SNAUT

EIN FORTH83 COMPILER VOM COMPY-SHOP

S N A U T

E i n F o r t h - 8 3 C o m p i l e r

- H a n d b u c h -

H. Köhler

SNAUT HANDBUCH
(c) 1986 Compy-Shop Ohg, 4330 Mülheim Ruhr
Alle Rechte vorbehalten

Druck: DER DRUCKER, BOCHUM
Printed in Germany

INHALTSVERZEICHNIS

Einleitung

Forth, ein compilierender Interpreter, von Charles Moore Ende der 60er Jahre entwickelt, läßt sich am besten durch folgende Merkmale beschreiben :

Interaktivität - ermöglicht das komfortable und schnelle Entwickeln und Testen von Programmen.

Erweiterbarkeit - der vorhandene Wortschatz kann (und soll) durch eigene Worte erweitert werden. Es ist auch ohne weiteres möglich, neue Compilerkontrollstrukturen hinzuzufügen. Das System 'wächst' demnach mit den Fähigkeiten des Programmierers.

Portabilität - Forth ist für alle bekannten Prozessoren eingerichtet worden. Programme können daher, wenn sie keine hardwarespezifischen Eigenarten unterstützen, auf andere Systeme übertragen werden.

Strukturiertes Programmieren - durch bedingte Verzweigungen und Programmschleifen, wie IF ELSE THEN, DO LOOP, DO +LOOP, BEGIN UNTIL, BEGIN WHILE REPEAT, BEGIN AGAIN ... Es gibt keine GOTO's, keine Zeilennummern.

Schnelligkeit - in der Programmausführung und der Entwicklung. Durch das gleichzeitige Vorhandensein von Interpreter, Compiler, Assembler, Editor und dem Forth-eigenen Betriebssystem entfällt der bei anderen Sprachen übliche ECLG-Zyklus, (Edit, Compile, Link und Go).

Über diese Forthversion

Im Hinblick auf die Übertragbarkeit von Anwendersoftware wurden durch die European Forth User's Group (EFUG) und durch das Forth Standards Team mehrere 'Standards' herausgegeben, den 77-, 79- und als letzten den 83-Standard. Dieser Compiler entspricht voll dem 83er Standard, zusätzlich wurden ein Assembler, zwei Editoren , I/O Worte inclusive Grafik und Sound, ein 80-Zeichen-pro-Zeile-Interface, ein spezielles Tape-Vokabular und diverse andere Utilities implementiert.
Dieses Handbuch enthalt unter anderem eine Systembeschreibung, Erläuterungen zu den einzelnen Vokabularen, Vorschlage zur Erstellung von Arbeitsversionen und zur Speicherverwaltung, Unterschiede zwischen dem 83-Standard und älteren Sprachversionen und ein Glossary, also eine Kurzbeschreibung der Worte des Sprachkerns. Es ist in erster Linie als Möglichkeit zum Nachschlagen gedacht und weniger als ein Einführungskurs für Starter, dafür gibt es sehr gute, inzwischen auch deutschsprachige, Literatur (siehe Anhang).

Booten des Programms:

Diskettenversion:

Schalten Sie die Stromversorgung von Laufwerk 1 an. Stecken Sie dann die Diskette (Vorderseite oben) in Laufwerk 1. Vergewissern Sie sich, daß keine Programmodule im Modulschacht stecken. Schalten Sie den Computer ein und warten Sie, bis das Programm eingelesen ist und sich meldet. Bei den XL- und XE-Computern muß während des Einschaltvorgangs die OPTION-Taste gedrückt werden, um das eingebaute Basic auszuschalten.

Kassettenversion:

Spulen Sie die Kassette bis zum Anfang zurück. Achten Sie darauf, daß keine Module im Schacht stecken. Drücken Sie beim 400 oder 800er während des Einschaltens des Computers die START-Taste, beim XL oder XE zusätzlich die OPTION-Taste. Der Computer gibt dann einen Ton von sich. Drücken Sie jetzt die PLAY-Taste am Recorder und irgendeine Taste am Computer. Das Programm wird jetzt vom Recorder eingelesen und gestartet.

Was ist Forth ?

Forth ist interaktiv, nimmt also Eingaben von der Tastatur oder einem anderen Eingabemedium (Slang: Inputstream) entgegen und führt sie nach dem Drücken der Returntaste aus.

Forth ist ein Compiler, die Forthbefehle : und ; öffnen bzw.beschließen die Compilation und definieren so neue 'Worte', also neue Forthbefehle.

Forth benutzt zwei Stacks, den **Parameterstack** für die schnelle Zwischenspeicherung von Zahlen, Parametern, lokalen Variablen, und den **Returnstack** für Rücksprungadressen und Indexgrenzen von Programmschleifen. Der Anwender hat vollen Zugriff auf beide Stacks !.

Forth arbeitet als stackorientierte Sprache mit 'umgekehrt polnischer Notation', beispielsweise wird eine Addition interaktiv folgendermaßen ausgeführt: 10 20 + <RET>
Ergebnisausdruck mit . <RET> ergibt 30.
Rechenoperatoren wie '+' greifen also auf den Parameterstack zu, addieren in diesen Fall die oberen beiden Stackwerte, zerstören sie und legen das Resultat wieder auf den Stack.

Forth bietet die Möglichkeit, mit einfach genauen oder doppelt genauen Zahlen zu rechnen.

Forthprogramme werden aus dem vorhandenen Wortkern per Definition (: und ;) zusammengestellt, kompilierte Worte werden wieder von neuen Worten benutzt, bis das Endprogramm durch das oberste Wort aufgerufen wird.

Laufzeitintensive Programmteile lassen sich mit dem Assembler formulieren.

Der Quelltext wird mit Hilfe eines Editors editiert. Dieser Editor arbeitet Screenorientiert, bearbeitet also ganze Informationsblöcke, im Gegensatz zu zeilenorientierten Editoren, die in manchen Forthsystemen installiert sind.

Editor und Assembler sind, wie der Forthkern, eigenständige Vokabulare. Das Vokabularkonzept ist ein sehr wichtiges Thema innerhalb von Forth und wird an anderer Stelle noch ausführlich behandelt.

Kernal und Arbeitsversion (Disk)

Wenn sie die Disk bzw. die Kassette gebootet haben, meldet sich der Compiler in einem relativ komprimierten 'jungfräulichen' Zustand. Das hat den Sinn, daß man sich, aufbauend auf diesem 'Kernal', seine eigene Arbeitsversion zusammenstellen bzw. ein sich selbst startendes Anwenderprogramm erzeugen kann. Es folgen Vorschläge für Arbeitsversionen, die in dieser Form und Kombination sinnvoll sind.

Folgende Möglichkeiten sind denkbar:

Vorschlag Arbeitsversion 1 (Disk):

- 512 bytes/block
- Utilities
- Assembler
- IO und Grafikworte
- Editor 1
- bei Bedarf TAPE

1. Kernal booten.

2. Den 'FULL LOAD' durchführen (Startscreen steht im Directory. Das Directory sieht man durch 178 LIST 179 LIST).

3. Die gewünschten Utilities bzw. Vokabulare dazu'load'en.

4. MAKEDBOOT 'load'en, neu formatierte Diskette in Laufwerk 1 einlegen (VORSICHT, nicht aus Versehen die Originaldiskette verwenden !) MAKEDBOOT eintippen und die RETURN-Taste drücken. Auf die Diskette wird nun die neue Boot-Arbeitsversion geschrieben.

5. Irgendeine Taste drücken. Die neue Bootdisk wird nun geladen.

Vorschlag Arbeitsversion 2 (Disk):

- 1024 bytes/block
- Utilities
- Assembler
- IO und Grafikworte
- 80Z und Editor 2 (unbedingt zusammen laden, Editor 2 benutzt das 80-Zeichen-Vokabular).
- bei Bedarf TAPE

1. Kernal booten.

2. Buffer relozieren !
Folgendes eingeben:
FIRST HEX 7000 RELOC DP ! DECIMAL <RET>

3. Den 'FULL LOAD' durchführen (Startscreen steht im Directory. Das Directory sieht man durch 178 LIST 179 LIST).

4. Die gewünschten Utilities bzw. Vokabulare dazucompilieren.

5. EXPAND-BUFFERS 'load'en. EXPAND-BUFFERS eingeben, RETURN-Taste drücken.

6. Die Buffer ans Ende des Wörterbuches verschieben:
HERE RELOC <RET>

7. Den Dictionarypointer auf das Ende der Buffer justieren:
LIMIT IS DP <RET>

8. MAKEDBOOT Version 2 'load'en (Screen 20), neu formatierte Diskette in Laufwerk 1 einlegen (VORSICHT, nicht aus Versehen die Originaldiskette verwenden !) MAKEDBOOT eintippen und die RETURN-Taste drücken. Auf die Diskette wird nun die neue Boot-Arbeitsversion geschrieben.

9. Irgendeine Taste drücken. Die neue Arbeitsversion wird nun gebootet.

Kernal und Arbeitsversion (Kassette)

Bevor Sie sich Ihre Arbeitsversion zusammenstellen, lesen Sie bitte unbedingt den Abschnitt 'Anmerkungen über das Kassettensystem'.

Folgende Arbeitsversionen sind denkbar:

Vorschlag Arbeitsversion 1 (Kassette):

- 512 bytes/block
- Utilities
- Assembler
- Grafik (benötigt Assembler)
- Editor 1

1. Kernal booten.

2. Das Kassettensystem enthält nach dem Booten schon fast alle wichtigen Funktionen. Laden Sie den ersten Kassettenfile (TAPE CLOAD). 'LOAD'en Sie mindestens DEBUG (2 4 THRU <RET>) und die CASE Statements (10 LOAD), bei Bedarf noch eine der CASES: Versionen (5 LOAD oder 6 LOAD).

3. Die gewünschten Utilities bzw. Vokabulare (Assembler, Grafik, Editor 1) dazu'load'en.

4. Die Kassette auf den Anfangspunkt des ersten Datenfiles zurückspulen, den ersten File laden (TAPE CLOAD). MAKECBOOT 'LOAD'en (7 LOAD). Dann eine neue Kassette einlegen, MAKECBOOT eintippen und die RETURN-Taste drücken. Auf die Kassette wird nun die neue Boot-Arbeitsversion geschrieben.

5. Wenn das Schreiben der Bootversion beendet ist, erscheint auf dem Bildschirm die Meldung 'Now boot cassette !'. Spulen Sie nun die Kassette bis auf den Startpunkt zurück und drücken sie eine Taste. Der Computer führt nun einen Kaltstart aus. Vergessen Sie beim Kassettenboot das Drücken der START-Taste nicht (beim XE, XL zusätzlich die OPTION-Taste). Die neue Kassettenversion wird nun geladen.

Vorschlag Arbeitsversion 2 (Kassette):

- 1024 bytes/block
- Utilities
- Assembler
- Grafik (benötigt Assembler)
- BOZ und Editor 2 (beides laden, denn Editor 2 benutzt das 80-Zeichen-Vokabular).

1. Kernal booten.

2. Buffer verschieben !
Folgendes eingeben:
FIRST HEX 7000 RELOC IS DP <RET>

3. Laden Sie den ersten Kassettenfile (TAPE CLOAD). 'LOAD'en Sie DEBUG (2 4 THRU <RET>), die CASE Statements und EXPAND-BUFFERS (10 11 THRU), bei Bedarf noch eine der CASES: Versionen (5 LOAD oder 6 LOAD).

4. Die gewünschten Utilities bzw. Vokabulare (siehe oben) aus den entsprechenden Kassettenfiles dazu'load'en. Nicht den ersten Editor (File 3) verwenden.

5. Wenn sie alle benötigten Worte und Vokabulare kompiliert haben, stellen sie das System um:
Die Buffer wieder ans Ende des Wörterbuches verschieben:
HERE RELOC <RET>
Auf die neue Buffergröße umstellen:
EXPAND-BUFFERS <RET>
Den Dictionarypointer auf das Ende der Buffer justieren:
LIMIT IS DP <RET>
Ramdisk zurückstellen:
TAPEID <RET>

6. Nun wird die neue Version auf Kassette geschrieben. Zuerst BOZ INIT <RET> eingeben. Der auf den Editor 2 folgende Kassettenfile enthält im 1024-Byte-Format MAKECBOOT. Also diesen File laden (TAPE CLOAD), und MAKECBOOT 'load'en (1 LOAD). Die neue kassette einlegen, MAKECBOOT eintippen und die RETURN-Taste drücken. Auf die Kassette wird nun die neue Boot-Arbeitsversion geschrieben.

7. Wenn auf dem Bildschirm die Meldung 'Now boot cassette !' erscheint, spulen Sie die Kassette bis auf den Startpunkt zurück, drücken Sie dann eine beliebige Taste. Der Computer führt nun eine Kaltstart aus. Das Drücken der START-Taste (und beim XE, XL gleichzeitig der OPTION-Taste) nicht vergessen. Die neue Kassettenversion wird nun geladen.

Editor 1 (512 bytes/screen)

Dieser Editor ist relativ kurz, ermöglicht aber trotzdem relativ komfortables Editieren der Screens. Der Editor wird als ein eigenständiges Vokabular mit dem Namen EDITOR geladen, seine Worte werden also nur gefunden, wenn EDITOR in der Suchordnung der Vokabulare enthalten ist (siehe Abschnitt Vokabular und Suchordnung).
Der Aufruf des Editors kann auf zwei Arten erfolgen. Es soll beispielsweise der Screen # 100 bearbeitet werden.

Entweder:

```
EDITOR 100 L <RET>
```

(ruft den Editor auf, lädt Screen # 100 von der (Ram-) Disk und geht in den Editiermodus).

Oder:

```
KL <RET>
```

KL ist als Forthwort definiert, der Editor braucht also nicht in der Suchordnung vorhanden zu sein. KL ruft den Editor auf, (d.h.: stellt die Suchordnung folgendermaßen auf: EDITOR FORTH ONLY), lädt dann den zuletzt benutzten Screen, dessen Nummer in der Variablen SCR enthalten war, und geht in den Editiermodus.

Ist man im Editiermodus angelangt, hat man zuerst die Möglichkeit, mit den Tasten + (Pfeil links) oder * (Pfeil rechts) sich benachbarte Screens anzusehen bzw. auszuwählen, oder nach dem Drücken der Taste - (Pfeil oben) den gerade angezeigten Screen zu editieren.

Der Editor bietet jetzt folgende Funktionen:

(^ = Controltaste)

```
^ + ------------ Cursor links
^ * ------------ Cursor rechts
^ - ------------ Cursor nach oben
^ = ------------ Cursor nach unten
TAB ------------ Tabulatursprung
^ > ------------ Leerzeichen in Zeile einfügen
^ delete/back s - Zeile um ein Zeichen zusammenschieben
delete/back s --- Backspace
^ < ------------ Home ( Cursor nach links oben )
ESC ------------  Einmaliges Betätigen ermöglicht einem wieder
```

das Wechseln in benachbarte Screens (s.o.), durch zweimaliges Drücken verläßt man den Editor ganz.

Das Betätigen anderer Tasten läßt das entsprechende Zeichen auf dem Bildschirm erscheinen und schreibt es in den Buffer. Grafikzeichen werden nicht akzeptiert.

Wollen Sie die Weite des Tabulatursprungs ändern, ändern Sie die EDITOR-Variable #TAB, z.B.: 7 #TAB !. Die Voreinstellung ist hier 5.

Im Editor sind zwei weitere wichtige Worte enthalten:

COPY (n1 n2 ---) erwartet zwei Zahlen (n1 n2) auf dem Stack. Der Screen mit der Nummer n1 wird in den Screen mit der Nummer n2 kopiert.
CLEAR (n1 ---) erwartet eine Zahl auf dem Stack. Der Screen mit der Nummer n1 wird gelöscht.

Sollte sich das Forthsystem beim Kompilieren eines Programmes von der Diskette oder von der Ramdisk mit einer Fehlermeldung bemerkbar machen, können Sie mit Hilfe des Wortes **WHERE** sich die Stelle im Quelltext zeigen lassen, wo der Fehler auftrat. Das Wort WHERE ist ebenfalls im FORTH Vokabular enthalten, der Editor braucht dazu nicht aufgerufen zu werden.

Denken Sie daran, daß der Inhalt der Screens spätestens vor dem Abschalten des Computers mit SAVE-BUFFERS oder FLUSH beim Diskettensystem bzw. mit SAVE-BUFFERS oder FLUSH und TAPE CSAVE beim Kassettensystem gesichert werden sollte.

Editor 2 (1024 bytes/screen)

Dieser Editor ist ebenfalls als Vokabular verfügbar und baut auf dem B♦Z-Vokabular auf. Das Forthsystem muß auf 1024 Bytes pro Screen umgestellt sein (siehe Vorschlag Arbeitsversion 2 , Disk bzw. Kassette).
Der Aufruf des Editor erfolgt mit dem EDITOR-Wort E :

E (n ---) E (für Edit) erwartet eine Screennummer auf dem Stack, lädt und listet diesen Screen im 80-Zeichen Format und geht in den Editiermodus. Wenn der Stack leer war, wird der Screen benutzt, dessen Nummer in der Variablen SCR abgelegt war. Falls noch keine Identifikationsmarkierung eingegeben wurde, wird der Anwender aufgefordert, diese einzugeben. Diese I.D. wird beim Verlassen des Editors in die Kopfzeile des Screens eingefügt und bietet die Möglichkeit, beispielsweise ein Namenskürzel mit Datum einzugeben. Wollen Sie das I.D. löschen, geben Sie interaktiv CLEAR-ID ein.

L (n ---) benutzen Sie interaktiv das Wort L zum Listen von Screens im 80-Zeichenformat.

Weitere Editierbefehle (^ = Controltaste):

^ +	Cursor links
^ *	Cursor rechts
^ -	Cursor nach oben
^ =	Cursor nach unten
Tab	Tabulatursprung nach rechts
^ Q	Tabulatursprung nach oben
^ A	Tabulatursprung nach unten
^ D	das rechts vom Cursor liegende Wort wird gelöscht, restlicher Text wird zusammengerückt
^ E	Editor verlassen, Screen wird nicht gesichert
ESC	Editor verlassen, veränderter Screen wird gesichert
^ C	aktuelle Linie löschen
^ L	Cursor auf Anfang linkes Wort
^ R	Cursor auf Anfang rechtes Wort
Shift > (Insert)	Linie einfügen, herausgeschobene Linien gehen verloren
Shift Delete	Linie löschen, alle darunter liegenden Linien wandern um eins nach oben
Shift < (Clear)	gesamten Screen löschen
Back S	aktuellen Buchstaben löschen, Text wird zusammengerückt
^ M	Insertmode an- bzw. abschalten

Wenn der Insertmode aktiviert ist, werden eingetippte Zeichen in den Text eingefügt, vorhandener Text wird nicht überschrieben. Aus

dem Screenfeld herausgeschobene Buchstaben gehen verloren.

Folgene Worte sind ebenfalls noch nützlich, definieren Sie bei Bedarf folgendes :

```
ONLY FORTH DEFINITIONS DECIMAL

: WHERE ( --- )
[ 8●Z ] DECIMAL ERRAT 2+ @ DUP LIST SCR !
ERRAT @ C/L MOD 12 +
ERRAT @ C/L / 2+
PCINIT POS. KEY DROP
PCEXIT EDITOR QUIT ;


ONLY FORTH ALSO EDITOR DEFINITIONS

: COPY ( from to --- )
SWAP BLOCK 2- ! UPDATE FLUSH ;

: CLEAR ( n --- )
BLOCK B/BUF BLANK UPDATE ;

ONLY FORTH DEFINITIONS
```

WHERE (---) zeigt bei Kompilationsfehlern den Screen und die Position innerhalb des Screens, wo der Fehler auftrat. Außerdem wird der Editor aufgerufen und man kann mit dem Editorwort E gleich anfangen, den Fehler zu korrigieren.

COPY (n1 n2 ---) erwartet zwei Zahlen auf dem Stack. Der Screen mit der Nummer n1 wird in den Screen mit der Nummer n2 kopiert.

CLEAR (n1 ---) erwartet eine Zahle n1 auf dem Stack. Der Screen mit der Nummer n1 wird gelöscht.

Denken Sie auch hier daran, daß der Inhalt der Screens spätestens vor dem Abschalten des Computers beim Diskettensystem mit FLUSH oder SAVE-BUFFERS, beim Kassettensystem ebenfalls mit FLUSH oder SAVE-BUFFERS und TAPE CSAVE gesichert werden sollte.

I/O und Grafik

Das IO-Vokabular enthält Worte, die den Zugriff auf das Betriebssystem des Atari gestatten, zum Basic kompatible Grafikbefehle und Worte zur Ansteuerung eines Druckers.

Das IO Vokabular stellt dem Benutzer folgende Worte zur Verfügung:

S: legt eine Adresse auf den Stack, in der ASCII 'S' abgelegt ist. Für Screen/Bildschirm-Operationen.
P: legt eine Adresse auf den Stack, in der ASCII 'P' abgelegt ist. Für Printer/Drucker-Operationen.
K: legt eine Adresse auf den Stack, in der ASCII 'K' abgelegt ist. Für Keyboard/Tastatur-Operationen.
C: legt eine Adresse auf den Stack, in der ASCII 'C' abgelegt ist. Für Cassette/Kassetten-Operationen.
E: legt eine Adresse auf den Stack, in der ASCII 'E' abgelegt ist. Für Editor-Operationen.
R: legt eine Adresse auf den Stack, in der ASCII 'R' abgelegt ist. Für RS232/Serielle Operationen.
R2: legt eine Adresse auf den Stack, in der ASCII 'R2' abgelegt ist, für den seriellen Handler, Kanal 2.
R3: legt eine Adresse auf den Stack, in der ASCII 'R3' abgelegt ist, für den seriellen Handler, Kanal 3.
R4: legt eine Adresse auf den Stack, in der ASCII 'R4' abgelegt ist, für den seriellen Handler, Kanal 4.

IOCB (--- adr) Variable, enthält Basisadresse des aktuellen IOCB (Input/Output Control Block)
IO.X (--- adr) Hilfsvariable in der Zeropage
IO.A (--- adr) Hilfsvariable in der Zeropage
IO# (n ---) wählt IOCB mit der Nummer n aus
<IO> - internes Definitionswort
ICCOM (--- adr) aktuelle Kommandoadresse, 1 Byte
ICSTA (--- adr) aktuelle Statusadresse, 1 Byte
ICBAL (--- adr) aktuelle Pufferadresse, 2 Bytes
ICBLL (--- adr) aktuelle Pufferlänge, 2 Bytes
ICAX1 ICAX2 ICAX3 ICAX4 ICAX5 ICAX6 (--- adr) jeweils ein Byte für Zusatzparameter bei I/O Operationen
>CIO (--- n) Das X-Register wird mit dem Inhalt von IO.X, der Akkumulator mit dem Inhalt von IO.A geladen, dann wird die zentrale IO-Betriebssystemroutine (CIOV) angesprungen und der Statuscode der Operation auf dem Stack übergeben
CLOSE (n ---) Schließt IOCB-Kanal, gibt bei Fehlschlag Fehlermeldung aus (siehe Check)
(CHECK) (n ---) führt I/O-Fehlertest durch
CHECK - Fehlertestwort
' (CHECK) IS CHECK (Fehlertest aktiv)
' DROP IS CHECK (Fehlertest inaktiv)
OPEN (adr icax1 icax2 c# ---) öffnet IO-Kanal
z.B.: P: 8 0 1 OPEN öffnet Kanal 1 für die Druckerausgabe.

PUT (b c# ---) schreibt ein Byte b in Kanal c#
GET (c# --- b) liest ein Byte b aus Kanal c#
PUTREC (adr len c# ---) schreibt einen Record (einen mit Return abgeschlossenen oder maximal 'len' langen Datenfile) in Kanal c#
GETREC (adr len c# ---) liest einen Record (s.o.) aus Kanal c# nach Adresse adr
STATUS (c# --- n) liest aktuelles Statusbyte aus Kanal c#
XIO (adr icax1 icax2 iccom c# ---) Spezialbefehl z.B.: für den Screenhandler zum Füllen von Flächen benutzt
GR. (n ---) Kurzform für GRAPHICS, öffnet Grafikmodus n mit Textfenster
GR.16 (n ---) öffnet Grafikmodus n ohne Textfenster
CLR# (--- adr) Variable, enthält aktuelle Farbe
POS. (x y ---) Kurzform für POSITION, setzt Grafikmodecursor auf Position x y
POS@ (--- x y) übergibt aktuelle Grafikcursorposition
COLOR (n ---) setzt Farbe n
PLOT (x y ---) zeichnet Pixel auf Position x y mit der Farbe in CLR#
LOC. (x y --- n) Kurzform für LOCATE, übergibt Farbwert des Grafikpunktes x y
G" (str ---) Beispiel: G" TEST " schreibt 'TEST' in den Grafikbildschirm (nur lesbar in Textmodes 1,2, etc.). Interaktiv und in der Kompilation verwendbar
DR. (x y ---) Kurzform für DRAWTO, zeichnet von der aktuellen Cursorposition zur Koordinate x y mit der durch CLR# festgelegten Farbe
FILL (n ---) füllt Fläche mit der Farbe n
SE. (r c l ---) Kurzform für SETCOLOR, setzt Bildschirmfarbe, und zwar Farbregister r mit der Farbe c in der Helligkeit l
SHUTUP (---) schließt alle Soundkanäle
SO. (n1 n2 n3 n4 ---) Kurzform von SOUND, steuert Soundkanal n1 (0 - 3) mit Frequenz n2 (0 - 255) und Verzerrung n3 (0 - 15) und Lautstärke n4 an

Die Diskettenversion enthält für das IO-Vokabular noch Worte, die einen EPSON-kompatiblen Drucker ansteuern. Für Anwender, die Recorder und Drucker besitzen, ist im Anhang der Sourcecode der Druckerroutinen beigelegt.

PROTOKOLL (---) alle von der Tastatur eingegebenen Befehle und Bildschirmausgaben werden auch auf dem Drucker ausgegeben. QUIT oder IO PROF beenden diese Betriebsart.

DOCS (n1 n2 ---) von Startscreen n1 bis Screen n2 werden alle Screens paarweise nebeneinander ausgedruckt. Die Bildschirmausgabe ist während dieser Zeit unterbunden. DOCS erkennt, ob mit 1024 oder 512 byte langen Blocks gearbeitet wird und formatiert die Druckerausgabe entsprechend.

Anwendungsbeispiele Grafik:

```
IO DEFINITIONS DECIMAL

: MOIRE ( --- )
  8 GR.16  2 0 0 SE.  1 COLOR
  1 319 DO 319 189 PLOT
           I 0 DR.
        -2 +LOOP
  ● 318 DO 0 ● PLOT
           I 189 DR.
        -2 +LOOP
  KEY DROP ;

MOIRE
```

```
IO DEFINITIONS DECIMAL

: FILLDEMO ( --- )
  7 GR.  3 COLOR
  100 50 PLOT
  100 10 DR.
  10  10 DR.
  10  50 POS.
  3 FILL ;

FILLDEMO
```

```
IO DEFINITIONS

: +COLOR  ( --- )
  CLR @ 15 = IF CLR# OFF
             ELSE 1 CLR# +!
             THEN ;

: DEMO ( --- )
  11 GR.  CLR# OFF  4 ● 2 SE.
  192 0 DO
      80 0 DO +COLOR
              I J PLOT
           LOOP +COLOR
        LOOP KEY DROP ;

DEMO
```

Das 80 Zeichenvokabular

Die 80-Zeichendarstellung wird softwaremäßig innerhalb eines Grafikstufe 8 Bildschirms realisiert. Die Textausgabe und Positionierung wird mit den 80Z-Vokabularworten TYPE SPACE SPACES D. D.R und POS. durchgeführt. Diese Worte sind gleichlautend mit den Worten im Forthkern bzw. im IO Vokabular, haben dieselben Auswirkungen auf den Stack und führen die gleichen Aktionen, bezogen auf den 80-Zeichenbildschirm, durch. Vor dem Benutzen dieser 80Z Worte muß der Bildschirm mit **80Z INIT** vorbereitet werden.
Das Zurückstellen der Farb- und Kontrasteinstellung am Fernseher/Monitor verbessert die Lesbarkeit. Falls Sie mit den Farben nicht einverstanden sein sollten, ändern Sie im Sourcecode das Wort INIT.

Folgende Worte sind für den Anwender interessant:

LIST (n ---) LIST listet einen Screen auf dem 80-Zeichenbildschirm, sollte dieser nicht aktiviert sein, wird das nachgeholt.

INIT (---) Initialisierung, Farbgebung, Cursor auf Position 0,0.

CLS (---) loescht den 80 Zeichenbereich.

SPACE (---) gibt ein Leerzeichen aus.

SPACES (n ---) gibt ab aktueller Cursorposition n Leerzeichen aus.

TYPE (adr n ---) gibt den bei Adresse adr liegenden, n Zeichen langen String aus. Beispiel: " Test" TYPE

D. (d ---) gibt die doppelt genaue Zahl d an der aktuellen Cursorposition aus.

D.R (d n ---) die doppelt genaue Zahl d wird in einem Feld der Weite n rechtsbündig ausgegeben.

Die Zeichenausgabe kann wahlweise mit oder ohne Cursor geschehen. Der Cursor ist ein Player und benutzt Page sechs, ($ 600 bis 6FF). Der Inhalt dieses Adressraumes geht also verloren. Zur Steuerung stehen folgende Worte zur Verfügung:

PCINIT (---) bereitet die Ausgabe mit Cursor vor. Alle hierauf folgenden Zeichenausgaben werden mit Cursor ausgeführt.

PCON (---) und **PCOF** (---) schalten den Cursor an der aktuellen Bildschirmposition an bzw. aus.

PCEXIT (---) schaltet den Cursormodus vollständig aus.

Vokabular und Suchordnung

Vorweg eine kurze Begriffserklärung:

Vokabular bezeichnet in einem Forthsystem eine Gruppe von Worten, die einem Anwendungsbereich zugeordnet werden. Zum Beispiel sind der Editor und der Assembler zu Vokabularen zusammengefasst.

CONTEXT-Vokabular(e) ist/sind die Vokabulare, die beim Suchlauf nach einem Wort durchlaufen werden.

CURRENT-Vokabular ist das Vokabular, in das neue Worte hineinkompiliert werden.

Die **Suchordnung** ist im Forthsystem von zentraler Bedeutung. Suchordnung bezeichnet hier die Reihenfolge der Vokabulare, in denen nach Worten gesucht wird.
Im F.I.G.-Forth ist z.B.: eine baumähnliche Suchstruktur implementiert. Es gibt ein CONTEXT- und ein CURRENT-Vokabular, beide münden im Endeffekt in das FORTH-Vokabular. Bei jedem Suchlauf wird zuerst das CONTEXT-Vokabular (inkl. FORTH), und wenn das gesuchte Wort nicht gefunden wurde anschließend das CURRENT-Vokabular (inkl. FORTH) durchsucht. Vokabulare sind vereinbarungsgemäß IMMEDIATE, werden also im Kompilationsmodus nicht kompiliert, sondern ausgeführt.

In die vorliegende F-83 Version wurde Ragsdale's experimenteller Vorschlag des ONLY-Vokabulars eingebracht. Dieses Konzept ist (noch) nicht im Standard enthalten, wird aber von den meisten Forth-83 Versionen unterstützt.
Das ONLY-Konzept ermöglicht eine weitaus mächtigere Manipulation der Suchordnung.

Die CONTEXT-Suchordnung kann jetzt aus bis zu sechs Vokabularen bestehen, die nacheinander durchsucht werden. FORTH und das CURRENT-Vokabular müssen nicht, wie im F.I.G.-Forth, notwendigerweise in der Suchordnung enthalten sein. Vokabulare sind hier vereinbarungsgemäß nicht IMMEDIATE.
Die oberste Position in der Suchordnung ist das 'transiente' Vokabular, die anderen in der CONTEXT-Suchordnung werden als 'resident' bezeichnet. Das Ausführen eines Vokabularwortes (z.B.: EDITOR) macht dieses Vokabular transient, es wird also zuerst durchsucht und ersetzt damit das zuvor transiente Vokabular.

Glossary:

ORDER (---) gibt auf dem Bildschirm die aktuelle Suchordnung aus: CONTEXT- und CURRENT-Vokabular.

ONLY (---) reduziert die Suchordnung auf das ONLY-Vokabular.

ALSO (---) macht das transiente Vokabular resident, d.h.: schiebt es um eine Position in der Suchordnung nach unten.

FORTH (---) Name des Kernvokabulars. Die Ausführung des Wortes FORTH macht dieses Vokabular transient.

WORDS (---) gibt alle Worte im obersten Vokabular auf dem Ausgabegerät aus.

FORGET (---) FORGET XXX löscht das Wort XXX aus dem Wörterbuch und damit auch alle nach XXX definierten Worte. Das Wort FORGET 'vergisst' zuerst aus dem CURRENT-Vokabular !

DEFINITIONS (---) kopiert das transiente Vokabular in das CURRENT-Vokabular. Z.B.: EDITOR DEFINITIONS Alle neuen Definitionen werden jetz in das EDITOR-Vokabular kompiliert.

SEAL (---) löscht jedes ONLY aus der CONTEXT-Suchordnung. So kann man zum Beispiel nur für die spätere Anwendung bestimmte Vokabulare bzw. Worte zulassen.

NULL (---) kompiliert ein Nullwort in das Wörterbuch, ermöglicht so weiterhin das interaktive Benutzen des Systems, wenn das Anwendervokabular sich allein in der Suchordnung befindet.

Beispiele:

```
ONLY         ( Nur ONLY wird durchsucht )
FORTH        ( Suchordnung FORTH, dann ONLY )
ALSO EDITOR  ( Suchordnung EDITOR, FORTH, dann ONLY )
DEFINITIONS  ( neue Worte werden ins EDITOR-Vokabular )
             ( kompiliert )
```

und kompiliert:

```
: SETUP  ONLY FORTH  ALSO EDITOR DEFINITIONS ;
```

Der FORTH-Assembler

In Forth lassen sich zeitkritische Programmteile leicht und elegant mit Hilfe des Assemblers formulieren und in das restliche Hochsprachenprogramm einfügen.
Der Forth-Assembler benutzt, wie das restliche Forthsystem, die umgekehrt polnische Notation. Alle Opcodes werden üblicherweise zusätzlich mit einem Komma versehen, einmal um Verwechslungen mit hexadezimalen Zahlen zu vermeiden (z.B.: ADC) und um den Zusammenhang mit dem Forthwort ',' (Komma) darzustellen, dem Eintrag ins Wörterbuch. Für die Notation der Opcodes des 6502 ergibt sich daher folgendes:

Forth	konvent. Assembler	Adressierungsart
nn # LDA,	LDA #nn	immediate
nn LDA,	LDA nn	zero page
nn ,X LDA,	LDA nn,X	zero page,X
nnnn LDA,	LDA nnnn	absolut
nnnn ,X LDA,	LDA nnnn ,X	absolut,X
nnnn ,Y LDA,	LDA nnnn ,Y	absolut,Y
nn X) LDA,	LDA (nn,X)	(indirekt,X)
nn)Y LDA,	LDA (nn),Y	(indirekt),Y
.A ROR,	ROR A	Akkumulator
nnnn JMP,	JMP nnnn	Absoluter JMP
nnnn) JMP,	JMP (nnnn)	Indirekter JMP

Die Prozessor-Register:

Das Y-Register enthält beim Einsprung in eine Codedefinition den Wert Null, und kann, wie der Akkumulator, frei benutzt werden. Das X-Register des 6502-Prozessors enthält den Zeiger auf den obersten Parameterstackwert. Wenn Maschinenprogramme bzw. 'Code'-Worte das X-Register verändern, sollte der Inhalt des X-Registers vorher gerettet werden. Der Assembler stellt dafür die Variable XSAVE in der Zeropage zur Verfügung.

Z.B.: CODE TEST XSAVE STX, ... XSAVE LDX, NEXT JMP, END-CODE

CODE :

Die beiden Worte CODE und END-CODE sind in etwa mit den Hochsprachworten ':' und ';' zu vergleichen.
';' und END-CODE führen einen Syntaxcheck durch und schließen die Wortdefinition ab. ':' und CODE erwarten beide einen Namen, CODE ruft jedoch nach dem Aufbau des Headers den Assembler auf. Z.B.:

```
CODE SWITCH
HEX
2C8 LDA, 80 # EOR, 2C8 STA,
NEXT JMP,
END-CODE
```

Folgendes ist passiert:

CODE SWITCH

- Es wurde der Header (Link- Name- und Codefeld) eines neuen 'Primitives' mit dem Namen SWITCH aufgebaut, also eines Wortes, dessen Runtimecode in Maschinensprache und nicht in Hochsprache formuliert werden wird.
- Das Vokabular ASSEMBLER wird nun als erstes in den Contextvokabularen durchsucht.
- Die Codefeldadresse von SWITCH zeigt jetzt auf 'HERE', den aktuellen Dictionarypointer und den Anfang des Maschinencodes.
- Im Unterschied zu ':' befindet sich das Forthsystem nicht im Compile-modus, sondern im interaktiven Zustand. Das bedeutet unter anderem, daß man vollen Zugriff auf die darunterliegenden Forthworte hat, um Berechnungen anzustellen oder um die Zahlenbasis zu wechseln etc.

HEX

- Umschalten auf hexadezimale Zahlenbasis.

2C8 LDA, 80 # EOR, 2C8 STA,

- Die den Mnemonics entspechenden Opcodes und Operanden werden in das Wörterbuch eingetragen.

NEXT JMP,

- Wenn nach Abarbeitung des Maschinencodes wieder ins Forthsystem zurückgekehrt werden soll, müssen bestimmte legale Ausgänge oder 'Exits' benutzt werden. 'NEXT' ist einer davon. 'NEXT' legt die Adresse auf den Stack (für JMP,), wo der Code des 'inneren Interpreters' zu finden ist, der Routine, die für die Ausführung des nächsten Wortes sorgt.

END-CODE

- schließt nach erfolgreicher Syntaxprüfung die Wortdefinition ab. Die Eingabe von SWITCH <RET> führt jetzt das Maschinenprogramm aus.

LABEL :

LABEL ermöglicht es dem Assemblerprogrammierer, einzelne Unterprogramme zu formulieren, die dann zum Beispiel aus anderen CODE-definitionen angesprungen werden können. Denkbar ist zum Beispiel auch das Erstellen von Tabellen, Bytefelder usw. mit Hilfe von ',' und 'C,'.
LABEL erwartet wie ':' oder CODE einen Namen, compiliert eine Konstante mit diesem Namen und ruft den Assembler auf. Die daran anschließende Definition eines Maschinenprogramms sollte auch mit END-CODE abgeschlossen werden.

Z.B.:

```
LABEL SUB1
... ( Assemblercode )
RTS, END-CODE
```

Die Eingabe von SUB1 <RET> führt jetzt nicht wie bei einer CODE-Definition den entsprechenden Maschinencode aus, sondern legt die Adresse, bei der der Code assembliert worden ist, auf den Stack. Das wird in der folgenden CODE-Definition benutzt:

```
CODE TEST
XSAVE STX, SUB1 JSR, ...
XSAVE LDX, NEXT JMP, END-CODE
```

Assembler-Exits :

(Falls das X-Register verändert wird, XSAVE benutzen ! Der Prozessor muss im binären Modus zurückkehren.)

NEXT - Rückkehr ins Forthsystem ohne Stackänderung.
POP - DROP, dann Sprung nach 'NEXT'.
2POP - DROP DROP, dann Sprung nach 'NEXT'.
3POP - DROP DROP DROP, dann Sprung nach 'NEXT'.
PUSH - Der oberste Stackwert wird überschrieben mit Lowbyte auf dem Returnstack und Highbyte im Akkumulator, dann Sprung nach 'NEXT'.
PUT - wie PUSH, Lowbyte auf dem Returnstack und Highbyte im Akku wird zusätzlich auf den Stack gelegt.
PUSHOA - legt den Inhalt des Akkumulators auf den Stack und springt nach 'NEXT'.

Weitere Variablen und Macros :

N - N übergibt die Adresse eines Bereiches in der Zeropage, der 9 Bytes (N-1 bis N+7) umfasst und frei für Maschinenprogramme ist. Dieser Bereich wird ebenfalls vom Forthsystem benutzt,der Inhalt bleibt nach dem Rücksprung nach 'NEXT' also nicht erhalten !.

```
CODE CALL ( adr --- )
BOT LDA, N STA, BOT 1+ LDA, N 1+ STA,
4C # LDA, N 1- STA, N 1- JSR, POP JMP,
END-CODE
```

SETUP - SETUP übergibt die Adresse einer Routine, die bis zu 4 Stackwerte (bestehend aus je 2 Bytes) zum weiteren Benutzen in den N-Bereich kopiert. Der Inhalt des Akku's enthält die Anzahle der Werte (1 - 4).

Z.B.:

```
CODE TEST ( n n --- )
2 # LDA, SETUP JSR, ( der Stackpointer ist um 4 angewachsen,
entspricht DROP DROP )
... ( weiterer Code )
NEXT JMP, END-CODE
```

BOT und SEC - werden folgendermaßen benutzt.

```
BOT LDA,     - lädt das Lowbyte des obersten Stackwertes,
entspricht 00 ,X LDA, .
BOT 1+ LDA, - lädt das Highbyte des obersten Stackwertes,
entspricht 01 ,X LDA, .
SEC LDA,     - lädt das Lowbyte des zweiten Stackwertes,
entspricht 02 ,X LDA, .
SEC 1+ LDA, - lädt das Highbyte des zweiten Stackwertes,
entspricht 03 ,X LDA, .
```

RP) - wird benutzt, um auf den Prozessor-Returnstack zuzugreifen.

```
Z.B.:
CODE TEST
XSAVE STX, TSX, ( beides notwendig ! )
RP) LDA,        ( entspricht 101 ,X LDA, )
...             ( weiterer Code )
XSAVE LDX, NEXT JMP,
END-CODE
```

Assembler Kontrollstrukturen:

Der vorliegende FORTH-Assembler basiert auf dem von William Ragsdale in die Public-Domain entlassenen Assembler. Er bietet die Möglichkeit mit Hilfe der Kontrollstrukturen BEGIN, UNTIL, IF, ELSE, und THEN, auch im Maschinencode strukturiert zu programmieren. Der Programmfluß wird von den Flags im Statusregister des 6502 abhängig gesteuert. Folgende Verzweigungs-Bedingungen stellt der Assembler in der Originalform zur Verfügung:

```
CS - C arryflag    = 1 / Carry gesetzt
0< - N egativflag  = 1 / kleiner Null
0= - Z Nullflag    = 1 / gleich Null
>= - C arryflag    = 1 / größer oder gleich
```

testet Carry, nur gültig nach einer Subtraktion oder einem Vergleich.

und deren Umkehrung mit NOT:

```
CS NOT - Carry clear
0< NOT - größer oder gleich Null
0= NOT - ungleich Null
>= NOT - kleiner
```

Beim Arbeiten mit dem Assembler hat man jetzt die Möglichkeit, entweder die strukturierenden Makros IF, BEGIN, etc. zu benutzen oder innerhalb des Codes einzelne Labels zu setzen und mit den relativen Branchbefehlen BCC, BCS, BEQ, BMI, BNE, BPL, BVC, BVS, anzuspringen.

Beispiele:

Strukturierter Code:

```
CODE XXX
BEGIN, ... ( Test ) 0<
UNTIL, NEXT JMP,
END-CODE


CODE XXX
( Test ) CS NOT IF, ...
                THEN, NEXT JMP,
END-CODE


CODE XXX
( Test ) 0= IF,   ...
           ELSE, ...
           THEN, NEXT JMP,
END-CODE
```

```
CODE XXX
BEGIN, ...
FLAG CMP, >= IF, LEGALEXIT JMP,
             THEN,
AGAIN,
END-CODE


CODE XXX
BEGIN, ( Test ) 0< NOT
WHILE, ...
REPEAT, NEXT JMP,
END-CODE
```

Beispiel für das Programmieren mit 'Set':

```
CODE XXX
XSAVE STX, ...
1 SET      ( Setzen von Label 1 )
...
2 SET      ( Setzen von Label 2 )
...
1 BCC,     ( Bei Carry clear Sprung nach Label 1 )
2 BEQ,     ( Falls gleich Sprung nach Label 2 )
XSAVE LDX, NEXT JMP,
END-CODE
```

ACHTUNG: Unbedingt innerhalb einer Codedefinition entweder nur strukturiert oder mit Labels arbeiten. Die Strukturmakros und die Labels benutzen den Stack für Adressen und Fehlercheck, beides kombiniert führt mit Sicherheit zu unvorhergesehenen Resultaten.

Die Kontrollstrukturen des Ragsdale-Assemblers lassen sich bei Bedarf zum Beispiel mit folgenden Worten erweitern (Sourcecode im Anhang):

AGAIN, - Ermöglicht BEGIN, ... AGAIN, Strukturen, ein Verlassen der Schleife ist nur mit einem Sprung auf einen legalen Exit möglich (Siehe Beispiel).

WHILE, REPEAT, - solange die Bedingung in dem Programmteil zwischen BEGIN, und WHILE, wahr ist, wird der Code zwischen WHILE, und REPEAT, ausgeführt (Siehe Beispiel).

C; - Aliaswort für END-CODE, spart Schreibarbeit und Platz.

VS - Testbedingung wie 0= etc. Testet, ob die Überlaufflagge V gesetzt ist. Selten benutzt. Auch in Kombination mit NOT möglich.

Anmerkungen über das Kassettensystem:

Die Kassettenversion des Compilers ist meines Wissens nach die einzige Forthversion für den Atari, die es dem Benutzer ermöglicht, erstens den Kassettenrekorder ohne Einschränkungen als Massenspeicher zu benutzen und zweitens völlig autark ohne Diskettenstation auszukommen. Der Tapebetrieb ist naturgemäß erheblich langsamer, aber zweifellos auch um einiges kostengünstiger. Welches Medium man als Massenspeicherung auch benutzen mag, als Maß für 'Know How' und Kreativität ist es mit Sicherheit nicht entscheidend.

Beim Kassettenbetrieb hat es sich bewährt, relativ hochwertiges Bandmaterial zu benutzen, da die Mehrkosten von zwei bis drei DM pro Kassette wohl nicht mit dem Ärger zu vergleichen sind, den man sich mit billigen Bändern durch 'Dropouts', Ladefehler usw. einhandelt. Problematisch ist es auch, von anderen Recordern mit Musik bespieltes oder mit anderen Recordern gelöschtes Bandmaterial zu verwenden, weil durch eine möglicherweise geringfügig unterschiedliche Tonkopfjustierung der Recorder bei erneuten Aufnahmen Störgeräusche zurückbleiben, die dann beim Laden oft zu dubiosen Fehlern führen. Also, am besten neues, gutes Bandmaterial verwenden, und wenn mal überspielt werden muß, bei wichtigen Aufnahmen das Band mit dem Datenrecorder vorher löschen.
Der Löschvorgang geht so:
Beim Recorder die RECORD- und PLAY-Taste drücken. Folgendes Eintippen:
DECIMAL 52 54018 C! <RET> (schaltet Kassettenrecordermotor an)
Warten Sie nun ab, bis das für Ihre Zwecke benötigte Band gelöscht ist.
Oder definieren Sie für diesen Zweck zwei neue Worte für den Tapebetrieb:

```
DECIMAL : MOTOR.ON  52 54018 C! ; <RET>
        : MOTOR.OFF 60 54018 C! ; <RET>

Ausführung durch MOTOR.ON  <RET>
            bzw. MOTOR.OFF <RET>
oder
DECIMAL 54018 CONSTANT MOTOR <RET>
        : AN 52 SWAP C! ; : AUS 60 SWAP C! ; <RET>

Ausführung durch MOTOR AN  <RET>
            oder MOTOR AUS <RET>
```

Die Masterkassette enthält folgende Files:

File 1.: Bootkern
File 2.: Utilities, MAKECBOOT, EXPAND-BUFFERS
File 3.: Assembler
File 4.: Editor 1
File 5.: Grafik/Sound
File 6.: Bootprogramm für den R:-Handler der 850er-Interfacebox
File 7.: 80Z (80-Zeichenvokabular)
File 8.: Editor 2
File 9.: MAKECBOOT für Systeme im 1024er-Modus.

Zum Laden von der Kassette geben Sie
TAPE CLOAD <RET>
zum Abspeichern
TAPE CSAVE <RET> ein.

Alle Files sind direkt hintereinander aufgenommen worden. Wenn Sie einen File geladen haben, können Sie, ohne das Band umzuspulen, gleich den nächsten Abschnitt laden.
Es empfiehlt sich, sich die Zählerstände der Startpunkte der einzelnen Files zu notieren, um sie später exakt wiederfinden zu können.
Allgemeingültige Zählerstände hier mitzuteilen nützt wenig, da die unterschiedlichen Zählwerke der Recorder erhebliche Toleranzen aufweisen.

Sollten Sie Schwierigkeiten beim Laden der Files der Originalkassette haben, tun Sie folgendes:
MOTOR AN <RET> (oben beschrieben)
Bandstelle kurz vor Filebeginn anfahren.
Monitor- bzw. Fernsehlautsprecher laut stellen, PLAY-Taste drücken, das Band abhören und kurz vor Beginn des Vorspanntones stoppen.
MOTOR AUS <RET>
Laden Sie jetzt den File mit TAPE CLOAD und speichern Sie den File sofort mit TAPE CSAVE auf eine neue Kassette. Dieser File müßte sich jetzt problemlos laden lassen, da er mit Ihrem Recorder aufgenommen wurde.

Kassettensystem, Ramdisk und TAPE-Vocabular

Für das Kassettensystem wurde eine Ramdisk installiert, die es dem Benutzer ermöglicht, entsprechend dem vorhandenen freien Speicherplatz im Computer auf bis zu 64 Screens zuzugreifen. Um die Kompatibilität zu den 400- und 800er Modellen zu erhalten, benutzt die Ramdisk nicht den Speicher, der bei den Computern ab 800 XL aufwärts parallel bzw. hinter dem Betriebsysytem liegt, sondern belegt dynamisch den momentan freien Speicherplatz unterhalb des Bildschirmspeichers (siehe Speicheraufteilung).

Wenn Sie durch LIST, BLOCK oder den Gebrauch des Editors auf den Massenspeicher (in diesem Fall die Ramdisk) zugreifen, wird der entsprechende Block aus der Ramdisk erstmal in die Screenbuffer geladen.
Falls Screens in den Buffern geändert worden sind (durch Update) und man FLUSH oder SAVE-BUFFERS eingibt, wird der geänderte Screen in die Ramdisk kopiert. Die Ramdisk belegt den Raum direkt unterhalb des obersten freien Speicherplatzes. Dieses für Programme und Daten oberste freie Byte wird in einer Betriebsystemvariablen festgehalten (MEMTOP 741,742) und ist abhängig von der momentan verwendeten Grafikstufe.
Für den Benutzer bedeutet das einerseits, daß ihm der je nach Grafikstufe maximale Speicherplatz zur Verfügung steht, andererseits aber auch, daß er, wenn er einen Kassettenfile geladen hat oder wenn er ein Programm editiert, nicht den Grafikmode wechseln darf, weil dann ein Teil der Daten verlorengehen kann.
Screen # 1 belegt in der Ramdisk den Bereich direkt unter MEMTOP, alle folgenden Screens werden in Richtung fallende Adressen abgelegt. Nach dem Booten der Kassette ist ein Screen 'aktiv', weitere werden aktiviert, wenn der entsprechende Screen z.B.: mit LIST angefordert wird. Die Anzahl der momentan aktivierten Screens wird in der Forthvariable #RAMBUFS festgehalten.
Wird ein Screen angefordert, den die Ramdisk nicht mehr aufnehmen kann, wird eine entsprechende Fehlermeldung ausgegeben.
Die Ramdisk fasst nach dem Booten des Kerns, im 512er-Modus 64 Screens, im 1024er-Modus 30 Screens, das ist für viele Anwendungen sicherlich ausreichend.
Da der Inhalt der Ramdisk nach dem Ausschalten des Computers verloren geht, sollte man den Inhalt spatestens vor dem Ausschalten auf die Kassette retten. Das geschieht mit dem Befehl
TAPE CSAVE <RET>.
CSAVE rettet alle benutzten Screens auf die Kassette, gehen Sie also sicher (z.B.: mit INDEX), daß nicht unnotige Screens mit gerettet werden. Haben Sie zum Beispiel 5 Screens benutzt, wollen aber nur die ersten drei retten, so ändern Sie den Wert der Variable #RAMBUFS in 3:
3 #RAMBUFS ! <RET>

ACHTUNG: Es werden immer die Screens ab Nummer 1 gerettet. Sie können daher nicht z.B.: Screen 5 bis 10 separat saven. Sollte es notwendig sein, die Ramdisk zu löschen, verwenden Sie das im TAPE Vokabular enthaltene Wort CLEAR-RAMDISK. Es setzt die Anzahl der aktiven Screens auf 1 zurück und löscht den gesamten Ramdiskinhalt.

Die Worte des TAPE-Vokabulars :
(Siehe auch Sourcecode im Anhang)

SIZE (-- n) legt die momentane Größe der Ramdisk im Speicher auf den Stack.

HI (-- n) legt die Adresse des für den Benutzer obersten freien Bytes (abhängig von der Grafikstufe) auf den Stack.

LO (-- n) übergibt die Adresse der unteren Grenze der Ramdisk.

?RANGE (#rambufs --) erwartet auf dem Stack eine Blocknummer. Testet, ob der Block im Speicher untergebracht werden kann. Gibt bei nicht ausreichendem Speicherplatz eine Fehlermeldung aus.

RIO (adr bl f --) führt den Datentransfer aus der Ramdisk in die Buffer bzw. von den Buffern in die Ramdisk durch. 'adr' ist die Quell- bzw. Zieladresse, 'bl' die Blocknummer, 'f' ein Flag. Wenn das Flag 0 ist, wird in die Ramdisk geschrieben, alle anderen Werte veranlassen das Lesen eines Blockes von der Ramdisk.

DISKIO (--) schaltet auf Diskbetrieb um.

TAPEIO (--) schaltet auf Kassetten- und Ramdiskbetrieb um.

CLEAR-RAMDISK (--) löscht den benutzten Teil der Ramdisk und schaltet auf Kassettenbetrieb um.

CLOAD (--) und **CSAVE** (--) laden bzw. schreiben den 'aktiven' Teil der Ramdisk von bzw. auf die Kassette.

Massenspeicher und Screenbuffer :

Die Screens:

Ein im üblichen Forth-Slang häufig benutzter Begriff ist der 'Screen'. Ein Screen ist eine Informationseinheit, die vom Massenspeicher gelesen wird und im Computer benutzt oder geändert werden kann. Screens sind üblicherweise entweder 512 oder 1024 Bytes lang und werden zum Beispiel dazu benutzt, um mit Hilfe des Editors Quelltexte für Programme festzuhalten und sie im Massenspeicher abzulegen oder um auf anderswie produzierte Daten zugreifen zu können. Ein Screen besteht, je nach Implementation, aus einen oder mehreren 'Blocks', wobei die Blocklänge in Forth-79 und Forth-83 mit 1024 Bytes festgelegt ist. Der vorliegende Forthcompiler bietet beide Möglichkeiten, entweder Blocks mit 512 oder 1024 Bytes. Direkt nach dem Laden des Sprachkerns ist das System auf 512-byte-Blocks eingestellt. Der Quelltext für die auf der zweiten Diskettenseite enthaltenenen Zusatzvokabulare und Utilities sind in 512-byte-Screens abgelegt und können nur geladen werden, wenn sich das System im 512er Modus befindet. Will man in den 1024-Modus wechseln, so führt man die beim Abschnitt 'Kernal und Arbeitsversion' beschriebenen Maßnahmen durch.

Wichtige Systemkonstanten in diesem Zusammenhang sind:

Default Disk kernal		/ nach Umstellung	
C/L	32	64	
B/BUF	512	1024	
B/SCR	1	1	
SEC/BLK	4	8	
#BUFS	3	3	
Screens von 1 bis 179		1 bis 89	/ bei Single Density
Screens von 1 bis 259		1 bis 129	/ bei Double Density

Die Frage, in welchem Modus man sein Forthsystem 'fährt', wird wohl jeder für sich selbst beantworten und hängt sicherlich nicht zuletzt von den Anwenderprogrammen ab. Zumindest gibt es hier keine Einschränkungen. In vielen Fällen wird die Arbeit mit 512-byte Blocks die handlichere Lösung sein.

Die Buffer:

Das Forthsystem benutzt für die interne Speicherung der von der Diskette bzw. von der Kassette anfallenden Daten einen Bereich im Ram, die 'Screenbuffer'. In der Speicherbelegung liegen diese Buffer direkt über dem Sprachkern zwischen FIRST und Limit (siehe Memory Map). Alle später hinzugefügten Worte werden direkt im Anschluss an diese Buffer ins Ram kompiliert (natürlich in Richtung höhere Adressen). Das bedeutet aber, daß in einem späteren Anwenderprogramm, welches unter Umständen nicht auf den Massenspeicher zugreift, einiges an Speicherraum nutzlos brachliegt. Das kann, je nach Sytemkonfiguration, zwischen einem

halben bis drei oder mehr Kilobyte ausmachen, Speicher, der sicherlich anderweitig sinnvoller genutzt werden könnte. Die Lösung für das Problem: man verschiebt die Buffer. Bei dieser Verschiebung und/oder der Umstellung des Systems auf 1024-Byte-Blocks ist allerdings Vorsicht geboten, da an der falschen Stelle liegende Buffer meist unerwünschte Auswirkungen nach sich ziehen (Systemabsturz etc.). Man sollte sich vorher also genau überlegen, wohin man sie verschiebt.

Zum Beispiel:
(bezieht sich auf die Bootversion)

```
FIRST                   / legt untere Buffergrenze auf den
                        / Stack
HEX 7000 RELOC          / Relozierung nach $ 7000 aufwärts
                        / ( da ist Platz genug )
DP !                    / FIRST ( von vorhin ) wird in die
                        / Variable DP geschrieben. Neue Worte
                        / werden jetzt an den Kern
                        / 'nahtlos' angefügt.
---->                   / jetzt das Anwenderprogramm
                        / dazu'load'en.
HERE RELOC              / die Buffer werden an das
                        / Wörterbuchende angehängt.
LIMIT IS DP             / der Dictionarypointer wird neu
                        / gesetzt, damit neue Worte nicht
                        / direkt in die Buffer kompiliert
                        / werden.
```

Unterschiede zwischen F.I.G.-Forth und dem 83er Standard

#TIB Neue Variable, enthält die im Terminal Input Buffer enthaltenen gültigen Zeichen. Wird von QUERY gesetzt.

' 'Tick' ist nicht mehr IMMEDIATE und übergibt jetzt die Codefeldadresse.

(FIND) Die Funktion dieses F.I.G.-Forth Wortes hat das 83er Standardwort FIND

+LOOP Die Schleife wird beendet, wenn der Index die Grenze zwischen dem Limit und Limit-1 überschreitet. Die Schleife 1 1 DO ... 1 +LOOP wird 65536 mal durchlaufen (im F.I.G.-Forth und Forth-79 nur einmal).

+- Nicht mehr im neuen Standard enthalten. Kann ersetzt werden durch : ?NEGATE 0< IF NEGATE THEN ;

+ORIGIN gestrichen.

-DUP jetzt ?DUP

-FIND gestrichen. Kann ersetzt werden durch BL WORD FIND

." Wird nur noch innerhalb von Wortdefinitionen benutzt.

.(Kann innerhalb und außerhalb von Wortdefinitionen benutzt werden.

ID. jetzt .ID

<BUILDS Ersetzt durch CREATE. Siehe Glossary.

<CMOVE jetzt CMOVE>

>BODY cfa -> pfa . Standardwort.

>LINK cfa -> lfa . Kein Standardwort.

>NAME cfa -> nfa . Kein Standardwort.

?DNEGATE siehe D+-

?DO Wie DO, die Schleife wird jedoch nicht durchlaufen, wenn das Schleifenlimit gleich dem Index ist.

?DUP Neue Bezeichnung für -DUP

ABORT" Neu. Siehe Glossary.

AGAIN Wie 0 UNTIL. Kein Standardwort.

BLANK Neu für BLANKS

BODY> pfa -> cfa . Kein Standardwort.

CFA gestrichen. Siehe BODY> NAME> LINK>

CONVERT arbeitet wie das alte Wort (NUMBER)

CREATE übernimmt die Funktion des F.I.G.-Forth Wortes <BUILDS

D+- Nicht mehr im neuen Standard enthalten. Kann ersetzt werden durch : ?DNEGATE 0< IF DNEGATE THEN ;

DMINUS Umbenannt in DNEGATE

DNEGATE Neu für DMINUS

END gestrichen. Wie UNTIL

ENDIF gestrichen. Wie THEN

EXPECT verändert SPAN

FIND Funktion des alten -FIND

IN Umbenannt in >IN

LEAVE Sofortige Beendigung einer Schleife.

L>NAME lfa -> nfa . Kein Standardwort.

LINK> lfa -> cfa . Kein Standardwort.

LOAD Kompilation von Screen 0 ist nicht erlaubt.

LOOP Beendigung der Schleife erfolgt, wenn die Grenze zwischen Limit und Limit-1 überschritten wird. Siehe +LOOP

M* Kein Standardwort mehr.

M/ Kein Standardwort mehr.

MINUS Umbenannt in NEGATE

MOVE gestrichen.

N>LINK nfa -> lfa . Kein Standardwort.

NAME> nfa -> cfa . Kein Standardwort.

NEGATE Neuer Name für MINUS

NFA gestrichen. Siehe >NAME

NOT Liefert Einerkomplement.

ORDER Gibt die momentane Suchordnung der Vokabulare aus. Kein Standardwort.

PICK Der oberste Stackwert wird mit 0 adressiert. Siehe Glossary.

R@ Neuer Name für R

R Umbenannt in R@

ROLL Der oberste Stackwert wird mit 0 adressiert. Siehe Glossary.

RP! Nimmt jetzt den obersten Stackwert zum Initialisieren des Returnstacks. Kein Standardwort.

S->D Umbenannt in S>D

SAVE-BUFFERS Nach dem Retten der Buffer werden die Massenspeicherbuffer nicht gelöscht.

SIGN Nimmt sein Argument jetzt vom obersten Stackplatz.

SP! Nimmt jetzt den obersten Stackwert zum Initialisieren des Parameterstacks. Kein Standardwort.

SPAN Neue Variable. Siehe EXPECT

TIB TIB ist jetzt eine Konstante im Gegensatz zu der Variablen in F.I.G.-Forth.

U* Umbenannt in UM*

U/ Umbenannt in UM/MOD

UM* Neuer Name für U*

UM/MOD Neuer Name für U/

VARIABLE benötigt keinen Initialisierungswert.

VLIST Umbenannt in WORDS

VOCABULARY Siehe Abschnitt 'Vocabular und Suchordnung'.

WORD legt jetzt zusätzlich die Adresse des Strings auf den Stack.

WORDS Neuer Name für VLIST. Kein Standardwort.

['] wird innerhalb einer Definition benutzt, um die cfa des folgenden Wortes zu kompilieren.

Floored Division

Die Divisionsworte im Forth-83 Standard reagieren anders als die vom Forth-79 und F.I.G.-Forth bekannten Worte. Das zugrunde gelegte Konzept für die Integerdivision nennt sich 'Floored Division', ist in SmallTalk und APL ebenfalls implementiert und führt zu einem symmetrischen Verhalten des Quotienten um den Nullpunkt.

Der arithmetische 'Floor' einer Zahl z ist diejenige ganze Zahl, die kleiner oder gleich z ist.
Der 'Floor' von 1.7 ist 1.
Der 'Floor' von 0.4 ist 0.
Der 'Floor' von -0.7 ist -1.

'Floored Division' ist die Bezeichnung für die Division, bei der der Quotient auf den arithmetischen 'Floor' abgerundet wird, und der Rest (der Remainder) das Vorzeichen des Divisors annimmt oder gleich 0 ist.

Beispiele:

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

Die Integerdivision 5/4 ergibt den Quotienten 1 und den Rest 1, oder anders geschrieben 1 + 1/4 = 1.25

Die Integerdivision 5/-4 ergibt den Quotienten -2 und den Rest -3, oder anders geschrieben -2 + (-3/-4) = -2 + 3/4 = -1.25

Double Number Extension

Im Anhang sind noch einige Screens enthalten, die die im Forthkern enthaltenen Worte, die doppelt genaue Zahlen verarbeiten, zu dem im Standard als 'Double Number Extension Word Set' bezeichneten Wortschatz ergänzen.
Die im Kern enthaltenen Worte sind :
2! 2@ 2DROP 2DUP 2OVER 2SWAP D+ D- D0= D2/ D< D> DABS DMAX DMIN DNEGATE D. D.R

Die zusätzlichen Worte sind D0< DU< 2ROT D= 2CONSTANT 2VARIABLE 0. 1. DU. und DU.R und sollten anhand der Stackdiagramme und der Anlehnung an die Namen einfach genauer Worte in ihrer Funktion deutlich werden.

Die Speicherbelegung des Forthsystems (siehe auch folgende Seite):

Page 0 ($ 00-FF) : Der Forth-Compiler benutzt die obere Hälfte der Zeropage von $ 80 - $ FF für die internen Register der 'Forth-maschine' und für den Bereich des Parameterstacks.

Die Register:
XSAVE : $ 80 , wird von Maschinenspracheroutinen als Speicher für das X-Register genutzt. Das X-Register des 6502 ist der Zeiger auf den aktuellen Stand des Parameterstacks.
IP : $ 81,82 , der 'Instruction pointer', der auf das als nächstes auszuführende Wort zeigt.
W : $ 84,85 , das 'Word adress register', Zeiger auf die aktuelle Codefeldadresse (des Wortes, das gerade ausgeführt wird).
N : $ 86-8E , 9 Bytes, (N-1 bis N+7), die systemintern verwendet werden, aber auch von Primitives benutzt werden können.

Der Parameterstack fasst 49 16-bit Worte und belegt den Zeropagebereich von $ 99 bis einschließlich $ FA.

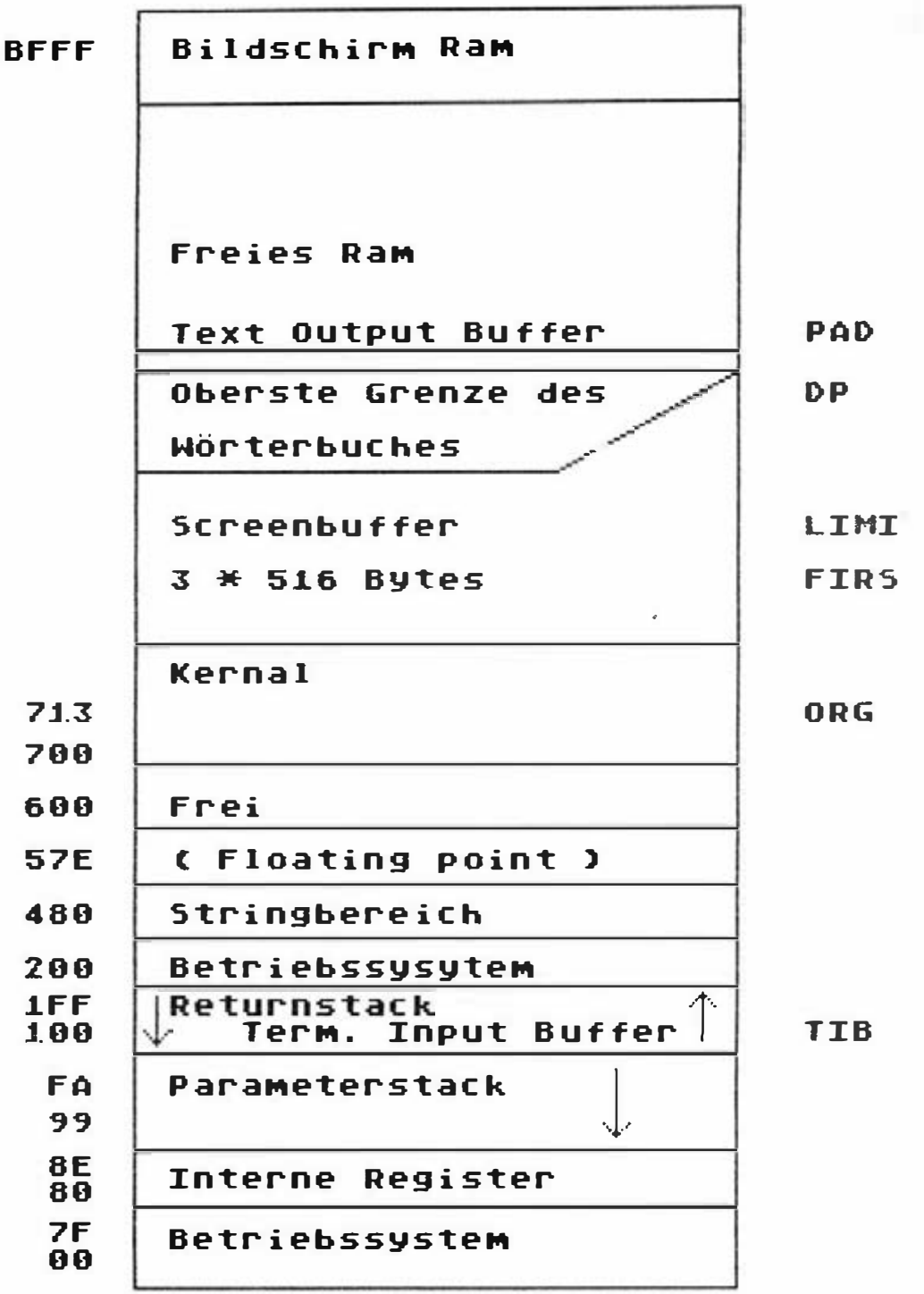
Page 1 ($ 100-1FF) wird belegt durch den 'Terminal input buffer' (der Tastatureingabebereich), der ab $ 100 beginnt und sich in Richtung höherwerdende Adressen ausbreitet - und durch den Returnstack, der umgekehrt ab $ 1FF nach unten wächst.

Der Speicherbereich von $ 200 bis $ 4FF wird wieder vom Betriebsystem benutzt. Ab $ 480 ist der Speicher für Zeichenketten reserviert, dieser Bereich kann aber durch Ändern der Systemkonstante STR auf andere Adressen verschoben werden.

Bereich $ 57E - $ 5FF wird von den Floating-Pointroutinen benutzt.

Page 6 ist frei für Anwenderzwecke, (der 80-Zeichen- Editor benutzt Page 6 für den Cursor - Vorsicht !)

Ab $ 700 beginnt der Sprachkern in Richtung aufsteigende Adressen bis zu den Screenpuffern, deren Grenzen durch die Konstanten FIRST und LIMIT festgelegt sind.
Die oberste Grenze des Wörterbuches wird in der Variablen DP festgehalten, ab HERE (HERE legt den Inhalt der Variablen DP auf den Parameterstack) beginnt die Compilation neuer Worte.
80 Bytes über HERE liegt PAD, ein Allzweckbuffer, der vom Systemkern benutzt wird (z.B.: .ID), aber auch dem Anwender für eigene Zwecke zugänglich ist.
Je nach gerade aktiviertem Grafikmode belegt der Bildschirmspeicher und die Displaylist den Speicherbereich von $ BFFF an abwärts. Beim Kassettensystem beginnt am ersten freien Byte unterhalb der Displaylist der Speicherbereich der Ram-Disk.


BFFF
Bildschirm Ram
Freies Ram
Text Output Buffer
PAD
Oberste Grenze des
Wörterbuches
DP
Screenbuffer
3 * 516 Bytes
LIMI
FIRS
Kernal
713
ORG
700
600
Frei
57E
(Floating point)
480
Stringbereich
200
Betriebssysytem
1FF
Returnstack
100
Term. Input Buffer
TIB
FA
99
Parameterstack
8E
80
Interne Register
7F
00
Betriebssystem

```
SCR#   84
\ PRON REV      START      180486

DCX
ONLY FORTH ALSO IO DEFINITIONS

85 89 THRU DCX

ONLY FORTH DEFINITIONS ALSO
```

```
SCR#   85
 0 \ PRON REV                  180486
 1
 2
 3  : (>E:P:)
 4  DUP (EMIT) DUP 126 = \ BS
 5  IF DROP 8
 6  THEN 5 PUT ;
 7
 8  : (>P:)   5 PUT ;
 9
10  : >E:P: ['] (>E:P:) IS EMIT ;
11  : >P:   ['] (>P:)   IS EMIT ;
12
13
14
15
```

```
SCR#   86
\ PRON REV                  180486
DCX
: CONDENSED 15 5 PUT ;
: NONDENSED 18 5 PUT ;
: PAGE      12 5 PUT ;
: RING       7 5 PUT ;
: ESC       27 5 PUT ;

: PRON 5 CLOSE P: 8 0 5 OPEN
ESC ASCII N 5 PUT 3 5 PUT
          \ SKIP PERFORATION
ESC ASCII R 5 PUT 0 5 PUT
          \ INTERN. ZSATZ
ESC ASCII 1 5 PUT
          5 5 PUT
155 5 PUT ;
```

```
SCR#   87
 0 \ PRON REV                  180486
 1
 2 : PROF
 3 155 EMIT 5 CLOSE
 4 ['] (EMIT) IS EMIT ;
 5
 6 : PROTOKOLL
 7   PRON >E:P: ;
 8
 9
10
11
12
13
14
15
```

```
SCR#   88
\ PRON REV                  180486

: DOC  ( l# r# - ) CR
." SCR# " OVER 0 4 D.R 2DUP <>
IF B/BUF 1024 = IF 55 ELSE 23
                THEN SPACES
." SCR# " DUP  0 4 D.R
THEN
16 0
DO CR
   OVER BLOCK I C/L * +
   C/L TYPE 2DUP <>
   IF SPACE I . 2 D.R SPACE
   DUP BLOCK I C/L * + C/L TYPE
   THEN
LOOP CR 2DROP ;
```

```
SCR#   89
 0 \ PRON REV                  180486
 1 : DOCS
 2 PRON B/BUF 1024 = IF
 3 CONDENSED ELSE NONDENSED THEN
 4 >P: CR SWAP
 5 BEGIN 3 0
 6  DO 2DUP < KEY? OR
 7       IF LEAVE
 8       ELSE DUP DUP 2OVER <>
 9       NEGATE + DOC THEN 2+
10  LOOP 2DUP < KEY? OR NOT
11 WHILE PAGE
12 REPEAT 2DROP
13 NONDENSED PAGE RING PROF ;
14
15
```

```
SCR#  123
\ TAPE
ONLY DEFINITIONS VOCABULARY TAPE
     ONLY FORTH DECIMAL
ALSO IO ALSO TAPE DEFINITIONS

VARIABLE #RAMBUFS
: SIZE #RAMBUFS @ B/BUF * ;
: HI 741 @ ;  : LO HI SIZE - ;
: ?RANGE ( #rambufs --- )
  DUP B/BUF UM* HI 0
  2SWAP D- PAD B/BUF + 0
    D< IF EMPTY-BUFFERS CR
       ." RAM-DISK ERROR !"
       ABORT THEN #RAMBUFS ! ;
124 126 THRU
ONLY FORTH ALSO DEFINITIONS
```

```
SCR#  124
 0 \ TAPE
 1 : RIO ( adr bl f --- )
 2   >R DUP 1 < ?BLOCK
 3   DUP #RAMBUFS @ >
 4   IF DUP ?RANGE THEN
 5   B/BUF * HI SWAP -
 6   R> IF SWAP THEN B/BUF CMOVE ;
 7
 8 FORTH DEFINITIONS
 9
10 : DISKIO ['] (R/W) IS R/W ;
11
12 : TAPEIO  [ TAPE ]
13   1 #RAMBUFS ! ['] RIO IS R/W ;
14
15 TAPE DEFINITIONS
```

```
SCR#  125
\ TAPE

: CLEAR-RAMDISK
  LO SIZE ERASE TAPEIO
  EMPTY-BUFFERS ;

: CLOAD
  CLEAR-RAMDISK
  1 CLOSE C: 4 128 1 OPEN
  1 GET ?RANGE
  128 1 DO 1 GET DROP LOOP
  LO SIZE BOUNDS
          DO 1 GET I C!
          LOOP 1 CLOSE ;
```

```
SCR#  126
 0 \ TAPE
 1
 2 : CSAVE
 3   FLUSH
 4   1 CLOSE C: 8 128 1 OPEN
 5   #RAMBUFS @ 1 PUT
 6   128 1 DO 0 1 PUT LOOP
 7   LO SIZE BOUNDS
 8           DO I C@ 1 PUT
 9           LOOP 1 CLOSE ;
10
11
12
13
14
15
```

```
SCR#   70
( ASSEMBLER START )

ONLY DEFINITIONS
VOCABULARY ASSEMBLER
FORTH ALSO ASSEMBLER
           DEFINITIONS
DCX 71 83 THRU



ONLY FORTH ALSO DEFINITIONS
DCX
```

```
SCR#   71
  0 ( ASSEMBLER )
  1
  2 VARIABLE MODE
  3
  4 : ZP  2 MODE ! ; STOP
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
SCR#   72
( ASSEMBLER )
HEX

  80 CONSTANT XSAVE
  84 CONSTANT W
  81 CONSTANT IP
  87 CONSTANT N
0A16 CONSTANT PUSHOA
072B CONSTANT PUSH
072D CONSTANT PUT
07E3 CONSTANT SETUP
07F4 CONSTANT 3POP
07F6 CONSTANT 2POP
07F8 CONSTANT POP
0732 CONSTANT NEXT
```

```
SCR#   73
  0 ( ASSEMBLER )
  1 VARIABLE INDEX -2 ALLOT
  2 0909 , 1505 , 0115 , 8011 ,
  3 8009 , 1D0D , 8019 , 8080 ,
  4 0080 , 1404 , 8014 , 8080 ,
  5 8080 , 1C0C , 801C , 2C80 ,
  6
  7
  8 : .A 0 MODE ! ; : #  1 MODE ! ;
  9 : ,X 3 MODE ! ; : ,Y 4 MODE ! ;
 10 : X) 5 MODE ! ; : )Y 6 MODE ! ;
 11 : ) 0F MODE ! ;
 12
 13 : BOT ,X 0   ;
 14 : SEC ,X 2   ;
 15 : RP) ,X 101 ;
```

```
SCR#   74
( ASSEMBLER )
: UPMODE
IF MODE @ 8 AND 0=
     IF 8 MODE +! THEN
THEN
1 MODE @ 0F AND ?DUP
IF 0 DO DUP + LOOP THEN
OVER 1+ @ AND 0= ;

: CPU
CREATE C, DOES> C@ C, ZP ;
```

```
SCR#   75
  0 ( ASSEMBLER )
  1 00 CPU BRK, 18 CPU CLC,
  2 D8 CPU CLD, 58 CPU CLI,
  3 B8 CPU CLV, CA CPU DEX,
  4 88 CPU DEY, E8 CPU INX,
  5 C8 CPU INY, EA CPU NOP,
  6 48 CPU PHA, 08 CPU PHP,
  7 68 CPU PLA, 28 CPU PLP,
  8 40 CPU RTI, 60 CPU RTS,
  9 38 CPU SEC, F8 CPU SED,
 10 78 CPU SEI, AA CPU TAX,
 11 A8 CPU TAY, BA CPU TSX,
 12 8A CPU TXA, 9A CPU TXS,
 13 98 CPU TYA,
 14
 15
```

```
SCR#   76
( ASSEMBLER )
: M/CPU
CREATE C, , DOES>
DUP 1+ @ 80 AND
IF 10 MODE +! THEN
OVER FF00 AND UPMODE UPMODE
IF ZP CR LAST @ .ID
 ABORT" INCORRECT ADRESS MODE"
THEN
C@ MODE C@ INDEX + C@ + C,
MODE C@ 7 AND
IF MODE C@ 0F AND 7 <
   IF C, ELSE , THEN
THEN ZP ;
```

```
SCR#   77
 0 ( ASSEMBLER )
 1 1C6E 60 M/CPU ADC,
 2 1C6E 20 M/CPU AND,
 3 1C6E C0 M/CPU CMP,
 4 1C6E 40 M/CPU EOR,
 5 1C6E A0 M/CPU LDA,
 6 1C6E 00 M/CPU ORA,
 7 1C6E E0 M/CPU SBC,
 8 1C6C 80 M/CPU STA,
 9 0D0D 01 M/CPU ASL,
10 0C0C C1 M/CPU DEC,
11 0C0C E1 M/CPU INC,
12 0D0D 41 M/CPU LSR,
13 0D0D 21 M/CPU ROL,
14 0D0D 61 M/CPU ROR,
15 0414 81 M/CPU STX,
```

```
SCR#   78
( ASSEMBLER )
0486 E0 M/CPU CPX,
0486 C0 M/CPU CPY,
1496 A2 M/CPU LDX,
0C8E A0 M/CPU LDY,
048C 80 M/CPU STY,
0480 14 M/CPU JSR,
8480 40 M/CPU JMP,
0484 20 M/CPU BIT,

: NOT 20 + ;

90 CONSTANT CS
D0 CONSTANT 0=
10 CONSTANT 0<
90 CONSTANT >=
```

```
SCR#   79
 0 ( ASSEMBLER )
 1 : BEGIN, HERE 1 ; IMMEDIATE
 2
 3 : UNTIL,
 4   >R 1 ?PAIRS R> C,
 5   HERE 1+ - C, ; IMMEDIATE
 6 : IF, C, HERE 0 C, 2 ; IMMEDIATE
 7
 8 : THEN,  2 ?PAIRS HERE OVER C@
 9   IF SWAP !
10   ELSE OVER 1+ - SWAP C!
11   THEN ; IMMEDIATE
12
13 : ELSE, 2 ?PAIRS
14   HERE 1+ 1 JMP, SWAP HERE OVER
15   1+ - SWAP C! 2 ; IMMEDIATE
```

```
SCR#   80
( ASSEMBLER )

DCX

VARIABLE SVAR 10 ALLOT

: ERA_SVAR ( --- )
SVAR 12 ERASE ;

: SET ( n --- )
6 MIN 1 MAX
HERE SWAP 1- 2* SVAR + ! ;
```

```
SCR#   81
 0 ( ASSEMBLER )
 1
 2 : RBR CREATE C, DOES> C@ C,
 3       HERE SWAP 0 C, ;
 4 ( n --- here n )
 5
 6 HEX
 7 90 RBR BCC,     B0 RBR BCS,
 8 F0 RBR BEQ,     30 RBR BMI,
 9 D0 RBR BNE,     10 RBR BPL,
10 50 RBR BVC,     70 RBR BVS,
11 DCX
12
13 : DIF      CSP @ SP@ - 2- 2/ ;
14
15
```

```
SCR#    82
( ASSEMBLER )

: FIX_IT
  DIF 2/ 0
  ?DO 1- 2* SVAR + @ OVER - 1-
      SWAP C!
  LOOP ;


: END-CODE
  FIX_IT CURRENT @ CONTEXT !
  ?EXEC ?CSP REVEAL ;

  IMMEDIATE
```

```
SCR#    83
  0 ( ASSEMBLER )
  1 ALSO FORTH DEFINITIONS
  2
  3
  4 : ;CODE   ?CSP COMPILE (;CODE)
  5   ASSEMBLER !CSP ZP ERA_SVAR
  6   [COMPILE] C ; IMMEDIATE
  7
  8 : CODE  ?EXEC CREATE
  9   HERE REDIRECT HIDE ASSEMBLER
 10   !CSP ZP ERA_SVAR ; IMMEDIATE
 11
 12 : LABEL ?EXEC CREATE HIDE
 13   ASSEMBLER !CSP ZP ERA_SVAR ;
 14 IMMEDIATE
 15
```

```
SCR#    84
( ASM 2 )
ASSEMBLER DEFINITIONS HEX
50 CONSTANT VS

LABEL BINARY
INX, INX, PUT JMP, END-CODE

: C;  [COMPILE] END-CODE ;
  IMMEDIATE     DECIMAL

( BINARY ist ein weiterer     )
( legaler Exit, der in        )
( manchen Programmen auf-     )
( taucht. Funktion von        )
( DROP und PUT. Uebergabe     )
( lo auf R.stack, hi im Akku )
```

```
SCR#    85
  0 ( ASM 3 )
  1 ASSEMBLER DEFINITIONS HEX
  2
  3 : AGAIN,  1 ?PAIRS 4C C, , ;
  4   IMMEDIATE
  5
  6 : WHILE,  SWAP 1 ?PAIRS
  7   [COMPILE] IF, DROP 3 ;
  8   IMMEDIATE
  9
 10 : REPEAT,  3 ?PAIRS SWAP 4C C, ,
 11   2 [COMPILE] THEN, ;
 12   IMMEDIATE
 13
 14 DECIMAL
 15
```

```
SCR#   86
\ GUELTIGKEIT SINGLE DOUBLE

n  16bit mit Vorzeichen
   -32768 bis 32767

u  16bit ohne Vorzeichen
   0 bis 65535

d  32bit mit Vorzeichen
   -2.147.483.648 bis
    2.147.483.647

ud 32bit ohne Vorzeichen
   0 bis 4.294.967.295
```

```
SCR#   87
  0 ( START DOUBLE NUMBER )
  1
  2 ONLY FORTH DEFINITIONS
  3 DECIMAL
  4
  5 88 90 THRU STOP
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
SCR#   88
( DOUBLE NUMBER 1 )

: D0<  ( d --- f )
  NIP 0< ;

: DU<  ( d d --- f )
  DUP 3 PICK XOR
  0< IF 2DROP D0< NOT
     ELSE D- D0<
     THEN ;
```

```
SCR#   89
  0 ( DOUBLE NUMBER 2 )
  1
  2 : 2ROT  ( d1 d2 d3 - d2 d3 d1 )
  3   5 ROLL 5 ROLL ;
  4
  5 : D=  ( d d --- f )
  6   D- D0= ;
  7
  8 : 2CONSTANT
  9     CREATE , , DOES> 2@ ;
 10
 11 : 2VARIABLE
 12     CREATE 0 , 0 , ;
 13
 14 0. 2CONSTANT 0.
 15 1. 2CONSTANT 1.
```

```
SCR#   90
( DOUBLE NUMBER 3 )

: (DU.)  ( d --- adr len )
  <# #S #> ;

: DU.  ( d --- )
  (DU.) TYPE SPACE ;

: DU.R  ( d len --- )
  >R (DU.) R> OVER - SPACES
  TYPE ;
```

```
QUIT

SCR # 80
 0 \ DEMO DEFINING WORDS                                          HK 020886
 1
 2 : MAKE1
 3   CREATE ." BUSY !"     ;CODE HEX 02C8 LDA, 80 # EOR,
 4                               02C8 STA, NEXT JMP,
 5                               END-CODE  DECIMAL
 6
 7
 8 : MAKE2
 9   CREATE ." BUSY !"     DOES>  DROP [ HEX ] 80 2C8 CTOGGLE ;
10
11   DECIMAL
12
13 \ MAKE1 TEST1
14 \ MAKE2 TEST2
15

SCR # 83
 0 \ DEMO SEAL , RESETPROOF                                       HK 020886
 1 ONLY FORTH DEFINITIONS  DECIMAL
 2
 3 : NOINIT  [  (INIT) 2+ @ , ] STANDARD ;
 4 ' NOINIT IS RESET     ' NOOP IS ERROR
 5
 6 ONLY DEFINITIONS  VOCABULARY TEST
 7 FORTH ALSO TEST DEFINITIONS
 8
 9 : GAME CLS CR ." THE VERY BIG CAVE ADVENTURE !" CR
10           CR ." YOU ARE AT THE BOTTOM OF A PIT."
11        BEGIN CR ." WHAT NOW ?" PAD 200 LISTEN SPO SP!
12              CR ." NO WAY ! THINK AGAIN !" AGAIN ;
13
14 ' GAME IS MAIN  ID 0 GR.  ONLY TEST SEAL GAME
15
```

Weiterführende Literatur

Goppold/Bouteiller
FORTH: Ein Programmiersystem ohne Grenzen

erschienen in der
Edition Aragon, Hombergerstr. 30, 4130 Moers

Leo Brodie
Programmieren in FORTH

und

Leo Brodie
In FORTH denken

beide erschienen im
Carl Hanser Verlag
Postfach 860420, 8000 München

Ronald Zech
Die Programmiersprache FORTH

erschienen im
Franzis' Verlag, 8000 München

Kontakte:

Forth - Gesellschaft Deutschland

Schanzenstr. 27, 2000 Hamburg 6

Forth Interest Group

P.O.BOX 1105, San Carlos, CA 94070, USA

Glossary

Die Worte des FORTH-Kerns sind hier alphabetisch aufgeführt.

Die Parameterstackaktionen werden nach jedem Wortnamen in runden Klammern angegeben. Vor dem '---' sind die Werte angegeben, die das entsprechende Wort vor der Ausführung erwartet, nach dem '---' sind die Werte angegeben, die das Wort nach der Ausführung auf dem Parameterstack hinterläßt.

Die Parameter werden durch folgende Symbole gekennzeichnet:

adr Adresse im Speicher
b 8-bit Byte
c Ascii-Zeichen
f Flag, Wahr = -1, Falsch = 0
n 16-bit vorzeichenbehaftete Zweierkomplementzahl
u 16-bit Absolutzahl
d doppelt genaue 32-bit Zweierkomplementzahl
ud 32-bit Absolutzahl
str Zeichenstring, str ist gleich (adr n), wobei 'adr' die Adresse ist, bei der der 'n' Zeichen lange String abgelegt ist.

Adressen im Adressbereich von Doppelpunktdefinitionen (Colondefinitions) werden abgekürzt :
LFA - Linkfeldadresse
NFA - Namensfeldadresse
CFA - Codefeldadresse
PFA - Parameterfeldadresse

! (n adr ---)
speichert n nach Adresse adr.

!CSP (---) speichert den aktuellen Parameterstackpointer in die Variable CSP.

" (--- (str)) im Kompilationsmodus wird der zwischen '"' eingeschlossene String kompiliert, im Exekutivmodus wird str auf dem Stack übergeben. Bei Runtime (einer Colondefinition) wird ebenfalls str (zum Beispiel für Type) übergeben. Doppelte " werden als ein " verarbeitet. IMMEDIATE.

(ud1 --- ud2) wird zwischen <# und #> beim Aufbau eines Ziffernstrings benutzt. ud2 ist der Quotient der Division ud1 durch die aktuelle Zahlenbasis BASE.

#> (d --- str) beendet Ziffernumwandlung, hinterläßt str des umgewandelten Strings.

#BUFS (--- n) Konstante, liefert Anzahl der momentan aktiven

Massenspeicherbuffer.

#DENSITY (--- adr) Variable, enthält Flag für das Diskettenformatieren in einfacher oder doppelter Dichte, wird von DENSITY gesetzt.

#LINE (--- adr) Variable, wird mit jedem CR um eins erhöht.

#OUT (--- adr) Variable, wird mit jedem Zeichen, das mit TYPE ausgegeben wird, um eins erhöht.

#S (ud1 --- ud2) zwischen <# und #> benutzt, wandelt den verbleibenden Rest der Ziffer in einen String um.

#TIB (--- adr) Variable, enthält die Anzahl der im Terminalinputbuffer enthaltenen Zeichen.

#VOCS (--- n) Konstante, maximale Anzahl der Kontextvokabulare.

' (--- adr) übergibt die Codefeldadresse (im F.I.G.-Forth die Parameterfeldadresse) des nächsten Wortes im Inputstream. Ist im Gegensatz zu F.I.G.-Forth nicht mehr IMMEDIATE.

((---) wird zum Kommentieren benutzt. IMMEDIATE. Begrenzer ist) Z.B.: (Kommentar)

(") (--- str) Runtimecode für kompilierten String

(+LOOP) (n ---) Runtimeprozedur von +LOOP, erwartet den Wert n auf dem Stack, mit dem der Schleifenindex hochgezählt wird. Die Schleife wird beendet, wenn der Index die Grenze zwischen Limit-1 und Limit überschreitet.

(.") (---) druckt String aus. Runtimecode von ."

(;CODE) (---) Runtimecode von ;CODE. Siehe ;CODE und Beispiel im Anhang.

(?DO) (n1 n2 ---) Runtimecode von ?DO, n1 ist das Schleifenlimit, n2 der Startindex, springt zum Schleifenende, wenn n1=n2 ist, oder legt die Exitadresse der Schleife, n1 und n2 auf den Returnstack.

(ABORT") (f ---) Runtimecode von ABORT" , wenn f wahr ist, wird die im Wörterbuch stehende Fehlermeldung ausgegeben, ERROR USERR und ABORT aufgerufen.

(BLOCKS) (--- str) übergibt bei der Kompilation von der Diskette oder von der Ramdisk die Adresse und Länge des aktuellen Input-Massenspeicherblocks.

(D.) (d --- str) wandelt doppelt genaue Zahl d in String str.

(DEFER) (---) die CFA des zuletzt definierten Wortes XXX wird dergestalt verändert, das das erste Doppelbyte im Parameterfeld von XXX die Adresse der CFA angibt, die bei Ausführung von XXX angesprungen wird.

(DO) (n1 n2 ---) Runtimecode von DO, wie (?DO), es erfolgt jedoch kein Test ob n1=n2 ist, die Schleife wird mindestens einmal durchlaufen.

(EMIT) (b ---) Runtimecode von EMIT, gibt b auf dem Bildschirm aus.

(ERROR) (---) voreingestellte Fehlerroutine, das oberste Kontextvokabular wird gleich FORTH gesetzt. Falls der Fehler während einer Kompilation auftrat, wird das fehlerhafte Wort 'vergessen'.

(EXPOUND) (---) Runtime von EXPOUND. Äußerer Interpreter. abhängig von STATE wird kompiliert oder interpretiert.

(FIND) (adr1 nfa --- adr2 f) durchsucht das Vokabular ab nfa nach dem String, der ab Adresse adr1 abgelegt ist. Bei Erfolg ist adr2 die CFA des gesuchten Wortes, f ist 1 bei IMMEDIATE- und -1 bei 'normalen' Worten. Wenn das Wort nicht gefunden wurde, ist adr2=adr1 und f ist 0. Siehe FIND.

(FMT) (n --- b) Diskettenformatierroutine. n ist die Nummer der Diskettenstation, b der Statuscode des Betriebssystems. Siehe FORMAT.

(FORGET) (adr ---) wird von FORGET aufgerufen. adr ist die LFA des Wortes, das vergessen werden soll.

(HEADER) (str ---) baut aus dem String str einen neuen Wortkopf im Wörterbuch auf. Die Länge des Namensfeldes ist abhängig von WIDTH. Das neue Wort legt bei Ausführung seine PFA auf den Stack. Siehe HEADER.

(INIT) (---) öffnet Betriebssysytemkanal 3 zur Tastatureingabe (K:), Kanal 4 zur Bildschirmausgabe (E:), und Kanal 5 für den Drucker (P:). Setzt CONTEXT und CURRENT auf FORTH.

(IS) (cfa ---) wird von IS kompiliert. Schreibt cfa in das Parameterfeld des in der Eingabe folgenden DEFER-Wortes.

(.) (---) Compilerschleife. Kompiliert den aus der Eingabe (Tastatur oder Disk/Ramdisk) kommenden Text, bzw. führt IMMEDIATE-Worte aus. Abbruch, wenn STATE = 0.

(KEY) (--- b) Runtimecode von KEY. Fragt ein Zeichen von der Tastatur ab.

(KEY?) (--- f) das Flag ist TRUE (-1), wenn die letzte Taste

immer noch gedrückt ist.

(LEAVE) (----) beendet eine Schleife sofort, wird von LEAVE kompiliert.

(LINE) (n1 n2 --- adr n3) die Zeile n1 im Screen n2 wird in die Bufferadresse adr und Zeilenläge n3 umgewandelt. Wenn der Screen nicht im Buffer vorliegt, wird er vom Massenspeicher geladen.

(LIT) (--- n) wird von LITERAL kompiliert. Runtime: legt kompilierte Zahl n auf den Stack.

(LOOP) (---) wird von LOOP kompiliert. Zählt den Schleifenindex um eins hoch. Die Schleife wird beendet, wenn der Index die Grenze zwischen Limit-1 und Limit überschreitet.

(MASS!) (---) initialisiert den Massenspeicherbetrieb (Diskette).

(NUMBER) (adr --- d f) wandelt den bei Adresse adr liegenden String in die doppelt genaue Zahl d um. Ist diese Umwandlung erfolgreich, ist das Flag TRUE. Vorangestelltes Minuszeichen und Dezimalpunkt sind im String erlaubt, der String muß mit einem Leerzeichen enden. Siehe DPL.

(QUIT) (---) Interpreterhauptschleife. Fragt die Tastatur ab und interpretiert die Eingabe.

(R/W) (adr n f ---) Massenspeicherroutine (schreibt oder liest von der Ramdisk bzw. Diskette). adr zeigt auf die Adresse des Buffers, n ist die Nummer des angewählten Blocks, f ist 0 zum Schreiben, andere Werte bewirken Lesebetrieb.

(SOURCE) (--- str) übergibt Speicheradresse und Länge der Tastatureingabe.

(TELL) (str ---) gibt String auf dem Bildschirm aus.

(VOCAB) (---) Runtimecode eines Vokabularwortes, z.B.: EDITOR

* (n1 n2 --- n3) multipliziert n1 mit n2, n3 ist das Produkt.

*/ (n1 n2 n3 --- n4) n4 ist das Resultat der Berechnung n1*n2/n3. Das Zwischenergebnis n1*n2 ist zur Erhöhung der Rechengenauigkeit doppelt genau.

*/MOD (n1 n2 n3 --- n4 n5) selbe Funktion wie */ , n5 ist das Resultat, zusätzlich wird der Rest der Division n4 übergeben.

+ (n1 n2 --- n3) addiert n1 und n2 zur Summe n3.

+! (n adr ---) addiert n zu dem Wert in Adresse adr.

+BUF (adr1 --- adr2 f) errechnet aus der Massenspeicherbufferadresse adr1 die Adresse adr2 des folgenden Buffers. Das Flag f ist 0, wenn der Inhalt der Variable PREV gleich adr2 ist.

+LOOP (n ---) beendet Schleife, die mit DO begonnen wurde. Erwartet auf dem Stack den Wert n, mit dem der Schleifenindex hochgezählt wird. Kompiliert (+LOOP). IMMEDIATE.

, (n ---) trägt die Zahl n ins Wörterbuch ein, erhöht die Variable DP um 2.

," (---) trägt String aus dem Inputstream ins Wörterbuch ein.

- (n1 n2 --- n3) subtrahiert n2 von n1, n3 ist die Differenz.

-DISC (adr f1 --- f2) greift auf die Diskette zu. adr ist die Adresse im Speicher, ab der gelesen, bzw. von der geschrieben wird. f1 ist 0 für Schreibbetrieb, andere Werte bedeuten Lesebetrieb. f2 ist der Errorcode des Betriebssysytems. SEC# enthält die Nummer des betroffenen Sektors.

-ROT (n1 n2 n3 --- n3 n1 n2) rotiert die obersten drei Stackwerte.

-TRAILING (str --- str) die Lange eines Strings wird um die am Stringende enthaltenen Leerzeichen gekürzt.

. (n ---) die Zahl n wird in der aktuellen Zahlenbasis ausgegeben.

." (---) druckt Text aus, im Gegensatz zu F.I.G.-Forth nur während der Kompilation anzuwenden. IMMEDIATE. Begrenzer ist " . Gebrauch: ." TEXT" . Runtimecode ist (.")

.((---) druckt Text aus. Begrenzer ist) . IMMEDIATE. Gebrauch: .(TEXT)

.ID (n ---) der Name eines Wortes wird ausgedruckt, n ist die Nfa dieses Wortes.

.LINE (n1 n2 ---) die Textzeile n1 im Screen n2 wird ausgedruckt.

.S (---) der momentane Stackinhalt wird ausgedruckt.

.BASE (---) gibt die momentane Zahlenbasis aus.

/ (n1 n2 --- n3) n3 ist der Quotient der Division von n1/n2. Siehe Abschnitt 'Floored Division'.

/MOD (n1 n2 --- n3 n4) n4 ist der Quotient der Division von n1/n2, n3 der Rest. Siehe Abschnitt 'Floored Division'.

/STRING (str1 n --- str2) liefert Teilstring str2, n ist der Anfang des neuen Strings.

0 (--- n) Konstante 0

0< (n --- f) Vergleich. f ist TRUE, wenn n kleiner 0 ist.

0<> (n --- f) Vergleich. f ist TRUE, wenn n ungleich 0 ist.

0= (n --- f) Vergleich. f ist TRUE, wenn n gleich 0 ist.

0> (n --- f) Vergleich. f ist TRUE, wenn n größer 0 ist.

1 (--- n) Konstante 1

1+ (n1 --- n2) n2 ist die Summe von n1+1.

1- (n1 --- n2) n2 ist die Differenz von n1-1.

2 (--- n) Konstante 2

2! (d adr ---) die doppelt genaue Zahl d wird ab Adresse adr gespeichert (4 Bytes).

2* (n --- 2n) n wird mit 2 multipliziert.

2+ (n1 --- n2) n2 ist die Summe von n1+2.

2- (n1 --- n2) n2 ist die Differenz von n1-2.

2/ (n1 --- n2) n2 ist der Quotient von n1/2.

2@ (adr --- d) Die doppelt genaue Zahl d wird von Adresse adr auf den Stack geholt.

2DROP (d ---) d wird vom Stack entfernt.

2DUP (d --- d d) d wird dupliziert.

2OVER (d1 d2 --- d1 d2 d1) wie OVER, für doppelt genaue Zahlen.

2SWAP (d1 d2 --- d2 d1) wechselt die obersten beiden doppelt genauen Zahlen aus.

3 (--- n) Konstante 3

: (---) eröffnet Definition eines Wortes (einer Colondefinition), erwartet einen Namen im Inputstream, legt Link-, Namens- und Codefeld an. Abschluss der Definition mit ;

; (---) schließt Definitionen ab, kompiliert UNNEST. IMMEDIATE.

;CODE (---) ruft Assembler auf. Wird zum Erstellen von Definitionsworten benutzt. IMMEDIATE. Siehe Beispiel im Anhang.

< (n1 n2 --- f) Vergleich. Das Flag f ist TRUE, wenn n1<n2.

<# (---) initialisiert die Umwandlung einer Zahl in einen String.

<> (n1 n2 --- f) Flag ist TRUE, wenn n1 ungleich n2 ist.

<MARK (--- f adr) legt TRUE und HERE auf den Stack. 'Markiert' Rückwärtssprung für Kontrollstrukturworte wie BEGIN UNTIL etc.

<RESOLVE (f adr ---) trägt von <MARK auf dem Stack gelegte Adresse ins Wörterbuch ein.

<SHIFT (n1 n2 --- n3) die Zahl n1 wird um n2 Bits nach links geschoben.

= (n1 n2 --- f) Vergleich. Das Flag f ist TRUE, wenn n1 gleich n2 ist.

> (n1 n2 --- f) Vergleich. Das Flag f ist TRUE, wenn n1>n2.

>BODY (cfa --- pfa) wandelt Codefeldadresse in Parameterfeldadresse.

>IN (--- adr) Variable. Enthält Zeiger in den Eingabetext.

>LINK (cfa --- lfa) wandelt Codefeldadresse in Linkfeldadresse.

>MARK (--- f adr) 'Markiert' Vorwärtssprung für Kontrollstrukturworte.

>NAME (cfa --- nfa) wandelt Codefeldadresse in Namensfeldadresse.

>R (n ---) legt n auf den Returnstack.

>RESOLVE (f adr ---) schreibt HERE in die von >MARK 'markierte' Adresse.

>SHIFT (n1 n2 --- n3) schiebt n1 um n2 Bits nach rechts.

? (adr ---) gibt den Inhalt von adr aus.

?BLOCK (f ---) wenn f TRUE ist, wird die Fehlermeldung "BAD BLOCK #" ausgegeben.

?BRANCH (f ---) führt bedingten Sprung auf die nach ?BRANCH im Wörterbuch abgelegte Adresse aus.

?CSP (---) gibt Fehlermeldung aus, wenn der Stackpointer vom Wert in der Variablen CSP abweicht.

?COMPILE (---) gibt Fehlermeldung aus, wenn sich das System nicht im Compilemodus befindet.

?CONDITION (f ---) alle von -1 abweichenden Werte für f geben die Fehlermeldung "CONDITIONALS WRONG" aus.

?DO (n1 n2 ---) Beginn einer Schleife, die mit LOOP oder +LOOP abgeschlossen wird. Kompiliert (?DO). n1 ist das Limit, n2 der Startindex der Schleife. IMMEDIATE. Siehe (?DO).

?DUP (n --- n n) (0 --- 0) dupliziert den obersten Stackwert, wenn er ungleich 0 ist.

?EXEC (---) gibt Fehlermeldung aus, wenn sich das System nicht im interpretativen Zustand befindet.

?MISSING (f ---) falls f gleich TRUE ist, wird (ABORT") aufgerufen und damit ein Fragezeichen ausgegeben.

?PAIRS (n1 n2 ---) Fehlermeldung, falls n1 ungleich n2 ist.

?PRINTABLE (c --- f) f ist TRUE, wenn c kein Sonderzeichen und damit normal ausdruckbar ist.

?STACK (---) Fehlermeldung, falls der Stack seinen Bereich überschritten hat.

?TERMINAL (--- n) fragt die Option-, Select- und Starttaste ab.

?UNIQUE (adr ---) der ab Adresse adr liegende String und "ISN'T UNIQUE" wird ausgegeben.

@ (adr --- n) die bei der Adresse adr liegende Zahl wird auf den Stack gelegt.

ABORT (---) Initilisierung beider Stacks und Sprung zum äußeren Interpreter QUIT.

ABORT" (---) Gebrauch: ABORT" FEHLERMELDUNG" kompiliert (ABORT"). IMMEDIATE. Siehe (ABORT")

ABS (n1 --- n2) n2 ist der Absolutwert der Zahl n1.

AGAIN (---) wird im Zusammenhang mit BEGIN gebraucht. Kompiliert BRANCH und absolute Sprungadresse. IMMIDIATE. Stackdiagramm während der Kompilation (f adr ---)

ALLOT (n ---) zu der Variablen DP, die den aktuellen Zeiger auf das Ende des Wörterbuches enthält, wird n addiert.

ALONE (---) im Kern ohne Funktion. DEFER-Wort, ist bei ausgebautem System durch ONLY ersetzt.

ALSO (---) schiebt das transiente Vokabular um eine Position in der Suchordnung nach unten.

AND (n1 n2 --- n3) n3 ist das Ergbenis der bitweisen UND-Verknüpfung von n1 und n2.

ASCII (--- b) interpretativ: ASCII A legt den Ascii-Wert von A auf den Stack. In der Kompilation kompiliert der Ausdruck 'ASCII A' den Asciiwert als Zahl. IMMEDIATE.

B/BUF (--- n) Konstante, Bytes/Buffer, Anzahl der Bytes pro Massenspeicherbuffer.

B/SCR (--- n) Konstante, Blocks/Screen, Anzahl der Blocks pro Screen.

BACKSPACE (---) gibt ein Backspace auf dem Bildschirm aus.

BASE (---) Variable, enthält die aktuelle Zahlenbasis.

BEGIN (---) Kontrollstrukturwort. IMMEDIATE. Gebrauch wie BEGIN ... AGAIN, BEGIN ... UNTIL und BEGIN ... WHILE ... REPEAT.

BELL (---) gibt Klingelton aus.

BL (--- n) Konstante, ein Leerzeichen (Blank) wird auf den Stack gelegt.

BLANK (adr n ---) füllt den Adressbereich adr mit n Leerzeichen (hex 20).

BLK (--- adr) Variable, enthält Zeiger auf den Eingabetext (Inputstream). Wenn der Inhalt gleich 0 ist, kommt die Eingabe von der Tastatur, ansonsten enthält BLK die Nummer des Massenspeicherblocks, auf den gerade zugegriffen wird.

BLOCK (n --- adr) übergibt die dem Massenspeicherblock n zugewiesene Bufferspeicheradresse. Falls der adressierte Block nicht im Buffer vorhanden ist, wird er vom Massenspeicher geladen. Ist der alte Buffer geändert (durch UPDATE), wird er in den Massenspeicher gerettet.

BODY> (pfa --- cfa) wandelt Parameterfeldadresse in Codefeldadresse.

BOOT (---) Initialisiert das Forthsystem so weit wie möglich auf den Originalzustand (nach dem Booten) zurück. Initialisiert die RAM-LINK und VOC-LINK Listen.

BOUNDS (str --- n1 n2) wandelt str (adr len) in passende Form für DO ... LOOP. n1 wird Limit, n2 wird Startindex.

BRANCH (---) Runtimecode für unbedingten Sprung. Die im

Wörterbuch nach BRANCH kompilierte Adresse wird in den IP geladen.

BUFFER (n --- adr) übergibt die Adresse eines freien Massenspeicherbuffers und weist diesem Buffer die Blocknummer n zu. Ist der alte Inhalt des Buffers geändert, wird er gerettet.

C! (n adr ---) die Zahl n (8 Bit) wird in die Adresse adr gespeichert.

C, (n ---) die Zahl n (8 Bit) wird in das nächste freie Byte im Wörterbuch eingetragen. DP wird um eins erhöht.

C@ (adr --- n) liest aus der Adresse adr deren Inhalt (n).

CAPS (--- adr) Variable. Nach CAPS ON werden alle eingegebenen Kleinbuchstaben in Großbuchstaben gewandelt. CAPS OFF schaltet diese Betriebsart wieder aus.

C/L (--- n) Konstante, liefert die Anzahl der Zeichen pro Screenzeile.

CLIP (adr1 adr2 ---) 'vergißt' Wörter eines Vokabulars unterhalb von adr1. adr2 ist der Zeiger auf einen Vokabularnamen.

CLS (---) löscht Bildschirminhalt.

CMOVE (adr1 adr2 n ---) kopiert n Bytes von adr1 nach adr2, niedrige Adressen zuerst.

CMOVE> (adr1 adr2 n ---) kopiert n Bytes von adr1 nach adr2, höhere Adressen zuerst.

CODE (---) IMMEDIATE. siehe Abschnitt ASSEMBLER.

COMBINE (n1 n2 --- n3) kombiniert n3 aus dem niedrigwertigen Byte n1 und dem höherwertigen Byte n2.

COMPILE (---) Beispiel: : TEST COMPILE DROP ; Bei Ausführung des Wortes TEST wird DROP ins Wörterbuch kompiliert.

CONSTANT (n ---) Definitionswort. Z.B.: 1024 CONSTANT 1K bildet eine Konstante mit dem Wert 1024.

CONTEXT (--- adr) Variable, zeigt auf das Transientvokabular.

CONVERT (ud1 adr1 --- ud2 adr2) wandelt die Zeichenfolge ab adr1+1 in eine doppelt genaue Zahl ud2 um, addiert das Ergebnis zu ud1. adr2 ist die Adresse des ersten, nicht wandelbaren Zeichens.

COUNT (adr1 --- adr2 n) erzeugt aus dem String, der ab adr1 im Speicher liegt, die Adresse adr2 und die Stringlänge n passend für TYPE.

CPACK (adr1 adr2 n ---) kopiert eine Zeichenfolge von adr1 nach adr2+1, legt bei adr2 das Längenbyte ab.

CR (---) bewegt den Cursor auf den Anfang der nächsten Zeile.

CRASH (---) warnt vor nicht initialisierten DEFER-Worten.

CREATE (---) CREATE XXX baut einen Wortkopf des Namens XXX auf, d.h. Link, Namens- und Codefeld, aber kein Parameterfeld. Ausführung von XXX legt pfa auf den Stack.

CRESET (n adr ---) löscht mit Hilfe der 8-Bit Maske n die Bits in adr.

CROP (adr ---) alle über adr liegenden, durch RAM-LINK verketteten Worte werden aus dieser Liste entfernt. RAM-LINK wird neu gesetzt. DEFER-Worte werden, falls sie auf eine Adresse höher adr zeigen, reinitialisiert.

CSET (n adr ---) setzt mit Hilfe der 8-Bit Maske n die Bits in adr.

CSP (--- adr) Variable für die Stackpointerposition.

CTEM (--- adr) Zwischenspeicher für CTO"

CTO" (b -- str) wandelt ein Zeichen b in einen String.

CTOGGLE (n adr ---) 'exort' den Inhalt der Adresse adr mit der 8-Bit Maske n.

CURRENT (--- adr) Variable, zeigt auf das Currentvokabular.

D+ (d1 d2 --- d3) d3 ist die Summe aus d1+d2.

D- (d1 d2 --- d3) d3 ist die Differenz aus d1-d2.

D. (d ---) gibt die Zahl d auf dem Bildschirm aus.

D.R (d n ---) gibt die Zahl d rechtsbündig in einem Feld der Weite n aus.

D0= (d --- f) f ist wahr, wenn d=0 ist.

D2/ (d1 --- d2) d2 ist der Quotient aus d1/2.

D< (d1 d2 --- f) f ist wahr, wenn d1<d2.

D> (d1 d2 --- f) f ist wahr, wenn d1>d2.

DABS (d1 --- d2) d2 ist der absolute Wert von d1.

DECIMAL (---) schaltet auf dezimale Zahlenbasis um.

DCX (---) Alias für DECIMAL.

DEFER (---) Beispiel: DEFER TEST definiert ein Wort TEST, dessen Runtimeaktivität noch nicht festgelegt ist. TEST wird initialisiert z.B.: durch ' NOOP IS TEST. Wenn TEST nicht initialisiert wurde, wird bei Ausführung von TEST das Wort CRASH aufgerufen. Alle DEFER-Worte sind durch RAM-LINK miteinander verkettet.

DEFINED (--- adr f) testet, ob das folgende Wort im Inputstream definiert ist. Wenn ja, ist adr die cfa des gefundenen Wortes, und f ist TRUE. Wenn nein, ist adr die Adresse, wo der Text im Speicher liegt, und f ist FALSE.

DEFINITIONS (---) kopiert den Inhalt von CONTEXT nach CURRENT.

DENSITY (n ---) erwartet durch SINGLE oder DOUBLE auf den Stack gelegten Wert. Siehe FORMAT.

DEPTH (--- n) liefert Anzahl der Stackwerte vor der Ausführung von DEPTH.

DIGIT (c n1 --- n2 f) wenn das Asciizeichen c in der Zahlenbasis n1 gültig ist, ist f TRUE und n2 ist der Zahlenwert. Andernfalls wird nur ein FALSE-Flag übergeben.

DISKID (---) siehe Abschnitt TAPE-Vokabular.

DLITERAL (d ---) erwartet während der Kompilation d auf dem Stack und trägt diese Zahl ins Wörterbuch ein. Bei Runtime wird d wieder auf den Stack gelegt. IMMEDIATE.

DMAX (d1 d2 --- d3) liefert das Maximum von d1 und d2.

DMIN (d1 d2 --- d3) liefert das Minimum von d1 und d2.

DNEGATE (d --- d) negiert d.

DO (n1 n2 ---) mit LOOP bzw. +LOOP benutzt. Erwartet n1 (Schleifenlimit) und n2 (Startindex) auf dem Stack. IMMEDIATE. Siehe (LOOP) und (+LOOP).

DOES> (----) mit CREATE bei der Bildung von Definitionsworten benutzt. Nach DOES> steht in der Definition des Definitionswortes der Runtimeteil der Worte, die mit dem Definitionswort definiert werden. IMMEDIATE. Siehe Anhang, Beispiel DEMO DEFINING WORDS.

DOUBLE (--- n) liefert Flag für DENSITY. Siehe FORMAT.

DP (--- adr) Variable. In dieser Variablen befindet sich die Adresse des ersten freien Bytes nach dem Ende des Wörterbuches.

DPL (--- adr) enthält nach der Eingabe von doppelt genauen Zahlen die Position des Dezimalpunktes.

DR# (--- adr) Variable, enthält Nummer der angewählten Diskettenstation.

DR1 und DR2 (---) schalten zwischen Diskettenstation 1 und 2 um.

DROP (n ---) löscht obersten Stackwert.

DUMP (adr n ---) liefert Speicherausdruck, n Bytes ab Adresse adr.

DUP (n1 --- n1 n1) dupliziert obersten Stackwert.

ECHO (str ---) DEFER-Wort, kann beispielsweise für eine Druckerausgabe mit einer eigenen Routine belegt werden. Default ist 2DROP.

ELSE (---) mit IF und THEN benutztes Kontrollstrukturwort. IMMEDIATE.

EMIT (c ---) gibt das Asciizeichen c auf dem Bildschirm aus.

EMPTY-BUFFERS (---) löscht die Massenspeicherbuffer.

ENDLINE (--- str) liefert String für TELL oder TYPE.

ENVIRON (---) DEFER-Wort. Nach dem Booten mit NOOP initialisiert.

ERASE (adr n ---) löscht n Bytes ab adr.

ERRAT (--- adr) enthält nach einem Fehler die Position im Inputstream, wo der Fehler auftrat. Wenn der Fehler bei der Kompilation vom Massenspeicher auftrat, enthält ERRAT + 2 die Nummer des Massenspeicherblocks.

ERROR (---) DEFER-Wort. Siehe (ERROR).

EOF? (--- f) f ist TRUE, wenn bei der letzten Tastaturbetätigung CTRL und 3 gedrückt wurde.

EOL-C# (--- b) liefert das CR-Zeichen.

EXECUTE (cfa ---) führt das Wort aus, dessen cfa auf dem Stack liegt.

EXIT (---) bricht die Ausführung eines Wortes ab, kehrt in die rufende Ebene zurück.

EXPECT (adr n ---) fragt von der Tastatur n Zeichen ab und kopiert sie nach adr. Bricht bei <RET> oder EOF ab.

EXPAND-BUFFERS (---) stellt System auf 1024 Bytes/Screen um. Siehe Abschnitt 'Kernal und Arbeitsversion'.

EXPOUND (---) DEFER-Wort. Siehe (EXPOUND).

FALSE (--- f) liefert Falsch-Flag (0).

FCB (--- adr) Zeiger auf aktuellen Werte von >IN und BLK.

FENCE (--- adr) Variable. Worte unterhalb von FENCE sind vor FORGET geschützt.

FILL (adr n b ---) füllt den Speicher mit n Bytes des Wertes b.

FIND (adr1 --- adr2 f) durchsucht die Vokabulare entsprechend der Suchordnung nach dem String, der im Speicher ab adr1 liegt. Falls der String nicht gefunden wurde, ist adr2=adr1 und f ist FALSE. Falls er gefunden wurde, ist adr2 die cfa des Wortes. Wenn das Wort IMMEDIATE ist, ist f=1, andernfalls ist f=-1 (TRUE).

FIRST (--- adr) Konstante, liefert die Adresse des ersten Massenspeicherbuffers.

FLIP (n --- n) tauscht Low- und Highbyte eines Wertes aus.

FLUSH (---) rettet alle geänderten Massenspeicherbuffer in den Massenspeicher (Ramdisk bzw. Diskette).

FORGET (---) FORGET XXX 'vergißt' XXX und alle nach XXX definierten Worte. Benutzt zuerst das CURRENT-Vokabular !

FORMAT (---) fragt nach gewünschter Diskettenstation und formatiert die Diskette. Voreinstellung ist einfache Dichte. Für die 1050er Diskettenstation kann man den Formatiervorgang auf 1.5 fache Dichte mit DOUBLE DENSITY umstellen. Zurück mit SINGLE DENSITY.

FORTH (---) Vokabularname. Name des Sprachkerns.

FORTH-83 (---) zeigt an, daß alle Worte des 83er-Standards vorhanden sind.

GETLINE (adr n1 --- str) oder (adr n1 --- f) liest eine n1 Zeichen lange neue Zeile Text (über GETREC) nach adr ein, übergibt str oder FALSE-Flag.

GETREC (---) DEFER-Wort. Siehe LISTEN.

GO (adr ---) Prozessor springt zum Maschinencode bei adr.

HEADER (---) DEFER-Wort. Siehe (HEADER).

HELLO (---) Systemmeldung. Runtime von MAIN.

HERE (--- adr) erstes freies Byte nach dem Wörterbuchende.

HEX (---) schaltet auf hexadezimale Zahlenbasis um.

HIDE (---) die letzte Definition wird nicht gefunden. Siehe REVEAL.

HLD (--- adr) Variable, bei der Umwandlung einer Zahl in einen String benutzt.

HOLD (c ---) fügt ein Asciizeichen in einen Ziffernstring ein.

I (--- n) legt in einer DO ... LOOP oder DO ... +LOOP Schleife den Index auf den Stack.

IF (f ---) Kontrollstrukturwort. IMMEDIATE. Wertet f aus,wird mit ELSE und THEN verwendet. Z.B.: IF ... THEN oder IF ... ELSE ... THEN.

IMMEDIATE (---) verändert das zuletzt definierte Wort so, daß es während der Kompilation nicht kompiliert, sondern ausgeführt wird.

INDEX (n1 n2 ---) gibt beim Diskettensystem die Kopfzeilen der Screens n1 bis n2 aus. Beim Kassettensystem werden keine Parameter auf dem Stack verlangt, es werden die Kopfzeilen der momentan aktiven Screens ausgegeben.

INTERPRET (---) exekutiert Worte oder legt Zahlen auf den Stack. Siehe EXPOUND und QUIT.

IS (cfa ---) legt die Runtimeaktion von DEFER-Worten fest. IMMEDIATE. Z.B.: ' (QUIT) IS QUIT.

J (--- n) liefert in zwei verschachtelten Schleifen den äußeren Schleifenindex.

KEY (---) DEFER-Wort. Siehe (KEY).

KEY? (---) DEFER-WORT. Siehe (KEY?).

KFCB (--- adr) Inputstreamfeld. Der Vector FCB zeigt auf KFCB. >IN zeigt auf FCB (KFCB), BLK zeigt auf FCB 2+ (KFCB 2+).

KILL (---) Kurzwort für EMPTY-BUFFERS.

L>NAME (lfa --- nfa) wandelt Linkfeldadresse in Namensfeldadresse.

LABEL (---) IMMEDIATE. Siehe Abschnitt Assembler.

LAST (--- adr) Variable, zeigt auf die nfa des zuletzt definierten Wortes.

LEAVE (---) kompiliert (LEAVE). IMMEDIATE. Siehe (LEAVE).

LEVEL (---) DEFER-Wort, initialisiert Massenspeicher. Siehe (MASS!).

LIMIT (--- adr) Konstante, liefert das erste freie Byte über den Massenspeicherbuffern.

LINK> (lfa --- cfa) wandelt Linkfeldadresse in Codefeldadresse.

LIST (n ---) gibt den Screen mit der Nummer n auf dem Bildschirm aus.

LISTEN (adr n1 --- n2 f) fragt die Tastatur nach n1 Zeichen ab, kopiert den Text nach adr aufwärts. Abbruch durch Returntaste oder EOF. n2 ist die Anzahl der gültigen Zeichen. f ist, bis auf EOF, TRUE. Die Eingaberoutine reagiert auf Back Space und Zeilenlöschen (Shift Delete).

LITERAL (n ---) während der Kompilation wird (LIT) und die Zahl n ins Wörterbuch kompiliert. IMEDIATE. Siehe (LIT).

LOAD (n ---) Kompiliere den Quelltext im Screen Nummer n. Siehe THRU.

LOOP (---) Kompiliert (LOOP). IMMEDIATE. Siehe DO und (LOOP).

M* (n1 n2 --- d) d ist das doppelt genaue Produkt der einfach genauen Zahlen n1 und n2.

M/ (d n1 --- n2 n3) n3 ist der Quotient von d/n1, n2 der Rest.

MAIN (---) DEFER-Wort. Siehe HELLO.

MAX (n1 n2 --- n3) n3 ist das Maximum der Zahlen n1 und n2.

MIN (n1 n2 --- n3) n3 ist das Minimum der Zahlen n1 und n2.

MOD (n1 n2 --- n3) n3 ist der Rest der Division n1/n2.

MYSELF (---) während der Kompilation angewandt, wird die cfa des gerade entstehenden Wortes kompiliert. Dadurch sind sich selbst aufrufende Worte möglich. IMMEDIATE.

N>LINK (nfa --- lfa) wandelt Namensfeldadresse in Linkfeldadresse.

NAME> (nfa --- cfa) wandelt Namensfeldadresse in Codefeldadresse.

NEGATE (n1 --- n2) n2 ist das Zweierkomplement von n1.

NEWLINE (---) stellt Cursor auf neuen Zeilenanfang.

NIP (n1 n2 --- n2) führt ein SWAP DROP aus.

NOOP (---) tut nichts. Platzhalterfunktion.

NOT (n1 --- n2) n2 ist das Einerkomplement von n1.

NUMBER (---) DEFER-Wort. Siehe (NUMBER).

OFF (adr ---) setzt den Inhalt von adr auf FALSE (0). Siehe ON.

OFFSET (--- adr) Variable, der Inhalt von adr wird von BLOCK und BUFFER zur aktuellen Blocknummer addiert.

ON (adr ---) setzt den Inhalt von adr auf TRUE (-1). Siehe OFF.

ONLY (---) Vokabularwort. Siehe Abschnitt Vokabular und Suchordnung.

OR (n1 n2 --- n3) n3 ist das Resultat der ODER-Verknüpfung von n1 und n2.

ORG (--- adr) Konstante, ab hier beginnt der FORTH-Kern.

OVER (n1 n2 --- n1 n2 n1) der zweite Stackwert wird auf den obersten Stackplatz kopiert.

PAD (--- adr) Textzwischenspeicher, befindet sich 80 Bytes über HERE.

PARSE (c --- str) durchsucht den Inputstream nach dem Asciibegrenzer c, übergibt str, justiert >IN.

PERFORM (adr ---) wie @ EXECUTE. adr muss auf eine cfa zeigen.

PICK (n1 ---n2) n2 ist eine Kopie des n1ten Stackeintrages. 0 PICK ist gleich DUP, 1 PICK ist gleich OVER etc.

PREV (--- adr) Variable, ihr Inhalt zeigt auf die Adresse des zuletzt benutzten Massenspeicherbuffers.

PUTREC (---) DEFER-Wort. Ausgaberoutine. Siehe TELL und (TELL).

QUERY (---) liest maximal 80 Zeichen in den Tastaturbuffer (Terminal Input Buffer).

QUIT (---) DEFER-Wort. Siehe (QUIT).

R/W (adr n f ---) Massenspeicherwort. adr ist die Adresse, ab der die Daten geschrieben bzw. gelesen werden, n die Nummer des dazugehörigen Blocks, f = 0 für Schreibbetrieb, andere Werte für f

bedeuten Lesebetrieb.

R> (--- n) entfernt die Zahl n vom Returnstack und legt sie auf den Parameterstack.

R@ (--- n) kopiert den obersten Returnstackwert auf den Parameterstack.

RAM-LINK (--- adr) Variable, verbindet alle DEFER-Worte und bestimmte Konstanten und Variablen miteinander. Wird benutzt, um diese Worte mit einem bestimmten Wert zu initialisieren.

REDIRECT (adr ---) das Codefeld des zuletzt definierten Wortes wird mit adr überschrieben.

RELOC (adr ---) Die Massenspeicherbuffer werden nach adr verschoben. Die Buffer werden gelöscht, FIRST und LIMIT werden danach justiert.

REPEAT (---) Kontrollstrukturwort. IMMEDIATE. Innerhalb von Wortdefinitionen wie folgt benutzt: BEGIN ... WHILE ... REPEAT.

RESET (---) DEFER-WORT. Siehe (INIT).

REVEAL (---) das mit HIDE 'versteckte' Wort läßt sich jetzt wieder auffinden. Siehe HIDE.

ROLL (n ---) der nte Stackwert wird auf den obersten Stackplatz kopiert, alle anderen wandern um eins nach unten.

ROT (n1 n2 n3 --- n2 n3 n1) die obersten Stackwerte werden rotiert, die unterste Zahl wandert nach oben.

RP! (n ---) der Returnstackpointer wird mit dem Wert n geladen.

RP@ (-- adr) der aktuelle Stand des Returnstackpointers wird auf den Parameterstack gebracht.

RP0 (--- adr) Konstante zum Initialisieren des Returnstacks.

S>D (n --- d) wandelt die vorzeichenbehaftete Zahl n in die doppelt genaue Zahl d um.

SAVE-BUFFERS (---) schreibt alle geänderten Buffer in die Ramdisk oder auf Diskette. Der Inhalt der Buffer bleibt im Gegensatz zu FLUSH erhalten. Siehe FLUSH.

SCAN (str1 c --- str2) tastet den String str1 nach dem Asciibegrenzer c ab. str2 ist der gekürzte String ab und mit dem Begrenzerzeichen. Die Stringlänge von str2 ist gleich 0, wenn str1 leer war oder keine Begrenzer enthält. Siehe SKIP.

SCR (--- adr) Variable, enthält die Nummer des zuletzt

geLlSTeten Screens.

SEC# (--- adr) Zeiger auf die Sektornummer für Diskettenoperationen.

SEC/BLK (--- n) Konstante, Anzahl der Diskettensektoren pro Block.

SIGN (n ---) wenn n negativ ist, wird bei der Umwandlung von einer Zahl in einen String (zwischen <# und #>) ein Minuszeichen in den String eingefügt.

SINGLE (--- n) liefert Wert für DENSITY. Siehe FORMAT.

SKIP (str1 c --- str2) tastet str1 nach dem Asciibegrenzer c ab. Liefert gekürzten String ohne den (voranstehenden) Begrenzer. Die Stringlänge von str2 ist gleich 0, wenn str1 leer war oder nur aus Begrenzern besteht. Siehe SCAN.

SOURCE (---) DEFER-Wort. Siehe (SOURCE).

SP! (n ---) der Parameterstackpointer wird mit dem Wert n geladen.

SP@ (--- n) n ist die Position des Parameterstackpointers vor dem Aufruf von SP@.

SPACE (---) gibt ein Leerzeichen auf dem Bildschirm aus.

SPACES (n ---) gibt n Leerzeichen auf dem Bildschirm aus.

SPAN (--- adr) Variable, enthält Anzahl der von EXPECT entgegengenommenen Zeichen.

SPLIT (n1 --- n2 n3) n2 ist das niedrigwertige, n3 das höherwertige Byte von n1.

SP0 (--- n) Konstante, übergibt Initialisierungswert für den Parameterstack.

STANDARD (---) setzt die Systemein- und Ausgabe auf die Standardworte.

STATE (--- adr) Variable, enthält 0, wenn interpretiert wird, ist ungleich 0, wenn kompiliert wird.

STATUS (---) DEFER-Wort, wird von STON und STOF manipuliert, befindet sich in der (QUIT)-Hauptschleife.

STLO (--- n) Konstante, liefert untersten legalen Stackpointerwert.

STOF (---) schaltet automatische Stackausgabe ab.

STON (---) schaltet automatische Stackausgabe an.

STOP (---) beendet die Kompilation eines Screens sofort. IMMEDIATE.

STR (--- adr) Variable, zeigt auf Zwischenspeicher für Strings. Siehe "

SWAP (n1 n2 --- n2 n1) Austausch der beiden oberen Stackwerte.

TAPEIO (---) siehe Abschnitt über das TAPE-Vokabular.

TELL (---) DEFER-Wort. Siehe (TELL).

THEN (---) Kontrollstrukturwort. IMMEDIATE. Wird mit IF und ELSE verwendet.

THRU (n1 n2 ---) Die Screens von n1 bis n2 werden nacheinander kompiliert.

TIB (--- adr) Konstante, liefert Adresse des Terminal Input Buffers (Texteingabe).

TRAVERSE a. (nfa 1 --- cfa-1)
b. (cfa-1 -1 --- nfa)
durchsucht das Namensfeld von Forthworten in Richtung auf- (a.) bzw. absteigender Adressen (b.).

TRUE (--- f) Konstante, übergibt den WERT -1 (Hex FFFF). Siehe FALSE.

TUCK (n1 n2 --- n2 n1 n2) Funktion von SWAP OVER.

TYPE (str ---) Haupt-Stringausdruck-routine. Verändert #OUT.

U. (u ---) gibt vorzeichenlose Zahl u auf dem Bildschirm aus.

U2/ (u1 --- u2) u2 ist das Resultat von u1/2.

U< (u1 u2 --- f) f ist TRUE, wenn u1 kleiner als u2 ist.

U> (u1 u2 --- f) f ist TRUE, wenn u1 größer als u2.

UM* (un1 un2 --- ud) multipliziert die vorzeichenlosen Zahlen n1 und n2, liefert das vorzeichenlose, doppelt genaue Produkt ud.

UM/MOD (ud u1 --- u2 u3) dividiert ud durch u1. u2 ist der Rest der Division, u3 der Quotient. Alle Zahlen sind vorzeichenlos. Bei einer Division durch 0 ist der Quotient gleich ♦, der Rest besteht aus dem unteren 16bit-Wort von ud. Siehe 'Floored Division'.

UMAX (u1 u2 --- u3) u3 ist das Maximum von u1 und u2.

UNNEST (---) wird am Ende einer Colondefinition ins Wörterbuch kompiliert. Bei der Ausführung wird in die rufende Ebene zurückgekehrt.

UNTIL (f ---) Kontrollstrukturwort. IMMEDIATE. Wird im Zusammenhang mit BEGIN verwendet. Kompiliert ?BRANCH. Wenn f True ist, wird das Programm mit dem Code nach UNTIL fortgesetzt, andernfalls wird zu dem mit BEGIN markierten Programmteil zurückgesprungen.

UPPER (str ---) wandelt String in Großbuchstaben.

UPDATE (---) markiert den zuletzt benutzten Block als geändert.

USE (--- adr) Variable, enthält die Adresse des ältesten Massenspeicherbuffers, derjenige, der als nächster benutzt wird.

USERR (---) DEFER-Wort. Für spezielle Anwender-Fehlerroutinen gedacht. Wird von (ABORT") aufgerufen.

VARIABLE (---) Definitionswort. VARIABLE XXX baut eine Variable mit dem Namen XXX auf. Wenn XXX aufgerufen wird, legt das Wort seine Parameterfeldadresse auf den Stack. Der Inhalt dieser Adresse wurde bei der Definition mit 0 initialisiert.

VOCABULARY (---) Bildet Vokabularworte wie EDITOR, ASSEMBLER etc. Aufruf: VOCABULARY XXX . XXX ist das neue Vokabularwort.

VOC-LINK (--- adr) Variable, verkettet alle Vokabularworte miteinander.

WARM (---) Warmstart des Systems. Beide Stacks werden initialisiert. Dann werden nacheinander RESET ENVIRON MAIN und QUIT aufgerufen.

WARN (---) DEFER-Wort. Siehe ?UNIQUE. Abschaltung durch: ' DROP IS WARN.

WHILE (f ---) Kontrollstrukturwort. IMMEDIATE. Wird mit BEGIN und REPEAT verwendet. Solange f TRUE ist, wird der Programmteil zwischen WHILE und REPEAT abgearbeitet.

WIDTH (--- adr) Variable. Enthält die maximale Anzahl von Zeichen in einem Namensfeld.

WORD (c --- adr) durchsucht den Inputstring nach dem Asciibegrenzer c. Liefert die Adresse eines Strings am Ende des Wörterbuches (HERE). Der String wird mit einem Leerzeichen abgeschlossen.

WORDS (---) gibt die Wörter des obersten Kontextvokabulars auf den Bildschirm aus.

XOR (n1 n2 --- n3) n3 ist das Resultat einer Exor-Verknüpfung

von n1 und n2.

[(---) stoppt die Kompilation, beginnt Interpretation.
IMMEDIATE.

['] (---) wird innerhalb einer Colondefinition gebraucht.
Kompiliert das nächste Wort im Inputstream als Zahl. IMMEDIATE.

[COMPILE] (---) erzwingt die Kompilation eines
IMMEDIATE-Wortes. IMMEDIATE.

\ (---) Kommentarzeichen. Rest der Zeile wird ignoriert.
IMMEDIATE.