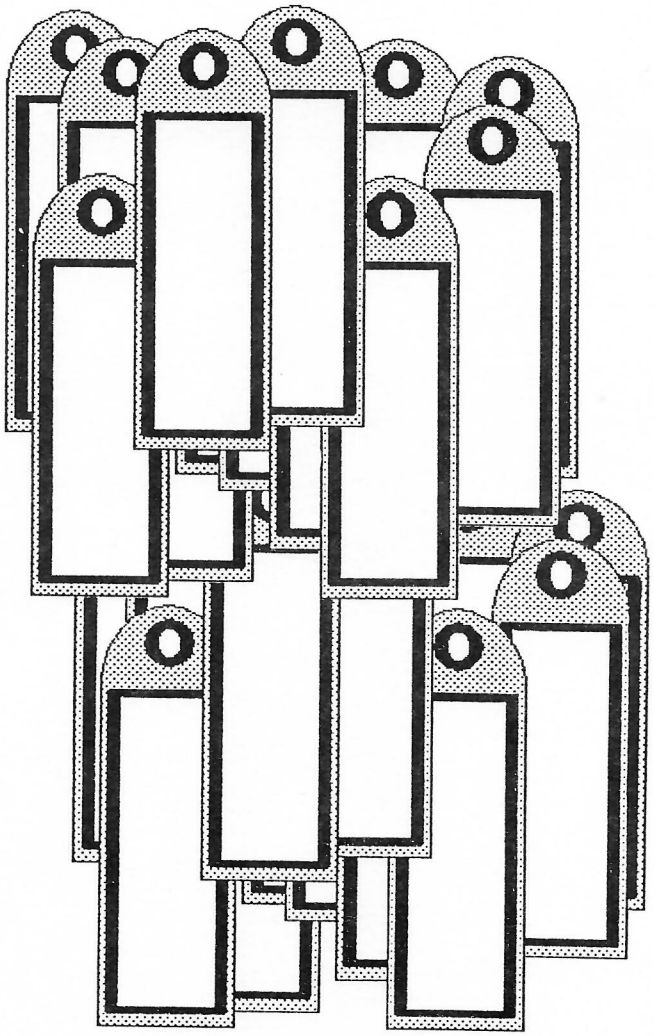


Anhang

Atari ST Fullscreen Editor

Ältere FORTH-Systeme enthalten meist Editoren, die diesen Namen höchstens zu einer Zeit verdient haben, als man noch die Bits einzeln mit Lochzange und Streifenleser an die Hardware übermitteln mußte. Demgegenüber bietet ein Editor, mit dem man 'im Blindflug' jeweils eine ganze Zeile bearbeiten kann, sicher schon einigen Komfort. Heute kann man jedoch mit solch einem Zeileneditor keinen Staat mehr machen und erst recht nicht mit anderen Sprachen konkurrieren.

Wir haben daher dem Atari ST einen komfortablen Fullscreen-Editor spendiert, der vollkommen in GEM eingebunden ist. Dies erspart uns auch umfangreiche Befehlslisten aller Editorfunktionen: Der Editor hat eine Menüzeile, und beim Anklicken eines Menüpunktes bekommt man eine kurze Erklärung der jeweiligen Funktion. Die 'Profis' können alle Editorfunktionen auch ohne Maus über die Tastatur erreichen, die entsprechenden Control-codes stehen hinter den Menüeinträgen. Auch die Hilfsfunktion ist abschaltbar, wenn man den Menüpunkt DAUERHILFE anwählt. Die Hilfstexte erscheinen aber ohnedies nicht, wenn man die Kommandos von der Tastatur aus eingibt.

Im Editorfenster wird immer ein FORTH-Screen - also 1024 Bytes - in der üblichen Aufteilung in 16 Zeilen mit je 64 Spalten dargestellt. Es gibt einen Zeichen- und einen Zeilenstack. Damit lassen sich Zeichen bzw. Zeilen innerhalb eines Screens oder auch zwischen zwei Screens bewegen oder kopieren. Dabei wird verhindert, daß versehentlich Text verloren geht, indem Funktionen nicht ausgeführt werden, wenn dadurch Zeichen nach unten oder zur Seite aus dem Bildschirm geschoben würden.

Der Editor unterstützt das 'Shadow-Konzept'. Zu jedem Quelltext-Screen gibt es einen Kommentar-Screen. Dieser erhöht die Lesbarkeit von FORTH-Programmen erheblich. (Sie wissen ja, guter FORTH-Stil ist selbstdokumentierend !) Auf Tastendruck stellt der Editor den Kommentar-Screen zur Verfügung. So können Kommentare 'deckungsgleich' angefertigt werden. Die meisten mitgelieferten Quelltexte sind übrigens mit Shadow-Screens versehen, und auch das Printer-Interface unterstützt dieses Konzept.

Um in den Editor zu gelangen gibt es drei Möglichkeiten:

```
<screennr> L
```

ruft den Screen mit Nummer screennr auf. Dabei muß das File vorher z.B. mit USE ausgewählt sein.

```
V
```

ruft den zuletzt bearbeiteten Screen wieder auf. Dies ist der zuletzt editierte oder aber, und das ist sehr hilfreich, derjenige, der einen Abbruch beim Kompilieren verursacht hat. Wenn Sie also ein File kompilieren, und der Compiler bricht mit

einer Fehlermeldung ab, brauchen Sie nur ein V einzugeben. Der fehlerhafte Screen wird in den Editor geladen, und der Cursor steht hinter dem Wort, das den Abbruch verursacht hat.

Häufig möchte man sich die Definition eines Wortes ansehen, um z.B. den Stackkommentar oder die genaue Arbeitsweise nachzulesen. Dafür gibt es das Kommando

```
VIEW <wort>
```

Damit wird der Screen - und natürlich auch das File - aufgerufen, auf dem wort definiert wurde. Dieses Verfahren ersetzt (fast) einen Decompiler, weil es natürlich sehr viel bequemer ist und Ihnen ja auch sämtliche Quelltexte des Systems zur Verfügung stehen. Natürlich müssen die entsprechenden Files auf den Laufwerken 'griffbereit' sein, sonst erscheint eine Fehlermeldung. Die VIEW-Funktion steht auch innerhalb des Editors zur Verfügung, man kann dann mit einem Tastendruck zwischen dem gerade bearbeiteten und dem Screen, auf dem man eine Definition gesucht hat, hin- und herschalten. Dies ist insbesondere nützlich, wenn man eine Definition aus einem anderen File übernehmen möchte oder nicht mehr sicher ist, wie der Stackkommentar eines Wortes lautet oder ....

Noch ein paar Hinweise :

-) Wie Files erzeugt und verlängert werden, steht im Teil 1 des Handbuchs und wird weiter unten in dem Kapitel über das Fileinterface ausführlich erklärt.

-) Der Editor unterstützt nur das Kopieren von Zeilen. Man kann auf diese Art auch Screens kopieren, aber beim gelegentlich erforderlichen Einfügen von Screens in der Mitte eines Files ist das etwas mühselig. Zum Kopieren ganzer Screens innerhalb eines Files oder von einem File in ein anderes werden im volksFORTH83 die Worte COPY und CONVEY verwendet. Schauen Sie bitte deren genaue Funktion im Glossar unter "Massenspeicher" nach.
   Beispiel : Sie wollen in ihr File FIRST.SCR vor den Screen 4 einen weiteren Screen einfügen. Dazu geben Sie ein :

```
use first.scr          \ FIRST.SCR ist aktuelles File
1 more                 \ Verlängere um einen Screen
4 capacity 2- 5  convey
                       \ kopiere 4..."vorletzer" nach
                       \         5..."letzter"   Screen
```

   Anschließend sind die Screens 4 und 5 gleich und Sie können den Screen 4 löschen. Beachten Sie bitte, daß Sie bei Einfügen von Screens evtl. die Argumente von +THRU +LOAD THRU und LOAD ändern müssen!

Die GEM-Bibliothek des volksFORTH83

Diese volksFORTH83-Version enthält eine umfangreiche Bibliothek der GEM-Routinen. Diese Bibliothek gliedert sich in die Teile AES ("Application Environment System") und VDI ("Virtual Device Interface"). Im volksFORTH gehört auch ein Teil BASICS dazu, der die VDI und AES gemeinsamen Teile enthält. Ferner gibt es Files, die den Parameter-Konstanten der GEM-Routinen symbolische Namen zuordnen und eine "Super"-Bibliothek.

Die Namen der von GEM zur Verfügung gestellten Funktionen enstprechen völlig den Namen, die vom "C" des Entwicklungspakets her bekannt sind. Viele Präfixe haben wir jedoch weggelassen, da sie nur unnötig viel Platz verbrauchen würden. Die Funktionen benötigen zumeist auch dieselben Argumente, wobei wir uns bemüht haben, überflüssige Eingangswerte wegzulassen (z.B. bestimmte Handles). Manche Aufrufe liefern soviele Ausgangswerte, daß wir uns entschlossen haben, sie nicht alle auf den Stack zu packen. In diesem Fall muß man sie selbst aus dem enstprechenden Array holen. Wo, findet man ggf. in der Literatur. Die Eingangswerte stehen jedoch (mit Ausnahme von EVNT-MULTI) auf den Stack.

-) Wer Zugang zur GEM-Programmierung sucht, sollte sich die Literatur des Enwicklungspakets von Atari verschaffen. Dort sind (fast) alle Funktionen (viele fehlerhaft!) beschrieben. Die Qualität ist aber nicht besonders, z.T auch auf den IBM-PC abgestellt.

-) Empfehlenswert, wenn auch ausserordentlich knapp und daher nur als Nachschlagewerk geeignet, ist das Handbuch zum Megamax C-Compiler.

-) Einige Zeitschriften des Atari-Marktes beginnen Fortsetzungsreihen zur GEM-Programmierung, die aber bisher nicht empfehlenswert sind.

Für die GEM-Programmierung ist außerdem ein sog. Resource Construction Set erforderlich. Das ist ein Programm, mit dem die Menüleisten, Dialog- und Alertboxen sowie die Icons hergestellt werden. Das Programm erzeugt ein File, daß auf ".RSC" endet (das sog. Resourcefile) und ein ".H"-File, daß die Anbindung des Resourcefiles an "C"-Programme erlaubt. Dieses C-File enthält Konstantendefinitionen für jedes "Objekt" (jedes noch so kleine Teil eines Resourcefiles ist so ein Objekt), die analog auch in ihrem Forth-Programm auftreten müssen. Vergleichen Sie dazu bitte das File EDIICON.H (für "C") mit EDIICON.SCR, das dessen Übersetzung in Forth darstellt. Ein Resource Construction Set gehört sowohl zum Lieferumfang des Entwicklungspakets als auch zum Megamax-"C"-Paket.

Verzeichnis der Worte der GEM-Bibliothek

Die GEM-Bibliothek besteht aus den Files BASICS.SCR, VDI.SCR und AES.SCR. Die in diesen Files enthaltenen Worte werden im folgenden aufgeführt. Des weiteren soll sich eine Bibliothek von "Super-worten" entwickeln, die die Handhabung der GEM-Routinen vereinfacht. Diese Files sind über Shadowscreens dokumentiert und daher hier nicht aufgeführt.

BASICS.SCR

```
Vocabulary GEM   GEM definitions also
     \ Das Vokabular, das die GEM-Worte enthält.

Create intin         Create ptsin
Create intout        Create ptsout
Create addrin        Create addrout
     \    Diese Arrays werden für die diversen Parameter  benö-
          tigt.

Variable grhandle
     \    Diese Variable nimmt die Handle auf,  die von  OPNVWK
          geliefert wird.

Create contrl   $16 allot
contrl 2 gemconstant opcode
       2 gemconstant #intin
       2 gemconstant #intout      ' #intout Alias #ptsout
       2 gemconstant #addrin
       2 gemconstant #addrout
       2 gemconstant function
     \    Die  Komponenten  dieses  Arrays  tragen  Namen   und
          enthalten die Zahl der Parameter,  die in den betref-
          fenden Arrays (s.o.) übergeben werden.

Create global
Constant ap_ptree
     \    Dieses Element des Arrays GLOBAL enthält die  Adresse
          eines Baumes, die von RSRC_LOAD initialisiert wird.

Create AESpb
Create VDIpb
     \    Diese beiden Arrays enthalten Zeiger auf die  Arrays,
          die die Parameter enthalten...

Code array!   ( n0 ... nk-1 adr k --)
     \ Speichert k Werte ab Adresse adr in einem Array.
Code 4!       ( n1 .. n4 addr -- )
Code 4@       ( addr -- n1 .. n4 )
     \    Speichert  bzw.  liest 4 Werte ab Adresse  addr.  Sie
          werden dazu benutzt,  Rechtecke in die div. Arrays zu
          schreiben bzw. herauszuholen.
Code AES      ( opcode #intin #intout #addrin #addrout --
                intout@)
     \    Dieses Wort wickelt alle AES-Aufrufe ab.
Code VDI      ( opcode #ptsin #intin --)
     \    Dieses Wort wickelt alle VDI-Aufrufe ab.
```

```
: appl_init
     \    Dieses Wort initialisiert die Applikation und  sollte
          vor  dem ersten Aufruf einer AES-Funktion  aufgerufen
          werden.
: appl_exit
     \    Dieses Wort sollte am Ende einer Applikation aufgeru-
          fen werden.

Create sizes 8 allot
     \    Dieses Array enthält die Größe eines Buchstabens  und
          einer Box in Pixeln
Sizeconst c_width   Sizeconst c_height
Sizeconst b_width   Siezconst b_height
     \    Diese  Worte lesen die Felder in SIZES aus,  die  die
          Höhe und Breite eines Zeichens bzw. einer Box enthal-
          ten.
: graf_handle
     \    Liefert die VDI-Handle in der Variablen GRHANDLE  und
          die Buchstaben- bzw. Boxgröße in SIZES
: opnvwk
     \    Öffnet  eine "virtual workstation" und muß zu  Beginn
          einer Applikation aufgerufen werden.
: clrwk
     \    Löscht die Workstation.  Der Hintergrund wird in  der
          gewählten Farbe gemalt.
: clsvwk
     \    Schließt die "virtual workstation".  Muß vor  Beenden
          des Programms aufgerufen werden.
: updwk
     \    Update virtual workstation. Alle VDI-Kommandos werden
          zuende ausgeführt.
: s_clip      ( x1 y1 x2 y2 clipflag -- )
     \    Setze  Größe und Position des Clipping rectangle  für
          alle VDI-Aufrufe.

: grinit    appl_init graf_handle opnvwk ;
     \    Faßt alle benötigten Aufrufe am Anfang einer Applika-
          tion zusammen.
: grexit    clsvwk appl_exit   ;
     \    dto. für die Beendigung einer Applikation.
Variable c_flag
     \    gibt  an,  ob  die  Aufrufe  von  SHOW_C  und  HIDE_C
          akkumuliert werden sollen.
: show_c       ( -- )
     \    Schalte die Maus an.
: hide_c       ( -- )
     \    Schalte die Maus aus.
2Variable objc_tree
     \    Enthält  den Objektbaum,  auf das sich die  folgenden
          Worte beziehen :  MENU_BAR  MENU_ICHECK  MENU_IENABLE
          MENU_TNORMAL  MENU_TNEXT  FORM_DO  FORM_CENTER
```

AES.SCR

Event :

```
: evnt_keybd   ( -- key )
     \     Wartet  auf  einen Keyboard-event.  key  besteht  aus
           scan-code und ASCII-Wert.
: evnt_button  ( #clicks0 bmask bstate -- #clicks1 )
     \     Wartet  auf  einen button-event.  #clicks0  gibt  die
           Anzahl der Clicks an,  bmask und bstate geben die  zu
           drückende Taste an. #clicks1 ist die Zahl der Clicks.
: evnt_mouse   ( f leftX topY widht height -- )
     \     Wartet auf einen mouse-movement-event. f ist null für
           das Betreten und 1 für das Verlassen des durch  leftX
           topY width und height spezifizierten Rechtecks.
Create message
     \     Dieses Array ist der message-event-buffer.
: evnt_mesag   ( -- )
     \     Wartet auf einen message-event.
: evnt_timer   ( dtime -- )
     \     Wartet auf einen Timer-event.  dtime ist die (doppelt
           genaue) zu wartende Zeit in msec.
Create events
     \     Dieses Array enthält die Eingangsparameter des Wortes
           EVNT_MULTI. Der Inhalt dieses Feldes ist wie folgt :
           +0 : Art der erkannten Events :
                Bit 0     Keyboard
                Bit 1     Button
                Bit 2     Mouse rectangle 1
                Bit 3     Mouse rectangle 2
                Bit 4     Message
                Bit 5     Timer
           +2   #clicks0  Zahl der zum "Auslösen" erforderlichen
                          Mausklicks.
           +4   bmask     Bit 0 : linker, Bit 1 : rechter ist zu
                          betätigen.
           +6   bstate    Gibt  an,  ob der entsprechende  Knopf
                          von  bmask gedrückt oder  gelöst  sein
                          muß.
           +8   f  leftX topY width height,    gibt an,  ob  der
                Eintritt  (f=0)  oder  Austritt  (f=1)  aus  dem
                angegebenen Rechteck signalisiert werden soll.

           +18   ------- " ------------- ,  dto.
           +28  dtime     Zu wartende Zeit in msec.

: prepare
     \     Dieses  Wort  kopiert den Inhalt von  EVENTS  in  die
           enstprechenden  Parameterarrays  und  sollte  vor
           EVNT_MULTI aufgerufen werden.
: evnt_multi   ( -- which )
     \     Dieses  Wort wartet auf einen von mehreren  möglichen
           Events. Das Array EVENTS enthält die Parameter dieser
           Routine.  which  ist  die  Nummer  der  aufgetretenen
           Events nach dem Schlüssel wie für das erste Wort  von
           EVENTS.  Die  Ausgabeparameter müssen den  GEM-Arrays
           entnommen werden...
```

```
: evnt_dclick  ( dnew dgetset -- dspeed )
     \     Dieses Wort setzt oder liest die Zeitspanne für einen
           erkannten Mehrfachclick.  dgetset = 1  bedeutet,  daß
           dnew  als  neue Zeit gesetzt  wird.  dspeed  ist  die
           eingestellte Zeitspanne.
```

Menu :

```
: menu_bar     ( showflag -- )
     \     Löscht oder setzt die Menüleiste.
: menu_icheck  ( item showflag -- )
     \     Setzt oder löscht den Haken vor einem menu item
: menu_ienable ( item enableflag -- )
     \     Schaltet einen menu item ein oder aus. Ausgeschaltete
           menu items erscheinen in heller Schrift.
: menu_tnormal ( title normalflag -- )
     \     Stellt einen Menütitel normal oder invers dar.
: menu_text    ( item laddr -- )
     \     Ändert den Text eines menu item.  Die Länge darf sich
           aber nicht ändern !
: menu_register( apid laddr -- menuid )
     \     Installiert ein Desktop Accessory in der  Menüleiste.
           menuid ist die Position im Menü
```

Object

```
: objc_add     ( parent child -- )
     \     Fügt ein Objekt child an das Ende der child list  des
           Objektes parent an.
: objc_delete  ( object -- )
     \     Löscht ein Objekt vom Baum.
: objc_draw    ( startob depth x y width height -- )
     \     Zeichnet  den  Baum von startob mit der  Tiefe  depth
           neu.  x  y  width  und  height  geben  ein  Clipping-
           Rectangle an.
: objc_find    ( startob depth x y -- obnum )
     \     Sucht das Objekt unter der Mausposition x y . Liefert
           die Objektnummer oder -1.
: objc_offset  ( object -- x y )
     \     Liefert die Bildschirmpositon eines Objektes
: objc_order   ( object newpos -- )
     \     Bewegt ein Objekt von einer Stelle des Baums zu einer
           anderen.
: objc_edit    ( object char index kind -- newindex )
     \     Wird zum Editieren des Textes innerhalb eines  Objek-
           tes benutzt.
: objc_change  ( object x y width height newstate redraw -- )
     \     Ändert den Objektstatus und zeichnet das Objekt neu.
```

Form :

```
: form_do     ( startobj -- objectno )
     \     Diese  Routine  zeigt ein Objekt an und  erlaubt  dem
           Benutzer,  es zu ändern. objectno. ist die Nummer des
           exit button.
: form_dial   ( diflag lix liy liw lih  bix biy biw bih )
     \     Diese Routine besteht aus vier Routinen,  ihre  Funk-
           tion wird von diflag bestimmt:
                0    Reserviere einen Bildschirmbereich für  ein
                     Objekt.
                1    Male eine wachsende Box
                2    Male eine schrumpfende Box
                3    Gib   den  reservierten   Bildschirmbereich
                     frei.
: form_alert  ( defbttn 0string -- exbttn )
     \     Erlaubt  auf  einfache  Art,  Alert-Boxen  zu  bauen.
           defbttn ist der Defaultknopf,  0string ist ein  durch
           $00  begrenzter  String,  der das  Aussehen  der  Box
           bestimmt und exbttn ist der gedrückte Knopf.
: form_error  ( enum -- exbttn )
     \     Malt  eine Alertbox mit dem  Text  "TOS-Fehler-Nummer
           soundso"
: form_center ( -- x y width height )
     \     Rechne  die  Position eines in der  Mitte  des  Bild-
           schirms zentrierten Objektes aus.
```

Graphic :

```
: graf_dragbox      ( startx starty width height boundx  boundy
                    boundw boundh -- finishx finishy )
     \     Malt den Umriß einer Box,  die auf dem Bildschirm mit
           der  Maus bewegt werden kann.  Die Bewegung der  Maus
           wird durch ein äußeres Rechteck beschränkt.
: graf_movebox      ( sourcex sourcey width height destx  desty
                    --)
     \     Malt eine Box, die sich von source nach dest bewegt.
: graf_growbox      ( stx sty stw sth  fix fiy fiw fih -- )
     \     Malt eine expandierende Box.
: graf_shrinkbox    ( fix fiy fiw fih  stx sty stw sth -- )
     \     Malt eine schrumpfende Box.
: graf_watchbox     ( object instate outstate --
                     inside/outside )
     \     Überwacht  die  Bewegung  der  Maus  innerhalb  eines
           Objektes....
: graf_slidebox     ( parent object vhflag -- vhpos )
     \     Überwacht  das  Ziehen  einer kleinen  Box  in  einer
           großen Box mit der Maus....

2Variable mofaddr
     \     Zeiger auf eine selbst definierte Mausform
: graf_mouse        ( mouseform -- )
     \     Setzt die Mausform :
           0    Pfeil
           1    Cursor
           2    Biene
           3    Zeigefinger
           4    Hand
           5    dünnes Fadenkreuz
```

```
          6    dickes Fadenkreuz
          7    Fadenkreuz mit Umriß
          255  selbst definiert
          256  hide mouse
          257  show mouse
: graf_mkstate      ( -- )
     \    liefert  Mausposition und Knöpfe in den  div.  Arrays
          zurück.

Fileselect :

Create inpath
     \    Der Defaultpath steht in diesem Array mit einem  $00-
          Byte abgeschlossen.
Create insel
     \    Der Name des selektierten Files ...
: fsel_input   ( -- button )
     \    malt  die sog.  File selector box und wartet auf  die
          Auswahl eines Files.   button ist der gedrückte Knopf
          (0 = Cancel)

Window :

: wind_create  (  components leftX topY maxWIDTH  maxHEIGTH  --
               handle)
     \    dieses Wort erzeugt ein Fenster mit div. Bestandtei-
          len und einer Maximalgröße. Liefert eine Handle für's
          Fenster.
: wind_open    ( W-handle leftX topY width height -- )
     \    Malt  ein  Fenster  in der angegeben  Größe  auf  dem
          Bildschirm.
: wind_close   ( Whandle -- )
     \    Schließt und löscht ein Fenster.  Es kann auch wieder
          geöffnet werden.
: wind_delete  ( Whandle -- )
     \    Löscht  ein Fenster,  so daß es nicht  mehr  geöffnet
          werden kann.
: wind_get     ( Whandle funktion# -- )
     \    Liefert div. Informationen über ein Fenster...
: wind_set     ( Whandle funktion# par0 par1 par2 par3 -- )
     \    Setzt  div.  Attribute eines Fenster wie  Titelzeile,
          Sliderpos. usw.
: wind_find    ( mouseX mouseY -- Whandle )
     \    Sucht das Fenster unter der Mausposition.
: wind_update  ( funktion# -- )
     \    wird benutzt, um die Manipulation an anderen Fenstern
          oder  Menüauswahl während des Zeichnens zu  unterbin-
          den.
: wind_calc    ( 0/1 components leftX topY width height -- )
     \    Konvertiert bei Fenstern die Innen- in Außenmaße  (0)
          oder umgekehrt (1).  Die Ausgabe erfolgt in den  GEM-
          Arrays.
```

```
RSRC : : rsrc_load  ( 0$ -- )   \ needs address of 0-termina-
                                  ted $
     \    Lädt ein Resourcefile in den Speicher.
: rsrc_load"
     \    Lädt  das  Resourcefile,  dessen Name,  durch  ein  "
          begrenzt, auf dieses Wort folgt.
: rsrc_free    ( -- )
     \    Gibt  den durch ein Resourcefile beanspruchten  Spei-
          cherbereich wieder frei.
: rsrc_gaddr   ( type index -- laddr )
     \    Liefert die Adresse eines Objektes im Resourcefile.
: rsrc_saddr   ( type index laddr --)
     \    Speichert die Adresse eines Objektes ab.
: rsrc_obfix   ( index laddr --)
     \    Konvertiert  Objektlage und Größe von Character-  in
          Pixeleinheiten.
```

VDI.SCR

Output Function

```
: pline         ( x1 y1 x2 y2 ... xn yn count -- )
     \     malt eine geknickte Linie von x1,y1 zu x2,y2 usw.
: pmarker       ( x1 y1 x2 y2 ... xn yn count -- )
     \     Gibt Polymarker aus.
: gtext         ( addr count x y -- )
     \     Gibt Text an der angegebenen Stelle aus.
: fillarea      ( x1 y1 x2 y2 ... xn yn count -- )
     \     Malt ein ausgefülltes Vieleck.
: contourfill   ( color x y -- )
     \     füllt ein gemalten Bereich aus.
: r_recfl       ( x1 y1 x2 y2 -- )
     \     Malt ein Rechteck ohne Rand.
: GDP           ( #ptsin #intin functionno -- )
     \     Allg. Malroutine
: bar           ( x1 y1 x2 y2 -- )
     \     Malt ein gefülltes Rechteck mit Rand.
: arc           ( startwinkel endwinkel x y radius -- )
     \     Malt den Ausschnitt eines Kreises.
: pie    ( startwinkel endwinkel x y radius -- )
     \     Malt ein Stück Torte.
: circle        ( x y radius -- )
     \     Malt einen Kreis
: ellarc        ( startwinkel endwinkel x y xradius yradius -- )
     \     Malt einen Ellipsenabschnitt
: ellpie        ( startwinkel endwinkel x y xradius yradius -- )
     \     Jetzt gefüllt !
: ellipse       ( x y xradius yradius -- )
     \     Und nochmal, bloß ganz.
: rbox          ( x1 y1 x2 y2 -- )
     \     Rechteck mit abgerundeten Ecken
: rfbox         ( x1 y1 x2 y2 -- )
     \     gefüllt.
: justified     ( string x y length wordspace charspace -- )
     \     Textausgabe  mit definierter Länge,  maximalem  Wort-
           und Buchstabenabstand.
```

Attribute Function :

```
: swr_mode      ( mode -- )
     \     Setzt Ausgabemodus...
1 Setmode replace               2 Setmode transparent
3 Setmode exor                  4 Setmode revtransparent

: sl_type       ( style -- )
     \     Setzt Polyline type
1 Settype solid                 2 Settype longdash
3 Settype dot                   4 Settype dashdot
5 Settype dash                  6 Settype dashdotdot
7 Settype userdef
```

```
: sl_udsty      ( pattern -- )
     \     setzt user defined line style
: sl_width      ( width -- )
     \     setzt polyline line width
: sl_color      ( color -- )
     \     setzt polyline color index
: sl_ends       ( begstyle endstyle -- )
     \     setzt polyline end styles
: sm_type       ( symbol -- )
     \     setzt polymarker type

1 Setmtype point                2 Setmtype plus
3 Setmtype asterisk             4 Setmtype square
5 Setmtype cross                6 Setmtype diamond

: sm_height          ( height -- )
     \     setzt polymarker height
: sm_color           ( color -- )
     \     setzt polymarker color index
: st_height          ( height  -- )
     \     setzt text character height (absolut)
: st_point           ( point -- )
     \     setzt text character height (points)
: st_rotation        ( winkel -- )
     \     setzt character baseline rotation
: st_font            ( font -- )
     \     setzt character font
: st_color           ( color --
     \     setzt text colour
: st_effects         ( effect -- )
     \     setzt fett, kursiv usw.
: st_alignement      ( horin vertin -- )
     \     setzt character alignment
: sf_interior        ( style  --
     \     setzt fill interior style
: sf_style           ( styleindex -- )
     \     setzt fill style index
: sf_color           ( color -- )
     \     setzt fill colour index für Vielecke
: sf_perimeter       ( pervis -- )
     \     schaltet fill outline um.
```

Raster Operation :

```
     \ Die  Rasteroperationen dienen zum schnellen  Verschieben
       von Bildschirmbereichen sowohl auf dem Bildschirm selbst
       als auch vom Bildschirm in den Speicher und  zurück.  Da
       es  sich um sehr schnelle Routinen handelt,  sollte  von
       ihnen  immer dann Gebrauch gemacht  werden,  wenn  Bild-
       schirminhalte  restauriert werden müssen.  Alle -  sonst
       notwendigen  - Ausgaberoutinen brauchen  erheblich  mehr
       Zeit.
```

```
Create scrMFDB
Variable >memMFDB

     \ Die Memory Form Definition Blocks beschreiben den Aufbau
       eines  Pixelblocks im Speicher oder auf dem  Bildschirm.
       Um  mit mehreren Speicherbereichen arbeiten  zu  können,
       enthält >memMFDB einen Pointer auf den gerade  benutzten
       Bereich.

: copyopaque   ( Xfr Yfr width height Xto Yto mode --)
     \ Grundroutine für alle Rasteroperationen
: scr>mem      ( addr_of_memMFDB -- )
     \ Definierendes  Wort  für  Rasteroperation   (Bildschirm-
       >Speicher)
: mem>scr      ( addr_of_memMFDB -- )
     \ Definierendes   Wort  für   Rasteroperation   (Speicher-
       >Bildschirm)
: scr>scr      ( Xfr Yfr width height Xto Yto --)
     \ verschiebt ein Rechteck auf dem Bildschirm.
Create memMFDB1
     \ Ein Speicherblock, der den gesamten Bildschirm speichern
       kann.
: scr>mem1     ( Xleft Ytop Width Heigth -- )
     \ verschiebt den gesamten Bildschirm in den Speicher.
: mem>scr1     ( Xleft Ytop Width Heigth -- )
     \ verschiebt den Speicherblock wieder in den Bildschirm.
: r_trnfm      ( -- )
     \ rechnet  Standard- in gerätespezifische  Koordinaten  um
       und umgekehrt.
: get_pixel    ( x y -- color flag )
     \ ermittelt die Farbe eines Pixels. flag ist 1, wenn Punkt
       gesetzt.
```

Input :

```
: sin_mode     ( devtype mode -- )
     \    legt den Inputmodus fest....
: sm_locater   ( x y -- status )
     \    Position der Maus,  Angabe der Anfangspos. erforder-
          lich
: sm_valuator  ( val_in -- status )
     \    Verwaltung von Wertänderungen ....
: sm_choice    ( -- status )
     \    Teste die Funktionstasten
: sm_string    ( addr max_len echomode x y -- status )
     \    Eingabe eines Strings mit Echo.
: sc_form      ( addr -- )
     \    setzt Mausform
: ex_time      ( tim_addr -- long_otim_addr )
     \    Ersetzt Timerinterrupt-Vektor
: q_mouse      ( -- x y status )
     \    Liest den Mauszustand aus.
: ex_butv      ( pusrcode -- long_psavcode )
     \    Ersetzt Mausbuttoninterruptroutine
: ex_motv      ( pusrcode -- long_psavcode )
     \      ... Mausbewegungsinterruptroutine...
: ex_curv      ( pusrcode -- long_psavcode )
     \    ... Mausforminterruptroutine...
```

```
: q_key_s       ( -- status )
    \    liefert Zustand der Shift-Tasten.
```

Inquire :

```
: q_extnd           ( info_flag -- )
    \    Diese und die folgenden Funktionen dienen dazu,
         festzustellen, welche der obigen VDI-Aufrufe man
         getätigt hat. Man kann sie nicht kurz beschreiben,
         daher muß die Eingangs erwähnte Literatur zu Rate
         gezogen werden.
: q_color           ( color_index info_flag )
: ql_attributes     ( -- )
: qm_attributes     ( -- )
: qf_attributes     ( -- )
: qt_attributes     ( -- )
: qt_extent         ( string -- )
: qt_width          ( char -- status )
: qt_name           ( element_num -- )
: q_cellarray       ( cols rows x1 y1 x2 y2 -- )
: qin_mode          ( dev_type -- mode )
: qt_fontinfo       ( -- )
```

Escape :

```
: q_chcells         ( -- rows cols )
    \    Größe des Bildschirms
: exit_cur          ( -- )
    \    textcursor aus
: enter_cur         ( -- )
    \    textcursor ein
: curup             ( -- )
    \    hoch
: curdown           ( -- )
    \    runter
: curright          ( -- )
    \    ????
: curleft           ( -- )
    \    !!!!
: curhome           ( -- )
    \    cursor nach links oben
: eeos              ( -- )
    \    erase to end of screen
: eeol              ( -- )
    \    ----- " ------- line
: s_curaddress      ( row col -- )
    \    positioniere cursor
: curtext           ( addr count -- )
    \    textausgabe
: rvon              ( -- )
    \    reverse video on
: rvoff             ( -- )
    \    ---- " ------ off
: q_curaddress      ( -- row col )
    \    frage nach cursor adress
: q_tabstatus       ( -- status )
    \    frage nach Mausstatus
: hardcopy          ( -- )
    \    screen dump
```

```
: dspcur            ( x y -- )
     \    positioniere maus
: rmcur             ( -- )
     \    entferne Maus
```

Die folgenden Befehle dienen zur Arbeit mit verschiedenen Output-Devices. Dies ist ein besonders schlecht dokumentierter Teil. Einige Hinweise findet man in der Dokumentation zum Entwicklungspaket von Digital Research. Es ist fraglich, ob alle Befehle tatsächlich auf dem Atari arbeiten. Versuche in dieser Richtung sind bislang noch nicht erfolgreich gewesen.

```
: form_adv          ( -- )
     \    Für Drucker: Seitenvorschub
: output_window     ( x1 y1 x2 y2 -- )
     \    Für Drucker: Gibt ein Fenster aus.
: clear_disp_list   ( -- )
     \    Für Drucker:  bricht Ausdruck ab,  wie Clear Worksta-
          tion, aber ohne abschließendes Linefeed.
: bit_image         ( string aspect scaling num_pts x1 y1 x2 y2 -
                    - )
     \    Für Drucker: Gibt ein 'Bit Image File' aus.
```

Die folgenden Befehle beziehen sich auf zusätzliche Output-Treiber; ein Effekt auf dem Atari ist bisher nicht bekannt. Sie sind nur der Vollständigkeit halber aufgeführt.

```
: s_palette         ( palette -- selected )
     \    setzt Farbpalette auf IBM-Farbmonitor ??!!
: qp_films          ( -- )
: qp_state          ( -- )
: sp_state          ( addr -- )
: sp_save           ( -- )
: sp_message        ( -- )
: qp_error          ( -- )
: meta_extents      ( x1 y1 x2 y2 -- )
: write_meta        ( intin num_intin ptsin num_ptsin -- )
: m_filename        ( string -- )
```

Der Assembler

Der im volksFORTH83 für den Atari 520 ST enthaltene 68000-Assembler entspricht im Wesentlichen dem von Michael Perry für das F83 entwickelten. Daher bringen wir hier eine Übersetzung eines Artikels aus 'Dr. Dobbs Journal', Nr.83 vom September 1983, in dem Michael Perry seinen Assembler beschrieben hat. Abweichungen des volksFORTH-Assemblers werden besonders erwähnt. Im Anschluß an die Übersetzung werden die zusätzlichen Funktionen des volksFORTH- Assemblers beschrieben.

Ein 68000 Forth Assembler

In diesem Artikel werde ich die Eigenarten eines Assemblers in FORTH, seinen Gebrauch und die Implementation eines Beispiels beschreiben: Ein FORTH Assembler für den 68000. Ich hoffe, die Leistungsfähigkeit eines solchen Assemblers darlegen zu können, einige der damit verbundenen Eigenarten, und warum er so und nicht anders programmiert wurde. Um den Leser nicht zu verwirren, verzichte ich auf Verallgemeinerungen. Ich werde gelegentlich Dinge darstellen, die sich speziell auf mein System beziehen und auf anderen Systemen leicht abweichen können.

Kurz gesagt ist ein FORTH-System eine interaktive Programmierumgebung, in der einzelne Module, 'Worte' genannt, in einer Datenstruktur namens Dictionary abgelegt werden. Der Programmierer kann neue Worte hinzufügen, die entweder aus vorhandenen FORTH-Worten oder aus Maschinencodedefinitionen bestehen. Ein Assembler in einem FORTH-System ist ein Werkzeug, um Coderoutinen zu definieren. Er ist nicht dazu vorgesehen, eigenständige Applikationen in Maschinensprache zu schreiben. Ein FORTH Cross-Assembler ist ganz ähnlich aufgebaut. Mit ihm kann man Code erzeugen, der auf einem anderen System läuft, eventuell sogar auf einem anderen Prozessor. Dieser Artikel bezieht sich nur auf 'normale' FORTH-Assembler. FORTH-Assembler sind kurz, da sie auf vorhandene FORTH-Worte zurückgreifen können; dieser Assembler z.B. benötigt nur ca. 3 kByte, dazu kommt das System mit 12 kByte. Zum Vergleich: Der Assembler, der zum CPM 68k gehört, benötigt 44 kByte, zusätzlich etwa 6 kByte für die Symboltabelle.

Wenn man Applikationen in FORTH schreibt, wird der Assembler selten eingesetzt, bevor die einzelnen Programmteile nicht in High-Level geschrieben und ausgetestet sind. In den Anfangsstadien einer Entwicklung ist die Zeit, die der Programmierer braucht, wesentlich wertvoller als die der Maschine. Wenn eine Applikation lauffähig ist, mag es sich herausstellen, daß sie in High-Level zu langsam ist. In diesem Falle muß man herausfinden, welche Routinen zeitkritisch sind und dann nur diese in Code neu schreiben. Diesen Vorgang wiederholt man so lange, bis das Programm schnell genug ist. Vermeiden Sie mehr Coderoutinen als erforderlich, da diese die Übertragbarkeit Ihres Programms stark einschränken. In seltenen Fällen, wenn man eine sehr zeitkritische Anwendung vor sich hat, wird man letztlich fast alles in Code schreiben.

Sogar in solchen Fällen wird die Entwicklung in der oben beschriebenen Reihenfolge am schnellsten zu Resultaten und zur fertigen Anwendung führen. Seien Sie immer bereit, frühere Entwürfe über den Haufen zu werfen und von vorn zu beginnen. Der Schlüssel zum Erfolg ist schrittweise Annäherung: Schreiben, Testen, Überarbeiten, bis Sie endgültig zufrieden sind. Das ist der Grund, warum es so wichtig ist, zunächst eine einfache Version zu entwickeln, um zu sehen, ob die Grundidee richtig und durchführbar ist.

Der Name eines FORTH-Wortes kann aus bis zu 31 Ascii-Zeichen bestehen, ausgenommen sind Leerzeichen. Worte im Dictionary sind in Gruppen zusammengefaßt, die man Vokabulare nennt. Der Assembler ist ein solches Vokabular namens ASSEMBLER. Die meisten Worte im Assembler haben die Namen der üblichen Mnemonics des Prozessors, in unserem Falle des 68000. Wenn so ein Wort ausgeführt wird, legt es die zugehörige Bytefolge im Dictionary ab. Andere Worte im Assembler behandeln die Adressierungsart, Kontrollstrukturen, Makros und möglicherweise andere Erweiterungen. Hält man sich an eine FORTH-übliche Syntax, ist es mit wenig Aufwand möglich, einen sehr leistungsfähigen Assembler zu implementieren.

Die zwei wichtigsten Einschränkungen sind die Syntax und der Verzicht auf Vorwärtsreferenzen. Wie in FORTH üblich sind Vorwärtsreferenzen nicht erlaubt. Das heißt, ein Wort muß vor seinem ersten Aufruf definiert sein. Ich bin der Überzeugung, dies ist eine gute Sache, aber diese Meinung beschwört endlose Debatten herauf, und ich werde sie hier nicht beeenden können. Es ist sehr viel einfacher (und damit auch erheblich schneller), wenn man eine Syntax verwendet, bei der der Operator hinten steht. Das bedeutet, die Befehle werden in folgender Form geschrieben:

```
Source Destination Operation
```

Wenn auch ungewöhnlich, so ist dieses Format doch sehr flexibel und einfach zu verwenden. Ein Pre-Prozessor, der die übliche Schreibweise verarbeiten kann, könnte relativ leicht eingebaut werden, wenn man die damit verbundenen Geschwindigkeitsnachteile in Kauf nimmt.

Das Dictionary wächst in Richtung steigender Adressen, wenn neue Worte hinzugefügt werden. Die meisten Datenstrukturen werden ebenfalls im Dictionary abgelegt. Die Systemvariable DP zeigt auf die nächste freie Adresse. Das Wort HERE übergibt den Wert von DP auf dem Stack. Das Wort , (comma) trägt einen 16-Bit-Wert ins Dictionary ein, das Wort c, (c-comma) einen 8-Bit-Wert (ein Byte). Der Assembler ist nur auf comma und c-comma aufgebaut.

Fehlerbehandlung

Wenn ich einen Assembler benutze, erwarte ich von ihm einige wichtige Leistungsmerkmale. An erster Stelle steht natürlich die richtige Übersetzung: Richtige Eingaben müssen zu richtigen Ausgaben führen. Das zweite ist die Geschwindigkeit. Ich möchte, daß der Assembler seine Arbeit so schnell wie möglich

erledigt. Das dritte ist die Genauigkeit der Übersetzung: Wenn ich Assemblercode schreibe, möchte ich ihn selbst optimieren. Ich möchte keinen optimierenden Assembler benutzen - ich hasse Überraschungen. Schließlich verwende ich ungern allzu 'schlaue' Operatoren, d.h. solche, die mir ein gewisses Maß an Denkfaulheit erlauben, wenn z.B. ADD manchmal ADDI, manchmal auch ADDQ, ADDA oder sonst etwas assembliert. Solche Operatoren sind langsamer und ihr Verhalten weniger durchsichtig. Da FORTH-Assembler erweiterbar sind, kann jeder Benutzer eigene 'schlaue' Operatoren hinzufügen, wenn er möchte.

Bei der Fehlerbehandlung kann im Assembler beliebiger Aufwand getrieben werden. Im Idealfall sollte ein Assembler nur korrekte Eingaben akzeptieren. Es kann allerdings vor allem in Bezug auf die Geschwindigkeit teuer werden, wenn man übertriebene Fehlerkontrolle einbaut. Zum Glück können viele Fehler sehr leicht entdeckt werden. Es ist einfach zu prüfen, ob sich die Stacktiefe während einer Definition verändert (kein Wert bleibt unzulässigerweise übrig bzw. wird verbraucht), ob die Kontrollstrukturen ausgeglichen sind usw.

Die nächste Stufe der Fehlererkennung ist die Prüfung auf erlaubte Adressierungsarten bei jedem Befehl. Bei einer sehr geradlinigen Prozessorarchitektur ist das sehr einfach. Unglücklicherweise ist der 68000 nicht ganz dazu geeignet, auch wenn oft das Gegenteil behauptet wird. Trotzdem können viele Befehle einfach überprüft werden. Obwohl ich so etwas gewöhnlich nicht benutze, habe ich einige Worte eingebaut, die nachprüfen, ob den Befehlen gültige Adressierungsarten zugeordnet sind. ??DN bricht ab, wenn keine direkte Adressierung eines Datenregisters vorliegt. ??AN führt das gleiche für ein Adreßregister durch. ??JMP bricht ab, wenn beim JMP-Befehl eine ungültige Adressierung benutzt wurde.

Für weitergehende Fehlererkennung muß zunehmender Aufwand bei abnehmender Wirkung getrieben werden.

Gebrauchsanleitung für den Assembler

Eine detaillierte und ziemlich genaue Beschreibung des Motorola MC68000 findet man im entsprechenden User's Manual. Als Beispiel für die Benutzung des Assemblers nehmen Sie bitte die Definition des Wortes FILL, das einen Speicherbereich mit einem vorgegebenen Byte füllt. Es wird folgendermaßen benutzt:

```
adresse länge byte  FILL
```

Beachten Sie, daß FILL drei Parameter vom Stack benötigt und nichts übrig läßt.
(Das folgende Beispiel wurde so abgeändert, daß es dem volksFORTH-83 entspricht. Näheres zu den Macros s.u. - Anm. d. Übers.)

```
Code fill   ( adr len val -- )
 SP )+ D0 move      \ Wert nach D0
 SP )+ D1 move      \ Länge nach D1
 SP )+ D6 move      \ Adresse nach D6,
 D6 reg) A0 lea     \ reg)  ist ein Macro,  das aus der  16  Bit
```

```
                    Systemadresse eine absolute 32-Bit Adresse
                    berechnet.
 D1 tst   0<> IF    \ Wenn Länge von 0 verschieden
  1 D1 subq         \ decrement D1; dbra läuft bis -1, nicht 0
  D1 DO    .b D0 A0 )+ move   LOOP
                    \ Schleife  bis D1 = -1;  jedesmal wird  ein
                      Byte in die
                    \ Adresse,  die in A0 steht, geschrieben und
                      A0 incrementiert.
  THEN
Next                \ ein Macro, das zum nächsten Wort springt.
end-code            \ beendet die Definition
```

Das Wort CODE ist ein definierendes Wort. Es erzeugt einen Kopf für das neue Wort FILL und setzt dessen Codefeld auf das Parameterfeld. Das System bleibt im interpretierenden Modus. Der Assembler benutzt den FORTH-Compiler nicht, wie häufig fälschlich angenommen wird. Der Kopf ist so etwas wie ein Eintrag in eine Symboltabelle. Das Codefeld eines jeden Wortes zeigt auf den Code, den dieses Wort ausgeführen soll. Normalerweise zeigen alle Worte, die mit denselben defining words erzeugt worden sind, auf den gleichen Code. Worte, die mit CODE erzeugt werden, bestehen aus einem einzigartigen Code-Segment, das immer auf das Parameterfeld eben dieses Wortes zeigt. Die übrigen Worte der Codedefinition erzeugen eine Bytefolge im Parameterfeld des Wortes.

Assembler-Opcode-Worte wie MOVE benutzen das Wort comma, um der Reihe nach Bytes in das Parameterfeld einzutragen. Wenn das neue Wort nach seiner Ausführung in den FORTH-Interpreter zurückkehren soll, muß die letzte Anweisung NEXT sein. Next ist ein Makro, das einen Sprung in den FORTH-Interpreter assembliert. Seine Definition lautet (im volksFORTH83):

```
: Next     IP )+ D7 move      \ D7 enthält cfa
           D7 reg) D6 move    \ D6 enthält cfa@
           D6 reg) jmp        \ Sprung auf cfa@
;
```

(Das Macro reg) wird weiter unten beschrieben. Anm. d. Übers.) JMP benutzt comma, um den richtigen Opcode und die Adresse einzutragen. END-CODE markiert das Ende, prüft auf Fehler und räumt ein biβchen auf.

SP ist der Name des Stackpointers der virtuellen FORTH-Maschine. Das Wort SP hinterläßt einen Wert auf dem Stack, der den 'direct-adressing' Modus mit Register A6 (volksFORTH83) darstellt. Das Wort A6 hat genau die gleiche Wirkung; beides sind einfache Konstanten. Das Wort)+ modifiziert den Wert auf dem Stack, den SP hinterlassen hat, um die 'indirect mit autoincrement' Adressierung anzuzeigen. Wie das funktioniert, wird später erklärt. Das Wort D0 stellt Datenregister 0 dar.

Das Wort MOVE assembliert einen 68000 move-Befehl. Es benötigt zwei Werte, die jeweils eine Adressierungsart beinhalten. In unserem Beispiel wird der assemblierte Code 16 Bit aus der Adresse, auf die SP zeigt, nach D0 transportieren und dabei SP

um zwei erhöhen. Die Länge der Operation wird von der Variablen SIZE festgelegt , die auf 16-Bit voreingestellt ist. SIZE wird durch .B (Byte), .W (Word) und .L (Long) entsprechend gesetzt.

Das Wort CODE schaltet das Assembler-Vokabular ein, damit bei gleichen Worten im Assembler und im übrigen System (z.B. SWAP) das richtige Wort gefunden wird. Das Wort LMOVE wurde zusätzlich definiert als Spezialfall von MOVE für die oben angesprochene Registerverschiebung. LMOVE assembliert immer einen Long move, ohne dabei SIZE zu verändern. Beachten Sie den Gebrauch von DO und LOOP im Assembler. DO erhält ein Datenregister zugeordnet, das den Schleifenzähler für die Ausführungsphase enthält. DO übergibt HERE und das Register an LOOP, welches einen dbra zurück auf DO assembliert, bei dem das angegebene Datenregister benutzt wird. (In ähnlicher Weise assembliert 0<> IF einen beq (!), dessen Offset beim folgenden THEN berechnet wird. Beachten Sie bitte, daß die Sprungbedingungen vor IF immer gerade entgegengesetzt den Sprungbefehlen sind, also beq bei 0<> oder bne bei 0=. Anm. d. Übers.)

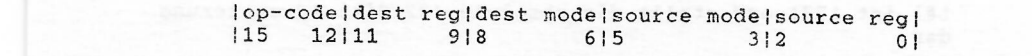
Implementation

Es gibt viele mögliche und darunter zwei häufiger beschrittene Wege, um einen Assembler in FORTH zu schreiben. Eine Methode ist, viele Variable mit Status-Informationen zu benutzen, die ihrerseits von den Mnemonic-Worten verwendet werden, um den Assembliervorgang zu steuern. Nach jeder Instruktion werden sie gelöscht, um von der nächsten Instruktion wieder verwendet werden zu können. Bei diesem Assembler ist eine weiter verbreitete und auch wünschenswertere Methode gewählt worden. Fast sämtliche Informationen werden auf dem Stack übergeben, der auch nicht initialisiert werden muß.

Ebenso gibt es zwei verbreitete Arten, die Adressierungsart an das assemblierende Wort zu übergeben. Eine Möglichkeit besteht in einer Art geschachtelter IF...ELSE Strukturen, die aus einer Folge von Möglichkeiten die richtige heraussucht. Der andere Weg, hier eingesetzt, besteht darin, daß die Worte, die die Adressierungsart bestimmen, die Werte, die ihnen übergeben wurden, in irgendeiner Form verändern. Dies geschieht durch Ausmaskieren mit AND und Setzen einzelner Bits mit OR. Solche Logikoperationen arbeiten bekanntlich viel schneller als Verzweigungen, sodaß der Assembler insgesamt mit solchen Operationen schneller wird.

Wenn Sie die folgenden Beschreibungen lesen, sollten Sie sich den Quelltext des Assemblers zur Hand nehmen. Er befindet sich auf Ihrer Diskette im File ASSEMBLE.SCR.

Die Grundidee, die hinter diesem Assembler steckt, ist die Betrachtung einer Maschinencodeinstruktion als Reihe von Bit-Feldern. Diese Bit-Felder sind im Manual der CPU beschrieben. Einige sind für viele Instruktionen gültig wie source und destination, mode und register Felder.



Wie bereits erwähnt, benutzen Instruktionen, die die Datenlänge kennen müssen, die Variable SIZE. Die Position des Bit-Felds, das die Datenlänge bestimmt, wechselt von Befehl zu Befehl mehr als alle anderen. Fast immer werden die benutzten Werte in die Variable SIZE übergeben, und zwar durch .B, .W oder .L. Beachten Sie, daß ich an diesem Punkt BASE auf OCTAL umgeschaltet habe. Die 68000-Befehle enthalten viele 3-Bit-Felder und können daher besonders übersichtlich als Oktalzahlen dargestellt werden. Ich war gezwungen, meine Vorliebe für Hexzahlen zeitweilig zurückzustellen.

Bei der Definition der Worte, die Register und Adressierungsarten festlegen, habe ich zu einem kleinen Trick gegriffen. Ich benutzte ein 'Multi-defining word' REGS, das in einer Schleife CONSTANT ausführt, um ähnliche Konstanten zu erzeugen. REGS wird nur zweimal benutzt. Einmal für Datenregister und einmal für Adreßregister, die Modus 0 bzw. 1 darstellen.

Modus 0 ist 'data register direct', daher ist D5 eine Konstante mit dem Wert 5005.

Modus 1 ist 'adress register direct', daher ist A3 eine Konstante mit dem Wert 3113.

Worte, die mit MODE definiert wurden, werden nach einem Adreßregister benutzt und ersetzen die zwei Modusziffern (in unserem Fall 1) mit den neuen Modus-Werten. Dies geschieht durch Ausmaskieren der alten Werte mit AND und Setzen der neuen mit OR. Alle MODE-Worte sind 'adress register indirect' mit Zusätzen.

Modus 2 ist 'adress register indirect',
daher ergibt A6) 6226.

Modus 3 ist dasselbe mit 'post-increment',
daher ergibt A7)+ 7337.

Modus 4 ist dasselbe mit 'pre-decrement',
daher ergibt A7 -) 7447.

Modus 5 ist dasselbe mit 'displacement',
daher ergibt 123 A1 D) 1551
mit dem 'displacement' Wert von 123, der zunächst unter dem Register/Modus Wert auf dem Stack liegt.

Modus 6 ist dasselbe mit index und displacement,
daher ergibt 123 D4 A1 DI) 1661.
Auf dem Stack liegen darunter 4004 und 123.

Modus 7 wird für alle übrigen Adressierungsarten verwendet, die sich durch ihre Registerfelder unterscheiden. Diese Modi sind als Konstanten definiert.

#) ist 0770 und stellt die absolute (16-Bit) Adressierung dar. Der Name bedeutet 'immediate indirect'. (Denken Sie darüber nach!)

L#) ist 1771 und stellt die absolute (32-Bit) Adressierung dar.

PCD) ist 2772 und stellt den 'program counter relative mit displacement' Modus dar. 123 PCD) ergibt 2772 und darunter liegt 123 auf dem Stack.

PCDI) ist 3773 und stellt den 'program counter relative displaced, indexed' Modus dar. 123 D4 PCDI) ergibt 3773 und darunter 4004 und 123 auf dem Stack.

ist 4774 und stellt den 'immediate data' Modus für 16 oder 32 Bit dar. 456 # ist 4774, darunter liegt 456.

(Anmerkung d. Übers.: Zusätzlich haben wir ins volksFORTH83 einige weitere Adressierungsarten aufgenommen. Mehr dazu weiter unten.)

Beachten Sie, daß immer 1 bis 3 16-Bit-Werte auf dem Stack hinterlassen werden, die die Adressierungsart kennzeichnen. Der oberste Wert wird normalerweise Teil der ersten 16 Bit eines Befehls zusammen mit dem Opcode. Falls zusätzliche Werte vorhanden sind, werden sie im Anschluß an den Opcode assembliert.

Manche 3-Bit-Felder werden häufiger (durch Ausmaskierung) selektiert als andere. Das Wort FIELD erzeugt Worte, mit denen man solche Felder selektieren kann. RS und RD wählen die source und destination register Felder (s. Bild oben) aus. MS selektiert das source mode Feld. Der erzeugende Ausdruck für eine vollständig festgelegte Adressierungsart ist eine effektive Adresse (EA). Das Wort EAS wählt die 'source effective adress', die aus den source mode und register Feldern besteht. LOW selektiert die unteren 8 Bits. Das Opcode-Wort enthält oft ein EAS Feld. Das Wort SRC führt

```
OVER EAS OR
```

aus, womit es dieses Feld ins Opcode-Wort überträgt. Das Wort DST baut das destination register Feld ein.

Die virtuelle FORTH-Maschine enthält fünf Register. Diese sind einzelnen 68000 Registern zugeordnet. Es ist im Assembler möglich, sowohl die 68000 Register-Namen als auch die Namen der Register der virtuellen Maschine zu benutzen. (Bemerkung d. Übers.: Sie sollten die 68000 Registernamen nur dann benutzen, wenn sie keine FORTH-Register meinen. Sie vermeiden so unerklärliche Systemabstürze, falls die Registerzuordnung sich ändert.)

Adressierungsarten, die nach dem Opcode weitere Werte assemblieren, nennt man 'extended adressing'. Solche Adressierungsarten werden mit sechs Worten und einem Buffer abgehandelt. DOUBLE? hinterläßt ein Flag, das wahr ist, falls der Modus, der oben auf dem Stack liegt, zusätzliche 32 Bit verlangt. INDEX? sucht nach einer Adressierungsart und verändert seine zusätzlichen Werte in das passende Format, falls es sich um 'indexed' Adressierung handelt. MORE? hinterläßt ein True-Flag, wenn die Adressierungsart weitere Werte benötigt. MORE trägt alle zusätzlichen Werte hinter dem Opcode-Wort ins Dictionary ein.

Einige Instruktionen brauchen zwei Adressierungsarten, eine für source und eine für destination. Der source Modus wird zuerst

festgelegt, sodaß er unter dem destination Modus auf dem Stack liegt. Jede Adressierungsart besteht aus ein bis drei Werten, die auf dem Stack liegen. Der source Modus wird vor dem destination Modus verarbeitet, daher müssen die Werte für den destination Modus so lange in einem Buffer abgelegt werden. EXTRA? rettet alle zusätzlichen Werte in einen Buffer namens EXTRA und hinterläßt nur den Wert für die Adressierungsart. ,EXTRA nimmt die zusätzlichen Werte aus dem Buffer und trägt sie ins Dictionary ein.

Fast der ganze Rest des Assemblers besteht aus Definitionen und der Anwendung von definierenden Worten, die Gruppen von Mnemonics herstellen. Zwei Beispiele werden genügen. Wenn Sie mit dem Gebrauch von definierenden Worten nicht vertraut sind - Sie sollten es sein; sie sind die leistungsfähigste Struktur in FORTH.

Das Wort IMM erzeugt Worte, die 'immediate' Befehle assemblieren. Ich werde die Definition hier wiederholen und genau erläutern:

```
: IMM     CONSTANT
    DOES>  @ >R  EXTRA?  EAS  R>  OR
           SZ3 ,  LONG?  ?,  ,EXTRA ;

Gebrauch bei der Definition:   3000 IMM ADDI
Gebrauch im Assembler:         n ea ADDI
Beispiel:                      123 A5 ) ADDI
```

Jedes Mal, wenn mit IMM ein Mnemonic-Wort definiert wird, speichert es einen konstanten Wert in der Definition dieses Wortes ab, der es von anderen 'immediate' Worten unterscheidet. Dieser Wert ist der Opcode des Befehls. Immediate Befehle beinhalten folgende Bit-Felder.

```
| op-code |  size  |  mode  |   reg  |
|15     8|7     6|5     3|2      0|
```

Diesen folgen 16 oder 32 Bit Daten. Wenn das Befehlswort ausgeführt wird, führt es den Code nach DOES> in IMM aus mit der Adresse seines eigenen Parameterfeldes auf dem Stack. An dieser Adresse ist die Konstante (der Opcode) kompiliert. Das Wort @ liest diesen Wert und rettet ihn mit >R auf den Returnstack. ADDI erhält die 'immediate' Daten auf dem Stack unterhalb der Werte für die Adressierungsart. EXTRA? rettet alle zusätzlichen Werte, EAS selektiert die mode und register Felder, die benutzt werden sollen. Dann wird der Opcode vom Returnstack mit R> geholt und durch OR mit EAS verknüpft. SZ3 setzt die zugehörigen Längenbits aus SIZE, und das Wort comma trägt das Opcode-Wort ins Dictionary ein. Jetzt liegen nur noch die Daten auf dem Stack, und LONG? entscheidet, ob ?, 16 oder 32 Bit anhängen soll. Zum Schluß holt ,EXTRA die geretteten zusätzlichen Werte, falls vorhanden, zurück und hängt sie ebenfalls an.

Zahlreiche andere ähnlich aufgebaute definierende Worte werden benutzt, um die meisten übrigen Befehle zu definieren. Viele dieser Wort bilden für sich eine Gruppe und sind deswegen mit dem Wort : definiert, gerade so wie Makros. Die conditional

Befehle sind so regelmäßig, daß ich noch ein 'trick defining word' benutze. SETCLASS verwendet wiederholt ein vorhandenes defining word, jedesmal mit einem anderen Argument, um mehrere Mnemonic-Worte auf einmal zu definieren. Alle 46 conditional Befehle werden definiert, indem jedes der drei defining words 16 mal aufgerufen wird. Dabei entstehen auch zwei ungültige Mnemonics, die nicht weiter benutzt werden. Vielleicht wäre es in diesem Falle besser gewesen, alle 46 Befehle einzeln zu definieren, aber ich wollte zeigen, was alles machbar ist.

Zum Schluß kommen wir zu den structured conditionals. Betrachten Sie folgendes Beispiel:

```
          A3 )+ D1 CMP  0<
IF        D0 A7 )  MOVE
ELSE      A7 ) D0  MOVE
THEN

BEGIN     A3 D2 CMP  0=
WHILE     A0 )+ D0 MOVE
REPEAT
```

Im ersten Beispiel beeinflußt das Ergebnis des Vergleichs bestimmte Flags im Status-Register. IF assembliert einen bedingten Sprung, dessen Opcode (mit Bedingung) durch 0< festgelegt ist. Dieser assembliert also den Wert für einen BGE (Branch greater or equal). ELSE berechnet den Sprungoffset für IF und assembliert einen unbedingten Sprung; THEN berechnet dessen Offset.

Ebenso berechnet WHILE einen bedingten Sprung hinter REPEAT, welches wiederum einen unbedingten Sprung zurück auf BEGIN assembliert.

Beachten Sie, daß keine Labels nötig sind. Der meiste Wirrwarr in normalen Assembler-Quelltexten entsteht durch die riesige Anzahl an bedeutungslosen Labelnamen für Sprungziele. Beachten Sie auch, daß die structured conditionals, die wir hier definiert haben, nur 1-Byte Offsets benutzen. Da der Assembler nur einen Pass durchläuft, muß der Platz für den Sprungoffset frei gehalten werden, bevor seine Größe bekannt ist. Da Coderoutinen in FORTH immer sehr kurz sind, genügt ein Byte für den Offset. Sollte das nicht ausreichen, ersetze ich einfach diese Definitionen durch sehr ähnliche, die einen 16-Bit Offset benutzen.

Schließlich sollte bemerkt werden, daß es keine Worte für die Einrichtung von Datenstrukturen in diesem Assembler gibt. Ein FORTH-Assembler ist Teil einer FORTH-Umgebung; und auf alle Datenstrukturen, die mit normalen FORTH-Worten erzeugt wurden, kann der Assembler zugreifen.
Ein Beispiel:

```
VARABLE  FOO
CODE BAR    FOO R#) NEG  NEXT END-CODE
```

BAR negiert den Inhalt der Variablen FOO.

volksFORTH83 Assembler

Wie bereits gesagt, beruht der Assembler im volksFORTH83 im Wesentlichen auf dem von Michael Perry. Es wurden jedoch einige neue Befehle implementiert, die die besonderen Möglichkeiten von volksFORTH83, z.B. den Heap, ausnutzen, oder sich aufgrund der relokatiblen Struktur als notwendig herausgestellt haben.

Struktur des relokatiblen volksFORTH83

volksFORTH83 ist ein 16-Bit-System, d.h. es lassen sich 64 kByte Speicher adressieren. Bisher lagen diese 64 kByte in einer Speicherbank (ab $50000), sodaß die 16-Bit-Adressen im FORTH-System mit Hilfe einer Konstanten namens mempage (=0005) auf 32-Bit-Adressen erweitert werden konnten. Diese Struktur hat sich jedoch als sehr unflexibel erwiesen, so war es z.B. nicht möglich, RAM-Disks zu benutzen. Wenn man stand-alone-Applikationen erzeugen wollte, mußte man dazu eigens ein Ladeprogramm schreiben, dessen einziger Sinn darin bestand, das eigentliche Programm nach $50000 zu laden und dort zu starten.

Die neue Struktur geht davon aus, daß das System an beliebiger Stelle im Speicher lauffähig sein soll (relokatibel). Dementsprechend dürfen keine absoluten 32-Bit-Adressen im System mehr vorkommen, weil sonst zusätzliche Relokationsinformationen mit abgespeichert werden müßten, was ein SAVESYSTEM nahezu unmöglich machen würde. Glücklicherweise bietet der 68000-Prozessor eine recht leistungsfähige Adressierungsart, nämlich die 'indirekte Adressierung mit Index und Displacement'. Grundlage ist dabei ein Adreßregister, daß auf Byte 0 des FORTH-Systems zeigt und Forthpointer (FP) heißt. Alle Speicheroperationen müssen nun relativ zu diesem Zeiger erfolgen.

Die 16-Bit-Adressen im FORTH-System werden nun als Offset zum Byte 0 des Systems aufgefaßt und als Index in einem Datenregister benutzt. Unglücklicherweise werden Datenregister, die als Index verwendet werden, auf 32 Bit vorzeichenerweitert, wenn sie nur wortweise adressiert werden. Der Assembler mußte daher so abgeändert werden, daß bei den Adressierungsarten DI) und PCDI) die Länge der Operation unabhängig von der Länge des Indexregisters angegeben werden kann. Ein Beispiel: Die Befehlsfolge

```
.l 0 D6 FP DI) .w D0 move
```

addiert zu FP den Wert des Datenregisters D6 lang, also ohne Vorzeichenerweiterung, geMOVEd werden aber nur 16 Bit. Steht also in D6 die 16-Bit-Adresse einer FORTH-Variablen, z.B. $9000 und zeigt FP auf $12300, würde diese Befehlsfolge den 16-Bit-Inhalt der Variablen, die absolut bei $1B300 liegt, ins Register D0 bringen.

Zusätzliche Adressierungsarten

Um das Programmieren in Assembler nun nicht noch schwieriger zu machen als es ohnedies schon ist, haben wir einige Macros

entwickelt, die den Charakter von Adressierungsarten haben.

```
: reg)    size push   .l  0 swap  FP DI) ;
```

Diese Adressierungsart könnte 'Datenregister indirect' benannt werden. Ein Beispiel:

```
Code @   ( addr -- 16b )
   SP )+ D6 move   D6 reg) SP -) move   Next
```

assembliert genau dasselbe wie

```
Code @   ( addr - 16b )
   SP )+ D6 move   .l 0 D6 FP DI) .w SP -) move   Next
```

Die Adresse auf dem Stack wird in D6 geladen; D6 wird lang zum Forthpointer addiert und der Inhalt dieser Speicherstelle wieder auf den Stack gebracht. (Anmerkung: @ im System ist etwas komplizierter definiert, da auch der Zugriff auf ungerade Adressen gestattet ist, was bei der obigen Definition eine Fehlermeldung auslösen würde.)

Zum Zugriff auf Datenstrukturen, vor allem auf Variablen, dient das Macro

```
: R#)   ( addr -- )      size push  .w
   dup 0< IF  # D6 move   D6 reg)   exit THEN
   FP D) ;
```

Dieses Macro gab es auch schon im 'alten' System, der Effekt ist auch der gleiche, sodaß vorhandene Assemblerquelltexte nicht umgeschrieben werden müssen. Die Wirkungsweise hat sich jedoch geändert: Ist addr negativ - also größer als $7FFF - wird die Adresse direkt nach D6 geladen und dann wie bereits beschrieben mit D6 indiziert. Liegt addr jedoch in den unteren 32 kByte des FORTH-Systems, erfolgt der Zugriff mit addr als Displacement ohne Indexregister, was erhebliche Geschwindigkeitsvorteile bringt. Vor allem die FORTH-Systemvariablen lassen sich auf diese Art und Weise schnell erreichen. Ein weiteres Beispiel:

```
Variable test
Code test@   ( -- 16b )
   test R#) SP -) move   Next
```

TEST@ legt den Inhalt der Variablen TEST auf den Datenstack.

Schließlich gibt es noch das Macro PCREL), dessen genauere Definition im Assemblerquelltext ASSEMBLE.SCR nachzulesen ist. Es arbeitet analog zu R#), ist aber schneller und kürzer. Nachteil ist, daß der 68000 die Adressierungsart PC-relativ nicht bei allen Operatoren zuläßt, und daß nicht der gesamte 64 kByte Adreßraum erreichbar ist. PCREL) ist mit Fehlermeldungen gegen falsche Adreßdistanzen abgesichert.

Register der virtuellen FORTH-Maschine

Folgende Register werden vom volksFORTH83 benutzt:

A3 Forthpointer (FP). Dieses Register enthält die Startadresse des Systems und gibt damit die Basis für alle relativen Adreßzugriffe ins System. Grundsätzlich gilt: Die absolute 32-Bit-Adresse erhält man durch Addition der 16-Bit-Forth-Adresse und FP. Die 16-Bit-Forth-Adressen lassen sich also als Offset zum Systemanfang auffassen.

A4 Instructionpointer (IP). Dieses Register enthält die (absolute) 32-Bit-CFA des nächsten auszuführenden Wortes.

A5 Returnstackpointer (RP). Dieses Register enthält den 'Systemstackpointer' der virtuellen FORTH-Maschine. Hier werden 'Rücksprungadressen' in aufrufende Worte usw. abgelegt.

A6 Datenstackpointer (SP). Dieses Register enthält den Datenstackpointer der virtuellen FORTH-Maschine. Über diesen Stack werden bekanntlich nahezu sämtliche Werte zwischen einzelnen Funktionen übergeben.

Neben diesen vier Adreßregistern werden noch zwei Datenregister benutzt:

D7 Work-Register (W) der virtuellen FORTH-Maschine; der 16-Bit-Wert in diesem Register zeigt auf die CFA des Wortes, das gerade ausgeführt wird.

D6 Ein Hilfsregister, das zur Umrechnung von relativen 16-Bit-Adressen in absolute 32-Bit-Adressen benutzt wird. Dem Programmierer stehen also die Datenregister D0 - D5 sowie die Adreßregister A0 - A2 zur freien Verfügung. D6 und D7 dürfen verändert werden, jedoch müssen die oberen 16 Bit immer auf 0 stehen. Sollen weitere Register verwendet werden, müssen sie vorher gerettet und anschließend restauriert werden.

Zusätzliche Befehle

;c: Schaltet den Assembler ab und den FORTH-Compiler an. Damit ist es möglich, von Maschinencode in FORTH überzuwechseln. Ein Gegenstück ist nicht vorhanden.

Ein Beispiel für die Verwendung von ;c: ist:

```
.... 0< IF  ;c: ." Fehler" ; Assembler  THEN
```

Ist irgendwas kleiner als Null, so wird 'Fehler' ausgedruckt und die Ausführung des Wortes abgebrochen, sonst geht es weiter im Code.

Schließlich gibt es noch die Worte >LABEL und LABEL. >LABEL erzeugt eine Konstante auf dem Heap, wobei es den Wert des

Labels vom Stack nimmt. LABEL erzeugt eine Konstante mit dem Wert von HERE. Beispiel:

```
LABEL  SCHLEIFE   1 D0 subq  SCHLEIFE BNE
```

SCHLEIFE verbraucht keinen Dictionaryspeicher, weil es vollständig - mit Header und Wert - auf dem Heap liegt. Es sind also echte Assemblerlabels möglich. Von der Verwendung solcher Labels kann allerdings nur abgeraten werden, wenn wie im obigen Beispiel ebensogut strukturiert programmiert werden könnte.

Übrigens ist >LABEL statesmart, d.h es verhält sich verschieden, je nachdem, ob das System kompiliert oder nicht. Während der Kompilation werden Labels als Literals kompiliert. Damit können Labels auch in Colon-Definitionen verwendet werden.

Der Disassembler

Der Disassembler wird geladen mit

```
include disass.scr
```

Mit DISW <word> kann man in Code geschriebene Worte disassemblieren. Das funktioniert natürlich mit Systemworten ebensogut wie mit eigenen Definitionen. Die Ausgabe stoppt automatisch, wenn ein NEXT erkannt wurde, und kann dann mit einer beliebigen Taste fortgesetzt oder mit Escape abgebrochen werden. Auch während der Ausgabe kann mit einer beliebigen Taste unterbrochen oder mit Escape abgebrochen werden.

Der Disassembler disassembliert die Befehle in der üblichen Motorola-Syntax, nicht in der Schreibweise des FORTH-Assemblers. Wer häufiger mit Assembler-Quelltexten zu tun hat, wird das zu schätzen wissen....

Weitere Benutzer-Worte des Disassemblers sind

```
dis   ( addr -- )
```

und

```
ldis   ( laddr -- )
```

Beide disassemblieren von einer vorgegebenen FORTH- bzw. Langadresse ab. Mit LDIS lassen sich auch Routinen außerhalb von FORTH disassemblieren, z.B. im TOS oder GEM. Die Ausgabe wird genauso gesteuert wie bei disw.

Sonstiges

Wie im Artikel von Michael Perry bereits zu lesen war, verzichtet der Assembler auf 'übertriebenes Errorchecking'. Im Klartext heißt das, daß man sich schon recht gut mit dem 68000-Befehlsumfang auskennen sollte, insbesondere mit den erlaubten Adressierungsarten und Operandenlängen bei den einzelnen Befehlen. Anfängern - und nicht nur solchen (!) - sei dringend empfohlen, grundsätzlich den erzeugten Maschinencode mit dem Disassembler zu überprüfen. Man reduziert damit die Anzahl der sonst unvermeidlichen Systemabstürze erheblich.

Der FORTH-Assembler ist etwas gewöhnungsbedürftig. Beispiele für verschiedenartige Anwendungen sind vor allem im File FORTH_83.SCR zu finden. Diese Beispiele sollte man studieren, bevor man sich an eigene Experimente heranwagt. Man muß ja nicht gleich mit (WORD beginnen....

Fileinterface für das Voksforth83 auf dem Atari ST

Erster Einstieg !

Bevor Sie das Glossar lesen, sollten Sie diese kleine Einführung lesen und auf einer leeren Diskette die Beispiele ausprobieren.

Wie erzeuge ich ein File, in das ich ein Programm eingeben kann?

Geben Sie bitte folgendes ein:

```
MAKEFILE test.scr
```

Das File test.scr wird auf dem Laufwerk erzeugt, auf dem Sie das Forth gebootet haben. Als nächstes schätzen Sie bitte ab, wie lang Ihr Programm etwa wird. Beachten Sie dabei bitte, daß der Screen 0 eines Files für Hinweise zur Handhabung ihres Programms und der Screen 1 für einen sog. Loadscreen (das ist ein Screen, der den Rest des Files lädt) reserviert sind. Wollen Sie also z.B. 3 Screens Programm eingeben, so sollte das File 5 Screens lang sein; Sie geben also ein:

```
5 MORE
```

Fertig ! Sie haben jetzt ein File, das die Screens 0..4 enthält. Geben Sie jetzt

```
1 L
```

ein. Sie editieren jetzt den Screen 1 Ihres neuen Files test.scr . Sie können, falls der Platz nicht ausreicht, Ihr File später einfach mit MORE verlängern. Ein File kann leider nicht verkürzt werden. **Wie spreche ich ein bereits auf der Diskette vorhandenes File an?**

Das geht noch einfacher. Geben Sie einfach den Filenamen ein. Reagiert das System mit der Meldung "Haeh ?" , so kennt das Forth dieses File noch nicht. Sie müssen in diesem Fall das Wort USE vor dem Filenamen eingeben, also z.B.

```
USE test.scr
```

Jetzt können Sie wie oben beschrieben mit 1 L (oder einer anderen Zahl) das File editieren. Das Wort USE erzeugt übrigens im Forthsystem das Wort TEST.SCR, falls es noch nicht vorhanden war. Wissen Sie also nicht mehr, ob Sie ein File schon benutzt haben, so können Sie mit WORDS nachsehen oder das Wort USE voranstellen.

Wie erzeuge ich Files auf einem bestimmten Laufwerk, z.B. A: ?

Entweder durch Voranstellen des Pfadnamens oder durch Eingabe von :

A: Hierbei wird A: zum aktuellen Laufwerk gemacht.

Wie spreche ich Directories an ?

Hierbei gibt es verschiedene Methoden. Für den Anfang versuchen wir, Files in einem Directory "test.dir" zu suchen bzw. zu erzeugen. Dazu geben Sie ein :

dir test.dir

Jetzt werden in test.dir alle Files und Directories erzeugt und gesucht. Ist das Directory "test.dir" noch nicht vorhanden, so erzeugen Sie es mit :

makedir test.dir

Anschließend können Sie wieder DIR benutzen.

0) Allgemeines

Im folgenden wird die Benutzung des Fileinterfaces beschrieben. Dieses Fileinterface benutzt die Files des GEM-Dos und dessen Subdirectories.

Benutzt man ein File von Forth aus, so wird es in Blöcke zu je 1024 Bytes aufgeteilt, die in gewohnter Weise anzusprechen sind. Dies trifft auch für Files zu, die nicht vom Forth aus erzeugt wurden. Als Konvention wird vorgeschlagen, daß Files, die Forth-Screens, also Quelltexte, enthalten, mit .SCR erweitert werden. Files, die Daten enthalten, die nicht unmittelbar lesbar sind, sollten auf .BLK enden.

Zum Umschalten vom Filesystem auf Direktzugriff und umgekehrt gibt es das Wort

DIRECT (--)
schaltet auf Direktzugriff um. Auf den Filezugriff schalten wir durch das Nennen eines Filenamens um.

DOS (--)
Viele Worte des Fileinterfaces, die normalerweise nicht benötigt werden, sind in diesem Vokabular enthalten.

1) Der Directory-Mechanismus

Die Verwaltung von Directories entspricht in etwa der des GEM-Dos Command-interpreters COMMAND.PRG

PATH (--)

Dieses Wort gestattet es, mehrere Directories nach einem zu öffnenden File durchsuchen zu lassen. Das ist dann praktisch, wenn man mit mehreren Files arbeitet, die sich in verschiedenen Directories oder Diskstationen befinden. Es wird in den folgenden Formen benutzt:

```
PATH dir1;dir2;....
```

Hierbei sind <dir1>, <dir2> etc. Pfadnamen. Wird ein File auf dem Massenspeicher gesucht, so wird zunächst <dir1>, dann <dir2> etc. durchsucht. Zum Schluß wird das File dann im aktuellen Directory (siehe DIR) gesucht. Beachten Sie bitte, daß keine Leerzeichen in der Reihe der Pfadnamen auftreten dürfen.

Beispiel: PATH A:\;B:\COPYALL.DEM;\AUTO\

In diesem Beispiel wird zunächst die Diskstation A, dann das Directory COPYALL.DEM auf Diskstation B und schließlich das Directory AUTO auf dem aktuellen Laufwerk (siehe SETDRIVE, A: und B:) gesucht. Beachten Sie bitte das Zeichen "\" vor und hinter AUTO . Das vordere Zeichen "\" steht für das Hauptdirectory. Wird es weggelassen, so wird AUTO\ an das mit DIR gewählte Directory angehängt. Das hintere Zeichen "\" trennt den Filenamen vom Pfadnamen.

Außerdem können Sie eingeben :

```
PATH ;
```

In diesem Fall werden alle Pfade gelöscht und es wird nur noch das aktuelle Directory durchsucht.
Schließlich geht noch :

```
PATH
```

Dann druckt PATH die Reihe der Pfadnamen aus.
Dieses Wort entspricht dem Kommando PATH des Kommandointeters.

MAKEDIR cccc (--)

Erzeugt im aktuellen Directory (siehe DIR) ein Directory mit Namen CCCC.
Dieses Wort entspricht dem Kommando MD des Kommandointerpreters.

A: (--)

Macht die Diskstation A: zum aktuellen Laufwerk (siehe SETDRIVE).
Dieses Wort entspricht dem Kommando A: des Kommandointerpreters.

B: (--)
Macht die Diskstation B: zum aktuellen Laufwerk (siehe SETDRIVE).
Dieses Kommando entspricht dem Kommando B: des Kommandointerpreters.

C: (--)
D: (--)
Analog zu A: und B: .

SETDRIVE (n --)
Macht die Diskstation mit der Nummer n zu aktuellen Laufwerk. Hiebei entspricht n=0 der Diskstation A, n=1 der Diskstation B usw.
Auf dem aktuellen Laufwerk werden Files und Directories erzeugt (und nach Durchsuchen von PATH gesucht). Das Directory, in dem Files erzeugt werden, wird mit DIR angegeben.

DIR (--)
Wird in den folgenden Formen benutzt:

```
DIR cccc
```

<cccc> ist ein Pfadname, an den neu zu erzeugende Files und Directories angehängt werden.
Beispiel:

```
A: DIR AUTO
```

erzeugt alle folgenden Files im Directory AUTO auf der Diskstation A. In <cccc> können alle vom GEM-Dos zugelassenen Zeichen außer Leerzeichen verwendet werden.

Außerdem geht noch

```
DIR
```

Ohne weitere Eingaben wird der aktuelle Suchpfad einschließlich des Laufwerks angezeigt.
Dieses Wort entspricht dem Kommando CD des Kommandointerpreters.

FILES (--)
Listet den Inhalt des aktuellen Directories (siehe DIR und SETDRIVE) auf dem Bildschirm auf. Subdirectories werden durch ein vorangestelltes "D" gekennzeichnet.
Dieses Wort, zusammen mit dem Wort FILES" entspricht dem Kommando DIR des Kommandointerpreters.

FILES" (--)
Benutzt in der Form

```
FILES" cccc"
```

Listet die Files auf, deren Name CCCC ist. Der String CCCC darf die bekannten Wildcards sowie einen Pfadnamen enthalten. Wird kein Pfadname bzw. Laufwerk angegeben, so

werden die Files des aktuellen Directories bzw. Laufwerks ausgegeben.

SAVESYSTEM (--)
Benutzt in der Form

```
SAVESYSTEM <name>
```

Damit läßt sich ein FORTH-System im gegenwärtigen Zustand unter dem Namen <name> abspeichern. Dieses Wort wird benutzt, um fertige Applikationen zu erzeugen. In diesem Fall sollte das Wort, das die Applikation ausführt, in 'COLD gepatched werden. Ein Beispiel: Sie haben ein Kopierprogramm geschrieben; das oberste ausführende Wort heißt COPYDISK. Mit der Sequenz

```
' COPYDISK IS 'COLD    <Return>
SAVESYSTEM COPY.PRG    <Return>
```

erhalten Sie ein Progamm namens COPY.PRG, das sofort beim Start COPYDISK ausführt.

2) Files

Files bestehen aus einem Forthnamen und einem GEM-Dos-Namen, die nicht übereinstimmen müssen.
Ist das Forthwort, unter dem ein File zugreifbar ist, gemeint, so wird im folgenden vom Forthfile gesprochen. Ist das File auf der Diskette gemeint, das vom GEM-Dos verwaltet wird, so wird vom GEM-File gesprochen. Durch das Nennen des Forthnamens wird das Forthfile (und das zugeordnete GEM-File) zum aktuellen File, auf das sich alle Operationen wie LIST, LOAD, CONVEY usw. beziehen.
Beim Öffnen eines Files wird die mit PATH angegebene Folge von Pfadnamen durchsucht. Ist ein File einmal geöffnet, so kann diese Folge beliebig geändert werden, ohne daß der Zugriff auf das File behindert wird. Aus Sicherheitsgründen empfiehlt es sich aber, Files so oft und so lange wie irgend möglich geschlossen zu halten. Dann kann eine Änderung der Folge von Pfadnamen dazu führen, daß ein File nicht mehr gefunden wird.

FILE (--)
Wird in der Form:

```
FILE <name>
```

benutzt.
Erzeugt ein Forthwort mit Namen <name>. Wird <name> später ausgeführt, so vermerkt es sich als aktuelles File. . Ebenso vermerkt es sich als FROMFILE, was für CONVEY wichtig ist. Einem Forthfile wird mit MAKE oder ASSIGN ein GEM-File zugeordnet.

MAKE (--)
Wird in der Form:

```
MAKE  cccc
```

benutzt. Erzeugt ein GEM-File mit Namen cccc im aktuellen Directory und ordnet es dem aktuellen Forthfile zu. Das File wird auch gleich geöffnet. Es hat die Länge Null (siehe MORE).

Beispiel: FILE test.scr test.scr MAKE test.scr

erzeugt ein Forthwort TEST.SCR und ein File gleichen Namens. Alle Operationen wie LOAD , LIST usw. beziehen sich nun auf den entsprechenden Screen in TEST.SCR. Beachten Sie bitte, daß dieses File noch leer ist und daher eine Fehlerbedingung besteht, wenn Zugriffsoperationen ausgeführt werden sollen.

MAKEFILE (--)

Wird in der folgenden Form benutzt:

MAKEFILE <name>

Erzeugt ein Forthfile mit Namen <NAME> und erzeugt anschließend ein GEM-File mit demselben Namen. Die Sequenz

FILE <name> <name> MAKE <name>

würde genau dasselbe bewirken.

ASSIGN (--)

Wird in der Form

ASSIGN cccc

benutzt. Ordnet dem aktuellen File das GEM-File mit Namen CCCC zu. Eine Fehlerbedingung besteht, wenn das File nicht gefunden werden kann.

USE (--)

Dieses Wort ist das wichtigste Wort zum Auswählen von Files.

Es wird in der folgenden Form benutzt:

USE <name>

Dieses Wort macht das File mit Namen <NAME> zum aktuellen File, auf das sich LOAD, LIST usw. beziehen. Es erzeugt ein Forthfile mit Namen <NAME> , falls der Name noch nicht vorhanden war.
Anschließend wird das File in den Directories, die mit PATH angegeben wurden, gesucht. Wird es dort nicht gefunden, wird das mit SETDRIVE (bzw. A: und B:) und DIR angegebene Directory durchsucht. Wird auch dort kein GEM-Dos File mit dem Namen <NAME> gefunden, so wird die Fehlermeldung FILE NOT FOUND ausgegeben. Das (automatisch) erzeugte Forthfile verbleibt im Dictionary und muß ggf. mit FORGET vergessen werden.

CLOSE (--)

Schließt das aktuelle File. Dabei wird das Inhaltsverzeichnis der Diskette aktualisiert und die sog. Handlenum-

mer wieder freigegeben. Es werden die zu diesem File gehörenden geänderten Blöcke auf die Diskette zurückgeschrieben und alle zu diesem File gehörenden Blöcke gelöscht.

OPEN (--)
Öffnet das aktuelle File. Eine Fehlerbedingung besteht, wenn das File nicht gefunden werden kann. Die Benutzung dieses Wortes ist in den meisten Fällen überflüssig, da Files automatisch bei einem Zugriff geöffnet werden.

FROM (--)
Wird in der folgenden Form benutzt:

```
FROM <name>
```

<NAME> ist der Name eines Forthfiles, aus dem beim Aufruf von CONVEY und COPY Blöcke herauskopiert werden sollen.

Beispiel: filea 1 FROM fileb 3 copy

Kopiert den Block 1 aus FILEB auf den Block 3 von FILEA. Dieses Wort benutzt USE um das File auszuwählen. Das bedeutet, daß fileb automatisch als Forthfile angelegt wird, wenn es noch nicht im System vorhanden ist.

LOADFROM (n --)
Wird in der folgenden Form benutzt:

```
LOADFROM <name>
```

<NAME> ist der Name eines Forthfiles, aus dem der Block n geladen wird.

Beispiel: 15 LOADFROM filea

Lädt den Block 15 aus FILEA. Dieses Wort ist wichtig, wenn während des Ladens eines Files Teile eines anderen Files geladen werden sollen. Damit kann die Funktion eines Linkers imitiert werden. Dieses Wort benutzt USE , um filea zu selektieren. Das bedeutet, daß automatisch ein Forthfile mit Namen filea erzeugt wird, falls es im System noch nicht vorhanden war.
Beachten Sie bitte, daß dieses Wort nichts mit FROM oder FROMFILE zu tun hat, obwohl es ähnlich heißt !

INCLUDE (--)
Wird in der folgenden Form benutzt:

```
INCLUDE <name>
```

<NAME> ist der Name eines Forthfiles, das vollständig geladen wird. Dabei ist Voraussetzung, daß auf Screen 1 dieses Files Anweisungen stehen, die zum Laden aller Screens dieses Files führen. Siehe auch LOADFROM.

CAPACITY (-- u)
u ist die Länge des aktuellen Files. Beachten Sie bitte,

daß die Länge des Files um eins größer ist als die Nummer des letzten Blocks, da der Block 0 mitgezählt wird.

FORTHFILES (--)
Druckt eine Liste aller Forthfiles, zusammen mit den Namen der zugehörigen GEM-Files, deren Länge und Handlenummer aus.

FROMFILE (-- addr)
Addr ist die Adresse einer Variablen, die auf das Forth-File zeigt, aus dem CONVEY und COPY Blöcke lesen. Siehe auch FROM. Bei Nennen eines Forth-Files wird diese Variable gesetzt.

FILE? (--)
Druckt den Namen des aktuellen Forthfiles.

MORE (n --)
Verlängert das aktuelle File um n Screens. Die Screens werden hinten angehängt. Anschließend wird das File geschlossen.

(MORE (n --)
Wie MORE, jedoch wird das File nicht geschlossen.

EOF (-- f)
f ist ein Flag, das wahr ist, falls über das Ende des Files hinaus gelesen wurde. f ist falsch, falls auf das zuletzt gelesene Byte im File noch weitere Bytes folgen.

3) Verschiedenes

Beim Vergessen eines Forth-Files mit Hilfe von FORGET, EMPTY usw. werden automatisch alle Blockpuffer, die aus diesem File stammen, gelöscht, und, wenn sie geändert waren, auf die Diskette zurückgeschrieben. Das File wird anschließend geschlossen.

Bei Verwendung von FLUSH werden alle Files geschlossen. FLUSH sollte VOR jedem Diskettenwechsel ausgeführt werden, und zwar nicht nur, um die geänderten Blöcke zurückzuschreiben, sondern auch damit alle Files geschlossen werden. Sind nämlich Files gleichen Namens auf der neuen Diskette vorhanden, so wird sonst eine abweichende Länge des neuen Files vom Forth nicht erkannt. Bei Verwendung von VIEW wird automatisch das richtige File geöffnet.

Diverses

Im File DIVERSES.SCR sind einige Worte zusammengefaßt, die sich nicht so recht zuordnen lassen.

>absaddr (addr -- abs_laddr)
rechnet eine - relative - Adresse im FORTH-System in die entsprechende absolute 32-Bit-Adresse um.

.blk (--)
gibt die Nummer des gerade kompilierten Screens aus. Wenn Screen 1 eines Files kompiliert wird, wird zusätzlich auch der Filename in einer neuen Zeile ausgegeben.

abort((f --)
Wird benutzt in der Form
Bedingung ABORT(string)
Wenn f wahr ist, wird der String ausgegeben; entspricht abort" , ist jedoch im Direktmodus zu benutzen.

arguments (n --)
prüft, ob mindestens n Werte auf dem Stack liegen ; andernfalls wird mit einer Fehlermeldung abgebrochen.

cpush (addr len --)
wie PUSH, aber nicht für eine Variable, sondern für Speicherbereiche, die nach Ausführung eines Wortes auf die alten Werte zurückgesetzt werden sollen. Auf diese Art werden 'lokale Arrays' möglich.

bell (--)
gibt einen Piepston auf der Konsole aus.

blank (addr len --)
Ab addr werden len Bytes mit Leerzeichen überschrieben.

setvec (--)
setzt den Critical Error Handler des Betriebssystems auf eine neue Routine, die die Ausgabe von Fehlerboxen bei Schreib- oder Lesefehlern bei Diskettenzugriffen verhindert. Andere Boxen, die über diesen Handler laufen, erscheinen nach wie vor, z.B. die Aufforderung, bei einem Laufwerk eine andere Diskette einzulegen. Damit entfällt das Geduldsspiel, ohne sichtbare Maus das Abbruchfeld zu finden.

restvec (--)
setzt den Critical Error Handler auf den alten Wert zurück. Diese Routine muß unbedingt ausgeführt werden, bevor man FORTH verläßt, da sonst das System abstürzt.

Allocate

MALLOC und MFREE enthalten die Betriebssystemaufrufe, mit denen zusätzlicher Speicher beim Betriebssystem an- und abgemeldet werden muß.

malloc (d -- laddr)

Es werden d Byte beim Betriebssystem angefordert; laddr ist die Anfangsadresse des reservierten Speicherbereichs. Eine Fehlerbedingung besteht, wenn nicht genug Speicherplatz vorhanden ist. In diesem Fall wird mit der Meldung 'No more RAM' abgebrochen.

mfree (laddr --)

Gibt den reservierten Speicher ab laddr wieder frei. Eine Fehlerbedingung besteht, falls kein Speicher reserviert war. In diesem Fall wird mit der Meldung 'mfree Error' abgebrochen.

Relocate

Hier sind Worte aufgeführt, mit denen sich die Speicheraufteilung des volksFORTH ändern läßt. Von der Größe des Return- und Datenstacks (also der Dictionarygröße) hängt der Platz für die Diskbuffer ab. Wenig Buffer erlauben ein großes Dictionary, bieten aber beim Editieren wenig Komfort, weil ständig auf das Laufwerk zugegriffen werden muß.

relocate (stacklen rstacklen --)
richtet das System neu ein mit stacklen Dictionarygröße und rstacklen Returnstackgröße (Vgl. die Memory Map des volksFORTH im Kapitel über den Multitasker). Abschließend wird COLD ausgeführt, um u.a. die Diskbuffer neu zu initialisieren. Vor RELOCATE sollte daher SAVE aufgerufen werden.

buffers (+n --)
stellt ein System mit +n Diskbuffern her. Es gilt das bei RELOCATE Gesagte.

Strings

Hier befinden sich grundlegende Routinen zur Stringverarbeitung. Vor allem wurden auch Worte aufgenommen, die den Umgang mit den vom Betriebssystem geforderten 0-terminated Strings ermöglichen. FORTH hat hier gegenüber 'C' den Nachteil, daß FORTH-Strings standardmäßig mit einem Count-Byte beginnen, das die Länge des Stings enthält. Ein abschließendes Zeichen (z.B. Null) ist daher unnötig. Da das Betriebssystem aber in 'C' geschrieben wurde, müssen Strings entsprechend umgewandelt werden.

Beachten Sie bitte auch im Glossar die Wortgruppe "Strings".

caps (-- adr)
Adr ist die Adresse einer Variablen, die angibt, ob beim Stringvergleich mit COMPARE auf Groß- und Kleinschreibung geachtet werden soll.

-text (addr0 len addr1 -- n) " minus-text"
addr0 und addr1 sind die Adressen von zwei counted strings, len die Anzahl von Zeichen, die verglichen werden soll. Die Strings werden zeichenweise voneinander subtrahiert. n enthält die Differenz der beiden ersten nicht übereinstimmenden Zeichen. Ist n = 0, so sind die Strings gleich.

compare (addr0 len addr1 -- n)
wie -TEXT, jedoch wird der Inhalt der Variablen CAPS ausgewertet. Ist CAPS wahr, wird nicht auf Groß- oder Kleinschreibung geachtet. Die Vergleichsroutine ist dann allerdings erheblich langsamer.

search (text textlen buf buflen -- offset f)
Im Text ab Adresse text wird in der Länge textlen nach dem String, der im Buffer buf mit der Länge buflen steht, gesucht. Zurückgeliefert wird der Offset in den durchsuchten Text an die Stelle, an der der String gefunden wurde, sowie das Flag f . Ist f wahr, wurde der String gefunden, sonst nicht.

delete (buffer size count --)
Im Buffer der Länge size werden count Zeichen entfernt. Der Rest des Buffers wird 'herangezogen' und das Ende mit Leerzeichen aufgefüllt.

insert (string len buffer size --)
Der String ab Adresse string mit der Länge len wird in den Buffer ab buffer mit der Größe size eingefügt. Der Rest des Buffers wird nach hinten geschoben, Zeichen am Ende fallen weg.

$sum (-- adr) " string-sum"
Adr ist die Adresse einer Variablen, die auf einen String zeigt. An diesen String kann ein weiterer String mit $ADD hinten angefügt werden.

$add (addr len --) " string-add"
hängt den String ab addr mit der Länge len an den String bei $SUM an. Der Count wird dabei addiert.

c>0" (addr --) " counted-to-zero-string"
wandelt den counted String ab addr in einen 0-terminated String um. Die Länge des Strings im Speicher bleibt gleich.

0>c" (addr --) " zero-to-counted-string"
wandelt den 0-terminated String ab addr in einen counted String um. Die Länge des Strings im Speicher bleibt gleich.

,0" (--) " comma-zero-string"
Wird in der folgenden Form benutzt :
,0" ccc"
Legt den String ccc mit führendem Countbyte und abschließender Null im Dictionary beginnend bei HERE ab. Vergleiche ,"

0" (-- adr) " zero-string"
(--) compiling
Wird in der folgenden Form benutzt :
0" ccc"
Liest den Quelltext bis zum nächsten " und legt ihn als counted String mit abschließender Null im Dictionary ab. Zur Laufzeit wird die Adresse des ersten Zeichens des Strings adr auf dem Stack abgelegt. Dieses Wort ist statesmart und kann daher auch im Interpretierenden Zustand verwendet werden.

Abweichungen des volksFORTH83 von Programmieren in Forth

Die Gründe für die zahlreichen Abweichungen des Buches "Starting Forth" (Prentice Hall, 1982) bzw. "Programmieren in Forth" (Hanser, 1984) von Leo Brodie wurden bereits im Prolog aufgeführt. Sie sollen nicht wiederholt werden. Das Referenzbuch ist, trotz offenkundiger Mängel in Übersetzung und Satz, die deutsche Version, da sie erheblich weiter verbreitet sein dürfte. Im folgenden werden nicht nur Abweichungen aufgelistet, sondern auch einige Fehler mit angegeben. Die Liste erhebt keinen Anspruch auf Vollständigkeit. Wir bitten alle Leser, uns weitere Tips und Hinweise mitzuteilen. Selbstverständlich gilt das auch für alle anderen Teile dieses Handbuchs. Vielleicht wird es in der Zukunft ein Programm geben, das es ermöglicht, Programme aus "Starting Forth" direkt, ohne daß man diese Liste im Kopf haben muß, einzugeben.

Kapitel 1 - Grundlagen

7 -) im volksFORTH System wurde statt "Lexikon" konsequent der Begriff "Dictionary" verwendet.

9 -) Statt "?" durckt das volksFORTH "haeh?" , wenn XLERB eingegeben wird.

-) Selbstverständlich akzeptiert das volksFORTH Namen mit bis zu 31 Zeichen Länge.

11 -) Im volksFORTH kann auch der Hochpfeil "^" als Zeichen eines Namens verwendet werden.

-) Fußnote 6 : Der 83-Standard legt fest, daß ." nur innerhalb von Definitionen verwendet werden darf. Außerhalb muß man .(verwenden.

17 -) Das volksFORTH gibt bei Stackleerlauf irgendeine Zahl aus, nicht unbedingt eine Null !

18 -) Druckfehler : Bei dem Beispiel (n --) muß zwischen "(" und "n" ein Leerzeichen stehen. Das gilt auch für viele der folgenden Kommentare.

19 -) Bei dem Wort EMIT muß man natürlich angeben, in welchem Code das Zeichen kodiert ist. Beim volksFORTH auf dem C64 ist das nicht der ASCII-Zeichensatz, sondern der Commodore-spezifische. Am Besten ist es, statt 42 EMIT lieber ASCII * EMIT einzugeben.

Kapitel 2 - Wie man in Forth rechnet

23 -) Druckfehler : Selbstverständlich ist 10 1000 * kleiner als 32767, kann also berechnet werden. Kurios wird es nämlich erst bei 100 1000 * .

31 -) Die Operatoren /MOD und MOD arbeiten laut 83-Standard mit vorzeichenbehafteten Zahlen. Die Definition von Rest und Quotient bei negativen Zahlen findet man unter "Division, floored" in "Definition der Begriffe" oder unter /MOD im Glossar.
Seien Sie bitte bei allen Multiplikations- und Divisionsoperatoren äußerst mißtrauisch und schauen Sie ins Handbuch. Es gibt keinen Bereich von Forth, wo die Abweichungen der Systeme untereinander größer sind als hier.

38 -) .S ist im System als Wort vorhanden. Unser .S druckt nichts aus, wenn der Stack leer ist. Auch die Reihenfolge der Werte ist andersherum, der oberste Wert steht ganz links. Alle Zahlen werden vorzeichenlos gedruckt, d.h. -1 erscheint als 65535 .

-) 'S heißt im volksFORTH SP@ .

-) Das Wort DO funktioniert nach dem 83-Standard anders als im Buch. .S druckt nämlich, falls kein Wert auf dem Stack liegt, 32768 Werte aus. Ersetzt man DO durch ?DO , so funktioniert das Wort .S , aber nur dann, wenn man 2- wegläßt. Nebenbei bemerkt ist es sehr schlechter Stil, den Stack direkt zu addressieren !

39 -) <ROT ist im volksFORTH unter dem Namen -ROT vorhanden.

Kapitel 3 - Der Editor und externe Textspeicherung

Beim volksFORTH auf dem ST wird ein leistungsfähiger Bildschirmeditor mitgeliefert, dessen Bedienung sich von dem im Buch benutzten Zeileneditor stark unterscheidet.

Kapitel 4 - Entscheidungen

75 -) Die Vergleichsoperatoren müssen nach dem 83-Standard -1 auf den Stack legen, wenn die Bedingung wahr ist. Dieser Unterschied zieht sich durch das ganze Buch und wird im folgenden nicht mehr extra erwähnt. Also : -1 ist "wahr" , 0 ist "falsch" . Daher entspricht 0= nicht mehr NOT. NOT invertiert nämlich jedes der 16 Bit einer Zahl, während 0= falsch liefert,

wenn mindestens eins der 16 Bit gesetzt ist. Überall, wo NOT steht, sollten Sie 0= verwenden.

84 -) Das Wort ?STACK im volksFORTH liefert keinen Wert, sondern bricht die Ausführung ab, wenn der Stack über- oder leerläuft. Daher ist das Wort ABORT" überflüssig.

-) Statt -< muß es in der Aufgabenstellung 6 natürlich =< heißen. Für positive Zahlen tut UWITHIN im volksFORTH dasselbe.

Kapitel 5 - Die Philosophie der Festkomma-Arithmetik

89 -) Eine Kopie des obersten Wertes auf dem Returnstack bekommt man beim volksFORTH mit R@ . Die Worte I und J liefern die Indices von DO-Schleifen. Bei einigen Forth-Systemen stimmt der Index mit dem ersten bzw. dritten Element auf dem Returnstack überein. Der dann mögliche und hier dokumentierte Mißbrauch dieser Worte stellt ein Beispiel für schlechten Programmierstil dar. Also: Benutzen sie I und J nur in DO-Schleifen.

91 -) Die angedeutete Umschaltung zwischen Vokabularen geschieht anders als beim volksFORTH.

93 -) Fußnote: Es trifft nicht immer zu, daß die Umsetzung von Fließkommazahlen länger dauert als die von Integerzahlen. Insbesondere dauert die Umsetzung von Zahlen beim volksFORTH länger als z.B. bei verschiedenen BASIC-Interpretern. Der Grund ist darin zu suchen, daß die BASICs nur Zahlen zur Basis 10 ausgeben und daher für dieses Basis optimierte Routinen verwenden können während das volksFORTH Zahlen zu beliebigen Basen verarbeiten kann. Es ist aber durchaus möglich, bei Beschränkung auf die Basis 10 eine erheblich schnellere Zahlenausgabe zu programmieren.

98 -) */MOD im volksFORTH arbeitet mit vorzeichenbehafteten Zahlen. Der Quotient ist nur 16 Bit lang.

Kapitel 6 - Schleifenstrukturen

104 -) Die Graphik auf dieser Seite stellt Implementationsdetails dar, die für das polyFORTH gelten, nicht aber für das volksFORTH. Reißen Sie bitte diese Seite aus dem Buch heraus.

108 -) J liefert zwar den Index der äußeren Schleife, aber nicht den 3. Wert auf dem Returnstack.

110 -) Das Beispiel TEST funktioniert nicht. Beim 83-Standard sind DO und LOOP geändert worden, so daß sie jezt den gesamten Zahlenbereich von 0...65535 durchlaufen können. Eine Schleife n n DO ... LOOP exekutiert also jetzt 65536 - mal und nicht nur einmal, wie es früher war.

111 -) Beim volksFORTH wird beim Eingeben von nichtexistenten Worten nicht der Stack geleert, denn der Textinterpreter führt nicht ABORT" aus, sondern ERROR" . Den Stack leert man durch Eingabe der Worte CLEARSTACK oder ABORT .

114 -) LEAVE wurde im 83-Standard so geändert, daß sofort bei Ausführen von LEAVE die Schleife verlassen wird und nicht erst beim nächsten LOOP. Daher ändert LEAVE auch nicht die obere Grenze.

Kapitel 7 - Zahlen, Zahlen, Zahlen

125 -) Es ist systemabhängig, ob EMIT Steuerzeichen ausgeben kann.

130 -) Seien Sie mißtrauisch ! Das Wort U/MOD heißt im 83-Standard UM/MOD .

-) Das Wort /LOOP ist nun überflüssig geworden, da das normale Wort LOOP bereits alle Zahlen durchlaufen kann.

132 -) Beim volksFORTH werden nur Zahlen, die "," oder "." enthalten, als 32-Bit Zahlen interpretiert. "/" und ":" sind nicht erlaubt.

137 -) Der 83-Standard legt fest, daß SIGN ein negatives Vorzeichen schreibt, wenn der oberste (und nicht der dritte) Wert negativ ist. Ersetzen Sie bitte jedes SIGN durch ROT SIGN , wenn Sie Beispiele aus dem Buch abtippen.

-) Für HOLD gilt dasselbe wie das für EMIT (Abweichung Seite 19) gesagte. Benutzen Sie statt 46 HOLD besser ASCII - HOLD .

140 -) D- DMAX DMIN und DU< sind nicht vorhanden.

141 -) 32-Bit Zahlen können in eine Definition geschrieben werden, indem sie durch einen Punkt gekennzeichnet werden. Die Warnung für experimentierfreudige Leser trifft beim volksFORTH also nicht zu.

-) M+ M/ und M*/ sind nicht vorhanden. Für M/ können Sie M/MOD NIP einsetzen und für M+ etwa EXTEND D+ .

143 -) U* heißt im 83-Standard UM* und U/MOD UM/MOD

149 -) Aufgabe 7 ist großer Quark ! Die Tatsache, daß viele Forth-Systeme .. als doppelt genaue Null interpretieren, bedeutet nicht, daß es sich um keinen Fehler der Zahlenkonversion handelt. Das volksFORTH toleriert solche Fehleingaben nicht.

Kapitel 9 - Forth intern

178 -) Zu den Fußnoten 1,2 : Der 83-Standard schreibt für das Wort FIND ein anderes Verhalten vor, als hier angegeben wird. Insbesondere sucht FIND nicht nach dem nächsten Wort im Eingabestrom, sondern nimmt als Parameter die Adresse einer Zeichenkette (counted String), nach der gesucht werden soll. Auch die auf dem Stack abgelegten Werte sind anders vorgeschrieben.

-) Das Beispiel 38 ' GRENZE ! ist schlechter Forth Stil; es ist verpönt, Konstanten zu ändern. Da das Wort ' nach dem 83-Standard die Kompilationsadresse des folgenden Wortes liefert (und nicht die Parameterfeldadresse), der Wert einer Konstanten aber häufig in ihrem Parameterfeld aufbewahrt wird, funktioniert das Beispiel auf vielen Forth Systemen, die dem 83-Standard entsprechen, folgendermaßen: 38 ' GRENZE >BODY ! .

179 -) Fußnote 3 : Die Fußnote trifft für den 83-Standard nicht zu. ' verhält sich wie im Text beschrieben.

180 -) Das volksFORTH erkennt 32-Bit Zahlen bereits seriemäßig, daher gibt es kein Wort 'NUMBER . Wollen Sie

eigene Zahlenformate erkennen (etwa Floating Point Zahlen) , so können Sie dafür das deferred Wort NOTFOUND benutzen.

181 -) Die vorgestellte Struktur eines Lexikoneintrags ist nur häufig anzutreffen, aber weder vorgeschrieben noch zwingend. Zum Beispiel gibt es Systeme, die keinen Code Pointer aufweisen. Das volksFORTH sieht jedoch ähnlich wie das hier vorgestellte polyFORTH aus.

188 -) Druckfehler : Statt ABORT' muß es ABORT" heißen.

189 -) Der 83-Standard schreibt nicht vor, daß EXIT die Interpretation eines Disk-Blockes beendet. Es funktioniert zwar auch beim volksFORTH, aber Sie benutzen besser das Wort \\ .

190 -) Die Forth Geographie gilt für das volksFORTH nicht; einige der Abweichungen sind :

-) Die Variable H heißt im volksFORTH DP (=Dictionary Pointer) .

-) Der Eingabepuffer befindet sich nicht bei S0 , sondern TIB liefert dessen Adresse.

-) Der Bereich der Benutzervariablen liegt woanders.

191 -) Zusätzliche Definitionen : Beim volksFORTH ist die Bibliothek anders organisiert. Ein Inhaltsverzeichnis der Disketten können Sie sich mit FILES ausgeben lassen.

197 -) Der Multitasker beim volksFORTH ist gegenüber dem des polyFORTH vereinfacht. So besitzen alle Terminal-Einheiten (wir nennen sie Tasks) gemeinsam nur ein Lexikon und einen Eingabepuffer. Es darf daher nur der OPERATOR (wir nennen in Main- oder Konsolen-Task) kompilieren.

198 -) Im volksFORTH sind SCR R# CONTEXT CURRENT >IN und BLK keine Benutzervariablen.

200 -) Das über Vokabulare gesagte trifft auch für das volksFORTH zu, genaueres finden Sie unter Vokabularstruktur im Handbuch.

202 -) LOCATE heißt im volksFORTH VIEW .

203 -) EXECUTE benötigt nach dem 83-Standard die Kompilationsadresse und nicht mehr die Parameterfeldadresse eines Wortes. Im Zusammenhang mit ' stört das nicht, da auch ' geändert wurde.

Kapitel 10 - Ein- und Ausgabeoperationen

211 -) Die angesprochene Funktion von FLUSH führt nach dem 83-Standard das Wort SAVE-BUFFERS aus. Es schreibt alle geänderten Puffer auf die Diskette zurück. Das Wort FLUSH existiert ebenfalls. Es unterscheidet sich von SAVE-BUFFERS dadurch, daß es nach dem zurückschreiben alle Puffer löscht. Die Definition ist einfach :

```
: flush  save-buffers  empty-buffers ;
```

FLUSH wird benutzt, wenn man die Diskette wechseln will, damit von BLOCK auch wirklich der Inhalt dieser Diskette geliefert wird.

-) Fußnote 4 trifft für das volksFORTH nicht zu.

212 -) Der 83-Standard schreibt vor, daß BUFFER sehr wohl darauf achtet, ob ein Puffer für diesen Block bereits existiert. BUFFER verhält sich genauso wie BLOCK , mit dem einzigen Unterschied, daß das evtl. erforderliche Lesen des Blocks von der Diskette entfällt.

213 -) Wie schon erwähnt, muß bei den Beispielen auf dieser Seite S0 @ durch TIB ersetzt werden. Ebenso muß ' TEST durch ' TEST >BODY ersetzt werden. Das gilt auch für das folgende Beispiel BEZEICHNUNG .

217 -) Druckfehler : WORT ist durch QUATSCH zu ersetzen.

219 -) <CMOVE heißt nach dem 83-Standard CMOVE> . Der Pfeil soll dabei andeuten, daß man das Wort benutzt, um im Speicher vorwärts zu kopieren. Vorher war gemeint, daß das Wort am hinteren Ende anfängt.

-) Für MOVE wird im 83-Standard ein anderes Verhalten vorgeschlagen. MOVE wählt zwischen CMOVE und CMOVE> , je nachdem , in welche Richtung verschoben wird.

223 -) Fußnote 10 : QUERY ist auch im 83-Standard vorgeschrieben.

-) WORD kann, muß aber nicht seine Zeichenkette bei HERE ablegen.

224 -) Nach dem 83-Standard darf EXPECT dem Text keine Null mehr anfügen.

229 -) Fußnote 14 : PTR heißt beim volksFORTH DPL (= decimal point location) .

230 -) Ersetzen sie WITHIN durch UWITHIN oder ?INNERHALB. NOT muß durch 0= ersetzt werden, da die beiden Worte nicht mehr identisch sind.

232 -) Druckfehler in Fußnote 15 : Der Name des Wortes ist natürlich -TEXT und nicht TEXT . Außerdem müssen die ersten beiden "/" durch "@" ersetzt werden. Das dritte "/" ist richtig.

Kapitel 11 - Wie man Compiler erweitert...

247 -) BEGIN DO usw. sehen im volksFORTH anders aus, damit mehr Fehler erkannt werden können.

248 -) Fußnote 2 : Die Erläuterungen beziehen sich auf polyFORTH, im volksFORTH siehts wieder mal anders aus. Die Frage ist wohl nicht unberechtigt, warum in einem Lehrbuch solche implementationsabhängigen Details vorgeführt werden.

-) Fußnote 3 : Eine Konstante im volksFORTH kommt, falls sie namenlos (siehe Kapitel "Der Heap" im Handbuch) definiert wurde, mit 2 Zellen aus.

249 -) Die zweite Definition von 4DAZU funktioniert beim volksFORTH nicht, da das Wort : dem ; während der Kompilation Werte auf dem Stack übergibt. Das ist durch den 83-Standard erlaubt.

250 -) Druckfehler : Statt [SAG-HALLO] muß es [SAG-HALLO] heißen.

251 -) Die Beispiele für LITERAL auf dieser Seite funktionieren, da LITERAL standardgemäß benutzt wird.

-) Druckfehler : Statt ICH SELBST muß es ICH-SELBST

heißen.

252 -) Bei Definitionen, die länger als eine Zeile sind, druckt das volksFORTH am Ende jeder Zeile "compiling" statt "ok" aus.

255 -) Die Beispiele für INTERPRET und] entstammen dem polyFORTH. Eine andere Lösung wurde im volksFORTH verwirklicht. Hier gibt es zwei Routinen, je eine für den interpretierenden und kompilierenden Zustand. INTERPRET führt diese Routinen vektoriell aus.

-) Fußnote 4 trifft für das volksFORTH nicht zu.

Kapitel 12 - Drei Beispiele

270 -) Druckfehler : In WÖRTER ist ein 2DROP zuviel; ferner sollte >- >IN sein.

-) In BUZZ muß es statt WORT .WORT heißen.
(Reingefallen : Es muß WÖRTER sein!)

239 -) Die Variable SEED muß wohl SAAT heißen. Ob der Übersetzer jemals dieses Programm kompiliert hat?

-) Nebenbei: Im amerikanischen Original hatten die Worte NÄCHSTES WÖRTER FÜLLWÖRTER und EINLEITUNG noch einen Stackkommentar, ohne die das Programm unverständlich wird, besonders bei diesem fürchterlichen Satz. Also, wenn Sie an Ihren Fähigkeiten zweifeln, sollten Sie sich das amerikanische Original besorgen.

Abweichungen des volksFORTH83 von 'FORTH TOOLS' von A. Anderson & M. Tracy

p.15 CLEAR macht im volksFORTH83 etwas anderes als in FORTH TOOLS. Benutze statt CLEAR das Wort CLEARSTACK oder definiere

```
' clearstack Alias clear
```

p.27 .S druckt die Werte auf dem Stack anders herum aus, ausserdem fehlt der Text "Stack:"

p.34 2ROT fehlen. Benutze

```
: 2rot    5 roll 5 roll ;
```

p.42 Statt DIRECTORY heißt es FILES

p.45 THRU druckt keine Punkte aus.

p.46 Der Editor funktioniert anders.

p.64 volksFORTH83 enthaelt kein Wort ? . Benutze

```
: ?   @ . ;
```

p.99 AGAIN gibt es nicht. Benutze stattdessen REPEAT oder definiere

```
' REPEAT  Alias  AGAIN  immediate restrict
```

p.103 Benutze ' extend Alias s>d

p.107 DU< fehlt.

p.116 SPAN enthaelt 6 Zeichen, da "SPAN ?" genau 6 Zeichen lang ist. Das System benutzt naemlich ebenfalls EXPECT. Daher geht das Beispiel auf Seite 117 auch nicht. Damit es geht, muss man alle Worte in einer Definition zusammenfassen.

p.118 CPACK entspricht dem Wort PLACE.

p.125 Benutze

```
: String   Create dup , 0 c, allot  Does> 1+ count ;
```

p.126 " kann nicht interpretiert werden. Zwei Gänsefüßchen hintereinander sind ebenfalls nicht erlaubt.

p.141 Das Wort IS aus dem volksFORTH83 kann innerhalb von Definitionen benutzt werden.

p.146 Es gibt keine Variable WIDTH , da bei Namen alle Zeichen gültig sind.

p.149 Benutze

```
: >link      >name 2- ;
: link>      2+ name> ;
: n>link     2- ;
: l>name     2+ ;
```

p.166 STOP entspricht \\

p.167 Bei FIND kann das Flag auch die Werte -2 oder +2 annehmen.

p.184 ORDER ist nicht in ONLY , sondern in FORTH enthalten. Außerdem druckt es auch nicht "Context:" oder "Current:" aus. Current wird stattdessen einfach zwei Leerzeichen hinter Context ausgegeben.

Fehlermeldungen des volksFORTH83

Das volksFORTH83 gibt verschiedene Fehlermeldungen und Warnungen aus. Zum Teil wird dabei das zuletzt eingegebene Wort wiederholt. Diese Meldungen sind im folgenden nach Gruppen getrennt, aufgelistet:

Kernel

?
Der eingegebene String konnte nicht mit NUMBER in eine Zahl umgewandelt werden. Beachten Sie den Inhalt von BASE.

Adress Error !
Dies ist ein fataler Fehler. Der Prozessor hat versucht, mit einer Wortoperation auf eine ungerade Adresse zuzugreifen. Sollten Sie keinen fehlerhaften Maschinencode geschrieben haben, so ist das Forth "zerschossen".

beyond capacity
Der angesprochene Block ist physikalisch nicht vorhanden. Beachten Sie den Inhalt von OFFSET.

Bus Error !
Der Prozessor hat versucht, auf einen nichtexistenten Speicher zuzugreifen. Prüfen Sie bitte alle verwendeten Langwortoperationen.

compile only
Das eingegebene Wort darf nur innerhalb von :-Definitionen verwendet werden.

crash
Es wurde ein deferred Wort aufgerufen, dem noch kein auszuführendes Wort mit IS zugewiesen wurde.

Dictionary full
Der Speicher für das Dictionary ist erschöpft. Sie müssen die Speicherverteilung ändern oder Worte mit FORGET vergessen.

Division by 0 !
Es wurde versucht, durch Null zu teilen.

division overflow
Die Division zweier Zahlen wurde abgebrochen, da das Ergebnis nicht im Zahlenbereich darstellbar ist.

exists
Das zuletzt definierte Wort ist im Definitons-Vokabular schon vorhanden. Dies ist kein Fehler, sondern nur ein Hinweis!

haeh?
Das eingegebene Wort konnte weder im Dictionary gefunden noch als Zahl interpretiert werden.

Illegal Instruction !
Dies ist ein fataler Fehler. Der Prozessor hat versucht, einen nicht erlaubten Befehl auszuführen. Sollten Sie Maschinencode geschrieben haben, so prüfen Sie ihn bitte auf unzulässige Kombinationen von Befehl und Adressierungsart. Sind Sie sich keiner Schuld bewußt, so ist das Forth zerstört.

invalid name
Der Name des definierten Wortes ist zu lang (mehr als 31 Zeichen) oder zu kurz (Der Name sollte dem definierenden Wort unmittelbar folgen).

is symbol
Das Wort, das mit FORGET vergessen werden sollte, befindet sich auf dem Heap. Benutzen Sie dafür CLEAR.

nein
Der Bereich von Blöcken, der mit CONVEY kopiert werden sollte, ist leer oder viel zu groß.

no file
Auf Ihrem System sind keine Files benutzbar. Die Variable FILE muß daher den Wert Null haben. Siehe auch die Fehlermeldungen des Fileinterfaces

not deferred
Es wurde versucht, mit IS einem Wort, das nicht deferred ist, eine auszuführende Prozedur zuzuweisen.

not enough parameters
Ein Wort erwartete mehr Werte auf dem Stack, als vorhanden waren. Dieser Fehler wird bei Ausführung des Wortes ARGUMENTS erzeugt.

protected
Es wurde versucht, ein Wort zu vergessen, das mit SAVE geschützt wurde. Benutzen Sie die Phrase:
```
' <name> >name 4 - (forget  save
```

read error !
Bei einem Lesezugriff auf die Diskette trat ein Fehler auf. Der zu lesende Buffer wurde nicht markiert.

stack empty
Es wurden mehr Werte vom Stack genommen als vorhanden waren.

tight stack
Es befanden sich zuviele Werte auf dem Stack, sodaß der Speicher erschöpft war. Legen Sie weniger Werte auf den Stack, oder sparen Sie Speicherplatz im Dictionary.

unstructured
Kontrollstrukturen wurden falsch verschachtelt oder weggelassen. Diese Fehlermeldung wird auch ausgegeben, wenn sich die Zahl der Werte während der Kompilation einer :- Definition zwischen : und ; geändert hat.

Userarea full
Es wurden mehr als 124 Uservariablen definiert. Das ist nicht zulässig.

Vocabulary stack full
Es wurden zuviele Vokabulare mit ALSO in den festen Teil der Suchreihenfolge übernommen. Benutzen Sie ONLY oder TOSS, um Vokabulare zu entfernen.

write error !
Siehe " read error ..."

Fileinterface

close error
Das mit SAVESYSTEM erzeugte File konnte nicht ordnungsgemäß geschlossen werden.

Datei nicht gefunden
Eine nach USE oder anderen Worten angegebene Datei konnte nicht auf dem Massenspeicher gefunden werden. Prüfen Sie bitte, ob PATH korrekt gesetzt ist.

Disk schreibgeschützt
Es wurde versucht, auf eine schreibgeschützte Diskette zu schreiben. Prüfen Sie bitte, ob ein Block fälschlich als UPDATEd markiert wurde.

Disk voll
Die Diskette ist voll, daher kann MORE nicht mehr ausgeführt werden.

Dos-Error #00
Es trat ein Fehler im TOS auf. Die Nummer hat folgende Bedeutung :

Bios	01	Error
	02	Drive not ready
	03	Unknown command
	04	CRC Error
	05	Bad request
	06	Seek error
	07	Unknown media
	08	Sector not found
	09	Out of paper
	10	Write fault
	11	Read fault
	14	Media change detected
	15	Unknown device
	16	Bad sectors on format
	17	Insert other disk (request)
GEM-Dos	32	Invalid function number
	35	Handle pool exhausted
	36	Acess denied
	40	Invalid memory block adress

46 Invalid drive specification
47 No more files
64 Range error
65 GEMDOS internal error
66 Invalid executable file format
67 Memory block growth failure

Wenn Sie mit diesen Fehlerbeschreibungen nicht viel anfangen können, so trösten Sie sich: wir können es auch nicht. Es ist auch nicht erforderlich; erscheinen doch die Fehlernummern ziemlich unmotiviert und selten fehlerspezifisch, oft liefern sie keinerlei Hinweis auf die Art des auftretenden Fehlers.

missing filename
Auf das Wort SAVESYSTEM muß ein Filename folgen, der aber nicht vorhanden war.

no device
SAVESYSTEM konnte nicht das gewünschte File erzeugen. Prüfen Sie bitte, ob die Diskette vorhanden, fehlerfrei und nicht schreibgeschützt ist.

No file !
Es wurde versucht, die Diskette im Directmodus mit MORE zu verlängern oder ihr mit ASSIGN ein File zuzuweisen. Prüfen Sie bitte mit FILE? das aktuelle File.

Pfad nicht gefunden
Der mit DIR angegebene Pfad existiert nicht. Überzeugen Sie sich bitte, daß die richtige Diskette im Laufwerk steckt.

string too long
Der nach ASSIGN, USE, MAKEFILE, LOADFROM, MAKE oder INCLUDE angegebene Filename ist zu lang. Entfernen sie bitte Subdirectories aus dem Pfadnamen, und setzen Sie DIR entsprechend.

Ungültige Handle#
Es liegt ein Fehler bei der Verwaltung der Files vor. Das Forth versucht, mit Hilfe einer durch das TOS vergebenen Zahl, der "Handle", auf ein File zuzugreifen, das bereits wieder geschlossen worden ist. Versuchen Sie, von Hand die Handle in der Liste der FCBs zu löschen.

Ungültiges Laufwerk
Das durch SETDRIVE bzw. A: B: C: oder D: angegebene Laufwerk existiert nicht.

write error
SAVESYSTEM konnte nicht das gesamte System speichern. Prüfen Sie bitte, ob die Diskette voll ist.

Wrong range
CONVEY kann die angegebenen Blöcke nicht kopieren, da sie nicht existieren. Prüfen Sie bitte, ob das File, in das kopiert werden soll, ausreichend lang ist.

Zugriff nicht möglich
Es wurde versucht, auf einen Block zuzugreifen, der nicht vorhanden ist. Überprüfen Sie bitte den Inhalt von OFFSET sowie das File, auf das zugegriffen werden sollte.

ALLOCATION

No more RAM
Der Arbeitsspeicher des ST ist aufgebraucht, so daß kein Speicher mehr angefordert werden kann.

mfree Error!
Der Speicherbereich, der mit MFREE freigegeben werden sollte, wurde nicht allokiert. Prüfen Sie bitte, ob MFREE zweimal aufgerufen wurde oder ob sich die Anfangsadresse, die sie bei MALLOC erhalten haben und die Adresse, die sie MFREE mitgeben, übereinstimmen.

GEM-Library

Menu-Error
Object-Error
Graphic-Error
File Error
Window-Error
Resource-Error
Eine Routine der entsprechenden AES-Bibliothek signalisierte einen Fehler.

RELOCATION

a ticket to the moon with no return ...
Der Speicherplatz, der nach Ausführung von RELOCATE für den Returnstack übrig bliebe, ist zu klein.

cuts the dictionary
Der Speicherplatz, der nach Ausführung von BUFFERS oder RELOCATE für das Dictionary übrig bliebe, ist zu klein.

kills all buffers
Bei Ausführung würde RELOCATE keinen Blockbuffer im System lassen. Prüfen Sie die Argumente von RELOCATE. Die Anweisung 0 BUFFERS ist ebenfalls nicht zulässig.

Targetcompiler-Worte

Das volksFORTH83-System wurde mit einem sog. Targetcopiler erzeugt. Dieser Targetcompiler compiliert ähnlichen Quelltext wie das volksFORTH83-System. Es gibt jedoch Unterschiede. Insbesondere sind im Quelltext des volksFORTH83 Worte enthalten, die der Steuerung des Targetcompilers dienen, also nicht im Quelltext definiert werden. Wenn Sie sich also über eine Stelle des Quelltextes sehr wundern, sollten Sie in der folgenden Liste die fragwürdigen Worte suchen.

Eine andere Besonderheit betrifft Uservariablen. Wenn im Quelltext der Ausdruck [<name>] LITERAL auftaucht und <name> eine Uservariable ist, so wird bei der späteren Ausführung des Ausdrucks NICHT die Adresse der Uservariablen in der Userarea sondern die Adresse in dem Speicherbereich, in dem sich die Kaltstartwerte der Uservariablen befinden, auf den Stack gelegt.
Wenn die Kaltstartwerte von Uservariablen benötigt werden, wird dieser Ausdruck benutzt.

Folgende, im Quelltext oft auftretende Worte sind im Targetcompiler definiert:

displace Target here! .unresolved origin! Compiler H T there Tnext-link Onlypatch Host Tudp Tvoc-link Tnext-link move-threads