on:  sub               Ascii S   off: super
14
15
```

```
  PRINTER.SCR Scr 5 Dr 0
 0 \ Printer Escapes Epson RX-80/FX-80                       12sep86re
 1
 2 : 10cpi   (-17cpi (+10cpi ;      ' 10cpi   Alias pica
 3 : 12cpi   (-17cpi (+12cpi ;      ' 12cpi   Alias elite
 4 : 17cpi   (+10cpi (+17cpi ;      ' 17cpi   Alias small
 5
 6 : lines  ( #.of.lines -- )   Ascii C esc2 ;
 7
 8 : "long  ( inches -- )   0 lines p! ;
 9
10 : american   0 Ascii R esc2 ;
11
12 : german     2 Ascii R esc2 ;
13
14 : normal     10cpi  american  suoff  1/6"  &12 "long  RET ;
15
```

```
  PRINTER.SCR Scr 18 Dr 0
 0 \ Printer controls                                        10oct86we
 1
 2 gibt Escape-Sequenzen an den Drucker aus.
 3
 4 gibt Escape und zwei Zeichen aus.
 5
 6 gibt Escape, ein Zeichen und eine 1 an den Drucker aus.
 7
 8 gibt Escape, ein Zeichen und eine 0 an den Drucker aus.
 9
10
11
12
13
14
15
```

```
  PRINTER.SCR Scr 19 Dr 0
 0 \ Printer Escapes Epson RX-80/FX-80                       10oct86we
 1
 2 setzt bzw. löscht Ausgabe komprimierter Schrift.
 3
 4 setzt Zeichenbreite auf 10 bzw. 12 cpi.
 5 Zeilenabstand in Zoll.
 6                                    schaltet Super- und Subscript ab
 7 Perforation überspringen ein- und ausschalten.
 8 Es folgen die Steuercodes für Fettdruck, Kursivschrift, Breit-
 9  schrift, Unterstreichen, Subscript und Superscript.
10  Diese müssen ggf. an Ihren Drucker angepaßt werden.
11  Selbstverständlich können auch weitere Fähigkeiten Ihres Druk-
12  kers genutzt werden wie Proportionalschrift, NLQ etc.
13
14
15
```

```
  PRINTER.SCR Scr 20 Dr 0
 0 \ Printer Escapes Epson RX-80/FX-80                       13oct86we
 1
 2 Hier wird die Zeichenbreite eingestellt. Dazu kann man sowohl
 3  Worte mit der Anzahl der characters per inch (cpi) als auch
 4  pica, elite und small benutzen.
 5
 6 setzt Anzahl der Zeilen pro Seite; Einstellung:
 7  &66 lines      oder     &12 "long
 8
 9
10 schaltet auf amerikanischen Zeichensatz.
11
12 schaltet auf deutschen Zeichensatz.
13
14 Voreinstellung des Druckers auf 'normale' Werte; wird beim
15  Einschalten mit PRINT ausgeführt.
```

```
  PRINTER.SCR Scr 6 Dr 0
 0 \ Umlaute                                                14oct86we
 1
 2 ¦ Create DIN
 3 Ascii ä c,      Ascii ö c,      Ascii ü c,      Ascii ß c,
 4 Ascii Ä c,      Ascii Ö c,      Ascii Ü c,      Ascii § c,
 5
 6 ¦ Create AMI
 7 Ascii { c,      Ascii ¦ c,      Ascii } c,      Ascii ~ c,
 8 Ascii [ c,      Ascii \ c,      Ascii ] c,      Ascii @ c,
 9
10 here AMI - ¦ Constant tablen
11
12 ¦ : p!  ( char -- )   dup $80 < IF  p! exit  THEN
13    tablen 0 DO  dup  I DIN + c@  =
14                 IF  drop  I AMI + c@  LEAVE  THEN  LOOP
15    german p! american ;
```

```
  PRINTER.SCR Scr 7 Dr 0
 0 \ Printer Output                                         12sep86re
 1
 2 ¦ Variable pcol   pcol off      ¦ Variable prow   prow off
 3
 4 ¦ : pemit  ( 8b -- )     p!  1 pcol +! ;
 5 ¦ : pcr  ( -- )          RET LF  1 prow +!  pcol off ;
 6 ¦ : pdel  ( -- )         DEL  pcol @ 1- 0 max pcol ! ;
 7 ¦ : ppage  ( -- )        FF  prow off  pcol off ;
 8 ¦ : pat  ( row col -- )   over  prow @ <  IF  ppage  THEN
 9      swap  prow @ -  0 ?DO  pcr  LOOP
10      dup  pcol @ <  IF  RET  pcol off  THEN  pcol @ - spaces ;
11 ¦ : pat?  ( -- row col )   prow @  pcol @ ;
12 ¦ : ptype  ( adr len -- )
13      dup pcol +!  bounds ?DO  I c@ p!  LOOP ;
14
15
```

```
  PRINTER.SCR Scr 8 Dr 0
 0 \ Printer output                                         18nov86we
 1
 2 Output: >printer   pemit pcr ptype pdel ppage pat pat? ;
 3
 4 Forth definitions
 5
 6 : print   >printer  normal ;
 7
 8
 9
10
11
12
13
14
15
```

```
  PRINTER.SCR Scr 21 Dr 0
 0 \ Umlaute                                              bp 12oct86
 1
 2 Auf diesem Screen werden die Umlaute aus dem IBM-(ATARI)-Zeichen
 3  satz in DIN-Umlaute aus dem deutschen Zeichensatz gewandelt.
 4
 5 Wenn Sie einen IBM-kompatiblen Drucker benutzen, kann dieser
 6  Screen mit \\ in der ersten Zeile wegkommentiert werden.
 7
 8
 9
10
11
12 p! wird neu definiert. Daher brauchen die folgenden Worte p!
13  nicht zu ändern, egal, ob mit oder ohne Umlautwandlung gearbei-
14  tet wird.
15
```

```
  PRINTER.SCR Scr 22 Dr 0
 0 \ Printer Output                                       10oct86we
 1
 2 aktuelle Druckerzeile und -spalte.
 3 Routinen zur Druckerausgabe      entspricht Befehl
 4 ein Zeichen auf Drucker          emit
 5 CR und LF auf Drucker            cr
 6 ein Zeichen löschen (?!)         del
 7 neue Seite                       page
 8 Drucker auf Zeile und Spalte     at
 9  positionieren; wenn nötig,
10  neue Seite.
11 Position feststellen            at?
12 Zeichenkette ausgeben           type
13
14 Damit sind die Worte für eine eigene Output-Struktur vorhanden.
15
```

```
  PRINTER.SCR Scr 23 Dr 0
 0 \ Printer output                                       10oct86we
 1
 2 erzeugt die Output-Tabelle >printer.
 3
 4 Die folgenden Worte sind von FORTH aus zugänglich.
 5
 6 schaltet Ausgabe auf Printer um. (Zurückschalten mit DISPLAY)
 7
 8
 9
10
11
12
13
14
15
```

```
  PRINTER.SCR Scr 9 Dr 0
 0 \ Variables and Setup                                    bp 12oct86
 1
 2 Printer definitions
 3
 4 ' 0 ¦ Alias logo
 5
 6 ¦ : header  ( pageno -- )
 7      12cpi  +dark  ."   volksFORTH-83    FORTH-Gesellschaft eV "
 8      -dark  17cpi  ." (c) 1985/86 we/bp/re/ks  "  12cpi +dark
 9      file?  -dark  17cpi ."  Seite "  . ;
10
11
12
13
14
15
```

```
  PRINTER.SCR Scr 10 Dr 0
 0 \ Print 2 screens across on a page                    26oct86we
 1
 2 ¦ : 2lines  ( scr#1 scr#2 line# -- )
 3      cr  dup 2 .r space  c/l * >r
 4      pad  c/l 2* 1+  bl fill  swap
 5      block  r@ +  pad           c/l cmove
 6      block  r> +  pad c/l + 1+  c/l cmove
 7      pad  c/l 2* 1+  -trailing  type ;
 8
 9 ¦ : 2screens  ( scr#1 scr#2 -- )
10      cr cr  &30 spaces
11      +wide +dark over 4 .r  &28 spaces  dup 4 .r  -wide -dark
12      cr  l/s 0 DO  2dup  I 2lines  LOOP  2drop ;
13
14
15
```

```
  PRINTER.SCR Scr 11 Dr 0
 0 \ print 6 screens on a page                           18sep86we
 1
 2 ¦ : pageprint  ( last+1 first pageno -- )
 3      header  2dup - 1+  2/  dup 0
 4      ?DO  >r  2dup under r@ + >
 5           IF  dup r@ +  ELSE  logo  THEN  2screens 1+ r>  LOOP
 6      drop 2drop  page ;
 7
 8 ¦ : >shadow   ( n1 -- n2 )
 9      capacity 2/  2dup < IF + ELSE - THEN ;
10
11 ¦ : shadowprint  ( last+1 first pageno -- )
12      header  2dup -  0
13      ?DO  dup dup >shadow  2screens  1+  LOOP
14      2drop page ;
15
```

```
  PRINTER.SCR Scr 24 Dr 0
 0 \ Variables and Setup                                    10oct86we
 1
 2 Diese Worte sind nur im Printer-Vokabular enthalten.
 3
 4 Dieser Screen wird gedruckt, wenn es nichts besseres gibt.
 5
 6 Druckt die Überschrift der Seite pageno.
 7
 8
 9
10
11
12
13
14
15
```

```
  PRINTER.SCR Scr 25 Dr 0
 0 \ Print 2 screens across on a page                       10oct86we
 1
 2 druckt nebeneinander die Zeilen line# der beiden Screens.
 3  Die komplette Druck-Zeile wird erst in PAD aufbereitet.
 4
 5
 6
 7
 8
 9 formatierte Ausgabe der beiden Screens nebeneinander
10  mit fettgedruckten Screennummern. Druck erfolgt mit 17cpi, also
11  in komprimierter Schrift.
12
13
14
15
```

```
  PRINTER.SCR Scr 26 Dr 0
 0 \ print 6 screens on a page                              10oct86we
 1
 2 gibt eine Seite aus. Anordnung der Screens auf der Seite:  1 4
 3  Wenn weniger als 6 Screens vorhanden sind, werden         2 5
 4  Lücken auf der rechten Seite mit dem Logo-Screen (0)      3 6
 5  aufgefüllt.
 6
 7
 8 berechnet zu Screen n1 den Shadowscreen n2 (Kommentarscreen wie
 9  dieser hier).
10
11 wie pageprint, aber anstelle der Screens 4, 5 und 6 werden die
12  Shadowscreens zu 1, 2 und 3 gedruckt.
13
14
15
```

```
  PRINTER.SCR Scr 12 Dr 0
 0 \ Printing without Shadows                                  b11nov86we
 1
 2 Forth definitions  also
 3
 4 ! Variable printersem    0 printersem !    \ for multitasking
 5
 6 : pthru  ( first last -- )       2 arguments
 7    printersem lock   output push  print
 8    1+  capacity umin  swap  2dup -  6 /mod  swap 0<> -  0
 9    ?DO  2dup 6 + min  over  I 1+  pageprint  6 +  LOOP
10    2drop  printersem unlock ;
11
12 : printall  ( -- )   0 capacity 1- pthru ;
13
14
15
```

```
  PRINTER.SCR Scr 13 Dr 0
 0 \ Printing with Shadows                                      bp 12oct86
 1
 2 : document  ( first last -- )
 3    printersem lock   output push   print
 4    1+  capacity 2/ umin  swap  2dup -  3 /mod  swap 0<> -  0
 5    ?DO  2dup 3+ min  over  I 1+  shadowprint  3+  LOOP
 6    2drop   printersem unlock ;
 7
 8 : listing  ( -- )   0 capacity 2/ 1- document ;
 9
10
11
12
13
14
15
```

```
  PRINTER.SCR Scr 14 Dr 0
 0 \ Printerspool                                               14oct86we
 1
 2 \needs Task        \\
 3
 4 $100 $200 Task spooler
 5
 6 : spool'   ( -- )    \ reads word
 7    '  isfile@  offset @  base @   spooler  depth 1-  6 min  pass
 8    base !  offset !  isfile !  execute
 9    true abort" SPOOL' ready for next job!" stop ;
10
11
12
13
14
15
```

```
  PRINTER.SCR Scr 27 Dr 0
 0 \ Printing without Shadows                                  b22oct86we
 1
 2 Die folgenden Definitionen stellen das Benutzer-Interface dar.
 3  Daher sollen sie in FORTH gefunden werden.
 4
 5 PRINTERSEM ist ein Semaphor für das Multitasking, der den Zugang
 6  auf den Drucker für die einzelnen Tasks regelt.
 7
 8 PTHRU gibt die Screens von  from  bis  to  aus.
 9  Ausgabegerät merken und Drucker einschalten. Multitasking wird,
10  sofern es den Drucker betrifft, gesperrt.
11  Die Screens werden mit pageprint ausgegeben.
12
13
14 wie oben, jedoch wird das komplette File gedruckt.
15
```

```
  PRINTER.SCR Scr 28 Dr 0
 0 \ Printing with Shadows                                     10oct86we
 1
 2 wie pthru, aber mit Shadowscreens.
 3
 4
 5
 6
 7
 8 wie printall, aber mit Shadowscreens.
 9
10
11
12
13
14
15
```

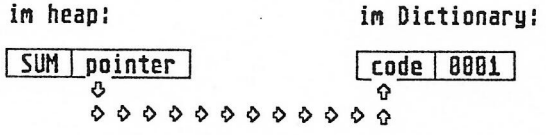
```
  PRINTER.SCR Scr 29 Dr 0
 0 \ Printerspool
 1                                                             10oct86we
 2 Falls der Multitasker nicht vorhanden ist, wird abgebrochen.
 3
 4 Der Arbeitsbereich der Task wird erzeugt.
 5
 6 Mit diesem Wort wird das Drucken im Hintergrund gestartet.
 7 Aufruf mit :
 8   spool' listing
 9   spool' printall
10   from to spool' pthru
11   from to spool' document
12 Vor (oder auch nach) dem Aufruf von spool' muß der Multitasker
13  mit multitask eingeschaltet werden.
14
15
```

5) Der Heap

Eine der ungewöhnlichen und fortschrittlichen Konzepte des volksFORTH83 besteht in der Möglichkeit, Namen von Worten zu entfernen, ohne den Rumpf zu vernichten.
Das ist insbesondere während der Kompilation nützlich, denn Namen von Worten, deren Benutzung von der Tastatur aus nicht sinnvoll wäre, tauchen am Ende der Kompilation auch nicht mehr im Dictionary auf. Man kann dem Quelltext sofort ansehen, ob ein Wort für den Gebrauch außerhalb des Programmes bestimmt ist oder nicht.
Die Namen, die entfernt wurden, verbrauchen natürlich keinen Speicherplatz mehr. Damit wird die Verwendung von mehr und längeren Namen und dadurch die Lesbarkeit gefördert.
Namen, die später eliminiert werden sollen, werden durch das Wort ¦ gekennzeichnet. Das Wort ¦ muß unmittelbar vor dem Wort stehen, das den zu eliminierenden Namen erzeugt. Der so erzeugte Name wird in einem Speicherbereich abgelegt, der Heap heißt. Der Heap kann später mit dem Wort CLEAR gelöscht werden. Dann sind natürlich auch alle Namen, die sich im Heap befanden, verschwunden.

Beispiel:
```
¦ Variable sum
1 sum !
```

ergibt :



Es werden weitere Worte definiert und dann CLEAR ausgeführt:

```
: clearsum      ( -- )     0 sum ! ;
: add           ( n -- )   sum +! ;
: show          ( -- )     sum @ . ;
clear
```

liefert die Worte CLEARSUM ADD SHOW , während der Name SUM durch CLEAR entfernt wurde; das Codefeld und der Wert 0001 existieren jedoch noch. (Das Beispiel soll eine Art Taschenrechner darstellen.)

Man kann den Heap auch dazu "mißbrauchen", Code, der nur zeitweilig benötigt wird, nachher wieder zu entfernen. Der Assembler wird auf diese Art geladen, so daß er nach Fertigstellen der Applikation mit CLEAR wieder entfernt werden kann und keinen Platz im Speicher mehr benötigt. Schauen Sie bitte in den Quelltext des Assemblers, um zu sehen, wie man das macht.

6) Der Multitasker

Das volksFORTH besitzt einen recht einfachen, aber leistungsfähigen Multitasker. Er ermöglicht die Konstruktion von Druckerspoolern, Uhren, Zählern und anderen einfachen Tasks. Als Beispiel soll gezeigt werden, wie man einen einfachen Druckerspooler konstruiert. Dieser Spooler ist in verbesserter Form auch im Quelltext des Multitaskers enthalten, hier wird er aus didaktischen Gründen möglichst simpel gehalten.

6.1) Anwendungsbeispiel: Ein Kochrezept

Das Programm für einen Druckerspooler lautet:

```
$F0 $100  Task background
: spool    background activate   1 100 pthru stop ;
multitask   spool
```

Normalerweise würde PTHRU den Rechner "lahmlegen", bis die Screens von 1 bis 100 ausgedruckt worden sind. Bei Aufruf von SPOOL ist das nicht so; der Rechner kann sofort weitere Eingaben verarbeiten. Damit alles richtig funktioniert, muß PTHRU allerdings einige Voraussetzungen erfüllen, die dieses Kapitel erklären will.
Das Wort TASK ist ein definierendes Wort, das eine Task erzeugt. Die Task besitzt übrigens Userarea, Stack, Returnstack und Dictionary unabhängig von der sog. Konsolen- oder Main-Task. Im Beispiel ist $F0 die Länge des reservierten Speicherbereichs für Returnstack und Userarea, $100 die Länge für Stack und Dictionary, jeweils in Bytes. Der Name der Task ist in diesem Fall BACKGROUND. Die neue Task tut nichts, bis sie aufgeweckt wird. Das geschieht durch das Wort SPOOL .
MULTITASK sagt dem Rechner, daß in Zukunft womöglich noch andere Tasks außer der Konsolentask auszuführen sind. Es schaltet also den Taskwechsler ein.
Bei Ausführen von SINGLETASK wird der Taskwechsler abgeschaltet. Dann wird nur noch die gerade aktive Task ausgeführt.

Bei Ausführung von SPOOL geschieht nun folgendes:

Die Task BACKGROUND wird aufgeweckt und ihr wird der Code hinter ACTIVATE (nämlich 1 100 PTHRU STOP) zur Ausführung übergeben. Damit ist die Ausführung von SPOOL beendet, es können jetzt andere Worte eingetippt werden. Die Task jedoch führt unverdrossen 1 100 PTHRU aus, bis sie damit fertig ist. Dann stößt sie auf STOP und hält an. Man sagt, die Task schläft.
Will man die Task während des Druckvorganges anhalten, z.B. um Papier nachzufüllen, so tippt man BACKGROUND SLEEP ein. Dann wird BACKGROUND vom Taskwechsler übergangen. Soll es weitergehen, so tippt man BACKGROUND WAKE ein.

Häufig möchte man erst bei Aufruf von SPOOL den Bereich als Argument angeben, der ausgedruckt werden soll. Das geht wie folgt:

```
: newspool  ( from to -- )
            2 background pass   pthru  stop ;
```

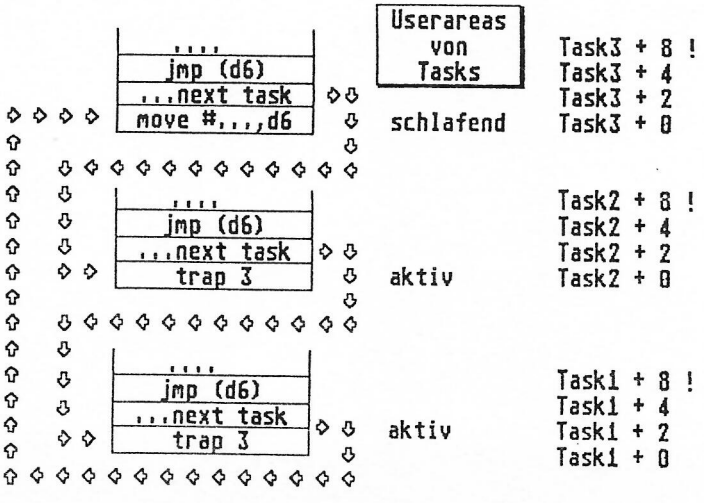
Die Phrase 2 BACKGROUND PASS funktioniert ähnlich wie BACKGROUND ACTIVATE , jedoch werden der Task auf dem Stack zusätzlich die beiden obersten Werte (hier from und to) übergeben. Um die Screens 1 bis 100 auszudrucken, tippt man jetzt ein:

```
1 100 newspool
```

Es ist klar, daß BACKGROUND ACTIVATE gerade der Phrase 0 BACKGROUND PASS entspricht.

6.2) Implementation

Der Unterschied dieses Multitaskers zu herkömmlichen liegt in seiner kooperativen Struktur begründet. Damit ist gemeint, daß jede Task explizit die Kontrolle über den Rechner und die Ein/Ausgabegeräte aufgeben und damit für andere Tasks verfügbar machen muß. Jede Task kann aber selbst "wählen", wann das geschieht. Es ist klar, daß das oft genug geschehen muß, damit alle Tasks ihre Aufgaben wahrnehmen können.
Die Kontrolle über den Rechner wird durch das Wort PAUSE aufgegeben. PAUSE führt den Code aus, der den gegenwärtigen Zustand der gerade aktiven Task rettet und die Kontrolle des Rechners an den Taskwechsler übergibt. Der Zustand einer Task besteht aus den Werten von Interpreterpointer (IP) , Returnstackpointer (RP) und des Stackpointers (SP).
Der Taskwechsler besteht aus einer geschlossenen Schleife. Jede Task enthält einen Maschinencodesprung auf die nächste Task, gefolgt von der Aufweckprozedur. Beim volksFORTH83 für den Atari besteht dieser Sprung aus zwei Befehlen, nämlich "move.w #next task,d6" und "jmp 0(FP,d6)". Zunächst wird also die Adresse (vom Anfang des Forthsystems aus gerechnet) geladen, durch den Sprungbefehl wird diese Adresse auf den Anfang des Speichers umgerechnet und dann auf diese Adresse gesprungen (mehr zu dieser Umrechnung finden Sie im Kapitel über den Assembler). Dort befindet sich die entsprechende Instruktion der nächsten Task. Ist die Task gestoppt, so wird dort ebenfalls ein Maschinencodesprung zur nächsten Task ausgeführt. Ist die Task dagegen aktiv, so ist der Sprung durch den Trap #3 ersetzt worden, der die Aufweckprozedur auslöst. Diese Prozedur lädt den Zustand der Task (bestehend aus SP , RP und IP) und setzt den Userpointer (UP), so daß er auf diese Task zeigt (siehe Bild).

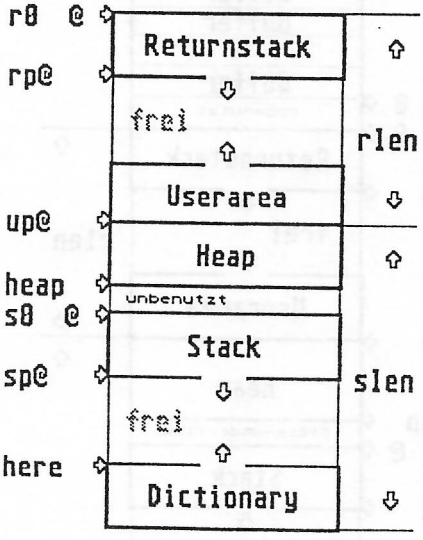


SINGLETASK ändert nun PAUSE so, daß überhaupt kein Taskwechsel stattfindet, wenn PAUSE aufgerufen wird. Das ist in dem Fall sinnvoll, wenn nur eine Task existiert, nämlich die Konsolentask, die beim Kaltstart des Systems "erzeugt" wurde. Dann würde PAUSE unnötig Zeit damit verbrauchen, einen Taskwechsel auszuführen, der sowieso wieder auf dieselbe Task führt. STOP entspricht PAUSE , jedoch mit dem Unterschied, daß die leere Anweisung durch einen Sprungbefehl ersetzt wird. Das System unterstützt den Multitasker, indem es während vieler Ein/Ausgabeoperationen wie KEY , TYPE und BLOCK usw. PAUSE ausführt. Häufig reicht das schon aus, damit eine Task (z.B. der Druckerspooler) gleichmäßig arbeitet.

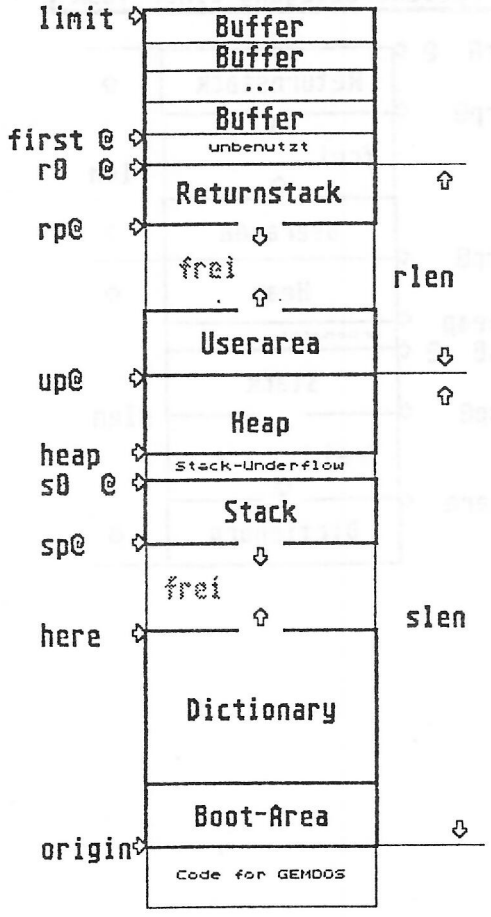
Tasks werden im Dictionary der Konsolentask erzeugt. Jede besitzt ihre eigene Userarea mit einer Kopie der Uservariablen. Die Implementation des Systems wird aber durch die Einchränkung vereinfacht, daß nur die Konsolentask Eingabetext interpretieren bzw. kompilieren kann. Es gibt z.B. nur eine Suchreihenfolge, die im Prinzip für alle Tasks gilt. Da aber nur die Konsolentask von ihr Gebrauch macht, ist das nicht weiter störend.
Es ist übrigens möglich, aktive Tasks mit FORGET usw. zu vergessen. Das ist eine Eigenschaft, die nicht viele Systeme aufweisen! Allerdings geht das manchmal auch schief... Nämlich dann, wenn die vergessene Task einen "Semaphor" (s.u.) besaß. Der wird beim Vergessen nämlich nicht freigegeben und damit ist das zugehörige Gerät blockiert.

Schließlich sollte man noch erwähnen, daß beim Ausführen eines Tasknamens der Beginn der Userarea dieser Task auf dem Stack hinterlassen wird.

Speicherbelegung einer Task



Memory Map des volksFORTH83



6.3) Semaphore und "Lock"

Ein Problem, daß bisher noch nicht erwähnt wurde, ist: Was passiert, wenn zwei Tasks gleichzeitig drucken (oder Daten von der Diskette lesen) wollen?
Es ist klar: Um ein Durcheinander oder Fehler zu vermeiden, darf das immer nur eine Task zur Zeit.
Programmtechnisch wird das Problem durch "Semaphore" gelöst:

```
Create disp 0 ,
: newtype   disp lock  type  disp unlock;
```

Der Effekt ist der folgende:
Wenn zwei Tasks gleichzeitig NEWTYPE ausführen, so kann doch nur eine zur Zeit TYPE ausführen, unabhängig davon, wie viele PAUSE in TYPE enthalten sind. Die Phrase DISP LOCK schaltet nämlich hinter der ersten Task, die sie ausführt, die "Ampel auf rot" und läßt keine andere Task durch. Die anderen machen solange PAUSE , bis die erste Task die Ampel mit DISP UNLOCK wieder auf grün umschaltet. Dann kann eine (!) andere Task die Ampel hinter sich umschalten usw. .
Übrigens wird die Task, die die Ampel auf rot schaltete, bei DISP LOCK nicht aufgehalten, sondern durchgelassen. Das ist notwendig, da ja TYPE ebenfalls DISP LOCK enthalten könnte (Im obigen Beispiel natürlich nicht, aber es ist denkbar) .
Die Implementation sieht nun folgendermaßen aus:

(Man muß sich noch vor Augen halten, daß jede Task eindeutig durch den Anfang ihrer Userarea identifizierbar ist.)

DISP ist ein sog. Semaphor; er muß den Anfangswert 0 haben!

LOCK schaut sich nun den Semaphor an:
Ist er Null, so wird die gerade aktive Task (bzw. der Anfang ihrer Userarea) in den Semaphor eingetragen und die Task darf weitermarschieren.
Ist der Wert des Semaphors gerade die aktive Task, so darf sie natürlich auch weiter.
Wenn aber der Wert des Semaphors von dem Anfang der Userarea der aktiven Task abweicht, dann ist gerade eine andere Task hinter der Ampel aktiv und die Task muß solange PAUSE machen, bis die Ampel wieder grün, d.h. der Semaphor null ist.

UNLOCK muß nun nichts anderes mehr tun, als den Wert des Semaphors wieder auf Null setzen.

BLOCK und BUFFER sind übrigens auf diese Weise für die Benutzung durch mehrere Tasks gesichert: Es kann immer nur eine Task das Laden von Blöcken von der Diskette veranlassen.
Ob TYPE , EMIT usw. ebenfalls gesichert sind, hängt von der Implementation ab.

6.4) Eine Bemerkung bzgl. BLOCK und anderer Dinge

Wie man dem Glossar entnehmen kann, ist immer nur die Adresse des zuletzt mit BLOCK oder BUFFER angeforderten Blockpuffers gültig, d.h. ältere Blöcke sind, je nach der Zahl der Blockpuffer, womöglich schon wieder auf die Diskette ausgelagert worden. Auf der sicheren Seite liegt man, wenn man sich vorstellt, daß nur ein Blockpuffer im gesamten System existiert.
Nun kann jede Task BLOCK ausführen und damit anderen Tasks die Blöcke "unter den Füßen" wegnehmen. Daher sollte man nicht die Adresse eines Blocks nach einem Wort, das PAUSE ausführt, weiter benutzen, sondern lieber neu mit BLOCK anfordern. Ein Beispiel:

```
: .line  ( block -- )
   block  c/l  bounds  DO  I c@ emit  LOOP ;
```

ist FALSCH, denn nach EMIT stimmt der Adressbereich, den der Schleifenindex überstreicht, womöglich gar nicht mehr.

```
: .line  ( block -- )
   c/l 0  DO  dup block  I +  c@  emit  LOOP  drop ;
```

ist RICHTIG, denn es wird nur die Nummer des Blocks, nicht die Adresse seines Puffers aufbewahrt.

```
: .line  ( block -- )    block  c/l  type ;
```

ist FALSCH, da TYPE ja EMIT wiederholt ausführen kann und somit die von BLOCK gelieferte Adresse in TYPE ungültig wird.

```
: >type  ( adr len -- ) >r  pad  r@  cmove  pad r> type ;
: .line  ( block -- )    block  c/l  >type ;
```

ist RICHTIG, denn PAD ist für jeden Task verschieden.

7) Debugging - Techniken

Fehlersuche ist in allen Programmiersprachen die aufwendigste Aufgabe des Programmierers.
Für verschiedene Programme sind in der Regel auch verschiedene Hilfsmittel erforderlich. Daher kann dieses Kapitel die Fehlersuche nicht erschöpfend behandeln. Da aber Anfänger häufig typische Fehler machen, kann man gerade für diese Gruppe brauchbare Hilfsmittel angeben.

7.1) Voraussetzungen für die Fehlersuche

Voraussetzung für die Fehlersuche ist immer ein übersichtliches und verständliches Programm. In Forth bedeutet das :

-) suggestive und prägnante Namen für Worte

-) starke Faktorisierung, d.h. sinnvoll zusammengehörende Teile eines Wortes sind zu einem eigenen Wort zusammengefaßt. Worte sollten durchschnittlich nicht länger als 2 - 3 Zeilen lang sein !

-) Übergabe von Parametern auf dem Stack statt in Variablen, überall wo das möglich ist.

Guter Stil in Forth ist nicht schwer, erleichtert aber sehr die Fehlersuche. Ein Buch, das auch diesen Aspekt behandelt, sei unbedingt empfohlen :
"Thinking Forth" von Leo Brodie, Prentice Hall 1984.

Sind diese Bedingungen erfüllt, ist es meist möglich, die Worte interaktiv zu testen. Damit ist gemeint, daß man bestimmte Parameter auf dem Stack versammelt und anschließend das zu testende Wort aufruft. Anhand der abgelieferten Werte auf dem Stack kann man dann beurteilen, ob das Wort korrekt arbeitet.
Sinnvollerweise testet man natürlich die zuerst definierten Worte auch zuerst, denn ein Wort, das fehlerhafte Worte aufruft, funktioniert natürlich nicht korrekt.
Wenn nun ein Wort auf diese Weise als fehlerhaft identifiziert wurde, muß man das Geschehen innerhalb des Wortes verfolgen. Das geschieht meist durch "tracen".

7.2) Der Tracer

Angenommen, Sie wollen folgendes Wort auf Fehler untersuchen: (bitte tippen Sie ein, alle nötigen Eingaben sind fett und alle Ausgaben des volksFORTH83 sind unterstrichen dargestellt.)

```
: -trailing  ( addr1 n1 -- addr1 n2 )  compiling
   2dup bounds  ?DO  2dup + 1-  c@ bl = compiling
   IF  LEAVE  THEN  1-  LOOP ;  ok
```

Die Funktion dieses Wortes können Sie, wenn sie Ihnen unklar ist, dem Glossar entnehmen. Zum Testen des Wortes wird ein String benötigt. Sie definieren:

```
Create teststring ," Dies ist ein Test        "  ok
```

wobei Sie bitte absichtlich einige zusätzliche Leerzeichen eingefügt haben. Sie geben nun ein :

```
teststring count .s 16 7E39  ok
-trailing .s 16 7E39  ok
```

und stellen zu Ihrem Erstaunen fest, daß -TRAILING kein einziges Leerzeichen abgeschnitten hat. (Spätestens jetzt sollten Sie am Rechner sitzen und den Tracer laden, wenn er noch nicht im System vorhanden ist. Prüfen Sie, ob es das Wort DEBUG im FORTH Vokabular gibt, dann ist der Tracer vorhanden. Der Tracer gehört zu den sogenannten Tools. Die Quelltexte finden Sie auf Ihrer Diskette).

Mit dem Tracer können Sie Worte, die mit dem : definiert wurden, schrittweise testen. Um den Tracer zu benutzen, geben Sie folgendes ein:

```
debug <name1>  ok
```

Hierbei ist <name1> das zu tracende Wort. Zunächst geschieht noch gar nichts. DEBUG hat nur den Tracer "scharf gemacht". Geben Sie nun eine Sequenz ein, die <name1> enthält, unterbricht der Tracer die Ausführung, wenn er auf <name1> stößt. Es erscheint folgendes Bild.

```
addr1 addr2 <name2>                    (Werte)
```

Hierbei ist addr1 eine Adresse im Parameterfeld von <name1>, nämlich die, in der addr2 steht. addr2 ist die Kompilations-Adresse von <name2>. <name2> ist das Wort, das als nächstes ausgeführt werden soll. (Werte) sind die Werte, die gerade auf dem Stack liegen.

Bleiben wir bei unserem Beispiel, Sie geben ein:

```
debug -trailing  ok
```

Es geschieht zunächst nichts.

Nun versuchen Sie es wieder mit der Sequenz

```
teststring count .s 16 7E39  ok
-trailing
```

und erhalten folgendes Bild:

```
7E04  536  2DUP                    16 7E39
```

16 7E39 ist der Stackinhalt, wie er von TESTSTRING COUNT geliefert wurde, nämlich Adresse und Länge des Strings TESTSTRING .

Natürlich können die Zahlen bei Ihnen anders aussehen, je nachdem, wohin TESTSTRING und -TRAILING kompiliert wurde. 536 ist die Kompilationsadresse von 2DUP , 7E04 die Position von 2DUP in -TRAILING . (Auch diese Adressen können sich geändert haben!) Diese Zahlen werden mit ausgegeben, so daß auch im Falle mehrerer Worte mit gleichem Namen eine Identifizierung möglich ist.

Drücken Sie jetzt so lange <Return>, bis OK erscheint. Insgesamt wird dabei folgendes ausgegeben :

```
debug -trailing   teststring count .s 16 7E39  ok
-trailing
7E04  536  2DUP                    16 7E39
7E06  954  BOUNDS                  16 7E39 16 7E39
7E08  92C  (?DO                    7E39 7E4F 16 7E39
7E0C  536  2DUP                    16 7E39
7E0E  546  +                       16 7E39 16 7E39
7E10  64C  1-                      7E4F 16 7E39
7E12  34E  C@                      7E4E 16 7E39
7E14 215C  BL                      20 16 7E39
7E16  816  =                       20 20 16 7E39
7E18  A0C  ?BRANCH                 FFFF 16 7E39
7E1C  BE2  LEAVE                   16 7E39
7E24  2F4  UNNEST                  16 7E39    ok
```

Sehen wir uns die Ausgabe nun etwas genauer an. Bei den ersten beiden Zeilen wächst der Wert ganz links immer um 2. Es ist der Inhalt des Instructionpointers (IP), der immer auf die nächste auszuführende Adresse zeigt. Der Inhalt dieser Adresse ist jeweils eine Kompilationsadresse (536 bei 2DUP usw.). Jede Kompilationsadresse benötigt zwei Bytes, daher muß der IP immer um 2 erhöht werden.

Immer? Nein, denn schon die nächste Zeile zeigt eine Ausnahme. Das Wort (?DO erhöht den IP um 4 ! Woher kommt eigentlich (?DO , in der Definition von -TRAILING stand doch nur ?DO . (?DO ist ein von ?DO kompiliertes Wort, das zusätzlich zur Kompilationsadresse noch einen 16-Bit-Wert benötigt, nämlich für den Sprungoffset hinter LOOP, wenn die Schleife beendet ist. Zwei ähnliche Fälle treten noch auf. Das IF aus dem Quelltext hat ein ?BRANCH kompiliert. Es wird gesprungen, wenn der oberste Stackwert FALSE (= 0) ist. Auch ?BRANCH benötigt einen zusätzlichen 16-Bit-Wert für den Sprungoffset.

Nach LEAVE geht es hinter LOOP weiter, es wird UNNEST ausgeführt, das vom ; in -TRAILING kompiliert wurde und das gleiche wie EXIT bewirkt, und damit ist das Wort -TRAILING auch beendet. Das hier gelistete Wort UNNEST ist nicht zu verwechseln mit dem UNNEST des Tracers (siehe unten).

Wo liegt nun der Fehler in unserer Definition von -TRAILING ? Bevor Sie weiterlesen, sollten Sie die Fehlerbeschreibung, den Tracelauf und Ihre Vorstellung von der korrekten Arbeitsweise des Wortes noch einmal unter die Lupe nehmen.

Der Stack ist vor und nach -TRAILING gleich geblieben, die Länge des Strings also nicht verändert worden. Offensichtlich wird die Schleife gleich beim ersten Mal verlassen, obwohl das letzte Zeichen des Textes ein Leerzeichen war. Die Schleife hätte also eigentlich mit dem vorletzten Zeichen weiter machen müssen. Mit anderen Worten:

Die Abbruchbedingung in der Schleife ist falsch. Sie ist genau verkehrt herum gewählt. Ersetzt man = durch = NOT oder - , so funktioniert das Wort korrekt. Überlegen Sie bitte, warum - statt = NOT eingesetzt werden kann. (Tip: der IF -Zweig wird nicht ausgeführt, wenn der oberste Stackwert FALSE (also = 0) ist.)

Der volksFORTH83-Tracer gestattet es, jederzeit Befehle einzugeben, die vor dem Abarbeiten des nächsten Trace-Kommandos ausgeführt werden. Damit kann man z. B. Stack-Werte verändern oder das Tracen abbrechen. Ändern Sie probehalber beim nächsten Trace-Lauf von -TRAILING das TRUE-Flag ($FFFF) auf dem Stack vor der Ausführung von ?BRANCH durch Eingabe von NOT auf 0 und verfolgen Sie den weiteren Trace-Lauf. Sie werden bemerken, daß die LOOP ein zweites Mal durchlaufen wird.

Wollen Sie das Tracen und die weitere Ausführung des getraceten Wortes abbrechen, so geben Sie **RESTART** ein. RESTART führt einen Warm-Start des FORTH-Systems aus und schaltet den Tracer ab. RESTART ist auch die Katastrophen-Notbremse, die man einsetzt, wenn man sieht, daß das System mit dem nächsten Befehl zwangsläufig im Computer-Nirwana entschwinden wird.

Nützlich ist auch die Möglichkeit, das Wort, das als nächstes

zur Ausführung ansteht, solange zu tracen, bis es ins aufrufende Wort zurückkehrt. Dafür ist das Wort **NEST** vorgesehen. Wollen Sie also wissen, was BOUNDS macht, so geben Sie bitte, wenn BOUNDS als nächstes auszuführendes Wort angezeigt wird, NEST ein und Sie erhalten dann:

```
7E04  536  2DUP                    16 7E39
7E06  954  BOUNDS                  16 7E39 16 7E39  nest
 956   44C  OVER                    16 7E39 16 7E39
 958   546  +                       7E39 16 7E39 16 7E39
 95A   40A  SWAP                    7E4F 7E39 16 7E39
 95C   2F4  UNNEST                  7E39 7E4F 16 7E39
7E08  92C  (?DO                     7E39 7E4F 16 7E39
...
```

Beachten Sie bitte, daß die Zeilen jetzt eingerückt dargestellt werden, bis der Tracer automatisch in das aufrufende Wort zurückkehrt. Der Gebrauch von NEST ist nur dadurch eingeschränkt, daß sich einige Worte, die den Return-Stack manipulieren, mit NEST nicht tracen lassen, da der Tracer selbst Gebrauch vom Return-Stack macht. Auf solche Worte muß man den Tracer mit DEBUG ansetzen.

Wollen Sie das Tracen eines Wortes beenden, ohne die Ausführung des Wortes abzubrechen, so benutzen Sie **UNNEST** . (Ist der Tracer geladen, so kommen Sie an das tief im System steckende UNNEST, einem Synonym für EXIT, das ausschließlich vom ; kompiliert wird, nicht mehr heran und benutzen statt dessen das Tracer-UNNEST, das Sie eine Ebene im Trace-Lauf zurückbringt.)

Manchmal hat man in einem Wort vorkommende Schleifen beim ersten Durchlauf als korrekt erkannt und möchte diese nicht weiter tracen. Das kann sowohl bei DO...LOOPs der Fall sein als auch bei Konstruktionen mit BEGIN...WHILE...REPEAT oder BEGIN...UNTIL. In diesen Fällen gibt man am Ende der Schleife das Wort **ENDLOOP** ein. Die Schleife wird dann in Echtzeit abgearbeitet und der Tracer meldet sich erst wieder, wenn er nach dem Wort angekommen ist, bei dem ENDLOOP eingegeben wurde.

Haben Sie den Fehler gefunden und wollen deshalb nicht mehr tracen, so müssen Sie nach dem Ende des Tracens END-TRACE oder jederzeit RESTART eingeben, ansonsten bleibt der Tracer "scharf", was zu merkwürdigen Erscheinungen führen kann; außerdem verringert sich bei eingeschaltetem Tracer die Laufzeit des Systems.

Der Tracer läßt sich auch mit dem Wort **TRACE'** starten, das seinerseits ein zu tracendes Forth-Wort erwartet. Sie könnten also auch im obigen Beispiel eingeben:

```
teststring count trace' -trailing
```

und hätten damit dieselbe Wirkung erzielt. TRACE' schaltet aber im Gegensatz zu DEBUG nach Durchlauf des Wortes den Tracer automatisch mit END-TRACE wieder ab.

Beachten Sie bitte auch, daß die Worte DEBUG und TRACE' das Vokabular TOOLS mit in die Suchreihenfolge aufnehmen. Sie sollten also nach - hoffentlich erfolgreichem - Tracelauf die Suchordnung wieder umschalten.

Wenn man sich eingearbeitet hat, ist der Tracer ein wirklich verblüffendes Werkzeug, mit dem man sehr viele Fehler schnell finden kann. Er ist gleichsam ein Mikroskop, mit dem man sehr tief ins Innere von Forth schauen kann.

7.3) Stacksicherheit

Anfänger neigen häufig dazu, Fehler bei der Stackmanipulation zu machen. Erschwerend kommt häufig hinzu, daß sie viel zu lange Worte schreiben, in denen es dann von unübersichtlichen Stackmanipulationen nur so wimmelt. Es gibt einige Worte, die sehr einfach sind und Fehler bei der Stackmanipulation früh erkennen helfen. Denn leider führen schwerwiegende Stackfehler zu "mysteriösen" Systemcrashs.

In Schleifen führt ein nicht ausgeglichener Stack oft zu solchen Fehlern. Während der Testphase eines Programms oder Wortes sollte man daher bei jedem Schleifendurchlauf prüfen, ob der Stack evtl. über- oder leerläuft.
Das geschieht durch Eintippen von :

```
: LOOP   compile ?stack  [compile] LOOP ; immediate restrict
: +LOOP  compile ?stack  [compile] +LOOP ; immediate restrict
: UNTIL  compile ?stack  [compile] UNTIL ; immediate restrict
: REPEAT compile ?stack  [compile] REPEAT ; immediate restrict
: :      : compile ?stack ;
```

Versuchen Sie ruhig, herauszufinden wie die letzte Definition funktioniert. Es ist nicht kompliziert.
Durch diese Worte bekommt man sehr schnell mitgeteilt, wann ein Fehler auftrat. Es erscheint dann die Fehlermeldung :

```
<name> stack full
```

wobei <name> der zuletzt vom Terminal eingegebene Name ist.
Wenn man nun überhaupt keine Ahnung hat, wo der Fehler auftrat, so gebe man ein :

```
: unravel   rdrop rdrop rdrop \ delete errorhandler-nest
      cr  ." trace dump on  abort is :"  cr
      BEGIN  rp@  r0 @ -      \ until stack empty
      WHILE  r> dup  8 u.r  space
             2- @ >name  .name  cr  REPEAT
      (error ;
' unravel  errorhandler !
```

Sie bekommen dann bei Eingabe von

```
1 2 0 */
```

ungefähr folgenden Ausdruck :

```
trace dump on  abort is:
    4678 M/MOD
    4692 */MOD
    9248 EXECUTE
   10060 INTERPRET
   10104 'QUIT/ division overflow
```

'QUIT INTERPRET und EXECUTE rühren vom Textinterpreter her. Interessant wird es bei */MOD . Wir wissen nämlich, daß */MOD von */ aufgerufen wird. */MOD ruft nun wieder M/MOD auf, in M/MOD gehts weiter nach UM/MOD . Dieses Wort ist in Code geschrieben und "verursachte" den Fehler, indem es eine Division durch Null ausführte.

Nicht immer treten Fehler in Schleifen auf. Es kann auch der Fall sein, daß ein Wort zu wenig Argumente auf dem Stack vorfindet, weniger nämlich, als Sie für dieses Wort vorgesehen haben. Diesen Fall sichert ARGUMENTS .
Die Definition dieses Wortes ist:

```
: ARGUMENTS   ( n -- )
    depth 1- <  abort" not enough arguments" ;
```

Es wird folgendermaßen benutzt:

```
: -trailing  ( adr len -- )  2 arguments   ... ;
```

wobei die drei Punkte den Rest des Quelltextes andeuten sollen. Findet -TRAILING nun weniger als zwei Werte auf dem Stack vor, so wird eine Fehlermeldung ausgegeben. Natürlich kann man damit nicht prüfen, ob die Werte auf dem Stack wirklich für -TRAILING bestimmt waren.

Sind Sie als Programmierer sicher, daß an einer bestimmten Stelle im Programm eine bestimmte Anzahl von Werten auf dem Stack liegt, so können Sie das ebenfalls sicherstellen:

```
: is-depth  ( n -- )  depth 1-  - abort" wrong depth" ;
```

IS-DEPTH bricht das Programm ab, wenn die Zahl der Werte auf dem Stack nicht n ist, wobei n natürlich nicht mitgezählt wird. Es wird analog zu ARGUMENTS benutzt.
Mit diesen Worten lassen sich Stackfehler oft feststellen und lokalisieren.

7.4) Aufrufgeschichte

Möchte man wissen, was geschah, bevor ein Fehler auftrat und nicht nur, wo er auftrat (denn nur diese Information liefert UNRAVEL), so kann man einen modifizierten Tracer verwenden, bei dem man nicht nach jeder Zeile <RETURN> drücken muß:

```
: : :
    Does>  cr  rdepth 2* spaces
           dup 2- >name .name  >r ;
```

Hierbei wird ein neues Wort mit dem Namen : definiert, das zunächst das alte Wort : aufruft und so ein Wort im Dictionary erzeugt. Das Laufzeitverhalten dieses Wortes wird aber so geändert, daß es sich jedesmal wieder ausdruckt, wenn es aufgerufen wird. Alle Worte, die nach der Redefinition (so nennt man das erneute Definieren eines schon bekannten Wortes) des : definiert wurden, weisen dieses Verhalten auf.

Beispiel :

```
: / / ;
: rechne ( -- n)  1 2 3 / / ;

RECHNE
  /
  /   RECHNE  division overflow
```

Wir sehen also, daß erst bei der zweiten Division der Fehler auftrat. Das ist auch logisch, denn 2 3 / ergibt 0 .

Sie sind sicher in der Lage, die Grundidee dieses zweiten Tracers zu verfeinern. Ideen wären z.B. :

Ausgabe der Werte auf dem Stack bei Aufruf eines Wortes

Die Möglichkeit, Teile eines Tracelaufs komfortabel zu unterdrücken.

7.5) Speicherdump

Ein Dump des Speichers benötigt man beim Programmieren sehr oft, mindestens dann, wenn man eigene Datenstrukturen anschauen will. Oft ist es dann hinderlich, eigene Worte zur (womöglich gar formatierten) Ausgabe der Datenstrukturen schreiben zu müssen. In diesen Fällen benötigt man ein Wort, das einen Speicherdump ausgibt. Das volksFORTH besitzt zwei Worte zum Dumpen von Speicherblöcken sowie einen Dekompiler, der auch für Datenstrukturen verwendet werden kann.

```
DUMP    ( addr n -- )
```

Ab addr werden n Bytes formatiert ausgegeben. Dabei steht am Anfang einer Zeile die Adresse (abgerundet auf das nächste Vielfache von $10) , dann folgen 16 Byte, in der Mitte zur besseren Übersicht getrennt und dann die Ascii-Darstellung. Dabei werden nur Zeichen im Bereich zwischen $20 und $7F ausgegeben, alle anderen Werte werden als Punkt angezeigt. Die Ausgabe läßt sich jederzeit mit einer beliebigen Taste unterbrechen oder mit <Esc> abbrechen. Mit PRINT läßt sich die Ausgabe auf einen Drucker umleiten. Beispiel:

```
print  pad 40 dump  display
```

Das abschließende DISPLAY sorgt dafür, daß wieder der Bildschirm als Ausgabegerät gesetzt wird.

Möchte man Speicherbereich außerhalb des FORTH-Systems dumpen, gibt es dafür das Wort

```
LDUMP    ( laddr n -- )
```

laddr muß eine - doppelt lange - absolute Speicheradresse sein, n gibt wie oben die Anzahl der Bytes an. Da das FORTH-System immer im sogenannten Supervisormodus arbeitet, können alle Speicherbereiche einschließlich der Trap-Vektoren usw. angezeigt werden. Die Ausgabe auf einen Drucker geschieht genau so wie oben beschrieben.

7.6) Der Dekompiler

Ein Dekompiler gehört so zu sagen zum guten Ton eines FORTH-Systems, war er bisher doch die einzige Möglichkeit, wenigstens ungefähr den Aufbau eines Systems zu durchschauen. Bei volks-FORTH-83 ist das anders, und zwar aus zwei Gründen:

-) Sie haben sämtliche Quelltexte vorliegen, und es gibt die VIEW-Funktion. Letztere ist normalerweise sinn-voller als der beste Dekompiler, da kein Dekompiler in der Lage ist, z.B. Stackkommentare zu rekonstruieren.

-) Der Tracer ist beim Debugging sehr viel hilfreicher als ein Dekompiler, da er auch die Verarbeitung von Stackwerten erkennen läßt. Damit sind Fehler leicht aufzufinden.

Dennoch gibt es natürlich auch im volksFORTH einen Dekompiler, allerdings in einfacher, von Hand zu bedienender, Form. Er befindet sich, wie der Tracer, im Vokabular TOOLS. Folgende Worte stehen zur Verfügung.

N (name) (addr -- addr')
druckt den Namen des bei addr kompilierten Wortes aus und setzt addr auf das nächste Wort.

K (konstante) (addr -- addr')
wird nach LIT benutzt und druckt den Inhalt von addr als Zahl aus. Es wird also nicht versucht, den Inhalt, wie bei N, als Forthwort zu interpretieren.

S (string) (addr -- addr')
wird nach (ABORT" (" (." und allen anderen Worten benutzt, auf die ein String folgt. Der String wird ausgedruckt und addr entsprechend erhöht, sodaß sie hinter den String zeigt.

C (character) (addr -- addr')
druckt den Inhalt von addr als Ascii-Zeichen aus und geht ein Byte weiter. Damit kann man eigene Datenstrukturen ansehen.

B (branch) (addr -- addr')
wird nach BRANCH oder ?BRANCH benutzt und druckt den Inhalt einer Adresse als Sprungoffset und Sprungziel aus.

D (dump) (addr n -- addr')
Dumped n Bytes. Wird benutzt, um selbstdefinierte Daten-strukturen anzusehen. (s.a. DUMP und LDUMP)

Sehen wir uns nun ein Beispiel zur Benutzung des Dekompilers an. Geben Sie bitte folgende Definition ein:

```
: test   ( n -- )
   12 = IF  cr ." Die Zahl ist zwölf !"  THEN ;
```

Rufen Sie das Vokabular TOOLS - durch Nennen seines Namens auf und ermitteln Sie die Adresse des ersten in TEST kompilierten Wortes:

```
' test >body
```

Jetzt können Sie TEST nach folgendem Muster dekompilieren

```
n A988:    B54  LIT  ok
k A98A:     12  ok
n A98C:    C52  =  ok
n A98E:    E74  ?BRANCH  ok
b A990:     1A  A9AA  ok
n A992:   32BC  CR  ok
n A994:   1776  (."  ok
s A996:     12    Die Zahl ist zwölf  ok
n A99A:    416  UNNEST
drop
```

Die erste Adresse ist die, an der im Wort TEST die anderen Worte kompiliert sind. Die zweite ist jeweils die Kompilationsadresse der Worte, danach folgen die sonstigen Ausgaben des Dekompilers.

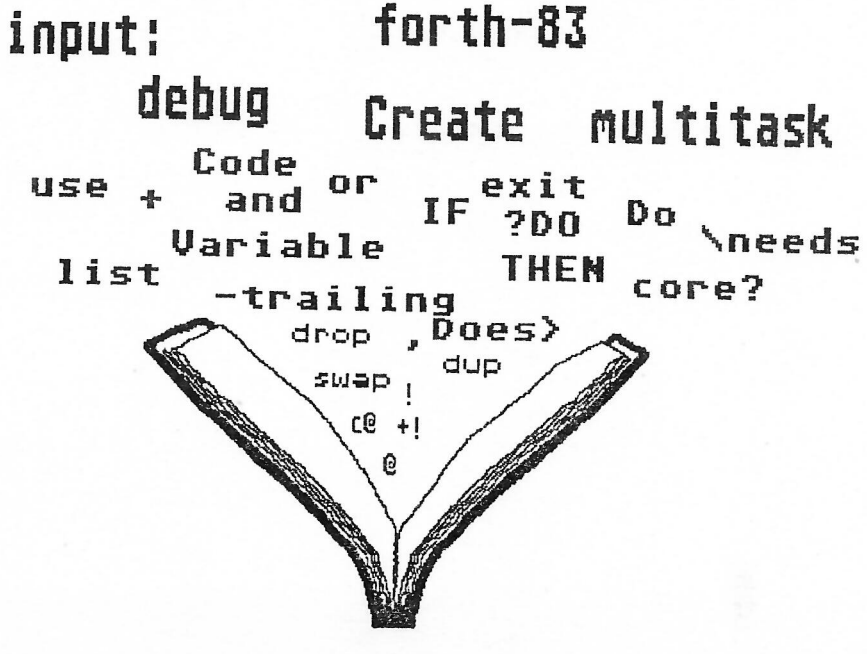
Probieren Sie dieses Beispiel auch mit dem Tracer aus:

```
20 trace' test
```

und achten Sie auf die Unterschiede. Sie werden sehen, daß der Tracer aussagefähiger und dazu noch einfacher zu bedienen ist.

Wenn Sie sich die Ratschläge und Tips zu Herzen genommen haben und noch etwas den Umgang mit den Hilfsmitteln üben, werden Sie sich sicher nicht mehr vorstellen können, wie Sie jemals in anderen Sprachen ohne diese Hilfsmittel ausgekommen sind. Bedenken Sie bitte auch : Diese Mittel sind kein Heiligtum ; oft wird Sie eine Modifikation schneller zum Ziel führen. Scheuen Sie sich nicht, sich vor dem Bearbeiten einer umfangreichen Aufgabe erst die geeigneten Hilfsmittel zu verschaffen.

Glossar


input:
forth-83
debug
Create
multitask
Code
use
+
and
or
exit
IF
?DO
Do
\needs
Variable
list
THEN
core?
-trailing
drop
,
Does>
dup
swap
!
c@
+!
@

Notation

Die Worte des volksFORTH83 sind in Wortgruppen zusammengefasst. Die Worte jeder Wortgruppe sind alphabetisch sortiert. Die Wortgruppen sind nicht geordnet.

Worte werden in der in Kapitel 1 angegeben Schreibweise aufgeführt. Wortnamen im Text werden gross geschrieben.

Die Wirkung des Wortes auf den Stack wird in Klammern angegeben und zwar in folgender Form:

```
( vorher -- nachher )
```

vorher : Werte auf dem Stack vor Ausführung des Wortes
nachher : Werte auf dem Stack nach Ausführung des Wortes

In dieser Notation wird das oberste Element des Stacks immer ganz rechts geschrieben. Sofern nicht anders angegeben, beziehen sich alle Stacknotationen auf die spätere Ausführung des Wortes. Bei immediate Worten wird auch die Auswirkung des Wortes auf den Stack während der Kompilierung angegeben.

Worte werden ferner durch folgende Symbole gekennzeichnet:

C
: Dieses Wort kann nur während der Kompilation einer :-Definition benutzt werden.

I
: Dieses Wort ist ein immediate Wort, das auch im kompilierenden Zustand ausgeführt wird.

83
: Dieses Wort wird im 83-Standard definiert und muß auf allen Standardsystemen äquivalent funktionieren.

U
: Kennzeichnet eine Uservariable.

Weicht die Aussprache eines Wortes von der natürlichen englischen Aussprache ab, so wird sie in Anführungszeichen angegeben. Gelegentlich folgt auch eine deutsche Übersetzung.

Die Namen der Stackparameter folgen, sofern nicht suggestive Bezeichnungen gewählt wurden, dem nachstehendem Schema. Die Bezeichnungen können mit einer nachfolgenden Ziffer versehen sein.

Stack-not.	Zahlentyp	Wertebereich in Dezimal	minimale Feldbreite
flag	logischer Wert	0=falsch, sonst=true	16 Bit
true	logischer Wert	-1 (als Ergebnis)	16
false	logischer Wert	0	16
b	Bit	0..1	1
char	Zeichen	0..127 (0..256)	7
8b	8 beliebige Bits	nicht anwendbar	8
16b	16 beliebige Bits	nicht anwendbar	16
n	Zahl, bewertete Bits	-32768..32767	16
+n	positive Zahl	0..32767	16
u	vorzeichenlose Zahl	0..65535	16
w	Zahl, n oder u	-32768..65535	16
addr	Adresse, wie u	0..65535	16
32b	32 beliebige Bits	nicht anwendbar	32
d	doppelt genaue Zahl	-2,147,483,648... 2,147,483,647	32
+d	pos. doppelte Zahl	0..2,147,483,647	32
ud	vorzeichenlose doppelt genaue Zahl	0..4,294,967,295	32
sys	0, 1 oder mehr System-abhängige Werte	na	nicht anwendbar

Arithmetik

***** (w1 w2 -- w2) 83 " times"
Der Wert w1 wird mit w2 multipliziert. w2 sind die niederwertigen 16 Bits des Produktes. Ein Überlauf wird nicht angezeigt.

***/** (n1 n2 n3 -- n4) 83 " times-divide"
Zuerst wird n1 mit n2 multipliziert und ein 32-bit Zwischenergebnis erzeugt. n4 ist der Quotient aus dem 32-bit Zwischenergebnis und dem Divisior n3. Das Produkt von n1 mal n2 wird als 32-bit Zwischenergebnis dargestellt, um eine größere Genauigkeit gegenüber dem sonst gleichwertigen Ausdruck n1 n2 * n3 / zu erhalten. Eine Fehlerbedingung besteht, wenn der Divisor Null ist, oder der Quotient außerhalb des Intervalls (-32768.. 32767) liegt.

***/mod** (n1 n2 n3 -- n4 n5) 83 " times-divide-mod "
Zuerst wird n1 mit n2 multipliziert und ein 32-bit Zwischenergebnis erzeugt. n4 ist der Rest und n5 der Quotient aus dem 32-bit-Zwischenergebnis und dem Divisor n3. n4 hat das gleiche Vorzeichen wie n3 oder ist Null. Das Produkt von n1 mal n2 wird als 32-bit Zwischenergebnis dargestellt, um eine größere Genauigkeit gegenüber dem sonst gleichwertigen Ausdruck n1 n2 * n3 /mod zu erhalten. Eine Fehlerbedingung besteht, falls der Divisior Null ist oder der Quotient außerhalb des Intervalls (-32768...32767) liegt.

+ (w1 w2 -- w3) 83 " plus "
w1 und w2 addiert ergibt w3.

- (w1 w2 -- w3) 83 " minus "
w2 von w1 subtrahiert ergibt w3.

-1 (-- -1)
Oft benutzte Zahlenwerte wurden zu Konstanten gemacht. Definiert in der Form :

```
n Constant n
```

Dadurch wird Speicherplatz eingespart und die Ausführungszeit verkürzt. Siehe auch CONSTANT.

/ (n1 n2 -- n3) 83 " divide "
n3 ist der Quotient aus der Division von n1 durch den Divisor n2. Eine Fehlerbedingung besteht, wenn der Divisor Null ist oder der Quotient außerhalb des Intervalls (-32768...32767) liegt.

/mod (n1 n2 -- n3 n4) 83 " divide-mod "
n3 ist der Rest und n4 der Quotient aus der Division von n1 durch den Divisor n2. n3 hat dasselbe Vorzeichen wie n2 oder ist Null. Eine Fehlerbedingung besteht, wenn der Divisor Null ist oder der Quotient außerhalb des Intervalls (-32768..32767) liegt.

0 (-- 0)
Siehe -1.

1 (-- 1)
Siehe -1.

1+ (w1 -- w2) 83 " one-plus "
w2 ist das Ergebnis von Eins plus w1. Die Operation 1 + wirkt genauso.

1- (w1 -- w2) 83 " one-minus "
w2 ist das Ergebnis von w1 minus Eins. Die Operation 1 - wirkt genauso.

2 (-- 2)
Siehe -1.

2* (w1 -- w2) " two-times "
w1 wird um ein Bit nach links geschoben und das ergibt w2. In das niederwertigste Bit wird eine Null geschrieben. Die Operation 2 * wirkt genauso.

2+ (w1 -- w2) 83 " two-plus "
w2 ist das Ergebnis von w1 plus Zwei. Die Operation 2 + wirkt genauso.

2- (w1 -- w2) 83 " two-minus "
w2 ist das Ergebnis von w1 minus Zwei. Die Operation 2 - wirkt genauso.

2/ (n1 -- n2) 83 " two-divide "
n1 wird um ein Bit nach rechts verschoben und das ergibt n2. Das Vorzeichen wird berücksichtigt und bleibt unverändert. Die Operation 2 / wirkt genauso.

3 (-- 3)
Siehe -1.

3+ (w1 -- w2) " three-plus "
w2 ist das Ergebnis von w1 plus Drei. Die Operation 3 + wirkt genauso.

4 (-- 4)
Siehe -1.

abs (n -- u) 83 " absolute "
u ist der Betrag von n. Wenn n gleich -32768 ist, hat u den selben Wert wie n. Vergleiche auch "Arithmetik, Zweierkomplement".

even (u1 -- u2)
u2 ist die nächstgrößere gerade Zahl zu u1.

max (n1 n2 -- n3) 83 " maximum
n3 ist die Größere der beiden Werte n1 und n2. Benutzt die > Operation. Die größte Zahl für n1 oder n2 ist 32767.

min (n1 n2 -- n3) 83 " minimum "
n3 ist die Kleinere der beiden Werte n1 und n2. Benutzt die < Operation. Die kleinste Zahl für n1 oder n2 ist - 32768.

mod (n1 n2 -- n3) 83 " mod "
n3 ist der Rest der Division von n1 durch den Divisor n2. n3 hat daßelbe Vorzeichen wie n2 oder ist Null. Eine Fehlerbedingung besteht, wenn der Divisor Null ist oder der Quotient außerhalb des Intervalls (-32768..32767) liegt.

negate (n1 -- n2) 83
n2 hat den gleichen Betrag, aber das umgekehrte Vorzeichen von n. n2 ist gleich der Differenz von Null minus n1.

u/mod (u1 u2 -- u3 u4) " u-divide-mod "
u3 ist der Rest und u4 der Quotient aus der Division von u1 durch den Divisor u2. Die Zahlen u sind vorzeichenlose 16-Bit Werte (unsigned integer). Eine Fehlerbedingung besteht, wenn der Divisor Null ist.

umax (u1 u2 -- u3) " u-maximum "
u3 ist der Größere der beiden Werte u1 und u2. Benutzt die U> Operation. Die größte Zahl für u1 oder u2 ist 65535.

umin (u1 u2 -- u3) " u-minimum "
u3 ist der Kleinere der beiden Werte u1 und u2. Benutzt die U< Operation. Die kleinste Zahl für u1 oder u2 ist Null.

Logik und Vergleiche

0< (n -- flag) 83 " zero-less "
Wenn n kleiner als Null (negativ) ist, ist flag wahr. Dies ist immer dann der Fall, wenn das höchstwertige Bit von n gesetzt ist. Deswegen kann dieser Operator zum Testen dieses Bits benutzt werden.

0<> (n -- flag)
Wenn n verschieden von Null ist, ist flag wahr.

0= (w -- flag) 83 " zero-equals "
Wenn w gleich Null ist, ist flag wahr.

0> (n -- flag) 83 " zero-greater "
Wenn n größer als Null ist, ist flag wahr.

< (n1 n2 -- flag) 83 " less-than "
Wenn n1 kleiner als n2 ist, ist flag wahr.
z.B. -32768 32767 < ist wahr.
-32768 0 < ist wahr.

= (w1 w2 -- flag) 83 " equals "
Wenn w1 gleich w2 ist, ist flag wahr.

> (n1 n2 -- flag) 83 " greater-than "
Wenn n1 größer als n2 ist, ist flag wahr.
z.B. -32768 32767 > ist falsch.
-32768 0 > ist falsch.

and (w1 w2 -- w3) 83
w1 wird mit w2 bitweise logisch UND verknüpft und das ergibt w3.

case? (16b1 16b2 -- 16b1 false)
oder (16b1 16b2 -- true) " case-question "
Vergleicht die beiden Werte 16b1 und 16b2 miteinander. Sind sie gleich, verbleibt TRUE auf dem Stack. Sind sie verschieden, verbleibt FALSE und der darunterliegende Wert 16b1 auf dem Stack. Wird z.B. in der folgenden Form benutzt :

```
key Ascii a case? IF ... exit THEN
    Ascii b case? IF ... exit THEN
      ..
drop
```

Entspricht dem Ausdruck OVER = DUP IF NIP THEN .

false (-- 0)
Hinterläßt Null als Zeichen für logisch-falsch auf dem Stack.

not (w1 -- w2) 83
Jedes Bit von w1 wird einzeln invertiert und das ergibt w2.

or (w1 w2 -- w3) 83
w1 wird mit w2 logisch ODER verknüpft und das ergibt w3.

true (-- -1)
Hinterläßt -1 als Zeichen für logisch wahr auf dem Stack.

u< (u1 u2 -- flag) 83 " u-less-than "
Wenn u1 kleiner als u2 ist, ist flag wahr. Die Zahlen u sind vorzeichenlose 16-Bit Werte. Wenn Adressen verglichen werden sollen, muß U< benutzt werden, sonst passieren oberhalb von 32K seltsame Dinge !

U> (u1 u2 -- flag) 83 " u-greater-than "
Wenn u1 größer als u2 ist, ist flag wahr. Ansonsten gilt das gleiche wie für U< .

uwithin (u u1 u2 -- flag)
Wenn u1 kleiner oder gleich u und u kleiner u2 ist (u1<=u<u2), ist flag wahr. Benutzt die U< Operation.

xor (w1 w2 -- w3) 83 " x-or "
w1 wird mit w2 bitweise logisch EXKLUSIV ODER verknüpft und das ergibt w3.

Speicheroperationen

! (16b adr --) 83 " store "
16b werden in den Speicher auf die Adresse adr geschrieben. In 8-Bitweise adressierten Speichern werden die zwei Bytes adr und adr+1 überschrieben.

+! (w1 adr --) 83 " plus-store "
w1 wird zu dem Wert w in der Adresse adr addiert. Benutzt die + Operation. Die Summe wird in den Speicher in die Adresse adr geschrieben. Der alte Speicherinhalt wird überschrieben.

2! (32b adr --) 83 " two-fetch "
32b werden in den Speicher ab Adresse adr geschrieben.

2@ (adr -- 32b) 83 " two-fetch "
Von der Adresse adr wird der Wert 32b aus dem Speicher geholt.

@ (adr -- 16b) 83 " fetch "
Von der Adresse adr wird der Wert 16b aus dem Speicher geholt. Siehe auch ! .

c! (16b adr --) 83 " c-store "
Von 16b werden die niederwertigsten 8 Bit in den Speicher in die Adresse adr geschrieben.

c@ (adr -- 8b) 83 " c-fetch "
Von der Adresse adr wird der Wert 8b aus dem Speicher geholt.

cmove (adr1 adr2 u --) 83 " c-move "
Beginnend bei adr1 werden u Bytes zur Adresse adr2 kopiert. Zuerst wird das Byte von adr1 nach adr2 bewegt und dann aufsteigend fortgefahren. Wenn u Null ist, wird nichts kopiert.

cmove> (adr1 adr2 u --) 83 " c-move-up "
Beginnend bei adr1 werden u Bytes zur Adresse adr2 kopiert. Zuerst wird das Byte von adr1-plus-u-minus-1 nach adr2-plus-u-minus-1 kopiert und dann absteigend fortgefahren. Wenn u Null ist, wird nichts kopiert. Das Wort wird benutzt, um Speicherinhalte auf höhere Adressen zu verschieben, wenn die Speicherbereiche sich überlappen.

count (adr1 -- adr2 +n) 83
adr2 ist adr1-plus-1 und +n der Inhalt von adr1. Das Byte mit der Adresse adr1 enthält die Länge des Strings angegeben in Bytes. Die Zeichen des Strings beginnen bei adr1+1. Die Länge +n eines Strings darf im Bereich (0..255) liegen. Vergleiche auch "String, counted"

ctoggle (8b adr --) 83 " c-toggle "
Für jedes gesetzte Bit in 8b wird im Byte mit der Adresse adr das entsprechende Bit invertiert (d.h. ein zuvor gesetztes Bit ist danach gelöscht und ein gelösch-

tes Bit ist danach gesetzt). Für alle gelöschten Bits in 8b bleiben die entsprechenden Bits im Byte mit der Adresse adr unverändert. Der Ausdruck DUP C@ ROT XOR SWAP C! wirkt genauso.

erase (adr u --)
Von der Adresse adr an werden u Bytes im Speicher mit $00 überschrieben. Hat u den Wert Null, passiert nichts.

fill (adr u 8b --)
Von der Adresse adr an werden u Bytes des Speichers mit 8b überschrieben. Hat u den Wert Null, passiert nichts.

move (adr1 adr2 u --)
Beginnend bei adr1 werden u Bytes nach adr2 kopiert. Dabei ist es ohne Bedeutung, ob überlappende Speicherbereiche aufwärts oder abwärts kopiert werden, weil MOVE die passende Routine datzu auswählt. Hat u den Wert Null, passiert nichts. Siehe auch CMOVE und CMOVE>.

off (adr --)
Schreibt den Wert FALSE in den Speicher mit der Adresse adr.

on (adr --)
Schreibt den Wert TRUE in den Speicher mit der Adresse adr.

pad (-- adr) 83
adr ist die Startadresse einer "scratch area". In diesem Speicherbereich können Daten für Zwischenrechnungen abgelegt werden. Wenn die nächste verfügbare Stelle für das Dictionary verändert wird, ändert sich auch die Startadresse von PAD. Die vorherige Startadresse von PAD geht ebenso wie die Daten dort verloren.

place (adr1 +n adr2 --)
Bewegt +n Bytes von der Adresse adr1 zur Adresse adr2+1 und schreibt +n in die Speicherstelle mit der Adresse adr2. Wird in der Regel benutzt, um Text einer bestimmten Länge als "counted string" abzuspeichern. adr2 darf gleich, größer und auch kleiner als adr1 sein.

32-Bit-Worte

d* (d1 d2 -- d3) " d-times "
d1 multipliziert mit d2 ergibt d3.

d+ (d1 d2 -- d3) 83 " d-plus "
d1 und d2 addiert ergibt d3.

d- (d1 d2 -- d3) " d-minus"
d2 minus d1 ergibt d3.

d0= (d -- flag) 83 " d-zero-equals "
Wenn d gleich Null ist, ist flag wahr.

d< (d1 d2 -- flag) 83 " d-less-than "
Wenn d1 kleiner als d2 ist, ist flag wahr.

d= (d1 d2 -- flag) " d-equal"
Wenn d1 gleich d2 ist, ist flag wahr.

dabs (d -- ud) 83 " d-absolut "
ud ist der Betrag von d. Wenn d gleich -2.147.483.648 ist, hat ud den selben Wert wie d.

dnegate (d1 -- d2) 83 " d-negate "
d2 hat den gleichen Betrag aber ein anderes Vorzeichen als d1.

extend (n -- d)
Der Wert n wird auf den doppelt genauen Wert d vorzeichenrichtig erweitert.

m* (n1 n2 -- d) " m-times "
Der Wert von n1 wird mit n2 multiplizert und d ist das doppelt genaue Produkt.

m/mod (d n1 -- n2 n3) " m-divide-mod "
n2 ist der Rest und n3 der Quotient aus der Division der doppelt genauen Zahl d durch den Divisor n1. Der Rest n2 hat dasselbe Vorzeichen wie n1 oder ist Null. Eine Fehlerbedingung besteht, wenn der Divisor Null ist oder der Quotient außerhalb des Intervalls (-32768..32767) liegt.

ud/mod (ud1 u1 -- u2 ud2) " u-d-divide-mod "
u2 ist der Rest und ud2 der doppelt genaue Quotient aus der Divison der doppelt genauen Zahl ud1 durch den Divisor u1. Die Zahlen u sind vorzeichenlose 16-Bit Werte (unsigned integer). Eine Fehlerbedingung besteht, wenn der Divisor Null ist.

um* (u1 u2 -- ud) 83 " u-m-times "
Die Werte u1 und u2 werden mulitpliziert und das ergibt das doppelt genaue Produkt ud. UM* ist die anderen multiplizierenden Worten zugrundeliegende Routine.

um/mod (ud u1 -- u2 u3) 83 " u-m-divide-mod "
u2 ist der Rest und u3 der Quotient aus der Division von ud durch den Divisor u1. Die Zahlen u sind vorzeichenlose Zahlen. Eine Fehlerbedingung besteht, wenn der Divisor Null ist oder der Quotient außerhalb des Intervalls (0..65535) liegt.

Stack

-roll (16bn .. 16b1 16b0 +n -- 16b0 16bn .. 16b1)
" minus-roll "
Das oberste Glied eier Kette von +n Werten wird an die n-te Position gerollt. Dabei wird +n selbst nicht mitgezählt. 2 -ROLL wirkt wie -ROT. 0 -ROLL verändert nichts.

-rot (16b1 16b2 16b3 -- 16b3 16b1 16b2)
" minus-rot "
Die drei obersten 16b Werte werden rotiert, sodaß der oberste Wert zum Untersten wird. Hebt ROT auf.

.s (--) " dot-s "
Gibt alle Werte, die auf dem Stack liegen aus, ohne den Stack zu verändern. Oft benutzt, um neue Worte auszutesten. Die Ausgabe der Werte erfolgt von links nach rechts, der oberste Stackwert zuerst !

2dup (32b -- 32b 32b) 83 " two-dup "
Der Wert 32b wird dupliziert.

2drop (32b --) 83 " two-drop "
Der Wert 32b wird vom Stack entfernt.

2over (32b1 32b2 -- 32b1 32b2 31b1) " two-over "
Der Wert 32b1 wird über den Wert 32b2 herüber kopiert.

2swap (32b1 32b2 -- 32b2 32b1) 83 " two-swap "
Die beiden obersten 32-Bit Werte 32b1 und 32b2 werden vertauscht.

?dup (16b -- 16b 16b) 83 " question-dup "
oder (0 -- 0)
Nur wenn der Wert 16b von Null verschieden ist, wird er verdoppelt.

clearstack (-- empty)
Löscht den Datenstack. Alle Werte, die sich vorher auf dem Stack befanden, sind verloren.

depth (-- n)
n ist die Anzahl der Werte, die auf dem Stack lagen, bevor DEPTH ausgeführt wurde.

drop (16b --) 83
Der Wert 16b wird vom Stack entfernt.

dup (16b -- 16b 16b) 83
Der Wert 16b wird dupliziert.

nip (16b1 16b2 -- 16b2)
Der Wert 16b1, der unter 16b2 auf dem Stack liegt, wird vom Stack entfernt.

over (16b1 16b2 -- 16b1 16b2 16b1) 83
Der Wert 16b1 wird über 16b2 herüberkopiert.

pick (16bn..16b0 +n -- 16bn..16b0 16bn) 83
Der +n-te Wert auf dem Stack wird nach oben auf den Stack kopiert. Dabei wird +n selbst nicht mitgezählt. 0 PICK wirkt wie DUP, 1 PICK wie OVER.

roll (16bn 16bm..16b0 +n -- 16bm..16b0 16bn) 83
Das +n-te Glied einer Kette von n Werten wird nach oben auf den Stack gerollt. Dabei wird +n selbst nicht mitgezählt.

rot (16b1 16b2 16b3 -- 16b2 16b3 16b1) 83
Die drei obersten Werte auf dem Stack werden rotiert, sodaß der unterste zum obersten wird.

s0 (-- adr) " s-zero "
adr ist die Adresse einer Uservariablen, in der die Startadresse des Stacks steht. Der Ausdruck S0 @ SP! wirkt wie CLEARSTACK und leert den Stack.

swap (16b1 16b2 -- 16b2 16b1)83
Die beiden obersten 16-Bit Werte werden vertauscht.

sp! (adr --) " s-p-store "
Setzt den Stackzeiger (stack pointer) auf die Adresse adr. Der oberste Wert auf dem Stack ist dann der, welcher in der Adresse adr steht.

sp@ (-- adr) " s-p-fetch "
Holt die Adresse adr aus dem Stackzeiger. Der oberste Wert im Stack stand in der Speicherstelle bei adr, bevor SP@ ausgeführt wurde.

under (16b1 16b2 -- 16b2 16b1 16b2)
Eine Kopie des obersten Wertes auf dem Stack wird unter dem zweiten Wert eingefügt.

Returnstack

>r (16b --) C,83 " to-r "
Der Wert 16b wird auf den Returnstack gelegt. Siehe auch R> .

push (adr --)
Der Inhalt aus der Adresse adr wird auf den Returnstack bis zum nächsten EXIT oder ; verwahrt und sodann nach adr zurück geschrieben. Dies ermöglicht die lokale Verwendung von Variablen innerhalb einer :-Definition. Wird z.B. benutzt in der Form :

```
: hex.  ( n -- ) base push hex . ;
```

Hier wird innerhalb von HEX. in der Zahlenbasis HEX gearbeitet, um eine Zahl auszugeben. Nachdem HEX. ausgeführt worden ist, besteht die gleiche Zahlenbasis wie vorher, durch HEX wird sie also nur innerhalb von HEX. verändert.

r> (-- 16b) C,83 " r-from "
Der Wert 16b wird vom Returnstack geholt. Vergleiche R>.

rp! (adr --) " r-p-store "
Setzt den Returnstackzeiger (return stack pointer) auf die Adresse adr. Der oberste Wert im Returnstack ist nun der, welcher in der Speicherstelle bei adr steht.

r0 (-- adr) U " r-zero "
adr ist die Adresse einer Uservariablen, in der die Startadresse des Returnstacks steht.

r@ (-- 16b) C,83 " r-fetch "
Der Wert 16b ist eine Kopie des obersten Wertes auf dem Returnstack.

rdepth (-- n) " r-depth "
n ist die Anzahl der Werte, die auf dem Returnstack liegen.

rdrop (--) C " r-drop "
Der oberste Wert wird vom Returnstack entfernt. Der Datenstack wird nicht verändert. Entspricht der Operation R> DROP .

rp@ (-- adr) " r-p-fetch "
Holt die Adresse adr aus dem Returnstackzeiger. Der oberste Wert im Returnstack steht in der Speicherstelle bei adr.

Strings

Siehe auch im Anhang : "Strings"

" (-- adr) C,I " string "
(--) (compiling)
Liest den Text bis zum nächsten " und legt ihn als counted string im Dictionary ab. Kann nur während der Kompilation verwendet werden. Zur Laufzeit wird die Startadresse des counted string auf den Stack gelegt. Wird in der folgenden Form benutzt :
: <name> ... " <text>" ... ;
Das Leerzeichen, das auf das erste Anführungszeichen folgt, ist nicht Bestandteil des Strings.

(+d1 -- +d2) 83 " sharp "
Der Rest von +d1 geteilt durch den Wert in BASE wird in ein Ascii-Zeichen umgewandelt und dem Ausgabestring in Richtung absteigender Adressen hinzugefügt. +d2 ist der Quotient und verbleibt auf dem Stack zur weiteren Bearbeitung. Üblicherweise zwischen <# und #> benutzt.

#> (32b -- adr +n) 83 " sharp-greater "
Am Ende der strukturierten Zahlenausgabe wird der 32b Wert vom Stack entfernt. Hinterlegt werden die Adresse des erzeugten Ausgabestrings und +n als die Anzahl der Zeichen im Ausgabstring, passend für TYPE .

#s (+d -- 0 0) 83 " sharp-s "
+d wird mit # umgewandelt, bis der Quotient zu Null geworden ist. Dabei wird jedes Zwischenergebnis in ein Ascii-Zeichen umgewandelt und dem String für die strukturierte Zahlenausgabe angefügt. Wenn +d von vornherein den Wert Null hatte, wird eine einzelne Null in den String gegeben. Wird üblichwerweise zwischen <# und #> benutzt.

/string (adr1 n1 n2 -- adr2 n3) " cut-string "
n2 ist ein Index, welcher in den String weist, der im Speicher bei der Adresse adr1 beginnt. Der String hat die Länge n1. adr2 ist die Adresse und n3 die Länge des rechten Teilstrings, der bei Teilung des ursprünglichen Strings vor dessen n2-ten Element entsteht.

<# (--) 83 " less-sharp "
Leitet die strukturierte Zahlenausgabe ein. Um eine doppelt genaue Zahl in einen Ascii-String umzuwandeln, benutze man die Worte : <# # #S HOLD SIGN #>

accumulate (+d1 adr char -- +d2 adr)
Dient der Umwandlung von Ziffern in Zahlen. Multipliziert die Zahl +d1 mit BASE, um sie eine Stelle in der aktuellen Zahlenbasis nach links zu rücken, und addiert den Zahlenwert von char. char stellt eine Ziffer dar. adr wird nicht verändert. wird z.B. in CONVERT benutzt. char muß eine in der Zahlenbasis gültige Ziffer darstellen.

capital (cha1 -- char2)
Die Zeichen im Berich a bis z werden in die Großbuchstaben A bis Z umgewandelt. Andere Zeichen werden nicht verändert.

capitalize (adr -- adr)
Wandelt alle Kleinbuchstaben im counted string bei der Adresse adr in GrOßbuchstaben um. adr wird nicht verändert. Siehe auch CAPITAL .

convert (+d1 adr1 -- +d2 adr2) 83
Wandelt den Ascii-Text ab adr1+1 in eine Zahl entsprechend der Zahlenbasis BASE um. Der enstehende Wert wird in d1 akkumuliert und als d2 hinterlassen. adr2 ist die Adresse des ersten nicht umwandelbaren Zeichens im Text.

digit? (char -- digit true)
oder (char -- false)
Prüft, ob das Zeichen char eine gültige Ziffer entsprechend der aktuellen Zahlenbasis in BASE ist. Ist das der Fall, so wird der Zahlenwert der Ziffer und TRUE auf den Stack gelegt. Ist char keine gültige Ziffer, wird FALSE hinterlegt.

hold (char --) 83
Das Zeichen char wird in den Ausgabestring für die Zahlenausgabe eingefügt. Üblicherweise zwischen <# und #> benutzt.

nullstring? (adr -- adr false)
oder (adr -- true)
Prüft, ob der counted string bei der Adresse adr ein String der Länge Null ist. Wenn dies der Fall ist, wird TRUE hinterlegt. Sonst bleibt adr erhalten und FALSE wird obenauf gelegt.

number (adr -- d)
Wandelt den counted string bei der Adresse adr in eine Zahl d um. Die Umwandlung erfolgt entsprechend der Zahlenbasis in BASE. Eine Fehlerbedingung besteht, wenn die Ziffern des Strings nicht in eine Zahl verwandelt werden können. Durch Angabe eines Präfix (siehe NUMBER?) kann die Basis für diese Zahl modifiziert werden.

number? (adr -- d 0>)
oder (adr -- n 0<)
oder (adr -- adr false)
Wandelt den counted string bei der Adresse adr in eine Zahl n um. Die Umwandlung erfolgt entsprechend der Zahlenbasis in BASE oder wird vom ersten Zeichen im String bestimmt. Enthält der String zwischen den Ziffer auch die Asciizeichen für Punkt oder Komma, so wird er als doppelt genaue Zahl interpretiert und 0> gibt die Zahl der Ziffern hinter dem Punkt einschließlich an. Sonst wird der String in eine einfach genaue Zahl n umgewandelt und eine Zahl kleiner als Null hinterlassen. Wenn die Ziffern des Strings nicht in eine Zahl umgewandelt werden können, bleibt die Adresse des String erhalten und FALSE wird auf den Stack gelegt. Die Zeichen, die zur Bestimmung der Zahlenbasis dem Ziffernstring vorangestellt werden können, sind :

```
%       ( Basis 2    "binär")
&       ( Basis 10   "dezimal")
$       ( Basis 16   "hexadezimal")
h       ( Basis 16   "hexadezimal")
```

Der Wert in BASE wird dadurch nicht verändert.

scan (adr1 n1 char -- adr2 n2)
Der String mit der Länge n1, der im Speicher ab Adresse adr1 steht, wird nach dem zeichen char durchsucht. adr2 ist die Adresse, bei der das Zeichen char gefunden wurde. n2 ist die Länge des verbleibenden Strings. Wird char nicht gefunden, so ist adr2 die Adrese des ersten Bytes hinter dem String und n2 ist Null.

sign (n --) 83
Wenn n negativ ist, wird ein Minuszeichen in den Ausgabestring für die Zahlenausgabe eingefügt. Wird üblicherweise zwischen <# und #> benutzt.

skip (adr1 n1 car -- adr2 n2)
Der String mit der Länge n1, der im Speicher ab Adresse adr1 steht, wird nach dem ersten Zeichen, das verschieden von char ist, durchsucht. adr2 ist die Adresse dieses Zeichens und n2 die Länge des verbleibenden Strings. Besteht der gesamte String aus dem Zeichen char so ist adr2 die Adresse des Bytes hinter dem String und n2 ist Null.

Datentypen

: (-- sys) 83 "colon"
ein definierendes Wort, das in der Form:
: <name> ... ;
benutzt wird. Es erzeugt die Wortdefinition für <name> im Kompilations-Vokabular und schaltet den Kompiler an. Das erste Vokabular der Suchreihenfolge (das "transient" Vokabular) wird durch das Kompilations-Vokabular ersetzt (ACHTUNG!). Das Kompilations-Vokabular wird nicht geändert. Der Quelltext wird anschließend kompiliert. <name> wird als "colon-definition" oder ":-Definition" bezeichnet. Die neue Wortdefinition für <name> kann nicht im Dictionary gefunden werden, bis das zugehörige ; oder ;CODE erfolgreich ausgeführt wurde. RECURSIVE macht <name> jedoch sofort auffindbar. Vergleiche HIDE und REVEAL .
Eine Fehlerbehandlung wird eingeleitet, wenn ein Wort während der Kompilation nicht gefunden bzw. nicht in eine Zahl (siehe auch BASE) gewandelt werden kann.
Der auf dem Stack hinterlassene Wert sys dient der Kompiler-Sicherheit und wird durch ; bzw. ;CODE abgebaut.

; (--) 83 I C "semi-colon"
(sys --) compiling
beendet die Kompilation einer :-Definition; macht den Namen dieser :-Definition im Dictionary auffindbar, schaltet den Kompiler aus, den Interpreter ein und kompiliert ein UNNEST (Siehe auch EXIT). Der Stackparameter sys, der in der Regel von : hinterlassen wurde, wird geprüft und abgebaut. Eine Fehlerbehandlung wird eingeleitet, wenn sys nicht dem erwarteten Wert entspricht.

2Constant (32b --) 83
ein definierendes Wort, das in der Form:
32b 2Constant <name>
benutzt wird. Erzeugt einen Kopf für <name> und legt 32b in dessen Parameterfeld so ab, daß bei Ausführung von <name> 32b wieder auf den Stack gelegt wird.

2Variable (--) 83
ein definierendes Wort, das in der Form:
2Variable <name>
benutzt wird. Erzeugt einen Kopf für <name> und hält 4 Byte in seinem Parameterfeld frei, die den Inhalt der 2VARIABLEN aufnehmen. Die 2VARIABLE wird nicht initialisiert. Wenn <name> ausgeführt wird, wird die Adresse des Parameterfelds auf den Stack gelegt. Siehe VARIABLE .

Alias (cfa --)
ein definierendes Wort, das typisch in der Form:
' <oldname> Alias <newname>
benutzt wird. ALIAS erzeugt einen Kopf für <newname> im Dictionary. Wird <newname> aufgerufen, so verhält es sich wie <oldname>. Insbesondere wird beim Kompilieren nicht <newname>, sondern <oldname> im Dictionary ein-

getragen. Im Unterschied zu
: <newname> <oldname> ;
ist es mit ALIAS möglich, Worte, die den Returnstack beeinflussen (z.B. >R oder R>), mit anderem Namen zu definieren. Außer dem neuen Kopf für <newname> wird kein zusaetzlicher Speicherplatz verbraucht.
Gegenwärtig wird bei Ausführung von >NAME aus einer CFA in der Regel der letzte mit ALIAS erzeugte Name gefunden.

Constant (16b --) 83
ein definierendes Wort, das in der Form:
16b Constant <name>
benutzt wird. Wird <name> später ausgeführt, so wird 16b auf den Stack gelegt.

Create (--) 83
ein definierendes Wort, das in der Form:
Create <name>
benutzt wird. Erzeugt einen Kopf für <name>. Die nächste freie Stelle im Dictionary (vergleiche HERE und DP) ist nach einem CREATE <name> das erste Byte des Parameterfelds von <name>. Wenn <name> ausgeführt wird, legt es die Adresse seines Parameterfelds auf den Stack. CREATE reserviert keinen Speicherplatz im Parameterfeld von <name>.

Defer (--)
ein definierendes Wort, das in der Form:
Defer <name>
benutzt wird. Erzeugt den Kopf für ein neues Wort <name> im Dictionary, hält 2 Byte in dessen Parameterfeld frei und speichert dort zunächst die Kompilationsadresse einer Fehlerroutine. Wird <name> nun ausgeführt, so wird eine Fehlerbehandlung eingeleitet. Man kann dem Wort <name> jedoch zu jeder Zeit eine andere Funktion zuweisen mit der Sequenz:
' <action> Is <name>
Nach dieser Zuweisung verhält sich <name> wie <action>. Mit diesem Mechanismus kann man zwei Probleme elegant lösen: Einerseits läßt sich <name> bereits kompilieren, bevor ihm eine sinnvolle Aktion zugewiesen wurde. Damit ist die Kompilation erst später definierter Worte (Vorwärts-Referenzen) indirekt möglich. Andererseits ist die Veränderung des Verhaltens von <name> für spezielle Zwecke auch nachträglich möglich, ohne neu kompilieren zu müssen.

Deferred Worte im System sind:
R/W , 'COLD , 'RESTART , 'ABORT , 'QUIT ,
NOTFOUND , .STATUS und DISKERR .
Diese Worte sind DEFERred, damit ihr Verhalten für die Anwendung geändert werden kann. Ein spezielles DEFERred Wort ist >INTERPRET .

Input: (--) "input-colon"
ein definierendes Wort, benutzt in der Form:
Input: <name>
newKEY newKEY? newDECODE newEXPECT ;

INPUT: erzeugt einen Kopf für <name> im Dictionary und kompiliert einen Satz von Zeigern auf Worte, die für die Eingabe von Zeichen zuständig sind. Wird <name> ausgeführt, so wird ein Zeiger auf das Parameterfeld von <name> in die Uservariable INPUT geschrieben. Alle Eingaben werden jetzt über die neuen Eingabewörte abgewickelt. Die Reihenfolge der Worte nach INPUT: <name> bis zum semi-colon muß eingehalten werden: (..key ..key? ..decode ..expect).

Im System ist das mit INPUT: definierte Wort KEYBOARD enthalten, nach dessen Ausführung alle Eingaben von der Tastatur geholt werden und ein einfacher Zeilen-Editor wirksam ist. Siehe DECODE und EXPECT .

Is (cfa --)
ein Wort, mit dem das Verhalten eines deferred Wortes verändert werden kann. IS wird in der Form:

```
' <action> Is <name>
```

benutzt. Wenn <name> kein deferred Wort ist, wird eine Fehlerbehandlung eingeleitet, sonst verhält sich <name> anschließend wie <action>. Siehe DEFER .

Output: (--) "output-colon"
ein definierendes Wort, benutzt in der Form:

```
Output: <name>
  newEMIT newCR newTYPE newDEL newPAGE newAT
  newAT? ;
```

OUTPUT: erzeugt einen Kopf für <name> im Dictionary und kompiliert einen Satz von Zeigern auf Worte, die für die Ausgabe von Zeichen zuständig sind. Wird <name> ausgeführt, so wird ein Zeiger auf das Parameterfeld von <name> in die Uservariable OUTPUT geschrieben. Alle Ausgaben werden jetzt über die neuen Ausgabeworte abgewickelt. Die Reihenfolge der Worte nach OUTPUT: <name> bis zum semi-colon muß eingehalten werden: (..emit ..cr ..type ..del ..page ..at ..at?).

Im System ist das mit OUTPUT: definierte Wort DISPLAY enthalten, nach dessen Ausführung alle Ausgaben auf den Bildschirm geleitet werden. Vergleiche auch das Wort PRINT aus dem Printer-Interface.

User (--) 83
ein definierendes Wort, benutzt in der Form:

```
User <name>
```

USER erzeugt einen Kopf für <name> im Dictionary und hält 2 Byte in der Userarea frei. Siehe UALLOT . Diese 2 Byte werden für den Inhalt der Uservariablen benutzt und werden nicht initialisiert. Im Parameterfeld der Uservariablen im Dictionary wird nur ein Offset zum Beginn der Userarea abgelegt. Wird <name> ausgeführt, so wird die Adresse des Wertes der Uservariablen in der Userarea auf den Stack gegeben.

Uservariablen werden statt normaler Variablen z.B. dann benutzt, wenn der Einsatz des Multitaskers geplant ist und mindestens eine Task die Variable unbeeinflußt von anderen Tasks benötigt. (Jede Task hat ihre eigene Userarea).

Variable (--) 83
ein definierendes Wort, benutzt in der Form:
Variable <name>
VARIABLE erzeugt einen Kopf für <name> im Dictionary und hält 2 Byte in seinem Parameterfeld frei. Siehe ALLOT . Dies Parameterfeld wird für den Inhalt der Variablen benutzt, wird jedoch nicht initialisiert. Wird <name> ausgeführt, so wird die Adresse des Parameterfeldes von <name> auf den Stack gelegt. Mit @ und ! kann der Wert von <name> gelesen und geschrieben werden. Siehe auch +! .

Vocabulary (--) 83
ein definierendes Wort, das in der Form:
Vocabulary <name>
benutzt wird. VOCABULARY erzeugt einen Kopf für <name>, das den Anfang einer neuen Liste von Worten bildet. Wird <name> ausgeführt, so werden bei der Suche im Dictionary zuerst die Worte in der Liste von <name> berücksichtigt.

Wird das VOCABULARY <name> durch die Sequenz:
<name> DEFINITIONS
zum Kompilations-Vokabular, so werden neue Wort-Definitionen in die Liste von <name> gehängt. Vergleiche auch CONTEXT , CURRENT , ALSO , TOSS , ONLY , FORTH , ONLYFORTH .

Dictionary - Worte

' (-- addr) 83 "tick"
Wird in der Form ' <name> benutzt.
addr ist die Kompilationsadresse von <name>. Wird <name> nicht in der Suchreihenfolge gefunden, so wird eine Fehlerbehandlung eingeleitet.

(forget (addr --) "paren-forget"
Entfernt alle Worte, deren Kompilationsadresse oberhalb von addr liegt, aus dem Dictionary und setzt HERE auf addr. Ein Fehler liegt vor, falls addr im Heap liegt.

, (16b --) 83 "comma"
2 ALLOT für 16b und speichere 16b ab HERE 2- .

.name (addr --) "dot-name"
addr ist die Adresse des Countfeldes eines Namens. Dieser Name wird ausgedruckt. Befindet er sich im Heap, so wird das Zeichen ¦ vorangestellt. Ist addr null, so wird "???" ausgegeben.

align (--)
rundet HERE auf den nächsten geraden Wert auf. Ist HERE gerade, so geschieht nichts.

allot (w --) 83
Allokiere w Bytes im Dictionary. Die Adresse des nächsten freien Dictionaryplatzes wird entsprechend verstellt.

c, (16b --) "c-comma"
ALLOT ein Byte und speichere die unteren 8 Bit von 16b in HERE 1-.

clear (--)
Löscht alle Namen und Worte im Heap.

custom-remove (dic symb -- dic symb)
Ein deferred Wort, daß von FORGET , CLEAR usw. aufgerufen wird. dic ist die untere Grenze des Dictonaryteils, der vergessen wird und symb die obere. Gewöhnlich zeigt symb in den Heap. Dieses Wort kann dazu benutzt werden, eigene Datenstrukturen, die Zeiger enthalten, bei FORGET korrekt abzuabrbeiten. Es wird vom Fileinterface verwendet, daher darf es nicht einfach überschrieben werden. Man kann es z.B. in folgender Form benutzen :

```
: <name>  [ ' custom-remove >body @ , ]
          <liststart> @ remove ;
' <name> Is custom-remove
```

Auf diese Weise stellt man sicher, daß das Wort, das vorher in CUSTOM-REMOVE eingetragen war, weiterhin ausgeführt wird. Siehe auch REMOVE

dp (-- addr) "d-p"
Eine Uservariable, die die Adresse des nächsten freien Dictionaryplatzes enthält.

empty (--)
Löscht alle Worte, die nach der letzten Ausführung von SAVE oder dem letzten Kaltstart definiert wurden. DP wird auf seinen Kaltstartwert gesetzt und der Heap gelöscht.

forget (--) 83
Wird in der Form FORGET <name> benutzt.
Falls <name> in der Suchreihenfolge gefunden wird, so werden <name> und alle danach definierten Worte aus dem Dictionary entfernt. Wird <name> nicht gefunden, so wird eine Fehlerbehandlung eingeleitet. Liegt <name> in dem durch SAVE geschützten Bereich, so wird ebenfalls eine Fehlerbehandlung eingeleitet. Es wurden Vorkehrungen getroffen, die es ermöglichen, aktive Tasks und Vokabulare, die in der Suchreihenfolge auftreten, zu vergessen.

here (-- addr) 83
addr ist die Adresse des nächsten freien Dictionaryplatzes.

hide (--)
Entfernt das zuletzt definierte Wort aus der Liste des Vokabulars, in das es eingetragen wurde. Dadurch kann es nicht gefunden werden. Es ist aber noch im Speicher vorhanden. (s.a. REVEAL LAST)

last (-- addr)
Variable, die auf das Countfeld des zuletzt definierten Wortes zeigt.

name> (addr1 -- addr2) "name-from"
addr2 ist die Kompilationsadresse die mit dem Countfeld in addr1 korrespondiert.

origin (-- addr)
addr ist die Adresse, ab der die Kaltstartwerte der Uservariablen abgespeichert sind.

remove (dic sym thread -- dic sym)
Dies ist ein Wort, das zusammen mit CUSTOM-REMOVE verwendet wird. dic ist die untere Grenze des Dictionarybereiches, der vergessen werden soll und sym die obere. Typisch zeigt sym in den Heap. thread ist der Anfang einer Kette von Zeigern, die durch einen Zeiger mit dem Wert Null abgeschlossen wird. Wird REMOVE dann ausgeführt, so werden alle Zeiger (durch Umhängen der übrigen Zeiger) aus der Liste entfernt, die in dem zu vergessenden Dictionarybereich liegen. Dadurch ist es möglich, FORGET und ähnliche Worte auf Datenstrukturen anzuwenden.

reveal (--)
Trägt das zuletzt definierte Wort in die Liste des Vokabulars ein, in dem es definiert wurde.

save (--)
Kopiert den Wert aller Uservariablen in den Speicherbereich ab ORIGIN und sichert alle Vokabularlisten. Wird später COLD ausgeführt, so befindet sich das System im gleichen Speicherzustand wie bei Ausführung von SAVE.

uallot (n1 -- n2)
Allokiere bzw. deallokiere n1 Bytes in der Userarea. n2 gibt den Anfang des allokierten Bereiches relativ zum Beginn der Userarea an. Eine Fehlerbehandlung wird eingeleitet, wenn die Userarea voll ist.

udp (-- addr) "u-d-p"
Eine Uservariable, in dem das Ende der bisher allokierten Userarea vermerkt ist.

>body (addr1 -- addr2) "to-body"
addr2 ist die Parameterfeldadresse, die mit der Kompilationsadresse addr1 korrespondiert.

>name (addr1 -- addr2) "to-name"
addr2 ist die Adresse eines Countfeldes, das mit der Kompilationsadresse addr1 korrespondiert. Es ist möglich, daß es mehrere addr2 für ein addr1 gibt. In diesem Fall ist nicht definiert, welche ausgewählt wird.

Vokabular - Worte

also (--)
Ein Wort, um die Suchreihenfolge zu spezifizieren. Das Vokabular im auswechselbarem Teil der Suchreihenfolge wird zum ersten Vokabular im festen Teil gemacht, wobei die anderen Vokabulare des festen Teils nach hinten rücken. Ein Fehler liegt vor, falls der feste Teil sechs Vokabulare enthält.

Assembler (--)
Ein Vokabular, das Prozessor-spezifische Worte enthält, die für Code-Definitionen benötigt werden.

context (-- addr)
addr ist die Adresse des auswechselbaren Teils der Suchreihenfolge. Sie enthält einen Zeiger auf das erste zu durchsuchende Vokabular.

current (-- addr)
addr ist die Adresse eines Zeigers, der auf das Kompilationsvokabular zeigt, in das neue Worte eingefügt werden.

definitions (--) 83
Ersetzt das gegenwärtige Kompilationsvokabular durch das Vokabular im auswechselbaren Teil der Suchreihenfolge, d.h. neue Worte werden in dieses Vokabular eingefügt.

Forth (--) 83
Das ursprüngliche Vokabular.

forth-83 (--) 83
Lt. Forth83-Standard soll dieses Wort sicherstellen, daß ein Standardsystem benutzt wird. Im volksFORTH funktionslos.

Only (--)
Löscht die Suchreihenfolge vollständig und ersetzt sie durch das Vokabular ONLY im festen und auswechselbaren Teil der Suchreihenfolge. ONLY enthält nur wenige Worte, die für die Erzeugung einer Suchreihenfolge benötigt werden.

Onlyforth (--)
entspricht der häufig benötigten Sequenz
ONLY FORTH ALSO DEFINITIONS.

seal (--)
Löscht das Vokabular ONLY , so daß es nicht mehr durchsucht wird. Dadurch ist es möglich, nur die Vokabulare des Anwenderprogramms durchsuchen zu lassen.

toss (--)
Entfernt das erste Vokabular des festen Teils der Suchreihenfolge. Insofern ist es das Gegenstück zu ALSO .

words (--)
Gibt die Namen der Worte des Vokabulars, das im auswechselbaren Teil der Suchreihenfolge steht, aus, beginnend mit dem zuletzt erzeugtem Namen.

voc-link (-- addr)
Eine Uservariable. Sie enthält den Anfang einer Liste mit allen Vokabularen. Diese Liste wird u.a. für FORGET benötigt.

vp (-- addr) "v-p"
Eine Variable, die das Ende der Suchreihenfolge markiert. Sie enthält ausserdem Informationen über die Länge der Suchreihenfolge.

Heap - Worte

?head (-- addr) "question-head"
Eine Variable, die angibt, ob und wieviele der nächsten zu erzeugenden Namen im Heap angelegt werden sollen (s.a. ¦).

halign (--) "h-align"
HEAP wird auf den nächsten geradzahligen Wert abgerundet. Ist HEAP gerade, geschieht nichts.

hallot (n --)
Allokiere bzw. deallokiere n Bytes auf dem Heap. Dabei wird der Stack verschoben, ebenso wie der Beginn des Heap.

heap (-- addr)
addr ist der Anfang des Heap. Er wird u.A. durch HALLOT geändert.

heap? (addr -- flag) "heap-question"
Das Flag ist wahr, wenn addr ein Byte im Heap adressiert, ansonsten falsch.

¦ (--) "headerless"
Setzt bei Ausführung ?HEAD so, daß der nächste erzeugte Name nicht im normalen Dictionaryspeicher angelegt wird, sondern auf dem Heap.

Kontrollstrukturen

+LOOP (n --) 83,I,C "plus-loop"
(sys --) compiling

n wird zum Loopindex addiert. Falls durch die Addition die Grenze zwischen limit-1 und limit überschritten wurde, so wird die Schleife beendet und die Loop-Parameter werden entfernt. Wurde die Schleife nicht beendet, so wird sie hinter dem korrespondierenden DO bzw. ?DO fortgesetzt.

?DO (w1 w2 --) 83,I,C "question-do"
(-- sys) compiling

Wird in der folgenden Art benutzt:

```
?DO ... LOOP   bzw.  ?DO ... +LOOP
```

Beginnt eine Schleife. Der Schleifenindex beginnt mit w2, limit ist w1. Details über die Beendigung von Schleifen: s. +LOOP. Ist w2=w1 , so wird der Schleifenrumpf überhaupt nicht durchlaufen.

?exit (flag --) "question-exit"

Führt EXIT aus, falls das flag wahr ist. Ist das flag falsch, so geschieht nichts.

BEGIN (--) 83,I,C
(sys ---) compiling

Wird in der folgenden Art benutzt:

```
BEGIN ( ...flag WHILE ) ... flag UNTIL
oder:  BEGIN ( ...flag WHILE ) ... REPEAT
```

BEGIN markiert den Anfang einer Schleife. Der ()-Ausdruck ist optional und kann beliebig oft auftreten. Die Schleife wird wiederholt, bis das flag bei UNTIL wahr oder oder das flag bei WHILE falsch ist. REPEAT setzt die Schleife immer fort.

bounds (start count -- limit start)

Dient dazu, ein Intervall, das durch Anfangswert start und Länge count gegeben ist, in ein Intervall umzurechnen, das durch Anfangswert start und Endwert+1 limit beschrieben wird.

Beispiel : 10 3 bounds DO ... LOOP führt dazu, das I die Werte 10 11 12 annimmt.

DO (w1 w2 --) 83,I,C
(sys --) compiling

Entspricht ?DO , jedoch wird der Schleifenrumpf mindestens einmal durchlaufen. Ist w1=w2 , so wird der Schleiferumpf 65536-mal durchlaufen.

ELSE (--) 83,I,C
(sys1 -- sys2) compiling

Wird in der folgenden Art benutzt:

```
flag IF ... ELSE ... THEN
```

ELSE wird unmittelbar nach dem Wahr-Teil , der auf IF folgt, ausgeführt. ELSE setzt die Ausführung unmittelbar hinter THEN fort.

execute (addr --) 83
Das Wort, dessen Kompilationsadresse addr ist, wird ausgeführt.

I (-- w) 83,C
Wird zwischen DO und LOOP benutzt, um eine Kopie des Schleifenindex auf den Stack zu holen.

IF (flag --) 83,I,C
(-- sys) compiling
Wird in der folgenden Art benutzt:
flag IF ... ELSE ... THEN
oder: flag IF ... THEN
Ist das flag wahr, so werden die Worte zwischen IF und ELSE ausgeführt und die Worte zwischen ELSE und THEN ignoriert. Der ELSE-Teil ist optional.
Ist das flag falsch, so werden die Worte zwischen IF und ELSE (bzw. zwischen IF und THEN , falls ELSE nicht vorhanden ist) ignoriert.

J (-- w) 83,C
Wird zwischen DO .. DO und LOOP .. LOOP benutzt, um eine Kopie des Schleifenindex der äusseren Schleife auf den Stack zu holen.

LEAVE (--) 83,C
Setzt die Ausführung des Programmes hinter dem nächsten LOOP oder +LOOP fort, wobei die zugehöhrige Schleife beendet wird. Mehr als ein LEAVE pro Schleife ist möglich, ferner kann LEAVE zwischen anderen Kontrollstrukturen auftreten. Der Forth83-Standard schreibt abweichend vom volksFORTH vor, daß LEAVE ein immediate Wort ist.

LOOP (--) 83,I,C
(-- sys) compiling
Entspricht +LOOP, jedoch mit n=1 fest gewählt.

perform (addr --)
addr ist eine Adresse, unter der sich ein Zeiger auf die Kompilationsadresse eines Wortes befindet. Dieses Wort wird ausgeführt. Entspricht der Sequenz @ EXECUTE .

REPEAT (--) 83,I,C
(-- sys) compiling
Wird in der folgenden Form benutzt:
BEGIN (.. WHILE) .. REPEAT
REPEAT setzt die Ausführung der Schleife unmittelbar hinter BEGIN fort. Der ()-Ausdruck ist optional und kann beliebig oft auftreten.

THEN (--) 83,I,C
(sys --) compiling
Wird in der folgenden Art benutzt:
IF (...ELSE) ... THEN
Hinter THEN ist die Programmverzweigung zuende.

UNTIL (flag --) 83,I,C
(sys --) compiling

Wird in der folgenden Art benutzt:

BEGIN (... flag WHILE) ... flag UNTIL

Markiert das Ende einer Schleife, deren Abbruch durch flag herbeigeführt wird. Ist das flag vor UNTIL wahr, so wird die Schleife beendet, ist es falsch, so wird die Schleife unmittelbar hinter BEGIN fortgesetzt.

WHILE (flag --) 83,I,C
(sys1 -- sys2)compiling

Wird in der folgenden Art benutzt:

BEGIN .. flag WHILE .. REPEAT

oder:

BEGIN .. flag WHILE .. flag UNTIL

Ist das flag vor WHILE wahr, so wird die Ausführung der Schleife bis UNTIL oder REPEAT fortgesetzt, ist es falsch, so wird die Schleife beendet und das Programm hinter UNTIL bzw. REPEAT fortgesetzt. Es können mehre WHILE in einer Schleife verwendet werden.

Compiler - Worte

," (--) "comma-quote"

Speichert einen counted String im Dictionary ab HERE. Dabei wird die Länge des Strings in dessen erstem Byte, das nicht zur Länge hinzugezählt wird, vermerkt.

Ascii (-- char) I
       (--) compiling

Wird in der folgenden Art benutzt:

```
Ascii ccc
```

wobei ccc durch ein Leerzeichen beendet wird. char ist der Wert des ersten Zeichens von ccc im benutzten Zeichensatz (gewöhnlich ASCII). Falls sich das System im kompilierenden Zustand befindet, so wird char als Konstante compiliert. Wird die :-definition später ausgeführt, so liegt char auf dem Stack.

compile (--) 83,C

Typischerweise in der folgenden Art benutzt:

```
: <name> ... compile <namex> ... ;
```

Wird <name> ausgeführt, so wird die Kompilationsadresse von <namex> zum Dictionary hinzugefügt und nicht ausgeführt. Typisch ist <name> immediate und <namex> nicht immediate.

Does> (-- addr) 83,I,C "does"
       (--) compiling

Definiert das Verhalten des Wortes, das durch ein definierendes Wort erzeugt wurde. Wird in der folgenden Art benutzt:

```
: <namex> ... <create> ... Does> ... ;
```

und später:

```
<namex> <name>
```

wobei <create> CREATE oder ein anderes Wort ist, das CREATE ausführt.

Zeigt das Ende des Wort-erzeugenden Teils des definierenden Wortes an. Beginnt die Kompilation des Codes, der ausgeführt wird, wenn <name> aufgerufen wird. In diesem Fall ist addr die Parameterfeldadresse von <name>. addr wird auf den Stack gebracht und die Sequenz zwischen DOES> und ; wird ausgeführt.

immediate (--) 83

Markiert das zuletzt definierte Wort als "immediate", d.h. dieses Wort wird auch im kompilierenden Zustand ausgeführt.

Literal (-- 16b) 83,I,C
         (16b --) compiling

Typisch in der folgenden Art benutzt:

```
[ 16b ] Literal
```

Kompiliert ein systemabhängiges Wort, so daß bei Ausführung 16b auf den Stack gebracht wird.

recursive (--) I,C
(--) compiling
Erlaubt die rekursive Kompilation des gerade definierten Wortes in diesem Wort selbst. Ferner kann Code für Fehlerkontrolle erzeugt werden.

restrict (--)
Markiert das zuletzt definierte Wort als "restrict", d.h. dieses Wort kann nicht vom Textinterpreter interpretiert sondern ausschliesslich in anderen Worten kompiliert werden.

[(--) 83,I "left-bracket"
(--) compiling
Schaltet den interpretierenden Zustand ein. Der Quelltext wird sukzessive ausgeführt. Typische Benutzung : s. LITERAL

['] (-- addr) 83,I,C "bracket-tick"
(--) compiling
Wird in der folgenden Art benutzt:
['] <name>
Kompiliert die Kompilationsadresse von <name> als eine Konstante. Wenn die :-definition später ausgeführt wird, so wird addr auf den Stack gebracht. Ein Fehler tritt auf, wenn <name> in der Suchreihenfolge nicht gefunden wird.

[compile] (--) 83,I,C "bracket-compile"
(--) compiling
Wird in der folgenden Art benutzt:
[compile] <name>
Erzwingt die Kompilation des folgenden Wortes <name>. Damit ist die Kompilation von immediate-Worten möglich.

Interpreter - Worte

((--) 83,I "paren"
(--) compiling
Wird in der folgenden Art benutzt:
(ccc)
Die Zeichen ccc, abgeschlossen durch) , werden als Kommentar betrachtet. Kommentare werden ignoriert. Das Leerzeichen zwischen (und ccc ist nicht Teil des Kommentars. (kann im interpretierenden oder kompilierenden Zustand benutzt werden. Fehlt) , so werden alle Zeichen im Quelltext als Kommentar betrachtet.

+load (n --) "plus-load"
LOAD den Block, dessen Nummer um n höher ist, als die Nummer des gegenwärtig interpretierten Blockes.

+thru (n1 n2 --) "plus-thru"
LOAD die Blöcke hintereinander, die n1..n2 vom gegenwärtigen Block entfernt sind.
Beispiel : 1 2 +thru lädt die nächsten beiden Blöcke.

--> (--) I "next-block"
(--) compiling
Setze die Interpretation auf dem nächsten Block fort.

>in (-- addr) 83 "to-in"
Eine Variable, die den Offset auf das gegenwärtige Zeichen im Quelltext enthält. s. WORD

>interpret (--) "to-interpret"
Ein deferred Wort, das die gegenwärtige Interpretationsroutine aufruft, ohne eine Rückkehradresse auf dem Returnstack zu hinterlassen (was INTERPRET tut). Es kann als spezielles GOTO angesehen werden.

blk (-- addr) 83 "b-l-k"
Eine Variable, die die Nummer des gerade als Quelltext interpretierten Blockes enthält. Ist der Wert von BLK Null, so wird der Quelltext vom Texteingabepuffer genommen.

find (addr1 -- addr2 n)83
addr1 ist die Adresse eines counted string. Der String enthält einen Namen, der in der aktuellen Suchreihenfolge gesucht wird. Wird das Wort nicht gefunden, so ist addr2 = addr1 und n = Null. Wird das Wort gefunden, so ist addr2 dessen Kompilationsadresse und n erfüllt folgende Bedingungen:
n ist vom Betrag 2 , falls das Wort restrict ist, sonst vom Betrag 1
n ist positiv, wenn das Wort immediate ist, sonst negativ.

interpret (--)
Beginnt die Interpretation des Quelltextes bei dem Zeichen, das durch den Inhalt von >IN indiziert wird. >IN indiziert relativ zum Anfang des Blockes, dessen Nummer in BLK steht. Ist BLK Null, so werden Zeichen aus dem Texteingabepuffer interpretiert.

load (n --) 83
Die Inhalte von >IN und BLK, die den gegenwärtigen Quelltext angeben, werden gespeichert. Der Block mit der Nummer n wird dann zum Quelltext gemacht. Der Block wird interpretiert. Die Interpretation wird bei Ende des Blocks abgebrochen, sofern das nicht explizit geschieht. Dann wird der alte Inhalt nach BLK und >IN zurückgebracht. s.a. BLK >IN BLOCK

loadfile (-- addr)
Addr ist die Adresse einer Variablen, die auf das Forth-File zeigt, das gerade geladen wird. Diese Variable wird bei Aufruf von LOAD , THRU usw. auf das aktuelle File gesetzt.

name (-- addr)
Holt den nächsten String, der durch Leerzeichen eingeschlossen wird, aus dem Quelltext, wandelt ihn in Grossbuchstaben um und hinterlässt die Adresse addr, ab der der String im Speicher steht. s. WORD

notfound (addr --)
Ein deferred Wort, das aufgerufen wird, wenn der Text aus dem Quelltext weder als Name in der Suchreihenfolge gefunden wurde, noch als Zahl interpretiert werden kann. Kann benutzt werden, um eigene Zahl- oder String-eingabeformate zu erkennen. Ist mit NO.EXTENSIONS vorbesetzt. Dieses Wort bricht die Interpretation des Quelltextes ab und druckt die Fehlermeldung "haeh?" aus.

parse (char -- addr +n)
Liefert die Adresse addr und Länge +n des nächsten Strings im Quelltext, der durch den Delimiter char abgeschlossen wird. +n ist Null, falls der Quelltext erschöpft oder das erste Zeichen char ist. >IN wird verändert.

quit (--) 83
Entleert den Returnstack, schaltet den interpretierenden Zustand ein, akzeptiert Eingaben von der aktuellen Eingabeeinheit und beginnt die Interpretation des eingegebenen Textes.

source (-- addr +n)
liefert Anfang addr und maximale Länge +n des Quelltextes. Ist BLK Null, beziehen sich Anfang und Länge auf den Texteingabepuffer, sonst auf den Block, dessen Nummer in BLK steht und der in den Rechnerspeicher kopiert wurde. s. BLOCK >IN

state (-- addr) 83
Eine Variable, die den gegenwärtigen Zustand enthält. Der Wert Null zeigt den interpretierenden Zustand an, ein von Null verschiedener Wert den kompilierenden Zustand.

thru (n1 n2 --)
LOAD die Blöcke von n1 bis inklusive n2.

word (char -- addr)83
erzeugt einen counted String durch Lesen von Zeichen vom Quelltext, bis dieser erschöpft ist oder der Delimiter char auftritt. Der Quelltext wird nicht zerstört. Führende Delimiter werden ignoriert. Der gesamte String wird im Speicher beginnend ab Adresse addr als eine Sequenz von Bytes abgelegt. Das erste Byte enthält die Länge des Strings (0..255). Der String wird durch ein Leerzeichen beendet, das nicht in der Längenangabe enthalten ist. Ist der String länger als 255 Zeichen, so ist die Länge undefiniert. War der Quelltext schon erschöpft, als WORD aufgerufen wurde, so wird ein String der Länge Null erzeugt.
Wird der Delimiter nicht im Quelltext gefunden, so ist der Wert von >IN die Länge des Quelltextes. Wird der Delimiter gefunden, so wird >IN so verändert, dass >IN das Zeichen hinter dem Delimiter indiziert. #TIB wird nicht verändert.
Der String kann sich oberhalb von HERE befinden.

] (--) 83,I "right-bracket"
(--) compiling
Schaltet den kompilierenden Zustand ein. Der Text vom Quelltext wird sukzessive kompiliert. Typische Benutzung s. LITERAL

\ (--) I "skip-line"
(--) compiling
Ignoriere den auf dieses Wort folgenden Text bis zum Ende der Zeile. s. C/L

\\ (--) I "skip-screen"
(--) compiling
Ignoriere den auf dieses Wort folgenden Text bis zum Ende des Blockes. s. B/BLK

\needs (--) "skip-needs"
Wird in der folgenden Art benutzt:
\needs <name>
Wird <name> in der Suchreihenfolge gefunden, so wird der auf <name> folgende Text bis zum Ende der Zeile ignoriert. Wird <name> nicht gefunden, so wird die Interpretation hinter <name> fortgesetzt.
Beispiel: \needs Editor 1+ load
Lädt den folgenden Block, falls EDITOR im Dictionary nicht vorhanden ist.

Fehlerbehandlung

(error (string --) "paren-error"
Dieses Wort steht normalerweise in der Variablen ERRORHANDLER und wird daher bei ABORT" und ERROR" ausgeführt. string ist dann die Adresse des auf ABORT" bzw. ERROR" folgenden Strings. (ERROR gibt das letzte Wort des Quelltextes gefolgt von dem String auf dem Bildschirm aus. Die Position des letzten Wortes im Quelltext, bei dem der Fehler auftrat, wird in SCR und R# abgelegt.

?pairs (n1 n2 --) "question-pairs"
Ist n1 <> n2 , so wird die Fehlermeldung "unstructured" ausgegeben. Dieses Wort wird benutzt, um die korrekte Schachtelung der Kontrollstrukturen zu überprüfen.

?stack (--) "question-stack"
Prüft, ob der Stack über- oder leerläuft. Der Stack läuft leer, falls der Stackpointer auf eine Adresse oberhalb von S0 zeigt. In diesem Fall wird die Fehlermeldung "stack empty" ausgegeben. Der Stack läuft über, falls der Stackpointer zwischen HERE und HERE + $100 liegt. In diesem Fall wird die Fehlermeldung "tight stack" ausgegeben, falls mehr als 31 Werte auf dem Stack liegen. Ist das nicht der Fall, so versucht das System, das zuletzt definierte Wort zu vergessen und es wird die Fehlermeldung "dictionary full" ausgegeben.

abort (--) 83,I
Leert den Stack, führt END-TRACE STANDARDI/O und QUIT aus.

abort" (flag) 83,I,C "abort-quote"
(--) compiling
Wird in der folgenden Form benutzt:
flag Abort" ccc"
Wird ABORT" später ausgeführt, so geschieht nichts, wenn flag falsch ist. Ist flag wahr, so wird der Stack geleert und der Inhalt von ERRORHANDLER ausgeführt. Beachten Sie bitte, daß im Gegensatz zu ABORT kein END-TRACE ausgeführt wird.

error" (flag) I,C "error-quote"
(--) compiling
Dieses Wort entspricht ABORT" , jedoch mit dem Unterschied, daß der Stack nicht geleert wird.

errorhandler (-- adr)
adr ist die Adresse einer Uservariablen, deren Inhalt die Kompilationsadresse eines Wortes ist. Dieses Wort wird ausgeführt, wenn das flag, das ABORT" bzw. ERROR" verbrauchen, wahr ist. Der Inhalt von ERRORHANDLER ist normalerweise (ERROR.

warning (-- adr)

adr ist die Adresse einer Variablen. Ist der Inhalt der Variablen null, so wird die Warnung "exists" ausgegeben, wenn ein Wort redefiniert wird. Ist der Wert nicht null, so wird die Warnung unterdrückt. Kurioserweise ist werden die Warnungen also durch WARNING ON abgeschaltet.

Sonstiges

'abort (--) "tick-abort"
Dies ist ein deferred Wort, das mit NOOP vorbesetzt ist. Es wird in ABORT ausgeführt, bevor QUIT aufgerufen wird. Es kann benutzt werden, um "automatisch" selbst definierte Stacks zu löschen.

'cold (--) "tick-cold"
Dies ist ein deferred Wort, das mit NOOP vorbesetzt ist. Es wird in COLD aufgerufen, bevor die Einschaltmeldung ausgegeben wird. Es wird benutzt, um Geräte zu initialisieren oder Anwenderprogramme automatisch zu starten.

'quit (--) "tick-quit"
Dies ist ein deferred Wort, das normalerweise mit (QUIT besetzt ist. Es wird in QUIT aufgerufen, nachdem der Returnstack enleert und der interpretierende Zustand eingeschaltet wurde. Es wird benutzt, um spezielle Kommandointerpreter (wie z.B. im Tracer) aufzubauen.

'restart (--) "tick-restart"
Dies ist ein deferred Wort, das mit NOOP vorbesetzt ist. Es wird in RESTART aufgerufen, nachdem 'QUIT mit (QUIT besetzt wurde. Es wird benutzt, um Geräte nach einem Warmstart zu reinitialisieren.

(quit (--) "paren-quit"
Dieses Wort ist normalerweise der Inhalt von 'QUIT. Es wird von QUIT benutzt. Es akzeptiert eine Zeile von der aktuellen Eingabeeinheit, führt sie aus und druckt "ok" bzw. "compiling".

.status (--) "dot-status"
Dieses Wort ist ein deferred Wort, das vor dem Einlesen einer Zeile bzw. dem Laden eines Blocks ausgeführt wird. Es ist mit NOOP vorbesetzt und kann dazu benutzt werden, Informationen über den Systemzustand oder den Quelltext auszugeben.

bye (--)
Dieses Wort führt FLUSH und EMPTY aus. Anschliessend wird der Monitor des Rechners angesprungen oder eine andere implementationsabhängige Funktion ausgeführt.

cold (--)
Bewirkt den Kaltstart des Systems. Dabei werden alle nach der letzten Ausführung von SAVE definierten Worte entfernt, die Uservariablen auf den Wert gesetzt, den sie bei SAVE hatten, die Blockpuffer neu initialisiert, der Bildschirm gelöscht und die Einschaltmeldung "volksFORTH-83 rev..." ausgegeben. Anschliessend wird RESTART ausgeführt.

end-trace (--)
Schaltet den Tracer ab, der durch "patchen" der Next-Routine arbeitet. Die Ausführung des aufrufenden Wortes wird fortgesetzt.

makeview (-- 16b)
Ein deferred Wort, dem mit IS ein Wort zugewiesen wurde. Dieses Wort erzeugt aus dem gerade kompilierten Block eine 16b Zahl, die von Create in das Viewfeld eingetragen wird. Das deferred Wort wird benötigt, um das Fileinterface rausschmeißen zu können.

next-link (-- addr)
addr ist die Adresse einer Uservariablen, die auf die Liste aller Nextroutinen zeigt. Der Assembler erzeugt bei Verwendung des Wortes NEXT Code, für den ein Zeiger in diese Liste eingetragen wird. Die so entstandene Liste wird vom Tracer benutzt, um alle im System vorhandenen Kopien des Codes für NEXT zu modifizieren.

noop (--)
Tut gar nichts.

r# (-- adr) "r-sharp"
adr ist die Adresse einer Variablen, die den Abstand des gerade editierten Zeichens vom Anfang des gerade editierten Screens enthält. Vergleiche (ERROR und SCR.

restart (--)
Bewirkt den Warmstart des Systems. Es setzt 'QUIT , ERRORHANDLER und 'ABORT auf ihre normalen Werte und führt ABORT aus.

scr (-- adr) 83 "s-c-r"
adr ist die Adresse einer Variablen, die die Nummer des gerade editierten Screens enthält. Vergleiche R# (ERROR und LIST .

Massenspeicher

>drive (block #drv -- block')"to-drive"
block' ist die Nummer des Blocks block auf dem Laufwerk #drv, bezogen auf das aktuelle Laufwerk (Vergleiche OFFSET und DRIVE) . block' kann positiv oder negativ sein. Beispiel:

```
23 1 >drive block
```

holt den Block mit der Nummer 21 vom Laufwerk 1, egal welches Laufwerk gerade das aktuelle ist.

all-buffers (--)
Belegt den gesamten Speicherbereich von LIMIT abwärts bis zum oberen Ende des Returnstacks mit Blockpuffern. Siehe ALLOTBUFFER .

allotbuffer (--)
Fügt der Liste der Blockpuffer noch einen weiteren hinzu, falls oberhalb vom Ende des Retunstacks dafür noch Platz ist. FIRST wird entsprechend geändert. Vergleiche FREEBUFFER und ALL-BUFFERS .

b/blk (-- &1024) "bytes pro block"
Die Länge eines Blocks (bzw. Screens, immer 1 KByte) wird auf den Stack gelegt. Siehe B/BUF .

b/buf (-- n) "bytes pro buffer"
n ist die Länge eines Blockpuffers incl. der Verwaltungsinformationen des Systems.

blk/drv (-- n) "blocks pro drive"
n ist die Anzahl der auf dem aktuellen Laufwerk verfügbaren Blöcke.

block (u -- addr) 83
addr ist die Adresse des ersten Bytes des Blocks u in dessen Blockpuffer. Der Block u stammt aus dem File in ISFILE . BLOCK prüft den Pufferbereich auf die Existenz des Blocks Nummer u. Befindet sich der Block u in keinem der Blockpuffer, so wird er vom Massenspeicher in einen an ihn vergebenen Blockpuffer geladen. Falls der Block in diesem Puffer UPDATEd , wird er auf den Massenspeicher gesichert, bevor der Blockpuffer an den Block u vergeben wird. Nur die Daten im letzten Puffer, der über BLOCK oder BUFFER angesprochen wurde, sind sicher zugreifbar. Alle anderen Blockpuffer dürfen nicht mehr als gültig angenommen werden (möglicherweise existiert nur 1 Blockpuffer). Vorsicht ist bei Benutzung des Multitaskers geboten, da eine andere Task BLOCK oder BUFFER ausführen kann. Der Inhalt eines Blockpuffers wird nur auf den Massenspeicher gesichert, wenn der Block mit UPDATE als verändert gekennzeichnet wurde.

buffer (u -- adr) 83
Vergibt einen Blockpuffer an den Block u. addr ist die Adresse des ersten Bytes des Blocks in seinem Puffer.

Dieses Wort entspricht BLOCK , jedoch mit der Ausnahme, das, wenn der Block noch nicht im Speicher ist, er nicht vom Massenspeicher geholt wird. Daher ist der Inhalt dieses Blockpuffers undefiniert.

convey (1st.block last.block to.blk --)
Verschiebt die Blöcke von 1st.blk bis last.blk einschließlich nach to.blk folgende. Die Bereiche dürfen sich überlappen. Eine Fehlerbehandlung wird eingeleitet, wenn last.blk kleiner als 1st.blk ist.
Die Blöcke werden aus dem File in FROMFILE ausgelesen und in das File in ISFILE geschrieben. FROMFILE und ISFILE dürfen gleich sein.
Beispiel :

```
4 6 5 convey
```

kopiert die Blöcke 4,5,6 nach 5,6,7. Der alte Inhalt des Blockes 7 ist verloren, der Inhalt der Blöcke 4 und 5 ist gleich.

copy (u1 u2 --)
Der Block u1 wird in den Block u2 kopiert. Der alte Inhalt des Blocks u2 ist verloren. Vergleiche auch CONVEY

core? (blk file -- addr/false)"core-question"
Prüft, ob sich der Block blk des Files file in einem der Blockpuffer befindet und liefert dann die Adresse addr des ersten Datenbytes. Befindet sich der Block nicht im Speicher, so wird false als Ergebnis geliefert. Vergleiche BLOCK , BUFFER und ISFILE .

drive (#drv --)
Selektiert das Laufwerk mit der Nummer #drv als aktuelles Laufwerk. 0 BLOCK liefert die Adresse des ersten Blocks auf dem aktuellen Laufwerk. Vergleiche >DRIVE und OFFSET.

drv? (block -- #drv) "drive-question"
#drv ist die Nummer des Laufwerks auf dem sich der Block block befindet. Vergleiche OFFSET >DRIVE und DRIVE.

empty-buffers (--)
Löscht den Inhalt aller Blockpuffer. UPDATEd Blockpuffer werden nicht auf den Massenspeicher zurückgeschrieben.

first (-- addr)
addr ist die Adresse einer Variablen, in der sich ein Zeiger auf den Blockpuffer mit der niedrigsten Adresse befindet. Vergleiche ALLOTBUFFER

flush (--)
Sichert alle UPDATEd Blocks auf den Massenspeicher und löscht dann alle Blockpuffer. Vergleiche SAVE-BUFFERS und EMPTY-BUFFERS .

freebuffer (--)
Entfernt den Blockpuffer mit der niedrigsten Adresse aus der Liste der Blockpuffer und gibt den dadurch belegten Speicherbereich frei. FIRST wird entsprechend verändert (um B/BUF erhöht). Ist der Inhalt des Puffers UPDATEd , so wird er zuvor auf den Massenspeicher gesichert. Gibt es im System nur noch einen Blockpuffer, so geschieht nichts. Vergleiche ALLOTBUFFER .

fromfile (-- adr)
adr ist die Adresse einer Variablen, deren Wert die Nummer eines Files ist, aus dem CONVEY und COPY die Blöcke auslesen, um sie in das File, das in ISFILE steht, zu kopieren. Siehe das Kapitel zum Fileinterface

isfile (-- addr)
addr ist die Adresse einer Uservariablen, deren Wert die Nummer des aktuellen Files, auf das sich alle Massenspeicheroperationen beziehen, ist. Typisch entspricht die Nummer eines Files der Adresse seines File Control Blocks. Ist der Wert von FILE Null, so wird direkt, ohne ein File, auf den Massenspeicher (z.B. die Sektoren einer Diskette) zugegriffen. Siehe das Kapitel zum Fileinterface.

limit (-- addr)
Unterhalb von addr befinden sich die Blockpuffer. Das letzte Byte des obersten Blockpuffers befindet sich in addr-1. Vergleiche ALL-BUFFERS ALLOTBUFFER

offset (-- addr)
addr ist die Adresse einer Uservariablen, deren Inhalt zu der Blocknummer addiert wird, die sich bei Aufruf von BLOCK BUFFER usw. auf dem Stack befindet. OFFSET wird durch DRIVE verändert.

prev (-- addr)
addr ist die Adresse einer Variablen, deren Wert der Anfang der Liste aller Blockpuffer ist. Der erste Blockpuffer in der Liste ist der zuletzt durch BLOCK oder BUFFER vergebene.

r/w (addr block file n -- flag)"r-w"
Ein deferred Wort, bei dessen Aufruf das systemabhängige Wort ausgeführt wird, das einen Block vom Massenspeicher holt. Dabei ist addr die Anfangsadresse des Speicherbereichs für den Block block, file die Filenummer des Files, in dem sich der Block befindet und n=0, falls der Block vom Speicherbereich auf den Massenspeicher geschrieben werden soll. Ist n=1, so soll der Block vom Massenspeicher in den Speicherbereich gelesen werden. Das flag ist Falsch, falls kein Fehler auftrat, sonst Wahr.

save-buffers (--) 83
Sichert alle als UPDATEd gekennzeichneten Blöcke aus den Blockpuffern auf den Massenspeicher. Das UPDATE - Kennzeichen wird für alle Blöcke zurückgesetzt, die Blöcke werden jedoch nicht verändert und bleiben für weitere Zugriffe im Speicher erhalten.

update (--) 83
Der zuletzt mit BLOCK BUFFER usw. zugegriffene Block wird als verändert gekennzeichnet. Blöcke mit solcher Kennzeichnung werden auf den Massenspeicher zurückgeschrieben, wenn ihr Blockpuffer für einen anderen Block benötigt oder wenn SAVE-BUFFERS ausgeführt wird. Vergleiche PREV.

Atari ST - spezifische Worte

#col (-- addr) "number-col"
addr ist die Adresse einer Variablen, die die Nummer der aktuellen Cursorzeile enthält.

#esc (-- n) "number-escape"
n ist der Ascii-Wert für Escape.

#lf (-- n) "number-linefeed"
n ist der Ascii-Wert für Linefeed.

#row (-- addr) "number-row"
addr ist die Adresse einer Variablen, die die Nummer der aktuellen Cursorspalte enthält.

bconin (dev# -- char) "b-con-in"
char ist ein Zeichen, das vom Gerät mit der Nummer dev# eingelesen wird. BCONIN wartet (und hängt dadurch), bis ein Zeichen bereitsteht. Typische Gerätenummern sind:

0 PRT; Centronics-Schnittstelle (Druckerport)
1 AUX; RS232-Schnittstelle
2 CON; Tastatur und Bildschirm
3 MIDI; Midi-Schnittstelle
4 IKBD; Tastatur-Port

Gerätenummer 4 ist bei dieser Funktion nicht erlaubt.

bconout (char dev# --) "b-con-out"
char ist ein Zeichen, das an das Gerät mit der Nummer dev# ausgegeben wird. BCONIN wartet (hängt), bis das Zeichen ausgegeben wurde. Die Gerätenummern sind die gleichen wie bei BCONIN . Alle Gerätenummern sind erlaubt.

bconstat (dev# -- flag) "b-con-stat"
flag ist TRUE, wenn ein Zeichen auf dem Gerät mit der Nummer dev# bereitsteht. Sonst ist flag = FALSE. Die Gerätenummern sind die gleichen wie bei BCONIN . Die Gerätenummern 0 und 4 sind bei dieser Funktion nicht erlaubt.

bcostat (dev# -- flag) "b-co-stat"
flag ist TRUE, wenn das Gerät mit der Nummer dev# bereit ist, ein Zeichen zu empfangen. Sonst ist flag = FALSE. Die Gerätenummern sind die gleichen wie bei BCONIN . Alle Gerätenummern sind bei dieser Funktion erlaubt.

con! (8b --) "con-store"
gibt 8b auf die CONsole (Bildschirm) aus. Ascii-Werte < $20 werden als Steuercodes interpretiert.

curleft (--)
setzt den Cursor um eine Spalte nach links.

curoff (--)
schaltet den Cursor aus.

curon (--)
schaltet den Cursor ein.

currite (--)
setzt den Cursor um eine Spalte nach rechts.

display (--)
ein mit OUTPUT: definiertes Wort, das den Bildschirm als Ausgabegerät setzt, wenn es ausgeführt wird. Die Worte EMIT , CR , TYPE , DEL , PAGE , AT und AT? beziehen sich dann auf den Bildschirm.

drv0 (--)
Kurzform für 0 DRIVE .
Das Laufwerk 0 (bzw. A) wird als aktuelles Laufwerk selektiert. Siehe DRIVE.

drv1 (--)
Wie DRV0.

drvinit (--)
Ein Wort, das von RESTART ausgeführt wird. Dieses Wort kann dazu verwendet werden, Floppycontroller, Files o.Ä. zu initialieren.

getkey (-- n)
die unteren 7 Bit von n enthalten den Ascii-Code, die unteren 8 Bit den ANSI-Code (Ach Nein! Sch... IBM) des nächsten Tastendrucks, die oberen 8 Bit den Tastatur-Scancode. War keine Taste gedrückt, ist n = FALSE . (Vergleiche KEY? und KEY bzw. STKEY? und STKEY .)

keyboard (--)
ein mit INPUT: definiertes Wort, das die Tastatur als Eingabegerät setzt. Die Worte KEY , KEY? , DECODE und EXPECT beziehen sich auf die Tastatur. Siehe STKEY , STKEY? , STDECODE und STEXPECT .

rwabs (r/wf addr rec# -- flag) "r-w-abs"
Ist r/wf = TRUE, wird ein Block Daten ($400 bzw. &1024) Bytes vom Disketten-Sektor mit der Nummer rec# nach addr gelesen. Ist r/wf = FALSE, werden Daten von addr auf die Diskette geschrieben. Diese Routine wird von STR/W benutzt.

STat (row col --) "s-t-at"
positioniert den Cursor in die Zeile row und die Spalte col. Ein Fehler liegt vor, wenn row > $18 (&24) oder col > $40 (&64) ??? ist. Vergleiche AT .

STat? (-- row col) "s-t-at-question"
row ist die aktuelle Zeilennummer des Cursors, col die aktuelle Spaltennummer. Vergleiche AT? .

STcr (--) "s-t-c-r"
setzt den Cursor in die erste Spalte der nächsten Zeile. Ein PAUSE wird ausgeführt.

STdecode (addr pos1 key -- addr pos2) "s-t-decode"
wertet key aus. key wird in der Speicherstelle addr+pos1 abgelegt und als Echo auf dem Bildschirm ausgegeben. Die Variable SPAN und pos werden inkrementiert. Folgende Tasten werden besonders behandelt: Cursor rechts und Cursor links beeinflussen nur pos1 und den Cursor. Delete löscht das Zeichen unter dem Cursor und dekrementiert SPAN . Backspace löscht das Zeichen links vom Cursor und dekrementiert pos1 und SPAN . Insert fügt an der Cursorposition ein Leerzeichen ein. SPAN wird inkrementiert. Return positioniert den Cursor auf das letzte Zeichen. Vergleiche INPUT: und STexpect .

STdel (--) "s-t-del"
löscht ein Zeichen links vom Cursor. Vergleiche DEL .

STemit (8b --) "s-t-emit"
gibt 8b auf dem Bildschirm aus. Ein PAUSE wird ausgeführt. Alle Werte werden als Zeichen ausgegeben, Steuercodes sind nicht möglich, d. h. alle Werte < $20 werden als ATARI-Spezifische Zeichen ausgegeben. Vergleiche CON! und EMIT .

STexpect (addr len --) "s-t-expect"
erwartet len Zeichen vom Eingabegerät, die ab addr im Speicher abgelegt werden. Ein Echo der Zeichen wird ausgegeben. Return beendet die Eingabe vorzeitig. Ein abschließendes Leerzeichen wird immer ausgegeben. Die Länge der Zeichenkette wird in der Variablen SPAN übergeben. Vergleiche EXPECT .

STkey (-- 16b) "s-t-key"
wartet auf einen Tastendruck. Während der Wartezeit wird PAUSE ausgeführt. Die unteren 7 Bit von 16b enthalten den Ascii-Code, die unteren 8 Bit den ANSI-Code der Taste, die oberen 8 Bit den Tastatur-Scancode. Steuerzeichen werden nicht ausgewertet, sondern unverändert abgeliefert. Vergleiche KEY .

STkey? (-- flag) "s-t-key-question"
flag ist TRUE, wenn eine Taste gedrückt wurde, sonst FALSE . Vergleiche KEY? .

STpage (--) "s-t-page"
löscht den Bildschirm und positioniert den Cursor in die linke obere Ecke. Vergleiche PAGE .

STr/w (adr blk r/w file -- f) "s-t-r-w"
liest oder schreibt einen Block Daten ($400 oder &1024 Bytes) vom Diskettenblock blk nach adr bzw. von adr auf den Block blk. Ist r/w falsch, so werden die Daten von der Adresse adr geschrieben, ist r/w wahr, so werden sie nach Adresse adr gelesen. STR/W ermittelt das Laufwerk, auf dem sich der Block blk befindet.
file ist die Nummer eines Files, dem der zu bearbeitende Block entstammt. file muß Null sein, da dieses Wort nicht mit Files arbeiten kann !

STtype (addr len --)
gibt den String, der im Speicher bei addr beginnt und die Länge len hat, auf dem Bildschirm aus. Ein PAUSE wird ausgeführt. Vergleiche TYPE , OUTPUT: und STEMIT .

Multitasking

's (Tadr -- usradr) I "tick-s"
wird benutzt in der Form:
... <taskname> 's <username> ...
'S liest den Namen einer USERvariablen <username> und hinterläßt die Adresse usradr dieser USERvariablen in der durch Tadr gekennzeichneten Task. Typisch wird Tadr durch Ausführung von <taskname> erzeugt. Eine Fehlerbedingung liegt vor, wenn <username> nicht der Name einer USERvariablen ist. Vergleiche USER und TASK . 'S ist für den Zugriff auf USERvariablen einer Task durch eine andere Task vorgesehen.

activate (Tadr --)
aktiviert die Task, die durch Tadr gekennzeichnet ist, und weckt sie auf. Vergleiche SLEEP , STOP , PASS , PAUSE , UP@ , UP! und WAKE .

lock (semadr --)
Der Semaphor (eine VARIABLE), dessen Adresse auf dem Stack liegt, wird von der Task, die LOCK ausführt, blockiert. Dazu prüft LOCK den Inhalt des Semaphors. Zeigt der Inhalt an, daß eine andere Task den Semaphor blockiert hat, so wird PAUSE ausgeführt, bis der Semaphor freigegeben ist. Ist der Semaphor freigegeben, so schreibt LOCK das Kennzeichen der Task, die LOCK ausführt, in den Semaphor und sperrt ihn damit für alle anderen Tasks. Den FORTH-Code zwischen semadr LOCK ... und ... semadr UNLOCK kann also immer nur eine Task zur Zeit ausführen. Semaphore schützen gemeinsam benutzte Geräte (z.B. den Drucker) vor dem gleichzeitigen Zugriff durch verschiedene Tasks. Vergleiche UNLOCK , RENDEZVOUS und die Beschreibung des Taskers.

multitask (--)
schaltet das Multitasking ein. Das Wort PAUSE ist dann keine NOOP -Funktion mehr, sondern gibt die Kontrolle über die FORTH-Maschine an eine andere Task weiter.

pass (n_0 .. n_{r-1} Tadr r --)
aktiviert die Task, die durch Tadr gekennzeichnet ist, und weckt sie auf. r gibt die Anzahl der Parameter n_0 bis n_{r-1} an, die vom Stack der PASS ausführenden Task auf den Stack der durch Tadr gekennzeichneten Task übergeben werden. Die Parameter n_0 bis n_{r-1} stehen dieser Task dann in der gleichen Reihenfolge auf ihrem Stack zur weiteren Verarbeitung zur Verfügung. Vergleiche ACTIVATE .

pause (--)
ist eine NOOP -Funktion, wenn der Singletask-Betrieb eingeschaltet ist; bewirkt jedoch, nach Ausführung von MULTITASK , daß die Task, die PAUSE ausführt, die Kontrolle über die FORTH-Maschine an eine andere Task abgibt. Existiert nur eine Task, oder schlafen alle anderen Tasks, so wird die Kontrolle unverzüglich an die Task zurückgegeben, die PAUSE ausführte. Ist

mindestens eine andere Task aktiv, so wird die Kontrolle von dieser übernommen und erst bei Ausführung von PAUSE oder STOP in dieser Task an eine andere Task weitergegeben. Da die Tasks ringförmig miteinander verkettet sind, erhält die Task, die zuerst PAUSE ausführte, irgendwann die Kontrolle zurück. Eine Fehlerbedingung liegt vor, wenn eine Task weder PAUSE noch STOP ausführt. Vergleiche STOP , MULTITASK und SINGLETASK .

rendezvous (semadr --)
gibt den Semaphor (die VARIABLE) mit der Adresse semadr frei (siehe UNLOCK) und führt PAUSE aus, um anderen Tasks den Zugriff auf das, durch diesen Semaphor geschützte, Gerät zu ermöglichen. Anschließend wird LOCK ausgeführt, um das Gerät zurück zu erhalten.

singletask (--)
schaltet das Multitasking aus. PAUSE ist nach Ausführung von SINGLETASK eine NOOP -Funktion. Eine Fehlerbedingung besteht, wenn eine Hintergrund-Task SINGLETASK ohne anschließendes MULTITASK ausführt, da die Main- oder Terminal-Task dann nie mehr die Kontrolle bekommt. Vergleiche UP@ und UP! .

sleep (Tadr --)
bringt die Task, die durch Tadr gekennzeichnet ist, zum Schlafen. SLEEP hat den gleichen Effekt, wie die Ausführung von STOP durch die Task selbst. Der Unterschied ist, daß STOP in der Regel am Ende des Jobs der Task ausgeführt wird, SLEEP trifft die Task zu einem nicht vorhersehbaren Zeitpunkt, so daß die laufende Arbeit der Task abgebrochen wird. Vergleiche WAKE .

stop (--)
bewirkt, daß die Task, die STOP ausführt, sich schlafen legt. Wichtige Zeiger der FORTH-Maschine, die den Zustand der Task kennzeichnen, werden gerettet, dann wird die Kontrolle an die nächste Task abgegeben, deren Zeiger wieder der FORTH-Maschine übergeben werden, so daß diese Task ihre Arbeit an der alten Stelle aufnehmen kann. PAUSE führt diese Aktionen ebenfalls aus, der Unterschied zu STOP ist, daß die ausführende Task bei PAUSE aktiv, bei STOP hingegen schlafend hinterlassen wird. Vergleiche PAUSE , WAKE und SLEEP .

Task (rlen slen --)
ist ein definierendes Wort, das in der Form:
rlen slen Task <cccc>
benutzt wird. TASK erzeugt einen Arbeitsbereich für einen weiteren Job, der gleichzeitig zu schon laufenden Jobs ausgeführt werden soll. Die Task erhält den Namen cccc, hat einen Stack-Bereich der Länge slen und einen Returnstack-Bereich der Länge rlen. Im Stack-Bereich liegen das Task-eigene Dictionary einschließlich PAD, das in Richtung zu höheren Adressen wächst, und der

Daten-Stack, der zu niedrigen Adressen wächst. Im Returnstack-Bereich befinden sich die Task-eigene USER-Area (wächst zu höheren Adressen) und der Returnstack, der gegen kleinere Adressen wächst. Eine Task ist ein verkleinertes Abbild des FORTH-Systems, allerdings ohne den Blockpuffer-Bereich, der von allen Tasks gemeinsam benutzt wird. Zur Zeit ist es nicht zugelassen, daß Jobs einer Hintergrundtask kompilieren. Die Task ist nur der Arbeitsbereich für einen Hintergrund-Job, nicht jedoch der Job selbst.

Die Ausführung von cccc in einer beliebigen Task hinterläßt die gleiche Adresse, die die Task cccc selbst mit UP@ erzeugt und ist zugleich die typische Adresse, die von LOCK , UNLOCK und RENDEZVOUS im Zusammenhang mit Semaphoren verwendet wird, bzw. von 'S , ACTIVATE , PASS , SLEEP und WAKE erwartet wird.

tasks (--)
listet die Namen aller eingerichteten Tasks und zeigt, ob sie schlafen oder aktiv sind.

unlock (semadr --)
gibt den Semaphor (die VARIABLE), dessen Adresse auf dem Stack ist, für alle Tasks frei. Ist der Semaphor im Besitz einer anderen Task, so wartet UNLOCK mit PAUSE auf die Freigabe. Vergleiche LOCK und die Beschreibung des Taskers.

up@ (-- Tadr) "u-p-fetch"
liefert die Adresse Tadr des ersten Bytes der USER-Area der Task, die UP@ ausführt. Siehe TASK . In der USER-Area sind Variablen und andere Datenstrukturen hinterlegt, die jede Task für sich haben muß. Vergleiche UP! .

up! (adr --) "u-p-store"
richtet den UP (User Pointer) der FORTH-Maschine auf adr. Vorsicht ist bei der Verwendung von UP! in Hintergrund-Tasks geboten. Vergleiche UP@ .

wake (Tadr --)
weckt die Task, die durch Tadr gekennzeichnet ist, auf. Die Task führt ihren Job dort weiter aus, wo sie durch SLEEP angehalten wurde oder wo sie sich selbst durch STOP beendet hat (Vorsicht!). Vergleiche SLEEP , STOP , ACTIVATE und PASS .

Input und Output Worte

#bs (-- n) "number-b-s"
n ist der Wert, den man durch KEY erhält, wenn die Backspace-Taste gedrückt wird.

#cr (-- n) "number-c-r"
n ist der Wert, den man durch KEY erhält, wenn die Return- (Enter-) Taste gedrückt wird.

#tib (-- adr) 83 "number-t-i-b"
adr ist die Adresse einer Variablen, die die Länge des aktuellen Textes im Text-Eingabe-Puffer enthält. Vergleiche TIB .

-trailing (adr +n0 -- adr +n1) 83 "minus-trailing"
adr ist die Anfangsadresse und +n0 die Länge eines Strings. -TRAILING verändert +n0 so zu +n1, daß eventuell am Ende vorhandene Leerzeichen nicht mehr in der Stringlänge +n1 enthalten sind. Der String selbst bleibt unangetastet. Ist +n0 = 0 oder besteht der ganze String aus Leerzeichen, so ist +n1 = 0.

. (n --) 83 "dot"
n wird vorzeichenbehaftet ausgegeben.

." (--) 83 I C "dot-quote"
(--) compiling
wird in :-Definitionen in der Form verwendet:
: <name> ... ." cccc" ... ;
Der String cccc wird bis zum abschließenden " so kompiliert, daß bei Ausführung von <name> der String cccc ausgedruckt wird. Das Leerzeichen nach ." und das abschließende " sind Pflicht und gehören nicht zum String.

.((--) 83 I "dot-paren"
(--) compiling
wird in der Form :
... .(cccc) ...
benutzt und druckt den String cccc bis zur abschließenden Klammer sofort aus. Das Leerzeichen nach .(und die schließende Klammer werden nicht mit ausgedruckt.

.r (n +n --) "dot-r"
druckt die Zahl n in einem +n Zeichen langen Feld mit Vorzeichen rechtsbündig aus. Reicht +n nicht zur Darstellung der Zahl aus, so wird über den rechten Rand hinaus ausgegeben. Die Zahl n wird in jedem Fall vollständig dargestellt.

>tib (-- adr) "to-tib"
adr ist die Adresse eines Zeigers auf den Text-Eingabe-Puffer. Siehe TIB

?cr (--) "question-c-r"
prüft, ob in der aktuellen Zeile mehr als C/L Zeichen ausgegeben wurden und führt in diesem Fall CR aus.

at (row col --)
positioniert die Schreibstelle des Ausgabegerätes in die Zeile row und die Spalte col. AT ist eines der über OUTPUT vektorisierten Worte. Siehe AT?

at? (-- row col) "at-question"
ermittelt die aktuelle Position der Schreibstelle des Ausgabegerätes und legt Zeilen- und Spaltennummer auf den Stack. Eines der OUTPUT-Worte.

base (-- adr) 83 U
adr ist die Adresse einer Uservariablen, die die Zahlenbasis enthält, die zur Wandlung von Zahlenein- und -ausgaben benutzt wird.

bl (-- n) "b-l"
n ist der ASCII-Wert für ein Leerzeichen.

c/l (-- +n) "characters-per-line"
+n ist die Anzahl der Zeichen pro Screenzeile. Aus historischen Gründen ist dieser Wert &64 bzw. $40 .

col (-- +n)
+n ist die Spalte in der sich die Schreibstelle des Ausgabegerätes gerade befindet. Vergleich ROW und AT? .

cr (--) 83 "c-r"
bewirkt, daß die Schreibstelle des Ausgabegerätes an den Anfang der nächsten Zeile verlegt wird. Eines der OUTPUT-Worte.

d. (d --) "d-dot"
druckt d vorzeichenbehaftet aus.

d.r (d +n --) "d-dot-r"
druckt d vorzeichenbehaftet in einem +n Zeichen breiten Feld rechtsbündig aus. Reicht +n nicht zur Darstellung der Zahl aus, so wird über den rechten Rand des Feldes hinaus ausgegeben. Die Zahl d wird in jedem Fall vollständig dargestellt.

decimal (--)
stellt BASE auf den Wert $0A bzw. &10 ein. Alle Zahlenein- und -ausgaben erfolgen nun im dezimalen System.

decode (adr +n0 key -- adr +n1)
wertet key aus. Typisch werden normale druckbare ASCII-Zeichen in die Speicherstelle adr + +n0 übertragen, als Echo zum Ausgabegerät gesandt und +n0 inkrementiert. Andere Zeichen (#BS #CR Steuercodes) können andere Aktionen zur Folge haben. Eines der über INPUT vektorisierten Worte. Wird von EXPECT benutzt. Siehe STDECODE .

del (--)
löscht das zuletzt ausgegebene Zeichen. Eines der OUTPUT-Worte. Bei Druckern ist die korrekte Funktion nicht immer garantiert.

emit (16b --) 83
die unteren 8 Bit werden an das Ausgabegerät ausgegeben. Ist das Zeichen nicht druckbar (insbesondere Steuercodes), so wird ein Punkt ausgegeben. Eines der OUTPUT-Worte.

expect (adr +n --) 83
empfängt Zeichen und speichert sie ab Adresse adr im Speicher. Die Übertragung wird beendet, bis ein #CR erkannt oder +n Zeichen übertragen wurden. Ein #CR wird nicht mit abgespeichert. Ist +n = 0 , so werden keine Zeichen übertragen. Alle Zeichen werden als Echo, statt #CR wird ein Leerzeichen ausgegeben. Eines der INPUT-Worte. Vergleiche SPAN

hex (--)
stellt BASE auf $10 bzw. &16 . Alle Zahlenein- und -ausgaben erfolgen im hexadezimalen Zahlensystem.

input (-- adr) U
adr ist die Adresse einer Uservariablen, die einen Zeiger auf ein Feld von vier Kompilationsadressen enthält, die für ein Eingabegerät die Funktionen KEY KEY? DECODE und EXPECT realisieren. Vergleiche die gesonderte Beschreibung der INPUT- und OUTPUT-Struktur.

key (-- 16b) 83
empfängt ein Zeichen vom Eingabegerät. Die niederwertigen 8 Bit enthalten den ASCII-Code des zuletzt empfangenen Zeichens. Alle gültigen ASCII-Codes können empfangen werden. Die oberen 8 Bit enthalten systemspezifische Informationen. Es wird kein Echo ausgesandt. KEY wartet, bis tatsächlich ein Zeichen empfangen wurde. Eines der INPUT-Worte.

key? (-- flag) "key-question"
flag ist TRUE , falls ein Zeichen zur Eingabe bereitsteht, sonst ist flag FALSE. Eines der INPUT-Worte.

list (u --) 83
zeigt den Inhalt des Screens u auf dem aktuellen Ausgabegerät an. SCR wird auf u gesetzt. Siehe BLOCK.

l/s (-- +n) "lines-per-screen"
+n ist die Anzahl der Zeilen pro Screen.

ouput (-- adr) U
adr ist die Adresse einer Uservariablen, die einen Zeiger auf sieben Kompilationsadressen enthält, die für ein Ausgabegerät die Funktionen EMIT CR TYPE DEL PAGE AT und AT? realisieren. Vergleiche die gesonderte Beschreibung der Ouput-Struktur.

page (--)
bewirkt, daß die Schreibstelle des Ausgabegerätes auf eine leere neue Seite bewegt wird. Eines der OUTPUT-Worte.

query (--) 83
Zeichen werden von einem Eingabegerät geholt und in den Text-Eingabe-Puffer übertragen, der bei TIB beginnt. Die Übertragung endet beim Empfang von #CR oder wenn die Länge des Text-Eingabe-Puffers erreicht wird. Der Inhalt von >IN und BLK wird zu 0 gesetzt und #TIB enthält die Zahl der empfangenen Zeichen. Um Text aus dem Puffer zu lesen, kann WORD benutzt werden. Siehe EXPECT.

row (-- +n)
+n ist die Zeile, in der sich die Schreibstelle des Ausgabegerätes befindet. Vergleiche COL und AT? .

space (--) 83
sendet ein Leerzeichen an das Ausgabegerät.

spaces (+n --) 83
sendet +n Leerzeichen an das Ausgabegerät.

span (-- adr) 83
Der Inhalt der Variablen SPAN gibt an, wieviele Zeichen vom letzten EXPECT übertragen wurden. Siehe EXPECT .

standardi/o (--) "standard-i-o"
stellt sicher, daß die beim letzten SAVE bestimmten Ein- und Ausgabegeräte wieder eingestellt sind.

stop? (-- flag) "stop-question"
Ein komfortables Wort, das es dem Benutzer gestattet, einen Programmablauf anzuhalten oder zu beenden. Steht vom Eingabegerät ein Zeichen zur Verfügung, so wird es eingelesen. Ist es #ESC oder CTRL-C , so ist flag TRUE, sonst wird auf das nächste Zeichen gewartet. Ist dieses jetzt #ESC oder CTRL-C , so wird STOP? mit TRUE verlassen, sonst mit FALSE . Steht kein Zeichen zur Verfügung, so ist flag FALSE .

tib (-- adr) 83
liefert die Adresse des Text-Eingabe-Puffers. Er wird benutzt, um die Zeichen vom Quelltext des aktiven Gerätes zu halten. Siehe >TIB .

type (adr +n --) 83
sendet +n Zeichen, die ab adr im Speicher abgelegt sind, an ein Ausgabegerät. Ist +n = 0 , so wird nichts ausgegeben.

u. (u --) "u-dot"
die Zahl u wird vorzeichenlos ausgegeben.

u.r (u +n --) "u-dot-r"

die Zahl u wird vorzeichenlos rechtsbündig in einem Feld der Breite +n ausgegeben. Reicht +n zur Darstellung von u nicht aus, so wird über den rechten Rand hinaus ausgegeben.

Erweiterte Adressierung

Auf den modernen Rechnern stehen zumeist mehr als 64 Kbyte Speicher zur Verfügung. Forth ist gewöhnlich auf einen Adreßbereich von 64 Kbyte beschränkt, da die Stacks nur 16 Bit breit sind. Daher kann Forth zunächst nicht sehr gut mit den 32Bit Adressen, die für solche Betriebssysteme erforderlich sind, umgehen.
Das Forth wird bei diesen Betriebssystemen normalerweise nicht an den Anfang des Speichers geladen. Da die meisten Operationen nur 16Bit Adressen verarbeiten (wie ! @ TYPE COUNT usw.), zählen sie diese Adressen ab dem Anfang des Forthsystems.
Die folgenden Operationen verarbeiten dagegen absolute 32Bit Adressen.

forthstart (-- laddr)
laddr ist die (erweiterte) Anfangsadresse des Forthsystems im Speicher des Rechners. d.h.
forthstart l@ und
0 @
lesen dieselbe Speicherzelle aus.

l! (16b laddr --) " long-store "
16b werden in den Speicher ab Adresse laddr geschrieben. Im Gegensatz zu ! wird bei dieser Operation die Adresse ab dem Anfang des Speichers und nicht ab dem Anfang des Forthsystems gezählt. Siehe ! und FORTHSTART

l2! (32b laddr --) " long-two-store "
Analog zu L! , jedoch werden 32b geschrieben.

l2@ (laddr -- 32b) " long-two-fetch "
Analog zu L@ , jedoch werden 32b gelesen.

l@ (laddr -- 16b) " long-fetch "
16b wurden aus dem Speicher ab Adresse laddr gelesen. Im Gegensatz zu ! wird bei dieser Operation die Adresse ab dem Anfang des Speichers und nicht ab dem Anfang des Forthsystems gezählt. Siehe @ und FORTHSTART.

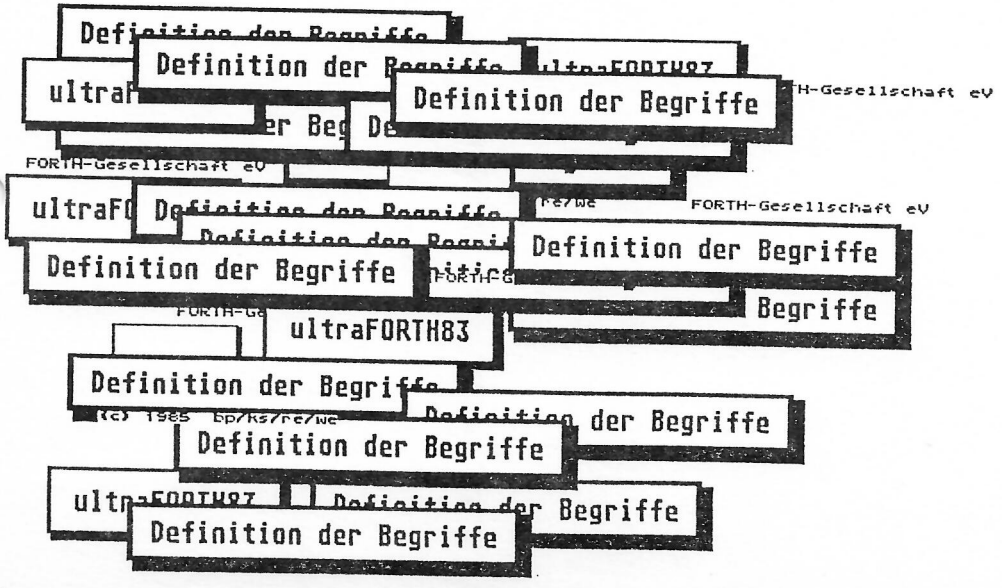
lc! (8b laddr --) " long-c-store "
Analog zu L! , jedoch werden 8b geschrieben.

lc@ (8b laddr --) " long-c-fetch "
Analog zu L@ , jedoch werden 8b gelesen.

lcmove (laddr1 laddr2 u --) " long-cmove "
Beginnend bei Adresse laddr1 werden u Bytes zur Adresse laddr2 kopiert. Wenn u Null ist, wird nichts kopiert. Siehe L! und CMOVE .

ln+! (w laddr --) " l-n-plus-store "
w wird zu dem Wert in der Adresse laddr addiert. Benutzt die + Operation. Die Summe wird in die Adresse ab laddr geschrieben. Siehe L! und +! .

Wortname	S.	Gruppe	S.
fromfile	III-42	Massenspeich	III-40
fsel_input	A-9	GEM	A-4
function	A-4	GEM	A-4
GDP	A-11	GEM	A-4
GEM	A-4	GEM	A-4
get_pixel	A-13	GEM	A-4
getkey	III-45	ST-spezifisch	III-44
global	A-4	GEM	A-4
graf_dragbox	A-8	GEM	A-4
graf_growbox	A-8	GEM	A-4
graf_handle	A-5	GEM	A-4
graf_mkstate	A-9	GEM	A-4
graf_mouse	A-8	GEM	A-4
graf_movebox	A-8	GEM	A-4
graf_shrinkbox	A-8	GEM	A-4
graf_slidebox	A-8	GEM	A-4
graf_watchbox	A-8	GEM	A-4
grexit	A-5	GEM	A-4
grhandle	A-4	GEM	A-4
grinit	A-5	GEM	A-4
gtext	A-11	GEM	A-4
hallot	III-27	Heap	III-27
hardcopy	A-14	GEM	A-4
heap	III-27	Heap	III-27
heap?	III-27	Heap	III-27
here	III-23	Dictionary	III-22
hex	III-53	In/Output	III-51
hide	III-23	Dictionary	III-22
hide_c	A-5	GEM	A-4
hold	III-16	Strings	III-15
I	III-29	Kontroll	III-28
IF	III-29	Kontroll	III-28
immediate	III-31	Compiler	III-31
include		Fileinterface	A-33
inpath	A-9	GEM	A-4
input	III-53	In/Output	III-51
Input:	III-19	Datentypen	III-18
insel	A-9	GEM	A-4
insert		Strings	A-47
interpret	III-34	Interpreter	III-33
intin	A-4	GEM	A-4
intout	A-4	GEM	A-4
Is	III-20	Datentypen	III-18
isfile	III-42	Massenspeich	III-40
J	III-29	Kontroll	III-28
justified	A-11	GEM	A-4
k		Tools	II-41
key	III-53	In/Output	III-51
key?	III-53	In/Output	III-51
keyboard	III-45	ST-spezifisch	III-44
l		Editor	A-1
l/s	III-53	In/Output	III-51
Label		Assembler	A-17
last	III-23	Dictionary	III-22
ldis		Disassembler	A-31


Definition der Begriffe
ultraFORTH83
FORTH-Gesellschaft eV
(c) 1985 bp/ks/re/we

Entscheidungskriterien

Bei Konflikten läßt sich das Standardteam von folgenden Kriterien in Reihenfolge ihrer Wichtigkeit leiten:

1. Korrekte Funktion - bekannte Einschränkungen, Eindeutigkeit
2. Transportabilität - wiederholbare Ergebnisse, wenn Programme zwischen Standardsystemen portiert werden
3. Einfachheit
4. Klare, eindeutige Namen - die Benutzung beschreibender statt funktionaler Namen, zB [COMPILE] statt 'c, und ALLOT statt dp+!
5. Allgemeinheit
6. Ausführungsgeschwindigkeit
7. Kompaktheit
8. Kompilationsgeschwindigkeit
9. Historische Kontinuität
10. Aussprechbarkeit
11. Verständlichkeit - es muß einfach gelehrt werden können

Definition der Begriffe

Es werden im allgemeinen die amerikanischen Begriffe beibehalten, es sei denn, der Begriff ist bereits im Deutschen geläufig. Wird ein deutscher Begriff verwendet, so wird in Klammern der engl. Originalbegriff beigefügt; wird der Originalbegriff beibehalten, so wird in Klammern eine möglichst treffende Übersetzung angegeben.

Adresse, Byte (address, byte)
: Eine 16bit Zahl ohne Vorzeichen, die den Ort eines 8bit Bytes im Bereich <0...65,535> angibt. Adressen werden wie Zahlen ohne Vorzeichen manipuliert.
Siehe: "Arithmetik, 2er-komplement"

Adresse, Kompilation (address, compilation)
: Der Zahlenwert, der zur Identifikation eines Forth Wortes kompiliert wird. Der Adressinterpreter benutzt diesen Wert, um den zu jedem Wort gehörigen Maschinencode aufzufinden.

Adresse, Natürliche (address, native machine)
: Die vorgegebene Adressdarstellung der Computerhardware.

Adresse, Parameterfeld (address, parameter field) "pfa"
: Die Adresse des ersten Bytes jedes Wortes, das für das Ablegen von Kompilationsadressen (bei :-definitionen) oder numerischen Daten bzw. Textstrings usw. benutzt wird.

anzeigen (display)
: Der Prozess, ein oder mehrere Zeichen zum aktuellen Ausgabegerät zu senden. Diese Zeichen werden normalerweise auf einem Monitor angezeigt bzw. auf einem Drucker gedruckt.

Arithmetik, 2er-komplement (arithmetic, two's complement)
: Die Arithmetik arbeitet mit Zahlen in 2er-komplementdarstellung; diese Zahlen sind, je nach Operation, 16bit oder 32bit weit. Addition und Subtraktion von 2er-komplementzahlen ignorieren Überlaufsituationen. Dadurch ist es möglich, daß die gleichen Operatoren benutzt werden können, gleichgültig, ob man die Zahl mit Vorzeichen (<-32,768...32,767> bei 16bit) oder ohne Vorzeichen (<0...65,535> bei 16bit) benutzt.

Block (block)
: Die 1024byte Daten des Massenspeichers, auf die über Blocknummern im Bereich <0...Anzahl_existenter_Blöcke-1> zugegriffen wird. Die exakte Anzahl der Bytes, die je Zugriff auf den Massenspeicher übertragen werden, und die Übersetzung von Blocknummern in die zugehörige Adresse des Laufwerks und des physikalischen Satzes, sind rechnerabhängig.
Siehe: "Blockpuffer" und "Massenspeicher"

Blockpuffer (block buffer)
Ein 1024byte langer Hauptspeicherbereich, in dem ein Block vorübergehend benutzbar ist. Ein Block ist in höchstens einem Blockpuffer enthalten.

Byte (byte)
Eine Einheit von 8bit. Bei Speichern ist es die Speicherkapazität von 8bits.

Kompilation (compilation)
Der Prozess, den Quelltext in eine interne Form umzuwandeln, die später ausgeführt werden kann. Wenn sich das System im Kompilationszustand befindet, werden die Kompilationsadressen von Worten im Dictionary abgelegt, so dass sie später vom Adresseninterpreter ausgeführt werden können. Zahlen werden so kompiliert, daß sie bei Ausführung auf den Stack gelegt werden. Zahlen werden aus dem Quelltext ohne oder mit negativem Vorzeichen akzeptiert und gemäß dem Wert von BASE umgewandelt.
Siehe: "Zahl", "Zahlenumwandlung", "Interpreter, Text" und "Zustand"

Definition (Definition)
Siehe: "Wortdefinition"

Dictionary (Wörterbuch)
Eine Struktur von Wortdefinitionen, die im Hauptspeicher des Rechners angelegt ist. Sie ist erweiterbar und wächst in Richtung höherer Speicheradressen. Einträge sind in Vokabularen organisiert, so daß die Benutzung von Synonymen möglich ist, d.h. gleiche Namen können, in verschiedenen Vokabularen enthalten, vollkommen verschiedene Funktionen auslösen.
Siehe: "Suchreihenfolge"

Division, floored (division, floored)
Ganzzahlige Division, bei der der Rest das gleiche Vorzeichen hat wie der Divisor oder gleich Null ist; der Quotient wird gegen die nächstkleinere ganze Zahl gerundet.
Bemerkung: Ausgenommen von Fehlern durch Überlauf gilt:
N1 N2 SWAP OVER /MOD ROT * + ist identisch mit N1.
Siehe: "floor, arithmetisch"

Beispiele:

Dividend	Divisor	Rest	Quotient
10	7	3	1
-10	7	4	-2
10	-7	-4	-2
-10	-7	-3	1

Empfangen (receive)
Der Prozess, der darin besteht, ein Zeichen von der aktuellen Eingabeeinheit zu empfangen. Die Anwahl einer Einheit ist rechnerabhängig.

Falsch (false)
Die Zahl Null repräsentiert den "Falschzustand" eines Flags.

Fehlerbedingung (error condition)
Eine Ausnahmesituation, in der ein Systemverhalten erfolgt, das nicht mit der erwarteten Funktion übereinstimmt. Im der Beschreibung der einzelnen Worte sind die möglichen Fehlerbedingungen und das dazugehörige Systemverhalten beschrieben.

Flag (logischer Wert)
Eine Zahl, die eine von zwei möglichen Werten hat, falsch oder wahr.
Siehe: "Falsch" "Wahr"

Floor, arithmetic
Z sei eine reelle Zahl. Dann ist der Floor von Z die größte ganze Zahl, die kleiner oder gleich Z ist.
Der Floor von +0,6 ist 0
Der Floor von -0,4 ist -1

Glossar (glossary)
Eine umgangssprachliche Beschreibung, die die zu einer Wortdefinition gehörende Aktion des Computers beschreibt - die Beschreibung der Semantik des Wortes.

Interpreter, Adressen (interpreter, address)
Die Maschinencodeinstruktionen, die die kompilierten Wortdefinitionen ausführen, die aus Kompilationsadressen bestehen.

Interpreter, Text (interpreter, text)
Eine Wortdefinition, die immer wieder einen Wortnamen aus dem Quelltext holt, die zugehörige Kompilationsadresse bestimmt und diese durch den Adressinterpreter ausführen läßt. Quelltext, der als Zahl interpretiert wird, hinterläßt den entsprechenden Wert auf dem Stack.
Siehe: "Zahlenumwandlung"

Kontrollstrukturen (structure, control)
Eine Gruppe von Worten, die, wenn sie ausgeführt werden, den Programmfluss verändern.
Beispiele von Kontrollstrukturen sind:
```
DO ... LOOP
BEGIN ... WHILE ... REPEAT
IF ... ELSE ... THEN
```

laden (load)
Das Umschalten des Quelltextes zum Massenspeicher. Dies ist die übliche Methode, dem Dictionary neue Definitionen hinzuzufügen.

Massenspeicher (mass storage)
Speicher, der ausserhalb des durch FORTH adressierbaren Bereiches liegen kann. Auf den Massenspeicher wird in Form von 1024byte grossen Blöcken zugegriffen. Auf einen Block

kann innerhalb des Forth-Adressbereichs in einem Blockpuffer zugegriffen werden. Wenn ein Block als verändert (UPDATE) gekennzeichnet ist, wird er letztendlich wieder auf den Massenspeicher zurückgeschrieben.

Programm (program)
Eine vollständige Ablaufbeschreibung in FORTH-Quelltext, um eine bestimmte Funktion zu realisieren.

Quelltext (input stream)
Eine Folge von Zeichen, die dem System zur Bearbeitung durch den Textinterpreter zugeführt wird. Der Quelltext kommt üblicherweise von der aktuellen Eingabeeinheit (über den Texteingabepuffer) oder dem Massenspeicher (über einen Blockpuffer). BLK, >IN, TIB und #TIB charakterisieren den Quelltext. Worte, die BLK, >IN, TIB oder #TIB benutzen und/oder verändern, sind dafür verantwortlich, die Kontrolle des Quelltextes aufrechtzuerhalten oder wiederherzustellen.
Der Quelltext reicht von der Stelle, die durch den Relativzeiger >IN angegeben wird, bis zum Ende. Wenn BLK Null ist, so befindet sich der Quelltext an der Stelle, die durch TIB adressiert wird, und er ist #TIB Bytes lang. Wenn BLK ungleich Null ist, so ist der Quelltext der Inhalt des Blockpuffers, der durch BLK angegeben ist, und er ist 1024byte lang.

Rekursion (recursion)
Der Prozess der direkten oder indirekten Selbstreferenz.

Screen (Bildschirm)
Ein Screen sind Textdaten, die zum Editieren aufbereitet sind. Nach Konvention besteht ein Screen aus 16 Zeilen zu je 64 Zeichen. Die Zeilen werden von 0 bis 15 durchnumeriert. Screens enthalten normalerweise Quelltext, können jedoch auch dazu benutzt werden, um Massenspeicherdaten zu betrachten. Das erste Byte eines Screens ist gleichzeitig das erste Byte eines Massenspeicherblocks; dies ist auch der Anfangspunkt für Quelltextinterpretation während des Ladens eines Blocks.

Suchreihenfolge (search order)
Eine Spezifikation der Reihenfolge, in der ausgewählte Vokabulare im Dictionary durchsucht werden. Die Suchreihenfolge besteht aus einem auswechselbaren und einem festen Teil, wobei der auswechselbare Teil immer als erstes durchsucht wird. Die Ausführung eines Vokabularnamens macht es zum ersten Vokabular in der Suchreihenfolge, wobei das Vokabular, das vorher als erstes durchsucht worden war, verdrängt wird. Auf dieses erste Vokabular folgt, soweit spezifiziert, der feste Teil der Suchreihenfolge, der danach durchsucht wird. Die Ausführung von ALSO übernimmt das Vokabular im auswechselbaren Teil in den festen Teil der Suchreihenfolge. Das Dictionary wird immer dann durchsucht, wenn ein Wort durch seinen Namen aufgefunden werden soll.

stack, data (Datenstapel)
Eine "Zuletzt-rein, Zuerst-raus" (last-in, first-out) Struktur, die aus einzelnen 16bit Daten besteht. Dieser Stack wird hauptsächlich zum Ablegen von Zwischenergebnissen während des Ausführens von Wortdefinitionen benutzt. Daten auf dem Stack können Zahlen, Zeichen, Adressen, Boole'sche Werte usw. sein.
Wenn der Begriff "Stapel" oder "Stack" ohne Zusatz benutzt wird, so ist immer der Datenstack gemeint.

stack, return (Rücksprungstapel)
Eine "Zuletzt-rein, Zuerst-raus" Struktur, die hauptsächlich Adressen von Wortdefinitionen enthält, deren Ausführung durch den Adressinterpreter noch nicht beendet ist. Wenn eine Wortdefinition eine andere Wortdefinition aufruft, so wird die Rücksprungadresse auf dem Returnstack abgelegt.
Der Returnstack kann zeitweise auch für die Ablage anderer Daten benutzt werden.

String, counted (abgezählte Zeichenkette)
Eine Hintereinanderfolge von 8bit Daten, die im Speicher durch ihre niedrigste Adresse charakterisiert wird. Das Byte an dieser Adresse enthält einen Zahlenwert im Bereich <0...255>, der die Anzahl der zu diesem String gehörigen Bytes angibt, die unmittelbar auf das Countbyte folgen. Die Anzahl beinhaltet nicht das Countbyte selber. Counted Strings enthalten normalerweise ASCII-Zeichen.

String, Text (Zeichenkette)
Eine Hintereinanderfolge von 8bit Daten, die im Speicher durch ihre niedrigste Adresse und ihre Länge in Bytes charakterisiert ist. Strings enthalten normalerweise ASCII-Zeichen. Wenn der Begriff "String" alleine oder in Verbindung mit anderen Begriffen benutzt wird, so sind Textstrings gemeint.

Userarea (Benutzerbereich)
Ein Gebiet im Speicher, das zum Ablegen der Uservariablen benutzt wird und für jeden einzelnen Prozeß/Benutzer getrennt vorhanden ist.

Uservariable (Benutzervariable)
Eine Variable, deren Datenbereich sich in der Userarea befindet. Einige Systemvariablen werden in der Userarea gehalten, sodaß die Worte, die diese benutzen, für mehrere Prozesse/Benutzer gleichzeitig verwendbar sind.

Vokabular (vocabulary)
Eine geordnete Liste von Wortdefinitionen. Vokabulare werden vorteilhaft benutzt, um Worte voneinander zu unterscheiden, die gleiche Namen haben (Synonyme). In einem Vokabular können mehrere Definitionen mit dem gleichen Namen existieren; diesen Vorgang nennt man redefinieren. Wird das Vokabular nach einem Namen durchsucht, so wird die jüngste Redefinition gefunden.

Vokabular, Kompilation (vocabulary, compilation)
Das Vokabular, in das neue Wortdefinitionen eingetragen werden.

Wahr (true)
Ein Wert, der nicht Null ist, wird als "wahr" interpretiert. Wahrheitswerte, die von Standard-FORTH-Worten errechnet werden, sind 16bit Zahlen, bei denen alle 16 Stellen auf "1" gesetzt sind, so daß diese zum Maskieren benutzt werden können.

Wort, Definierendes (defining word)
Ein Wort, das bei Ausführung einen neuen Dictionary-Eintrag im Kompilationsvokabular erzeugt. Der Name des neuen Wortes wird dem Quelltext entnommen. Wenn der Quelltext erschöpft ist, bevor der neue Name erzeugt wurde, so wird die Fehlermeldung "ungültiger Name" ausgegeben.
Beispiele von definierenden Worten sind:
: CONSTANT CREATE

Wort, immediate (immediate word)
Ein Wort, das ausgeführt wird, wenn es während der Kompilation oder Interpretation aufgefunden wird. Immediate Worte behandeln Sondersituationen während der Kompilation.
Siehe z.B. IF LITERAL ." usw.

Wortdefinition (word definition)
Eine mit einem Namen versehene, ausführbare FORTH-Prozedur, die ins Dictionary kompiliert wurde. Sie kann durch Maschinencode, als eine Folge von Kompilationsadressen oder durch sonstige kompilierte Worte spezifiziert sein. Wenn der Begriff "Wort" ohne Zusatz benutzt wird, so ist im allgemeinen eine Wortdefinition gemeint.

Wortname (word name)
Der Name einer Wortdefinition. Wortnamen sind maximal 31 Zeichen lang und enthalten kein Leerzeichen. Haben zwei Definitionen verschiedene Namen innerhalb desselben Vokabulars, so sind sie eindeutig auffindbar, wenn das Vokabular durchsucht wird.
Siehe: "Vokabular"

Zahl (number)
Wenn Werte innerhalb eines größeren Feldes existieren, so sind die höherwertigen Bits auf Null gesetzt. 16bit Zahlen sind im Speicher so abgelegt, dass sie in zwei benachbarten Byteadressen enthalten sind. Die Bytereihenfolge ist rechnerabhängig. Doppeltgenaue Zahlen (32bit) werden auf dem Stack so abgelegt, daß die höherwertigen 16bit (mit dem Vorzeichenbit) oben liegen. Die Adresse der niederwertigen 16bit ist um zwei größer als die Adresse der höherwertigen 16bit, wenn die Zahl im Speicher abgelegt ist.
Siehe: "Arithmetik, 2er-komplement" und "Zahlentypen"

Zahlenausgabe, bildhaft (pictured numeric output)
Durch die Benutzung elementarer Worte für die Zahlenausgabe (z.B. <# # #s #>) werden Zahlenwerte in Textstrings umgewandelt. Diese Definitionen werden in einer Folge benutzt, die ein symbolisches Bild des gewünschten Ausgabeformates darstellen. Die Umwandlung schreitet von der niedrigstwertigen zur höchstwertigen Ziffer fort und die umgewandelten Zeichen werden von höheren gegen niedrigere Speicheradressen abgelegt.

Zahlenausgabe, freiformatiert (free field format)
Zahlen werden in Abhängigkeit von BASE umgewandelt und ohne führende Nullen, aber mit einem folgenden Leerzeichen, angezeigt. Die Anzahl von Stellen, die angezeigt werden, ist die Minimalanzahl von Stellen - mindestens eine - die notwendig sind, um die Zahl eindeutig darzustellen.
Siehe: "Zahlenumwandlung"

Zahlentypen (number types)
Alle Zahlentypen bestehen aus einer spezifischen Anzahl von Bits. Zahlen mit oder ohne Vorzeichen bestehen aus bewerteten Bits. Bewertete Bits innerhalb einer Zahl haben den Zahlenwert einer Zweierpotenz, wobei das am weitesten rechts stehende Bit (das niedrigstwertige) einen Wert von zwei hoch null hat. Diese Bewertung setzt sich bis zum am weitesten links stehenden Bit fort, wobei sich die Bewertung für jedes Bit um eine Zweierpotenz erhöht. Für eine Zahl ohne Vorzeichen ist das am weitesten links stehende Bit in diese Bewertung eingeschlossen, sodaß für eine solche 16bit Zahl das ganz linke Bit den Wert 32.768 hat. Für Zahlen mit Vorzeichen wird die Bewertung des ganz linken Bits negiert, sodaß es bei einer 16bit Zahl den Wert -32.768 hat. Diese Art der Wertung für Zahlen mit Vorzeichen wird 2er-komplementdarstellung genannt.
Nicht spezifizierte, bewertete Zahlen sind mit oder ohne Vorzeichen; der Programmkontext bestimmt, wie die Zahl zu interpretieren ist.

Zahlenumwandlung (number conversion)
Zahlen werden intern als Binärzahlen geführt und extern durch graphische Zeichen des ASCII Zeichensatzes dargestellt. Die Umwandlung zwischen der internen und externen Form wird unter Beachtung des Wertes von BASE durchgeführt, um die Ziffern einer Zahl zu bestimmen. Eine Ziffer liegt im Bereich von Null bis BASE-1. Die Ziffer mit dem Wert Null wird durch das ASCII-zeichen "0" (Position 3/0, dezimalwert 48) dargestellt. Diese Zifferndarstellung geht den ASCII-code weiter aufwärts bis zum Zeichen "9", das dem dezimalen Wert neun entspricht. Werte, die jenseits von neun liegen, werden durch die ASCII-zeichen beginnend mit "A", entsprechend dem Wert zehn usw. bis zum ASCII-zeichen "~", entsprechend einundsiebzig, dargestellt.Bei einer negativen Zahl wird das ASCII-zeichen "-" den Ziffern vorangestellt. Bei der Zahleneingabe kann der

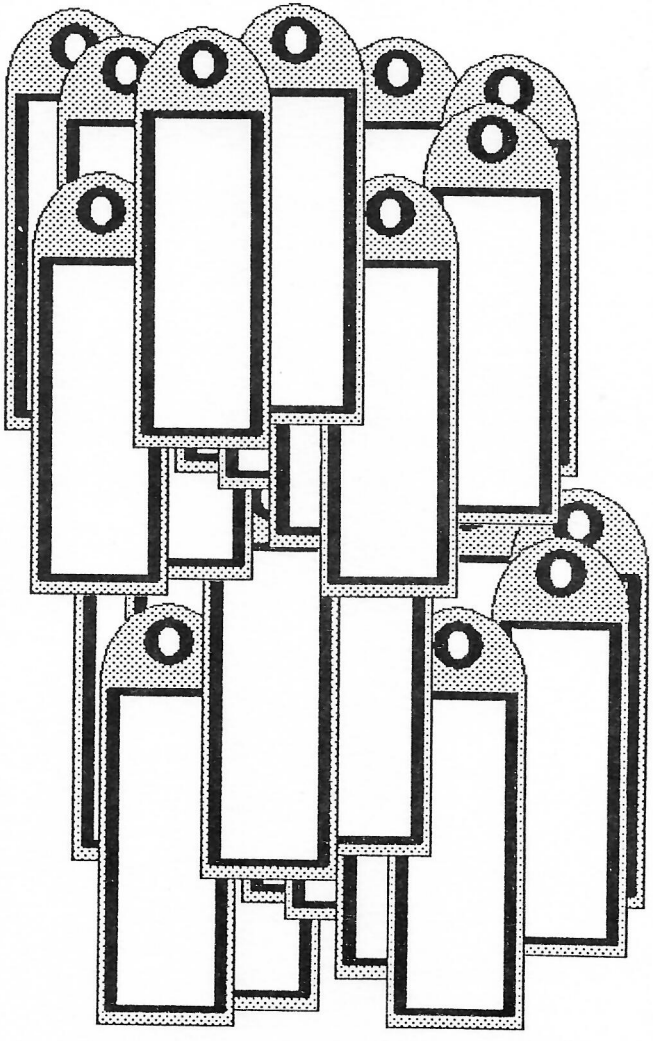
aktuelle Wert von BASE für die gerade umzuwandelnde Zahl dadurch umgangen werden, daß den Ziffern ein "Zahlenbasisprefix" vorangestellt wird. Dabei wird durch das Zeichen "%" die Basis vorübergehend auf den Wert zwei gesetzt, durch "&" auf den Wert zehn und durch "$" oder "h" auf den Wert sechzehn. Bei negativen Zahlen folgt das Zahlenbasisprefix dem Minuszeichen. Enthält die Zahl ein Komma oder einen Punkt, so wird sie als 32bit Zahl umgewandelt.

Zeichen (character)

Eine 8bit Zahl, deren Bedeutung durch den ASCII-Standard festgelegt ist. Wenn es in einem größeren Feld gespeichert ist, so sind die höherwertigen Bits auf Null gesetzt.

Zustand (mode)

Der Textinterpreter kann sich in zwei möglichen Zuständen befinden: dem interpretierenden oder dem kompilierenden Zustand. Die Variable STATUS wird vom System entsprechend gesetzt, und zwar enthält sie bei der Interpretation eine False-flag, bei der Kompilation eine True-flag.
Siehe: "Interpreter, Text" und "Kompilation"



Anhang

Atari ST Fullscreen Editor

Ältere FORTH-Systeme enthalten meist Editoren, die diesen Namen höchstens zu einer Zeit verdient haben, als man noch die Bits einzeln mit Lochzange und Streifenleser an die Hardware übermitteln mußte. Demgegenüber bietet ein Editor, mit dem man 'im Blindflug' jeweils eine ganze Zeile bearbeiten kann, sicher schon einigen Komfort. Heute kann man jedoch mit solch einem Zeileneditor keinen Staat mehr machen und erst recht nicht mit anderen Sprachen konkurrieren.

Wir haben daher dem Atari ST einen komfortablen Fullscreen-Editor spendiert, der vollkommen in GEM eingebunden ist. Dies erspart uns auch umfangreiche Befehlslisten aller Editorfunktionen: Der Editor hat eine Menüzeile, und beim Anklicken eines Menüpunktes bekommt man eine kurze Erklärung der jeweiligen Funktion. Die 'Profis' können alle Editorfunktionen auch ohne Maus über die Tastatur erreichen, die entsprechenden Control-codes stehen hinter den Menüeinträgen. Auch die Hilfsfunktion ist abschaltbar, wenn man den Menüpunkt DAUERHILFE anwählt. Die Hilfstexte erscheinen aber ohnedies nicht, wenn man die Kommandos von der Tastatur aus eingibt.

Im Editorfenster wird immer ein FORTH-Screen - also 1024 Bytes - in der üblichen Aufteilung in 16 Zeilen mit je 64 Spalten dargestellt. Es gibt einen Zeichen- und einen Zeilenstack. Damit lassen sich Zeichen bzw. Zeilen innerhalb eines Screens oder auch zwischen zwei Screens bewegen oder kopieren. Dabei wird verhindert, daß versehentlich Text verloren geht, indem Funktionen nicht ausgeführt werden, wenn dadurch Zeichen nach unten oder zur Seite aus dem Bildschirm geschoben würden.

Der Editor unterstützt das 'Shadow-Konzept'. Zu jedem Quelltext-Screen gibt es einen Kommentar-Screen. Dieser erhöht die Lesbarkeit von FORTH-Programmen erheblich. (Sie wissen ja, guter FORTH-Stil ist selbstdokumentierend !) Auf Tastendruck stellt der Editor den Kommentar-Screen zur Verfügung. So können Kommentare 'deckungsgleich' angefertigt werden. Die meisten mitgelieferten Quelltexte sind übrigens mit Shadow-Screens versehen, und auch das Printer-Interface unterstützt dieses Konzept.

Um in den Editor zu gelangen gibt es drei Möglichkeiten:

```
<screennr> L
```

ruft den Screen mit Nummer screennr auf. Dabei muß das File vorher z.B. mit USE ausgewählt sein.

```
V
```

ruft den zuletzt bearbeiteten Screen wieder auf. Dies ist der zuletzt editierte oder aber, und das ist sehr hilfreich, derjenige, der einen Abbruch beim Kompilieren verursacht hat. Wenn Sie also ein File kompilieren, und der Compiler bricht mit

einer Fehlermeldung ab, brauchen Sie nur ein V einzugeben. Der fehlerhafte Screen wird in den Editor geladen, und der Cursor steht hinter dem Wort, das den Abbruch verursacht hat.

Häufig möchte man sich die Definition eines Wortes ansehen, um z.B. den Stackkommentar oder die genaue Arbeitsweise nachzulesen. Dafür gibt es das Kommando

```
VIEW <wort>
```

Damit wird der Screen - und natürlich auch das File - aufgerufen, auf dem wort definiert wurde. Dieses Verfahren ersetzt (fast) einen Decompiler, weil es natürlich sehr viel bequemer ist und Ihnen ja auch sämtliche Quelltexte des Systems zur Verfügung stehen. Natürlich müssen die entsprechenden Files auf den Laufwerken 'griffbereit' sein, sonst erscheint eine Fehlermeldung. Die VIEW-Funktion steht auch innerhalb des Editors zur Verfügung, man kann dann mit einem Tastendruck zwischen dem gerade bearbeiteten und dem Screen, auf dem man eine Definition gesucht hat, hin- und herschalten. Dies ist insbesondere nützlich, wenn man eine Definition aus einem anderen File übernehmen möchte oder nicht mehr sicher ist, wie der Stackkommentar eines Wortes lautet oder ....

Noch ein paar Hinweise :

-) Wie Files erzeugt und verlängert werden, steht im Teil 1 des Handbuchs und wird weiter unten in dem Kapitel über das Fileinterface ausführlich erklärt.

-) Der Editor unterstützt nur das Kopieren von Zeilen. Man kann auf diese Art auch Screens kopieren, aber beim gelegentlich erforderlichen Einfügen von Screens in der Mitte eines Files ist das etwas mühselig. Zum Kopieren ganzer Screens innerhalb eines Files oder von einem File in ein anderes werden im volksFORTH83 die Worte COPY und CONVEY verwendet. Schauen Sie bitte deren genaue Funktion im Glossar unter "Massenspeicher" nach.
Beispiel : Sie wollen in ihr File FIRST.SCR vor den Screen 4 einen weiteren Screen einfügen. Dazu geben Sie ein :

```
use first.scr          \ FIRST.SCR ist aktuelles File
1 more                 \ Verlängere um einen Screen
4 capacity 2-  5   convey
                       \ kopiere 4..."vorletzer" nach
                       \         5..."letzter"   Screen
```

Anschließend sind die Screens 4 und 5 gleich und Sie können den Screen 4 löschen. Beachten Sie bitte, daß Sie bei Einfügen von Screens evtl. die Argumente von +THRU +LOAD THRU und LOAD ändern müssen!

Die GEM-Bibliothek des volksFORTH83

Diese volksFORTH83-Version enthält eine umfangreiche Bibliothek der GEM-Routinen. Diese Bibliothek gliedert sich in die Teile AES ("Application Environment System") und VDI ("Virtual Device Interface"). Im volksFORTH gehört auch ein Teil BASICS dazu, der die VDI und AES gemeinsamen Teile enthält. Ferner gibt es Files, die den Parameter-Konstanten der GEM-Routinen symbolische Namen zuordnen und eine "Super"-Bibliothek.

Die Namen der von GEM zur Verfügung gestellten Funktionen enstprechen völlig den Namen, die vom "C" des Entwicklungspakets her bekannt sind. Viele Präfixe haben wir jedoch weggelassen, da sie nur unnötig viel Platz verbrauchen würden. Die Funktionen benötigen zumeist auch dieselben Argumente, wobei wir uns bemüht haben, überflüssige Eingangswerte wegzulassen (z.B. bestimmte Handles). Manche Aufrufe liefern soviele Ausgangswerte, daß wir uns entschlossen haben, sie nicht alle auf den Stack zu packen. In diesem Fall muß man sie selbst aus dem enstprechenden Array holen. Wo, findet man ggf. in der Literatur. Die Eingangswerte stehen jedoch (mit Ausnahme von EVNT-MULTI) auf den Stack.

-) Wer Zugang zur GEM-Programmierung sucht, sollte sich die Literatur des Enwicklungspakets von Atari verschaffen. Dort sind (fast) alle Funktionen (viele fehlerhaft!) beschrieben. Die Qualität ist aber nicht besonders, z.T auch auf den IBM-PC abgestellt.

-) Empfehlenswert, wenn auch ausserordentlich knapp und daher nur als Nachschlagewerk geeignet, ist das Handbuch zum Megamax C-Compiler.

-) Einige Zeitschriften des Atari-Marktes beginnen Fortsetzungsreihen zur GEM-Programmierung, die aber bisher nicht empfehlenswert sind.

Für die GEM-Programmierung ist außerdem ein sog. Resource Construction Set erforderlich. Das ist ein Programm, mit dem die Menüleisten, Dialog- und Alertboxen sowie die Icons hergestellt werden. Das Programm erzeugt ein File, daß auf ".RSC" endet (das sog. Resourcefile) und ein ".H"-File, daß die Anbindung des Resourcefiles an "C"-Programme erlaubt. Dieses C-File enthält Konstantendefinitionen für jedes "Objekt" (jedes noch so kleine Teil eines Resourcefiles ist so ein Objekt), die analog auch in ihrem Forth-Programm auftreten müssen. Vergleichen Sie dazu bitte das File EDIICON.H (für "C") mit EDIICON.SCR, das dessen Übersetzung in Forth darstellt. Ein Resource Construction Set gehört sowohl zum Lieferumfang des Entwicklungspakets als auch zum Megamax-"C"-Paket.

Verzeichnis der Worte der GEM-Bibliothek

Die GEM-Bibliothek besteht aus den Files BASICS.SCR, VDI.SCR und AES.SCR. Die in diesen Files enthaltenen Worte werden im folgenden aufgeführt. Des weiteren soll sich eine Bibliothek von "Super-worten" entwickeln, die die Handhabung der GEM-Routinen vereinfacht. Diese Files sind über Shadowscreens dokumentiert und daher hier nicht aufgeführt.

BASICS.SCR

```
Vocabulary GEM   GEM definitions also
     \ Das Vokabular, das die GEM-Worte enthält.

Create intin          Create ptsin
Create intout         Create ptsout
Create addrin         Create addrout
     \    Diese Arrays werden für die diversen Parameter  benö-
          tigt.

Variable grhandle
     \    Diese Variable nimmt die Handle auf,  die von  OPNVWK
          geliefert wird.

Create contrl   $16 allot
contrl 2 gemconstant opcode
       2 gemconstant #intin
       2 gemconstant #intout      ' #intout Alias #ptsout
       2 gemconstant #addrin
       2 gemconstant #addrout
       2 gemconstant function
     \    Die  Komponenten  dieses  Arrays  tragen  Namen  und
          enthalten die Zahl der Parameter,  die in den betref-
          fenden Arrays (s.o.) übergeben werden.

Create global
Constant ap_ptree
     \    Dieses Element des Arrays GLOBAL enthält die  Adresse
          eines Baumes, die von RSRC_LOAD initialisiert wird.

Create AESpb
Create VDIpb
     \    Diese beiden Arrays enthalten Zeiger auf die  Arrays,
          die die Parameter enthalten...

Code array!    ( n0 ... nk-1 adr k --)
     \ Speichert k Werte ab Adresse adr in einem Array.
Code 4!        ( n1 .. n4 addr -- )
Code 4@        ( addr -- n1 .. n4 )
     \    Speichert  bzw.  liest 4 Werte ab Adresse  addr.  Sie
          werden dazu benutzt,  Rechtecke in die div. Arrays zu
          schreiben bzw. herauszuholen.
Code AES       ( opcode #intin #intout #addrin #addrout --
                 intout@)
     \    Dieses Wort wickelt alle AES-Aufrufe ab.
Code VDI       ( opcode #ptsin #intin --)
     \    Dieses Wort wickelt alle VDI-Aufrufe ab.
```

```
: appl_init
     \    Dieses Wort initialisiert die Applikation und sollte
          vor dem ersten Aufruf einer AES-Funktion aufgerufen
          werden.
: appl_exit
     \    Dieses Wort sollte am Ende einer Applikation aufgeru-
          fen werden.

Create sizes 8 allot
     \    Dieses Array enthält die Größe eines Buchstabens und
          einer Box in Pixeln
Sizeconst c_width   Sizeconst c_height
Sizeconst b_width   Siezconst b_height
     \    Diese Worte lesen die Felder in SIZES aus, die die
          Höhe und Breite eines Zeichens bzw. einer Box enthal-
          ten.
: graf_handle
     \    Liefert die VDI-Handle in der Variablen GRHANDLE und
          die Buchstaben- bzw. Boxgröße in SIZES
: opnvwk
     \    Öffnet eine "virtual workstation" und muß zu Beginn
          einer Applikation aufgerufen werden.
: clrwk
     \    Löscht die Workstation. Der Hintergrund wird in der
          gewählten Farbe gemalt.
: clsvwk
     \    Schließt die "virtual workstation". Muß vor Beenden
          des Programms aufgerufen werden.
: updwk
     \    Update virtual workstation. Alle VDI-Kommandos werden
          zuende ausgeführt.
: s_clip     ( x1 y1 x2 y2 clipflag -- )
     \    Setze Größe und Position des Clipping rectangle für
          alle VDI-Aufrufe.

: grinit    appl_init graf_handle opnvwk ;
     \    Faßt alle benötigten Aufrufe am Anfang einer Applika-
          tion zusammen.
: grexit    clsvwk appl_exit   ;
     \    dto. für die Beendigung einer Applikation.
Variable c_flag
     \    gibt an, ob die Aufrufe von SHOW_C und HIDE_C
          akkumuliert werden sollen.
: show_c       ( -- )
     \    Schalte die Maus an.
: hide_c       ( -- )
     \    Schalte die Maus aus.
2Variable objc_tree
     \    Enthält den Objektbaum, auf das sich die folgenden
          Worte beziehen : MENU_BAR MENU_ICHECK MENU_IENABLE
          MENU_TNORMAL MENU_TNEXT FORM_DO FORM_CENTER
```

AES.SCR

Event :

```
: evnt_keybd   ( -- key )
     \     Wartet auf einen Keyboard-event. key besteht aus
           scan-code und ASCII-Wert.
: evnt_button  ( #clicks0 bmask bstate -- #clicks1 )
     \     Wartet auf einen button-event. #clicks0 gibt die
           Anzahl der Clicks an, bmask und bstate geben die zu
           drückende Taste an. #clicks1 ist die Zahl der Clicks.
: evnt_mouse   ( f leftX topY widht height -- )
     \     Wartet auf einen mouse-movement-event. f ist null für
           das Betreten und 1 für das Verlassen des durch leftX
           topY width und height spezifizierten Rechtecks.
Create message
     \     Dieses Array ist der message-event-buffer.
: evnt_mesag   ( -- )
     \     Wartet auf einen message-event.
: evnt_timer   ( dtime -- )
     \     Wartet auf einen Timer-event. dtime ist die (doppelt
           genaue) zu wartende Zeit in msec.
Create events
     \     Dieses Array enthält die Eingangsparameter des Wortes
           EVNT_MULTI. Der Inhalt dieses Feldes ist wie folgt :
           +0 : Art der erkannten Events :
                Bit 0     Keyboard
                Bit 1     Button
                Bit 2     Mouse rectangle 1
                Bit 3     Mouse rectangle 2
                Bit 4     Message
                Bit 5     Timer
           +2   #clicks0  Zahl der zum "Auslösen" erforderlichen
                          Mausklicks.
           +4   bmask     Bit 0 : linker, Bit 1 : rechter ist zu
                          betätigen.
           +6   bstate    Gibt an, ob der entsprechende Knopf
                          von bmask gedrückt oder gelöst sein
                          muß.
           +8   f leftX topY width height,   gibt an, ob der
                Eintritt (f=0) oder Austritt (f=1) aus dem
                angegebenen Rechteck signalisiert werden soll.

           +18   ------- " ------------- , dto.
           +28  dtime     Zu wartende Zeit in msec.

: prepare
     \     Dieses Wort kopiert den Inhalt von EVENTS in die
           enstprechenden Parameterarrays und sollte vor
           EVNT_MULTI aufgerufen werden.
: evnt_multi   ( -- which )
     \     Dieses Wort wartet auf einen von mehreren möglichen
           Events. Das Array EVENTS enthält die Parameter dieser
           Routine. which ist die Nummer der aufgetretenen
           Events nach dem Schlüssel wie für das erste Wort von
           EVENTS. Die Ausgabeparameter müssen den GEM-Arrays
           entnommen werden...
```

```
: evnt_dclick  ( dnew dgetset -- dspeed )
     \     Dieses Wort setzt oder liest die Zeitspanne für einen
           erkannten Mehrfachclick.  dgetset = 1  bedeutet,  daß
           dnew  als  neue Zeit gesetzt  wird.  dspeed  ist  die
           eingestellte Zeitspanne.
```

Menu :

```
: menu_bar      ( showflag -- )
     \     Löscht oder setzt die Menüleiste.
: menu_icheck   ( item showflag -- )
     \     Setzt oder löscht den Haken vor einem menu item
: menu_ienable  ( item enableflag -- )
     \     Schaltet einen menu item ein oder aus. Ausgeschaltete
           menu items erscheinen in heller Schrift.
: menu_tnormal  ( title normalflag -- )
     \     Stellt einen Menütitel normal oder invers dar.
: menu_text     ( item laddr -- )
     \     Ändert den Text eines menu item.  Die Länge darf sich
           aber nicht ändern !
: menu_register( apid laddr -- menuid )
     \     Installiert ein Desktop Accessory in der  Menüleiste.
           menuid ist die Position im Menü
```

Object

```
: objc_add      ( parent child -- )
     \     Fügt ein Objekt child an das Ende der child list  des
           Objektes parent an.
: objc_delete   ( object --  )
     \     Löscht ein Objekt vom Baum.
: objc_draw     ( startob depth x y width height -- )
     \     Zeichnet  den  Baum von startob mit der  Tiefe  depth
           neu.  x  y  width  und  height  geben  ein  Clipping-
           Rectangle an.
: objc_find     ( startob depth x y -- obnum )
     \     Sucht das Objekt unter der Mausposition x y . Liefert
           die Objektnummer oder -1.
: objc_offset   ( object -- x y )
     \     Liefert die Bildschirmpositon eines Objektes
: objc_order    ( object newpos -- )
     \     Bewegt ein Objekt von einer Stelle des Baums zu einer
           anderen.
: objc_edit     ( object char index kind -- newindex )
     \     Wird zum Editieren des Textes innerhalb eines  Objek-
           tes benutzt.
: objc_change   ( object x y width height newstate redraw -- )
     \     Ändert den Objektstatus und zeichnet das Objekt neu.
```

Form :

```
: form_do      ( startobj -- objectno )
     \     Diese  Routine  zeigt ein Objekt an und  erlaubt  dem
           Benutzer,  es zu ändern. objectno. ist die Nummer des
           exit button.
: form_dial    ( diflag lix liy liw lih  bix biy biw bih )
     \     Diese Routine besteht aus vier Routinen,  ihre  Funk-
           tion wird von diflag bestimmt:
                0    Reserviere einen Bildschirmbereich für  ein
                     Objekt.
                1    Male eine wachsende Box
                2    Male eine schrumpfende Box
                3    Gib   den  reservierten   Bildschirmbereich
                     frei.
: form_alert   ( defbttn 0string -- exbttn )
     \     Erlaubt  auf  einfache  Art,  Alert-Boxen  zu  bauen.
           defbttn ist der Defaultknopf,  0string ist ein  durch
           $00  begrenzter  String,  der das  Aussehen  der  Box
           bestimmt und exbttn ist der gedrückte Knopf.
: form_error   ( enum -- exbttn )
     \     Malt  eine Alertbox mit dem  Text  "TOS-Fehler-Nummer
           soundso"
: form_center  ( -- x y width height )
     \     Rechne  die  Position eines in der  Mitte  des  Bild-
           schirms zentrierten Objektes aus.
```

Graphic :

```
: graf_dragbox       ( startx starty width height boundx  boundy
                       boundw boundh -- finishx finishy )
     \     Malt den Umriß einer Box,  die auf dem Bildschirm mit
           der  Maus bewegt werden kann.  Die Bewegung der  Maus
           wird durch ein äußeres Rechteck beschränkt.
: graf_movebox       ( sourcex sourcey width height destx  desty
                       --)
     \     Malt eine Box, die sich von source nach dest bewegt.
: graf_growbox       ( stx sty stw sth  fix fiy fiw fih -- )
     \     Malt eine expandierende Box.
: graf_shrinkbox     ( fix fiy fiw fih  stx sty stw sth -- )
     \     Malt eine schrumpfende Box.
: graf_watchbox      ( object instate outstate --
                       inside/outside )
     \     Überwacht  die  Bewegung  der  Maus  innerhalb  eines
           Objektes....
: graf_slidebox      ( parent object vhflag -- vhpos )
     \     Überwacht  das  Ziehen  einer kleinen  Box  in  einer
           großen Box mit der Maus....

2Variable mofaddr
     \     Zeiger auf eine selbst definierte Mausform
: graf_mouse         ( mouseform -- )
     \     Setzt die Mausform :
           0     Pfeil
           1     Cursor
           2     Biene
           3     Zeigefinger
           4     Hand
           5     dünnes Fadenkreuz
```

```
          6    dickes Fadenkreuz
          7    Fadenkreuz mit Umriß
          255  selbst definiert
          256  hide mouse
          257  show mouse
: graf_mkstate      ( -- )
     \    liefert  Mausposition und Knöpfe in den  div.  Arrays
          zurück.

Fileselect :

Create inpath
     \    Der Defaultpath steht in diesem Array mit einem  $00-
          Byte abgeschlossen.
Create insel
     \    Der Name des selektierten Files ...
: fsel_input   ( -- button )
     \    malt  die sog.  File selector box und wartet auf  die
          Auswahl eines Files.   button ist der gedrückte Knopf
          (0 = Cancel)

Window :

: wind_create  (  components leftX topY maxWIDTH  maxHEIGTH  --
               handle)
     \    dieses Wort erzeugt ein Fenster mit div. Bestandtei-
          len und einer Maximalgröße. Liefert eine Handle für's
          Fenster.
: wind_open    ( W-handle leftX topY width height -- )
     \    Malt  ein  Fenster  in der angegeben  Größe  auf  dem
          Bildschirm.
: wind_close   ( Whandle -- )
     \    Schließt und löscht ein Fenster.  Es kann auch wieder
          geöffnet werden.
: wind_delete  ( Whandle -- )
     \    Löscht  ein Fenster,  so daß es nicht  mehr  geöffnet
          werden kann.
: wind_get     ( Whandle funktion# -- )
     \    Liefert div. Informationen über ein Fenster...
: wind_set     ( Whandle funktion# par0 par1 par2 par3 -- )
     \    Setzt  div.  Attribute eines Fenster wie  Titelzeile,
          Sliderpos. usw.
: wind_find    ( mouseX mouseY -- Whandle )
     \    Sucht das Fenster unter der Mausposition.
: wind_update  ( funktion# -- )
     \    wird benutzt, um die Manipulation an anderen Fenstern
          oder  Menüauswahl während des Zeichnens zu  unterbin-
          den.
: wind_calc    ( 0/1 components leftX topY width height -- )
     \    Konvertiert bei Fenstern die Innen- in Außenmaße  (0)
          oder umgekehrt (1).  Die Ausgabe erfolgt in den  GEM-
          Arrays.
```

```
RSRC : : rsrc_load  ( 0$ -- )   \ needs address of 0-termina-
                                  ted $
     \    Lädt ein Resourcefile in den Speicher.
: rsrc_load"
     \    Lädt das Resourcefile, dessen Name, durch ein "
          begrenzt, auf dieses Wort folgt.
: rsrc_free    ( -- )
     \    Gibt den durch ein Resourcefile beanspruchten Spei-
          cherbereich wieder frei.
: rsrc_gaddr   ( type index -- laddr )
     \    Liefert die Adresse eines Objektes im Resourcefile.
: rsrc_saddr   ( type index laddr --)
     \    Speichert die Adresse eines Objektes ab.
: rsrc_obfix   ( index laddr --)
     \    Konvertiert Objektlage und Größe von Character- in
          Pixeleinheiten.
```

VDI.SCR

Output Function

```
: pline          ( x1 y1 x2 y2 ... xn yn count -- )
     \     malt eine geknickte Linie von x1,y1 zu x2,y2 usw.
: pmarker        ( x1 y1 x2 y2 ... xn yn count -- )
     \     Gibt Polymarker aus.
: gtext          ( addr count x y -- )
     \     Gibt Text an der angegebenen Stelle aus.
: fillarea       ( x1 y1 x2 y2 ... xn yn count -- )
     \     Malt ein ausgefülltes Vieleck.
: contourfill    ( color x y -- )
     \     füllt ein gemalten Bereich aus.
: r_recfl        ( x1 y1 x2 y2 -- )
     \     Malt ein Rechteck ohne Rand.
: GDP            ( #ptsin #intin functionno -- )
     \     Allg. Malroutine
: bar            ( x1 y1 x2 y2 -- )
     \     Malt ein gefülltes Rechteck mit Rand.
: arc            ( startwinkel endwinkel x y radius -- )
     \     Malt den Ausschnitt eines Kreises.
: pie    ( startwinkel endwinkel x y radius -- )
     \     Malt ein Stück Torte.
: circle         ( x y radius -- )
     \     Malt einen Kreis
: ellarc         ( startwinkel endwinkel x y xradius yradius -- )
     \     Malt einen Ellipsenabschnitt
: ellpie         ( startwinkel endwinkel x y xradius yradius -- )
     \     Jetzt gefüllt !
: ellipse        ( x y xradius yradius -- )
     \     Und nochmal, bloß ganz.
: rbox           ( x1 y1 x2 y2 -- )
     \     Rechteck mit abgerundeten Ecken
: rfbox          ( x1 y1 x2 y2 -- )
     \     gefüllt.
: justified      ( string x y length wordspace charspace -- )
     \     Textausgabe  mit definierter Länge,  maximalem  Wort-
           und Buchstabenabstand.
```

Attribute Function :

```
: swr_mode       ( mode -- )
     \     Setzt Ausgabemodus...
1 Setmode replace                2 Setmode transparent
3 Setmode exor                   4 Setmode revtransparent

: sl_type        ( style -- )
     \     Setzt Polyline type
1 Settype solid                  2 Settype longdash
3 Settype dot                    4 Settype dashdot
5 Settype dash                   6 Settype dashdotdot
7 Settype userdef
```

```
: sl_udsty      ( pattern -- )
     \     setzt user defined line style
: sl_width      ( width -- )
     \     setzt polyline line width
: sl_color      ( color -- )
     \     setzt polyline color index
: sl_ends       ( begstyle endstyle -- )
     \     setzt polyline end styles
: sm_type       ( symbol -- )
     \     setzt polymarker type

1 Setmtype point                2 Setmtype plus
3 Setmtype asterisk             4 Setmtype square
5 Setmtype cross                6 Setmtype diamond

: sm_height          ( height -- )
     \     setzt polymarker height
: sm_color           ( color -- )
     \     setzt polymarker color index
: st_height          ( height  -- )
     \     setzt text character height (absolut)
: st_point           ( point -- )
     \     setzt text character height (points)
: st_rotation        ( winkel -- )
     \     setzt character baseline rotation
: st_font            ( font -- )
     \     setzt character font
: st_color           ( color --
     \     setzt text colour
: st_effects         ( effect -- )
     \     setzt fett, kursiv usw.
: st_alignement      ( horin vertin -- )
     \     setzt character alignment
: sf_interior        ( style  --
     \     setzt fill interior style
: sf_style           ( styleindex -- )
     \     setzt fill style index
: sf_color           ( color -- )
     \     setzt fill colour index für Vielecke
: sf_perimeter       ( pervis -- )
     \     schaltet fill outline um.
```

Raster Operation :

```
     \ Die  Rasteroperationen dienen zum schnellen  Verschieben
       von Bildschirmbereichen sowohl auf dem Bildschirm selbst
       als auch vom Bildschirm in den Speicher und  zurück.  Da
       es  sich um sehr schnelle Routinen handelt,  sollte  von
       ihnen  immer dann Gebrauch gemacht  werden,  wenn  Bild-
       schirminhalte  restauriert werden müssen.  Alle  -  sonst
       notwendigen  - Ausgaberoutinen brauchen  erheblich  mehr
       Zeit.
```

```
Create scrMFDB
Variable >memMFDB

     \ Die Memory Form Definition Blocks beschreiben den Aufbau
       eines  Pixelblocks im Speicher oder auf dem  Bildschirm.
       Um  mit mehreren Speicherbereichen arbeiten  zu  können,
       enthält >memMFDB einen Pointer auf den gerade  benutzten
       Bereich.

: copyopaque   ( Xfr Yfr width height Xto Yto mode --)
     \ Grundroutine für alle Rasteroperationen
: scr>mem      ( addr_of_memMFDB -- )
     \ Definierendes  Wort  für  Rasteroperation   (Bildschirm-
       >Speicher)
: mem>scr      ( addr_of_memMFDB -- )
     \ Definierendes   Wort   für   Rasteroperation   (Speicher-
       >Bildschirm)
: scr>scr      ( Xfr Yfr width height Xto Yto --)
     \ verschiebt ein Rechteck auf dem Bildschirm.
Create memMFDB1
     \ Ein Speicherblock, der den gesamten Bildschirm speichern
       kann.
: scr>mem1     ( Xleft Ytop Width Heigth -- )
     \ verschiebt den gesamten Bildschirm in den Speicher.
: mem>scr1     ( Xleft Ytop Width Heigth -- )
     \ verschiebt den Speicherblock wieder in den Bildschirm.
: r_trnfm      ( -- )
     \ rechnet  Standard- in gerätespezifische  Koordinaten  um
       und umgekehrt.
: get_pixel    ( x y -- color flag )
     \ ermittelt die Farbe eines Pixels. flag ist 1, wenn Punkt
       gesetzt.

Input :

: sin_mode     ( devtype mode -- )
     \    legt den Inputmodus fest....
: sm_locater   ( x y -- status )
     \    Position der Maus,  Angabe der Anfangspos. erforder-
          lich
: sm_valuator  ( val_in -- status )
     \    Verwaltung von Wertänderungen ....
: sm_choice    ( -- status )
     \    Teste die Funktionstasten
: sm_string    ( addr max_len echomode x y -- status )
     \    Eingabe eines Strings mit Echo.
: sc_form      ( addr -- )
     \    setzt Mausform
: ex_time      ( tim_addr -- long_otim_addr )
     \    Ersetzt Timerinterrupt-Vektor
: q_mouse      ( -- x y status )
     \    Liest den Mauszustand aus.
: ex_butv      ( pusrcode -- long_psavcode )
     \    Ersetzt Mausbuttoninterruptroutine
: ex_motv      ( pusrcode -- long_psavcode )
     \      ... Mausbewegungsinterruptroutine...
: ex_curv      ( pusrcode -- long_psavcode )
     \    ... Mausforminterruptroutine...
```

```
: q_key_s       ( -- status )
     \    liefert Zustand der Shift-Tasten.
```

Inquire :

```
: q_extnd            ( info_flag -- )
     \    Diese und die folgenden Funktionen dienen dazu,
          festzustellen, welche der obigen VDI-Aufrufe man
          getätigt hat. Man kann sie nicht kurz beschreiben,
          daher muß die Eingangs erwähnte Literatur zu Rate
          gezogen werden.
: q_color            ( color_index info_flag )
: ql_attributes      ( -- )
: qm_attributes      ( -- )
: qf_attributes      ( -- )
: qt_attributes      ( -- )
: qt_extent          ( string -- )
: qt_width           ( char -- status )
: qt_name            ( element_num -- )
: q_cellarray        ( cols rows x1 y1 x2 y2 -- )
: qin_mode           ( dev_type -- mode )
: qt_fontinfo        ( -- )
```

Escape :

```
: q_chcells          ( -- rows cols )
     \    Größe des Bildschirms
: exit_cur           ( -- )
     \    textcursor aus
: enter_cur          ( -- )
     \    textcursor ein
: curup              ( -- )
     \    hoch
: curdown            ( -- )
     \    runter
: curright           ( -- )
     \    ????
: curleft            ( -- )
     \    !!!!
: curhome            ( -- )
     \    cursor nach links oben
: eeos               ( -- )
     \    erase to end of screen
: eeol               ( -- )
     \    ----- " ------- line
: s_curaddress       ( row col -- )
     \    positioniere cursor
: curtext            ( addr count -- )
     \    textausgabe
: rvon               ( -- )
     \    reverse video on
: rvoff              ( -- )
     \    ---- " ------ off
: q_curaddress       ( -- row col )
     \    frage nach cursor adress
: q_tabstatus        ( -- status )
     \    frage nach Mausstatus
: hardcopy           ( -- )
     \    screen dump
```

```
: dspcur              ( x y -- )
     \    positioniere maus
: rmcur               ( -- )
     \    entferne Maus
```

Die folgenden Befehle dienen zur Arbeit mit verschiedenen Output-Devices. Dies ist ein besonders schlecht dokumentierter Teil. Einige Hinweise findet man in der Dokumentation zum Entwicklungspaket von Digital Research. Es ist fraglich, ob alle Befehle tatsächlich auf dem Atari arbeiten. Versuche in dieser Richtung sind bislang noch nicht erfolgreich gewesen.

```
: form_adv            ( -- )
     \    Für Drucker: Seitenvorschub
: output_window       ( x1 y1 x2 y2 -- )
     \    Für Drucker: Gibt ein Fenster aus.
: clear_disp_list     ( -- )
     \    Für Drucker:  bricht Ausdruck ab,  wie Clear Workstation, aber ohne abschließendes Linefeed.
: bit_image           ( string aspect scaling num_pts x1 y1 x2 y2 -
                      - )
     \    Für Drucker: Gibt ein 'Bit Image File' aus.
```

Die folgenden Befehle beziehen sich auf zusätzliche Output-Treiber; ein Effekt auf dem Atari ist bisher nicht bekannt. Sie sind nur der Vollständigkeit halber aufgeführt.

```
: s_palette           ( palette -- selected )
     \    setzt Farbpalette auf IBM-Farbmonitor ??!!
: qp_films            ( -- )
: qp_state            ( -- )
: sp_state            ( addr -- )
: sp_save             ( -- )
: sp_message          ( -- )
: qp_error            ( -- )
: meta_extents        ( x1 y1 x2 y2 -- )
: write_meta          ( intin num_intin ptsin num_ptsin -- )
: m_filename          ( string -- )
```

Der Assembler

Der im volksFORTH83 für den Atari 520 ST enthaltene 68000-Assembler entspricht im Wesentlichen dem von Michael Perry für das F83 entwickelten. Daher bringen wir hier eine Übersetzung eines Artikels aus 'Dr. Dobbs Journal', Nr.83 vom September 1983, in dem Michael Perry seinen Assembler beschrieben hat. Abweichungen des volksFORTH-Assemblers werden besonders erwähnt. Im Anschluß an die Übersetzung werden die zusätzlichen Funktionen des volksFORTH- Assemblers beschrieben.

Ein 68000 Forth Assembler

In diesem Artikel werde ich die Eigenarten eines Assemblers in FORTH, seinen Gebrauch und die Implementation eines Beispiels beschreiben: Ein FORTH Assembler für den 68000. Ich hoffe, die Leistungsfähigkeit eines solchen Assemblers darlegen zu können, einige der damit verbundenen Eigenarten, und warum er so und nicht anders programmiert wurde. Um den Leser nicht zu verwirren, verzichte ich auf Verallgemeinerungen. Ich werde gelegentlich Dinge darstellen, die sich speziell auf mein System beziehen und auf anderen Systemen leicht abweichen können.

Kurz gesagt ist ein FORTH-System eine interaktive Programmierumgebung, in der einzelne Module, 'Worte' genannt, in einer Datenstruktur namens Dictionary abgelegt werden. Der Programmierer kann neue Worte hinzufügen, die entweder aus vorhandenen FORTH-Worten oder aus Maschinencodedefinitionen bestehen. Ein Assembler in einem FORTH-System ist ein Werkzeug, um Coderoutinen zu definieren. Er ist nicht dazu vorgesehen, eigenständige Applikationen in Maschinensprache zu schreiben. Ein FORTH Cross-Assembler ist ganz ähnlich aufgebaut. Mit ihm kann man Code erzeugen, der auf einem anderen System läuft, eventuell sogar auf einem anderen Prozessor. Dieser Artikel bezieht sich nur auf 'normale' FORTH-Assembler. FORTH-Assembler sind kurz, da sie auf vorhandene FORTH-Worte zurückgreifen können; dieser Assembler z.B. benötigt nur ca. 3 kByte, dazu kommt das System mit 12 kByte. Zum Vergleich: Der Assembler, der zum CPM 68k gehört, benötigt 44 kByte, zusätzlich etwa 6 kByte für die Symboltabelle.

Wenn man Applikationen in FORTH schreibt, wird der Assembler selten eingesetzt, bevor die einzelnen Programmteile nicht in High-Level geschrieben und ausgetestet sind. In den Anfangsstadien einer Entwicklung ist die Zeit, die der Programmierer braucht, wesentlich wertvoller als die der Maschine. Wenn eine Applikation lauffähig ist, mag es sich herausstellen, daß sie in High-Level zu langsam ist. In diesem Falle muß man herausfinden, welche Routinen zeitkritisch sind und dann nur diese in Code neu schreiben. Diesen Vorgang wiederholt man so lange, bis das Programm schnell genug ist. Vermeiden Sie mehr Coderoutinen als erforderlich, da diese die Übertragbarkeit Ihres Programms stark einschränken. In seltenen Fällen, wenn man eine sehr zeitkritische Anwendung vor sich hat, wird man letztlich fast alles in Code schreiben.

Sogar in solchen Fällen wird die Entwicklung in der oben beschriebenen Reihenfolge am schnellsten zu Resultaten und zur fertigen Anwendung führen. Seien Sie immer bereit, frühere Entwürfe über den Haufen zu werfen und von vorn zu beginnen. Der Schlüssel zum Erfolg ist schrittweise Annäherung: Schreiben, Testen, Überarbeiten, bis Sie endgültig zufrieden sind. Das ist der Grund, warum es so wichtig ist, zunächst eine einfache Version zu entwickeln, um zu sehen, ob die Grundidee richtig und durchführbar ist.

Der Name eines FORTH-Wortes kann aus bis zu 31 Ascii-Zeichen bestehen, ausgenommen sind Leerzeichen. Worte im Dictionary sind in Gruppen zusammengefaßt, die man Vokabulare nennt. Der Assembler ist ein solches Vokabular namens ASSEMBLER. Die meisten Worte im Assembler haben die Namen der üblichen Mnemonics des Prozessors, in unserem Falle des 68000. Wenn so ein Wort ausgeführt wird, legt es die zugehörige Bytefolge im Dictionary ab. Andere Worte im Assembler behandeln die Adressierungsart, Kontrollstrukturen, Makros und möglicherweise andere Erweiterungen. Hält man sich an eine FORTH-übliche Syntax, ist es mit wenig Aufwand möglich, einen sehr leistungsfähigen Assembler zu implementieren.

Die zwei wichtigsten Einschränkungen sind die Syntax und der Verzicht auf Vorwärtsreferenzen. Wie in FORTH üblich sind Vorwärtsreferenzen nicht erlaubt. Das heißt, ein Wort muß vor seinem ersten Aufruf definiert sein. Ich bin der Überzeugung, dies ist eine gute Sache, aber diese Meinung beschwört endlose Debatten herauf, und ich werde sie hier nicht beeenden können. Es ist sehr viel einfacher (und damit auch erheblich schneller), wenn man eine Syntax verwendet, bei der der Operator hinten steht. Das bedeutet, die Befehle werden in folgender Form geschrieben:

```
Source Destination Operation
```

Wenn auch ungewöhnlich, so ist dieses Format doch sehr flexibel und einfach zu verwenden. Ein Pre-Prozessor, der die übliche Schreibweise verarbeiten kann, könnte relativ leicht eingebaut werden, wenn man die damit verbundenen Geschwindigkeitsnachteile in Kauf nimmt.

Das Dictionary wächst in Richtung steigender Adressen, wenn neue Worte hinzugefügt werden. Die meisten Datenstrukturen werden ebenfalls im Dictionary abgelegt. Die Systemvariable DP zeigt auf die nächste freie Adresse. Das Wort HERE übergibt den Wert von DP auf dem Stack. Das Wort , (comma) trägt einen 16-Bit-Wert ins Dictionary ein, das Wort c, (c-comma) einen 8-Bit-Wert (ein Byte). Der Assembler ist nur auf comma und c-comma aufgebaut.

Fehlerbehandlung

Wenn ich einen Assembler benutze, erwarte ich von ihm einige wichtige Leistungsmerkmale. An erster Stelle steht natürlich die richtige Übersetzung: Richtige Eingaben müssen zu richtigen Ausgaben führen. Das zweite ist die Geschwindigkeit. Ich möchte, daß der Assembler seine Arbeit so schnell wie möglich

erledigt. Das dritte ist die Genauigkeit der Übersetzung: Wenn ich Assemblercode schreibe, möchte ich ihn selbst optimieren. Ich möchte keinen optimierenden Assembler benutzen - ich hasse Überraschungen. Schließlich verwende ich ungern allzu 'schlaue' Operatoren, d.h. solche, die mir ein gewisses Maß an Denkfaulheit erlauben, wenn z.B. ADD manchmal ADDI, manchmal auch ADDQ, ADDA oder sonst etwas assembliert. Solche Operatoren sind langsamer und ihr Verhalten weniger durchsichtig. Da FORTH-Assembler erweiterbar sind, kann jeder Benutzer eigene 'schlaue' Operatoren hinzufügen, wenn er möchte.

Bei der Fehlerbehandlung kann im Assembler beliebiger Aufwand getrieben werden. Im Idealfall sollte ein Assembler nur korrekte Eingaben akzeptieren. Es kann allerdings vor allem in Bezug auf die Geschwindigkeit teuer werden, wenn man übertriebene Fehlerkontrolle einbaut. Zum Glück können viele Fehler sehr leicht entdeckt werden. Es ist einfach zu prüfen, ob sich die Stacktiefe während einer Definition verändert (kein Wert bleibt unzulässigerweise übrig bzw. wird verbraucht), ob die Kontrollstrukturen ausgeglichen sind usw.

Die nächste Stufe der Fehlererkennung ist die Prüfung auf erlaubte Adressierungsarten bei jedem Befehl. Bei einer sehr geradlinigen Prozessorarchitektur ist das sehr einfach. Unglücklicherweise ist der 68000 nicht ganz dazu geeignet, auch wenn oft das Gegenteil behauptet wird. Trotzdem können viele Befehle einfach überprüft werden. Obwohl ich so etwas gewöhnlich nicht benutze, habe ich einige Worte eingebaut, die nachprüfen, ob den Befehlen gültige Adressierungsarten zugeordnet sind. ??DN bricht ab, wenn keine direkte Adressierung eines Datenregisters vorliegt. ??AN führt das gleiche für ein Adreßregister durch. ??JMP bricht ab, wenn beim JMP-Befehl eine ungültige Adressierung benutzt wurde.

Für weitergehende Fehlererkennung muß zunehmender Aufwand bei abnehmender Wirkung getrieben werden.

Gebrauchsanleitung für den Assembler

Eine detaillierte und ziemlich genaue Beschreibung des Motorola MC68000 findet man im entsprechenden User's Manual. Als Beispiel für die Benutzung des Assemblers nehmen Sie bitte die Definition des Wortes FILL, das einen Speicherbereich mit einem vorgegebenen Byte füllt. Es wird folgendermaßen benutzt:

```
adresse länge byte  FILL
```

Beachten Sie, daß FILL drei Parameter vom Stack benötigt und nichts übrig läßt.
(Das folgende Beispiel wurde so abgeändert, daß es dem volksFORTH-83 entspricht. Näheres zu den Macros s.u. - Anm. d. Übers.)

```
Code fill   ( adr len val -- )
 SP )+ D0 move      \ Wert nach D0
 SP )+ D1 move      \ Länge nach D1
 SP )+ D6 move      \ Adresse nach D6,
 D6 reg) A0 lea     \ reg)  ist ein Macro,  das aus der  16  Bit
```

```
                    Systemadresse eine absolute 32-Bit Adresse
                    berechnet.
 D1 tst   0<> IF    \ Wenn Länge von 0 verschieden
  1 D1 subq         \ decrement D1; dbra läuft bis -1, nicht 0
  D1 D0    .b D0 A0 )+ move   LOOP
                    \ Schleife  bis D1 = -1;  jedesmal wird  ein
                    Byte in die
                    \ Adresse,  die in A0 steht, geschrieben und
                    A0 incrementiert.
  THEN
Next                \ ein Macro, das zum nächsten Wort springt.
end-code            \ beendet die Definition
```

Das Wort CODE ist ein definierendes Wort. Es erzeugt einen Kopf für das neue Wort FILL und setzt dessen Codefeld auf das Parameterfeld. Das System bleibt im interpretierenden Modus. Der Assembler benutzt den FORTH-Compiler nicht, wie häufig fälschlich angenommen wird. Der Kopf ist so etwas wie ein Eintrag in eine Symboltabelle. Das Codefeld eines jeden Wortes zeigt auf den Code, den dieses Wort ausgeführen soll. Normalerweise zeigen alle Worte, die mit denselben defining words erzeugt worden sind, auf den gleichen Code. Worte, die mit CODE erzeugt werden, bestehen aus einem einzigartigen Code-Segment, das immer auf das Parameterfeld eben dieses Wortes zeigt. Die übrigen Worte der Codedefinition erzeugen eine Bytefolge im Parameterfeld des Wortes.

Assembler-Opcode-Worte wie MOVE benutzen das Wort comma, um der Reihe nach Bytes in das Parameterfeld einzutragen. Wenn das neue Wort nach seiner Ausführung in den FORTH-Interpreter zurückkehren soll, muß die letzte Anweisung NEXT sein. Next ist ein Makro, das einen Sprung in den FORTH-Interpreter assembliert. Seine Definition lautet (im volksFORTH83):

```
: Next     IP )+ D7 move      \ D7 enthält cfa
           D7 reg) D6 move    \ D6 enthält cfa@
           D6 reg) jmp        \ Sprung auf cfa@
;
```

(Das Macro reg) wird weiter unten beschrieben. Anm. d. Übers.) JMP benutzt comma, um den richtigen Opcode und die Adresse einzutragen. END-CODE markiert das Ende, prüft auf Fehler und räumt ein bißchen auf.

SP ist der Name des Stackpointers der virtuellen FORTH-Maschine. Das Wort SP hinterläßt einen Wert auf dem Stack, der den 'direct-adressing' Modus mit Register A6 (volksFORTH83) darstellt. Das Wort A6 hat genau die gleiche Wirkung; beides sind einfache Konstanten. Das Wort)+ modifiziert den Wert auf dem Stack, den SP hinterlassen hat, um die 'indirect mit autoincrement' Adressierung anzuzeigen. Wie das funktioniert, wird später erklärt. Das Wort D0 stellt Datenregister 0 dar.

Das Wort MOVE assembliert einen 68000 move-Befehl. Es benötigt zwei Werte, die jeweils eine Adressierungsart beinhalten. In unserem Beispiel wird der assemblierte Code 16 Bit aus der Adresse, auf die SP zeigt, nach D0 transportieren und dabei SP

um zwei erhöhen. Die Länge der Operation wird von der Variablen SIZE festgelegt , die auf 16-Bit voreingestellt ist. SIZE wird durch .B (Byte), .W (Word) und .L (Long) entsprechend gesetzt.

Das Wort CODE schaltet das Assembler-Vokabular ein, damit bei gleichen Worten im Assembler und im übrigen System (z.B. SWAP) das richtige Wort gefunden wird. Das Wort LMOVE wurde zusätzlich definiert als Spezialfall von MOVE für die oben angesprochene Registerverschiebung. LMOVE assembliert immer einen Long move, ohne dabei SIZE zu verändern. Beachten Sie den Gebrauch von DO und LOOP im Assembler. DO erhält ein Datenregister zugeordnet, das den Schleifenzähler für die Ausführungsphase enthält. DO übergibt HERE und das Register an LOOP, welches einen dbra zurück auf DO assembliert, bei dem das angegebene Datenregister benutzt wird. (In ähnlicher Weise assembliert 0<> IF einen beq (!), dessen Offset beim folgenden THEN berechnet wird. Beachten Sie bitte, daß die Sprungbedingungen vor IF immer gerade entgegengesetzt den Sprungbefehlen sind, also beq bei 0<> oder bne bei 0=. Anm. d. Übers.)

Implementation

Es gibt viele mögliche und darunter zwei häufiger beschrittene Wege, um einen Assembler in FORTH zu schreiben. Eine Methode ist, viele Variable mit Status-Informationen zu benutzen, die ihrerseits von den Mnemonic-Worten verwendet werden, um den Assembliervorgang zu steuern. Nach jeder Instruktion werden sie gelöscht, um von der nächsten Instruktion wieder verwendet werden zu können. Bei diesem Assembler ist eine weiter verbreitete und auch wünschenswertere Methode gewählt worden. Fast sämtliche Informationen werden auf dem Stack übergeben, der auch nicht initialisiert werden muß.

Ebenso gibt es zwei verbreitete Arten, die Adressierungsart an das assemblierende Wort zu übergeben. Eine Möglichkeit besteht in einer Art geschachtelter IF...ELSE Strukturen, die aus einer Folge von Möglichkeiten die richtige heraussucht. Der andere Weg, hier eingesetzt, besteht darin, daß die Worte, die die Adressierungsart bestimmen, die Werte, die ihnen übergeben wurden, in irgendeiner Form verändern. Dies geschieht durch Ausmaskieren mit AND und Setzen einzelner Bits mit OR. Solche Logikoperationen arbeiten bekanntlich viel schneller als Verzweigungen, sodaß der Assembler insgesamt mit solchen Operationen schneller wird.

Wenn Sie die folgenden Beschreibungen lesen, sollten Sie sich den Quelltext des Assemblers zur Hand nehmen. Er befindet sich auf Ihrer Diskette im File ASSEMBLE.SCR.

Die Grundidee, die hinter diesem Assembler steckt, ist die Betrachtung einer Maschinencodeinstruktion als Reihe von Bit-Feldern. Diese Bit-Felder sind im Manual der CPU beschrieben. Einige sind für viele Instruktionen gültig wie source und destination, mode und register Felder.

```
|op-code|dest reg|dest mode|source mode|source reg|
|15   12|11    9|8       6|5         3|2        0|
```

Wie bereits erwähnt, benutzen Instruktionen, die die Datenlänge kennen müssen, die Variable SIZE. Die Position des Bit-Felds, das die Datenlänge bestimmt, wechselt von Befehl zu Befehl mehr als alle anderen. Fast immer werden die benutzten Werte in die Variable SIZE übergeben, und zwar durch .B, .W oder .L. Beachten Sie, daß ich an diesem Punkt BASE auf OCTAL umgeschaltet habe. Die 68000-Befehle enthalten viele 3-Bit-Felder und können daher besonders übersichtlich als Oktalzahlen dargestellt werden. Ich war gezwungen, meine Vorliebe für Hexzahlen zeitweilig zurückzustellen.

Bei der Definition der Worte, die Register und Adressierungsarten festlegen, habe ich zu einem kleinen Trick gegriffen. Ich benutzte ein 'Multi-defining word' REGS, das in einer Schleife CONSTANT ausführt, um ähnliche Konstanten zu erzeugen. REGS wird nur zweimal benutzt. Einmal für Datenregister und einmal für Adreßregister, die Modus 0 bzw. 1 darstellen.

Modus 0 ist 'data register direct', daher ist D5 eine Konstante mit dem Wert 5005.

Modus 1 ist 'adress register direct', daher ist A3 eine Konstante mit dem Wert 3113.

Worte, die mit MODE definiert wurden, werden nach einem Adreßregister benutzt und ersetzen die zwei Moduszíffern (in unserem Fall 1) mit den neuen Modus-Werten. Dies geschieht durch Ausmaskieren der alten Werte mit AND und Setzen der neuen mit OR. Alle MODE-Worte sind 'adress register indirect' mit Zusätzen.

Modus 2 ist 'adress register indirect',
daher ergibt A6) 6226.

Modus 3 ist dasselbe mit 'post-increment',
daher ergibt A7)+ 7337.

Modus 4 ist dasselbe mit 'pre-decrement',
daher ergibt A7 -) 7447.

Modus 5 ist dasselbe mit 'displacement',
daher ergibt 123 A1 D) 1551
mit dem 'displacement' Wert von 123, der zunächst unter dem Register/Modus Wert auf dem Stack liegt.

Modus 6 ist dasselbe mit index und displacement,
daher ergibt 123 D4 A1 DI) 1661.
Auf dem Stack liegen darunter 4004 und 123.

Modus 7 wird für alle übrigen Adressierungsarten verwendet, die sich durch ihre Registerfelder unterscheiden. Diese Modi sind als Konstanten definiert.

#) ist 0770 und stellt die absolute (16-Bit) Adressierung dar. Der Name bedeutet 'immediate indirect'. (Denken Sie darüber nach!)

L#) ist 1771 und stellt die absolute (32-Bit) Adressierung dar.

PCD) ist 2772 und stellt den 'program counter relative mit displacement' Modus dar. 123 PCD) ergibt 2772 und darunter liegt 123 auf dem Stack.

PCDI) ist 3773 und stellt den 'program counter relative displaced, indexed' Modus dar. 123 D4 PCDI) ergibt 3773 und darunter 4004 und 123 auf dem Stack.

ist 4774 und stellt den 'immediate data' Modus für 16 oder 32 Bit dar. 456 # ist 4774, darunter liegt 456.

(Anmerkung d. Übers.: Zusätzlich haben wir ins volksFORTH83 einige weitere Adressierungsarten aufgenommen. Mehr dazu weiter unten.)

Beachten Sie, daß immer 1 bis 3 16-Bit-Werte auf dem Stack hinterlassen werden, die die Adressierungsart kennzeichnen. Der oberste Wert wird normalerweise Teil der ersten 16 Bit eines Befehls zusammen mit dem Opcode. Falls zusätzliche Werte vorhanden sind, werden sie im Anschluß an den Opcode assembliert.

Manche 3-Bit-Felder werden häufiger (durch Ausmaskierung) selektiert als andere. Das Wort FIELD erzeugt Worte, mit denen man solche Felder selektieren kann. RS und RD wählen die source und destination register Felder (s. Bild oben) aus. MS selektiert das source mode Feld. Der erzeugende Ausdruck für eine vollständig festgelegte Adressierungsart ist eine effektive Adresse (EA). Das Wort EAS wählt die 'source effective adress', die aus den source mode und register Feldern besteht. LOW selektiert die unteren 8 Bits. Das Opcode-Wort enthält oft ein EAS Feld. Das Wort SRC führt

```
OVER EAS OR
```

aus, womit es dieses Feld ins Opcode-Wort überträgt. Das Wort DST baut das destination register Feld ein.

Die virtuelle FORTH-Maschine enthält fünf Register. Diese sind einzelnen 68000 Registern zugeordnet. Es ist im Assembler möglich, sowohl die 68000 Register-Namen als auch die Namen der Register der virtuellen Maschine zu benutzen. (Bemerkung d. Übers.: Sie sollten die 68000 Registernamen nur dann benutzen, wenn sie keine FORTH-Register meinen. Sie vermeiden so unerklärliche Systemabstürze, falls die Registerzuordnung sich ändert.)

Adressierungsarten, die nach dem Opcode weitere Werte assemblieren, nennt man 'extended adressing'. Solche Adressierungsarten werden mit sechs Worten und einem Buffer abgehandelt. DOUBLE? hinterläßt ein Flag, das wahr ist, falls der Modus, der oben auf dem Stack liegt, zusätzliche 32 Bit verlangt. INDEX? sucht nach einer Adressierungsart und verändert seine zusätzlichen Werte in das passende Format, falls es sich um 'indexed' Adressierung handelt. MORE? hinterläßt ein True-Flag, wenn die Adressierungsart weitere Werte benötigt. MORE trägt alle zusätzlichen Werte hinter dem Opcode-Wort ins Dictionary ein.

Einige Instruktionen brauchen zwei Adressierungsarten, eine für source und eine für destination. Der source Modus wird zuerst

festgelegt, sodaß er unter dem destination Modus auf dem Stack liegt. Jede Adressierungsart besteht aus ein bis drei Werten, die auf dem Stack liegen. Der source Modus wird vor dem destination Modus verarbeitet, daher müssen die Werte für den destination Modus so lange in einem Buffer abgelegt werden. EXTRA? rettet alle zusätzlichen Werte in einen Buffer namens EXTRA und hinterläßt nur den Wert für die Adressierungsart. ,EXTRA nimmt die zusätzlichen Werte aus dem Buffer und trägt sie ins Dictionary ein.

Fast der ganze Rest des Assemblers besteht aus Definitionen und der Anwendung von definierenden Worten, die Gruppen von Mnemonics herstellen. Zwei Beispiele werden genügen. Wenn Sie mit dem Gebrauch von definierenden Worten nicht vertraut sind - Sie sollten es sein; sie sind die leistungsfähigste Struktur in FORTH.

Das Wort IMM erzeugt Worte, die 'immediate' Befehle assemblieren. Ich werde die Definition hier wiederholen und genau erläutern:

```
: IMM     CONSTANT
    DOES>  @ >R  EXTRA?  EAS  R>  OR
           SZ3 ,  LONG?  ?,  ,EXTRA ;

Gebrauch bei der Definition:   3000 IMM ADDI
Gebrauch im Assembler:         n ea ADDI
Beispiel:                      123 A5 ) ADDI
```

Jedes Mal, wenn mit IMM ein Mnemonic-Wort definiert wird, speichert es einen konstanten Wert in der Definition dieses Wortes ab, der es von anderen 'immediate' Worten unterscheidet. Dieser Wert ist der Opcode des Befehls. Immediate Befehle beinhalten folgende Bit-Felder.

```
| op-code |  size  |  mode  |   reg  |
|15     8|7     6|5     3|2     0|
```

Diesen folgen 16 oder 32 Bit Daten. Wenn das Befehlswort ausgeführt wird, führt es den Code nach DOES> in IMM aus mit der Adresse seines eigenen Parameterfeldes auf dem Stack. An dieser Adresse ist die Konstante (der Opcode) kompiliert. Das Wort @ liest diesen Wert und rettet ihn mit >R auf den Returnstack. ADDI erhält die 'immediate' Daten auf dem Stack unterhalb der Werte für die Adressierungsart. EXTRA? rettet alle zusätzlichen Werte, EAS selektiert die mode und register Felder, die benutzt werden sollen. Dann wird der Opcode vom Returnstack mit R> geholt und durch OR mit EAS verknüpft. SZ3 setzt die zugehörigen Längenbits aus SIZE, und das Wort comma trägt das Opcode-Wort ins Dictionary ein. Jetzt liegen nur noch die Daten auf dem Stack, und LONG? entscheidet, ob ?, 16 oder 32 Bit anhängen soll. Zum Schluß holt ,EXTRA die geretteten zusätzlichen Werte, falls vorhanden, zurück und hängt sie ebenfalls an.

Zahlreiche andere ähnlich aufgebaute definierende Worte werden benutzt, um die meisten übrigen Befehle zu definieren. Viele dieser Wort bilden für sich eine Gruppe und sind deswegen mit dem Wort : definiert, gerade so wie Makros. Die conditional

Befehle sind so regelmäßig, daß ich noch ein 'trick defining word' benutze. SETCLASS verwendet wiederholt ein vorhandenes defining word, jedesmal mit einem anderen Argument, um mehrere Mnemonic-Worte auf einmal zu definieren. Alle 46 conditional Befehle werden definiert, indem jedes der drei defining words 16 mal aufgerufen wird. Dabei entstehen auch zwei ungültige Mnemonics, die nicht weiter benutzt werden. Vielleicht wäre es in diesem Falle besser gewesen, alle 46 Befehle einzeln zu definieren, aber ich wollte zeigen, was alles machbar ist.

Zum Schluß kommen wir zu den structured conditionals. Betrachten Sie folgendes Beispiel:

```
          A3 )+ D1 CMP  0<
IF        D0 A7 )  MOVE
ELSE      A7 ) D0  MOVE
THEN


BEGIN     A3 D2 CMP  0=
WHILE     A0 )+ D0 MOVE
REPEAT
```

Im ersten Beispiel beeinflußt das Ergebnis des Vergleichs bestimmte Flags im Status-Register. IF assembliert einen bedingten Sprung, dessen Opcode (mit Bedingung) durch 0< festgelegt ist. Dieser assembliert also den Wert für einen BGE (Branch greater or equal). ELSE berechnet den Sprungoffset für IF und assembliert einen unbedingten Sprung; THEN berechnet dessen Offset.

Ebenso berechnet WHILE einen bedingten Sprung hinter REPEAT, welches wiederum einen unbedingten Sprung zurück auf BEGIN assembliert.

Beachten Sie, daß keine Labels nötig sind. Der meiste Wirrwarr in normalen Assembler-Quelltexten entsteht durch die riesige Anzahl an bedeutungslosen Labelnamen für Sprungziele. Beachten Sie auch, daß die structured conditionals, die wir hier definiert haben, nur 1-Byte Offsets benutzen. Da der Assembler nur einen Pass durchläuft, muß der Platz für den Sprungoffset frei gehalten werden, bevor seine Größe bekannt ist. Da Coderoutinen in FORTH immer sehr kurz sind, genügt ein Byte für den Offset. Sollte das nicht ausreichen, ersetze ich einfach diese Definitionen durch sehr ähnliche, die einen 16-Bit Offset benutzen.

Schließlich sollte bemerkt werden, daß es keine Worte für die Einrichtung von Datenstrukturen in diesem Assembler gibt. Ein FORTH-Assembler ist Teil einer FORTH-Umgebung; und auf alle Datenstrukturen, die mit normalen FORTH-Worten erzeugt wurden, kann der Assembler zugreifen.
Ein Beispiel:

```
VARABLE  FOO
CODE BAR     FOO R#) NEG  NEXT END-CODE
```

BAR negiert den Inhalt der Variablen FOO.

volksFORTH83 Assembler

Wie bereits gesagt, beruht der Assembler im volksFORTH83 im Wesentlichen auf dem von Michael Perry. Es wurden jedoch einige neue Befehle implementiert, die die besonderen Möglichkeiten von volksFORTH83, z.B. den Heap, ausnutzen, oder sich aufgrund der relokatiblen Struktur als notwendig herausgestellt haben.

Struktur des relokatiblen volksFORTH83

volksFORTH83 ist ein 16-Bit-System, d.h. es lassen sich 64 kByte Speicher adressieren. Bisher lagen diese 64 kByte in einer Speicherbank (ab $50000), sodaß die 16-Bit-Adressen im FORTH-System mit Hilfe einer Konstanten namens mempage (=0005) auf 32-Bit-Adressen erweitert werden konnten. Diese Struktur hat sich jedoch als sehr unflexibel erwiesen, so war es z.B. nicht möglich, RAM-Disks zu benutzen. Wenn man stand-alone-Applikationen erzeugen wollte, mußte man dazu eigens ein Ladeprogramm schreiben, dessen einziger Sinn darin bestand, das eigentliche Programm nach $50000 zu laden und dort zu starten.

Die neue Struktur geht davon aus, daß das System an beliebiger Stelle im Speicher lauffähig sein soll (relokatibel). Dementsprechend dürfen keine absoluten 32-Bit-Adressen im System mehr vorkommen, weil sonst zusätzliche Relokationsinformationen mit abgespeichert werden müßten, was ein SAVESYSTEM nahezu unmöglich machen würde. Glücklicherweise bietet der 68000-Prozessor eine recht leistungsfähige Adressierungsart, nämlich die 'indirekte Adressierung mit Index und Displacement'. Grundlage ist dabei ein Adreßregister, daß auf Byte 0 des FORTH-Systems zeigt und Forthpointer (FP) heißt. Alle Speicheroperationen müssen nun relativ zu diesem Zeiger erfolgen.

Die 16-Bit-Adressen im FORTH-System werden nun als Offset zum Byte 0 des Systems aufgefaßt und als Index in einem Datenregister benutzt. Unglücklicherweise werden Datenregister, die als Index verwendet werden, auf 32 Bit vorzeichenerweitert, wenn sie nur wortweise adressiert werden. Der Assembler mußte daher so abgeändert werden, daß bei den Adressierungsarten DI) und PCDI) die Länge der Operation unabhängig von der Länge des Indexregisters angegeben werden kann. Ein Beispiel: Die Befehlsfolge

```
.l 0 D6 FP DI) .w D0 move
```

addiert zu FP den Wert des Datenregisters D6 lang, also ohne Vorzeichenerweiterung, geMOVEd werden aber nur 16 Bit. Steht also in D6 die 16-Bit-Adresse einer FORTH-Variablen, z.B. $9000 und zeigt FP auf $12300, würde diese Befehlsfolge den 16-Bit-Inhalt der Variablen, die absolut bei $1B300 liegt, ins Register D0 bringen.

Zusätzliche Adressierungsarten

Um das Programmieren in Assembler nun nicht noch schwieriger zu machen als es ohnedies schon ist, haben wir einige Macros

entwickelt, die den Charakter von Adressierungsarten haben.

```
: reg)    size push   .l  0 swap  FP DI) ;
```

Diese Adressierungsart könnte 'Datenregister indirect' benannt werden. Ein Beispiel:

```
Code @   ( addr -- 16b )
   SP )+ D6 move   D6 reg) SP -) move   Next
```

assembliert genau dasselbe wie

```
Code @   ( addr - 16b )
   SP )+ D6 move   .l 0 D6 FP DI) .w SP -) move   Next
```

Die Adresse auf dem Stack wird in D6 geladen; D6 wird lang zum Forthpointer addiert und der Inhalt dieser Speicherstelle wieder auf den Stack gebracht. (Anmerkung: @ im System ist etwas komplizierter definiert, da auch der Zugriff auf ungerade Adressen gestattet ist, was bei der obigen Definition eine Fehlermeldung auslösen würde.)

Zum Zugriff auf Datenstrukturen, vor allem auf Variablen, dient das Macro

```
: R#)   ( addr -- )      size push  .w
  dup 0< IF  # D6 move   D6 reg)   exit THEN
  FP D) ;
```

Dieses Macro gab es auch schon im 'alten' System, der Effekt ist auch der gleiche, sodaß vorhandene Assemblerquelltexte nicht umgeschrieben werden müssen. Die Wirkungsweise hat sich jedoch geändert: Ist addr negativ - also größer als $7FFF - wird die Adresse direkt nach D6 geladen und dann wie bereits beschrieben mit D6 indiziert. Liegt addr jedoch in den unteren 32 kByte des FORTH-Systems, erfolgt der Zugriff mit addr als Displacement ohne Indexregister, was erhebliche Geschwindigkeitsvorteile bringt. Vor allem die FORTH-Systemvariablen lassen sich auf diese Art und Weise schnell erreichen. Ein weiteres Beispiel:

```
Variable test
Code test@   ( -- 16b )
   test R#) SP -) move   Next
```

TEST@ legt den Inhalt der Variablen TEST auf den Datenstack.

Schließlich gibt es noch das Macro PCREL), dessen genauere Definition im Assemblerquelltext ASSEMBLE.SCR nachzulesen ist. Es arbeitet analog zu R#), ist aber schneller und kürzer. Nachteil ist, daß der 68000 die Adressierungsart PC-relativ nicht bei allen Operatoren zuläßt, und daß nicht der gesamte 64 kByte Adreßraum erreichbar ist. PCREL) ist mit Fehlermeldungen gegen falsche Adreßdistanzen abgesichert.

Register der virtuellen FORTH-Maschine

Folgende Register werden vom volksFORTH83 benutzt:

A3 Forthpointer (FP). Dieses Register enthält die Startadresse des Systems und gibt damit die Basis für alle relativen Adreßzugriffe ins System. Grundsätzlich gilt: Die absolute 32-Bit-Adresse erhält man durch Addition der 16-Bit-Forth-Adresse und FP. Die 16-Bit-Forth-Adressen lassen sich also als Offset zum Systemanfang auffassen.

A4 Instructionpointer (IP). Dieses Register enthält die (absolute) 32-Bit-CFA des nächsten auszuführenden Wortes.

A5 Returnstackpointer (RP). Dieses Register enthält den 'Systemstackpointer' der virtuellen FORTH-Maschine. Hier werden 'Rücksprungadressen' in aufrufende Worte usw. abgelegt.

A6 Datenstackpointer (SP). Dieses Register enthält den Datenstackpointer der virtuellen FORTH-Maschine. Über diesen Stack werden bekanntlich nahezu sämtliche Werte zwischen einzelnen Funktionen übergeben.

Neben diesen vier Adreßregistern werden noch zwei Datenregister benutzt:

D7 Work-Register (W) der virtuellen FORTH-Maschine; der 16-Bit-Wert in diesem Register zeigt auf die CFA des Wortes, das gerade ausgeführt wird.

D6 Ein Hilfsregister, das zur Umrechnung von relativen 16-Bit-Adressen in absolute 32-Bit-Adressen benutzt wird. Dem Programmierer stehen also die Datenregister D0 - D5 sowie die Adreßregister A0 - A2 zur freien Verfügung. D6 und D7 dürfen verändert werden, jedoch müssen die oberen 16 Bit immer auf 0 stehen. Sollen weitere Register verwendet werden, müssen sie vorher gerettet und anschließend restauriert werden.

Zusätzliche Befehle

;c: Schaltet den Assembler ab und den FORTH-Compiler an. Damit ist es möglich, von Maschinencode in FORTH überzuwechseln. Ein Gegenstück ist nicht vorhanden.

Ein Beispiel für die Verwendung von ;c: ist:

```
.... 0< IF  ;c: ." Fehler" ; Assembler  THEN
```

Ist irgendwas kleiner als Null, so wird 'Fehler' ausgedruckt und die Ausführung des Wortes abgebrochen, sonst geht es weiter im Code.

Schließlich gibt es noch die Worte >LABEL und LABEL. >LABEL erzeugt eine Konstante auf dem Heap, wobei es den Wert des

Labels vom Stack nimmt. LABEL erzeugt eine Konstante mit dem Wert von HERE. Beispiel:

```
LABEL SCHLEIFE  1 D0 subq  SCHLEIFE BNE
```

SCHLEIFE verbraucht keinen Dictionaryspeicher, weil es vollständig - mit Header und Wert - auf dem Heap liegt. Es sind also echte Assemblerlabels möglich. Von der Verwendung solcher Labels kann allerdings nur abgeraten werden, wenn wie im obigen Beispiel ebensogut strukturiert programmiert werden könnte.

Übrigens ist >LABEL statesmart, d.h es verhält sich verschieden, je nachdem, ob das System kompiliert oder nicht. Während der Kompilation werden Labels als Literals kompiliert. Damit können Labels auch in Colon-Definitionen verwendet werden.

Der Disassembler

Der Disassembler wird geladen mit

```
include disass.scr
```

Mit DISW <word> kann man in Code geschriebene Worte disassemblieren. Das funktioniert natürlich mit Systemworten ebensogut wie mit eigenen Definitionen. Die Ausgabe stoppt automatisch, wenn ein NEXT erkannt wurde, und kann dann mit einer beliebigen Taste fortgesetzt oder mit Escape abgebrochen werden. Auch während der Ausgabe kann mit einer beliebigen Taste unterbrochen oder mit Escape abgebrochen werden.

Der Disassembler disassembliert die Befehle in der üblichen Motorola-Syntax, nicht in der Schreibweise des FORTH-Assemblers. Wer häufiger mit Assembler-Quelltexten zu tun hat, wird das zu schätzen wissen....

Weitere Benutzer-Worte des Disassemblers sind

```
dis   ( addr -- )
```

und

```
ldis   ( laddr -- )
```

Beide disassemblieren von einer vorgegebenen FORTH- bzw. Langadresse ab. Mit LDIS lassen sich auch Routinen außerhalb von FORTH disassemblieren, z.B. im TOS oder GEM. Die Ausgabe wird genauso gesteuert wie bei disw.

Sonstiges

Wie im Artikel von Michael Perry bereits zu lesen war, verzichtet der Assembler auf 'übertriebenes Errorchecking'. Im Klartext heißt das, daß man sich schon recht gut mit dem 68000-Befehlsumfang auskennen sollte, insbesondere mit den erlaubten Adressierungsarten und Operandenlängen bei den einzelnen Befehlen. Anfängern - und nicht nur solchen (!) - sei dringend empfohlen, grundsätzlich den erzeugten Maschinencode mit dem Disassembler zu überprüfen. Man reduziert damit die Anzahl der sonst unvermeidlichen Systemabstürze erheblich.

Der FORTH-Assembler ist etwas gewöhnungsbedürftig. Beispiele für verschiedenartige Anwendungen sind vor allem im File FORTH_83.SCR zu finden. Diese Beispiele sollte man studieren, bevor man sich an eigene Experimente heranwagt. Man muß ja nicht gleich mit (WORD beginnen....

Fileinterface für das Voksforth83 auf dem Atari ST

Erster Einstieg !

Bevor Sie das Glossar lesen, sollten Sie diese kleine Einführung lesen und auf einer leeren Diskette die Beispiele ausprobieren.

Wie erzeuge ich ein File, in das ich ein Programm eingeben kann?

Geben Sie bitte folgendes ein:

```
MAKEFILE test.scr
```

Das File test.scr wird auf dem Laufwerk erzeugt, auf dem Sie das Forth gebootet haben. Als nächstes schätzen Sie bitte ab, wie lang Ihr Programm etwa wird. Beachten Sie dabei bitte, daß der Screen 0 eines Files für Hinweise zur Handhabung ihres Programms und der Screen 1 für einen sog. Loadscreen (das ist ein Screen, der den Rest des Files lädt) reserviert sind. Wollen Sie also z.B. 3 Screens Programm eingeben, so sollte das File 5 Screens lang sein; Sie geben also ein:

```
5 MORE
```

Fertig ! Sie haben jetzt ein File, das die Screens 0..4 enthält. Geben Sie jetzt

```
1 L
```

ein. Sie editieren jetzt den Screen 1 Ihres neuen Files test.scr . Sie können, falls der Platz nicht ausreicht, Ihr File später einfach mit MORE verlängern. Ein File kann leider nicht verkürzt werden. **Wie spreche ich ein bereits auf der Diskette vorhandenes File an?**

Das geht noch einfacher. Geben Sie einfach den Filenamen ein. Reagiert das System mit der Meldung "Haeh ?" , so kennt das Forth dieses File noch nicht. Sie müssen in diesem Fall das Wort USE vor dem Filenamen eingeben, also z.B.

```
USE test.scr
```

Jetzt können Sie wie oben beschrieben mit 1 L (oder einer anderen Zahl) das File editieren. Das Wort USE erzeugt übrigens im Forthsystem das Wort TEST.SCR, falls es noch nicht vorhanden war. Wissen Sie also nicht mehr, ob Sie ein File schon benutzt haben, so können Sie mit WORDS nachsehen oder das Wort USE voranstellen.

Wie erzeuge ich Files auf einem bestimmten Laufwerk, z.B. A: ?

Entweder durch Voranstellen des Pfadnamens oder durch Eingabe von :

A: Hierbei wird A: zum aktuellen Laufwerk gemacht.

Wie spreche ich Directories an ?

Hierbei gibt es verschiedene Methoden. Für den Anfang versuchen wir, Files in einem Directory "test.dir" zu suchen bzw. zu erzeugen. Dazu geben Sie ein :

```
dir test.dir
```

Jetzt werden in test.dir alle Files und Directories erzeugt und gesucht. Ist das Directory "test.dir" noch nicht vorhanden, so erzeugen Sie es mit :

```
makedir test.dir
```

Anschließend können Sie wieder DIR benutzen.

0) Allgemeines

Im folgenden wird die Benutzung des Fileinterfaces beschrieben. Dieses Fileinterface benutzt die Files des GEM-Dos und dessen Subdirectories.

Benutzt man ein File von Forth aus, so wird es in Blöcke zu je 1024 Bytes aufgeteilt, die in gewohnter Weise anzusprechen sind. Dies trifft auch für Files zu, die nicht vom Forth aus erzeugt wurden. Als Konvention wird vorgeschlagen, daß Files, die Forth-Screens, also Quelltexte, enthalten, mit .SCR erweitert werden. Files, die Daten enthalten, die nicht unmittelbar lesbar sind, sollten auf .BLK enden.

Zum Umschalten vom Filesystem auf Direktzugriff und umgekehrt gibt es das Wort

DIRECT (--)
schaltet auf Direktzugriff um. Auf den Filezugriff schalten wir durch das Nennen eines Filenamens um.

DOS (--)
Viele Worte des Fileinterfaces, die normalerweise nicht benötigt werden, sind in diesem Vokabular enthalten.

1) Der Directory-Mechanismus

Die Verwaltung von Directories entspricht in etwa der des GEM-Dos Command-interpreters COMMAND.PRG

PATH (--)

Dieses Wort gestattet es, mehrere Directories nach einem zu öffnenden File durchsuchen zu lassen. Das ist dann praktisch, wenn man mit mehreren Files arbeitet, die sich in verschiedenen Directories oder Diskstationen befinden. Es wird in den folgenden Formen benutzt:

```
PATH dir1;dir2;....
```

Hierbei sind <dir1>, <dir2> etc. Pfadnamen. Wird ein File auf dem Massenspeicher gesucht, so wird zunächst <dir1>, dann <dir2> etc. durchsucht. Zum Schluß wird das File dann im aktuellen Directory (siehe DIR) gesucht. Beachten Sie bitte, daß keine Leerzeichen in der Reihe der Pfadnamen auftreten dürfen.

Beispiel: PATH A:\;B:\COPYALL.DEM;\AUTO\

In diesem Beispiel wird zunächst die Diskstation A, dann das Directory COPYALL.DEM auf Diskstation B und schließlich das Directory AUTO auf dem aktuellen Laufwerk (siehe SETDRIVE, A: und B:) gesucht. Beachten Sie bitte das Zeichen "\" vor und hinter AUTO . Das vordere Zeichen "\" steht für das Hauptdirectory. Wird es weggelassen, so wird AUTO\ an das mit DIR gewählte Directory angehängt. Das hintere Zeichen "\" trennt den Filenamen vom Pfadnamen.

Außerdem können Sie eingeben :

```
PATH ;
```

In diesem Fall werden alle Pfade gelöscht und es wird nur noch das aktuelle Directory durchsucht.
Schließlich geht noch :

```
PATH
```

Dann druckt PATH die Reihe der Pfadnamen aus.
Dieses Wort entspricht dem Kommando PATH des Kommandointeters.

MAKEDIR cccc (--)

Erzeugt im aktuellen Directory (siehe DIR) ein Directory mit Namen CCCC.
Dieses Wort entspricht dem Kommando MD des Kommandointerpreters.

A: (--)

Macht die Diskstation A: zum aktuellen Laufwerk (siehe SETDRIVE).
Dieses Wort entspricht dem Kommando A: des Kommandointerpreters.

B: (--)
Macht die Diskstation B: zum aktuellen Laufwerk (siehe SETDRIVE).
Dieses Kommando entspricht dem Kommando B: des Kommandointerpreters.

C: (--)
D: (--)
Analog zu A: und B: .

SETDRIVE (n --)
Macht die Diskstation mit der Nummer n zu aktuellen Laufwerk. Hiebei entspricht n=0 der Diskstation A, n=1 der Diskstation B usw.
Auf dem aktuellen Laufwerk werden Files und Directories erzeugt (und nach Durchsuchen von PATH gesucht). Das Directory, in dem Files erzeugt werden, wird mit DIR angegeben.

DIR (--)
Wird in den folgenden Formen benutzt:

```
DIR cccc
```

<cccc> ist ein Pfadname, an den neu zu erzeugende Files und Directories angehängt werden.
Beispiel:

```
A: DIR AUTO
```

erzeugt alle folgenden Files im Directory AUTO auf der Diskstation A. In <cccc> können alle vom GEM-Dos zugelassenen Zeichen außer Leerzeichen verwendet werden.

Außerdem geht noch

```
DIR
```

Ohne weitere Eingaben wird der aktuelle Suchpfad einschließlich des Laufwerks angezeigt.
Dieses Wort entspricht dem Kommando CD des Kommandointerpreters.

FILES (--)
Listet den Inhalt des aktuellen Directories (siehe DIR und SETDRIVE) auf dem Bildschirm auf. Subdirectories werden durch ein vorangestelltes "D" gekennzeichnet.
Dieses Wort, zusammen mit dem Wort FILES" entspricht dem Kommando DIR des Kommandointerpreters.

FILES" (--)
Benutzt in der Form

```
FILES" cccc"
```

Listet die Files auf, deren Name CCCC ist. Der String CCCC darf die bekannten Wildcards sowie einen Pfadnamen enthalten. Wird kein Pfadname bzw. Laufwerk angegeben, so

werden die Files des aktuellen Directories bzw. Laufwerks ausgegeben.

SAVESYSTEM (--)
Benutzt in der Form

```
SAVESYSTEM <name>
```

Damit läßt sich ein FORTH-System im gegenwärtigen Zustand unter dem Namen <name> abspeichern. Dieses Wort wird benutzt, um fertige Applikationen zu erzeugen. In diesem Fall sollte das Wort, das die Applikation ausführt, in 'COLD gepatched werden. Ein Beispiel: Sie haben ein Kopierprogramm geschrieben; das oberste ausführende Wort heißt COPYDISK. Mit der Sequenz

```
' COPYDISK IS 'COLD    <Return>
SAVESYSTEM COPY.PRG    <Return>
```

erhalten Sie ein Progamm namens COPY.PRG, das sofort beim Start COPYDISK ausführt.

2) Files

Files bestehen aus einem Forthnamen und einem GEM-Dos-Namen, die nicht übereinstimmen müssen.
Ist das Forthwort, unter dem ein File zugreifbar ist, gemeint, so wird im folgenden vom Forthfile gesprochen. Ist das File auf der Diskette gemeint, das vom GEM-Dos verwaltet wird, so wird vom GEM-File gesprochen. Durch das Nennen des Forthnamens wird das Forthfile (und das zugeordnete GEM-File) zum aktuellen File, auf das sich alle Operationen wie LIST, LOAD, CONVEY usw. beziehen.
Beim Öffnen eines Files wird die mit PATH angegebene Folge von Pfadnamen durchsucht. Ist ein File einmal geöffnet, so kann diese Folge beliebig geändert werden, ohne daß der Zugriff auf das File behindert wird. Aus Sicherheitsgründen empfiehlt es sich aber, Files so oft und so lange wie irgend möglich geschlossen zu halten. Dann kann eine Änderung der Folge von Pfadnamen dazu führen, daß ein File nicht mehr gefunden wird.

FILE (--)
Wird in der Form:

```
FILE <name>
```

benutzt.
Erzeugt ein Forthwort mit Namen <name>. Wird <name> später ausgeführt, so vermerkt es sich als aktuelles File. . Ebenso vermerkt es sich als FROMFILE, was für CONVEY wichtig ist. Einem Forthfile wird mit MAKE oder ASSIGN ein GEM-File zugeordnet.

MAKE (--)
Wird in der Form:

```
MAKE cccc
```

benutzt. Erzeugt ein GEM-File mit Namen cccc im aktuellen Directory und ordnet es dem aktuellen Forthfile zu. Das File wird auch gleich geöffnet. Es hat die Länge Null (siehe MORE).

Beispiel: FILE test.scr test.scr MAKE test.scr

erzeugt ein Forthwort TEST.SCR und ein File gleichen Namens. Alle Operationen wie LOAD , LIST usw. beziehen sich nun auf den entsprechenden Screen in TEST.SCR. Beachten Sie bitte, daß dieses File noch leer ist und daher eine Fehlerbedingung besteht, wenn Zugriffsoperationen ausgeführt werden sollen.

MAKEFILE (--)
Wird in der folgenden Form benutzt:

MAKEFILE <name>

Erzeugt ein Forthfile mit Namen <NAME> und erzeugt anschließend ein GEM-File mit demselben Namen. Die Sequenz

FILE <name> <name> MAKE <name>

würde genau dasselbe bewirken.

ASSIGN (--)
Wird in der Form

ASSIGN cccc

benutzt. Ordnet dem aktuellen File das GEM-File mit Namen CCCC zu. Eine Fehlerbedingung besteht, wenn das File nicht gefunden werden kann.

USE (--)
Dieses Wort ist das wichtigste Wort zum Auswählen von Files.
Es wird in der folgenden Form benutzt:

USE <name>

Dieses Wort macht das File mit Namen <NAME> zum aktuellen File, auf das sich LOAD, LIST usw. beziehen. Es erzeugt ein Forthfile mit Namen <NAME> , falls der Name noch nicht vorhanden war.
Anschließend wird das File in den Directories, die mit PATH angegeben wurden, gesucht. Wird es dort nicht gefunden, wird das mit SETDRIVE (bzw. A: und B:) und DIR angegebene Directory durchsucht. Wird auch dort kein GEM-Dos File mit dem Namen <NAME> gefunden, so wird die Fehlermeldung FILE NOT FOUND ausgegeben. Das (automatisch) erzeugte Forthfile verbleibt im Dictionary und muß ggf. mit FORGET vergessen werden.

CLOSE (--)
Schließt das aktuelle File. Dabei wird das Inhaltsverzeichnis der Diskette aktualisiert und die sog. Handlenum-

mer wieder freigegeben. Es werden die zu diesem File gehörenden geänderten Blöcke auf die Diskette zurückgeschrieben und alle zu diesem File gehörenden Blöcke gelöscht.

OPEN (--)
Öffnet das aktuelle File. Eine Fehlerbedingung besteht, wenn das File nicht gefunden werden kann. Die Benutzung dieses Wortes ist in den meisten Fällen überflüssig, da Files automatisch bei einem Zugriff geöffnet werden.

FROM (--)
Wird in der folgenden Form benutzt:

FROM <name>

<NAME> ist der Name eines Forthfiles, aus dem beim Aufruf von CONVEY und COPY Blöcke herauskopiert werden sollen.

Beispiel: filea 1 FROM fileb 3 copy

Kopiert den Block 1 aus FILEB auf den Block 3 von FILEA. Dieses Wort benutzt USE um das File auszuwählen. Das bedeutet, daß fileb automatisch als Forthfile angelegt wird, wenn es noch nicht im System vorhanden ist.

LOADFROM (n --)
Wird in der folgenden Form benutzt:

LOADFROM <name>

<NAME> ist der Name eines Forthfiles, aus dem der Block n geladen wird.

Beispiel: 15 LOADFROM filea

Lädt den Block 15 aus FILEA. Dieses Wort ist wichtig, wenn während des Ladens eines Files Teile eines anderen Files geladen werden sollen. Damit kann die Funktion eines Linkers imitiert werden. Dieses Wort benutzt USE , um filea zu selektieren. Das bedeutet, daß automatisch ein Forthfile mit Namen filea erzeugt wird, falls es im System noch nicht vorhanden war.
Beachten Sie bitte, daß dieses Wort nichts mit FROM oder FROMFILE zu tun hat, obwohl es ähnlich heißt !

INCLUDE (--)
Wird in der folgenden Form benutzt:

INCLUDE <name>

<NAME> ist der Name eines Forthfiles, das vollständig geladen wird. Dabei ist Voraussetzung, daß auf Screen 1 dieses Files Anweisungen stehen, die zum Laden aller Screens dieses Files führen. Siehe auch LOADFROM.

CAPACITY (-- u)
u ist die Länge des aktuellen Files. Beachten Sie bitte,

daß die Länge des Files um eins größer ist als die Nummer des letzten Blocks, da der Block 0 mitgezählt wird.

FORTHFILES (--)
Druckt eine Liste aller Forthfiles, zusammen mit den Namen der zugehörigen GEM-Files, deren Länge und Handlenummer aus.

FROMFILE (-- addr)
Addr ist die Adresse einer Variablen, die auf das Forth-File zeigt, aus dem CONVEY und COPY Blöcke lesen. Siehe auch FROM. Bei Nennen eines Forth-Files wird diese Variable gesetzt.

FILE? (--)
Druckt den Namen des aktuellen Forthfiles.

MORE (n --)
Verlängert das aktuelle File um n Screens. Die Screens werden hinten angehängt. Anschließend wird das File geschlossen.

(MORE (n --)
Wie MORE, jedoch wird das File nicht geschlossen.

EOF (-- f)
f ist ein Flag, das wahr ist, falls über das Ende des Files hinaus gelesen wurde. f ist falsch, falls auf das zuletzt gelesene Byte im File noch weitere Bytes folgen.

3) Verschiedenes

Beim Vergessen eines Forth-Files mit Hilfe von FORGET, EMPTY usw. werden automatisch alle Blockpuffer, die aus diesem File stammen, gelöscht, und, wenn sie geändert waren, auf die Diskette zurückgeschrieben. Das File wird anschließend geschlossen.

Bei Verwendung von FLUSH werden alle Files geschlossen. FLUSH sollte VOR jedem Diskettenwechsel ausgeführt werden, und zwar nicht nur, um die geänderten Blöcke zurückzuschreiben, sondern auch damit alle Files geschlossen werden. Sind nämlich Files gleichen Namens auf der neuen Diskette vorhanden, so wird sonst eine abweichende Länge des neuen Files vom Forth nicht erkannt. Bei Verwendung von VIEW wird automatisch das richtige File geöffnet.

Diverses

Im File DIVERSES.SCR sind einige Worte zusammengefaßt, die sich nicht so recht zuordnen lassen.

>absaddr (addr -- abs_laddr)
rechnet eine - relative - Adresse im FORTH-System in die entsprechende absolute 32-Bit-Adresse um.

.blk (--)
gibt die Nummer des gerade kompilierten Screens aus. Wenn Screen 1 eines Files kompiliert wird, wird zusätzlich auch der Filename in einer neuen Zeile ausgegeben.

abort((f --)
Wird benutzt in der Form
Bedingung ABORT(string)
Wenn f wahr ist, wird der String ausgegeben; entspricht abort" , ist jedoch im Direktmodus zu benutzen.

arguments (n --)
prüft, ob mindestens n Werte auf dem Stack liegen ; andernfalls wird mit einer Fehlermeldung abgebrochen.

cpush (addr len --)
wie PUSH, aber nicht für eine Variable, sondern für Speicherbereiche, die nach Ausführung eines Wortes auf die alten Werte zurückgesetzt werden sollen. Auf diese Art werden 'lokale Arrays' möglich.

bell (--)
gibt einen Piepston auf der Konsole aus.

blank (addr len --)
Ab addr werden len Bytes mit Leerzeichen überschrieben.

setvec (--)
setzt den Critical Error Handler des Betriebssystems auf eine neue Routine, die die Ausgabe von Fehlerboxen bei Schreib- oder Lesefehlern bei Diskettenzugriffen verhindert. Andere Boxen, die über diesen Handler laufen, erscheinen nach wie vor, z.B. die Aufforderung, bei einem Laufwerk eine andere Diskette einzulegen. Damit entfällt das Geduldsspiel, ohne sichtbare Maus das Abbruchfeld zu finden.

restvec (--)
setzt den Critical Error Handler auf den alten Wert zurück. Diese Routine muß unbedingt ausgeführt werden, bevor man FORTH verläßt, da sonst das System abstürzt.

Allocate

MALLOC und MFREE enthalten die Betriebssystemaufrufe, mit denen zusätzlicher Speicher beim Betriebssystem an- und abgemeldet werden muß.

malloc (d -- laddr)

Es werden d Byte beim Betriebssystem angefordert; laddr ist die Anfangsadresse des reservierten Speicherbereichs. Eine Fehlerbedingung besteht, wenn nicht genug Speicherplatz vorhanden ist. In diesem Fall wird mit der Meldung 'No more RAM' abgebrochen.

mfree (laddr --)

Gibt den reservierten Speicher ab laddr wieder frei. Eine Fehlerbedingung besteht, falls kein Speicher reserviert war. In diesem Fall wird mit der Meldung 'mfree Error' abgebrochen.

Relocate

Hier sind Worte aufgeführt, mit denen sich die Speicheraufteilung des volksFORTH ändern läßt. Von der Größe des Return- und Datenstacks (also der Dictionarygröße) hängt der Platz für die Diskbuffer ab. Wenig Buffer erlauben ein großes Dictionary, bieten aber beim Editieren wenig Komfort, weil ständig auf das Laufwerk zugegriffen werden muß.

relocate (stacklen rstacklen --)
richtet das System neu ein mit stacklen Dictionarygröße und rstacklen Returnstackgröße (Vgl. die Memory Map des volksFORTH im Kapitel über den Multitasker). Abschließend wird COLD ausgeführt, um u.a. die Diskbuffer neu zu initialisieren. Vor RELOCATE sollte daher SAVE aufgerufen werden.

buffers (+n --)
stellt ein System mit +n Diskbuffern her. Es gilt das bei RELOCATE Gesagte.

Strings

Hier befinden sich grundlegende Routinen zur Stringverarbeitung. Vor allem wurden auch Worte aufgenommen, die den Umgang mit den vom Betriebssystem geforderten 0-terminated Strings ermöglichen. FORTH hat hier gegenüber 'C' den Nachteil, daß FORTH-Strings standardmäßig mit einem Count-Byte beginnen, das die Länge des Stings enthält. Ein abschließendes Zeichen (z.B. Null) ist daher unnötig. Da das Betriebssystem aber in 'C' geschrieben wurde, müssen Strings entsprechend umgewandelt werden.

Beachten Sie bitte auch im Glossar die Wortgruppe "Strings".

caps (-- adr)
Adr ist die Adresse einer Variablen, die angibt, ob beim Stringvergleich mit COMPARE auf Groß- und Kleinschreibung geachtet werden soll.

-text (addr0 len addr1 -- n) " minus-text"
addr0 und addr1 sind die Adressen von zwei counted strings, len die Anzahl von Zeichen, die verglichen werden soll. Die Strings werden zeichenweise voneinander subtrahiert. n enthält die Differenz der beiden ersten nicht übereinstimmenden Zeichen. Ist n = 0, so sind die Strings gleich.

compare (addr0 len addr1 -- n)
wie -TEXT, jedoch wird der Inhalt der Variablen CAPS ausgewertet. Ist CAPS wahr, wird nicht auf Groß- oder Kleinschreibung geachtet. Die Vergleichsroutine ist dann allerdings erheblich langsamer.

search (text textlen buf buflen -- offset f)
Im Text ab Adresse text wird in der Länge textlen nach dem String, der im Buffer buf mit der Länge buflen steht, gesucht. Zurückgeliefert wird der Offset in den durchsuchten Text an die Stelle, an der der String gefunden wurde, sowie das Flag f . Ist f wahr, wurde der String gefunden, sonst nicht.

delete (buffer size count --)
Im Buffer der Länge size werden count Zeichen entfernt. Der Rest des Buffers wird 'herangezogen' und das Ende mit Leerzeichen aufgefüllt.

insert (string len buffer size --)
Der String ab Adresse string mit der Länge len wird in den Buffer ab buffer mit der Größe size eingefügt. Der Rest des Buffers wird nach hinten geschoben, Zeichen am Ende fallen weg.

$sum (-- adr) " string-sum"
Adr ist die Adresse einer Variablen, die auf einen String zeigt. An diesen String kann ein weiterer String mit $ADD hinten angefügt werden.

$add (addr len --) " string-add"
hängt den String ab addr mit der Länge len an den String bei $SUM an. Der Count wird dabei addiert.

c>0" (addr --) " counted-to-zero-string"
wandelt den counted String ab addr in einen 0-terminated String um. Die Länge des Strings im Speicher bleibt gleich.

0>c" (addr --) " zero-to-counted-string"
wandelt den 0-terminated String ab addr in einen counted String um. Die Länge des Strings im Speicher bleibt gleich.

,0" (--) " comma-zero-string"
Wird in der folgenden Form benutzt :
,0" ccc"
Legt den String ccc mit führendem Countbyte und abschließender Null im Dictionary beginnend bei HERE ab. Vergleiche ,"

0" (-- adr) " zero-string"
(--) compiling
Wird in der folgenden Form benutzt :
0" ccc"
Liest den Quelltext bis zum nächsten " und legt ihn als counted String mit abschließender Null im Dictionary ab. Zur Laufzeit wird die Adresse des ersten Zeichens des Strings adr auf dem Stack abgelegt. Dieses Wort ist statesmart und kann daher auch im Interpretierenden Zustand verwendet werden.

Abweichungen des volksFORTH83 von Programmieren in Forth

Die Gründe für die zahlreichen Abweichungen des Buches "Starting Forth" (Prentice Hall, 1982) bzw. "Programmieren in Forth" (Hanser, 1984) von Leo Brodie wurden bereits im Prolog aufgeführt. Sie sollen nicht wiederholt werden. Das Referenzbuch ist, trotz offenkundiger Mängel in Übersetzung und Satz, die deutsche Version, da sie erheblich weiter verbreitet sein dürfte. Im folgenden werden nicht nur Abweichungen aufgelistet, sondern auch einige Fehler mit angegeben. Die Liste erhebt keinen Anspruch auf Vollständigkeit. Wir bitten alle Leser, uns weitere Tips und Hinweise mitzuteilen. Selbstverständlich gilt das auch für alle anderen Teile dieses Handbuchs. Vielleicht wird es in der Zukunft ein Programm geben, das es ermöglicht, Programme aus "Starting Forth" direkt, ohne daß man diese Liste im Kopf haben muß, einzugeben.

Kapitel 1 - Grundlagen

7 -) im volksFORTH System wurde statt "Lexikon" konsequent der Begriff "Dictionary" verwendet.

9 -) Statt "?" durckt das volksFORTH "haeh?" , wenn XLERB eingegeben wird.

-) Selbstverständlich akzeptiert das volksFORTH Namen mit bis zu 31 Zeichen Länge.

11 -) Im volksFORTH kann auch der Hochpfeil "^" als Zeichen eines Namens verwendet werden.

-) Fußnote 6 : Der 83-Standard legt fest, daß ." nur innerhalb von Definitionen verwendet werden darf. Außerhalb muß man .(verwenden.

17 -) Das volksFORTH gibt bei Stackleerlauf irgendeine Zahl aus, nicht unbedingt eine Null !

18 -) Druckfehler : Bei dem Beispiel (n --) muß zwischen "(" und "n" ein Leerzeichen stehen. Das gilt auch für viele der folgenden Kommentare.

19 -) Bei dem Wort EMIT muß man natürlich angeben, in welchem Code das Zeichen kodiert ist. Beim volksFORTH auf dem C64 ist das nicht der ASCII-Zeichensatz, sondern der Commodore-spezifische. Am Besten ist es, statt 42 EMIT lieber ASCII * EMIT einzugeben.

Kapitel 2 - Wie man in Forth rechnet

23 -) Druckfehler : Selbstverständlich ist 10 1000 * kleiner als 32767, kann also berechnet werden. Kurios wird es nämlich erst bei 100 1000 * .

31 -) Die Operatoren /MOD und MOD arbeiten laut 83-Standard mit vorzeichenbehafteten Zahlen. Die Definition von Rest und Quotient bei negativen Zahlen findet man unter "Division, floored" in "Definition der Begriffe" oder unter /MOD im Glossar.
Seien Sie bitte bei allen Multiplikations- und Divisionsoperatoren äußerst mißtrauisch und schauen Sie ins Handbuch. Es gibt keinen Bereich von Forth, wo die Abweichungen der Systeme untereinander größer sind als hier.

38 -) .S ist im System als Wort vorhanden. Unser .S druckt nichts aus, wenn der Stack leer ist. Auch die Reihenfolge der Werte ist andersherum, der oberste Wert steht ganz links. Alle Zahlen werden vorzeichenlos gedruckt, d.h. -1 erscheint als 65535 .

-) 'S heißt im volksFORTH SP@ .

-) Das Wort DO funktioniert nach dem 83-Standard anders als im Buch. .S druckt nämlich, falls kein Wert auf dem Stack liegt, 32768 Werte aus. Ersetzt man DO durch ?DO , so funktioniert das Wort .S , aber nur dann, wenn man 2- wegläßt. Nebenbei bemerkt ist es sehr schlechter Stil, den Stack direkt zu addressieren !

39 -) <ROT ist im volksFORTH unter dem Namen -ROT vorhanden.

Kapitel 3 - Der Editor und externe Textspeicherung

Beim volksFORTH auf dem ST wird ein leistungsfähiger Bildschirmeditor mitgeliefert, dessen Bedienung sich von dem im Buch benutzten Zeileneditor stark unterscheidet.

Kapitel 4 - Entscheidungen

75 -) Die Vergleichsoperatoren müssen nach dem 83-Standard -1 auf den Stack legen, wenn die Bedingung wahr ist. Dieser Unterschied zieht sich durch das ganze Buch und wird im folgenden nicht mehr extra erwähnt. Also : -1 ist "wahr" , 0 ist "falsch" . Daher entspricht 0= nicht mehr NOT. NOT invertiert nämlich jedes der 16 Bit einer Zahl, während 0= falsch liefert,

wenn mindestens eins der 16 Bit gesetzt ist. Überall, wo NOT steht, sollten Sie 0= verwenden.

84 -) Das Wort ?STACK im volksFORTH liefert keinen Wert, sondern bricht die Ausführung ab, wenn der Stack über- oder leerläuft. Daher ist das Wort ABORT" überflüssig.

-) Statt -< muß es in der Aufgabenstellung 6 natürlich =< heißen. Für positive Zahlen tut UWITHIN im volksFORTH dasselbe.

Kapitel 5 - Die Philosophie der Festkomma-Arithmetik

89 -) Eine Kopie des obersten Wertes auf dem Returnstack bekommt man beim volksFORTH mit R@ . Die Worte I und J liefern die Indices von DO-Schleifen. Bei einigen Forth-Systemen stimmt der Index mit dem ersten bzw. dritten Element auf dem Returnstack überein. Der dann mögliche und hier dokumentierte Mißbrauch dieser Worte stellt ein Beispiel für schlechten Programmierstil dar. Also: Benutzen sie I und J nur in DO-Schleifen.

91 -) Die angedeutete Umschaltung zwischen Vokabularen geschieht anders als beim volksFORTH.

93 -) Fußnote: Es trifft nicht immer zu, daß die Umsetzung von Fließkommazahlen länger dauert als die von Integerzahlen. Insbesondere dauert die Umsetzung von Zahlen beim volksFORTH länger als z.B. bei verschiedenen BASIC-Interpretern. Der Grund ist darin zu suchen, daß die BASICs nur Zahlen zur Basis 10 ausgeben und daher für dieses Basis optimierte Routinen verwenden können während das volksFORTH Zahlen zu beliebigen Basen verarbeiten kann. Es ist aber durchaus möglich, bei Beschränkung auf die Basis 10 eine erheblich schnellere Zahlenausgabe zu programmieren.

98 -) */MOD im volksFORTH arbeitet mit vorzeichenbehafteten Zahlen. Der Quotient ist nur 16 Bit lang.

Kapitel 6 - Schleifenstrukturen

104 -) Die Graphik auf dieser Seite stellt Implementationsdetails dar, die für das polyFORTH gelten, nicht aber für das volksFORTH. Reißen Sie bitte diese Seite aus dem Buch heraus.

108 -) J liefert zwar den Index der äußeren Schleife, aber nicht den 3. Wert auf dem Returnstack.

110 -) Das Beispiel TEST funktioniert nicht. Beim 83-Standard sind DO und LOOP geändert worden, so daß sie jezt den gesamten Zahlenbereich von 0...65535 durchlaufen können. Eine Schleife n n DO ... LOOP exekutiert also jetzt 65536 - mal und nicht nur einmal, wie es früher war.

111 -) Beim volksFORTH wird beim Eingeben von nichtexistenten Worten nicht der Stack geleert, denn der Textinterpreter führt nicht ABORT" aus, sondern ERROR" . Den Stack leert man durch Eingabe der Worte CLEARSTACK oder ABORT .

114 -) LEAVE wurde im 83-Standard so geändert, daß sofort bei Ausführen von LEAVE die Schleife verlassen wird und nicht erst beim nächsten LOOP. Daher ändert LEAVE auch nicht die obere Grenze.

Kapitel 7 - Zahlen, Zahlen, Zahlen

125 -) Es ist systemabhängig, ob EMIT Steuerzeichen ausgeben kann.

130 -) Seien Sie mißtrauisch ! Das Wort U/MOD heißt im 83-Standard UM/MOD .

-) Das Wort /LOOP ist nun überflüssig geworden, da das normale Wort LOOP bereits alle Zahlen durchlaufen kann.

132 -) Beim volksFORTH werden nur Zahlen, die "," oder "." enthalten, als 32-Bit Zahlen interpretiert. "/" und ":" sind nicht erlaubt.

137 -) Der 83-Standard legt fest, daß SIGN ein negatives Vorzeichen schreibt, wenn der oberste (und nicht der dritte) Wert negativ ist. Ersetzen Sie bitte jedes SIGN durch ROT SIGN , wenn Sie Beispiele aus dem Buch abtippen.

-) Für HOLD gilt dasselbe wie das für EMIT (Abweichung Seite 19) gesagte. Benutzen Sie statt 46 HOLD besser ASCII - HOLD .

140 -) D- DMAX DMIN und DU< sind nicht vorhanden.

141 -) 32-Bit Zahlen können in eine Definition geschrieben werden, indem sie durch einen Punkt gekennzeichnet werden. Die Warnung für experimentierfreudige Leser trifft beim volksFORTH also nicht zu.

-) M+ M/ und M*/ sind nicht vorhanden. Für M/ können Sie M/MOD NIP einsetzen und für M+ etwa EXTEND D+ .

143 -) U* heißt im 83-Standard UM* und U/MOD UM/MOD

149 -) Aufgabe 7 ist großer Quark ! Die Tatsache, daß viele Forth-Systeme .. als doppelt genaue Null interpretieren, bedeutet nicht, daß es sich um keinen Fehler der Zahlenkonversion handelt. Das volksFORTH toleriert solche Fehleingaben nicht.

Kapitel 9 - Forth intern

178 -) Zu den Fußnoten 1,2 : Der 83-Standard schreibt für das Wort FIND ein anderes Verhalten vor, als hier angegeben wird. Insbesondere sucht FIND nicht nach dem nächsten Wort im Eingabestrom, sondern nimmt als Parameter die Adresse einer Zeichenkette (counted String), nach der gesucht werden soll. Auch die auf dem Stack abgelegten Werte sind anders vorgeschrieben.

-) Das Beispiel 38 ' GRENZE ! ist schlechter Forth Stil; es ist verpönt, Konstanten zu ändern. Da das Wort ' nach dem 83-Standard die Kompilationsadresse des folgenden Wortes liefert (und nicht die Parameterfeldadresse), der Wert einer Konstanten aber häufig in ihrem Parameterfeld aufbewahrt wird, funktioniert das Beispiel auf vielen Forth Systemen, die dem 83-Standard entsprechen, folgendermaßen: 38 ' GRENZE >BODY ! .

179 -) Fußnote 3 : Die Fußnote trifft für den 83-Standard nicht zu. ' verhält sich wie im Text beschrieben.

180 -) Das volksFORTH erkennt 32-Bit Zahlen bereits seriemäßig, daher gibt es kein Wort 'NUMBER . Wollen Sie

eigene Zahlenformate erkennen (etwa Floating Point Zahlen) , so können Sie dafür das deferred Wort NOTFOUND benutzen.

181 -) Die vorgestellte Struktur eines Lexikoneintrags ist nur häufig anzutreffen, aber weder vorgeschrieben noch zwingend. Zum Beispiel gibt es Systeme, die keinen Code Pointer aufweisen. Das volksFORTH sieht jedoch ähnlich wie das hier vorgestellte polyFORTH aus.

188 -) Druckfehler : Statt ABORT' muß es ABORT" heißen.

189 -) Der 83-Standard schreibt nicht vor, daß EXIT die Interpretation eines Disk-Blockes beendet. Es funktioniert zwar auch beim volksFORTH, aber Sie benutzen besser das Wort \\ .

190 -) Die Forth Geographie gilt für das volksFORTH nicht; einige der Abweichungen sind :

-) Die Variable H heißt im volksFORTH DP (=Dictionary Pointer) .

-) Der Eingabepuffer befindet sich nicht bei S0 , sondern TIB liefert dessen Adresse.

-) Der Bereich der Benutzervariablen liegt woanders.

191 -) Zusätzliche Definitionen : Beim volksFORTH ist die Bibliothek anders organisiert. Ein Inhaltsverzeichnis der Disketten können Sie sich mit FILES ausgeben lassen.

197 -) Der Multitasker beim volksFORTH ist gegenüber dem des polyFORTH vereinfacht. So besitzen alle Terminal-Einheiten (wir nennen sie Tasks) gemeinsam nur ein Lexikon und einen Eingabepuffer. Es darf daher nur der OPERATOR (wir nennen in Main- oder Konsolen-Task) kompilieren.

198 -) Im volksFORTH sind SCR R# CONTEXT CURRENT >IN und BLK keine Benutzervariablen.

200 -) Das über Vokabulare gesagte trifft auch für das volksFORTH zu, genaueres finden Sie unter Vokabularstruktur im Handbuch.

202 -) LOCATE heißt im volksFORTH VIEW .

203 -) EXECUTE benötigt nach dem 83-Standard die Kompilationsadresse und nicht mehr die Parameterfeldadresse eines Wortes. Im Zusammenhang mit ' stört das nicht, da auch ' geändert wurde.

Kapitel 10 - Ein- und Ausgabeoperationen

211 -) Die angesprochene Funktion von FLUSH führt nach dem 83-Standard das Wort SAVE-BUFFERS aus. Es schreibt alle geänderten Puffer auf die Diskette zurück. Das Wort FLUSH existiert ebenfalls. Es unterscheidet sich von SAVE-BUFFERS dadurch, daß es nach dem zurückschreiben alle Puffer löscht. Die Definition ist einfach :

```
: flush  save-buffers  empty-buffers ;
```

FLUSH wird benutzt, wenn man die Diskette wechseln will, damit von BLOCK auch wirklich der Inhalt dieser Diskette geliefert wird.

-) Fußnote 4 trifft für das volksFORTH nicht zu.

212 -) Der 83-Standard schreibt vor, daß BUFFER sehr wohl darauf achtet, ob ein Puffer für diesen Block bereits existiert. BUFFER verhält sich genauso wie BLOCK , mit dem einzigen Unterschied, daß das evtl. erforderliche Lesen des Blocks von der Diskette entfällt.

213 -) Wie schon erwähnt, muß bei den Beispielen auf dieser Seite S0 @ durch TIB ersetzt werden. Ebenso muß ' TEST durch ' TEST >BODY ersetzt werden. Das gilt auch für das folgende Beispiel BEZEICHNUNG .

217 -) Druckfehler : WORT ist durch QUATSCH zu ersetzen.

219 -) <CMOVE heißt nach dem 83-Standard CMOVE> . Der Pfeil soll dabei andeuten, daß man das Wort benutzt, um im Speicher vorwärts zu kopieren. Vorher war gemeint, daß das Wort am hinteren Ende anfängt.

-) Für MOVE wird im 83-Standard ein anderes Verhalten vorgeschlagen. MOVE wählt zwischen CMOVE und CMOVE> , je nachdem , in welche Richtung verschoben wird.

223 -) Fußnote 10 : QUERY ist auch im 83-Standard vorgeschrieben.

-) WORD kann, muß aber nicht seine Zeichenkette bei HERE ablegen.

224 -) Nach dem 83-Standard darf EXPECT dem Text keine Null mehr anfügen.

229 -) Fußnote 14 : PTR heißt beim volksFORTH DPL (= decimal point location) .

230 -) Ersetzen sie WITHIN durch UWITHIN oder ?INNERHALB. NOT muß durch 0= ersetzt werden, da die beiden Worte nicht mehr identisch sind.

232 -) Druckfehler in Fußnote 15 : Der Name des Wortes ist natürlich -TEXT und nicht TEXT . Außerdem müssen die ersten beiden "/" durch "@" ersetzt werden. Das dritte "/" ist richtig.

Kapitel 11 - Wie man Compiler erweitert...

247 -) BEGIN DO usw. sehen im volksFORTH anders aus, damit mehr Fehler erkannt werden können.

248 -) Fußnote 2 : Die Erläuterungen beziehen sich auf polyFORTH, im volksFORTH siehts wieder mal anders aus. Die Frage ist wohl nicht unberechtigt, warum in einem Lehrbuch solche implementationsabhängigen Details vorgeführt werden.

-) Fußnote 3 : Eine Konstante im volksFORTH kommt, falls sie namenlos (siehe Kapitel "Der Heap" im Handbuch) definiert wurde, mit 2 Zellen aus.

249 -) Die zweite Definition von 4DAZU funktioniert beim volksFORTH nicht, da das Wort : dem ; während der Kompilation Werte auf dem Stack übergibt. Das ist durch den 83-Standard erlaubt.

250 -) Druckfehler : Statt [SAG-HALLO] muß es [SAG-HALLO] heißen.

251 -) Die Beispiele für LITERAL auf dieser Seite funktionieren, da LITERAL standardgemäß benutzt wird.

-) Druckfehler : Statt ICH SELBST muß es ICH-SELBST

heißen.

252 -) Bei Definitionen, die länger als eine Zeile sind, druckt das volksFORTH am Ende jeder Zeile "compiling" statt "ok" aus.

255 -) Die Beispiele für INTERPRET und] entstammen dem polyFORTH. Eine andere Lösung wurde im volksFORTH verwirklicht. Hier gibt es zwei Routinen, je eine für den interpretierenden und kompilierenden Zustand. INTERPRET führt diese Routinen vektoriell aus.

-) Fußnote 4 trifft für das volksFORTH nicht zu.

Kapitel 12 - Drei Beispiele

270 -) Druckfehler : In WÖRTER ist ein 2DROP zuviel; ferner sollte >- >IN sein.

-) In BUZZ muß es statt WORT .WORT heißen. (Reingefallen : Es muß WÖRTER sein!)

239 -) Die Variable SEED muß wohl SAAT heißen. Ob der Übersetzer jemals dieses Programm kompiliert hat?

-) Nebenbei: Im amerikanischen Original hatten die Worte NÄCHSTES WÖRTER FÜLLWÖRTER und EINLEITUNG noch einen Stackkommentar, ohne die das Programm unverständlich wird, besonders bei diesem fürchterlichen Satz. Also, wenn Sie an Ihren Fähigkeiten zweifeln, sollten Sie sich das amerikanische Original besorgen.

Abweichungen des volksFORTH83 von 'FORTH TOOLS' von A. Anderson & M. Tracy

p.15 CLEAR macht im volksFORTH83 etwas anderes als in FORTH TOOLS. Benutze statt CLEAR das Wort CLEARSTACK oder definiere
```
' clearstack Alias clear
```

p.27 .S druckt die Werte auf dem Stack anders herum aus, ausserdem fehlt der Text "Stack:"

p.34 2ROT fehlen. Benutze
```
: 2rot    5 roll 5 roll ;
```

p.42 Statt DIRECTORY heißt es FILES

p.45 THRU druckt keine Punkte aus.

p.46 Der Editor funktioniert anders.

p.64 volksFORTH83 enthaelt kein Wort ? . Benutze
```
: ?   @ . ;
```

p.99 AGAIN gibt es nicht. Benutze stattdessen REPEAT oder definiere
```
' REPEAT  Alias  AGAIN  immediate restrict
```

p.103 Benutze
```
' extend Alias s>d
```

p.107 DU< fehlt.

p.116 SPAN enthaelt 6 Zeichen, da "SPAN ?" genau 6 Zeichen lang ist. Das System benutzt naemlich ebenfalls EXPECT. Daher geht das Beispiel auf Seite 117 auch nicht. Damit es geht, muss man alle Worte in einer Definition zusammenfassen.

p.118 CPACK entspricht dem Wort PLACE.

p.125 Benutze
```
: String   Create dup , 0 c, allot  Does> 1+ count ;
```

p.126 " kann nicht interpretiert werden. Zwei Gänsefüßchen hintereinander sind ebenfalls nicht erlaubt.

p.141 Das Wort IS aus dem volksFORTH83 kann innerhalb von Definitionen benutzt werden.

p.146 Es gibt keine Variable WIDTH , da bei Namen alle Zeichen gültig sind.

p.149 Benutze
```
: >link      >name 2- ;
: link>      2+ name> ;
: n>link     2- ;
: l>name     2+ ;
```

p.166 STOP entspricht \\

p.167 Bei FIND kann das Flag auch die Werte -2 oder +2 annehmen.

p.184 ORDER ist nicht in ONLY , sondern in FORTH enthalten. Außerdem druckt es auch nicht "Context:" oder "Current:" aus. Current wird stattdessen einfach zwei Leerzeichen hinter Context ausgegeben.

Fehlermeldungen des volksFORTH83

Das volksFORTH83 gibt verschiedene Fehlermeldungen und Warnungen aus. Zum Teil wird dabei das zuletzt eingegebene Wort wiederholt. Diese Meldungen sind im folgenden nach Gruppen getrennt, aufgelistet:

Kernel

?
Der eingegebene String konnte nicht mit NUMBER in eine Zahl umgewandelt werden. Beachten Sie den Inhalt von BASE.

Adress Error !
Dies ist ein fataler Fehler. Der Prozessor hat versucht, mit einer Wortoperation auf eine ungerade Adresse zuzugreifen. Sollten Sie keinen fehlerhaften Maschinencode geschrieben haben, so ist das Forth "zerschossen".

beyond capacity
Der angesprochene Block ist physikalisch nicht vorhanden. Beachten Sie den Inhalt von OFFSET.

Bus Error !
Der Prozessor hat versucht, auf einen nichtexistenten Speicher zuzugreifen. Prüfen Sie bitte alle verwendeten Langwortoperationen.

compile only
Das eingegebene Wort darf nur innerhalb von :-Definitionen verwendet werden.

crash
Es wurde ein deferred Wort aufgerufen, dem noch kein auszuführendes Wort mit IS zugewiesen wurde.

Dictionary full
Der Speicher für das Dictionary ist erschöpft. Sie müssen die Speicherverteilung ändern oder Worte mit FORGET vergessen.

Division by 0 !
Es wurde versucht, durch Null zu teilen.

division overflow
Die Division zweier Zahlen wurde abgebrochen, da das Ergebnis nicht im Zahlenbereich darstellbar ist.

exists
Das zuletzt definierte Wort ist im Definitons-Vokabular schon vorhanden. Dies ist kein Fehler, sondern nur ein Hinweis!

haeh?
Das eingegebene Wort konnte weder im Dictionary gefunden noch als Zahl interpretiert werden.

Illegal Instruction !
Dies ist ein fataler Fehler. Der Prozessor hat versucht, einen nicht erlaubten Befehl auszuführen. Sollten Sie Maschinencode geschrieben haben, so prüfen Sie ihn bitte auf unzulässige Kombinationen von Befehl und Adressierungsart. Sind Sie sich keiner Schuld bewußt, so ist das Forth zerstört.

invalid name
Der Name des definierten Wortes ist zu lang (mehr als 31 Zeichen) oder zu kurz (Der Name sollte dem definierenden Wort unmittelbar folgen).

is symbol
Das Wort, das mit FORGET vergessen werden sollte, befindet sich auf dem Heap. Benutzen Sie dafür CLEAR.

nein
Der Bereich von Blöcken, der mit CONVEY kopiert werden sollte, ist leer oder viel zu groß.

no file
Auf Ihrem System sind keine Files benutzbar. Die Variable FILE muß daher den Wert Null haben. Siehe auch die Fehlermeldungen des Fileinterfaces

not deferred
Es wurde versucht, mit IS einem Wort, das nicht deferred ist, eine auszuführende Prozedur zuzuweisen.

not enough parameters
Ein Wort erwartete mehr Werte auf dem Stack, als vorhanden waren. Dieser Fehler wird bei Ausführung des Wortes ARGUMENTS erzeugt.

protected
Es wurde versucht, ein Wort zu vergessen, das mit SAVE geschützt wurde. Benutzen Sie die Phrase:
```
' <name> >name 4 - (forget  save
```

read error !
Bei einem Lesezugriff auf die Diskette trat ein Fehler auf. Der zu lesende Buffer wurde nicht markiert.

stack empty
Es wurden mehr Werte vom Stack genommen als vorhanden waren.

tight stack
Es befanden sich zuviele Werte auf dem Stack, sodaß der Speicher erschöpft war. Legen Sie weniger Werte auf den Stack, oder sparen Sie Speicherplatz im Dictionary.

unstructured
Kontrollstrukturen wurden falsch verschachtelt oder weggelassen. Diese Fehlermeldung wird auch ausgegeben, wenn sich die Zahl der Werte während der Kompilation einer :- Definition zwischen : und ; geändert hat.

Userarea full
Es wurden mehr als 124 Uservariablen definiert. Das ist nicht zulässig.

Vocabulary stack full
Es wurden zuviele Vokabulare mit ALSO in den festen Teil der Suchreihenfolge übernommen. Benutzen Sie ONLY oder TOSS, um Vokabulare zu entfernen.

write error !
Siehe " read error ..."

Fileinterface

close error
Das mit SAVESYSTEM erzeugte File konnte nicht ordnungsgemäß geschlossen werden.

Datei nicht gefunden
Eine nach USE oder anderen Worten angegebene Datei konnte nicht auf dem Massenspeicher gefunden werden. Prüfen Sie bitte, ob PATH korrekt gesetzt ist.

Disk schreibgeschützt
Es wurde versucht, auf eine schreibgeschützte Diskette zu schreiben. Prüfen Sie bitte, ob ein Block fälschlich als UPDATEd markiert wurde.

Disk voll
Die Diskette ist voll, daher kann MORE nicht mehr ausgeführt werden.

Dos-Error #00
Es trat ein Fehler im TOS auf. Die Nummer hat folgende Bedeutung :

Bios	01	Error
	02	Drive not ready
	03	Unknown command
	04	CRC Error
	05	Bad request
	06	Seek error
	07	Unknown media
	08	Sector not found
	09	Out of paper
	10	Write fault
	11	Read fault
	14	Media change detected
	15	Unknown device
	16	Bad sectors on format
	17	Insert other disk (request)
GEM-Dos	32	Invalid function number
	35	Handle pool exhausted
	36	Acess denied
	40	Invalid memory block adress

46 Invalid drive specification
47 No more files
64 Range error
65 GEMDOS internal error
66 Invalid executable file format
67 Memory block growth failure

Wenn Sie mit diesen Fehlerbeschreibungen nicht viel anfangen können, so trösten Sie sich: wir können es auch nicht. Es ist auch nicht erforderlich; erscheinen doch die Fehlernummern ziemlich unmotiviert und selten fehlerspezifisch, oft liefern sie keinerlei Hinweis auf die Art des auftretenden Fehlers.

missing filename
Auf das Wort SAVESYSTEM muß ein Filename folgen, der aber nicht vorhanden war.

no device
SAVESYSTEM konnte nicht das gewünschte File erzeugen. Prüfen Sie bitte, ob die Diskette vorhanden, fehlerfrei und nicht schreibgeschützt ist.

No file !
Es wurde versucht, die Diskette im Directmodus mit MORE zu verlängern oder ihr mit ASSIGN ein File zuzuweisen. Prüfen Sie bitte mit FILE? das aktuelle File.

Pfad nicht gefunden
Der mit DIR angegebene Pfad existiert nicht. Überzeugen Sie sich bitte, daß die richtige Diskette im Laufwerk steckt.

string too long
Der nach ASSIGN, USE, MAKEFILE, LOADFROM, MAKE oder INCLUDE angegebene Filename ist zu lang. Entfernen sie bitte Subdirectories aus dem Pfadnamen, und setzen Sie DIR entsprechend.

Ungültige Handle#
Es liegt ein Fehler bei der Verwaltung der Files vor. Das Forth versucht, mit Hilfe einer durch das TOS vergebenen Zahl, der "Handle", auf ein File zuzugreifen, das bereits wieder geschlossen worden ist. Versuchen Sie, von Hand die Handle in der Liste der FCBs zu löschen.

Ungültiges Laufwerk
Das durch SETDRIVE bzw. A: B: C: oder D: angegebene Laufwerk existiert nicht.

write error
SAVESYSTEM konnte nicht das gesamte System speichern. Prüfen Sie bitte, ob die Diskette voll ist.

Wrong range
CONVEY kann die angegebenen Blöcke nicht kopieren, da sie nicht existieren. Prüfen Sie bitte, ob das File, in das kopiert werden soll, ausreichend lang ist.

Zugriff nicht möglich
Es wurde versucht, auf einen Block zuzugreifen, der nicht vorhanden ist. Überprüfen Sie bitte den Inhalt von OFFSET sowie das File, auf das zugegriffen werden sollte.

ALLOCATION

No more RAM
Der Arbeitsspeicher des ST ist aufgebraucht, so daß kein Speicher mehr angefordert werden kann.

mfree Error!
Der Speicherbereich, der mit MFREE freigegeben werden sollte, wurde nicht allokiert. Prüfen Sie bitte, ob MFREE zweimal aufgerufen wurde oder ob sich die Anfangsadresse, die sie bei MALLOC erhalten haben und die Adresse, die sie MFREE mitgeben, übereinstimmen.

GEM-Library

Menu-Error
Object-Error
Graphic-Error
File Error
Window-Error
Resource-Error
Eine Routine der entsprechenden AES-Bibliothek signalisierte einen Fehler.

RELOCATION

a ticket to the moon with no return ...
Der Speicherplatz, der nach Ausführung von RELOCATE für den Returnstack übrig bliebe, ist zu klein.

cuts the dictionary
Der Speicherplatz, der nach Ausführung von BUFFERS oder RELOCATE für das Dictionary übrig bliebe, ist zu klein.

kills all buffers
Bei Ausführung würde RELOCATE keinen Blockbuffer im System lassen. Prüfen Sie die Argumente von RELOCATE. Die Anweisung 0 BUFFERS ist ebenfalls nicht zulässig.

Targetcompiler-Worte

Das volksFORTH83-System wurde mit einem sog. Targetcopiler erzeugt. Dieser Targetcompiler compiliert ähnlichen Quelltext wie das volksFORTH83-System. Es gibt jedoch Unterschiede. Insbesondere sind im Quelltext des volksFORTH83 Worte enthalten, die der Steuerung des Targetcompilers dienen, also nicht im Quelltext definiert werden. Wenn Sie sich also über eine Stelle des Quelltextes sehr wundern, sollten Sie in der folgenden Liste die fragwürdigen Worte suchen.

Eine andere Besonderheit betrifft Uservariablen. Wenn im Quelltext der Ausdruck [<name>] LITERAL auftaucht und <name> eine Uservariable ist, so wird bei der späteren Ausführung des Ausdrucks NICHT die Adresse der Uservariablen in der Userarea sondern die Adresse in dem Speicherbereich, in dem sich die Kaltstartwerte der Uservariablen befinden, auf den Stack gelegt.
Wenn die Kaltstartwerte von Uservariablen benötigt werden, wird dieser Ausdruck benutzt.

Folgende, im Quelltext oft auftretende Worte sind im Targetcompiler definiert:

displace Target here! .unresolved origin! Compiler H T there Tnext-link Onlypatch Host Tudp Tvoc-link Tnext-link move-threads